[redacted]

Project Report

on

"Microcontroller Communication using CAN Protocol"

Bachelor of Engineering

Electronics & Telecommunication Engineering

Sant Gadge Baba Amravati University, Amravati.

Submitted by

Atharva Tirtharaj Thakare

Mihir Prashant Deshmukh

Shruti Pradip Lahane

Guided by

Prof. B. P. Fuladi

DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION ENGINEERING,

PROF. RAM MEGHE INSTITUTE OF TECHNOLOGY & RESEARCH,

BADNERA-AMRAVATI

2022-2023

CERTIFICATE

This is to certify that the project report entitled

"Microcontroller Communication using CAN Protocol"

has been successfully completed by

Atharva Tirtharaj Thakare

Mihir Prashant Deshmukh

Shruti Pradip Lahane

in satisfactory manner as a partial fulfilment of
Degree of Bachelor of Engineering in
(Electronics & Telecommunication Engineering)
Sant Gadge baba Amravati University, Amravati
during the academic year 2022-2023

Prof. B. P. Fuladi
Guide
Department of EXTC Engineering.
PRM Institute of Technology & Research,
Badnera

Dr. S. W. Mohod
Head,
Department of EXTC Engineering.
PRM Institute of Technology & Research,
Badnera

External Examiner

Vidarbha Youth Welfare Society's

Prof. Ram Meghe Institute of Technology & Research, Badnera-Amravati.

Department of Electronics & Telecommunication Engineering.

B.E. (Electronics and Telecommunication)

Certificate of Originality

This is to certify that, I am responsible for the work submitted in this project. The original work is my own except as specified in the references & acknowledgement. The original work contained herein have not been undertaken or done by unspecified sources or persons.

Atharva T. Thakare

Mihir P. Deshmukh

Shruti P. Lahane

ACKNOWLEDGEMENT

It is my moral duty & responsibility to be loyal & grateful to those who have shown me the path by throwing their knowledged rays during the Project.

It is worth mentioning here that as a guide ***Prof. B. P. Fuladi*** *has encouraged me time to time during the Project. He all the while was my co-rider. He is the reinforcement of my dissertation. Let me be honest to pay him utmost regards for his guidance to which my project proved to be a successful one.*

Words are insufficient to show my thankfulness to HOD ***Dr. S. W. Mohod*** *who at every point showed me the telescopic way in respect of my project.*

I am grateful to all the ***authors*** *of books & papers which have been referred for this Project. Last but not the least, I am thankful to* ***all teaching, non-teaching staff and everyone*** *who directly or indirectly helped me for the completion of this project.*

Atharva T. Thakare

Mihir P. Deshmukh

Shruti P. Lahane

ABSTRACT

The core concept of the project is to communicate between 4 nodes of Microcontroller such as Arduino Uno, STM32 F446RE Nucleo Board, ESP32 & Raspberry Pi Pico using CAN (Controller Area Network) protocol without any host computer. Further we have used CAN Transceiver i.e. MCP 2515 as there is no inbuild CAN system in Arduino Uno and Raspberry Pi Pico. The assembly of these four microcontrollers is called as CAN Node. We have taken CAN Node 1, CAN Node 2, CAN Node 3, CAN & Node 4. These nodes is connected to CAN Bus for further communications between nodes. The received data on CAN Bus is shown on LCD Display.

CAN is network protocol that enables communication between different electronic control units (ECUs) or devices within a system without the need for a central computer. Overall, the CAN protocol provides a robust, reliable, and flexible communication solution for various applications, particularly in the automotive and industrial domains, where real-time performance and fault tolerance are critical requirements. Therefore in these project we simply demonstrate the working principles, characteristics and operation of CAN protocol using 4 microcontrollers.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

1.1 DEFINITION:

Microcontroller communication using the Controller Area Network (CAN) protocol involves the exchange of messages between microcontrollers or other devices over a network using the CAN protocol, which is a widely used communication standard in embedded systems. CAN is a message-based, multi-master, multi-node protocol that enables reliable communication in real-time, with high fault tolerance and deterministic behavior.

1.2 ANALOGY:

In 21st century we generally used Computers for commanding n number of electronic devices for ex. in Indian Railways they use single mega computer to communicate with large number of trains in that specific region. In broad vision it is costlier to use expensive computer assembly in small electronics architecture. At this moment microcontrollers comes into the pictures. Microcontroller itself is mini computer throughout which we can control n numbers of microcontrollers using single microcontroller by inducing single embedded c code in master microcontroller.

1.3 OVERVIEW:

The project is demonstration of communication between 4 nodes of Microcontroller such as Arduino Uno, STM32 F446RE Board, ESP32 & Raspberry Pi Pico using CAN (Controller Area Network)_protocol without any host computer. Further we have used CAN Transceiver such as MCP 2515 as there is no inbuild CAN system in Arduino Uno and Raspberry Pi Pico. The assembly of these four microcontrollers is called as CAN Node. We have taken CAN Node 1, CAN Node 2, CAN Node 3, CAN & Node 4. These nodes is connected to CAN Bus_for further communications between nodes. The received data on CAN Bus is shown on LCD Display.

1.4 CONCEPT:

CAN (Controller Area Network) communication can be established between Arduino Uno, STM32 F446RE, ESP32, and Raspberry Pi using the CAN protocol. Here's a general overview of how we have achieved this:

1. Hardware Setup: First, we have connected the CAN transceivers to the respective microcontrollers or boards. CAN transceivers convert the digital signals from the microcontroller into differential signals for communication over the CAN bus. The exact pinout and connection details may vary depending on the specific transceiver module that we are using, so be sure to refer to the respective datasheets and documentation.

2. CAN Configuration: Next, we have to configured the CAN parameters on each microcontroller or board. This includes setting the baud rate, setting the mode of operation (e.g., normal mode, loopback mode, etc.), and configuring the CAN message filters if required. The configuration process may involve setting registers or using software libraries provided by the respective microcontroller platforms.

3. Message Transmission: Once the CAN parameters are configured, we can start transmitting messages between the microcontrollers or boards. Messages are typically composed of an identifier (ID) that identifies the content of the message and its priority, and a data field that contains the actual data being transmitted. Each microcontroller or board can send messages by constructing CAN frames with the appropriate ID and data, and then sending them onto the CAN bus.

4. Message Reception: Microcontrollers or boards that are intended to receive CAN messages need to continuously monitor the CAN bus for incoming messages. When a message is received, it can be parsed and processed according to the intended functionality. This may involve filtering messages based on their IDs, extracting data from the data field, and taking appropriate actions based on the received data.

5. Software Libraries: There are various software libraries available for implementing CAN communication on different microcontroller platforms, such as Arduino, STM32, ESP32, and Raspberry Pi. For example we are using Cube IDE for STM32 f446re & Arduino IDE along with Github libraries and board manager for the rest of the boards.

1.5 HISTORY:

The CAN (Controller Area Network) protocol is a communication protocol originally developed by Bosch in the 1980s for use in automotive applications. It was designed to allow electronic control units (ECUs) in vehicles to communicate with each other in a reliable and efficient manner.

The history of the CAN protocol can be traced back to the increasing complexity of automotive systems in the 1980s. As vehicles became more computerized, there was a need for different ECUs to communicate with each other to coordinate various functions, such as engine control, transmission control, and anti-lock braking systems. Traditional point-to-point wiring systems were becoming impractical due to the large number of wires required, increased weight, and decreased reliability in response to these challenges, Bosch developed the CAN protocol in the mid-1980s as a solution for in-vehicle communication. The CAN protocol was based on a multi-master, message-based, and event-triggered architecture, where multiple ECUs could transmit messages over a shared communication bus. This allowed for efficient communication between ECUs, reduced wiring complexity, and improved reliability.

The CAN protocol was first introduced in production vehicles by Mercedes-Benz in 1991, and it quickly gained popularity in the automotive industry due to its robustness, reliability, and cost-effectiveness. It was widely adopted by other automakers and has since become the de facto standard for in-vehicle communication in modern vehicles.

1.6 IMPORTANCE:

The main objective of microcontroller communication using the CAN (Controller Area Network) protocol is to establish reliable, efficient, and standardized communication between different microcontrollers or electronic control units (ECUs) in a networked system. This can include systems in various industries, such as automotive, industrial automation, aerospace, and more.

The importance of microcontroller communication using the CAN protocol are:

Reliable Communication: CAN protocol is designed to provide reliable communication even in harsh environments with high levels of electrical noise, temperature variations, and electromagnetic interference. It uses a robust error detection and error correction mechanism to ensure that transmitted data is received accurately without corruption.

Efficient Communication: CAN protocol is a message-based and event-triggered protocol, allowing for efficient communication between different ECUs. It uses a prioritized message arbitration mechanism, where messages with higher priority are transmitted first, ensuring that critical messages are transmitted promptly.

Scalability: CAN protocol allows for easy expansion of the network by adding or removing ECUs without disrupting the overall communication. It supports multi-master communication, enabling multiple ECUs to transmit messages on the same network, and it can be implemented in different network topologies, such as star, bus, or ring.

Standardization: CAN protocol has well-defined and widely accepted standards, such as CAN 2.0A, CAN 2.0B, and ISO 11898, which ensure interoperability and compatibility between different CAN-enabled devices from various manufacturers. This allows for easy integration of different microcontrollers or ECUs from different vendors into a single network.

Real-time Communication: CAN protocol supports real-time communication, allowing for timely and deterministic data exchange between microcontrollers or ECUs. This is particularly important in safety-critical applications, where precise timing and synchronization are required.

Cost-effectiveness: CAN protocol is known for its simplicity and low implementation cost, making it an economical choice for communication in microcontroller-based systems. It uses a two-wire differential bus, which reduces the number of required wires and associated hardware costs, and it requires minimal computational resources, making it suitable for resource-constrained microcontrollers.

CHAPTER 2

LITERATURE SURVEY

2. LITERATURE SURVEY

"CAN Communication with Arduino Uno and MCP2515: A Step-by-Step Tutorial" by Jhon Moreno (2018): This technical article presents a detailed tutorial on implementing CAN communication with Arduino Uno using the MCP2515 CAN controller and the MCP2551 CAN transceiver. It covers topics such as setting up the hardware, wiring the MCP2515 and MCP2551 modules to Arduino Uno, installing the necessary libraries, and writing Arduino code for message transmission and reception.

"Implementing CAN Communication using Arduino Uno and CAN-BUS Shield" by Akshay Garg and Dr. Mohd. Abdul Ahad (2016): This research paper presents an implementation of CAN communication using Arduino Uno and the CAN-BUS Shield. It covers topics such as setting up the hardware, installing the necessary libraries, and writing Arduino code for message transmission and reception. The paper includes experimental results and performance evaluation of the implemented CAN communication system.

"Implementation of CAN Protocol using STM32F446RE" by Ankush Pharkya et al. (2020): This paper presents the implementation of the CAN protocol using STM32F446RE microcontroller. It provides an overview of the CAN protocol, including message format, arbitration, error detection, and fault tolerance. The paper discusses the hardware and software requirements for implementing CAN communication on STM32F446RE, including the use of the STM32Cube HAL library, CAN peripheral configuration, and example code. The authors present a practical implementation of CAN communication on STM32F446RE, including message transmission, reception, and error handling.

"ESP32 CAN Bus Development with Arduino IDE" by Sachin Soni (2020): This technical article presents an overview of ESP32 CAN bus development with Arduino IDE. It discusses the basics of CAN communication, including message formats, bit rates, and frame types. The article provides step-by-step instructions on how to implement CAN communication on ESP32 using Arduino IDE, including setting up the hardware, installing the necessary libraries, and writing code for message transmission and reception.

"Raspberry Pi Pico CAN Bus Communication using Arduino IDE" by Krzysztof Kaczmarek (2021): This technical article provides a comprehensive guide on how to implement CAN communication on Raspberry Pi Pico using the Arduino IDE. It covers topics such as setting up the hardware, installing the necessary libraries, and writing Arduino code for message transmission and reception. The article discusses the basics of CAN communication, including message formats, bit rates, and frame types, and provides example code for implementing CAN communication using the MCP_CAN library in Arduino IDE.

"Efficient data transmission in CAN networks for microcontroller-based systems" by Haris Orak et al. (2019): This paper focuses on efficient data transmission techniques in CAN networks for microcontroller-based systems. It discusses various methods for improving the data transmission performance in CAN networks, including priority assignment, message packing, and scheduling algorithms.

"CAN protocol implementation for embedded systems" by Sarath Pillai et al. (2014): This paper provides an in-depth review of CAN protocol implementation in embedded systems, with a focus on microcontrollers. It discusses various aspects of CAN protocol, including message format, arbitration, error detection, and fault tolerance. The paper also presents a review of CAN software and hardware implementations, including different architectures, programming techniques, and tools for implementing CAN in microcontroller-based systems. This review can serve as a valuable resource for researchers and practitioners involved in the design and implementation of microcontroller communication using the CAN protocol.

"**A review on CAN bus-based control for electric vehicles" by Bo Jiang et al. (2019):** This review article provides an overview of the use of the CAN bus for control in electric vehicles (EVs). It discusses the various control functions in EVs that can be implemented using the CAN protocol, such as motor control, battery management, and charging control. The article also reviews recent advancements in using CAN for communication in advanced EV features, such as vehicle-to-grid (V2G) communication, energy management, and autonomous driving. This review article can be a valuable resource for researchers and practitioners working on EVs and interested in utilizing the CAN protocol for communication.

"Microcontroller-Based CAN Communication for Industrial Automation" by A. B. Jaganathan et al. (2015): This research paper presents an implementation of CAN communication for industrial automation using microcontrollers. It covers topics such as

hardware interfacing, CAN message formats, message transmission, and reception using microcontrollers. The paper includes experimental results and performance evaluation of the implemented CAN communication system. It also discusses the challenges and considerations in implementing CAN communication on microcontrollers, such as timing constraints, error handling, and fault tolerance. It provides insights into the practical aspects of microcontroller communication using CAN protocol in industrial automation.

"Controller Area Network (CAN) Based Industrial Automation: A Literature Review" by S. Sujatha et al. (2016): This literature review article provides an overview of CAN-based industrial automation. It covers topics such as CAN communication principles, CAN network configuration, CAN message formats, and CAN applications in industrial automation. The review also discusses the advantages and limitations of CAN in industrial automation, including its reliability, determinism, and scalability. It provides insights into the use of CAN protocol for microcontroller communication in industrial automation applications.

CHAPTER 3

METHODOLOGY

3. METHODOLOGY

3.1 BLOCK DIAGRAM:

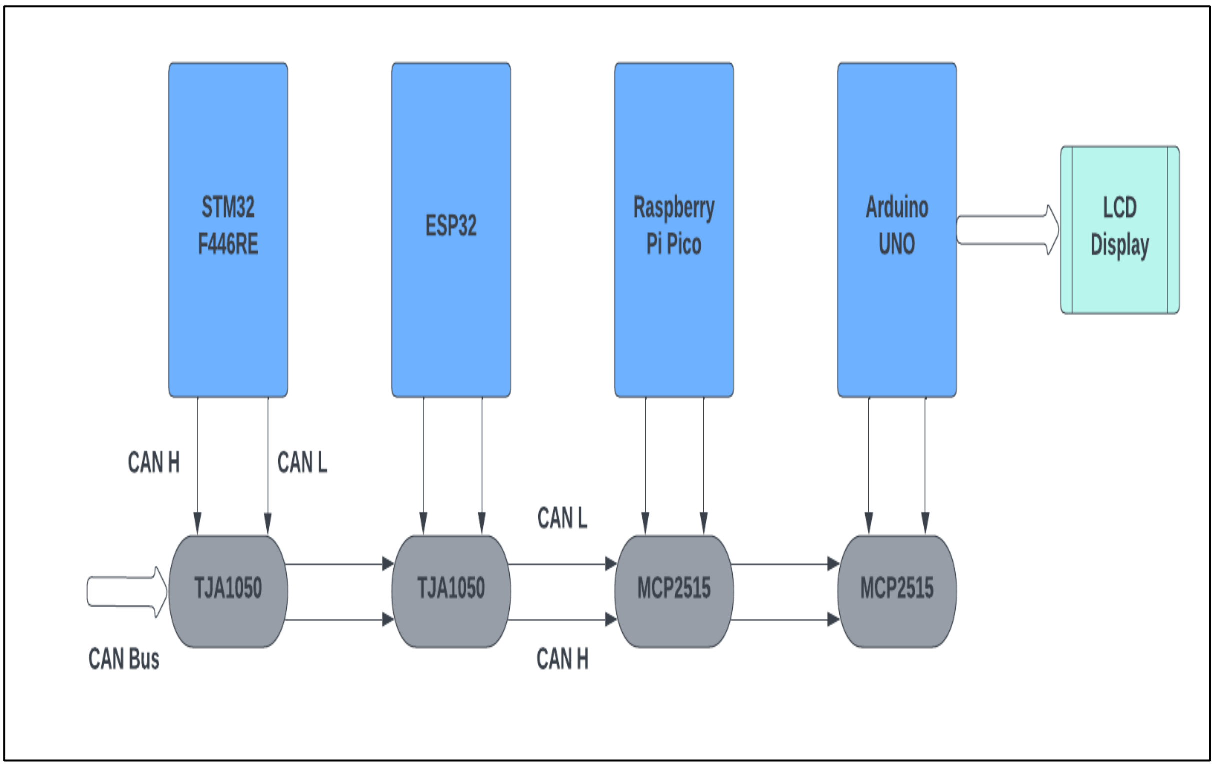


Fig 3.1 Block Diagram of System

3.2 DESCRIPTION:

Our project aimed Microcontroller Communication using CAN Protocol involves four microcontroller boards which are STM32 F446RE, ESP32, Raspberry Pi Pico & Arduino UNO. While we are using inbuild CAN controller for STM32 F446RE and ESP32 therefore we do not need external CAN controller for STM32 F446RE and ESP32 we are just using CAN transceiver i.e. TJA 1050 which offers CAN High and CAN Low for the two board. Whereas for Arduino UNO and Raspberry Pi Pico we are using external CAN controller because there is no inbuild CAN module in Arduino UNO and Raspberry Pi Pico. We are using MCP 2515 as external CAN controller for Arduino UNO and Raspberry Pi Pico. The four CAN Transceiver and CAN Controller which are 2 TJA 1050 & 2 MCP 2515 creates CAN But having two terminals CAN High & CAN Low. While these CAN Bus is created by using twisted pair cable to avoid distortion and noise of signal.

Now the assembly of project involves 4 nodes comprising of 4 microcontrollers for that purpose we have created CAN Bus of four microcontroller in which we have implemented Master-Slave concept for that 3 boards STM32 F446RE, ESP32 & Raspberry Pi Pico is master controller which means Hexadecimal data is being transmitted by these 3 boards. Thus, STM32 F446RE, ESP32 & Raspberry Pi Pico are our Master Microcontroller Transmitter or Sender. On other hand Arduino UNO is our Receiver which means Arduino UNO will receive all Hexadecimal data coming from 3 master board. Therefore, Arduino UNO is our Slave Part.

The received data on Slave Arduino UNO is sent to the Liquid Crystal Display (LCD) of size 20×4 which is further soldered with I2C module for input data from Arduino UNO and received data is showed there. Each line on LCD Display will represent data from that particular Master Microcontroller received by Arduino UNO.

3.3 ANALYSIS:

3.3.1 What is CAN Protocol?

The CAN protocol is a standard designed to allow the microcontroller and other devices to communicate with each other without any host computer. The feature that makes the CAN protocol unique among other communication protocols is the broadcast type of bus. The CAN is a two-wired communication protocol as the CAN network is connected through the two-wired bus. The wires are twisted pair having 120Ω characteristics impedance connected at each end. Initially, it was mainly designed for communication within the vehicles.

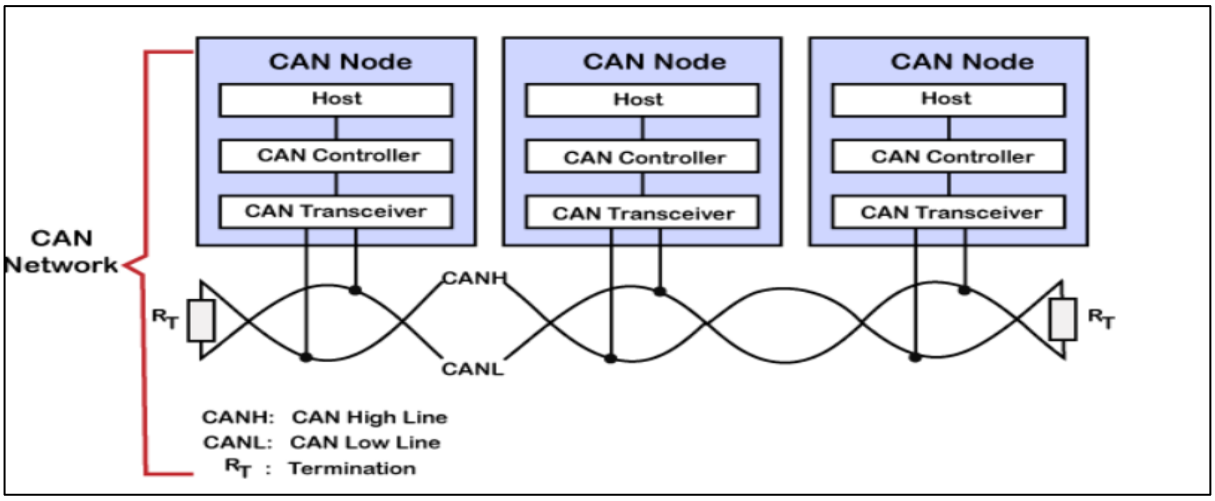


Fig 3.2 Basic CAN Assembly

CAN layered architecture:

1 CAN specifications define CAN protocol and CAN physical layer, which are defined in the CAN standard ISO 110898.

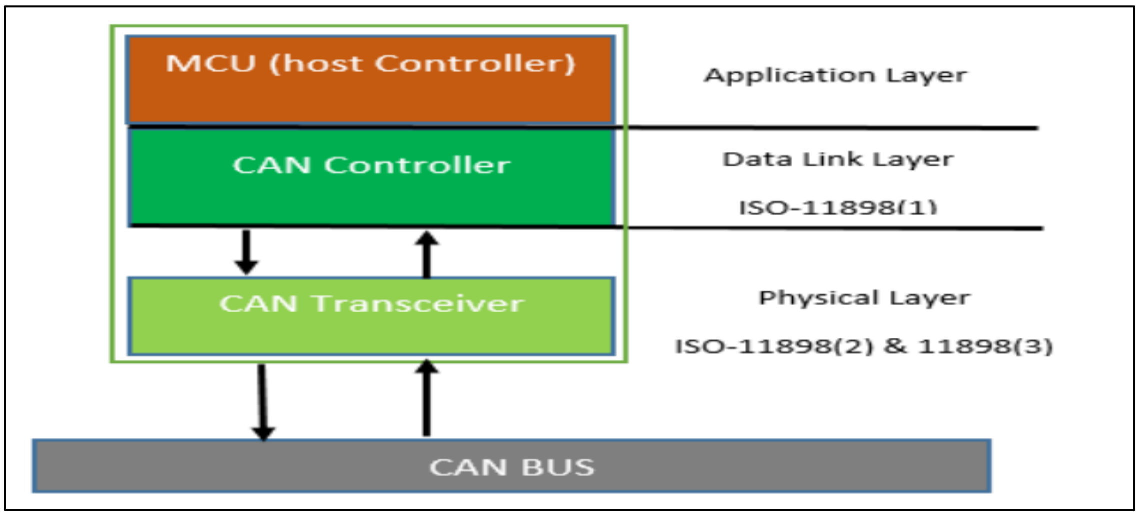


Fig 3.2 Layered architecture of CAN Protocol

ISO 11898 has three parts:

- ISO 11898-1: This part contains the specification of the Data-link layer and physical signal link.
- ISO 11898-2: This part comes under CAN physical layer for high speed CAN. The high- speed CAN allows data rate upto 1 Mbps used in the power train and the charges area of the vehicle.
- ISO 11898-3: This part also comes under CAN physical layer for low-speed CAN. It allows data rate upto 125 kbps, and the low speed CAN is used where the speed of communication is not a critical factor.

2 CiA DS-102: The full form of CiA is CAN in Automation, which defines the specifications for the CAN connector.

3 As far as the implementation is concerned, the CAN controller and CAN transceiver are implemented in the software with the help of the application, operating system, and network management functions.

CAN Framing:

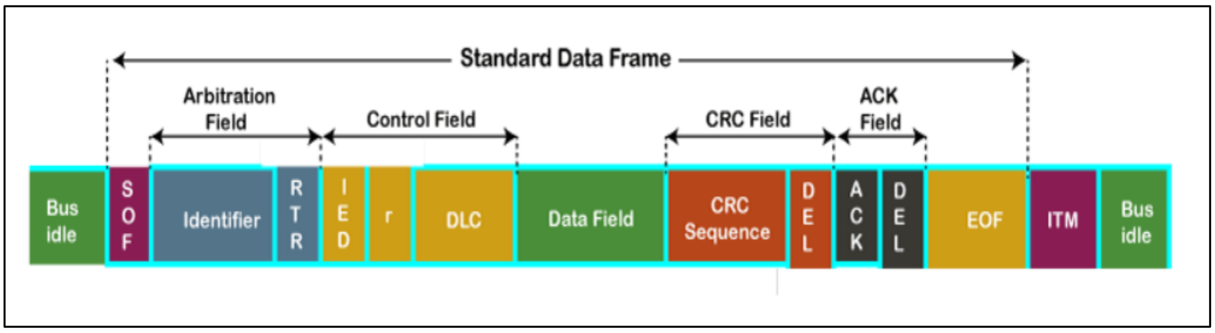


Fig 3.3 CAN Frame

SOF: SOF stands for the start of frame, which indicates that the new frame is entered in a network. It is of 1 bit.

Identifier: A standard data format defined under the CAN 2.0 A specification uses an 11-bit message identifier for arbitration. This message identifier sets the priority of the data frame.

RTR: RTR stands for Remote Transmission Request, which defines the frame type, whether it is a data frame or a remote frame. It is of 1-bit.

Control field: It has user-defined functions.

> **IDE:** An IDE bit in a control field stands for identifier extension. A dominant IDE bit defines the 11-bit standard identifier, whereas recessive IDE bit defines the 29-bit extended identifier.
>
> **DLC:** DLC stands for Data Length Code, which defines the data length in a data field. it is of 4 bits.
>
> **Data field:** The data field can contain upto 8 bytes.

CRC field: The data frame also contains a cyclic redundancy check field of 15 bit, which is used to detect the corruption if it occurs during the transmission time. The sender will compute the CRC before sending the data frame, and the receiver also computes the CRC and then compares the computed CRC with the CRC received from the sender.

ACK field: This is the receiver's acknowledgment. In other protocols, a separate packet for an acknowledgment is sent after receiving all the packets, but in case of CAN protocol, no separate packet is sent for an acknowledgment.

3.2.2 Why CAN?

The need for a centralized standard communication protocol came because of the increase in the number of electronic devices. For example, there can be more than 7 TCU for various subsystems such as dashboard, transmission control, engine control unit, and many more in a modern vehicle. If all the nodes are connected one-to-one, then the speed of the communication would be very high, but the complexity and cost of the wires would be very high. In the above example, a single dashboard requires 8 connectors, so to overcome this issue, CAN was introduced as a centralized solution that requires two wires, i.e., CAN high and CAN low. The solution of using CAN protocol is quite efficient due to its message prioritization, and flexible as a node can be inserted or removed without affecting the network**.**

3.2.3 What is CAN Bus?

CAN bus, or Controller Area Network bus, is a widely used communication protocol that allows devices within a network to communicate with each other over a shared medium, typically using twisted pair cables. Originally developed by Bosch for the automotive industry, CAN bus has found applications in various other industries as well. It is known for its reliability, real-time performance, and noise immunity, making it suitable for distributed control applications. CAN bus supports multi-master communication, allowing multiple devices to send and receive messages simultaneously, and is used for communication between devices in a peer-to-peer or broadcast fashion.

3.3.4 What is CAN High & CAN Low?

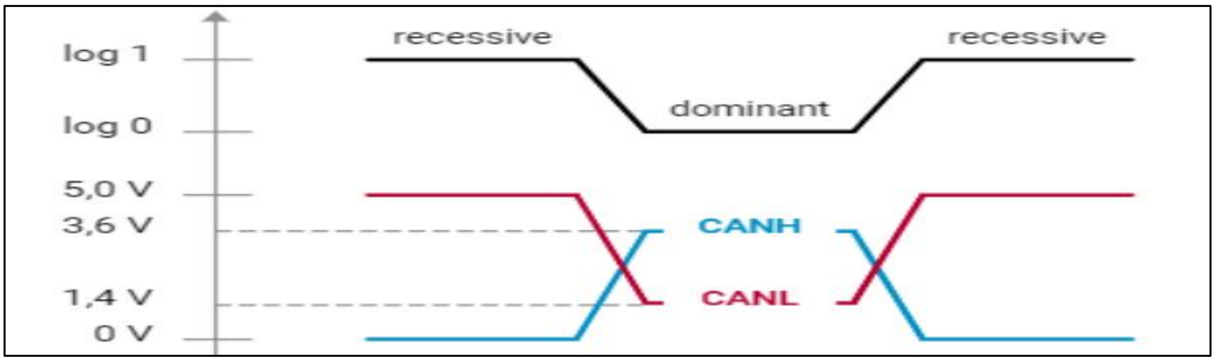


Figure 3.2 CAN Bus representation on graph with CAN H & CAN L

CAN Transceiver develops two pair of wires called as CAN H & CAN L. CAN H and CAN L is basically CAN High and CAN Low. In CAN terminology CAN High and CAN Low is dependent on potential difference and logic of AND gate. Let's see CAN terminology in brief, we know that logic 1 is Recessive and logic 0 is Dominant in CAN.

CAN Bus Terminology chart:

Logic	Potential difference	Std Voltage
1 (Recessive)	CAN High – CAN Low	0V
0 (Dominant)	CAN High – CAN Low	2V

As we are observing in above chart the ideal standard voltage of logic 1 is 0 V and ideal standard voltage of logic 0 is 2v.

This CAN H and CAN L of one CAN Transceiver terminals are connected to same terminals of another CAN Transceiver by Twisted pair cables.

3.2.5 Why discrete signals are used in CAN rather than digital signals?

Sampling discrete-time signals, i.e., using only every Nth sample of a sequence of samples, is efficient for processing, transmitting, or storing information. Also, digital signals are not continues and can not be transmit at long distance efficiently whereas discrete signals can travel long distance via CAN Protocol.

3.3.6 Why Twisted pair cables is used for connection in CAN Bus?

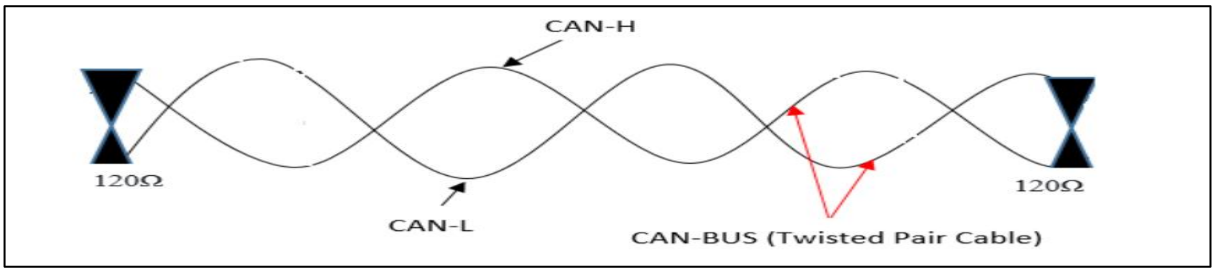


Figure 3.3 CAN Bus twisted pair cable of CAN H & CAN L

Twisted pair cables are used in CAN (Controller Area Network) protocol due to their excellent noise immunity, high bandwidth, cost-effectiveness, availability, and compatibility, making them reliable and practical for industrial and automotive applications.

1. Noise Immunity: Twisted pair cables are designed to minimize electromagnetic interference (EMI) and radio frequency interference (RFI). The twisting of the pairs of wires helps to cancel out external electromagnetic fields, reducing the impact of noise on the signal quality. This makes twisted pair cables highly immune to electrical noise, which is prevalent in industrial and automotive environments where CAN protocols are commonly used.
2. Differential Signalling: CAN protocol uses differential signalling, where the data is transmitted as voltage levels between two wires in a twisted pair cable. The voltage levels on the two wires are opposite in polarity, and the difference between them represents the transmitted data. This differential signalling helps to improve noise immunity and allows for longer cable runs without significant signal degradation.
3. High Bandwidth: Twisted pair cables are capable of supporting high data rates, making them suitable for modern CAN protocols that require fast data transmission. The use of twisted pair cables allows for efficient data transmission at high speeds, ensuring reliable communication between devices on the CAN network.

CHAPTER 4:

IMPLEMENTATION

4. IMPLEMENTATION

4.1 PROJECT LAYOUT:

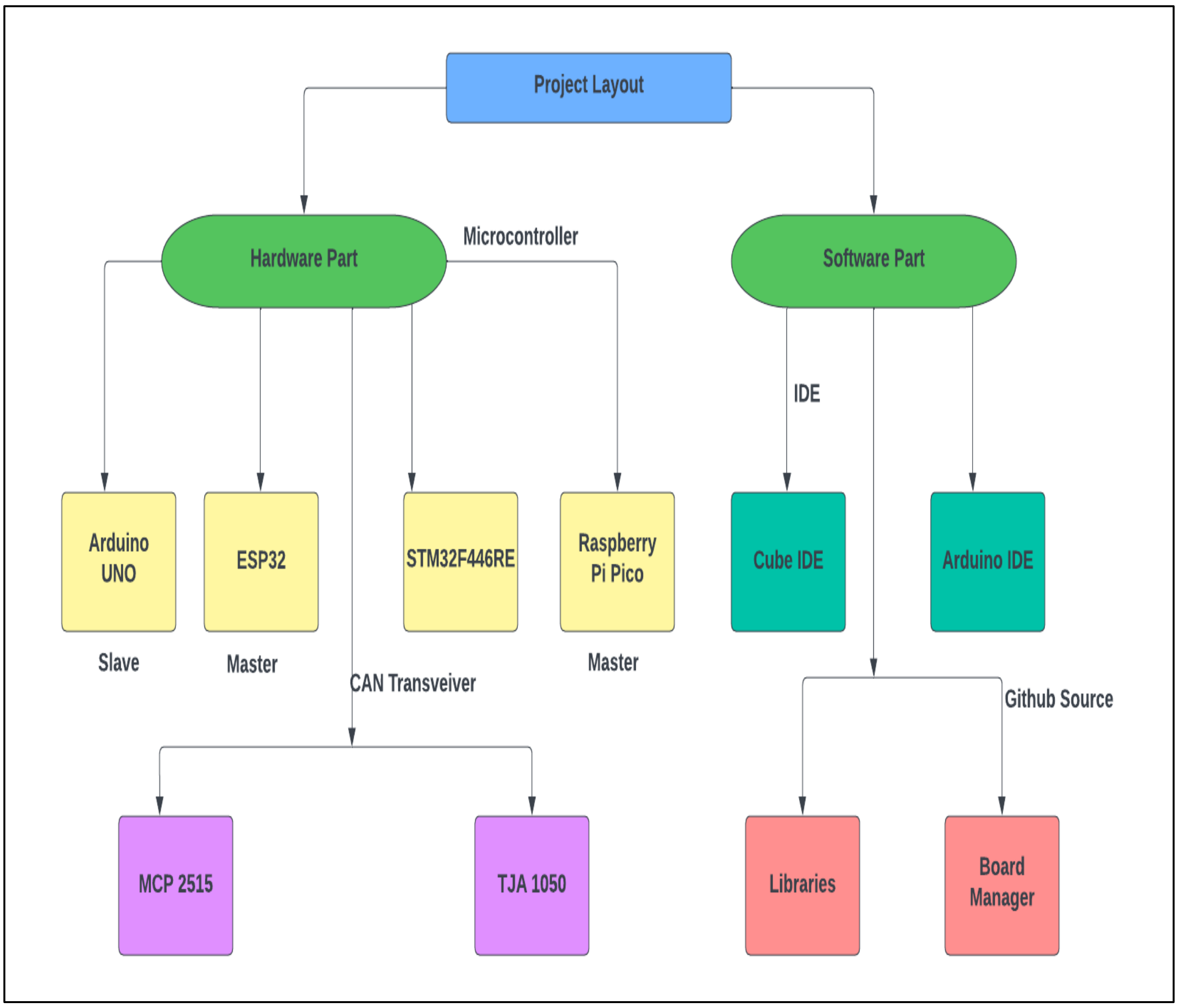


Fig 4.1 Project Layout

The Project layout consists of two parts software part & hardware part. Hardware part involves all 4 microcontroller along with CAN transceiver which we have seen before while software part involves Cube IDE for the coding part of STM32 F446RE & Arduino IDE for coding in the Arduino UNO, ESP32 & Raspberry Pi Pico. In Arduino IDE we are using Github board libraries such as Earlephilhower/arduino-pico and Espressif/arduino-esp32. Also we are using MCP Libraries for the CAN Interface of specific microcontroller with MCP2515. Further we are using Liquid Crystal Library for LCD Display for showing the received data of Arduino UNO on LCD Display.

4.2 HARDWARE ASSEMBLY:

Hardware assembly involves interfacing of STM32 F446RE with TJA1050, ESP32 with another TJA 1050 and interfacing of Arduino UNO with MCP2515 and Raspberry pi pico with another MCP2515 because Arduino UNO & Raspberry Pi Pico do not have inbuilt CAN controller. All connection are done by using Jumper wires all types are involves i.e. male-male, male-female & female-female. While twisted pair cable used for two CAN tranceivers. Now let's see pin configuration of each microcontroller with respective CAN transceivers.

4.2.1 Connections Arduino UNO with MCP2515:

Arduino UNO	**MCP 2515**
Arduino 5V	VCC
Arduino GND	GND
D13 (SPI SCK)	SCK
D12 (SPI MISO)	SO
D11 (SPI MOSI)	SI
D10	CS
D2	INT

Table 4.1 Connections Arduino UNO with MCP2515

1. VCC: Connects to the 5V power supply of the Arduino UNO, providing power to the MCP2515 module.
2. GND: Connects to the ground (GND) of the Arduino UNO, establishing a common ground reference between the Arduino UNO and the MCP2515 module.

3. SPI SCK: Connects to the Serial Clock (SCK) pin of the Arduino UNO, which is used for the SPI (Serial Peripheral Interface) communication protocol to transfer data between the Arduino UNO and the MCP2515 module.

4. SPI MISO: Connects to the Master In Slave Out (MISO) pin of the Arduino UNO, which is used for SPI communication to receive data from the MCP2515 module.

5. SPI MOSI: Connects to the Master Out Slave In (MOSI) pin of the Arduino UNO, which is used for SPI communication to send data to the MCP2515 module.

6. CS: Connects to a digital pin on the Arduino UNO (e.g., D10), which is used as the Chip Select (CS) pin for the MCP2515 module. It is used to enable or disable the MCP2515 module for communication.

7. INT: Connects to a digital pin on the Arduino UNO (e.g., D2), which is used as the interrupt (INT) pin for the MCP2515 module. The MCP2515 module can generate an interrupt to the Arduino UNO when a CAN message is received or when other events occur.

4.2.2 Connection of Raspberry Pi Pico with MCP2515:

Raspberry Pi Pico	**MCP2515**
3.3V	VCC
GND	GND
GP2 (SPIO SCK)	SCK
GP3 (SPIO MISO)	SO
GP (SPI0 MOSI)	SI

GP6 (SPI0 CSn)	CS
INT	INT

Table 4.2 Connection of Raspberry Pi Pico with MCP2515

These connections enable the Raspberry Pi Pico to communicate with the MCP2515 module using the SPI protocol, allowing the Raspberry Pi Pico to send and receive CAN messages over a Controller Area Network (CAN) bus.

4.2.3 Connection of STM32 F446RE with TJA 1050:

STM32F446RE (Microcontroller)	**TJA1050 (CAN Transceiver)**
CAN1_TX (PA12)	TX
CAN1_RX (PA11)	RX
VCC (3.3V)	VCC
GND	GND

Table 4.3 Connection of STM32 F446RE with TJA 1050

1. CAN1_TX (PA12) of STM32F446RE should be connected to TX transmitter pin of the TJA1050. This is the transmit pin of the CAN interface on the microcontroller, and it is used to send data over the CAN bus.

2. CAN1_RX (PA11) of STM32F446RE should be connected to RX receiver pin of the TJA1050. This is the receive pin of the CAN interface on the microcontroller, and it is used to receive data from the CAN bus.

3. VCC (3.3V) of STM32F446RE should be connected to the VCC pin of the TJA1050. This provides power to the CAN transceiver and should be connected to the appropriate voltage level (typically 3.3V) for the STM32F446RE microcontroller.

4. GND of STM32F446RE should be connected to the GND pin of the TJA1050. This provides the ground reference for the CAN bus communication and ensures proper grounding between the microcontroller and the transceiver.

4.2.4 Connection of ESP32 with TJA1050:

ESP32 (Microcontroller)	**TJA1050 (CAN Transceiver)**
GPIO21 (TX)	TX Terminal
GPIO22 (RX)	RX Terminal
3.3V	VCC
GND	GND

Table 4.4 Connection of ESP32 with TJA1050

As we know there is inbuilt CAN Transceiver in ESP32 we are using TJA1050 for the CAN interfacing. The voltage levels used are compatible with both the ESP32 microcontroller (which typically operates at 3.3V) and the TJA1050 CAN transceiver (which typically operates at 5V or 3.3V depending on the version).

All four boards are connected to each other using CAN module which offers two terminal CAN High & CAN Low. This CAN H & CAN L relesed from each CAN module are connected to each other by using twisted wire cable which is called as CAN Bus having 120Ω resistor on both end to avoid distortion and noise in signal.

4.2.5 Connection of Arduino Uno with LCD display & I2C module:

Now we have to display received data on receiver microcontroller or Slave Arduino UNO to the LCD (Liquid Crystal Display) of 20×4 for convinience we have connected LCD Display to I2C module which provide proper connection with Ardino UNO. We have soldered I2C module to LCD Display. We have displayed 3 hexadecimal input values coming from 3 microcontrollers each on respective line.

Arduino UNO	**I2C Module(soldered with LCD)**
Arduino Uno GND	GND
Arduino Uno +5V	VCC
Arduino Uno A4 pin (or SDA pin)	SDA (Serial Data)
Arduino Uno A5 pin (or SCL pin)	SCL (Serial Clock)

Table 4.5 Connection of Arduino Uno with LCD display & I2C module

1. I2C Data (SDA) and I2C Clock (SCL): Connect the SDA (Serial Data) pin and SCL (Serial Clock) pin of the I2C module to the corresponding SDA and SCL pins of the Arduino Uno (usually A4 and A5 pins, respectively). These pins are used for communication between the Arduino Uno and the I2C module using the I2C protocol.

2. Ground (GND): Connect the GND pin of the Arduino Uno to the GND pin of the LCD display and the GND pin of the I2C module. This establishes a common ground reference for all components.

3. Power (VCC/+5V): Connect the +5V pin of the Arduino Uno to the VCC pin of the LCD display and the VCC pin of the I2C module. This provides power to both the LCD display and the I2C module.

4.3 SOFTWARE ASSEMBLY:

Software assembly involves two softwares Arduino IDE for coding in Arduino UNO, Raspberry pi pico & ESP32 while we are using Cube IDE for STM 32F446RE microcontroller. Further we are using additional Github Board libraries such as Earlephilhower/arduino-pico for Raspberry pi pico and Espressif/arduino-esp32 for ESP32 as we are coding these both board on Arduino IDE also we have installed libraries related to CAN protocol such as MCP_CAN, CAN, etc. Also we have added liquid crystal library for interfacing the arduino data on LCD display (20×4).

4.3.1 Arduin IDE Setup:

We are using Arduino 1.8.19 version of official Arduino IDE which is open source and free software. Now let’s discuss which libraries and board manager we using for coding different microcontrollers on Arduino IDE.

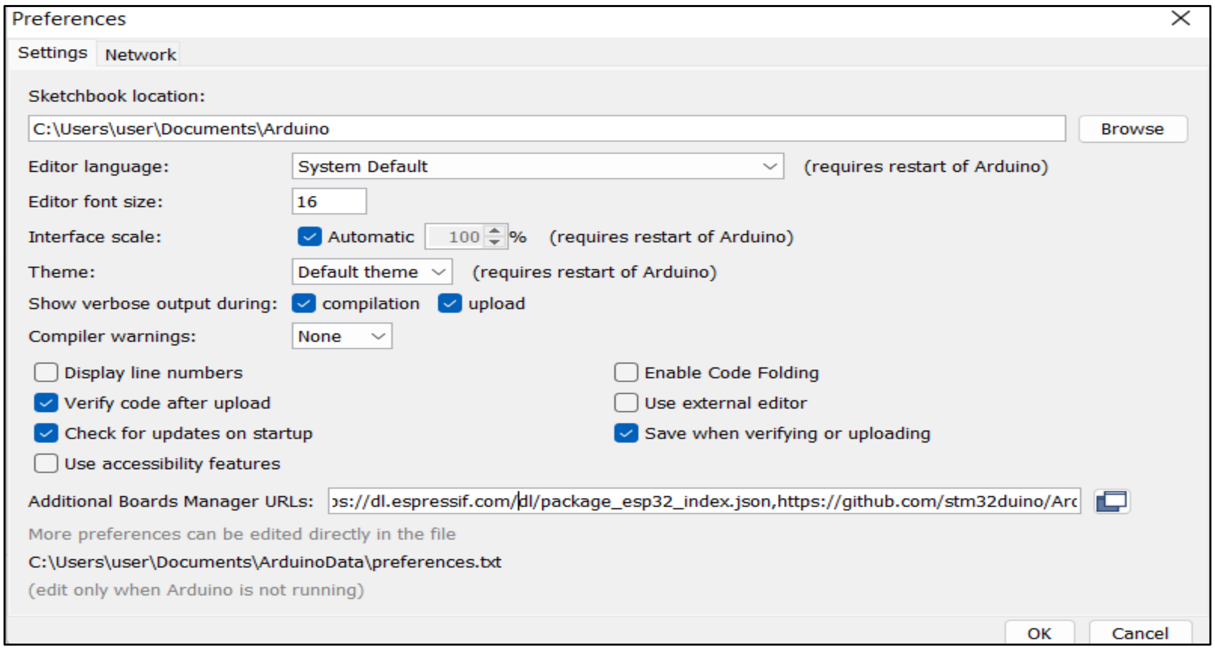
ESP32 Setup:

Step 1: For ESP32 we are using Espressif/arduino-esp32 which is available on Github we have to simply log in to particular domain which is Espressif/arduino-esp32 where we have to click on code tab and copy the https link given below.

Step 2: Now we have to paste that particular link in File → Preference → Aditional board manager URL’s of Arduino IDE and press OK tab.

Step 3: Now open Tools → Board → Board Manager which is on upper left side of IDE wait for some installation proceess as our files are being downloading.

Step 4: After successful downloading of file search for ESP32 and then we have to click on install tab to download particular library we can also select version for the paricular board library.



Preferences

Settings | Network

Sketchbook location: C:\Users\user\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 16

Interface scale: ✓ Automatic 100 % (requires restart of Arduino)

Theme: Default theme (requires restart of Arduino)

Show verbose output during: ✓ compilation ✓ upload

Compiler warnings: None

- ☐ Display line numbers
- ✓ Verify code after upload
- ✓ Check for updates on startup
- ☐ Use accessibility features
- ☐ Enable Code Folding
- ☐ Use external editor
- ✓ Save when verifying or uploading

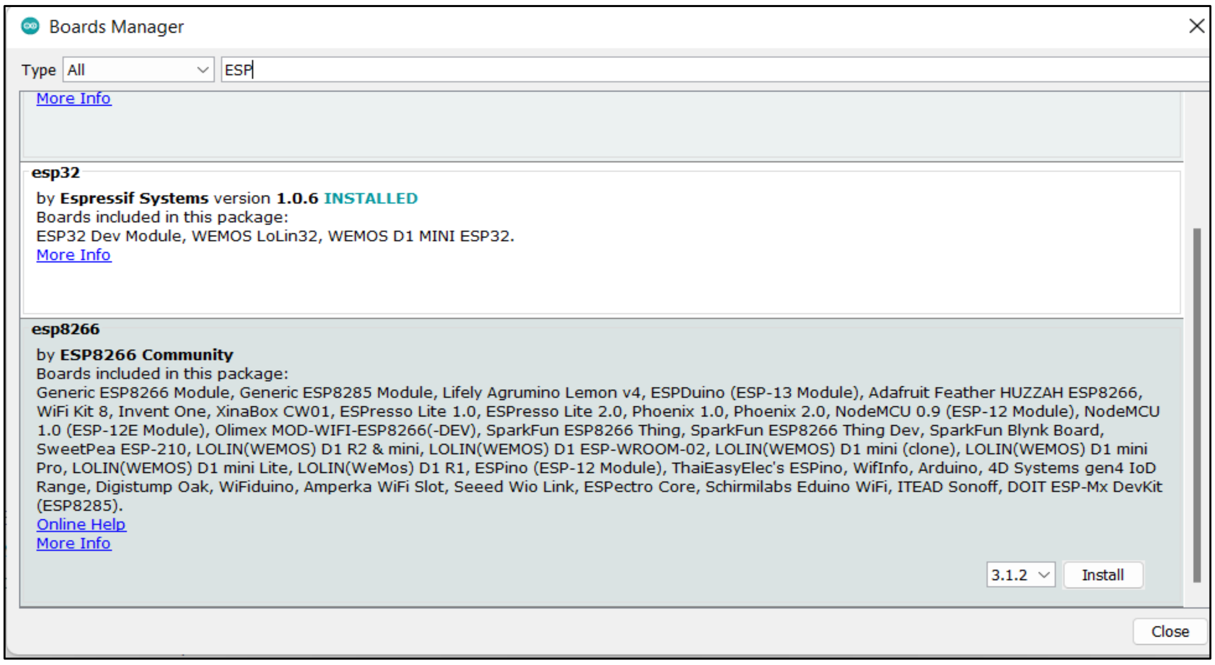
Additional Boards Manager URLs: ps://dl.espressif.com/dl/package_esp32_index.json,https://github.com/stm32duino/Arc

More preferences can be edited directly in the file

C:\Users\user\Documents\ArduinoData\preferences.txt

(edit only when Arduino is not running)

Fig 4.3 Pasting the link in IDE



Boards Manager

Type All | ESP

More Info

esp32

by **Espressif Systems** version **1.0.6** **INSTALLED**
Boards included in this package:
ESP32 Dev Module, WEMOS LoLin32, WEMOS D1 MINI ESP32.
More Info

esp8266

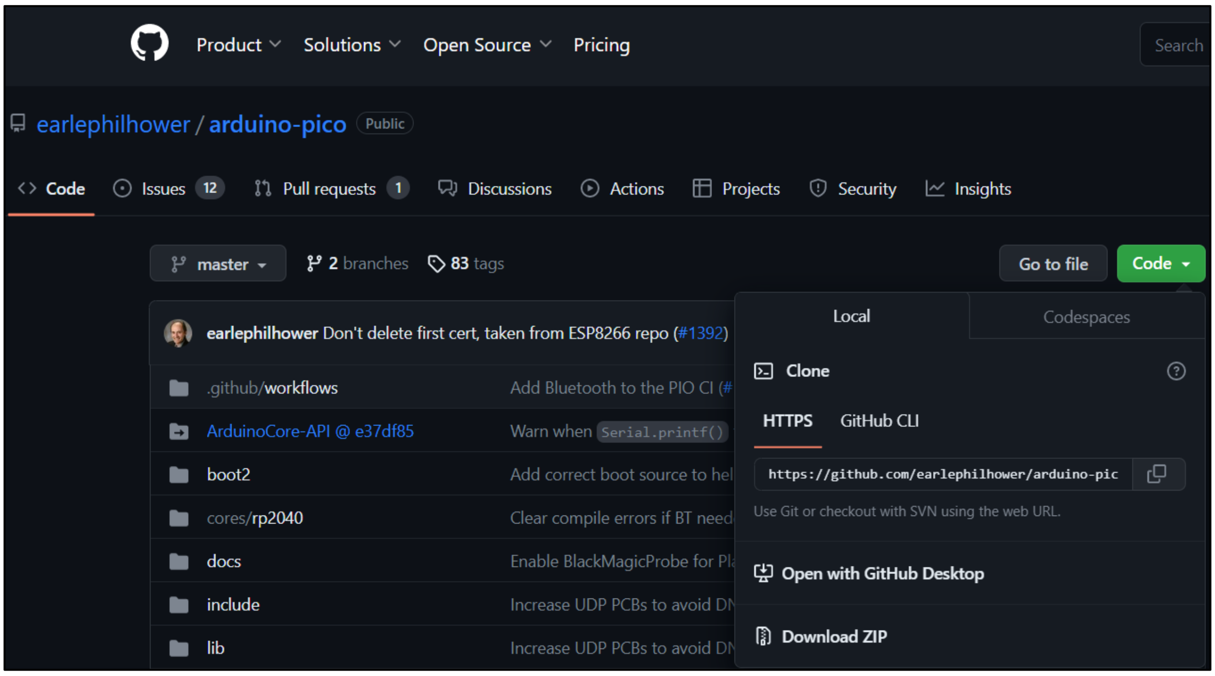
by **ESP8266 Community**
Boards included in this package:
Generic ESP8266 Module, Generic ESP8285 Module, Lifely Agrumino Lemon v4, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, WiFi Kit 8, Invent One, XinaBox CW01, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SparkFun Blynk Board, SweetPea ESP-210, LOLIN(WEMOS) D1 R2 & mini, LOLIN(WEMOS) D1 ESP-WROOM-02, LOLIN(WEMOS) D1 mini (clone), LOLIN(WEMOS) D1 mini Pro, LOLIN(WEMOS) D1 mini Lite, LOLIN(WeMos) D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WifInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak, WiFiduino, Amperka WiFi Slot, Seeed Wio Link, ESPectro Core, Schirmilabs Eduino WiFi, ITEAD Sonoff, DOIT ESP-Mx DevKit (ESP8285).
Online Help
More Info

Fig 4.4 Searching board in board manger

Raspberry Pi Pico Setup:

As we are using same Arduino IDE for coding in Raspberry Pi Pico we have to follow same steps as we have seen in case of ESP32 only difference is we have to paste

Earlephilhower/arduino-pico for Raspberry pi pico in Preference → Additional Board URL's and we have to search for Rasberry pi pico in Tools → Board → Board Manager



earlephilhower Don't delete first cert, taken from ESP8266 repo (#1392)

.github/workflows	Add Bluetooth to the PIO CI (#
ArduinoCore-API @ e37df85	Warn when `Serial.printf()`
boot2	Add correct boot source to hel
cores/rp2040	Clear compile errors if BT need
docs	Enable BlackMagicProbe for Pl
include	Increase UDP PCBs to avoid DN
lib	Increase UDP PCBs to avoid DN

```
https://github.com/earlephilhower/arduino-pic
```

Use Git or checkout with SVN using the web URL.

Fig: Github based Earlephilhower's library for Pico on Arduino IDE

We can also code raspberry pi pico with the help Thonny software we have to just install Micro Python Firmware as Raspberry pi pico interprets Micro Python and Cpp so we can code Raspberry pi pico on Thonny too using Micro Python Language.

Libraries for CAN & MCP:

We are using various cutomisable libraries for our project such as autowp-mcp2515, ACAN2515, CAN_BUS_SHIELD, mcp_can ,etc and liquid crystal library for LCD display in the Arduino IDE. We will predominantly use autowp-mcp2515, CAN and liquid crystal in our project and cutomizing the code according tt need of project.Now let us see how to download libraries.

Step 1: In Arduino IDE at the left corner click on Sketch → Include library → manage library to get custom library on Arduino IDE.

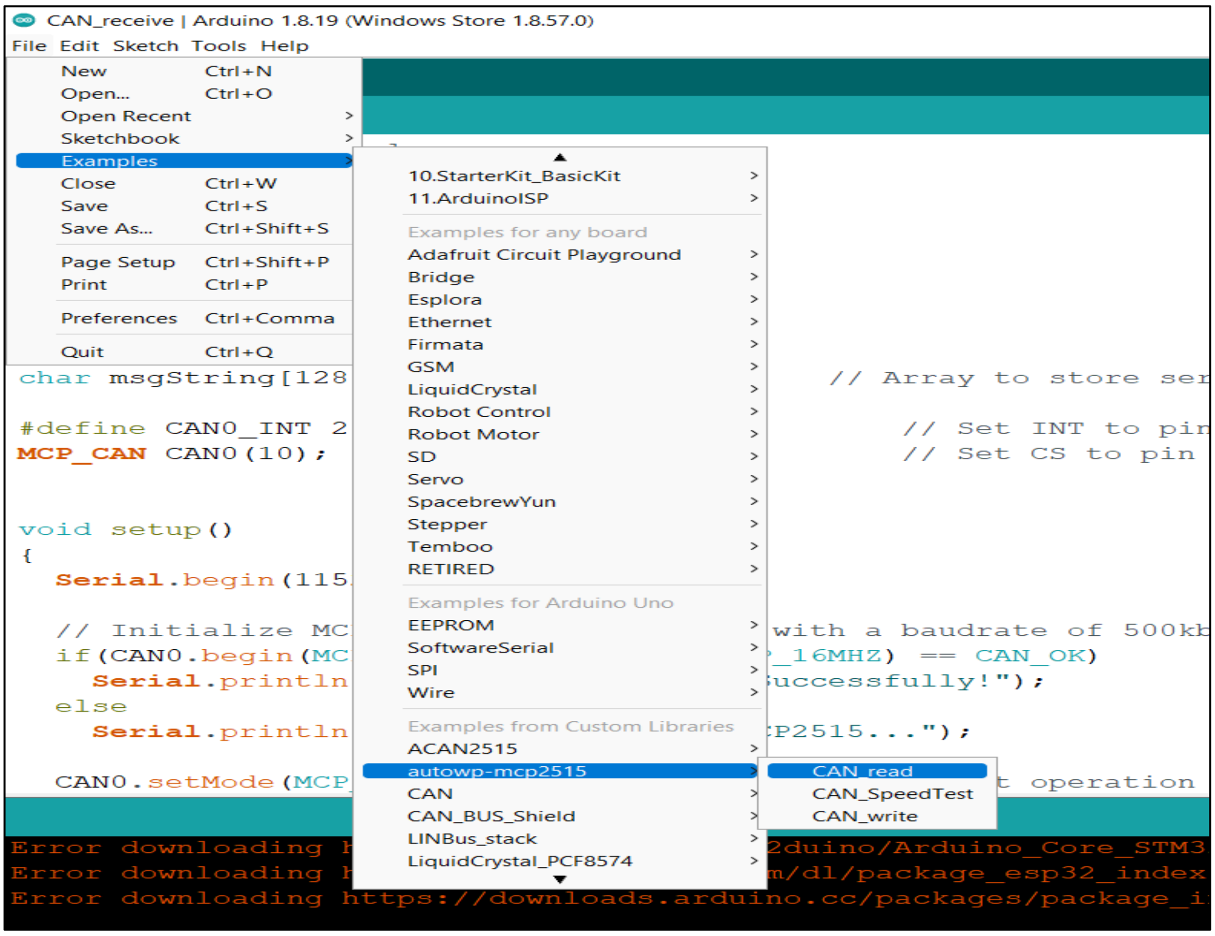
Step 2: In next step search for autowp-mcp2515, CAN and liquid crystal one by one click on Install tab and download latest version or according to need. In some cases we could founf errors while downloading the specific library for these we have to follow following steps.

Another way for installing libraries includes:

Step 4: We just have to search for the required library on Chrome in our case they are autowp-mcp2515, CAN and liquid crystal. Several result will dsplay on screen we have to select Github library because Github contains source code of particular library.

Step 5: Now inside Github click on Code → Download zip files as we are downloading zip files of particular library wait for some moment to download the zip file.

Step 6: Now in Arduino IDE on upper left corner select on Sketch → Include library → Add zip library now search for the location of zip file inside PC genrally it will be in Download folder click on file. This process automaticlly download library in Arduino IDE



```
char msgString[128                    // Array to store ser

#define CAN0_INT 2                       // Set INT to pin
MCP_CAN CAN0(10);                        // Set CS to pin


void setup()
{
  Serial.begin(115

  // Initialize MC                  with a baudrate of 500kb
  if(CAN0.begin(MC                  _16MHZ) == CAN_OK)
    Serial.println                  uccessfully!");
  else
    Serial.println                  P2515...");

  CAN0.setMode(MCP                  t operation

Error downloading h                  2duino/Arduino_Core_STM3
Error downloading h                  m/dl/package_esp32_index
Error downloading https://downloads.arduino.cc/packages/package_i
```

Fig 4.5 To open code of from custom library

Step 6: After downloading specific library on left upper corner select File → Examples → Examples from custom library → Desired library follow the code and we get particular code on IDE we can make changes in that code according to our need.

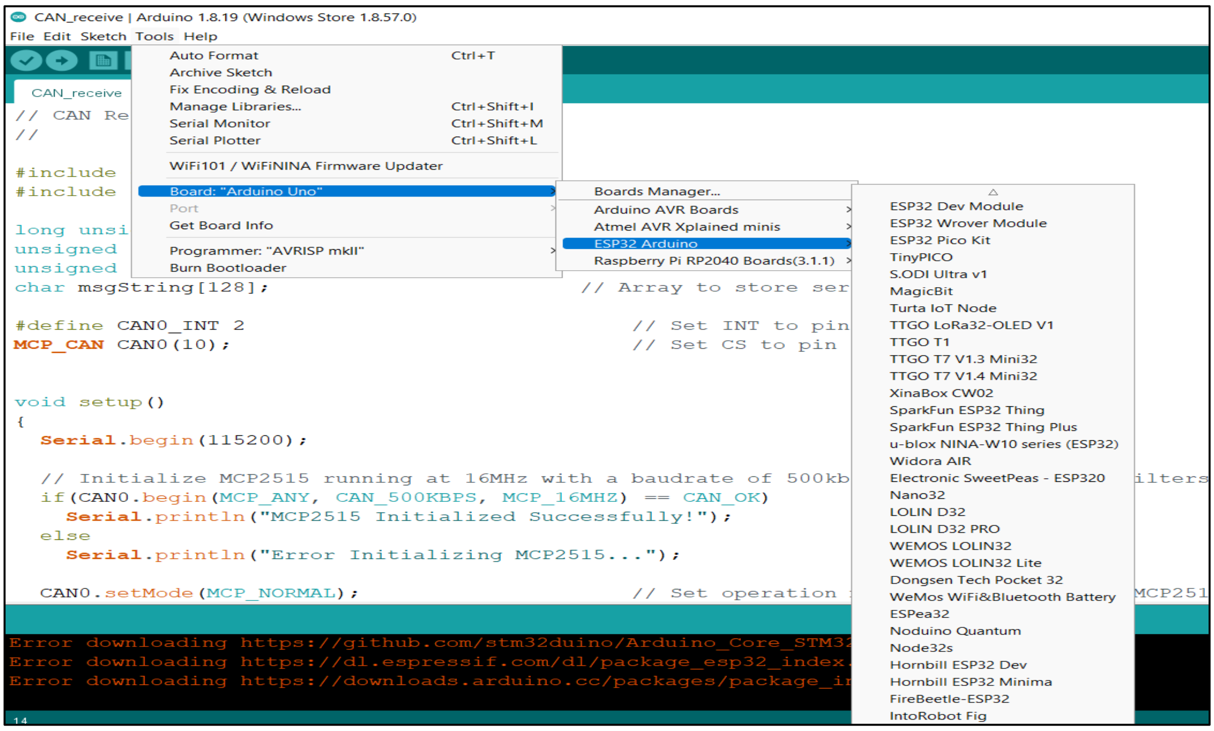
Coding Instructions :

Now we have to make changes in custom library code according to our conditions we have to simply transmit and receive some data at both end in our project we transmitting and receiving hexadecimal data on both sides. Thus we selected CAN send and CAN receive example also we have to remember that we can not use the code which contain MCP 2515 because we are using inbuilt CAN module for ESP32. Thus we have select custom library without CAN.

In following part we have to select Tools → Board → Desired specific board the board which we have connected to PC. Now adjust the com by selecting Tools → Port we have to select port which is not currently in use. Now Run the specific code after succesfully runnig we uploaded that code in particular board. Also observe the Serial monitor for observing the results we are getting on Microcontroller

Keypoints :

- Adjust Baudrate of serial monitor according to code.
- Adjust Bitrate of both codes which are send and receive generally Bit rate is 500Kbps
- Also we have to observe Serial plotter for detail analysis of data that we are receiving or generating on board.



```
// CAN Re
//

#include
#include

long unsi
unsigned
unsigned
char msgString[128];                          // Array to store ser

#define CAN0_INT 2                                    // Set INT to pin
MCP_CAN CAN0(10);                                     // Set CS to pin


void setup()
{
  Serial.begin(115200);

  // Initialize MCP2515 running at 16MHz with a baudrate of 500kb          ilters
  if(CAN0.begin(MCP_ANY, CAN_500KBPS, MCP_16MHZ) == CAN_OK)
    Serial.println("MCP2515 Initialized Successfully!");
  else
    Serial.println("Error Initializing MCP2515...");

  CAN0.setMode(MCP_NORMAL);                           // Set operation          MCP251
```

```
Error downloading https://github.com/stm32duino/Arduino_Core_STM32
Error downloading https://dl.espressif.com/dl/package_esp32_index.
Error downloading https://downloads.arduino.cc/packages/package_in
```

Fig 4.6 To select particular custom board

4.3.2 Cube IDE Setup:

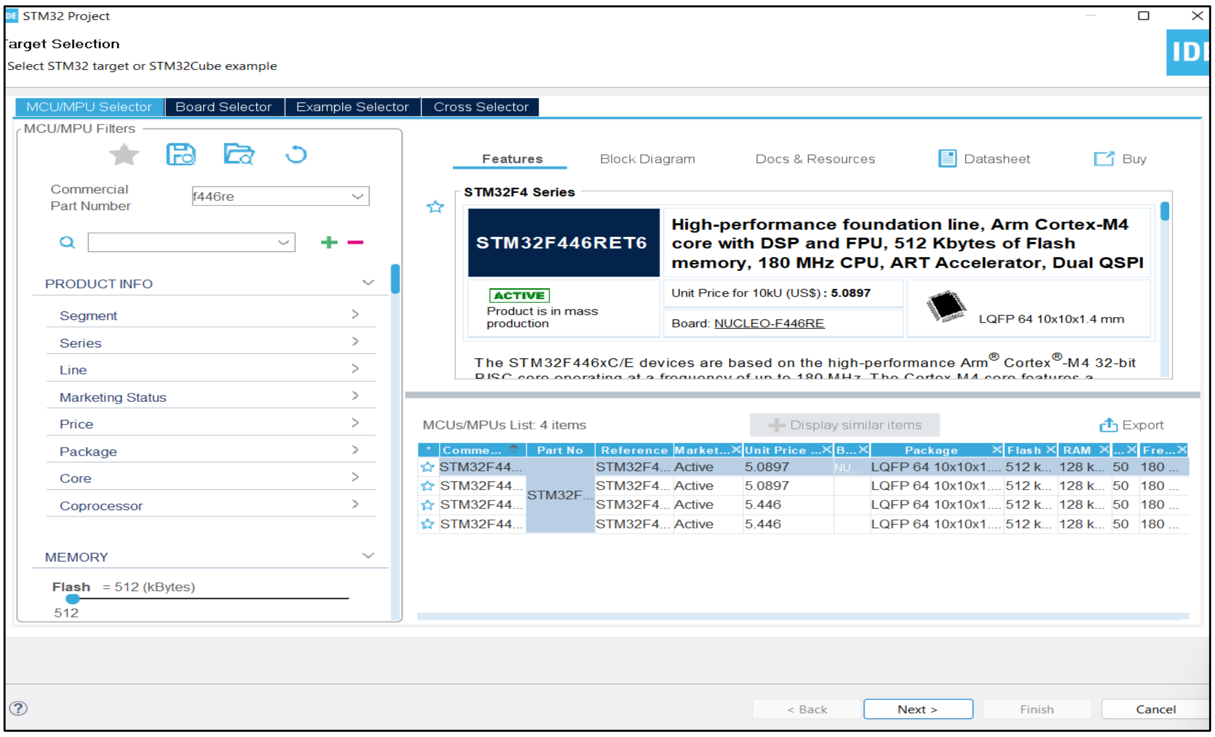
For the coding part of STM32 F446RE we are using open source official software of ST Microcontrollers called Cube IDE. Cube IDE provides proper environment for coding in STM32 F446RE. STM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, 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.

We have to install the Cube IDE from official website of ST Microcontrollers. It is necessary to have C/C++ interpreter & compiler available on our PC. If not available we have to download C/C++ interpreter & compiler from MinGW. Now install Cube IDE and follow the setup instruction for coding in STM32 F446RE microcontroller.

Steps of Cube IDE setup for STM32 F446RE:

Step 1: First of all we have to open installed Cube IDE it is advised to keep the Internet connection active while using Cube IDE as it fetched the data from server

Step 2: In another step we have to select on file which is on upper left part File → New → STM32 Project. The new tab called Target selection comes on screen


STM32 Project

arget Selection

Select STM32 target or STM32Cube example

Commercial Part Number: f446re

STM32F4 Series

STM32F446RET6

High-performance foundation line, Arm Cortex-M4 core with DSP and FPU, 512 Kbytes of Flash memory, 180 MHz CPU, ART Accelerator, Dual QSPI

ACTIVE
Product is in mass production

Unit Price for 10kU (US$) : **5.0897**

Board: NUCLEO-F446RE

LQFP 64 10x10x1.4 mm

The STM32F446xC/E devices are based on the high-performance Arm® Cortex®-M4 32-bit

MCUs/MPUs List: 4 items

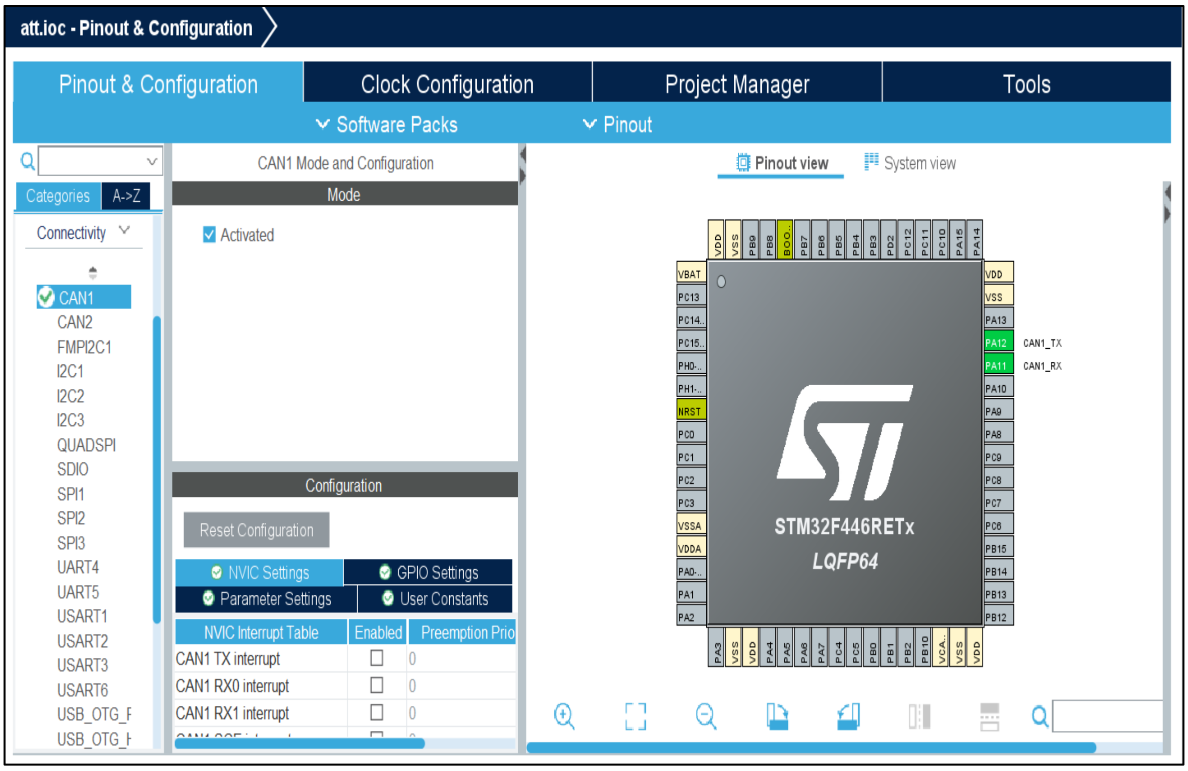
*	Comme...	Part No	Reference	Market...	Unit Price ...	B...	Package	Flash	RAM	...	Fre...
☆	STM32F44...	STM32F...	STM32F4...	Active	5.0897	NU...	LQFP 64 10x10x1....	512 k...	128 k...	50	180 ...
☆	STM32F44...		STM32F4...	Active	5.0897		LQFP 64 10x10x1....	512 k...	128 k...	50	180 ...
☆	STM32F44...		STM32F4...	Active	5.446		LQFP 64 10x10x1....	512 k...	128 k...	50	180 ...
☆	STM32F44...		STM32F4...	Active	5.446		LQFP 64 10x10x1....	512 k...	128 k...	50	180 ...

Flash = 512 (kBytes)

Fig 4.7 Searching for F446RE on Target Selection tab

Step 3: In Target Select tab we have to search for STM32 F446RE in commercial part number tab as shown in fig 4.6. Now we have to select active board from MCU list we are selecting STM32F446RET6 tab as it satisfies all criteria of our project and click on Next button. Save the file under specific name in newly open Project name tab IDE also provide option to select language C/C++ we are selecting C lang. after these select on Finish tab.

Step 4: It could take some time for rendering the UI and installing the zip files of respective board. The new tab called Pinout & Configuration will be open as shown in fig 4.7



Mode: ✓ Activated

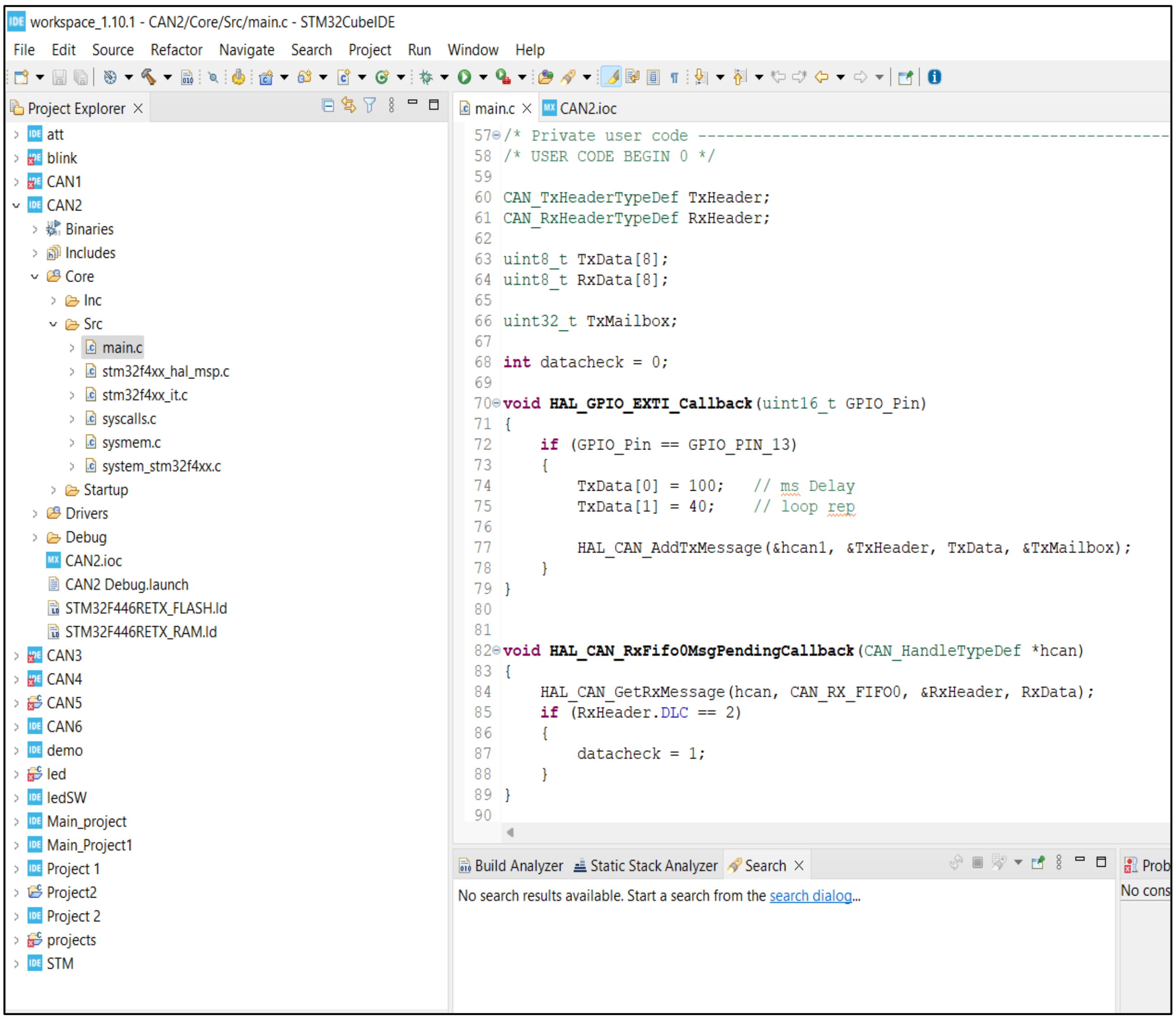
NVIC Interrupt Table	Enabled	Preemption Prio
CAN1 TX interrupt	☐	0
CAN1 RX0 interrupt	☐	0
CAN1 RX1 interrupt	☐	0

PA12: CAN1_TX
PA11: CAN1_RX

Fig 4.8 Pinout &Configuration tab settings

Step 5: In this step we have to configure the CAN for STM32F446RE as there are two CAN interfaces in STM32F446RE named CAN1 and CAN2. In our project we are using only one CAN node. To activate CAN1 we have to select Categories → Connectivity → CAN1 and click on activate. We also have to configure parameter & GPIO settings according to standards. We also have to activate interrupts, maintain baud rate and mange bitrate which can adjust with remaining 3 boards for proper CAN communication between node to node respectively.

Step 6: This step involves generation of code for that purpose we have to click on upper left corner Project → Generate code it will generate main.c file in respective folder. To explore that main.c code we have to search our project title name in Project explorer to open main.c we are following Project name(CAN) → Core → Src → main.c this step will generate the code for particular setting which we have done for CAN transmission with CAN library.



```c
/* Private user code ---------------------------------------------------------
/* USER CODE BEGIN 0 */

CAN_TxHeaderTypeDef TxHeader;
CAN_RxHeaderTypeDef RxHeader;

uint8_t TxData[8];
uint8_t RxData[8];

uint32_t TxMailbox;

int datacheck = 0;

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == GPIO_PIN_13)
    {
        TxData[0] = 100;   // ms Delay
        TxData[1] = 40;    // loop rep

        HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox);
    }
}


void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &RxHeader, RxData);
    if (RxHeader.DLC == 2)
    {
        datacheck = 1;
    }
}

```

No search results available. Start a search from the search dialog...

Fig: Code generation in Cube IDE

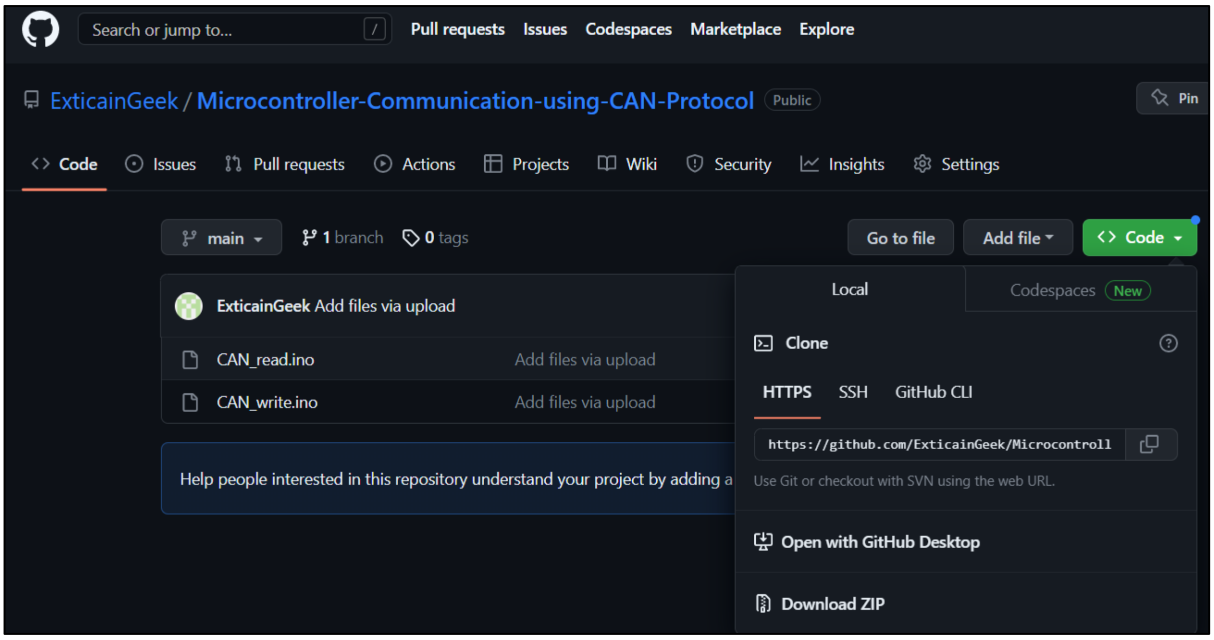
Now we have to do some necessary changes in generated code as we are sending hexadecimal data on CAN bus we have to declare function related to hexadecimal value. We will write that code in while loop as we need continues data generation from i.e. Hexadecimal numbers from STM32F446RE. For reference we can observe these values on serial monitor of Arduino IDE because serial monitor is common for all external microcontrollers we just have to select proper Com number to access it.

4.3.3 Code Access:

To access all codes we have uploaded all codes on Github repsitory named ExticainGeek/Microcontroller-Communication-using-CAN-Protocol where one can access all codes which are required for CAN communication between Arduino UNO(Slave), STM32F446RE, ESP32 & Raspberry pi pico. All codes are uploaded with proper titles for the convinience of user.

The link of Github Repsitory:

https://github.com/ExticainGeek/Microcontroller-Communication-using-CAN-Protocol.git also we can download zip files of all codes.



ExticainGeek Add files via upload

CAN_read.ino	Add files via upload
CAN_write.ino	Add files via upload

Help people interested in this repository understand your project by adding a

https://github.com/ExticainGeek/Microcontroll

Use Git or checkout with SVN using the web URL.

Fig 4.9 Our GitHub Repository of all codes related to the project

CHAPTER 5:

COMPONENT DESCRIPTION

5. COMPONENT DESCRIPTION

In this chapter we are going to analysed the Component by their characterstics, specifications and with respect to CAN terminology as we are using various microcontrollers it is important to be familiar with such new board. Thus in this chapter we are analysing following Microcontrollers, CAN Transceivers & LCD display module :

5.1 MICROCONTROLLERS:

As we are using several types of microcontrollers for CAN communication which are Arduino UNO, STM32F446RE, ESP32 and Raspberry pi pico. We are going to analysed these 4 microcontrollers in these descrption**.**

5.1.1 STM32 F446RE Microcontroller:

Definition:

The STM32F446RE microcontroller is based on the ARM Cortex-M4 processor, which provides advanced processing capabilities, including a high-performance 32-bit RISC CPU, floating-point unit (FPU), and digital signal processing (DSP) instructions, making it suitable for computation-intensive applications. It operates at a maximum frequency of up to 180 MHz and provides a wide range of flash memory sizes (up to 512 KB) and RAM sizes (up to 128 KB), allowing for ample storage and execution space for firmware and data.

CAN Relation:

The STM32F446RE MCU comes with multiple CAN interfaces, typically referred to as CAN1 and CAN2, which can be configured to send and receive CAN messages. These CAN interfaces are connected to specific pins on the MCU, and the pin configuration is an important aspect of using CAN communication on the STM32F446RE MCU.

The pin configuration for CAN communication on the STM32F446RE MCU involves assigning specific pins for CAN communication functions, such as CAN_TX (transmit) and CAN_RX (receive), as well as other pins for CAN-related features like CAN_STBY (standby), CAN_SILENT (silent mode), and CAN_WAKEUP (wake-up). The specific pins used for CAN communication may vary depending on the selected CAN interface (CAN1 or CAN2) and the STM32F446RE MCU package variant

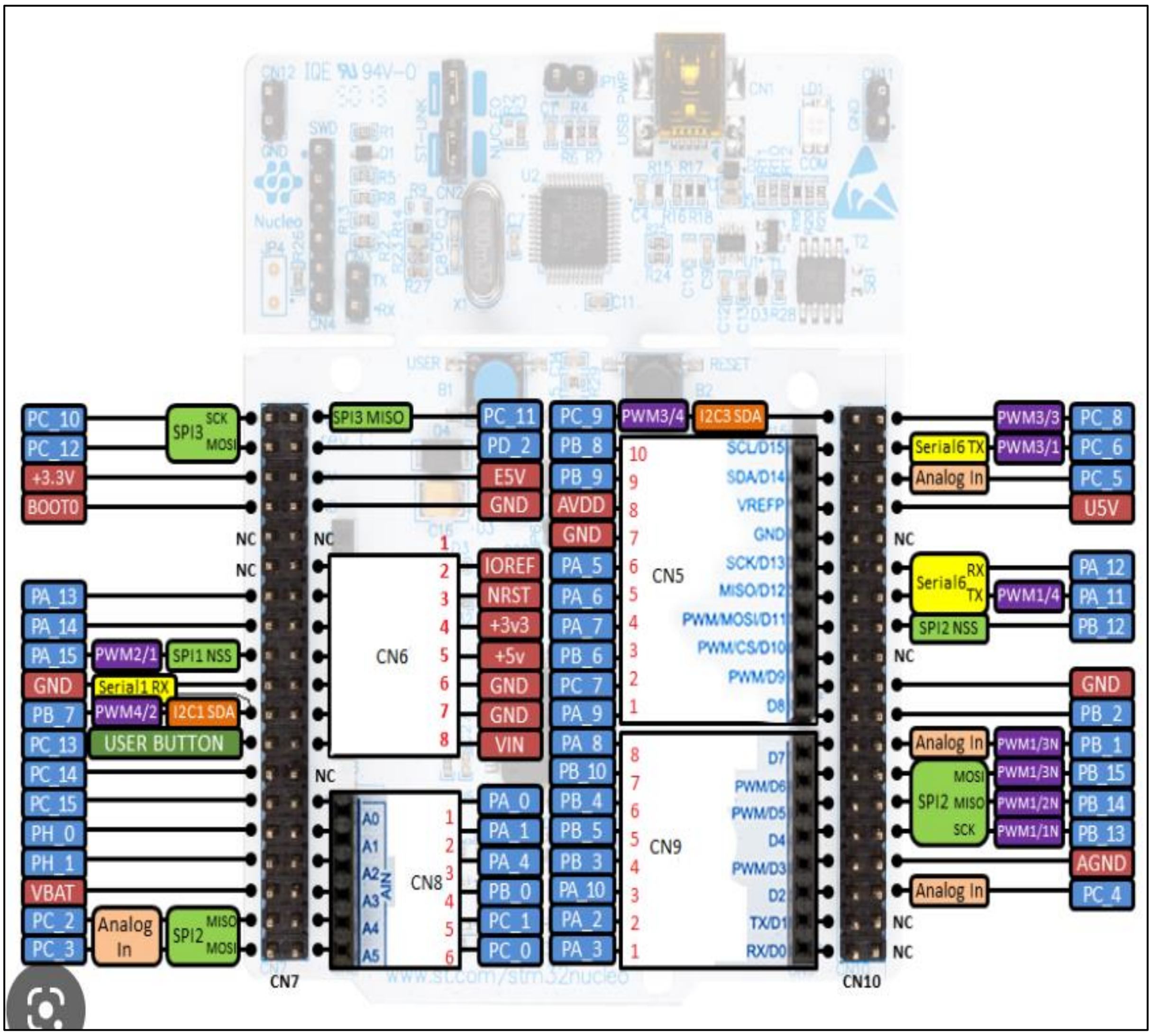


Fig 5.1 STM32F446RE Microcontroller Pinout

Features:

Integrated Debugging: It includes support for on-chip debugging with a built-in SWD (Serial Wire Debug) interface, making it easy to develop and debug applications.

Security: It has built-in hardware security features such as a CRC unit, a hardware random number generator, and memory protection features to enhance the security of the system.

Packaging: The STM32F446RE is available in various package options, including LQFP (Low Quad Flat Package) and UFBGA (Ultra Fine Ball Grid Array), to suit different application requirements.

5.1.2 ESP32 Microcontroller:

Definition:

The ESP32 is a highly popular, low-power, dual-core microcontroller unit (MCU) developed by Espressif Systems. It is widely used for a variety of Internet of Things (IoT) applications, including smart home devices, industrial automation, wearables, and more. The ESP32 is based on the Xtensa LX6 32-bit microprocessor, and it offers advanced Wi-Fi and Bluetooth capabilities, along with a rich set of peripheral interfaces, making it a versatile and powerful MCU for IoT applications.

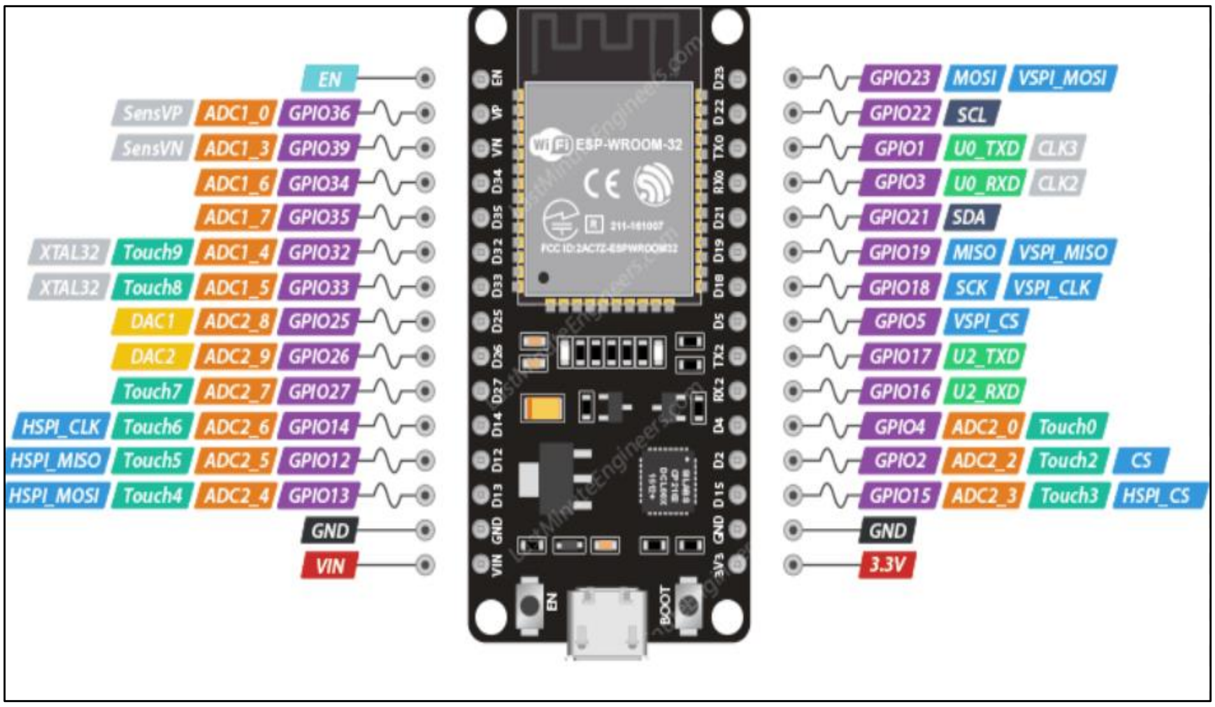


Fig 5.2 Esp32 Microcontroller & Pinout

Specification:

The ESP32 comes with built-in Wi-Fi and Bluetooth support, allowing it to connect to wireless networks, communicate with other devices, and transfer data wirelessly. The Wi-Fi support includes 802.11 b/g/n/e/i protocols, while the Bluetooth support includes Bluetooth Classic (BR/EDR) and Bluetooth Low Energy (BLE) protocols. This makes the ESP32 suitable for a wide range of wireless communication scenarios in IoT applications. In addition to wireless connectivity, the ESP32 offers a rich set of peripheral interfaces, including multiple UART, SPI, I2C, and GPIO pins, as well as support for analog-to-digital conversion (ADC).

5.1.3 Raspberry Pi Pico:

Introduction:

The Raspberry Pi Pico is a microcontroller development board developed by the Raspberry Pi Foundation. It is based on the Raspberry Pi RP2040 microcontroller, which is a powerful and versatile microcontroller chip designed specifically for embedded systems and IoT applications**.**

Specification:

The Raspberry Pi Pico features a compact form factor, with a single-board design that includes a microcontroller, flash memory, and various I/O pins for connecting to external sensors, actuators, and other electronic components. It also includes built-in support for popular communication protocols such as UART, SPI, I2C, PWM, and GPIO, making it suitable for a wide range of projects and applications. The Raspberry Pi Pico is programmable in C/C++ and Micro Python, and it supports a variety of development tools, including the official Raspberry Pi Pico SDK, Thonny and Visual Studio code.

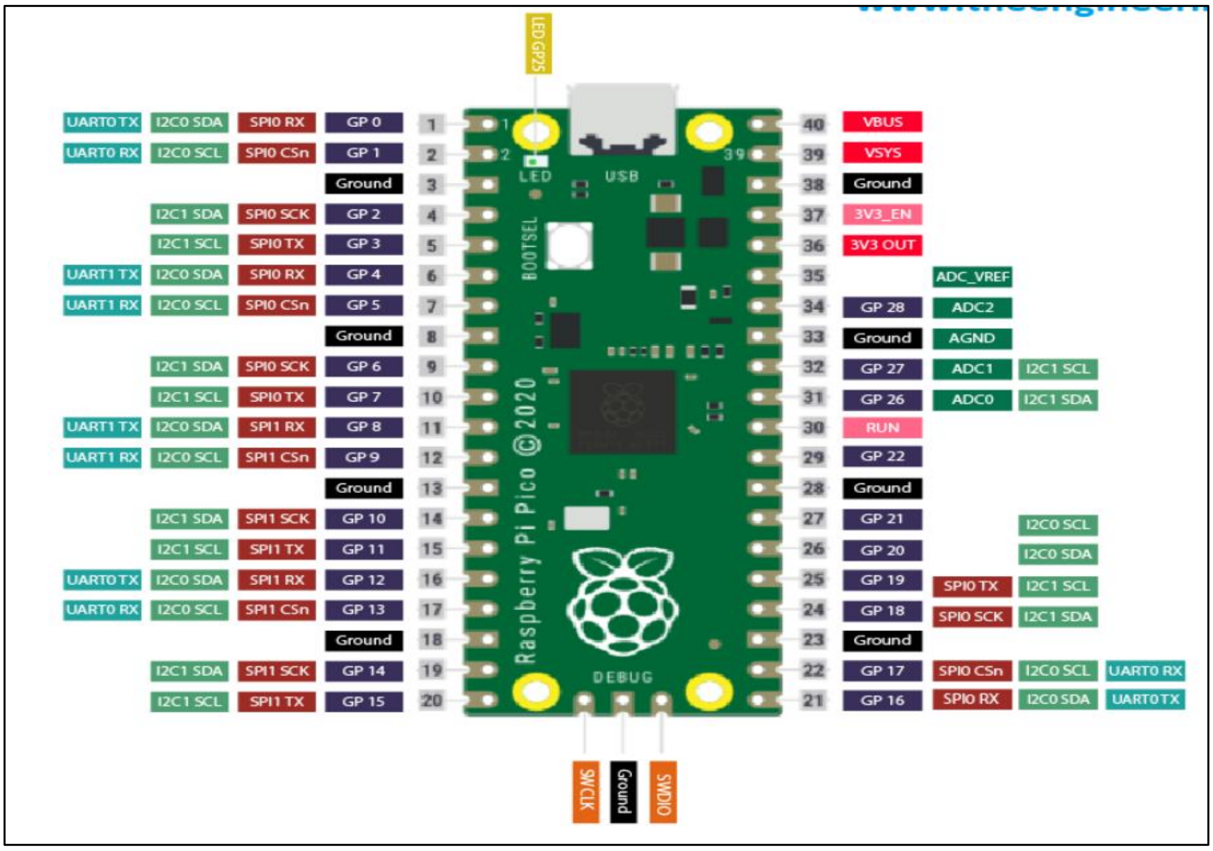


Fig 5.3 Raspberry Pi Pico & pinout

5.1.4 Arduino UNO:

Introduction:

The Arduino Uno is a popular microcontroller development board that is part of the Arduino ecosystem. It is based on the Atmega328P microcontroller chip and provides a simple and easy-to-use platform for prototyping and building electronic projects.

Specification:

The Arduino Uno features a compact form factor with a single-board design that includes a microcontroller, flash memory, and various I/O pins for connecting to external sensors, actuators, and other electronic components. It also includes built-in support for popular communication protocols such as UART, SPI, I2C, and GPIO, making it suitable for a wide range of projects and applications. The Arduino Uno is programmable using the Arduino programming language, which is based on C/C++, and the Arduino Integrated Development Environment (IDE).

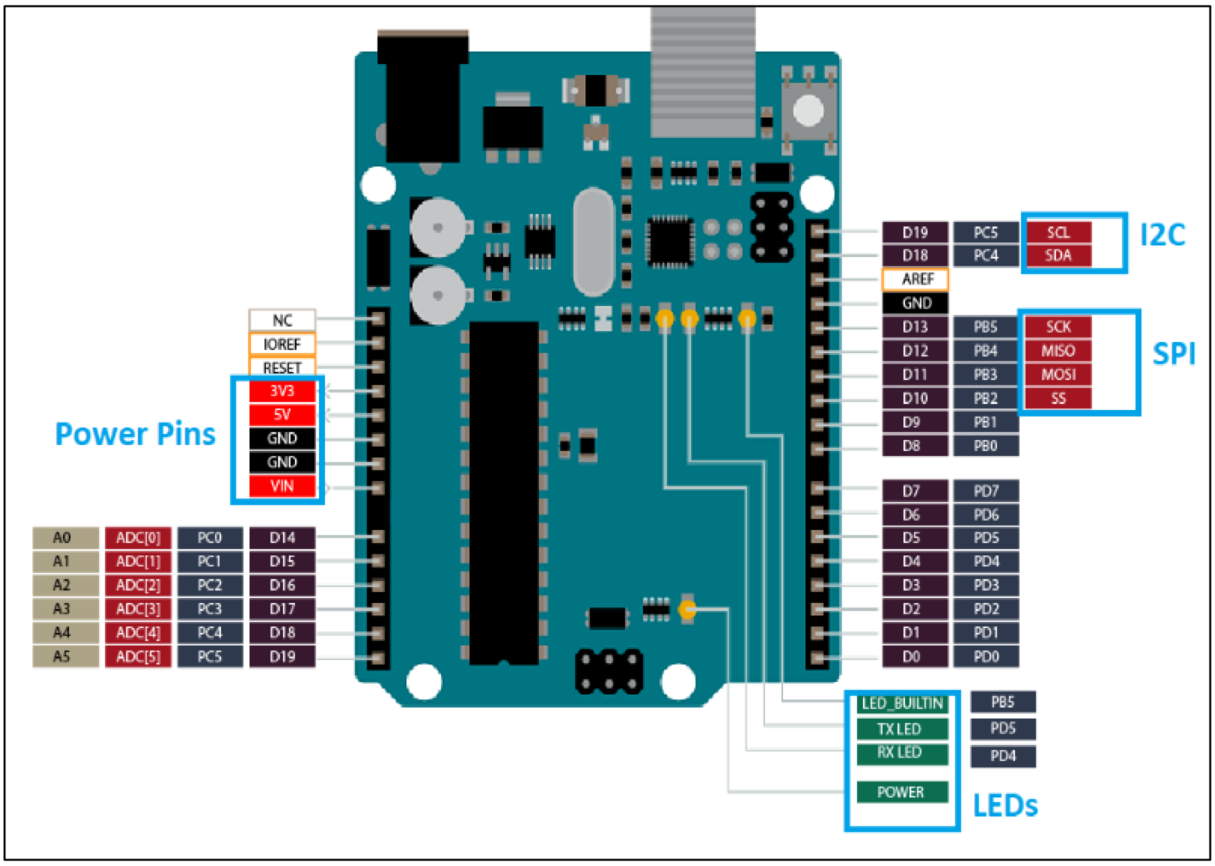


Fig 5.4 Arduino UNO & pinout

5.2 CAN Transceivers:

CAN transceivers are electronic devices that provide the physical interface between a CAN controller (such as a microcontroller or other device) and the CAN bus. They handle the transmission and reception of CAN messages, as well as voltage level shifting, bus arbitration, and other tasks required for CAN communication. In this project we are using MCP2515 & TJA 1050 thus we will be analysed this two CAN Transceivers as following:

5.2.1 MCP2515:

Introduction:

The MCP2515 is a standalone CAN (Controller Area Network) controller chip developed by Microchip Technology. It is widely used in embedded systems for implementing CAN communication using a microcontroller or other microprocessor. The MCP2515 provides an SPI (Serial Peripheral Interface) interface for communication with the host microcontroller and supports both CAN 2.0A and CAN 2.0B standards.

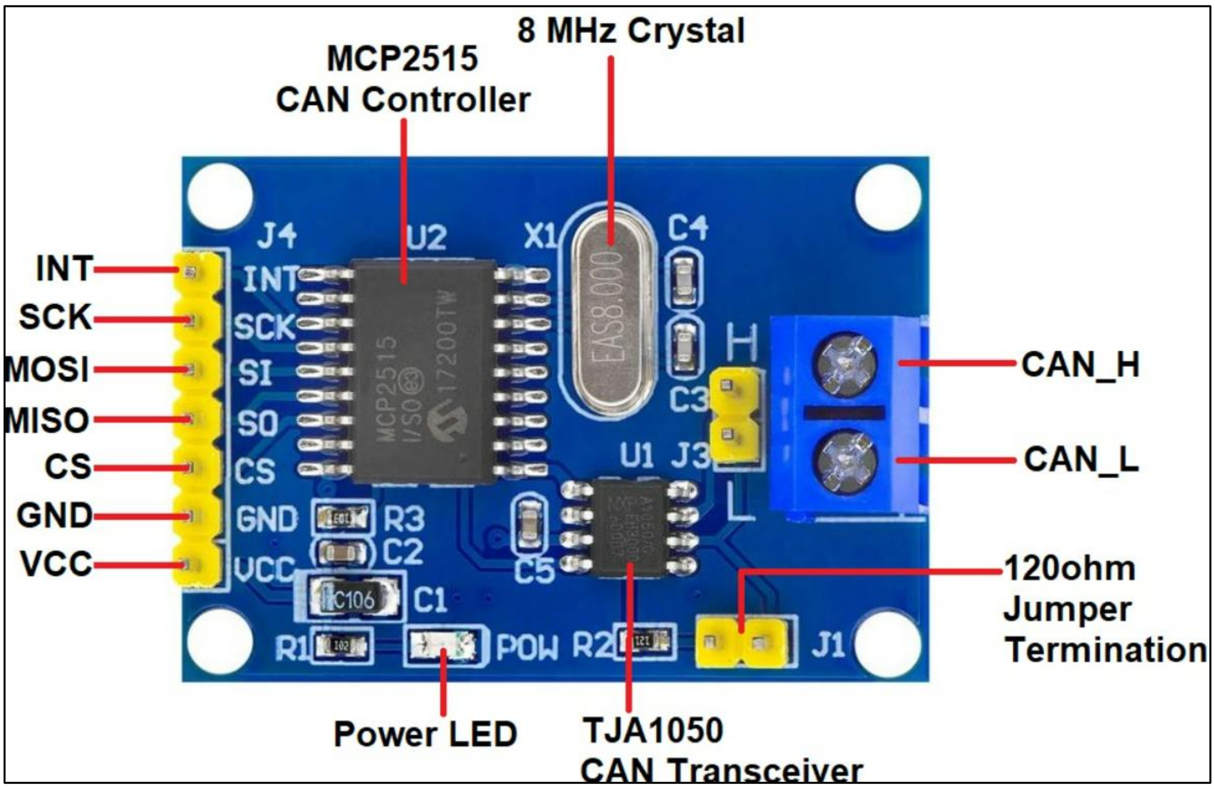


Fig 5.5 MCP2515 & Pinout

Features & Specifications:

1. CAN Communication: The MCP2515 provides support for both CAN 2.0A and CAN 2.0B standards, which are widely used in automotive, industrial, and other embedded systems for reliable and robust communication between multiple devices over a shared bus.

2. SPI Interface: The MCP2515 communicates with the host microcontroller using the SPI interface, which is a popular synchronous serial communication protocol used in many microcontroller-based systems. The SPI interface allows for fast and efficient communication between the MCP2515 and the host microcontroller.

3. Integrated CAN Transceiver: The MCP2515 includes an integrated CAN transceiver, which provides the physical interface between the MCP2515 and the CAN bus. The integrated transceiver simplifies the design of CAN communication systems by eliminating the need for an external transceiver.

4. Buffering and Filtering: The MCP2515 includes multiple receive buffers and filtering options, which allow for efficient handling of incoming CAN messages. The buffering and filtering options help reduce the overhead on the host microcontroller and allow for efficient handling of large volumes of CAN messages.

5. Message Filtering and Masking: The MCP2515 allows for flexible message filtering and masking, which enables the selection of specific CAN messages based on their identifiers, data length, and data content. This helps reduce the processing overhead on the host microcontroller and allows for efficient handling of only relevant CAN messages.

6. Error Handling: The MCP2515 includes built-in error detection and error handling features, such as error detection on transmitted and received messages, error flag reporting, and automatic retransmission of failed messages. These features help ensure reliable and error-free communication on the CAN bus.

5.2.2 TJA1050:

Introduction:

The TJA1050 is a popular standalone CAN (Controller Area Network) transceiver chip developed by NXP Semiconductors. It is used for interfacing with CAN bus networks, enabling communication between different devices over a shared CAN bus. The TJA1050 is compliant with the CAN 2.0A and CAN 2.0B standards and supports both 11-bit and 29-bit CAN identifiers. It provides a physical interface between the microcontroller or other host processor and the CAN bus, converting the digital signals from the microcontroller into differential voltage levels that are used for communication over the CAN bus.

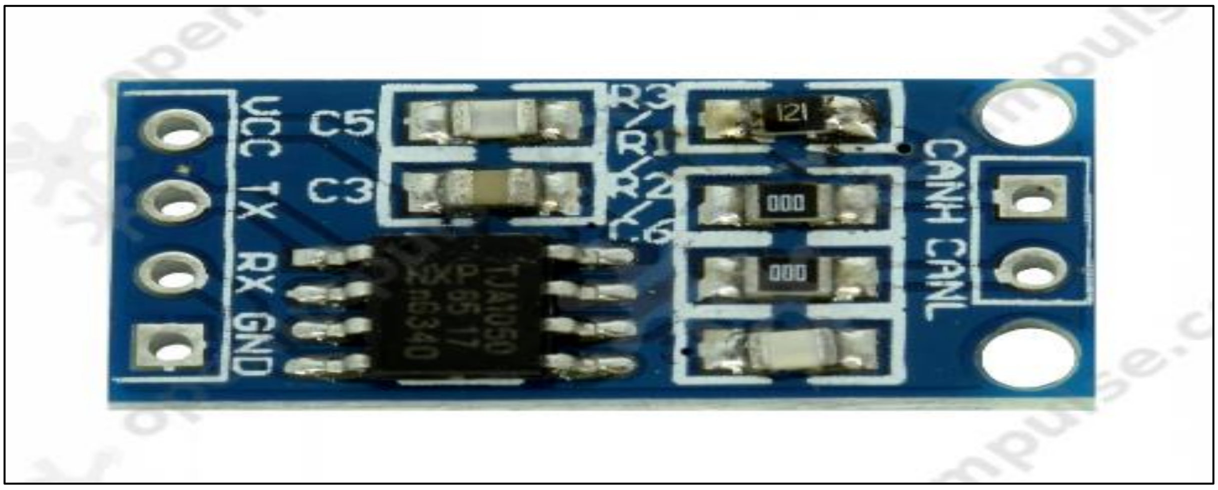


Fig 5.6 TJA 1050

Features & Specifications:

1. High-Speed Communication: The TJA1050 supports high-speed communication on the CAN bus, allowing for data rates of up to 1 Mbps. This makes it suitable for applications that require fast and efficient communication, such as automotive systems.

2. Wide Supply Voltage Range: The TJA1050 operates over a wide supply voltage range of typically 4.5V to 5.5V, making it compatible with a variety of microcontrollers and other host processors.

3. Differential Signaling: The TJA1050 uses differential signaling to transmit and receive data over the CAN bus, which provides good noise immunity and allows for reliable communication in noisy environments.

5.3 LCD Display & I2C Module:

When used together, an LCD module and an I2C module can enable a microcontroller or other host processor to communicate with an LCD display using the I2C protocol. The I2C module acts as an interface between the microcontroller and the LCD module, translating the commands and data from the microcontroller into I2C messages that can be understood by the LCD module. This allows for simple and efficient communication between the microcontroller and the LCD display, reducing the number of wires required for communication and simplifying the overall system design.

LCD Module: An LCD module is a type of display that uses liquid crystals to produce visual output. It typically consists of a display panel, a controller, and other components such as backlighting and interface circuitry. LCD modules are commonly used for displaying text, numbers, symbols, and graphics in various electronic devices, including embedded systems.

I2C Module: I2C, also known as I²C or Inter-Integrated Circuit, is a communication protocol that allows for communication between microcontrollers and other devices, such as sensors, modules, and peripherals. I2C uses a two-wire serial interface for communication, typically labeled as SDA (Serial Data Line) and SCL (Serial Clock Line). By providing the necessary circuitry for sending and receiving I2C data.

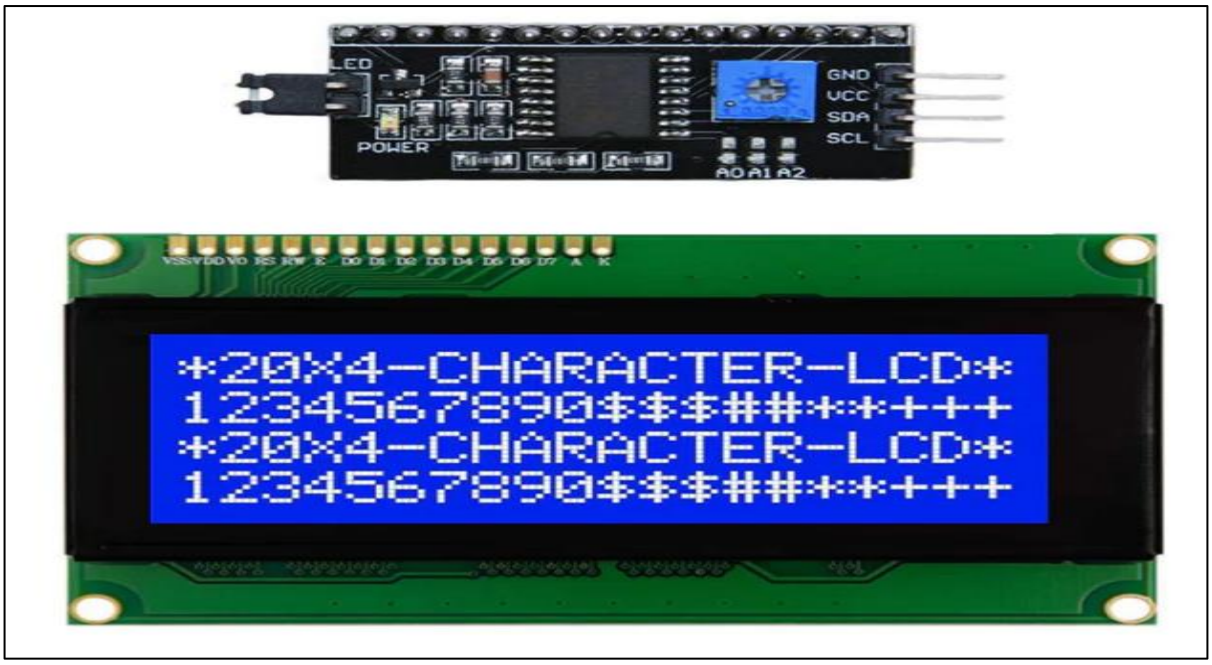


Fig 5.7 I2C Module & LCD Display(20×4)

CHAPTER 6:

APPLICATIONS

6. APPLICATIONS

1. **Automotive Systems:** CAN is widely used in automotive systems for communication between various components, such as engine control units (ECUs), transmission control units (TCUs), body control modules (BCMs), and instrument clusters. Microcontrollers can use CAN to communicate critical data, such as sensor readings, control commands, and diagnostic information, among different parts of the vehicle for seamless operation.

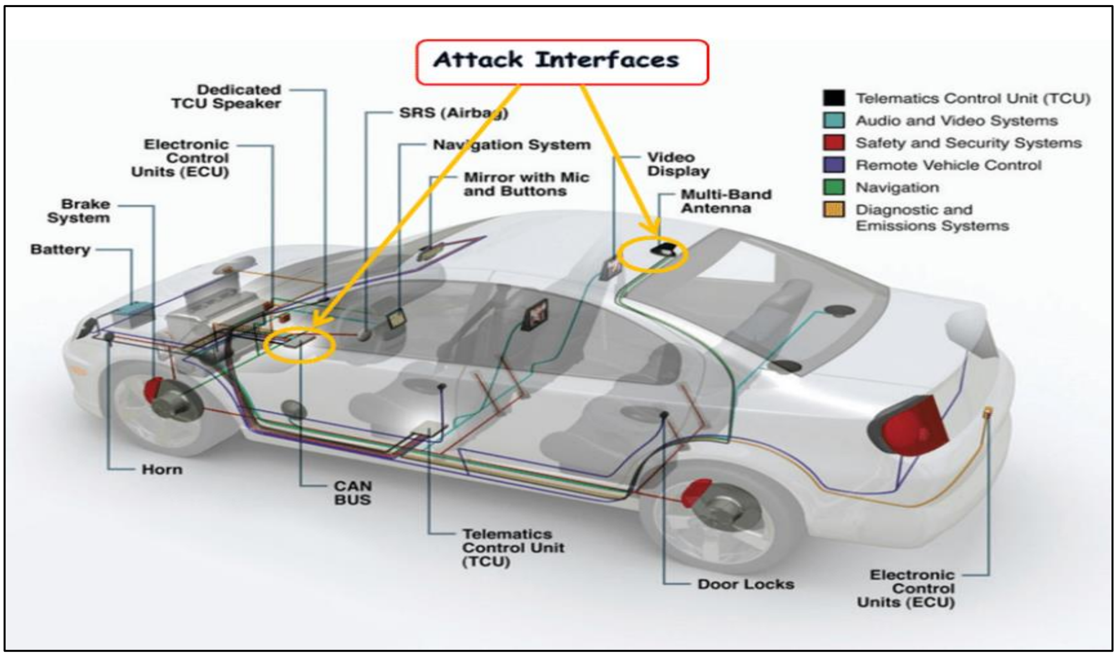

Attack Interfaces
Dedicated TCU Speaker
SRS (Airbag)
Electronic Control Units (ECU)
Navigation System
Mirror with Mic and Buttons
Video Display
Multi-Band Antenna
Brake System
Battery
Horn
CAN BUS
Telematics Control Unit (TCU)
Door Locks
Electronic Control Units (ECU)
⬛ Telematics Control Unit (TCU)
🟦 Audio and Video Systems
🟥 Safety and Security Systems
🟪 Remote Vehicle Control
🟩 Navigation
🟧 Diagnostic and Emissions Systems

Fig6.1 CAN Protocol Implementation in Automobiles

2. **Advanced Driver Assistance Systems (ADAS):** Microcontrollers in ADAS applications, such as collision avoidance systems, adaptive cruise control, and lane departure warning systems, can use CAN to exchange sensor data, control commands, and status information among different components of the ADAS system. This enables timely and accurate communication between the microcontrollers, allowing for effective operation of the ADAS system.

3. **Industrial Automation**: CAN is also commonly used in industrial automation systems to enable communication between different industrial devices, such as programmable logic controllers (PLCs), human-machine interfaces (HMIs), motor drives, and sensors.

Microcontrollers can use CAN to exchange data related to process control, monitoring, and fault detection, allowing for efficient and reliable communication in industrial environments.

4. **Medical Devices:** CAN protocol is utilized in medical devices, such as patient monitoring systems, infusion pumps, and surgical instruments, to facilitate communication among different devices and components. Microcontrollers can use CAN to transmit real-time data, alarms, and control commands within medical devices, enabling seamless integration and interoperability in healthcare settings.

5. **Home Automation:** CAN can be used in home automation systems for communication among different devices, such as smart thermostats, lighting controls, security systems, and smart appliances. Microcontrollers can utilize CAN to enable communication and coordination among various smart devices in a home automation network, allowing for centralized control and automation of home systems.

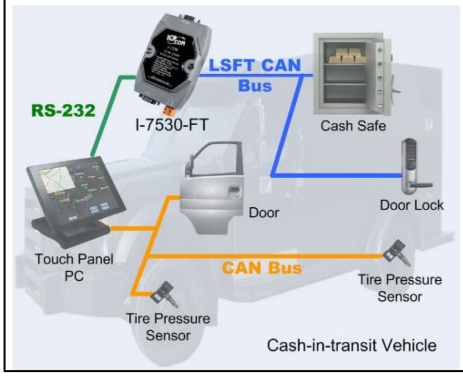
6. **Aerospace and Defense:** CAN is also used in aerospace and defense applications for communication between different subsystems and components of aircraft, spacecraft, and military equipment. Microcontrollers can use CAN to exchange data related to navigation, communication, sensor readings, and control commands in aerospace and defense systems, ensuring reliable and secure communication in demanding environments.

7. **Internet of Things (IoT):** CAN can be used as a communication protocol in IoT applications where there is a need for reliable, robust, and deterministic communication between embedded devices. Microcontrollers can use CAN to enable communication among different IoT devices, such as sensors, actuators, and gateways, in industrial automation, smart cities, and smart agriculture applications.

8. **Energy Management Systems:** CAN can be used in energy management systems for communication between microcontrollers in devices such as solar inverters, battery management systems, and smart grid equipment. Microcontrollers can use CAN to exchange data related to power generation, consumption, and storage, allowing for efficient management and control of energy resources in smart grid and renewable energy applications.

Few ongoing projects in different part of World on CAN & Microcontroller concept:

Cash-in-transit Vehicle



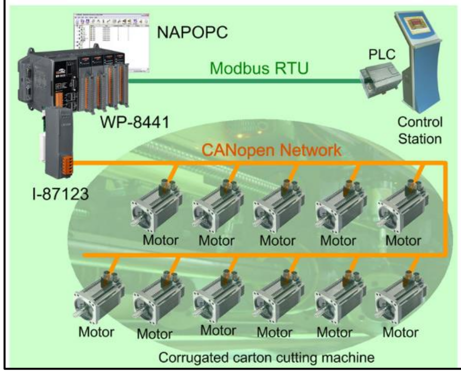
Location: England, United Kingdom

Product: I-7530-FT

Description: A telematics and vehicle control system need to be closely integrated with each other by some data-exchange interfaces. LSFT (Low speed fault tolerance) CAN is a kind of familiar interfaces in the autotronics applications. An I-7530-FT is specially designed for solving the interface transformation between LSFT CAN and RS-232. Through it, the cash-in-transit of the manufacturer can control the door of the cash safely and monitor the conditions of the back door directly and reliably on the telematics.

Figure 6.2 England Based Cash in transit Vehicles project

Corrugated Carton Cutting Machine



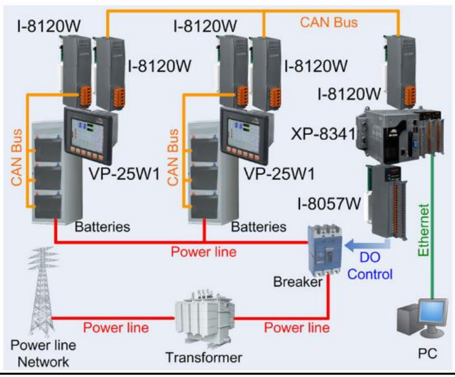
Location: Taichung, Taiwan

Product: WP-8441, I-87123

Description: In this case, the orientation and cutting speeds seriously affect the quantity of output. Because all of the cutting knives and rollers must be controlled by 31 motors, the customer selects the CANopen motors to do that. The WP-8441 and I-87123 play the role of a CANopen master to control all of the motors simultaneously by the CANopen features of the synchronization and high speed. By using this architecture, all of the motors can quickly move to the target position at the same time by just sending one command.

Figure 6.3 Taiwan based corrugated carton cutting machine

Energy Storage System



Location: Guangdong, China

Product: I-8120W, I-8057W, VP-25W1, XP-8341

Description: This system can improve the usage efficiency of the electrical power. During the off-peak time of the electricity use, the unused electricity can be stored in the battery. When the peak time is coming, these batteries supply the power to the electric grid. The customer utilizes one of the I-8120Ws in VP-25W1 to monitor the battery status, and another is used to feedback these data to the XP-8341. The XP-8341 transmits the status to the PC via Ethernet and control the charge time by the breaker.

Figure 6.4 China Based Energy storage system

CHAPTER 7

ADVANTAGES & DISADVANTAGES

7. ADVANTAGES & DISADVANTAGES

7.1 ADVANTAGES:

1. **Robustness and Reliability**: CAN is designed for high-reliability communication in harsh environments, making it suitable for applications that require robustness, such as automotive and industrial systems. CAN utilizes differential signaling, which provides noise immunity and allows for long-distance communication without signal degradation.

2. **Easy Integration**: Many microcontroller platforms, including STM32F446RE, Arduino Uno, Raspberry Pi Pico, and ESP32, have built-in hardware support for CAN communication, making it easy to integrate CAN communication into the system without the need for additional external components or modules.

3. **Scalability:** CAN supports multi-node communication, allowing for the connection of multiple microcontrollers and other devices in a network, making it highly scalable. It also supports both peer-to-peer and broadcast communication, enabling efficient communication in various network topologies.

4. **Deterministic Communication**: CAN provides deterministic communication, where each message has a priority level (message identifier) that determines its order of transmission. This allows for precise and predictable communication timing, making it suitable for real-time applications that require time-sensitive data exchange.

5. **High Data Rates**: CAN supports high data rates, typically ranging from 125 kbps to 1 Mbps, allowing for fast data transfer between microcontrollers or other devices in the network. Higher data rates can be achieved using CAN FD (Flexible Data Rate), a newer version of CAN protocol that allows for higher baud rates and larger data payloads.

6. **Noise Immunity:** CAN uses differential signaling, which provides excellent noise immunity, allowing for reliable communication in noisy environments commonly found in industrial or automotive settings.

7. **Interoperability:** CAN is a widely used and standardized protocol, with standardized communication interfaces, frame formats, and error-handling mechanisms. This ensures interoperability between different microcontroller platforms and devices from different manufacturers, making it easy to integrate CAN into a wide range of applications.

7.2 DISADVANTAGES:

STM32F446RE:

- Hardware Complexity: Implementing CAN communication on STM32F446RE microcontroller may require additional hardware components, such as CAN transceivers and external crystal oscillators, which can add complexity to the system design and may require additional board space.
- Software Configuration: Configuring CAN communication on STM32F446RE may require understanding and configuration of various registers and settings in the microcontroller's CAN peripheral, which can be complex for users who are not familiar with the CAN protocol or STM32 microcontroller family.

Arduino Uno:

- Limited Hardware Support: Arduino Uno does not have built-in CAN hardware support, so using CAN communication would require additional hardware components, such as CAN shields or external CAN modules, which may add complexity to the system and increase costs.
- Limited Processing Power: Arduino Uno is based on an ATmega328P microcontroller, which has limited processing power and memory compared to some other microcontroller platforms. This may limit the performance and capabilities of CAN communication in terms of data rates, message handling, and error detection and correction.

Raspberry Pi Pico:

- Lack of Built-in CAN Support: Raspberry Pi Pico does not have built-in CAN hardware support, so using CAN communication would require additional hardware components, such as CAN shields or external CAN modules, which may add complexity to the system and increase costs.
- Limited GPIO Pins: Raspberry Pi Pico has a limited number of GPIO pins, which may limit the available pins for CAN communication, especially if other peripherals or sensors are also used in the system.

ESP32:

Pin Conflicts: ESP32 is a popular microcontroller platform that has built-in CAN hardware support. However, some ESP32 development boards may have pin conflicts with other peripherals, such as SPI or I2C, which may require careful pin configuration and management to avoid conflicts.

Power Consumption: ESP32 is known to have higher power consumption compared to some other microcontrollers, which may be a consideration in battery-powered or low-power applications that use CAN communication.

There may be some potential disadvantages when used with specific microcontroller platforms like STM32F446RE, Arduino Uno, Raspberry Pi Pico, and ESP32, including hardware complexity, limited hardware support, limited processing power, pin conflicts, and power consumption, which should be carefully considered when planning to use CAN communication in these platforms. It's important to carefully review the documentation and specifications of the specific microcontroller platform and CAN hardware components to ensure compatibility and proper configuration for reliable and efficient CAN communication.

CHAPTER 8:

CONCLUSION AND SCOPE

8. CONCLUSION & SCOPE

8.1 CONCLUSION:

In conclusion, the CAN (Controller Area Network) protocol offers several advantages for communication in microcontroller platforms such as STM32F446RE, Arduino Uno, Raspberry Pi Pico, and ESP32. These advantages include robust and reliable communication, scalability, real-time capabilities, high data rates, message prioritization, easy integration, and wide industry support. CAN protocol is well-suited for applications that require reliable and efficient communication, especially in industrial or automotive environments.

However, it's important to carefully review the documentation and specifications of the specific microcontroller platform and CAN hardware components to ensure proper configuration and compatibility for successful implementation of CAN communication. Overall, CAN protocol is a powerful and widely-used communication protocol for microcontroller platforms, offering numerous benefits for various applications.

8.2 FUTURE SCOPE:

There is huge scope for CAN protocol as it reliably communicate between microcontrollers without any distortion with cost efficiency and very high data rates. The Common fields od scope of particular project in future are:

- Robotics
- Industrial & Home automation
- Aviation Industry
- Space Industry
- Defence Industry
- Medical Industry
- Electric Vehicles
- Broadcasting & Telecommunication Industry .

REFERENCES

REFERENCES

[1] Hyeran Mun, Kyusuk Han and Dong Hoon Lee, "Ensuring Safety and Security in CAN-Based Automotive Embedded Systems: A Combination of Design Optimization and Secure Communication", *proc IEEE trans-actions on vehicular technology*, vol. 69, no. 7, July 2020.

[2] de Andrade, R.; Hodel, K. N.; Justo, J. F.; Laganá, A. M.; Santos, M. M.; Gu, Z. (2018). "Analytical and Experimental Performance Evaluations of CAN-FD Bus". *IEEE Access*. 6: 21287–21295. doi:10.1109/ACCESS.2018.2826522.

[3] S. Woo, H. J. Jo, I. S. Kim and D. H. Lee, "A practical security architecture for in-vehicle CAN-FD", *IEEE Trans. Intell. Transp. Syst.*, vol. 17, pp. 2248-2261, Aug. 2016.

[4] Swathy Lakshmi; Renjith H Kumar, “Secure Communication between Arduinos using Controller Area Network(CAN) Bus”,IEEE, Kollam, India, 16-18 December 2022

[5] ESP32 Overview by Espressif Systems, (Shanghai) Co., Ltd, [online] Available: https://www.espressif.com/en/products/hardware/esp32/overview.

[6] Louis E. FrenzelJr, “Controller Area Network (CAN)’’, *Handbook of Serial Communications Interfaces*, Science Direct, 25 May 1016.

[7] Y. Kuang, "Communication between PLC and Arduino Based on Modbus Protocol", *International Conference on Instrumentation and Measurement Computer Communication and Control*, pp. 370-373, 2014.

[8] L. P. Pinho, F. Vasques, E. Tovar, "Integrating inaccessibility in response time analysis of CAN networks." Proceedings of the IEEE International Workshop on Factory Communication Systems, Porto, Portugal, 2000, pp. 77-84.

[9] "Datasheet - STM32F446xC/e - arm® cortex®-M4 32-bit MCU+FPU 225 DMIPS up to 512 KB flash/128+4 KB RAM USB OTG HS/FS seventeen TIMs three ADCs and twenty communication Available: https://www.st.com/resource/en/datasheet/stm32f446re.pdf.