SRI KRISHNA COLLEGE OF TECHNOLOGY

[An Autonomous Institution | Affiliated to Anna University and
Approved by AICTE | Accredited by NBA & NAAC with 'A' Grade]
KOVAIPUDUR, COIMBATORE – 641 042.

DEPARTMENT OF INTERNET OF THINGS

22SBE16 – IOT DESIGN USING MICROCONTROLLER LABORATORY MANUAL

Regulation: 2022

IV Year B.Tech. IOT / VII Semester

Academic Year: 2025-2026

DEPARTMENT OF INTERNET OF THINGS

ACADEMIC YEAR: 2025 – 2026 (ODD)

LAB MANUAL

22SBE16 – IOT DESIGN USING MICROCONTROLLER LABORATORY MANUAL

(For IV Year IOT - VII Semester)

Prepared by

1. Dr.K.Bagyalakshmi (AP/ECE)

2. Mr.V.Suresh Babu (AP/ECE)

Approved By

Prof & Program

DEPARTMENT OF INTERNET OF THINGS

SRI KRISHNA COLLEGE OF TECHNOLOGY

[An Autonomous Institution | Affiliated to Anna University and
Approved by AICTE | Accredited by NBA & NAAC with 'A' Grade]
KOVAIPUDUR, COIMBATORE – 641 042.

VISION

The department of CSE fosters a conducive ambience to meet the global standards by equipping the students with modern techniques in the area of Computer Science and relevant research to address the societal needs.

MISSION

M1: To provide positive working environment that would help the students perform to their highest abilities in various fields of computer science.

M2: To enable students and faculty with the best of technologies and knowledge emerging in the domain of Computer Science and Engineering.

M3: To establish nationally and internationally recognized research centres and expose the students to broad research experience.

PROGRAM EDUCTIONAL OBJECTIVES

PEO1. Graduates will be able to solve computer science and engineering problems with solid foundation in mathematical, scientific and engineering fundamentals.

PEO2. Graduates will be able to apply programming skills & computing techniques to design and maintain hardware and software with social relevance.

PEO3. Graduates will be able to adapt, contribute and innovate new technologies in the key domains of Computer Science and Engineering and to productively extend their interest by pursuing higher studies and research.

PEO4. Graduates will be professionally competent and ethically responsible entrepreneurs in providing pioneering solutions to contemporary problems with effective multidisciplinary, communication and team work skills

PROGRAM OUTCOMES

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

PSO1: Will be able to practice algorithmic thinking with deep understanding of/in basic concepts of computer science to develop software solutions

PSO2: Ability to design complex software systems by deploying programming, networking and data management strategies.

PSO3: Will be able to employ computing intelligence and software development concepts to design and develop secured systems/applications for societal benefits.

DEPARTMENT OF INTERNET OF THINGS

COMMON INSTRUCTIONS

DO's

1. Login-on with your username and password.
2. Log off the computer every time when you leave the Lab.
3. Arrange your chair properly when you are leaving the lab.
4. Put your bags in the designated area.
5. Ask permission to print.

DONT's

1. Do not share your username and password.

2. Do not remove or disconnect cables or hardware parts.

3. Do not personalize the computer setting.

4. Do not run programs that continue to execute after you log off.

5. Do not download or install any programs, games or music on computer in Lab.

6. Personal Internet use chat room for Instant Messaging (IM) and Sites Strictly Prohibited.

7. No Internet gaming activities allowed.

8. Tea, Coffee, Water & Eatables are not allowed in the Computer Lab.

INDEX

S. No	Name of the Experiment

1	Demonstrate the LED Blinking and GPIO Control.
2	Perform Analog Sensor Interfacing.
3	Demonstrate PWM and Servo Motor Control.
4	Demonstrate UART communication between the microcontroller and a PC or another microcontroller.
5	Perform basic HTTP requests with microcontroller and Wifi Network.
6	Experiment interfacing pc.
7	Implement multitasking the ESP32.
8	Demonstrate Bluetooth Communication.
CONTENT BEYOND SYLLABUS	
10	Design Solutions for Smart Home Using Arduino Processor
OPEN ENDED EXPERIMENT	
11	Simulation and Analysis of Real-Time Task Scheduling using Free RTOS in Keil µVision or Proteus Simulator

ARDUINO UNO

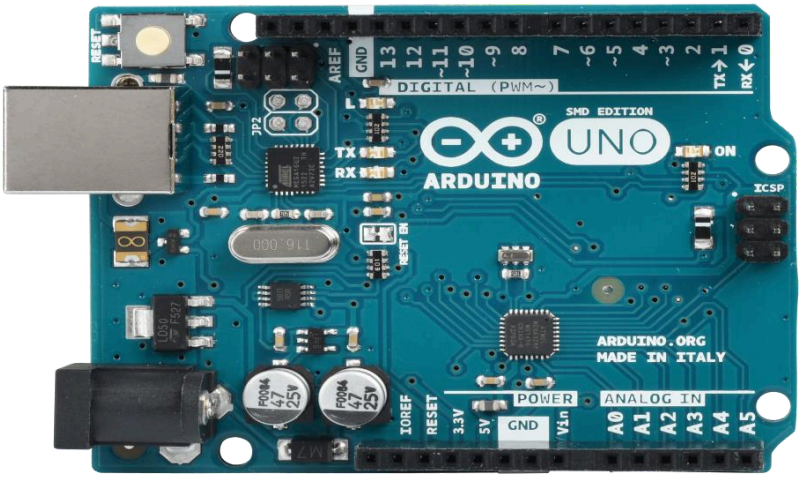
INTRODUCTION:

The Arduino Uno is an open-source microcontroller board based on the Microchip ATmega328P microcontroller and developed by Arduino.cc. The board is equipped with

sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) and other circuits. The board has 14 digital I/O pins (six capable of PWM output), 6 analog I/O pins, and is programmable with the Arduino IDE (Integrated Development Environment), via a type B USB cable. It can be powered by theUSB cable or by an external 9-volt battery, though it accepts voltages between 7 and 20 volts. The word "uno" means "one" in Italian and was chosen to markthe initial release of Arduino Software.

Features of the Arduino

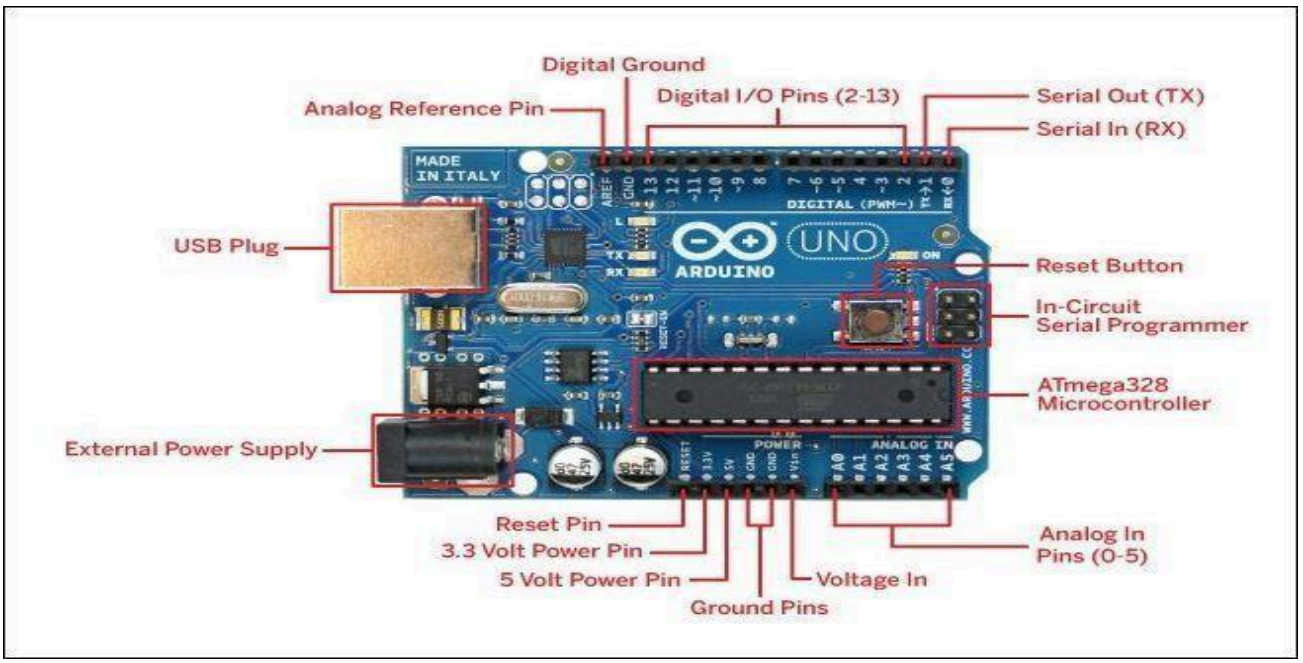
1. Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.



2. The board functions can be controlled by sending a set of instructions to the microcontroller on the board via Arduino IDE.
3. Arduino IDE uses a simplified version of C++, making it easier to learn to program.
4. Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

Arduino UNO Pin Configuration

Arduino boards senses the environment by receiving inputs from many sensors, and affects their surroundings by controlling lights, motors, and other actuators. Arduino boards are the microcontroller development platform that will be at the heart of your projects. When making something you will be building the circuits and interfaces for interaction, and telling the microcontroller how to interface with other components. Here the anatomy of Arduino UNO.



1. **Digital pins** Use these pins with digitalRead(), digitalWrite(), and analogWrite(). analogWrite() works only on the pins with the PWM symbol.
2. **Pin 13 LED** The only actuator built-in to your board. Besides being a handy target for your first blink sketch, this LED is very useful for debugging.
3. **Power LED** Indicates that your Arduino is receiving power. Useful for debugging.
4. **ATmega** microcontroller The heart of your board.
5. **Analog in** Use these pins with analogRead().
6. **GND and 5V pins** Use these pins to provide +5V power and ground to your circuits.
7. **Power connector** This is how you power your Arduino when it's not plugged into a USB port for power. Can accept voltages between 7-12V.
8. **TX and RX LEDs** These LEDs indicate communication between your Arduino and your computer. Expect them to flicker rapidly during sketch upload as well as during serial communication. Useful for debugging.
9. **USB port** Used for powering your Arduino UNO, uploading your sketches to your Arduino, and for communicating with your Arduino sketch (via Serial. println() etc.).
10. **Reset button** Resets the ATmega microcontroller.

Arduino IDE

(Integrated Development Environment)

Introduction: The Arduino Software (IDE) is easy-to-use and is based on the Processing programming environment. The Arduino Integrated Development Environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in functions from C and C++.The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.

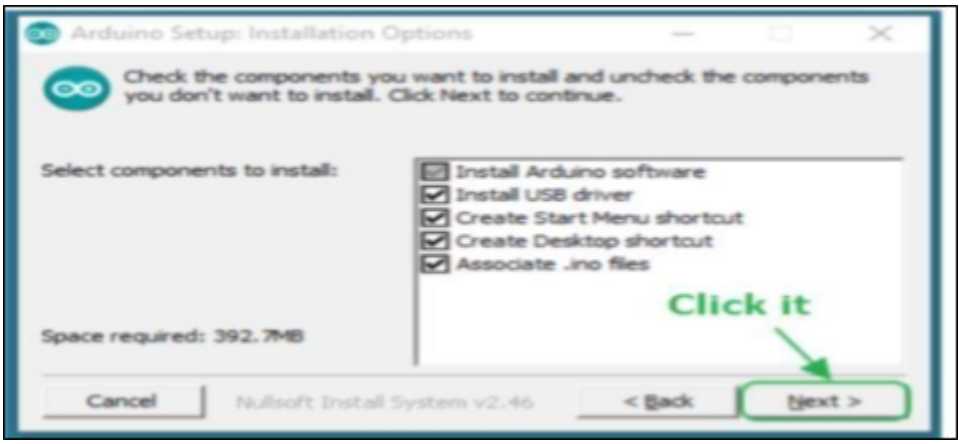
The Arduino Software (IDE) – contains:

- A text editor for writing code
- A message area
- A text consoles
- A toolbar with buttons for common functions and a series of menus.It connects to the Arduino hardware to upload programs and communicate with them.

Installation of Arduino Software (IDE)

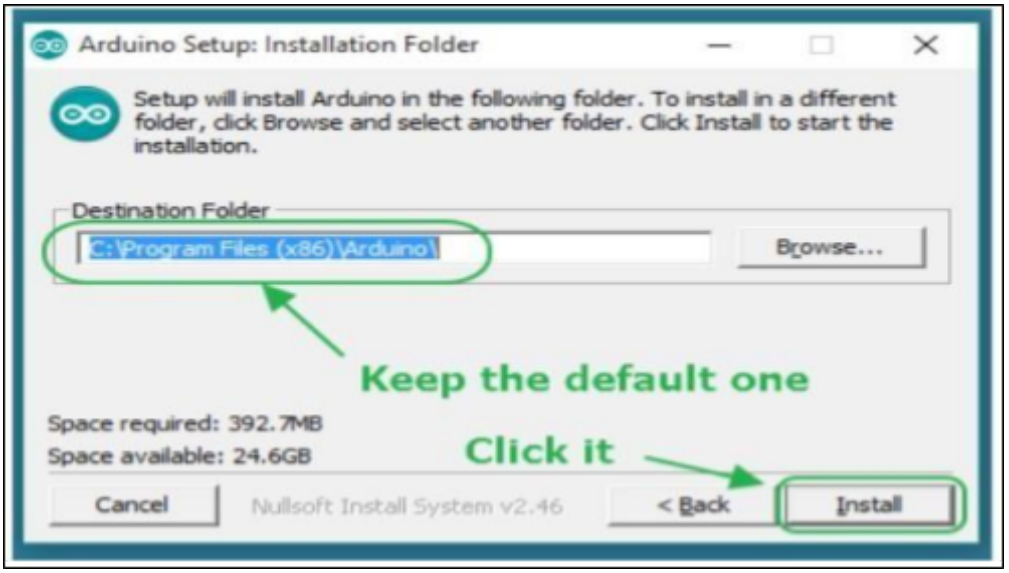
Step1: Downloading

- To install the Arduino software, download this page: http://arduino.cc/en/Main/Software and proceed with the installation by allowing the driver installation process.



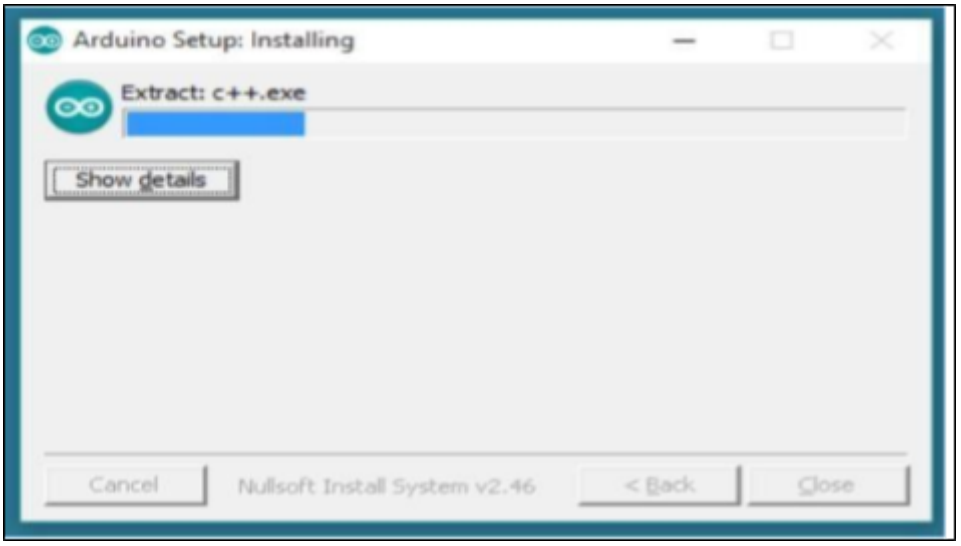
Step 2: Directory Installation

Choose the installation directory.



Step 3: Extraction of Files

- The process will extract and install all the required files to execute properlythe Arduino Software (IDE).

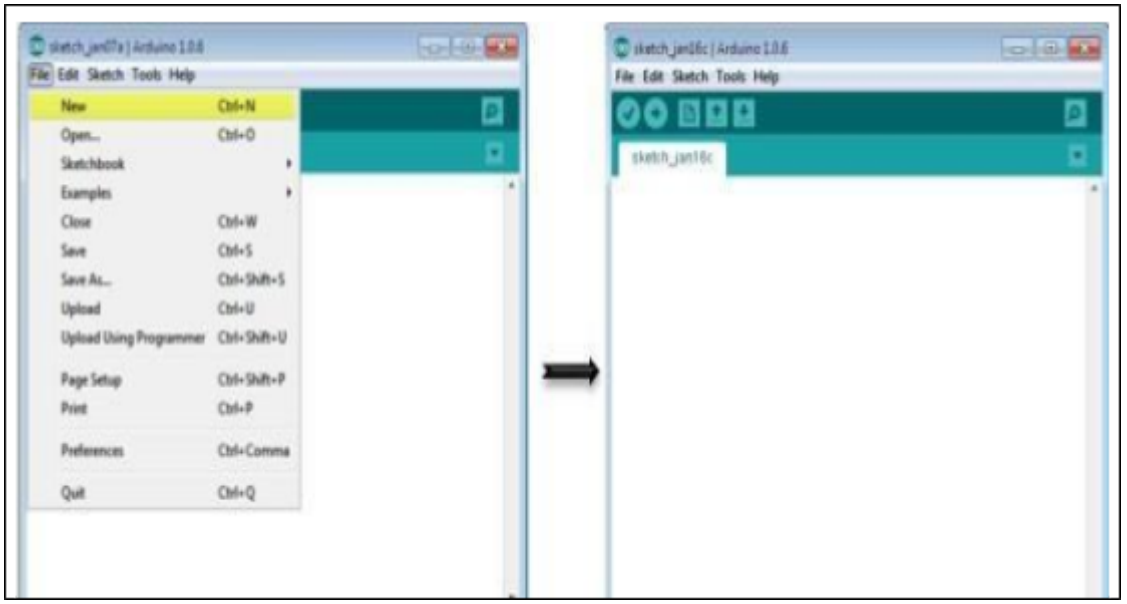


Step 4: Connecting the board

- The USB connection with the PC is necessary to program the board and not just to power it up. The Uno and Mega automatically draw power fromeither the USB or an external power supply. Connect the board to the computer using the USB cable. The green power LED (labelled PWR) should go on.

Step 5: Working on the new project

- Open the Arduino IDE software on your computer. Coding in the Arduinolanguage will control your circuit.
- Open a new sketch File by clicking on New.



Exp. No. 1	Demonstrate the LED Blinking and GPIO Control
Date:	

AIM:

To demonstrate digital output control using GPIO pins by blinking an LED with a Microcontroller

HARDWARE & SOFTWARE REQUIRED:

S. No.	Component	Specification / Example	Quantity
1	Microcontroller Board	ESP32 / Arduino Uno	1
2	LED	5mm	1
3	Resistor	220Ω or 330Ω	1
4	Breadboard	Standard	1
5	Jumper Wires	Male to Male	As required
6	USB Cable	Micro USB / Type B	1
7	PC with Arduino IDE	Installed with ESP32/Arduino support	1

THEORY:

GPIO (General Purpose Input Output) pins are used in microcontrollers to control external devices like LEDs, sensors, or motors.
When configured as **digital output**, a GPIO pin can be set to:

- HIGH (typically 3.3V or 5V) → LED turns ON
- LOW (0V) → LED turns OFF

LED blinking is the basic test to validate that:

- The microcontroller is powered and programmable
- GPIO configuration and timing delays are functional

PROCEDURE

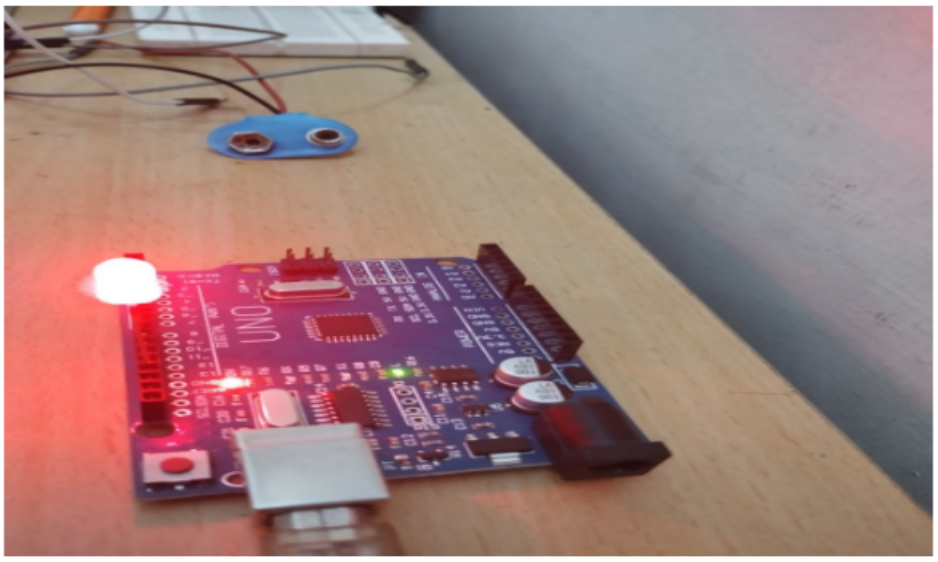
Connect the longer leg (anode) of the LED to GPIO pin (e.g., D2 or GPIO 2) through a 220Ω resistor.

1. Connect the **shorter leg (cathode)** of the LED to GND.
2. Open **Arduino IDE** and select the correct board (ESP32/Arduino UNO) and COM port.
3. Write a code to set the GPIO as output and toggle it ON/OFF with a delay.
4. Upload the code to the microcontroller.
5. Observe the LED blinking on and off periodically.

CODE

```
#define LED_PIN 2  // Use GPIO 2 for ESP32 or pin 13 for Arduino Uno
void setup() {
  pinMode(LED_PIN, OUTPUT); // Set GPIO as output
}
void loop() {
  digitalWrite(LED_PIN, HIGH);  // LED ON
  delay(1000);                  // Wait 1 second
  digitalWrite(LED_PIN, LOW);   // LED OFF
  delay(1000);                  // Wait 1 second
}
```

OUTPUT



PRE-VIVA QUESTIONS

1. What is a GPIO pin and what are its modes?
2. How does an LED behave when connected to a GPIO pin?
3. Why do we need a resistor in series with the LED?
4. What is the purpose of `pin Mode()` in Arduino?
5. What is the default voltage level for HIGH and LOW in ESP32/Arduino?

POST-VIVA QUESTIONS

1. Which pin did you use for LED blinking and why?
2. What will happen if you don't use a resistor with the LED?
3. Can we use multiple GPIOs to blink multiple LEDs simultaneously?
4. How would you change the blink speed in the code?
5. Suggest one real-world application of GPIO-based digital output.

RESULTS:

The LED connected to the GPIO pin blinked at 1-second intervals, confirming successful GPIO configuration and digital output control using the microcontroller.

Exp. No. 2	Perform Analog Sensor Interfacing
Date:	

AIM:

To interface an analog sensor with a microcontroller and read real-time analog values using the ADC, enabling digital monitoring of physical parameters like temperature or light.

HARDWARE & SOFTWARE REQUIRED:

S. No.	Component	Specification/Example	Quantity
1	Microcontroller Board	Arduino UNO / ESP32	1
2	Analog Sensor	LM35 / LDR / Potentiometer	1
3	Jumper Wires	Male to Male	As required
4	Breadboard	Standard	1
5	USB Cable	For Programming	1
6	PC with Arduino IDE	With board support installed	1

THEORY:

THEORY

Analog sensors produce voltage signals that vary with physical conditions. These signals are read using ADC pins on microcontrollers like Arduino or ESP32. The ADC converts the continuous voltage input into digital values (0–1023 for Arduino Uno). By applying calibration formulas (e.g., LM35 → 10mV/°C), the readings can be interpreted into real-world quantities.

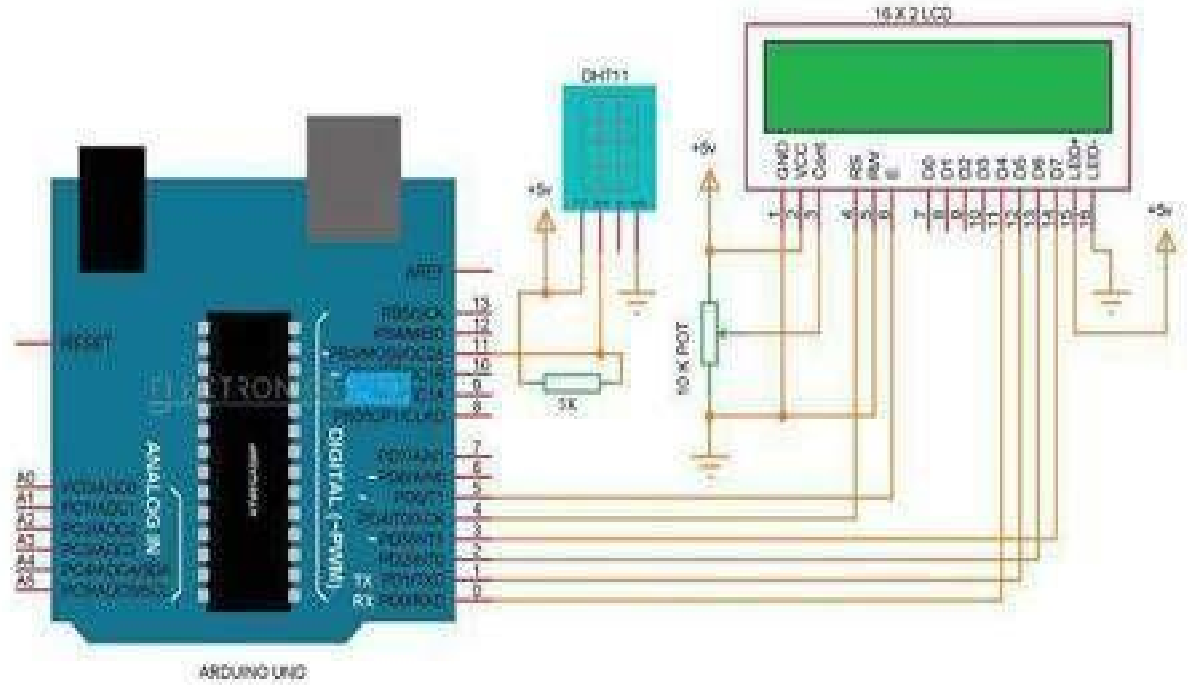
PROCEDURE

1. Connect Vcc and GND of the sensor to the 5V/3.3V and GND pins.
2. Connect the sensor's output to an analog input pin (A0 on Arduino or GPIO36 on ESP32).
3. Launch Arduino IDE and select the board and COM port.
4. Write code to read from analog pin using `analogRead()`.
5. Upload the code and open the Serial Monitor.
6. Observe the real-time analog values and their interpretation (e.g., temperature).

CODE

```
#define SENSOR_PIN A0  // Use GPIO36 for ESP32
void setup() {
  Serial.begin(9600);
}
void loop() {
  int analogValue = analogRead(SENSOR_PIN);
  float voltage = analogValue * (5.0 / 1023.0);
  float temperature = voltage * 100.0;

  Serial.print("Analog Value: ");
  Serial.print(analogValue);
  Serial.print(" | Temperature (°C): ");
  Serial.println(temperature);
  delay(1000);
}
```

OUTPUT



PRE-VIVA QUESTIONS

1. What is an analog signal?
2. Define ADC and its purpose in microcontrollers.
3. What does the `analogRead()` function return?
4. What is the range of ADC output for Arduino Uno?
5. Name two analog sensors used in embedded systems.

POST-VIVA QUESTIONS

1. What analog values did you observe during the experiment?
2. How was the sensor voltage converted into temperature?
3. What resolution is provided by Arduino's ADC?
4. Can the analog input pin be used for digital output?
5. Suggest one IoT application using analog sensor data.

RESULTS:

The analog signal from the sensor was successfully read and processed by the microcontroller. The temperature/light levels were displayed in real time on the Serial Monitor.

Exp. No. 3	**Demonstrate PWM and Servo Motor Control**
Date:	

AIM:

To demonstrate the generation of PWM signals and control the position of a servo motor using a microcontroller.

HARDWARE & SOFTWARE REQUIRED:

S. No.	Component Name	Specification / Example	Quantity
1	Microcontroller Board	Arduino UNO / ESP32 / STM32	1
2	Servo Motor	SG90 / MG996R	1
3	Breadboard	Standard Size	1
4	Jumper Wires	Male-to-Male / Male-to-Female	As required
5	Power Supply	USB 5V or External 5V Adapter	1
6	Resistors (Optional)	220Ω or as needed	As required
7	PC with Programming Software	Arduino IDE / PlatformIO	1

THEORY:

PWM and Servo Motor Control

Pulse Width Modulation (PWM) is a technique used to generate analog-like control using digital signals. By varying the **duty cycle** (i.e., the percentage of time the signal stays high in one cycle), devices such as motors, LEDs, and fans can be controlled with high precision and efficiency.

A **servo motor** is a rotary actuator that allows precise control of angular position. It consists of a DC motor, a gearbox, a position sensor (usually a potentiometer), and a control circuit. Servo motors typically operate between 0° to 180°.

Working Principle of PWM with Servo Motor:

- Servo motors are controlled by PWM signals, where the width of the pulse determines the angle of the shaft.
- A standard servo expects a pulse every **20 ms (50 Hz)**:
    - A **1 ms** pulse rotates the motor to **0°**
    - A **1.5 ms** pulse moves it to **90°**
    - A **2 ms** pulse rotates it to **180°**
-

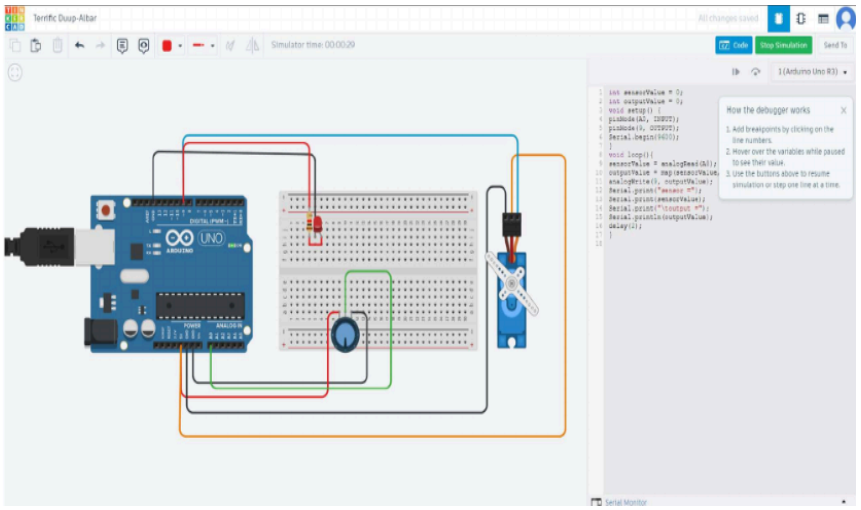
CODE

-
-
-
- {
- servo_9.attach(9);
- }
- void loop()
- {
- for (pos = 0; pos<= 180; pos += 1) {
- servo_9.write(pos);
- delay(15);
- }
- for (pos = 180; pos>= 0; pos -= 1) {
- servo_9.write(pos);
- delay(15);
- }
- }
-
-
-
- CODE:
- int sensorValue = 0;
- int outputValue = 0;
- void setup() {
- pinMode(A0, INPUT);
- pinMode(9, OUTPUT);
- Serial.begin(9600);
- }
- void loop(){
- sensorValue = analogRead(A0);
- outputValue = map(sensorValue, 0, 1023, 0, 255);
- analogWrite(9, outputValue);
- Serial.print("sensor =");
- Serial.print(sensorValue);
- Serial.print("\toutput =");
- Serial.println(outputValue);
- delay(2);

```
#include <Servo.h>
Servo myServo;      // Create servo object
int angle = 0;      // Variable to store servo position
void setup() {
  myServo.attach(9); // Connect servo signal pin to digital pin 9
  Serial.begin(9600);
}
void loop() {
  for (angle = 0; angle <= 180; angle += 10) { // Sweep from 0 to 180 degrees
    myServo.write(angle);
    Serial.print("Moving to angle: ");
    Serial.println(angle);
    delay(500);
```

```
  }
  for (angle = 180; angle >= 0; angle -= 10) { // Sweep back from 180 to 0
    myServo.write(angle);
    Serial.print("Moving to angle: ");
    Serial.println(angle);
    delay(500);
  }
}
```

OUTPUT



Pre-Viva Questions

1. What is PWM and how is it generated in microcontrollers?
2. How does PWM control the angle of a servo motor?
3. What is the role of the `Servo.h` library in Arduino?
4. What are the duty cycles for 0°, 90°, and 180° positions in servo motors?
5. Can a digital output pin provide PWM signals? If yes, how?

Post-Viva Questions

1. What is the angle range you observed during the servo sweep?
2. Which pin was used for PWM output in your circuit?
3. How did the delay affect the servo motion?
4. What would happen if the PWM signal frequency was too low or too high?
5. Suggest one real-time application where PWM and servo motors are used.

RESULT

The servo motor successfully responded to PWM signals generated by the microcontroller. It rotated smoothly from 0° to 180° and back, demonstrating precise angle control using Pulse Width Modulation.

Exp. No. 4	**Demonstrate UART communication between the microcontroller and a PC or another microcontroller**
Date:	

AIM

To implement and verify UART-based asynchronous serial communication between a microcontroller (Arduino UNO) and a PC or peer microcontroller, enabling real-time data exchange for use in embedded system applications such as sensor data logging, inter-device communication, and debugging.

HARDWARE & SOFTWARE REQUIRED:

S. No.	Component	Specification	Quantity
1	Arduino UNO Board	ATmega328P	2
2	USB Cable	Type-A to B	1
3	Jumper Wires	Male to Male	As required

4	PC with Arduino IDE	Serial Monitor Enabled	1
5	Breadboard (Optional)	Standard Size	1

THEORY

UART (Universal Asynchronous Receiver/Transmitter) is a serial communication protocol that uses **asynchronous** data transfer. It involves two main pins:

- **TX (Transmit)**
- **RX (Receive)**

The Arduino uses the Serial library for UART communication. Data is transmitted at a specified **baud rate** (bits per second), e.g., 9600 bps.

In this experiment:

- Arduino A sends data through TX
- Arduino B or PC receives data through RX

PROCEDURE

- Connect **the Arduino to the PC** using a USB cable to establish UART communication through the built-in USB-to-serial converter.
- Open **Arduino IDE** on your computer and select the correct **Board** and **COM Port** from the **Tools** menu.
- Write **the UART communication code** using `Serial.begin()`, `Serial.print()`, or `Serial.read()` functions to transmit or receive data.
- Upload **the code** to the Arduino board using the **Upload** button in the IDE.
- Open **Serial Monitor** from the Tools menu and set the baud rate (e.g., 9600) to match the `Serial.begin()` setting in the code.
- Observe **the transmitted or received data** in the Serial Monitor or on the receiving device (another Arduino, if used), verifying UART communication is working.

CODE

```
void setup() {
  Serial.begin(9600);   // Start UART
}
void loop() {
  Serial.println("Hello from Arduino!");
  delay(1000);
}
```

Transmitter Code (Arduino A):

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.println("Data to Arduino B");
  delay(1000);
```

}

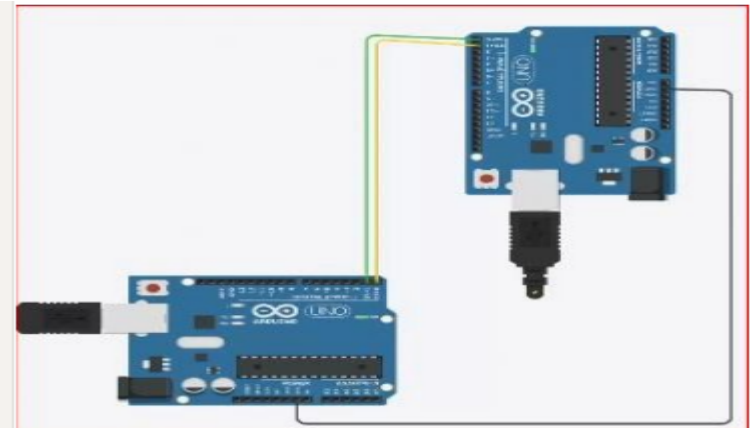
Receiver Code (Arduino B):

```
 void setup() {
   Serial.begin(9600);
 }
 void loop() {
   if (Serial.available()) {
     String data = Serial.readStringUntil('\n');
     Serial.print("Received: ");
     Serial.println(data);
   }
 }
```

-
-
-
- {
- servo_9.attach(9);
- }
- void loop()
- {
- for (pos = 0; pos<= 180; pos += 1) {
- servo_9.write(pos);
- delay(15);
- }
- for (pos = 180; pos>= 0; pos -= 1) {
- servo_9.write(pos);
- delay(15);
- }
- }
-
-
-
- CODE:
- int sensorValue = 0;
- int outputValue = 0;
- void setup() {
- pinMode(A0, INPUT);
- pinMode(9, OUTPUT);
- Serial.begin(9600);
- }
- void loop(){
- sensorValue = analogRead(A0);
- outputValue = map(sensorValue, 0, 1023, 0, 255);
- analogWrite(9, outputValue);
- Serial.print("sensor =");
- Serial.print(sensorValue);
- Serial.print("\toutput =");
- Serial.println(outputValue);

- delay(2);

OUTPUT



Pre-Viva Questions

- What **is UART? Explain its basic working principle.**
- Differentiate **between synchronous and asynchronous communication.**
- What **are the key pins used in UART communication on an Arduino?**
- What **is the function of the** `Serial.begin()` **command in Arduino?**
- What **is a baud rate, and why must it match on both devices in UART?**

Post -Viva Questions

1. **What output did you observe on the Serial Monitor?**
2. **What happens if TX and RX pins are connected incorrectly?**
3. **How did you verify successful data transmission and reception?**
4. **Can UART communication occur simultaneously in both directions? Explain.**
5. **How would you implement UART communication between two Arduino boards?**

RESULTS:

UART communication was successfully established between the Arduino and the PC/another Arduino. Data was accurately transmitted and received using serial functions at the specified baud rate.

Exp. No.5	Perform basic HTTP requests with microcontroller and Wifi Network
Date:	

AIM

To implement and verify HTTP client functionality on a Wi-Fi-enabled microcontroller (ESP32) for executing HTTP GET and POST requests, enabling RESTful communication with web servers in IoT-based networked applications.

HARDWARE & SOFTWARE REQUIRED:

S.No	Component	Specification	Quantity
1	ESP32 Board	Wi-Fi enabled	1
2	USB Cable	Micro USB	1
3	PC with Arduino IDE	Installed with ESP32	1
4	Wi-Fi Connection	SSID + Password	1

Theory

- **HTTP (Hypertext Transfer Protocol)** is the standard protocol for web communication.
- **GET Request**: Requests data from a server (e.g., read sensor data).
- **POST Request**: Sends data to a server (e.g., upload sensor readings).
- ESP32, with Wi-Fi, can connect to the internet and communicate with web APIs or cloud servers using these methods.

PROCEDURE

- Connect **ESP32 to PC** and open **Arduino IDE**.
- Install **ESP32 Board Support** via Boards Manager if not already done.
- Include **`WiFi.h` and `HTTPClient.h`** libraries.
- Write **code** for HTTP GET/POST requests using Wi-Fi credentials and server URL.
- Upload **the code** to ESP32 and open the **Serial Monitor**.
- Observe **responses** from the web server printed to the Serial Monitor.

CODE

HTTP GET Request

```
#include <WiFi.h>
#include <HTTPClient.h>
const char* ssid = "Your_SSID";
const char* password = "Your_PASSWORD";
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting...");
  }
  Serial.println("Connected!");
  HTTPClient http;
  http.begin("http://example.com/data"); // Replace with actual URL
  int httpCode = http.GET(); // Send GET request
  if (httpCode > 0) {
    String payload = http.getString();
    Serial.println(payload); // Print response
  } else {
    Serial.println("GET request failed");
  }
  http.end();
}
void loop() {
}
```

HTTP POST Request

```
#include <WiFi.h>
#include <HTTPClient.h>
const char* ssid = "Your_SSID";
const char* password = "Your_PASSWORD";
void setup() {
  Serial.begin(115200);
```

```
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting...");
  }
  Serial.println("Connected!");
  HTTPClient http;
  http.begin("http://example.com/post"); // Replace with real server URL
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");
  int httpCode = http.POST("sensor=temperature&value=30"); // key-value pair
  if (httpCode > 0) {
    String response = http.getString();
    Serial.println(response);
  } else {
    Serial.println("POST request failed");
  }
  http.end();
}
void loop() {
}
```

Pre-Viva Questions

1. What is the difference between HTTP GET and POST methods?
2. What is the role of the `WiFi.h` and `HTTPClient.h` libraries in Arduino?
3. How does a microcontroller like ESP32 connect to a Wi-Fi network?
4. What is RESTful communication in the context of IoT?
5. Why is HTTP preferred for web-based IoT applications?

Post-Viva Questions

1. What was the observed output from the GET/POST request in the Serial Monitor?
2. How did you ensure the ESP32 was connected to the internet before sending a request?
3. What would happen if the server URL is incorrect or the server is down?
4. How can you handle HTTP response codes in ESP32?
5. Suggest a real-time application where HTTP requests from ESP32 are practically used.

RESULT

The ESP32 microcontroller successfully established a Wi-Fi connection and performed HTTP GET and POST requests.

Exp. No.6	**Experiment interfacing sensors /LCD using PC**
Date:	

AIM

To implement interfacing of sensors or an LCD module with a microcontroller and establish serial communication with a PC.

HARDWARE & SOFTWARE REQUIRED:

S. No.	Component	Specification / Example	Quantity
1	Microcontroller Board	Arduino UNO / ESP32	1
2	Sensor Module	e.g., LM35 / DHT11 / LDR	1
3	LCD Display	16x2 LCD (with I2C optional)	1
4	USB Cable	Type A to B / Micro USB	1
5	Jumper Wires	Male to Male	As required
6	Breadboard (optional)	Standard	1
7	PC with Arduino IDE	With Serial Monitor	1

THEORY

Interfacing peripherals like **sensors** or **LCDs** with microcontrollers allows embedded systems to interact with the physical world.

- Sensors detect physical parameters (e.g., temperature, light) and provide electrical signals.
- LCDs are used to display sensor values or messages locally.
- Using **UART**, the microcontroller communicates with a **PC**, transmitting sensor data for visualization or logging.

PROCEDURE

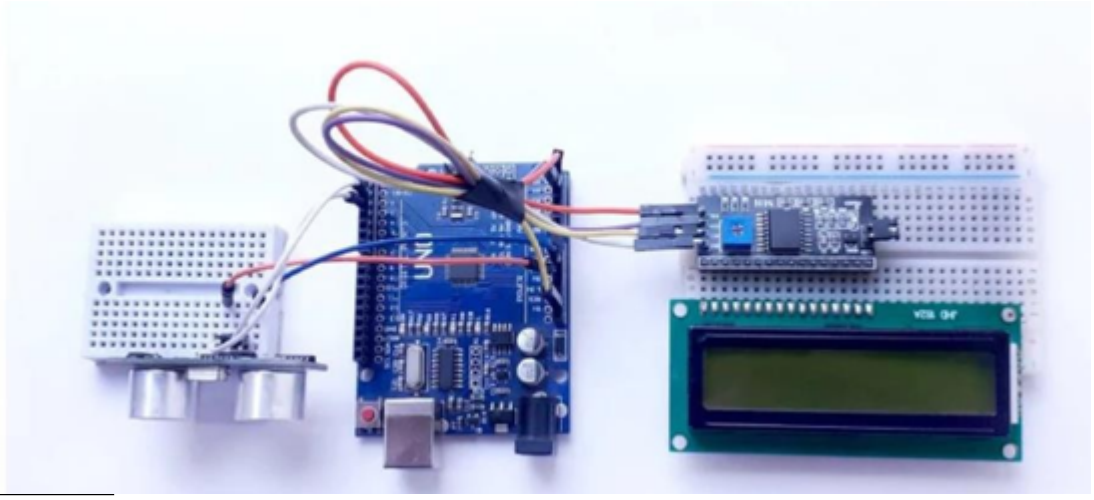
1. Connect the sensor or LCD to the Arduino as per the pin configuration.
2. Open Arduino IDE, select the correct board and COM port.
3. Write code to read sensor values or print data to the LCD.
4. Use `Serial.begin()` and `Serial.print()` to send data to PC.
5. Upload the code to the Arduino.
6. Open Serial Monitor to view sensor data or message output.

CODE

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
 LiquidCrystal_I2C lcd(0x27, 16, 2);
```

```
const int trigPin = 3;
const int echoPin = 2; long duration;
int distance; void setup() {
lcd.begin(); // Initializes the interface to the LCD display lcd.backlight();
pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
Serial.begin(9600);
}
 void loop() { lcd.clear();
digitalWrite(trigPin, LOW);
delayMicroseconds(2); digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
duration = pulseIn(echoPin, HIGH);
distance = duration * 0.0340 / 2;
Serial.println("Distance"); Serial.println(distance);
lcd.setCursor(0, 0); lcd.print("Distance: ");
lcd.print(distance);
lcd.print("cm"); delay(1000);
}
```

OUTPUT



PRE-VIVA QU

1. What is UART and how is it used in Arduino?
2. How do sensors communicate with microcontrollers?
3. What are the standard pins for connecting a 16x2 LCD to Arduino?
4. What is the purpose of `Serial.begin()` and `Serial.print()`?

Exp. No.7	Implement Multitasking On ESP 232 Using Free RTOS
Date:	

5. What role does the PC play in embedded interfacing experiments?

POST-VIVA QUESTIONS:

1. What output did you observe on the Serial Monitor or LCD?
2. How did the microcontroller process and send sensor data to the PC?
3. What challenges did you face in interfacing and how did you resolve them?
4. How can this system be used in a practical IoT application?
5. How would you modify the code to send data to the cloud instead of a PC?

RESULT

The microcontroller successfully interfaced with the sensor/LCD and communicated with the PC via UART.Real-time data was accurately transmitted to or received from the PC, validating the interface and communication.

AIM:

To implement multitasking on an ESP32 microcontroller using Free RTOS to manage concurrent task execution.

HARDWARE & SOFTWARE REQUIRED:

S. No.	Component	Specification / Example	Quantity

1	ESP32 Development Board	Dual-core, Wi-Fi + Bluetooth	1
2	USB Cable	Micro USB	1
3	LED (Optional)	5mm	2
4	Resistors	220Ω	2
5	Jumper Wires	Male to Male	As required
6	Breadboard (Optional)	Standard	1
7	PC with Arduino IDE	ESP32 board package installed	1

THEORY

Free RTOS (Free Real-Time Operating System) is a lightweight open-source RTOS designed for embedded systems. It supports **pre-emptive multitasking**, enabling multiple tasks (threads) to run seemingly in parallel. The ESP32 supports FreeRTOS natively and can execute multiple functions simultaneously using `xTaskCreate()`.

In embedded applications, multitasking allows better CPU utilization, modular code, and responsiveness, such as blinking LEDs while monitoring a sensor or handling communication tasks.

FreeRTOS handles:

- **Task scheduling** (based on priority)
- **Task delays (`vTaskDelay`)**
- **Resource sharing** (using semaphores, mutexes)
- **Concurrent execution** using context switching

CODE

```
void Task1(void *pvParameters) {
  while (1) {
    Serial.println("Task 1 is running");
    vTaskDelay(1000 / portTICK_PERIOD_MS);  // Delay 1 second
  }
}
void Task2(void *pvParameters) {
  while (1) {
```

```
    Serial.println("Task 2 is running");
    vTaskDelay(500 / portTICK_PERIOD_MS);  // Delay 0.5 seconds
  }
}
void setup() {
  Serial.begin(115200);
  xTaskCreate(Task1, "Task 1", 1000, NULL, 1, NULL);
  xTaskCreate(Task2, "Task 2", 1000, NULL, 1, NULL);
}
void loop() {
  // Empty - FreeRTOS manages tasks
}
```

Pre-Viva Questions

1. What is FreeRTOS and why is it used in embedded systems?
2. How does multitasking improve system performance in microcontrollers?
3. What is the syntax of the `xTaskCreate()` function in FreeRTOS?
4. How does FreeRTOS decide which task to run when multiple tasks are ready?
5. What are task priority and stack size in the context of FreeRTOS?

Post-Viva Questions

1. How many tasks did you implement in your FreeRTOS code on ESP32?
2. What was the behavior observed when two tasks had equal priority?
3. How does `vTaskDelay()` affect task scheduling in FreeRTOS?
4. Can you explain how task switching happens in FreeRTOS on ESP32?
5. What would happen if one task enters an infinite loop without a delay?

RESULT

Multitasking was successfully implemented on the ESP32 using FreeRTOS.

Exp. No.8	**Demonstrate Bluetooth Low Energy Communication**
Date:	

AIM

To demonstrate Bluetooth Low Energy (BLE) communication using the ESP32 microcontroller. This enables wireless data exchange between ESP32 and mobile devices using the BLE GATT protocol for IoT applications.

HARDWARE & SOFTWARE REQUIRED:

S. No.	Component	Specification / Example	Quantity
1	ESP32 Development	Wi-Fi + BLE supported	1

	Board		
2	USB Cable	Micro USB	1
3	Smartphone	With BLE Scanner app	1
4	Jumper Wires (Optional)	For sensors/indicators	As needed
5	PC with Arduino IDE	With ESP32 board support	1

THEORY

Bluetooth Low Energy (BLE) is a wireless communication technology designed for short-range, low-power data exchange. Unlike classic Bluetooth, BLE is optimized for intermittent communication and reduced energy consumption, making it ideal for IoT applications.

The ESP32 microcontroller has built-in BLE capabilities and supports acting as:

- **Peripheral** (advertises services to a central device like a phone)
- **Central** (connects to a peripheral like a sensor)

Communication is based on the **GATT** (Generic Attribute Profile), consisting of:

- **Services**: Grouped characteristics
- **Characteristics**: Actual data points (e.g., temperature value)

PROCEDURE

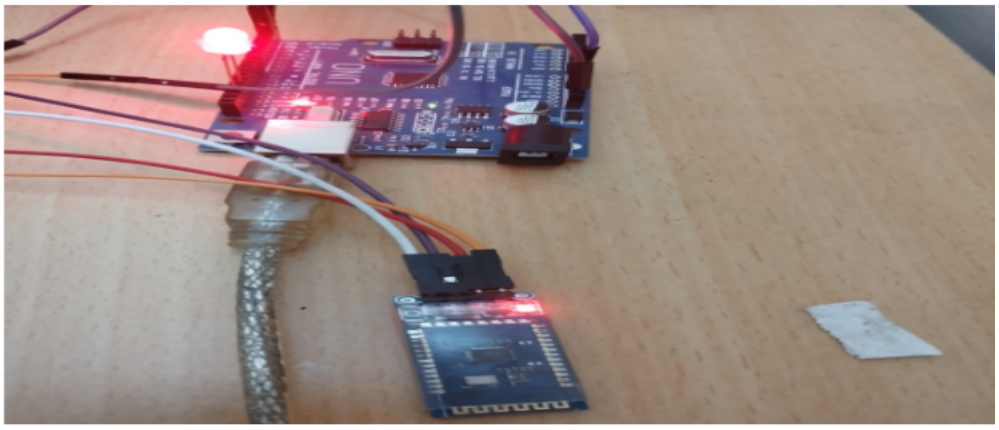
1. Connect ESP32 to PC via USB and open Arduino IDE.
2. Install **ESP32 board support** and include the BLE library.
3. Write code to initialize BLE, advertise a custom service and characteristic.
4. Upload the code to ESP32.
5. Open a **BLE scanner app** (e.g., nRF Connect or LightBlue) on your phone.
6. Connect to the ESP32 and read/write the characteristic to verify communication.

CODE

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
BLECharacteristic *pCharacteristic;
bool deviceConnected = false;
#define SERVICE_UUID "91bad492-b950-4226-aa2b-4ede9fa42f59"
#define CHARACTERISTIC_UUID "cba1d466-344c-4be3-ab3f-189f80dd7518"
class MyServerCallbacks : public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
  };
  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
  }
};

void setup() {
```

```
  Serial.begin(115200);
  BLEDevice::init("ESP32_BLE");
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  BLEService *pService = pServer->createService(SERVICE_UUID);
  pCharacteristic = pService->createCharacteristic(
                      CHARACTERISTIC_UUID,
                      BLECharacteristic::PROPERTY_READ |
                      BLECharacteristic::PROPERTY_WRITE
                    );

  pCharacteristic->setValue("Hello from ESP32!");
  pService->start();

  BLEAdvertising *pAdvertising = pServer->getAdvertising();
  pAdvertising->start();
  Serial.println("BLE service is now advertising...");
}
void loop() {
  if (deviceConnected) {
    Serial.println("Device is connected!");
  }
  delay(2000);
}
```

OUTPUT



PRE-VIVA QUESTIONS

1. What is the difference between classic Bluetooth and Bluetooth Low Energy (BLE)?
2. What are GATT services and characteristics in BLE?
3. What is the role of a BLE peripheral and central?
4. Why is BLE suitable for IoT applications?
5. Which BLE libraries are required for ESP32 in Arduino IDE?
6.

POST-VIVA QUESTIONS

1. How did you verify successful BLE communication between ESP32 and smartphone?
2. What was the UUID of your custom service and characteristic?
3. How can you enable notification/indication for a BLE characteristic?
4. What happens when the central disconnects from the ESP32 BLE peripheral?
5. Suggest a real-world IoT use case for BLE-enabled ESP32.

RESULT

BLE communication was successfully established between ESP32 and a smartphone.

Ex No: 9	CONTENT BEYOND SYLLABUS
Date:	DESIGN SOLUTIONS FOR SMART HOME USING ARDUINO PROCESSOR

AIM

To Design a smart home system using an Arduino processor involves integrating various sensors, actuators, and communication modules to automate and control different aspects of the home environment.

SYSTEM AND SOFTWARE TOOLS REQUIRED

1. ARM Evaluation Kit
2. ARM Development tools
3. LCD
4. KEIL C software

PROGRAM: 1 Lighting Control

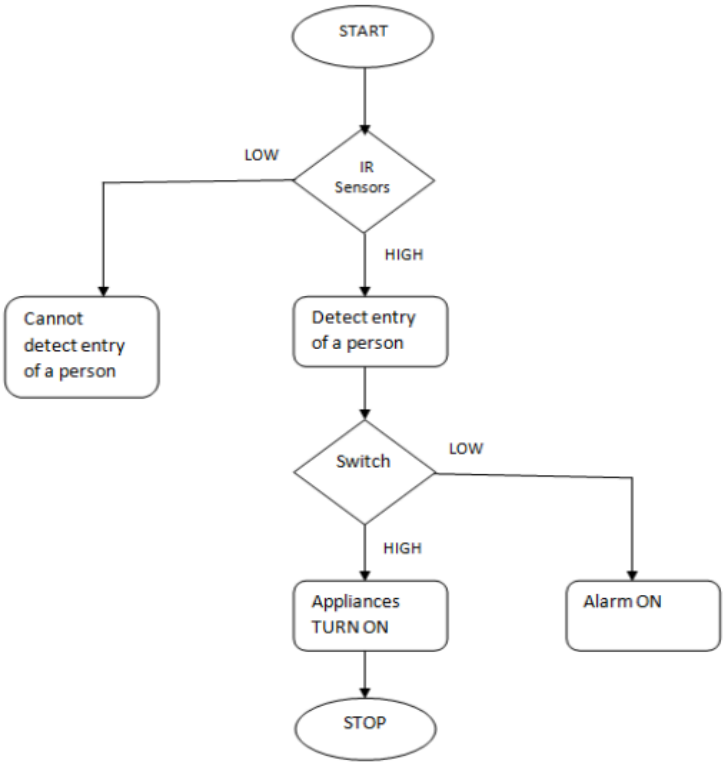
```
int ldrPin = A0;      // LDR connected to analog pin A0
int relayPin = 8;     // Relay module connected to digital pin 8
int ldrValue;
void setup() {
  pinMode(relayPin, OUTPUT);
```

```
}

void loop() {
  ldrValue = analogRead(ldrPin);
  if (ldrValue < 300) { // Dark condition
    digitalWrite(relayPin, HIGH); // Turn on light
  } else {
    digitalWrite(relayPin, LOW); // Turn off light
  }
  delay(1000); // Delay for 1 second
}
```

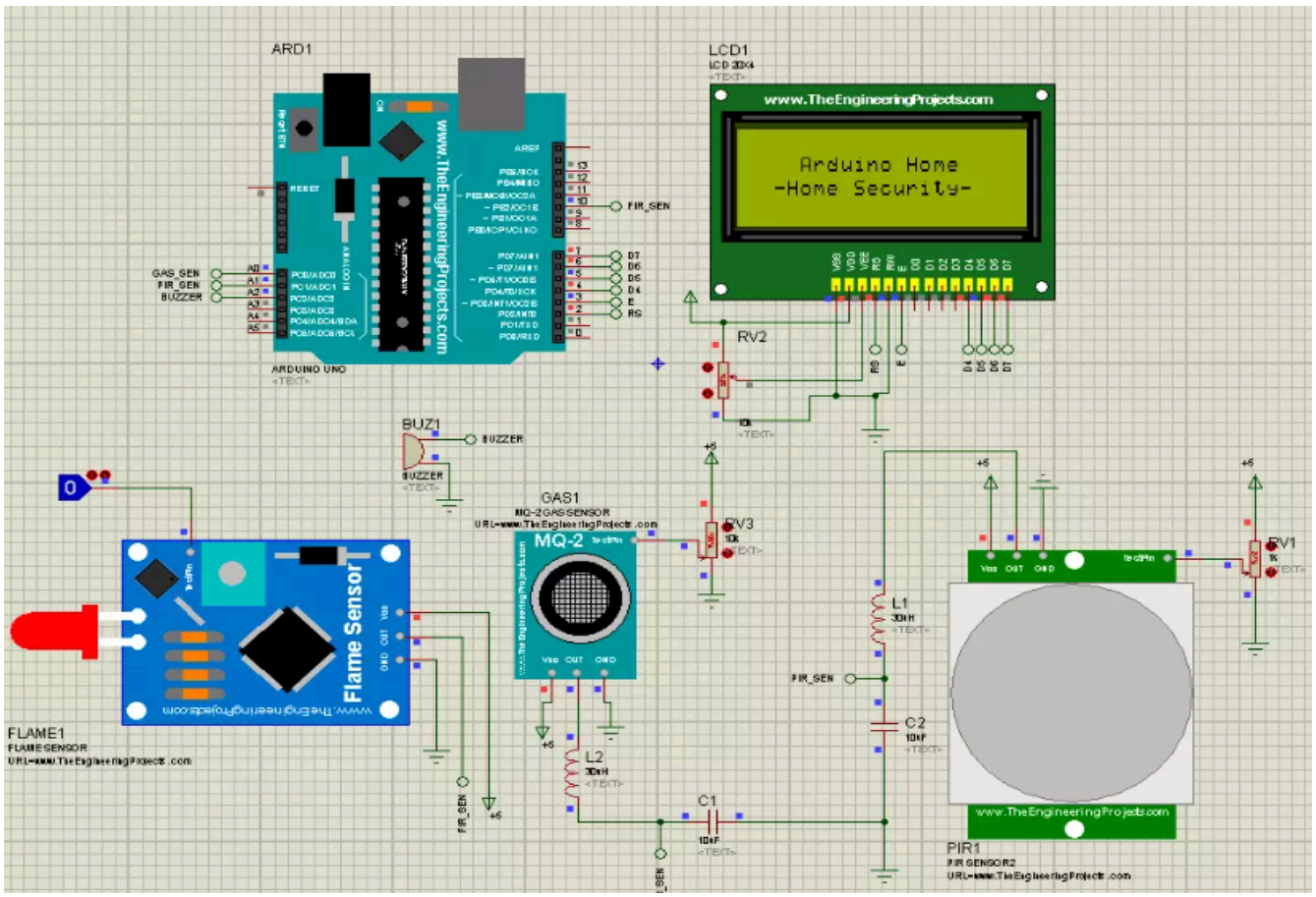
PROGRAM: 2 Temperature Control

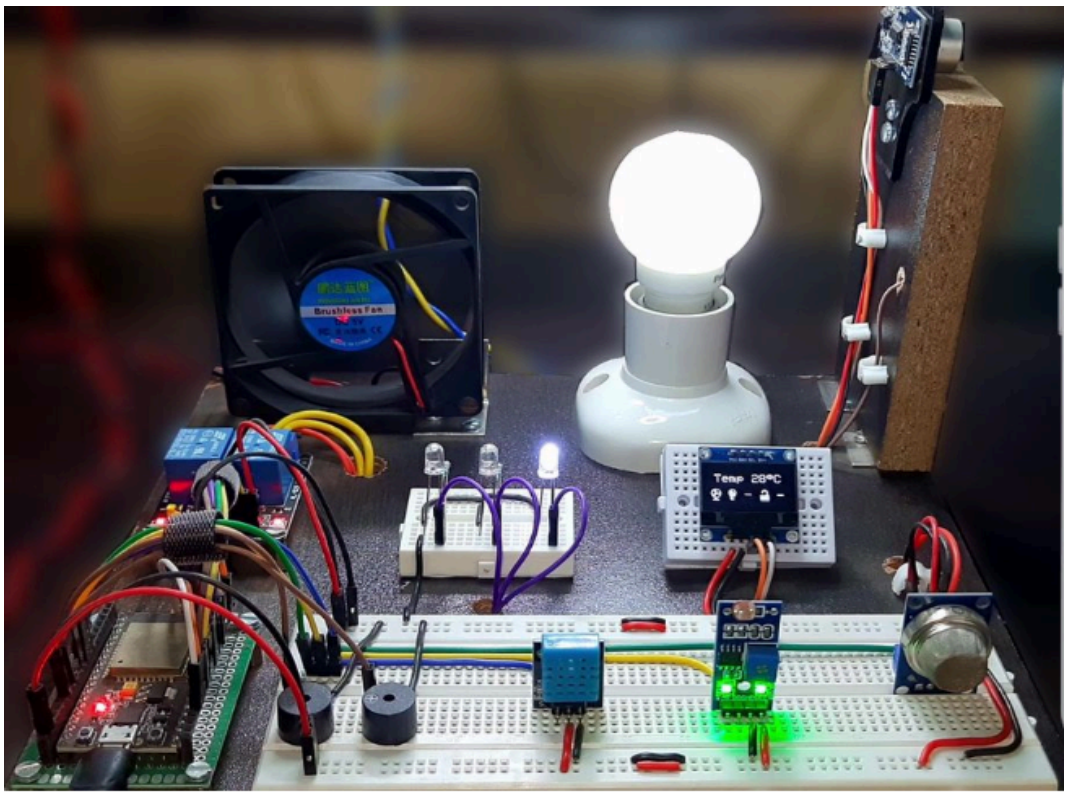
```
#include <DHT.h>
#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

int relayPin = 8; // Relay module connected to digital pin 8
float temperature;

void setup() {
  pinMode(relayPin, OUTPUT);
  dht.begin();
}

void loop() {
  temperature = dht.readTemperature();
  if (temperature > 25.0) { // If temperature is above 25°C
    digitalWrite(relayPin, HIGH); // Turn on fan
  } else {
    digitalWrite(relayPin, LOW); // Turn off fan
  }
  delay(2000); // Delay for 2 seconds
}
```

FLOWCHART:


START
IR Sensors
LOW
HIGH
Cannot detect entry of a person
Detect entry of a person
Switch
LOW
HIGH
Appliances TURN ON
Alarm ON
STOP

OUTPUT:


ARD1
ARDUINO UNO
www.TheEngineeringProjects.com
LCD1
LCD 20X4
www.TheEngineeringProjects.com
Arduino Home
-Home Security-
RV2
BUZ1
BUZZER
GAS1
MQ-2 GAS SENSOR
MQ-2
RV3
RV1
L1
30mH
C2
10nF
L2
30mH
C1
10nF
FLAME1
FLAME SENSOR
Flame Sensor
PIR1
PIR SENSOR2
GAS_SEN
FIR_SEN
BUZZER


Brushless Fan
Temp 28°C

Pre-Lab Viva Questions:

1. What are the primary goals of the smart home system you intend to design?
2. List the sensors and actuators are required for your smart home system.
3. What is the role of the Arduino in your smart home system?
4. List some o the home automations applications.
5. What do you by user interface which is needed to control or monitor the new features?

Post-Lab Viva Questions:

1. What were the major challenges or limitations encountered during the design process?
2. List the communication module effective for remote control and monitoring.
3. How will you integrate additional features or expand the functionality of your smart home system?
4. What were the major challenges or limitations encountered during the design and implementation of the smart home system?
5. How did you ensure the reliability and robustness of the system during operation?

RESLUT

Thus, an embedded C program is developed to integrate additional features and functionality of the smart home automation system and the output is verified in the ARM kit.