PROJECT REPORT

on

Group no 12

BareMetal implementation of CAN-Based Data Communication with RTOS on STM32F407VGT6

Submitted in partial fulfillment for the award of

Post Graduate Diploma in
EMBEDDED SYSTEMS AND DESIGN

From C-DAC, ACTS (Pune)

Guided by:

Mr. Tarun Bharani

Presented by:

Ms. Anchal Maurya	**240340130007**
Ms. Maithili Balkawade	**240340130013**
Mr. Digpal Singh Bisht	**240340130015**
Mr. Harsh Bhati	**240340130019**
Mr. Shubham Jadhav	**240340130043**

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

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Ms. Anchal Maurya	**240340130007**
Ms. Maithili Balkawade	**240340130013**
Mr. Digpal Singh Bisht	**240340130015**
Mr. Harsh Bhati	**240340130019**
Mr. Shubham Jadhav	**240340130043**

Have successfully completed their project on

BareMetal implementation of CAN-Based Data Communication with RTOS on STM32F407VGT6

Under the guidance of

Mr. Tarun Bharani

Project Guide **Project Supervisor**

Acknowledgment

The success of this project, titled "**BareMetal implementation of CAN-Based Data Communication with RTOS on STM32F407VGT6**" is due to the invaluable guidance and support I received from many individuals. I am deeply grateful for their assistance, which played a crucial role in completing my work.

I am deeply grateful to **Mr. Tarun Bharani** for his expert guidance, continuous supervision, and invaluable insights throughout the project. His unwavering support was crucial to the successful completion of this work.

I am also thankful to **Ms. Risha P.R.** and all the staff members for their cooperation and assistance. It is with great pleasure that I acknowledge **Ms. Namrata Ailawar** for her invaluable guidance and continuous support during this work. I am equally grateful to my parents and the members of CDAC ACTS, Pune, for their encouragement and cooperation, which significantly contributed to the successful completion of this project.

My sincere thanks go to **Ms. Srujana B**, Course Coordinator of PG-DESD, for her comprehensive support.

Lastly, I extend my heartfelt appreciation to my batchmates for their collaborative efforts and to everyone who generously offered their help and expertise during this project.

Ms. Anchal Maurya	**240340130007**
Ms. Maithili Balkawade	**240340130013**
Mr. Digpal Singh Bisht	**240340130015**
Mr. Harsh Bhati	**240340130019**
Mr. Shubham Jadhav	**240340130043**

Abstract

This project involves the development and implementation of a CAN-based data transmission system using the STM32F407VGT6 microcontroller. The system is composed of two CAN nodes: the first node transmits sensor data, while the second node receives this data. The receiving node utilizes a Real-Time Operating System (RTOS) to manage the incoming data, with a focus on the deferred task mechanism to optimize task scheduling and execution.

The project is implemented at the bare metal level, meaning that the code interacts directly with the hardware without relying on the hardware abstraction layer, allowing for precise control over system behavior. This approach minimizes latency and maximizes efficiency, which is crucial in real-time data transmission scenarios.

Throughout the development process, key challenges such as efficient data handling, task prioritization, and maintaining system stability were addressed. The resulting system demonstrates the feasibility of using CAN communication in conjunction with RTOS in a bare metal environment. The project provides a foundation for understanding how real-time data transmission can be effectively managed in embedded systems, particularly in environments where reliability and timing are critical.

Index

List of Figures

Figure 21	Configuration Figure 13

List of Tables

Abbreviations

Abbreviation	Full Form
CAN	Controller Area Network
RTOS	Real-Time Operating System
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
RNG	Random Number Generator
FPU	Floating-point unit
MPU	Memory Protection Unit
RTC	Real-Time Clock
APB	Advanced Peripheral Bus
AHB	Advanced High-Performance Bus
PWM	Pulse Width Modulation
SRAM	Random Access Memory
DMIPS	Dhrystone Million Instructions per Second
ART	Adaptive Real-Time Accelerator
DSP	Digital Signal Processing
CCM	Core Coupled Memory
DMA	Direct Memory Access
FIFO	First In First Out
SWD	Serial Wire Debug
JTAG	Joint Test Action Group
CRC	Cyclic Redundancy Check
RTR	Remote Transmission Request

IDE	Extended Identifier
DLC	Data Length Code
SOF	Start of Frame
DEL	Delimiter
ACK	Acknowledgment

1. Introduction

Effective communication within embedded systems is crucial for achieving high performance and reliability, particularly in demanding environments such as automotive systems. This project addresses the development of a sophisticated data transmission system using the STM32F407VGT6 microcontroller, leveraging CAN (Controller Area Network) technology to facilitate communication between two distinct nodes.

The system design involves two CAN nodes: one responsible for sending sensor data and the other for receiving and processing this information. The receiving node integrates a Real-Time Operating System (RTOS), specifically utilizing a deferred task mechanism to manage and prioritize incoming data efficiently. This design choice ensures that the system remains responsive and can handle multiple tasks with minimal latency.

What distinguishes this project is its bare metal approach, where the system is programmed directly on the hardware level without relying on a high-level operating system. This bare metal implementation allows for greater control over hardware interactions and optimizes performance by reducing overhead.

The primary objective of this project is to demonstrate how a CAN-based communication system, combined with an RTOS and bare metal programming, can effectively manage real-time data transmission. By tackling the challenges of task scheduling, data handling, and system stability, this project provides insights into designing reliable and efficient embedded systems.

1.1. Key Components & Project Scope

- **CAN Communication**: Establishing reliable data exchange between the two nodes.
- **RTOS Integration**: Utilizing real-time task management features to handle data efficiently.
- **Bare metal Programming**: Directly interacting with hardware to minimize latency and maximize performance.
- **Conduct Testing and Validation**: Perform functional and performance testing to ensure system reliability and data handling efficiency.

1.2. System Requirements

Hardware Requirements:

1. STM32F407VGT6 Discovery Boards (2)
2. SN65HVD230 CAN Transceivers (2)

Software Requirements:

The software used for implementing the project was STM32CubeIDE 1.15.1 for STM32F407VGT6 Discovery Board.

2. Literature Survey

- **Controller Area Network (CAN) Protocol**

The Controller Area Network (CAN), developed by Bosch in the 1980s, is a robust, multi-master, broadcast communication protocol that has become a standard in the automotive industry and other fields like industrial automation, medical devices, and aerospace. CAN's fault tolerance, real-time communication capabilities, and simplicity in wiring have made it ideal for applications requiring reliable and efficient data transmission. CAN operates on a two-wire differential signaling scheme, which enhances noise immunity and allows communication over long distances. Over the years, CAN has evolved to support higher data rates and extended data lengths with the introduction of CAN 2.0 and CAN FD (Flexible Data Rate). [1]

- **Importance of RTOS in Embedded Systems**

In embedded systems, particularly those with real-time requirements, an RTOS is indispensable. It ensures that tasks are executed with precise timing, providing mechanisms like task scheduling, inter-task communication, and synchronization. RTOSs such as FreeRTOS are commonly used in conjunction with bare metal programming to manage the complexities of modern embedded applications. In systems where CAN-based data transmission is just one of many tasks, an RTOS ensures that critical tasks are prioritized and executed with minimal latency, which is crucial for maintaining the integrity of real-time communications. [2]

- **STM32F407VGT6 Microcontroller Overview**

The STM32F407VGT6 microcontroller from STMicroelectronics is a high-performance device in the STM32F4 series, based on the ARM Cortex-M4 core. It is designed for real-time applications and includes advanced features such as floating-point unit (FPU) support, high-speed embedded memories, and a rich set of peripherals including multiple CAN controllers. The STM32F407VGT6 is particularly well-suited for applications requiring high processing power and precise timing control, making it an excellent choice for implementing CAN-based data transmission in real-time environments. [3]

- **Bare metal Implementation of CAN on STM32F407VGT6**

Implementing CAN on the STM32F407VGT6 in a bare metal environment allows for direct control over the CAN peripheral, minimizing latency and maximizing performance. In bare metal programming, the software is written to directly interact with the CAN hardware registers, providing a highly efficient and responsive system. This is particularly important in real-time applications where the timing of message transmission and reception is critical. By carefully managing the CAN peripheral through direct register manipulation, the system can meet stringent real-time requirements. [3]

- **Combining RTOS with Bare metal CAN Implementation**

Integrating an RTOS with a bare metal CAN implementation on the STM32F407VGT6 offers the best of both worlds: the efficiency and speed of bare metal control and the advanced task management capabilities of an RTOS. In this setup, the RTOS handles task scheduling, allowing CAN-related tasks to be prioritized appropriately while still managing other system tasks efficiently. This integration is critical in complex systems where CAN communication must coexist with other time-sensitive operations, such as sensor data processing or actuator control. [4]

- **Applications of CAN with RTOS in Embedded Systems**

The combination of CAN and RTOS is widely used across various industries. In the automotive sector, it facilitates real-time communication between ECUs, enabling the efficient operation of systems such as powertrain control, ABS, and ADAS. In industrial automation, CAN with RTOS ensures reliable data transmission in complex control systems. In medical devices, the combination is crucial for real-time monitoring and response in critical care equipment. The STM32F407VGT6, with its robust CAN and RTOS support, is well-suited for these demanding applications. [1]

3. System Description and Design

3.1. System Implementation

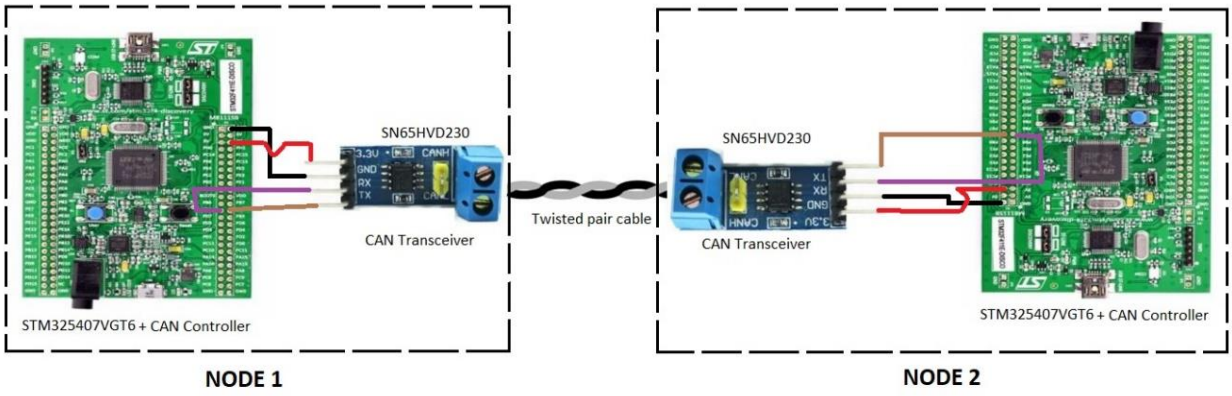

Figure 1: CAN Implementation

Block Diagram Description:

The above block diagram shows the circuit implementation for our project. We developed a system using the STM32F407VGT6 microcontroller, where two CAN nodes are utilized: the first node transmits sensor data, and the second node receives it. The receiving node implements an RTOS with a deferred task mechanism to manage data processing, all at the bare metal level.

3.2. Block Diagram

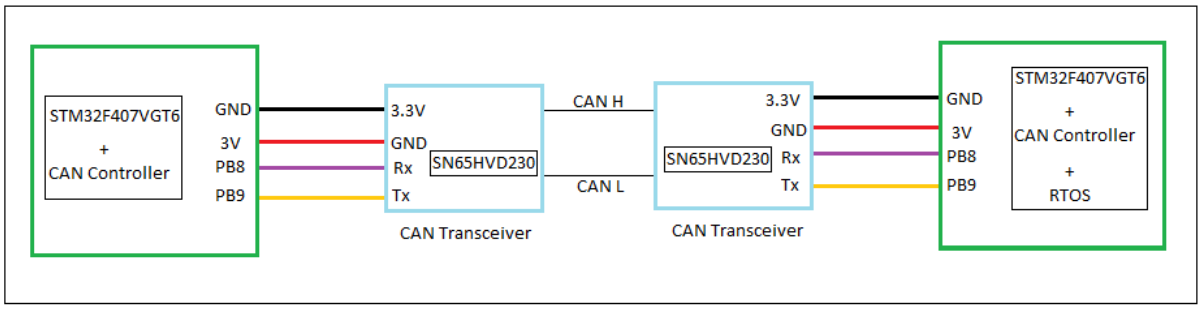

Figure 2: Project Block Diagram

- According to the block diagram first data will be generated randomly using RNG in STM32 CUBE IDE software here we can also take sensor data as input.

- This data is processed by CAN Node 1 and transmitted to CAN Node 2 via CAN transceiver nodes SN65HVD230 modules.
- In CAN Node 2 Data is received using the Transmit function written in bare metal code and receive is done in interrupt mode in deferred task using FreeRTOS.
- Further the data received we can observe and analyze using the Segger System view and Ozone software which is supported by STM CUBE IDE.

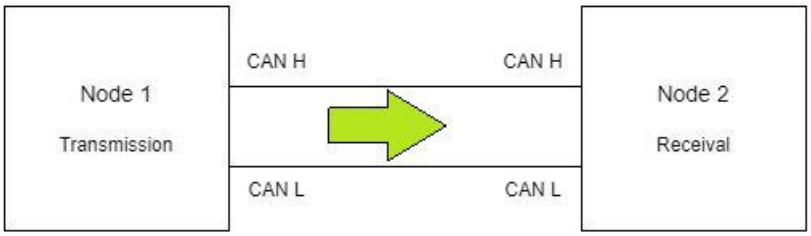

Figure 3: Overall Block Diagram

3.3. Flow Diagram

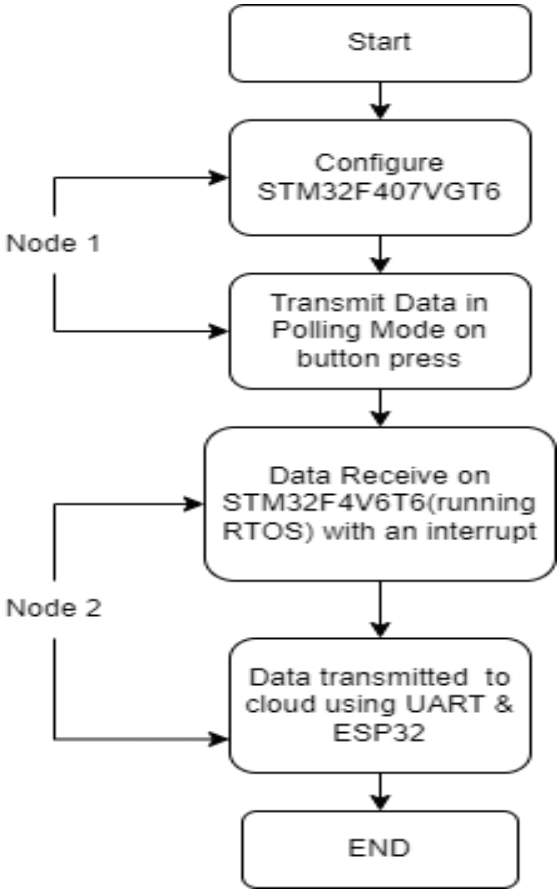

Figure 3.1: Flow Diagram

3.4. STM32F407VGT6 Discovery Board

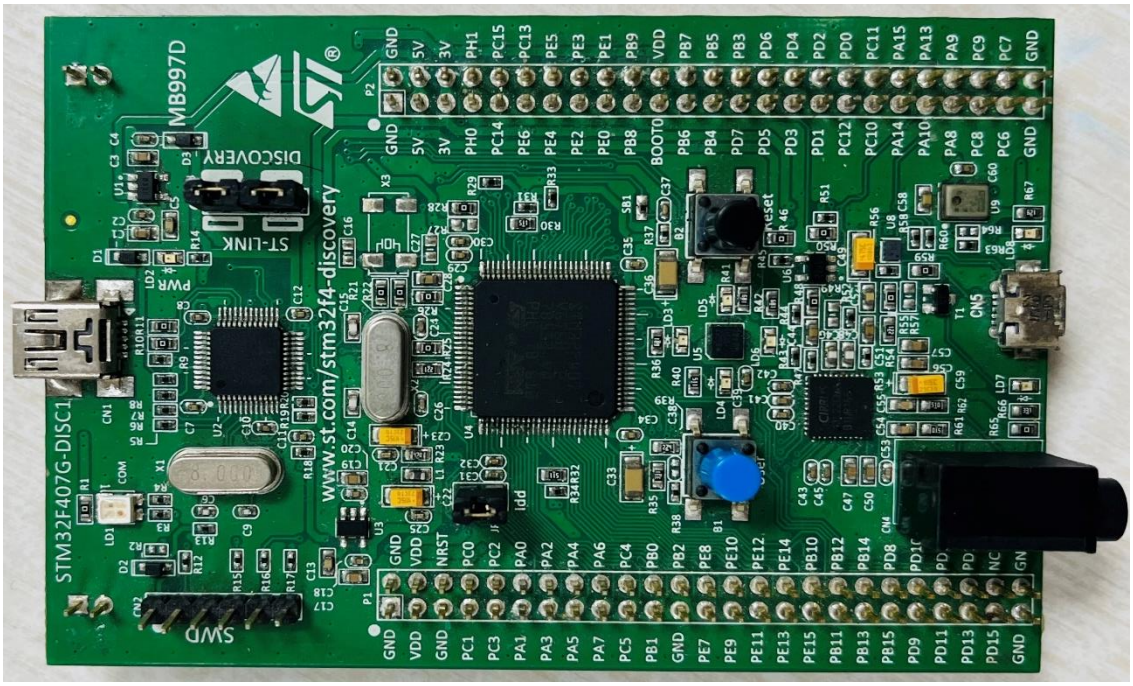

Figure 4: STM32F407VGT6 Discovery Board

The STM32F405xx and STM32F407xx family is based on the high-performance Arm® Cortex®-M4 32-bit RISC core operating at a frequency of up to 168 MHz The Cortex-M4 core features a Floating-point unit (FPU) single precision which supports all Arm single-precision data-processing instructions and data types. It also implements a full set of DSP instructions and a memory protection unit (MPU) which enhances application security.

The STM32F405xx and STM32F407xx family incorporates high-speed embedded memories (Flash memory up to 1 Mbyte, up to 192 Kbytes of SRAM), up to 4 Kbytes of backup SRAM, and an extensive range of enhanced I/Os and peripherals connected to two APB buses, three AHB buses and a 32-bit multi-AHB bus matrix.

All devices offer three 12-bit ADCs, two DACs, a low-power RTC, twelve general-purpose 16-bit timers including two PWM timers for motor control, two general-purpose 32-bit timers. a true random number generator (RNG). They also feature standard and advanced communication interfaces.

Features

- Core
  - Arm® 32-bit Cortex®-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator) allowing 0-wait state execution from Flash memory,

frequency up to 168 MHz, memory protection unit, 210 DMIPS/1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions

- Memories
  - Up to 1 Mbyte of Flash memory
  - Up to 192+4 Kbytes of SRAM including 64-Kbyte of CCM (core coupled memory) data RAM
  - 512 bytes of OTP memory
  - Flexible static memory controller supporting Compact Flash, SRAM, PSRAM, NOR, and NAND memories
- LCD parallel interface, 8080/6800 modes
- Clock, reset and supply management
  - 1.8 V to 3.6 V application supply and I/Os
  - POR, PDR, PVD and BOR
  - 4-to-26 MHz crystal oscillator
  - Internal 16 MHz factory-trimmed RC (1% accuracy)
  - 32 kHz oscillator for RTC with calibration
  - Internal 32 kHz RC with calibration
- Low-power operation
  - Sleep, Stop, and Standby modes
  - V_{BAT} supply for RTC, 20×32bit backup registers + optional 4 KB backup SRAM
- 3×12-bit, 2.4 MSPS A/D converters
  - up to 24 channels and 7.2 MSPS in triple interleaved mode
- 2×12-bit D/A converters
- General-purpose DMA
  - 16-stream DMA controller with FIFOs and burst support
- Up to 17 timers
  - up to twelve 16-bit and two 32-bit timers up to 168 MHz, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input
- Debug mode
  - Serial wire debug (SWD) & JTAG interfaces
  - Cortex-M4 Embedded Trace Macrocell™
- Up to 140 I/O ports with interrupt capability
  - Up to 136 fast I/Os up to 84 MHz
  - Up to 138 5 V-tolerant I/Os
- Up to 15 communication interfaces
  - Up to 3 × I^2C interfaces (SMBus/PMBus)
  - Up to 4 USARTs/2 UARTs (10.5 Mbit/s, ISO 7816 interface, LIN, IrDA, modem control)
  - Up to 3 SPIs (42 Mbits/s), 2 with muxed full-duplex I^2S to achieve audio class accuracy via internal audio PLL or external clock
  - 2 × CAN interfaces (2.0B Active)
  - SDIO interface
- Advanced connectivity
  - USB 2.0 full-speed device/host/OTG controller with on-chip PHY
  - USB 2.0 high-speed/full-speed device/host/OTG controller with dedicated DMA, on-chip full-speed PHY and ULPI
  - 10/100 Ethernet MAC with dedicated DMA: supports IEEE 1588v2 hardware, MII/RMII

- 8- to 14-bit parallel camera interface up to 54 Mbytes/s
- True random number generator
- CRC calculation unit
- 96-bit unique ID
- RTC
    - subsecond accuracy, hardware calendar

3.5. SN65HVD230 CAN Transceivers

Figure 5: SN65HVD230 CAN Transceiver Module

The SN65HVD230 is a high-speed CAN, fault-tolerant device that serves as the interface between a CAN protocol controller and the physical bus. The SN65HVD230 device provides differential transmit and receive capability for the CAN protocol controller and is fully compatible with the ISO-11898 standard, including 24V requirements. It will operate at speeds of up to 1 Mb/s. Typically, each node in a CAN system must have a device to convert the digital signals generated by a CAN controller to signals suitable for transmission over the bus cabling (differential output). It also provides a buffer between the CAN controller and the high-voltage spikes that can be generated on the CAN bus by outside sources. Some of its features are:

- Supports 1 Mb/s operation
- Implements ISO-11898 standard physical layer requirements
- Suitable for 12V and 24V systems
- 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 batteryvoltage)
- Up to 112 nodes can be connected
- High-noise immunity due to differential bus implementation

- Temperature ranges: - Industrial (I): -400C to +850C - Extended (E): -400C to +1250C

3.6. CAN Protocol in STM32

The Basic Can protocol in STM32. Here we will see, how to communicate between two STM32 boardsusing the CAN protocol.

CAN (Controlled Area Network) Protocol is a way of communication between different devices butunder certain rules. These rules must be followed when a message is transmitted over the CAN bus.

Shown below is the Standard CAN Frame

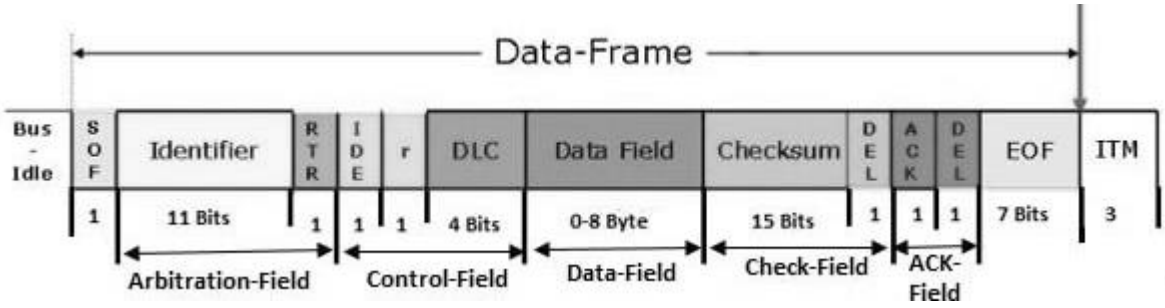

Figure 6: Standard CAN Frame

- The CRC and ACK will be handled by the HAL Library.
- Here, the Identifier is the ID of the transmitting Device
- RTR (Remote Transmission Request) Specifies if the data is Remote frame or Data frameIDE specifies if we are using Standard ID or Extended ID
- r is the Reserved bit
- DLC specifies the data length in Bytes
- Data Field is where we can send the data, which should be up to 8 bytesChecksum and DEL are the CRC data, and its Delimiter
- ACK and DEL are the acknowledgment bit and their Delimiter

3.7. Bare Metal Approach

CAN FIFO & Mailbox:

The architecture of the CAN (Controller Area Network) controller in many microcontrollers, such as those from the STM32 family, typically includes multiple mailboxes for transmission and multiple FIFOs for reception. This design provides several advantages for managing CAN messages efficiently in real-time systems. Let's delve into the details:

Three Mailboxes for Transmission

1. Concurrency: Multiple mailboxes allow the CAN controller to queue up multiple messages for transmission. This is particularly useful in systems where multiple messages need to be sent in quick succession. By having three mailboxes, the CAN controller can manage three different transmission requests simultaneously, which reduces latency and increases the throughput of the system.

2. Priority Management: CAN supports message prioritization based on the message ID. If there are messages in all three mailboxes, the CAN controller will determine the priority of each message and transmit the one with the highest priority first. This ensures that critical messages are sent with minimal delay.

3. Buffering: Having multiple mailboxes acts as a buffer to hold messages temporarily until the CAN bus is free for transmission. This buffering capability helps prevent data loss when the bus is busy and ensures that all messages are eventually transmitted.

Two FIFOs for Reception

1. Message Sorting: The two receive FIFOs (FIFO0 and FIFO1) provide a mechanism to sort and prioritize received messages. The CAN controller can use different filters to direct certain messages to FIFO0 and others to FIFO1 based on their IDs or other criteria. This allows the system to handle high-priority messages immediately while storing lower-priority messages for later processing.

2. Efficient Processing: By using two FIFOs, the CAN controller can continue to receive new messages even if the application is busy processing messages from one FIFO. This dual FIFO structure reduces the chance of message loss due to FIFO overflow and ensures that incoming messages are not missed.

3. Separation of Concerns: In some applications, it may be beneficial to separate different types of messages. For example, control messages could be directed to FIFO0 while diagnostic messages go to FIFO1. This separation can simplify the handling and processing logic in the application.

The following registers from the STM32F407VGT6 Reference manual are used for bare metal code.

1**. CAN transmit status register (CAN_TSR)**

Address offset: 0x08

Reset value: 0x1C00 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LOW2	LOW1	LOW0	TME2	TME1	TME0	CODE[1:0]		ABRQ2	Reserved			TERR2	ALST2	TXOK2	RQCP2
r	r	r	r	r	r	r	r	rs				rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRQ1	Reserved Res.			TERR1	ALST1	TXOK1	RQCP1	ABRQ0	Reserved			TERR0	ALST0	TXOK0	RQCP0
rs				rc_w1	rc_w1	rc_w1	rc_w1	rs				rc_w1	rc_w1	rc_w1	rc_w1

Table 01: CAN_TSR (transmit status register)

(Bit 26) **TME0**: Transmit mailbox 0 empty

This bit is set by hardware when no transmit request is pending for mailbox 0.

(Bit 0) **RQCP0**: Request completed mailbox0

Set by hardware when the last request (transmit or abort) has been performed.

Cleared by software writing a “1” or by hardware on transmission request (TXRQ0 set in CAN_TI0R register).

Clearing this bit clears all the status bits (TXOK0, ALST0, and TERR0) for Mailbox 0.

2. **CAN receive FIFO 0 register (CAN_RF0R)**

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										RFOM0	FOVR0	FULL0	Res.	FMP0[1:0]	
										rs	rc_w1	rc_w1		r	r

Table 02: CAN_RF0R (receive FIFO 0 register)

(Bits 1:0) **FMP0[1:0]**: FIFO 0 message pending

These bits indicate how many messages are pending in the receive FIFO.

FMP is increased each time the hardware stores a new message into the FIFO. FMP is decreased each time the software releases the output mailbox by setting the RFOM0 bit.

CAN Tx Rx Mailboxes:

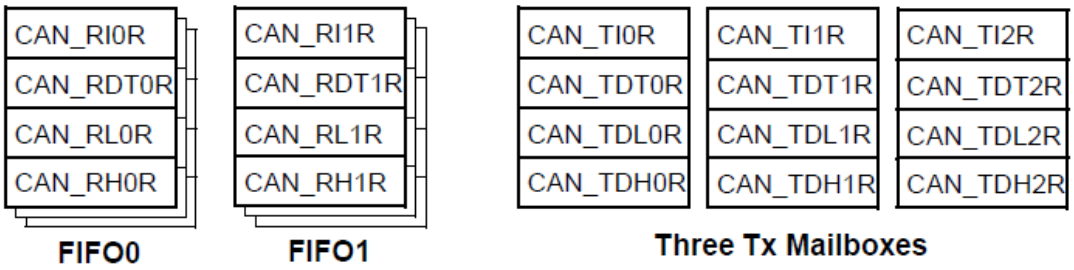

Figure 7: CAN Tx Rx Mailboxes Registers

3. **CAN TX mailbox identifier register (CAN_TIxR) (x=0..2)**

Address offsets: 0x180, 0x190, 0x1A0

Reset value: 0xXXXX XXXX (except bit 0, TXRQ = 0)

All TX registers are write-protected when the mailbox is pending transmission (TMEx reset).

This register also implements the TX request control (bit 0) - reset value 0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STID[10:0]/EXID[28:18]											EXID[17:13]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXID[12:0]													IDE	RTR	TXRQ
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Table 03: CAN_TIxR (TX mailbox identifier register)

(Bits 31:21) **STID [10:0]/EXID [28:18]**: Standard identifier or extended identifier

The standard identifier or the MSBs of the extended identifier (depending on the IDE bit value).

(Bits 20:3) **EXID [17:0]**: Extended identifier

The LSBs of the extended identifier.

(Bit 2) **IDE**: Identifier extension

This bit defines the identifier type of message in the mailbox.

0: Standard identifier.

1: Extended identifier.

(Bit 1) **RTR**: Remote transmission request

0: Data frame

1: Remote frame

(Bit 0) **TXRQ**: Transmit mailbox request

Set by software to request the transmission for the corresponding mailbox.

Cleared by hardware when the mailbox becomes empty.

4. **CAN mailbox data length control and time stamp register (CAN_TDTxR) (x=0..2)**

All bits of this register are write-protected when the mailbox is not in empty state.

Address offsets: 0x184, 0x194, 0x1A4

Reset value: 0xXXXX XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TIME[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							TGT	Reserved				DLC[3:0]			
							rw					rw	rw	rw	rw

Table 04: CAN_TDTxR (mailbox data length control and time stamp register)

(Bits 3:0) **DLC [3:0]**: Data length code

This field defines the number of data bytes a data frame contains or a remote frame request.

A message can contain from 0 to 8 data bytes, depending on the value in the DLC field.

5. **CAN mailbox data low register (CAN_TDLxR) (x=0..2)**

All bits of this register are write protected when the mailbox is not in an empty state.

Address offsets: 0x188, 0x198, 0x1A8

Reset value: 0xXXXX XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA3[7:0]								DATA2[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA1[7:0]								DATA0[7:0]							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Table 05: CAN_TDLxR (mailbox data low register)

(Bits 31:24**) DATA3[7:0]**: Data byte 3

Data byte 3 of the message.

(Bits 23:16) **DATA2[7:0]**: Data byte 2

Data byte 2 of the message.

(Bits 15:8) **DATA1[7:0]**: Data byte 1

Data byte 1 of the message.

(Bits 7:0) **DATA0[7:0]**: Data byte 0

Data byte 0 of the message.

A message can contain from 0 to 8 data bytes and starts with byte 0.

6. **CAN receive FIFO mailbox identifier register (CAN_RIxR) (x=0..1)**

Address offsets: 0x1B0, 0x1C0

Reset value: 0xXXXX XXXX

All RX registers are write-protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STID[10:0]/EXID[28:18]											EXID[17:13]				
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXID[12:0]													IDE	RTR	Res.
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table 06: CAN_RIxR (receive FIFO mailbox identifier register)

(Bits 31:21) **STID [10:0]/EXID [28:18]**: Standard identifier or extended identifier

The standard identifier or the MSBs of the extended identifier (depending on the IDE bit value).

(Bits 20:3) **EXID [17:0]:** Extended identifier

The LSBs of the extended identifier.

(Bit 2) **IDE**: Identifier extension

This bit defines the identifier type of message in the mailbox.

0: Standard identifier.

1: Extended identifier.

(Bit 1) **RTR:** Remote transmission request

0: Data frame

1: Remote frame

(Bit 0) Reserved, must be kept at reset value.

7. **CAN receive FIFO mailbox data length control and time stamp register (CAN_RDTxR) (x=0..1)**

Address offsets: 0x1B4, 0x1C4

Reset value: 0xXXXX XXXX

All RX registers are write-protected.

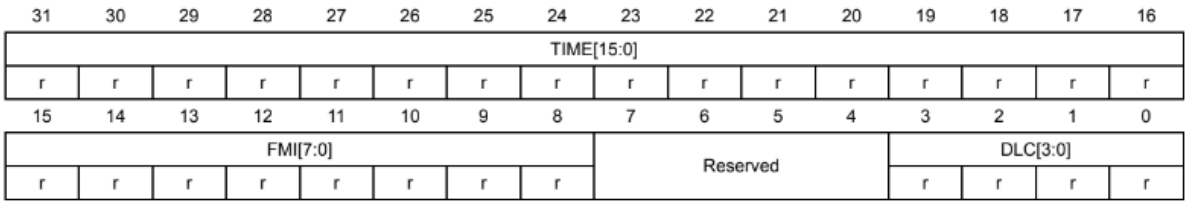

Table 07: CAN_ RDTxR (receive FIFO mailbox data length control and time stamp register)

(Bits 3:0) **DLC [3:0]:** Data length code

This field defines the number of data bytes a data frame contains (0 to 8). It is 0 in the case of a remote frame request.

8. **CAN receive FIFO mailbox data low register (CAN_RDLxR) (x=0..1)**

All bits of this register are write-protected when the mailbox is not in an empty state.

Address offsets: 0x1B8, 0x1C8

Reset value: 0xXXXX XXXX

All RX registers are write-protected.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DATA3[7:0]								DATA2[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA1[7:0]								DATA0[7:0]							
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Table 08: CAN_RDLxR (receive FIFO mailbox data low register)

(Bits 31:24) **DATA3[7:0]**: Data Byte 3

Data byte 3 of the message.

(Bits 23:16) **DATA2[7:0]**: Data Byte 2

Data byte 2 of the message.

(Bits 15:8) **DATA1[7:0]:** Data Byte 1

Data byte 1 of the message.

(Bits 7:0) **DATA0[7:0]:** Data Byte 0

Data byte 0 of the message.

A message can contain from 0 to 8 data bytes and starts with byte 0.

3.8. IOT Implementation

Figure 7.1: IoT Implementation

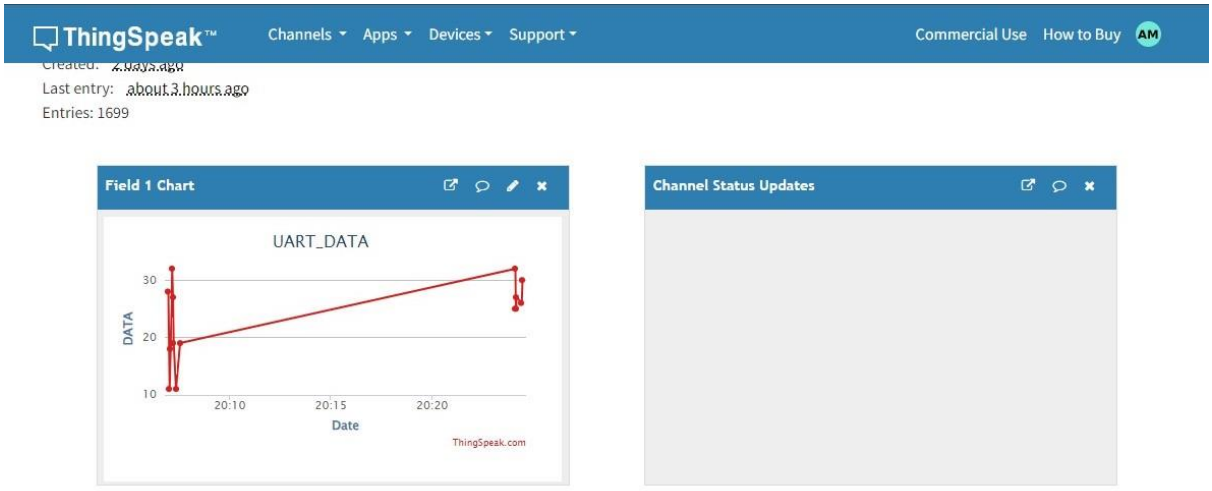

Figure 7.2: IoT in Think speak

MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe protocol ideal for IoT applications. It allows devices to communicate efficiently with minimal bandwidth. Devices publish messages to specific topics, and other devices subscribe to these topics to receive updates. MQTT is known for its simplicity, low overhead, and support for reliable message delivery, making it perfect for resource-constrained devices.

ESP32

ESP32 is a powerful, low-cost microcontroller with integrated Wi-Fi and Bluetooth, widely used in IoT projects. It supports various communication protocols, including MQTT, making it ideal for connecting devices to the internet and cloud platforms like Thing Speak.

Thing Speak

Thing Speak is an IoT analytics platform that allows users to aggregate, visualize, and analyze live data streams in the cloud. It is often used in conjunction with devices like the ESP32 to send sensor data via MQTT, enabling real-time monitoring and analysis.

4. Program Implementation

4.1. CAN Initialization Calculations

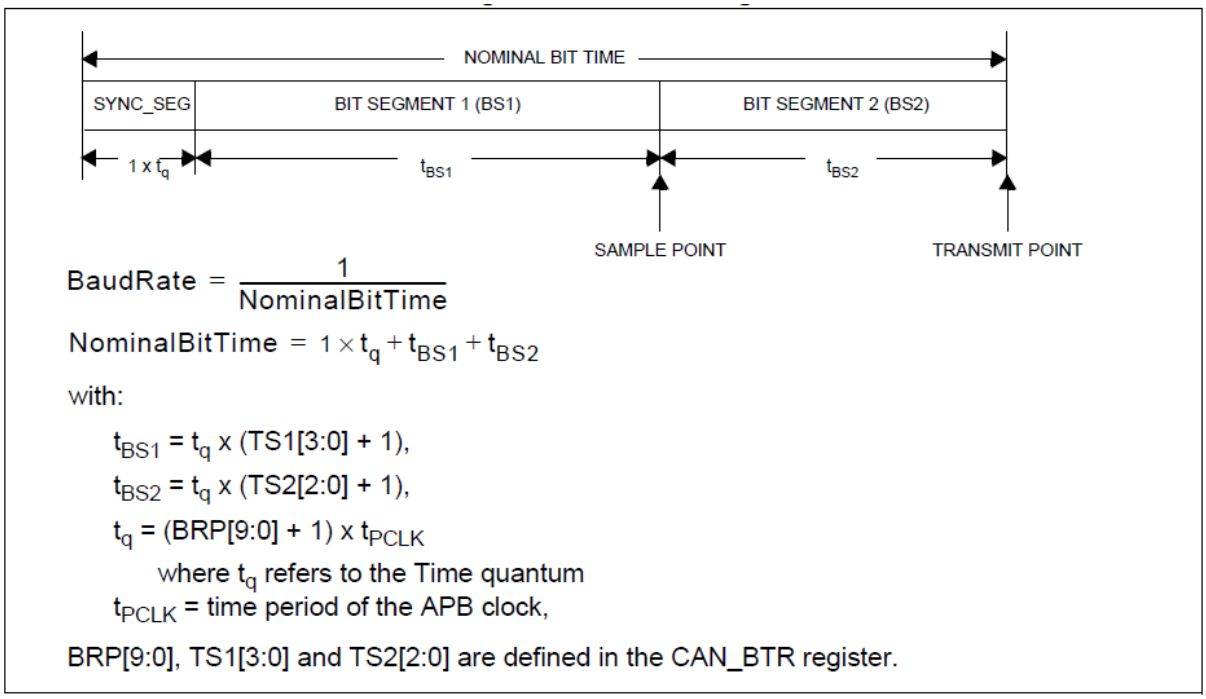

Figure 8: CAN Baud Rate Calculation Formula

For setting 1Mbit,

CAN Baud Rate = 125Kbps) $\Rightarrow \frac{CAN\_CLK}{(BRP*(BS1+BS2+1))}$

$\Rightarrow \frac{24000000}{BRP*(13+2+1)} = 125000$

$\Rightarrow \frac{24000000}{BRP*16} = 125000$

$\Rightarrow \frac{24000000}{125000} = BRP * 16$

$\Rightarrow \frac{24000}{125*16} = BRP$

$\Rightarrow \frac{3000}{125*2} = BRP$

$$=> \frac{1500}{125} = BRP$$

$=> BRP = 12$ => [Prescalar (12)]

[CANCLK →APB1→Bus Speed (24MHz)]

4.2. Connections and Configurations

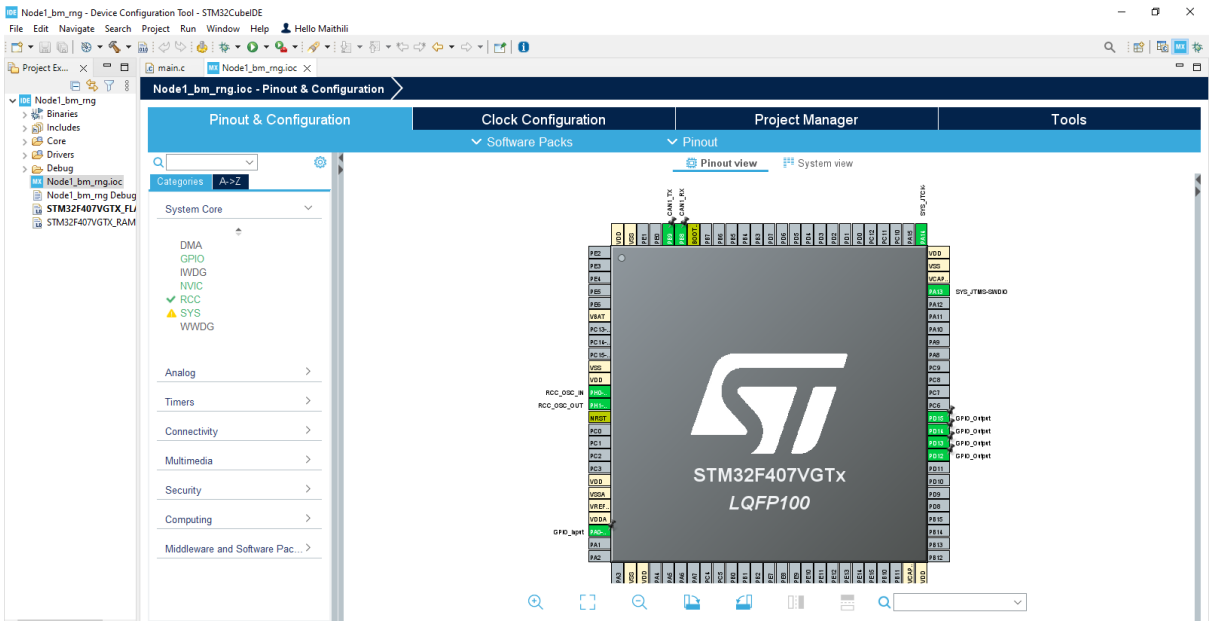

Figure 9: Configuration Figure 1

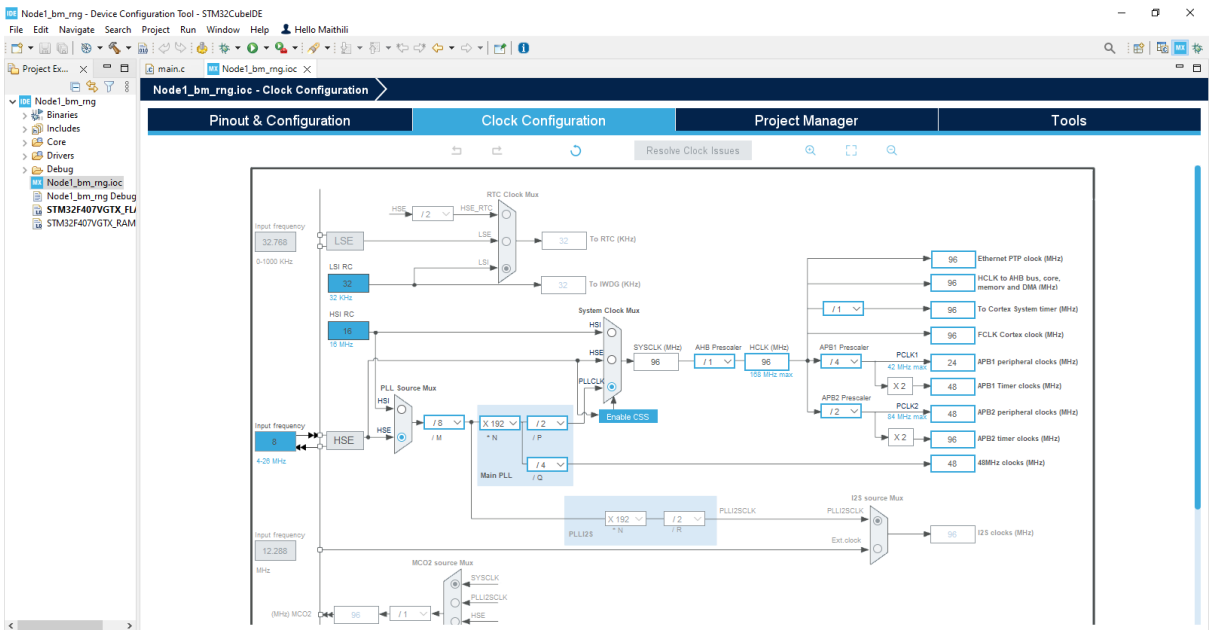

Figure 10: Configuration Figure 2

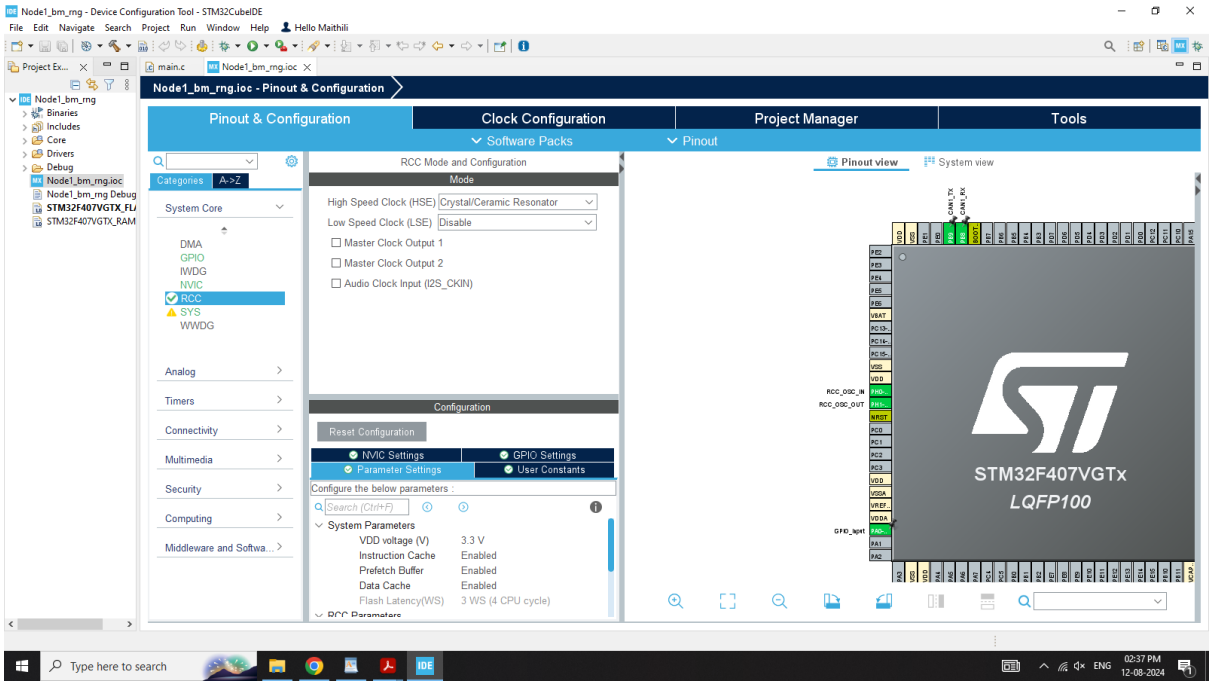

Figure 11: Configuration Figure 3

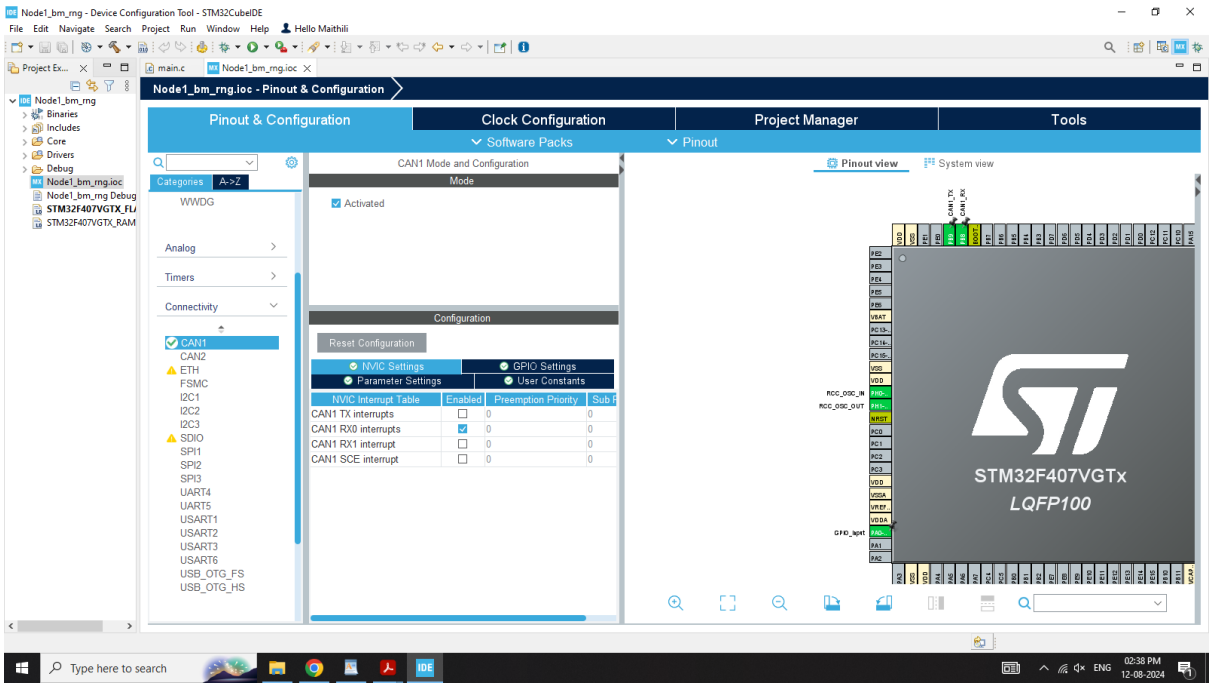

Figure 12: Configuration Figure 4

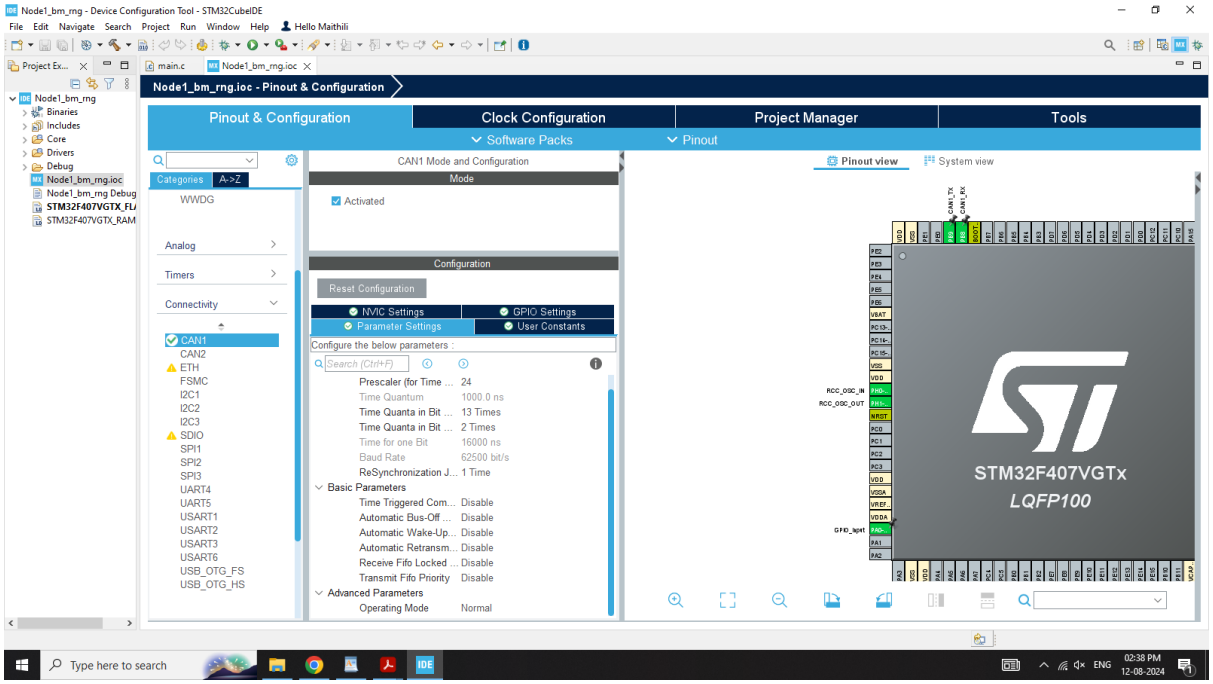

Figure 13: Configuration Figure 5

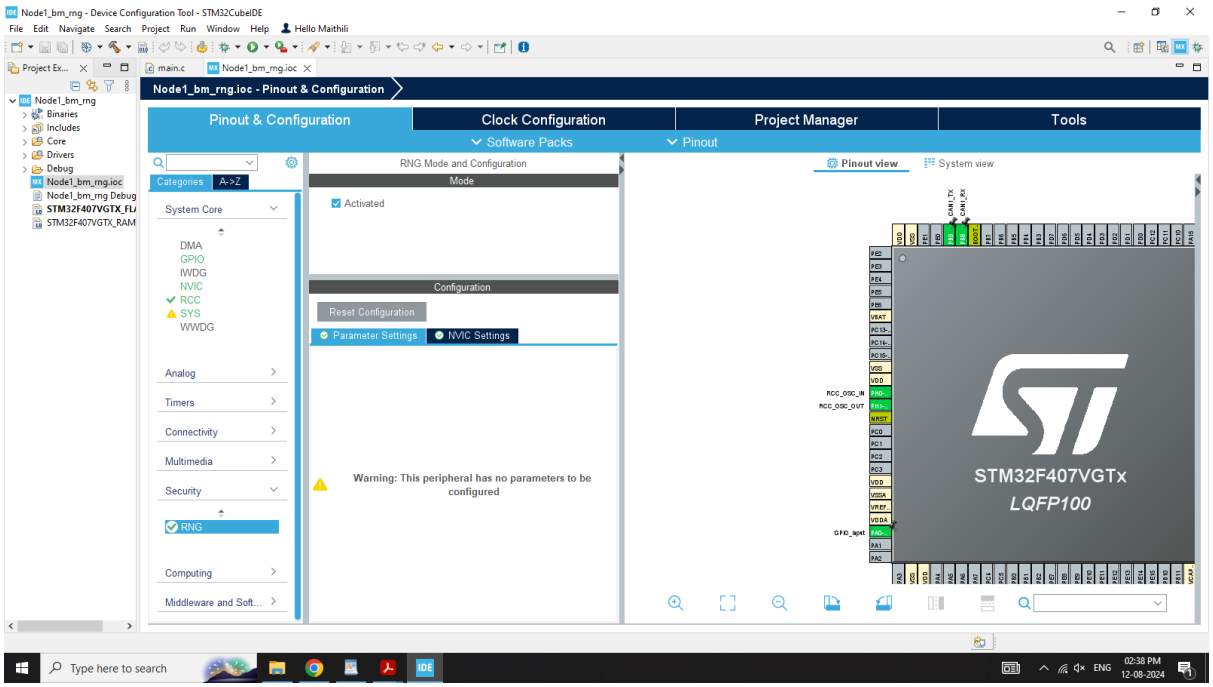

Figure 14: Configuration Figure 6

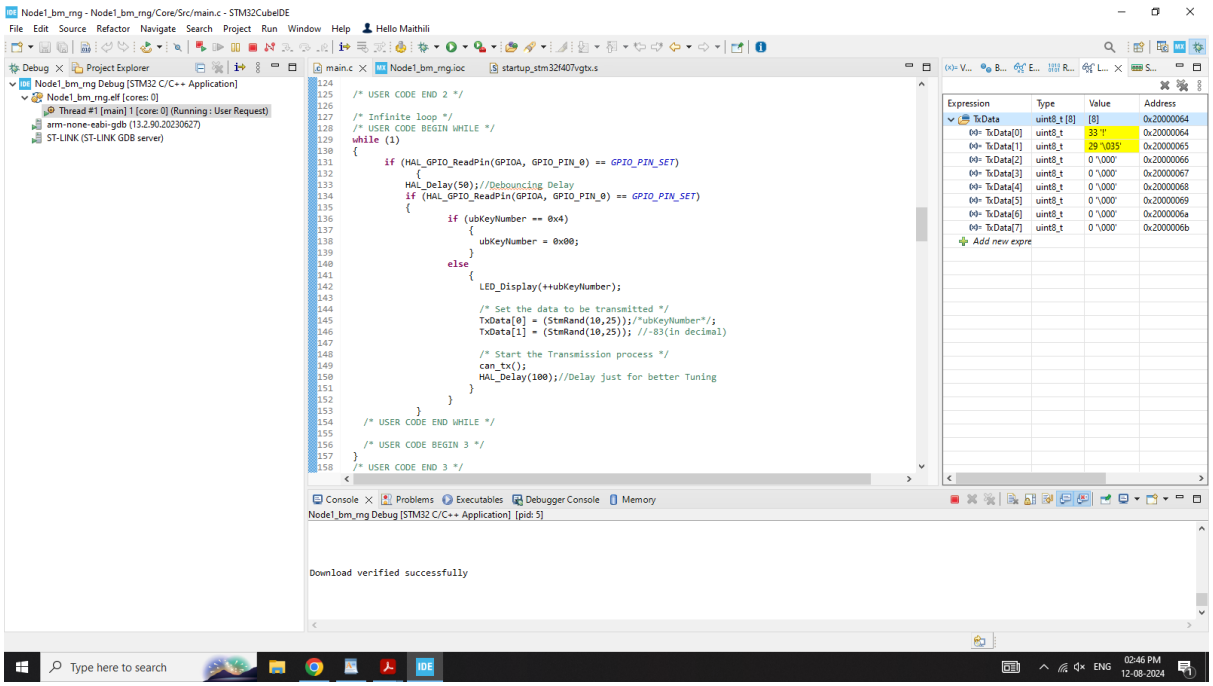

Figure 15: Configuration Figure 7

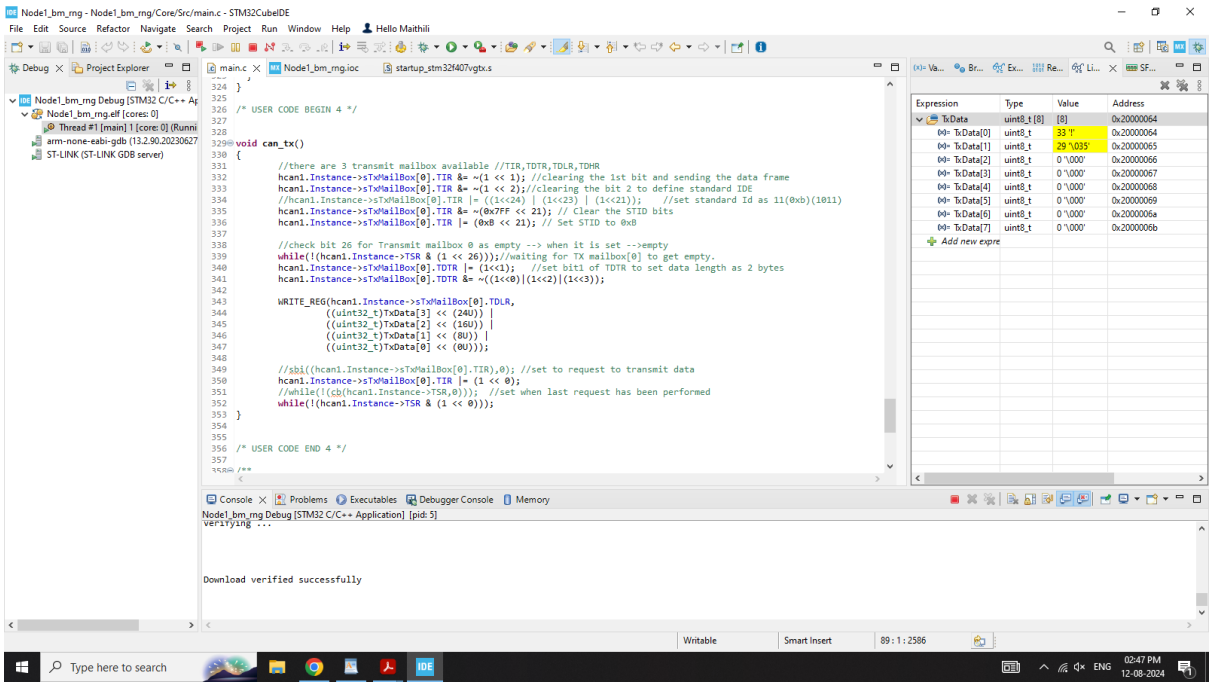

Figure 16: Configuration Figure 8

Node 2:

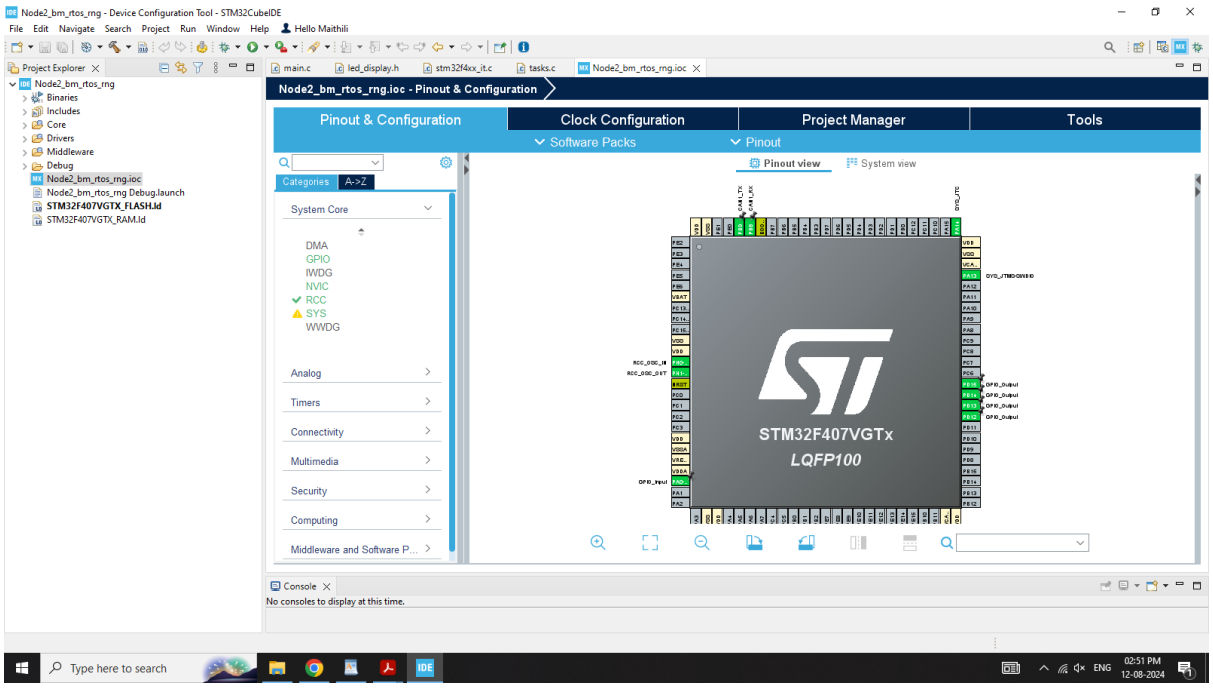

Figure 15: Configuration Figure 9

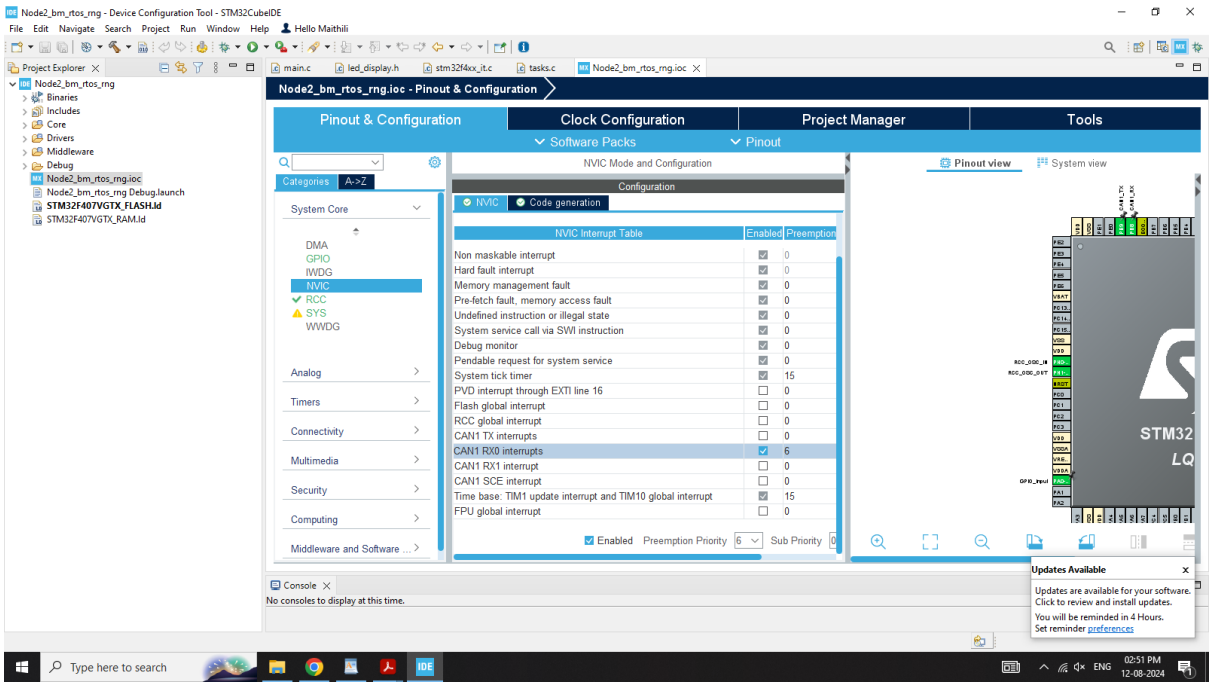

Figure 16: Configuration Figure 10

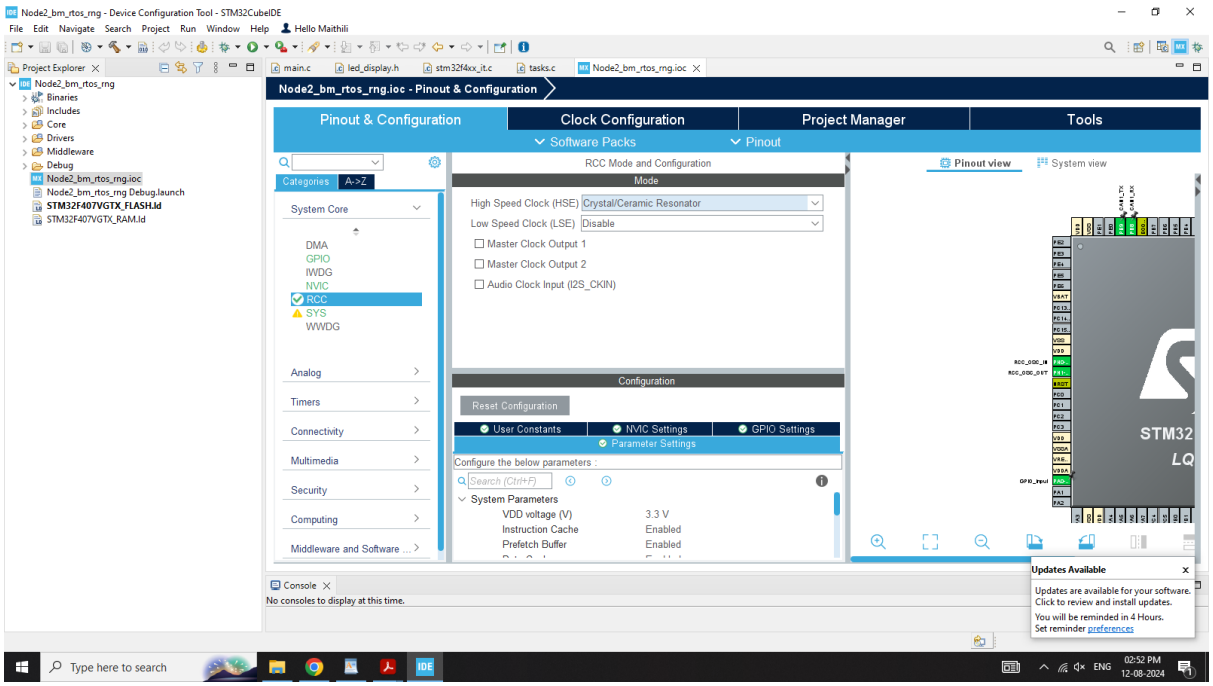

Figure 17: Configuration Figure 11

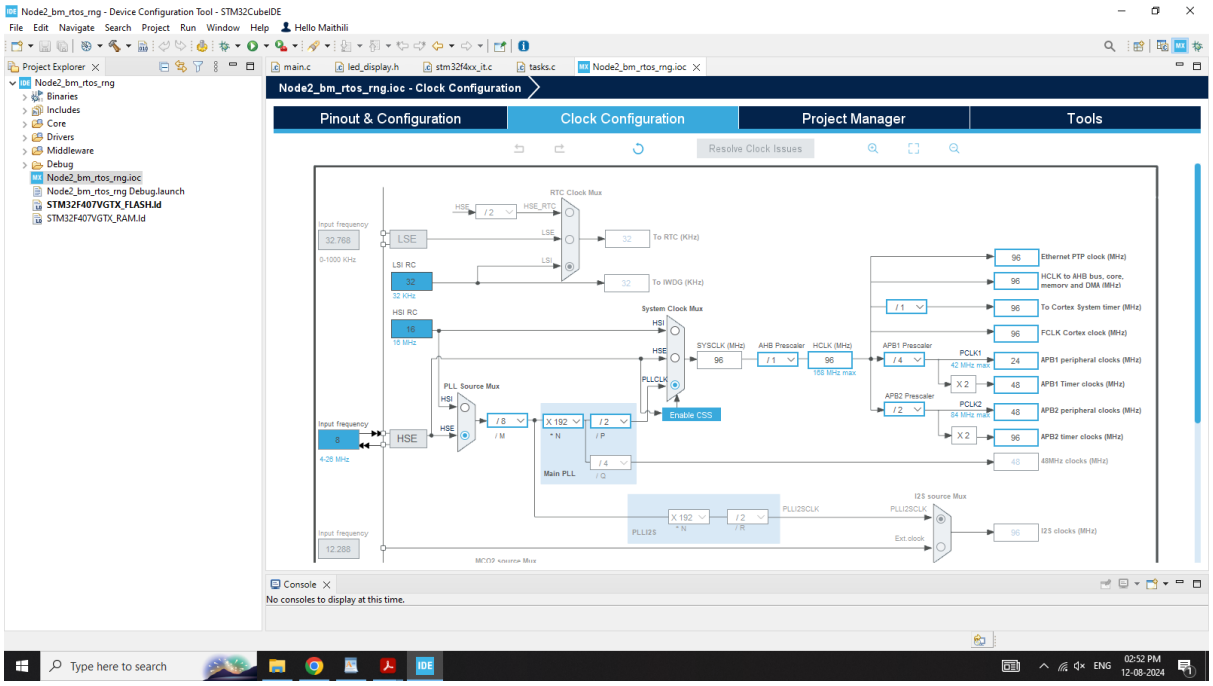

Figure 18: Configuration Figure 12

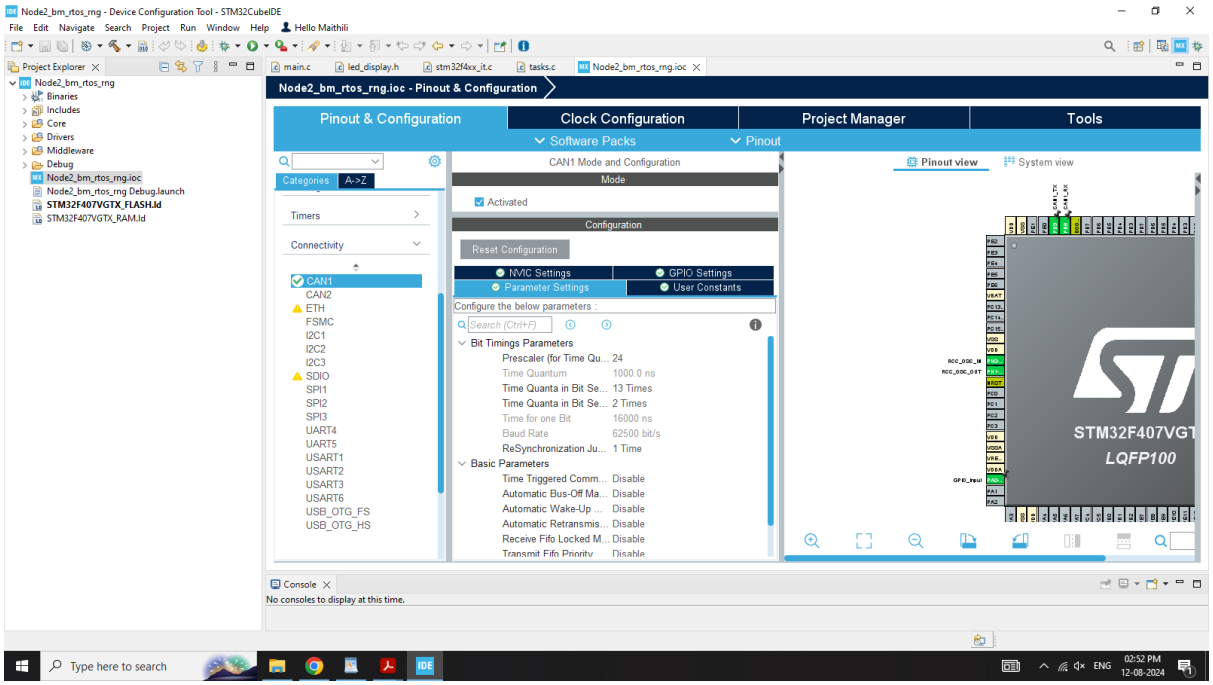

Figure 19: Configuration Figure 12

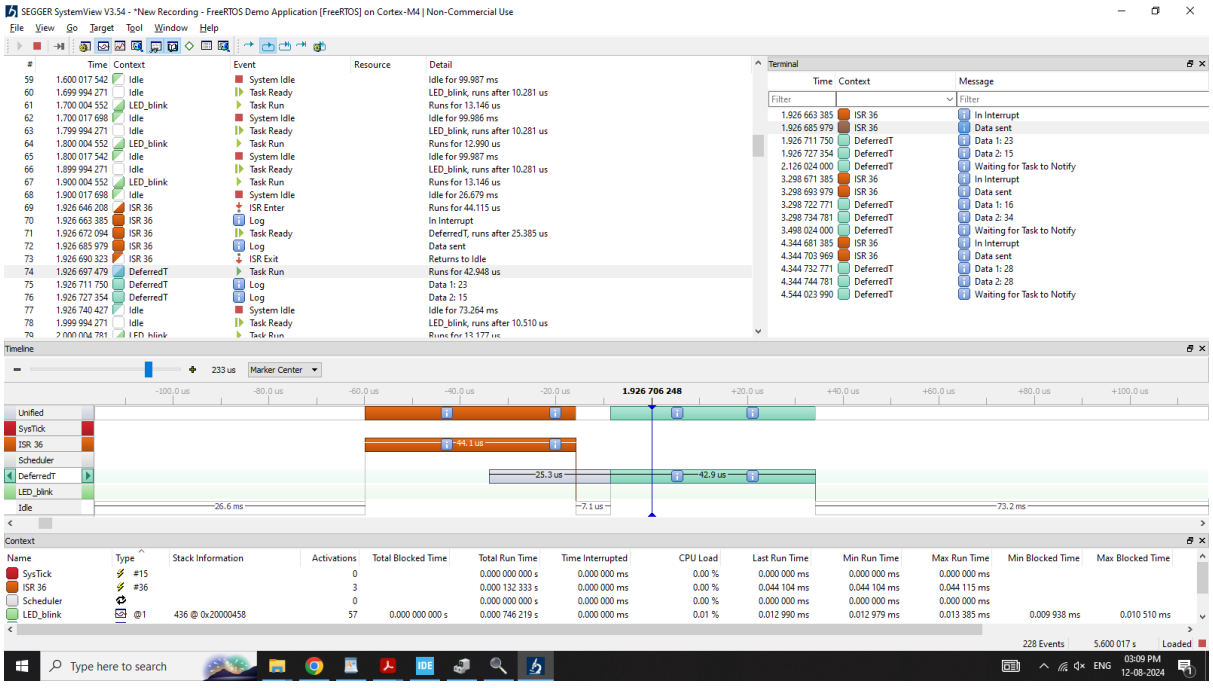

Figure 20: Configuration Figure 13

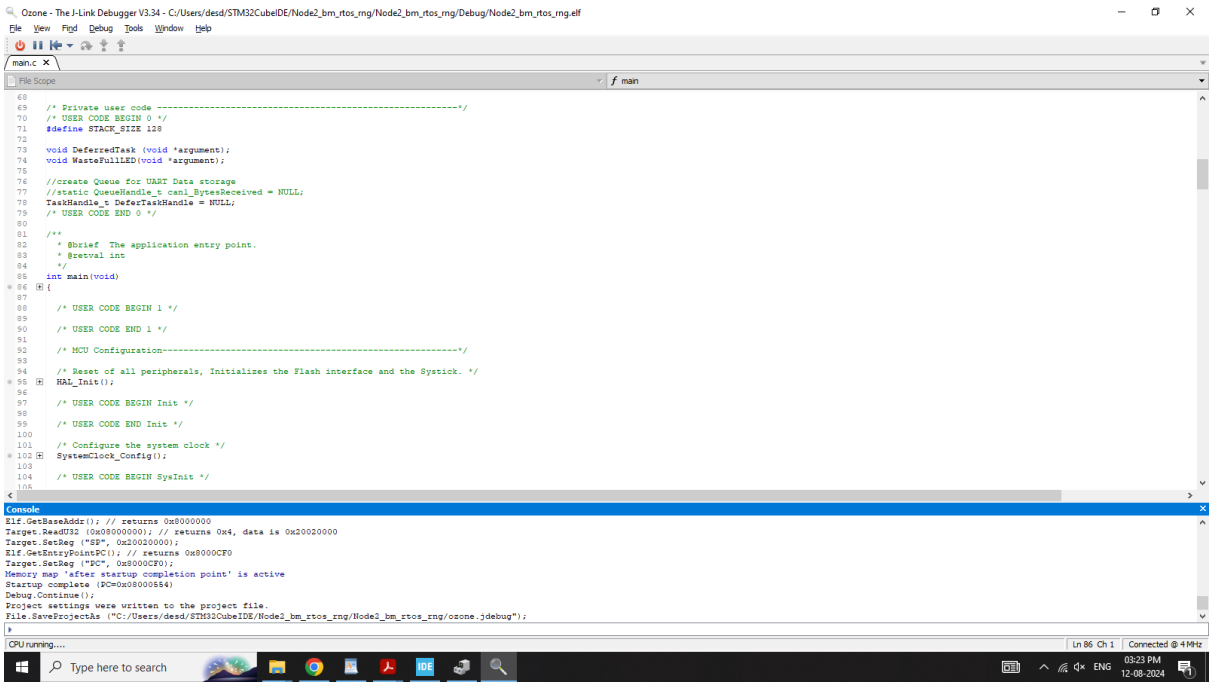

Figure 21: Configuration Figure 14

4.3. RTOS in Receiving Node

FreeRTOS, a popular open-source Real-Time Operating System (RTOS), is well-suited for embedded systems like the STM32F407VGT6 microcontroller. In this project, FreeRTOS provides specific features and tools that enhance the efficiency and reliability of the CAN-based data transmission system. Here's how FreeRTOS contributes:

1. Task Management and Scheduling:

- **Task Creation**: FreeRTOS allows the creation of multiple tasks, each performing different functions, such as CAN message reception, data processing, and communication with other peripherals. These tasks can run concurrently, with FreeRTOS ensuring that each task gets the CPU time it needs.
- **Task Prioritization**: With FreeRTOS, tasks can be assigned different priority levels. In your project, critical tasks like CAN message handling can be assigned higher priority, ensuring they are executed before less critical tasks, improving response time and system efficiency.

2. Deferred Interrupt Processing:

- **Task Notifications and Semaphores**: FreeRTOS supports task notifications and semaphores, which are useful for deferred interrupt processing. For instance, when a CAN message is received, an interrupt service routine (ISR) can quickly capture the

data and use a task notification or semaphore to signal a lower-priority task to process the data. This allows the system to handle high-priority interrupts swiftly while deferring more complex processing to a later time, optimizing overall performance.

3. Queue Management:

- **Data Queues**: FreeRTOS provides robust queue management mechanisms, which are ideal for buffering incoming CAN messages. Queues allow the system to temporarily store messages until they can be processed, ensuring no data is lost even if messages arrive in rapid succession. This is particularly important in real-time systems where data integrity is critical.

4. Real-Time Operation:

- **Deterministic Behavior**: FreeRTOS offers deterministic behavior, meaning that tasks are executed within predictable time frames. This is essential for maintaining the timing requirements of real-time data transmission. FreeRTOS ensures that high-priority tasks, such as those managing CAN communication, meet their deadlines consistently.

5. Inter-Task Communication:

- **Message Passing**: FreeRTOS provides mechanisms like queues, direct task notifications, and message buffers for inter-task communication. In your project, this enables different tasks to exchange data and synchronize operations, such as passing processed CAN data to another task for further action.

6. Resource Management:

- **Mutexes and Semaphores**: FreeRTOS offers mutexes and semaphores to manage shared resources, such as CAN bus access or memory buffers. This prevents conflicts and ensures that resources are accessed in a controlled and predictable manner, which is vital for system stability.

7. Low Power Operation:

- **Idle Task and Tick less Mode**: FreeRTOS includes features like the idle task and tick less mode, which can help reduce power consumption by putting the CPU to sleep when there are no tasks to run. This is beneficial for embedded systems that need to operate efficiently in power-constrained environments.

8. Portability and Scalability:

- **Cross-Platform Support**: FreeRTOS is portable across different microcontroller architectures, making it easier to scale or migrate the project to different hardware if needed. It also supports various extensions and middleware that can be added as the project evolves.

5. Conclusion and Future Scope

5.1. Conclusion

The "Bare metal Implementation of CAN-Based Data Transmission with RTOS on STM32F407VGT6" project demonstrates a robust approach to developing reliable and efficient real-time communication systems for embedded applications. This project capitalizes on the strengths of the Controller Area Network (CAN) protocol, Real-Time Operating Systems (RTOS), and bare metal programming to create a system capable of high-performance data transmission with minimal latency.

1. CAN Communication: The project successfully established reliable data exchange between two nodes via CAN, a critical requirement for systems where deterministic communication is essential. By leveraging the inherent advantages of CAN, such as its multi-master capability and fault tolerance, the system ensures that data is transmitted and received with high reliability, even in environments prone to electromagnetic interference.

2. RTOS Integration: Incorporating an RTOS into the system allowed for efficient task management and scheduling, which is vital in managing the concurrent execution of multiple tasks. The RTOS ensured that CAN communication tasks were given appropriate priority, thereby reducing the risk of data loss or delay. This integration highlights the importance of RTOS in enhancing the performance and reliability of embedded systems where real-time data handling is crucial.

3. Bare metal Programming: By employing bare metal programming techniques, the project was able to interact directly with the hardware, bypassing the overhead associated with traditional operating systems. This direct control over the hardware significantly reduced latency, enabling faster response times, which are critical in real-time applications. The use of the STM32F407VGT6 microcontroller, with its high processing power and advanced peripherals, further contributed to the system's performance, making it an ideal choice for the project.

4. Testing and Validation: Extensive testing and validation were conducted to ensure the system's reliability and efficiency. Functional tests verified the correct operation of CAN-based data transmission under various conditions, while performance tests evaluated the system's ability to handle real-time data with minimal latency. The successful completion of these tests underlines the robustness of the system and its suitability for real-time applications.

5.2. Future Scope:

The future scope of this project encompasses several avenues for enhancement and application in various real-time scenarios:

1. Advanced Real-Time Applications: The system developed in this project can be extended to more complex real-time applications, such as autonomous vehicles, industrial automation, and medical devices. Real-time data processing and communication are critical for safety and efficiency in these fields. For instance, in autonomous vehicles, the CAN bus can manage communication between sensors and control units, ensuring that the vehicle responds promptly to environmental changes. The integration of RTOS would ensure that high-priority tasks, such as obstacle detection and avoidance, are handled with minimal delay.

2. Scalability to Multi-Node Systems: While the current project focused on two-node communication, future work could explore the scalability of the system to larger networks with multiple nodes. This would involve implementing more complex communication protocols and ensuring the system can handle increased data traffic without compromising performance. Such advancements would make the system suitable for use in large-scale industrial automation systems, where multiple machines and sensors must communicate in real time.

3. Integration with Advanced Security Protocols: As embedded systems become more interconnected, the need for secure communication becomes paramount. Future developments could focus on integrating advanced security protocols into the CAN communication system to protect against potential cyber threats. This would be particularly relevant in applications such as automotive systems, where unauthorized access to the network could have serious consequences.

4. Exploring Alternative RTOS Solutions: While the current project utilized a specific RTOS, future work could explore the use of alternative RTOS solutions that offer different features or performance benefits. For example, some RTOSs are specifically designed for safety-critical applications, such as those used in aerospace or medical devices. Evaluating these alternatives could provide insights into optimizing the system for different application domains.

5. Real-Time Data Analytics: The integration of real-time data analytics capabilities could further enhance the system's functionality. By analyzing the data transmitted over the CAN bus in real time, the system could make predictive decisions or provide feedback to improve overall system performance. This would be particularly useful in applications such as predictive maintenance in industrial settings, where early detection of potential failures could prevent costly downtime.

6. Wireless Communication Integration: Another promising direction for future research is the integration of wireless communication technologies with the CAN protocol. This would enable the development of hybrid systems that combine the reliability of CAN with the flexibility of wireless communication. Such systems could be used in scenarios where wired communication is impractical, such as in mobile robotics or remote monitoring systems.

In conclusion, the project lays a strong foundation for the development of real-time communication systems in embedded applications. The successful implementation of CAN-

based data transmission, coupled with the advantages of RTOS and bare metal programming, showcases the potential of this approach in various real-time scenarios. Future work can build on this foundation by exploring advanced applications, enhancing security, and integrating new technologies to create even more powerful and versatile embedded systems

References

[1] Bosch, "CAN Specification," [Bosch CAN Protocol Documentation] (https://www.bosch-semiconductors.com/can).

[2] FreeRTOS, “Introduction to RTOS” [FreeRTOS Documentation]

(https://www.freertos.org/RTOS.html).

[3] STMicroelectronics, "STM32F407VGT6 Reference Manual," [STM32F4 Series Documentation] (https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-high-performance-mcus/stm32f4-series.html).

[4] STMicroelectronics, "Using FreeRTOS with STM32," [STMicroelectronics Application Notes] (https://www.st.com/resource/en/application_note/an4435-using-freertosstm32cube-with-stm32-microcontrollers-stmicroelectronics.pdf).