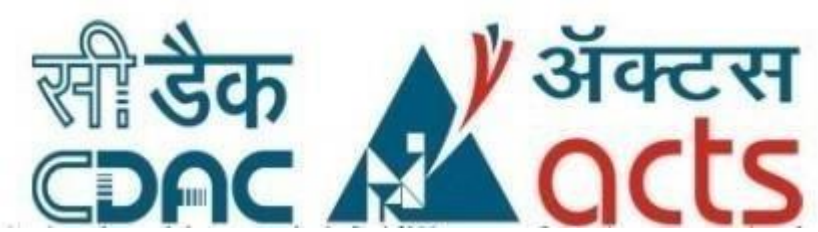
Project Report

On

Smart Real Time Battery Monitoring System



Submitted
In partial fulfilment
For the award of the Degree of

PG-Diploma in Embedded Systems and Design (PG-DESD)

C-DAC, ACTS (Pune)

Guided by:

Mr. Shripad Deshpande.

Presented by:

Mr. Akshay Uttam Navale	**PRN:-230940130007**
Mr. Ganesh Pandurang Karande	**PRN:-230940130034**
Mr. Saurav Mishra	**PRN:-230940130054**
Mr. Ashish Kannaujiya	**PRN:-230940130013**

Centre for Development of Advanced Computing(C-DAC), ACTS

(Pune- 411008)

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Mr. Akshay Uttam Navale	PRN:230940130007
Mr. Ganesh Pandurang Karande	PRN:230940130034
Mr. Saurav Mishra	PRN230940130054
Mr. Ashish Kannaujiya	PRN230940130013

Have successfully completed their project on

Smart Battery Monitoring System

Mr. Sripad Deshpande.

Project Guide **Project Supervisor**

Acknowledgement

This acknowledgment expresses our sincere gratitude to our guide, **Mr. Sripad Deshpande**, for his unwavering support and invaluable guidance in the development of our project, "***Smart Real Time Battery Monitoring System.***" His constant encouragement, insightful suggestions, and personal involvement have been instrumental in shaping the success of this project. Beyond technical expertise, his constructive criticism and shared enthusiasm have taught us patience and acted as a morale booster throughout the project. We extend our thanks to **Mr. Gaur Sunder** Head of the department, and all staff members for their cooperation and support in various aspects. Special appreciation is directed towards to **Ms. Namrata Ailawar** for his continuous guidance and assistance, which played a pivotal role in the successful completion of this work and further exploration of additional studies.

We also express our deepest appreciation to **Ms. Risha P R** and **Mrs. Srujana Bhamidi** Course Coordinator for PG-DESD, for his unwavering support and coordination, providing the necessary resources and encouragement for the project's realization. This collaborative effort and the support from our mentors and colleagues have significantly contributed to the development of the IOT-based solution for the Smart Battery Monitoring System, marking a successful culmination of our academic endeavors. Also, our warm thanks to **C-DAC ACTS Pune**, which provided us this opportunity to carry out, this prestigious Project and enhance our learning in various technical fields.

Mr. Akshay Uttam Navale	PRN:-230940130007
Mr. Ganesh Pandurang Karande	PRN:-230940130034
Mr. Saurav Mishra	PRN:-230940130054
Mr. Ashish Kannaujiya	PRN:-230940130013

Abstract

The Smart Real Time Battery Monitoring System using *FreeRTOS, Embedded OS and MQTT, CAN, and SPI Protocols* is a groundbreaking project aimed at enhancing battery management and safety in automotive vehicles. Inspired by the significance of Smart Battery monitoring system, this project leverages the STM32F407VG Discovery Board and ESP32 microcontrollers to implement the IOT Based smart Battery Monitoring System. The system utilizes a NTC @10K, ACS712 Curren sensor Module sensor to detect the Temperature and Current of the LI-Ion Battery. Through Controller Area Network (CAN) based communication, the project integrates the microcontrollers, allowing seamless data exchange for efficient Battery Monitoring.

The STM32F407VGT6 microcontroller acts as the central processing unit, orchestrating the sensor data processing Using SAR -ADC while the ESP32 WROOM 32D facilitates wi-fi communication. The system not only prevents Burning of the Li- Ion Battery, but also save the Human Life from this accidental problem. MQTT protocol is employed for efficient data transmission between the microcontrollers, enabling seamless communication. Additionally, the project extends its innovation by sending Battery status to the ThingsBoard cloud platform, ensuring accessibility and monitoring beyond the vehicle. This comprehensive approach ensures a robust and Smart Battery Monitoring that combines the precision of sensors, the power of microcontrollers, and the connectivity of IoT protocols.

The integration of MQTT, CAN, BLE and SPI protocols, along with the incorporation of STM32F407VGT6 and ESP32 WROOM 32D, TFT Display positions this project at the forefront of intelligent battery management systems in automotive applications.

Table of Contents

Chapter 1

Introduction

1.1 Introduction

The "Smart Real-Time Battery Monitoring System" embodies an ingenious fusion of state-of-the-art technologies, meticulously crafted to enhance the monitoring and control of batteries while adhering to stringent automotive and communication standards. This project seamlessly integrates the STM32F407VGT6 microcontroller and the ESP32 WROOM 32D module, ensuring compliance with industry-established norms and standards. The STM32F407VGT6, serving as the central processing unit, aligns with automotive-grade standards for reliability, performance, and compatibility with a diverse range of peripherals. This selection ensures that the system meets and surpasses rigorous automotive industry requirements.

Within the communication domain, the project utilizes the industry-standard Controller Area Network (CAN) protocol, following the ISO 11898 standard. This adherence to standardization ensures that the CAN communication within the system is robust, reliable, and interoperable, meeting the stringent criteria set by the automotive industry. The ESP32 WROOM 32D module, acclaimed for its dependable wireless connectivity, aligns with the Institute of Electrical and Electronics Engineers (IEEE) standards. Specifically, the Bluetooth functionality adheres to IEEE Std 802.15.1, while the Wi-Fi capabilities conform to IEEE Std 802.11, ensuring a standardized and secure communication framework. To establish a secure and reliable Bluetooth connection, the project adheres to the IEEE standards, incorporating secure pairing and encryption protocols. This ensures that the Bluetooth communication between the system and mobile devices is not only seamless but also meets industry-established security benchmarks. Furthermore, the inclusion of a TFT display and its integration into the system align with automotive industry standards, ensuring durability, readability, and optimal performance in diverse automotive environments. In conclusion, the "Smart Real-Time Battery Monitoring System" harmoniously

incorporates the STM32F407VGT6, ESP32 WROOM 32D, ISO 11898-compliant CAN protocol, and IEEE standards for Bluetooth (802.15.1) and Wi-Fi (802.11). This meticulous adherence to established standards positions the project as a robust and compliant solution, setting a new benchmark for intelligent and standardized battery management systems within the automotive sector.

1.2 Objective

The objectives of the project work are as Enhanced Battery Performance: Improve the overall performance of automotive batteries by continuously monitoring key parameters such as voltage, current, and temperature in real-time, leading to optimal usage and increased lifespan.

- **Prevention of Overcharging and Discharging**: Implement intelligent algorithms to detect and prevent overcharging or discharging, ensuring the battery operates within safe voltage limits, thereby reducing the risk of damage and enhancing safety.
- **Optimized Vehicle Efficiency**: Contribute to increased fuel efficiency and electric vehicle range by providing accurate insights into the battery's health and usage patterns, enabling efficient power management strategies.
- **Safety Assurance on Hills**: Incorporate the capability to monitor and control the vehicle on hills, preventing rollbacks and ensuring the safety of occupants and surrounding vehicles in hilly terrains.
- **Compliance with Automotive Standards**: Adhere to automotive industry standards for communication protocols (e.g., ISO 11898 for CAN), ensuring seamless integration with existing vehicle systems and compliance with industry norms.
- **User-Friendly Interface**: Provide a user-friendly interface through a TFT display, allowing drivers and technicians to easily access and interpret real-time battery parameters, fostering better user engagement and understanding.
- **Integration with IoT Ecosystem:** Enable seamless integration with the Internet of Things (IoT) ecosystem, facilitating remote monitoring and control. This connectivity enhances user convenience and allows for centralized fleet

management.

- **Data Analytics for Continuous Improvement:** Implement data analytics capabilities to analyze historical battery performance data. This enables manufacturers to identify patterns, improve design, and optimize future battery systems for enhanced reliability and efficiency.

1.3 Block Diagram:

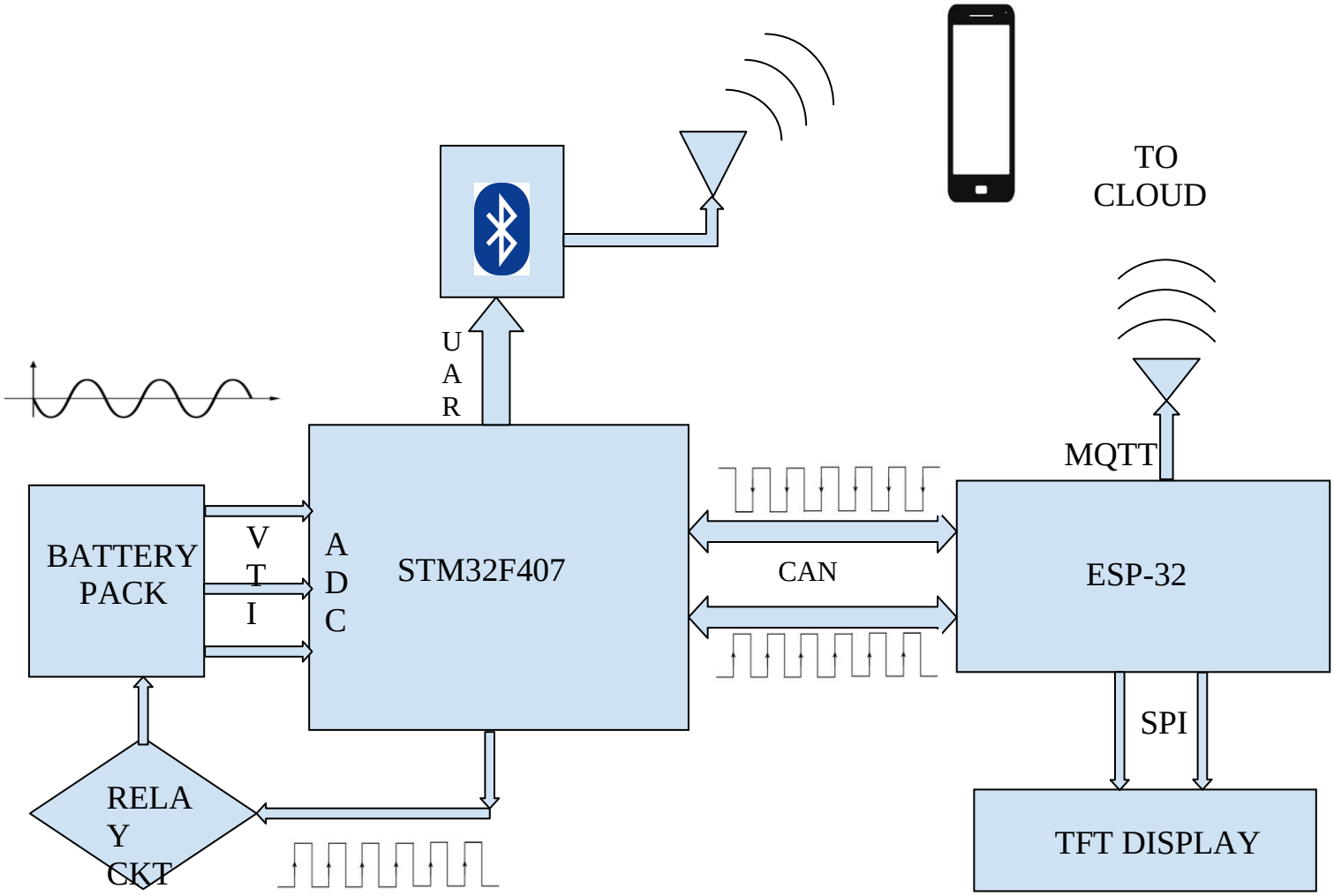
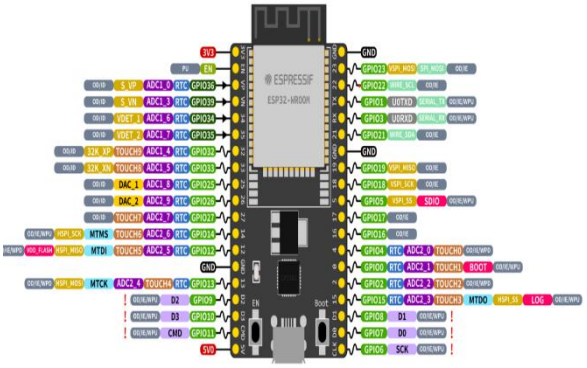


Fig (1): Smart Real Time Battery Monitoring System.

Block Diagram consist of Following Hardware modules:

1) STM32F407VGT6 Board 2) ESP32 WROOM32-D 3) HC-05 Bluetooth module 4) TFT 2.8 inch 240×320 pixels ILI9341 module 5) TJ1051 CAN Trans receiver module 6) 1 channel 5V Optocoupler Relay 7) LI-ion 12V 2A Battery 8) NTC-10K temperature sensor 9) ACS-712 30A Hall effect sensor. 10) 775 DC motor.



Chapter 2
LITERATURE REVIEW

The literature surrounding Smart Real-Time Battery Monitoring Systems reveals a growing interest in developing effective solutions for monitoring and managing batteries in various applications. Researchers have recognized the critical importance of real-time monitoring to ensure the optimal performance and safety of batteries. Previous works have explored different methodologies, technologies, and communication protocols to address the challenges associated with battery monitoring. In particular, the use of communication protocols such as Controller Area Network (CAN), Universal Asynchronous Receiver-Transmitter (UART), and Serial Peripheral Interface (SPI) has been widely discussed. These protocols enable efficient data transmission and real-time communication between battery monitoring systems and external devices. Additionally, the integration of the Message Queuing Telemetry Transport (MQTT) protocol has gained attention, providing a lightweight and scalable solution for connecting battery monitoring systems to the Internet of Things (IoT) or cloud platforms.

The STM32F407VGT6 microcontroller emerges as a key player in this domain, known for its advanced features and capabilities. Literature on the STM32F407VGT6 highlights its suitability for real-time systems, making it an ideal choice for battery monitoring applications. Researchers have explored the integration of this microcontroller with various communication protocols, showcasing successful implementations in diverse contexts. The literature also emphasizes the importance of selecting a microcontroller that meets the specific requirements of real-time battery monitoring, and the STM32F407VGT6 stands out for its performance and versatility.

As the literature review unfolds, a comparative analysis of different approaches and methodologies becomes apparent. Researchers have evaluated the strengths and weaknesses of various communication protocols and microcontrollers in the context of real-time battery monitoring. Identified gaps in the existing literature suggest

opportunities for further research and improvement. Overall, the literature underscores the need for comprehensive and efficient battery monitoring systems, laying the groundwork for the present project that leverages CAN, UART, SPI protocols, MQTT, and the STM32F407VGT6 microcontroller to contribute to this evolving field.

Chapter 3

Embedded System Design

3.1 Introduction: This chapter explains in detail about hardware Circuit Desing that are being used, and software design for Embedded their features and applications

3.1.1 STM32F407VGT6 Microcontroller:

The STM32F4 Discovery board is small devices based on STMF407 ARM microcontroller, which is a high-performance microcontroller. This board allows users to develop and design applications. It has multiple modules within itself which allows the user to communicate and design the interface of different kinds without relying on any third device. The board has all the modern system modules peripherals like DAC, ADC, audio port, UART, etc which makes it one of the best-developing devices. The device may be for developing modern applications but some protocols will need to be followed to use the device, like the compiler, voltage potential. The STM32F407xx family is based on the high-performance ARM® Cortex®-M4 32- bit RISC core with FPU operating at a frequency of up to 168 MHz, and embedding a floating-point unit (FPU), a memory protection unit (MPU) and an embedded trace macro cell (ETM). The family incorporates high-speed embedded memories (up to 1 Mbytes of Flash memory, up to 192 Kbytes of RAM) and an extensive range of enhanced I/O ’s and peripherals connected to two APB buses. They also feature standard and advanced communication interfaces: up to two I2Cs, up to three SPIs (two SPIs are with multiplexed full-duplex I2Ss), three USARTs, up to two UARTs, CAN and USB. To achieve audio class accuracy, the I2S peripherals can be clocked via an external PLL. The STM32F303xB/STM32F303xC family operates in the -40 to +85°C and -40 to +105 °C temperature ranges from a 2.0 to 3.3 V power supply. A comprehensive set of power-saving mode allows the design of low-power applications

3.1.2 ESP32 WROOM32-D Microcontroller:

The ESP32-WROOM-32D is a versatile wireless module based on the ESP32 microcontroller, renowned for its capabilities in IoT and wireless communication applications. Its dual-core Tensilica LX6 microprocessors, running at a clock frequency of 240 MHz, provide substantial processing power. With integrated Wi-Fi (802.11 b/g/n) and Bluetooth (Low Energy - BLE) capabilities, the module facilitates seamless wireless

connectivity. Memory features include 520 KB of SRAM, 448 KB of ROM, and support for external SPI flash memory up to 16 MB. The ESP32-WROOM-32D offers a rich set of peripheral interfaces, including GPIO pins, UART, I2C, CAN, SPI, I2S, PWM, ADC, DAC, and a hall effect sensor. Security features include secure boot, WPA/WPA2, and WPA3 protocols, ensuring secure data transmission. With power management options and support for low-power modes, the module is designed for energy-efficient operation. Operating at an input voltage of 2.2V to 3.6V and over a temperature range of -40°C to 125°C, the ESP32-WROOM-32D is suitable for various environments. Its small form factor, integrated antenna, and external antenna support make it adaptable for diverse applications. Certifications such as FCC and CE attest to regulatory compliance. Development support is provided through the Arduino IDE and Espressif IoT Development Framework (ESP-IDF), backed by a vibrant community offering assistance and resources. The ESP32-WROOM-32D excels in ultra-low power consumption, supports Over-the-Air updates, and operates in dual-mode, making it an excellent choice for a broad range of projects, from IoT devices to home automation and wearables.

3.1.3 Introduction to SAR ADC: In almost all STM32 microcontrollers, the ADC is implemented as a 12-bit Successive Approximation Register ADC1. Depending on the sales type and packaged used, it can have a variable number of multiplexed input channels (usually more than ten channels in the most of STM32 MCUs), allowing to measure signals from external sources. Moreover, some internal channels are also available: a channel for internal temperature sensor (VSENSE), one for internal reference voltage (VREF INT), one for monitoring external VBAT power supply and a channel for monitoring LCD voltage in those MCUs providing a native monochrome passive LCD controller (for example, the STM32L053 is one of these). ADCs implemented in STM32F3 and in majority of STM32L4 MCUs are also capable of converting fully differential inputs**.**

A/D conversion of the various channels can be performed in single, continuous, scan or discontinuous mode. The result of the ADC is stored in a left- or right-aligned 16-bit data register. Moreover, the ADC also implements the analog watchdog feature, which allows the application to detect if the input voltage goes outside the user-defined higher or lower thresholds: if this happens, a dedicated IRQ fires.

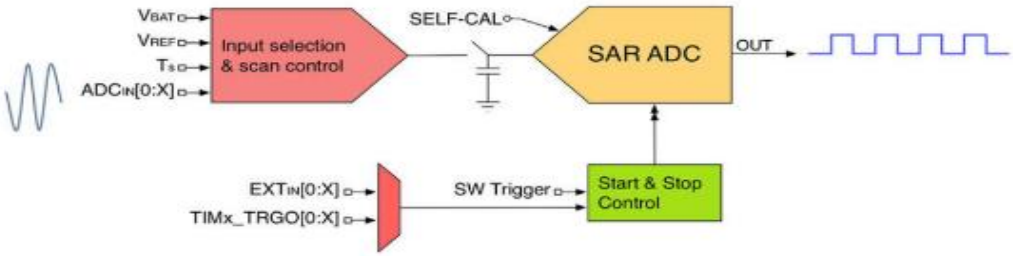


Fig (2): Simplified structure of SAR ADC.

The Successive Approximation algorithm computes the voltage of the input signal by comparing it with the one generated by the internal DAC, which is a fraction of the VREF voltage: if the input signal is higher than this internal reference voltage, then this is further increased until the input signal is lower. The final result will correspond to a number ranging from zero to the maximum 12-bit unsigned integer, that is $Resolution = 2\text{^}12 - 1 = 4095$. Assuming VREF = 3300mV, we have that 3300mV is represented with 4095. This means that Resolution=3.3\4095 ≈ 0.8mV.

Scan Continuous Conversion Mode:

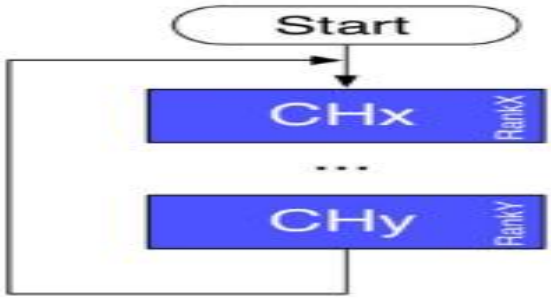


Fig (3): Independent Scan continuous conversion mode.

This mode is also called multichannel continuous mode and it can be used to convert some channels successively with the ADC in independent mode. Using ranks, you can configure any sequence of up to 16 channels successively with different sampling times and different orders. This mode is similar to the multichannel single conversion mode except that it does not stop converting after the last channel of the sequence but it restarts the conversion sequence from the first channel and continues indefinitely. Scan conversions are carried out in DMA mode.

Channel Selection:

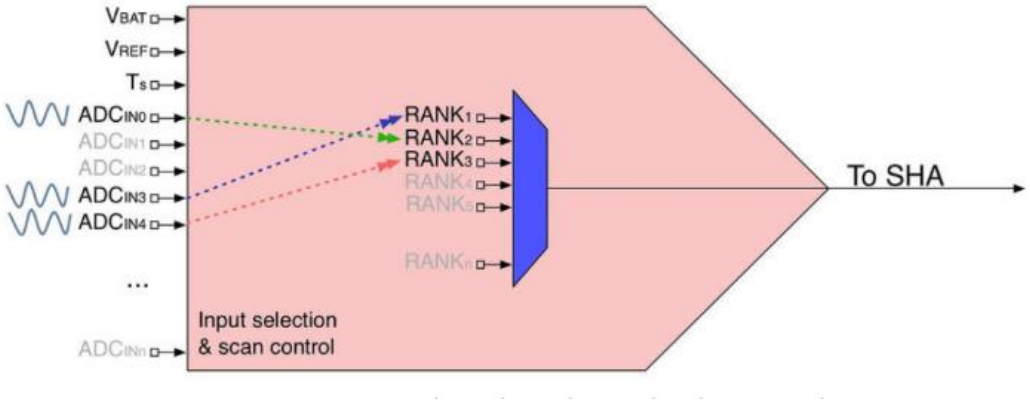


Fig (4): How input channels can be reordered using ranks.

STM32 MCUs, instead, offer the notion of group. A group consists of a sequence of conversions that can be done on any channel and in any order. While input channels are fixed and bound to specific MCU pins (that is, IN0 is the first channel, IN1 the second and so on), they can be logically reordered to form custom sampling sequences. The reordering of channels is performed by assigning to them an index ranging from 1 to 16. This index is called rank in the Cube HAL. The Figure 4 shows this concept. Although the IN4 channel is fixed (for example, it is connected to PA4 pin in an STM32F401RE MCU), it can be logically assigned to the rank 1 so that it will be the first channel to be sampled. Those MCUs offering this possibility also allow to select the sampling speed of each channel individually, differently from F0/L0 MCUs where the configuration is ADC-wide. The channel/rank configuration is performed by using an instance of the C struct ADC_Channel-ConfTypeDef, which is defined in the following way:

```
typedef struct {
uint32_t Channel; /* Specifies the channel to configure into ADC rank */
uint32_t Rank; /* Specifies the rank ID */
uint32_t SamplingTime; /* Sampling time value for the selected channel */
uint32_t Offset; /* Reserved for future use, can be set to 0 */
} ADC_ChannelConfTypeDef;
```

ADC Resolution and Conversion Speed: It is possible to perform faster conversions by reducing the ADC resolution8. The sampling time, in fact, is defined by a fixed number of cycles (usually 3) plus a variable number of cycles depending the A/D resolution. The minimum conversion time for each resolution is then as follows:

• 12 bits: 3 + ∼12 = 15 ADCCLK cycles

• 10 bits: 3 + ∼10 = 13 ADCCLK cycles

• 8 bits: 3 + ∼8 = 11 ADCCLK cycles

• 6 bits: 3 + ∼6 = 9 ADCCLK cycles

By reducing the resolution is so possible to increase the number of maximum samples per seconds, reaching even more then 15Msps in some STM32 MCUs. Remember that the ADCCLK is derived from the peripheral clock: this means that SYSCLK and PCLK speeds impact on the maximum number of samples per second.

A/D Conversions in Polling Mode:

Like the majority of STM32 peripherals, the ADC can be driven in three modes: polling, interrupt and DMA mode. As we will see later, a timer can eventually drive this last mode so that A/D conversions take place at regular interval. This is extremely useful when we need to sample signals at a given frequency, like in audio applications. Once the ADC controller is configured by using an instance of the ADC_InitTypeDef struct passed to the ***HAL_ADC_Init()*** routine, we can start the peripheral using the ***HAL_ADC_Start()*** function.

Depending on the conversion mode chosen, ADC will convert each selected input continuously or once: in this case, to convert again selected inputs we need to call the HAL_ADC_Stop() function before calling again the HAL_ADC_Start() one. In polling mode we use the function

HAL_StatusTypeDef ***HAL_ADC_PollForConversion(ADC_HandleTypeDef* hadc, uint32_t Timeout);*** to determine when the A/D conversion is complete and the result is available inside the ADC data register. The function accepts the pointer to the ADC handler descriptor and a Timeout value, which represents the maximum time expressed in milliseconds we are willing to wait. Alternatively, we can pass the HAL_MAX_DELAY to wait indefinitely.

3.2.0 Circuit Diagram:

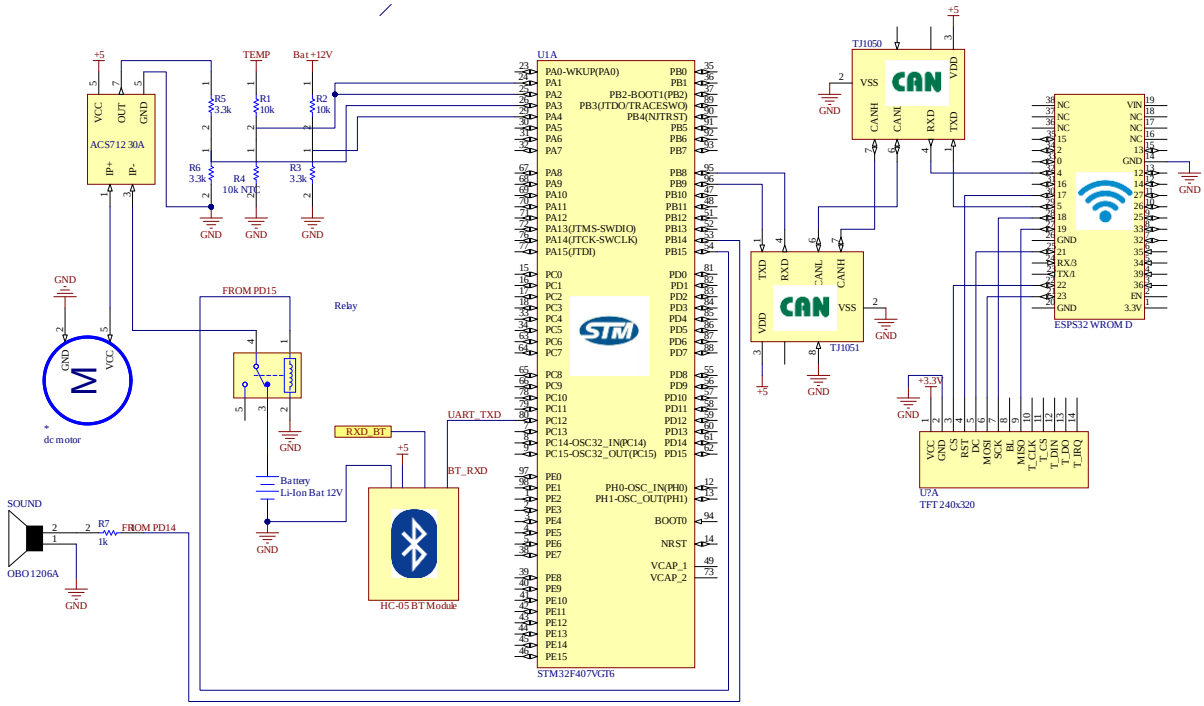


Fig (5): Smart Real Time Battery Monitoring System circuit.

In the Smart Real-Time Battery Monitoring System circuit, the design incorporates components for signal conditioning, analog-to-digital conversion, and relay-based temperature monitoring using the STM32F407VGT6 microcontroller. Here's a summary of the key elements:

Voltage Divider for Signal Conditioning: To adapt the voltage levels of the battery parameters to the operating range of the STM32F407VGT6, which operates at 3.3V.

Details: For the 5V signal, a voltage divider is employed to convert it to 2.5V, ensuring compatibility with the STM32F407VGT6. Another voltage divider is used for the 12V signal, reducing it to 3.3V, aligning with the microcontroller's voltage requirements.

Analog-to-Digital Conversion (ADC) using STM32F407VGT6: To digitize the analog signals obtained from the voltage dividers for further processing and monitoring by the microcontroller.

Details: The STM32F407VGT6 integrates a built-in ADC, which converts the conditioned analog signals into digital values. These digital values can be processed by the microcontroller to obtain accurate readings of battery parameters such as voltage and current.

5V Optocoupler Relay for Temperature Monitoring.

Purpose: To monitor the battery temperature and trigger a response if it exceeds a specified threshold.

Details: An optocoupler relay driven by a 5V signal is used to interface with the STM32F407VGT6. The relay is configured to activate when the battery temperature surpasses a predefined level. Upon activation, the relay can trigger actions such as sending alerts or activating cooling mechanisms. Overall, the circuit ensures compatibility between the analog signals from the batteries and the STM32F407VGT6 microcontroller. The voltage dividers play a crucial role in scaling down the battery voltages to levels suitable for the microcontroller. The ADC of the STM32F407VGT6 facilitates the conversion of these analog signals into digital form for efficient processing. Additionally, the integration of an optocoupler relay enhances the system's functionality by providing a mechanism to monitor and respond to battery temperature fluctuations. This comprehensive approach ensures accurate and real-time monitoring of critical battery parameters in the Smart Real-Time Battery Monitoring System.

ESP32 for Communication: Facilitates communication between the STM32F407VGT6 microcontroller and other devices using the Controller Area Network (CAN) protocol.

Details: The ESP32 serves as a bridge between the STM32F407VGT6 and external devices, allowing seamless data exchange over the CAN bus. CAN communication enables reliable and real-time transmission of battery parameters.

CAN Transceiver Module (TJ1051: Purpose: Interfaces with CANH and CANL lines, ensuring proper communication on the CAN bus.

Details: The TJ1051 CAN transceiver module provides the necessary hardware support for CAN communication. It ensures signal conditioning and proper voltage levels for CAN communication between the STM32F407VGT6 and other CAN-enabled devices.

TFT Display: Provides a visual interface for users to monitor battery parameters such as temperature, voltage, and current.

Details: The TFT display is driven by the ESP32 microcontroller and presents real-time data to the user. Users can conveniently observe and assess the battery status through a graphical user interface on the display.

HC-05 Bluetooth Module: Enables wireless communication of battery parameters to a user's device using Bluetooth technology. The HC-05 Bluetooth module allows the STM32F407VGT6 to transmit battery data to user devices, enhancing accessibility.

Users can receive and monitor battery parameters on their smartphones or other Bluetooth-

enabled devices.

Overall, the extended circuit enhances the functionality of the Smart Real-Time Battery Monitoring System by introducing capabilities for CAN communication, visual representation through a TFT display, and wireless data transmission via wi-Fi and Bluetooth. This comprehensive approach ensures that users have multiple avenues for accessing and monitoring critical battery parameters, making the system more versatile and user-friendly.

3.3.0 Communication Protocols:

In the realm of embedded systems, communication plays a pivotal role in facilitating seamless interaction between components. Several protocols are commonly employed to enable efficient data exchange and control. Controller Area Network (CAN) is a robust and widely adopted serial communication protocol known for its reliability in real-time applications, particularly in the automotive and industrial sectors. Universal Asynchronous Receiver-Transmitter (UART) is a simple and widely used asynchronous protocol suitable for point-to-point communication between devices. Serial Peripheral Interface (SPI) is another synchronous protocol that excels in high-speed communication, often used to connect microcontrollers to peripheral devices. Message Queuing Telemetry Transport (MQTT) is a lightweight and scalable protocol tailored for the Internet of Things (IoT) applications, enabling efficient publish-subscribe messaging between devices. Each protocol serves specific communication needs, and their judicious selection is critical in designing effective and efficient embedded systems for a variety of applications.

3.3.1: Controller Area Network Protocol: CAN stands for *Controller Area Network* protocol. It is a protocol that was developed by **Robert Bosch** in around 1986. The CAN protocol is a standard designed to allow the microcontroller and other devices to communicate with each other without any host computer. The protocol is well-suited for applications requiring high-speed data exchange, such as engine control, vehicle diagnostics, and various control systems within modern automobiles.

ISO Standards and Specifications: CAN protocol is standardized by the International Organization for Standardization (ISO). The two main ISO standards related to CAN are ISO 11898-1 and ISO 11898-2. ISO 11898-1 specifies the data link layer and physical signaling, while ISO 11898-2 defines the high-speed CAN physical layer.

Speed Modes and Communication Range: CAN supports multiple speed modes, and the choice of speed depends on the specific application requirements. The two primary speed

modes are:

High-Speed CAN (HS-CAN): Speed Range: Typically operates at speeds up to 1 Mbps. Communication Range: Suited for short to medium-range communication within a vehicle.

Uses: High-Speed CAN is commonly employed for critical real-time applications such as engine control, transmission control, and chassis systems.

Low-Speed CAN (LS-CAN): Speed Range: Operates at speeds up to 125 kbps.

Communication Range: Suited for longer communication distances within a vehicle.

Uses: Low-Speed CAN is often used for non-critical functions like interior lighting, seat controls, and infotainment systems.

Uses in the Automotive Industry: CAN has become the backbone of automotive communication systems due to its key features:

Reliability: CAN's differential signaling and error-checking mechanisms enhance communication reliability in noisy environments.

Efficiency: The protocol efficiently handles multiple electronic control units (ECUs) communicating on the same network. Real-time Capabilities: CAN's deterministic nature is crucial for real-time applications in vehicles, ensuring timely data

transmission

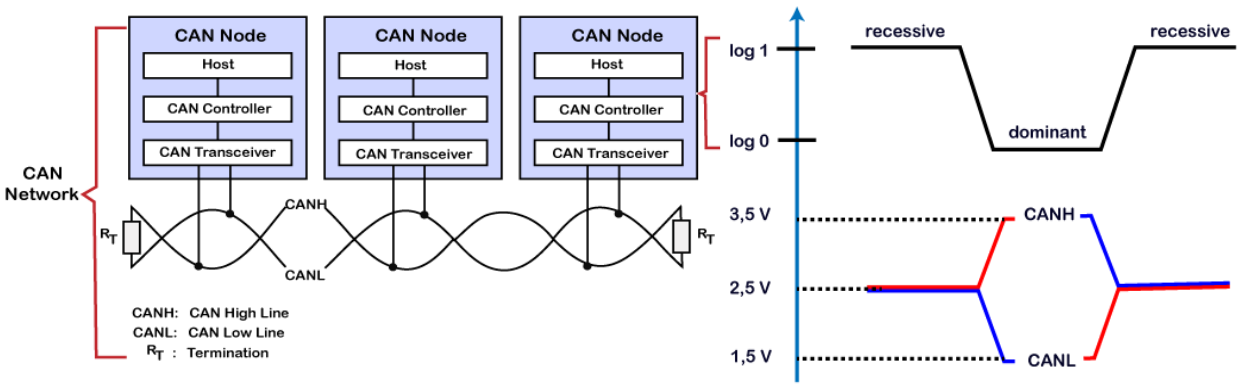


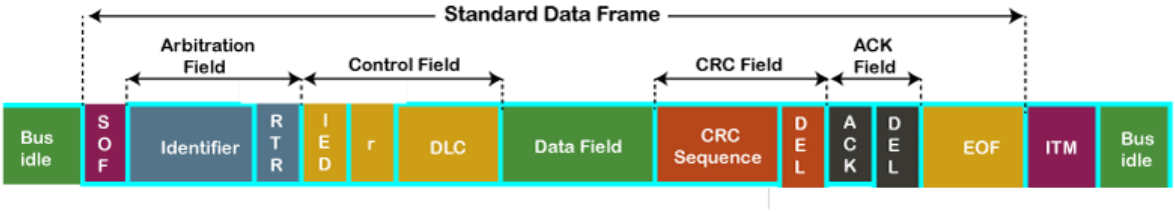
Fig (6): CAN BUS Network & Characteristics.

Key points learnt from the CAN characteristics

- Logic 1 is a recessive state. To transmit 1 on CAN bus, both CAN high and CAN low should be applied with 2.5V.

- Logic 0 is a dominant state. To transmit 0 on CAN bus, CAN high should be applied at 3.5V and CAN low should be applied at 1.5V.
- The ideal state of the bus is recessive.
- If the node reaches the dominant state, it cannot move back to the recessive state by any other node.

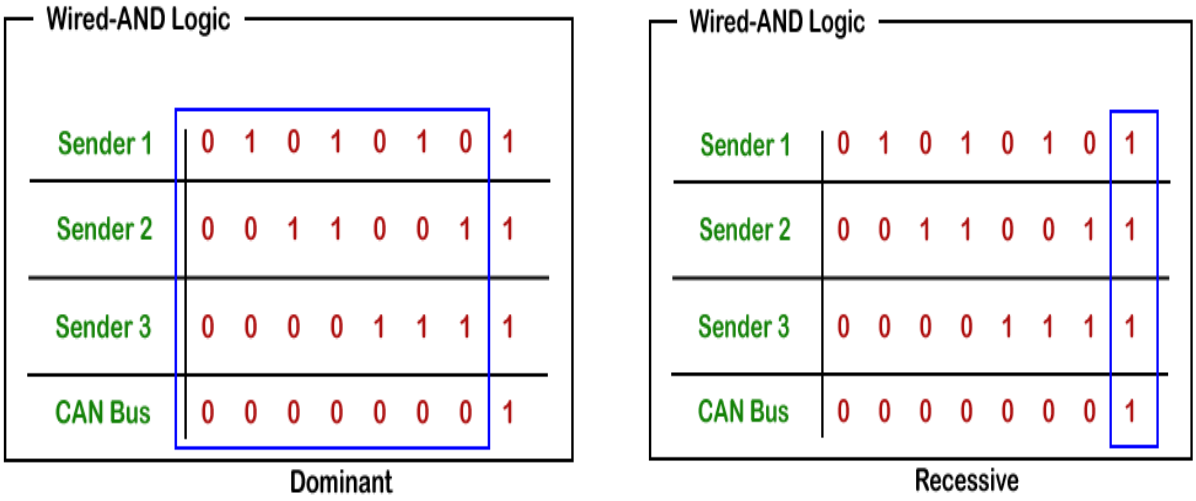
CAN Framing:



- **SOF:** SOF stands for the start of frame, which indicates that the new frame is entered in a network. It is of 1 bit.
- **Identifier:** A standard data format defined under the CAN 2.0 A specification uses an 11-bit message identifier for arbitration. Basically, this message identifier sets the priority of the data frame.
- **RTR:** RTR stands for Remote Transmission Request, which defines the frame type, whether it is a data frame or a remote frame. It is of 1-bit.
- **Control field:** It has user-defined functions.
    1. **IDE:** An IDE bit in a control field stands for identifier extension. A dominant IDE bit defines the 11-bit standard identifier, whereas recessive IDE bit defines the 29-bit extended identifier.
    2. **DLC:** DLC stands for Data Length Code, which defines the data length in a data field. It is of 4 bits.
    3. **Data field:** The data field can contain upto 8 bytes.
- **CRC field:** The data frame also contains a cyclic redundancy check field of 15 bit, which is used to detect the corruption if it occurs during the transmission time. The sender will compute the CRC before sending the data frame, and the receiver also computes the CRC and then compares the computed CRC with the CRC received from the sender. If the CRC does not match, then the receiver will generate the error.

- **ACK field:** This is the receiver's acknowledgment. In other protocols, a separate packet for an acknowledgment is sent after receiving all the packets, but in case of CAN protocol, no separate packet is sent for an acknowledgment.
- **EOF:** EOF stands for end of frame. It contains 7 consecutive recessive bits known End of frame.

CAN bus logic: From the above scenario, we get to know that the dominant state overwrites the recessive state. When the node sends the dominant and the recessive bit simultaneously, then the bus remains dominant. The recessive level occurs only when all the nodes send the recessive bit. Such logic is known as AND logic, and physically it is implemented as an open collector circuit.



CAN Communication Principle: As we know that the message is sent based on the priority set in the arbitration field. For the standard frame, the message identifier is 11 bit, while for the extended frame, the message identifier is 29 bit. It allows the system designer to design the message identifier at the design itself. The smaller the message identifier, the higher, would be the message priority.

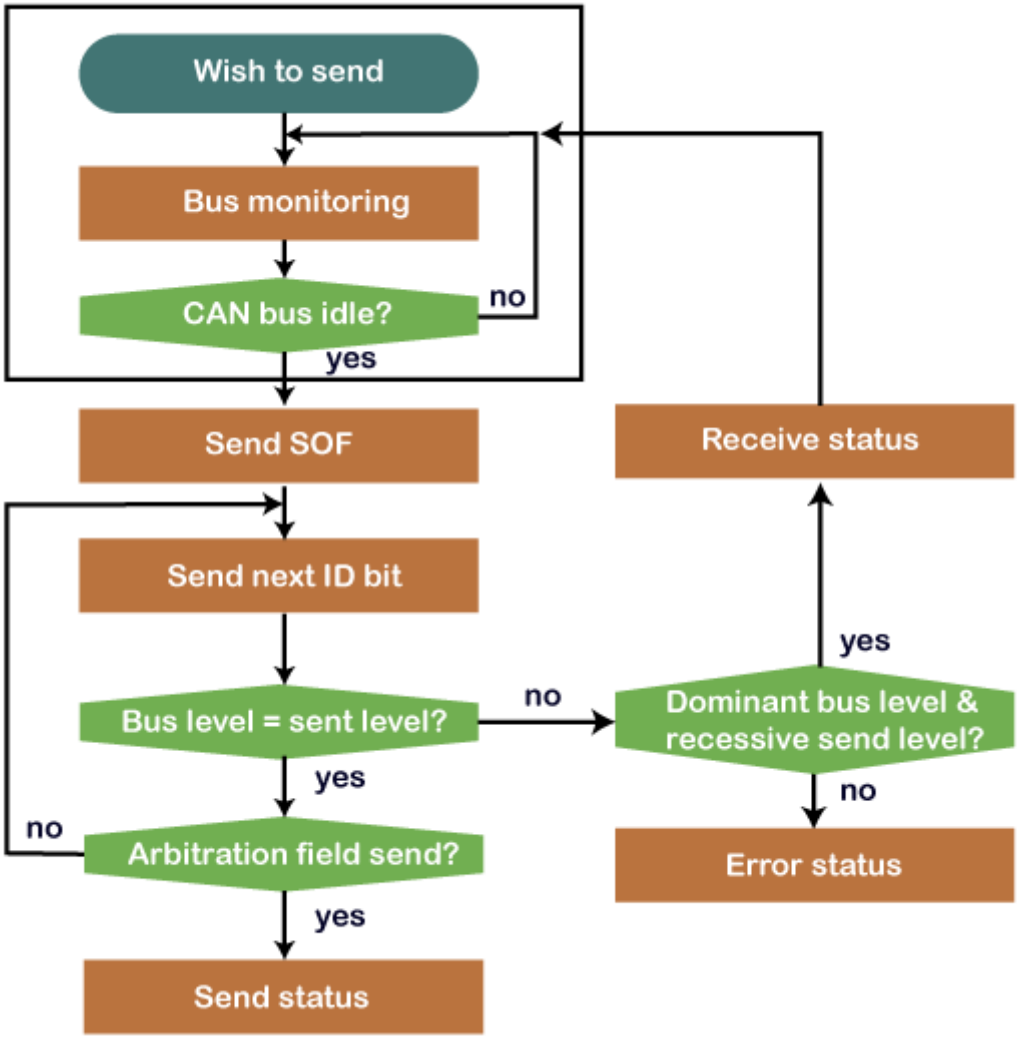
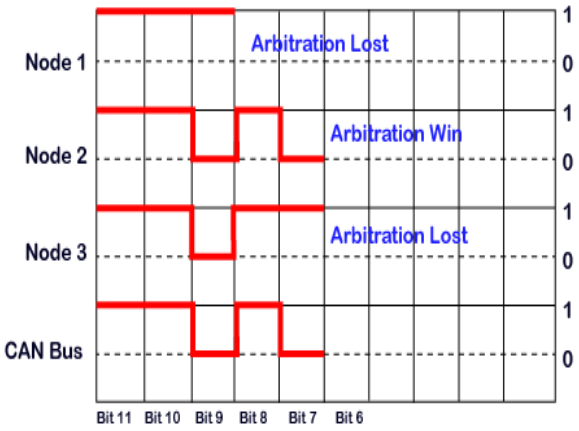


Fig (7): Working Principle of CAN.

The sender wants to send the message and waiting for the CAN bus to become idle. If the CAN bus is idle, then the sender sends the SOF or the dominant bit for the bus access. Then, it sends the message identifier bit in the most significant bit. If the node detects the dominant bit on the bus while it has transmitted the recessive bit, it means that the node has lost the arbitration and stops transmitting further bits. The sender will wait and resend the message once the bus is free.

CAN Arbitration Example

CAN Node	Identifier (Hex)	Identifier (Binary)
1	0x7F3	11111110011
2	0x6B3	11010110011
3	0x6D9	11011011001



If we consider three nodes, i.e., **Node 1, Node 2, and Node 3**, the message identifiers of these nodes are **0x7F3**, **0x6B3**, and **0x6D9**, respectively. The transmission of all the three nodes with the most significant bit is shown in the above diagram **11th** bit: As all the three bits of nodes are recessive, so bus bit will also remain recessive. **10th** bit: All the nodes have 10th bit as recessive, so the bus will also remain recessive. **9th** bit: Node 1 has recessive bit while other nodes have a dominant bit, so the bus will also remain dominant. In this case, node 1 has lost the arbitration, so it stops sending bits. **8th** bit: Both **node 2** and **node 3** are sending recessive bit, so that the bus state will remain recessive. **7th** bit: The **node 2** is sending dominant bit while **node 3** has sent recessive bit, so that the bus state will remain dominant. In this case, **the node 3** has lost the arbitration, so it stops sending the message while the **node 2** has won the arbitration means that it will continue to hold the bus until the message is received.

3.3.2: Universal Asynchronous Receiver-Transmitter Protocol (UART):

UART is a popular asynchronous serial communication protocol used for transmitting and receiving data between devices. It facilitates point-to-point communication and is widely utilized in embedded systems, microcontrollers, and various electronic devices. The term "asynchronous" refers to the fact that the data is sent without a shared clock signal between the communicating devices.

Key Features of UART:

1. **Two-Wire Communication:** UART uses two wires: Transmit (Tx) and Receive (Rx), for bidirectional communication between devices.
2. **Start and Stop Bits:** Each data byte in UART communication is framed by start and stop bits. The start bit signals the beginning of a data byte, and the stop bit(s) indicate the end.
3. **Baud Rate:** Baud rate determines the speed of communication and is expressed in bits per second (bps). Devices communicating via UART must be configured with the same baud rate for successful data transfer.
4. **No Master-Slave Restriction:** Unlike some other communication protocols, UART does not inherently have a master or slave device. Any device with UART capability can communicate with any other device.
5. **Synchronous vs. Asynchronous:** While UART is typically asynchronous, meaning that data is transmitted without a shared clock, there are synchronous variants like Synchronous UART (SUART) that use a shared clock signal for synchronization.
6. **Versatility:** UART is versatile and commonly used in various applications, including data communication between microcontrollers, interfacing with sensors, connecting peripherals like GPS modules and Bluetooth modules, and debugging through serial consoles.
7. **Communication Process:** Transmission: The sender (transmitter) initiates communication by sending a start bit, followed by the data bits, an optional parity bit, and one or more stop bits. Reception: The receiver samples the incoming signal at the middle of each bit time to decode the transmitted data.
8. **Baud Rate Configuration:** Both the transmitting and receiving devices must be configured with the same baud rate to ensure proper synchronization.

Applications:

1) **Microcontroller Communication**: UART is extensively used for communication between microcontrollers and other devices within embedded systems.
2) **Wireless Communication**: UART is employed for connecting wireless modules, such as Bluetooth and Wi-Fi modules, to microcontrollers.

3) **Serial Consoles:** UART is commonly used for debugging and interacting with devices through serial consoles.

UART is a fundamental and widely adopted communication protocol in the realm of embedded systems, offering simplicity, versatility, and reliability in data transmission between devices.

3.3.3: SPI (Serial Peripheral Interface) Protocol: SPI stands for the **Serial Peripheral Interface**. It is a serial communication protocol that is used to connect low-speed devices. It was developed by **Motorola** in the **mid-1980** for inter-chip communication. SPI, or Serial Peripheral Interface, is a synchronous serial communication protocol commonly used for short-distance communication between microcontrollers and peripheral devices. It is designed for high-speed, full-duplex communication and is widely employed in embedded systems for connecting microcontrollers with sensors, displays, memory devices, and other peripherals.

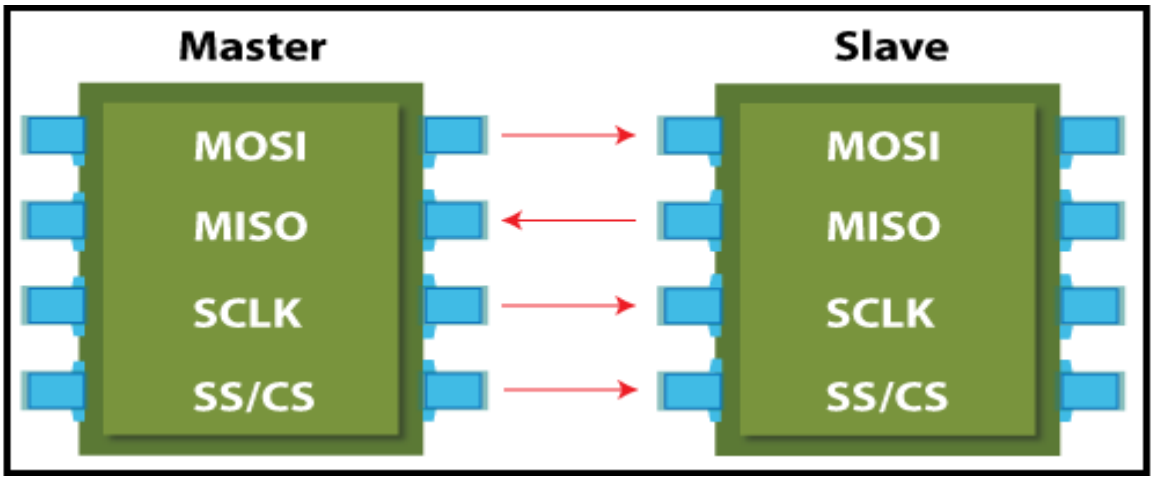


Fig (8): SPI Interface.

Key Features of SPI:

1. **Four-Wire Communication:** SPI uses four wires: MOSI (Master Out Slave In), MISO (Master in Slave Out), SCLK (Serial Clock), and SS/CS (Slave Select/Chip Select). These wires enable full-duplex communication between the master and slave devices.

2. **Master-Slave Configuration:** In SPI communication, there is typically one master device that initiates communication and one or more slave devices that respond to the master's commands.
3. **Synchronous Communication:** SPI is synchronous, meaning that data transmission is synchronized with a clock signal (SCLK). This allows for high-speed communication.
4. **Full-Duplex Operation:** SPI supports simultaneous data transmission and reception, allowing for efficient bidirectional communication between the master and slave devices.
5. **Variable Data Frame Size:** SPI allows for flexibility in the size of data frames, enabling devices to transmit and receive different amounts of data in each communication cycle.
6. **Configurable Clock Polarity and Phase:** SPI supports different clock polarities (CPOL) and phases (CPHA), providing flexibility in adapting to the requirements of various peripherals.
7. **Multiple Slave Devices:** The master device can communicate with multiple slave devices by using a separate SS/CS line for each slave.

Communication Process:

1. **Master Initialization:** The master device initiates communication by asserting the SS/CS line corresponding to the intended slave device.
2. **Data Transmission:** The master sends data (MOSI) to the slave while simultaneously receiving data (MISO) from the slave. The communication is synchronized with the clock signal (SCLK).
3. **Clock Configuration:** The master and slave devices must be configured with the same clock polarity and phase to ensure proper synchronization.
4. **Slave Deactivation:** Once the communication is complete, the master deactivates the SS/CS line, signalling the end of the transaction.

Applications:

1. **Sensor Interfacing:** SPI is commonly used for connecting sensors to microcontrollers in applications such as temperature sensing, accelerometers, and gyroscopes.

2. **Memory Devices:** SPI is employed for interfacing with memory devices like EEPROMs and Flash memory.
3. **Display Interfaces:** SPI is utilized for connecting microcontrollers with graphical displays and LCDs.
4. **Communication between Microcontrollers:** SPI facilitates communication between multiple microcontrollers in complex embedded systems.

SPI is a widely adopted communication protocol offering high-speed, full-duplex communication with a straightforward master-slave architecture. Its versatility and efficiency make it a preferred choice for various embedded applications.

3.4.0 Wireless Communication:

3.4.1 Wi-Fi Communication: The family of standards defining Wi-Fi is developed and maintained by the Institute of Electrical and Electronics Engineers (IEEE). The primary IEEE standard for Wi-Fi is known as the IEEE 802.11 standard. Various amendments and extensions to this standard have been introduced over the years, each labeled with a letter (e.g., 802.11a, 802.11b, etc.). Here are some key Wi-Fi standards within the 802.11 family. Wi-Fi, short for Wireless Fidelity, is a widely adopted wireless communication technology that enables devices to exchange data over a local area network (LAN) without the need for physical cables. It has become a ubiquitous technology, providing wireless internet connectivity for various devices, including smartphones, laptops, IoT devices, and more.

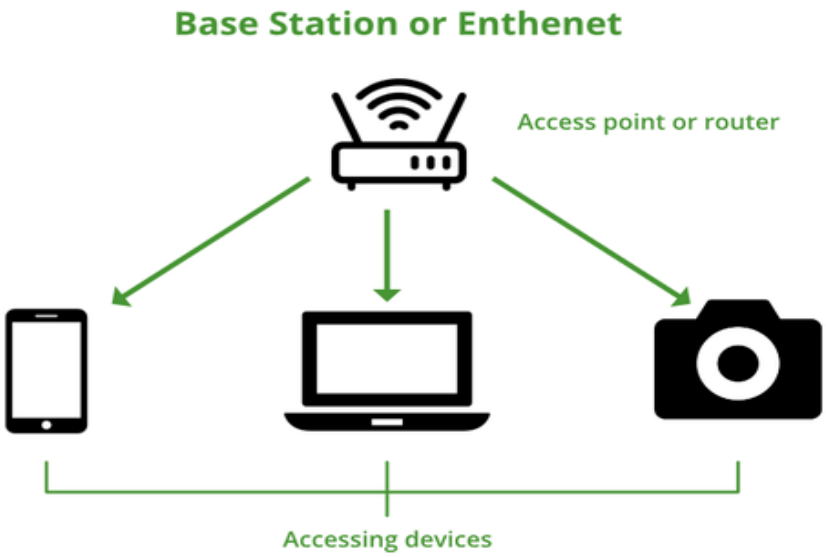


Fig (9): Wi-Fi Interface.

Key Features of Wi-Fi:

1. **Wireless Connectivity:** Wi-Fi enables devices to connect to a network wirelessly, allowing for seamless data transmission and internet access.
2. **Frequency Bands:** Wi-Fi operates in different frequency bands, including 2.4 GHz and 5 GHz, providing flexibility and reducing interference.
3. **Standards:** Wi-Fi standards, such as IEEE 802.11a/b/g/n/ac/ax, define the rules for wireless communication, ensuring interoperability among devices from different manufacturers.
4. **Security Protocols:** Wi-Fi supports various security protocols, including WEP, WPA, and WPA2/WPA3, to secure data transmissions and protect networks from unauthorized access.
5. **Range and Throughput:** Wi-Fi offers varying ranges and data transfer rates depending on the standard and environmental factors. Higher standards, such as Wi-Fi 6 (802.11ax), provide increased throughput and efficiency.
6. **Infrastructure and Ad-Hoc Modes:** Wi-Fi can operate in infrastructure mode, connecting devices through a central access point, or in ad-hoc mode, allowing devices to communicate directly with each other without a central point.

SSID and Password Protection. Wi-Fi networks are identified by Service Set Identifiers (SSIDs), and access is often protected by passwords, ensuring secure and controlled access.

Communication Process:

1. **Network Discovery:** Devices search for available Wi-Fi networks, and users select the desired network from the list of available options.
2. **Authentication and Association:** Users provide authentication credentials (passwords) to connect to a secured Wi-Fi network. Once authenticated, the device associates with the network.
3. **Data Transmission:** Devices connected to the Wi-Fi network can exchange data, access the internet, or communicate with other devices within the same network.
4. **Security Measures:** Wi-Fi networks use encryption protocols to secure data during transmission, preventing unauthorized access and eavesdropping.

Applications:

1. **Internet Connectivity**: Wi-Fi provides wireless internet access for devices such as smartphones, laptops, and tablets.
2. **Home and Business Networking:** Wi-Fi is extensively used for creating wireless networks within homes and businesses, connecting various devices to share resources and data.
3. **IoT Connectivity:** Wi-Fi is a common choice for connecting IoT devices, enabling them to communicate with each other and with cloud services.
4. **Public Hotspots:** Public places like cafes, airports, and hotels often provide Wi-Fi hotspots for visitors to access the internet.

Wi-Fi is a versatile and widely adopted wireless communication technology that has revolutionized how devices connect, communicate, and access the internet, contributing to the proliferation of wireless networking in both personal and professional settings.

Bluetooth Communication: Bluetooth is a wireless communication standard designed for short-range data exchange between devices. Developed by Ericsson in the late 1990s, Bluetooth has evolved into a versatile and widely adopted technology, enabling seamless connectivity among various electronic devices.

Key Features of Bluetooth:

1. **Wireless Connectivity:** Bluetooth facilitates wireless communication between devices within a short range, typically up to 10 meters, without the need for physical cables.
2. **Frequency Band:** Bluetooth operates in the unlicensed 2.4 GHz ISM (Industrial, Scientific, and Medical) frequency band, avoiding interference with other wireless technologies.
3. **Versions and Profiles:** Bluetooth has undergone multiple versions, each introducing improvements in speed, range, and power efficiency. Common profiles, such as Hands-Free Profile (HFP) and Advanced Audio Distribution Profile (A2DP), define specific use cases and functionalities.
4. **Low Energy (LE) Technology:** Bluetooth Low Energy (BLE or Bluetooth Smart) is an energy-efficient variant designed for power-sensitive devices, enabling longer battery life and supporting applications in healthcare, fitness, and IoT.
5. **Pairing and Security:** Bluetooth devices establish a secure connection through a process called pairing, typically requiring user authentication. Security features

include encryption to protect data during transmission.

6. **Point-to-Point and Broadcast Communication:** Bluetooth supports point-to-point communication between two devices and broadcast communication, allowing one device to communicate with multiple devices simultaneously.

Communication Process:

1. **Device Discovery:** Devices use a process called inquiry to discover nearby Bluetooth devices. Once discovered, devices can establish connections.
2. **Pairing:** Pairing involves establishing a secure connection between two devices, often requiring user input for authentication. This process ensures that only authorized devices can communicate.
3. **Connection Establishment:** After pairing, devices establish a connection, enabling the exchange of data. Devices can function in master or slave roles, allowing one device to initiate communication (master) while the other responds (slave).
4. **Data Exchange:** Devices exchange data over the established connection. The type of data exchanged depends on the Bluetooth profile used, ranging from audio streaming to file transfer and device control.

Applications:

1. **Peripheral Connectivity**: Bluetooth is employed for connecting peripherals such as keyboards, mice, and printers to computers and mobile devices.
2. **IoT and Wearables**: Bluetooth Low Energy is commonly used in IoT devices and wearables for energy-efficient communication.
3. **Wireless File Transfer**: Bluetooth facilitates the wireless transfer of files between devices, such as smartphones and tablets.

Bluetooth is a versatile wireless communication standard that enables the seamless exchange of data between devices in various applications, fostering connectivity and convenience in the digital ecosystem.

3.5.0 RTOS (Real Time Operating System):

FreeRTOS, an open-source real-time operating system, offers a robust set of features tailored for embedded systems. Its task management capabilities allow developers to organize code into concurrent threads, and a preemptive, priority-based scheduler ensures deterministic task execution in real-time applications. Semaphore and Mutex mechanisms enable resource sharing and synchronization, while queues and message passing facilitate inter-task communication. With software timers, interrupt management, and configurable memory allocation, FreeRTOS is adaptable to diverse embedded platforms, maintaining portability and efficiency. Its tickless operation minimizes power consumption during idle periods, making it suitable for power-sensitive applications. Renowned for reliability, FreeRTOS has widespread use in safety-critical environments. In applications, it is prevalent in IoT devices, consumer electronics, industrial automation, medical devices, automotive systems, telecommunications, and aerospace and defense. Its versatility, real-time capabilities, and efficiency make it a go-to choose for developers in various embedded domains.

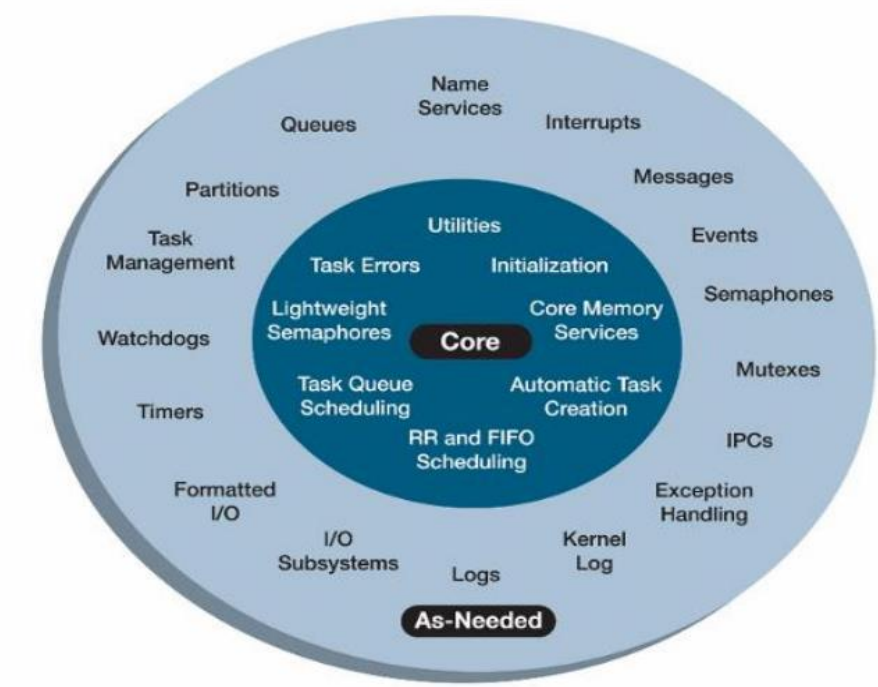


Figure (10). Core and Optional Components.

FreeRTOS (Real-Time Operating System) is an open-source, real-time operating system kernel designed for embedded systems. It provides a multitasking

environment for microcontrollers and microprocessors, allowing developers to create applications with multiple tasks that can run concurrently. Here's a brief overview of the key functionalities of FreeRTOS:

1. **Task Management:** FreeRTOS allows developers to create multiple tasks that can run independently.Each task has its own stack and program counter, enabling concurrent execution of different parts of the application.
2. **Task Scheduling:** FreeRTOS uses a priority-based preemptive scheduling algorithm. Tasks with higher priority are given preference and can preempt lower-priority tasks. Round-robin scheduling is used within tasks of the same priority to ensure fairness.
3. **Semaphore and Mutex:** FreeRTOS provides synchronization mechanisms such as semaphores and mutexes to manage access to shared resources among tasks. Semaphores are useful for signaling between tasks, while mutexes help in preventing data corruption due to simultaneous access.
4. **Queues:** Queues allow tasks to communicate with each other by passing messages. Tasks can send and receive data through queues, facilitating inter-task communication and synchronization.
5. **Event Groups:** FreeRTOS supports event groups, which are sets of flags that tasks can wait for or set. Tasks can be made to wait until specific flags are set.
6. **Timers:** Timers in FreeRTOS can be used for scheduling tasks at specific intervals or after a predefined delay. Timers provide a way to execute code periodically or after a specific time period.
7. **Memory Management:** FreeRTOS includes memory management functions for dynamic memory allocation and deallocation within tasks.
8. **Interrupt Service Routines (ISRs):** FreeRTOS is designed to work with both preemptive and cooperative multitasking within the context of interrupt service routines. It allows tasks to be interrupted by higher-priority tasks or interrupts.
9. **Portability:** FreeRTOS is highly portable and can be easily adapted to different microcontroller architectures and development environments.
10. **Configurability:** Developers can configure FreeRTOS based on the requirements of their applications, enabling them to include or exclude specific features to optimize the kernel's footprint.

FreeRTOS is widely used in various embedded systems, including IoT devices, automotive systems, and industrial applications, where real-time capabilities and resource efficiency are essential. Its open-source nature, portability, and rich feature set make it a popular choice for embedded developers.

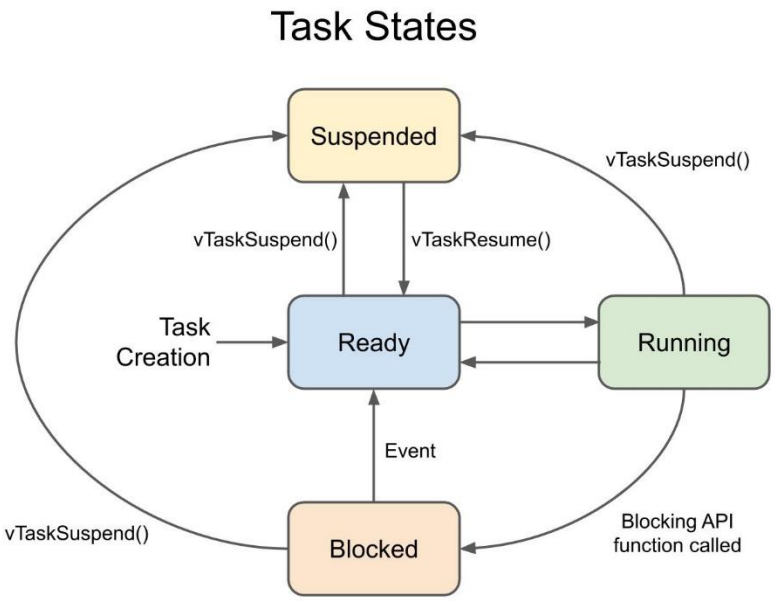


Figure (11). FreeRTOS Task States.

Battery Monitoring Task Scheduling:

```
void Bat_sensor_Data(void * prm)
{
            while(1) {
/* Battery Temperatuure calculation using the NTC 10K and Taking Vin To channel 1
& 2 */
                    ADC_Select_CH1();
                   HAL_ADC_Start(&hadc1);
                   HAL_ADC_PollForConversion(&hadc1, 1000);
                   delay();
                   Temp_sensor_count = HAL_ADC_GetValue(&hadc1);
                   Vout = Temp_sensor_count * (Vsupply/4095);
                   R_NTC = (Vout * R_10k) /(Vsupply - Vout);
                  //calculating the resistance of the thermistor
                   Temp_K = (T0*B_param)/(T0*log(R_NTC/R_10k)+B_param);
                   //Temperature in Kelvin
                   Temp_C = Temp_K - 273.15;  //converting into Celsius
                   HAL_ADC_Stop(&hadc1);
                   ADC_Select_CH2();
                   HAL_ADC_Start(&hadc1);
                   HAL_ADC_PollForConversion(&hadc1, 1000);
                   delay();
```

```
                    Temp_sensor_count1 = HAL_ADC_GetValue(&hadc1);
                    Vout1 = Temp_sensor_count * (Vsupply/4095);
                    R1_NTC = (Vout1 * R_10k) /(Vsupply - Vout1);
                    //calculating the resistance of the thermistor
                    Temp_K1 = (T0*B_param)/(T0*log(R_NTC/R_10k)+B_param);
                    //Temperature in Kelvin
                    Temp_C1 = Temp_K - 273.15;
                     //converting into Celsius
                    HAL_ADC_Stop(&hadc1);
/* Battery Current calculation using sensor current calculation  Channel 3 */
                    ADC_Select_CH3();
                     HAL_ADC_Start(&hadc1);
                     HAL_ADC_PollForConversion(&hadc1, 1000);
                     delay();
                     Current_sensor_count = HAL_ADC_GetValue(&hadc1);
                      voltage =((Current_sensor_count*3.3*2)/4095)*2;
                      Bat_current =((voltage-2.5)/sensitivity);
                      HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
                      HAL_ADC_Stop(&hadc1);


/* Battery voltage calculation using sensor voltage calculation  Channel 4 */

                              ADC_Select_CH4();
                              HAL_ADC_Start(&hadc1);
                              HAL_ADC_PollForConversion(&hadc1, 1000);
                              delay();
                              voltage_sensor_count = HAL_ADC_GetValue(&hadc1);
                              Bat_voltage = ((voltage_sensor_count/4095)*12.2);
                              HAL_ADC_Stop(&hadc1);

if(Temp_C > 37 || Temp_C1 > 37 || Bat_voltage < 9 || Bat_current > 2) {

/* when the Batter  Bank Exceeds the Temp > 37 c and Bat_voltasge is below 9V &
Bat_current > 2A */
                    /*Relay Switch off tghe Battery supply */
                              delay();
                            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, SET);
                            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, SET);
                    }
else if(Temp_C < 37 && Temp_C1 < 37 && Bat_voltage > 9 && Bat_current < 2) {

/* when the Batter  Bank Exceeds the Temp > 37 c and Bat_voltasge is below 9V &
Bat_current > 2A */
                    /*Relay Switch off tghe Battery supply */
                            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, RESET);
                            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, RESET);
                    }
                    vTaskSuspend(T1_Handle); // Task is self-suspended for the Dat

        }

}

void Bat_Sensor_Data_TXn(void *prm)
{
        /* CAN BUS Initilization---------------------------------------*/
                HAL_CAN_Start(&hcan1);
```

```
		CAN_TxHeaderInit();


	while(1) {

	// CAN Frame Transmission in the Task
		delay();
		TxData[0] = (uint16_t)Temp_C;
		delay();
		TxData[1] = (uint16_t)Bat_voltage;
		delay();
		TxData[2] =(uint16_t) Bat_current;
		delay();
		TxData[3] =(uint16_t) Bat_current;

	if(HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox) == HAL_OK)
				{
			HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, SET);
				}

		vTaskResume(T3_Handle);
		vTaskResume(T1_Handle);
		vTaskSuspend(T2_Handle);
	}

}

void Test_Task(void *prm)
{

	while(1) {
    /* Bluetooth Data trnasmission*/

	delay();

		sprintf(BT_Buff,"\nTemp= %.2fC\n",Temp_C);

		HAL_UART_Transmit(&huart5,(uint8_t *)BT_Buff, strlen(BT_Buff), 10);

		delay();

		 sprintf(BT_Buff,"\nBat_vtg= %.2fV\n",Bat_voltage);

		HAL_UART_Transmit(&huart5,(uint8_t *)BT_Buff, strlen(BT_Buff), 10);

		delay();

		sprintf(BT_Buff,"\nBat_crnt= %.2fA\n",Bat_current);

		 HAL_UART_Transmit(&huart5,(uint8_t *)BT_Buff, strlen(BT_Buff), 10);

		 delay();
		vTaskResume(T1_Handle);
		vTaskResume(T2_Handle);
		vTaskSuspend(T3_Handle);
	}
```

}

3.5.1 System-View Timing Diagrams of Running Tasks:

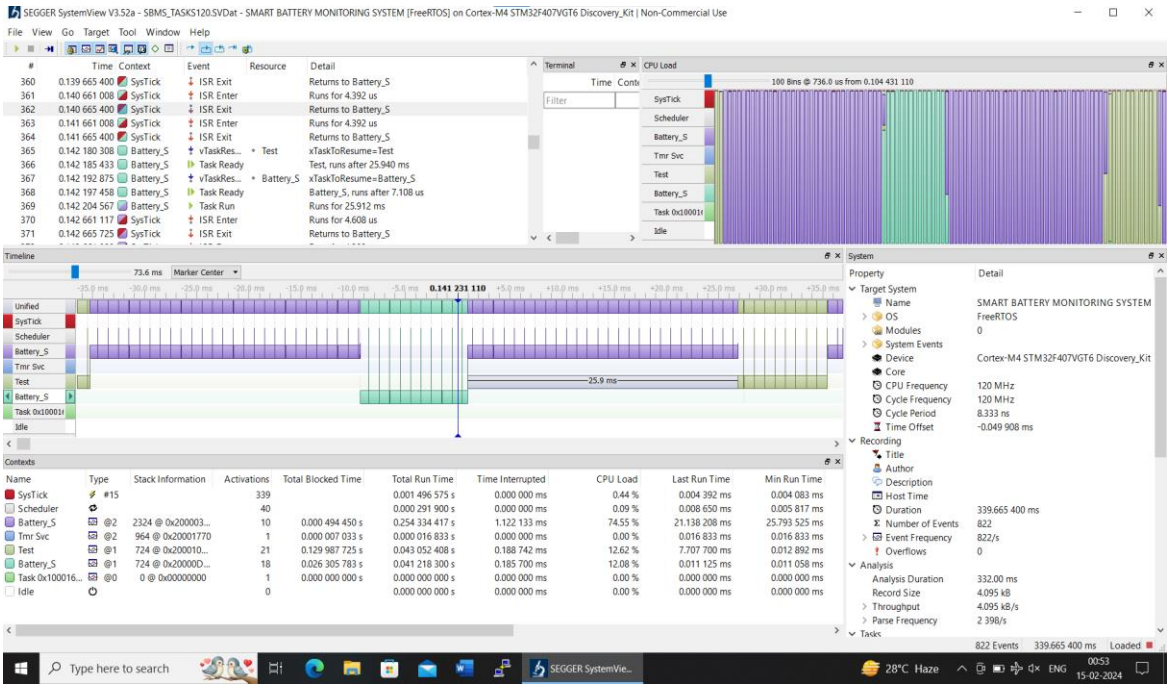


Figure (12). Task Timing Diagrams.

3.6.0 Things Board Cloud Platform:

"ThingsBoard" is an open-source IoT (Internet of Things) platform that allows you to collect, store, and visualize data from your connected devices.

Here's a brief introduction to ThingsBoard and its applications:

Overview: IoT Platform: ThingsBoard serves as a platform for managing and monitoring IoT devices and applications.

Open Source: It is an open-source platform, which means its source code is available for modification and customization.

Features:

1. **Device Management:** ThingsBoard provides functionalities for managing and controlling connected devices.
2. **Data Collection:** It can collect and store data from various IoT devices, sensors, and actuators.
3. **Real-time Visualization:** Offers real-time dashboards and widgets to visualize data from connected devices.
4. **Rule Engine:** Allows you to define rules and triggers based on device data

to automate actions.

Working of ThingsBoard:

1. **Device Integration:** Devices need to be integrated with ThingsBoard. This involves configuring devices to communicate with the platform using supported protocols such as MQTT, CoAP, HTTP, etc.
2. **Data Ingestion:** Once connected, devices send data to ThingsBoard. The platform can handle different types of data, including telemetry and attributes.
3. **Storage:** ThingsBoard stores the incoming data, making it available for historical analysis and reporting.
4. **Visualization:** Users can create dashboards using the web-based interface to visualize real-time and historical data. Widgets like charts, maps, and gauges can be used to represent device information.
5. **Rule Processing:** ThingsBoard has a rule engine that allows users to define actions based on certain conditions. For example, sending notifications, triggering alarms, or updating device attributes.

Applications:

1. **Industrial IoT (IIoT):** ThingsBoard is used in industrial settings to monitor and manage machines, equipment, and processes.
2. **Smart Cities:** It can be applied in smart city solutions, such as monitoring environmental parameters, managing public infrastructure, and optimizing resource usage.
3. **Home Automation:** ThingsBoard can be employed for home automation, allowing users to control and monitor smart home devices.
4. **Healthcare:** In healthcare IoT applications, it can be used to collect and analyze patient data from various medical devices.
5. **Security:** Access Control: ThingsBoard provides access control features to ensure that only authorized users can interact with the platform.

In summary, ThingsBoard simplifies the development and management of IoT solutions by providing a comprehensive platform for device integration, data visualization, and rule-based automation. Its open-source nature allows for flexibility and customization based on specific project requirements

Chapter 4

Results

Results Below figure shows the implementation results of our project that are carried out using STM32Cube IDE, Cloud9 IDE and STM Studio. One figure also shows the detailed connection diagram of CAN Connections for communicating between two boards

4.1 Project Image &STM32 Live Variable Output: As Shown in the below image

I. 1 indicate Main Component of Or Project i.e. STM32F407VGT6 Discovery Board.
II. 2 indicate Bluetooth Module.
III. 3 indicate the ACS712 30A Current Sensor Module.
IV. 4 indicate the 5V 1 channel Optocoupler Relay Module.
V. 5 indicate the 12V LI-ion Battery and NTC10K Sensor Mounted on Battery.
VI. 6 indicate the Zero PCB for Soldering all component on Board.
VII. 7 indicate the 775, 12V 1.5A DC Motor as load to Battery Pack.
VIII. 8 indicate the TFT 2.8 240X320 Display for the Showing the Battery Parameters.
IX. 9 indicate the ESP32 WROOM 32-D Wi-Fi and CAN Functionality.
X. 10 indicate the CAN TJ1050 Transceiver Module.
XI. 11 indicate the CAN Bus mounted on the Breadboard.
XII. 12 indicate Voltage Divider Resistor circuit.

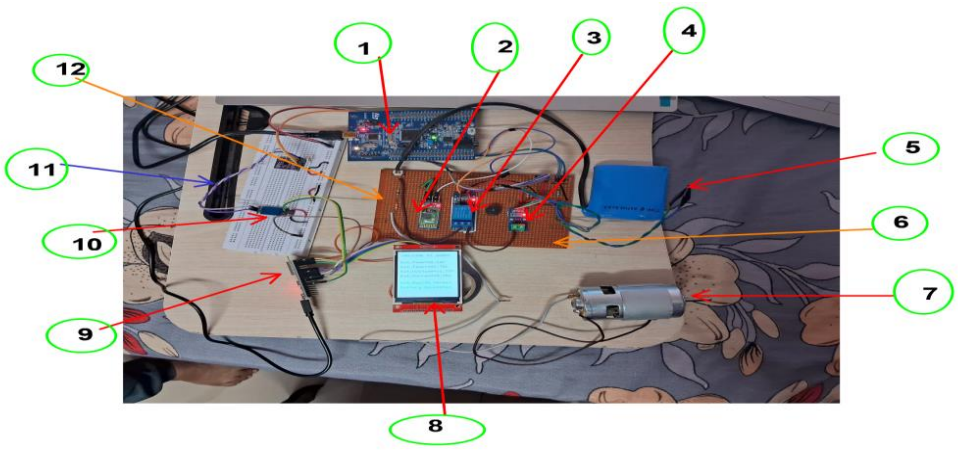


Figure (13). Project Module.

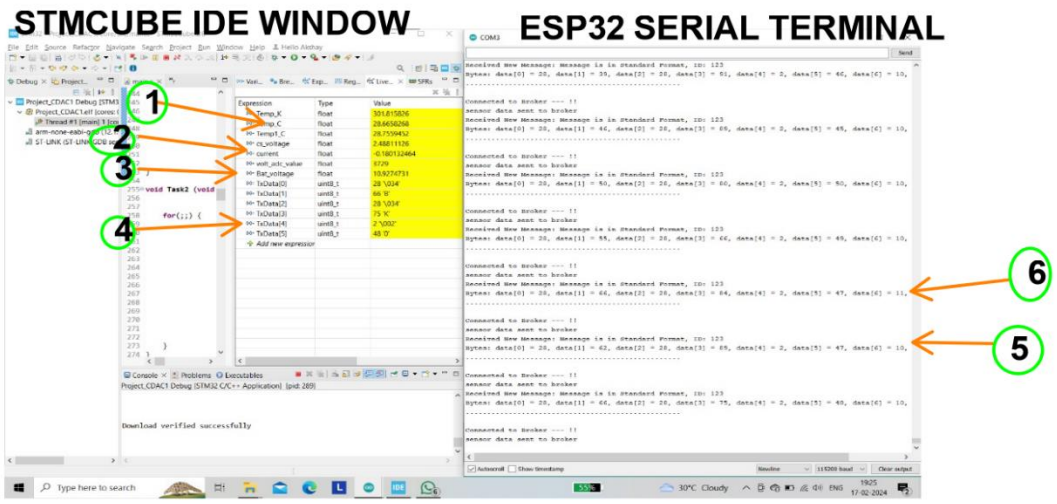


Figure (14). STM CUBEIDE Project Window and ESP32 Arduino Serial Terminal.

As Shown in the Screenshot of PC the STMCBEIDE Expression window we can see the Changing the output from various sensors like Temperature Current and Voltage are shown in the 1,2,3,4,5,6 and CAN Frame shown in 4 no in fig (13).

4.2 Segger Sysview Timing Diagram of Running Task in Given Manner:

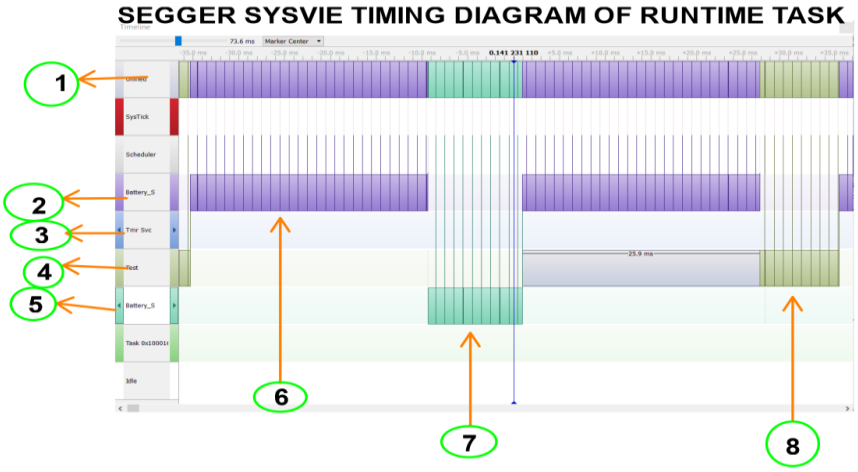


Figure (15). System-View Running Task Diagrams In the Given Priority 2,1,1; In the Fig (15) 1 indicate the overall the all Task on single graph .2,3,4 shows the Task name and Run Time FreeRTOS Os scheduling task in the given manner. The 6 shows Task NO. 1 Runtime ,7 shows the Task2 Runtime and 8 shows the Task 3 Runtime.

4.3 ThingsBoard Cloud Visual Display Result:

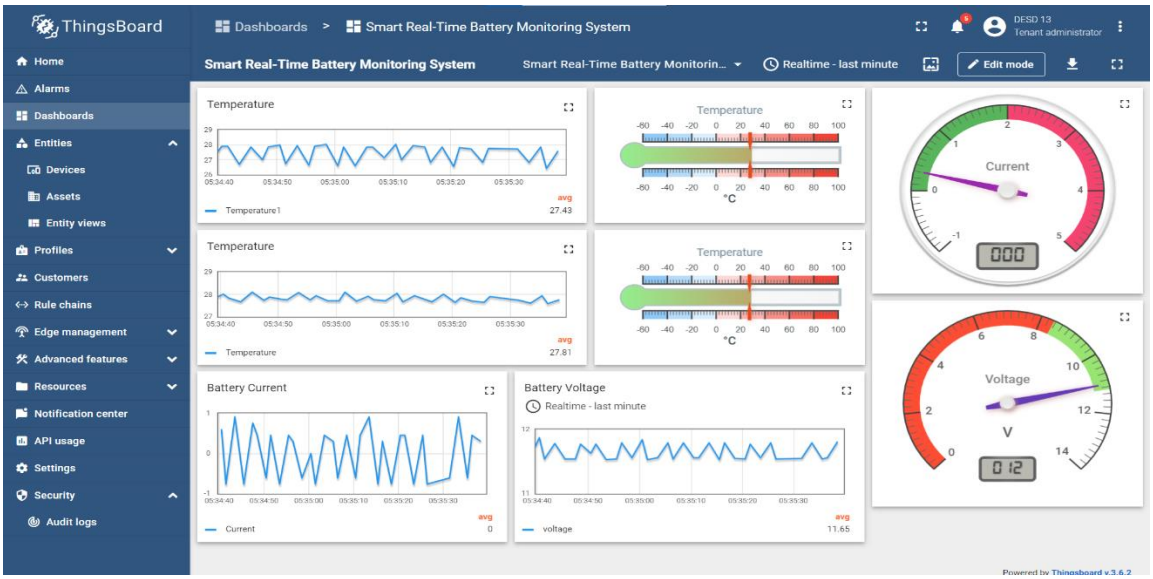


Figure (16). ThingsBoard GUI interface for the showing the Battery Parameters.

4.4Bluetooth Terminal Output: In the below image the Battery Parameters value are show on the user Mobile. As shown in the image Temperature, Voltage and Current are Shown.

Figure (17). Bluetooth Terminal App shows the Battery Parameters.

Chapter 5

Conclusion

5.1 Conclusion Thus, by the following system For the Real Time Monitoring the Battery Parameters and Avoid the Accidental Hazard of various batteries. The FreeRTOS OS Helps for the Task management in this project. CAN Based communication protocol provide a facility to transfer the DATA Of Battery parameters from the SMT32 Microcontroller to the ESP32 controller for the further send to Thingsboard cloud platform. As a result, we can save the battery from over temperature, Voltage and Current and accidental damage avoided, which may have cause the Hman life as well as Economic cost. This system more deterministic because implementation of RTOS, i.e. time critical response is achieved. We are sending the data to ThingBoard and displaying it there so it can be accessed from other location. In essence, the Smart Real-Time Battery Monitoring System not only showcases the technical prowess of the implemented components but also addresses practical challenges in vehicular safety and maintenance. The project's success opens doors for further exploration and advancements in the field of real-time monitoring and control systems.

5.2 Future Enhancement: A Real-Time Battery Monitoring System (BMS) has significant future scope in various industries, particularly in automobiles and other sectors where battery-powered devices or systems are prevalent. Here are some potential areas of growth and applications for a Real-Time Battery Monitoring System.

1. **Automotive Industry:** Electric Vehicles (EVs): As the adoption of electric vehicles continues to grow, the need for efficient battery monitoring becomes crucial. A Real-Time BMS can enhance the performance, safety, and longevity of electric vehicle batteries by providing continuous monitoring of key parameters like voltage, temperature, and state of charge.
2. **Hybrid Vehicles:** Hybrid vehicles, which combine internal combustion engines with electric propulsion, can benefit from advanced battery monitoring to optimize energy usage and improve overall efficiency.
3. **Renewable Energy:** Energy Storage Systems: Battery technologies play a vital role in storing energy from renewable sources such as solar and wind. Real-Time BMS can optimize charging and discharging cycles, ensuring the longevity and reliability of energy storage systems.

4. **Consumer Electronics:** Smartphones, Laptops, Wearables: Real-Time BMS can be implemented in portable electronic devices to monitor battery health, prevent overcharging, and enhance the overall safety and lifespan of batteries in consumer electronics.
5. **Industrial Equipment:** Forklifts and Material Handling Equipment: Battery-powered forklifts and other industrial equipment can benefit from a Real-Time BMS to monitor battery status, prevent unexpected failures, and optimize charging cycles.
6. **Uninterruptible Power Supplies (UPS):** Implementing BMS in UPS systems ensures the reliability of backup power in critical applications by actively monitoring battery health.
7. **Telecommunications:** Base Stations and Towers: Real-Time BMS can be integrated into the backup power systems of telecommunications infrastructure to ensure continuous operation during power outages.
8. **Aerospace:** Satellites and Spacecraft: In space applications, where reliability is paramount, a Real-Time BMS can play a crucial role in monitoring and managing the health of batteries in satellites and spacecraft.
9. **Medical Devices:** Implantable Medical Devices: Real-Time BMS can enhance the safety and performance of medical devices powered by batteries, such as pacemakers and insulin pumps, by providing continuous monitoring and early fault detection.
10. **Data Centers:** Uninterruptible Power Supplies (UPS): Data centers rely on backup power systems to ensure continuous operation. Real-Time BMS can optimize battery performance and provide early warnings for maintenance, reducing the risk of downtime.
11. **Military and Defense:** Military Vehicles and Equipment: Real-Time BMS can be essential for military applications, ensuring the reliability and performance of batteries in various equipment, including vehicles and portable devices.

Chapter 6

References

1. https://www.freertos.org/index.html
2. http://esp32.net/
3. https://github.com/futureLab-Nepal/mastering-stm32/blob/master/docs/datasheets-user-guides/stm32f407-black/schematics/STM32_F4VET6_black.PDF
4. https://thingsboard.io/docs/
5. https://www.ti.com/lit/an/sbaa127/sbaa127.pdf?ts=1708145131349&ref_url=https%253A%252F%252Fwww.google.com%252F
6. Mastering STM32 Guide Book