IoT Practicals

1. **To write a program to sense the available networks using Arduino**

➔

Arduino Code:

```
#include <SPI.h>
#include <WiFi.h>

void setup() {
  // initialize serial and wait for the port to open:
  Serial.begin(9600);
  while (!Serial);

  // initialize WiFi
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    while (true); // don't continue
  }

  // attempt to connect to WiFi (if required)
  Serial.println("Initializing WiFi...");
  printMacAddress();

  // scan for existing networks:
```

```
  Serial.println("Scanning available networks...");
  listNetworks();
}

void loop() {
  delay(10000);
  // scan for existing networks:
  Serial.println("Scanning available networks...");
  listNetworks();
}

void printMacAddress() {
  // the MAC address of your WiFi shield
  byte mac[6];
  WiFi.macAddress(mac);

  // print the MAC address in the correct format:
  Serial.print("MAC Address: ");
  for (int i = 5; i >= 0; i--) {
    if (mac[i] < 16) Serial.print("0"); // leading zero if necessary
    Serial.print(mac[i], HEX);
    if (i > 0) Serial.print(":");
  }
  Serial.println();
```

```
}

void listNetworks() {
  // scan for nearby networks:
  Serial.println("** Scan Networks **");
  int numSsid = WiFi.scanNetworks();

  // print the list of networks seen:
  Serial.print("Number of available networks: ");
  Serial.println(numSsid);

  // print the network number, name, signal strength, and
encryption type:
  for (int thisNet = 0; thisNet < numSsid; thisNet++) {
    Serial.print(thisNet);
    Serial.print(": ");
    Serial.print(WiFi.SSID(thisNet));
    Serial.print(" | Signal: ");
    Serial.print(WiFi.RSSI(thisNet));
    Serial.print(" dBm | Encryption: ");

    // Print encryption type in readable format
    int encryption = WiFi.encryptionType(thisNet);
    switch (encryption) {
            case WIFI_AUTH_OPEN:
              Serial.println("Open");
              break;
            case WIFI_AUTH_WEP:
              Serial.println("WEP");
              break;
            case WIFI_AUTH_WPA_PSK:
              Serial.println("WPA_PSK");
              break;
            case WIFI_AUTH_WPA2_PSK:
              Serial.println("WPA2_PSK");
              break;
            case WIFI_AUTH_WPA_WPA2_PSK:
              Serial.println("WPA_WPA2_PSK");
              break;
            default:
              Serial.println("Unknown");
              break;
          }
        }
      }
```

To simulate your WiFi-based Arduino code in **Proteus** with a **Virtual Terminal**, you can follow these step-by-step instructions to set up your circuit and connect it to the Virtual Terminal.

Circuit Diagram Instructions for Proteus

1. **Components Needed:**
    - **Arduino (NodeMCU or ESP8266)**: The microcontroller that will handle the WiFi connection and communicate with the virtual terminal.
    - **Virtual Terminal**: A component in Proteus to simulate serial output.
    - **WiFi Module (ESP8266 or NodeMCU)**: If you're using NodeMCU or ESP8266, this component will handle the WiFi connection.
    - **Power Supply**: A suitable power source for the microcontroller.
2. **Step 1: Adding Arduino/NodeMCU in Proteus**
    - Open Proteus and go to **Components Mode** (press the **P** key).
    - Search for **NodeMCU** or **ESP8266** in the components library.
    - Place the NodeMCU/ESP8266 on the schematic workspace.

If NodeMCU is unavailable, you can use an **Arduino Uno** as an alternative. However, the code in your sketch is for **WiFi-based modules**, so using a NodeMCU or ESP8266 is ideal.

3. **Step 2: Adding Virtual Terminal**
    - In **Proteus Components Mode**, search for **"Virtual Terminal"**.

- Place the **Virtual Terminal** onto the workspace.

4. **Step 3: Connecting Components**
    - **TX (Transmitter) Pin** of the NodeMCU/ESP8266: Connect it to the **RX** pin of the **Virtual Terminal**.
    - **RX (Receiver) Pin** of the NodeMCU/ESP8266: Connect it to the **TX** pin of the **Virtual Terminal**.

This forms the serial communication link between the microcontroller and the Virtual Terminal.

5. **Step 4: Powering the Circuit**
    - Connect a **5V power supply** to the **VCC** pin of the NodeMCU/ESP8266.
    - Connect the **GND** pin of the NodeMCU/ESP8266 to the **GND** line on the schematic.
    - Ensure that your **Virtual Terminal** is powered by the Proteus simulation (it should automatically connect to the same ground).
6. **Step 5: Adding a Serial Connection for Debugging (Optional)**
    - In addition to the Virtual Terminal, you may connect the **TX/RX pins** of NodeMCU/ESP8266 to an actual **Arduino** or **other serial debugging device** to monitor the outputs.
7. **Step 6: Loading the Arduino Code into Proteus**
    - Compile the Arduino code in the **Arduino IDE**.
    - In **Proteus**, right-click on the **NodeMCU/ESP8266** component and select **Edit properties**.
    - In the **Program File** field, browse to the .hex file generated by the Arduino IDE for your project and upload it to Proteus.

8. **Step 7: Starting the Simulation**
    - Start the **simulation** in Proteus by clicking the **Play** button in the toolbar.
    - The Virtual Terminal should now display the Wi-Fi network scanning output, including MAC address and network details, every time the loop runs.
9. **Step 8: Monitoring the Output**
    - The Virtual Terminal will display the scanned Wi-Fi networks and their signal strengths. It will also show the MAC address as per your code, which you can monitor and analyze.

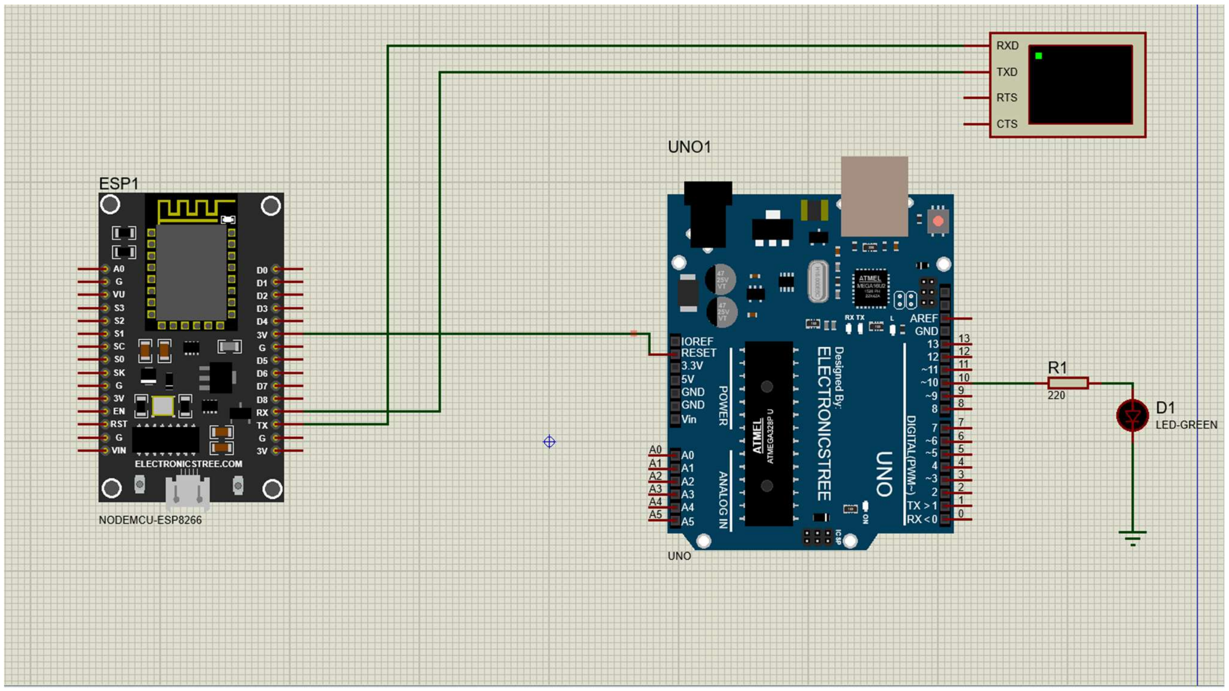
Example Proteus Circuit:

- **NodeMCU (or ESP8266)**: Used as the microcontroller.
- **Virtual Terminal**: Displays the serial output.
- **Power Supply**: Ensures proper power to the circuit.

Key Proteus Connections:

- **TX pin of NodeMCU → RX pin of Virtual Terminal**
- **RX pin of NodeMCU → TX pin of Virtual Terminal**
- **5V Power supply to NodeMCU/ESP8266**
- **GND to Common ground**

This setup will allow you to simulate the scanning of Wi-Fi networks and show the output in the virtual terminal within Proteus, as your Arduino code prints out the Wi-Fi details.


RXD
TXD
RTS
CTS
UNO1
ESP1
A0
G
VU
S3
S2
S1
SC
S0
SK
G
3V
EN
RST
G
VIN
D0
D1
D2
D3
D4
3V
G
D5
D6
D7
D8
RX
TX
G
3V
ELECTRONICSTREE.COM
NODEMCU-ESP8266
IOREF
RESET
3.3V
5V
GND
GND
Vin
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
Designed By:
ELECTRONICSTREE
UNO
DIGITAL(PWM~)
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX > 1
RX < 0
UNO
R1
220
D1
LED-GREEN

2.To write a program to measure the distance using ultrasonic sensor and make LED blink using Arduino

➔

```
// constants
const int TRIG_PIN = 6; // Arduino pin connected to Ultrasonic Sensor's TRIG pin
const int ECHO_PIN = 7; // Arduino pin connected to Ultrasonic Sensor's ECHO pin
const int LED_PIN = 3;  // Arduino pin connected to LED's pin
const int DISTANCE_THRESHOLD = 50; // centimeters

// variables
float duration_us, distance_cm;

void setup() {
  Serial.begin(9600); // initialize serial port
  pinMode(TRIG_PIN, OUTPUT); // set Arduino pin to output mode
  pinMode(ECHO_PIN, INPUT);  // set Arduino pin to input mode
  pinMode(LED_PIN, OUTPUT);  // set Arduino pin to output mode
}

void loop() {
  // generate 10-microsecond pulse to TRIG pin
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // measure duration of pulse from ECHO pin
  duration_us = pulseIn(ECHO_PIN, HIGH);

  // calculate the distance
  distance_cm = 0.017 * duration_us;

  if (distance_cm < DISTANCE_THRESHOLD)
    digitalWrite(LED_PIN, HIGH); // turn on LED
  else
    digitalWrite(LED_PIN, LOW);  // turn off LED

  // print the value to Serial Monitor
  Serial.print("distance: ");
  Serial.print(distance_cm);
  Serial.println(" cm");
  delay(500);
}
```

Instructions for Proteus Circuit Diagram:

1. **Components Needed**:
   - Arduino Uno (or any compatible board).

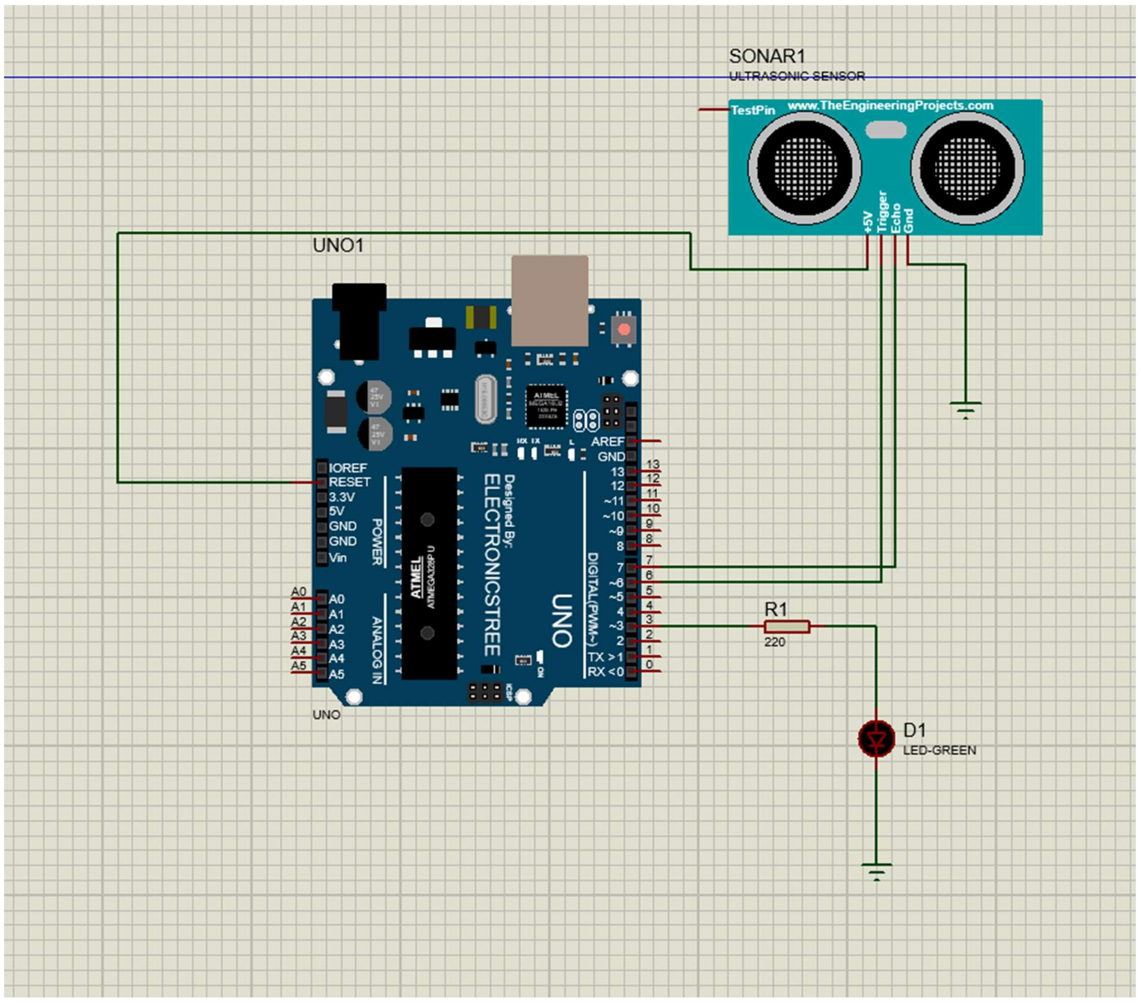
- HC-SR04 Ultrasonic Sensor.
- LED.
- 220Ω Resistor (for LED).
- 10kΩ Resistor (for ECHO pin pull-down).
- Wires for connections.

2. **Circuit Connections**:
   - **HC-SR04 Ultrasonic Sensor**:
     - VCC pin to 5V of Arduino.
     - GND pin to GND of Arduino.
     - TRIG pin to Arduino pin 6 (as defined in the code).
     - ECHO pin to Arduino pin 7 (as defined in the code). Attach a 10kΩ resistor to the ECHO pin to act as a pull-down resistor (optional, for stability).
   - **LED**:
     - The longer leg (anode) of the LED to Arduino pin 3 (as defined in the code).
     - The shorter leg (cathode) to one end of a 220Ω resistor.
     - The other end of the 220Ω resistor to the GND of the Arduino.
3. **Proteus Simulation Setup**:
   - Place all the components in the Proteus workspace.
   - Connect the TRIG and ECHO pins of the HC-SR04 to the corresponding Arduino pins as defined in the code.
   - Connect the LED to Arduino pin 3, with a current-limiting resistor.

- Make sure all components have correct connections for power and ground.

4. **Testing**:

- After placing and wiring the components, load the Arduino code into Proteus by clicking on the Arduino and selecting the appropriate hex file (compiled from Arduino IDE).
- Start the simulation, and you should see the LED turn on when the distance measured by the ultrasonic sensor is below the threshold of 50 cm.
- The serial monitor will show the measured distance in centimeters.



3.To write a program to detects the vibration of an object with sensor using Arduino.

➔

```
int vib_pin = 7;
int led_pin = 13;

void setup() {
  pinMode(vib_pin, INPUT);
  pinMode(led_pin, OUTPUT);
}

void loop() {
  int val = digitalRead(vib_pin);

  if (val == 1) {
    digitalWrite(led_pin, HIGH);
    delay(1000);
    digitalWrite(led_pin, LOW);
    delay(1000);
  } else {
    digitalWrite(led_pin, LOW);
  }
}
```

To create a circuit in Proteus using this code, follow these steps:

Components Needed:

1. **Arduino Uno** (or any compatible Arduino board)
2. **Vibration Sensor Module** (digital output type)
3. **LED**
4. **Resistors** (1KΩ for the LED, optional)
5. **Connecting Wires**

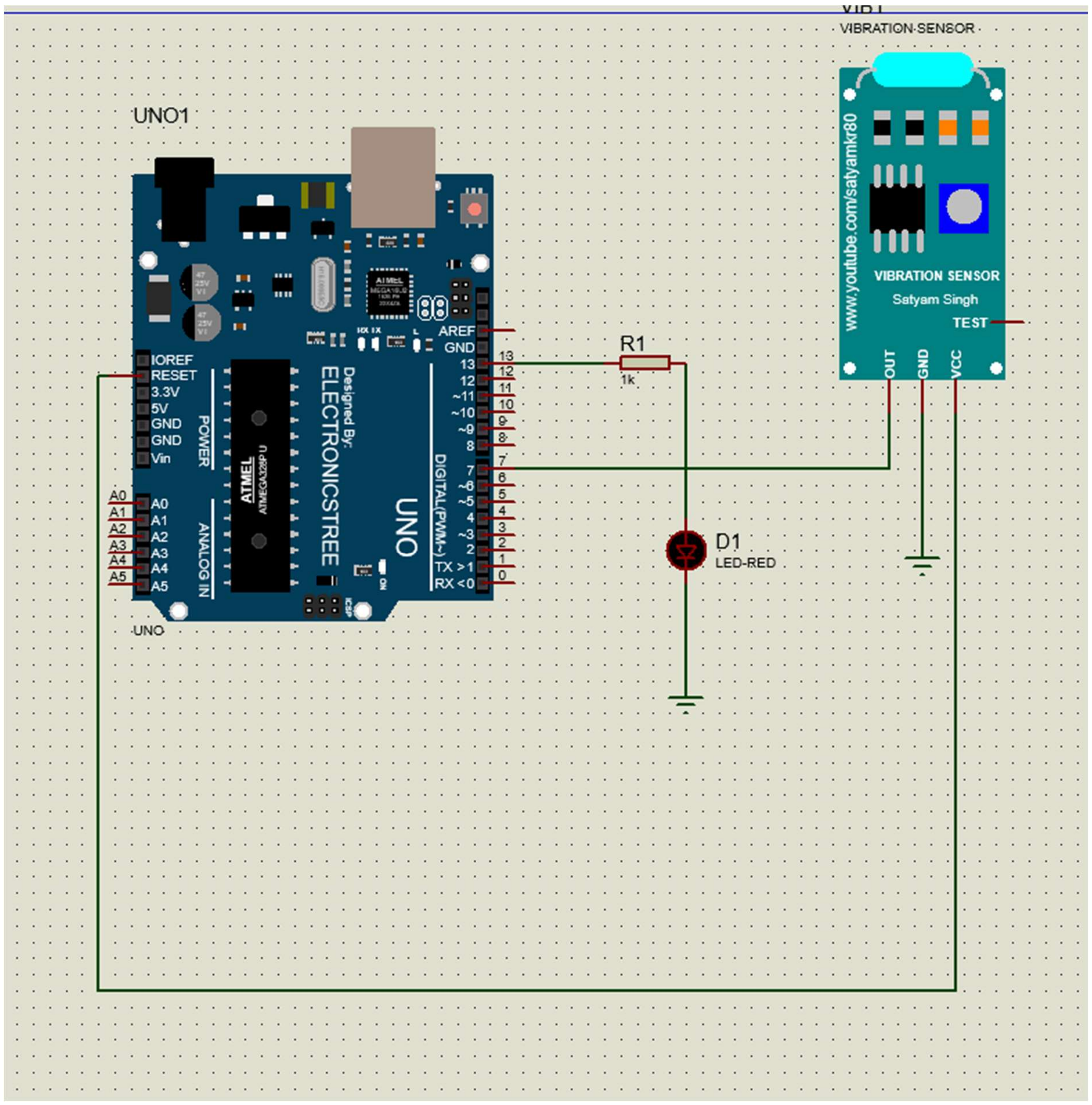
Step-by-Step Instructions:

1. **Place the Arduino Uno:**
    - Open Proteus and add the Arduino Uno to the workspace. You can find it in the "Components" library.
    - Double-click on the Arduino Uno to configure it.
    - Upload the code by going to **Program File** in the Arduino properties and selecting the compiled HEX file (generated in the Arduino IDE).
2. **Add the Vibration Sensor Module:**
    - Find a **Vibration Sensor Module** in the library (if unavailable, use a simple **Digital Input** pin to simulate the sensor).
    - Connect the **OUT** pin of the sensor module to **pin 7** on the Arduino Uno.
    - Connect the **GND** pin of the sensor to **GND** on the Arduino.
    - Connect the **VCC** pin of the sensor to **5V** on the Arduino.
3. **Connect the LED:**
    - Place an LED component on the workspace.

    - Connect the **anode (+)** of the LED to **pin 13** of the Arduino.
    - Connect the **cathode (-)** of the LED to one side of a **1KΩ resistor** (optional for current limiting).
    - Connect the other side of the resistor to **GND** on the Arduino.
4. **Run the Simulation:**
    - Start the simulation in Proteus.
    - The LED should turn ON for 1 second and OFF for 1 second whenever the vibration sensor is activated (detects vibrations).
    - If the sensor doesn't detect vibration, the LED should remain OFF.

Explanation of Circuit Functionality:

- When the vibration sensor sends a HIGH signal to pin 7, the Arduino reads this as **val = 1**. This turns the LED ON for 1 second, then OFF for 1 second.
- When no vibration is detected, the LED remains OFF.


VIB1
VIBRATION SENSOR
UNO1
www.youtube.com/satyamkr80
VIBRATION SENSOR
Satyam Singh
TEST
OUT
GND
VCC
R1
1k
D1
LED-RED
IOREF
RESET
3.3V
5V
GND
GND
Vin
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
AREF
GND
DIGITAL (PWM~)
ATMEL
ATMEGA328P U
Designed By:
ELECTRONICSTREE
UNO

4.To write a program to sense a finger when it is placed on the board Arduino.

➔

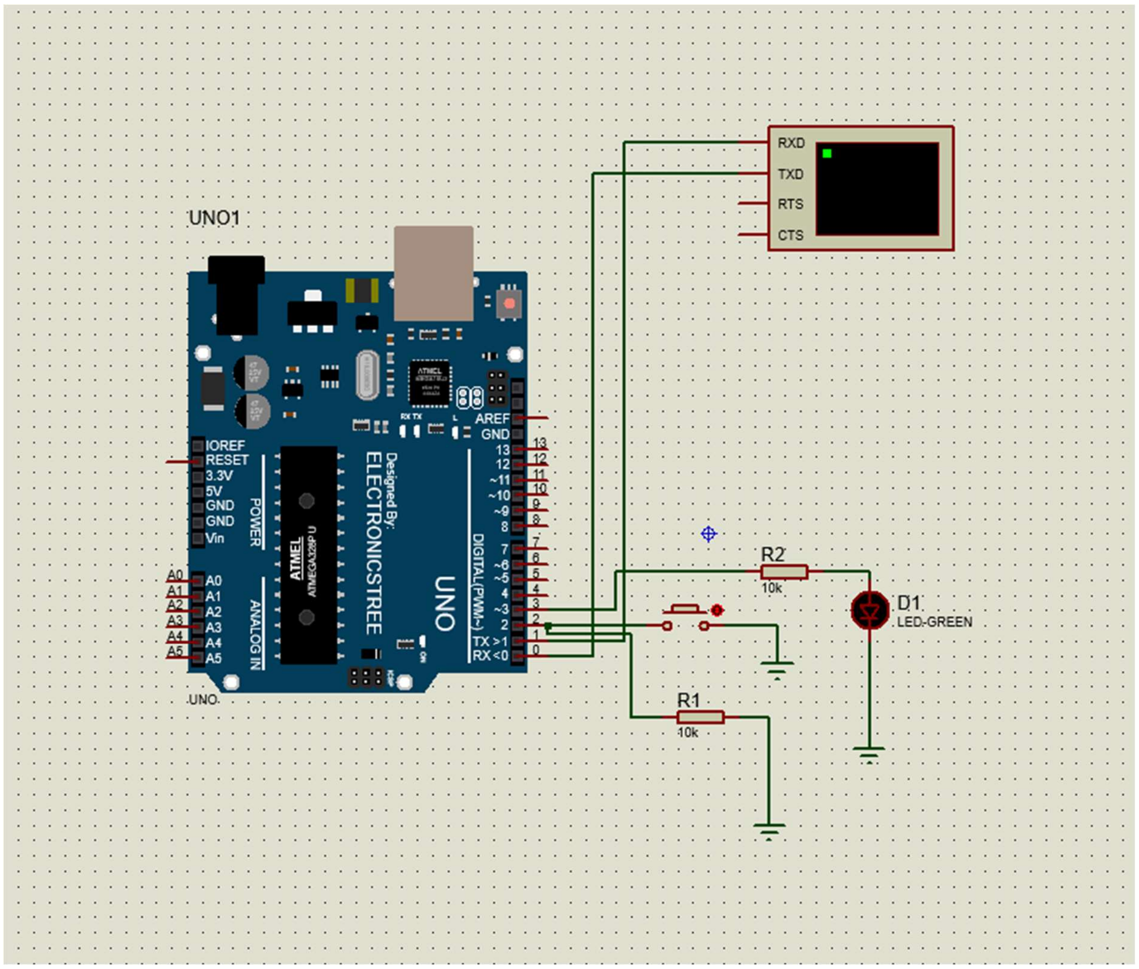
```
#include <CapacitiveSensor.h>
const int touchPin = 2;
const int ledPin = 3; // New LED pin
const int touchThreshold = 50;

CapacitiveSensor touchSensor = CapacitiveSensor(0, touchPin);

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT); // Set LED as output
}
void loop() {
  long touchValue = touchSensor.capacitiveSensor(30);

  if (touchValue >= touchThreshold) {
    Serial.println("Finger detected!");
    digitalWrite(ledPin, HIGH); // Turn on LED
    delay(1000); // wait to prevent multiple detections
    digitalWrite(ledPin, LOW); // Turn off LED after delay
  }
}
```

To simulate a **capacitive touch sensor** in Proteus, we'll use a similar circuit and configuration, but it's important to note that Proteus may have limited support for simulating capacitive touch behavior directly. However, I can guide you on how to set up a basic simulation for this setup using a **simple button or LED** as a substitute for demonstration purposes. This way, the circuit logic can be tested without the actual capacitive sensing feature.

Steps to Set Up the Circuit in Proteus

1. **Open Proteus**:
   - Start a new project.
2. **Add Components**:
   - From the **Components Library** (P button), add the following:
     - **Arduino Uno** (or other compatible microcontroller for which you're compiling).
     - **Push Button** (to mimic the capacitive touch sensor for testing purposes).
     - **Resistor** (10k ohm) for the pull-down configuration.
     - **Virtual Terminal** (for serial output, optional if you want to see the "Finger detected!" message).
     - **LED** (optional, to visually simulate detection).
3. **Component Placement and Connections**:
   - **Push Button** Setup:
     - Place the **push button** on the workspace.
     - Connect one side of the button to the **touchPin** on the Arduino (Pin 2 in this code).
     - Connect the other side of the button to **ground**.

        - Add a **10k ohm resistor** between **Pin 2 and ground** as a pull-down resistor.
    - **LED** Setup (Optional):
        - Connect the anode of the **LED** to any digital output pin on the Arduino (e.g., **Pin 3**).
        - Connect the cathode of the LED to ground through a **330-ohm resistor**.
    - **Virtual Terminal** Setup (Optional):
        - Connect the **TX (Transmit)** of the Arduino to the **RX** of the Virtual Terminal.
        - Set the **baud rate** of the Virtual Terminal to **9600**.



5. To write a program to connect with the available Wi-Fi using Arduino

➔

```
#include <WiFiNINA.h>

const char* ssid = "WiFi name";
const char* password = "secret password";

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

 // Check for WiFi module
 if (WiFi.status() == WL_NO_MODULE) {
  Serial.println("WiFi module not found");
  while (true);  // Halt if no WiFi module is found
 }

 // Attempt to connect to WiFi
 Serial.print("Connecting to WiFi...");
 WiFi.begin(ssid, password);

 // Wait for connection
 while (WiFi.status() != WL_CONNECTED) {
  Serial.print(".");
```

```
   delay(500);
  }

  Serial.println("\nWiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  // Your main code here
}
```

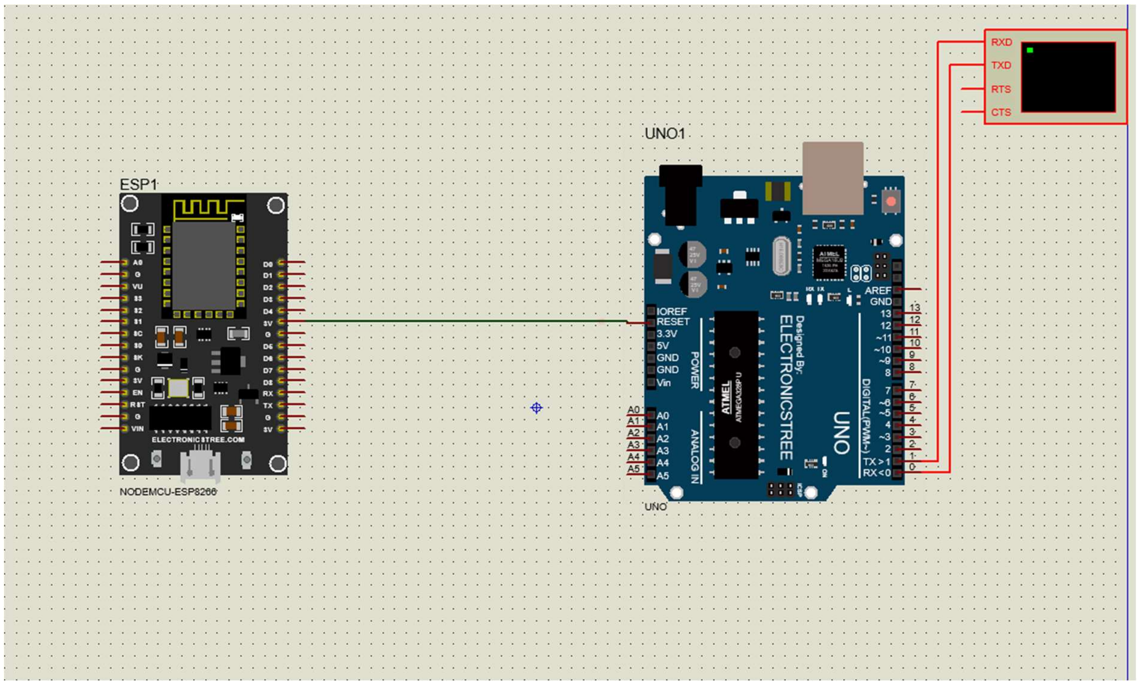
To simulate the code in Proteus using a virtual terminal and NodeMCU, you'll need to connect the components and configure them as follows:

Steps to Set Up the Circuit Diagram in Proteus

1. **Open Proteus**:
    - Start a new project and choose the right microcontroller (in this case, the **NodeMCU ESP8266**).
2. **Add Components**:
    - From the **Components Library** (P button), add:
        - **NodeMCU ESP8266** (make sure you've installed the ESP8266 library).

        - **WiFi module** (optional, but the NodeMCU will represent this).
        - **Virtual Terminal** (for serial output).

3. **Component Placement**:
    - Place the **NodeMCU** on the workspace.
    - Place the **Virtual Terminal** near the NodeMCU.
4. **Wiring Connections**:
    - Connect the **TX (Transmit)** pin of the NodeMCU to the **RX (Receive)** pin of the Virtual Terminal.
    - Connect the **RX (Receive)** pin of the NodeMCU to the **TX (Transmit)** pin of the Virtual Terminal.
    - Ground the Virtual Terminal as required.
5. **Configuration of the Virtual Terminal**:
    - Double-click the Virtual Terminal to set the **Baud Rate to 9600** (to match Serial.begin(9600) in your code).
6. **Add Power and Ground Connections**:
    - Ensure that the NodeMCU is connected to power and ground (if it's not powered by default).
7. **Program the NodeMCU in Proteus**:
    - Click on the NodeMCU and upload the compiled .hex file (compile the code in the Arduino IDE with the correct board settings, then locate the .hex file in the **Arduino build folder**).


ESP1
NODEMCU-ESP8266
UNO1
UNO
RXD
TXD
RTS
CTS

6.To write a program to get temperature notification using Arduino.

➔

```
#define ADC_VREF_MV 5000.0  // Reference voltage in millivolts
#define ADC_RESOLUTION 1024.0  // ADC resolution for 10-bit ADC

#define PIN_LM35 A0  // Pin where LM35 sensor is connected

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

void loop() {
  // Read the ADC value from the LM35 temperature sensor
  int adc_val = analogRead(PIN_LM35);

  // Convert the ADC value to voltage in millivolts
  float millivolt = (adc_val * ADC_VREF_MV) / ADC_RESOLUTION;

  // Convert the voltage to temperature in Celsius (LM35 gives 10mV
per °C)
  float temp_c = millivolt / 10;

  // Convert Celsius to Fahrenheit
  float temp_f = (temp_c * 9.0 / 5.0) + 32.0;
  // Print temperature in the Serial Monitor
  Serial.print("Temperature: ");
  Serial.print(temp_c);
  Serial.print("°C ");
  Serial.print(temp_f);
  Serial.println("°F");

  delay(1000);  // Delay for 1 second
}
```

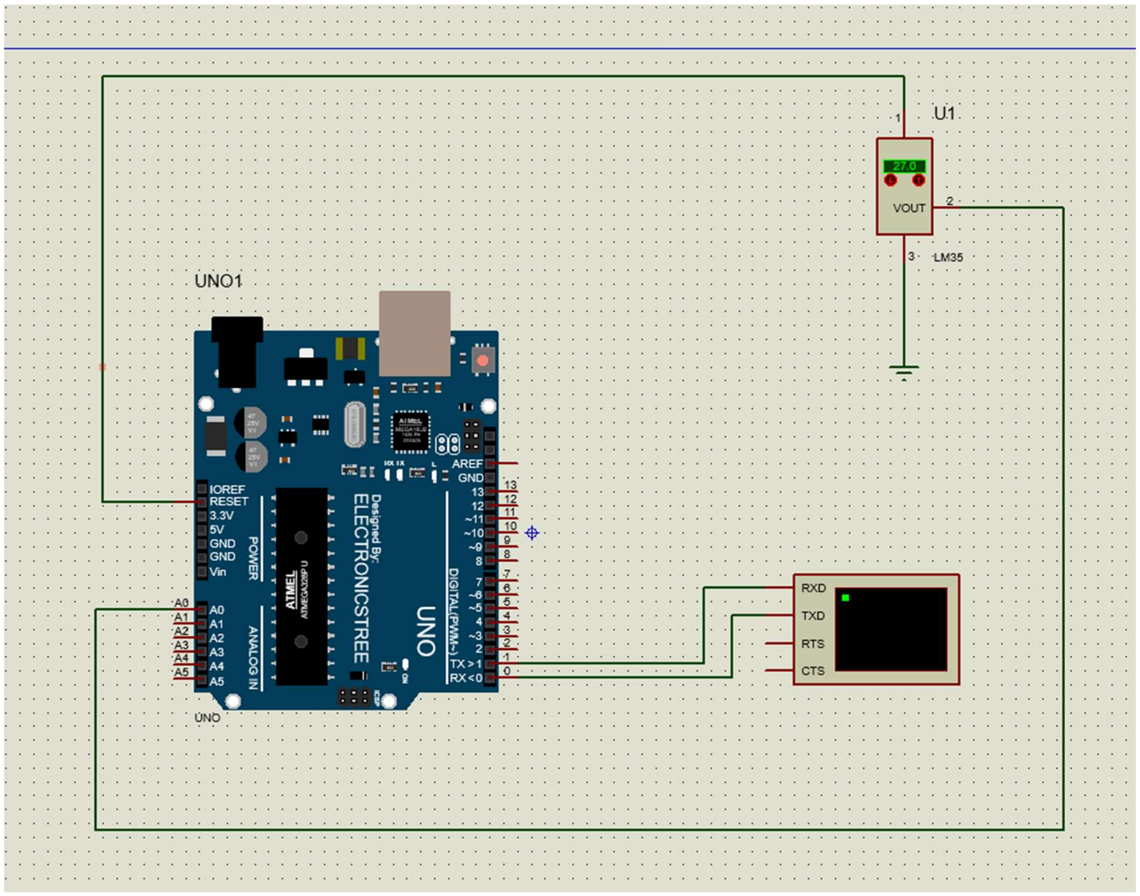
Instructions for Proteus Circuit Diagram

To simulate this in Proteus, follow these steps:

1. **Open Proteus**:
   - Open the Proteus software and create a new project.
2. **Add Components**:
   - Search and add the following components from the library:
     - **Arduino Uno** (or the microcontroller model you are using)
     - **LM35** (Temperature sensor)
     - **Virtual Terminal** (to simulate Serial Monitor)
3. **Place Components and Connections**:
   - **Arduino Uno**: Place the Arduino Uno on the workspace.
   - **LM35 Temperature Sensor**: Connect the LM35's pins as follows:

        - VCC (Pin 1) to +5V on Arduino.
        - GND (Pin 3) to GND on Arduino.
        - VOUT (Pin 2) to A0 (Analog Input) on Arduino.

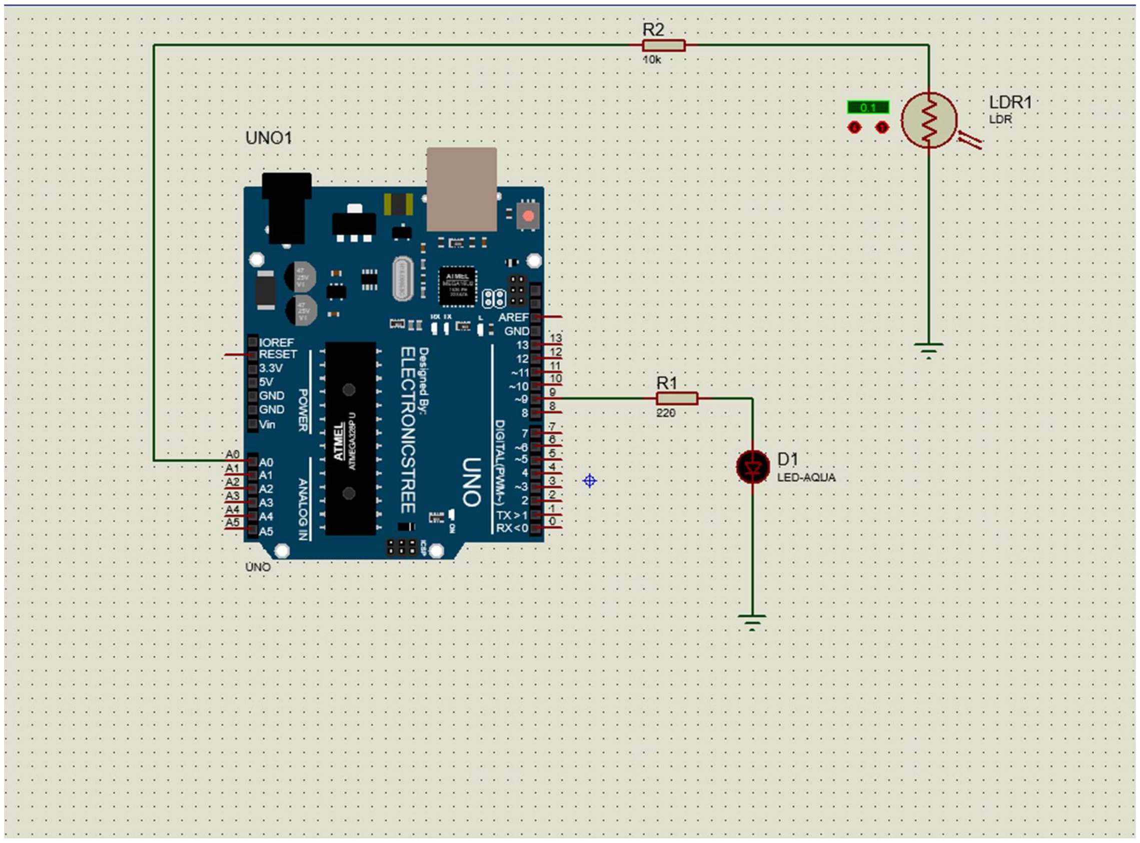
4. **Add the Virtual Terminal**:
    - Connect the TX pin of the Arduino Uno to the RXD of the Virtual Terminal.
    - Connect the GND pin of the Virtual Terminal to the Arduino GND.
5. **Program the Arduino in Proteus**:
    - Double-click the Arduino and upload the hex file of the code. To create the hex file, you can verify and compile the code in the Arduino IDE, which will automatically generate a hex file in the Arduino build folder.
6. **Run the Simulation**:
    - Run the simulation in Proteus.
    - The Virtual Terminal will display the temperature readings in Celsius and Fahrenheit, updating every second as per your code.


U1
27.0
VOUT
LM35
UNO1
IOREF
RESET
3.3V
5V
GND
GND
Vin
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX>1
RX<0
DIGITAL (PWM~)
Designed By:
ELECTRONICSTREE
UNO
ATMEL
ATMEGA328P U
RXD
TXD
RTS
CTS

7. **To write a program for LDR to vary the light intensity of LED using Arduino.**

   ➔

```
int sensor = A0;

int output = 9;


void setup() {

  pinMode(output, OUTPUT);

}


void loop() {

  int reading = analogRead(sensor);

  int bright = reading / 4;


  delay(500);

  analogWrite(output, bright);

}
```


R2
10k
LDR1
LDR
0.1
UNO1
R1
220
D1
LED-AQUA
UNO

Raspberry Pi in IoT

1.Start Raspberry Pi and try various Linux commands in command terminal window: ls, cd, touch, mv, rm, man, mkdir, rmdir, tar, gzip, cat, more, less, ps, sudo, cron, chown, chgrp, pingetc.

➔

2.Run some python programs on Pi like: a) Read your name and print Hello message with name b)Read two numbers and print their sum, difference, product and division. c) Word and character count of a given string. d)Area of a given shape (rectangle, triangle and circle) reading shape and appropriate values from standard input.

➔

a) Read your name and print Hello message with name

```
name = input("Enter your name: ")
print(f"Hello, {name}!")
```

b) Read two numbers and print their sum, difference, product, and division

```
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

sum_result = num1 + num2
difference = num1 - num2
```

```
product = num1 * num2
division = num1 / num2 if num2 != 0 else "undefined (division by
zero)"

print(f"Sum: {sum_result}")
print(f"Difference: {difference}")
print(f"Product: {product}")
print(f"Division: {division}")
```

c) Word and character count of a given string

```
text = input("Enter a string: ")

word_count = len(text.split())
character_count = len(text)

print(f"Word count: {word_count}")
print(f"Character count: {character_count}")
```

d) Area of a given shape (rectangle, triangle, or circle)

```
import math

shape = input("Enter the shape (rectangle, triangle, or circle): 
").lower()

if shape == "rectangle":
    length = float(input("Enter the length: "))
    width = float(input("Enter the width: "))
    area = length * width
    print(f"Area of the rectangle: {area}")

elif shape == "triangle":
    base = float(input("Enter the base: "))
    height = float(input("Enter the height: "))
    area = 0.5 * base * height
    print(f"Area of the triangle: {area}")

elif shape == "circle":
    radius = float(input("Enter the radius: "))
    area = math.pi * radius ** 2
    print(f"Area of the circle: {area}")
```

```
else:
    print("Invalid shape.")
```

3.Run some python programs on Pi like: a) Print a name 'n' times, where name and n are read from standard input, using for and while loops. b) Handle Divided by Zero Exception. c) Print current time for 10 times with an interval of10seconds. d) Read a fileline byline and print the word count of each line

➔

```
# a) Print a name 'n' times, where name and n are read from standard input


name = input("Enter your name: ")

n = int(input("Enter the number of times to print your name: "))


# Using a for loop

print("\nUsing for loop:")

for i in range(n):

    print(name)


# Using a while loop

print("\nUsing while loop:")

count = 0

while count < n:

    print(name)
    count += 1

# b) Handle Division by Zero Exception

try:
    numerator = float(input("Enter the numerator: "))
    denominator = float(input("Enter the denominator: "))
    result = numerator / denominator
    print(f"Result: {result}")
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")

# c) Print current time for 10 times with an interval of 10 seconds

import time
from datetime import datetime

print("Printing current time every 10 seconds:")
for _ in range(10):
    current_time = datetime.now().strftime("%H:%M:%S")
    print(f"Current time: {current_time}")
    time.sleep(10)  # Wait for 10 seconds
```

```
# d) Read a file line by line and print the word count of each line

# Make sure to replace 'filename.txt' with the path to your file
filename = "filename.txt"

try:
    with open(filename, 'r') as file:
        for line_number, line in enumerate(file, start=1):
            word_count = len(line.split())
            print(f"Line {line_number}: {word_count} words")
except FileNotFoundError:
    print(f"Error: The file '{filename}' was not found.")
```

4.Run some python programs on Pi like a) Light an LED through Python program b) Get input from two switches and switch on corresponding LEDs c) Flash an LED at a given on time and off time cycle, where the two times are taken from a file

➔

a) Light an LED through Python program

This program will light up an LED connected to a specific GPIO pin.

```
import RPi.GPIO as GPIO

import time


# Setup

GPIO.setmode(GPIO.BCM)

LED_PIN = 18  # Replace with the pin number where the LED is connected

GPIO.setup(LED_PIN, GPIO.OUT)


# Turn on LED

GPIO.output(LED_PIN, GPIO.HIGH)

time.sleep(5)  # Keep it on for 5 seconds


# Cleanup

GPIO.output(LED_PIN, GPIO.LOW)

GPIO.cleanup()
```

To create a Proteus simulation for an LED controlled by a Python program, you can simulate it with the Raspberry Pi model that supports GPIO control in Proteus. Here's a step-by-step guide on setting it up in Proteus:

Components Needed:

1. **Raspberry Pi** (preferably Raspberry Pi 3 or Raspberry Pi 4 model if available in Proteus).
2. **LED** (any color of your choice).
3. **Resistor** (330 ohms or 220 ohms to limit current for the LED).

Steps to Create the Circuit Diagram:

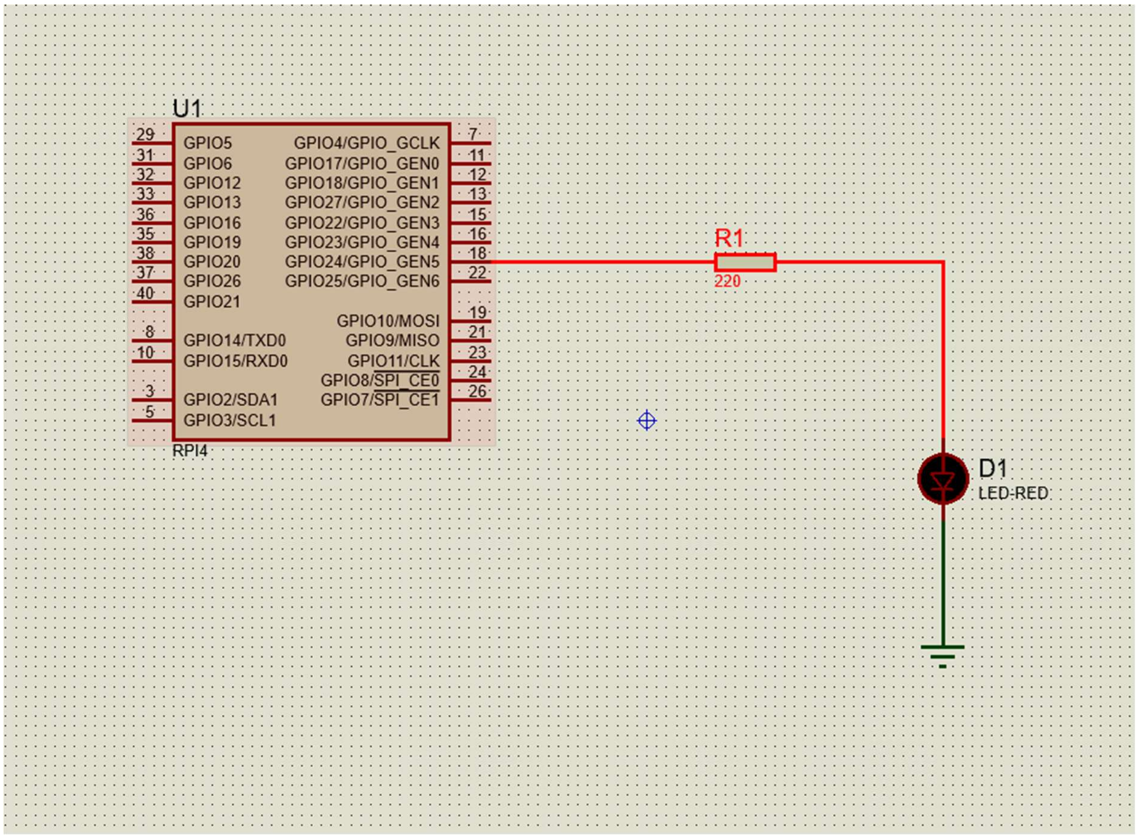
1. **Open Proteus**:
   - Open a new project in Proteus and go to the "Component Mode."
2. **Add Components**:
   - Search for **Raspberry Pi** in the component library (make sure you have a compatible Raspberry Pi model installed).
   - Add **LED** and **Resistor** components from the library.
3. **Connect Components**:
   - Place the **LED** in the circuit and connect its **anode (positive)** to **GPIO Pin 18** of the Raspberry Pi.
   - Connect the **cathode (negative)** of the LED to one end of the **resistor**.
   - Connect the other end of the resistor to the **Ground (GND)** pin on the Raspberry Pi.

Note: The resistor limits the current flowing through the LED, preventing it from burning out.

4. **Power and Ground Connections**:

  - Ensure that the **Raspberry Pi** is connected to a **power supply** (check the Raspberry Pi's datasheet for appropriate voltage).
  - Connect other necessary **GND** pins if required by the simulation settings.

5. **Configure Raspberry Pi GPIO Mode**:
   - In some Proteus simulations, you may need to set the GPIO mode directly through the Proteus properties or by programming it within your Python script.
6. **Python Code Execution**:
   - For the Python code, you typically use a Raspberry Pi setup to execute it in real life. In Proteus, if you need to simulate the GPIO signals, you may need additional scripts or a microcontroller that emulates Python code execution, as Proteus does not directly run Python.
7. **Simulation**:
   - Once everything is connected, run the simulation. Observe if the LED lights up as it would when GPIO pin 18 is set to HIGH.


U1
GPIO5
GPIO6
GPIO12
GPIO13
GPIO16
GPIO19
GPIO20
GPIO26
GPIO21
GPIO14/TXD0
GPIO15/RXD0
GPIO2/SDA1
GPIO3/SCL1
GPIO4/GPIO_GCLK
GPIO17/GPIO_GEN0
GPIO18/GPIO_GEN1
GPIO27/GPIO_GEN2
GPIO22/GPIO_GEN3
GPIO23/GPIO_GEN4
GPIO24/GPIO_GEN5
GPIO25/GPIO_GEN6
GPIO10/MOSI
GPIO9/MISO
GPIO11/CLK
GPIO8/SPI_CE0
GPIO7/SPI_CE1
29 31 32 33 36 35 38 37 40 8 10 3 5
7 11 12 13 15 16 18 22 19 21 23 24 26
RPI4
R1
220
D1
LED-RED

b) Get input from two switches and switch on corresponding LEDs

This program will read the input from two switches. Each switch will turn on its corresponding LED when pressed.

```
import RPi.GPIO as GPIO

import time


# Setup

GPIO.setmode(GPIO.BCM)

SWITCH1_PIN = 7  # Replace with your switch pin numbers

SWITCH2_PIN = 26

LED1_PIN = 22  # Corresponding LEDs for each switch

LED2_PIN = 23


GPIO.setup(SWITCH1_PIN, GPIO.IN,
pull_up_down=GPIO.PUD_DOWN)

GPIO.setup(SWITCH2_PIN, GPIO.IN,
pull_up_down=GPIO.PUD_DOWN)

GPIO.setup(LED1_PIN, GPIO.OUT)

GPIO.setup(LED2_PIN, GPIO.OUT)


try:

    while True:

        if GPIO.input(SWITCH1_PIN) == GPIO.HIGH:

            GPIO.output(LED1_PIN, GPIO.HIGH)

        else:
            GPIO.output(LED1_PIN, GPIO.LOW)


        if GPIO.input(SWITCH2_PIN) == GPIO.HIGH:
            GPIO.output(LED2_PIN, GPIO.HIGH)
        else:
            GPIO.output(LED2_PIN, GPIO.LOW)


        time.sleep(0.1)


except KeyboardInterrupt:
    pass
finally:
    GPIO.cleanup()
```

o simulate this program in Proteus, you can follow these steps to design a circuit where two switches control two LEDs. Here's the setup guide:

Components Needed:

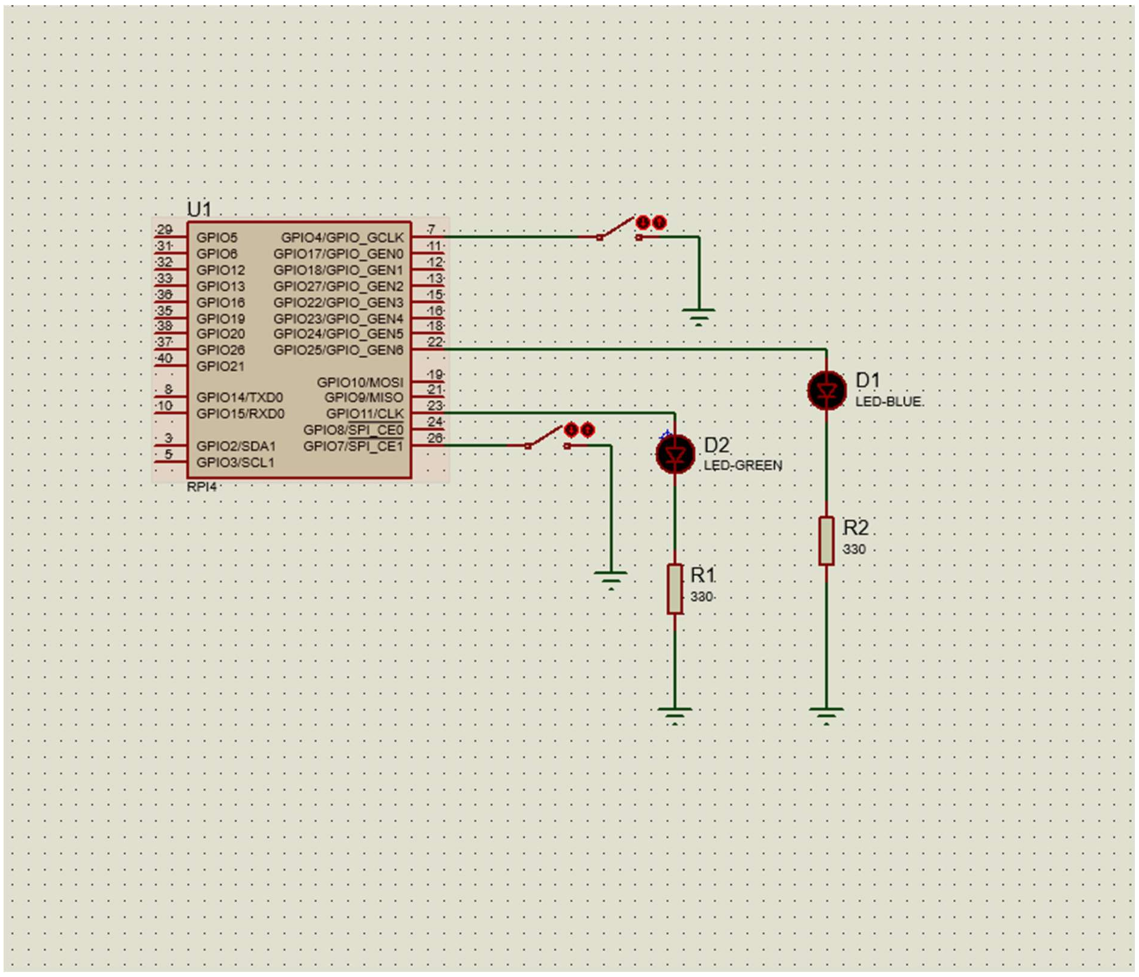
1. **Raspberry Pi** (e.g., Raspberry Pi 3 or 4 if available in Proteus).
2. **Two LEDs** (can be different colors to distinguish each).
3. **Two Resistors** (330 ohms or 220 ohms for each LED).
4. **Two Push Button Switches**.
5. **Resistors for Pull-down Configuration** (10k ohms for each switch if needed).

Steps to Create the Circuit Diagram:

1. **Open Proteus and Add Components**:
    - Open Proteus, start a new project, and go to "Component Mode."
    - Search for **Raspberry Pi** (if available in your Proteus library).
    - Add **2 LEDs**, **2 Push Buttons**, and **4 Resistors**.
2. **Set Up LEDs**:
    - Connect **LED1's anode** to **GPIO Pin 22** (LED1_PIN) and **LED2's anode** to **GPIO Pin 23** (LED2_PIN).
    - Connect the **cathode of each LED** to one end of the **330-ohm resistor**, and the other end of the resistor to the **ground (GND)** pin on the Raspberry Pi.
3. **Set Up Switches**:
    - Connect **one terminal of Switch1** to **GPIO Pin 7** (SWITCH1_PIN) and **one terminal of Switch2** to **GPIO Pin 26** (SWITCH2_PIN).
    - Connect the **other terminal of each switch** to **ground (GND)**.
4. **Pull-Down Resistors** (optional if needed):
    - Place a **10k-ohm resistor** between each GPIO pin (7 and 26) and ground to act as pull-down resistors, ensuring the GPIO pin reads LOW when the switch isn't pressed.
5. **Power and Ground Connections**:
    - Ensure that the **Raspberry Pi** is connected to a **power supply** and **GND** connections are consistent across components.

- If your Proteus Raspberry Pi model requires additional power configuration, ensure you follow the specific setup instructions for your model.

6. **Simulation**:
    - Start the simulation in Proteus.
    - When **Switch1** is pressed, **LED1** connected to GPIO Pin 22 should light up.
    - When **Switch2** is pressed, **LED2** connected to GPIO Pin 23 should light up.
    - Releasing each switch should turn off the corresponding LED.



Flash an LED at a given on-time and off-time cycle

```
import RPi.GPIO as GPIO
import time

# Setup
GPIO.setmode(GPIO.BCM)
LED_PIN = 18  # Replace with your LED pin
GPIO.setup(LED_PIN, GPIO.OUT)

# Sample on and off times for simulation
on_time = 1.0  # 1 second on
off_time = 0.5  # 0.5 second off

try:
    while True:
        GPIO.output(LED_PIN, GPIO.HIGH)
        time.sleep(on_time)
        GPIO.output(LED_PIN, GPIO.LOW)
        time.sleep(off_time)

except KeyboardInterrupt:
    pass
finally:
    GPIO.cleanup()
```

o simulate this program in Proteus, you'll create a circuit that flashes an LED based on on/off timings read from a file. While Proteus can handle the hardware simulation, it won’t natively read from external files. However, the circuit will represent the on/off timing control of an LED using a Raspberry Pi GPIO pin.

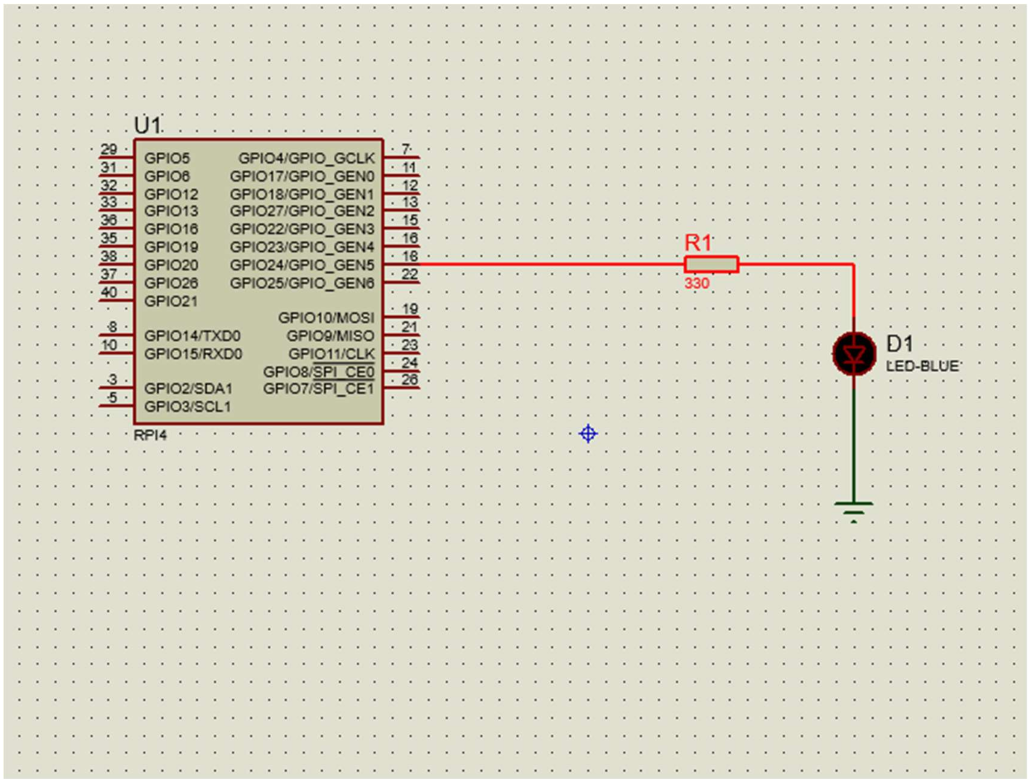
Components Needed:

1. **Raspberry Pi** (e.g., Raspberry Pi 3 or Raspberry Pi 4 model in Proteus).
2. **One LED**.
3. **One Resistor** (330 ohms or 220 ohms for current limiting).

Steps to Create the Circuit Diagram:

1. **Open Proteus and Add Components**:
   - Start a new project in Proteus.
   - Add the **Raspberry Pi**, **LED**, and **resistor** to the workspace from the Proteus library.
2. **Connect the LED**:
   - Connect the **anode** (positive leg) of the LED to **GPIO Pin 18** on the Raspberry Pi.
   - Connect the **cathode** (negative leg) of the LED to one end of the **330-ohm resistor**.
   - Connect the other end of the resistor to the **ground (GND)** pin on the Raspberry Pi.
3. **Power and Ground Connections**:
   - Connect the **power** and **GND** pins of the Raspberry Pi as per your Proteus configuration to ensure correct operation.
4. **Simulating the Timing Control**:

- Proteus doesn't directly support reading from external files like timing.txt. Instead, you can simulate the LED flashing by manually setting on_time and off_time values in a test program. Here's how to modify the code for Proteus simulation:


U1
GPIO5
GPIO6
GPIO12
GPIO13
GPIO16
GPIO19
GPIO20
GPIO26
GPIO21
GPIO14/TXD0
GPIO15/RXD0
GPIO2/SDA1
GPIO3/SCL1
GPIO4/GPIO_GCLK
GPIO17/GPIO_GEN0
GPIO18/GPIO_GEN1
GPIO27/GPIO_GEN2
GPIO22/GPIO_GEN3
GPIO23/GPIO_GEN4
GPIO24/GPIO_GEN5
GPIO25/GPIO_GEN6
GPIO10/MOSI
GPIO9/MISO
GPIO11/CLK
GPIO8/SPI_CE0
GPIO7/SPI_CE1
RPI4
R1
330
D1
LED-BLUE