SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

INTERNET OF THINGS LABORATORY

Course Code: CSE431

Semester: VII

Lab Manual

2024

SHANMUGHA ARTS, SCIENCE, TECHNOLOGY AND RESEARCH ACADEMY (SASTRA Deemed to be) University
Tirumalaisamudram, Thanjavur-613 401
School of Computing

Course Objective:

This course will help the learner to understand the networking mechanism of an IoT and to design IoT applications with Arduino (ATmega), ESP 32; Raspberry Pi& ARM based microcontroller boards.

List of Experiments:

1. Digital I/O Interface – Multicolour Led, IR Sensor, PIR, Slot Sensor
2. Analog Read and Write – Potentiometer, Temperature Sensor, Led Brightness Control
3. GPIO and Associated Peripheral Interfacing using Rpi
4. Camera Module Interface using Rpi
5. Deploying local server using Raspberry pi
6. Implementation of I2C / CAN
7. Configure and connection establishment of Wi-Fi with IoT development board (AP/RE/GW/Packages)
8. MQTT – Publish and Subscribe with IoT development board (Local Host as Broker)
9. MQTT – Publish and Subscribe with IoT development board (Web Broker) 10. Import Sensor data into cloud database (AWS/Azure/Firebase)

Using RTOS

11. Task Context Maintenance
12. Mutual Exclusion

Course Learning Outcomes:

- Experiment by interfacing Peripherals with Raspberry pi
- Connect two boards/devices using I2C / CAN
- Establish communication using MQTT protocols
- Incorporate Database for IoT applications
- Deploy Local/ web server to deliver the request from the user

Exercise No. 1 Digital I/O Interface – IR Sensor and LED

Objective:

To Interfacing IR sensor to the development board and turning ON/OFF the LED based on the object detection.

Components Required:

- ESP32 board, IR sensor, LED and Connecting Wires

Theory or Concept:

- An IR sensor is an electronic device that emits the light in order to sense some object of the surroundings. The emitter is simply an IR LED (Light Emitting Diode) and the

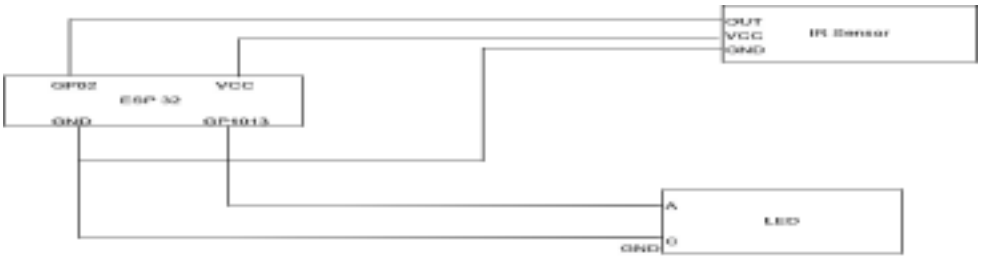
detector is simply an IR photodiode. When IR light falls on the photodiode, the resistances and the output voltages will change in proportion to the magnitude of the IR light received.

- The Light-emitting diode is a two-lead semiconductor light source. The LED is a special type of diode and they have similar electrical characteristics to a PN junction diode. The LED includes two terminals namely anode (+) and the cathode (-). Most of the LEDs have voltage ratings from 1 volt-3 volt whereas forward current ratings range from 200 mA-100 mA.

Procedure:

- Connect the IR sensor GND pin to the GND pin in the ESP32 board. •
Connect the IR sensor Vcc pin to the 3.3v pin in the ESP32 board.
- Connect the IR sensor OUT pin to any of the digital pins in the ESP32 board. •
Connect the LED Anode leg to any of the digital pins in the ESP32 board. •
Connect the LED Cathode leg to the GND pin in the ESP32 board.

Block Diagram:



Outcome:

This experiment demonstrates the basic concept of digital input (IR sensor) and output (LED) interfacing using a microcontroller, suitable for various applications like motion detection systems, presence sensing, or security alarms.

Sample use case:

Design and implement an automated entry system that utilizes a Passive Infrared (PIR) sensor for motion detection and an Infrared (IR) sensor for height estimation, enhancing security and user convenience.

Exercise No. 2 Analog Read and Write –Temperature Sensor and LED

Objective:

To Interfacing temperature sensor to the development board and turning ON/OFF

Components Required:

- ESP32 board, LM35 temperature sensor, LED and Connecting Wires

Theory or Concept:

The LM35 series are precision integrated-circuit temperature devices with an output voltage linearly- proportional to the Centigrade temperature. It uses the basic principle of a diode to measure known temperature value. As the temperature increases the voltage across a diode increases at a known rate. As the LM35 device draws only 60 μA from the supply, it has very low self-heating of less than 0.1°C in still air. The device is rated to operate over a -55°C to

150°C temperature range. Operating voltage range of LM35 is 4V-32V.

Procedure:

• Connect the GND pin of LM35 temperature sensor to the GND pin in ESP32 • board.
• Connect the Vcc pin of LM35 temperature sensor to the 5v pin in ESP32 board. • Connect the OUT pin of LM35 temperature sensor to the analog pin in ESP32 board. • Connect the LED Anode leg to any of the digital pins in the ESP32 board. • Connect the LED Cathode leg to the GND pin in the ESP32 board.

Block Diagram:

Outcome: The outcome of this experiment showcases how a temperature sensor's analog output can be read by a IoT board processed, and used to control room conditions.

Sample use case:

1. Design and implement an embedded system that controls the brightness of a Room according to temperature

2. Develop a real-time system using an ESP32 to read temperature data, and use a potentiometer to set threshold levels, triggering an LED to indicate temperature ranges

Exercise No. 3 GPIO and Associated Peripheral Interfacing using Rpi

Objective:

To interface IR Sensor to the Raspberry Pi.

Components Required:

• LED, IR Sensor, Wires, Raspberry Pi board

Theory or Concept:

When the circuit is operated the IR Sensor takes the input from the surrounding and passes digital values to the program and the LED is triggered based on the input from IR Sensor and output is reflected**.**

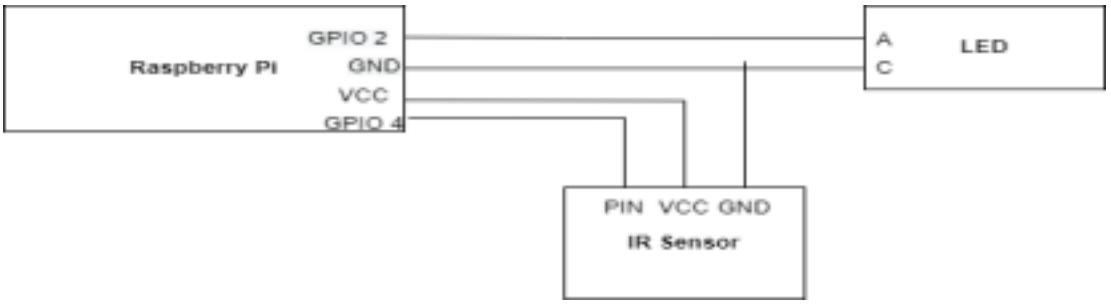
Basic Functions:
• setmode() : to set the mode in which Raspberry Pi is to be used GPIO.BCM or • GPIO.BOARD
• setup(): sets the corresponding pin as GPIO.OUT or GPIO.IN mode • output(): sets the mode in which output is to be received HIGH or LOW. •

sleep() : for delay in seconds
• cleanup() : to clean all the used ports

Procedure

1. Interface the LED to a bread board and connect the anode to raspberry pi GPIO pin 2 and cathode to GND.
2. Connect raspberry pi to GPU monitor

Block Diagram:



Outcome:

The outcome demonstrated successful control and interfacing of LEDs, amd sensors using Raspberry Pi GPIO pins, showcasing the versatility of GPIO for both input and output operations in various electronic applications

Sample use case:

1. Design and implement an efficient, secure, and safe operation of automated doors in a warehouse environment.
2. Extend a basic motion detection system with an IR sensor and Raspberry Pi to send notifications or alerts when motion is detected thro email or whatsapp.

Exercise No. 4 Interfacing Digital Camera to Raspberry

Objective:

To interface digital camera in Raspberry Pi

Components Required:

• Digital Camera, PIR Sensor, Wires and Raspberry Pi board

Theory or Concept:

When the circuit is powered DC, the PIR sensor becomes active and scans for motion. When motion is detected, the signal is passed to the camera to capture the images and save it to the local directory. The camera remains switched on. The clicked snapshots can be viewed in the corresponding directory

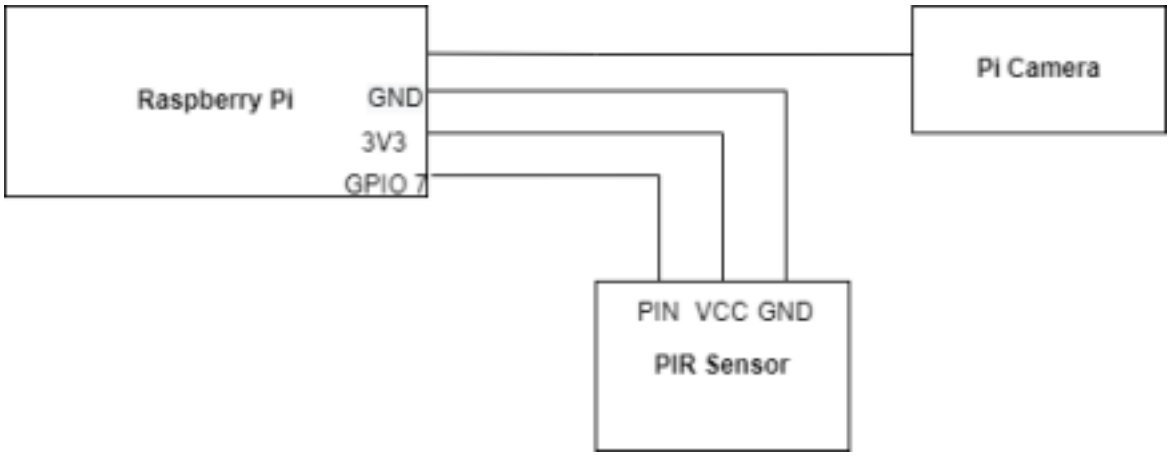
Basic Functions:

- capture() : captures images from camera
- start_preview(): starts live display of the cameras input
- stop_preview(): stops the display

Procedure:

- Interface PIR Sensor to Raspberry Pi board and connect its VCC to 3V3 and output to GPIO pin 7.Connect ground to GND pin.
- Interface the camera in camera I/O interface present in Raspberry Pi board.
- Write the corresponding code for interfacing

Block Diagram:



Outcome:

Successfully interfaced a digital camera with a Raspberry Pi, enabling image capture and processing, demonstrating the Pi's capability for photography and computer vision applications.

Sample use case:

Design and implement a cost-effective home security solution that can be tailored to meet specific needs.

Exercise No. 5 To connect Arduino and Raspberry Pi as Master-Slave I2C

Objective:

To interface Arduino based and Raspberry Pi board in master-slave approach and demonstrate I2C

Components Required:

- Raspberry Pi board, Arduino UNO, LED, Bread Board and Wires

Theory or Concept:

When the circuit is complete the user runs the pi program and passes a binary value while is passed to Arduino from pi. Arduino acts as slave and based on this input LED state turns into higher low The LED is toggled based on passed parameter in the console

Basic Functions:

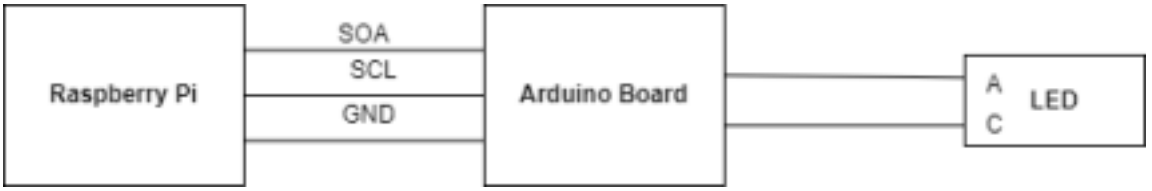
- wire.begin() : Initialize wire library and join I2C
- wire.OnReceive(): registers a function to called on to receiving data
- wire.read(): reads a byte that was transmitted from master

- wire.available(): returns number of bytes available for retrieval
- bus.write_byte(): sends a byte, value, data to I2C device

Procedure

- Deploy the LED code into Arduino board by interfacing the board with LED and writing corresponding code to toggle it. Upload the code to arduino.
- Since now the arduino behaves as slave node and pi as master. Connect SDA and SCL [ins of arduino with the Pi for chasing I2C communication.

Block Diagram:



Output:

Established I2C communication between Arduino (Slave) and Raspberry Pi (Master), enabling bi-directional data transfer and demonstrated the integration of sensor data or control signals between the two platforms for IoT or embedded system applications.

Sample use case:

Design and implement an environmental monitoring system that collects temperature, humidity, and air quality data using sensors connected to an Arduino, with the Raspberry Pi acting as the master device to collect, process, and display the data.

Exercise No. 6 Deploying local server using Raspberry pi

Objective:

To deploy a local server using Raspberry Pi, enabling hosting of web applications or services accessible within a local network.

Components Required:

- RaspberryPi

Theory or Concept:

Raspberry Pi, with its capabilities as a low-cost, energy-efficient computer, can be used as a local server. By installing server software like Apache (for web hosting), MQTT broker (for IoT communication), or Samba (for file sharing), the Raspberry Pi can serve various purposes within a local network. The server can host web pages, process HTTP requests, manage MQTT messages, or share files among connected devices.

Procedure:

- Setting Up Raspberry Pi

• Installing Server Software - Web Server (Apache) - Install Apache web server • Verify Apache is running by accessing Raspberry Pi's IP address in a web browser. • Access the deployed services (e.g., web pages, MQTT topics, shared files) from devices within the local network.
• Configure security settings (firewall, user access) for each service as needed.

Outcome:

Successfully deployed a local server using Raspberry Pi capable of hosting web applications, within a local network.

Exercise No. 7 WiFi Based communication between ESP32 and Raspberry Pi using HTTP

Objective:

To enable WiFi based communication between ESP32 and Raspberry

Pi. **Components Required:**

• Raspberry Pi board
• ESP32 board

Theory or Concept:

HTTP-based communication between an ESP32 and Raspberry Pi over WiFi involves the ESP32 acting as a client sending HTTP requests to a Raspberry Pi server. The ESP32 uses its WiFi module to establish a connection to the Raspberry Pi's WiFi network. HTTP requests are sent by the ESP32 to retrieve or send data from/to the Raspberry Pi, facilitating data exchange for IoT applications such as sensor data transmission or remote control.

Basic Functions:

- http.begin() - starts http
- app.route - mapping the URLs to a specific function that will handle the logic for that URL

Procedure:

• Install FLASK 'pip install flask' in the raspberry pi connected system. • Connect the ESP32 to the WiFi name and password through the code. • Find the IP address of the Raspberry Pi through command prompt using "ifconfig" command. In ESP32 code, change WiFi details, IP address and upload it.
• Name the RaspberryPi Code as app.py and execute the code.

Block Diagram:

Output:

Implemented WiFi-based communication between an ESP32 (client) and a Raspberry Pi (server) using HTTP, enabling data exchange and control signals, demonstrating effective

integration for IoT applications.

Sample use case:

Design and implement a smart home monitoring system where the ESP32 collects environmental data (e.g., temperature, humidity) and sends it to a Raspberry Pi server over WiFi using HTTP. The Raspberry Pi processes and displays the data on a web dashboard.

Exercise No. 8 MQTT Publish and Subscribe using I0T Development Board

Objective

Develop a real-time IoT application using the MQTT protocol where an ESP32 acts as both a publisher and a subscriber.

Components Required:

- ESP32 Board

Theory or Concept:

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed for constrained devices and low-bandwidth, high-latency, or unreliable networks. It operates on a publish/subscribe model:

- **Publisher**: Sends messages to a topic.
- **Broker**: Manages topics and delivers messages to subscribers.
- **Subscriber**: Receives messages from topics of interest.

Procedure

- The **ESP32** will read the potentiometer value and publish it to a specific MQTT topic. - The ESP32 will also subscribe to another MQTT topic to receive messages that will control an LED.

Block Diagram:

Output:

Successfully implemented MQTT publish and subscribe functionality on an IoT development board, demonstrating bi-directional communication for real-time data exchange and control in IoT applications.

Sample use case:

Design and implement a smart agriculture system where an ESP32 collect soil moisture data

and publishes it to an MQTT broker running on a ESP32 Pi. Another ESP32 subscribes to this data and controls a water pump based on the soil moisture level.

Exercise No. 9 MQTT Publish and Subscribe using IOT Development Board (Web server)

Objective:

To exchange the message using MQTT and visualization using Mosquitto broker.

Components Required:

- ESP32 Board
- Raspberry Pi

Theory or Concept:

MQTT-Message Telemetry Transport is message oriented and typical architecture is based on the client/server model. Sensor is a client and it connects to the broker over TCP. Message is published to a topic. Mosquitto is an open source MQTT message broker service. It uses MQTT protocol for devices to communicate by sending and receiving messages. Mosquitto is a small and lightweight implementation of MQTT v3.1/3.1.1. Mosquitto runs on small computing models like Raspberry Pi.

Block Diagram:

Output:

Successfully implemented MQTT Mosquito publish and subscribe functionality on an IoT development board, demonstrating bi-directional communication for real-time data exchange and control in IoT applications.

Sample use case:

Design and implement a smart agriculture system where an ESP32 collect soil moisture data and publishes it to an MQTT broker running on a Raspberry Pi. Another ESP32 subscribes to this data and controls a water pump based on the soil moisture level.

Exercise No. 10 Import Sensor data from ESP32 to Thing Speak cloud

Objective:

To implement a system where an ESP32 collects sensor data (e.g., temperature and humidity) and sends it to the Thing Speak cloud platform for real-time monitoring and data analysis.

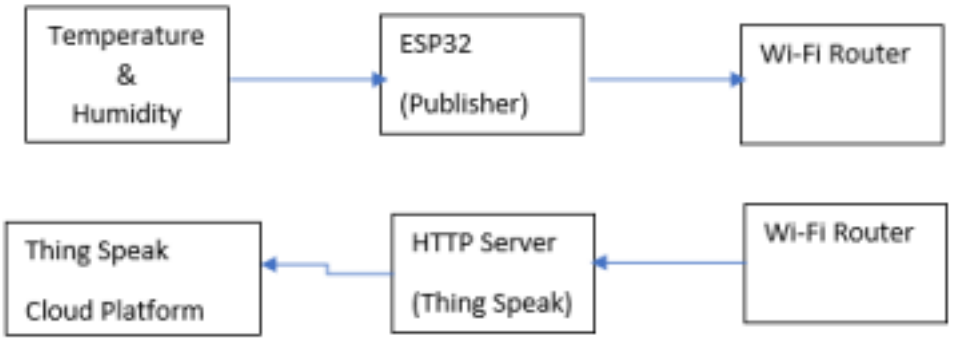
Components Required:

- ESP32:
- Temperature and Humidity Sensor
- WiFi Network:
- Thing Speak Account

Theory or Concept:

The Internet of Things (IoT) ecosystem involves collecting data from various sensors, processing this data, and sending it to cloud platforms for storage, analysis, and visualization. Thing Speak is a popular cloud-based platform designed specifically for IoT applications, providing tools for data collection, storage, analysis, and visualization.

Block Diagram:



Output:
Successfully imported sensor data from an ESP32 to ThingSpeak cloud, enabling real-time visualization and analysis of data such as temperature, humidity, or any other sensor readings, facilitating remote monitoring and data-driven decision-making.

Sample use case:

Design and implement a remote health monitoring system using an ESP32 microcontroller to monitor vital signs (such as heart rate and body temperature) of patients. Data will be sent securely to the Thing Speak cloud platform for real-time monitoring by healthcare providers and patients.

Exercise No. 11 Task Context Maintenance and FREERTOS

Objective:

To implement the task scheduling using Freertos.

Components Required:

- ESP32 Board
- LED

Theory or Concept:

The theory of concept of Task Context Maintenance revolves around the principles and practices involved in effectively managing and sustaining the context or environment within which tasks are executed. This concept is crucial in ensuring that tasks progress smoothly, remain aligned with objectives, and achieve successful outcomes.

Basic Functions:

- TaskHandle_t - Type by which tasks are referenced. For instance, a call

to xTaskCreate returns an TaskHandle_t variable that can then be used as a

parameter to vTaskDelete to delete the task.

● xTaskCreate(taskOne,"TaskOne",10000,NULL,1,&xHandle1) :

a. TaskCreate : creates a genericTask

b. 10000 : Stack size in words

c. NULL : Parameter passed as input of the task

d. &myTask : Priority of task and task handle

● vTaskDelete(xHandle) : The task is removed from all ready, blocked,

suspended and event lists

● vTaskSuspend(xHandle) : Suspends a task and the suspended task will

never get any microcontroller processing time regardless of its priority.

● vTaskResume(xHandle) : Resumes a suspended task.

Output:

The task scheduling using Freertos is implemented and the output is tested and verified.

Sample use case:

Illustrate how task context maintenance principles are applied to ensure efficient software development project management. By implementing these strategies, Company X can navigate complexities, mitigate risks, and deliver a high-quality web application to enhance customer satisfaction and operational efficiency.

Exercise No. 12 Mutual Exclusion using Semaphores and FREERTOS

Objective:

To perform mutual exclusion in freertos using semaphores.

Components Required:

- ESP32 Board
- LED

Theory or Concept:

The two functions taskled and taskblink must be executed mutually exclusive to each other. The critical section can be accessed only by one task at a time and the other task has to wait until the current task completes execution.

Basic Functions:

● xSemaphoreCreateBinary() : Creates a binary semaphore, and returns a handle

by which the semaphore can be referenced. The semaphore is created in the

'empty' state, meaning the semaphore must first be given using the function. ●

xSemaphoreTake(SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait) :

a. Parameters:

i. xSemaphore : A handle to the semaphore being taken - obtained when the semaphore was created.

ii. xTicksToWait : The time in ticks to wait for the semaphore to become available. The macro portTICK_PERIOD_MS can be used to convert this to a real time. A block time of zero can be used to poll the semaphore.If INCLUDE_vTaskSuspend is set to '1' then specifying the block time as portMAX_DELAY will cause the task to block indefinitely (without a timeout).

b. Return : pdTRUE if the semaphore was obtained. pdFALSE if xTicksToWait expired without the semaphore becoming available.

Output:

Mutual exclusion of two tasks using semaphores and FREERTOS is implemented. The output is tested and verified.

Sample use case:

Illustrate how mutual exclusion using semaphores in FreeRTOS, Company Y can effectively manage shared resources in their embedded system application, ensuring reliable and deterministic behavior across tasks running on the ESP32 microcontroller.