PROJECT REPORT

CAN and IoT Implementation for Vehicle Monitoring and Analysis

Submitted in partial fulfilment for the award of

Post Graduate Diploma in

EMBEDDED SYSTEMS AND DESIGN

From C-DAC, ACTS (Pune)

Guided by:
Mr. Devendra Dhande

Presented by:

Mr. SAMEER KALMEGH	**230944230036**
Mr. OM BADGUJAR	**230944230064**
Ms. PAYAL DUKARE	**230944230073**
Mr. SANKET LONKAR	**230944230047**

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

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Mr. SAMEER KALMEGH	**230944230036**
Mr. OM BADGUJAR	**230944230064**
Ms. PAYAL DUKARE	**230944230073**
Mr. SANKET LONKAR	**230944230047**

Have successfully completed their project on

CAN AND IoT IMPLEMENTATION FOR VEHICLE MONITORING AND ANALYSIS

Under the guidance of
Mr. Devendra Dhande

Project Guide **Project Supervisor**

Acknowledgement

Mr. SAMEER KALMEGH	**230944230036**
Mr. OM BADGUJAR	**230944230064**
Ms. PAYAL DUKARE	**230944230073**
Mr. SANKET LONKAR	**230944230047**

Abstract

This project focuses on the implementation of Controller Area Network (CAN) and Internet of Things (IoT) technologies for vehicle monitoring and analysis. With the growing complexity of modern vehicles and the increasing demand for real-time data analysis, there is a need for efficient systems to monitor vehicle performance, diagnose issues, and optimize operations. By integrating CAN, which is a widely used communication protocol in vehicles, with IoT devices and platforms, this project aims to create a comprehensive solution for monitoring and analyzing vehicle data remotely.

The implementation involves collecting data from various sensors and systems within the vehicle through the CAN bus. This data is then transmitted to IoT devices equipped with connectivity capabilities, such as Wi-Fi or cellular networks, enabling seamless communication with cloud-based platforms. Leveraging IoT platforms, the collected data can be analyzed in real-time or stored for later analysis.

Overall, the integration of CAN and IoT technologies for vehicle monitoring and analysis offers a scalable and efficient solution that can improve vehicle performance, reduce maintenance costs, and enhance overall operational efficiency in various transportation sectors.

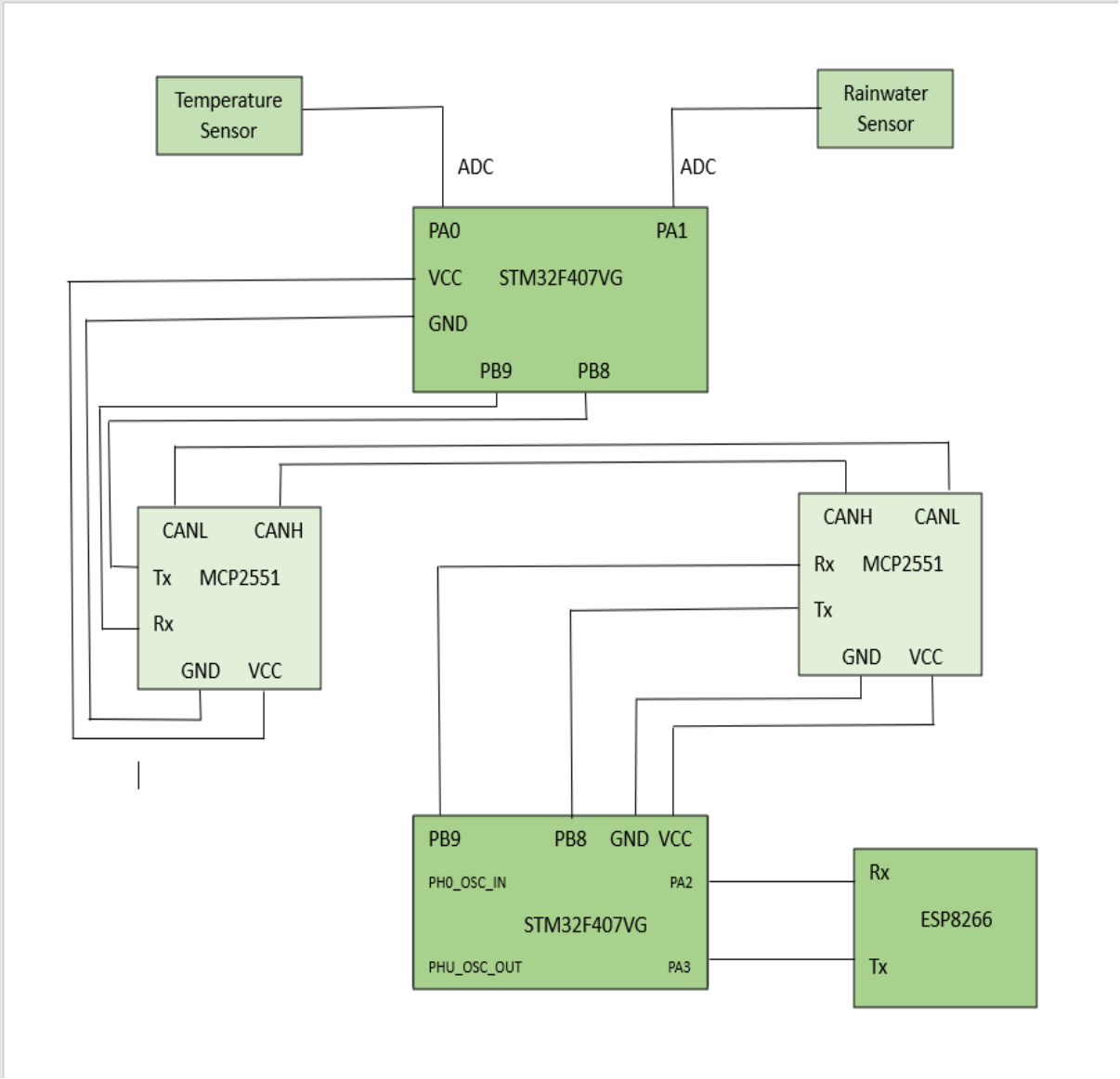
Contents

1. Introduction

1.1 Overview

In this project we proposed to develop a monitoring and analysis system for electric vehicles (E-Vehicles) by integrating Controller Area Network communication and the Internet of Things using STM32F407VG and ESP8266. In addition, a Rainwater Sensor is incorporated to detect water drops, specifically in areas of the car where water could lead to system failure, such as the relay box. Furthermore, temperature sensors are strategically placed in various locations of the car prone to overheating, with the aim of preventing potential fire hazards. The ESP8266 is employed to transmit this comprehensive data over the server for monitoring and analysis.

1.2 Block diagram



1.2 Hardware

Temperature Sensor

The LM35 is a precision integrated circuit temperature sensor that outputs a voltage linearly proportional to the Celsius temperature. Used to measure Hotness and Coldness of object. Output proportional to temperature voltage.
Inside the LM35 sensor, there's a precision temperature-sensitive voltage generator. This generator produces an output voltage that is linearly proportional to the Celsius temperature. The output voltage of the LM35 increases by 10mV per degree Celsius rise in temperature. This linear relationship simplifies the interface with other electronic circuits, as the output can directly represent the measured temperature.
The LM35 operates over a temperature range of -55°C to 150°C, making it suitable for a wide range of applications.

Pinout

Pin	**Function**
1- VCC	power supply
2- OUT	Increase in 10mV for raise of every 1°C
3 -GND	ground

Rainwater Sensor

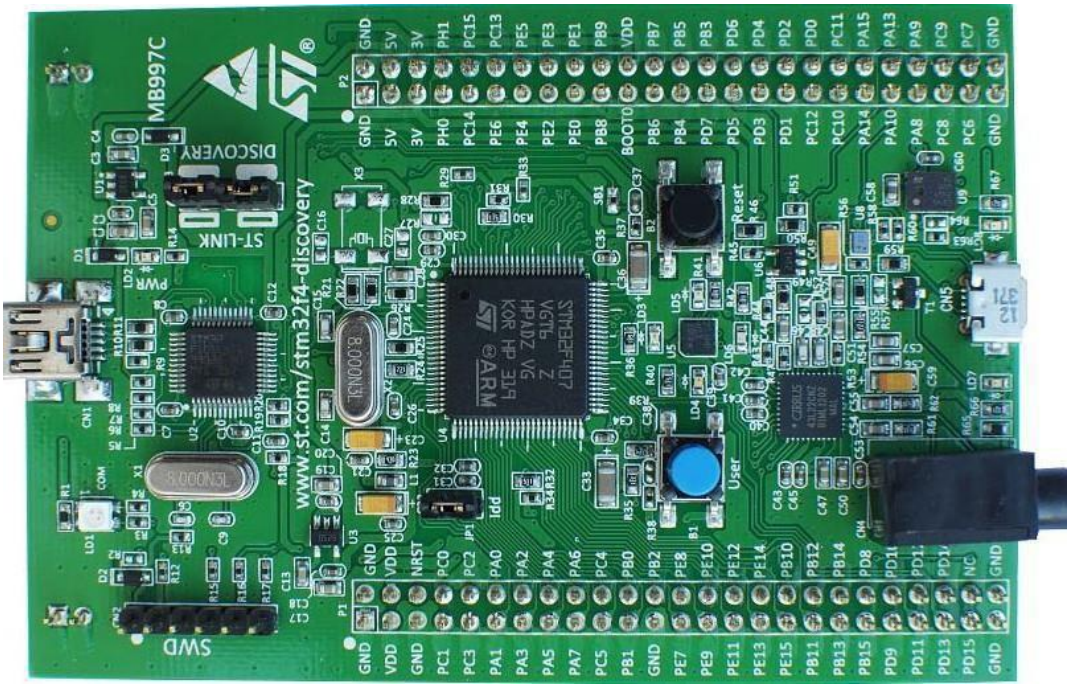
Rainwater sensors, also known as rain sensors or rain detectors, are devices designed to detect the presence and intensity of rainfall. These sensors are commonly used in various applications, automotive applications. A rainwater sensor refers to a component integrated into the vehicle's electrical system to detect the presence and intensity of rainfall. Typically installed on the windshield or in the vicinity of the windshield wipers, this sensor utilizes optical or capacitive principles to detect raindrops hitting the windshield surface.
The integration of a rainwater sensor within the relay box enhances vehicle safety by automating the operation of windshield wipers in response to rainfall. By activating the wipers as soon as rain is detected, the system helps maintain clear visibility for the driver, reducing the risk of accidents and improving overall driving safety, particularly in adverse weather conditions. Additionally, this integration contributes to driver convenience by relieving them of the need to manually adjust wiper settings, allowing them to focus more on the road ahead.

Pinout

Pin	Function
1-VDD	5V DC pin
2-GND	GND
3-DO	It is a low/ high output pin
4-AO	It is an analog output pin

ARM Cortex M4

The ARM Cortex -M4 processor is a high-performance embedded processor with DSP instructions developedto address digital signal control markets that demand an efficient, easy-to-use blend of control and signal processing capabilities. The processor is highly configurable enabling a wide range of implementations fromthose requiring floating point operations, memory protection and powerful trace technology to cost sensitive devices requiring minimal area.



MCP2551 CAN transceiver

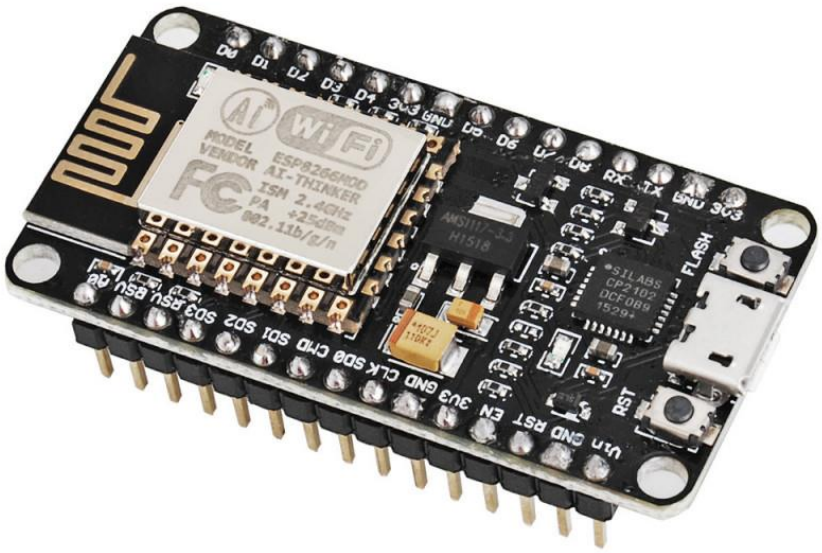
The MCP2551 CAN Board is an accessory board that features an on board CAN transceiver MCP2551 , which is pinout compatible with STM32F407 . It is powered from 3.3V and features ESD protection. The MCP2551 CAN Board is ideal for connecting microcontrollers to the CAN network.

Pinout

Pin	Name	Function
1	3V3	3.3V supply power
2	GND	Ground
3	CTX	CAN controller's TX
4	CTR	CAN controller's Rx
5	CANH	CAN High in twisted pair
6	CANL	CAN Low in twisted pair

ESP8266 NodeMCU

The NodeMCU ESP8266 is a low-cost, open-source development board based on the ESP8266 Wi-Fi module. It's widely used for Internet of Things (IoT) projects due to its affordability, ease of use, and built-in Wi-Fi capabilities. The board features GPIO pins for connecting sensors and actuators, and it can be programmed using the Arduino IDE or Lua scripting language. The NodeMCU ESP8266 enables prototyping and development of IoT applications such as home automation, sensor monitoring, remote control, and web servers.



2. SYSTEM DESIGN

The proposed monitoring and analysis system for electric vehicles (E-Vehicles) integrates Controller Area Network (CAN) communication and the Internet of Things (IoT) using STM32F407VG and ESP8266, incorporating a Rainwater Sensor to detect water drops in critical areas like the relay box and temperature sensors to prevent overheating hazards. The STM32F407VG facilitates CAN communication within the vehicle, collecting data from various sensors and systems, while the ESP8266 module provides Wi-Fi connectivity to transmit data from the temperature and rainwater sensors to a cloud-based server for real-time monitoring and analysis. Robust security measures ensure data integrity and confidentiality during transmission, and user-friendly interfaces offer easy access to insights for vehicle owners, fleet managers, and maintenance personnel. This integrated system aims to improve E-Vehicle performance, reduce maintenance costs, and enhance overall safety through comprehensive data collection, analysis, and remote monitoring capabilities.

2.1 Communication protocol

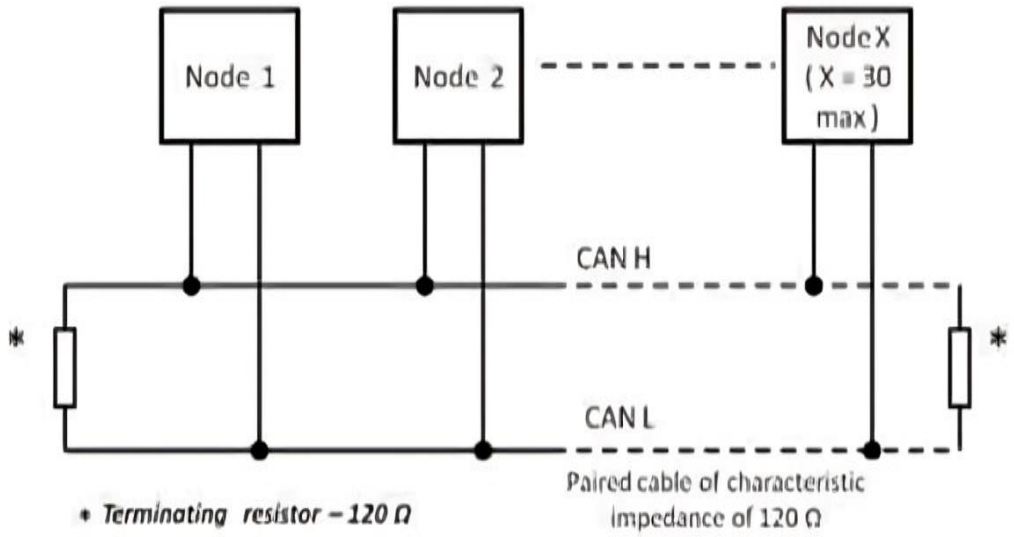
2.1.1 STM32F407

CAN overview

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 witha two-wire bus. The Controller Area Network bus, or CAN bus, is a vehicle bus standard designed to allow devicesand 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 through the use of prioritized message IDs. Bus arbitration is accomplished in CAN using bit dominance, a process where nodes begin to transmit their message headers on thebus, 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 heavilyloaded network CAN is ideal for periodic traffic. Lastly, CAN’s reliability features are suited for typically harsh embedded environments.
CAN was designed for a typical embedded system with periodic, busty traffic,
Where every node transmits at regular intervals. Ethernet, on the other hand, was designed for aperiodic, light traffic and allow number of active transmitters. More significantly, CAN was designed to achieve tight, real-timeschedules, unlike Ethernet

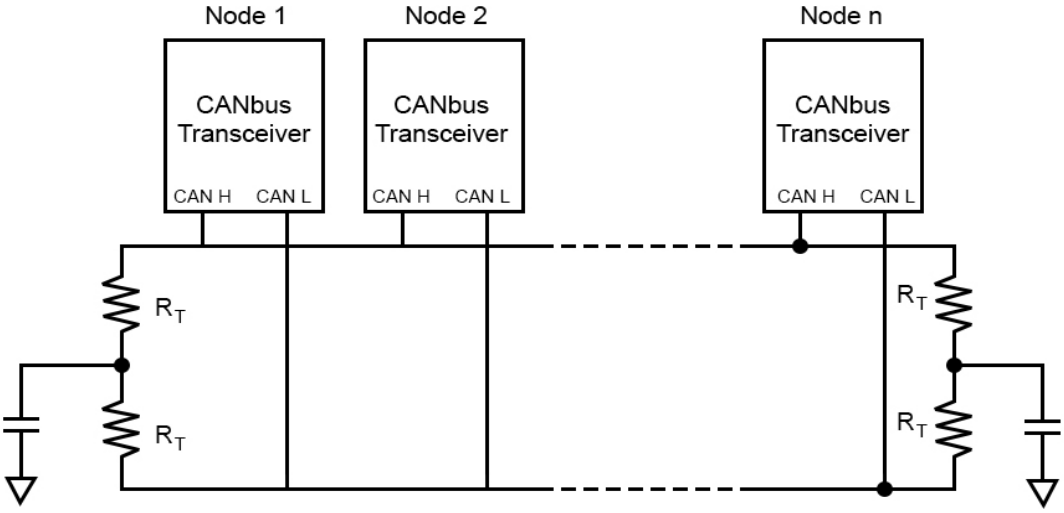
CAN Network

A CAN network consists of a number of CAN nodes which are linked via a physical transmission medium (CANbus). In practice, the CAN network is usually based on a Line Topology with a linear bus to which a number of 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. Themaximum data rate is 1 Mbit/s.
A maximum network extensionofabout40metersisallowed.AttheendsoftheCANnetwork, Bus Termination Resistors contribute to preventing transient phenomena (reflections). ISO 11898 specifies the maximum number of CAN nodes as 32.



CAN Bus

Physical signal transmission in a CAN network is based on transmission of differential voltages (differential signaltransmission). 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 Highand 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 Bus level

Physical signal transmission in a CAN network is based on differential signal trans- mission. The specific differential voltages depend on the bus interface that is used. A distinction is made here between the high-speed CAN bus interface (ISO 11898-2) and the low-speed bus interface (ISO 11898-3).
ISO 11898-2 assigns logical “1” to a typical differential voltage of 0 Volt. The logical “0” is assigned with a typicaldifferential voltage of 2 Volt. High -speed CAN.

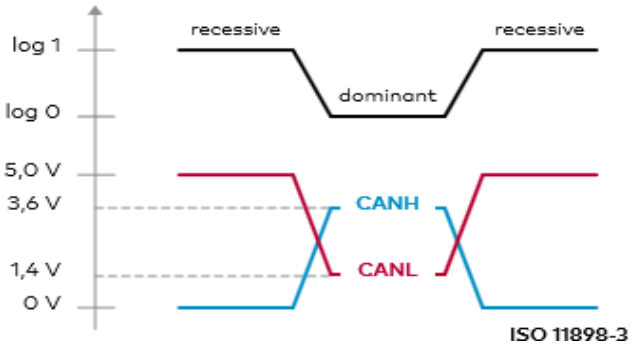


Fig. CAN Low speed

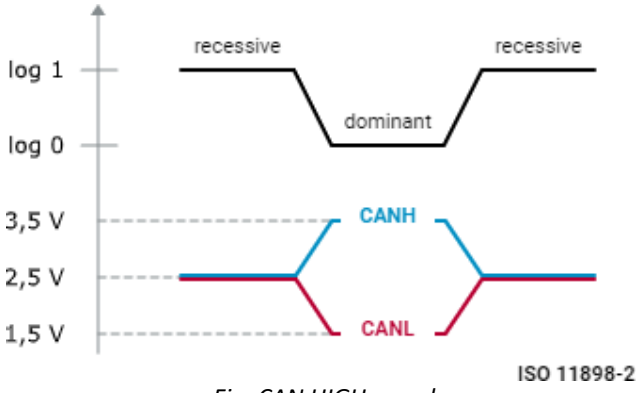


Fig. CAN HIGH speed

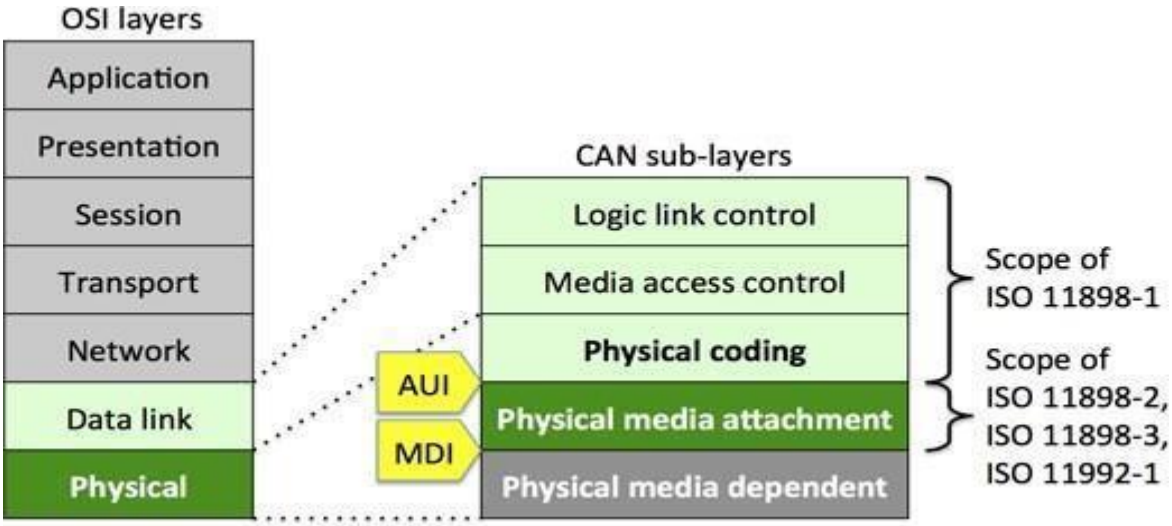
transceivers interpret a differential voltage of more than 0.9 Volt as a dominant level within the common mode operating range, typically between 12 Volt and -12 Volts. Below 0.5 Volt, however, the differential voltage is interpreted as a recessive level. A hysteresis circuit increases immunity to interference voltages. ISO 11898-3 assigns a typical differential voltage of 5 Volt to logical "1", and a typical differential voltage of 2 Volt corresponds to logical "0". The figure "High-Speed CAN Bus Levels" and the figure "Low -Speed CAN Bus Levels" depict the different voltage relationships on the CAN bus.

2.1.2 CAN layer

Data Link Layer

Most of the CAN standard applies to the transfer layer. The transfer layer receives messages from the physical layer and transmits those messages to the object layer. The transfer layer is responsible for bit timing and synchronization, message framing, arbitration, acknowledgement, error detection and signalling, and fault confinement. It performs-

- Error Detection
- Message Validation
- Acknowledgement
- Arbitration
- Message Framing



Physical Layer

CAN bus specify the link layer protocol with only abstract requirements for the physical layer. As a result, implementation often employs custom connector with various sorts of cables, of which two are the CAN buslines. Noise immunity is achieved by maintaining the differential impedance of the bus at a low level with low-value resistors (120 ohms) at each end of the bus.

CAN frames

For transmitting user data, ISO 11898-1 prescribes the so-called data frame. A data frame can transport amaximum payload of eight bytes.

While the generator ECU of relevant information takes the initiative in sending data frames, there also isthe remote frame — a frame type with which user data, i.e. data frames, can be requested from any other CANnode.

The error frame is available to indicate errors detected during communication. An ongoing erroneous datatransmission is terminated and an error frame is issued.

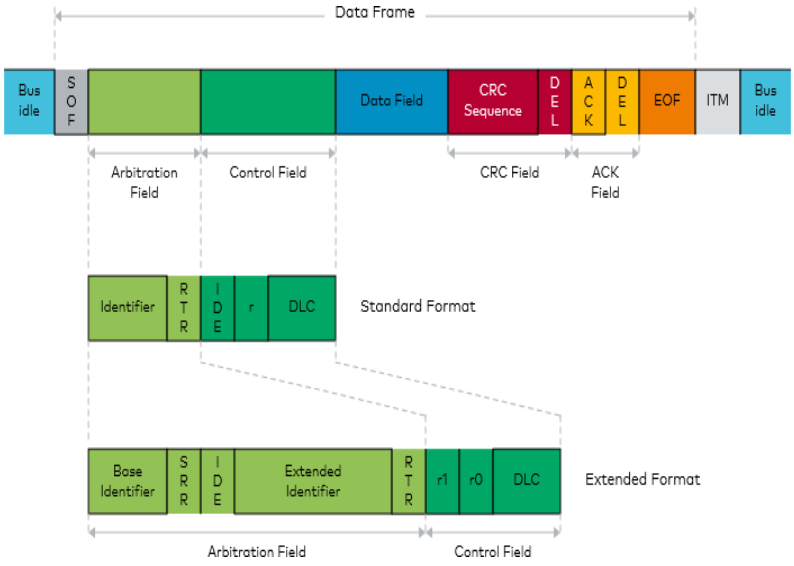


Fig. CAN data frame

2.1.3 STM32F407VG to ESP8266

Firstly, we have establish UART communication between the two devices. First, physically connect the UART TX pin of the STM32F407VG to the UART RX pin of the ESP8266, and vice versa. Then, we have configure the UART peripherals on both devices with compatible parameters. Written code in the STM32F407VG firmware to format and transmit data over UART, and code in the ESP8266 firmware to receive and process the data. Implement error handling and flow control mechanisms to ensure reliable communication, and thoroughly test the system to verify proper data transmission and reception. By doing these steps, you we have effectively transfer data between the STM32F407VG and ESP8266.

2.2Physical Connection

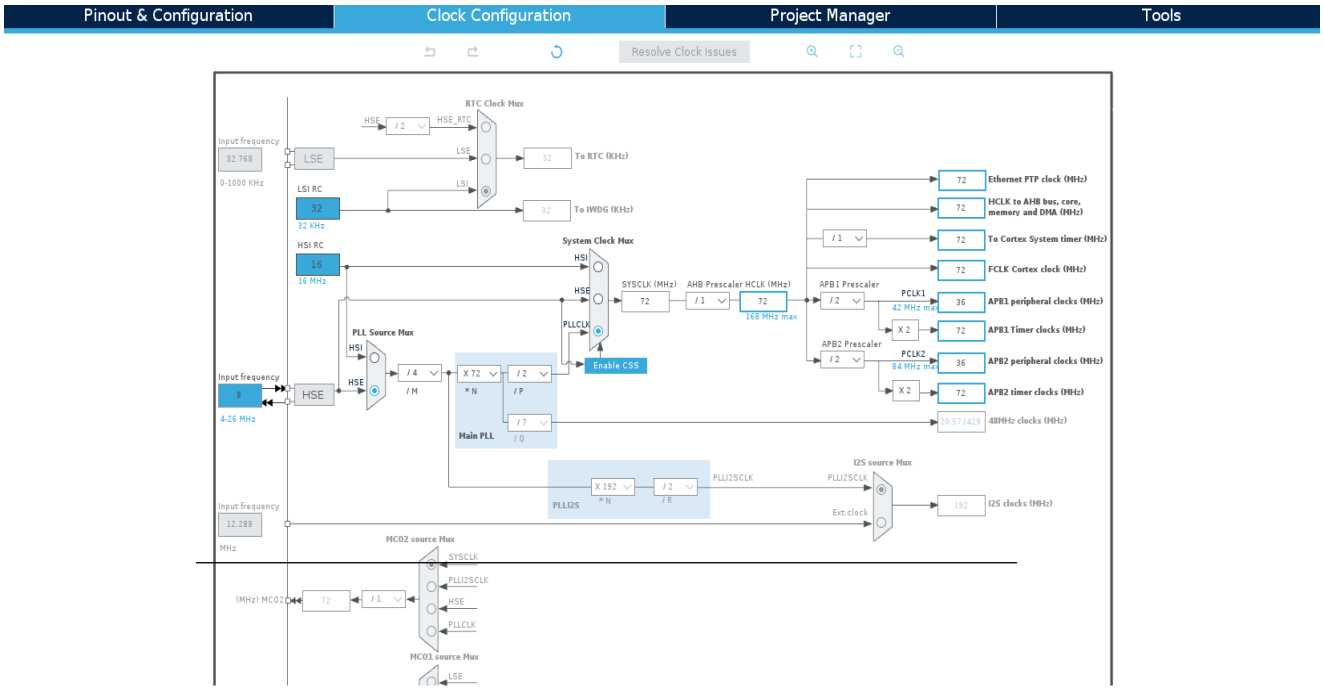
The STM32F407 has built-in CAN controller, CAN1 is connected on Port B. CAN transceiver is connected to CANcontroller pins to complete the physical layer of CAN node. Connection are as follows.

SN65HVD230	F407 discovery board
3.3v	3V3
GND	Ground
CTX	PB9 (Port B Pin 9)
CRX	PB8 (Port B Pin 8)

CANH and CANL of transceiver is connected to the CANH and CANL of the other transceiver connected to STM32F407VG respectively with a twisted pair of wire.

3. Clock Configuration

ST has provided three different clock sources can be used to drive the system clock (SYSCLK):
• HSI oscillator clock
• HSE oscillator clock
• Main PLL (PLL) clock



4.Code

4.1 CAN Transmitter code

```
        /* USER CODE BEGIN Header */
/**

******************************************************************************
***
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
***
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *

******************************************************************************
***
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include<string.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */
```

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;

CAN_HandleTypeDef hcan1;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CAN1_Init(void);
static void MX_ADC1_Init(void);
static void MX_ADC2_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
int Switch_Flag = 0;
CAN_TxHeaderTypeDef TxHeader;
uint8_t TxData[5];
uint32_t TxMailBox;
uint8_t adcValue = 0;
uint8_t adcValue1 = 0;

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
Switch_Flag = 1;
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
```

```
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
//                                          char str1[64] = "Temp\r\n";
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_CAN1_Init();
  MX_ADC1_Init();
  MX_ADC2_Init();
  /* USER CODE BEGIN 2 */
  HAL_CAN_Start(&hcan1);
  HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING);
  HAL_Delay(100);
  HAL_Delay(100);

  TxHeader.DLC=3;
  TxHeader.RTR=CAN_RTR_DATA;
  TxHeader.IDE=CAN_ID_STD;
  TxHeader.ExtId=0x0;
  TxHeader.StdId=0x0AA;
  TxHeader.TransmitGlobalTime=DISABLE;
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  if(Switch_Flag == 1){
 HAL_ADC_Start(&hadc1);
```

```
HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
int value = HAL_ADC_GetValue(&hadc1);
   adcValue = value * 0.08;
HAL_ADC_Start(&hadc2);

HAL_ADC_PollForConversion(&hadc2, HAL_MAX_DELAY);
adcValue1 = HAL_ADC_GetValue(&hadc2);
//sprintf(str,"%d\r\n",adcValue);
 TxData[0] = (uint8_t)( adcValue>>8);
//sprintf(str1,"%d\r\n",adcValue);
   TxData[1] = (uint8_t)( adcValue);

   TxData[2] = (uint8_t)( adcValue1>>8);
   TxData[3] = (uint8_t)( adcValue1);

//HAL_UART_Transmit(&huart2, (uint8_t *)str, strlen(str), HAL_MAX_DELAY);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);
   HAL_Delay(100);
   HAL_ADC_Stop(&hadc1);
    HAL_ADC_Stop(&hadc2);
  //TxData[1] = adcValue & 0xFF;

HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailBox);

HAL_Delay(1000);
   /* USER CODE END WHILE */

   /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE
1);

  /** Initializes the RCC Oscillators according to the specified parameters
```

```
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 72;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */
```

```
  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and
number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its corresponding rank in the sequencer
and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief ADC2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC2_Init(void)
{

  /* USER CODE BEGIN ADC2_Init 0 */

  /* USER CODE END ADC2_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};
```

```
  /* USER CODE BEGIN ADC2_Init 1 */

  /* USER CODE END ADC2_Init 1 */

  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and
number of conversion)
  */
  hadc2.Instance = ADC2;
  hadc2.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
  hadc2.Init.Resolution = ADC_RESOLUTION_12B;
  hadc2.Init.ScanConvMode = DISABLE;
  hadc2.Init.ContinuousConvMode = ENABLE;
  hadc2.Init.DiscontinuousConvMode = DISABLE;
  hadc2.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc2.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc2.Init.NbrOfConversion = 1;
  hadc2.Init.DMAContinuousRequests = DISABLE;
  hadc2.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc2) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its corresponding rank in the sequencer
and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_2;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC2_Init 2 */

  /* USER CODE END ADC2_Init 2 */

}

/**
  * @brief CAN1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_CAN1_Init(void)
{
```

```
  /* USER CODE BEGIN CAN1_Init 0 */

  /* USER CODE END CAN1_Init 0 */

  /* USER CODE BEGIN CAN1_Init 1 */

  /* USER CODE END CAN1_Init 1 */
  hcan1.Instance = CAN1;
  hcan1.Init.Prescaler = 18;
  hcan1.Init.Mode = CAN_MODE_NORMAL;
  hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan1.Init.TimeSeg1 = CAN_BS1_2TQ;
  hcan1.Init.TimeSeg2 = CAN_BS2_1TQ;
  hcan1.Init.TimeTriggeredMode = DISABLE;
  hcan1.Init.AutoBusOff = DISABLE;
  hcan1.Init.AutoWakeUp = DISABLE;
  hcan1.Init.AutoRetransmission = DISABLE;
  hcan1.Init.ReceiveFifoLocked = DISABLE;
  hcan1.Init.TransmitFifoPriority = DISABLE;
  if (HAL_CAN_Init(&hcan1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN CAN1_Init 2 */
  CAN_FilterTypeDef canFilterConfig;
   canFilterConfig.FilterActivation=CAN_FILTER_ENABLE;
   canFilterConfig.SlaveStartFilterBank=14;
   canFilterConfig.FilterBank=2;
   canFilterConfig.FilterFIFOAssignment=CAN_RX_FIFO0;
   canFilterConfig.FilterScale=CAN_FILTERSCALE_32BIT;
   canFilterConfig.FilterMode=CAN_FILTERMODE_IDMASK;
   canFilterConfig.FilterMaskIdLow=0x0000;
   canFilterConfig.FilterMaskIdHigh=0xFF00;
   canFilterConfig.FilterIdLow=0x0000;
   canFilterConfig.FilterIdHigh=0x1500;
   HAL_CAN_ConfigFilter(&hcan1, &canFilterConfig);
  /* USER CODE END CAN1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
```

```
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD,
GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

  /*Configure GPIO pin : PA0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : PD12 PD13 PD14 PD15 */
  GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void HAL_CAN_TxMailbox0CompleteCallback(CAN_HandleTypeDef *hcan)
{
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);
}


/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
```

```
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

4.2 CAN Receiver Code

```
    /* USER CODE BEGIN Header */
/**

******************************************************************************
***
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
***
  * @attention
  *
```

```
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *

******************************************************************************
***
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include<stdio.h>
#include<string.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
CAN_HandleTypeDef hcan1;
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CAN1_Init(void);
static void MX_USART2_UART_Init(void);
```

```
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
CAN_RxHeaderTypeDef RxHeader;
uint8_t RxData[10];
uint8_t flag =0;
char str[20];
char str1[20];


void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan){


HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &RxHeader, RxData);
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET); // blue
flag =1;

}


/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */
```

```
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_CAN1_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */
  HAL_CAN_Start(&hcan1);
  HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING);
  HAL_Delay(100);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  if(flag == 1){
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);

uint16_t receivedAdcValue = (uint16_t)(( RxData[0]<< 8) | RxData[1] );
sprintf(str,"%d\n",receivedAdcValue);
HAL_UART_Transmit(&huart2, (uint8_t *)str, strlen(str), HAL_MAX_DELAY);
HAL_Delay(1000);

uint16_t receivedAdcValue1 = (uint16_t)(( RxData[2]<< 8) | RxData[3] );


sprintf(str1,"%d\n",receivedAdcValue1);

HAL_UART_Transmit(&huart2, (uint8_t *)str1, strlen(str1), HAL_MAX_DELAY);

HAL_Delay(1000);
                                                    }


    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
```

```
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE
1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 72;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief CAN1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_CAN1_Init(void)
{
```

```
  /* USER CODE BEGIN CAN1_Init 0 */

  /* USER CODE END CAN1_Init 0 */

  /* USER CODE BEGIN CAN1_Init 1 */

  /* USER CODE END CAN1_Init 1 */
  hcan1.Instance = CAN1;
  hcan1.Init.Prescaler = 18;
  hcan1.Init.Mode = CAN_MODE_NORMAL;
  hcan1.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan1.Init.TimeSeg1 = CAN_BS1_2TQ;
  hcan1.Init.TimeSeg2 = CAN_BS2_1TQ;
  hcan1.Init.TimeTriggeredMode = DISABLE;
  hcan1.Init.AutoBusOff = DISABLE;
  hcan1.Init.AutoWakeUp = DISABLE;
  hcan1.Init.AutoRetransmission = DISABLE;
  hcan1.Init.ReceiveFifoLocked = DISABLE;
  hcan1.Init.TransmitFifoPriority = DISABLE;
  if (HAL_CAN_Init(&hcan1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN CAN1_Init 2 */
  CAN_FilterTypeDef canFilterConfig;
  canFilterConfig.FilterActivation=CAN_FILTER_ENABLE;
  canFilterConfig.SlaveStartFilterBank=14;
  canFilterConfig.FilterBank=2;
  canFilterConfig.FilterFIFOAssignment=CAN_RX_FIFO0;
  canFilterConfig.FilterScale=CAN_FILTERSCALE_32BIT;
  canFilterConfig.FilterMode=CAN_FILTERMODE_IDMASK;
  canFilterConfig.FilterMaskIdLow=0x0000;
  canFilterConfig.FilterMaskIdHigh=0xFF00;
  canFilterConfig.FilterIdLow=0x0000;
  canFilterConfig.FilterIdHigh=0x1500;
  HAL_CAN_ConfigFilter(&hcan1, &canFilterConfig);
  /* USER CODE END CAN1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */
```

```
  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 9600;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD,
GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

  /*Configure GPIO pins : PD12 PD13 PD14 PD15 */
  GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

4.3 Node MCU(WIFI-module)Code

```
  #include<ESP8266WiFi.h>
#include<WiFiClient.h>
#include<ESP8266HTTPClient.h>

const int bufferSize = 64;  // Adjust the size based on your requirement
char inputBuffer[bufferSize];

void setup() {
  Serial.begin(9600);  // Set the baud rate to match the Serial Monitor
}

void loop() {
  if (Serial.available() > 0) {
    // Read the incoming bytes until a newline character is encountered
    int bytesRead = Serial.readBytesUntil('\n', inputBuffer, bufferSize - 1);

    // Null-terminate the string
    inputBuffer[bytesRead] = '\0';

    // Print the received string to the Serial Monitor
    Serial.print("Received: ");
    Serial.println(inputBuffer);
  }
}
```

4.4 Server Code

```
  #include<ESP8266WiFi.h>
#include<WiFiClient.h>
#include<ESP8266HTTPClient.h>


const char *ssid = "Sanket";
const char *password = "sanket@123";

const int bufferSize=64;
char inputBuffer[bufferSize];
char inputBuffer1[bufferSize];

void setup() {
```

```
  // put your setup code here, to run once:
  Serial.begin(9600);
  //Serial.flush();

  //WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.println("Connecting to WiFi");
  while(WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
   }
  Serial.println("Serial Setup is completed");
  Serial.println("WiFi is connected");
  Serial.print("IP Address : ");
  Serial.println(WiFi.localIP());

  //pinMode(A0, INPUT);
}

void loop() {
  // put your main code here, to run repeatedly:

  if(Serial.available()>0){
    int bytesRead = Serial.readBytesUntil('\n',inputBuffer,bufferSize-1);
    inputBuffer[bytesRead]='\0';
    delay(1000);
     bytesRead = Serial.readBytesUntil('\n',inputBuffer1,bufferSize-1);
    inputBuffer1[bytesRead]='\0';
    Serial.print("Received1:");
    Serial.println(inputBuffer);
    Serial.print("\nReceived2 :");
     Serial.println(inputBuffer1);
 //float temp = atof(inputBuffer);
//ring body = String(temp);

 String body = "{ \"temperature\":"+String(inputBuffer)+",\"rain\":"+String(inputBuffer1)+"
}";

    Serial.println(body);
  HTTPClient httpclient;
  WiFiClient wificlient;
  httpclient.begin(wificlient, "http://192.168.47.254:5000/monitor");
httpclient.addHeader("content-type", "application/json");
  int statuscode = httpclient.POST(body);
  Serial.printf("Status Code : %d\n", statuscode);
  delay(5000);
  }
}
```

5.Future scope

The future scope of the proposed monitoring and analysis system for electric vehicles is vast, with opportunities for innovation and improvement in various aspects of vehicle monitoring, maintenance, and optimization. Continued research and development in these areas could lead to significant advancements in electric vehicle technology and contribute to the broader goals of sustainability and energy efficiency in transportation.

5.References

1."Controller Area Network (CAN) Overview - National Instruments," by National Instruments. This provides a comprehensive overview of the CAN protocol, its applications, and implementation details. [Link: https://www.ni.com/en-us/innovations/white-papers/05/controller-area-network--can--overview.html]

2."Introduction to the Internet of Things: Definition and Market Overview," by Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. This paper offers insights into IoT concepts and its applications in various domains. [Link: https://ieeexplore.ieee.org/document/8093055]

3."Smart Vehicles and The Internet of Things," by Michael Chui, Markus Löffler, and Roger Roberts. This article explores the integration of IoT in vehicles and its impact on transportation systems. [Link: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/smart-vehicles-and-the-internet-of-things]

4."Vehicle Health Monitoring and Fault Diagnosis Based on CAN Bus," by Zhou, Z., & Zeng, Q. This research paper focuses on vehicle health monitoring using CAN bus data. [Link: https://ieeexplore.ieee.org/document/7169200]

5."Integration of CAN Bus and Internet of Things for Urban Traffic Monitoring," by U. P. Martinez, E. Silva, and F. V. Paulovich. This paper discusses the integration of CAN bus with IoT for urban traffic monitoring. [Link: https://www.sciencedirect.com/science/article/pii/S1877050916317190]

6."IoT Applications for Smart and Connected Vehicles: A Review," by Amin Bashi, R., Rashid, R. A., & Yau, K. L. A comprehensive review of IoT applications in smart and connected vehicles. [Link: https://www.mdpi.com/1424-8220/17/6/1256]

7."Real-time Data Acquisition and Analysis in Automotive Systems Using IoT," by

Francisco de Oliveira Neto, Paulo Cortez, and Eduardo A. B. da Silva. This paper discusses real-time data acquisition and analysis in automotive systems through IoT technologies. [Link: https://www.sciencedirect.com/science/article/pii/S2405959521001445]

8."Automotive Cybersecurity: A Systematic Review of Recent Trends and Future Directions," by F. Mirzadeh, A. Dehghantanha, and K.-K. R. Choo. This review paper highlights the importance of cybersecurity in automotive systems, including those utilizing CAN and IoT technologies. [Link: https://www.sciencedirect.com/science/article/pii/S0167404820302201]

9."Implementation of an IoT-Based Vehicle Tracking System," by Oluwaseyi Feyisetan, S. P. Adeniyi, and M. O. Adigun. This paper presents an implementation of an IoT-based vehicle tracking system, which could provide valuable insights for your project. [Link: https://ieeexplore.ieee.org/document/7740461]

10."Internet of Things (IoT) in Transportation: Research Challenges and Opportunities," by Sherali Zeadally, Ray Hunt, Yuh-Shyan Chen, Angela Irwin-Hunt, and Aftab Ahmad. This article discusses research challenges and opportunities in applying IoT technologies to transportation systems, including vehicle analysis and monitoring. [Link: https://ieeexplore.ieee.org/document/7503195]