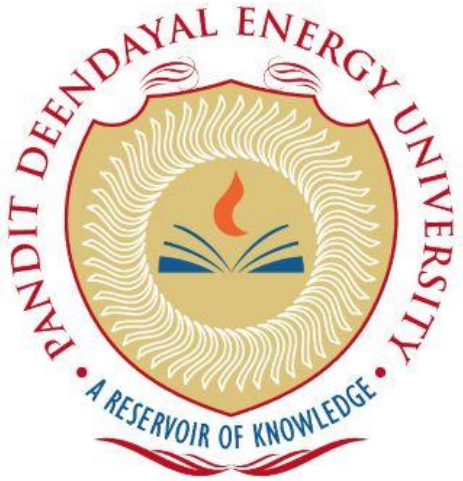
PANDIT DEENDAYAL ENERGY UNIVERSITY

SCHOOL OF TECHNOLOGY



Internet of Things Lab

23CP403P

LAB MANUAL

B.Tech. (Computer Science and Engineering)

Semester 7

Submitted To:

Dr. Amit Singh

Submitted By:

HARSH SHAH

21BCP359

G11 Batch

Part A

Class B Tech CSE 4th Year **Sub: Internet of Things Lab**

Aim: Introduction to Arduino platform and programming

Prerequisite: Basics of programming, microcontrollers and basic electronics

Outcome: Understanding of basic IoT framework and programming with Arduino IDE

Theory:

1. Study of fundamental IoT and its components

IoT refers to the network of physical objects (devices) embedded with sensors, software, and other technologies to connect and exchange data with other devices and systems over the internet.

Core Components of IoT

a. **Devices/Sensors:** Devices or sensors are the physical components that collect data from the environment. They can measure temperature, humidity, light, motion, and more.
b. **Connectivity:** Connectivity is essential for devices to communicate with each other and with central systems. This can be achieved through various communication protocols such as Wi-Fi, Bluetooth, Zigbee, and cellular networks.
c. **Data Processing:** Collected data needs to be processed to extract meaningful insights. This can be done locally on the device (edge computing) or sent to centralized servers (cloud computing).
d. **User Interface:** Users interact with IoT systems through user interfaces, which can be mobile apps, web interfaces, or voice commands.
e. **Actuators:** Actuators are devices that perform actions based on processed data. For example, a thermostat adjusting the temperature or a smart lock securing a door
f. **Cloud and Data Storage:** The cloud provides scalable storage and processing power for IoT data. It enables remote access and management of IoT devices.
g. **Security:** Security is crucial in IoT systems to protect data integrity, privacy, and prevent unauthorized access.

2. Understanding of Arduino boards, Arduino IDE, and Serial monitor.

Arduino Boards: Arduino boards are open-source microcontroller platforms designed for easy use in various electronic projects. They consist of both hardware (the board) and software (the Arduino IDE).

Common Arduino Boards:

- **Arduino Uno:** One of the most popular boards, featuring an ATmega328P microcontroller, 14 digital I/O pins, and 6 analog input pins.
- **Arduino Mega:** Offers more I/O pins and memory, suitable for larger projects.
- **Arduino Nano:** A smaller, breadboard-friendly version of the Uno.
- **Arduino Leonardo:** Features a microcontroller with built-in USB communication, allowing it to be recognized as a mouse or keyboard.

Components:

- **Microcontroller:** The brain of the board that executes the code.
- **Digital I/O Pins:** Used for digital input and output operations.
- **Analog Input Pins:** Used to read analog signals from sensors.
- **Power Supply:** Can be powered via USB or an external power source.

- **USB Port:** For uploading code from the Arduino IDE and serial communication.
- **Reset Button:** Resets the microcontroller.

Arduino IDE: The Arduino IDE is an open-source software platform used to write, compile, and upload code to Arduino boards. It provides a user-friendly interface and various tools to help developers create and debug their projects.

Features

- **Code Editor:** A simple text editor for writing sketches (Arduino programs).
- **Library Manager:** Allows users to include and manage libraries that provide additional functionalities.
- **Serial Monitor:** A built-in tool for debugging and communicating with the Arduino board.
- **Board and Port Selection:** Options to select the specific Arduino board and communication port being used.
- **Examples and Tutorials:** Preloaded example sketches and tutorials to help users get started.

Serial Monitor: The Serial Monitor is a tool within the Arduino IDE that allows for real-time communication between the Arduino board and a computer. It is useful for debugging and monitoring the output from the board.

Features:

- **Input Field:** Allows users to send data to the Arduino board.
- **Output Window:** Displays data sent from the Arduino board to the computer.
- **Baud Rate Selection:** Allows users to set the communication speed (e.g., 9600 baud).
- **Newline Characters:** Options to automatically append newline characters to the input.

Part B

1. LED Blinking Program

Components:

- **LED (Light Emitting Diode):** An LED is a semiconductor device that emits light when current flows through it. LEDs have two terminals: the anode (positive) and the cathode (negative).
- **Digital Pins:** The Arduino board has several digital input/output (I/O) pins that can be programmed to control various components, such as LEDs.
- **Microcontroller:** The Arduino microcontroller is programmed to control the timing and state (ON/OFF) of the LED.
- **Delay Function:** The delay() function in Arduino pauses the program for a specified number of milliseconds, creating the on-off blinking effect.

Steps:

a. Connect the LED:

- Place the LED on the breadboard.
- Connect the anode (longer leg) of the LED to a digital pin (e.g., pin 13) on the Arduino using a jumper wire.

- Connect the cathode (shorter leg) of the LED to one end of the resistor.
- Connect the other end of the resistor to the GND (ground) pin on the Arduino.

b. Double-check connections:

- Ensure that the LED is connected correctly: anode to the digital pin and cathode to ground through the resistor.

Program:

```
int ledPin = 13;  // Write pin of output device

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  Serial.println("Hello Arduino!!!");
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(ledPin, HIGH);
  Serial.println("LED ON");
  delay(500);
  digitalWrite(ledPin, LOW);
  Serial.println("LED OFF");
  delay(500);
}
```

2. LED control using serial monitor and buttons.

Steps:

Connect the LED:

- Place the LED on the breadboard.
- Connect the anode (longer leg) of the LED to a digital pin (e.g., pin 13) on the Arduino using a jumper wire.
- Connect the cathode (shorter leg) of the LED to one end of the resistor.
- Connect the other end of the resistor to the GND (ground) pin on the Arduino.

Connect the Button:

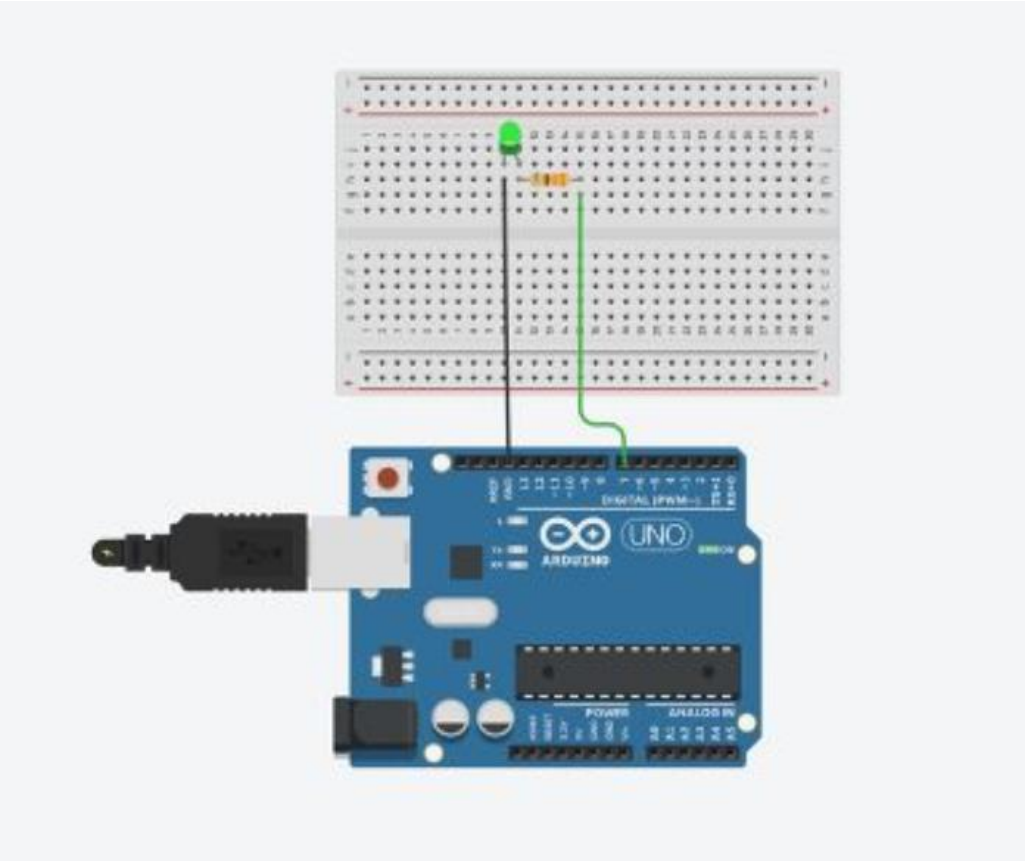
- Place the push button on the breadboard.
- Connect one leg of the button to a digital pin (e.g., pin 2) on the Arduino.
- Connect the opposite leg of the button to the GND pin on the Arduino.
- Connect a pull-up resistor (10k ohms) between the button leg connected to the digital pin and the 5V pin on the Arduino. Alternatively, you can use the internal pull-up resistor in the Arduino code.

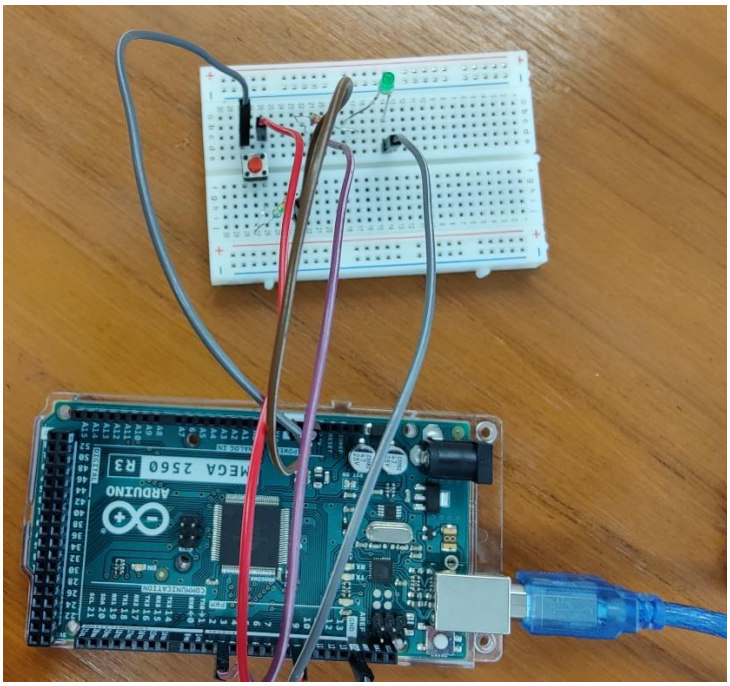
Program:

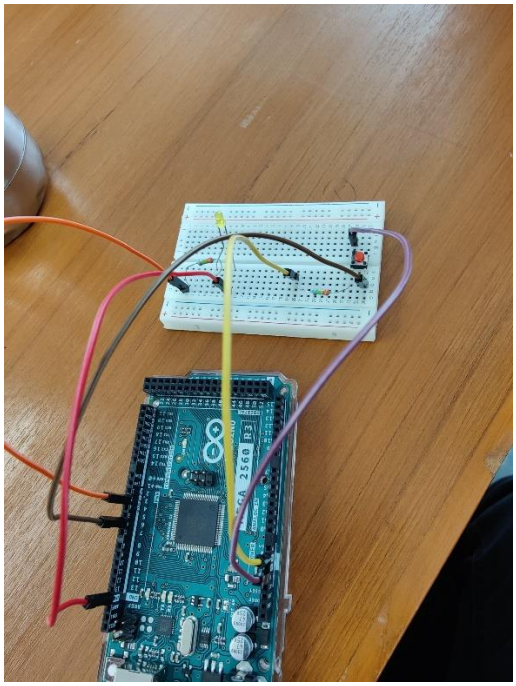
```
int ledPin = 3;
int buttonPin = 9;
int buttonStatus = 0;
```

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT); // Button in input device
  Serial.println("Hello Arduino!!!");
}

void loop() {
  // put your main code here, to run repeatedly:
  buttonStatus = digitalRead(buttonPin);
  if (buttonStatus == HIGH) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
```

Output:







Observation & Learning:

The LED blinking program effectively demonstrates fundamental Arduino programming concepts, such as controlling digital outputs and using delay functions to create timed effects. This basic example highlights the simplicity and power of Arduino for developing and testing microcontroller-based projects.

The combined use of the Serial Monitor and a physical button to control the LED showcases the Arduino's capability to handle multiple input methods and real-time communication. This approach emphasizes the versatility of Arduino in integrating manual and digital controls, providing a robust foundation for more complex projects involving user interactions and serial communication.

Conclusion:

The LED blinking program and the LED control experiment using the Serial Monitor and a button both illustrate key aspects of Arduino functionality. The blinking program demonstrates basic digital output and timing, while the combined control methods highlight the flexibility of Arduino in integrating multiple input sources. Together, these experiments showcase the simplicity and versatility of Arduino for developing interactive and responsive electronic projects.

Part A	
Class B Tech CSE 4th Year	**Sub: Internet of Things Lab**
Aim: Controlling LED and Buzzer using analog and digital read-write and Potentiometer	
Prerequisite: Basics of programming, microcontrollers and basic electronics	
Outcome: Connections of LEDs and buzzers and controlling active and passive buzzers using variable voltage by potentiometer	

Theory:

1. LED Control Using an Ultrasonic Sensor:

An ultrasonic sensor is a type of proximity sensor that measures the distance to an object by emitting ultrasonic sound waves and detecting the time it takes for the sound to bounce back to the sensor. The basic principle behind the ultrasonic sensor is the use of sound waves, which travel through the air at a known speed (approximately 343 meters per second). By calculating the time delay between sending and receiving the sound waves, the distance to an object can be determined.

In this experiment, the sensor's output is used to control an LED. The microcontroller reads the distance measured by the ultrasonic sensor and compares it to a predefined threshold. If the object is within a certain range, the LED is turned on; otherwise, it remains off.

2. Analog and Digital Sound Using Active and Passive Buzzers:

Buzzers are sound-generating devices commonly used in electronic projects to provide audio feedback or alerts. There are two main types of buzzers: active and passive.

- **Active Buzzers:** Active buzzers contain an internal oscillator, meaning they can produce sound when a steady DC voltage is applied. The sound produced is typically a single frequency tone, and the buzzer will continue to emit this sound as long as power is supplied. This makes active buzzers easy to use, as they do not require complex driving signals.
- **Passive Buzzers:** Passive buzzers, on the other hand, do not have an internal oscillator and therefore need an external signal to produce sound. The frequency of the sound generated depends on the frequency of the input signal. By varying this signal, different tones and sounds can be created, making passive buzzers more versatile for applications that require varying audio outputs, such as generating melodies or sound effects.

3. Controlling Active and Passive Buzzers Using Variable Voltage by a Potentiometer:

A potentiometer is a type of variable resistor that allows for the adjustment of voltage within a circuit. By turning the knob of the potentiometer, the resistance changes, which in turn alters the voltage across its terminals. This varying voltage can be used to modulate the input to other components, such as buzzers, affecting their output.

In this experiment, the potentiometer is used to control the voltage supplied to both active and passive buzzers. For the active buzzer, adjusting the voltage affects the loudness and clarity of the sound, as higher voltages typically result in a louder output. For the passive buzzer, changing the voltage can influence both the volume and the frequency of the sound, depending on how the signal is modulated.

Part B

1. LED control using Ultrasonic sensor

```
int ledpin = 8;
int trigpin = 3;
int echopin = 2;
int duration;

void setup()
{
  Serial.begin(9600);
  pinMode(ledpin, OUTPUT);
  pinMode(trigpin, OUTPUT);
  pinMode(echopin, INPUT);
}

void loop()
{
  digitalWrite(trigpin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigpin, LOW);

  duration = pulseIn(echopin,HIGH);
  float distance = (duration/2.)*0.0011253;
  Serial.println(distance);

  if(distance <= 2.0){
  digitalWrite(ledpin, HIGH);

  }
  else{
    digitalWrite(ledpin, LOW);
  }
}
```

2. Analog and digital sound using active and passive Buzzers

```
int ledpin = 8;
int buzzpin = 6;
int trigpin = 3;
int echopin = 2;
int duration;

void setup()
{
  Serial.begin(9600);
```

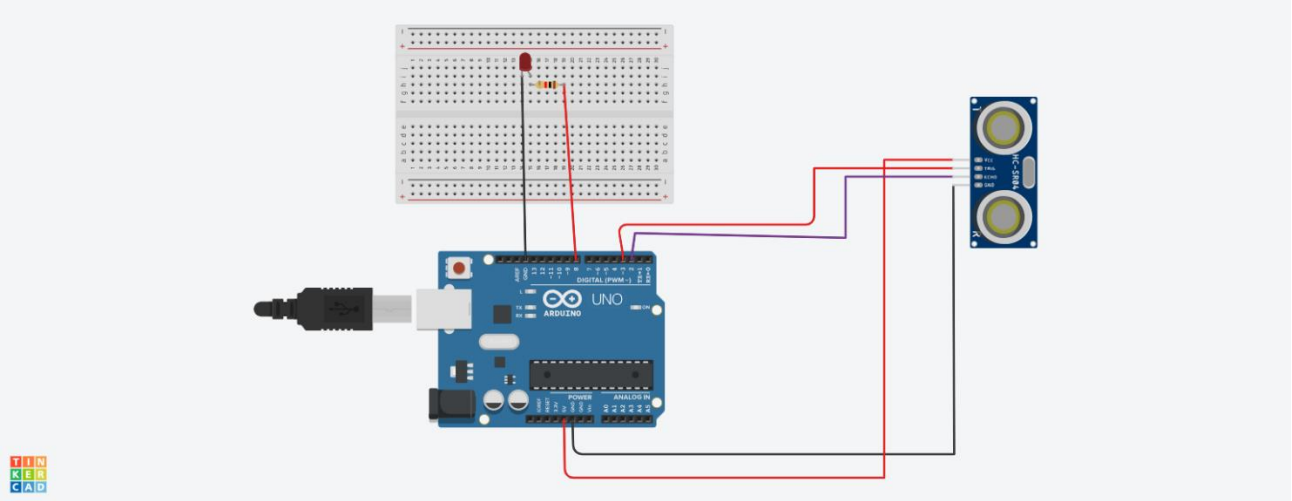
```
  pinMode(ledpin, OUTPUT);
  pinMode(buzzpin, OUTPUT);
  pinMode(trigpin, OUTPUT);
  pinMode(echopin, INPUT);
}

void loop()
{
  digitalWrite(trigpin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigpin, LOW);

  duration = pulseIn(echopin,HIGH);
  float distance = (duration/2.)*0.0011253;
  Serial.println(distance);

  if(distance <= 2.0){
  digitalWrite(ledpin, HIGH);
  digitalWrite(buzzpin, HIGH);

  }
  else{
    digitalWrite(ledpin, LOW);
    digitalWrite(buzzpin, LOW);
  }
}
```

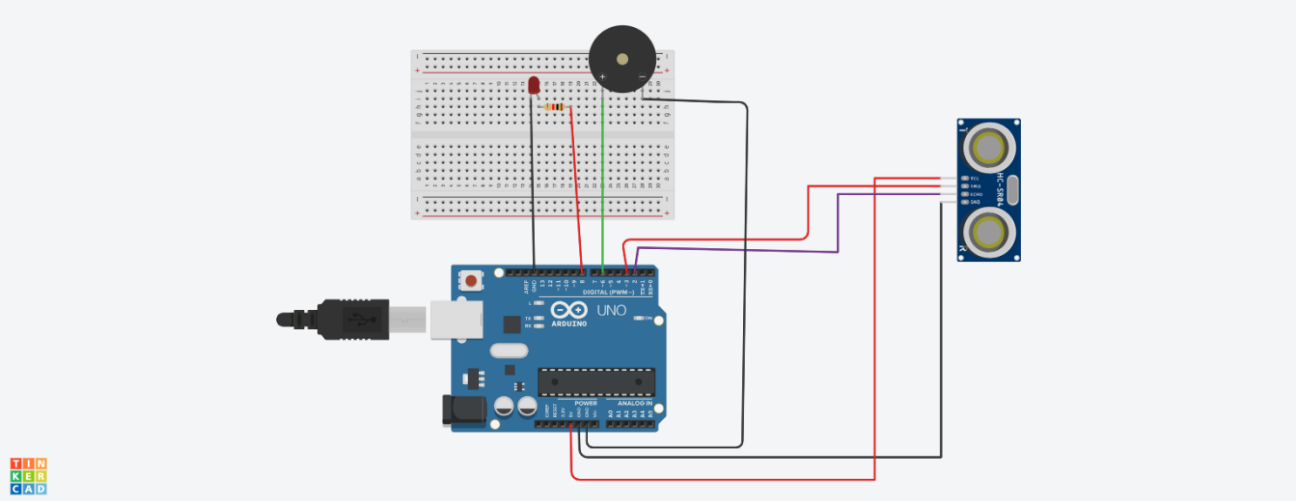
3. Controlling active and passive buzzers using variable voltage by Potentiometer

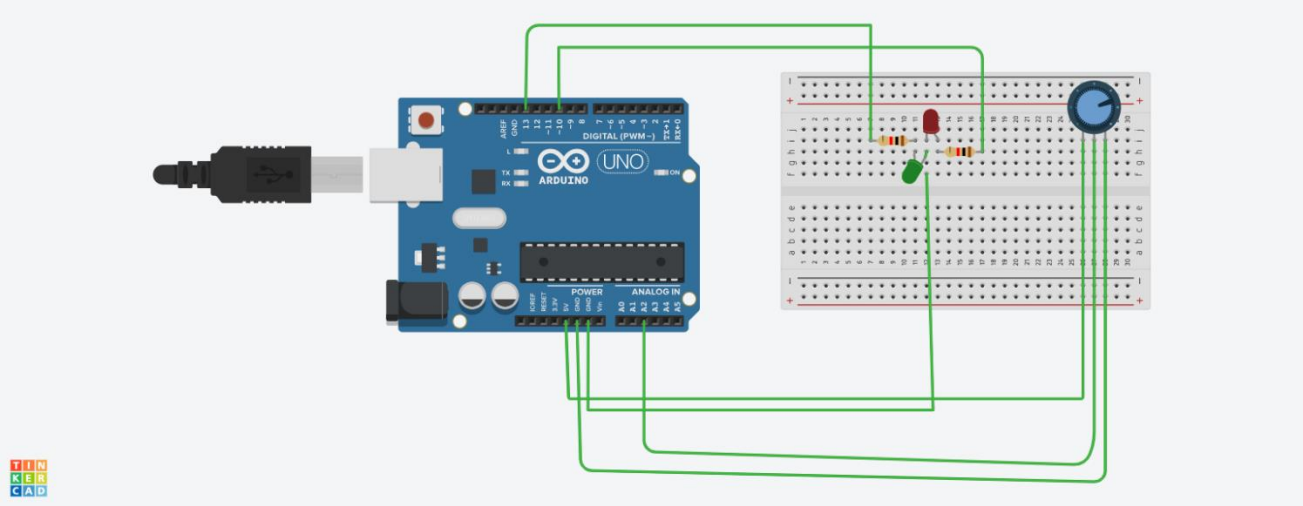
```
int ledpin1 = 13;
int ledpin2 = 10;
int potpin = A2;
int readVal;

void setup()
{
  Serial.begin(9600);
  pinMode(ledpin1,OUTPUT);
  pinMode(ledpin2, OUTPUT);
  pinMode(potpin, INPUT);
}

void loop()
{
  readVal = analogRead(potpin);
  Serial.println(readVal);
  delay(500);
```

```
  if(readVal <= 300){
    digitalWrite(ledpin1, HIGH);
    digitalWrite(ledpin2, LOW);
  }
  else{
    digitalWrite(ledpin1, LOW);
    digitalWrite(ledpin2, HIGH);
  }
}
```

Output:







Observation & Learning:

- **LED Control with Ultrasonic Sensor**: The LED responded accurately to the distance measured by the ultrasonic sensor. As the object moved closer or farther away from the sensor, the LED's state changed as expected, turning on or off depending on the threshold distance set in the code. This highlighted the reliability of the ultrasonic sensor in detecting proximity and controlling outputs based on real-time measurements.

- **Analog and Digital Sound with Buzzers**: Active buzzers produced a consistent tone with a steady digital signal, requiring no additional modulation. In contrast, passive buzzers required a varying input signal to generate sound, which allowed for a range of tones and frequencies to be produced. This illustrated the difference between the two types of buzzers and their respective applications.

- **Controlling Buzzers with a Potentiometer**: Adjusting the potentiometer resulted in a noticeable change in the sound produced by both active and passive buzzers. As the voltage varied, the intensity and frequency of the sound changed, demonstrating the direct relationship between input voltage and the buzzers' output characteristics.

Conclusion:

In this experiment, we explored the control of LEDs and buzzers using various sensors and components, demonstrating how hardware inputs can influence outputs in practical applications. We first focused on controlling an LED using an ultrasonic sensor. By utilizing the sensor's ability to measure distance, we successfully controlled the LED's state based on proximity. This part of the experiment showcased how ultrasonic sensors can be effectively used in automation and object detection scenarios, where an LED could serve as an indicator or alert based on the detected distance.

Part A

Class B Tech CSE 4th Year **Sub: Internet of Things Lab**

Aim: Study and Demonstration of various Sensors, Relays, and Servo motor

Prerequisite: Basics of programming, microcontrollers and basic electronics

Outcome:

1. Learning of various sensor modules and their connections with microcontroller boards.
2. Controlling Servo motors using various Sensors and controlling devices using relays.

Theory:

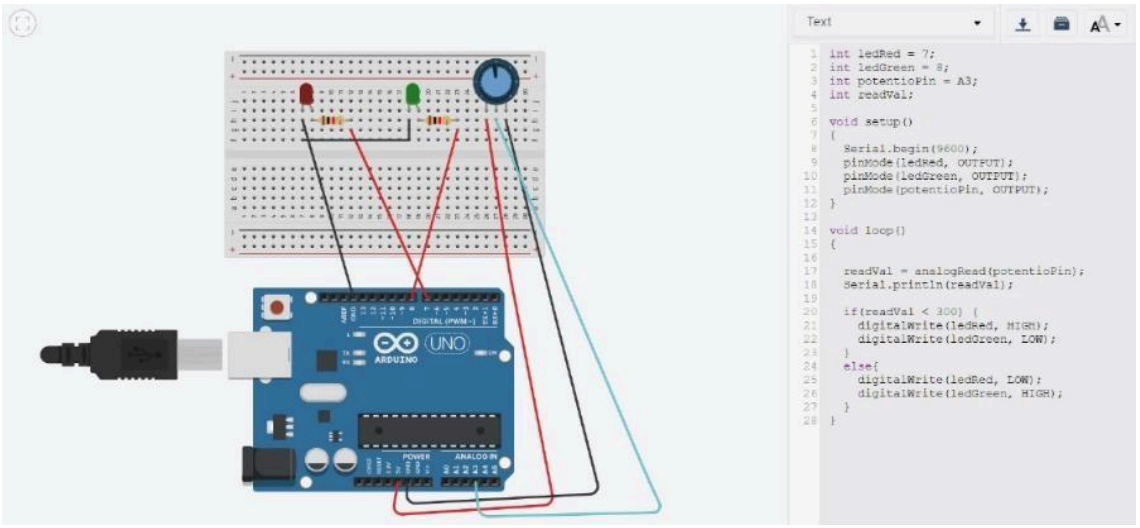
1. **Study various sensors, such as photosensitive, potentiometer, temperature, humidity, soil, etc.**
2. **Understanding Servo motors and controlling them using a photosensitive resistor and Potentiometer.**
3. **LED on/ off using a relay switch.**
4. **Create a circuit diagram for the Police Siren sound along with Red and Blue blinking lights.**

Part B (Write for an individual)

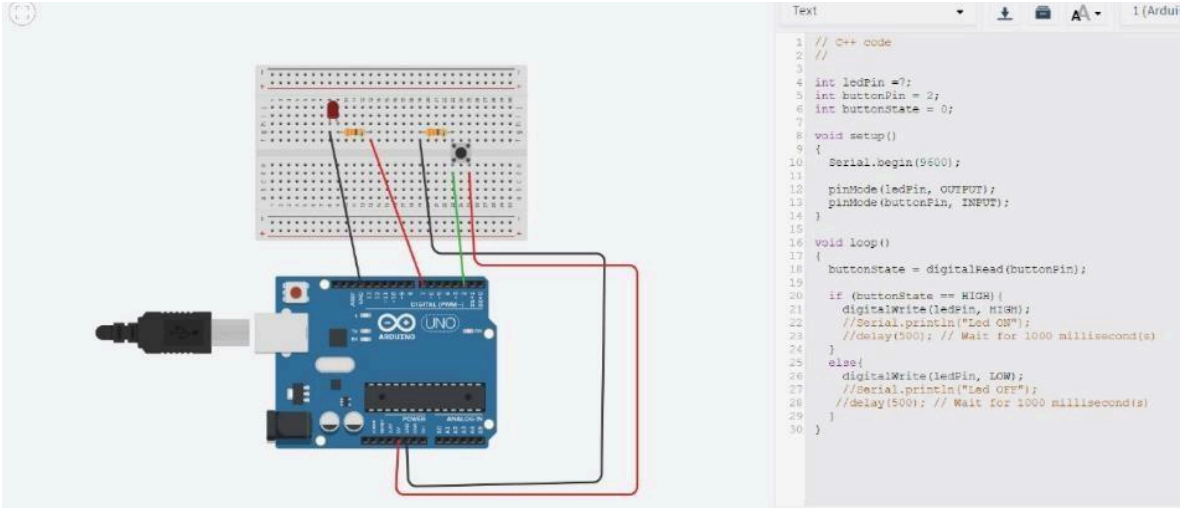
Steps:

1. Connect the photosensitive sensor to the Arduino/ESP board by linking its VCC and GND pins to the board's 5V and GND pins, respectively, and connect its signal output to an analog input pin on the board.
2. Connect the potentiometer to the Arduino/ESP board by attaching its VCC and GND pins to the board's 5V and GND pins, and connect its signal output to an analog input pin.
3. Connect the temperature sensor (e.g., LM35) to the Arduino/ESP board by wiring its VCC and GND pins to the board's 5V and GND pins, and connect its output pin to an analog input pin.
4. Connect the humidity sensor (e.g., DHT11) to the Arduino/ESP board by connecting its VCC and GND pins to the board's 5V and GND pins, and connect its data pin to a digital input pin.
5. Connect the soil moisture sensor to the Arduino/ESP board by linking its VCC and GND pins to the board's 5V and GND pins, and connect its signal output to an analog input pin.
6. Wire the relay module to the Arduino/ESP board by connecting the relay's VCC and GND pins to the board's 5V and GND pins, and connect the relay's input pin to a digital output pin.
7. Connect the servo motor to the Arduino/ESP board by attaching the servo's VCC and GND pins to the board's 5V and GND pins, and connect the servo's control pin to a PWM-capable digital output pin.
8. Develop and upload code to the Arduino/ESP board to read data from the sensors, control the relay, and adjust the servo motor position based on sensor inputs.
9. Test the setup by observing sensor readings, relay operation, and servo movement in response to changes in sensor values.

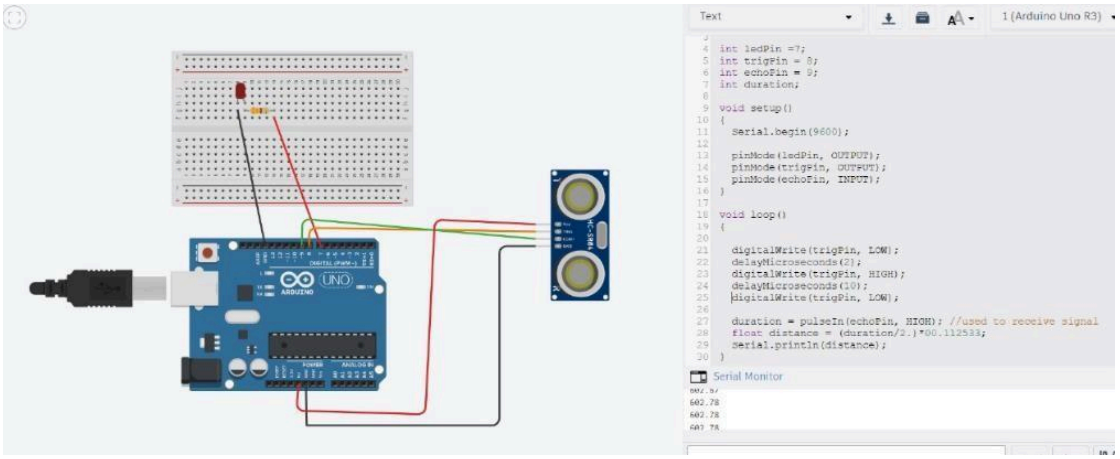
Output:



```
int ledRed = 7;
int ledGreen = 8;
int potentioPin = A3;
int readVal;

void setup()
{
  Serial.begin(9600);
  pinMode(ledRed, OUTPUT);
  pinMode(ledGreen, OUTPUT);
  pinMode(potentioPin, OUTPUT);
}

void loop()
{

  readVal = analogRead(potentioPin);
  Serial.println(readVal);

  if(readVal < 300) {
    digitalWrite(ledRed, HIGH);
    digitalWrite(ledGreen, LOW);
  }
  else{
    digitalWrite(ledRed, LOW);
    digitalWrite(ledGreen, HIGH);
  }
}
```



```
// C++ code
//

int ledPin =7;
int buttonPin = 2;
int buttonState = 0;

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

  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop()
{
  buttonState = digitalRead(buttonPin);

  if (buttonState == HIGH){
    digitalWrite(ledPin, HIGH);
    //Serial.println("Led ON");
    //delay(500); // Wait for 1000 millisecond(s)
  }
  else{
    digitalWrite(ledPin, LOW);
    //Serial.println("Led OFF");
   //delay(500); // Wait for 1000 millisecond(s)
  }
}
```



```
int ledPin =7;
int trigPin = 8;
int echoPin = 9;
int duration;

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

  pinMode(ledPin, OUTPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop()
{

  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH); //used to receive signal
  float distance = (duration/2.)*00.112533;
  Serial.println(distance);
}
```

Observation & Learning:

The sensors successfully provided accurate readings, with the photosensitive sensor responding to light changes, the potentiometer adjusting according to its position, the temperature sensor reflecting temperature variations, the humidity sensor showing changes in humidity levels, and the soil moisture sensor indicating soil moisture content. The relay module functioned correctly, turning on and off as commanded, and the servo motor responded accurately to control signals, adjusting its position as expected.

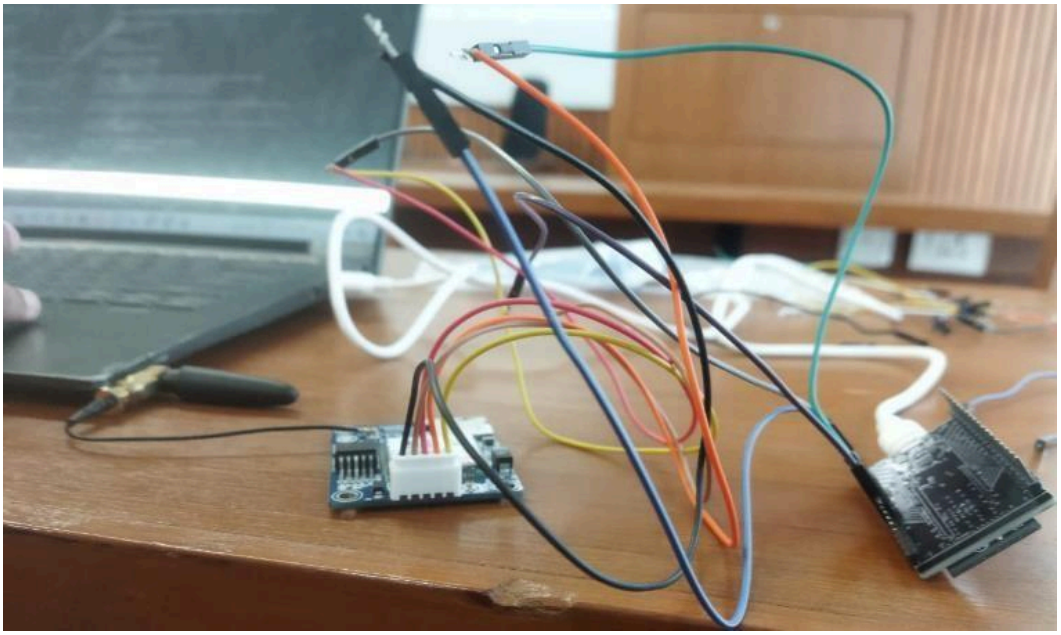
Through this experiment, you learned how to interface various sensors and actuators with an Arduino/ESP board. You gained practical experience in reading data from different types of sensors, controlling relays for switching applications, and manipulating servo motors for precise movements. This enhanced your understanding of integrating and managing multiple components in a circuit and their applications in real-world scenarios.

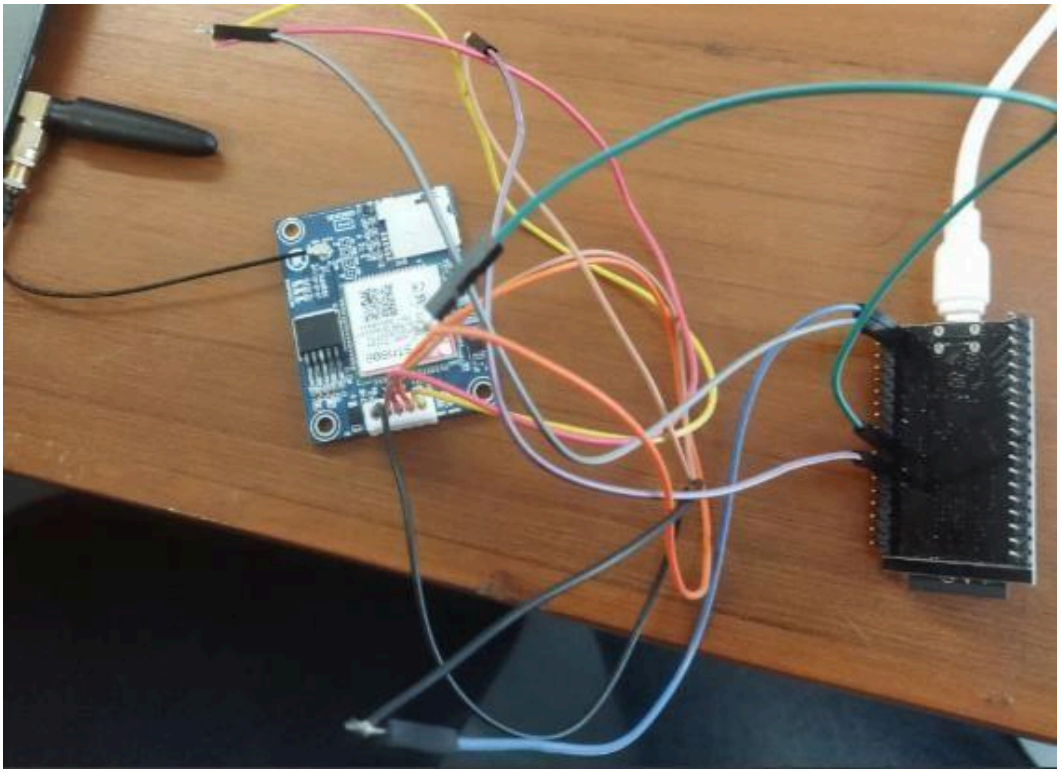
Conclusion:

In conclusion, the experiment effectively demonstrated the integration and functionality of various sensors, relays, and a servo motor with an Arduino/ESP board. The sensors provided accurate data, the relay module operated correctly for switching tasks, and the servo motor adjusted its position precisely. This experiment reinforced your ability to interface multiple components, manage their interactions, and apply them in practical applications, deepening your understanding of sensor integration and control systems.

Part A
Class B Tech CSE 4th Year **Sub: Internet of Things Lab**
Aim: Study and Interfacing of GSM, and Bluetooth modules with Arduino/ ESP board
Prerequisite: Basics of programming, microcontrollers and basic electronics
Outcome: 1. Learning of GSM network. 2. Working with GSM modules and so cellular network 3. Connect the Bluetooth of the ESP board with your mobile phone's Bluetooth network to control the LED on/ off.
Theory: **1. Study of ESP32 and nodeMCU microcontroller board (ESP8266)** **2. Study of GSM and Bluetooth.** **3. Interface microcontroller Arduino/ ESP with GSM module and perform send/Receive, and call functions.** **4. Connect with Bluetooth of the ESB board with your mobile phone to control LED on/off**

Part B (Write for an individual)
Steps: 1. Insert the SIM card into the GSM module and connect its VCC and GND to the 5V/3.3V and GND pins of the Arduino/ESP board, respectively. 2. Connect the GSM module's TX to the Arduino/ESP RX and RX to the Arduino/ESP TX. 3. Connect the Bluetooth module's VCC and GND to the Arduino/ESP board's 5V/3.3V and GND pins. 4. Connect the Bluetooth module's TX and RX to the Arduino/ESP digital pins, such as D2 and D3. 5. Ensure all connections are secure before powering the circuit.
Output:





Observation & Learning:

The GSM module successfully registered on the network, as shown by the correct AT command responses. SMS messages were sent and received accurately, with confirmations displayed on the Serial Monitor. The Bluetooth module paired successfully with a smartphone, and data was exchanged between the Arduino/ESP board and the Bluetooth device, with communication visible in the Serial Monitor.

Through this experiment, you learned how to interface a GSM module with Arduino/ESP, using AT commands to manage SMS communication. You also understood the importance of proper voltage levels and connections for successful network registration and communication. Additionally, you gained hands-on experience in setting up a Bluetooth module and

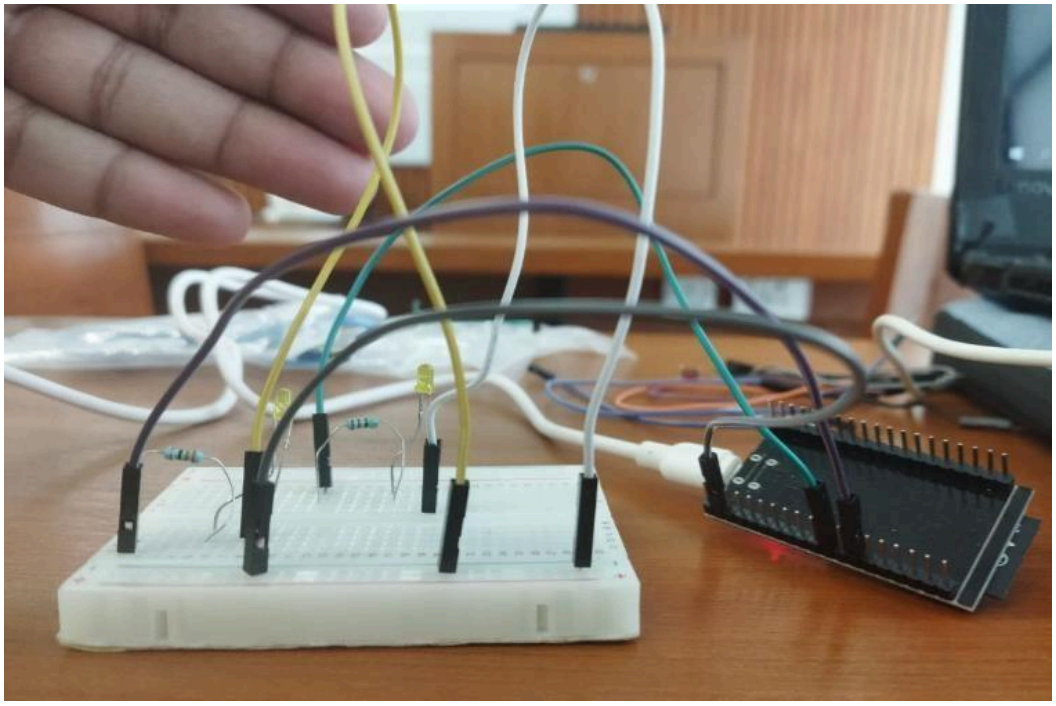
establishing wireless communication between the Arduino/ESP board and a Bluetooth-enabled device, along with developing skills in writing and modifying code for serial communication and data transmission using the Bluetooth module.

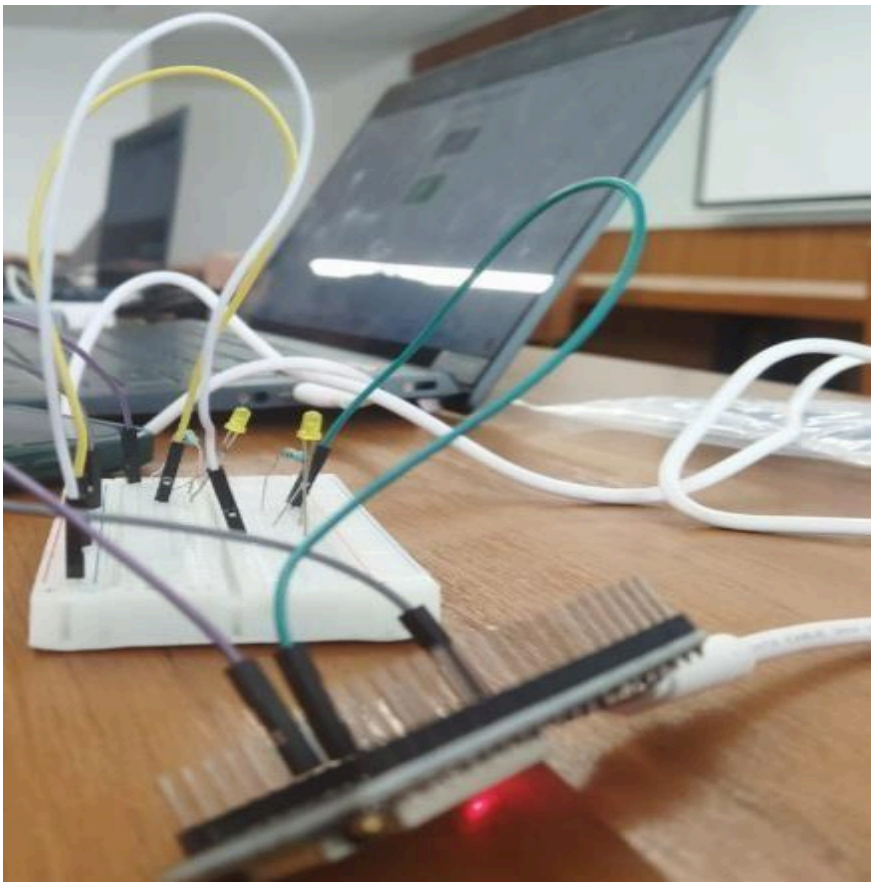
Conclusion:

In conclusion, the experiment successfully demonstrated the interfacing of GSM and Bluetooth modules with an Arduino/ESP board. The GSM module was effectively used for SMS communication, and the Bluetooth module enabled seamless wireless data exchange with a Bluetooth-enabled device. This experiment provided practical experience in serial communication, reinforced the importance of correct wiring and voltage levels, and deepened understanding of integrating wireless communication modules with microcontroller boards.

Part A
Class B Tech CSE 4th Year **Sub: Internet of Things Lab**
Aim: *M2M communication using WiFi*: Communication between two nodeMCU (ESP8266-based)/ ESP32 microcontroller board.
Prerequisite: Basics of programming, microcontrollers and basic electronics
Outcome: 1. Study and work of WiFi. 2. Connecting microcontroller board with mobile hotspot. *3.* Establishing a connection using WiFi and exchange of messages between two devices *(Client-Server architecture)*
Theory: **1. Study of WiFi and M2M communication.** **2. Connect with the WiFi of your mobile phone and control the LED on/off** **3. Connection and communication between two nodeMCU/ESP32 microcontroller boards.**

Part B (Write for an individual)
Steps: 1. Set up the first NodeMCU/ESP32 as the WiFi Access Point (AP) by configuring it in AP mode and assigning it an SSID and password. 2. Configure the second NodeMCU/ESP32 to connect to the first board's WiFi network by setting it up in Station mode and providing the SSID and password of the first board. 3. Connect both boards to their respective power supplies. 4. Write and upload code to the first NodeMCU/ESP32 to send data over WiFi to the second board using TCP or UDP protocols. 5. Write and upload code to the second NodeMCU/ESP32 to receive the data from the first board and process it accordingly. 6. Power both boards, and observe the communication between them, ensuring that the data sent from the first board is correctly received by the second.
Output:





Observation & Learning:

The first NodeMCU/ESP32 successfully created a WiFi network as an Access Point, and the second NodeMCU/ESP32 connected to this network without issues. Data transmission

between the two boards was achieved using TCP/UDP protocols, with the second board accurately receiving and processing the data sent by the first.

Through this experiment, you learned how to set up WiFi-based M2M communication between two microcontroller boards. You gained practical experience in configuring one board as an Access Point and the other as a Station, as well as in implementing and troubleshooting TCP/UDP data transmission. This reinforced your understanding of wireless communication protocols and their application in IoT systems.

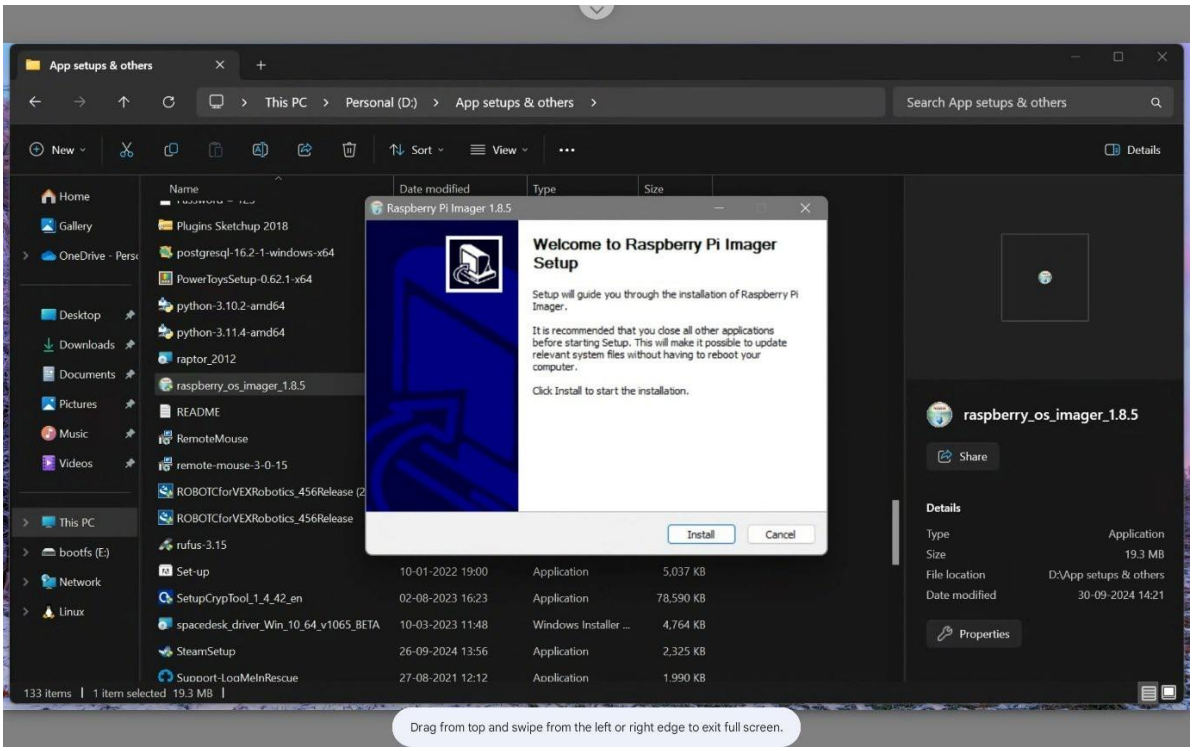
Conclusion:

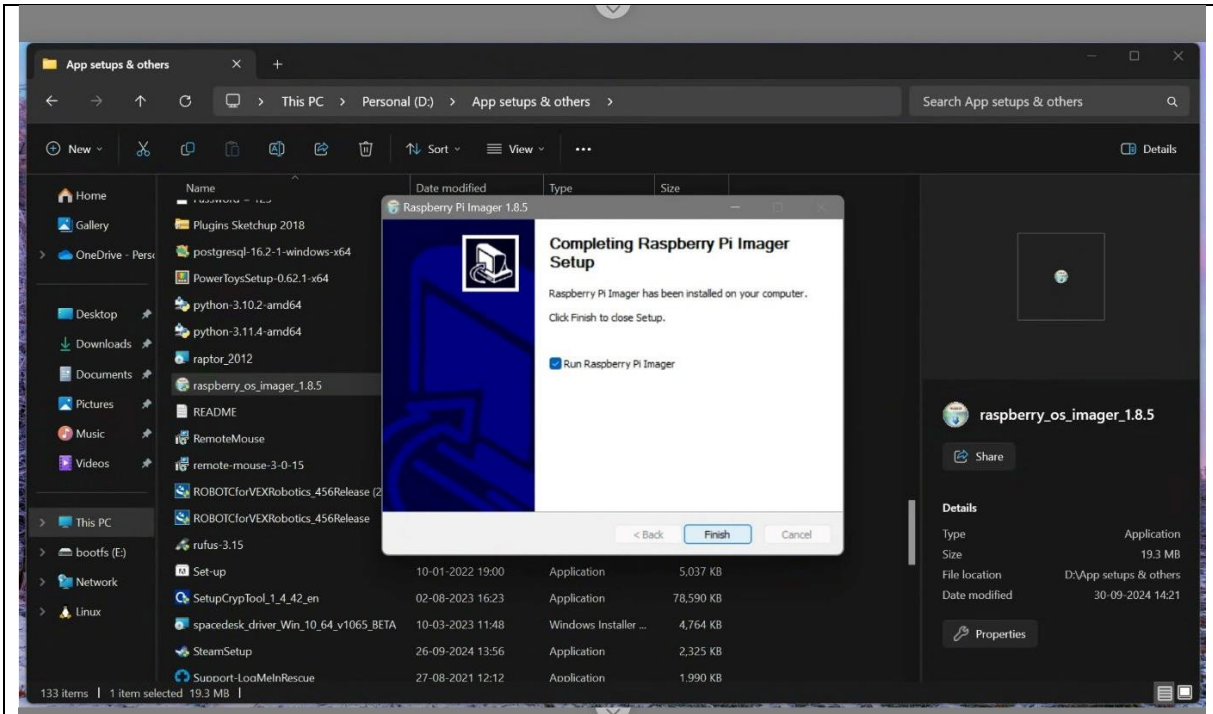
In conclusion, the experiment successfully demonstrated M2M communication using WiFi between two NodeMCU/ESP32 boards. The first board effectively functioned as a WiFi Access Point, while the second board connected and communicated with it. The accurate data transmission between the boards highlighted the practical application of TCP/UDP protocols in wireless communication. This experiment enhanced your understanding of configuring microcontrollers for WiFi-based communication and the principles underlying IoT systems.

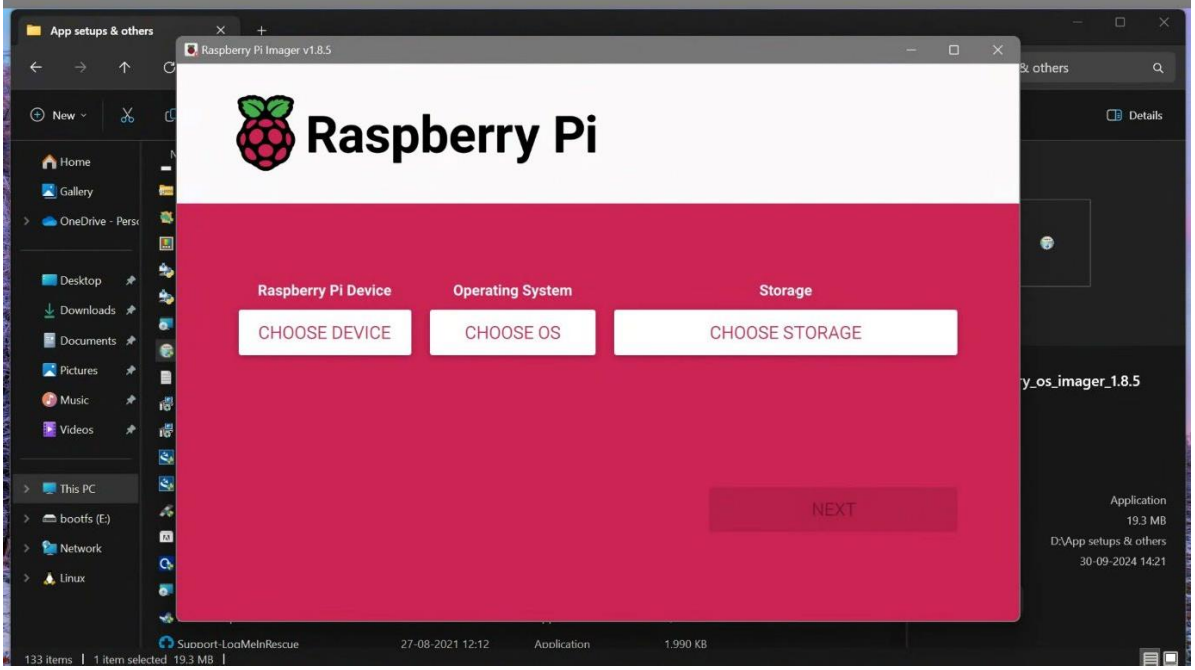
Part A
Class B Tech CSE 4th Year **Sub: Internet of Things Lab**
Aim: Raspberry Pi: To study the hardware features of the Raspberry Pi board and to install and set up the Raspbian OS (now known as Raspberry Pi OS) for initial configuration and operation.
Prerequisite: Basic Electronics
Outcome: Hands-on experience with Raspberry Pi hardware and will be capable of installing and configuring the Raspbian OS for various projects.

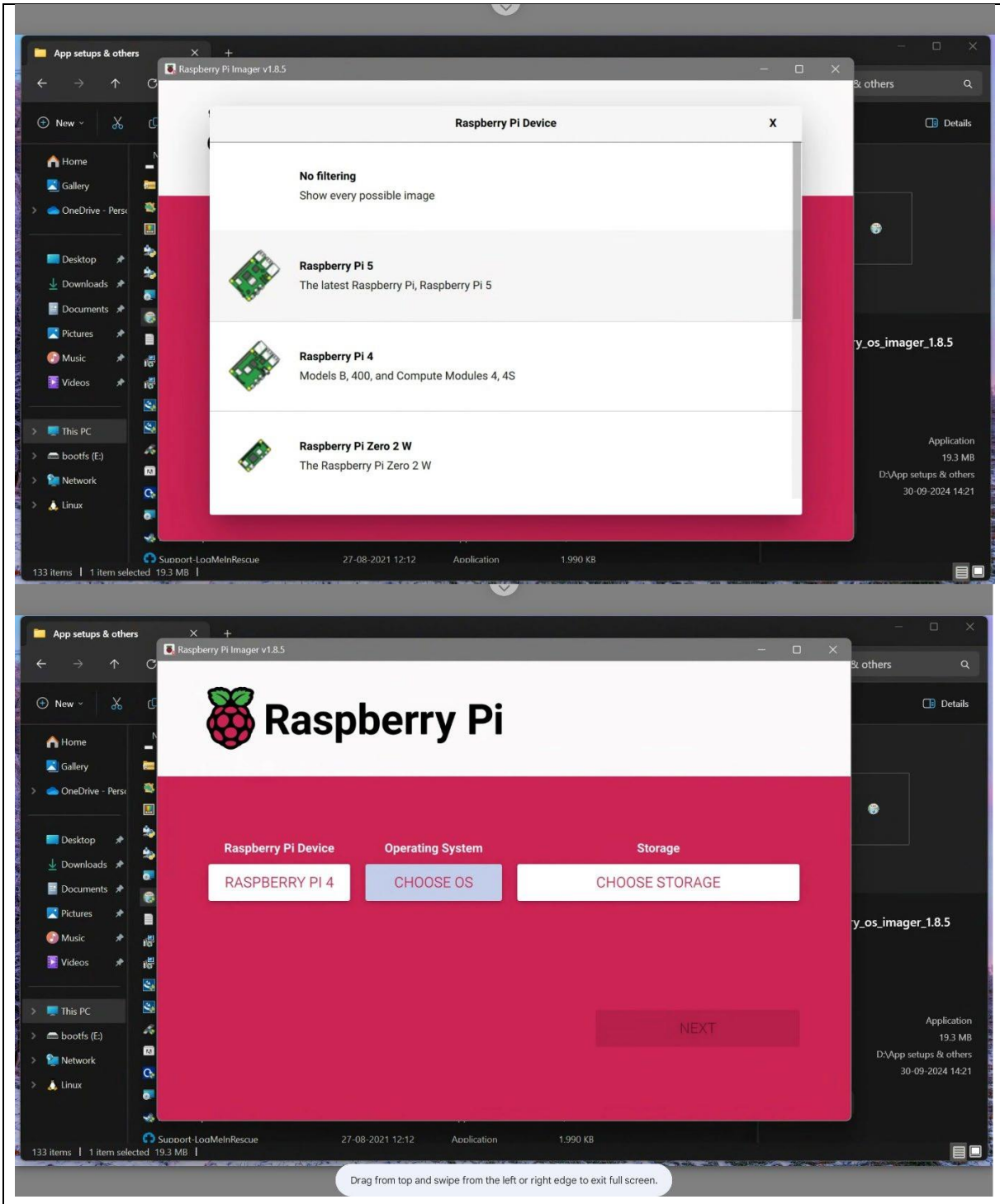
Part B

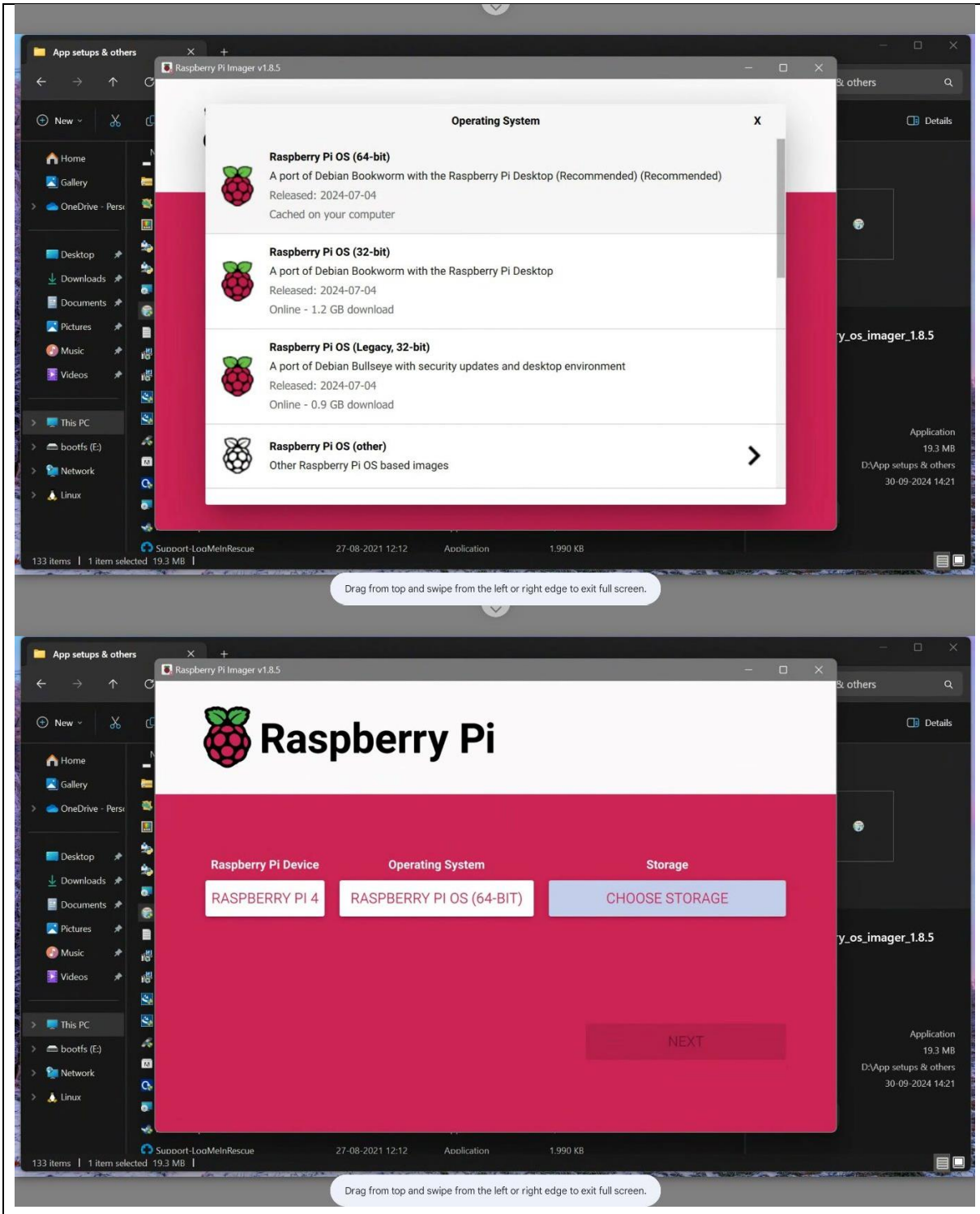
Steps:

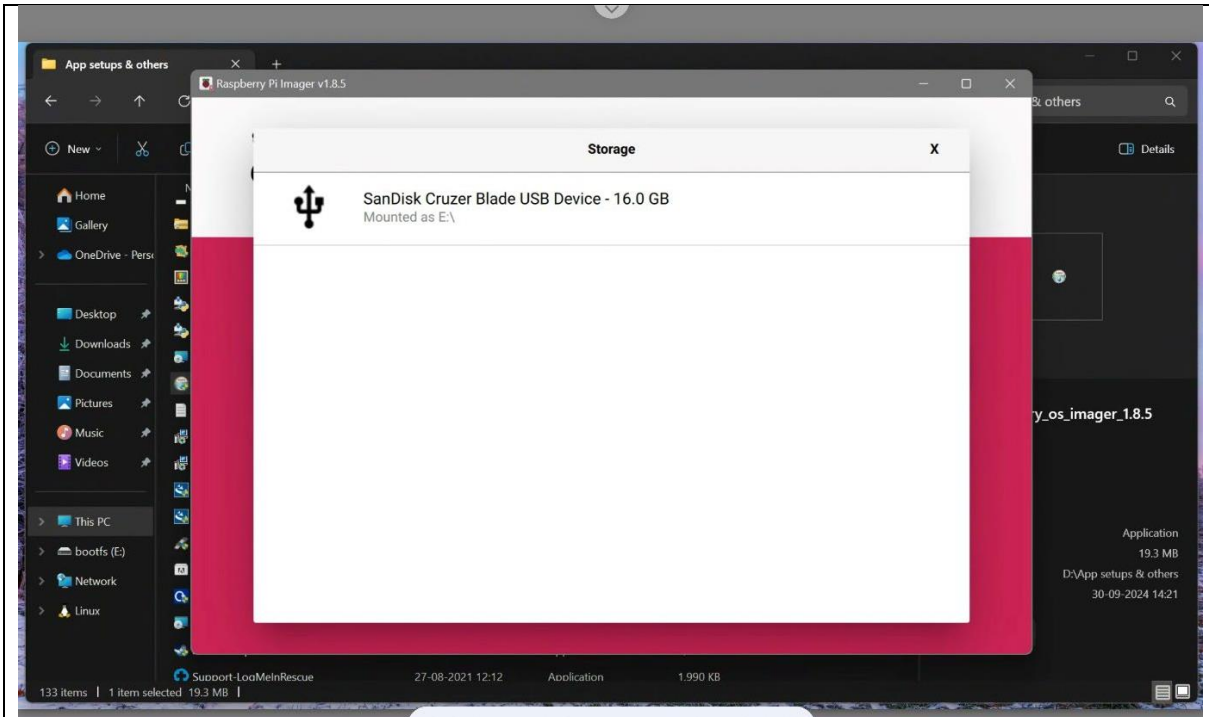


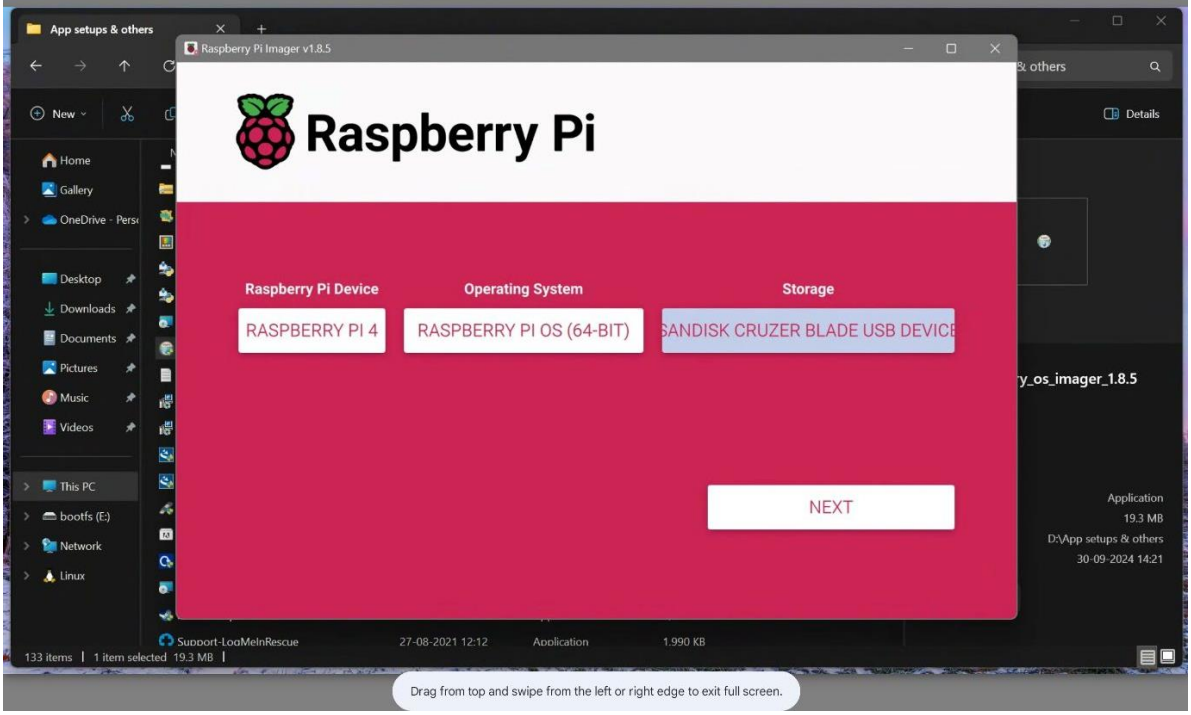

App setups & others
This PC › Personal (D:) › App setups & others ›
Search App setups & others
New
Sort
View
Details
Home
Gallery
OneDrive - Perso
Desktop
Downloads
Documents
Pictures
Music
Videos
This PC
bootfs (E:)
Network
Linux
Name
Date modified
Type
Size
Plugins Sketchup 2018
postgresql-16.2-1-windows-x64
PowerToysSetup-0.62.1-x64
python-3.10.2-amd64
python-3.11.4-amd64
raptor_2012
raspberry_os_imager_1.8.5
README
RemoteMouse
remote-mouse-3-0-15
ROBOTCforVEXRobotics_456Release
rufus-3.15
Set-up
SetupCrypTool_1_4_42_en
spacedesk_driver_Win_10_64_v1065_BETA
SteamSetup
Support-LogMeInRescue
10-01-2022 19:00 Application 5,037 KB
02-08-2023 16:23 Application 78,590 KB
10-03-2023 11:48 Windows Installer ... 4,764 KB
26-09-2024 13:56 Application 2,325 KB
27-08-2021 12:12 Application 1,990 KB
133 items | 1 item selected 19.3 MB
Raspberry Pi Imager 1.8.5
Completing Raspberry Pi Imager Setup
Raspberry Pi Imager has been installed on your computer.
Click Finish to close Setup.
Run Raspberry Pi Imager
< Back
Finish
Cancel
raspberry_os_imager_1.8.5
Share
Details
Type Application
Size 19.3 MB
File location D:\App setups & others
Date modified 30-09-2024 14:21
Properties

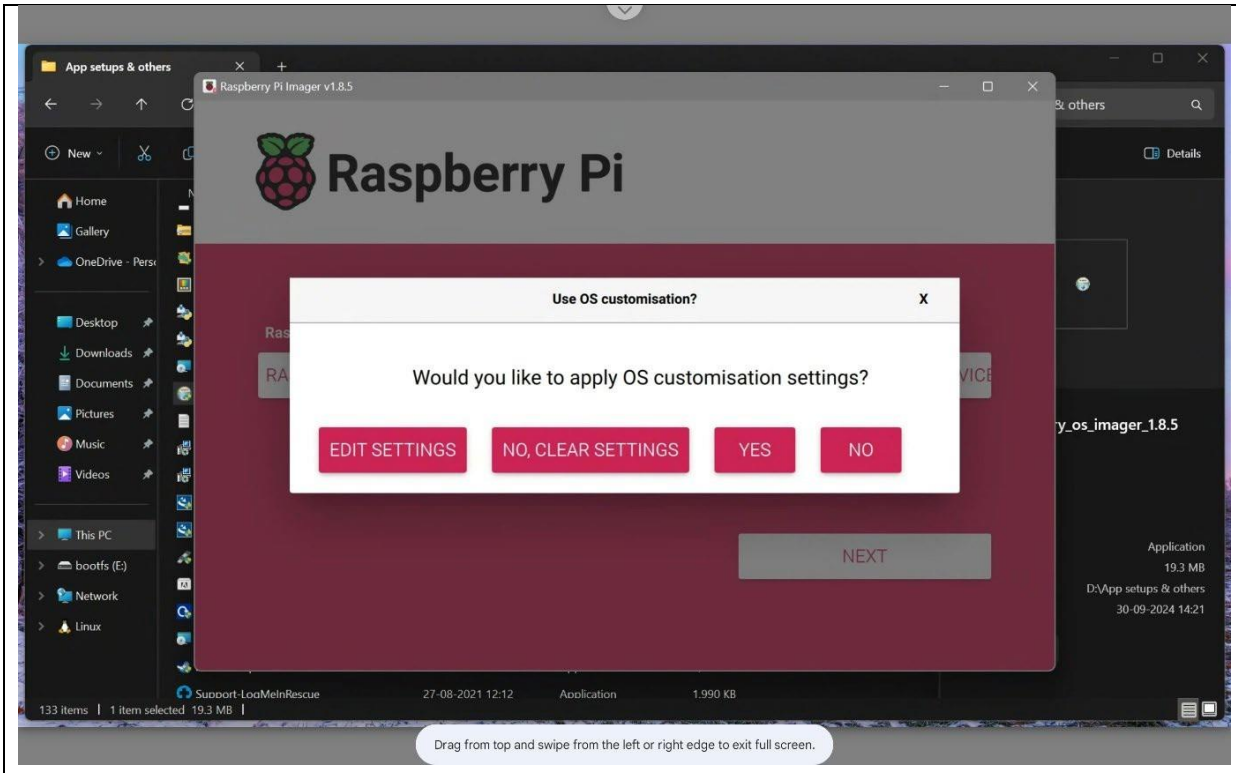

App setups & others
Raspberry Pi Imager v1.8.5
Raspberry Pi
Raspberry Pi Device
Operating System
Storage
CHOOSE DEVICE
CHOOSE OS
CHOOSE STORAGE
NEXT
Application
19.3 MB
D:\App setups & others
30-09-2024 14:21

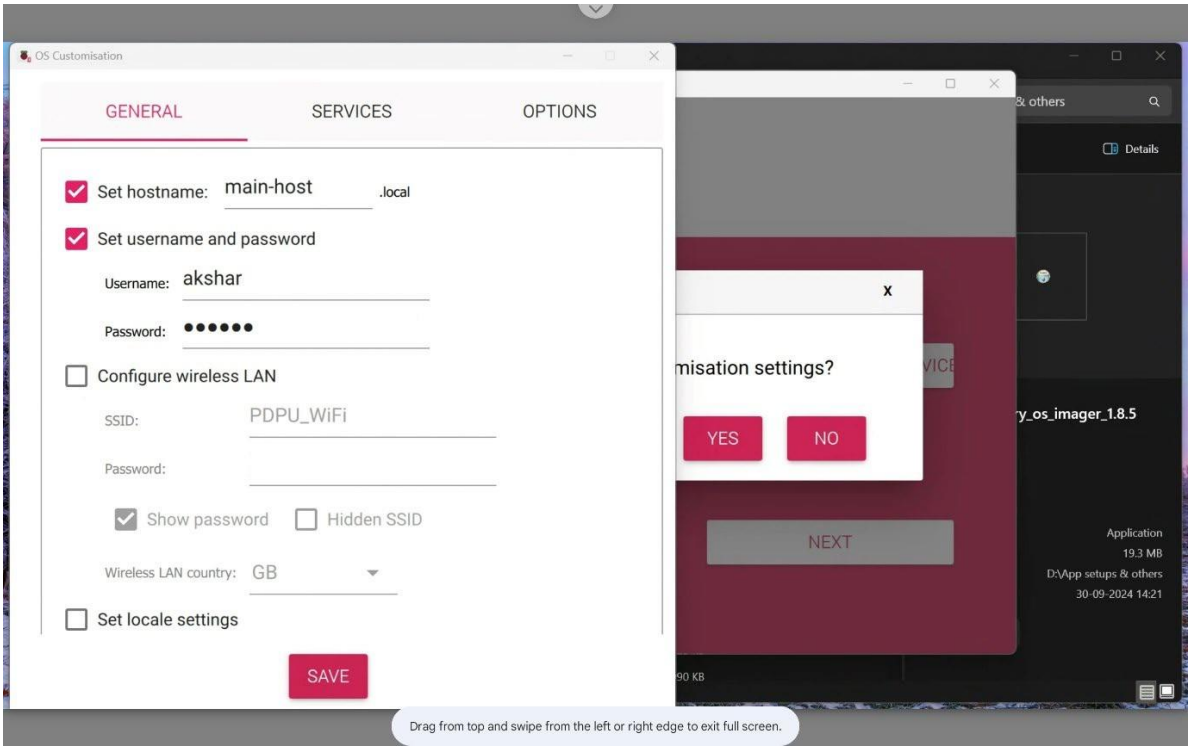

App setups & others
Raspberry Pi Imager v1.8.5
Raspberry Pi Device
X
No filtering
Show every possible image
Raspberry Pi 5
The latest Raspberry Pi, Raspberry Pi 5
Raspberry Pi 4
Models B, 400, and Compute Modules 4, 4S
Raspberry Pi Zero 2 W
The Raspberry Pi Zero 2 W
Raspberry Pi
Raspberry Pi Device
Operating System
Storage
RASPBERRY PI 4
CHOOSE OS
CHOOSE STORAGE
NEXT
Drag from top and swipe from the left or right edge to exit full screen.

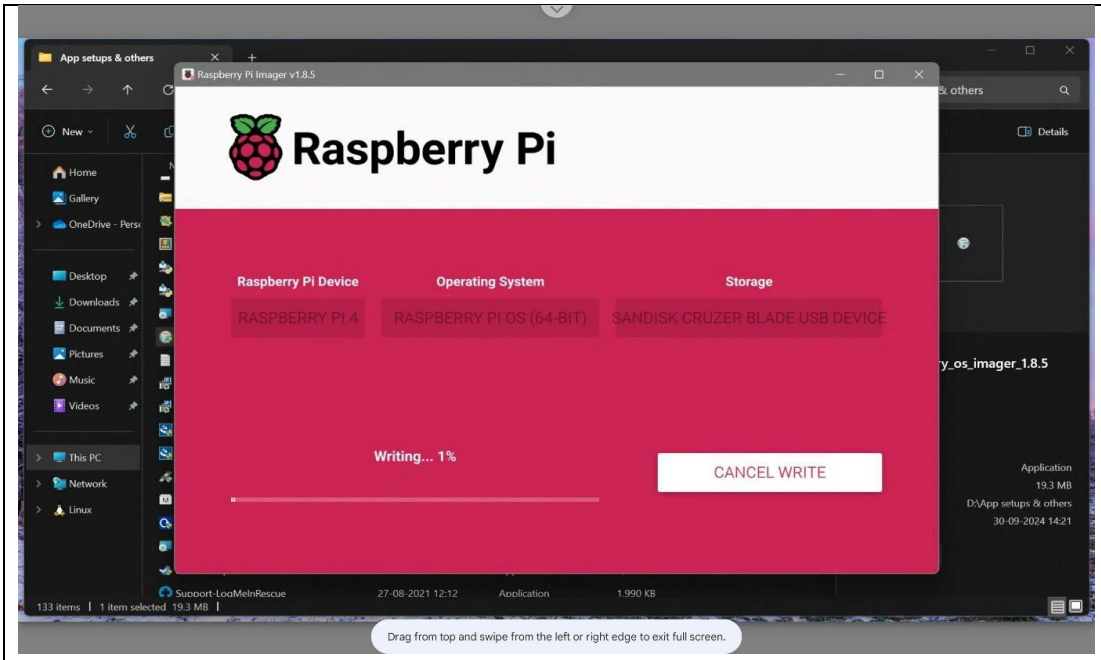

App setups & others
Raspberry Pi Imager v1.8.5
Operating System
Raspberry Pi OS (64-bit)
A port of Debian Bookworm with the Raspberry Pi Desktop (Recommended) (Recommended)
Released: 2024-07-04
Cached on your computer
Raspberry Pi OS (32-bit)
A port of Debian Bookworm with the Raspberry Pi Desktop
Released: 2024-07-04
Online - 1.2 GB download
Raspberry Pi OS (Legacy, 32-bit)
A port of Debian Bullseye with security updates and desktop environment
Released: 2024-07-04
Online - 0.9 GB download
Raspberry Pi OS (other)
Other Raspberry Pi OS based images
Drag from top and swipe from the left or right edge to exit full screen.
Raspberry Pi
Raspberry Pi Device
Operating System
Storage
RASPBERRY PI 4
RASPBERRY PI OS (64-BIT)
CHOOSE STORAGE
NEXT
Drag from top and swipe from the left or right edge to exit full screen.

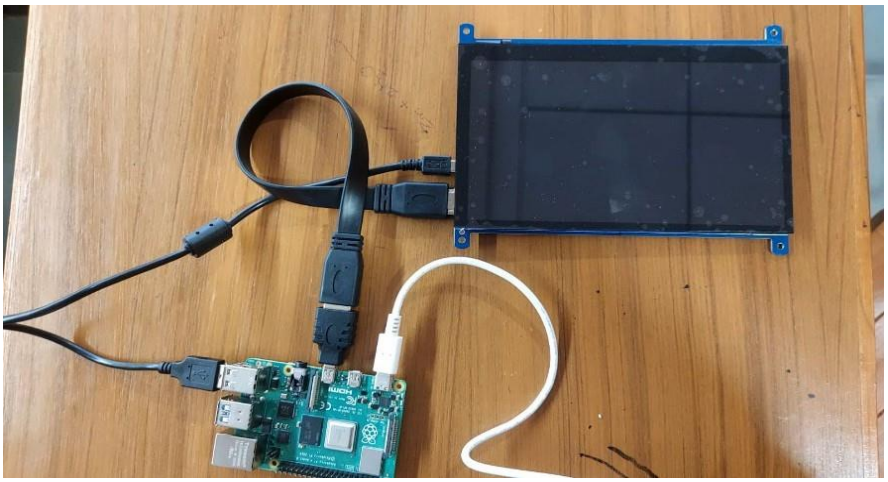

App setups & others
Raspberry Pi Imager v1.8.5
Storage
X
SanDisk Cruzer Blade USB Device - 16.0 GB
Mounted as E:\
Drag from top and swipe from the left or right edge to exit full screen.

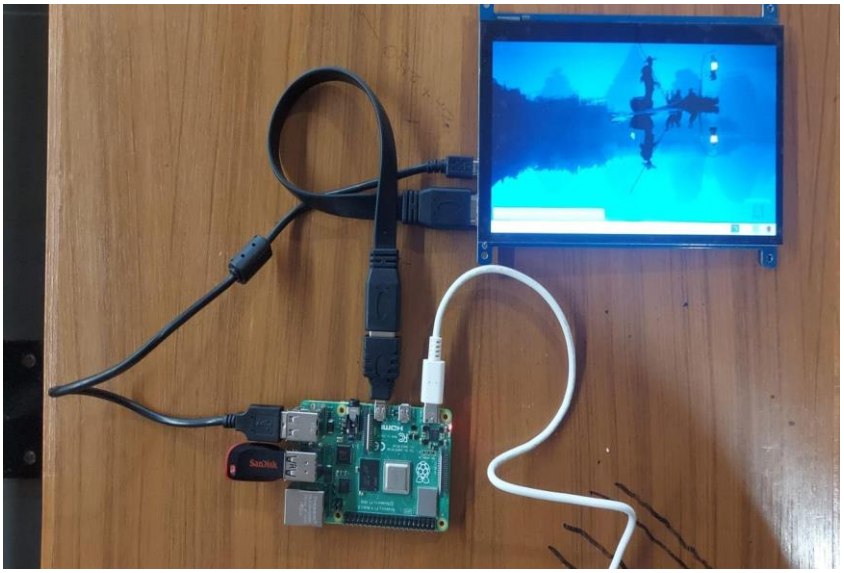

App setups & others
Raspberry Pi Imager v1.8.5
Raspberry Pi
Raspberry Pi Device
Operating System
Storage
RASPBERRY PI 4
RASPBERRY PI OS (64-BIT)
SANDISK CRUZER BLADE USB DEVICE
NEXT
Drag from top and swipe from the left or right edge to exit full screen.

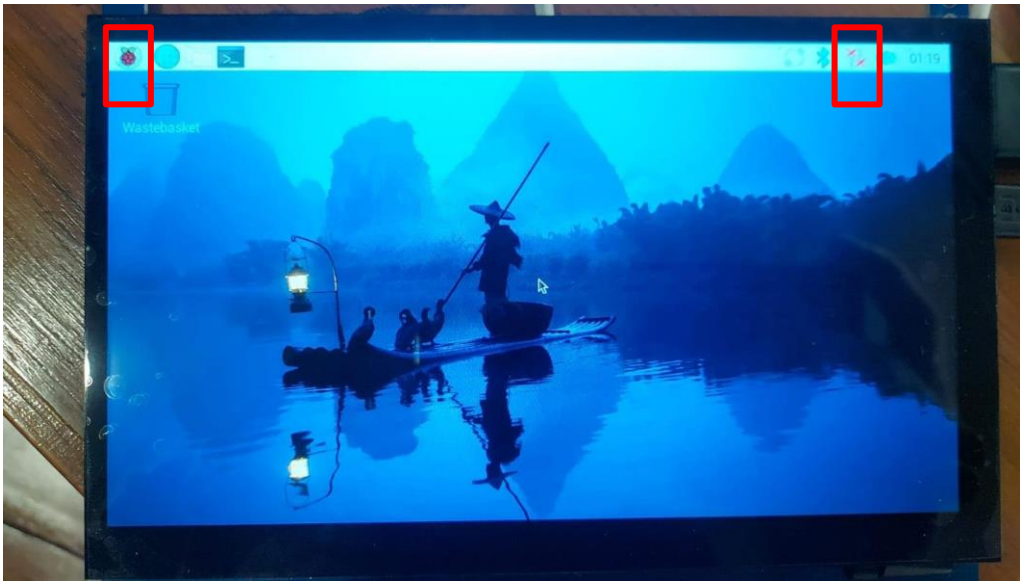

App setups & others
Raspberry Pi Imager v1.8.5
Raspberry Pi
Use OS customisation?
Would you like to apply OS customisation settings?
EDIT SETTINGS
NO, CLEAR SETTINGS
YES
NO
NEXT
Drag from top and swipe from the left or right edge to exit full screen.

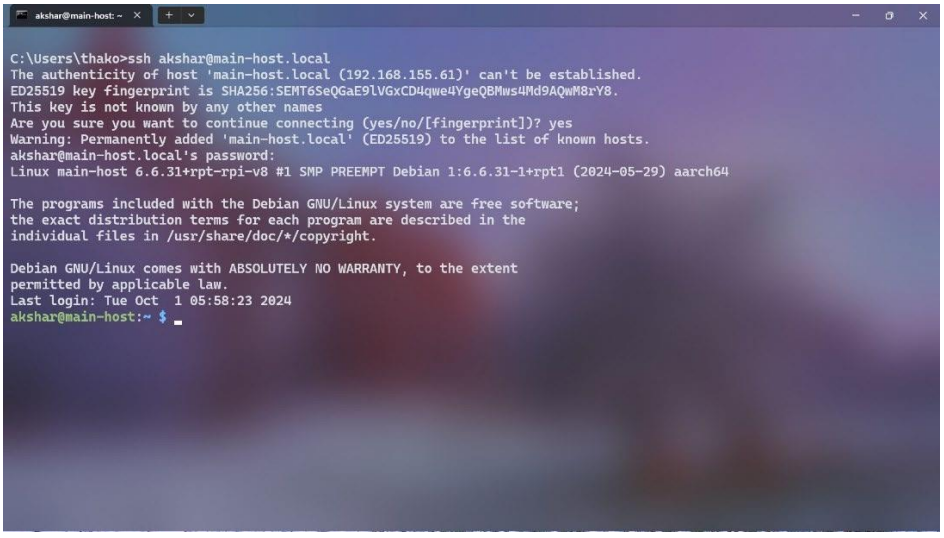

OS Customisation
GENERAL
SERVICES
OPTIONS
Set hostname: main-host .local
Set username and password
Username: akshar
Password: ••••••
Configure wireless LAN
SSID: PDPU_WiFi
Password:
Show password
Hidden SSID
Wireless LAN country: GB
Set locale settings
SAVE
misation settings?
YES
NO
NEXT
Drag from top and swipe from the left or right edge to exit full screen.

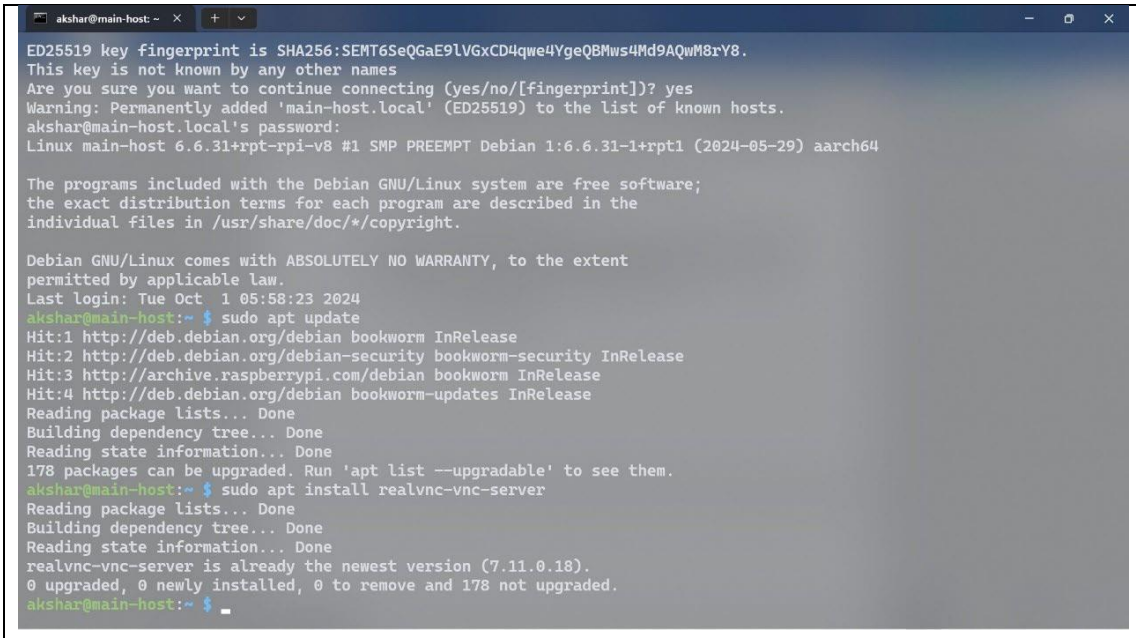

App setups & others
Raspberry Pi Imager v1.8.5
Raspberry Pi
Raspberry Pi Device
Operating System
Storage
RASPBERRY PI 4
RASPBERRY PI OS (64-BIT)
SANDISK CRUZER BLADE USB DEVICE
Writing... 1%
CANCEL WRITE
Drag from top and swipe from the left or right edge to exit full screen.

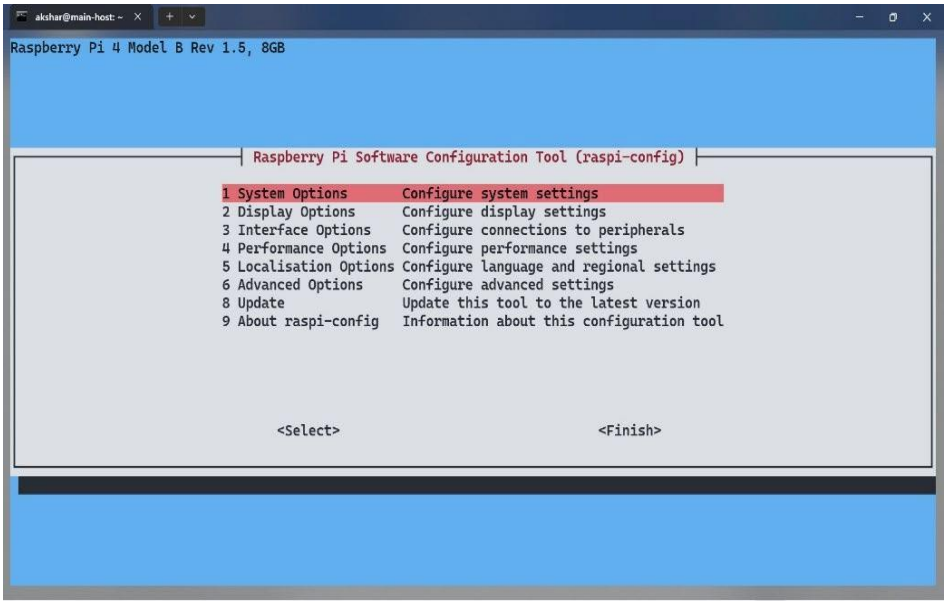




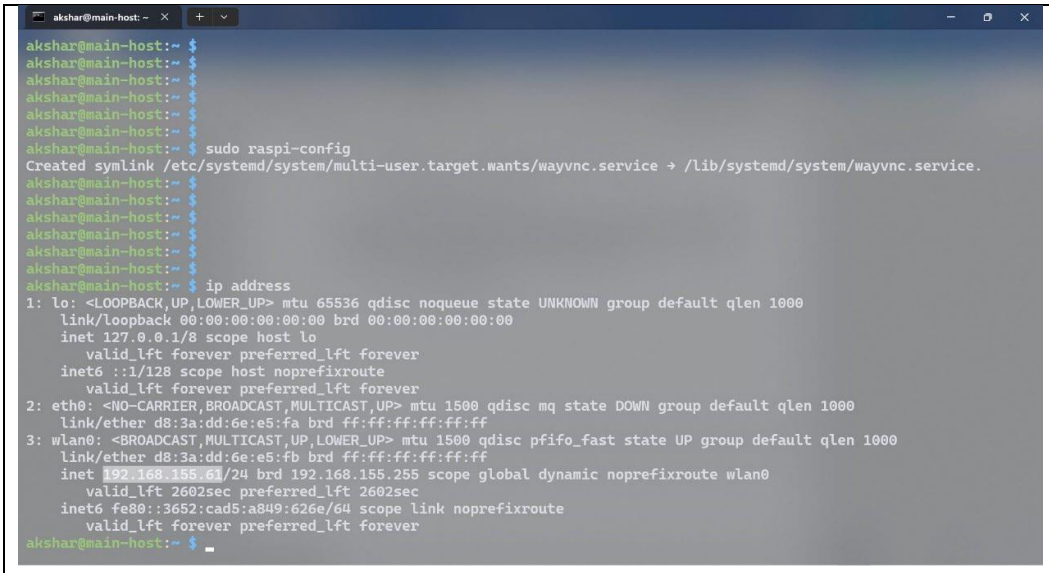

Wastebasket

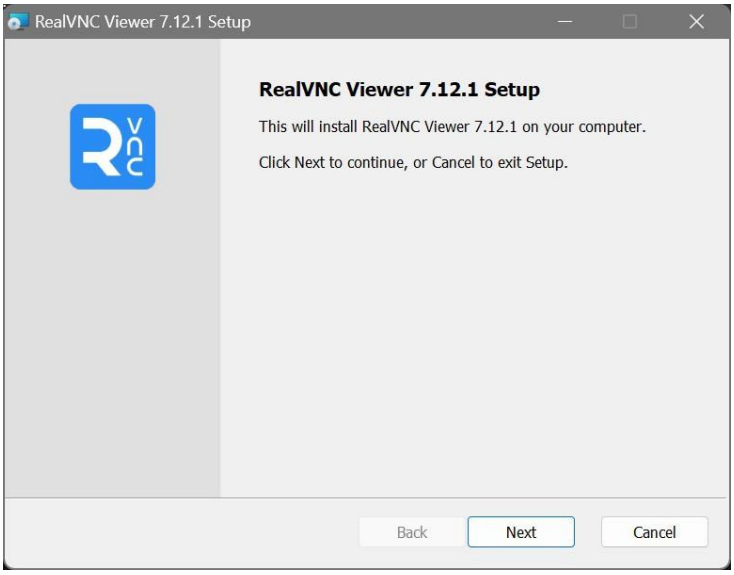

akshar@main-host: ~
C:\Users\thako>ssh akshar@main-host.local
The authenticity of host 'main-host.local (192.168.155.61)' can't be established.
ED25519 key fingerprint is SHA256:SEMT6SeQGaE9lVGxCD4qwe4YgeQBMws4Md9AQwM8rY8.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'main-host.local' (ED25519) to the list of known hosts.
akshar@main-host.local's password:
Linux main-host 6.6.31+rpt-rpi-v8 #1 SMP PREEMPT Debian 1:6.6.31-1+rpt1 (2024-05-29) aarch64
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Oct 1 05:58:23 2024
akshar@main-host:~ $

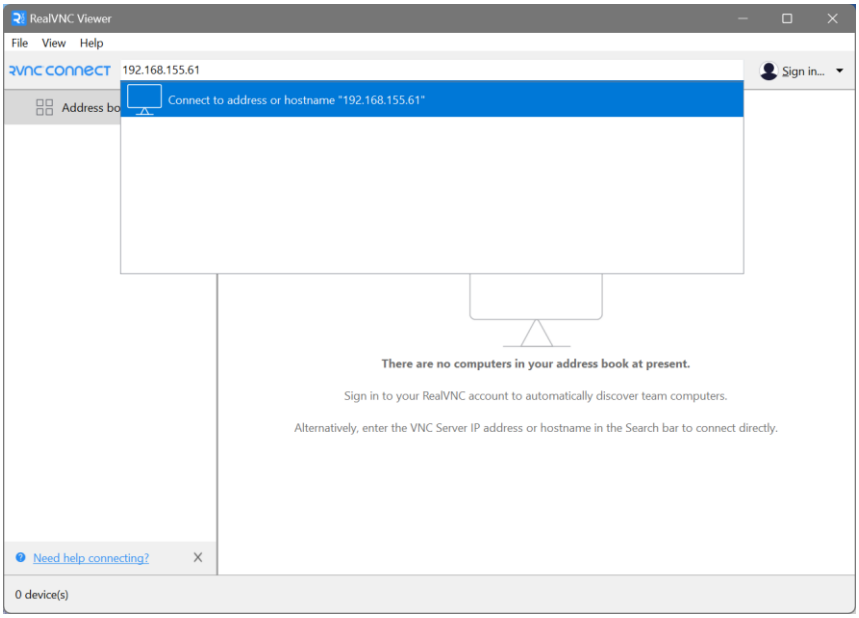

akshar@main-host: ~
ED25519 key fingerprint is SHA256:SEMT6SeQGaE9lVGxCD4qwe4YgeQBMws4Md9AQwM8rY8.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'main-host.local' (ED25519) to the list of known hosts.
akshar@main-host.local's password:
Linux main-host 6.6.31+rpt-rpi-v8 #1 SMP PREEMPT Debian 1:6.6.31-1+rpt1 (2024-05-29) aarch64
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Oct 1 05:58:23 2024
akshar@main-host:~ $ sudo apt update
Hit:1 http://deb.debian.org/debian bookworm InRelease
Hit:2 http://deb.debian.org/debian-security bookworm-security InRelease
Hit:3 http://archive.raspberrypi.com/debian bookworm InRelease
Hit:4 http://deb.debian.org/debian bookworm-updates InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
178 packages can be upgraded. Run 'apt list --upgradable' to see them.
akshar@main-host:~ $ sudo apt install realvnc-vnc-server
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
realvnc-vnc-server is already the newest version (7.11.0.18).
0 upgraded, 0 newly installed, 0 to remove and 178 not upgraded.
akshar@main-host:~ $


akshar@main-host: ~
Raspberry Pi 4 Model B Rev 1.5, 8GB
Raspberry Pi Software Configuration Tool (raspi-config)
1 System Options Configure system settings
2 Display Options Configure display settings
3 Interface Options Configure connections to peripherals
4 Performance Options Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options Configure advanced settings
8 Update Update this tool to the latest version
9 About raspi-config Information about this configuration tool
<Select>
<Finish>

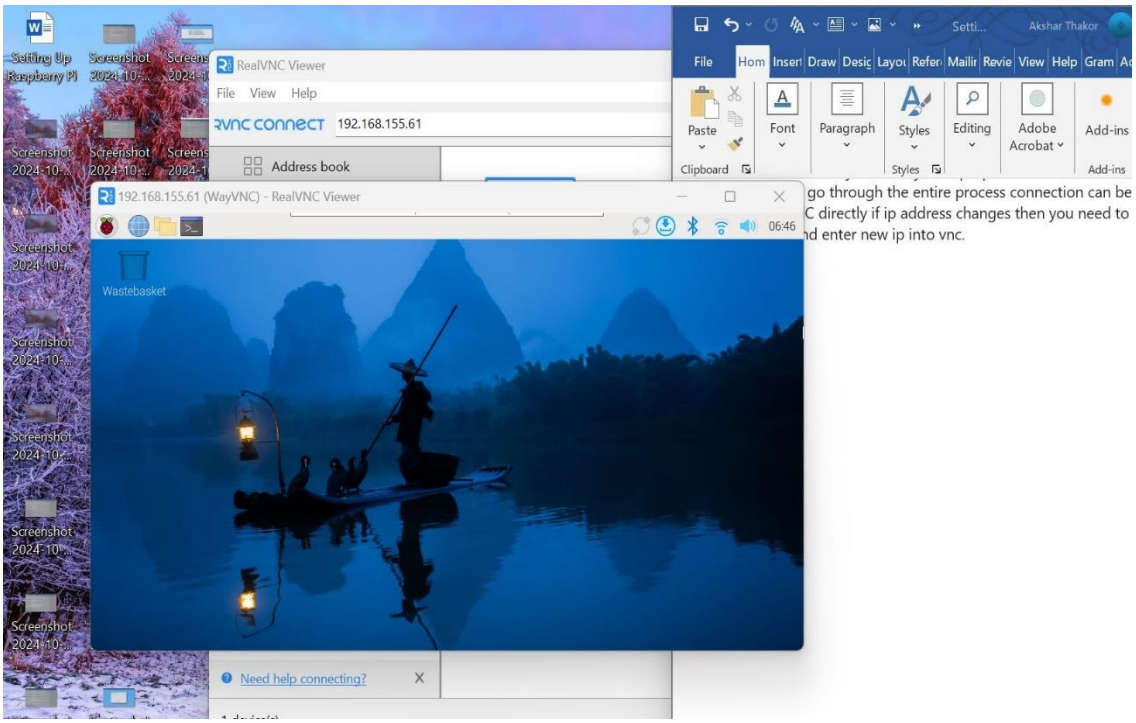

akshar@main-host: ~
The VNC Server is enabled
<Ok>


akshar@main-host: ~
akshar@main-host:~ $
akshar@main-host:~ $
akshar@main-host:~ $
akshar@main-host:~ $
akshar@main-host:~ $
akshar@main-host:~ $
akshar@main-host:~ $ sudo raspi-config
Created symlink /etc/systemd/system/multi-user.target.wants/wayvnc.service → /lib/systemd/system/wayvnc.service.
akshar@main-host:~ $
akshar@main-host:~ $
akshar@main-host:~ $
akshar@main-host:~ $
akshar@main-host:~ $
akshar@main-host:~ $
akshar@main-host:~ $ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host noprefixroute
valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group default qlen 1000
link/ether d8:3a:dd:6e:e5:fa brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
link/ether d8:3a:dd:6e:e5:fb brd ff:ff:ff:ff:ff:ff
inet 192.168.155.61/24 brd 192.168.155.255 scope global dynamic noprefixroute wlan0
valid_lft 2602sec preferred_lft 2602sec
inet6 fe80::3652:cad5:a849:626e/64 scope link noprefixroute
valid_lft forever preferred_lft forever
akshar@main-host:~ $


RealVNC Viewer 7.12.1 Setup
RealVNC Viewer 7.12.1 Setup
This will install RealVNC Viewer 7.12.1 on your computer.
Click Next to continue, or Cancel to exit Setup.
Back
Next
Cancel


RealVNC Viewer
File View Help
VNC CONNECT
192.168.155.61
Sign in...
Address bo
Connect to address or hostname "192.168.155.61"
There are no computers in your address book at present.
Sign in to your RealVNC account to automatically discover team computers.
Alternatively, enter the VNC Server IP address or hostname in the Search bar to connect directly.
Need help connecting?
0 device(s)

Output:



Observation & Learning:

The Raspberry Pi board includes various I/O ports, including USB, HDMI, and GPIO pins. Raspbian OS installation was straightforward, involving downloading, writing to a microSD card, and configuring initial settings.

Conclusion:

The practical session provided essential hands-on experience with Raspberry Pi hardware and OS setup. After completing the installation, users are equipped to configure Raspbian OS and utilize the board for diverse projects.

PRACTICAL 7

AIM: LCD will be used with Arduino / ESP32 with various sensors.

PREREQUISITE: Basics of programming, microcontrollers and basic electronics.

OUTCOME: 1. Study and work of LCD

2. Connecting microcontroller board with LCD
3. Display of sensor data over the 16X2 LCD.

PART 1

THEORY:

1. **Study and Work of LCD:**

   An LCD (Liquid Crystal Display) is a flat-panel display that uses liquid crystals to produce visible images. It works by modulating the light through liquid crystals sandwiched between polarized glass. LCDs are commonly used in electronic devices for displaying text, numbers, and graphics. A typical 16x2 LCD has 16 columns and 2 rows, allowing it to display up to 32 characters simultaneously.

2. **Connection of 16x2 LCD with Arduino and ESP32 Microcontroller Boards:**

   To connect a 16x2 LCD with Arduino or ESP32, the LCD requires connections to data pins (D4-D7 for 4-bit mode), RS (Register Select), E (Enable), and VSS/VDD for power. Using libraries like LiquidCrystal in Arduino and similar libraries for ESP32, you can easily control the LCD and send characters or messages to display. Both microcontrollers can interface with the LCD through digital pins and use serial communication to control the display.

3. **Display the Values Sensed by Various Sensors:**

   The 16x2 LCD can display sensor data such as distance from an ultrasonic sensor, light intensity from a photosensitive resistor, or resistance values from a potentiometer. The sensor readings are processed by the microcontroller (Arduino or ESP32), which converts analog/digital sensor values into readable text, and then sends this data to the LCD for real-time display.

4. **Use a DHT Sensor to Display the Temperature and Humidity:**

   A DHT sensor (like DHT11 or DHT22) is used to measure temperature and humidity. It is connected to a microcontroller like Arduino or ESP32, and the sensor values are fetched using appropriate libraries. These values are then displayed on the 16x2 LCD. The DHT sensor outputs digital values, which are easily interpreted and shown on the screen in a human-readable format (e.g., "Temp: 25°C, Humidity: 60%").

PART 2

Ultrasonic Sensor and LCD:

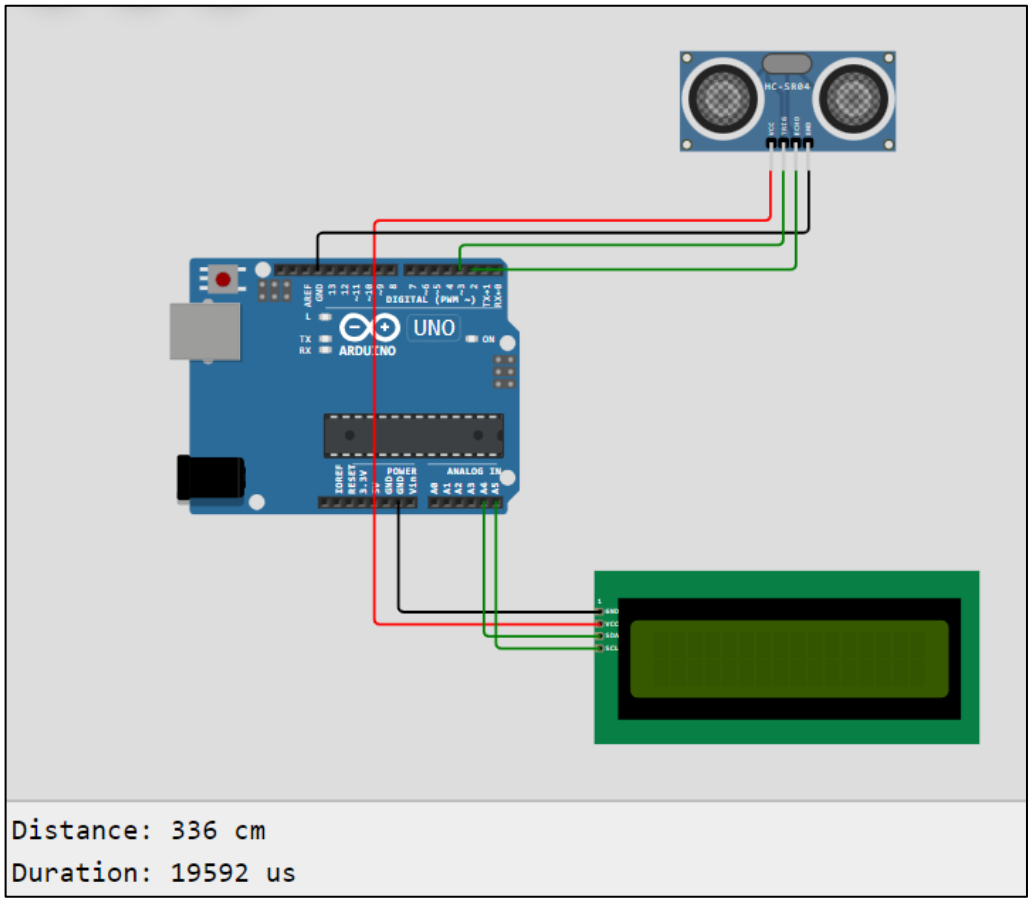
CODE:

```
#include <Wire.h>                 // Library for I2C communication
#include <LiquidCrystal_I2C.h>    // Library for I2C LCD

// Initialize the I2C LCD
LiquidCrystal_I2C lcd(0x27, 16, 2);   // Set the LCD address to 0x27 (may vary) for a 16 chars and 2 line display

// Define Ultrasonic Sensor Pins
const int trigPin = 3;
const int echoPin = 2;

long duration;
int distance;

void setup() {
  // Set up the ultrasonic sensor pins
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
```

```
  // Initialize LCD
  lcd.init();
  lcd.backlight();

  // Begin Serial Communication for debugging
  Serial.begin(9600);
}

void loop() {
  // Clear the trigger pin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  // Send a 10us pulse to trigger the ultrasonic sensor
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Read the echo pin, and calculate the duration in microseconds
  duration = pulseIn(echoPin, HIGH);

  // Calculate the distance in centimeters
  // Speed of sound is 343 m/s or 0.0343 cm/us
  distance = duration * 0.0343 / 2;

  // Print distance and duration on the LCD
  lcd.clear();
  lcd.setCursor(0, 0);   // Set cursor to the first line
  lcd.print("Dist: ");
  lcd.print(distance);
```

```
  lcd.print(" cm");

  lcd.setCursor(0, 1);   // Set cursor to the second line
  lcd.print("Dur: ");
  lcd.print(duration);
  lcd.print(" us");

  delay(3000);  // Delay before next reading
}
```

OUTPUT:



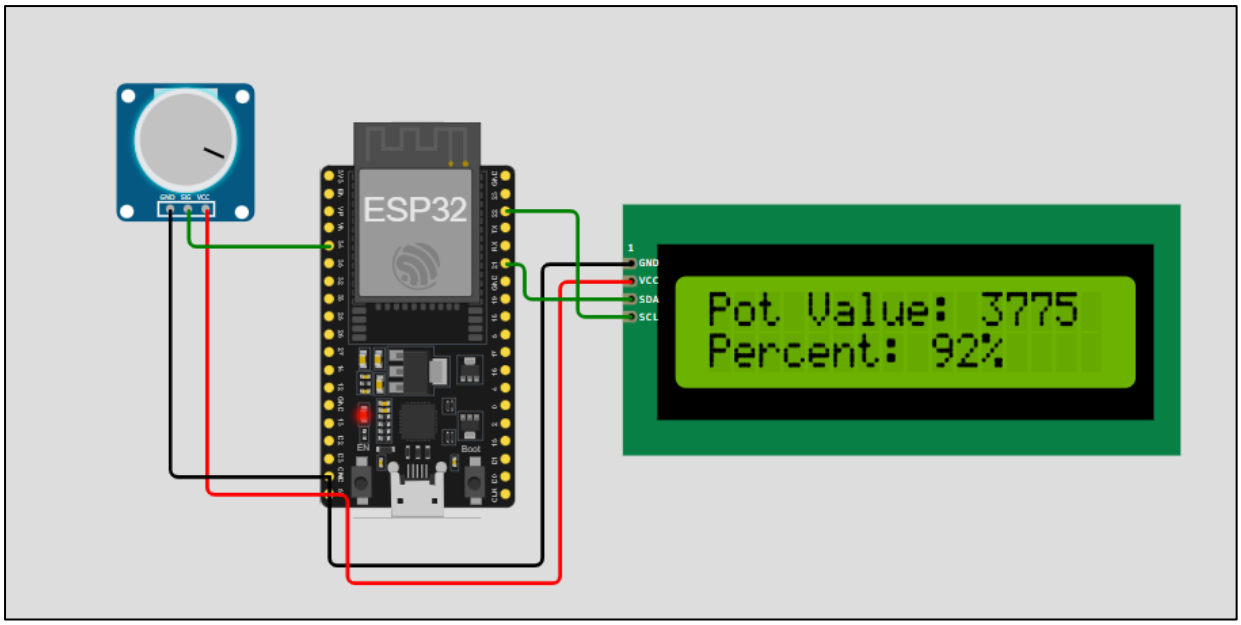
ESP32 and Potentiometer:

CODE:

```
#include <Wire.h>                 // Library for I2C communication
#include <LiquidCrystal_I2C.h>    // Library for I2C LCD

// Initialize the I2C LCD
LiquidCrystal_I2C lcd(0x27, 16, 2);   // Set the LCD address to 0x27 (adjust if needed)

// Define the pin for the potentiometer
#define POT_PIN 34                // Potentiometer connected to GPIO 34 (analog input on ESP32)

void setup() {
  // Initialize LCD
  lcd.init();
  lcd.backlight();

  // Display a welcome message on the LCD
  lcd.setCursor(0, 0);
  lcd.print("Potentiometer");
  lcd.setCursor(0, 1);
  lcd.print("Value:");
  delay(2000);                    // Wait 2 seconds before starting readings
  lcd.clear();
}

void loop() {
  // Read the potentiometer value (0 to 4095 for ESP32's 12-bit ADC)
  int potValue = analogRead(POT_PIN);
```

```
  // Map the potentiometer value to a percentage (0 to 100)
  int potPercent = map(potValue, 0, 4095, 0, 100);

  // Print potentiometer value on the LCD
  lcd.setCursor(0, 0);            // Set cursor to the first line
  lcd.print("Pot Value: ");
  lcd.print(potValue);

  lcd.setCursor(0, 1);            // Set cursor to the second line
  lcd.print("Percent: ");
  lcd.print(potPercent);
  lcd.print("%");

  delay(500);                     // Wait for 0.5 seconds before taking the next reading
}
```

OUTPUT:



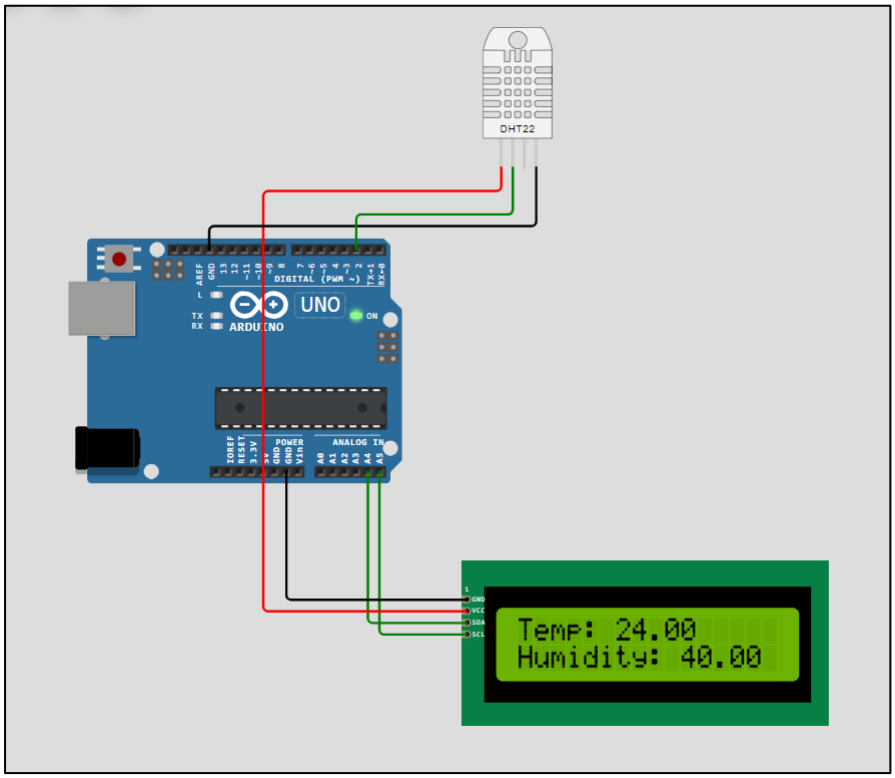
DHT and Arduino:

CODE:

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

LiquidCrystal_I2C lcd(0x27, 16, 2);

#define DHTPIN 2
#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

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

  dht.begin();

  lcd.setCursor(0, 0);
  lcd.print("Temp & Humidity");
  delay(2000);
  lcd.clear();
}

void loop() {
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();
  if (isnan(humidity) || isnan(temperature)) {
```

```
    lcd.setCursor(0, 0);
    lcd.print("Sensor Error!");
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  lcd.setCursor(0, 0);
  lcd.print("Temp: ");
  lcd.print(temperature);

  lcd.setCursor(0, 1);
  lcd.print("Humidity: ");
  lcd.print(humidity);
  delay(2000);
}
```

OUTPUT:



Observation:

While working with a 16x2 LCD connected to Arduino and ESP32, it was observed that the microcontrollers efficiently interface with the display to show real-time data from various sensors like ultrasonic, photosensitive resistors, and potentiometers. The LCD provided a clear and responsive visual output of sensor values. The DHT sensor accurately measured and displayed the temperature and humidity on the LCD. Both Arduino and ESP32 platforms successfully handled multiple inputs and output tasks, making them ideal for IoT applications requiring real-time monitoring.

Conclusion:

The integration of a 16x2 LCD with Arduino and ESP32 demonstrates how IoT systems can effectively communicate sensor data to users in a simple visual format. The ability to display real-time sensor readings, such as environmental conditions (temperature, humidity) or proximity, is vital for many practical applications, including home automation, weather stations, and smart systems. This combination of microcontrollers, sensors, and display units offers a powerful yet accessible tool for prototyping and deploying IoT projects.

Part A

Class B Tech CSE 4th Year **Sub: Internet of Things Lab**

Aim: *Arduino IoT cloud platform to control LED and DHT sensor data display*.

Prerequisite: Basics of programming, microcontrollers and basic electronics

Outcome:

1. Understanding of IoT Cloud and various service providers.
2. Creating things and converting them into smart things over IoT Cloud.
3. Connect and control smart things with the Arduino IoT Cloud.
4. Display the data of various sensors over the Arduino IoT Cloud dashboard

Theory:

1. **Study of IoT Cloud Service Providers:** Analyze key IoT cloud platforms such as Arduino IoT Cloud, AWS, Microsoft Azure IoT, and Blynk, focusing on their functionalities and features.
    - **Arduino IoT Cloud:** A cloud platform designed for users to develop IoT projects with Arduino hardware. It provides capabilities for real-time monitoring, device management, and automation, through a simplified, user-friendly interface.
    - **Microsoft Azure IoT:** A robust cloud solution offering services for device connectivity, data analytics, security, and machine learning, supporting comprehensive management of IoT ecosystems.
2. **LED Control via Arduino IoT Cloud:**
    - **Hardware Setup:** An LED is connected to an Arduino pin, forming a basic circuit that can be controlled using a virtual switch on the Arduino IoT Cloud Dashboard.
    - **Cloud Configuration:** A device (referred to as a "thing") is configured in the Arduino IoT Cloud to control the LED. A variable is assigned to represent the LED's status (on/off) and linked to the dashboard.
    - **Programming:** The Arduino Cloud API is utilized to develop code that maps the LED control variable to the physical pin, enabling the LED to respond to cloud commands sent via the dashboard.
3. **Displaying DHT Sensor Data on the Arduino IoT Cloud Dashboard:**
    - **Cloud Setup:** Two variables (temperature and humidity) are defined in the Arduino IoT Cloud, linked to the data received from the DHT sensor.
    - **Dashboard Display:** Real-time sensor data is transmitted to the Arduino IoT Cloud and displayed on the dashboard using widgets, such as gauges or text boxes, for temperature and humidity readings.
4. **Temperature and Humidity Charting on Arduino IoT Cloud Dashboard:**
    - **Analysis and Monitoring:** A live chart is implemented to monitor temperature and humidity trends, offering valuable insights for projects requiring continuous environmental data tracking.

Part B (Write for an individual)

Steps:

Project 1: LED On/Off Using the Arduino IoT Cloud Platform

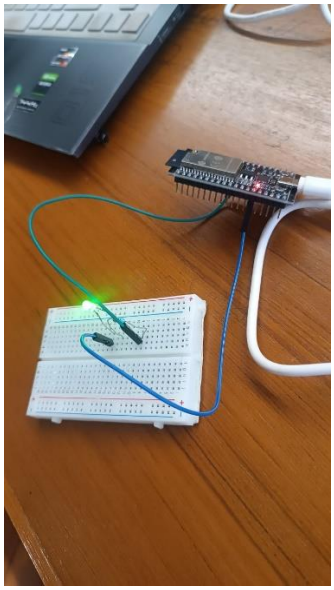
- **Setup the LED circuit**: Connect the LED to a digital pin (e.g., D2) on the Arduino board and the other end to ground (GND).
- **Create an Arduino IoT Cloud account**: Go to the Arduino IoT Cloud platform and create an account if you don't have one.
- **Create a new thing**: In the Arduino IoT Cloud, create a new "thing" and add your Arduino device (e.g., Arduino Uno, Nano, ESP32).
- **Add a variable**: Add a new boolean variable (e.g., ledStatus) that will control the LED's state (true for ON, false for OFF). This will be linked to the LED.
- **Configure a dashboard**: Create a dashboard in Arduino IoT Cloud and add a toggle switch widget that controls the ledStatus variable to turn the LED on/off.
- **Write the Code:**

```
#include "thingProperties.h"  // Include IoT properties

const int ledPin = 2;

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  initProperties();
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
}

void loop() {
  ArduinoCloud.update();  // Keep cloud connection alive

  // Control LED based on cloud variable status
 If(ledStatus){
  digitalWrite(ledPin, HIGH );
}else{
digitalWrite(ledPin, LOW);
}
}
```

- **Upload the code and control**: After uploading the code to the Arduino, open the Arduino IoT Cloud Dashboard and use the toggle switch to turn the LED on and off.

.

Output:



Observation & Learning:

1. The I2C interface simplifies the connection of the LCD with the ESP32 and Arduino, using only two wires (SDA and SCL).
2. Mapping the sensor values to readable formats is essential for presenting clear and accurate data on the LCD screen.
3. Real-time data from various sensors (potentiometer and humidity sensor) can be effectively displayed using libraries like LiquidCrystal_I2C.
4. Using cloud platforms like Arduino IoT Cloud makes it easy to control devices and visualize sensor data remotely.
5. Creating interactive dashboards with widgets (switches, gauges, charts) enhances monitoring and control of IoT devices.
6. Arduino IoT Cloud provides real-time data visualization tools, making it useful for environmental monitoring projects.

Conclusion:

The integration of IoT platforms like Arduino IoT Cloud with sensors and devices like LEDs, potentiometers, and DHT sensors offers a powerful way to create real-time, interactive projects. By using simple libraries and cloud-based dashboards, even beginners can set up

systems for data monitoring and remote control. Displaying sensor data such as temperature, humidity, and user-controlled devices on the IoT Cloud platform provides insight into the flexibility and scalability of IoT systems.

PRACTICAL 9

AIM: Raspberry Pi: Remote Access Setup and LED Control via Python Programming.

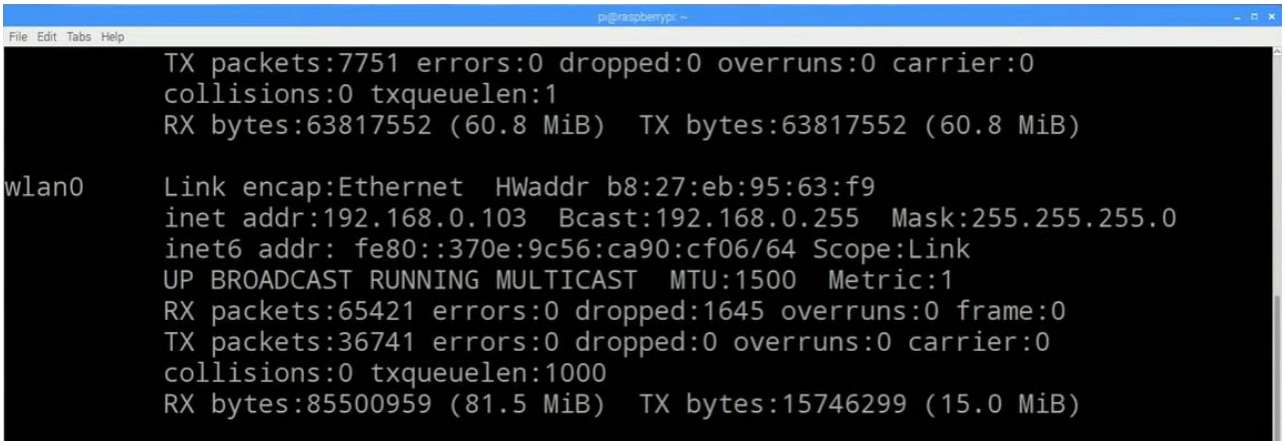
PREREQUISITE: Basics of programming, microcontrollers and basic electronics.

OUTCOME: Access the Raspberry Pi remotely using SSH and VNC, and demonstrate control of its GPIO to blink an LED using Python programming.

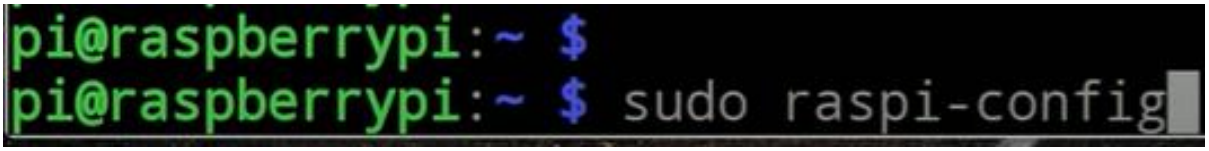
PROCEDURE:

A) For Remote Access:

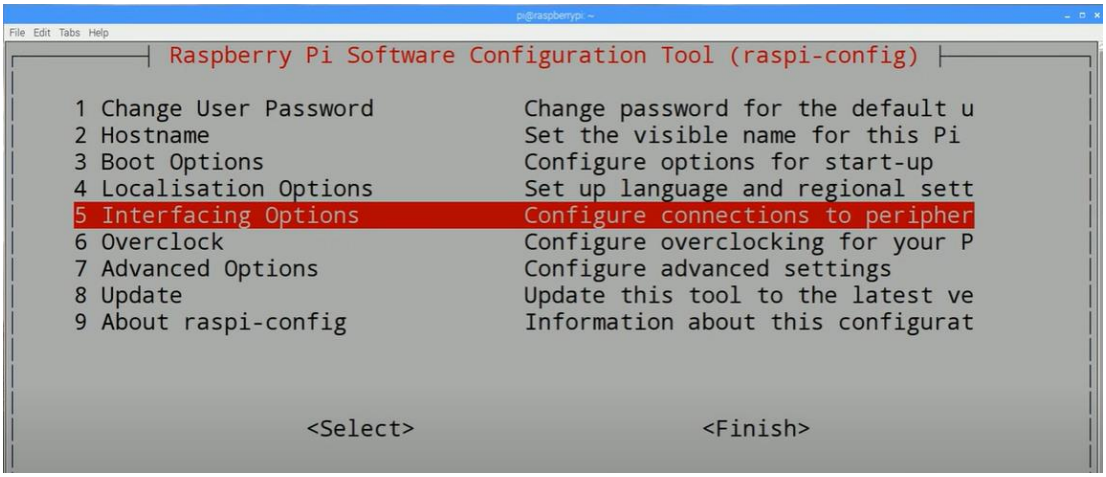
Step 1: Connect the Raspberry Pi to a Wi-Fi network and note its IP address.



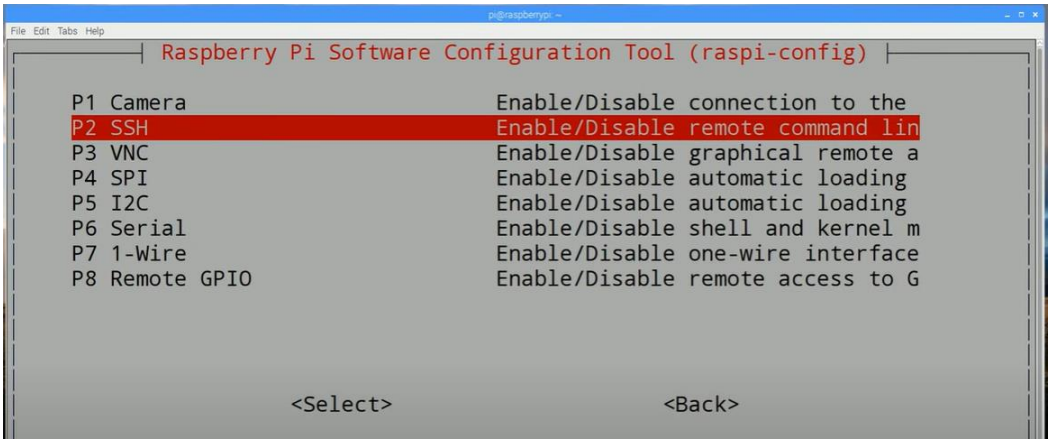
Step 2: Write command: sudo raspi-config

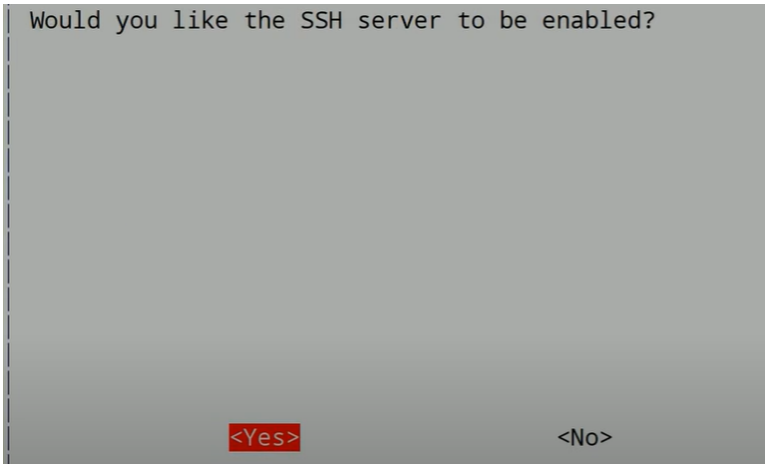


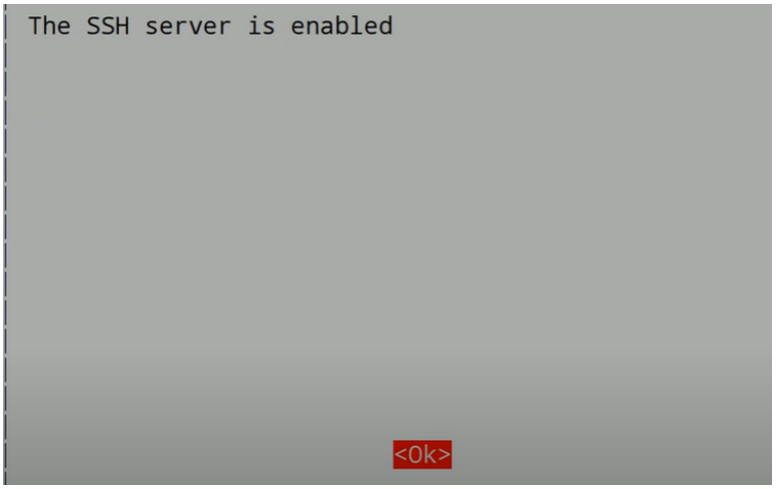
Step 3: Click on Interfacing Options.



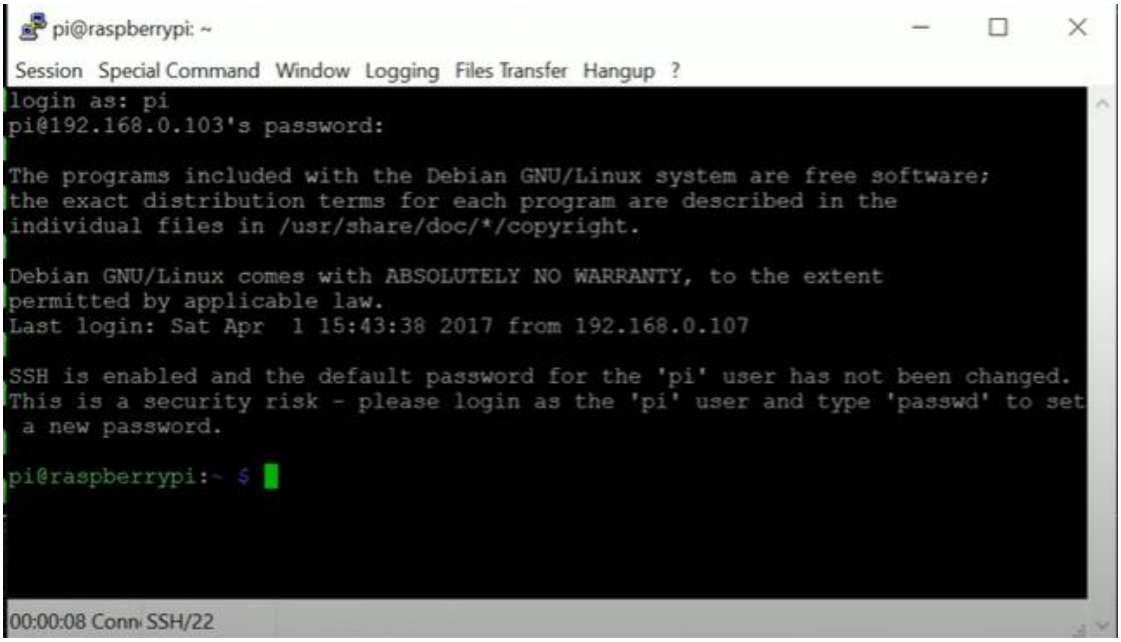
Step 4: Click on P2 SSH



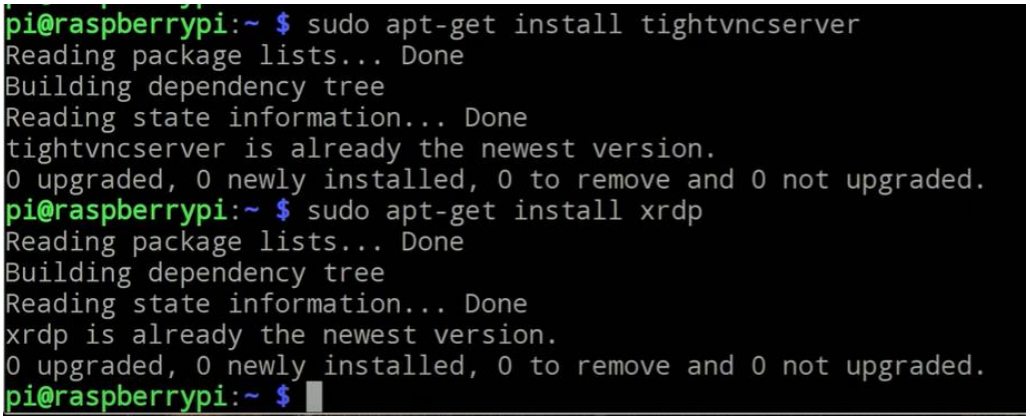




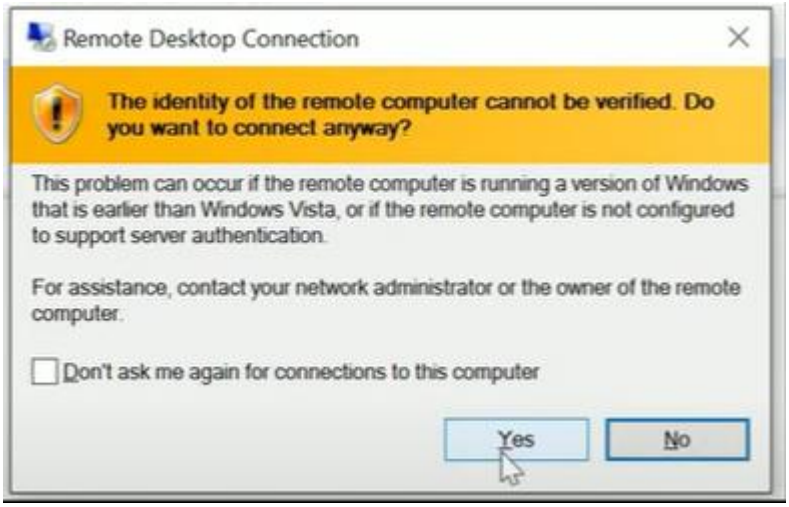
Step 5: Use the SSH client to remotely access the Raspberry Pi by entering its IP address.



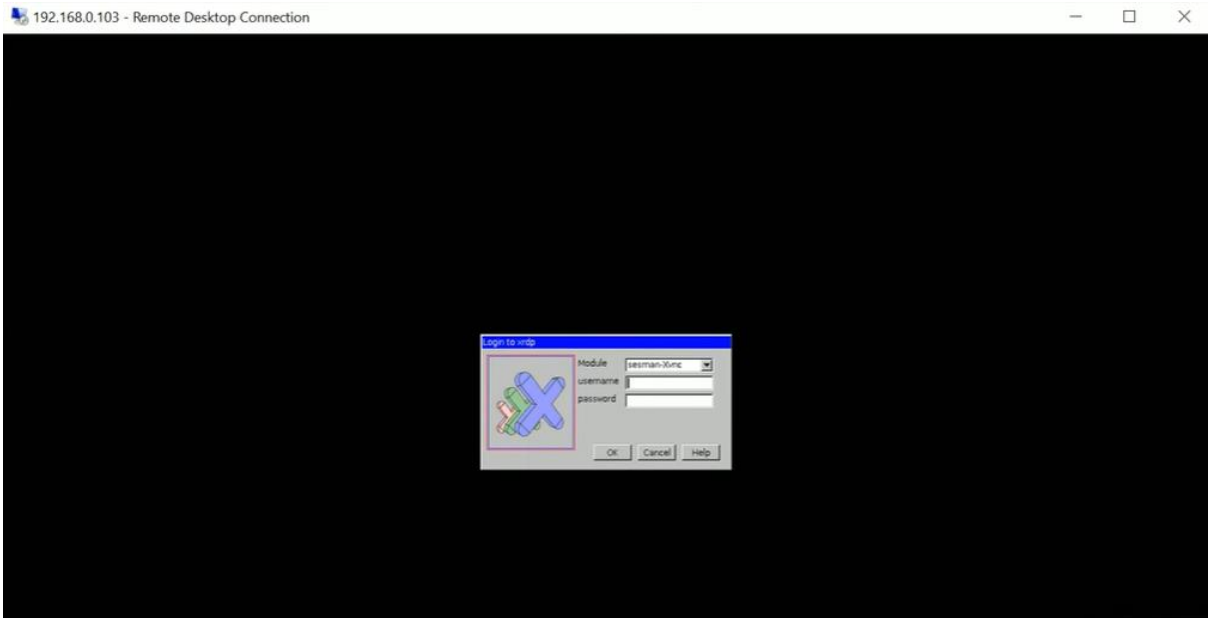
Step 6: Install an SSH client (e.g., PuTTY) and a VNC viewer (e.g., tightvnc Server).



Step 7: Open the VNC viewer, input the IP address of the Raspberry Pi, and log in to access the desktop remotely.



Step 8: Raspberry Pi remote access successful



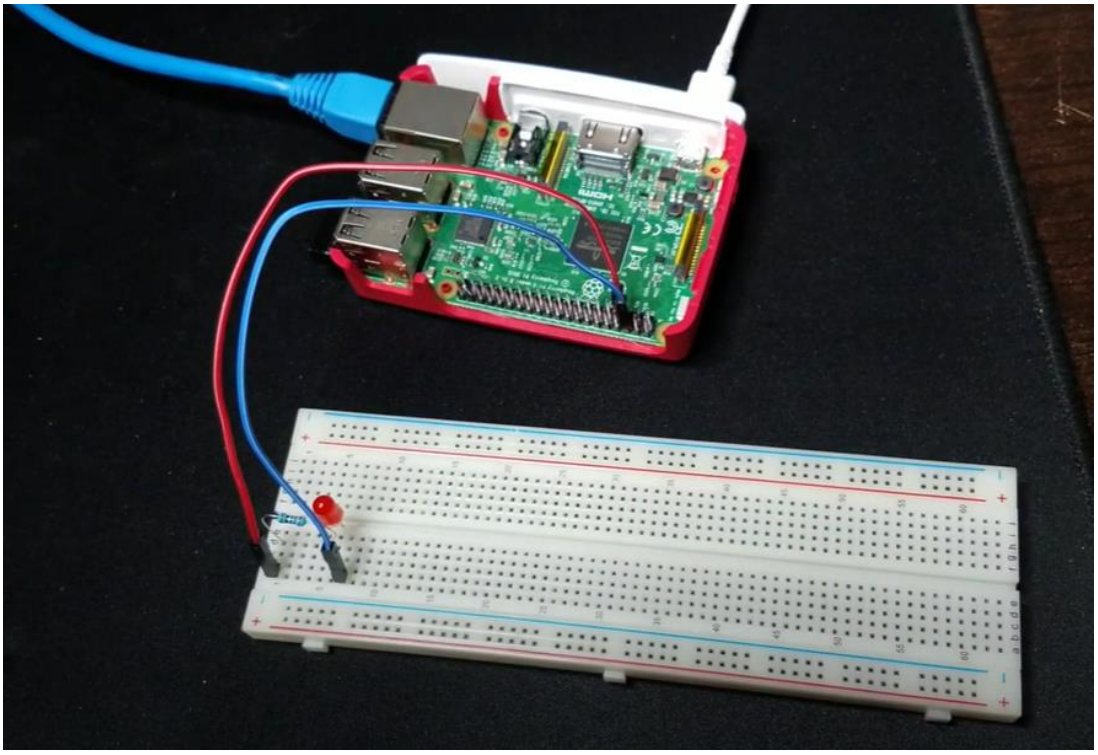
B) Blink an LED using Python programming.

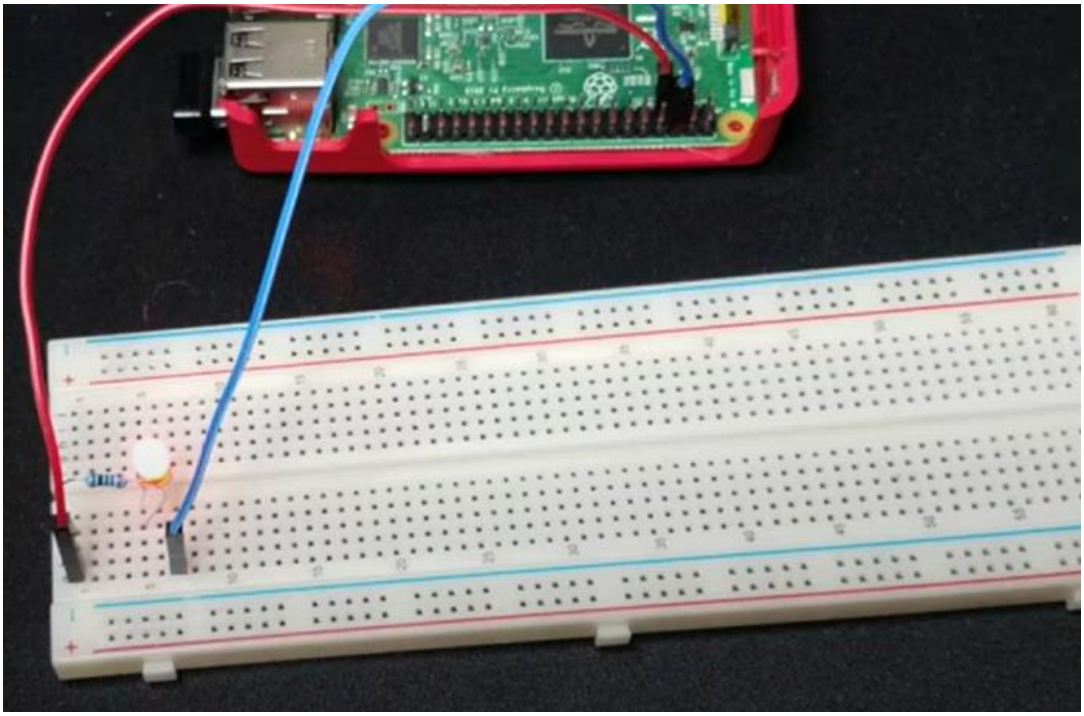
Code:

```
import RPi.GPIO as GPIO
import time

# Set up GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
```

```
# Blink LED
try:
    while True:
        GPIO.output(17, GPIO.HIGH)  # LED on
        time.sleep(1)
        GPIO.output(17, GPIO.LOW)   # LED off
        time.sleep(1)
except KeyboardInterrupt:
    pass
finally:
    GPIO.cleanup()
```

OUTPUT:





Observation

During the experiment, we successfully accessed the Raspberry Pi remotely through SSH and VNC, allowing us to manage it without a physical display. We learned to control GPIO pins using Python programming to blink an LED, gaining hands-on experience with basic circuit connections and Raspberry Pi's GPIO setup. This experiment reinforced our understanding of remote management of IoT devices and the flexibility of Python in hardware control.

Conclusion

In conclusion, remote access to Raspberry Pi through SSH and VNC is highly effective for managing and programming the device without a monitor. This experiment demonstrated how Raspberry Pi GPIO pins could be controlled using Python to interact with hardware components like LEDs. Such setups can be expanded to control more complex hardware, showcasing the Raspberry Pi's suitability for various IoT and automation projects.