Project Report

On

Vehicle Dashboard using CAN/IOT

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

Submitted By:

Aditya Gadhave (250240130002)

Harsh Mishra (250240130011)

Kartik Nambiar (250240130016)

Pavan Bhusare (250240130023)

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

(Pune- 411008)

Acknowledgment

We sincerely thank our project guide, **Dr. Shripad Deshpande**, for his invaluable guidance, technical insights, and constant encouragement throughout the development of our project, "**Vehicle Dashboard using CAN/IoT.**"

We extend our gratitude to **CDAC ACTS Pune** for providing the infrastructure and environment that supported our hands-on learning in Embedded Systems and IoT as part of the **PG-DESD program**. Special thanks to **Mr. Gaur Sunder** (HoD), **Mrs. Risha P R** (Program Head), **Mrs. Srujana Bhamidi** (Course Coordinator), and **Mrs. Namrata Ailawar** (Process Owner) for their support and coordination throughout this journey.

We also acknowledge the faculty, lab assistants, and peers for their collaboration and motivation during development and testing. Finally, heartfelt thanks to our families for their continuous support and encouragement. This project was a valuable experience in practical embedded design, collaboration, and problem-solving.

Aditya Gadhave (250240130002)

Harsh Mishra (250240130011)

Kartik Nambiar (250240130016)

Pavan Bhusare (250240130023)

ABSTRACT

In the automotive industry, real-time monitoring and system intelligence are key to safety and efficiency. This project, "Vehicle Dashboard using CAN & IoT" aims to provide a modern embedded solution that integrates multiple vehicle parameters and environmental data for real-time display and cloud-based logging. Using STM32 and ESP32 microcontrollers, the system acquires live data from various sensors (such as speed, temperature, air quality, and humidity etc) and transfers this data over the CAN protocol to ESP32. ESP32 then visualizes it graphically on a non-touch OLED display and uploads it to cloud platforms such as Blynk. The final system allows both local visualization and remote monitoring via cloud dashboards, providing a compact, modular, and efficient smart vehicle dashboard solution.

Table of Contents

Chapter 1
Introduction

1.1 Introduction

The increasing demand for intelligent transport systems and smart vehicles has made real-time sensor-based monitoring a fundamental requirement in modern automobiles. Traditional dashboards often provide limited data with minimal visualization. However, with the rise of microcontrollers, IoT platforms, and standardized communication protocols like CAN (Controller Area Network), it is now feasible to design dashboards that not only offer real-time insights but also support cloud connectivity.

1.2 Objective and Specifications

The primary objective of this project is to design and develop a Vehicle Dashboard that automates the process of acquisition of various vehicle parameters, while also enabling remote monitoring through cloud connectivity. This system aims to enhance vehicle safety monitoring and having access to various vehicle parameters.

This project titled **"Vehicle Dashboard using CAN & IoT"** is designed to achieve three primary objectives:

1. Acquisition of vehicle sensor data in real time using STM32.
2. Transmission of this data to ESP32 using the CAN protocol.
3. Visualization on a TFT/OLED screen and uploading to the cloud usingWi-Fi.

The specifications of the project are:

1. Two Microcontrollers are used to meet the objective of the project.
2. STM32 has 4 sensors connected, which are running on FreeRTOS to acquire data.
3. ESP32 receives data from STM32 via CAN protocol.
4. ESP32 has two sensors connected and sends the data to the display and cloud.

Chapter 2

LITERATURE REVIEW

Over the years, the automotive and embedded systems industries have seen a significant shift toward intelligent data acquisition and communication systems. This literature survey explores various technologies and related research that support and validate the relevance of our project.

CAN Protocol in Automotive Applications

The Controller Area Network (CAN) protocol, first developed by Bosch in 1986, has become a cornerstone of intra-vehicle communication. It provides a robust and real-time bus system that allows microcontrollers and devices to communicate without a host computer. CAN is widely used in modern automotive systems to facilitate communication between the Engine Control Unit (ECU), sensors, and display systems.

According to the SAE (Society of Automotive Engineers), CAN's reliability, error detection, and arbitration capabilities make it ideal for critical automotive functions. Studies have also shown that CAN is effective in reducing wiring complexity and improving fault tolerance within vehicles.

IoT in Smart Transportation

The Internet of Things (IoT) has revolutionized the transportation sector by enabling real-time data collection and remote diagnostics. Research from IEEE and other sources highlights the role of IoT in creating smart cities and enhancing vehicle performance monitoring and cloud communication is made possible, especially for embedded devices with limited resources.

In our project, we adopt real-time publishing of sensor data to cloud dashboards like Blynk, enabling remote access to vehicle health and behavior.

Use of STM32 and ESP32 in Embedded Projects

STM32 microcontrollers are known for their high-performance ARM Cortex-M cores, support for communication protocols like CAN, and efficient power usage. ESP32, on the other hand, offers Wi-Fi and Bluetooth capabilities with dual-core processing, making it ideal for IoT applications.

Numerous academic and industry projects have successfully integrated these microcontrollers for smart home automation, wearable health monitors, and vehicle telemetry systems. These real-world implementations highlight their practicality, cost-efficiency, and ease of integration.

Related Work and Gaps Identified

In a recent project titled "Vehicle Monitoring System using CAN and GSM" (IJAREEIE, 2021), researchers developed a system that used CAN for data collection and GSM for remote monitoring. However, the system lacked a graphical user interface and real-time cloud integration.

Similarly, a study from the International Journal of Computer Applications (IJCA) explored air pollution monitoring using Arduino and Wi-Fi, but it didn’t include inter-device communication or advanced visualization.

Our project addresses these gaps by combining CAN for sensor-to-processor communication, ESP32 for IoT data handling, and TFT/OLED graphics for an intuitive user interface.

Chapter 3

Methodology and Techniques

3.1 Methodology:

The architecture of the Vehicle Dashboard Using CAN & IOT follows a structured approach involving hardware design,software development, sensor interfacing , communication setup and cloud integration.

The overall methodology is divided into following stages:

1. System Architecture Design

   The system is divided into two main controllers:

   - STM32 Microcontroller : Handles real-time sensor data acquisition and sends the data over to ESP32 using CAN protocol.
   - ESP32 Microcontroller : Renders the data acquired from the STM32 onto the OLED screen along with the data acquired from the sensors connected to ESP32 and sends the required data to the cloud like Blynk.

2. Real Time Task Management Using FreeRTOS

   FreeRTOS is being used on the STM32 to manage multiple tasks, including:

   - Reading Temperature from LM35.
   - Reading Obstacle Distance through HC-SR04.
   - Reading Speed of Vehicle using HC-89.
   - Detects if Jerk occurs using MPU6050.
   - Sends all the data to ESP32 using CAN Protocol.

   FreeRTOS is being used on the ESP32 to manage multiple tasks, including:

   - Reading Temperature and Humidity using DHT-11.
   - Reading AQI of air using MQ-135.
   - Render the data onto an OLED Display.
   - Receive data from STM32 using CAN protocol.
   - Send all the data onto Cloud like Blynk.

3. CAN Bus Communication and WiFi Network Communication

- Integrates two microcontrollers—STM32F4 and ESP32 communicating over the CAN bus. The STM32 acquires data from vehicle-like sensors and sends it to ESP32.

- All the data from the sensors are sent onto Blynk using Wifi connected to ESP32.

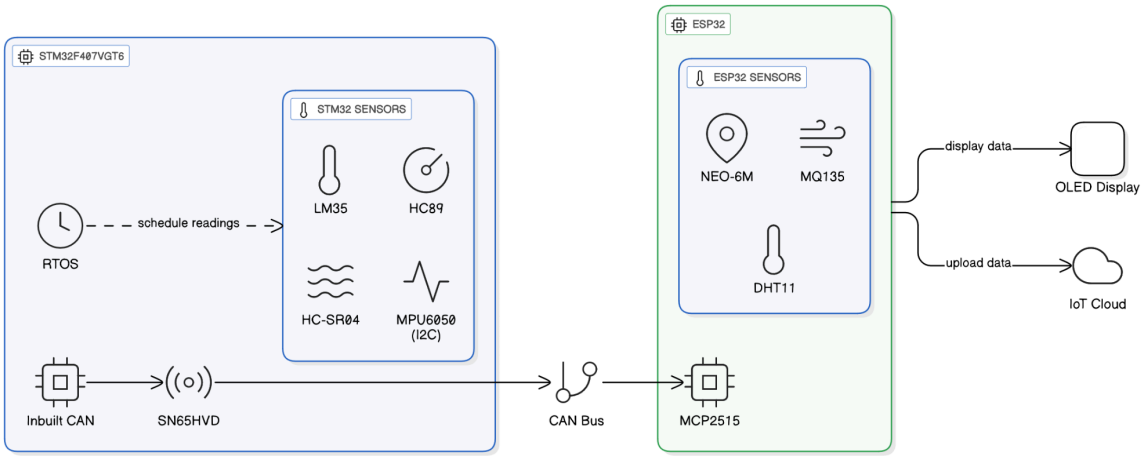

Fig 1 : Block Diagram

3.2 Hardware Modules Description

STM32F407VGT Discovery Board:

Fig 2 : STM32F407VGT6

- Development board based on ARM Cortex-M4 STM32F407VGT6 microcontroller
- Runs up to 168 MHz with 1 MB Flash and 192 KB RAM
- Includes ST-LINK/V2 debugger, accelerometer, microphone, DAC, USB OTG, and LEDs
- Ideal for embedded systems, audio processing, and motion-based applications

ESP32 WROOM-32 Module:

Fig 3 : ESP32 WROOM 32

- Wi-Fi + Bluetooth microcontroller module by Espressif
- Dual-core 32-bit LX6 processor up to 240 MHz
- Includes GPIOs, ADCs, DACs, SPI, I2C, UART, PWM, and touch sensors
- Ideal for IoT, smart home, and wireless applications

HC-89:

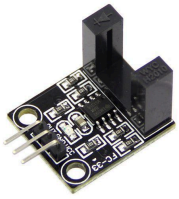

Fig 4 : HC-89

- Detects object interruption using a slot-type optical coupler
- Commonly used for RPM measurement and motor speed feedback
- Outputs digital signal when an object passes through the slot

LM35:

Fig 5 : LM35

- Precision analog temperature sensor with linear output
- Output increases by 10 mV per °C (e.g., 250 mV at 25°C)
- Measures temperature from −55°C to +150°C
- Easy to interface with ADC-enabled microcontrollers

DHT-11:

Fig 6 :DHT-11

- Digital sensor with calibrated output
- Measures temperature (0–50°C) and humidity (20–90% RH)
- Communicates via single-wire serial interface
- Used in weather stations and environmental monitoring

Ultrasonic Sensor:

Fig 7 : HC-SR04

- Measures distance using ultrasonic waves (2 cm to 400 cm range)
- Sends a 40 kHz pulse and calculates time for echo return
- Outputs digital signal via Echo pin
- Widely used in obstacle detection and robotics

Air Quality Sensor:

Fig 8 : MQ-135

- Detects gases like CO2, NH3, NOx, benzene, and smoke
- Analog output proportional to gas concentration
- Used in air quality monitoring and pollution detection
- Requires preheating for accurate readings

Jerk Sensor:

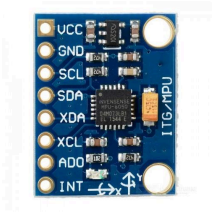

Fig 9 : MPU6050

- Combines 3-axis accelerometer and 3-axis gyroscope
- Communicates via I2C; includes Digital Motion Processor (DMP)
- Measures orientation, acceleration, and angular velocity
- Used in drones, robotics, and motion tracking systems

SN65HVD230 CAN Transceiver:

Fig 10 : SN65HVD230

- 3.3V CAN bus transceiver compliant with ISO 11898-2
- Enables communication between microcontrollers over CAN protocol
- Supports up to 1 Mbps data rate
- Used in automotive and industrial control systems

MCP2515 CAN Controller

Fig 10 : MCP2515

- Standalone CAN controller with SPI interface
- Works with transceivers like SN65HVD230
- Supports CAN 2.0B protocol with 3 transmit and 2 receive buffers
- Used for adding CAN capability to microcontrollers

SSD1306 OLED Display

Fig 10 : SSD1306

- 0.96" monochrome OLED display (128×64 resolution)
- Communicates via I2C
- Low power consumption and wide viewing angle
- Used for displaying text, graphics, and sensor data

3.3 Software Modules Description

STM32CubeIDE:

- **STM32CubeIDE** is an integrated development environment (IDE) developed by STMicroelectronics for STM32 microcontrollers. It combines the STM32CubeMX configuration tool with a powerful Eclipse-based code editor and debugger. STM32CubeIDE allows developers to configure peripherals, generate initialization code, write and compile C/C++ programs, and debug applications using ST-LINK or J-Link probes. It supports real-time variable monitoring, fault analysis, and RTOS-aware debugging, making it a comprehensive platform for embedded development across Windows, Linux, and macOS systems.

Arduino IDE:

- **Arduino IDE for ESP32** is a beginner-friendly development environment that enables programming of ESP32 microcontrollers using the Arduino framework. By installing the ESP32 board support package via the Board Manager, users can write, compile, and upload sketches using familiar Arduino syntax. The IDE supports a wide range of ESP32 boards and provides access to built-in libraries for Wi-Fi, Bluetooth, sensors, and peripherals. Its simplicity and cross-platform compatibility make it a popular choice for IoT projects, rapid prototyping, and wireless applications involving the ESP32.

3.4 Data Flow Chart

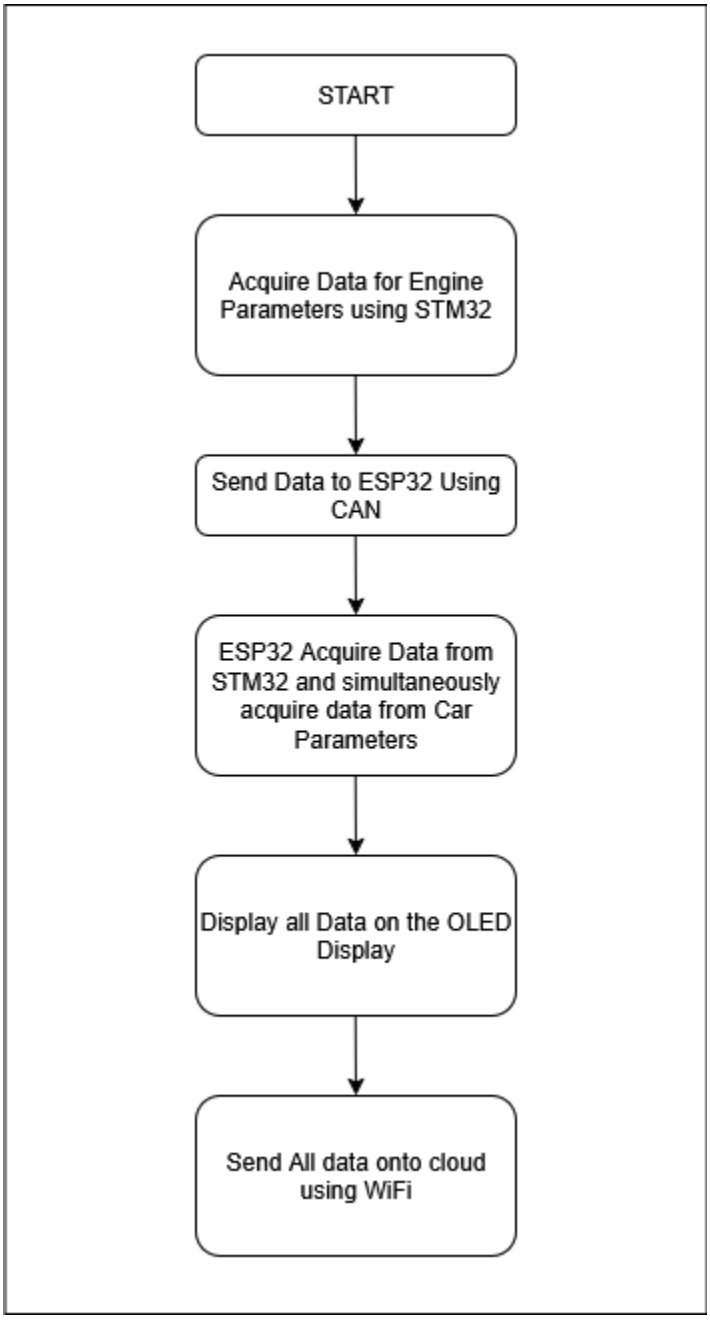

Fig 11 : Data Flowchart

Chapter 4

Implementation

4.1 System Overview

The system is a dual-MCU architecture comprising an STM32 microcontroller for real-time sensor acquisition and CAN transmission, and an ESP32 microcontroller for CAN frame parsing, local sensor interfacing, display rendering, and cloud synchronization via Blynk. FreeRTOS is used on both sides to ensure modularity, concurrency, and deterministic behavior.

4.2 Component List and Purpose

Below is the actual list of components used in our project, along with their functions:

Component	Description	Protocol/Interface Used
STM32F4 Discovery Board	Main sensor interface	CAN, GPIO, ADC,I2C
ESP32-WROOM-32	IoT controller + display driver	CAN, Wi-Fi, UART
MCP2515	CAN Transceiver ICs	CAN TX/RX
DHT11	Car cabin temperature & humidity	Digital GPIO
MQ-135	Air quality detection	Analog GPIO (ADC)
LM35	Engine temperature sensor	Analog GPIO (ADC)
Ultrasonic Sensor	Obstacle distance	GPIO (Trigger-Echo)
IR Speed Sensor	Rotational speed measurement	Digital GPIO
OLED Display (I2C 128x64)	Visual dashboard	I2C
NEO-6M GPS Module	Location tracking	UART
Power Supply (USB/Battery)	Power source	5V/3.3V Lines

4.3 Circuit Diagram

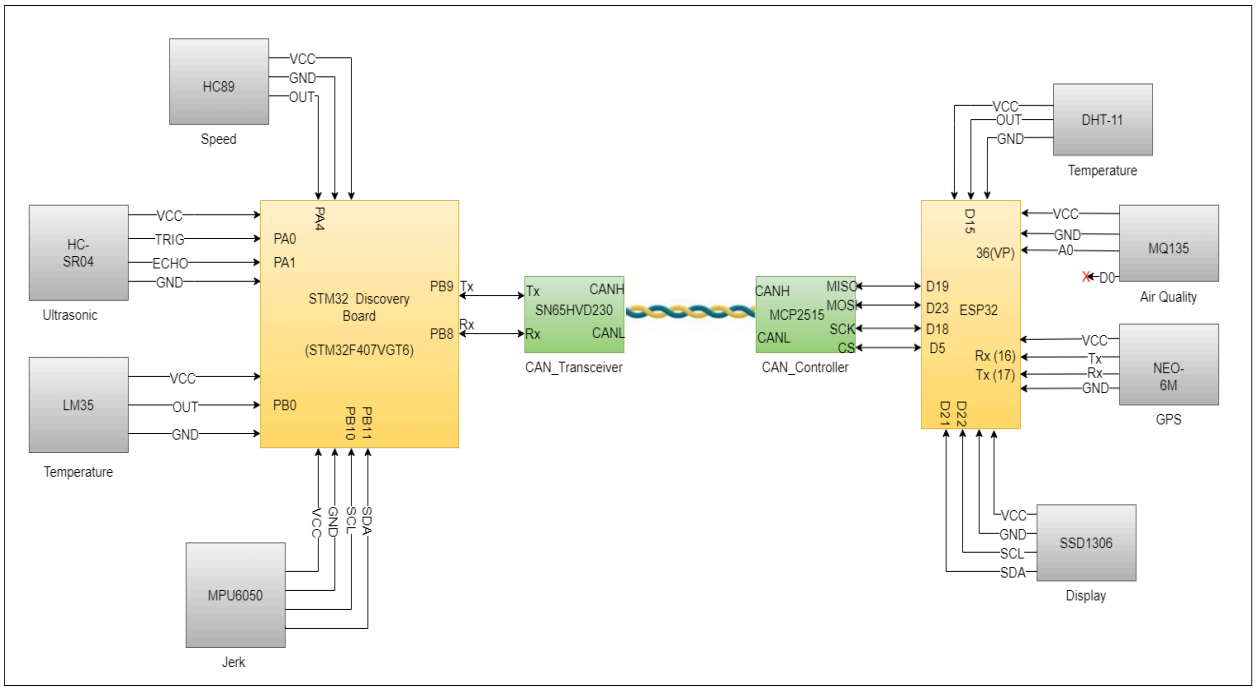

Fig 12 :Circuit Diagram

4.4 STM32F407VGT6 Pin Configuration

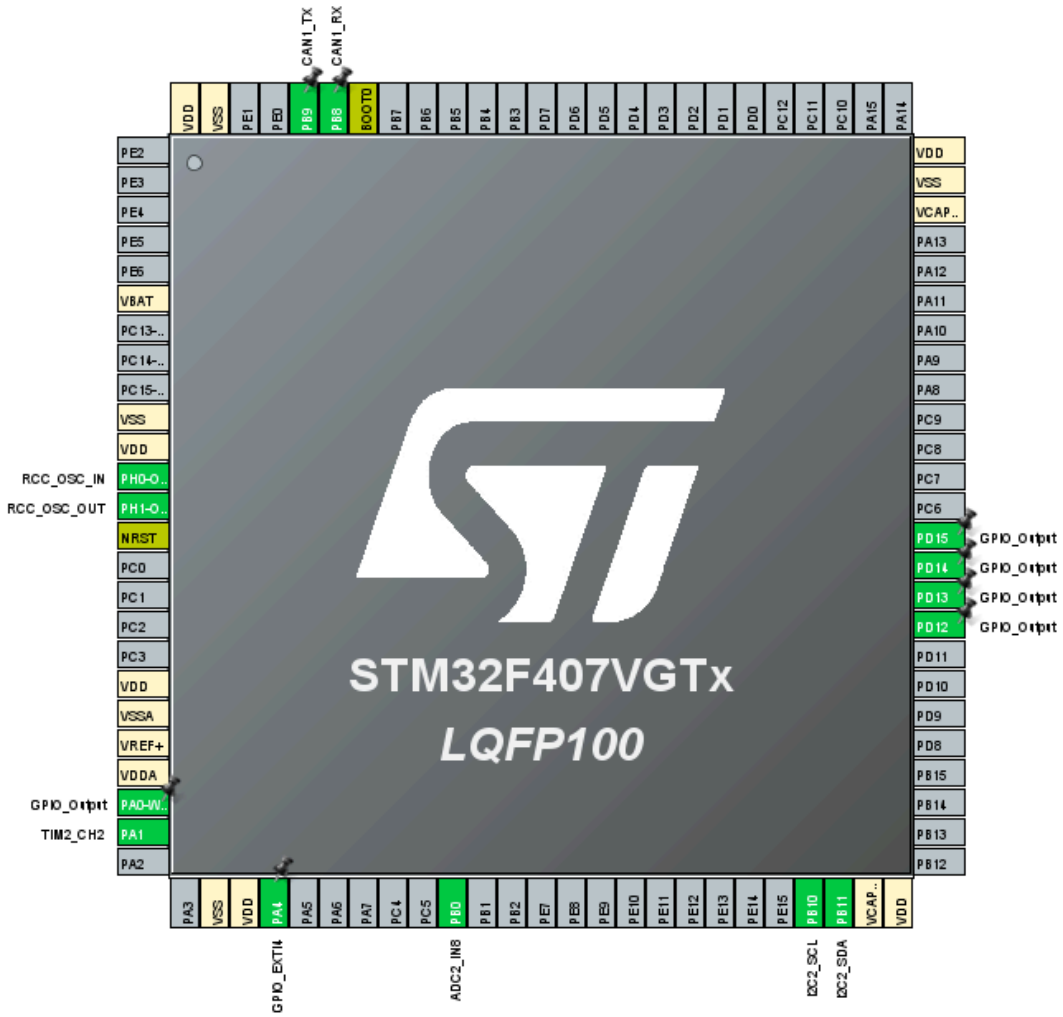

Fig 13 : Pin Configuration on STM32

4.5 Clock Configuration

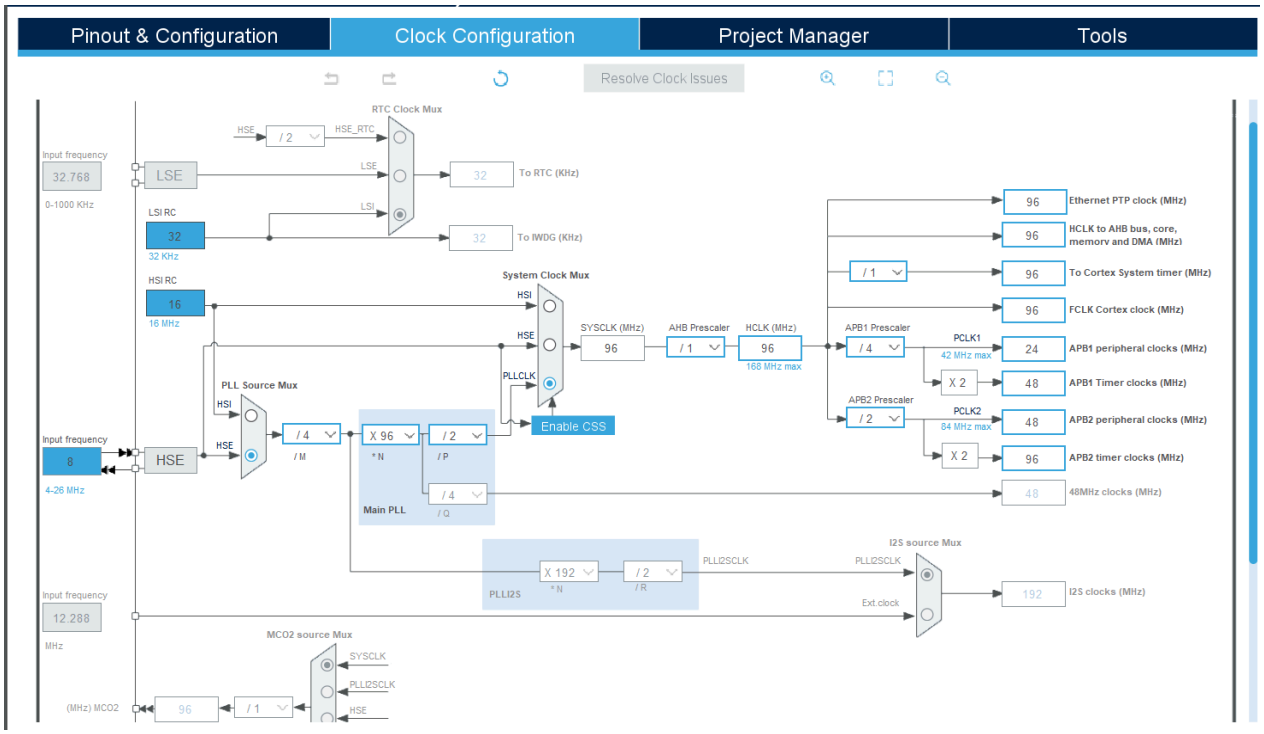

Fig 14 : Clock Configuration on STM32

4.6 Communication Flow

- STM32 acquires sensor data and transmits it over CAN using unique IDs.
- ESP32 receives CAN frames via MCP2515, parses them, and integrates with local sensors.
- ESP32 displays data on OLED and syncs with Blynk cloud dashboard.

4.7 FreeRTOS Task Architecture

STM32 Tasks

Task Name	**Functionality**
Task_Read Temp LM35	Temperature reading data
Task_ReadSpeed	Speed via input capture
Task_Read_MPU6050	Jerk Detection
Task_Read_Ultrasonic	Distance Measurement
Task_CANTransmitter	CAN Frame Dispatch

ESP32 Tasks

Task Name	**Functionality**
Task_CANReceiver	Parse CAN Frames
Task_ReadDHT	Temperature read
Task_ReadPM	Sense Air Quality
Task_ReadGPS	GPS location mapping
Task_Display	Display data
Task_SendCANtoBlynk	Sending data to Cloud

CAN Frame Protocol

CAN ID	Sensor	Action on ESP32
0x100	Temperature	Display, Blynk
0x101	Distance	Display, Blynk
0x102	Jerk Detection	Display, Blynk
0x103	Speed	Display, Blynk

4.8 Sensor Data Acquisition

STM32 Side

- LM35: Analog temperature via ADC.
- HC89: Speed via input capture and timer.
- MPU6050: Jerk detection via interrupt.
- Ultrasonic: Distance via echo pulse timing.

ESP32 Side

- DHT11: Temperature and humidity.
- MQ135 (PM2.5): Analog voltage conversion
- GPS: UART parsing for location, speed, and satellite count.

4.9 Communication Protocols

4.9.1 Controller Area Network

The Controller Area Network bus is an International Standardization Organization (ISO) defined serial communications bus originally developed for the automotive industry to replace the complex wiring harness with a two-wire bus. The Controller Area Network bus, or CAN bus, is a vehicle bus standard designed to allow devices and microcontrollers to communicate with each other within a vehicle without a host computer.

CAN is a reliable, real-time protocol that implements a multicast, data-push, publisher/subscriber model. CAN messages are short (data payloads are a maximum of 8 bytes, headers are 11 or 29 bits), so there is no centralized master or hub to be a single point of failure and it is flexible in size. Its real-time features include deterministic message delivery time and global priority with prioritized message IDs.

Bus arbitration is accomplished in CAN using bit dominance, a process where nodes begin to transmit their message headers on the bus, then drop out of the “competition” when a dominant bit is detected on the bus, indicating a message ID of higher priority being transmitted elsewhere. This means bus arbitration does not add overhead because

once the bus is “won” the node simply continues sending its message. Because there is no time lost to collisions on a heavily loaded network.

- CAN Network

A CAN network consists of several CAN nodes which are linked via a physical transmission medium (CAN bus). In practice, the CAN network is usually based on a line topology with a linear bus to which a few electronic control units are each connected via CAN interface. The passive star topology may be used as an alternative. An unshielded twisted two-wire line is the physical transmission medium used most frequently in applications(Unshielded Twisted Pair — UTP), over which symmetrical signal transmission occurs. The maximum data rate is 1 Mbit/s. A maximum network extension of about 40 meters is allowed. At the ends of the CAN network, bus termination resistors contribute to preventing transient phenomena (reflections). ISO 11898 specifies the maximum number of CAN nodes as 32.

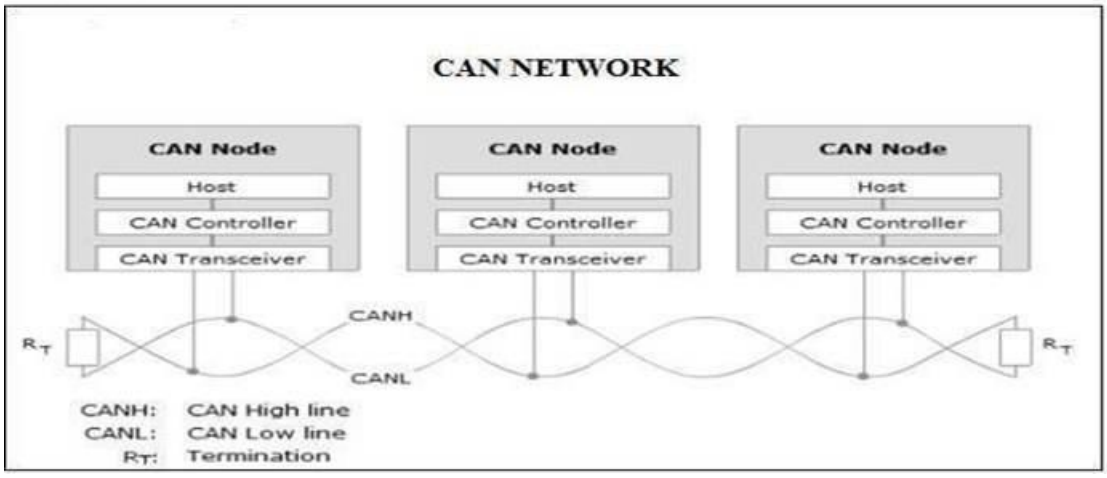

Fig 15 : CAN Network

- CAN BUS

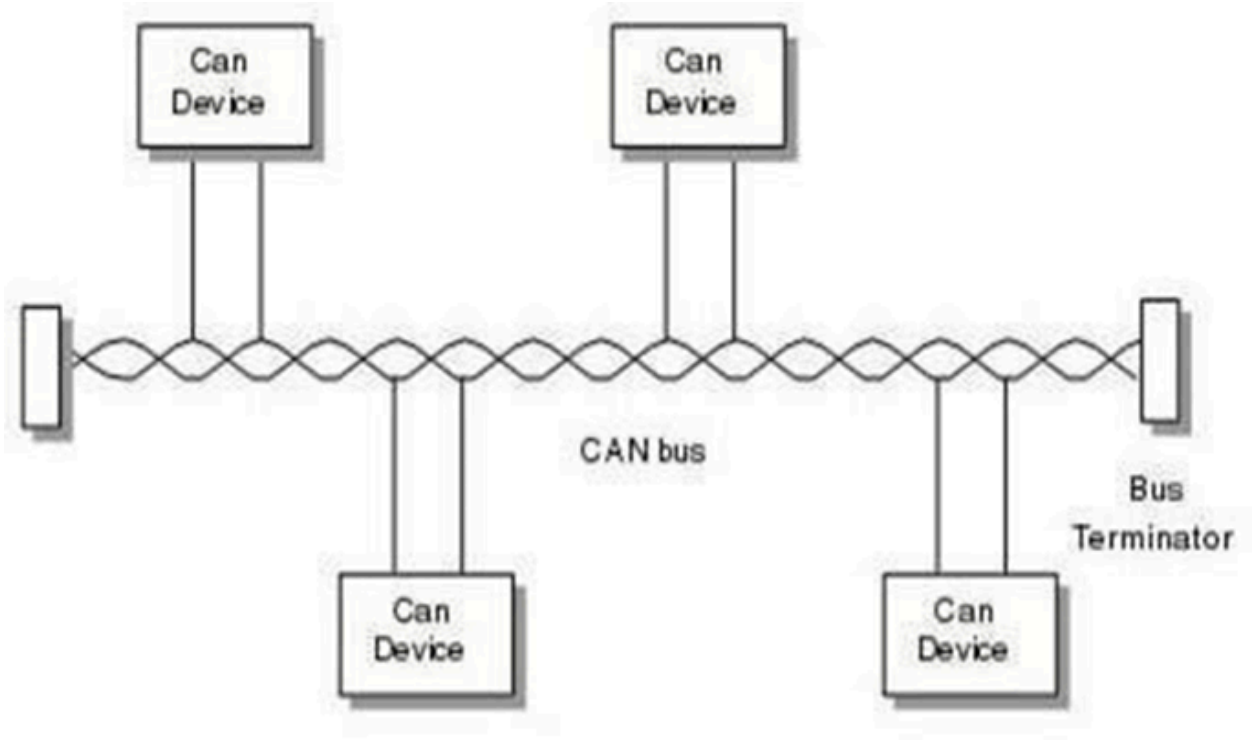

Fig 16 : CAN Bus

Physical signal transmission in a CAN network is based on transmission of differential voltages (differential signal transmission). This effectively eliminates the negative effects of interference voltages induced by motors, ignition systems and switch contacts. Consequently, the transmission medium (CAN bus) consists of two lines: CAN High and CAN Low.

Twisting of the two lines reduces the magnetic field considerably. Therefore, in practice twisted pair conductors are generally used as the physical transmission medium. Due to finite signal propagation speed, the effects of transient phenomena (reflections) grow with increasing data rate and bus extension. Terminating the ends of the communication channel using termination resistors (simulation of the electrical properties of the transmission medium) prevents reflections in a high-speed CAN network.

The key parameter for the bus termination resistor is the so-called characteristic impedance of the electrical line. This is 120 Ohm. In contrast to ISO 11898-2, ISO 11898-3 (low-speed CAN) does not specify any bus termination resistors due to the low maximum data rate of 125 kbit/s.

- CAN FRAMES

    1. Data Frame
    2. Overload Frame
    3. Remote Frame
    4. Error Frame

- Data Frame:

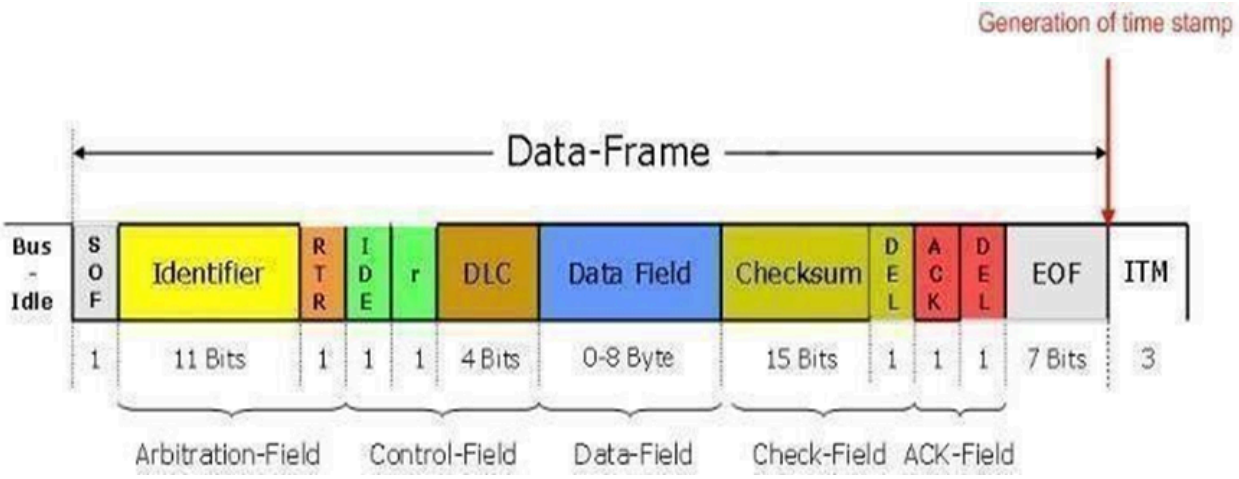

Fig 17 : CAN Data Frame

The data frame is the most common message type, and comprises the Arbitration Field, the Data Field, the CRC Field, and the Acknowledgment Field. The Arbitration Field contains an 11-bit identifier and the RTR bit, which is dominant for data frames. Next is the Data Field which contains zero to eight bytes o+f data, and the CRC Field which contains the 16-bit checksum used for error detection. Last is the Acknowledgment Field.

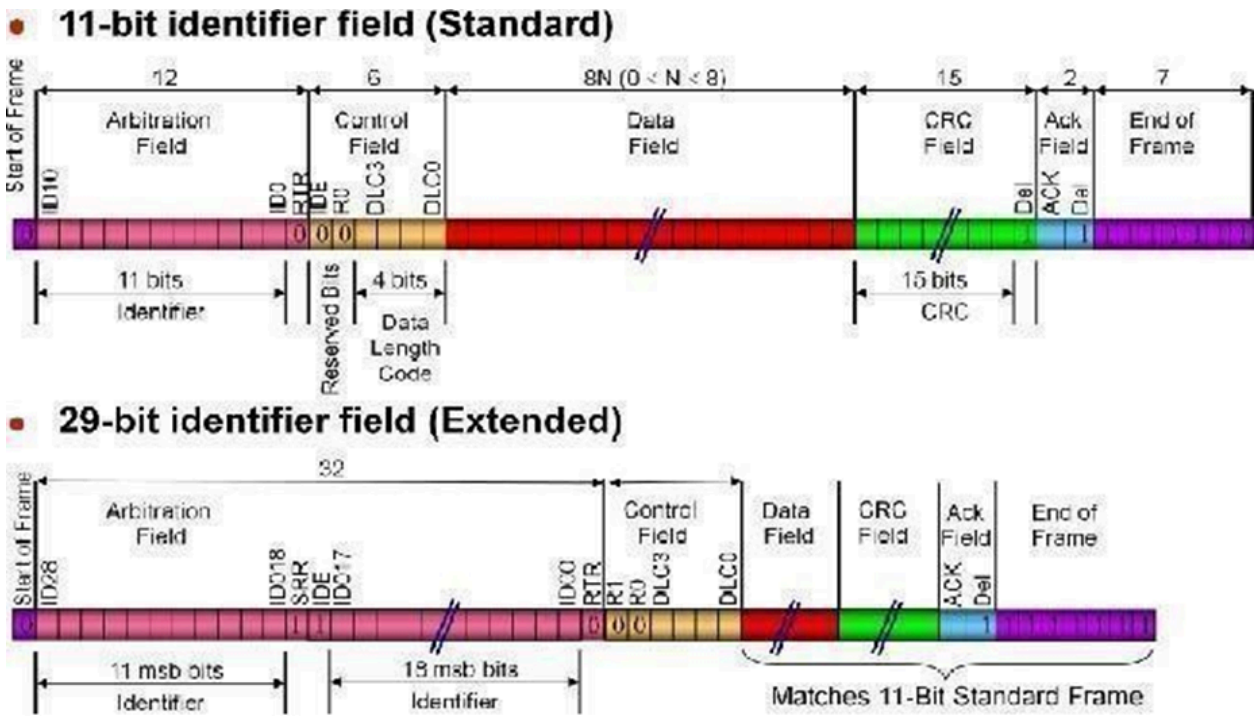

Fig 18 : CAN Data Frame

- Remote Frame:

  The intended purpose of the remote frame is to solicit the transmission of data from another node. The remote frame is similar to the data frame, with two important differences. First, this type of message is explicitly marked as a remote frame by a recessive RTR bit in the arbitration field, and secondly, there is no data.

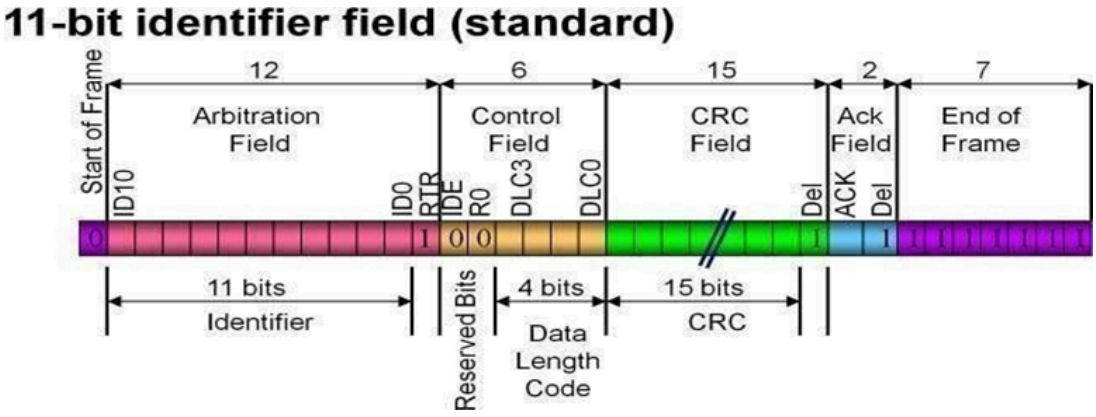

Fig 19 : CAN Remote Frame

- Error Frame:

The error frame is a special message that violates the formatting rules of a CAN message. It is transmitted when a node detects an error in a message and causes all other nodes in the network to send an error frame as well. The original transmitter then automatically retransmits the message. An elaborate system of error counters in the CAN controller ensures that a node cannot tie up a bus by repeatedly transmitting error frames.

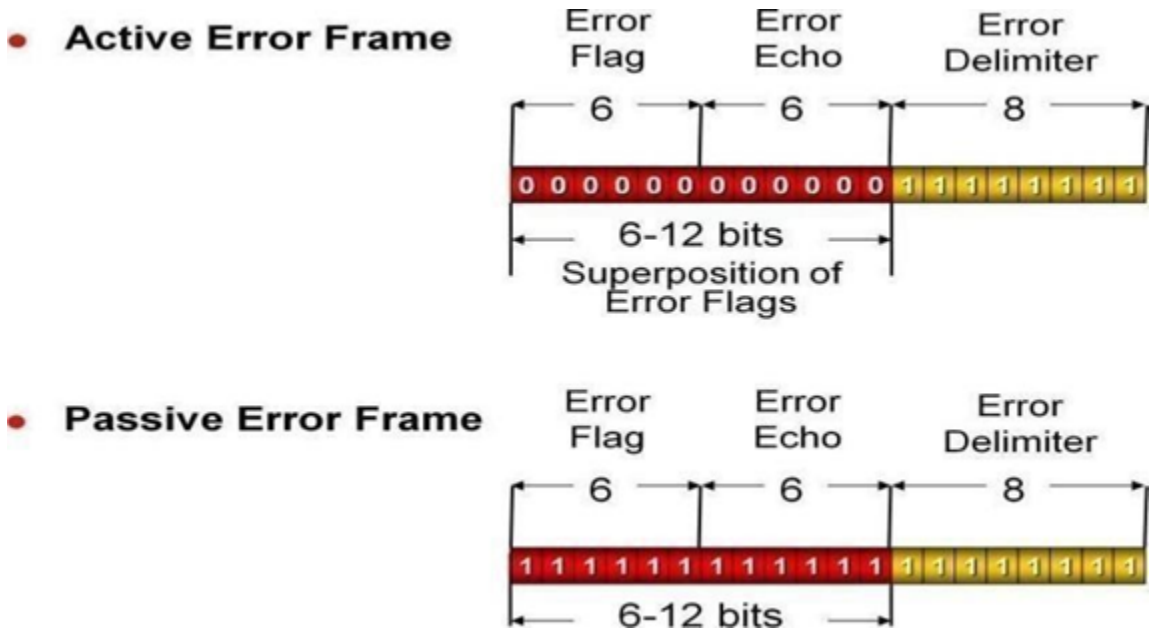

Fig 20 : CAN Error Frame

- Overload Frame:

The overload frame is mentioned for completeness. It is like the error frame with regardto the format, and it is transmitted by a node that becomes too busy. It is primarily used to provide for an extra delay between messages.

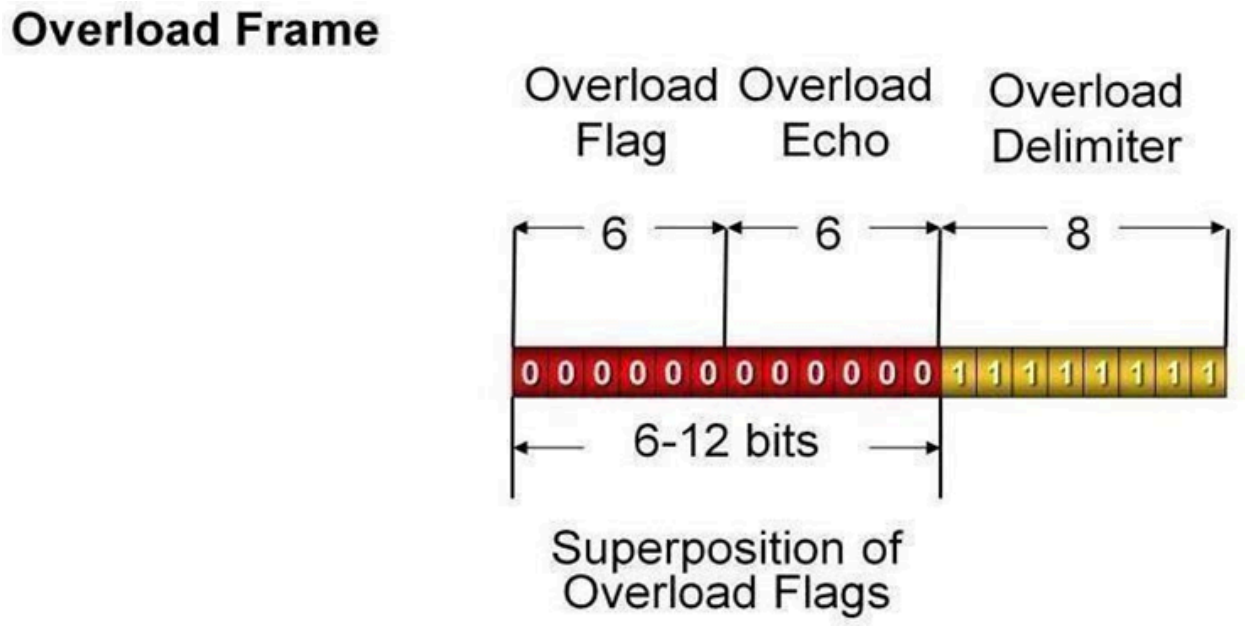

Fig 21 : CAN Overload Frame

- BUS ARBITRATION

The CAN communication protocol is a carrier-sense, multiple-access protocol with collision detection and arbitration on message priority (CSMA/CD+AMP). CSMA means that each node on a bus must wait for a prescribed period of inactivity before attempting to send a message. CD+AMP means that collisions are resolved through a bit-wise arbitration, based on a pre- programmed priority of each message in the identifier field of a message. The higher priority identifier always wins bus access. That is, the last logic high in the identifier keeps on transmitting because it is the highest priority.

Whenever the bus is free, any unit may start to transmit a message. If two or more units start transmitting messages at the same time, the bus access conflict is resolved by bit-wise arbitration using the Identifier. The mechanism of arbitration guarantees that neither information nor time is lost. If a data frame and a remote frame with the same identifier are initiated at the same time, the data frame prevails over the remote frame. During arbitration every transmitter compares the level of the bit transmitted with the level that is monitored on and a "dominant" level is monitored, the unit has lost arbitration and must withdraw without sending one more bit. the bus. If these levels are equal the unit may continue to send.

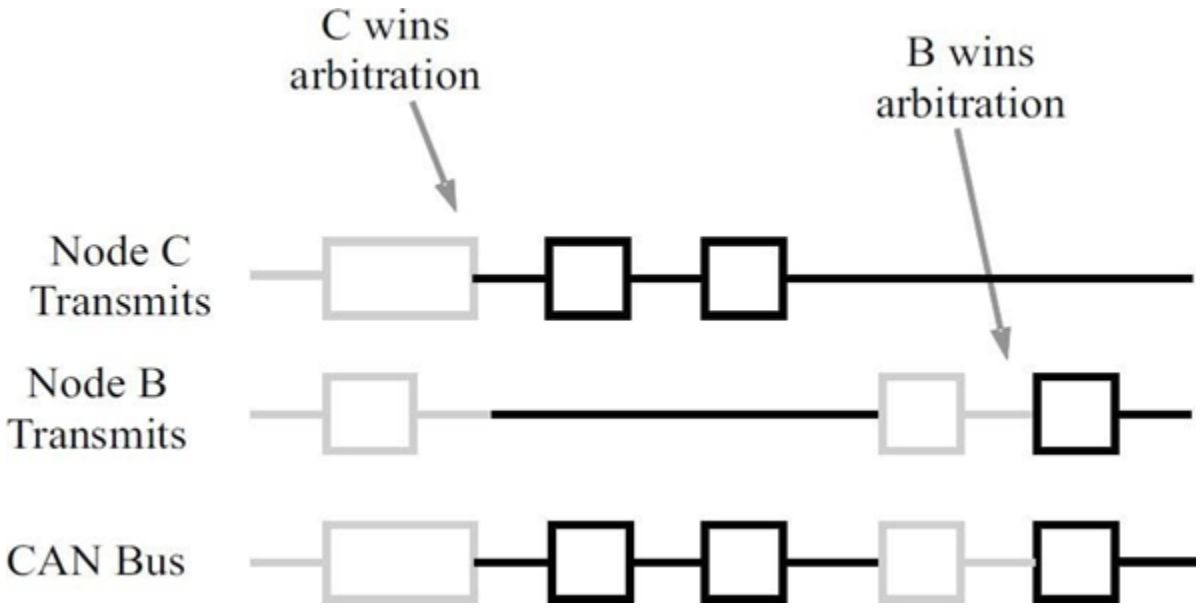

Fig 22 : Bus Arbitration

4.9.2 Inter-Integrated Circuit

I2C, short for Inter-Integrated Circuit, is a multi-master, multi-slave, packet-switched, single-ended, serial communication bus. It is used to connect low-speed peripherals to a microcontroller in a cost effective and simple manner. I2C was developed by Philips (now NXP Semiconductors) in the early 1980s.

Key Features

- Two-Wire Interface: I2C communication requires only two bi-directional lines: Serial Data Line (SDA) and Serial Clock Line (SCL).
- Addressing: Each device on the I2C bus has a unique address. The address can be 7-bit or 10-bit long.
- Data Transfer: Data is transferred in bytes (8 bits), with each byte accompanied by an acknowledgment bit.
- Clock Synchronization: The clock line (SCL) is controlled by the master device, which synchronizes the data transfer.

Communication Protocol Start

- Condition: Initiated by the master device by pulling the SDA line low while SCL is high. This signals the beginning of a communication session.
- Addressing : The master sends the address of the target slave device. The address is followed by a read/write bit indicating the desired operation.
- Data Transfer : Data is transferred in 8-bit bytes. After each byte, the receiver sends an acknowledgment bit to confirm receipt of the data.
- Stop Condition : Initiated by the master device by pulling the SDA line high while SCL is high. This signals the end of the communication session.

4.9.3 RTOS

Free RTOS

The firmware code for the project extensively utilizes FreeRTOS, an open-source real- time operating system kernel, to manage tasks, synchronize operations, and handle interrupts effectively. The implementation of FreeRTOS within the firmware code encompasses several key aspects:

Task Creation

- Two tasks are created using FreeRTOS APIs to enable concurrent execution of different functionalities within the embedded system.
- Task creation involves defining task functions, specifying task priorities, and allocating stack space for each task.

Semaphore Creation

- Semaphores are created using FreeRTOS APIs to facilitate synchronization and mutual exclusion between tasks.
- Semaphores are utilized to coordinate access to shared resources and prevent race conditions in critical sections of the firmware code.

Queue Creation

- Queues are implemented using FreeRTOS APIs to enable inter-task communication and data exchange.
- Queues allow tasks to send and receive messages, sensor readings, or control commands asynchronously, ensuring seamless communication between tasks.

Interrupt Handling

- FreeRTOS provides mechanisms for handling interrupts and integrating interrupt service routines (ISRs) into the real-time operating system.
- Interrupts are configured and managed using FreeRTOS APIs to ensure timely and deterministic response to hardware interrupts.
- Scheduler Configuration
- The FreeRTOS scheduler is configured to prioritize tasks based on their criticality and execution requirements.
- Task priorities are carefully assigned to optimize system performance and responsiveness, ensuring that high-priority tasks are executed with minimal latency.

Code:

https://github.com/GeniusGit4/Car_Dashboard_CAN_IOT.git

Chapter 5

Results

5.1 Project Setup and Working

The final prototype was successfully built using two microcontrollers STM32F407VGT6 for sensor data acquisition and ESP32 for CAN data reception, display, and cloud upload. All components were mounted on a Zero PCB, and the system was powered via USB and battery.

5.2 Data Display and Cloud Integration

- Real-time data from sensors such as speed (HC-89), temperature (LM35), distance (Ultrasonic), and jerk (MPU6050) were transmitted over CAN from STM32 to ESP32.

- ESP32 successfully parsed the received CAN frames and displayed them on a 0.96" I2C OLED screen.

- Simultaneously, data from DHT11, MQ135, and GPS (NEO-6M) was read locally by ESP32 and merged with incoming CAN data.

- All consolidated data was sent to Blynk cloud platform, enabling remote real-time visualization on mobile devices.

5.3 Performance Metrics

Parameter	Value Range Tested	Displayed on OLED	Uploaded to Blynk
Vehicle Speed	0 – 50 RPM	Yes	Yes
Engine Temp	25°C – 75°C	Yes	Yes
Obstacle Distance (US)	5 cm – 200 cm	Yes	Yes
Cabin Temp & Humidity	28°C / 50% RH	Yes	Yes
Air Quality (MQ135)	Varies	Yes	Yes
Jerk Detection (MPU6050)	Sudden tilt/motion detection	Yes (LED)	Yes
GPS (NEO-6M)	Live coordinates, speed	Yes	Yes

5.4 Integration Testing

- All FreeRTOS tasks ran concurrently without any system crashes or timing issues.
- CAN frame IDs and payloads were decoded correctly at ESP32.
- Data was successfully transmitted to Blynk even under fluctuating Wi-Fi conditions.
- The system ran continuously for over 3 hours without interruption, passing a stress test.

5.5 Project Setup

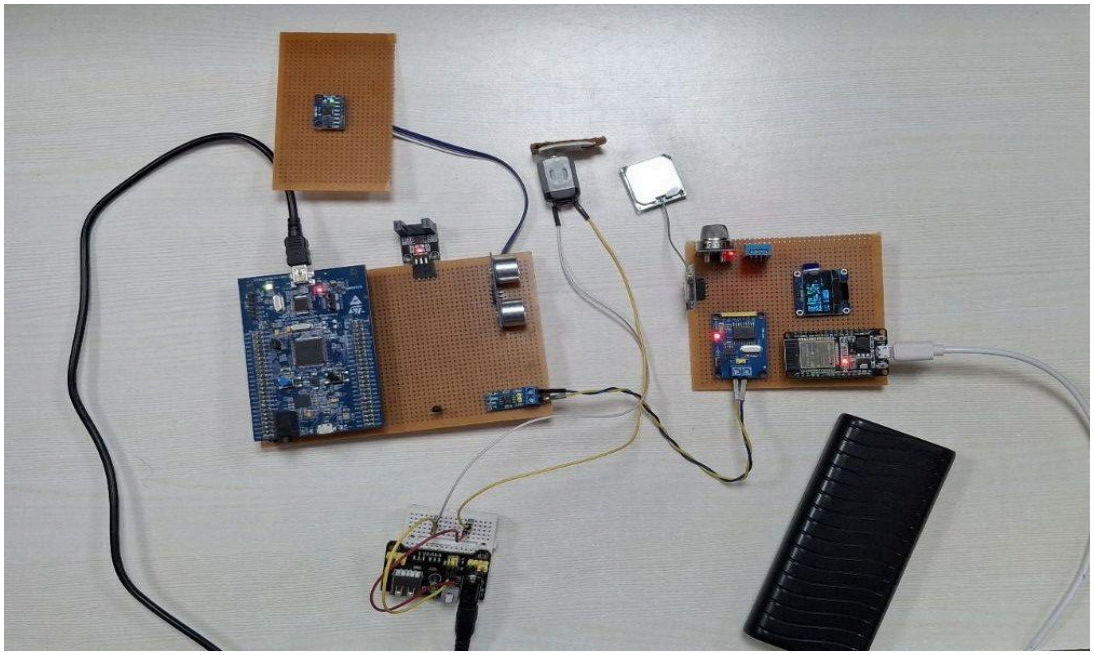

Fig 23 :Project Setup

5.6 CAR DASHBOARD

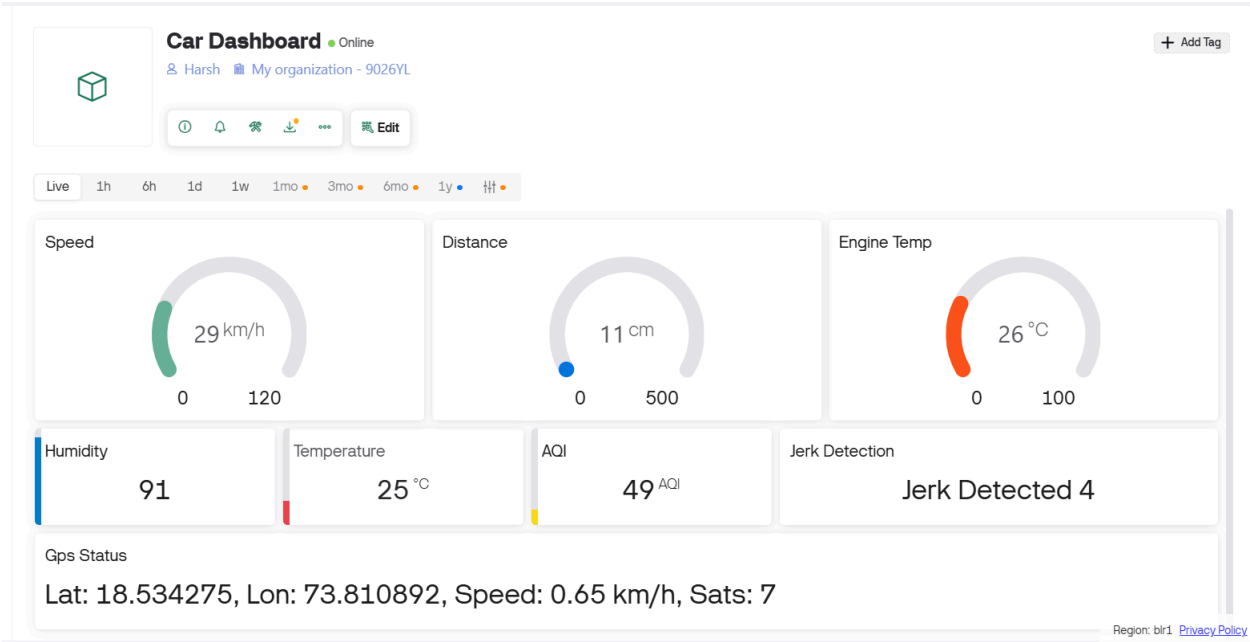

Fig 24 : Cloud Dashboard

5.7 Testing and Validation

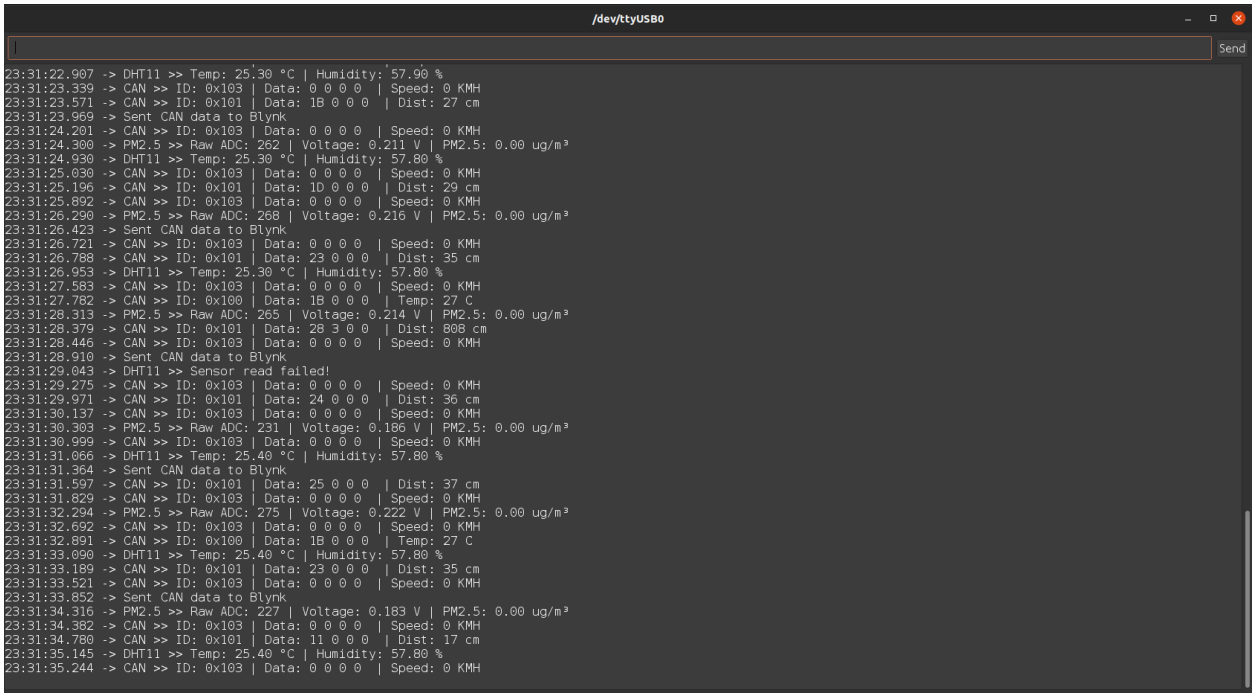

Fig 24 : Testing and Validation

Chapter 6

CONCLUSION

6.1 Conclusion

This project effectively showcases the practical implementation of an intelligent vehicle dashboard system by leveraging embedded microcontrollers, CAN bus communication, and IoT technologies. By integrating hardware components such as STM32, ESP32, MCP2515 CAN transceivers, non-touch graphical displays, and multiple environmental sensors, the project delivers a modular, compact, and scalable dashboard solution.

The system successfully accomplishes:

- Real-time acquisition of vital vehicle and environmental parameters such as speed, temperature, humidity, and air quality.
- Seamless and reliable data transmission between microcontrollers via the CAN protocol.
- Intuitive graphical visualization of live data on a TFT/OLED screen, improving driver awareness.
- Efficient cloud integration through MQTT, allowing remote data monitoring and logging on platforms like Blynk and ThingSpeak.

The outcome of this project not only meets the core objectives but also lays a robust groundwork for advanced automotive applications. With minimal enhancements, this system can evolve into a full-fledged smart car dashboard offering predictive analytics, mobile integration, and enhanced safety features. It also opens avenues for adoption in fields such as fleet management, vehicle diagnostics, smart transport systems, and connected car solutions.

6.2 Future Scope

The Smart Car Dashboard with CAN & IoT Integration presents a highly scalable platform that can be expanded and enhanced in several meaningful ways. Below are five promising directions for future improvement:

1. **Touch-enabled Graphical User Interface (GUI)**
   The current system can be upgraded by integrating a capacitive touch TFT display, enabling interactive features like real-time configuration, menu navigation, and touch-based alerts—enhancing the driver experience and dashboard usability.

2. **OBD-II Protocol Integration**
   By integrating the OBD-II standard via a CAN-compatible adapter, the system can retrieve official automotive diagnostic data like engine load, throttle position, and error codes. This would make the system compatible with real-world vehicles beyond prototypes.

3. **Battery Health Monitoring System**
   Addition of voltage, current, and power monitoring ICs (e.g., INA219) will help assess vehicle battery performance, charging status, and potential degradation over time—essential for electric vehicle diagnostics and predictive maintenance.

4. **AI-Based Predictive Maintenance**
   Data collected from sensors (temperature, distance, etc.) and stored in the cloud can be processed using basic machine learning models to detect abnormal patterns. This enables the prediction of possible system failures before they occur.

Chapter 7

References

1. **STM32F4 Series Reference Manual**
   STMicroelectronics
   https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html

2. **ESP32 Technical Reference Manual**
   Espressif Systems
   https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/index.html

3. **MCP2551 CAN Transceiver Datasheet**
   Microchip Technology Inc.

   https://ww1.microchip.com/downloads/en/DeviceDoc/MCP2551-High-Speed-CAN-Transceiver-DS20001822G.pdf

4. **Adafruit GFX and TFT Libraries for Arduino**
   Adafruit Industries
   https://learn.adafruit.com/adafruit-gfx-graphics-library

5. **PubSubClient MQTT Library for Arduino**
   Nick O'Leary
   https://pubsubclient.knolleary.net/

6. **ThingSpeak IoT Platform Documentation**

MathWorks
https://in.mathworks.com/help/thingspeak/

7. **DHT11 Temperature & Humidity Sensor Datasheet**
   Aosong Electronics
   https://www.componentsinfo.com/dht11-sensor-pinout-datasheet/

8. **MQ-135 Air Quality Sensor Datasheet**
   Winsen Electronics
   https://www.winsen-sensor.com/d/files/PDF/MQ-135.pdf

9. **CAN Protocol Overview – Bosch CAN Specification v2.0**
   Bosch Semiconductor
   https://www.semiconductors.bosch.de/media/ip_modules/pdf_2/can2spec.pdf

10. **TinyGPS++ Library for GPS Parsing**
    Mikal Hart
    https://github.com/mikalhart/TinyGPSPlus

11. **Arduino Official Documentation and Libraries**
    Arduino.cc
    https://www.arduino.cc/reference/en/

12. **ESP32 CAN Communication Using MCP2515**
    TutorialsPoint and GitHub examples
    https://github.com/coryjfowler/MCP_CAN_lib