Intro to Mechatronics:

Arduino, Sensors & Basic Interfaces

ASSIGNMENT 1

LED with Resistor

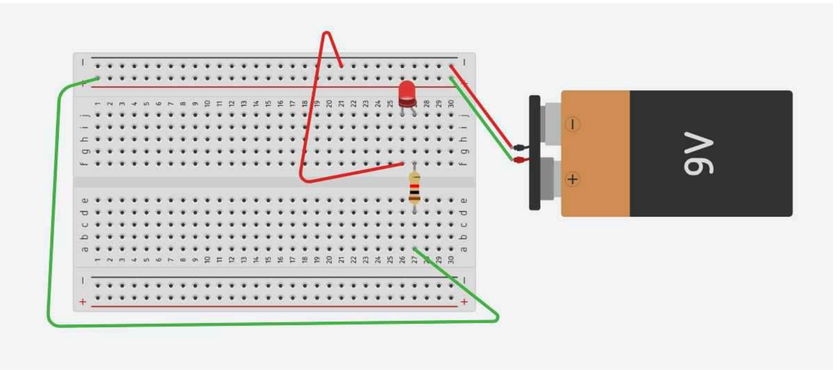
Components

- 9V Battery
- LED (red)
- Resistor (current-limiting)
- Breadboard
- Jumper wires

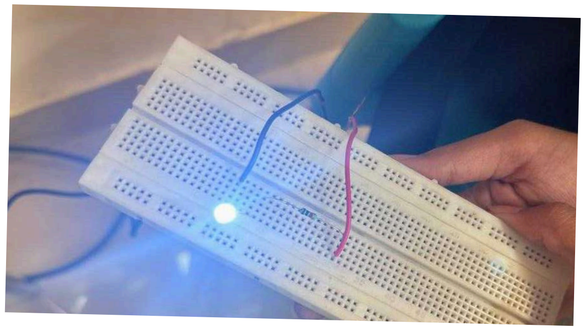
Learnings

- Learned importance of current-limiting resistor
- correct LED polarity.

IN TINKERCAD



IN BREADBOARD



Connection:

- 5V → Resistor → LED (+)
- LED (–) → GND
- Adapter + to +5 V rail, Adapter – to GND rail

ASSIGNMENT 2

Two LEDs in Series connection

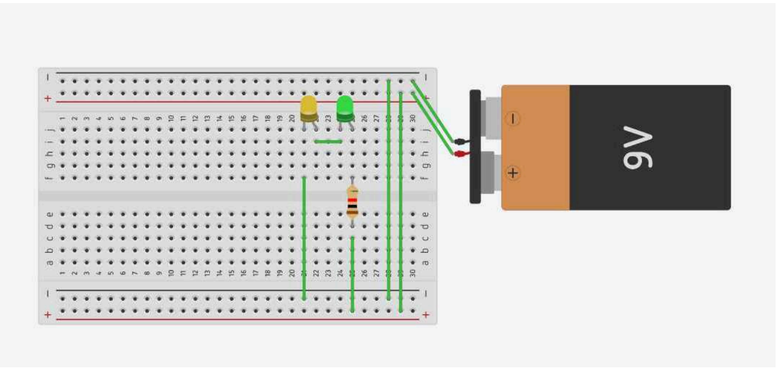
Components

- 2 × LEDs (white)
- 220Ω resistor (current limiting)
- Breadboard
- Jumper wires
- 5V adapter

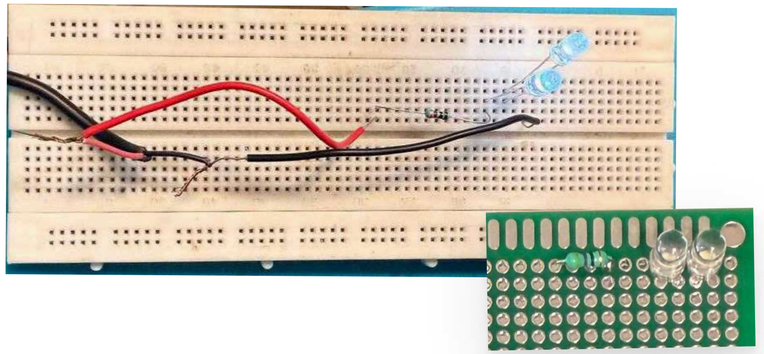
Learnings

- LEDs in series share current and divide voltage.
- Series connection reduces brightness.

IN TINKERCAD



IN BREADBOARD



Connection:

1. LED1 anode → LED2 anode
2. LED2 cathode → Resistor → GND rail
3. LED1 cathode → +5 V rail
4. Adapter + to +5 V rail, Adapter – to GND rail

ASSIGNMENT 3

Two LEDs in Parallel connection

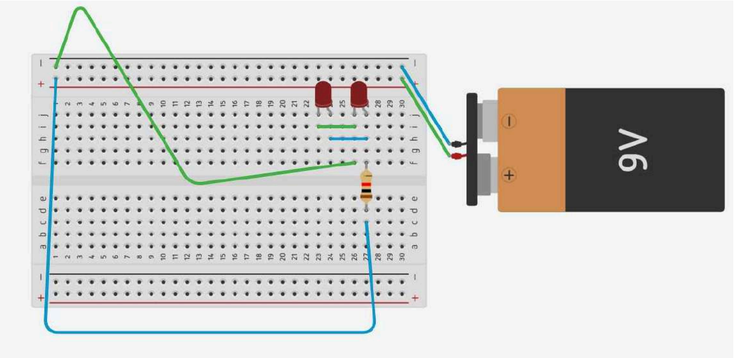
Components

- 2 × LEDs (white)
- 220Ω resistor (current limiting)
- Breadboard
- Jumper wires
- 5V adapter

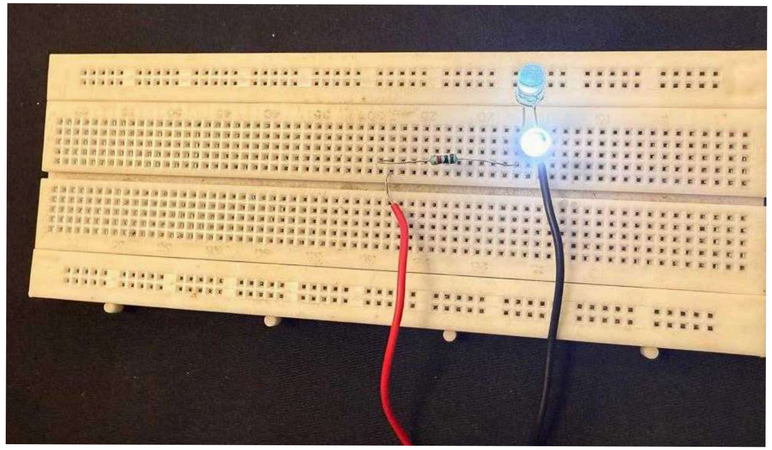
Learnings

- Parallel connection maintains brightness comparable to a single LED.
- Current draw increases with each additional LED.

IN TINKERCAD



IN BREADBOARD



Connection
Anode of each LED → +5 V rail
Cathode → resistor → GND rail
Adapter + → +5 V rail
Adapter – → GND rail

ASSIGNMENT 4

Controlling LED with Pontentiometer

To vary the brightness of an LED by directly adjusting the resistance using a potentiometer.

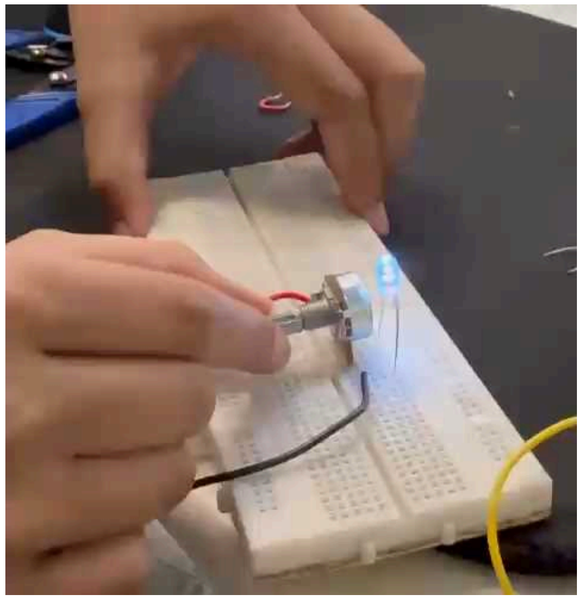
Components

- 1 × LED
- 1 × Potentiometer (10kΩ recommended)
- 1 × Resistor (220Ω–330Ω for LED protection)
- Breadboard or wires
- DC power supply (e.g., 3V–9V battery depending on LED)

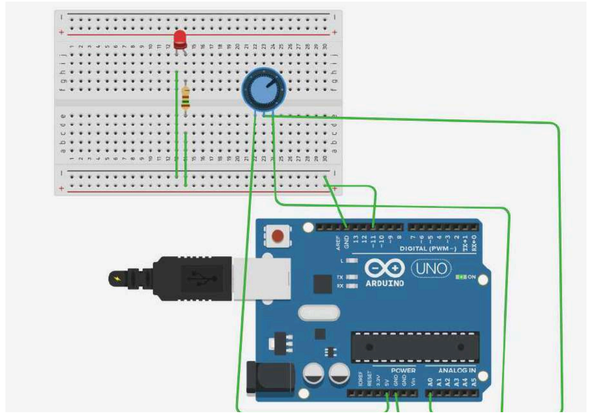
Circuit Setup

1. Battery positive terminal → Potentiometer outer leg
2. Potentiometer middle leg (wiper) → 220 ohm resistor → LED anode (positive leg)
3. LED cathode (negative leg) → Battery negative terminal

- Ask ChatGPT



IN TINKERCAD



ASSIGNMENT 3

Controlling LED with Pontentiometer

How It Works

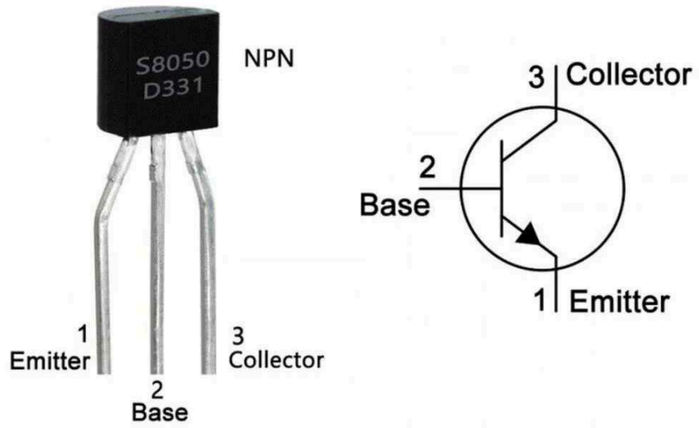
- The potentiometer changes resistance in the circuit.
- Lower resistance → More current → LED brighter.
- Higher resistance → Less current → LED dimmer.
- The fixed resistor ensures the LED never receives excessive current.

Learnings

- Don’t turn potentiometer to minimum resistance for long — it may overheat and damage components.
- Resistance controls LED brightness (Ohm’s Law).
- Potentiometer works as a variable resistor.
- Always check LED with Multimeter

ASSIGNMENT 5

LED Control Using NPN Transistor



- NPN transistor is a semiconductor device .
- A small current to the base allows a larger current to flow from collector to emitter.
- Commonly used as an electronic switch or amplifier in circuits.

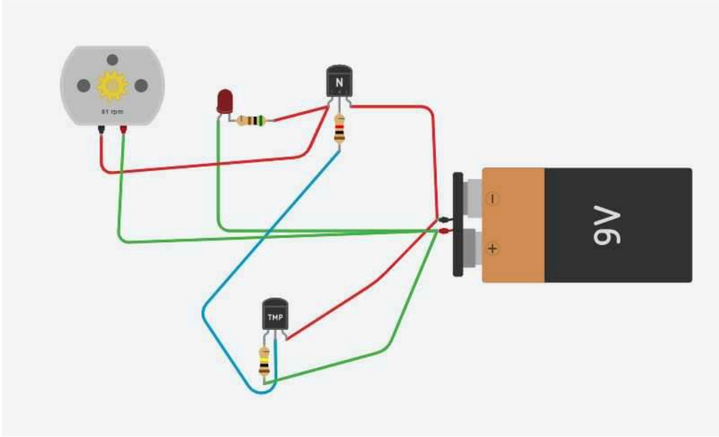
Components

- NPN transistor (e.g., BC547, 2N2222)
- LED
- 220 Ω resistor – limits current through the LED
- 1 kΩ resistor – limits current into the transistor's base
- Breadboard
- Jumper wires
- 5 V adapter

Circuit Setup

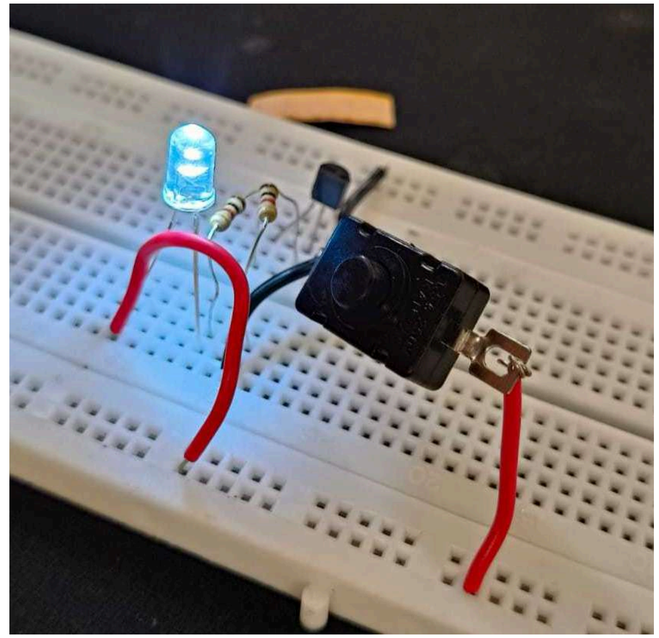
- LED: Anode→+5V, Cathode→Resistor→Collector
- NPN: Base→Control+Resistor, Emitter→GND
- Resistors: LED(220Ω-1kΩ), Base(1kΩ-10kΩ)

IN TINKERCAD



ASSIGNMENT 4

LED Control Using NPN Transistor

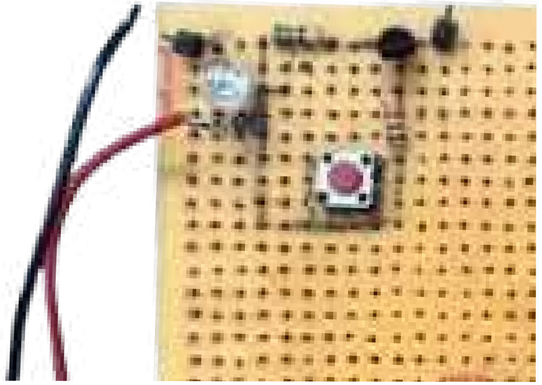


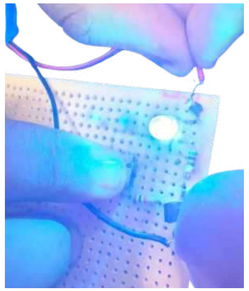
Connection

Anode of each LED → +5 V rail
Cathode → resistor → GND rail
Adapter + → +5 V rail
Adapter – → GND rail

How it works

NPN = electrically controlled switch
Small base current controls large LED current
Base >0.7V = transistor ON = LED ON
No base current = LED OFF





Learnings

- NPN transistor controls large current with small base current.
- Base positive vs emitter allows current flow collector to emitter.
- Acts as switch or amplifier.

ASSIGNMENT 6

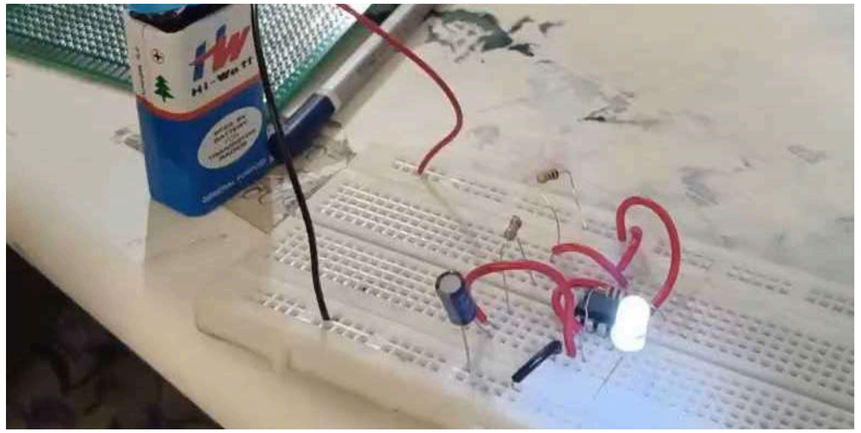
555

Connection

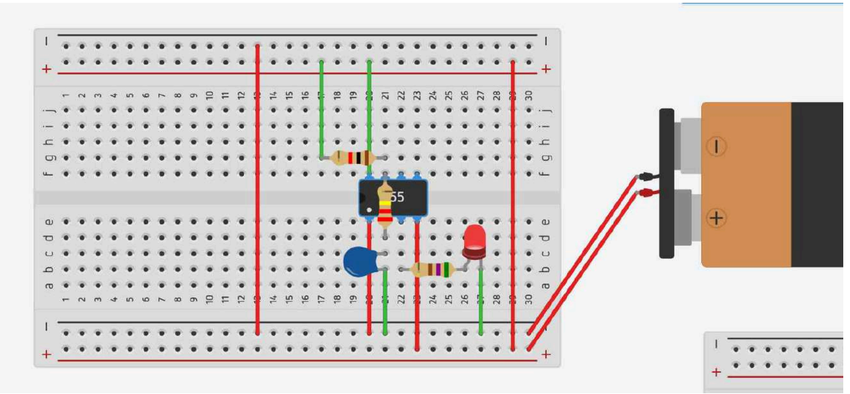
Pin 1→GND, Pin 8→VCC, Pin 4→Pin 8
Pin 2→Pin 6 (direct jumper)
R1: Pin 8 to Pin 7, R2: Pin 7 to Pin 6
Capacitor: +ve to Pin 2/6, -ve to Pin 1
Pin 3→LED→Resistor→GND

Components

555 Timer IC
R1 (Brown-Black-Orange) = 10kΩ
R2 (Yellow-Violet-Brown) = 470Ω
Capacitor (Blue) = 10µF-100µF
LED + current limiting resistor
9V Battery + breadboard + jumper wires



IN TINKERCAD



ASSIGNMENT 6

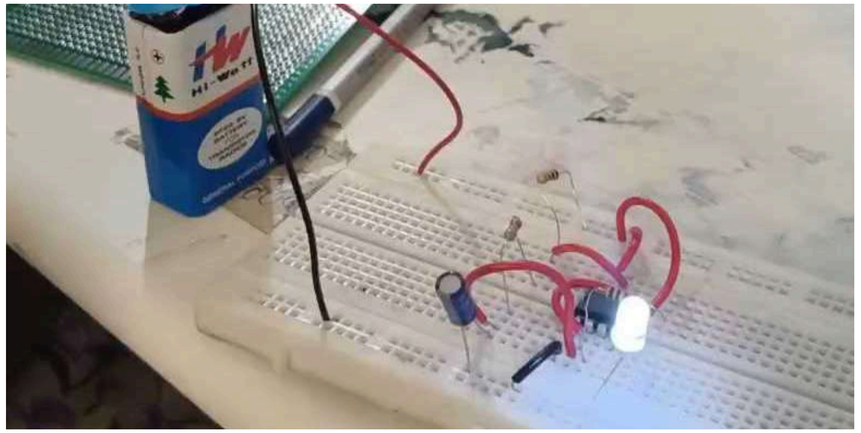
555

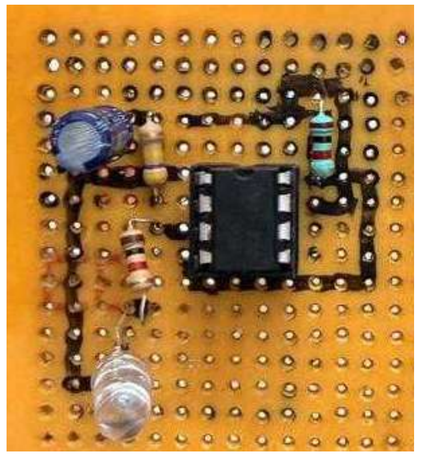
How it works

555 configured in astable mode (oscillator)
R1 & R2 control charging time, R2 controls discharging time
Capacitor charges through R1+R2, discharges through R2
Output (Pin 3) switches HIGH/LOW creating blinking pattern
Frequency = 1.44/((R1+2×R2)×C

Learnings

555 timer generates square wave without microcontroller
Timing controlled by RC values (resistor-capacitor)
Astable mode = continuous oscillation
Duty cycle = (R1+R2)/(R1+2×R2) × 100%
Versatile IC for timing, oscillation, and pulse generation

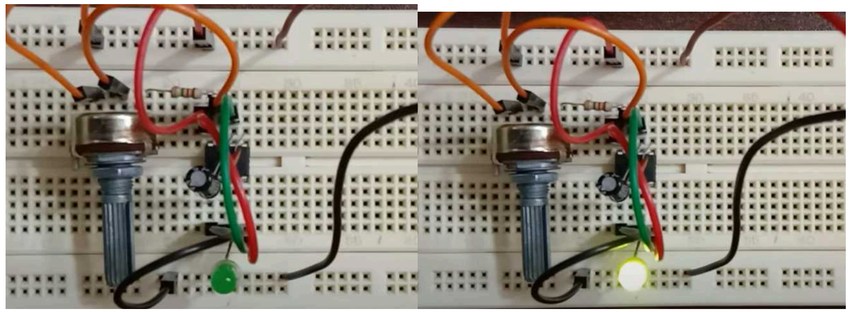




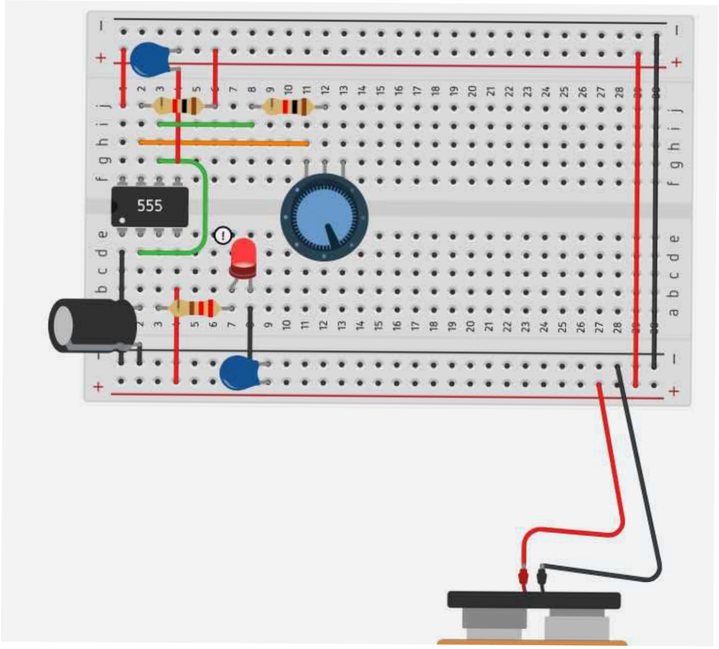
ASSIGNMENT 7

LED Control Using 555 Timer IC

- To use a 555 timer IC in an astable mode to blink an LED at a variable speed, controlled by a potentiometer.



IN TINKERCAD



Components

- 555 timer IC – 1
- LED – 1
- Potentiometer (10kΩ) – 1
- Resistors – 2 (e.g., 1kΩ, 470Ω)
- Capacitor (10µF–100µF) – 1
- Breadboard – 1
- 9V battery with clip – 1
- Jumper wires – few

Connection

- Pin 1 (GND) → Battery negative
- Pin 8 (VCC) → Battery positive
- Pin 2 (Trigger) connected to Pin 6 (Threshold)
- Potentiometer + resistors + capacitor connected between VCC, pins 2/6, and GND to set timing
- Pin 3 (Output) → LED + resistor → GND

ASSIGNMENT 8

Controlling LEDs with Push Buttons

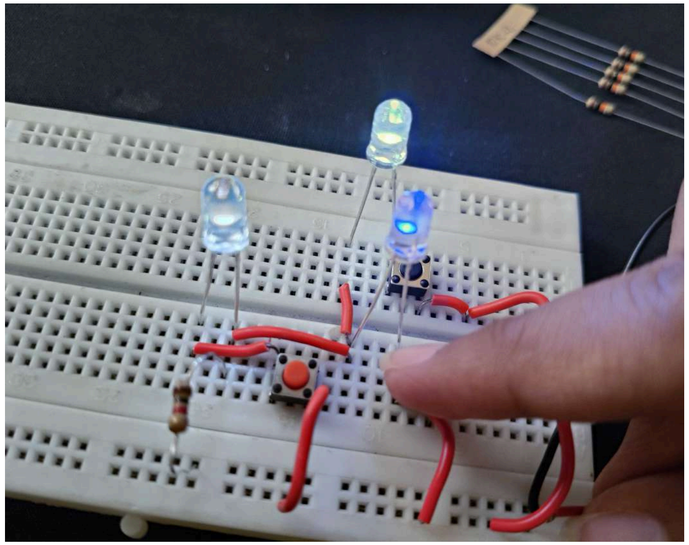
- To turn LEDs ON and OFF using push buttons in a basic breadboard circuit.

Components

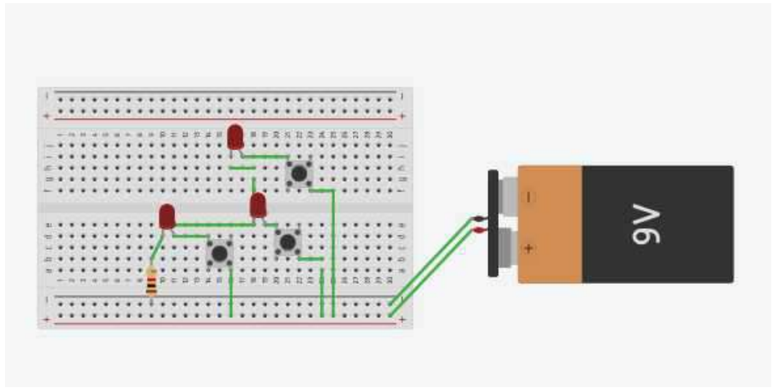
- LEDs – 3
- Push buttons – 3
- Resistor (220Ω) – 1 (for LED protection)
- Breadboard – 1
- 9V battery with clip – 1
- Jumper wires – few

Circuit

- Battery positive (+) → One leg of each push button.
- Other leg of each push button → LED anode (positive leg).
- LED cathode (negative leg) → 220 Ω resistor → Battery negative (–).



IN TINKERCAD

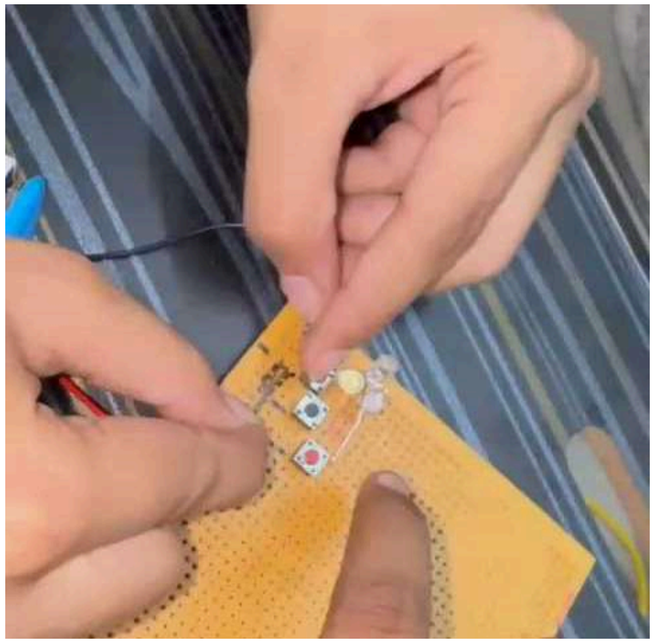


ASSIGNMENT 8

Controlling LEDs with Push Buttons

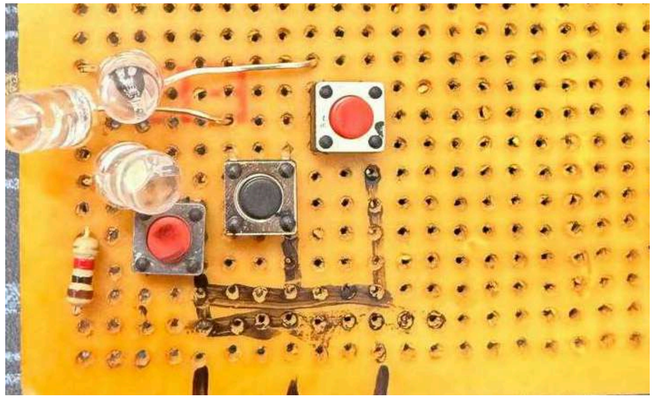
How it works

Push buttons can act as simple ON/OFF switches.
A resistor is always needed with LEDs to prevent damage.
Breadboard rows and columns have specific connection patterns—wrong placement may cause circuit failure.



Learning

LED needs resistor to prevent damage
Breadboard connection patterns matter
Push buttons = manual digital control
Base for complex switching circuits



ASSIGNMENT 9

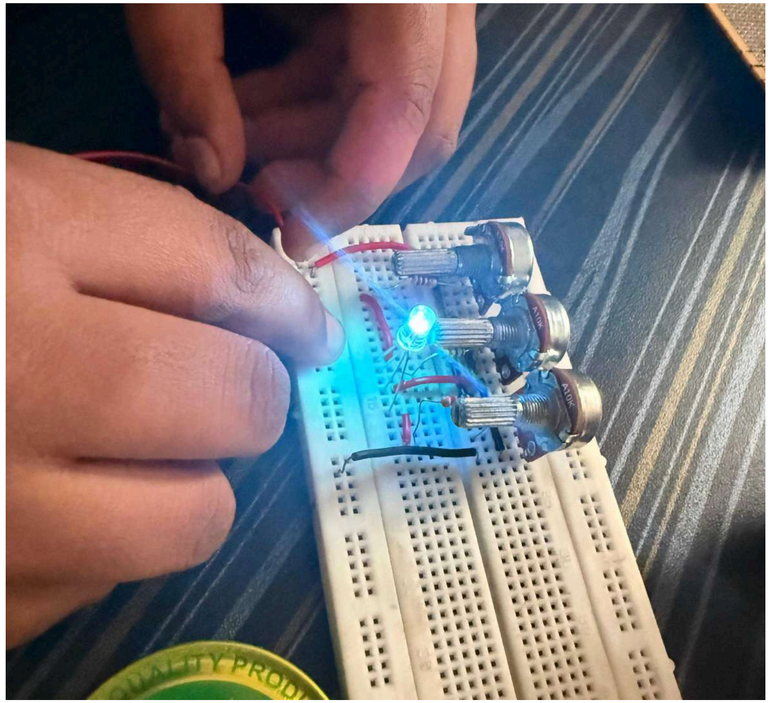
Controlling RGB LED with 3 Potentiometers

Components

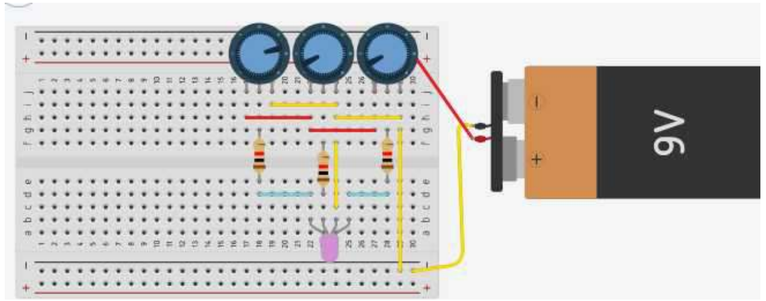
1. 9V Battery + connector
2. RGB LED (common cathode)
3. 3 × Potentiometers – 10 kΩ
4. 3 × Resistors – 330 Ω (current limiting, one for R/G/B each)
5. Breadboard + jumper wires

Circuit

- Battery + → breadboard + rail, Battery – → GND rail
- RGB LED common cathode → GND
- Each LED pin (R, G, B) → 330 Ω resistor → Potentiometer middle pin
- Potentiometer sides → +9V and GND (acts as variable voltage divider)
- Adjust pots → vary brightness of each color → mix colors



IN TINKERCAD



ASSIGNMENT 9

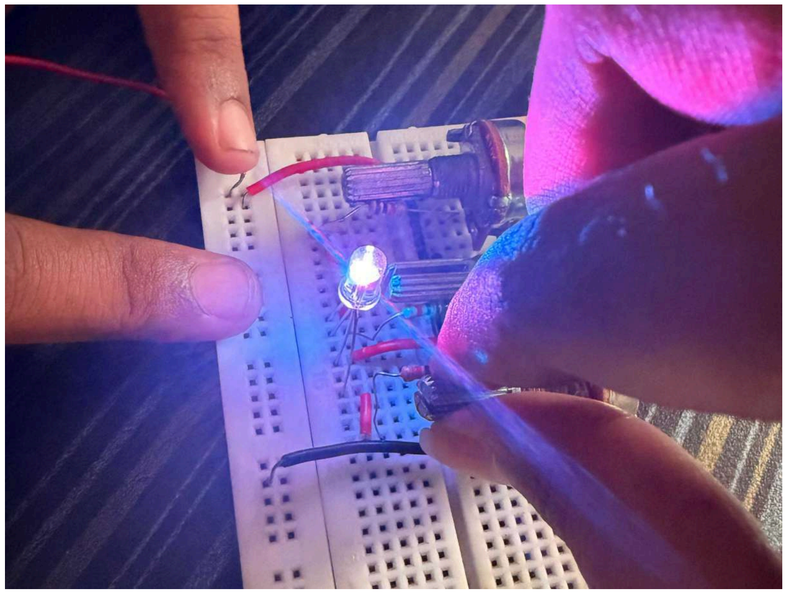
Controlling RGB LED with 3 Potentiometers

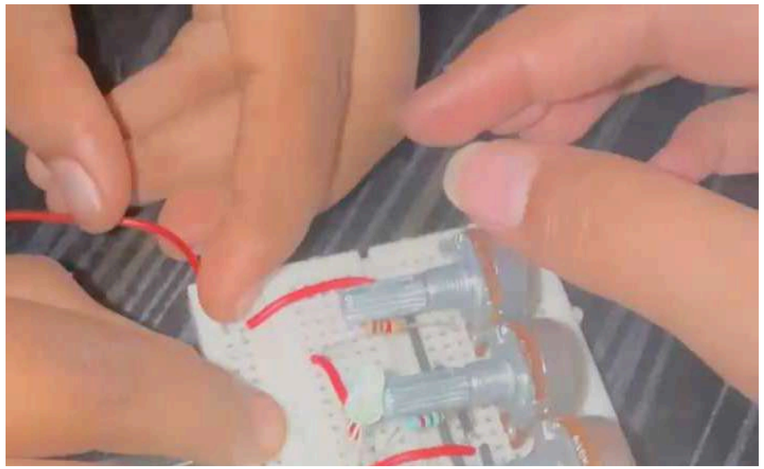
How it works

1. 9V Battery + connector
2. RGB LED (common cathode)
3. 3 × Potentiometers – 10 kΩ
4. 3 × Resistors – 330 Ω (current limiting, one for R/G/B each)
5. Breadboard + jumper wires

Learnings

- Potentiometers act as manual dimmers
- Current-limiting resistors protect LED
- Common ground is essential
- Forward voltage differs (Red ~2V, Green/Blue ~3V) → brightness varies
- Note: Do not turn potentiometers fully to maximum — may overdrive the LED and fuse it





ASSIGNMENT 10

Lighting an LED with Arduino

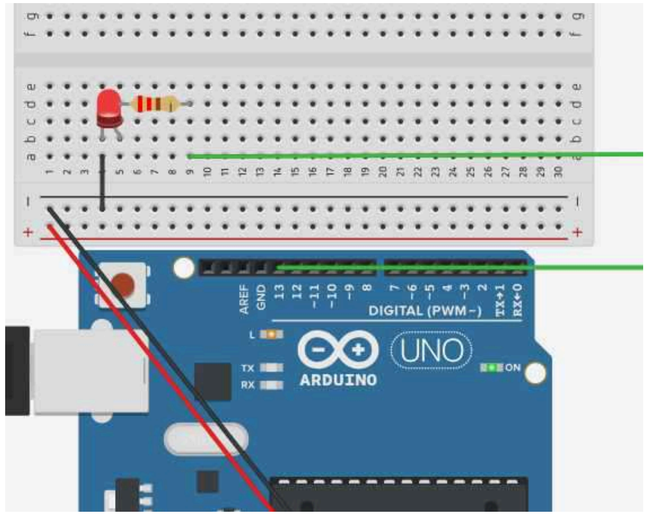
Components

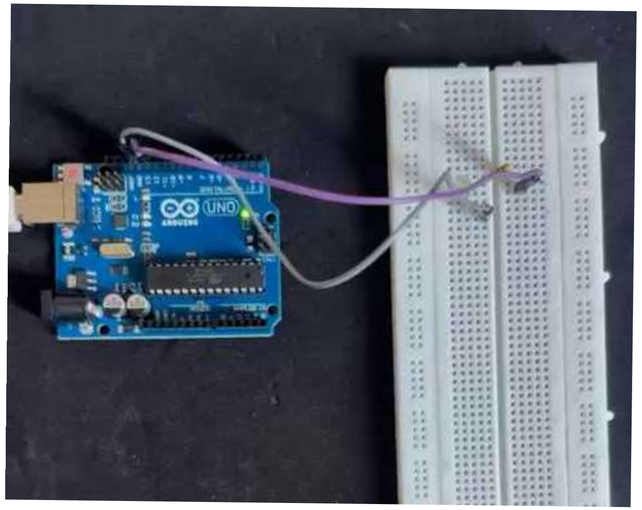
- Arduino UNO
- 1 × LED (any color)
- 1 × Resistor — 220 Ω (current limiting)
- Breadboard + jumper wires

Circuit

- LED long leg (anode) → Arduino digital pin (e.g., Pin 13) through 220 Ω resistor
- LED short leg (cathode) → Arduino GND

IN TINKERCAD





ASSIGNMENT 10

Lighting an LED with Arduino

How it works

- Arduino pin sends a HIGH signal (5V) → current flows through the resistor → LED lights up.
- When pin goes LOW (0V) → no current flows → LED turns off.
- The resistor (220 Ω) limits current (prevents LED damage).
- The delay(1000) makes the LED stay ON or OFF for 1 second → creating a blink.

Learnings

- Always use a resistor to protect LED from high current
- Arduino digitalWrite(HIGH) = LED ON, digitalWrite(LOW) = LED OFF
- Timing is controlled using delay(ms)

Code

```cpp
int ledPin = 13;  // LED connected to pin 13

void setup() {
  pinMode(ledPin, OUTPUT);   // set pin as output
}

void loop() {
  digitalWrite(ledPin, HIGH); // turn LED ON
  delay(1000);                // wait 1 second
  digitalWrite(ledPin, LOW);  // turn LED OFF
  delay(1000);                // wait 1 second
}
```

ASSIGNMENT 11

Arduino with 2 LEDs (Blink)

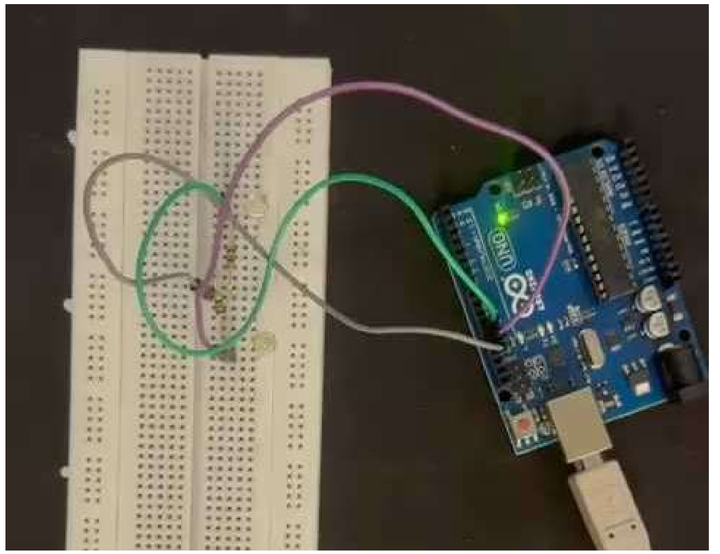
A simple Arduino project where two LEDs are connected to digital pins and programmed to blink alternately, creating a basic light pattern.

Components

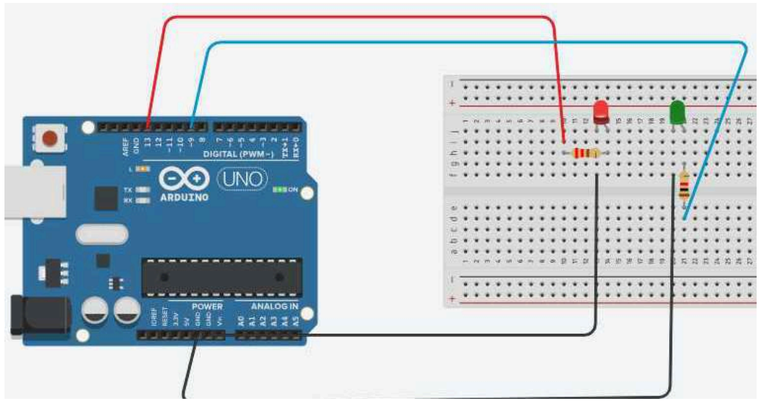
Arduino UNO (or similar)
2 × LEDs (any color)
2 × Resistors — 220 Ω each
Breadboard + jumper wires

Circuit

- Powering Arduino
  - Connect Arduino UNO via USB cable to a laptop/PC (for power + code upload)
  - Or use a 5V USB adapter as external power supply
- LED 1
  - Long leg (anode) → Arduino digital pin 8 through a 220 Ω resistor
  - Short leg (cathode) → Arduino GND
- LED 2
  - Long leg (anode) → Arduino digital pin 9 through a 220 Ω resistor
  - Short leg (cathode) → Arduino GND

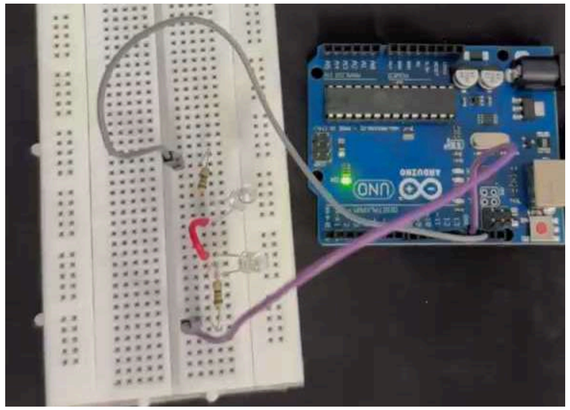


IN TINKERCAD



ASSIGNMENT 11

Arduino with 2 LEDs (Blink)



How it Works

- Arduino sends HIGH (5V) signal to one LED pin → LED lights up.
- The other LED pin stays LOW (0V) → LED stays OFF.
- After 1 second (delay(1000)), the signals swap → LEDs blink alternately.
- Arduino is powered through the USB cable, which also allows uploading the code.

Learnings

- delay(ms) controls timing between blinks.
- A common ground is essential for proper circuit operation.
- USB cable provides both power and code uploading.

Code

```cpp
int led1 = 8;
int led2 = 9;

void setup() {
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
}

void loop() {
  digitalWrite(led1, HIGH);  // LED 1 ON
  digitalWrite(led2, LOW);   // LED 2 OFF
  delay(1000);

  digitalWrite(led1, LOW);   // LED 1 OFF
  digitalWrite(led2, HIGH);  // LED 2 ON
  delay(1000);
}
```

ASSIGNMENT 12

Arduino LED with Push Button

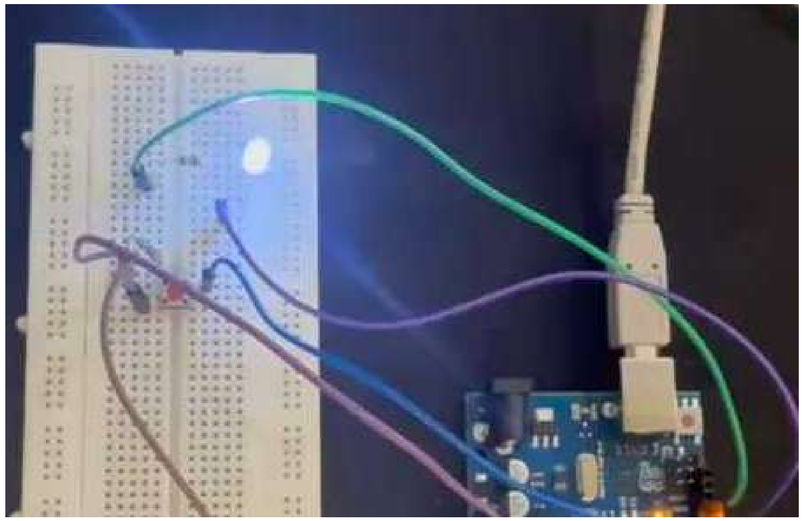
Control an LED using a push button — press to turn it ON, release to turn it OFF.

Components

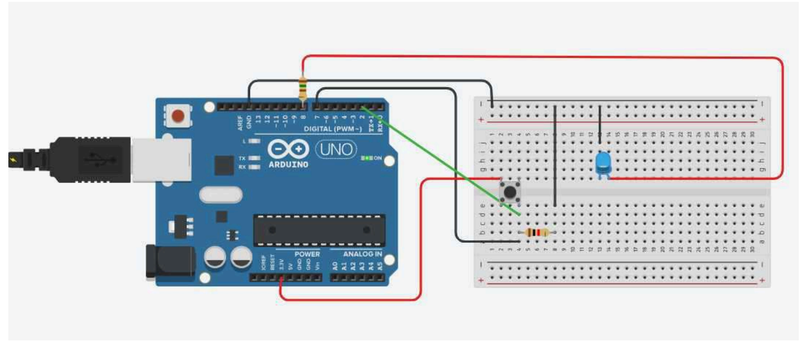
- Arduino UNO (or any board)
- 1 × LED (any color)
- 1 × Push Button
- 1 × Resistor — 220 Ω (for LED)
- 1 × Resistor — 10 kΩ (for button pull-down)
- Breadboard + jumper wires

Circuit

- LED: long leg (anode) → Arduino digital pin 8 through 220 Ω resistor
  short leg (cathode) → Arduino GND
- Push Button:
  - One side → Arduino digital pin 7
  - Other side → GND
  - 10 kΩ resistor → connects button pin to GND (pull-down)

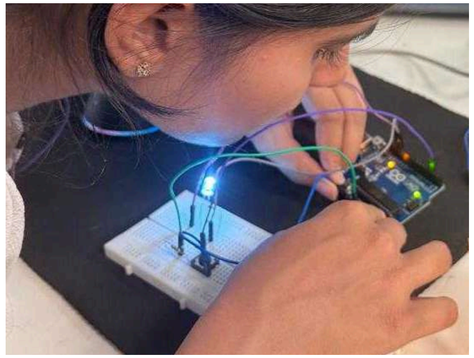


IN TINKERCAD



ASSIGNMENT 12

Arduino LED with Push Button



How it Works

- Arduino reads the button state (HIGH when pressed, LOW when not).
- When button is pressed, pin goes HIGH → LED turns ON.
- When button is released, pin goes LOW → LED turns OFF.
- Resistors: 220 Ω limits current to LED, 10 kΩ ensures button pin reads LOW when not pressed.

Code

```cpp

int ledPin = 8;     // LED connected to pin 8
int buttonPin = 7;  // Button connected to pin 7
int buttonState = 0;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  buttonState = digitalRead(buttonPin);  // read button state

  if (buttonState == HIGH) {             // if button pressed
    digitalWrite(ledPin, HIGH);          // turn LED ON
  } else {
    digitalWrite(ledPin, LOW);           // turn LED OFF
  }
}
```

Learnings

- digitalRead(pin) checks the state of a button or sensor.
- Conditional statements (if) control devices based on input.

ASSIGNMENT 13

Arduino LED with Potentiometer

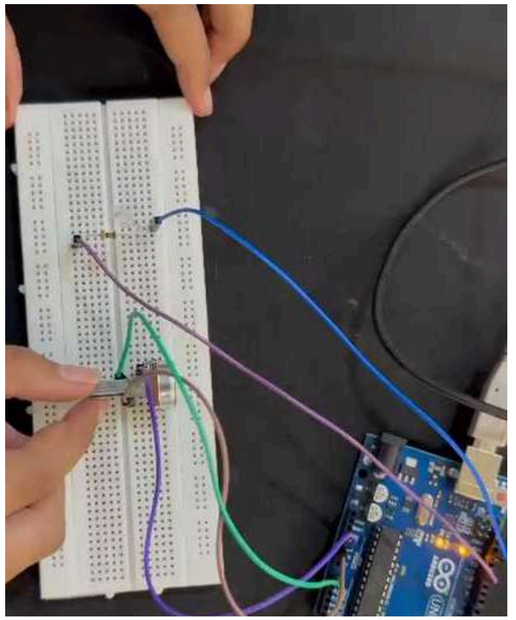
Control the brightness of an LED using a potentiometer — turn the knob to dim or brighten the LED.

Components

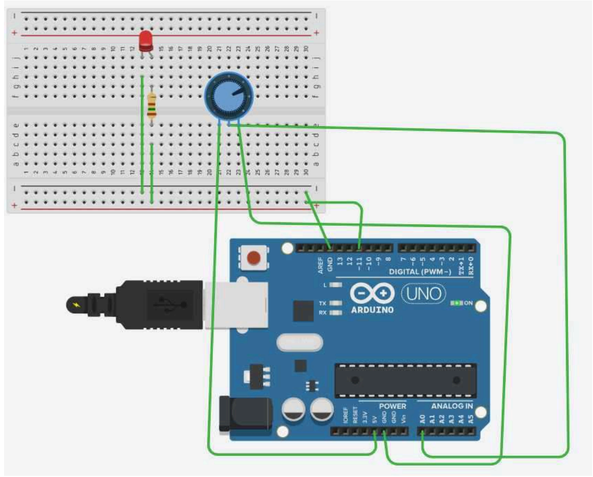
- Arduino UNO (or any board)
- 1 × LED (any color)
- 1 × Potentiometer (10 kΩ recommended)
- 1 × Resistor — 220 Ω (for LED)
- Breadboard + jumper wires

Circuit

- LED: long leg (anode) → Arduino digital pin 9 through 220 Ω resistor
  short leg (cathode) → Arduino GND
- Potentiometer:
  - Middle pin → Arduino A0 (analog input)
  - One side → 5V
  - Other side → GND



IN TINKERCAD



ASSIGNMENT 13

Arduino LED with Potentiometer

How it Works

- Arduino reads analog value from the potentiometer (0–1023).
- map() converts this range to PWM range (0–255).
- analogWrite() sends PWM signal to LED → controls brightness.
- Turning the potentiometer changes resistance → changes voltage → LED brightness varies smoothly.

Code

```cpp
int ledPin = 9;      // LED connected to PWM pin 9
int potPin = A0;     // Potentiometer connected to analog pin A0
int potValue = 0;
int ledBrightness = 0;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  potValue = analogRead(potPin);                // read potentiometer (0-1023)
  ledBrightness = map(potValue, 0, 1023, 0, 255); // map to PWM range (0-255)
  analogWrite(ledPin, ledBrightness);           // set LED brightness
}
```

Learnings

- analogWrite(pin, value) controls brightness using PWM.
- Potentiometer acts as a variable resistor to adjust voltage input.

ASSIGNMENT 14

Arduino RGB LED Control

Control the color of an RGB LED by adjusting the intensity of its Red, Green, and Blue channels using Arduino and PWM.

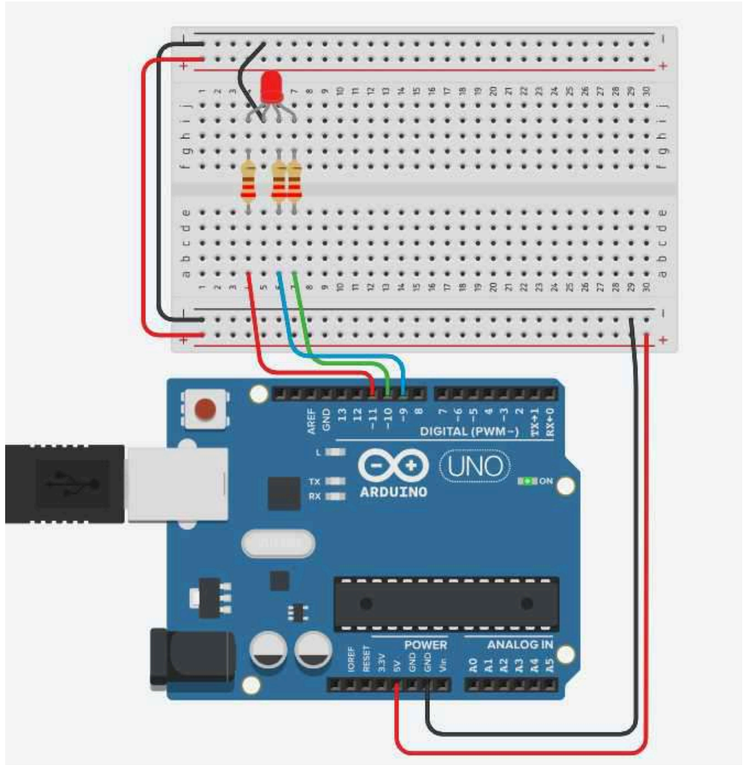
Components

- Arduino UNO (or any board)
- 1 × Common Cathode RGB LED
- 3 × Resistors — 220 Ω each
- Breadboard + jumper wires

Circuit

- RGB LED:
  - Red Anode → Arduino digital pin 9 through 220 Ω resistor
  - Green Anode → Arduino digital pin 10 through 220 Ω resistor
  - Blue Anode → Arduino digital pin 11 through 220 Ω resistor
  - Common Cathode → Arduino GND

IN TINKERCAD



ASSIGNMENT 14

Arduino RGB LED Control

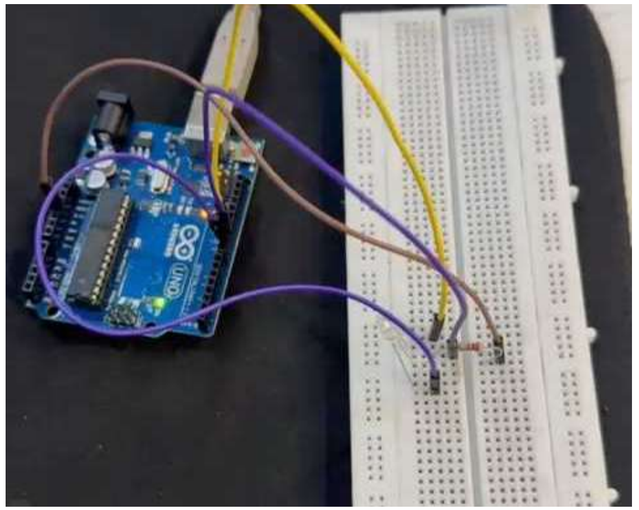
How it Works

- RGB LED has 3 channels → Red, Green, Blue.
- analogWrite(pin, value) uses PWM to set brightness (0–255) of each channel.
- Combining different intensities produces different colors.
- Delay between color changes allows visible transition effect.

Learnings

- PWM allows smooth control of LED brightness.
- Mixing RGB channels can create millions of colors.
- Can be extended to sensor-based color changes or BLE/Mobile app control.
- Useful for decorative lighting, mood lamps, and DIY projects.

```
// C++ code
//
void setup()
{
  pinMode(11, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(9, OUTPUT);
}

void loop()
{
  analogWrite(11, 255);
  analogWrite(10, 0);
  analogWrite(9, 0);
  delay(1000); // Wait for 1000 millisecond(s)
  analogWrite(11, 255);
  analogWrite(10, 255);
  analogWrite(9, 102);
  delay(1000); // Wait for 1000 millisecond(s)
}
```



ASSIGNMENT 15

Arduino Ultrasonic Sensor with LCD

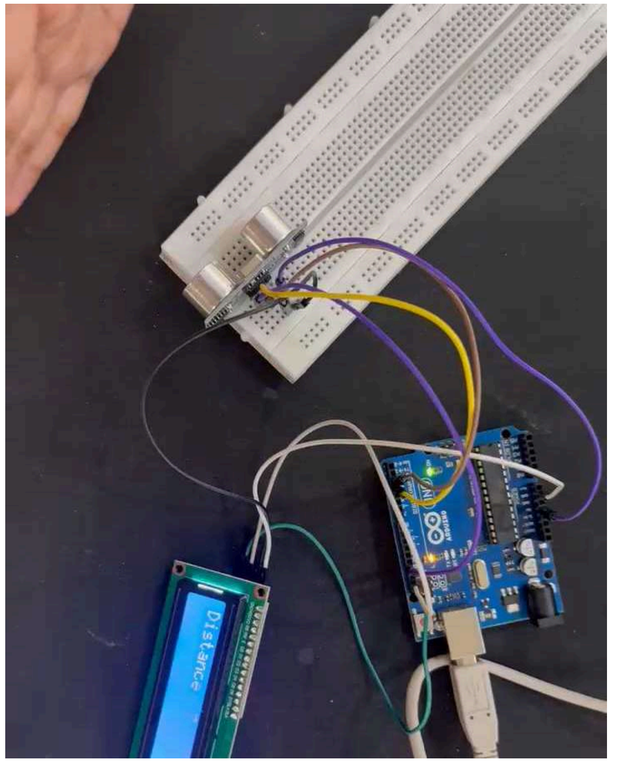
Control the brightness of an LED using a potentiometer — turn the knob to dim or brighten the LED.

Components

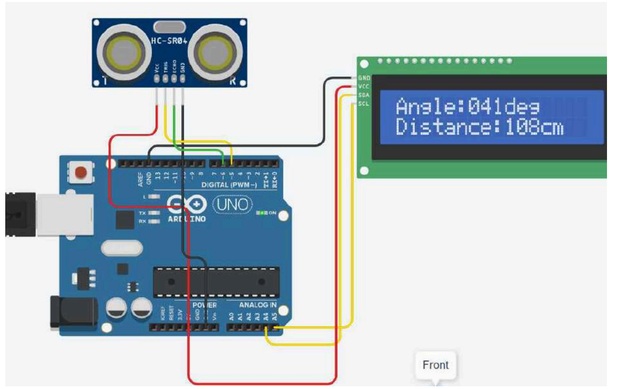
- Arduino UNO (or any board)
- 1 × Ultrasonic Sensor (HC-SR04)
- 1 × 16×2 LCD (with I2C module recommended)
- Breadboard + jumper wires

Circuit

- Ultrasonic Sensor HC-SR04:
  - VCC → Arduino 5V
  - GND → Arduino GND
  - TRIG → Arduino digital pin 9
  - ECHO → Arduino digital pin 10
- 16×2 LCD with I2C:
  - GND → Arduino GND
  - VCC → Arduino 5V
  - SDA → Arduino A4
  - SCL → Arduino A5



IN TINKERCAD



ASSIGNMENT 15

Arduino Ultrasonic Sensor with LCD

How it Works

- Ultrasonic sensor sends a pulse → reflects from an object → echo received.
- Arduino measures time between sending and receiving → calculates distance.
- Distance is displayed on LCD in centimeters.
- pulseIn() measures the duration of the pulse, and the formula distance = duration * 0.034 / 2 converts it to cm.

Learnings

- pulseIn(pin, HIGH/LOW) measures pulse duration.
- LCD with I2C simplifies wiring (SDA/SCL).
- Ultrasonic sensor can measure distance from ~2 cm to ~400 cm.
- Arduino can process sensor data and show real-time output on LCD.

CODE

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2); // I2C address 0x27, 16x2 LCD

const int trigPin = 9;
const int echoPin = 10;
long duration;
int distance;

void setup() {
  lcd.init();
  lcd.backlight();
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  // Send ultrasonic pulse
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read echo
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;  // Convert to cm

  // Display on LCD
  lcd.setCursor(0, 0);
  lcd.print("Distance:");
  lcd.setCursor(0, 1);
  lcd.print(distance);
  lcd.print(" cm");

  delay(500);
}
```

ASSIGNMENT 16

Arduino Ultrasonic Sensor with LCD and Servo Motor

Measure distance using an ultrasonic sensor, display it on an LCD, and control a servo motor based on the measured distance.

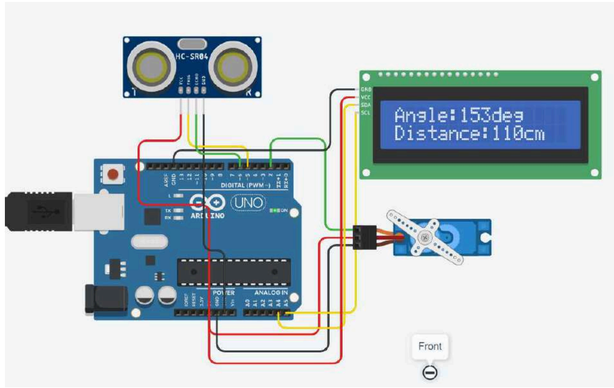
Components

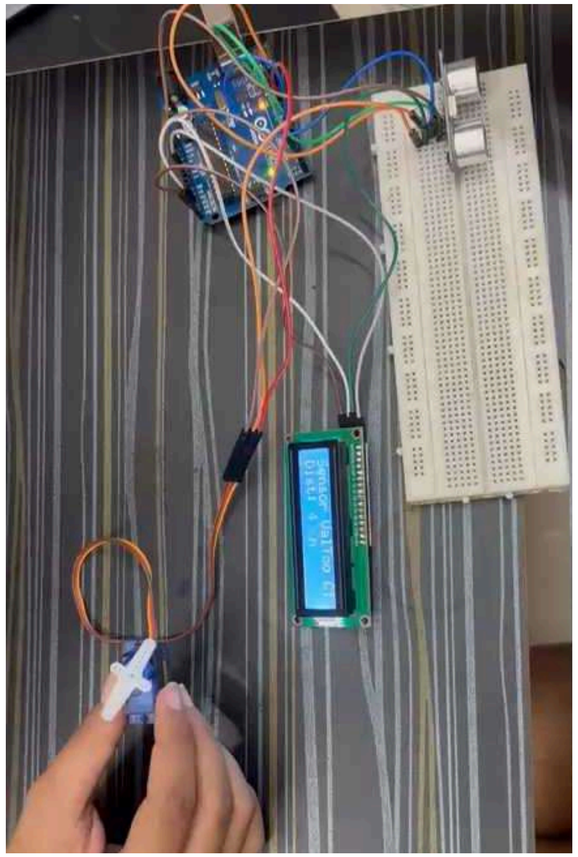
- Arduino UNO (or any board)
- 1 × Ultrasonic Sensor (HC-SR04)
- 1 × 16×2 LCD (with I2C module recommended)
- 1 × Servo Motor (SG90 or similar)
- Breadboard + jumper wires

Circuit

- Ultrasonic Sensor HC-SR04:
  - VCC → Arduino 5V
  - GND → Arduino GND
  - TRIG → Arduino digital pin 9
  - ECHO → Arduino digital pin 10
- 16×2 LCD with I2C:
  - GND → Arduino GND
  - VCC → Arduino 5V
  - SDA → Arduino A4
  - SCL → Arduino A5
- Servo Motor:
  - Signal → Arduino digital pin 6
  - VCC → Arduino 5V
  - GND → Arduino GND

IN TINKERCAD





ASSIGNMENT 16

Arduino Ultrasonic Sensor with LCD and Servo Motor

How it works

- Ultrasonic sensor measures distance by sending a pulse and detecting its echo.
- Arduino calculates distance using pulseIn() and displays it on LCD.
- Distance is mapped to a servo angle (0–180°) → closer object → smaller angle, farther object → larger angle.
- map() converts distance range to servo rotation range, constrain() ensures safe limits.

learnings

- Servo.write(angle) sets servo position based on calculation.
- Ultrasonic + LCD + Servo demonstrates real-time feedback control system.
- Useful for projects like automatic doors, obstacle indicators, or robotic arms.

CODE

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Servo.h>

LiquidCrystal_I2C lcd(0x27, 16, 2); // I2C LCD
Servo myServo;

const int trigPin = 9;
const int echoPin = 10;
const int servoPin = 8;
long duration;
int distance;

void setup() {
  lcd.init();
  lcd.backlight();
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  myServo.attach(servoPin);
}

void loop() {
  // Send ultrasonic pulse
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read echo
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.034 / 2;  // Convert to cm

  // Display on LCD
  lcd.setCursor(0, 0);
  lcd.print("Distance:");
  lcd.setCursor(0, 1);
  lcd.print(distance);
  lcd.print(" cm   ");

  // Control servo based on distance
  int angle = map(distance, 0, 100, 0, 180); // map distance 0-100cm to 0-180°
  angle = constrain(angle, 0, 180);         // limit angle
  myServo.write(angle);

  delay(200);
}
```

ASSIGNMENT 17

Arduino Piezo Buzzer with IR Sensor

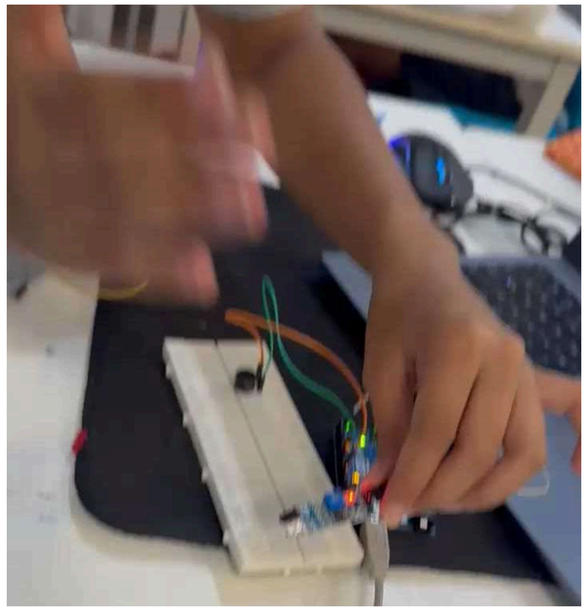
Detect objects using an IR sensor and trigger a piezo buzzer to sound an alert when an object is detected.

Components

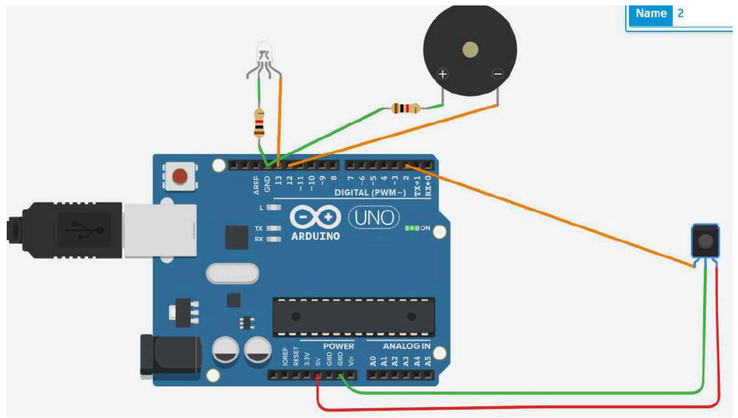
- Arduino UNO (or any board)
- 1 × IR Sensor Module
- 1 × Piezo Buzzer (Active or Passive)
- Breadboard + jumper wires

Circuit

- IR Sensor:
  - VCC → Arduino 5V
  - GND → Arduino GND
  - OUT → Arduino digital pin 7
- Piezo Buzzer:
  - Positive → Arduino digital pin 8
  - Negative → Arduino GND



IN TINKERCAD



ASSIGNMENT 17

Arduino Piezo Buzzer with IR Sensor

How it Works

- IR sensor detects presence of an object → output goes HIGH.
- Arduino reads sensor signal using digitalRead().
- If object is detected → buzzer sounds → alert.
- If no object → buzzer stays OFF.

Learnings

- digitalRead(pin) checks sensor output.
- digitalWrite(pin, HIGH/LOW) controls buzzer ON/OFF.
- IR sensor can detect obstacles or motion in its path.

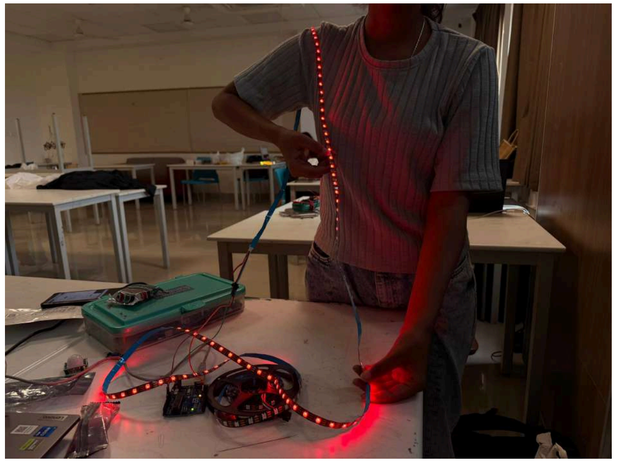
CODE

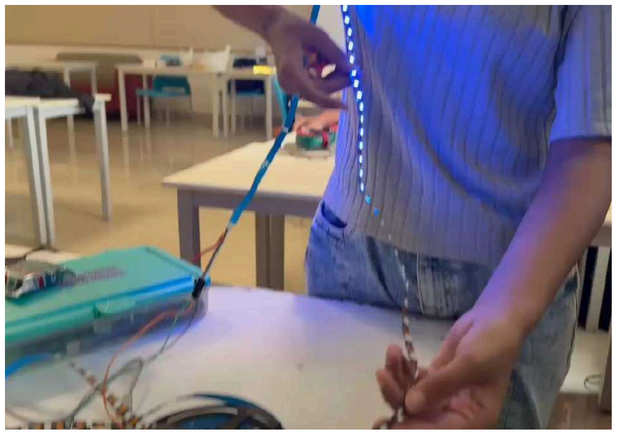
```
int irPin = 7;       // IR sensor output pin
int buzzerPin = 8;   // Buzzer pin
int irState = 0;

void setup() {
  pinMode(irPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  irState = digitalRead(irPin);  // read IR sensor state

  if (irState == HIGH) {         // object detected
    digitalWrite(buzzerPin, HIGH); // turn buzzer ON
  } else {
    digitalWrite(buzzerPin, LOW);  // turn buzzer OFF
  }
}
```

ASSIGNMENT 18

Arduino NeoPixel (WS2812) LED

Control individual RGB LEDs (NeoPixels) to create colors, patterns, and effects using Arduino.





Components

- Arduino UNO (or any board)
- 1 × NeoPixel Strip or Ring (WS2812)
- 1 × 470 Ω resistor (recommended for data line)
- 1 × 1000 µF capacitor (for power stabilization)
- Breadboard + jumper wires
- External 5V power supply (if using many LEDs)

Circuit

- NeoPixel Data In → Arduino digital pin 6 through 470 Ω resistor
- NeoPixel VCC → Arduino 5V (or external 5V supply for long strips)
- NeoPixel GND → Arduino GND
- Add 1000 µF capacitor across VCC and GND of NeoPixel to prevent power spikes

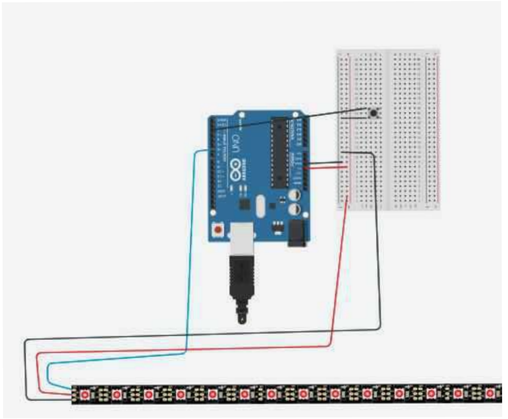
ASSIGNMENT 18

Arduino NeoPixel (WS2812) LED

How it Works

- Ultrasonic sensor sends a pulse → reflects from an object → echo received.
- Arduino measures time between sending and receiving → calculates distance.
- Distance is displayed on LCD in centimeters.
- pulseIn() measures the duration of the pulse, and the formula distance = duration * 0.034 / 2 converts it to cm.

IN TINKERCAD



CODE

```
#include <Adafruit_NeoPixel.h>

#define PIN 6       // Data pin connected to NeoPixel
#define NUM 8       // Number of LEDs

Adafruit_NeoPixel strip(NUM, PIN, NEO_GRB + NEO_KHZ800);

void setup() {
  strip.begin();
  strip.show();  // Initialize all pixels to 'off'
}

void loop() {
  for(int i=0; i<NUM; i++){
    strip.setPixelColor(i, strip.Color(255, 0, 0)); // Red color
    strip.show();
    delay(200);
  }

  for(int i=0; i<NUM; i++){
    strip.setPixelColor(i, strip.Color(0, 255, 0)); // Green color
    strip.show();
    delay(200);
  }

  for(int i=0; i<NUM; i++){
    strip.setPixelColor(i, strip.Color(0, 0, 255)); // Blue color
    strip.show();
    delay(200);
  }
}
```

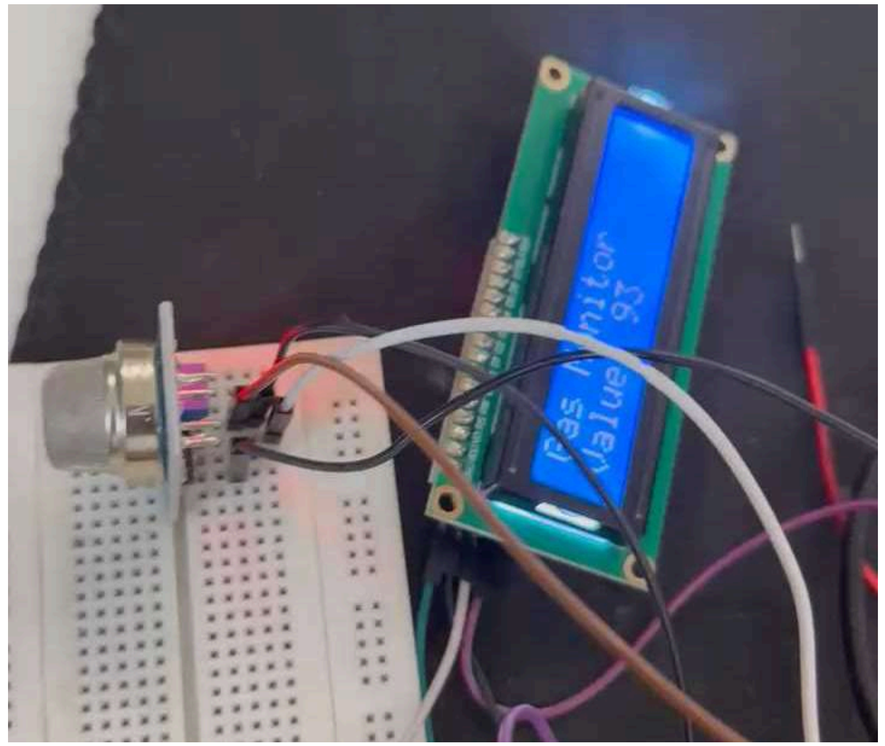
Learnings

- pulseIn(pin, HIGH/LOW) measures pulse duration.
- LCD with I2C simplifies wiring (SDA/SCL).
- Ultrasonic sensor can measure distance from ~2 cm to ~400 cm.
- Arduino can process sensor data and show real-time output on LCD.

ASSIGNMENT 19

Arduino Gas Sensor with LCD

Detect gas levels using a gas sensor (e.g., MQ-2) and display the readings on an LCD in real-time.



Components

- Arduino UNO (or any board)
- 1 × Gas Sensor (MQ-2 or similar)
- 1 × 16×2 LCD (with I2C module recommended)
- Breadboard + jumper wires

Circuit

- Gas Sensor (MQ-2):
  - VCC → Arduino 5V
  - GND → Arduino GND
  - AO (analog output) → Arduino A0
- 16×2 LCD with I2C:
  - GND → Arduino GND
  - VCC → Arduino 5V
  - SDA → Arduino A4
  - SCL → Arduino A5

ASSIGNMENT 19

Arduino Gas Sensor with LCD

How it Works

- Gas sensor outputs an analog voltage proportional to gas concentration.
- Arduino reads the analog voltage using analogRead() → value 0–1023.
- Value is displayed on LCD in real-time.
- Higher readings indicate higher gas concentration.

Learnings

- analogRead(pin) reads sensor values continuously.
- LCD with I2C simplifies wiring (SDA/SCL).
- MQ-2 can detect LPG, smoke, methane, propane, etc.

CODE

```cpp
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2); // I2C LCD

const int gasPin = A0; // Gas sensor analog output
int gasValue = 0;

void setup() {
  lcd.init();
  lcd.backlight();
}

void loop() {
  gasValue = analogRead(gasPin);  // Read gas sensor value

  lcd.setCursor(0, 0);
  lcd.print("Gas Level:");
  lcd.setCursor(0, 1);
  lcd.print(gasValue);
  lcd.print("      "); // Clear remaining chars

  delay(500);
}
```

ASSIGNMENT 20

Arduino Gas Sensor with LCD and Piezo Buzzer

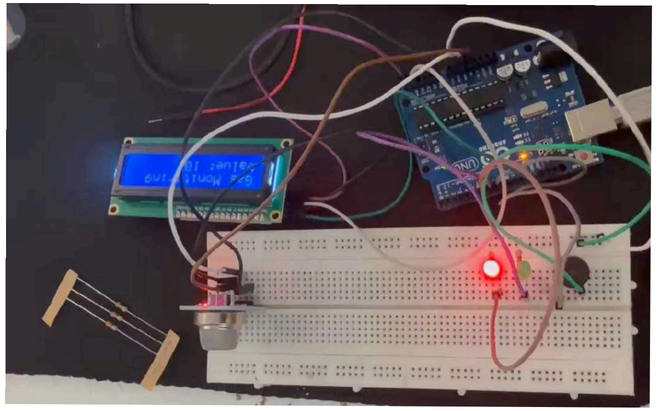
Detect gas levels using a gas sensor, display the readings on an LCD, and sound a buzzer alert when gas concentration exceeds a threshold.

Components

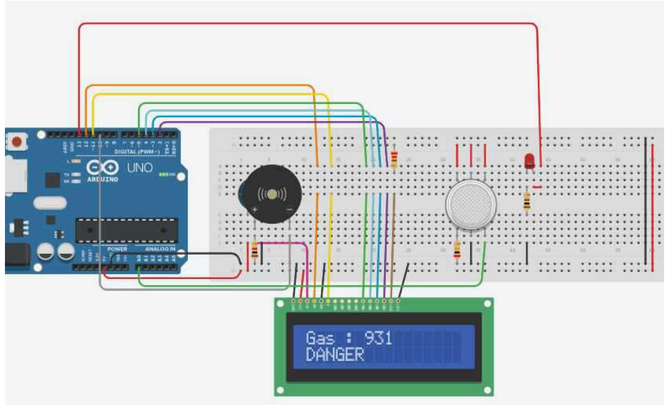
- Arduino UNO (or any board)
- 1 × Gas Sensor (MQ-2 or similar)
- 1 × 16×2 LCD (with I2C module recommended)
- 1 × Piezo Buzzer (Active or Passive)
- Breadboard + jumper wires

Circuit

- Gas Sensor (MQ-2):
  - VCC → Arduino 5V
  - GND → Arduino GND
  - AO (analog output) → Arduino A0
- 16×2 LCD with I2C:
  - GND → Arduino GND
  - VCC → Arduino 5V
  - SDA → Arduino A4
  - SCL → Arduino A5
- Piezo Buzzer:
  - Positive → Arduino digital pin 8
  - Negative → Arduino GND



IN TINKERCAD



ASSIGNMENT 20

Arduino Gas Sensor with LCD and Piezo Buzzer

How it Works

- Gas sensor outputs an analog voltage proportional to gas concentration.
- Arduino reads value using analogRead() → displayed on LCD.
- If gas level exceeds threshold, buzzer sounds → alert.
- Allows real-time monitoring and alarm for unsafe gas levels.

Learnings

- analogRead() monitors gas level continuously.
- Piezo buzzer provides audible warning for high gas levels.
- LCD gives real-time visual feedback for monitoring.

CODE

```cpp
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,16,2); // I2C LCD
const int gasPin = A0;            // Gas sensor analog output
const int buzzerPin = 8;          // Buzzer pin
int gasValue = 0;
int threshold = 300;              // Gas level threshold for alert

void setup() {
  lcd.init();
  lcd.backlight();
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  gasValue = analogRead(gasPin);  // Read gas sensor value

  // Display on LCD
  lcd.setCursor(0,0);
  lcd.print("Gas Level:");
  lcd.setCursor(0,1);
  lcd.print(gasValue);
  lcd.print("      "); // clear remaining chars

  // Buzzer alert
  if(gasValue > threshold){
    digitalWrite(buzzerPin, HIGH); // Turn buzzer ON
  } else {
    digitalWrite(buzzerPin, LOW);  // Turn buzzer OFF
  }

  delay(500);
}
```

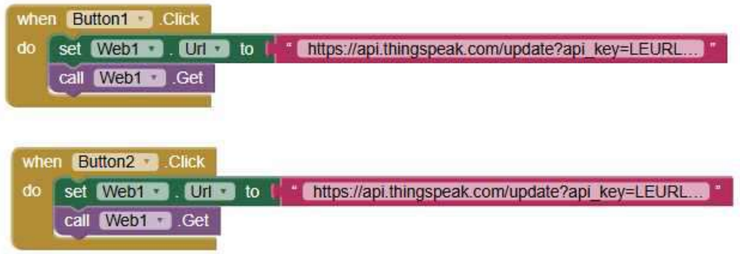
ASSIGNMENT 21

Controlling an LED with BLE on ESP32

Turn an LED ON or OFF wirelessly using a smartphone via Bluetooth Low Energy (BLE) and ESP32.

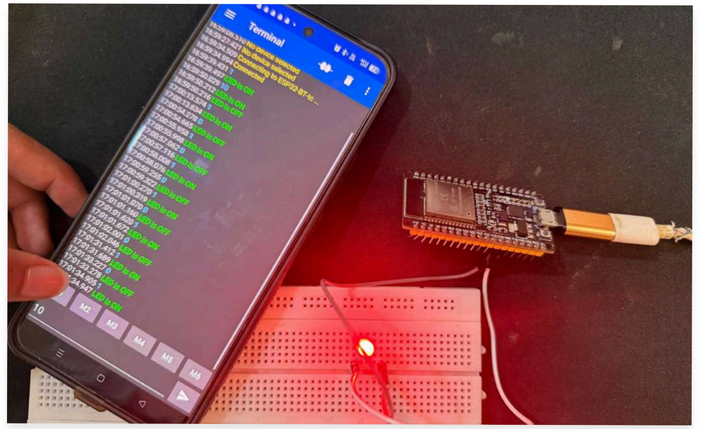
Components

- ESP32 Development Board
- 1 × LED (any color)
- 1 × Resistor — 220 Ω (for LED)
- Breadboard + jumper wires
- Smartphone with BLE app (e.g., Bluetooth terminal app)



Circuit

- Long leg (anode) → ESP32 digital pin 2 through 220 Ω resistor
- Short leg (cathode) → ESP32 GND



ASSIGNMENT 21

Controlling an LED with BLE on ESP32

How it Works

- ESP32 acts as a BLE server, advertising a service for LED control.
- Smartphone app connects to ESP32 → sends ON/OFF commands.
- BLE characteristic writes trigger callbacks → turn LED ON or OFF.
- LED responds in real-time to wireless commands.

Learnings

- BLE allows wireless control of devices using low power.
- BLECharacteristicCallbacks let you handle commands from smartphone.
- ESP32 integrates microcontroller + BLE, no extra modules needed.
- Useful for IoT home automation, remote switching, and smart lighting.

CODE

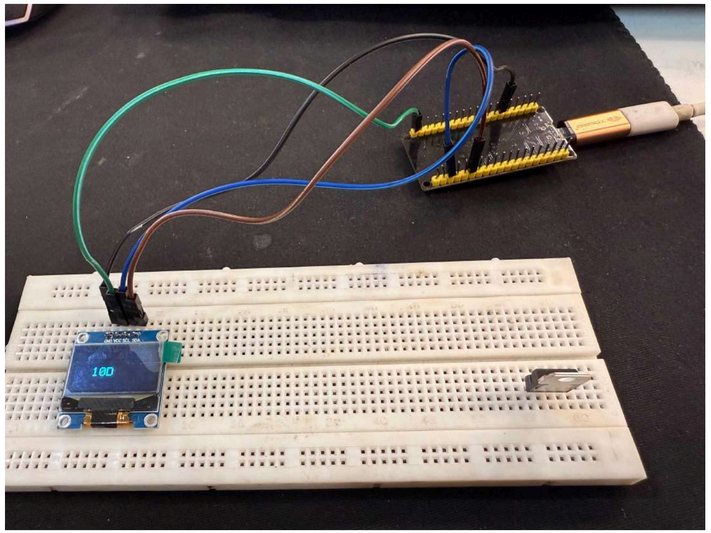
```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

#define LED_PIN 2

BLEServer* pServer = NULL;
BLECharacteristic* pCharacteristic = NULL;
bool deviceConnected = false;

class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
  };
  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
  }
};

class MyCallbacks: public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *pCharacteristic) {
    std::string value = pCharacteristic->getValue();
    if(value == "ON") {
      digitalWrite(LED_PIN, HIGH);
    } else if(value == "OFF") {
      digitalWrite(LED_PIN, LOW);
    }
  }
};
```

```
void setup() {
  pinMode(LED_PIN, OUTPUT);
  BLEDevice::init("ESP32_LED"); // Device name
  pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  BLEService *pService = pServer->createService(BLEUUID((uint16_t)0xFFE0));
  pCharacteristic = pService->createCharacteristic(
                      BLEUUID((uint16_t)0xFFE1),
                      BLECharacteristic::PROPERTY_WRITE
                    );
  pCharacteristic->setCallbacks(new MyCallbacks());
  pCharacteristic->addDescriptor(new BLE2902());
  pService->start();
  pServer->getAdvertising()->start();
}

void loop() {
  // Nothing needed here, BLE callbacks handle LED control
}
```

ASSIGNMENT 22

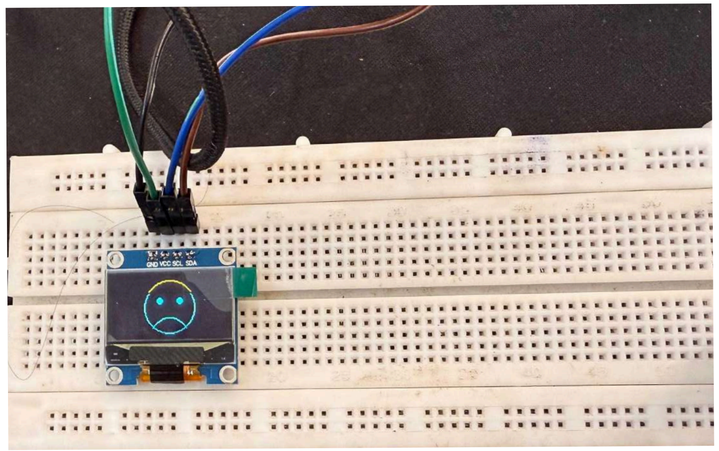
ESP32 with OLED Display

Display text, numbers, or sensor data on a small OLED screen using ESP32 for real-time visualization..



Components

- ESP32 Development Board
- 1 × OLED Display (0.96", 128×64, I2C)
- Breadboard + jumper wires



Circuit

- OLED Display (I2C):
  - VCC → ESP32 3.3V or 5V (depending on OLED)
  - GND → ESP32 GND
  - SDA → ESP32 GPIO 21
  - SCL → ESP32 GPIO 22

ASSIGNMENT 22

ESP32 with OLED Display

How it Works

- OLED communicates via I2C using SDA and SCL pins.
- Adafruit_SSD1306 library handles low-level OLED commands.
- display.println() prints text, display.display() updates the screen.
- Can be used to show sensor readings, time, or status messages in real-time.

Learnings

- ESP32 + OLED is ideal for compact displays in IoT projects.
- I2C allows two-wire communication, freeing up pins.
- You can display dynamic data like counters, sensor values, or alerts.

CODE

```cpp

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // I2C address
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }

  display.clearDisplay();
  display.setTextSize(2);      // Text size
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.println("Hello ESP32!");
  display.display();
}

void loop() {
  // Example: Display counter
  for(int i=0; i<=10; i++){
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println("Count:");
    display.setCursor(0, 20);
    display.println(i);
    display.display();
    delay(1000);
  }
}
```

ASSIGNMENT 23

ESP32 with DHT Sensor, ThingSpeak, and MIT App Inventor

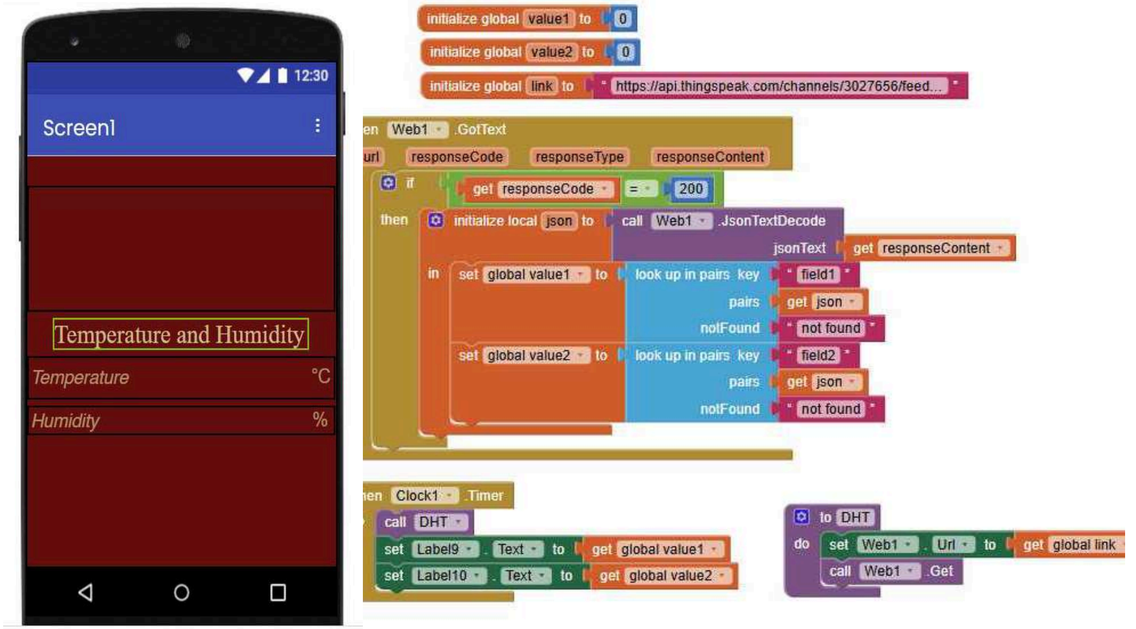
Measure temperature and humidity using a DHT sensor, send data to ThingSpeak for cloud logging, and display readings on a mobile app built with MIT App Inventor.

Components

- ESP32 Development Board
- 1 × DHT Sensor (DHT11 or DHT22)
- 1 × 10 kΩ Resistor (for DHT if required)
- Breadboard + jumper wires
- Wi-Fi network for ThingSpeak

Circuit

- DHT Sensor:
  - VCC → ESP32 3.3V or 5V (depending on sensor)
  - GND → ESP32 GND
  - Data → ESP32 GPIO 4 (example)
  - Optional: 10 kΩ pull-up resistor between VCC and Data



MIT APP INVENTOR

ASSIGNMENT 23

ESP32 with DHT Sensor, ThingSpeak, and MIT App Inventor

Arduino Code

- Use the code to read DHT sensor and send data to ThingSpeak using HTTP GET.
- Set SSID, password, and ThingSpeak Write API Key in the code.
- ThingSpeak allows a minimum 15-second interval between updates.

How to Set Up ThingSpeak

- Go to ThingSpeak and create an account.
- Create a new channel with fields for temperature and humidity.
- Copy the Write API Key → use it in Arduino code.
- Upload code to ESP32 → check ThingSpeak channel → values should update every 15s.

Code

```
#include <WiFi.h>
#include <HTTPClient.h>
#include "DHT.h"

#define DHTPIN 4
#define DHTTYPE DHT22

const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
String writeAPIKey = "YOUR_THINGSPEAK_WRITE_API_KEY";

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  dht.begin();
  WiFi.begin(ssid, password);

  Serial.print("Connecting to WiFi");
  while(WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected");
}

void loop() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

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

  if(WiFi.status() == WL_CONNECTED){
    HTTPClient http;
    String url = "http://api.thingspeak.com/update?api_key=" + writeAPIKey +
                 "&field1=" + String(temperature) + "&field2=" + String(humidity);
    http.begin(url);
    int httpResponseCode = http.GET();
    if(httpResponseCode > 0){
      Serial.println("Data sent to ThingSpeak");
    }
    else{
      Serial.println("Error sending data");
    }
    http.end();
  }

  delay(15000); // ThingSpeak free API allows 15s minimum interval
}
```

ASSIGNMENT 23

ESP32 with DHT Sensor, ThingSpeak, and MIT App Inventor

How to Set Up MIT App Inventor

- Go to MIT App Inventor and create a project.
- Add a Web component to the app.
- Set the URL to the ThingSpeak channel API:
- https://api.thingspeak.com/channels/YOUR_CHANNEL_ID/fields/1.json?api_key=YOUR_READ_API_KEY&results=1
- Use Label/Text components to display temperature and humidity.
- Use Clock component to fetch new data every 15–30 seconds.
- Test the app on your smartphone → live sensor data appears from ESP32 via ThingSpeak.

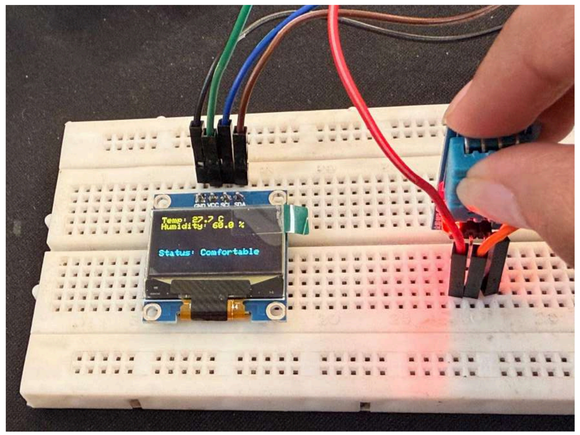
learnings

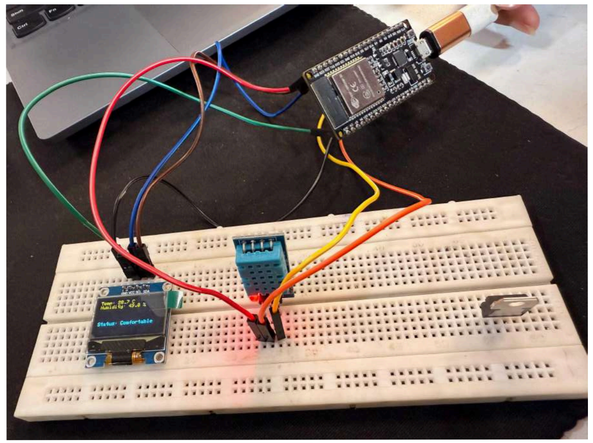
- ThingSpeak handles cloud storage & visualization.
- MIT App Inventor fetches ThingSpeak data → mobile dashboard.
- Ensure Wi-Fi is connected before uploading code.
- Adjust field URLs and update intervals according to your app design.
- Useful for IoT environmental monitoring, smart home projects, and learning cloud integration.

ASSIGNMENT 24

ESP32 with DHT Sensor and OLED Display

Measure temperature and humidity using a DHT sensor and display the readings in real-time on a small OLED screen with ESP32.





Components

- ESP32 Development Board
- DHT11 or DHT22 Sensor
- 0.96" OLED Display (128×64, I2C)
- 10 kΩ Resistor (optional, for DHT pull-up)
- Breadboard + jumper wires

Circuit

- DHT Sensor:
  - VCC → ESP32 3.3V/5V
  - GND → ESP32 GND
  - Data → ESP32 GPIO 4 (example)
  - Optional 10 kΩ pull-up resistor between VCC and Data
- OLED Display (I2C):
  - VCC → ESP32 3.3V or 5V
  - GND → ESP32 GND
  - SDA → ESP32 GPIO 21
  - SCL → ESP32 GPIO 22

ASSIGNMENT 24

ESP32 with DHT Sensor and OLED Display

How it Works

- DHT sensor measures temperature and humidity → sends digital data to ESP32.
- ESP32 reads values using dht.readTemperature() and dht.readHumidity().
- OLED communicates via I2C → displays the readings in real-time.
- Display updates every 2 seconds to show current environmental data.

Learnings

- ESP32 + OLED allows compact, real-time sensor monitoring.
- I2C simplifies wiring and frees up GPIO pins.
- Can be extended to log data, send to cloud, or trigger alerts.
- Useful for IoT weather monitoring, smart homes, and environmental projects.

Code

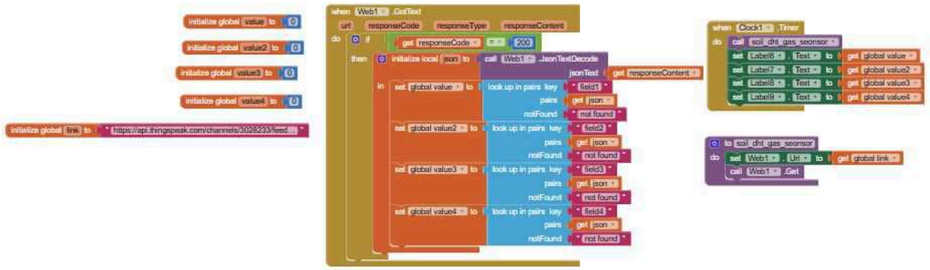
```cpp

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include "DHT.h"

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1

#define DHTPIN 4
#define DHTTYPE DHT22

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  dht.begin();

  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
}

void loop() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

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

  display.clearDisplay();
  display.setCursor(0,0);
  display.println("Temperature:");
  display.setCursor(0,10);
  display.print(temperature);
  display.println(" C");

  display.setCursor(0,30);
  display.println("Humidity:");
  display.setCursor(0,40);
  display.print(humidity);
  display.println(" %");

  display.display();
  delay(2000);
}
```

ASSIGNMENT 25

ESP32 with Soil Moisture, DHT, Gas Sensors and MIT App Inventor

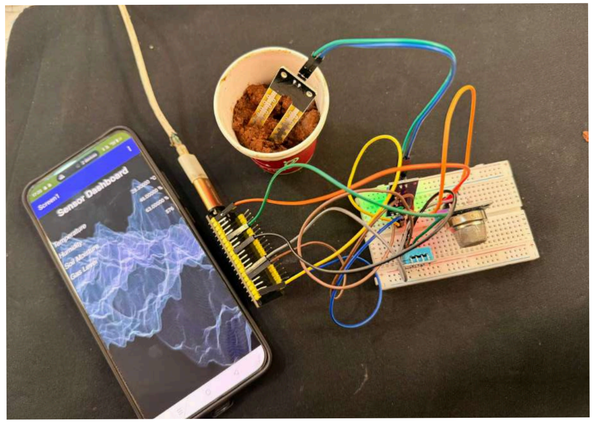
Monitor soil moisture, temperature, humidity, and gas levels using ESP32, and display the readings in real-time on a mobile app built with MIT App Inventor.

MIT APP INVENTOR



Components

- ESP32 Development Board
- 1 × DHT11 or DHT22 Sensor
- 1 × Soil Moisture Sensor (analog type)
- 1 × Gas Sensor (MQ-2)
- Breadboard + jumper wires
- Wi-Fi connection for MIT App Inventor (optional cloud logging)



Circuit

- DHT Sensor:
  - VCC → ESP32 3.3V/5V
  - GND → ESP32 GND
  - Data → ESP32 GPIO 4

ASSIGNMENT 25

ESP32 with Soil Moisture, DHT, Gas Sensors and MIT App Inventor

Circuit

- Soil Moisture Sensor:
  - VCC → ESP32 3.3V/5V
  - GND → ESP32 GND
  - Analog Output → ESP32 A0
- Gas Sensor (MQ-2):
  - VCC → ESP32 5V
  - GND → ESP32 GND
  - Analog Output → ESP32 A3

How to Set Up MIT App Inventor

- Open MIT App Inventor → create a project.
- Add Labels to display temperature, humidity, soil moisture, and gas levels.

CODE

```cpp
#include <WiFi.h>
#include <HTTPClient.h>
#include "DHT.h"

#define DHTPIN 4
#define DHTTYPE DHT22
#define SOILPIN 34
#define GASPIN 39

DHT dht(DHTPIN, DHTTYPE);

const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";

// Optional: your MIT App Inventor Web component URL
String appURL = "http://YOUR_APP_URL?temperature=";

void setup() {
  Serial.begin(115200);
  dht.begin();
  WiFi.begin(ssid, password);

  Serial.print("Connecting to WiFi");
  while(WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected");
}
```

```
void loop() {
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
  int soilMoisture = analogRead(SOILPIN);
  int gasLevel = analogRead(GASPIN);

  Serial.print("Temp: "); Serial.println(temperature);
  Serial.print("Humidity: "); Serial.println(humidity);
  Serial.print("Soil Moisture: "); Serial.println(soilMoisture);
  Serial.print("Gas: "); Serial.println(gasLevel);

  // Optional: Send to MIT App Inventor Web component
  if(WiFi.status() == WL_CONNECTED){
    HTTPClient http;
    String url = appURL + String(temperature) + "&humidity=" + String(humidity) +
                 "&soil=" + String(soilMoisture) + "&gas=" + String(gasLevel);
    http.begin(url);
    int code = http.GET();
    http.end();
  }

  delay(5000); // Send data every 5 seconds
}
```

ASSIGNMENT25

ESP32 with Soil Moisture, DHT, Gas Sensors and MIT App Inventor

How to Set Up MIT App Inventor

- Add a Web component → set URL to receive ESP32 data.
- Use Clock component to request data every few seconds.
- Parse web response → update Labels dynamically with sensor readings.
- Test app on your smartphone → live sensor data appears.

How to Set Up MIT App Inventor

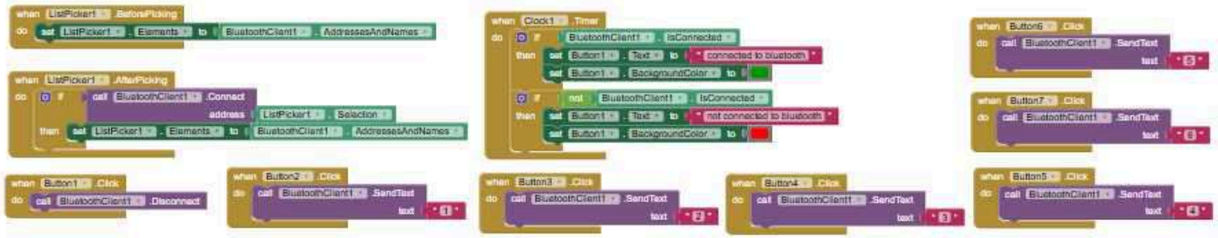
- ESP32 can read multiple analog and digital sensors simultaneously.
- Data can be sent to MIT App Inventor → mobile monitoring.
- Soil moisture + DHT + gas sensors allow environmental monitoring for smart agriculture or home projects.
- Combining sensors with Wi-Fi opens possibilities for IoT dashboards and alerts.

ASSIGNMENT 26

Controlling 3 LEDs with BLE on ESP32

Wirelessly control three LEDs independently using a smartphone via Bluetooth Low Energy (BLE) and ESP32.

MIT APP INVENTOR

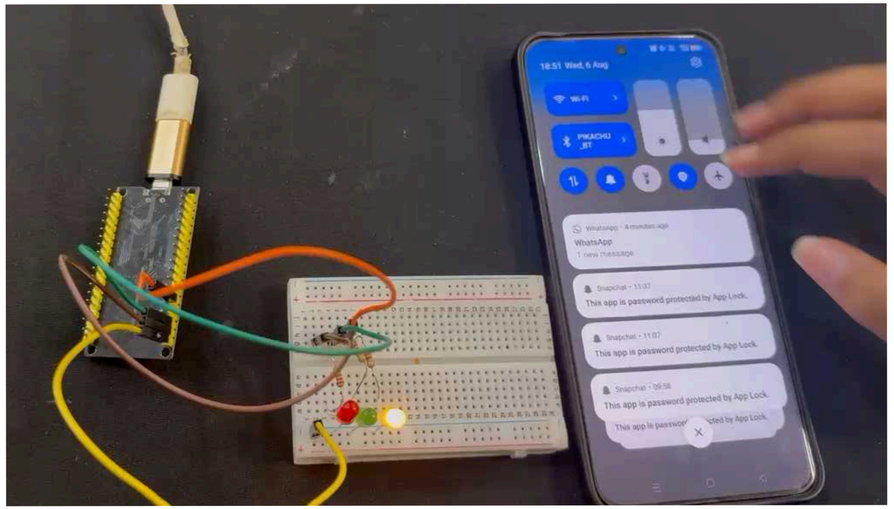


Components

- ESP32 Development Board
- 3 × LEDs (any color)
- 3 × Resistors — 220 Ω each
- Breadboard + jumper wires
- Smartphone with BLE app (e.g., nRF Connect)

Circuit

- LED1:
  - Anode → ESP32 GPIO 2 through 220 Ω resistor
  - Cathode → ESP32 GND
- LED2:
  - Anode → ESP32 GPIO 4 through 220 Ω resistor
  - Cathode → ESP32 GND
- LED3:
  - Anode → ESP32 GPIO 5 through 220 Ω resistor
  - Cathode → ESP32 GND



ASSIGNMENT 26

Controlling 3 LEDs with BLE on ESP32

How it Works

- ESP32 acts as a BLE server → advertises service for LED control.
- Smartphone sends commands like LED1_ON, LED2_OFF, etc.
- BLE characteristic onWrite() callback handles the commands → turns LEDs ON/OFF.
- LEDs respond in real-time to wireless commands.

CODE

```cpp
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

#define LED1 2
#define LED2 4
#define LED3 5

BLEServer* pServer = NULL;
BLECharacteristic* pCharacteristic = NULL;
bool deviceConnected = false;

class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer){ deviceConnected = true; }
  void onDisconnect(BLEServer* pServer){ deviceConnected = false; }
};

class MyCallbacks: public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *pCharacteristic) {
    std::string value = pCharacteristic->getValue();

    if(value == "LED1_ON") digitalWrite(LED1, HIGH);
    else if(value == "LED1_OFF") digitalWrite(LED1, LOW);
    if(value == "LED2_ON") digitalWrite(LED2, HIGH);
    else if(value == "LED2_OFF") digitalWrite(LED2, LOW);

    if(value == "LED3_ON") digitalWrite(LED3, HIGH);
    else if(value == "LED3_OFF") digitalWrite(LED3, LOW);
  }
};

void setup() {
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);

  BLEDevice::init("ESP32_3LED");
  pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  BLEService *pService = pServer->createService(BLEUUID((uint16_t)0xFFE0));
  pCharacteristic = pService->createCharacteristic(
                      BLEUUID((uint16_t)0xFFE1),
                      BLECharacteristic::PROPERTY_WRITE
                    );
  pCharacteristic->setCallbacks(new MyCallbacks());
  pCharacteristic->addDescriptor(new BLE2902());

  pService->start();
  pServer->getAdvertising()->start();
}

void loop() {
  // BLE callbacks handle LED control, no code needed here
}
```

Learnings

- BLE allows low-power wireless control of multiple devices ESP32 can handle multiple outputs independently using BLE commands.
- Useful for IoT home automation, smart lighting, or remote control projects.
- Adding more LEDs requires only additional pins and command handling.

ASSIGNMENT 28

PCB Design for RGB LED with Current-Limiting Resistors

This PCB connects RGB LEDs with 220Ω resistors on each color channel (Red, Green, Blue) for current limiting, ensuring safe operation and independent color control.

Schematic Design

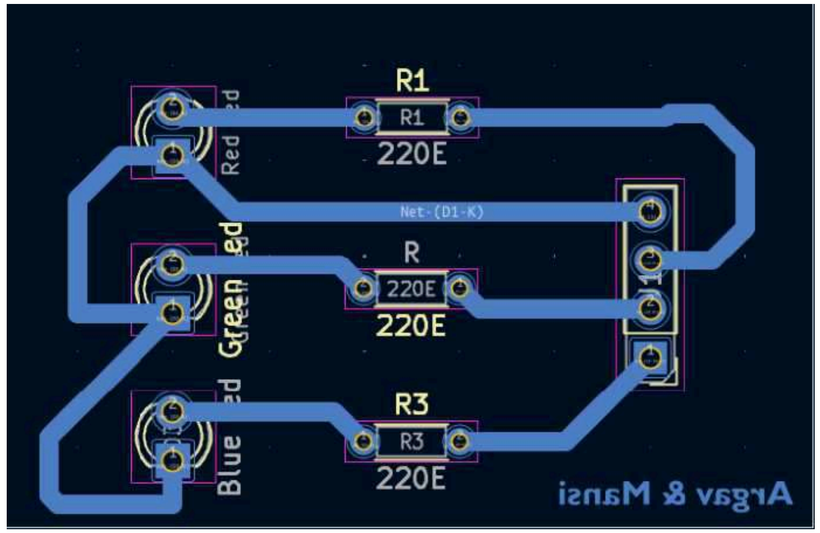
Open KiCAD and create a new project.
Place the following components:

- RGB LED (Common Cathode or Common Anode depending on requirement)
- Three resistors (220Ω each) labeled R1, R2, R3.
- Input header pins for external connection.

Connections:

- Red pin of LED → R1 → Input (Red control).
- Green pin of LED → R2 → Input (Green control).
- Blue pin of LED → R3 → Input (Blue control).
- Common pin of LED → GND (for common cathode) or VCC (for common anode).

kiCad

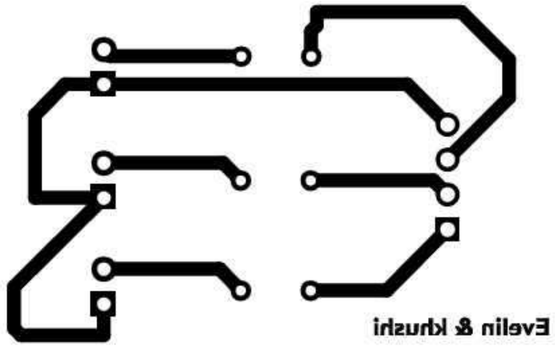


ASSIGNMENT 28

PCB Design for RGB LED with Current-Limiting Resistors

PCB Layout

1. Switch to PCB editor.
2. Place the RGB LED footprint on one side of the board.
3. Arrange the resistors near each color pin (R1, R2, R3).
4. Place a 3-pin header for Red, Green, Blue inputs and another pin for GND.
5. Route traces:
   - Each input → resistor → corresponding LED pin.
   - Common LED pin → GND (if common cathode) or VCC (if common anode).
6. Use wide traces if you plan to drive higher current.



ASSIGNMENT 28

PCB Design for RGB LED with Current-Limiting Resistors

Making PCB

1. Generate Gerber files (File → Plot in KiCAD).
2. Fabricate the PCB:
   DIY: print layout, transfer to copper board, etch, and drill holes.
   Soldering components:
   - Solder resistors R1, R2, R3 (220Ω).
   - Solder RGB LED.
   - Solder input header pins.

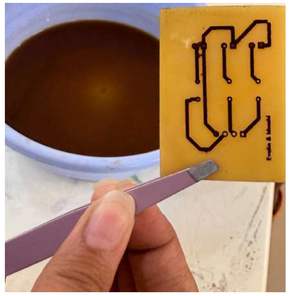
Connections

Connect input pins to a microcontroller (e.g., ESP32/Arduino):

- Red pin → GPIO (with PWM control)
- Green pin → GPIO
- Blue pin → GPIO
- Common pin → GND (for common cathode) or 3.3V/5V (for common anode).

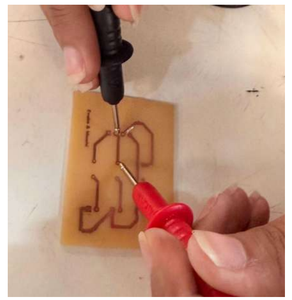
Printing & Transfer – The PCB layout is printed and transferred onto the copper board.

Washing – The board is cleaned with water to remove excess paper.

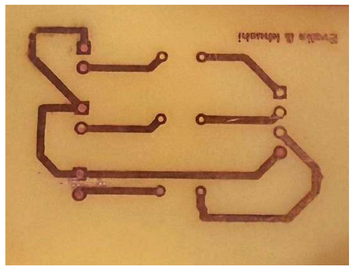


Etching – The board is placed in ferric chloride solution to remove unwanted copper.

it is ccleaned with thinner and copper will be visiblw



connections are checked with multimeter.



Final PCB – The etched and cleaned PCB is ready for soldering components.

ASSIGNMENT 29

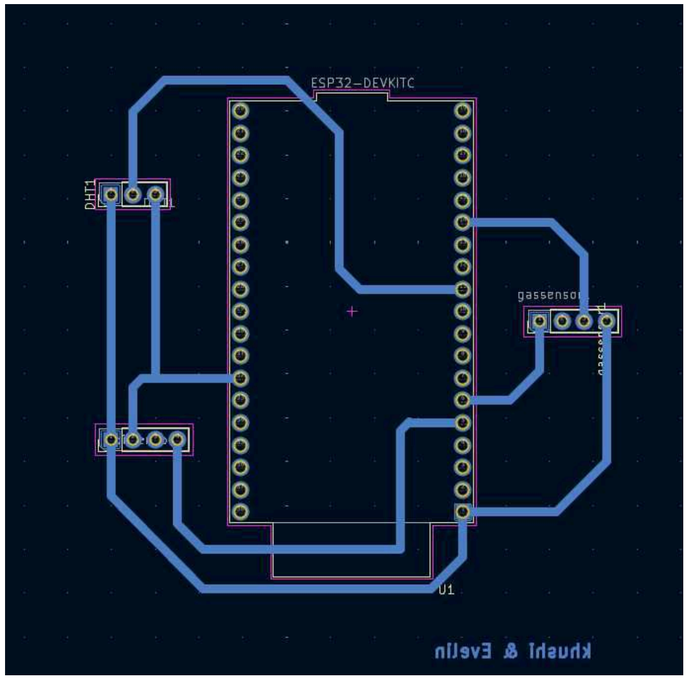
PCB Design for ESP32 with Soil, DHT, and Gas Sensors

This PCB integrates an ESP32 DevKitC microcontroller with three environmental sensors: a soil moisture sensor, a DHT sensor (temperature & humidity), and a gas sensor. The layout is designed in KiCAD, with clean routing to ensure stable connections and minimize interference. This compact board allows easy prototyping and testing of smart agriculture or environmental monitoring systems.

Schematic Design

1. Open KiCAD and create a new project.
2. Place the ESP32 DevKitC module footprint as the main controller.
3. Add the following sensor connectors:
   - Soil sensor (3-pin: VCC, GND, Signal).
   - DHT sensor (3-pin: VCC, GND, Data).
   - Gas sensor (4-pin: VCC, GND, AO, DO).
4. Connect sensor pins to appropriate GPIO pins on the ESP32.
   - Soil sensor signal → GPIO 34 (ADC).
   - DHT data → GPIO 4.
   - Gas sensor digital output → GPIO 15.
   - All sensors share 3.3V VCC and GND from ESP32.
5. Add a decoupling capacitor (100nF) near ESP32 VCC for power stability.

kiCad



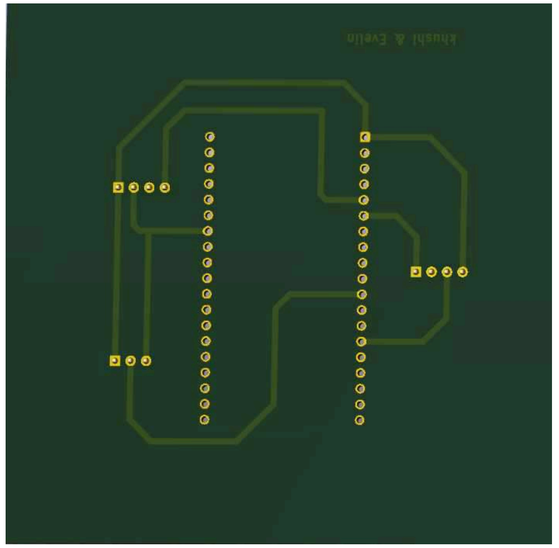
ASSIGNMENT 29

PCB Design for ESP32 with Soil, DHT, and Gas Sensors

PCB Layout

1. Import the schematic into PCB Editor.
2. Place ESP32 DevKitC footprint at the center.
3. Arrange sensor connectors around the board edges for easy access.
    - Soil sensor on the left.
    - DHT sensor on the top left.
    - Gas sensor on the right.
4. Use thicker traces (≥1mm) for VCC and GND lines to handle current properly.
5. Route signal traces using shortest paths to reduce noise.
6. Add ground fill (copper pour) to minimize EMI and improve stability.
7. Add mounting holes if needed for casing.

kiCad

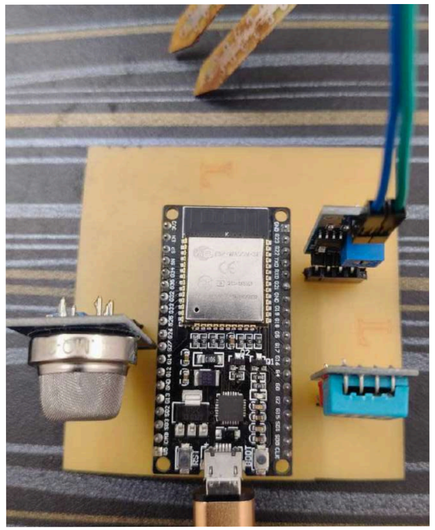


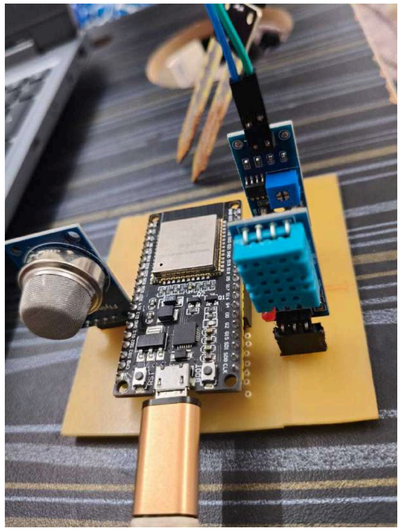
ASSIGNMENT 29

PCB Design for ESP32 with Soil, DHT, and Gas Sensors

Connections

1. Soil Sensor
   - VCC → 3.3V
   - GND → GND
   - Signal → GPIO34
2. DHT Sensor
   - VCC → 3.3V
   - GND → GND
   - Data → GPIO4
3. Gas Sensor
   - VCC → 3.3V
   - GND → GND
   - DO (Digital Output) → GPIO15





Making PCB

DIY PCB

- Print PCB layout on glossy paper (laser printer).
- Heat-transfer to copper board (iron/laminator).
- Etch in ferric chloride.
- Drill holes for components.
- (Optional) Apply solder mask & tin coat.

Soldering Components

- Solder male headers for ESP32 DevKitC.
- Add 3/4-pin female headers for sensors.
- Place capacitors/LEDs if needed.
- Connect sensors after soldering.