DESIGN AND IMPLEMENTATION OF A REAL-TIME OCCUPANCY DETECTION AND INTERACTIVE STAFF AVAILABILITY DISPLAY SYSTEM FOR SMART OFFICES

BY

GIDEON OSADEBAMWEN OSEMWENGIE

PG/ENG0303161

DEPARTMENT OF COMPUTER ENGINEERING

FACULTY OF ENGINEERING

UNIVERSITY OF BENIN

JANUARY, 2026

DESIGN AND IMPLEMENTATION OF A REAL-TIME OCCUPANCY DETECTION AND INTERACTIVE STAFF AVAILABILITY DISPLAY SYSTEM FOR SMART OFFICES

BY

GIDEON OSADEBAMWEN OSEMWENGIE

PG/ENG0303161

PROJECT SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING, UNIVERSITY OF BENIN IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF MASTER IN ENGINEERING (MENG) DEGREE IN COMPUTER ENGINEERING

JANUARY, 2026

DECLARATION

I hereby declare that this project write-up titled: Design and Implementation of a Real-Time Occupancy Detection and Interactive Staff Availability Display System for Smart Offices, is a collection of my original work and has not been presented for any other qualification anywhere. Information from other sources has been duly acknowledged.

…….. …… …….

Gideon Osadebamwen Osemwengie Date
PG/ENG0303161
Computer Engineering, Faculty of Engineering,
University of Benin, Benin City, Nigeria

CERTIFICATION

This is to certify that this research was carried out by Gideon Osadebamwen Osemwengie, with Matriculation number PG/ENG0303161 of the Department of Computer Engineering, University of Benin, Benin City in partial fulfillment of the requirements for the award of the Master (M.Eng) degree in Computer Engineering.

…….. ……

Engr. Dr. O.I. Omoifo Date

Project Supervisor

……. ……..

Engr. Dr. Isi Arthur Edeoghon Date

Head of Department

DEDICATION

This Master thesis is dedicated to Almighty God, who has been my guide all these years.

ACKNOWLEDGEMENT

First of all, I give honor, praise, and adoration to the almighty God for providing me the wherewithal to turn my dream into reality.

My gratitude goes to my project supervisor, Dr. O.I. Omoifo, my Head of Department, Dr. Isi Arthur Edeoghon, and to all teaching and non-teaching staff of the Department of Computer Engineering.

Finally, my thanks also go to my beautiful wife and all members of my family for their support.

Above all, I give God the glory for his enduring mercies and wisdom.

ABSTRACT

In a workplace environment, such as an academic department, knowing the availability of an office occupant remains a persistent challenge for staff and students. Traditional approaches, such as the use of indoor/outdoor tags, are outdated.

This research focuses on the design and Implementation of a Real-Time Occupancy Detection and Interactive Staff Availability Display System for Smart Offices. The system uses a Passive Infrared (PIR) motion sensor to detect when the office occupant is seated. The system provides five distinct status messages that can be automatically broadcast using push buttons on the input unit of the device, and the status communicated are: "In a Meeting - Please Wait" - "Available But Busy", "Available - Knock First", "In Class - Back Soon", "Unavailable" (Auto-triggered by inactivity or off-hours.

Testing was done in different stages of the design process. After construction, the system was tested, and it worked satisfactorily.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND TO THE STUDY

Recently, several ICT-based systems and applications have been designed to assist people in resolving problems that arise in daily life (Mingyu et al., 2016). ICT advancements have made it simple for companies to keep an eye on their workers' whereabouts in real time. The term "real-time occupancy detection" describes the ongoing surveillance and monitoring of how many individuals are in a certain place at any given time. According to Kyle and Spies (2015), the staff availability display system uses hardware and software to display employees' real-time availability in their designated office area.

Staff monitoring and availability determination are not novel concepts. Using RFID (Radio-frequency identification) tags, several systems have been developed to track employees as they move from one location to another. The three main parts of a conventional location monitoring system are RFID, software, and location tags, sometimes known as badges. Staff members wear the tag all day long, and wireless data transfer occurs. In addition to RFID, the global positioning system (GPS) and sensor networks have made real-time monitoring possible (Kyle and Spies, 2015).

The Office Occupancy Status Display System is a smart, automated status display that makes displaying notification messages simple and professional. It is a wireless messaging system that handles actual communication problems at work. It creates a polished, automated status message display system that boosts productivity and minimizes disruptions by fusing embedded technologies, wireless communication, sensor technology, and user interface design.

The aim is to automate messages to convey important information to students and visitors alike in an office environment. Imagine one has an important meeting coming up. Instead of putting a

paper sign on the door (which one might probably forget to remove), the operator simply presses a button on a small control panel. Instantly, a bright LED sign outside the door starts scrolling "IN A MEETING - PLEASE WAIT" and so on, depending on the message button one has chosen.

The benefits of occupancy detection and staff monitoring include the ability of employees to update their availability within the assigned office space at any time. It will also save students time by not waiting endlessly for a lecturer or staff member in the department without knowing when he/she will be available.

The developed system is aimed at notifying students, visitors, and other staff about the availability of the office occupant within their allocated office space. The developed system will provide real-time information regarding staff member availability within their allocated office space. This developed system has a display board that will be placed at the staff office door to show the availability of the staff members whom they intend to see.

1.2 PROBLEM STATEMENT

It is often very difficult for other staff members and students to know the availability of staff within their assigned office. Oftentimes, students are made to spend hours waiting endlessly for a particular lecturer in the department without knowing when he or she will be available. Monitoring the availability of a lecturer in the office for the students to meet them is an aspect that must be taken care of (Hemanth *et al.,* 2023). Therefore, there is a need for a suitable solution to overcome this problem, which is faced by students daily.

1.3 AIM AND OBJECTIVES

This study aims to design and implement an occupancy monitoring and interactive staff availability notification system for smart offices.

The specific objectives are:

i. To specify the requirements for an automated occupancy detection system, identify the most effective sensor types for accurate office occupancy tracking.

ii. To design a hardware architecture centered around the ESP32 microcontroller that integrates occupancy sensors with a physical status indicator

iii. To implement a management dashboard that allows staff to manually override their status (e.g., "In a Meeting" or "Be Right Back") and enables students to view availability via a detached display device.

1.4 SCOPE OF STUDY

The scope of this work is limited to designing and implementing a real-time occupancy detection and interactive staff availability display system. The real-time occupancy detection is limited only to when someone sits in an office chair and does not factor in the movement of staff within or outside their offices. The availability status is limited to display on an LED strip for visitors to see.

1.5 RELEVANCE OF STUDY

This study is perfect for office use, managers, or anyone who wants to automatically communicate their availability status to visitors without manual intervention. The system "knows" when someone is at their desk and makes it easy to quickly set an appropriate status message. The system is essentially a smart "Do Not Disturb" sign that activates automatically when you sit down and lets visitors know your availability to receive or not to receive visitors at a glance.

CHAPTER TWO

LITERATURE REVIEW

2.1 THEORETICAL REVIEW OF CONCEPTS

2.1.1 ESP32 Microcontroller

The ESP32 is a single 2.4 GHz Wi-Fi and Bluetooth combination chip that uses low-power 40 nm technology from TSMC. In a wide range of applications and power settings, it demonstrates resilience, versatility, and dependability while achieving the optimum power and RF performance. According to Pratama and Agus (2022), the ESP32 series of chips consists of the ESP32-D0WDQ6, ESP32-D0WD, ESP32-D2WD, and ESP32S0WD.

The ESP32 is a versatile and widely used microcontroller and Wi-Fi/Bluetooth system-on-chip (SoC) produced by Espressif Systems. ESP-Wroom-32 contains a low-power Tensilica Xtensa® Dual-Core 32-bit LX6 microprocessor at 240 MHz (Soy, 2021). It also has 448 KB of ROM for booting and core functions. 520 KB of on-chip SRAM for data and instructions.

Other features include 4MB of Flash Memory, 16 KB SRAM in RTC Wi-Fi 802.11b/g/n Bluetooth v4.2 BR/EDR and Bluetooth LE specifications. The ESP32 chip has about 500 KB of RAM and 4 Mb of FLASH built in. This is essential for supporting the IP and TLS stack as well as all other internet communication-related issues that use excessive amounts of resources.

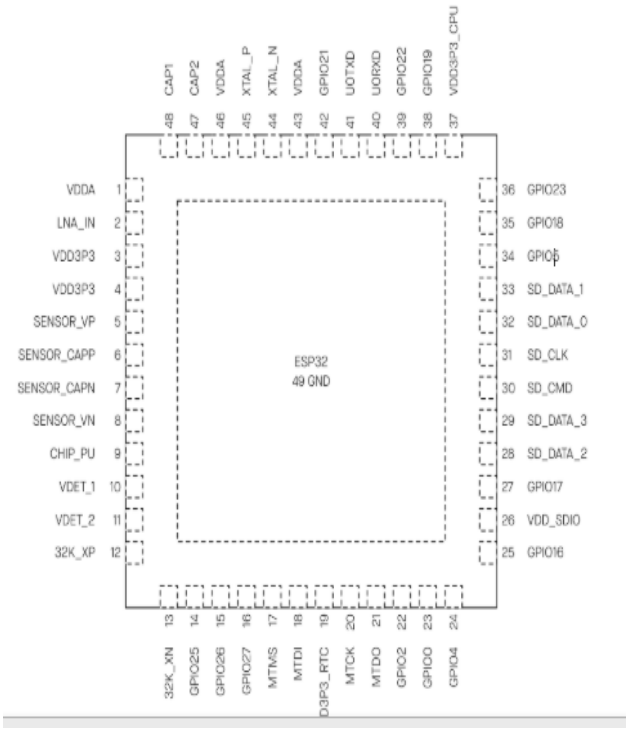

Figure 2.1: .ESP32 Pin Layout

Figure 2.1 shows the pin layout of the ESP32 microcontroller. Standard interfaces like UART and SPI are used by the processor to communicate with the peripherals. The ESP32 is intended for Internet-of-things (IoT) applications, wearable technology, and mobile devices. It has all of the cutting-edge features of low-power devices, such as dynamic power scaling, numerous power modes, and fine-grained clock gating. In addition to adding sophisticated calibration circuitry that enables the solution to eliminate external circuit flaws or adapt to changes in external conditions, ESP32 leverages CMOS for a single-chip fully integrated radio and baseband (Pratama and Agus, 2022).

2.1.2 Millimeter Wave Sensor

A millimeter wave (mmWave) sensor is a sophisticated detection tool that operates in the 30-300 GHz frequency range. It uses high-frequency electromagnetic waves to precisely detect motion, objects, and human presence. It produces and receives radio waves to examine minute variations

in reflected signals, in contrast to conventional infrared or camera-based systems. This allows for non-contact detection even through non-metallic surfaces like walls or textiles. This technology performs exceptionally well in situations that call for precision and discretion; it can discriminate between background noise and human activity, operates dependably in low light or darkness, and protects privacy by not collecting visual data. Millimeter wave sensors are reinventing automated sensing across industries while balancing performance, affordability, and ethical issues. Examples include providing occupancy analytics in retail environments and optimizing energy use in smart buildings.

2.1.3 PIR motion sensor

A PIR sensor is an electronic sensor that is used in motion detectors, such as protection systems and automatically activated lighting devices that measure infrared light-emitting devices within its field of vision Yogesh et al. (2018). Anybody who has a temperature higher than zero emits radiation, which is heat energy. Instead of measuring or detecting heat, PIR sensors identify infrared radiation that is reflected or emitted from the target. The temperature in the sensor's range of view rises to the intruder's body temperature from the surrounding air if it detects an animal, bug, or human.

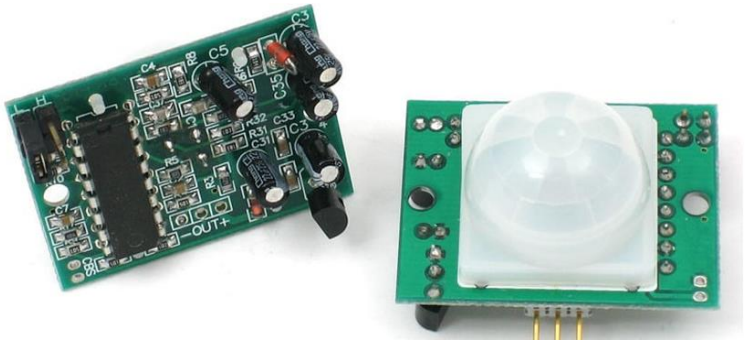

Figure 2.2 PIR motion sensor

Passive infrared, or PIR, detectors are most frequently found in intruder alarm systems. A remote thermal sensor has a higher probability than a PIR. These infrared sensors/detectors are the covert internal motion sensor security lights that turn on driveways when someone gets close. According to Yogesh et al. (2018), infrared light cannot be detected by our eyes.

2.1.4 RTC module

A crystal unit used as a clock source, an oscillator circuit, and a date/time counter circuit make up a basic RTC. A little electric charge is created when a voltage is applied to the crystal, a process known as the "piezoelectric effect." A constant frequency signal is produced by amplifying this electric charge and feeding it back into the crystal. Date-time data is produced by the date-time counter circuit using the oscillation signal of the crystal unit as the clock source. Quartz crystals produce extremely accurate time data because they have the most steady oscillation frequencies of any piezoelectric material.

2.2 RELATED WORKS

Kyle and Spies (2015) created and put into use an automated system for tracking lecturer availability that integrated hardware (such as Arduino Ethernet boards and magnetic switches) and software (such as Apache, PHP, MySQL, JavaScript, HTML, and C#). The client-server architecture is used by the system. There is a database on the server. In response to a data request, the web server retrieves all of the information from the pertinent database that contains the staff availability status. The client receives this data when it has been compiled into an appropriate HTML and JavaScript display format. An employee has the power to modify their availability status.

Mingyu et al. (2016) employed a variety of sensors attached to the chair together with an embedded Internet of Things device to ascertain the posture of the user. Using Bluetooth 4.0 connection, the

embedded IoT device transmits the measured sensor data to a smartphone or a server (via an Internet-connected smartphone or a gateway). The posture data transmitted by the Smart IoT Chair is received, processed, and displayed in real-time by the server or smartphone. In order to process data received from the Smart IoT Chairs, a wireless gateway is utilized to assess an individual's or a group's pattern and transmit the results to the server. To assist users in correcting their uneven posture, the developed Smart IoT Chair uses a smartphone application to record and visualize users' posture.

In another related work, Kommey et al. (2017) created a system that tracks seat occupancy in a hall. Kommey et al. (2017) used a base station, a seat sensor, and a microcontroller with WiFi capability to create the system. When a person is seated, the system uses a seat sensor to detect their presence in the hall; the Wi-Fi-enabled microcontroller receives the data. After that, these data are sent via LAN to the base station.

Mengjie et al. (2017) developed a smart chair system that can keep an eye on how each user sits. An artificial neural network classifier was used to automatically recognize the eight standardized sitting postures of human participants based on pressure patterns that were collected and sent to the computer. According to the findings, the system can accurately identify eight different sitting positions of people.

Aswin et al., (2017) designed a wireless office automation and security system that uses an Android smartphone and Raspberry Pi to allow officials to engage directly with visitors while keeping track of their information on the smartphone. In the visitor's area is a Raspberry Pi with an LCD, a push button, and a camera. The official also has an Android smartphone. The visitor's image is taken and sent to an Android phone using Bluetooth each time he touches the button.

Through the Android application, the official replies by displaying a pre-configured message on the LCD, such as "busy" or "in meeting."

Shaikh et al. (2017) developed an intelligent office space monitoring and control system using IoT. A Raspberry Pi and an Android smartphone running an Android app are used to monitor and manage the electrical appliances via voice commands.

A low-cost visitor control system for smart offices was created by Rao et al. (2018) using an Android smartphone, a nodeMCU-based wireless transmitter, and a nodeMCU-based wireless display. The office worker or consultant has an Android smartphone, and the nodeMCU-based transmitter and display are stored in the visitor's area. The visitor writes down the acknowledgement token number that the consultant's Android smartphone sent and types a message using a keyboard attached to a wireless transmitter. After the consultant has reviewed this message, the Wireless Display Unit receives the same token number and notifies the visitor to consult the consultant.

Mannino et al. (2019) implemented Image Recognition (Imr), a form of Artificial Intelligence (AI), on a Raspberry Pi board connected to an image sensor in order to identify human movement within the office building. When a person or multiple persons enter the camera's field of view, the system instantly recognizes their movements. The Raspberry Pi's artificial intelligence analyzes the frames to identify and follow the direction and movements of the people in the photos. Consequently, real-time building occupancy data is provided. The article provides an example of an Internet of Things system for managing visitors in a smart office.

The automation system is made up of a Raspberry Pi with a camera and an Android smartphone. A Raspberry Pi with a camera is kept in the guest waiting room, and the consultant's Android

smartphone is equipped with the visitor management software. The guest selects a time slot and receives a QR code to make an appointment using the client app on his Android phone.

Santhosha and Casbona (2019) developed an Internet of Things system for managing visitors in a smart office environment. The automation system is made up of a Raspberry Pi with a camera and an Android smartphone. A Raspberry Pi with a camera is kept in the guest waiting room, and the consultant's Android smartphone is equipped with the visitor management software. The guest selects a time slot and receives a QR code to make an appointment using the client app on his Android phone. The consultant has access to details about each session via a professional application that is loaded on his Android phone. Every time a visitor scans the QR code using the Raspberry Pi camera, the professional application is notified, enabling the consultant to see a list of every visitor in real time. An intelligent token system determines the order of passage when there are several visitors in the office.

Taraneh et al. (2022) created a smart chair sensor system that can detect users' sitting positions and calculate body pressure. The backrest and seating cushion of the chair were equipped with eight pressure sensors. The data produced by the sensor pressure was collected using a signal acquisition board and sent to a laptop via a Wi-Fi network for storage, preprocessing, and real-time user monitoring. Eight sitting postures were assessed using seven deep learning models: Multilayer Perceptron (MLP), CNN, Long Short-Term Memory (LSTM), bidirectional LSTM (BDLSTM), CNN-LSTM (CNLSTM), convolutional LSTM (CVLSTM), and an Echo Memory Network (EMN). The outcome demonstrates that an echo memory network model has the highest accuracy of 91.68%.

Venkateswar et al. created a smart office chair in 2022. If someone is seated for an extended period of time, the chair alerts them. This chair also serves as a reminder to drink water or take medication.

Using an IR sensor and an Arduino, a prototype module was created and put into use. Additionally, an Arduino controller was interfaced with a speech chip that provides spoken announcements to drink water, take medication, etc.

A smart office chair with movable textile sensors that can track users' sitting positions throughout the workday was created by Martínez et al. in 2023. Ten textile capacitive sensors with varying degrees of activation are used in the system along with a signal conditioning device. The device was included in an office chair to identify postures that can cause discomfort or musculoskeletal problems. A microcontroller used a cycle count approach to measure the capacitance and gave real-time position data. By analyzing the data, warnings might be set up to prevent bad postures or the need to move, hence reducing occupational dangers.

A smart chair for tracking sitting postures and detecting seat occupancy was created by Mohammed and Shaharil in 2024. Four ultrasonics were incorporated into the smart chair to actively track the distance between the user's back and the seat cushion. A NodeMCU-32S Base module serves as the primary controlling device for the ESP32 Wi-Fi module that powers the Smart Chair. The sensors allow the Smart Chair to send signals to the microcontroller for processing and detect physical pressure for seat occupancy. Eight positions were examined by the system: sitting straight, slouching, leaning forward, leaning backward, leaning left, leaning right, crossing one leg, and crossing the other leg. Every time bad posture is identified, a program called Blynk notifies the user on their smartphone.

Brijesh et al. (2024) designed a cloud-based IoT-enabled smart chair that concurrently stores data on the cloud and continuously monitors a person's sitting posture, alerting them when they are in an incorrect position. The cloud-based database aids physicians in determining the underlying cause of a spinal or joint issue. An ESP 32 microcontroller, force and flex sensors to identify an

individual's incorrect posture, and cloud-enabled technology to provide smooth connectivity and notification were all used in the system's design, making it appropriate for any type of setting.

The review of relevant works covered in the previous part is summarized in the Meta Analysis Table shown in Table 2.1.

Table 2.1: Meta-Analysis Table

S/N	Author name and year	Title of Work	Method used	Results obtained	Limitations
1	Kyle and Spies (2015)	Design and Implementation of an Automatic Staff Availability Tracking System	Integrated several software such as Apache, PHP, MySQL, JavaScript, HTML and C#: and hardware technologies (i.e. Magnetic Switches and Arduino Ethernet Board).). The system adopt the client-server architecture.	A staff member can change his/her availability status.	The work focus only on staff tracking
2	Mingyu *et al.,* 2016	Design and Implementation of a Smart Chair system for IoT	It uses sensors for pressure and iBeacon and Bluetooth Communication. Embedded IoT device was used to sends the measured sensor data to a smartphone or a server using Bluetooth 4.0 communication.	The developed Smart IoT Chair records and visualizes user's posture through a smartphone application to help users correct their unbalanced posture.	The research focuses only on posture viisualization

3	Kommey *et al.,* 2017	Design and implementation of seat occupancy Detection System	The system was implemented using WIFI Wi-Fi-enabled microcontroller, a seat sensor, and a base station..	The system detects the presence of a person within the hall when seated.	The work focuses only on seat occupancy.
4	Mannino *et al.,* 2019	Office building occupancy monitoring through image recognition sensors	used Artificial Intelligence (AI) installed on the Raspberry board attached to an image sensor to detect the movement of people within the office building.	The system recognizes movements of people immediately a person, or more than one, enters the field of view of the camera, the Artificial Intelligence on the Raspberry Pi analyses the frames in order to detect and track movement ments and direction of all people in the captured images.	The research focuses only on the detection of people within the building.
	Taraneh *et al.,* 2022	Development of a Smart Chair Sensors System and Classification of Sitting Postures with Deep Learning Algorithms,	Eight pressure sensors were embedded in the chair's sitting cushion and the backrest. A signal acquisition board was used to acquire the data generated from the sensor	The result shows that the best accuracy of 91.68% was achieved by an echo memory network model.	The work focused only on sitting posture.

			pressure. Seven deep learning models were used to evaluate the sitting posture.		
5	Mohamad and Shaharil (2024)	A Smart Chair for Sitting Postures Monitoring and Seat Occupancy Detection	The smart chair was integrated with four ultrasonic to actively monitor the distance between the user's back and the seat cushion.	The system analyzed eight postures which include upright sitting, slouching, leaning forward, leaning backward, leaning left, leaning right, right leg crossed and left leg crossed.	The work focused only on sitting posture monitoring

CHAPTER THREE

METHODOLOGY

3.1 METHODOLOGY

The research objectives were accomplished through the process described in Figure 3.1. The research workflow is broken down into stages as shown in Figure 3.1 in order to achieve the research objectives.

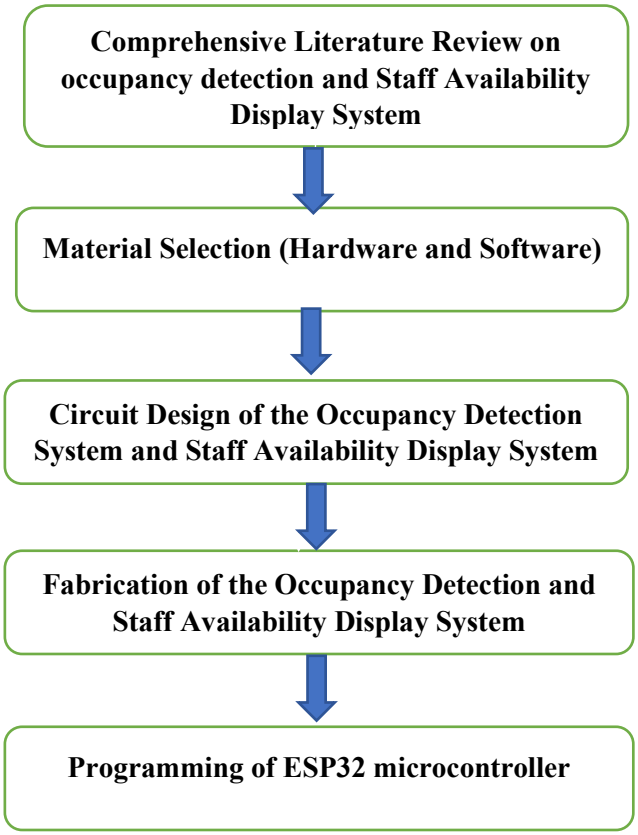

Figure 3.1: Workflow diagram showing the research methodology

This method is achieved through first, a comprehensive review of existing literature, then the hardware design, which involves component selection/system fabrication, and the next stage involves the programming of the ESP 32 microcontroller.

3.2 MATERIALS USED

To achieve the stated objectives of this research, the following tools and materials in Table 3.1 were used.

Table 3.1: Materials used

S/No	Item	Purpose
1	ESP32 microcontroller	Main processor (Main brain)
2	mmWave radar sensor (RD03)	Detects chair occupancy with high precision
3	PIR motion sensor	General motion detection backup
4	physical buttons	For status selection
5	Dual P10 LED matrix panels	Visual status display
6	RTC module	Time tracking
7	Wi-Fi capability	Web configuration interface

3.2.1 HARDWARE COMPONENTS

The hardware components used in the design of the occupancy detection and staff availability display system include an ESP32 microcontroller, which serves as the main processor, Dual P10 LED matrix panels (32x16 pixels each, arranged horizontally), a SPI-based display driver for high-speed LED multiplexing, and the hardware timer interrupt.

3.2.2 SOFTWARE FEATURES

The system was designed with the following software features:

1. **Persistent storage** - Last status saved to EEPROM survives power loss
2. **Auto-restart mechanism** - Device resets after status change to ensure a clean display
3. **Timer-based rendering** - Ensures consistent refresh rates regardless of message processing
4. **Non-blocking message reception** - Handles incoming ESP-NOW packets without display interruption

3.3 CORE FUNCTION (AUTOMATIC PRESENCE DETECTION)

The system uses a Passive Infrared (PIR) motion sensor to detect when the office occupant is seated and automatically switches between "AVAILABLE" and "UNAVAILABLE" states within working hours. The system is configurable for an inactivity timer (30 seconds to 5 minutes) to trigger the UNAVAILABLE status when no motion is detected.

3.3.1 STATUS COMMUNICATION

The system provides four distinct status messages that can be manually broadcast using push buttons on the device, and the status communicated are:

1. "**IN A MEETING - PLEASE WAIT**" - Status 1 (Red button).
2. **"AVAILABLE BUT BUSY"** - Status 2 (Orange button).
3. **"AVAILABLE - KNOCK FIRST"** - Status 3 (Green button).
4. **"IN CLASS - BACK SOON"** - Status 4 (Blue button).
5. **"UNAVAILABLE"** - Status 5 (Auto-triggered by inactivity or off-hours).

3.3.2 WIRELESS DISPLAY INTEGRATION

The office occupancy and staff availability display system utilizes the ESP-NOW protocol for communication with a remote display unit. The ESP-NOW permits communication between two (2) ESP32 microcontrollers wirelessly, and this protocol permits a distance of up to 400 meters for line-of-sight communication and over 220 meters when obstructed. The system sends status updates wirelessly to a hallway or door-mounted indicator. Voice announcements using text-to-speech confirm availability changes.

3.3.3 SMART AUTOMATIC SCHEDULING

The system has a built-in RTC (Real-Time Clock) to enforce working hours of operation from 8:00 am - 4:30 pm daily. It also automatically displays "UNAVAILABLE" when no occupant is

present and outside working hours. The system shows a countdown timer (when motion is detected), indicating the time until the next status change.

3.3.4 USER INTERFACE

The system uses an I2C 16x2 LCD to display current status and time, and an EEPROM for settings storage. Talkie library for audio feedback. The system uses a four-button interface for manual status selection. Also, the hidden menu system is accessed by pressing two buttons simultaneously (UP and DOWN buttons), which allows for the configuration of the system. The inactivity timer duration (by default) was set at 60 seconds.

3.4 SYSTEM INTEGRATION WITH THE INDOOR UNIT

The indoor unit detects activity/inactivity or receives a button press. The system automatically transmits the status code (1-5) wirelessly via ESP-NOW. The outdoor unit receives the message, updates the EEPROM, and restarts. The new status message will then scroll continuously on the LED matrix on the outdoor unit.

3.5 SYSTEM COMMUNICATION AUTHENTICATION

For each pair of devices to communicate, the paired devices must confirm their link using their MAC address. If the MAC addresses are confirmed, the transmitter and receiver units can then transmit and receive the information to be displayed.

3.6 SYSTEM ALGORITHM

Figure 3.2 shows the system algorithm of the indoor and outdoor units of the occupancy detection and staff availability display system.

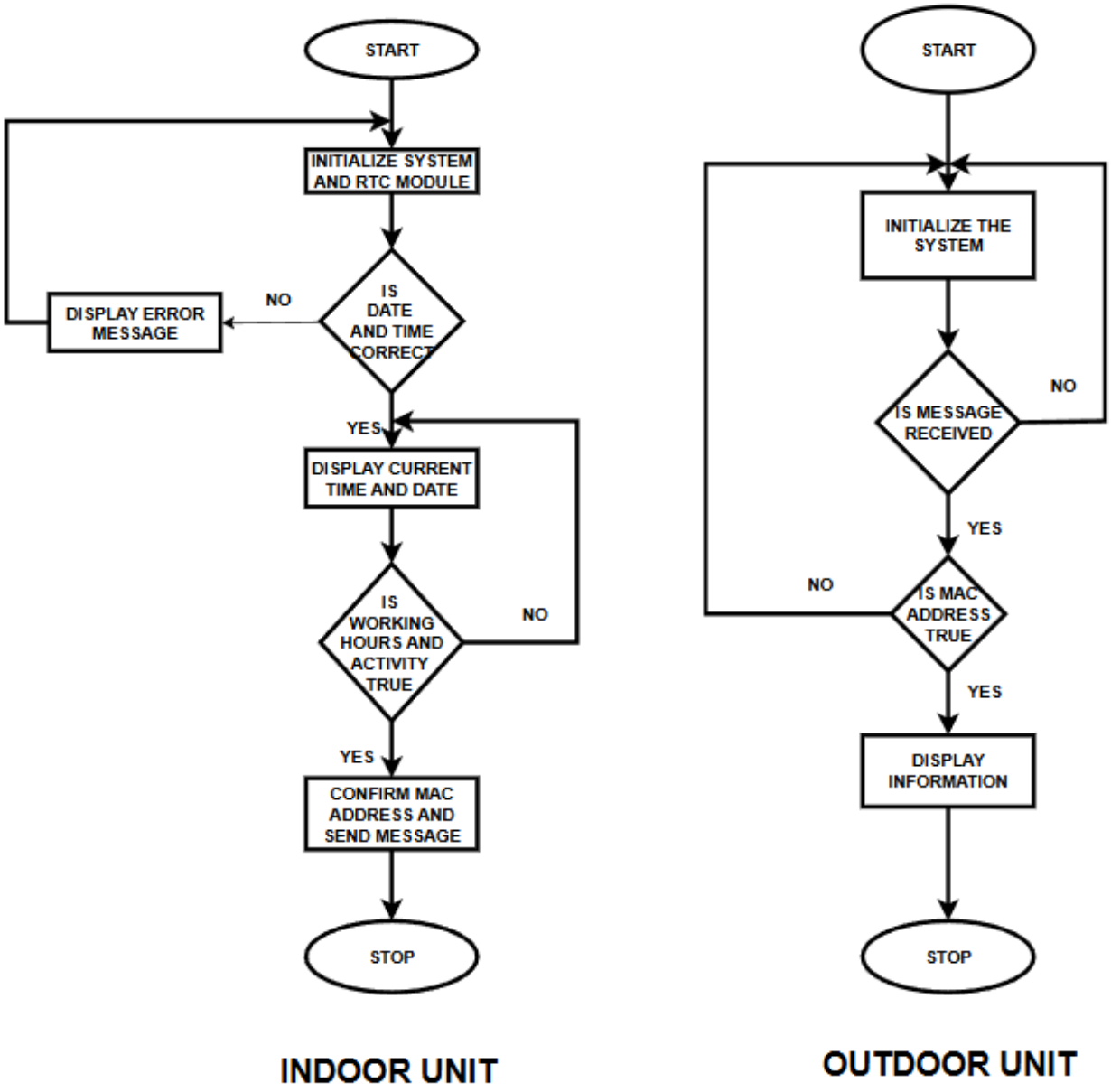

Figure 3.2: SYSTEM FLOW CHART OF THE INDOOR AND OUTDOOR UNIT

Figure 3.3 shows the indoor and outdoor units of the occupancy detection and staff availability system block diagram. Figure 3.4 shows the connection of the various components of the indoor unit of the occupancy detection and staff availability display system.

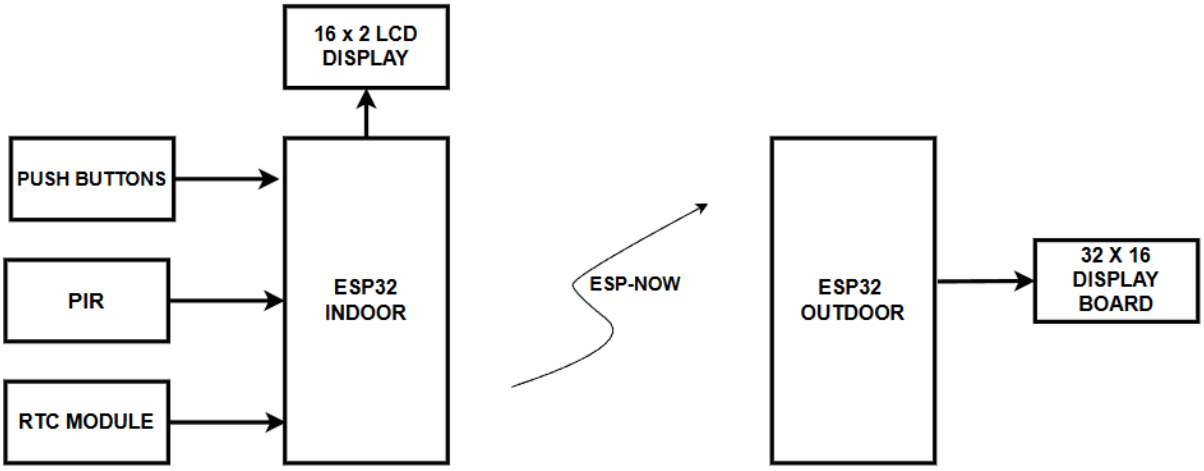

Figure 3.3: System block diagram of the indoor and outdoor unit

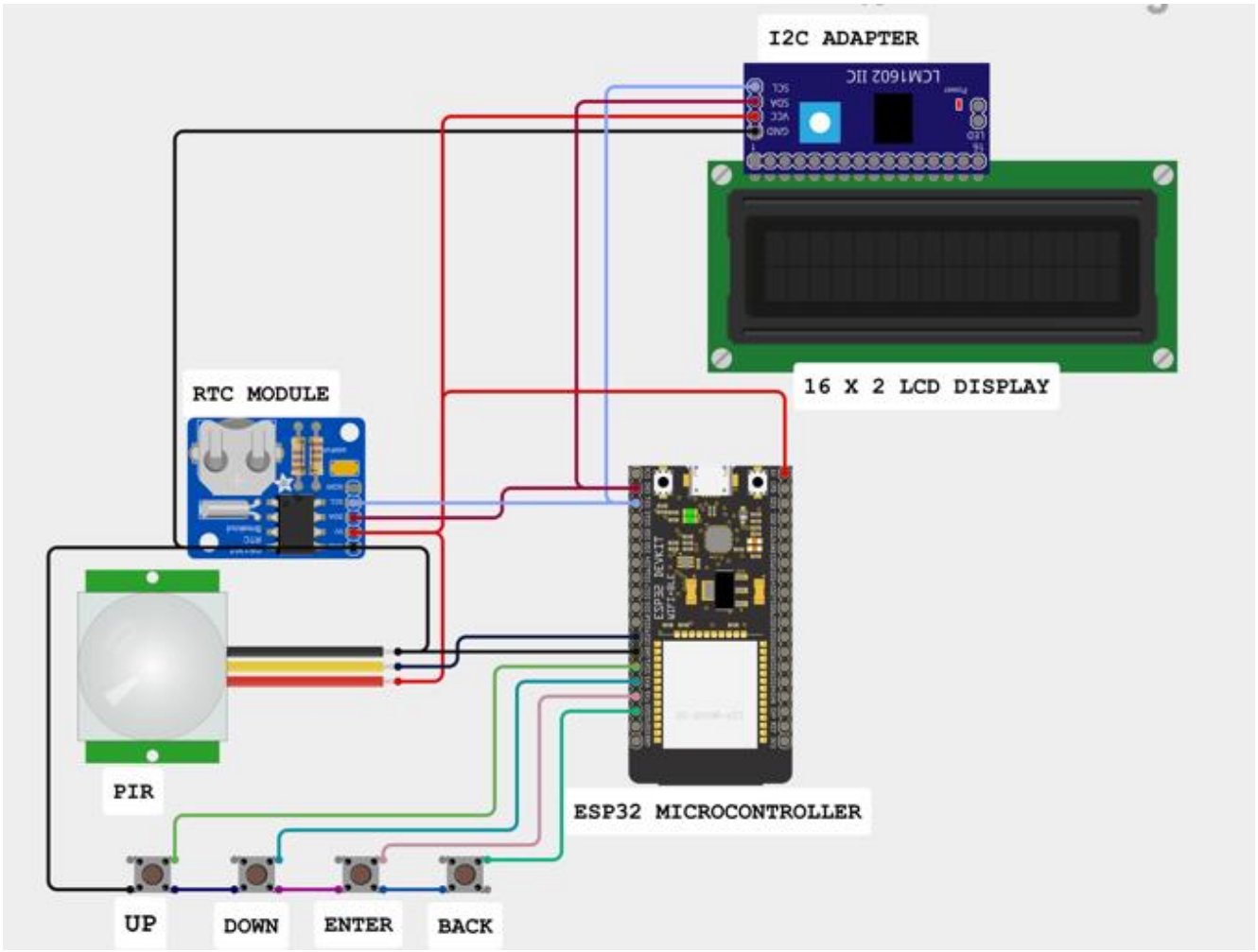

Figure 3.4: Component Connections of the indoor unit

3.7 DISPLAY AND INTERFACE

The system uses a 16x2 LCD Display that is connected via an I2C adapter module. The connection is through four (4) wires (VCC, GND, SDA, SCL). The display indicates the status messages, time, menu options, and countdown timer.

3.8 INPUT DEVICES

The system uses four Push Buttons (Function: Status selection and menu navigation). The buttons and their connection are:

i. UP Button → GPIO 12 (with internal pullup)

ii. DOWN Button → GPIO 14 (with internal pullup)

iii. ENTER Button → GPIO 27 (with internal pullup)

iv. BACK Button → GPIO 33 (with internal pullup)

3.9 PIR MOTION SENSOR

The PIR sensor is connected to GPIO 19. It uses a 3-wire connection: VCC (red), GND (black), and signal (yellow/white). The function of the PIR motion sensor is for automatic presence detection for activity monitoring

3.10 REAL-TIME CLOCK MODULE

The RTC Module (DS3231 or similar) uses an I2C connection. It shares the I2C bus with the LCD. The connection is through VCC, GND, SDA, and SCL. The function of the real-time module is to maintain accurate time/date for working hours enforcement.

3.11 POWER DISTRIBUTION

All the components share common power rails:

i. Red lines - VCC =+5V

ii. Black - GND (Ground)

Table 3.2 Connection Summary

Component	Interface	ESP32 Pins	Purpose
LCD Display	I2C	SDA, SCL	Status display
RTC Module	I2C	SDA, SCL (shared)	Timekeeping
PIR Sensor	Digital Input	GPIO 19	Motion detection
UP Button	Digital Input	GPIO 12	Status/menu control
DOWN Button	Digital Input	GPIO 14	Status/menu control
ENTER Button	Digital Input	GPIO 27	Status/confirmation
BACK Button	Digital Input	GPIO 33	Status/navigation

Some of the key Features of the system include:

i. Shared I2C bus - LCD and RTC use the same communication lines

ii. The system uses Pull-up resistors built into the ESP32 for all buttons

iii. Minimal wiring due to the I2C protocol, which reduces pin count significantly

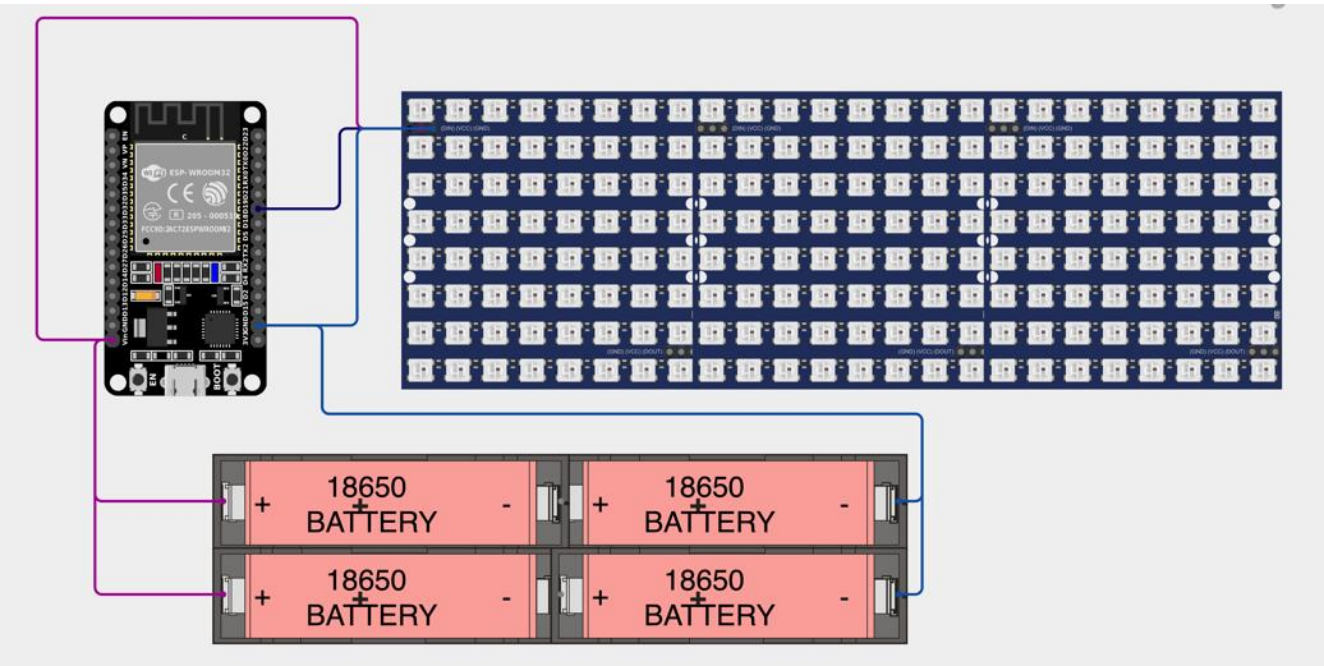

Figure 3.5: Component connection of the outdoor unit

3.12 COMPONENT CONNECTIONS OF THE OUTDOOR UNIT

The outdoor unit uses **a** P10 LED Matrix Panel (32x16 pixels, dual panel configuration). The total display area is 64 pixels wide × 16 pixels tall**.** It also adopts multiple connection pins for control signals and a high-visibility red LED matrix for scrolling text messages.

3.12.1 MICROCONTROLLER

The outdoor unit uses the ESP32 Development Board (left side) as the main processor controlling the display. It also handles ESP-NOW wireless communication and manages SPI data transfer to LED panels. It runs a hardware timer interrupt for display scanning

3.12.2 Connection Details

Table 3.3 shows the ESP32 to LED Panel Pins for the outdoor unit.

Table 3.3: ESP32 to LED Panel Pins

ESP32 GPIO	LED Panel Pin	Function
GPIO 22	nOE	Output Enable (brightness PWM)
GPIO 17	A	Row select bit A
GPIO 21	B	Row select bit B
GPIO 4	CLK	SPI Clock
GPIO 16	SCLK	Shift register latch clock
GPIO 0	R_DATA	Red data (SPI MOSI)

3.12.3 POWER CONNECTIONS

i. Positive (+) from battery pack to ESP32 and LED panel, and Negative (GND) return path

ii. ESP32 may use the onboard voltage regulator for 3.3V logic System Architecture

3.12.4 Display Technology

The display is via the P10 LED Module, which is the industry-standard format. It uses a 32×16 resolution per panel and a 1/4 scan multiplexing (4 rows scanned sequentially). It uses red monochrome LEDs with high brightness for daylight visibility

3.12.5 COMMUNICATION

The communication is through wireless only, as no physical connection to the control unit is required. It receives status updates via the ESP-NOW protocol and operates independently once powered on.

CHAPTER FOUR

CONSTRUCTION, TESTING AND RESULTS

4.1 CONSTRUCTION PROCEDURE

The circuit diagram was obtained after careful design and calculation had been done. The ESP32 board with a basic blink sketch was tested. The LCD with the I2C scanner was verified. The RTC module for time accuracy was checked, and the PIR sensor for motion detection was tested to ensure that it is working properly. Thereafter, the speaker audio output was verified by connecting it to a headphone plug

4.2 BREADBOARD PROTOTYPING

The power rails (5V and GND) were connected and wired to the I2C devices (LCD and RTC) to the shared bus. Thereafter, the PIR sensor was connected to GPIO 26. The push buttons with pull-up configuration were wired. The speaker was later connected to GPIO 25. Each component was tested individually. The complete program was uploaded and tested with the ESP32.

4.3 ENCLOSURE ASSEMBLY

The openings for the LCD were cut, and also the drill holes for buttons. The PIR sensor was mounted with a clear view, and the speaker for optimal audio output was positioned. Ventilation slots for heat dissipation were also created. Finally, a cable gland for power input was added.

4.4 FINAL ASSEMBLY

All soldered connections were verified using a multimeter, and some components were secured with hot glue or standoffs. The ESP32 socket was installed for easy replacement. The buttons were labelled with function names. Power supply was connected, and the circuit was tested. All button functions were tested to ensure functionality. The PIR detection range and sensitivity were

verified. The configuration of time, date, and timer settings was carried out. The system was checked for pairing with the remote display unit.

4.5 TESTING AND CALIBRATION

To ensure that the system works effectively after the assembly. All button functions were tested to ensure their functionality. The PIR detection range and sensitivity were also verified. Adjustment for the inactivity timer to the appropriate duration was made. Voice announcements' clarity was also tested. Finally, the wireless communication with the remote unit was verified.

4.6 RESULTS

4.6.1 OPERATIONAL MODES

Display Layout:

1. Top Line (Row 0): Current time and date (HH:MM DD/MM/YY)
2. Bottom Line (Row 1): Status message and countdown timer

Status Options:

1. **AVAILABLE - Occupant is present and available (default)**
    i. Shows a countdown timer until auto-unavailability
    ii. Voice: "GOOD DAY, WHAT DO I DISPLAY" (on motion detection)
2. **MEETING - Occupant is in a meeting**
    i. Activated by UP button
    ii. Voice: "BUTTON ONE"
3. **BUSY - Occupant is busy, do not disturb**
    i. Activated by DOWN button
    ii. Voice: "BUTTON TWO"
4. **KNOCK FIRST - Please knock before entering**
    i. Activated by the ENTER button
    ii. Voice: "BUTTON THREE"

5. **IN CLASS - Occupant is teaching/in class**
    i. Activated by the BACK button
    ii. Voice: "BUTTON FOUR"
6. **UNAVAILABLE - No occupant detected (automatic)**
    i. Triggered after the inactivity timeout expires
    ii. Waits for motion to return to AVAILABLE

Figure 4.1: Indoor unit

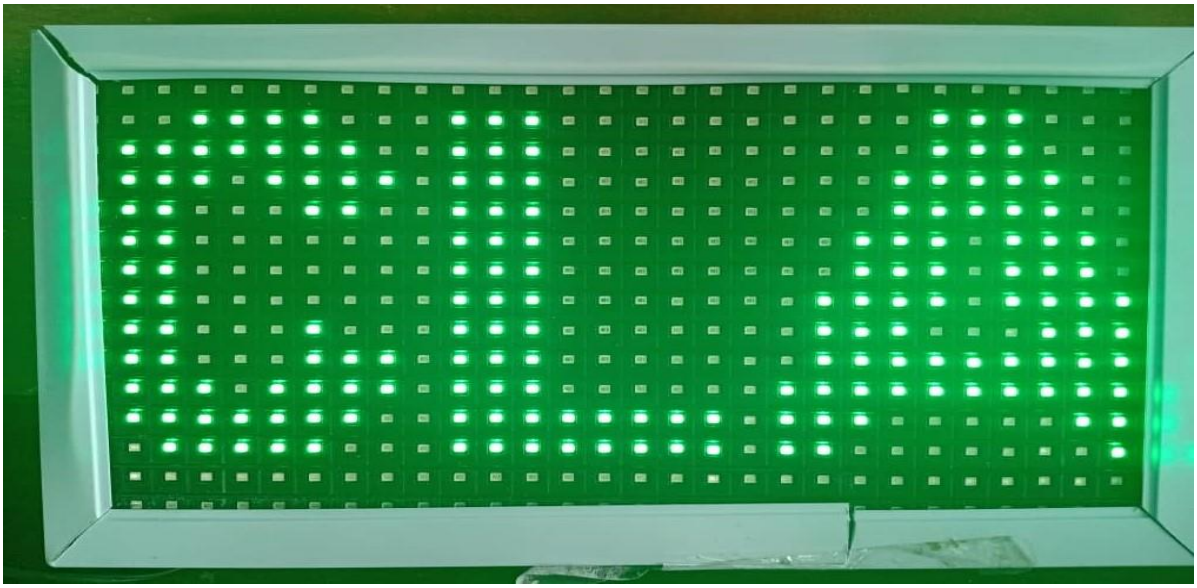

Figure 4.2: Outdoor Unit

4.1 BILL OF ENGINEERING MEASUREMENT AND EVALUATION (BEME)

S/NO	ITEMS	UNIT PRICE	QUANTITY	AMOUNT
1	ESP32 Dev Board	#13,500	10	#135,000
2	P10 Display	#12,500	5	#62,500
3	PIR Sensor	#3,500	10	#35,000
4	DS3231 RTC Module	#7,000	5	#35,000
5	16*2 LCD	#6,000	5	#30,000
6	8Ω Speaker	#600	5	#3,000
7	Project Enclosure	#700	5	#3,500
8	12C Adapter	#1,800	5	#9,000
9	Switch	#200	5	#1,000
10	5V Power Adapter	#4,500	10	#45,000
11	Jumper Wires	#3,000	10	#30,000
12	Audio Amplifier	#2,200	5	#11,000
13	Push Button	#300	20	#6,000
14	Resistors/Capacitors	#200	10	#2000
15	Back cover for display			#9,500
	Total			#417,500

4.7 TOOLS USED

1. Hand saw

2. Multimeter

3. Pliers

4. Screen driver

5. Soldering iron

6. Lead sucker

7. Wire cutter

4.7 CONTRIBUTION TO KNOWLEDGE

1. This study has developed a novel real-time occupancy detection and staff availability display system for smart offices.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 CONCLUSION

The real-time occupancy detection and interactive staff availability display system for smart offices integrates multiple sensors, displays, and communication protocols into a cohesive solution for office status management. The system provides automatic presence detection, real-time status updates, voice feedback, and wireless communication capabilities. The project is also a practical application of microcontroller technology, sensor integration, wireless communication, and embedded systems design suitable for academic and professional environments.

5.2 RECOMMENDAION

Possible Upgrades

1. Web Interface: Remote status control via browser
2. Mobile App: Smartphone control and notifications
3. Calendar Integration: Automatic status based on schedule
4. RFID Access: Employee identification and logging
5. Touch Screen: Replace buttons with capacitive touch LCD
6. Color Display: Use OLED or TFT for richer graphics
7. Multiple Languages: Multilingual voice announcements
8. Data Logging: Record occupancy patterns to SD card
9. Cloud Connectivity: MQTT integration for remote monitoring
10. Solar Power: Solar panel with battery backup

REFERENCES

Aswin S, K V Mahendra Prashanth, " Wireless Office Automation & Security System – A Novel Approach", International Conference on Recent Advances in Electronics and Communication Technology", IEEE, March 2017, Bangalore, India.

Brijesh Kundaliya, Upesh Patel, S.K.Hadia, Smit Patel, Jaanvi Patel, Parv Baro (2024), An IoT and Cloud Enabled Smart Chair for Detection and Notification of Wrong Seating Posture, Journal of Electrical Systems 20-3, pp 5199-5210

Hemanth Inti, Jaddu Swathi Kiran, and Darisipudi Saahil (2023), International Journal of recent development in Science and Technology, Volume 07, Issue 02, May 2023, pp 96-106

Kommy B., Addo E.O., and K.A. Adjei (2017), Design and implementation of seat occupancy Detection System, Journal of Science and Technology, Vol. 37, No. 2, pp. 26-42

Kyle Stone, and Jan Spies (2015), Design and Implementation of an Automatic Staff Availability Tracking System (2015). International conference on Computing Communication and Security (ICCCS), Ponte aox Pinents, Mauritus, pp 1-3

Mannino ANTONINO, Moretti Nicola, Dejaco Mario Claudio, Baresi Luciano & Re Cecconi Fulvio (2019), office building occupancy monitoring through image recognition sensors, Int. J. of Safety and Security Eng., Vol. 9, No. 4, pp. 371–380

Martínez-Estrada, M.; Vuohijoki, T.; Poberznik, A.; Shaikh, A.; Virkki, J.; Gil, I.; Fernández-García, R. A (2023), Smart Chair to Monitor Sitting Posture by Capacitive Textile Sensors. Materials 2023, 16, 4838. Pp 1-15, https://doi.org/ 10.3390/ma1613483

Mengjie Huang, Ian Gibson, and Rui Yang, (2017), "Smart Chair for Monitoring of Sitting Behavior," in The International Conference on Design and Technology, KEG, pages 274–280. DOI 10.18502/keg.v2i2.626

Mingyu Park, Younghoon Song, Jaewon Lee, Jeongyeup Paek (2016), Design and Implementation of a Smart Chair system for IoT, International conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea (South), pp. 1200-1203

Mohamad Syahmi Hilmi Zaharuddin, Shaharil Mohd Shah (2024), A Smart Chair for Sitting Postures Monitoring and Seat Occupancy Detection, Journal of Electronic Voltage and Application, Vol. 5 No. 2, pp 36-55

Ondrej Zeleny, Tomas Fryza, Tomas Bravenec, Shoaib Azizi, Gireesh Nair (2024), Detection of Room Occupancy in Smart Buildings

Pratama Erik Wahyu, and Agus Kiswantono (2022), Electrical Analysis Using Esp-32 Module In Realtime, Journal of Electrical Engineering and Computer Sciences, Vol. 7, No. 2, pp 1273 - 1284

Santhosha Rao, Casbona Jonathan (2019), QR code based Visitor Management System for Smart Offices, International Journal of Recent Technology and Engineering (IJRTE), Volume-8 Issue-4, pp 8787 - 8791

Santhosha Rao, Smitha A, Kunal Kulkarni (2018), "Smart Phone based Cost Effective Visitor Management System for Smart Offices", International Journal of Interactive Mobile Technologies, Vol. 12, No. 6.

Shaikh S.A., Aparna S.Kapare, "Intelligent Office Area Monitoring and Control using Internet of Things", International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering astics, Vol. 6, Issue 6, June 2017

Soy, H. (2021). ESP8266 and ESP32 Series of SoC Microcontrollers. In S. Kocer, O. Dundar & R. Butuner (Eds .), Programmable Smart Microcontroller Cards (pp. 110 –125). ISRES Publishing.

Taraneh Aminosharieh Najafi, Antonio Abramo, Kyandoghere Kyamakya, and Antonio Affanni (2022), Development of a Smart Chair Sensors System and Classification of Sitting Postures with Deep Learning Algorithms, Sensors 22, pp 1-24

Venkateswar Rao B., K.Raju, Md.Asma, D.Bhargavi, M.Rahul (2022), Smart Intelligence Office Chair, Turkish Journal of Computer and Mathematics Education Vol.13 No.03, pp 1199-1202

Yogesh Pawar, Abhay Chopde, Mandar Nandre (2018), Motion Detection Using PIR Sensor, International Research Journal of Engineering and Technology (IRJET), Volume: 5 Issue 4, pp 4753 - 4756

APPENDIX

INDOOR UNIT PROGRAM

```
#include <Arduino.h>
#include <WifiEspNow.h>//esp_now.h
#include <WiFi.h>
unsigned long lastActivityTime = 0;
#include <Wire.h>
#include "mylcd.h"
#include "myrtc.h"
LiquidCrystal_I2C lcd(0x27, 20, 4);
// In your main .ino file
#include <Talkie.h>
#include "Vocab_US_Acorn.h"
#include "Vocab_US_Large.h"
Talkie voice;  // Create Talkie object
#include <EEPROM.h>
char eprom[512];
 //char lcd_putc[30];
  char buff[15]={};
 const char numbers[12]={"0123456789 "};
#define up   !digitalRead(12)
#define down !digitalRead(14)
#define ent  !digitalRead(27)
#define next !digitalRead(27)
#define back !digitalRead(33) //33
//tone(8, notes[thisSensor], 20); pin, freq, duration
// The recipient MAC address. It must be modified for each device.
//static uint8_t PEER[]{0xA6, 0xE5, 0x7C, 0xB3, 0xA5, 0x62};
static uint8_t PEER[]{0xA0, 0xA3, 0xB3, 0x28, 0xD4, 0xC1};
//static uint8_t PEER[]{0xD8, 0xBC, 0x38, 0xE4, 0x99, 0x01};// MAC OF RX
int a=0;
void printReceivedMessage(const uint8_t mac[WIFIESPNOW_ALEN], const uint8_t* buf,
size_t count,void* arg)
{
  Serial.printf("Message from %02X:%02X:%02X:%02X:%02X:%02X\n", mac[0], mac[1],
mac[2], mac[3],
                mac[4], mac[5]);
  for (int i = 0; i < static_cast<int>(count); ++i) {
    Serial.print(static_cast<char>(buf[i]));
   //  lcd.setCursor(1,4);lcd.print(a);a++;
 //  Serial.print("message data=");Serial.println(buf[0]);
  }
  Serial.println();
}
void setup(){
  Serial.begin(115200);
```

```
  EEPROM.begin(512);
  pinMode(12, INPUT_PULLUP);
  pinMode(14, INPUT_PULLUP);
  pinMode(27, INPUT_PULLUP);
   pinMode(33, INPUT_PULLUP);
   pinMode(19, INPUT_PULLUP);
   pinMode(25, OUTPUT);
    lcd.begin();//SDA--SCL

 // Connect to Wi-Fi
 savo:
 lcd.clear();
lcd.setCursor(1,2);lcd.print("COURSE ADVISER");delay(2000); // lcd.clear();
 lcd.clear();
Serial.begin(115200);
 Serial.println();

 WiFi.persistent(false);
 WiFi.mode(WIFI_AP);
 WiFi.disconnect();
 WiFi.softAP("ESPNOW", nullptr, 3);
 WiFi.softAPdisconnect(false);
 // WiFi must be powered on to use ESP-NOW unicast.
 // It could be either AP or STA mode, and does not have to be connected.
 // For best results, ensure both devices are using the same WiFi channel.

 Serial.print("MAC address of this node is ");
 Serial.println(WiFi.softAPmacAddress());
 uint8_t mac[6];
 WiFi.softAPmacAddress(mac);
 Serial.println();
 Serial.println("You can paste the following into the program for the other device:");
 Serial.printf("static uint8_t PEER[]{0x%02X, 0x%02X, 0x%02X, 0x%02X, 0x%02X,
0x%02X};\n", mac[0],
 mac[1], mac[2], mac[3], mac[4], mac[5]);
 Serial.println();

 bool ok = WifiEspNow.begin();
 if (!ok) {
  Serial.println("WifiEspNow.begin() failed");
  ESP.restart();
 }
 WifiEspNow.onReceive(printReceivedMessage, nullptr);
 ok = WifiEspNow.addPeer(PEER);
 if (!ok) {
  Serial.println("WifiEspNow.addPeer() failed");
```

```
    ESP.restart();
  }
 lastActivityTime = millis();
}
uint16_t make16(byte high, byte low){
  return high*256+low;
  }
uint8_t connection_state = 0;
uint16_t reconnect_interval = 10000;
 void write_eeprom(uint16_t a, char d){
//  eprom[a]=d;
for(uint16_t i=0;i<=511;i++)eprom[i]=EEPROM.read(i);
eprom[a]=d;
for(uint16_t i=0;i<=511;i++)EEPROM.write(i,eprom[i]);
EEPROM.commit();
  }
    char eeprom_read(uint16_t a){
return EEPROM.read(a);
  }
   void eeprom_write(uint16_t a, char d){
//  eprom[a]=d;
for(uint16_t i=0;i<=511;i++)eprom[i]=EEPROM.read(i);
eprom[a]=d;
for(uint16_t i=0;i<=511;i++)EEPROM.write(i,eprom[i]);
EEPROM.commit();
  }
    char read_eeprom(uint16_t a){
return EEPROM.read(a);
  }
bool ava=0;
#define prox  !digitalRead(19)
//unsigned long lastActivityTime = 0;
//const unsigned long inactivityThreshold = 300 * 1000; // 5 minutes in milliseconds
//void setup() {
//  pinMode(pirPin, INPUT);
//  Serial.begin(9600);
//  lastActivityTime = millis();
//}
void loop() {
     voice.say(spa_AVAILABLE);delay(200);
     voice.say(spa_AVAILABLE);delay(200);
     voice.say(spa_AVAILABLE);delay(200);
stt:
const unsigned long inactivityThreshold = EEPROM.read(1) * 1000; // 5 minutes in milliseconds
//     goto stt;
//if(prox){x=0;}
```

```
   if (prox) {
   // Motion detected, update last activity time
   lcd.setCursor(1,1);lcd.print("    AVAILABLE       ");
   lastActivityTime = millis();
  }
  unsigned long currentTime = millis();
  if (currentTime - lastActivityTime > inactivityThreshold) {
   Serial.println("No activity detected for 0.5 minutes.");
   // Optionally reset or handle further
   lastActivityTime = currentTime; // Reset timer to avoid repeated messages immediately
   delay(1000); // Short delay to prevent flooding serial output
   lcd.clear();
    lcd.setCursor(1,1);lcd.print("   UNAVAILABLE   ");send_mssg(5);
    while(!prox){lastActivityTime = millis();}
    lcd.setCursor(1,1);lcd.print("Welcome!      ");delay(1000);
    lcd.setCursor(1,1);lcd.print("    AVAILABLE       ");delay(1000);
    voice.say(spa_AVAILABLE);delay(200);
    voice.say(spa_AVAILABLE);delay(200);
    voice.say(spa_AVAILABLE);delay(200);
    goto stt;
  }

 lcd.setCursor(1,2);lcd.print((inactivityThreshold- (currentTime -
lastActivityTime))/1000);lcd.print("     ");
       rtc.refresh();
int start_time=minutesSinceHour0(8, 0);
int stop_time=minutesSinceHour0(16, 30);
int current_time=minutesSinceHour0(rtc.hour(),rtc.minute());
if(current_time>=start_time && current_time<stop_time){}
else {
  lcd.clear();
  lcd.setCursor(1,1);lcd.print("   UNAVAILABLE   ");send_mssg(5);
  while(1){
   rtc.refresh();
   current_time=minutesSinceHour0(rtc.hour(),rtc.minute());
  if(current_time>=start_time && current_time<stop_time){goto stt;}
lcd.setCursor(2,2);sprintf(lcd_putc,"%02u:%02u:%02u
%02u/%02u/%02u",rtc.hour(),rtc.minute(),rtc.second(),rtc.day(),rtc.month(),rtc.year());
lcd.print(lcd_putc);
   delay(100);
   }
  }
lcd.setCursor(4,2);sprintf(lcd_putc,"%02u:%02u:%02u
%02u/%02u/%02u",rtc.hour(),rtc.minute(),rtc.second(),rtc.day(),rtc.month(),rtc.year());
lcd.print(lcd_putc);
 // voice.say(spa_AVAILABLE);
```

```
/*
   char smt1[100] = "IN A MEETING - PLEASE WAIT";
   char smt2[100] = "AVAILABLE BUT BUSY";
   char smt3[100] = "AVAILABLE - KNOCK FIRST";
   char smt4[100] = "IN CLASS - BACK SOON";
*/
//if(ava==0){send_mssg(5);}
 lcd.setCursor(3,3);lcd.print("MEETING");
 lcd.setCursor(13,3);lcd.print("BUSY");
 lcd.setCursor(3,4);lcd.print("KNOCK");
 lcd.setCursor(13,4);lcd.print("CLASS");
 if(up){while(up){if (down){menu(); goto stt;}}
 lcd.setCursor(1,3);lcd.print(">>");
 lcd.setCursor(11,3);lcd.print("  ");
 lcd.setCursor(1,4);lcd.print("  ");
 lcd.setCursor(11,4);lcd.print("  ");send_mssg(1);
 }
 if(down){while(down){if (up){menu(); goto stt;}}
 lcd.setCursor(1,3);lcd.print("  ");
 lcd.setCursor(11,3);lcd.print(">>");
 lcd.setCursor(1,4);lcd.print("  ");
 lcd.setCursor(11,4);lcd.print("  ");send_mssg(2);
 }
  if(ent){while(ent);
 lcd.setCursor(1,3);lcd.print("  ");
 lcd.setCursor(11,3);lcd.print("  ");
 lcd.setCursor(1,4);lcd.print(">>");
 lcd.setCursor(11,4);lcd.print("  ");send_mssg(3);
 }
  if(back){while(back);
 lcd.setCursor(1,3);lcd.print("  ");
 lcd.setCursor(11,3);lcd.print("  ");
 lcd.setCursor(1,4);lcd.print("  ");
 lcd.setCursor(11,4);lcd.print(">>");send_mssg(4);
 }
delay(100);
 goto stt;
}
//void send_mssg(int aa)
//{
//  char msg[100]="savotech";
//  int len = snprintf(msg, sizeof(msg), "%lu :%u" ,millis(),aa);
//  WifiEspNow.send(PEER, reinterpret_cast<const uint8_t*>(msg), len);
//  Serial.println("message sent");
//
//}
```

```
// static uint8_t PEER[]{0xA0, 0xA3, 0xB3, 0x2E, 0xD5, 0xD1};
/**
 * @file
 *
 * EspNowUnicast.ino demonstrates how to transmit unicast ESP-NOW messages with @c
WifiEspNow .
 * You need two ESP8266 or ESP32 devices to run this example.
 * Unicast communication requires the sender to specify the MAC address of the recipient.
 * Thus, you must modify this program for each device.
 *
 * The recommended workflow is:
 * @li 1. Flash the program onto device A.
 * @li 2. Run the program on device A, look at serial console for its MAC address.
 * @li 3. Copy the MAC address of device A, paste it in the @c PEER variable below.
 * @li 4. Flash the program that contains A's MAC address onto device B.
 * @li 5. Run the program on device A, look at serial console for its MAC address.
 * @li 6. Copy the MAC address of device B, paste it in the @c PEER variable below.
 * @li 7. Flash the program that contains B's MAC address onto device A.
 */
//#include <WifiEspNow.h>
//#if defined(ARDUINO_ARCH_ESP8266)
//#include <ESP8266WiFi.h>
//#elif defined(ARDUINO_ARCH_ESP32)
//#include <WiFi.h>
//#endif
// The recipient MAC address. It must be modified for each device.
//static uint8_t PEER[]{0x02, 0x00, 0x00, 0x45, 0x53, 0x50};
//static uint8_t PEER[]{0xD8, 0xBC, 0x38, 0xE4, 0x99, 0x01};// MAC OF RX
//static uint8_t PEER[]{0xA0, 0xA3, 0xB3, 0x2E, 0xD5, 0xD1};//MAC OF TX
//void
//printReceivedMessage(const uint8_t mac[WIFIESPNOW_ALEN], const uint8_t* buf, size_t
count,
//                     void* arg)
//{
//  Serial.printf("Message from %02X:%02X:%02X:%02X:%02X:%02X\n", mac[0], mac[1],
mac[2], mac[3],
//                mac[4], mac[5]);
//  for (int i = 0; i < static_cast<int>(count); ++i) {
//    Serial.print(static_cast<char>(buf[i]));
//  }
//  Serial.println();
//}
//void
//setup()
//{
//  Serial.begin(115200);
```

```
//  Serial.println();
//
//  WiFi.persistent(false);
//  WiFi.mode(WIFI_AP);
//  WiFi.disconnect();
//  WiFi.softAP("ESPNOW", nullptr, 3);
//  WiFi.softAPdisconnect(false);
//  // WiFi must be powered on to use ESP-NOW unicast.
//  // It could be either AP or STA mode, and does not have to be connected.
//  // For best results, ensure both devices are using the same WiFi channel.
//
//  Serial.print("MAC address of this node is ");
//  Serial.println(WiFi.softAPmacAddress());
//
//  uint8_t mac[6];
//  WiFi.softAPmacAddress(mac);
//  Serial.println();
//  Serial.println("You can paste the following into the program for the other device:");
//  Serial.printf("static uint8_t PEER[]{0x%02X, 0x%02X, 0x%02X, 0x%02X, 0x%02X,
0x%02X};\n", mac[0],
// mac[1], mac[2], mac[3], mac[4], mac[5]);
//  Serial.println();
//
//  bool ok = WifiEspNow.begin();
//  if (!ok) {
//    Serial.println("WifiEspNow.begin() failed");
//    ESP.restart();
//  }
//
//  WifiEspNow.onReceive(printReceivedMessage, nullptr);
//
//  ok = WifiEspNow.addPeer(PEER);
//  if (!ok) {
//    Serial.println("WifiEspNow.addPeer() failed");
//    ESP.restart();
//  }
//}
void send_mssg(int a)
{
  char msg[70];
  int len = snprintf(msg, sizeof(msg), "%u hello ESP-NOW from %s at %lu",a,
                WiFi.softAPmacAddress().c_str(), millis());
  WifiEspNow.send(PEER, reinterpret_cast<const uint8_t*>(msg), len);
  delay(100);
}
void menu(){
```

```
    time:
     lcd.clear();
    while (1)
    {
      lcd.setCursor(1,1);lcd.print("        Time:      >");
     rtc.refresh();
      lcd.setCursor(1,1)
      lcd.setCursor(1,2);sprintf(lcd_putc,"%02u:%02u:%02u
%02u/%02u/%02u",rtc.hour(),rtc.minute(),rtc.second(),rtc.day(),rtc.month(),rtc.year());
lcd.print(lcd_putc);
      lcd.setCursor(9,2)
           delay(50);
      if(next){while(next);edit_time();lcd.clear();}// this is to go to edit mode when button "next"
is pressed
      else if (up){while(up);goto node;}
       else if (back){while(back);lcd.clear(); return;}
    }
    node:
 lcd.clear();
         lcd.setCursor(1,1);lcd.print("<Active timer");
    while (1)
    {
           lcd.setCursor(1,2);sprintf(lcd_putc,"%02u",EEPROM.read(1));  lcd.print(lcd_putc);
      delay(50);
      if(next){while(next); lcd.clear();
      lcd.setCursor(1,1);lcd.print("Active timer Edit:");
     int gg=gen_edit(EEPROM.read(1),30, 250,9,2);
        EEPROM.write(1,gg);
        EEPROM.commit();
        goto node;
      }// this is to go to edit mode when button "next" is pressed
     // else if (up){while(up);goto sched;}
      else if (down){while(down);goto time;}
      else if (back){while(back);lcd.clear(); return;}
    }
}
int svt;
int gen_edit(signed int dat,signed int mini, int max,int xx,int yy){
    while (1)
    {
     svt=0;
    if(up){dat++;if
(dat>max){dat=mini;}else;while(up){delay(10);svt++;if(svt>100){while(up){dat++;if
(dat>23){dat=0;}else;lcd.setCursor(xx,yy);  printf (lcd_putc, "%02u",dat); delay(200);}}}}
```

```
else if(down){dat--;if
(dat<mini){dat=max;}else;while(down){delay(10);svt++;if(svt>100){while(down){dat--;if
(dat==-1){dat=2;}else;lcd.setCursor(xx,yy);  printf (lcd_putc, "%02u",dat); delay(200);}}}}
else if(next){while(next);return dat;}
else if(back){while(back);return 255;}
        lcd.setCursor(xx,yy);sprintf(lcd_putc,"%02u    ",dat);lcd.print(lcd_putc);
        delay (50) ;
   }
}
 void edit_time ()
 {
   int min=rtc.minute();
   int hr= rtc.hour();
   int day=rtc.day();
   int mth= rtc.month();
   int yr=rtc.year();
    lcd.clear();
//    rtc_get_time(hr,min,sec);
//    rtc_get_date(day,mth,yr,dow);
rtc.refresh();
 lcd.setCursor(1,1); lcd.print("Set Current Time/Dat");
 lcd.setCursor(1,2);sprintf(lcd_putc,"%02u:%02u   %02u/%02u/%02u",hr,min,day,mth,yr);
lcd.print(lcd_putc);
 lcd.setCursor(1,3); lcd.print("Set Hour:"); hr=gen_edit(rtc.hour(),0,23,10,3);if
(hr==255){return;} lcd.setCursor(1,2);sprintf(lcd_putc,"%02u:%02u
%02u/%02u/%02u",hr,min,day,mth,yr);lcd.print(lcd_putc);
 lcd.setCursor(1,3); lcd.print("Set Minute:"); min=gen_edit(rtc.minute(),0,59,12,3);if
(min==255){return;} lcd.setCursor(1,2);sprintf(lcd_putc,"%02u:%02u
%02u/%02u/%02u",hr,min,day,mth,yr);lcd.print(lcd_putc);
 lcd.setCursor(1,3); lcd.print("Set Day:"); day=gen_edit(rtc.day(),1,31,9,3);if
(day==255){return;} lcd.setCursor(1,2);sprintf(lcd_putc,"%02u:%02u
%02u/%02u/%02u",hr,min,day,mth,yr);lcd.print(lcd_putc);
 lcd.setCursor(1,3); lcd.print("Set Month:"); mth=gen_edit(rtc.month(),1,12,11,3);if
(mth==255){return;} lcd.setCursor(1,2);sprintf(lcd_putc,"%02u:%02u
%02u/%02u/%02u",hr,min,day,mth,yr);lcd.print(lcd_putc);
 lcd.setCursor(1,3); lcd.print("Set Year:"); yr=gen_edit(rtc.year(),20,99,10,3);if
(yr==255){return;} lcd.setCursor(1,2);sprintf(lcd_putc,"%02u:%02u
%02u/%02u/%02u",hr,min,day,mth,yr);lcd.print(lcd_putc);
// rtc_set_date_time(day,mth,yr,1,hr,min,0);
 rtc.set(0, min, hr, 0,day, mth, yr);
 }
/*
WORK REMINING
time setting
time display
automatic unavailable outside the time
```

```
*/
OUTDOOR PROGRAM
#define STATUS_LED 2
 void web1();
  #define lo 10
  #define hi 255
#include <WiFi.h>
#include <WifiEspNow.h>
#include <EEPROM.h>
//static uint8_t PEER[]{0x86, 0xCC, 0xA8, 0x9F, 0x10, 0x46};
static uint8_t PEER[]{0xA0, 0xA3, 0xB3, 0x2E, 0xD5, 0xD1};
char eprom[300];
#include <EEPROM.h>
byte read_eeprom(uint16_t a) {
  return EEPROM.read(a);
}
byte eeprom_read(uint16_t a) {
  return EEPROM.read(a);
}
//byte read_eeprom(uint16_t a)
//void read_eeprom(int);
//     uint8_t ff=read_eeprom(200);
//// //int bb=map(ff,0,100,255,1);
//// int bm=map(ff,100,0,255,10);
//// if(bm>255){bm=255;}
////  Serial.print("brightness of led =");Serial.println(bm);
////analogWrite(PIN_DMD_nOE,bm);//pin22
//=ff;//=map(ff,100,0,255,10);
#ifndef DMD_H_
#define DMD_H_
//Arduino toolchain header, version dependent
#include "Arduino.h"
#include "soc/soc_caps.h"
//SPI library must be included for the SPI scanning/connection method to the DMD
#include <SPI.h>
//
//////////////////////////////////////////////////////////////////////////////////////////
//
//////////////////////////////////////////////////////////////////////////////////////////
#warning CHANGE THESE TO SEMI-ADJUSTABLE PIN DEFS!
//ESP32 pins used for the display connection (Using VSPI)
#define PIN_DMD_nOE    22    // D22 active low Output Enable, setting this low lights all the
LEDs in the selected rows. Can pwm it at very high frequency for brightness control.
#define PIN_DMD_A   17    // D17
```

```
#define PIN_DMD_B   21    // D21
#define PIN_DMD_CLK   4   // D18_SCK  is SPI Clock if SPI is used
#define PIN_DMD_SCLK    16    // D016
#define PIN_DMD_R_DATA    0 // D23_MOSI is SPI Master Out if SPI is used
//Define this chip select pin that the Ethernet W5100 IC or other SPI device uses
//if it is in use during a DMD scan request then scanDisplayBySPI() will exit without conflict!
(and skip that scan)
#define PIN_OTHER_SPI_nCS SS
//
#############################################################################
#########################################
//
#############################################################################
#########################################

//DMD I/O pin macros
#define LIGHT_DMD_ROW_01_05_09_13()       { digitalWrite( PIN_DMD_B,  LOW );
digitalWrite( PIN_DMD_A,  LOW ); }
#define LIGHT_DMD_ROW_02_06_10_14()       { digitalWrite( PIN_DMD_B,  LOW );
digitalWrite( PIN_DMD_A, HIGH ); }
#define LIGHT_DMD_ROW_03_07_11_15()       { digitalWrite( PIN_DMD_B, HIGH );
digitalWrite( PIN_DMD_A,  LOW ); }
#define LIGHT_DMD_ROW_04_08_12_16()       { digitalWrite( PIN_DMD_B, HIGH );
digitalWrite( PIN_DMD_A, HIGH ); }
#define LATCH_DMD_SHIFT_REG_TO_OUTPUT() { digitalWrite( PIN_DMD_SCLK,
HIGH ); digitalWrite( PIN_DMD_SCLK,  LOW ); }
//void OE_DMD_ROWS_OFF()                  {pinMode(PIN_DMD_nOE,OUTPUT);
digitalWrite( PIN_DMD_nOE, LOW  );  }
////#define OE_DMD_ROWS_OFF()             {}//      { ledcWrite(0,0); }
//void OE_DMD_ROWS_ON()
{pinMode(PIN_DMD_nOE,OUTPUT);digitalWrite( PIN_DMD_nOE, HIGH ); }
// //int BRI;
//void OE_DMD_ROWS_ON2()                   {analogWrite( PIN_DMD_nOE, BRI ); }
//Pixel/graphics writing modes (bGraphicsMode)
#define GRAPHICS_NORMAL 0
#define GRAPHICS_INVERSE  1
#define GRAPHICS_TOGGLE 2
#define GRAPHICS_OR   3
#define GRAPHICS_NOR  4
//drawTestPattern Patterns
#define PATTERN_ALT_0 0
#define PATTERN_ALT_1 1
#define PATTERN_STRIPE_0  2
#define PATTERN_STRIPE_1  3
//display screen (and subscreen) sizing
#define DMD_PIXELS_ACROSS 32     //pixels across x axis (base 2 size expected)
```

```
#define DMD_PIXELS_DOWN 16      //pixels down y axis
#define DMD_BITSPERPIXEL    1      //1 bit per pixel, use more bits to allow for pwm screen
brightness control
#define DMD_RAM_SIZE_BYTES
((DMD_PIXELS_ACROSS*DMD_BITSPERPIXEL/8)*DMD_PIXELS_DOWN)
                        // (32x * 1 / 8) = 4 bytes, * 16y = 64 bytes per screen here.
//lookup table for DMD::writePixel to make the pixel indexing routine faster
static byte bPixelLookupTable[8] =
{
  0x80,   //0, bit 7
  0x40,   //1, bit 6
  0x20,   //2. bit 5
  0x10,   //3, bit 4
  0x08,   //4, bit 3
  0x04,   //5, bit 2
  0x02,   //6, bit 1
  0x01    //7, bit 0
};
// Font Indices
#define FONT_LENGTH     0
#define FONT_FIXED_WIDTH    2
#define FONT_HEIGHT     3
#define FONT_FIRST_CHAR   4
#define FONT_CHAR_COUNT   5
#define FONT_WIDTH_TABLE  6
typedef uint8_t (*FontCallback)(const uint8_t*);
//The main class of DMD library functions
class DMD
{
 public:
   //Instantiate the DMD
   DMD(byte panelsWide, byte panelsHigh);
 //virtual ~DMD();
 //Set or clear a pixel at the x and y location (0,0 is the top left corner)
 void writePixel( unsigned int bX, unsigned int bY, byte bGraphicsMode, byte bPixel );
 //Draw a string
 void drawString( int bX, int bY, const char* bChars, byte length, byte bGraphicsMode);
 //Select a text font
 void selectFont(const uint8_t* font);
 //Draw a single character
 int drawChar(const int bX, const int bY, const unsigned char letter, byte bGraphicsMode);

 //Find the width of a character
 int charWidth(const unsigned char letter);
 //Draw a scrolling string
 void drawMarquee( const char* bChars, byte length, int left, int top);
```

```
  //Move the maquee accross by amount
  boolean stepMarquee( int amountX, int amountY);
  //Clear the screen in DMD RAM
  void clearScreen( byte bNormal );
  //Draw or clear a line from x1,y1 to x2,y2
  void drawLine( int x1, int y1, int x2, int y2, byte bGraphicsMode );
  //Draw or clear a circle of radius r at x,y centre
  void drawCircle( int xCenter, int yCenter, int radius, byte bGraphicsMode );
  //Draw or clear a box(rectangle) with a single pixel border
  void drawBox( int x1, int y1, int x2, int y2, byte bGraphicsMode );
  //Draw or clear a filled box(rectangle) with a single pixel border
  void drawFilledBox( int x1, int y1, int x2, int y2, byte bGraphicsMode );
  //Draw the selected test pattern
  void drawTestPattern( byte bPattern );
  //Scan the dot matrix LED panel display, from the RAM mirror out to the display hardware.
  //Call 4 times to scan the whole display which is made up of 4 interleaved rows within the 16
total rows.
  //Insert the calls to this function into the main lobvcbop for the highest call rate, or from a timer
interrupt
  void scanDisplayBySPI();
  private:
    void drawCircleSub( int cx, int cy, int x, int y, byte bGraphicsMode );
    //Mirror of DMD pixels in RAM, ready to be clocked out by the main lohjvhchop or high
speed timer calls
    byte *bDMDScreenRAM;
    //Marquee values
    char marqueeText[256];
    byte marqueeLength;
    int marqueeWidth;
    int marqueeHeight;
    int marqueeOffsetX;
    int marqueeOffsetY;
    //Pointer to current font
    const uint8_t* Font;
    //Display information
    byte DisplaysWide;
    byte DisplaysHigh;
    byte DisplaysTotal;
    int row1, row2, row3;
    //scanning pointer into bDMDScreenRAM,  init @ 48 for the first valid scan
    volatile byte bDMDByte;
    //uninitalised pointer to SPI object
  SPIClass * vspi = NULL;
  static const int spiClk = 4000000; // 4 MHz SPI clock
};
#endif /* DMD_H_ */
```

```
/*--------------------------------------------------------------------------------------
 and instantiation of DMD library
 Note this currently uses the SPI port for the fastest performance to the DMD, be
 careful of possible conflicts with other SPI port devices
--------------------------------------------------------------------------------------*/
DMD::DMD(byte panelsWide, byte panelsHigh)
{
   uint16_t ui;
   DisplaysWide=panelsWide;
   DisplaysHigh=panelsHigh;
   DisplaysTotal=DisplaysWide*DisplaysHigh;
   row1 = DisplaysTotal<<4;
   row2 = DisplaysTotal<<5;
   row3 = ((DisplaysTotal<<2)*3)<<2;
   bDMDScreenRAM = (byte *) malloc(DisplaysTotal*DMD_RAM_SIZE_BYTES);
  //initialise instance of the SPIClass attached to vspi
  vspi = new SPIClass(VSPI);
   // initialize the SPI port
   vspi->begin();    // initiate VSPI with the default pinsinitiat VSPI with the defualt pins
   digitalWrite(PIN_DMD_A, LOW); //
   digitalWrite(PIN_DMD_B, LOW); //
   digitalWrite(PIN_DMD_CLK, LOW); //
   digitalWrite(PIN_DMD_SCLK, LOW);  //
   digitalWrite(PIN_DMD_R_DATA, HIGH); //
   digitalWrite(PIN_DMD_nOE, LOW); //
   pinMode(PIN_DMD_A, OUTPUT); //
   pinMode(PIN_DMD_B, OUTPUT); //
   pinMode(PIN_DMD_CLK, OUTPUT); //
   pinMode(PIN_DMD_SCLK, OUTPUT);  //
   pinMode(PIN_DMD_R_DATA, OUTPUT);  //
   pinMode(PIN_DMD_nOE, OUTPUT); //
   clearScreen(true);
   // init the scan line/ram pointer to the required start point
   bDMDByte = 0;
}
//DMD::~DMD()
//{
//   // nothing needed here
//}
/*--------------------------------------------------------------------------------------
 Set or clear a pixel at the x and y location (0,0 is the top left corner)
--------------------------------------------------------------------------------------*/
void
 DMD::writePixel(unsigned int bX, unsigned int bY, byte bGraphicsMode, byte bPixel)
{
   unsigned int uiDMDRAMPointer;
```

```
  if (bX >= (DMD_PIXELS_ACROSS*DisplaysWide) || bY >= (DMD_PIXELS_DOWN *
DisplaysHigh)) {
    return;
  }
  byte panel=(bX/DMD_PIXELS_ACROSS) + (DisplaysWide*(bY/DMD_PIXELS_DOWN));
  bX=(bX % DMD_PIXELS_ACROSS) + (panel<<5);
  bY=bY % DMD_PIXELS_DOWN;
  //set pointer to DMD RAM byte to be modified
  uiDMDRAMPointer = bX/8 + bY*(DisplaysTotal<<2);
  byte lookup = bPixelLookupTable[bX & 0x07];
  switch (bGraphicsMode) {
  case GRAPHICS_NORMAL:
   if (bPixel == true)
  bDMDScreenRAM[uiDMDRAMPointer] &= ~lookup;  // zero bit is pixel on
   else
  bDMDScreenRAM[uiDMDRAMPointer] |= lookup; // one bit is pixel off
   break;
  case GRAPHICS_INVERSE:
   if (bPixel == false)
    bDMDScreenRAM[uiDMDRAMPointer] &= ~lookup;  // zero bit is pixel on
   else
    bDMDScreenRAM[uiDMDRAMPointer] |= lookup; // one bit is pixel off
   break;
  case GRAPHICS_TOGGLE:
   if (bPixel == true) {
  if ((bDMDScreenRAM[uiDMDRAMPointer] & lookup) == 0)
    bDMDScreenRAM[uiDMDRAMPointer] |= lookup; // one bit is pixel off
  else
    bDMDScreenRAM[uiDMDRAMPointer] &= ~lookup;  // one bit is pixel off
   }
   break;
  case GRAPHICS_OR:
   //only set pixels on
   if (bPixel == true)
    bDMDScreenRAM[uiDMDRAMPointer] &= ~lookup;  // zero bit is pixel on
   break;
  case GRAPHICS_NOR:
   //only clear on pixels
   if ((bPixel == true) &&
    ((bDMDScreenRAM[uiDMDRAMPointer] & lookup) == 0))
    bDMDScreenRAM[uiDMDRAMPointer] |= lookup; // one bit is pixel off
   break;
  }
}
void DMD::drawString(int bX, int bY, const char *bChars, byte length,
      byte bGraphicsMode)
```

```
{
   if (bX >= (DMD_PIXELS_ACROSS*DisplaysWide) || bY >= DMD_PIXELS_DOWN *
DisplaysHigh)
  return;
   uint8_t height = pgm_read_byte(this->Font + FONT_HEIGHT);
   if (bY+height<0) return;
   int strWidth = 0;
  this->drawLine(bX -1 , bY, bX -1 , bY + height, GRAPHICS_INVERSE);
   for (int i = 0; i < length; i++) {
      int charWide = this->drawChar(bX+strWidth, bY, bChars[i], bGraphicsMode);
     if (charWide > 0) {
        strWidth += charWide ;
        this->drawLine(bX + strWidth , bY, bX + strWidth , bY + height,
GRAPHICS_INVERSE);
         strWidth++;
      } else if (charWide < 0) {
         return;
      }
      if ((bX + strWidth) >= DMD_PIXELS_ACROSS * DisplaysWide || bY >=
DMD_PIXELS_DOWN * DisplaysHigh) return;
   }
}
void DMD::drawMarquee(const char *bChars, byte length, int left, int top)
{
   marqueeWidth = 0;
   for (int i = 0; i < length; i++) {
     marqueeText[i] = bChars[i];
     marqueeWidth += charWidth(bChars[i]) + 1;
   }
   marqueeHeight=pgm_read_byte(this->Font + FONT_HEIGHT);
   marqueeText[length] = '\0';
   marqueeOffsetY = top;
   marqueeOffsetX = left;
   marqueeLength = length;
   drawString(marqueeOffsetX, marqueeOffsetY, marqueeText, marqueeLength,
    GRAPHICS_NORMAL);
}
boolean DMD::stepMarquee(int amountX, int amountY)
{
   boolean ret=false;
   marqueeOffsetX += amountX;
   marqueeOffsetY += amountY;
   if (marqueeOffsetX < -marqueeWidth) {
     marqueeOffsetX = DMD_PIXELS_ACROSS * DisplaysWide;
     clearScreen(true);
      ret=true;
```

```
  } else if (marqueeOffsetX > DMD_PIXELS_ACROSS * DisplaysWide) {
    marqueeOffsetX = -marqueeWidth;
    clearScreen(true);
      ret=true;
  }
  if (marqueeOffsetY < -marqueeHeight) {
    marqueeOffsetY = DMD_PIXELS_DOWN * DisplaysHigh;
    clearScreen(true);
      ret=true;
  } else if (marqueeOffsetY > DMD_PIXELS_DOWN * DisplaysHigh) {
    marqueeOffsetY = -marqueeHeight;
    clearScreen(true);
      ret=true;
  }
  // Special case horizontal scrolling to improve speed
  if (amountY==0 && amountX==-1) {
      // Shift entire screen one bit
      for (int i=0; i<DMD_RAM_SIZE_BYTES*DisplaysTotal;i++) {
          if ((i%(DisplaysWide*4)) == (DisplaysWide*4) -1) {
              bDMDScreenRAM[i]=(bDMDScreenRAM[i]<<1)+1;
          } else {
              bDMDScreenRAM[i]=(bDMDScreenRAM[i]<<1) + ((bDMDScreenRAM[i+1] &
0x80) >>7);
          }
      }
      // Redraw last char on screen
      int strWidth=marqueeOffsetX;
      for (byte i=0; i < marqueeLength; i++) {
          int wide = charWidth(marqueeText[i]);
          if (strWidth+wide >= DisplaysWide*DMD_PIXELS_ACROSS) {
              drawChar(strWidth, marqueeOffsetY,marqueeText[i],GRAPHICS_NORMAL);
              return ret;
          }
          strWidth += wide+1;
      }
  } else if (amountY==0 && amountX==1) {
      // Shift entire screen one bit
      for (int i=(DMD_RAM_SIZE_BYTES*DisplaysTotal)-1; i>=0;i--) {
          if ((i%(DisplaysWide*4)) == 0) {
              bDMDScreenRAM[i]=(bDMDScreenRAM[i]>>1)+128;
          } else {
              bDMDScreenRAM[i]=(bDMDScreenRAM[i]>>1) + ((bDMDScreenRAM[i-1] & 1)
<<7);
          }
      }
      // Redraw last char on screen
```

```
      int strWidth=marqueeOffsetX;
      for (byte i=0; i < marqueeLength; i++) {
         int wide = charWidth(marqueeText[i]);
         if (strWidth+wide >= 0) {
            drawChar(strWidth, marqueeOffsetY,marqueeText[i],GRAPHICS_NORMAL);
            return ret;
         }
         strWidth += wide+1;
      }
   } else {
      drawString(marqueeOffsetX, marqueeOffsetY, marqueeText, marqueeLength,
       GRAPHICS_NORMAL);
   }
   return ret;
}
/*--------------------------------------------------------------------------------------
 Clear the screen in DMD RAM
--------------------------------------------------------------------------------------*/
void DMD::clearScreen(byte bNormal)
{
   if (bNormal) // clear all pixels
      memset(bDMDScreenRAM,0xFF,DMD_RAM_SIZE_BYTES*DisplaysTotal);
   else // set all pixels
      memset(bDMDScreenRAM,0x00,DMD_RAM_SIZE_BYTES*DisplaysTotal);
}
/*--------------------------------------------------------------------------------------
 Draw or clear a line from x1,y1 to x2,y2
--------------------------------------------------------------------------------------*/
void DMD::drawLine(int x1, int y1, int x2, int y2, byte bGraphicsMode)
{
   int dy = y2 - y1;
   int dx = x2 - x1;
   int stepx, stepy;
   if (dy < 0) {
     dy = -dy;
     stepy = -1;
   } else {
     stepy = 1;
   }
   if (dx < 0) {
     dx = -dx;
     stepx = -1;
   } else {
     stepx = 1;
   }
   dy <<= 1;     // dy is now 2*dy
```

```
  dx <<= 1;     // dx is now 2*dx
  writePixel(x1, y1, bGraphicsMode, true);
  if (dx > dy) {
    int fraction = dy - (dx >> 1);  // same as 2*dy - dx
    while (x1 != x2) {
       if (fraction >= 0) {
         y1 += stepy;
         fraction -= dx; // same as fraction -= 2*dx
       }
       x1 += stepx;
       fraction += dy; // same as fraction -= 2*dy
       writePixel(x1, y1, bGraphicsMode, true);
    }
  } else {
    int fraction = dx - (dy >> 1);
    while (y1 != y2) {
       if (fraction >= 0) {
         x1 += stepx;
         fraction -= dy;
       }
       y1 += stepy;
       fraction += dx;
       writePixel(x1, y1, bGraphicsMode, true);
    }
  }
}
/*---------------------------------------------------------------------------------------
 Draw or clear a circle of radius r at x,y centre
---------------------------------------------------------------------------------------*/
void DMD::drawCircle(int xCenter, int yCenter, int radius,
        byte bGraphicsMode)
{
  int x = 0;
  int y = radius;
  int p = (5 - radius * 4) / 4;
  drawCircleSub(xCenter, yCenter, x, y, bGraphicsMode);
  while (x < y) {
    x++;
    if (p < 0) {
       p += 2 * x + 1;
    } else {
       y--;
       p += 2 * (x - y) + 1;
    }
    drawCircleSub(xCenter, yCenter, x, y, bGraphicsMode);
  }
```

```
}
void DMD::drawCircleSub(int cx, int cy, int x, int y, byte bGraphicsMode)
{
   if (x == 0) {
     writePixel(cx, cy + y, bGraphicsMode, true);
     writePixel(cx, cy - y, bGraphicsMode, true);
     writePixel(cx + y, cy, bGraphicsMode, true);
     writePixel(cx - y, cy, bGraphicsMode, true);
   } else if (x == y) {
     writePixel(cx + x, cy + y, bGraphicsMode, true);
     writePixel(cx - x, cy + y, bGraphicsMode, true);
     writePixel(cx + x, cy - y, bGraphicsMode, true);
     writePixel(cx - x, cy - y, bGraphicsMode, true);
   } else if (x < y) {
     writePixel(cx + x, cy + y, bGraphicsMode, true);
     writePixel(cx - x, cy + y, bGraphicsMode, true);
     writePixel(cx + x, cy - y, bGraphicsMode, true);
     writePixel(cx - x, cy - y, bGraphicsMode, true);
     writePixel(cx + y, cy + x, bGraphicsMode, true);
     writePixel(cx - y, cy + x, bGraphicsMode, true);
     writePixel(cx + y, cy - x, bGraphicsMode, true);
     writePixel(cx - y, cy - x, bGraphicsMode, true);
   }
}
/*--------------------------------------------------------------------------------------
 Draw or clear a box(rectangle) with a single pixel border
--------------------------------------------------------------------------------------*/
void DMD::drawBox(int x1, int y1, int x2, int y2, byte bGraphicsMode)
{
   drawLine(x1, y1, x2, y1, bGraphicsMode);
   drawLine(x2, y1, x2, y2, bGraphicsMode);
   drawLine(x2, y2, x1, y2, bGraphicsMode);
   drawLine(x1, y2, x1, y1, bGraphicsMode);
}
/*--------------------------------------------------------------------------------------
 Draw or clear a filled box(rectangle) with a single pixel border
--------------------------------------------------------------------------------------*/
void DMD::drawFilledBox(int x1, int y1, int x2, int y2,
     byte bGraphicsMode)
{
   for (int b = x1; b <= x2; b++) {
     drawLine(b, y1, b, y2, bGraphicsMode);
   }
}
/*--------------------------------------------------------------------------------------
 Draw the selected test pattern
```

```
-------------------------------------------------------------------------------*/
void DMD::drawTestPattern(byte bPattern)
{
  unsigned int ui;
  int numPixels=DisplaysTotal * DMD_PIXELS_ACROSS * DMD_PIXELS_DOWN;
  int pixelsWide=DMD_PIXELS_ACROSS*DisplaysWide;
  for (ui = 0; ui < numPixels; ui++) {
    switch (bPattern) {
    case PATTERN_ALT_0: // every alternate pixel, first pixel on
      if ((ui & pixelsWide) == 0)
        //even row
        writePixel((ui & (pixelsWide-1)), ((ui & ~(pixelsWide-1)) / pixelsWide),
GRAPHICS_NORMAL, ui & 1);
      else
        //odd row
        writePixel((ui & (pixelsWide-1)), ((ui & ~(pixelsWide-1)) / pixelsWide),
GRAPHICS_NORMAL, !(ui & 1));
      break;
    case PATTERN_ALT_1: // every alternate pixel, first pixel off
      if ((ui & pixelsWide) == 0)
        //even row
        writePixel((ui & (pixelsWide-1)), ((ui & ~(pixelsWide-1)) / pixelsWide),
GRAPHICS_NORMAL, !(ui & 1));
      else
        //odd row
        writePixel((ui & (pixelsWide-1)), ((ui & ~(pixelsWide-1)) / pixelsWide),
GRAPHICS_NORMAL, ui & 1);
      break;
    case PATTERN_STRIPE_0:  // vertical stripes, first stripe on
      writePixel((ui & (pixelsWide-1)), ((ui & ~(pixelsWide-1)) / pixelsWide),
GRAPHICS_NORMAL, ui & 1);
      break;
    case PATTERN_STRIPE_1:  // vertical stripes, first stripe off
      writePixel((ui & (pixelsWide-1)), ((ui & ~(pixelsWide-1)) / pixelsWide),
GRAPHICS_NORMAL, !(ui & 1));
      break;
      }
  }
}
uint8_t BRI;
/*-------------------------------------------------------------------------------
 Scan the dot matrix LED panel display, from the RAM mirror out to the display hardware.
 Call 4 times to scan the whole display which is made up of 4 interleaved rows within the 16 total
rows.
 Insert the calls to this function into the main logchgchop for the highest call rate, or from a timer
interrupt
```

```
----------------------------------------------------------------------------------*/
void DMD::scanDisplayBySPI()
{
   //if PIN_OTHER_SPI_nCS is in use during a DMD scan request then scanDisplayBySPI()
will exit without conflict! (and skip that scan)
  if( digitalRead( PIN_OTHER_SPI_nCS ) == HIGH )
  {
      //SPI transfer pixels to the display hardware shift registers
      int rowsize=DisplaysTotal<<2;
      int offset=rowsize * bDMDByte;
      for (int i=0;i<rowsize;i++) {
   vspi->beginTransaction(SPISettings(spiClk, MSBFIRST, SPI_MODE0));
   vspi->transfer(bDMDScreenRAM[offset+i+row3]);
   vspi->transfer(bDMDScreenRAM[offset+i+row2]);
   vspi->transfer(bDMDScreenRAM[offset+i+row1]);
   vspi->transfer(bDMDScreenRAM[offset+i]);
   vspi->endTransaction();
      }
     OE_DMD_ROWS_OFF();
      LATCH_DMD_SHIFT_REG_TO_OUTPUT();
      switch (bDMDByte) {
      case 0:     // row 1, 5, 9, 13 were clocked out
         LIGHT_DMD_ROW_01_05_09_13();
         bDMDByte=1;
         break;
      case 1:     // row 2, 6, 10, 14 were clocked out
         LIGHT_DMD_ROW_02_06_10_14();
         bDMDByte=2;
         break;
      case 2:     // row 3, 7, 11, 15 were clocked out
         LIGHT_DMD_ROW_03_07_11_15();
         bDMDByte=3;
         break;
      case 3:     // row 4, 8, 12, 16 were clocked out
         LIGHT_DMD_ROW_04_08_12_16();
         bDMDByte=0;
         break;
      }
     if (BRI>250){
      OE_DMD_ROWS_ON();
      }
     else {OE_DMD_ROWS_ON2();}
   }
}
void DMD::selectFont(const uint8_t * font)
{
```

```
  this->Font = font;
}
int DMD::drawChar(const int bX, const int bY, const unsigned char letter, byte bGraphicsMode)
{
  if (bX > (DMD_PIXELS_ACROSS*DisplaysWide) || bY >
(DMD_PIXELS_DOWN*DisplaysHigh) ) return -1;
  unsigned char c = letter;
  uint8_t height = pgm_read_byte(this->Font + FONT_HEIGHT);
  if (c == ' ') {
    int charWide = charWidth(' ');
    this->drawFilledBox(bX, bY, bX + charWide, bY + height, GRAPHICS_INVERSE);
    return charWide;
  }
  uint8_t width = 0;
  uint8_t bytes = (height + 7) / 8;
  uint8_t firstChar = pgm_read_byte(this->Font + FONT_FIRST_CHAR);
  uint8_t charCount = pgm_read_byte(this->Font + FONT_CHAR_COUNT);
  uint16_t index = 0;
  if (c < firstChar || c >= (firstChar + charCount)) return 0;
  c -= firstChar;
  if (pgm_read_byte(this->Font + FONT_LENGTH) == 0
    && pgm_read_byte(this->Font + FONT_LENGTH + 1) == 0) {
    // zero length is flag indicating fixed width font (array does not contain width data entries)
    width = pgm_read_byte(this->Font + FONT_FIXED_WIDTH);
    index = c * bytes * width + FONT_WIDTH_TABLE;
  } else {
    // variable width font, read width data, to get the index
    for (uint8_t i = 0; i < c; i++) {
      index += pgm_read_byte(this->Font + FONT_WIDTH_TABLE + i);
    }
    index = index * bytes + charCount + FONT_WIDTH_TABLE;
    width = pgm_read_byte(this->Font + FONT_WIDTH_TABLE + c);
  }
  if (bX < -width || bY < -height) return width;
  // last but not least, draw the character
  for (uint8_t j = 0; j < width; j++) { // Width
    for (uint8_t i = bytes - 1; i < 254; i--) { // Vertical Bytes
      uint8_t data = pgm_read_byte(this->Font + index + j + (i * width));
      int offset = (i * 8);
      if ((i == bytes - 1) && bytes > 1) {
        offset = height - 8;
        }
      for (uint8_t k = 0; k < 8; k++) { // Vertical bits
        if ((offset+k >= i*8) && (offset+k <= height)) {
          if (data & (1 << k)) {
            writePixel(bX + j, bY + offset + k, bGraphicsMode, true);
```

```
            } else {
              writePixel(bX + j, bY + offset + k, bGraphicsMode, false);
            }
          }
        }
      }
    }
    return width;
}
int DMD::charWidth(const unsigned char letter)
{
    unsigned char c = letter;
    // Space is often not included in font so use width of 'n'
    if (c == ' ') c = 'n';
    uint8_t width = 0;
    uint8_t firstChar = pgm_read_byte(this->Font + FONT_FIRST_CHAR);
    uint8_t charCount = pgm_read_byte(this->Font + FONT_CHAR_COUNT);
    uint16_t index = 0;
    if (c < firstChar || c >= (firstChar + charCount)) {
      return 0;
    }
    c -= firstChar;
    if (pgm_read_byte(this->Font + FONT_LENGTH) == 0
  && pgm_read_byte(this->Font + FONT_LENGTH + 1) == 0) {
      // zero length is flag indicating fixed width font (array does not contain width data entries)
      width = pgm_read_byte(this->Font + FONT_FIXED_WIDTH);
    } else {
      // variable width font, read width data
      width = pgm_read_byte(this->Font + FONT_WIDTH_TABLE + c);
    }
    return width;
}
//#include "SystemFont5x7.h"
#include "Arial_black_16.h"
//#include "Arial_38b.h"
//#include "Arial_black_16_ISO_8859_1.h"
//#include "Arial_black21.h"
//#include "Arial14.h"
//#include "BodoniMTBlack24.h"
//#include "Comic24.h"
//#include "Droid_Sans_24.h"
//#include "Tahoma_32.h"
//#include "Droid_Sans_16.h"
#include <Preferences.h>
//----------------------------------------
//----------------------------------------Defining the key.
```

```
// "Key" functions like a password. In order to change the text on the P10, the user must know
the "key".
// You can change it to another word.
#define key_Txt "88888888"
//----------------------------------------
// Fire up the DMD library as dmd.
#define DISPLAYS_ACROSS 2
#define DISPLAYS_DOWN 1
DMD dmd(DISPLAYS_ACROSS, DISPLAYS_DOWN);
// Timer p.
// create a hardware timer  of ESP32
hw_timer_t * timer = NULL;
//----------------------------------------SSID and PASSWORD of your WiFi network.
char ssid[15] = "SmartRoyal";
char password[15] = "";
//----------------------------------------
String display_Modes = "";
String single_Row_Txt = "";
String key = "";
String hymn0 = "";
String hymn1 = "";
String hymn2 = "";
String hymn3 = "";
String hymn4 = "";
String hymn5 = "";
String hymn6 = "";
String hymn7 = "";
String fff = "";
int dd;
String double_Row_First_Txt = "";
int double_Row_First_Txt_Pos = 0;
String double_Row_Second_Txt = "";
// Server on port 80.
// WebServer server(80);
// Initialize Preferences.
Preferences preferences;
//_____________________________________________________________________________
____IRAM_ATTR triggerScan()
// Interrupt handler for Timer1 (TimerOne) driven DMD refresh scanning,
// this gets called at the period set in Timer1.initialize();
void IRAM_ATTR triggerScan() {
  dmd.scanDisplayBySPI();
}
//_____________________________________________________________________________
//_____________________________________________________________________________
__handleRoot()
```

```
// This routine is executed when you open ESP32 IP Address in browser.
// void handleRoot() {
//   server.send(200, "text/html", MAIN_page); //Send web page
// }
// void handleAccessRoot() {
//   server.send(200, "text/html", access_page); //Send web page
// }
//_____

//_____
____handleSettings().
// Subroutine to handle settings. The displayed text and others are set here.
int extract_number2(String dd, char low, char high, int c) {
  int st = 0;
  int sp = 0;
  for (int x = c; x <=30+c; x++) {
    if (dd[x] == low) {
      st = x;
    }
    else if (dd[x] == high) {
      sp = x;
      break;
    }
  }
      Serial.println("start="); Serial.print(dd[st]);
       Serial.print("  stop="); Serial.println(dd[sp]);
proc:
  int res = (dd[sp - 1] - 48);
  if (isDigit(dd[sp - 2])) {
    res = res + (dd[sp - 2] - 48) * 10;
  }
  Serial.println("extracted number="); Serial.println(res);
  return res;
}
int extract_number(String dd, int tens, int unit) {
     Serial.print("data="); Serial.print(dd[tens]);Serial.println(dd[unit]);
proc:
  int res = (dd[tens] - 48)*10 +(dd[unit] - 48);
  Serial.print("extracted number="); Serial.println(res);
  return res;
}
/_____
//_____
____getValue()
// String function to split strings based on certain characters.
String getValue(String data, char separator, int index) {
```

```
  int found = 0;
  int strIndex[] = { 0, -1 };
  int maxIndex = data.length() - 1;
  for (int i = 0; i <= maxIndex && found <= index; i++) {
    if (data.charAt(i) == separator || i == maxIndex) {
      found++;
      strIndex[0] = strIndex[1] + 1;
      strIndex[1] = (i == maxIndex) ? i + 1 : i;
    }
  }
  return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}
//_____
//_____
____Single_Row_Display_Mode()
// Subroutine for displaying "running text" on P10 in Single Row mode.
void mssg() {
  char CA_single_Row_Txt[hymn0.length() + 1];
  hymn0.toCharArray(CA_single_Row_Txt, hymn0.length() + 1);
  dmd.clearScreen(true);
  dmd.selectFont(Arial_Black_16);
  dmd.drawMarquee(CA_single_Row_Txt, hymn0.length(), (32) - 1, 0);
  long start = millis();
  long timer = start;
  boolean ret = false;
  while (!ret) {
    if ((timer + 30) < millis()) {
      ret = dmd.stepMarquee(-1, 0);
      timer = millis();
    }
  }
  delay(1000);
}
void draw16bit(int x, uint16_t d) {
  for (int y = 0; y <= 15; y++)  {
    dmd.writePixel(x, y, GRAPHICS_NORMAL, bitRead(d, y));
  }
}
void draw32bit(int x, uint32_t d) {
  for (int y = 0; y <= 15; y++)  {
    dmd.writePixel(x, y, GRAPHICS_NORMAL, bitRead(d, y));
    dmd.writePixel(x-96, y, GRAPHICS_NORMAL, bitRead(d, y+16));
  }
}
void draw32bit_b(int x, uint32_t d) {
  for (int y = 0; y <= 15; y++)  {
```

```
    dmd.writePixel(x, y, GRAPHICS_NORMAL, bitRead(d, y));
    dmd.writePixel(x-64, y, GRAPHICS_NORMAL, bitRead(d, y+16));
  }
}
void write_eeprom(uint16_t a, char d) {
  //  eprom[a]=d;
  for (uint16_t i = 0; i <= 99; i++)eprom[i] = EEPROM.read(i);
  eprom[a] = d;
  for (uint16_t i = 0; i <= 99; i++)EEPROM.write(i, eprom[i]);
  EEPROM.commit();
}
void eeprom_write(uint16_t a, char d) {
  //  eprom[a]=d;
  for (uint16_t i = 0; i <= 99; i++)eprom[i] = EEPROM.read(i);
  eprom[a] = d;
  for (uint16_t i = 0; i <= 99; i++)EEPROM.write(i, eprom[i]);
  EEPROM.commit();
}
//WiFiServer server(80);
void setup(void) {
  pinMode(STATUS_LED, OUTPUT);
   digitalWrite(STATUS_LED,LOW);
  // put your s code here, to run once:
  EEPROM.begin(100);
  Serial.begin(115200);
  // delay(1000);
 pinMode(26, OUTPUT);
  pinMode(27, INPUT_PULLUP);
   WiFi.persistent(false);
  WiFi.mode(WIFI_AP);
  WiFi.disconnect();
  WiFi.softAP("ESPNOW", nullptr, 3);
  WiFi.softAPdisconnect(false);
  // WiFi must be powered on to use ESP-NOW unicast.
  // It could be either AP or STA mode, and does not have to be connected.
  // For best results, ensure both devices are using the same WiFi channel.
  Serial.print("MAC address of this node is ");
  Serial.println(WiFi.softAPmacAddress());
  uint8_t mac[6];
  WiFi.softAPmacAddress(mac);
  Serial.println();
  Serial.println("You can paste the following into the program for the other device:");
  Serial.printf("static uint8_t PEER[]{0x%02X, 0x%02X, 0x%02X, 0x%02X, 0x%02X,
0x%02X};\n", mac[0],
              mac[1], mac[2], mac[3], mac[4], mac[5]);
  Serial.println();
```

```
  bool ok = WifiEspNow.begin();
  if (!ok) {
    Serial.println("WifiEspNow.begin() failed");
    ESP.restart();
  }
  WifiEspNow.onReceive(printReceivedMessage, nullptr);
  ok = WifiEspNow.addPeer(PEER);
  if (!ok) {
    Serial.println("WifiEspNow.addPeer() failed");
    ESP.restart();
  }
  uint8_t cpuClock = ESP.getCpuFreqMHz();
  delay(50);
  // Use 1st timer of 4.
  // devide cpu clock speed on its speed value by MHz to get 1us for each signal  of the timer.
  timer = timerBegin(0, cpuClock, true);
  delay(50);
  timerAttachInterrupt(timer, &triggerScan, true);
  delay(50);
  timerAlarmWrite(timer, 300, true);
  delay(50);
  Serial.println();
  Serial.println("Start an alarm.");
  // Start an alarm.
  timerAlarmEnable(timer);
  dmd.selectFont(Arial_Black_16);
  Serial.println();
  Serial.println("Clear Screen.");
  // clear/init the DMD pixels held in RAM.
  // true is normal (all pixels off), false is negative (all pixels on).
  dmd.clearScreen(true);
  delay(50);
  // While the process of connecting to a WiFi network is in progress or when the process of
creating an access point is in progress,
  // the "Alarm Timer" must be disabled.
  timerAlarmDisable(timer);
  delay(100);
  //----------------------------------------Wait for connection
//  Serial.println(_ssid);
//  Serial.println(_pass);
  // Time Out for the process of connecting to the WiFi network. 20 = 20 seconds.
  // If after 20 seconds, it fails to connect to the WiFi network, then ESP32 restarts.
  // I created this condition because there are times when ESP32 cannot connect to a WiFi
network and needs to be restarted in order to connect to a WiFi network.
  int time_out = 20;
  time_out = time_out * 2;
```

```
//  timerAlarmDisable(timer);
  timerAlarmEnable(timer);
  delay(50);
   dmd.selectFont(Arial_Black_16);
}
//_____________________________________________________________________________
//_____________________________________________________________________________
____VOID LOkOP()
void dispp();
//int dd=0;
bool play = 0;
bool resett = 0;
void timerr();
int amp=0;
String output26State = "off";
String output27State = "off";
// Assign output variables to GPIO pins
const int output26 = 26;
const int output27 = 27;
// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;
unsigned long previousMillis = 0;        // will store last time LED was updated
// constants won't change:
const long interval = 200;
int ledState = LOW;
void led() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;
    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }
    // set the LED with the ledState of the variable:
    digitalWrite(STATUS_LED, ledState);
  }
}
bool tick_tock = 0;             // ledState used to set the LED
unsigned long previousMillis1 = 0;        // will store last time LED was updated
```

```
// constants won't change:
const long interval1 = 500;
void OE_DMD_ROWS_OFF()                  {pinMode(PIN_DMD_nOE,OUTPUT);
digitalWrite( PIN_DMD_nOE, LOW  );  }
//#define OE_DMD_ROWS_OFF()             {}//      { ledcWrite(0,0); }
void OE_DMD_ROWS_ON()                   {pinMode(PIN_DMD_nOE,OUTPUT);digitalWrite(
PIN_DMD_nOE, HIGH ); }
 //int BRI;
//void OE_DMD_ROWS_ON2()                    {analogWrite( PIN_DMD_nOE, BRI );
Serial.print("brightness of led =");Serial.println(BRI);}w
void OE_DMD_ROWS_ON2()
{pinMode(PIN_DMD_nOE,OUTPUT);analogWrite( PIN_DMD_nOE, 20 );}
/**
 * @file
 *
 * EspNowUnicast.ino demonstrates how to transmit unicast ESP-NOW messages with @c
WifiEspNow .
 * You need two ESP8266 or ESP32 devices to run this example.
 *
 * Unicast communication requires the sender to specify the MAC address of the recipient.
 * Thus, you must modify this program for each device.
 *
 * The recommended workflow is:
 * @li 1. Flash the program onto device A.
 * @li 2. Run the program on device A, look at serial console for its MAC address.
 * @li 3. Copy the MAC address of device A, paste it in the @c PEER variable below.
 * @li 4. Flash the program that contains A's MAC address onto device B.
 * @li 5. Run the program on device A, look at serial console for its MAC address.
 * @li 6. Copy the MAC address of device B, paste it in the @c PEER variable below.
 * @li 7. Flash the program that contains B's MAC address onto device A.
 */
int aa=0;
   char smt1[100] = "IN A MEETING - PLEASE WAIT";
   char smt2[100] = "AVAILABLE BUT BUSY";
   char smt3[100] = "AVAILABLE - KNOCK FIRST";
   char smt4[100] = "IN CLASS - BACK SOON";
   char smt5[100] = "UNAVAILABLE";
   char smt[100];
void loop()
{
int  dat=EEPROM.read(0);
Serial.print(" data="); Serial.println(dat);
    if (dat==1){strcpy(smt,smt1);}
else if (dat==2){strcpy(smt,smt2);}
else if (dat==3){strcpy(smt,smt3);}
else if (dat==4){strcpy(smt,smt4);}
```

```
else {strcpy(smt,smt5);}
  stt:
 // goto stt;
    Serial.println(smt);
    dmd.drawMarquee(smt, strlen(smt), 64, 0);
    long start = millis();
    long timer = start;
    boolean ret = false;
    while (!ret) {
      if ((timer + 70) < millis()) {
        ret = dmd.stepMarquee(-1, 0);
        timer = millis();
        //  server.handleClient(); web();
      }
    }
/*
o  Button 1 (Red): "IN A MEETING - PLEASE WAIT"
o Button 2 (Orange): "AVAILABLE BUT BUSY"
o Button 3 (Green): "AVAILABLE - KNOCK FIRST"
o Button 4 (Blue): "IN CLASS - BACK SOON"
*/
//  char msg[100];
////  int len = snprintf(msg, sizeof(msg), "hello ESP-NOW from %s at %lu NUMBER:%u"
,WiFi.softAPmacAddress().c_str(), millis(),aa);
//  int len = snprintf(msg, sizeof(msg), "%lu :%u" ,millis(),aa);
//  WifiEspNow.send(PEER, reinterpret_cast<const uint8_t*>(msg), len);
//  Serial.println("mssg sent");
//  delay(10);
//  aa++;
  goto stt;
}
void printReceivedMessage(const uint8_t mac[WIFIESPNOW_ALEN], const uint8_t* buf,
size_t count,void* arg)
{
  Serial.printf("Message from %02X:%02X:%02X:%02X:%02X:%02X\n", mac[0], mac[1],
mac[2], mac[3],
               mac[4], mac[5]);
  for (int i = 0; i < static_cast<int>(count); ++i) {
    Serial.print(static_cast<char>(buf[i]));
  }
    timerAlarmDisable(timer);
   Serial.print("message data=");Serial.println(buf[0]);
       if(buf[0]=='1'){ EEPROM.write(0,1);EEPROM.commit();}
   else if(buf[0]=='2'){ EEPROM.write(0,2);EEPROM.commit();}
  else  if(buf[0]=='3'){ EEPROM.write(0,3);EEPROM.commit();}
  else  if(buf[0]=='4'){ EEPROM.write(0,4);EEPROM.commit();}
```

```
  else  if(buf[0]=='5'){ EEPROM.write(0,5);EEPROM.commit();}
timerAlarmEnable(timer);
delay(100);
Serial.print("eeprom data="); Serial.println(EEPROM.read(0));
ESP.restart();
  Serial.println();
}
```