Project report on

Driver Alert System Using CAN

Guided by

Mr. Ankush Tembhurnikar

Presented by:

Vishal Gurav	**240844230095**
Shreya Ghewade	**240844230027**
Pranav Sangar	**240844230073**
Tejas Shirke	**240844230088**

At

Sunbeam Institute Of Information Technology,

Hinjewadi

SUNBEAM INSTITUTE OF INFORMATION TECHNOLOGY, HINJEWADI.

This is to certify that the project

Driver Alert System Using CAN

Has been submitted by

Vishal Gurav	**240844230095**
Shreya Ghewade	**240844230027**
Pranav Sangar	**240844230073**
Tejas Shirke	**240844230088**

In partial fulfillment of the requirement for the course of **PG Diploma In Embedded System Design (PG-DESD AUG-2024)** as prescribed by the **CDAC** ACTS, PUNE.

Place : Hinjewadi Date :

ACKNOWLEDGEMENT

This project " **Driver Alert System using CAN** " was a great learning experience for us and we are submitting this work to SUNBEAM (CDAC).
We all are very glad to mention the name of *Mr.Ankush Tembhurnikar* for his valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.
Our most heartfelt thanks goes to *Mr. Sohail Inamdar* (Course Coordinator, PG-*DESD*) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility and extra Lab hours to complete the project and throughout the course up to the last day here in C-DAC SUNBEAM,Pune.

ABSTRACT

This system is an attempt to analysis driver alert system. Implementation of controller area network (CAN) integrated with STM32F407 Microcontroller for communication between two nodes. CAN is the most versatile and popular technology of electronics industry which are used in many practical application.

Controller area network is a serial asynchronous , multi-master communication protocol for connection of electronics derives and control modules in automotive and industrial applications .CAN was designed for automotive applications needing high level of data integrity and data rates upto 1Mbits/sec.

The project flow consist of continuous monitoring of temperature sensor and ultrasonic sensor value which is interfaced with microcontroller .In this system the signal information like temperature(LM35 sensor) if temperature increase above 60 degree Celsius and ultrasonic sensor is adapted to measure the distance between the obstacle and vehicle, if obstacle is detected close; the data of distance and temperature continuously displayed on LCD. If temperature and distance crossed certain limit then buzzer gives signal.

TABLE OF INDEX

Topic Page No.

1] Introduction

This system helps in achieving effective communication between transmitter and receiver modules using CAN protocol with multiple sensors to monitor the various parameters and visualize them to the vehicle driver through a LCD display .The CAN modules interfaced with the sensors for this system are temperature sensor capable of detecting engine heat, ultrasonic sensor for detecting the distance between obstacles. This is important that human drivers control over the vehicle and check the parameters in vehicle on LCD screen at the same time of driving, parameters like engine temperature and obstacle's distance. CAN protocol (bus) are used for data transmission. A CPU is needed to manage the CAN protocol. The STM32F407 microcontroller is used as the CPU that can manage bus arbitration, assigning priority for the message addressing and identification. For the CAN bus– based designs it is easier to use a STM32F407 microcontroller with a built-in CAN module, such devices include built-in CAN controller hardware on the chip. For implementation of this digital circuitry need a different component the main part for controlling all information to check working for this purpose use a processor for the sensing purpose use a temperature sensor, obstacle detection sensor and power supply are main parts.

1] Literature survey

1] ARM- CORTEX M4 Architecture

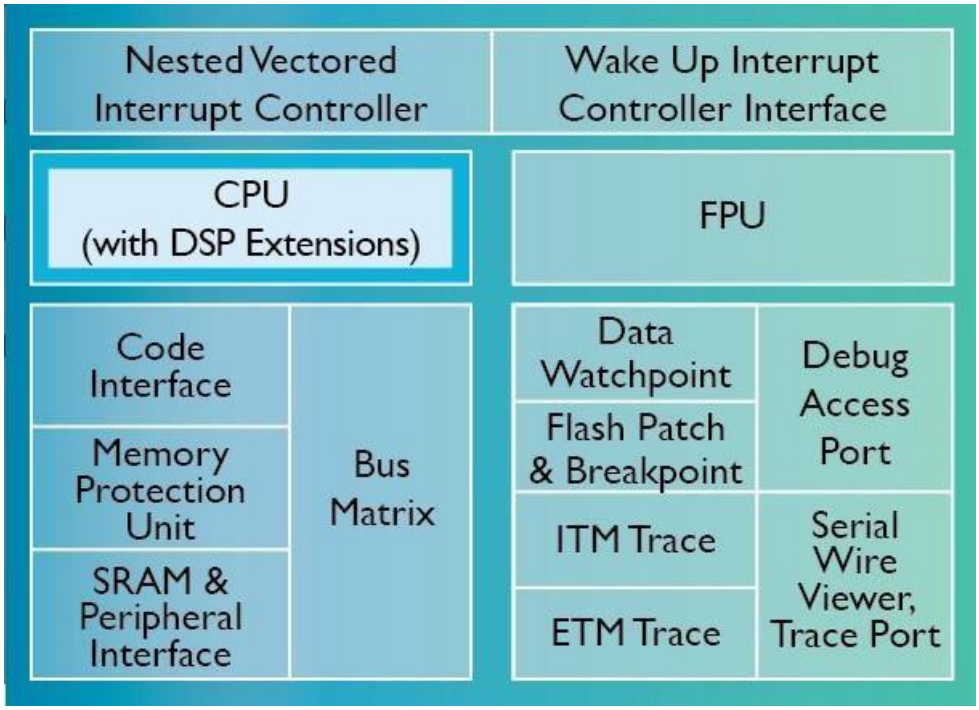


Fig. 1.1 ARM CORTEX M4 Architecture

The ARM cortex-M4 processor is a high performance embedded processor with DSP instructions developed to 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 implementation from those requiring floating point operations, memory protection and powerful trace technology to cost sensitive devices requiring minimal area.

Key benefits:

1. Gain the advantages of a microcontroller with integrated DSP, SIMD, and MAC instructions that simplify overall system design, software development and debug

2. Accelerate single precision floating point math operations up to 10x over the equivalent integer software library with the optional floating point unit (FPU)

3. Develop solutions for a large variety of markets with a full-featured ARMv7-M instruction set that has been proven across a broad set of embedded applications

4. Achieve exceptional 32-bit performance with low dynamic power, delivering leading system energy efficiency due to integrated software controlled sleep modes, extensive clock gating and optional state retention.

2] Controller Area Network

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 through the use of prioritized message IDs.

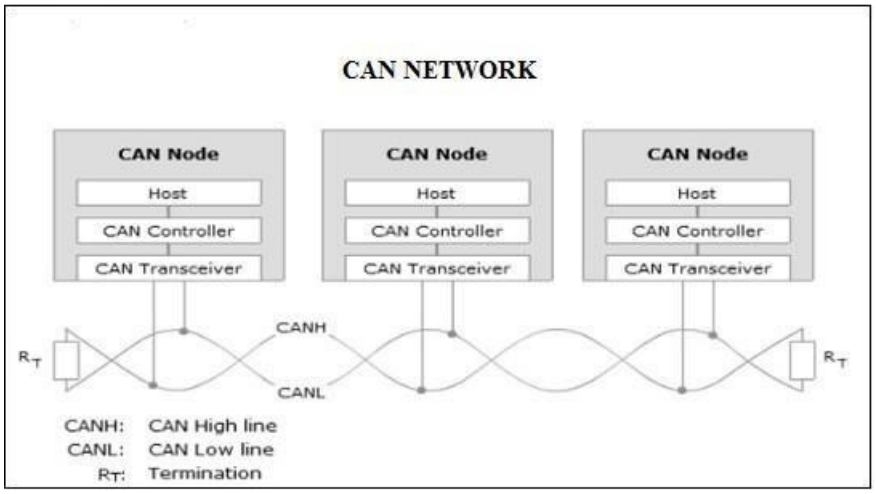


Fig. 2.1 CAN network

2] CAN Bus

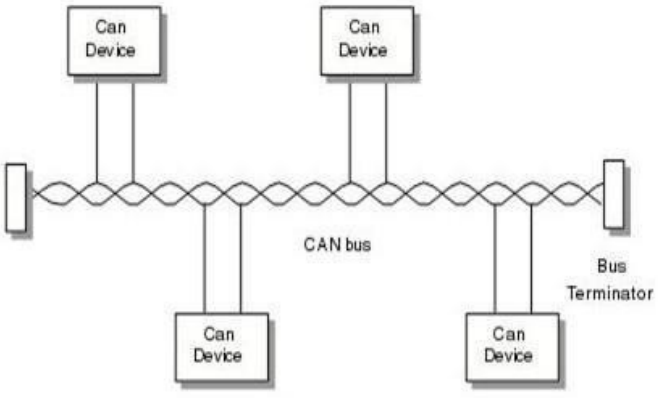


Fig .2.2 . CAN Bus

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. 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.

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.

3] CAN bus levels

Physical signal transmission in a CAN network is based on differential signal transmission. 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 typical differential voltage of 2 Volt. High-speed CAN 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.

High Speed CAN:

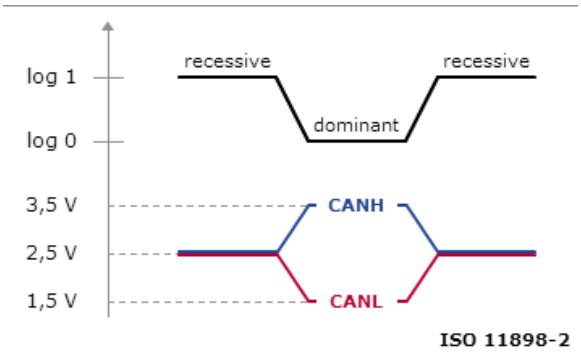


fig 2.3. high speed CAN voltage levels

Low Speed CAN:

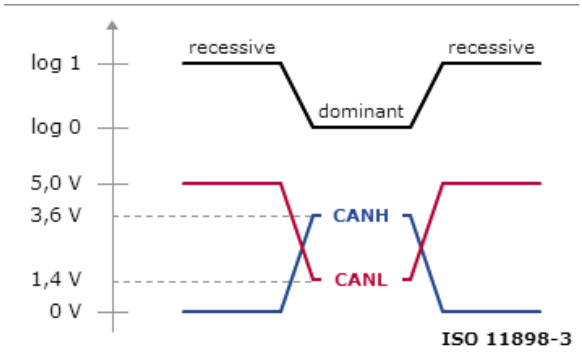
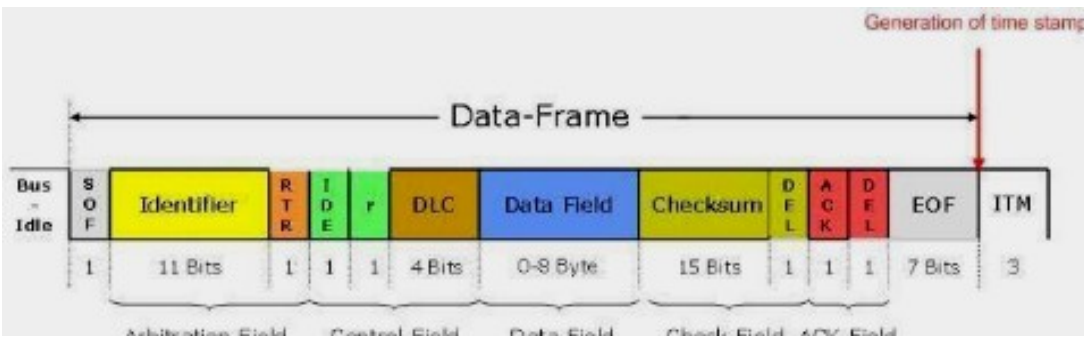
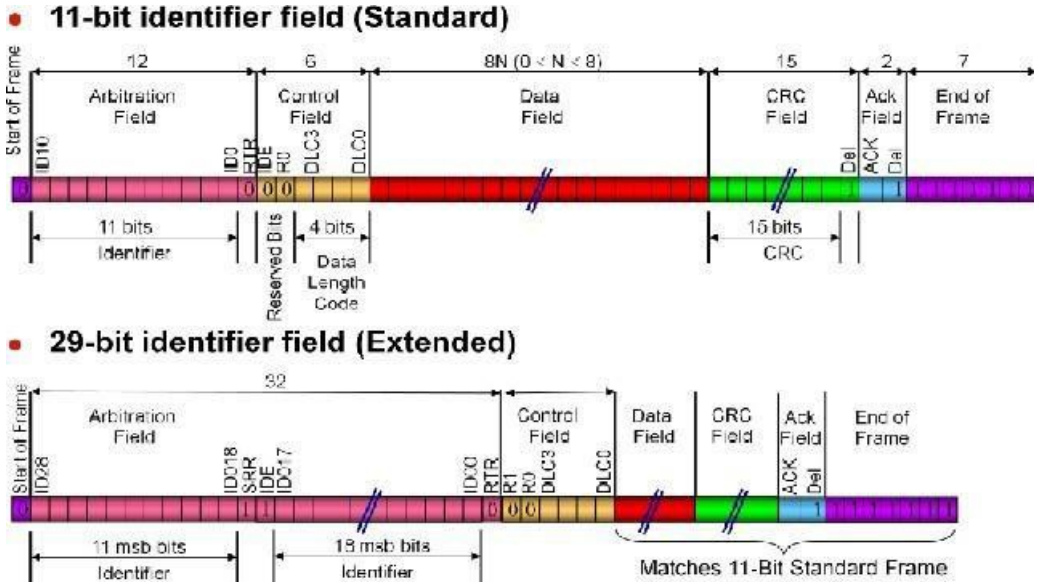


Figure 2.4 Low speed CAN voltage levels

4] CAN Frames

1. **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 of data, and the CRC Field which contains the 16-bit checksum used for error detection. Last is the Acknowledgment Field.





data frame fig. 2.5

2. 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.

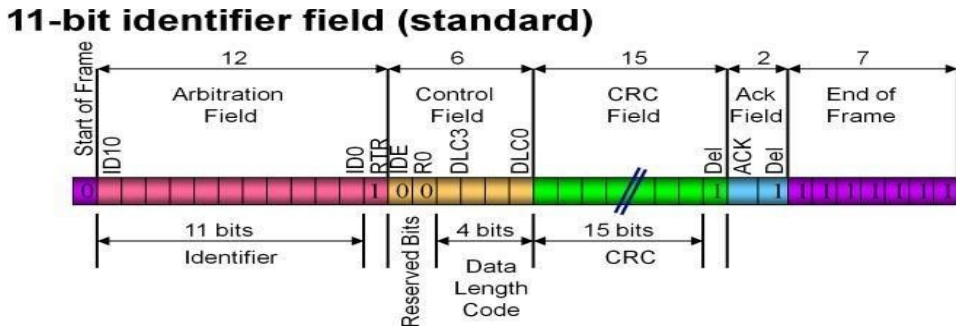


Fig. 2.6 remote frame

3. 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.

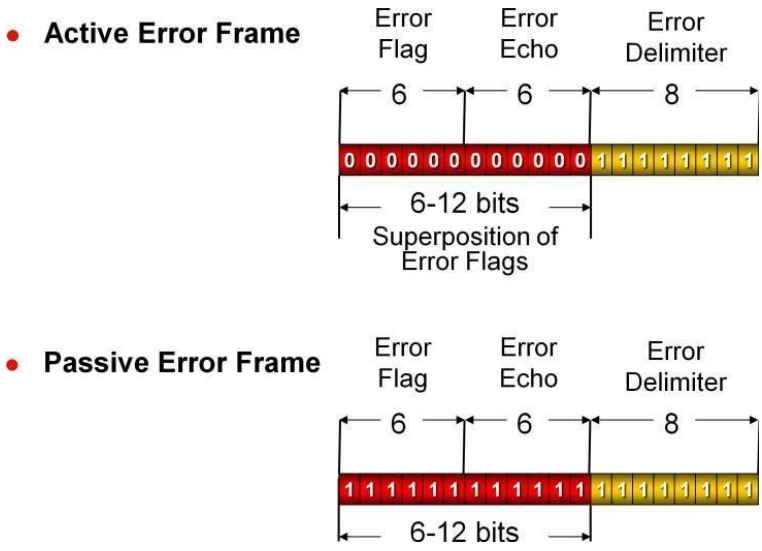


Fig 2.7 error frame

4. Overload frame
   The overload frame is mentioned for completeness. It is similar to the error frame with regard to 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.

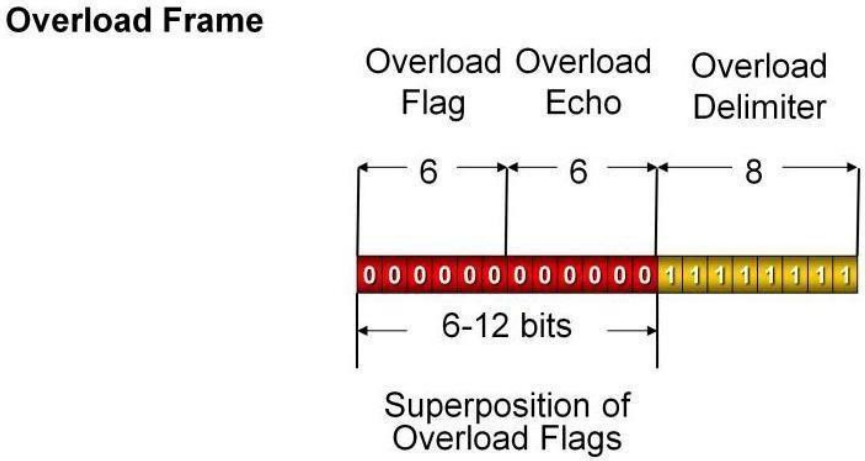


Fig.2.8. overload frame

5] 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 mean 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 the bus. If these levels are equal the unit may continue to send. When a "recessive" level is sent and a "dominant" level is monitored, the unit has lost arbitration and must withdraw without sending one more bit.

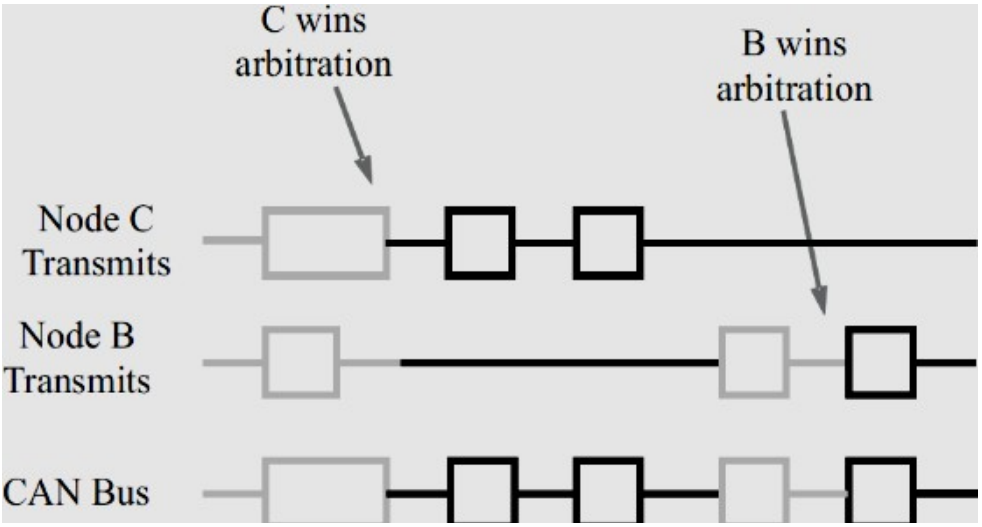


fig. 2.9. Bus arbitration

3] Requirements

1] Hardware Requirements

1.1] **STM32F407 Discovery Board**

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 72 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/Os 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 3.6 V power supply. A comprehensive set of power-saving mode allows the design of low-power applications.

1.2] LM35 temperature sensor:

- A temperature sensor is used for sensing the temperature of the environment and the system displays the temperature on an lcd.
- The LM35 series are precision integrated temperature sensor.
- Features-
  Calibrated Directly in Celsius (Centigrade)
  - Linear + 10-mV/°C Scale Factor
  - 0.5°C Ensured Accuracy (at 25°C)
  - Rated for Full -55°C to 150°C Range
  - Suitable for Remote Applications
  - Low-Cost Due to Wafer-Level Trimming
  - Operates From 4 V to 30 V
  - Less Than 60-µA Current Drain
  - Low Self-Heating, 0.08°C in Still Air
  - Non-Linearity Only ±¼°C Typical
  - Low-Impedance Output, 0.1 Ω for 1-mA Load

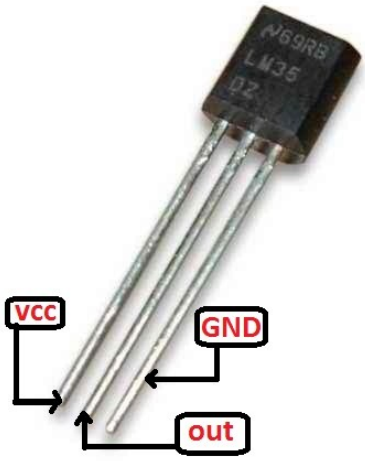


fig3.1. LM35.

1.3] HCSR04 Ultrasonic Sensor:

- These sensors generate high frequency sound waves and calculate the time interval between sending the signal and receiving the echo to determine the distance to an object.
- Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

(1) Using IO trigger for at least 10us high level signal,

(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.

(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.

Test distance = (high level time ×velocity of sound (340M/S)) / 2.

Fig 1.4. ultrasonic sensor

1.4] I2C serial interface 20 x 4 LCD module :

- This is I2C interface 20x4 LCD display module, a new high-quality 4 line 20 character LCD module with on-board contrast control adjustment, backlight and I2C communication interface.

Features :

- Display Type: Black on yellow green backlight. 20 x 4 display format .
- I2C Address:0x38-0x3F (0x3F default)
- Supply voltage: 5V
- Interface: I2C to 4bits LCD data and control lines.
- Contrast Adjustment : built-in Potentiometer.

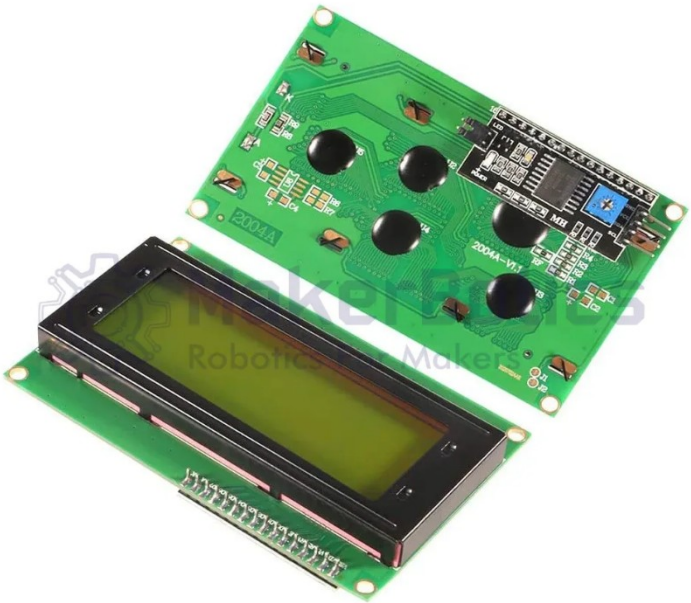


Fig .1.4]. LCD display with I2C interface

1.5] MCP2551(CAN transreceiver)

• Implements ISO-11898 standard physical layer requirements

• Supports 1 Mb/s operation

• Suitable for 12V and 24V systems

• Externally-controlled slope for reduced RFI emissions

• Detection of ground fault (permanent dominant) on TXD input

• Power-on reset and voltage brown-out protection

• An unpowered node or brown-out event will not disturb the CAN bus

• Low current standby operation

• Protection against damage due to short-circuit conditions (positive or negative battery voltage)

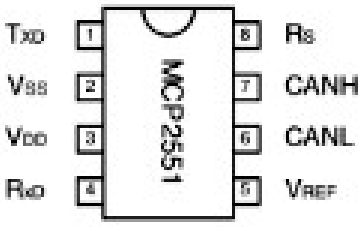
• Protection against high-voltage transients

• Automatic thermal shutdown protection

• Up to 112 nodes can be connected

• High noise immunity due to differential bus implementation

• Temperature ranges: - Industrial (I): -40°C to +85°C - Extended (E): -40°C to +125°C



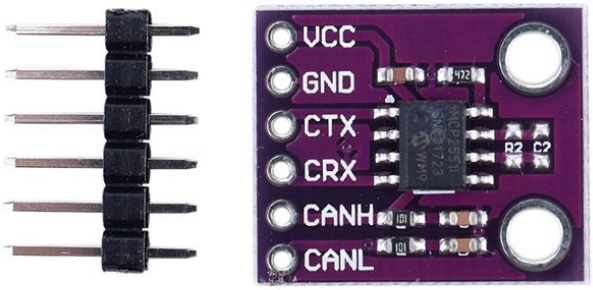


Figure 1.5 MCP2251 Module

1.6] Buzzer

This is an audio signaling device used at receivers side of system in this project.

Features :

- Diameter is 38mm Height is 22mm 2 Mounting holes distance:40mm 2 wires.
- rate voltage: 12v DC.
- operating voltage: 3 - 24v
- Buzzer type: piezoelectric sound pressure level 95

fig. 1.5. Buzzer

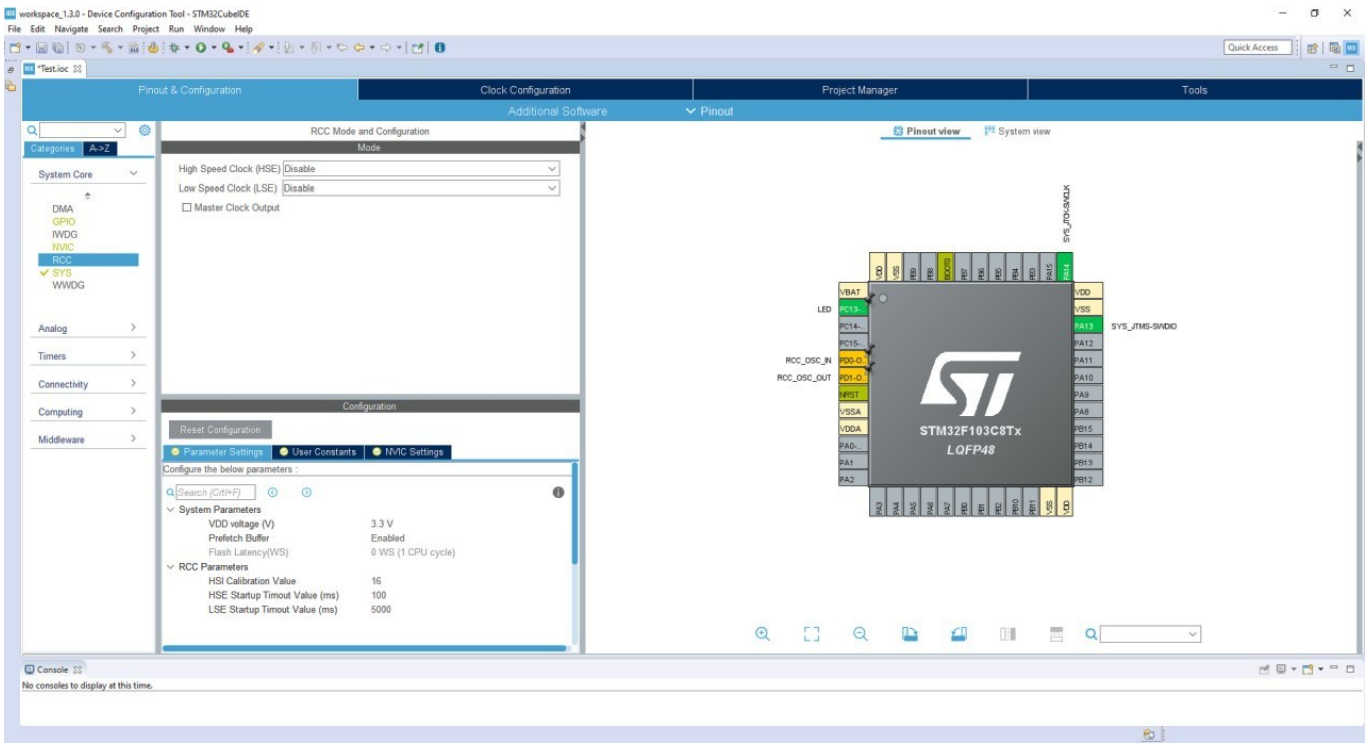
2] Software Requirements

2.1] STM32CubeIDE

STM32CubeIDE is an all-in-one multi-OS development tool, which is part of the STM32Cube software ecosystem.

STM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors. It is based on the Eclipse®/CDT™ framework and GCC toolchain for the development, and GDB for the debugging. It allows the integration of the hundreds of existing plugins that complete the features of the Eclipse® IDE. STM32CubeIDE integrates STM32 configuration and project creation functionalities from STM32CubeMX to offer all-in-one tool experience and save installation and development time. After the selection of an empty STM32 MCU or MPU, or preconfigured microcontroller or microprocessor from the selection of a board or the selection of an example, the project is created and initialization code generated..

fig 2.1. STM32 cube IDE interface



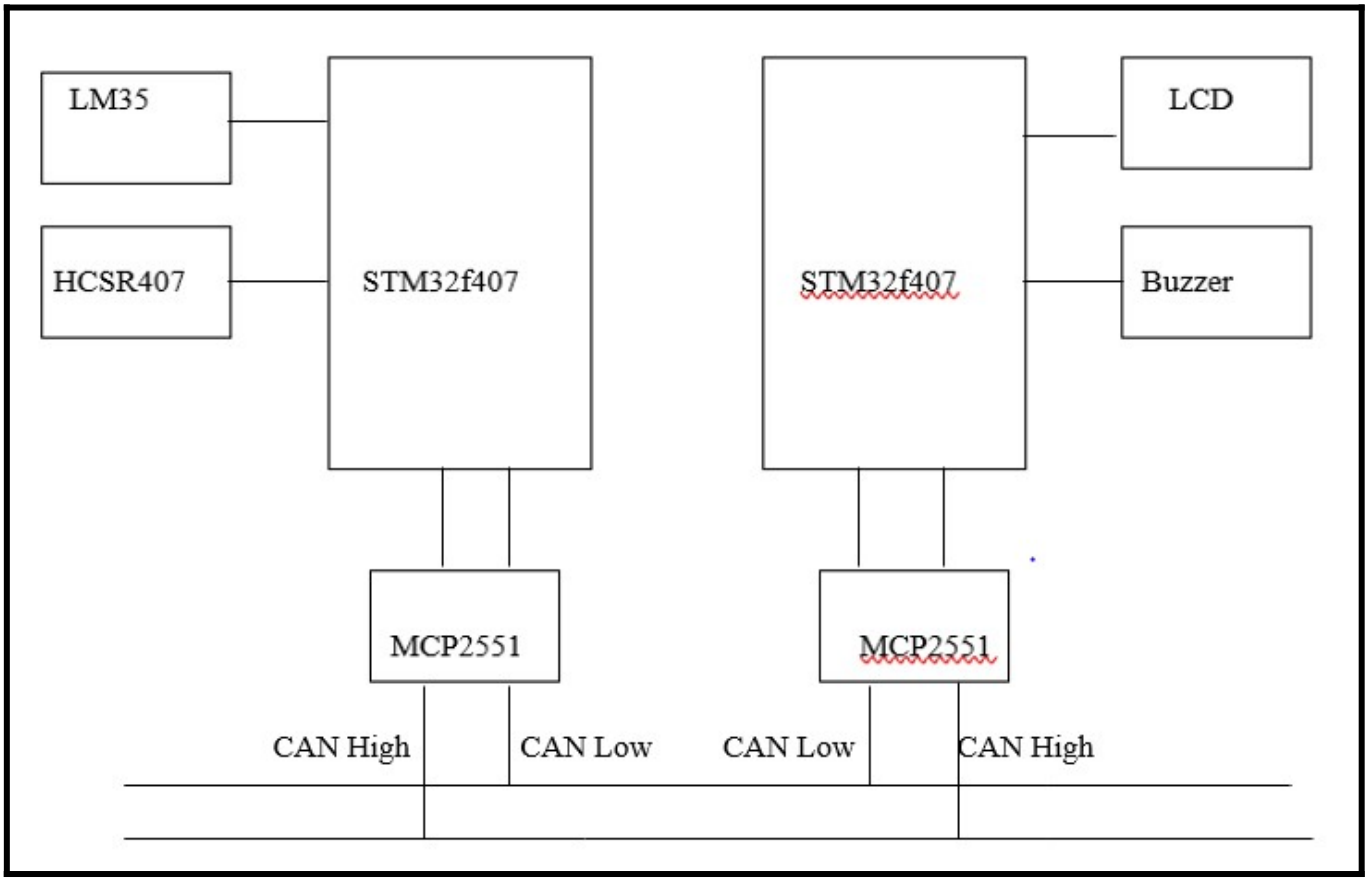
At any time during the development, the user can return to the initialization and configuration of the peripherals or middleware and regenerate the initialization code with no impact on the user code.

STM32CubeIDE includes build and stack analysers that provide the user with useful information about project status and memory requirements. STM32CubeIDE also includes standard and advanced debugging features including views of CPU core registers, memories, and peripheral registers, as well as live variable watch, Serial Wire Viewer interface, or fault analyser.

Features :

- Integration of services from STM32CubeMX:STM32 microcontroller, microprocessor, development platform and example project selection Pinout, clock, peripheral, and middleware configuration Project creation and generation of the initialization code Software and middleware completed with enhanced STM32Cube Expansion Packages.
- Based on Eclipse®/CDT™, with support for Eclipse® add-ons, GNU C/C++ for Arm® toolchain and GDB debugger
- STM32MP1 Series: Support for Open ST Linux projects: LinuxSupport for Linux
- Additional advanced debug features including:CPU core, peripheral register, and memory viewsLive variable watch viewSystem analysis and real-time tracing (SWV)CPU fault analysis tool RTOS-aware debug support including Azure.
- Support for ST-LINK (STMicroelectronics) and J-Link (SEGGER) debug probes
- Multi-OS support: Windows®, Linux®, and macOS®, 64-bit versions only

4] Project Architecture And Flowchart



4.1] Block Diagram

4.2] Description

As per block diagram here we are using two STM32F407 discovery board, one is used as a transmitter and another one is used as a receiver.

It uses a temperature sensor to detect engine heat and ultrasonic sensor to detect the distance between object and vehicle using STM32F407 MicroController.

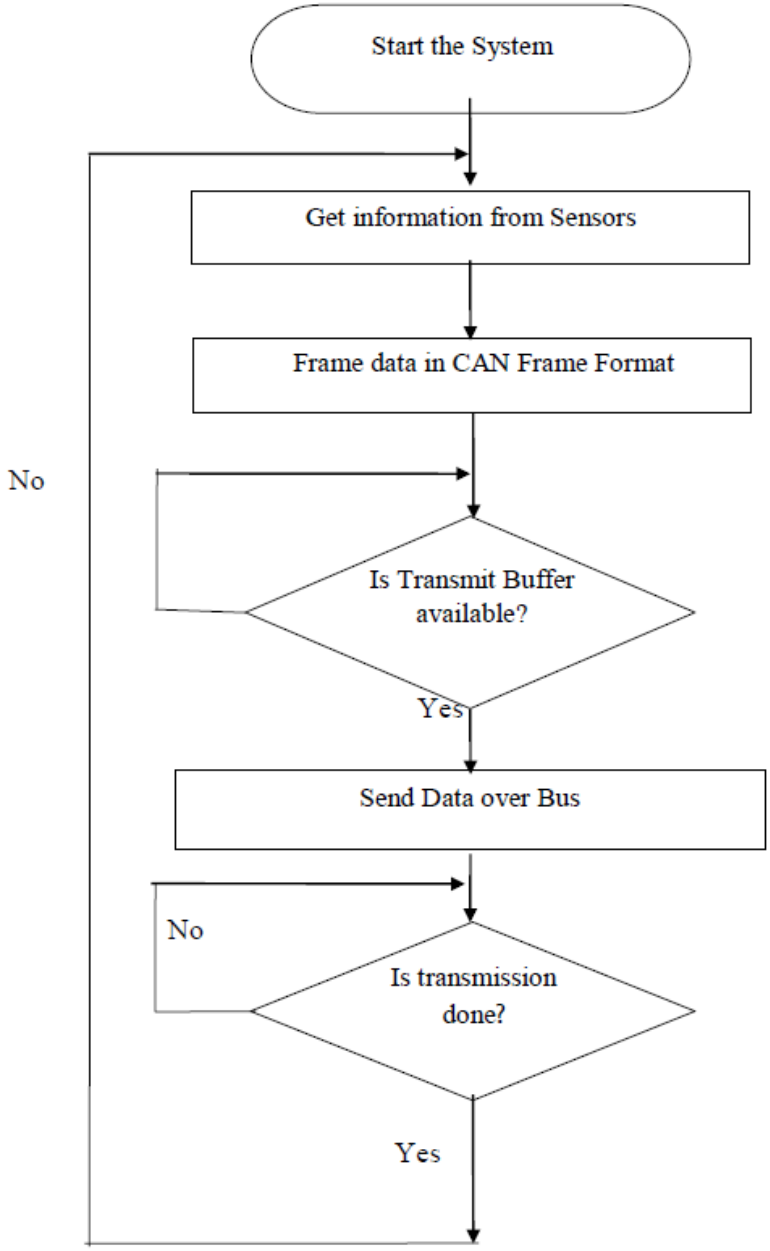
On transmitting end STM32F407 MicroController take data of temperature sensor and ultrasonic sensor then this data will be transmitted to the receiver's end.

This system uses CAN protocol for communication between sensors and display /alert system at the other end.

Driver alert system transmitting and receiving end. Transmitter end consists of inbuilt CAN module in discovery board and transreceiver (MCP2551). LM35 (temperature sensor) is connected to A0 pin at receivers side and PA0 and PA1 pins are connected to ultrasonic sensor (HCSR407). PB9 and PB8 (CAN tx

and CAN rx pins) are connected to MCP 2551 Rx and Tx end. Both transreceivers are connected together with connection of CAN high and CAN low connections. At the receivers end we I2C module of LCD is connected with PB6 for SDL signal and PB7 for SDA signal. For buzzer we have used PB14 as output pin.

Flow Chart :-



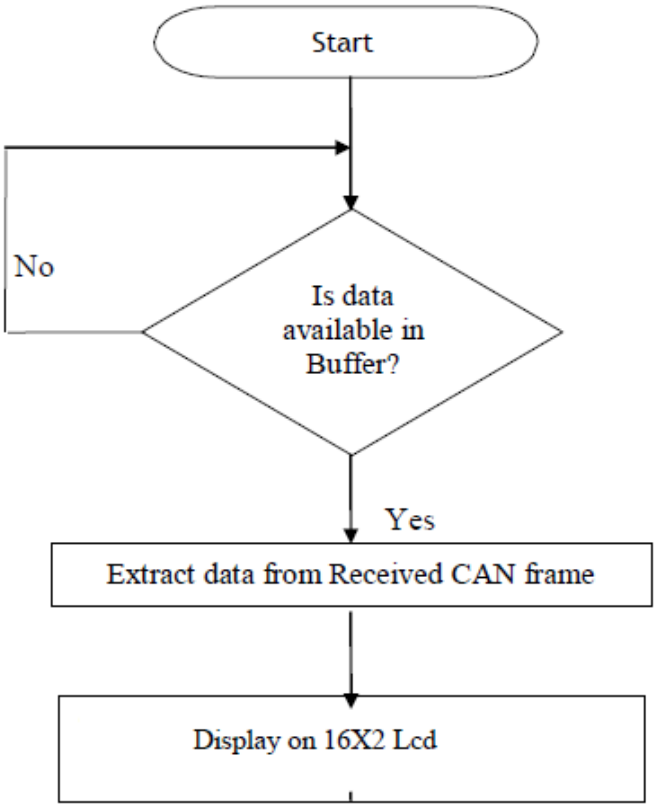


Figure 4.3 Node B Flowchart

Fig . 4.3 Node A flowchart

5] Source Code Explanation

5.1] transmitter node source code :

```
uint32_t IC_Val1 = 0;
uint32_t IC_Val2 = 0;
uint32_t Difference = 0;
uint8_t Is_First_Captured = 0;  // is the first value captured ?
uint8_t Distance  = 0;
uint32_t adc = 0;
int Temp = 0;

#define TRIG_PIN GPIO_PIN_11
#define TRIG_PORT GPIOE
```

- We have defined input output pins for trigger and echo port of ultrasonic module. Some variables for further calculations have initialized.

```
CAN_TxHeaderTypeDef TxHeader;
uint8_t LM35;
volatile int Switch_Flag = 0;
uint8_t TxData[5];
uint32_t TxMailBox;
void delay (uint16_t time)
{
    __HAL_TIM_SET_COUNTER(&htim1, 0);
    while (__HAL_TIM_GET_COUNTER (&htim1) < time);
}
```

- Data frame of 8 bit for transmitting through CAN protocol is declared . we have declared 32 bit mailbox of CAN frame to transmit over bus

```
/* 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;

TxData[0] = 0;

HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_1);

/* USER CODE END 2 */
```

- CAN is activated for RX_FIFO notification, transmission data length is been set to 3, RTR bit indicating data transmission . IDE indicating standard (11 bit frame) and standard id is defined.

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)  // if the interrupt source is channel1
    {
        if (Is_First_Captured==0) // if the first value is not captured
        {
            IC_Val1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1); // read the first value
            Is_First_Captured = 1;  // set the first captured as true
            // Now change the polarity to falling edge
            __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_FALLING);
        }

        else if (Is_First_Captured==1)   // if the first is already captured
        {
            IC_Val2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);  // read second value
            __HAL_TIM_SET_COUNTER(htim, 0);  // reset the counter

            if (IC_Val2 > IC_Val1)
            {
                Difference = IC_Val2-IC_Val1;
            }

            else if (IC_Val1 > IC_Val2)
            {
                Difference = (0xffff - IC_Val1) + IC_Val2;
            }

            Distance = Difference * .034/2;
            Is_First_Captured = 0; // set it back to false

            // set polarity to rising edge
            __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_RISING);
            __HAL_TIM_DISABLE_IT(&htim1, TIM_IT_CC1);
        }
    }
}
```

We have trigger the HC-SR04 ultrasonic sensor and enable a timer interrupt for measuring the echo pulse duration on an STM32 microcontroller.

```
void HCSR04_Read (void)
{
    HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_SET);  // pull the TRIG pin HIGH
    delay(10);  // wait for 10 us
    HAL_GPIO_WritePin(TRIG_PORT, TRIG_PIN, GPIO_PIN_RESET);  // pull the TRIG pin lo

    __HAL_TIM_ENABLE_IT(&htim1, TIM_IT_CC1);
}

/* USER CODE END 0 */
```

- For distance calculation Trigger pin and echo Pin of ultrasonic sensor being modified in above code . Finally distance is calculated using formula with values of high and low echo wave detection.
-

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
      HAL_ADC_Start(&hadc1);
     HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
              LM35 = HAL_ADC_GetValue(&hadc1);
              HAL_ADC_Stop(&hadc1);
```

- ADC Count for LM35 readings assigned to corresponding LM35 variable using HAL_ADC_GetValue() function.

- This is another alternate function to get exact value of temperature by converting ADC counts to temperature.

```
197       HCSR04_Read();
198       TxData[0] = !TxData[0]; //LED
199       TxData[1] = LM35;
200       TxData[2] = Distance;
201       if (HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailBox) != HAL_OK)
202               Error_Handler();
203       HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
204       HAL_Delay(500);
205     /* USER CODE END WHILE */
206
207     /* USER CODE BEGIN 3 */
208   }
209   /* USER CODE END 3 */
210 }
```

- We used LED as successful data transmission indication at receiver side it will turn on the LED

- We are sending distance and temperature data through mailbox attaching it to TxHeader declared above .

```
340   /* USER CODE BEGIN CAN1_Init 2 */
341   CAN_FilterTypeDef canFilterConfig;
342     canFilterConfig.FilterActivation=CAN_FILTER_ENABLE;
343     canFilterConfig.SlaveStartFilterBank=14;
344     canFilterConfig.FilterBank=2;
345     canFilterConfig.FilterFIFOAssignment=CAN_RX_FIFO0;
346     canFilterConfig.FilterScale=CAN_FILTERSCALE_32BIT;
347     canFilterConfig.FilterMode=CAN_FILTERMODE_IDMASK;
348     canFilterConfig.FilterMaskIdLow=0x0000;
349     canFilterConfig.FilterMaskIdHigh=0xFF00;
350     canFilterConfig.FilterIdLow=0x0000;
351     canFilterConfig.FilterIdHigh=0x1500;
352
353
354     HAL_CAN_ConfigFilter(&hcan1, &canFilterConfig);
355   /* USER CODE END CAN1_Init 2 */
```

- We have configured CAN filters for data transmission filteration. These filter configuration will help to avoid unnecessary data transmission and save data space.

5.2] Receiver node source code

```
64 CAN_RxHeaderTypeDef RxHeader;
65 uint8_t RxData[5];
66 char str[20];
67
68
69
70 void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan){
71
72     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET); // blue
73     HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &RxHeader, RxData);
74 }
75 /* USER CODE END 0 */
```

- Declared Rx data buffer to receive messages and a string ‘str ’ to display messages on lcd.

- Collected messages in Rxdata with Rxdata header ID and enabled data collection for Rx fifo filter configuration

```
/* USER CODE BEGIN 2 */
HAL_CAN_Start(&hcan1);
HAL_CAN_ActivateNotification(&hcan1, CAN_IT_RX_FIFO0_MSG_PENDING);
HAL_Delay(100);
HAL_Delay(100);

lcd_init();
lcd_clear();
```

- Initialized lcd and CAN to notify on indication of RX_fifo_message_pending ; so it can receive the message passed through filter.

```
while (1)
{
    lcd_clear();
    sprintf(str, "Temp:%d",RxData[1]);
    lcd_send_cmd(0x80|0x00);
    lcd_send_string(str);


    sprintf(str, "Dist:%d",RxData[2]);
    lcd_send_cmd(0x80|0x40);
    lcd_send_string(str);
              if(RxData[0] == 1){
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET); //
      }
      if(RxData[0] == 0){
          HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET); //
      }

  /* USER CODE END WHILE */
```

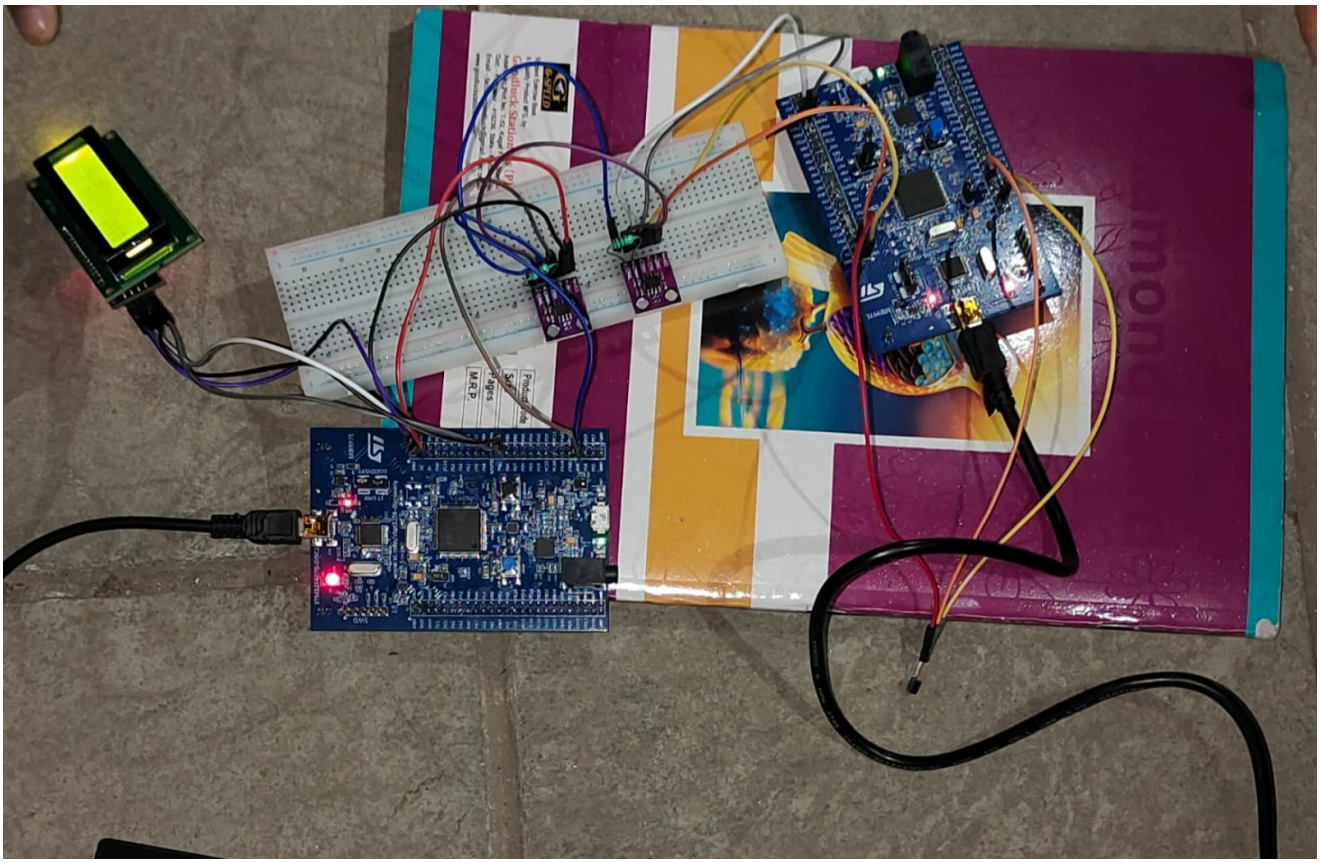
- Used a infinite while loop for data reception. Accepted temperature data and displayed it to line one of lcd as ‘str’ value. Similarly displayed distance data on line two of lcd.

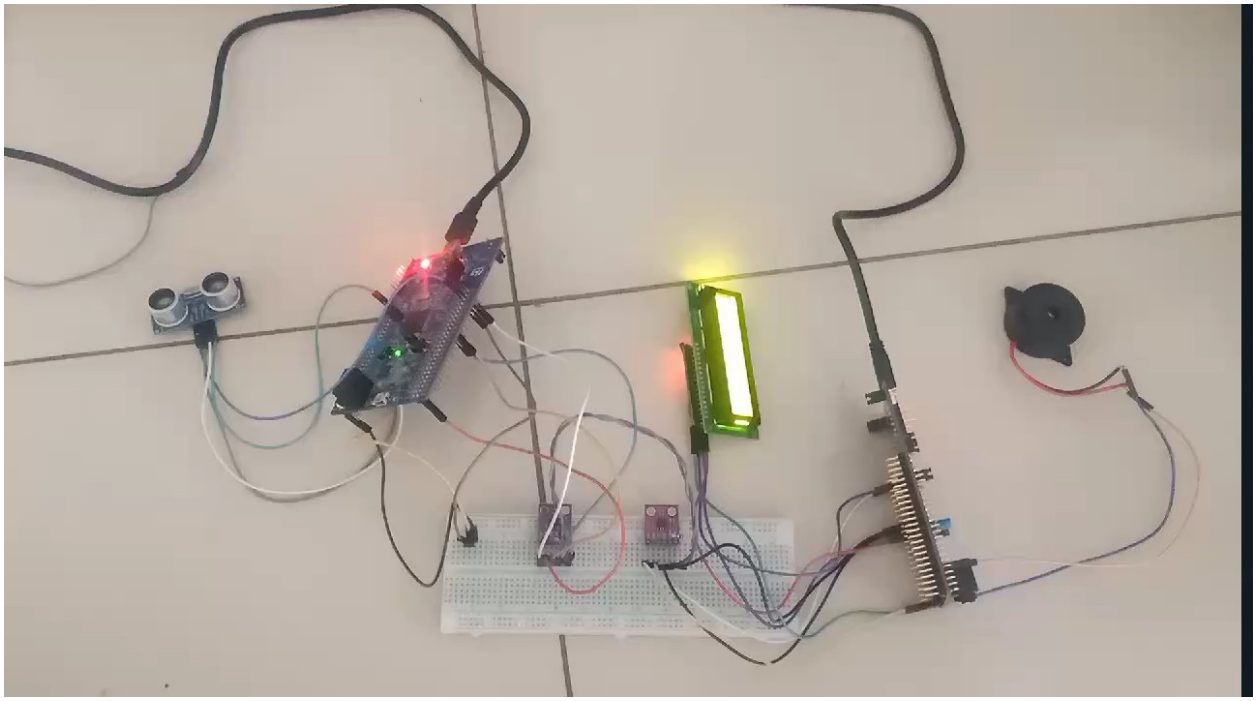
```
  /* USER CODE BEGIN 3 */
      if(RxData[2] <= 7)
          {
              HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, GPIO_PIN_SET);
              HAL_Delay(1000);
              HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10, GPIO_PIN_RESET);
              HAL_Delay(1000);
          }

}
/* USER CODE END 3 */
```

- Controlling a buzzer connected to GPIO Pin 10 of Port D on an STM32 microcontroller.
- The buzzer **beeps ON and OFF every second** when RxData[2] <= 7.
- This creates a **1 Hz beeping sound** whenever the condition is met.

6] Testing images

6.1] connections





6] Conclusion

In this system, the CAN bus based communication system for intelligent driver alert system is designed. The status of car like fuel level indication, the speed of the vehicle, obstacle detection and temperature of car engine are displayed on LCD digitally, controller will send the signal information and alert to the user. The proposed high-speed CAN bus system solves the problem of automotive system applications. This system features efficient data transfer among different nodes and safety the driver and car in the practical application

7]Future Scope

We successfully implemented a system working on CAN protocol that can detect and provide respective alerts to avoid extra heating of car and measure the distance between obstacle and object. this has been implemented using temperature sensor and ultrasonic sensor mounted on stm32F407 microcontroller(which is consider as a different hardware communicating with the another hardware that is also STM32F407 board).This data of temperature sensor and ultrasonic sensor is displayed on LCD mounted on second hardware.

Further extension to the project can be manipulating and controlling the system from a remote place(ex. Turning on AC, turning on buzzer, rolling windows) from the information collected from sensors.

Similar application was used by TATA motors for controlling the AC remote.Also for user information this sensor data can be uploaded over cloude using IOT protocol like MQTT

References

3.1.1 https://elearning.vector.com/vl_can_introduction_en.html

3.1.2 Renesas CAN pdf

3.1.3 Serial Bus System pdf

3.1.4 CAN primer

3.1.5 STM32F407 Discovery User Manual

3.1.6 https://www.arm.com/products/processors/cortex-m/cortex-m4-processor.php