HO CHI MINH UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY OF ELECTRICAL AND ELECTRONICS ENGINEERING

DEPARTMENT OF COMPUTER AND COMMUNICATIONS ENGINEERING

---o0o---

GRADUATE THESIS

DESIGN AND IMPLEMENTATION OF FIRMWARE OVER THE AIR MODEL FOR CAR LIGHTING AND COLLISION AVOIDANCE SYSTEMS

Major: Computer Engineering Technology

Student: **CHAU THANH DAT**

Student ID: 20119332

DINH THE DANH

Student ID: 20119327

Ho Chi Minh City , June 2024

HO CHI MINH UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY OF ELECTRICAL AND ELECTRONICS ENGINEERING

DEPARTMENT OF COMPUTER AND COMMUNICATIONS ENGINEERING

GRADUATE THESIS

DESIGN AND IMPLEMENTATION OF FIRMWARE OVER THE AIR MODEL FOR CAR LIGHTING AND COLLISION AVOIDANCE SYSTEMS

Major: Computer Engineering Technology

Student: **CHAU THANH DAT**

Student ID: 20119332

DINH THE DANH

Student ID: 20119327

Advisor: **DO DUY TAN, PhD**

Ho Chi Minh City , June 2024

PROJECT ASSIGNMENT

Student name: Chau Thanh Dat	**Student ID:** 20119332
Student name: Dinh The Danh	**Student ID:** 20119327
Major: Computer Engineering technology	**Class:** 201192A
Advisor: PhD. Do Duy Tan	**Phone number:** 0369393615
Date of assignment: 20/2/2024	**Date of submission**: 3/6/2024
Project title: Design and Implementation firmware over the air for car lighting and colision avoidance system.	
Initial material provided by the advisor: - Books and reports related to Firmware-over-the-air models in the automotive industry.	
Content of the project: - Design and implementation of Firmware-over-the-air system for vehicle lighting and collision avoidance systems.	
Final product: - Report documents explaining the project. - Full demo product FOTA system.	

ADVISOR'S EVALUATION SHEET

BẢN NHẬN XÉT KHÓA LUẬN TỐT NGHIỆP

(Dành cho giảng viên hướng dẫn)

Đề tài: THIẾT KẾ VÀ ỨNG DỤNG HỆ THỐNG FIRMWARE-OVER-THE-AIR (FOTA) CHO HỆ THỐNG ĐÈN VÀ PHÁT HIỆN VA CHẠM TRÊN XE HƠI

Sinh viên: + Châu Thành Đạt MSSV: 20119332

+ Đinh Thế Danh MSSV: 20119327

Hướng dẫn: TS. Đỗ Duy Tân

Nhận xét bao gồm các nội dung sau đây:

1. Tính hợp lý trong cách đặt vấn đề và giải quyết vấn đề; ý nghĩa khoa học và thực tiễn:

- Đề tài xác định rõ nhu cầu cập nhật phần mềm tự động trên hệ thống xe hơi. Đề tài có tính ứng dụng cao, giúp giảm thiểu thời gian và chi phí khi cập nhật phần mềm, đồng thời tăng cường tính bảo mật cho hệ thống.

2. Phương pháp thực hiện/ phân tích/ thiết kế:

- Phương pháp thiết kế dựa trên các cơ sở lý thuyết hợp lý.

3. Kết quả thực hiện/ phân tích và đánh giá kết quả/ kiểm định thiết kế:

- Kết quả đánh giá đáp ứng được các mục tiêu đề ra, chứng minh tính khả thi của hệ thống.

4. Kết luận và đề xuất:

- Kết luận đưa ra phù hợp với các vấn đề đã đặt ra và kết quả đạt được, có khả năng ứng dụng trong thực tế.

5. Hình thức trình bày và bố cục báo cáo:

- Văn phong báo cáo rõ ràng, logic và nhất quán. Bố cục rõ ràng, đúng định dạng mẫu. Văn bản chau chuốt.

6. Kỹ năng chuyên nghiệp và tính sáng tạo:

- Thể hiện tốt các kỹ năng giao tiếp và làm việc nhóm.

7. Tài liệu trích dẫn

- Sinh viên trích dẫn tài liệu tham khảo, phù hợp với nội dung nghiên cứu

- Trích dẫn theo đúng chỉ dẫn APA.

8. Đánh giá về sự trùng lặp của đề tài

- Đề tài kế thừa và phát triển từ các nghiên cứu trước đó. Chỉ số trùng lặp khoảng 20% trong ngưỡng cho phép.

9. Những nhược điểm và thiếu sót, những điểm cần được bổ sung và chỉnh sửa*

- Nên tìm hiểu sâu hơn về chuẩn AUTOSAR.

- Cải thiện các bảng biểu, các hình ảnh để tăng chất lượng hình ảnh.

10. Nhận xét tinh thần, thái độ học tập, nghiên cứu của sinh viên

- Sinh viên có tinh thần học tập và nghiên cứu nghiêm túc, kiên trì và sáng tạo trong thời gian thực hiện đề tài.

Đề nghị của giảng viên hướng dẫn

Ghi rõ: "Báo cáo đạt/ không đạt yêu cầu của một khóa luận tốt nghiệp kỹ sư, và được phép/ không được phép bảo vệ khóa luận tốt nghiệp"

- Báo cáo đạt yêu cầu của một KLTN kỹ sư, và được phép bảo vệ KLTN.

Tp. HCM, ngày 02 tháng 06 năm 2024

Người nhận xét

(Ký và ghi rõ họ tên)

Đỗ Duy Tân

TS. Đỗ Duy Tân

---

* *Giáo viên hướng dẫn có thể ghi tiếp vào mặt sau*

ACKNOWLEDGEMENTS

We, the project team, want to express our deepest gratitude to our guiding lecturer, Do Duy Tan, PhD from the Department of Computer Engineering and Telecommunications. His commitment, unwavering support, and insightful guidance on crucial knowledge provided an ideal environment for our project's success. With his invaluable assistance, we were able to see the project through to completion.

We also thank our families and friends who stood by our side and offered their unwavering support throughout this journey. Their genuine advice and constant encouragement proved instrumental in helping us overcome challenges and achieve the best possible outcome for this project. We couldn't have done it without them.

We express our sincere appreciation to Dr. Tan and wish him continued success and good health.

The project team.

Chau Thanh Dat

Dinh The Danh

DECLARATION OF AUTHORSHIP

We now declare that this thesis is our work and has been conducted under the guidance and supervision of Do Duy Tan, PhD. We affirm that the content and findings presented in this thesis are our own and do not violate any research ethics.

The data and figures presented in this thesis are for analysis, comments, and evaluations from various sources. All sources have been duly acknowledged and cited in the references section.

We take full responsibility for any plagiarism or misrepresentation detected in this thesis.

ABSTRACT

Modern car lighting systems and collision avoidance systems play a critical role in ensuring road safety and visibility. However, updating firmware using traditional diagnostic methods can be inconvenient and time-consuming for both users and the company's workforce. To address this issue, a Firmware Over-the-Air (FOTA) system has been developed. This system allows for remote updates and enhances safety, functionality, and user experience.

The FOTA system comprises a Telematics Unit responsible for identifying new firmware updates on the Cloud, downloading them, and sending them to the Gateway. The Gateway, in turn, receives the data and sends the firmware to the node's bootloader, which requires updating and allows the firmware to be reprogrammed. This comprehensive FOTA system offers a reliable, cost-effective, and convenient solution for users, long-term support for customers, ease of maintenance, and future improvements.

TABLE OF CONTENTS

LIST OF FIGURE

LIST OF TABLES

ABBREVIATIONS

ACK	Acknowledge
AUTOSAR	AUTomotive Open System Architecture
CAN	Control Area Network
DIP	Dual In-line Package
ECU	Electronic Control Unit
ESD	ElectroStatic Discharges
FOTA	Flash over the air
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GUI	Graphic User Interface
IVI	In-Vehicle Infotainment
MCU	Microcontroller unit
OEM	Original Equipment Manufacturing
OTA	Over the air update
UART	Universal asynchronous receiver transmitter
UDS	Unified Diagnostic Services

CHAPTER 1 : OVERVIEW

This chapter provides an overview of the thesis, including the reason for selecting this topic, research objectives, scope, and limitations.

1.1. PROBLEM STATEMENT

The auto industry has grown significantly over the past decade. The main factors behind this development are the increasing need for safety and improved driving experience. The cars we had were previously purely mechanical, and now they are being transformed in such a way that the vehicle itself can assist the driver by integrating multiple Electronic Control Units (ECU) into the vehicle, such as ECU brakes, power steering, and airbags [29]…. However, with the integration of more and more ECUs into vehicles, maintenance and updating of these ECUs is almost a big problem for car manufacturers. The concept of firmware updates existed before, but the process was complex. It typically involved connecting the device to a computer, downloading the update onto it, and transferring it to the device using specialized software or hardware. This process was inconvenient, time-consuming, and often required technical expertise. Firmware Over The Air (FOTA) technology was born to solve the problem. FOTA allows car manufacturers to update the software wirelessly to control various vehicle components. The development of FOTA technology revolutionized how firmware updates are delivered and increased user convenience and cost-effectiveness.

Therefore, we decided to choose the graduation thesis topic “DESIGN AND IMPLEMENTATION OF FIRMWARE OVER THE AIR MODEL FOR CAR LIGHTING AND COLLISION AVOIDANCE SYSTEM” because FOTA is a potential and valuable technology. Moreover, implementing it safely and reliably in a safety-critical system such as car lighting is a significant technical challenge. This topic helps us understand more about the FOTA system in the automotive industry.

1.2. RESEARCH OBJECTIVES

This thesis focuses on the design and development of a safe and effective FOTA system based on existing research concepts for updating automotive MCU software using ESP32 and STM32, learning related knowledge (UART, CAN, ...), analyzing potential benefits and solving associated challenges, especially adding AES security mechanisms when transferring data. Finally, different test cases will evaluate the system's performance in a realistic environment.

1.3. RESEARCH SCOPE

This thesis focuses on addressing the following problems:

- Set up a real-time database linked to the FOTA system and design an interface so that developers can easily upload firmware to the database.
- Designing a reliable data transfer sequence from the database to the MCU that needs to be reprogrammed.
- Design and implement a Bootloader to reprogram the application of the Light System and handle errors when the reprogram fails.
- Design of security communication between Gateway - MCU on CAN bus.
- User interface for reprogramming progress.

1.4. RESEARCH METHODS

This research will employ a multi-pronged approach, utilizing:

- Literature review: A crucial step, examining existing research on FOTA technologies, UART, CAN bus, …
- System design and development: Designing the FOTA protocol, communication framework, and update management system specifically for lighting and collision avoidance systems.
- Performance evaluation: Evaluating the efficiency and reliability of the FOTA system in a real-time environment.

1.5. RESEARCH LIMITS

There are some limitations of this project, including:

- The Telematic Unit needs to connect with the Cloud via Wi-Fi. Therefore, the operation depends on a stable Wi-Fi connection; without it, the system would not work correctly. Power supply will also not be considered because there will be many MCUs in use, so we decided to use a power supply directly to the MCU.
- The security of firmware should be considered in real projects as it can involve human life if hackers manipulate that firmware. Because of the scope of the thesis, we will not consider that aspect of the connection between the cloud and the telematic unit; we will assume that it is already securely protected.
- The thesis will not delve into the business aspects of FOTA adoption, such as cost analysis or market potential.
- Power consumption will also not be considered; we focus on the application and feasibility of the project.

1.6.STRUCTURE OF THESIS

The structure of the thesis is as follows:

Chapter 1: Introduction

This chapter provides an overview of the thesis, including the reason for selecting this topic, research objectives, scope, and limitations.

Chapter 2: Literature Review

In this chapter, we discuss the existing research on FOTA and provide detailed information about the components that will be used in the project, including STM32 and ESP32. We also cover the theory of the booting process on MCU ARM CORTEX, UDS standard, AUTOSAR, communication protocols such as CAN, I2C, and UART, and security in embedded systems.

Chapter 3: System Design and Implementation

This chapter, System Design and Implementation, is where the theoretical

knowledge is put into practice. It presents system specifications and requirements, the design for both hardware and software like the System diagram, Bootloader design, Telematic Unit, Gateway system, Collision avoidance system, and Lighting system, and explains the software flowchart. Then, we discuss the implementation.

Chapter 4: Results

This chapter presents the results of the researched topic through test cases and evaluation.

Chapter 5: Conclusion and Future Work

This chapter summarizes the main findings, concludes, and suggests directions for future improvement.

CHAPTER 2: LITERATURE REVIEW

In this chapter, we discuss the existing research on FOTA and provide detailed information about the components that will be used in the project, including STM32 and ESP32. We also cover the theory of the booting process on MCU ARM CORTEX, UDS standard, AUTOSAR, communication protocols such as CAN, I2C, and UART, and security in embedded systems.

2.1. STM32F103C8T6

The STM32 BluePill is a compact and powerful version of the STM32 series of boards, which is considered a popular choice in the DIY and small project development communities, or basically for those who want to learn about ST microchip, ARM Cortex 32bit. [1]

- Microcontroller: STM32 BluePill uses a STM32F103C8T6 chip, an ARM Cortex-M3 microcontroller with a clock speed of 72MHz. The chip comes with 64KB of Flash memory, 20KB of SRAM, and 64 or 128 KB of EEPROM.
- Dimensions: With dimensions of approximately 53mm x 22mm, the STM32 BluePill features a compact design that saves space in applications with size requirements.
- Operating voltage: The board supports operating voltage from 2V to 3.6V, which can operate stably at 5V with the help of a USB power supply pin or VBUS battery.
- Connector: STM32 BluePill usually has a mini-USB connector for loading programs and communicating with computers.
- Pins: The board comes with 37 GPIO connectors, supporting many functions such as GPIO, PWM, UART, SPI, and I2C, providing flexibility with peripheral components and modules.
- Other features: STM32 BluePill supports protocols such as UART, SPI, and

I2C and can interact with computers via USB. It can also perform various functions such as ADC measurement and PWM and support other features depending on the specific application.

- The STM32 BluePill can be programmed using software development environments such as the STM32CubeIDE or Arduino IDE using circuit boards and support libraries from the community. This is a popular choice for small projects, IoT projects, and applications that require flexibility and performance.

We create Table 2-1 summarizing the function of STM32 pins.

Table 2-1 : Table Pinout of STM32 [1]

	Pin Name	Feature
Power	3.3V, 5V, GND	3.3V: The output voltage is stabilized from the on-board regulator (large current operation is not recommended), can also be used to supply power to the microcontroller. 5V from USB or onboard regulator: Can be used to power the onboard 3.3V regulator. GND: The ground pins.
Analog Pins	PA0 – PA7 PB0 – PB1	The pins can read analog signal values with 12-bit resolution.
Input/output pins	PA0 – PA15 PB0 – PB15 PC13 – PC15	A total of 37 pins can be used to connect and control various components.
Serial	TX1, RX1 TX2, RX2	Provides communication via UART.
External interrupts	PA0 – PA15 PB0 – PB15	The number pins are likely to be used to create interrupts when an event occurs.

PWM	PA0 – PA3 PA6 – PA10 PB0 - PB1 PB6 – PB9	Total of 15 Pulse Width Modulation (PWM) pins are available to control pulse width and dim signals.
SPI	MISO0, MOSI0, SCK0, CS0 MISO1, MOSI1, SCK1, CS0	Support Serial Peripheral Interface (SPI) communication via 2 pins.
Onboard LED	PC13	Using the LED as a GPIO pin can be used to display status or a general indicator.
I^2C	SCL1, SDA1 SCL2, SDA2	Support Inter-Integrated Circuit (I2C) communication for connection with other devices.
CAN	CAN0TX, CAN0RX	Provide Controller Area Network (CAN) interfaces

2.2. ESP32

ESP32 is a low-power and cost-effective system-on-a-chip (SoC) microcontroller. Most ESP32 variants integrate Bluetooth and dual-mode Wi-Fi, making them highly versatile, powerful, and reliable for various applications.

As the successor to the widely popular NodeMCU ESP8266, the ESP32 microcontroller brings a significant performance boost and enhanced features. Manufactured by Espressif Systems, it has found extensive use in IoT, robotics, and automation applications.

The ESP32 microcontroller is designed with a robust power management system, enabling it to operate in sleep mode and wake up only when necessary. This energy-saving feature is a boon for battery-powered applications, potentially extending their lifespan significantly [2]**.**

- Low-power consumption: ESP32 is designed to be low-power, making it ideal for battery-powered applications. It has a power management system allows it to operate in sleep mode and wake up only when needed, which can significantly extend battery life.
- Integrated Wi-Fi and Bluetooth: ESP32 integrates Wi-Fi and Bluetooth, making it a versatile platform for various applications. Wi-Fi allows ESP32 to connect to the internet, while Bluetooth allows it to connect to other devices such as smartphones, sensors, and actuators.
- Powerful CPU: ESP32 is powered by a 32-bit Tensilica Xtensa LX6 microprocessor, which provides the performance needed for demanding applications, as shown in the figure below.
- Large memory: ESP32 has a large amount of memory, which allows it to store data and code. This makes it ideal for applications that require a lot of data storage or complex code.
- Wide range of peripherals: ESP32 has a wide range of peripherals, including GPIO pins, ADC, DAC, SPI, I2C, and UART. This makes it a versatile platform for various applications.

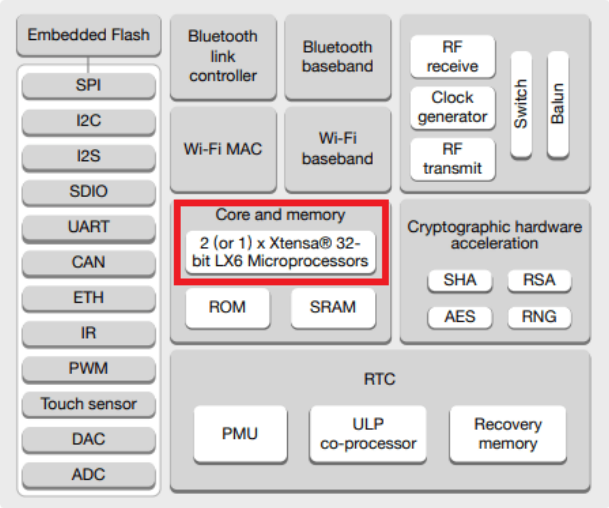


Figure 2-1 : ESP32 Functional Block Diagram [2, Fig 1]

Based on the functional blocks ESP32 specification as shown in Figure 2-1: [2]

- Wireless connectivityWiFi: 150.0 Mbps data rate with HT40
- Bluetooth: BLE (Bluetooth Low Energy) and Bluetooth Classic
- Processor: Tensilica Xtensa Dual-Core 32-bit LX6 microprocessor, running at 160 or 240 MHz
- Memory:
  - ROM: 448 KB (for booting and core functions).
  - SRAM: 520 KB (for data and instructions).
  - RTC fast SRAM: 8 KB (for data storage and main CPU during RTC Boot from the deep-sleep mode).
  - RTC slow SRAM: 8KB (for co-processor accessing during deep-sleep mode).
  - eFuse: 1 Kbit (of which 256 bits are used for the system (MAC address and chip configuration) and the remaining 768 bits are reserved for customer applications, including Flash-Encryption and Chip-ID).
- Low Power: ensures that you can still use ADC conversions, for example, during deep sleep.
- Peripheral Input/Output:
  - Peripheral interface with DMA that includes capacitive touch.
  - ADCs (Analog-to-Digital Converter).
  - DACs (Digital-to-Analog Converter).
  - I²C (Inter-Integrated Circuit).
  - UART (Universal Asynchronous Receiver/Transmitter).
  - SPI (Serial Peripheral Interface).
  - I²S (Integrated Interchip Sound).
  - PWM (Pulse-Width Modulation).
  - Security: hardware accelerators for AES and SSL/TLS.

In this thesis, we will use the ESP32 DEVKIT Development Board.

2.3. FOTA

2.3.1. Overview FOTA

Firmware Over The Air (FOTA) is a highly efficient technology that enables the upgrading and updating of a device's operating firmware over a network, all without the need for a direct connection to the device.

Devices that support FOTA may be able to download updates from the manufacturer shortly, depending on file size and connection speed. This technology saves businesses the time and money to send a technician to upgrade or update their equipment [3].

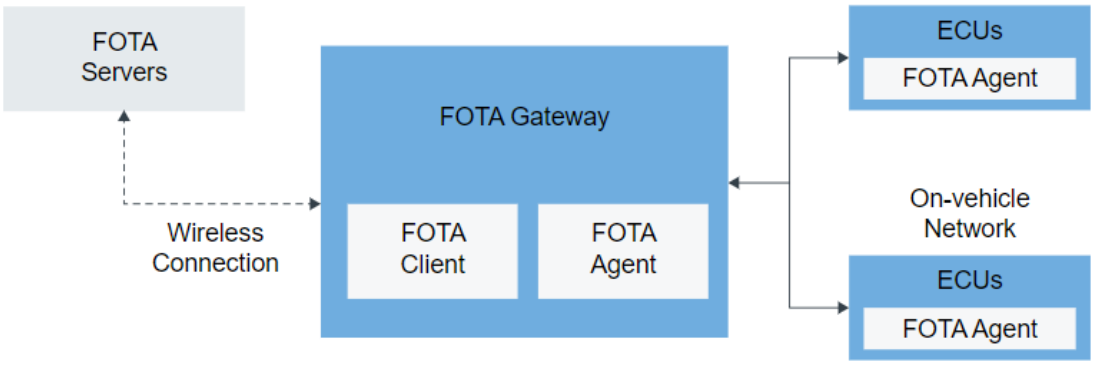


Figure 2-2 : Basic FOTA concept in automotive industry *[4]*

FOTA technology is a game-changer in the realm of IoT systems with a multitude of connected devices that necessitate frequent updates, especially in the automotive industry, with a concept as Figure 2-2. FOTA technology enables the remote management and updating of firmware for ECU in vehicles, leading to streamlined maintenance and ensuring that they are always equipped with the latest software improvements and security patches, thereby bolstering their efficiency and performance.

- *FOTA Server*: Responsible for managing vehicle firmware release.

- *FOTA Client*: Application responsible for communicating with a backend server and updating campaign management.
- *FOTA Agent*: Application that performs final updating of firmware for ECUs during run-time [4].

For example, if car manufacturers wanted to update the ECUs on thousands of cars being used by customers, it would be an almost impossible task to accomplish using traditional methods, in which each car would have to be taken to the manufacturer, connecting to a computer or specialized diagnostic equipment, reprogrammed with new updates and returned to the owner, creates unnecessary costs and disrupts the user experience. However, with this FOTA technology, most of the above problems have been solved.

2.3.2. FOTA Benefits

- Cost- Efficient and better-managed firmware update:

Developing car software can be a complex process involving multiple parties across different regions. In such a case, performing any firmware update can be challenging, requiring many modifications and revisions. Therefore, Updating over the air eases the task while reducing additional costs and time spent on multiple software/firmware updates. IHS Automotive has estimated that total OEM cost savings from FOTA updates will increase to more than $35 billion by 2022 from $2.7 billion in 2015. [5]

- Upgrade the firmware safely, continuous improvements:

Many OEMs have used online patches to fix errors in their automotive computer programs. With FOTA updates, most firmware errors can be fixed by installing the latest firmware or submitting a minor fix to improve the existing firmware. For instance, when a Tesla caught fire in an accident, Tesla resolved the issue by sending out an upgrade that changed the system settings. Since the update, no further fires have been reported. [5]

2.3.3. FOTA Requirements

- Safety:
  - The firmware after downloading the ECU must be verified as the correct image for the specific MCU by using CRC, checksums, and other version compatibility releases.
  - Therefore, we use CRC, a strong testing mechanism to ensure the new update is correctly installed.
- Security:
  - The security of the firmware is a top priority. To ensure this, data is sent via a secure protocol and encrypted. It will be encrypted at the Gateway and then will be decrypted MCU before flashing the update.
  - Protocols used in wireless communication must be secure. However the limitation of the scope of the thesis, we assume that the wireless communication is already secure, we only handle the security from the gateway transmitted through the MCU with AES-CBC-128 standard. The details of this standard will be discussed later.
- Reliability:
  - It is important to ensure that the data transmitted from the Gateway to another MCU is correct. All data must be fully transmitted before processing. Therefore, a reliable communication protocol must be used.
- Scalability:
  - Should be designed to support a large number of embedded devices.

2.4. BOOTING PROCESS OF ARM COTEX-M3

2.4.1. Vector table of ARM COTEX-M3

A vector table where we can find a list of error handlers and usable interrupts for a given ARM CORTEX microcontroller. This table is defined inside the boot file for our MCU, an assembly file ending in ".s" inside our project's Core/Startup directory.

Table 2-2 : Vector table of ARM Cortex-M3 [6, Tab 7.1]

Number	Exception type	Priority[a]	Function
1	Reset	-3	Reset
2	NMI	-2	Non-Maskable Interrupt
3	Hard Fault	-1	All classes of Fault, when the fault cannot activate because of priority or the Configurable Fault handler has been disabled.
4	Memory Management[c]	Configurable[b]	MPU mismatch, including access violation and no match. This is used even if the MPU is disabled or not present.
5	Bus Fault[c]	Configurable	Pre-fetch fault, memory access fault, and other address/memory related.
6	Usage Fault[c]	Configurable	Usage fault, such as Undefined instruction executed or illegal state transition attempt.
7	SecureFault[d]	Configurable	SecureFault is available when the CPU runs in *Secure state*. It is triggered by the various security checks that are performed. For example, when jumping from Non-secure code to an address in Secure code that is not marked as a valid entry point.
8-10	-	-	RESERVED
11	SVCall	Configurable	System service call with SVC instruction.
12	Debug Monitor[c]	Configurable	Debug monitor – for software based debug.
13	-	-	RESERVED
14	PendSV	Configurable	Pending request for system service.
15	SysTick	Configurable	System tick timer has fired.
16-[47/239/479][e]	IRQ	Configurable	IRQ Input

This vector table (Table 2-2) is important in starting the ARM Cortex MCU, looking up errors that occur, and programming the interrupt function to work.

Normally, one vector table for each MCU will be shown in Figure 2-3. However, having vector tables in memory is possible, especially if you have many programs in memory. Moving to different programs in memory means moving vector tables to areas of that program's memory.

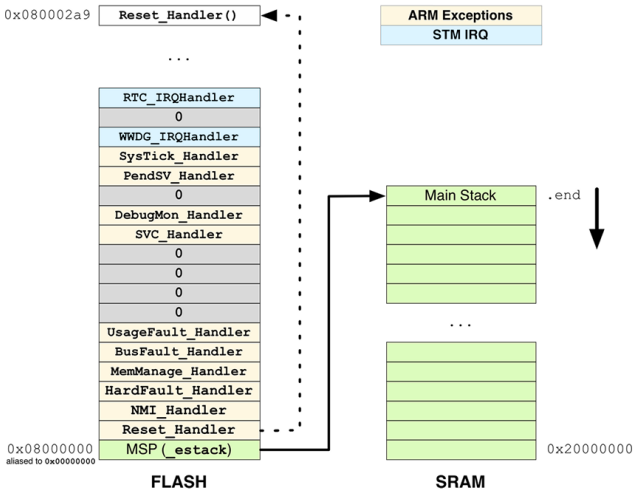


Figure 2-3 : Example structure of vector table in MCU memory [6, Fig 7.2]

2.4.2. The startup process of the MCU ARM COTEX-M3

The STM32 microcontroller implements a special mechanism called physical remapping to boot from memory other than the flash, including dedicated two-pin MCU sampling, BOOT0 and BOOT1. The electrical state of these pins establishes the boot start address and the source memory.

Table 2-3 : Boot mode table [6, Tab 22.1]

Boot mode selection pins		Boot Mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash	Main Flash is selected as boot space
0	1	System Memory	System Memory is selected as boot space
1	1	SRAM	SRAM is selected as boot space

The first row in Table 2-3 corresponds to the most common boot mode: The MCU will alias flash memory (0x0800_0000) to address 0x0000_0000. The other two boot modes correspond to booting from internal SRAM and System Memory, a ROM memory contains a bootloader in all STM32 MCUs and we did not discuss this in this thesis.

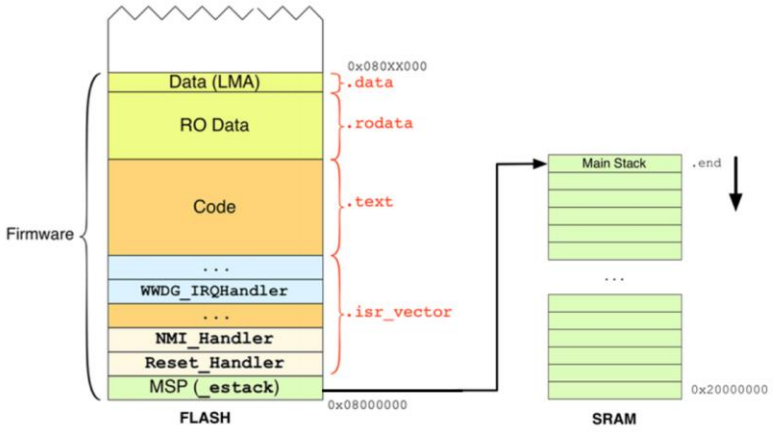


Figure 2-4 : Structure of program in MCU memory [6, Fig 22.2]

Figure 2-4 depicts the operation of a basic program in the MCU. When this boot time has passed, the CPU will fetch the Main Stack Pointer (MSP) from address

0x0000_0000 and execute code from boot memory starting from address 0x0000_0004. Reset_Handler has the function of performing some initialization: the .data section (store static, global variables), the .bss sections (store static, global variables but not initialized value), and from the Reset_Handler will then call the main() function of the program. That's how the main() function is executed every time the processor is powered [6].

2.4.3. The startup process of the MCU with Bootloader.

The Bootloader is just like a regular application. What sets it apart is that it only runs once per operation and returns control to the main firmware after its task.

This is similar to "The startup process of the MCU," but when it has already run the main() code of the Bootloader, it will return control to the main firmware. To do this, move the vector table to the main firmware address, disable all interrupts and peripherals, and then call the reset handler of the main firmware.

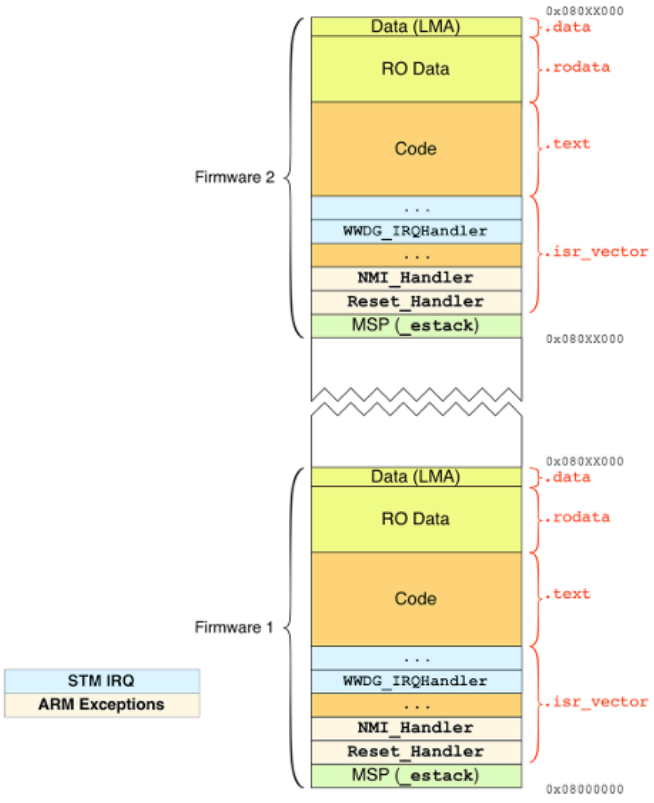


Figure 2-5 : Structure of multiple firmware in MCU memory [7, Fig 22.1]

For example, in Figure 2-5, firmware 1 is the Bootloader, and firmware 2 is the main firmware. First, MCU entry to Bootloader at 0x0800_0000, after the task, it disables GPIO, timer, clocks, and interrupts … then relocates vector table and reset the main stack pointer (MSP) based on the main firmware at 0x08xx000 as shown. Finally, it is called the Reset Handler of the main firmware.

2.4.4. Reprogram MCU with Bootloader

Bootloaders can come in various sizes and with a variety of types, but in general, the operation of a system with a bootloader is relatively standard. The main components of these systems can be seen in Figure 2-6 below: [7].

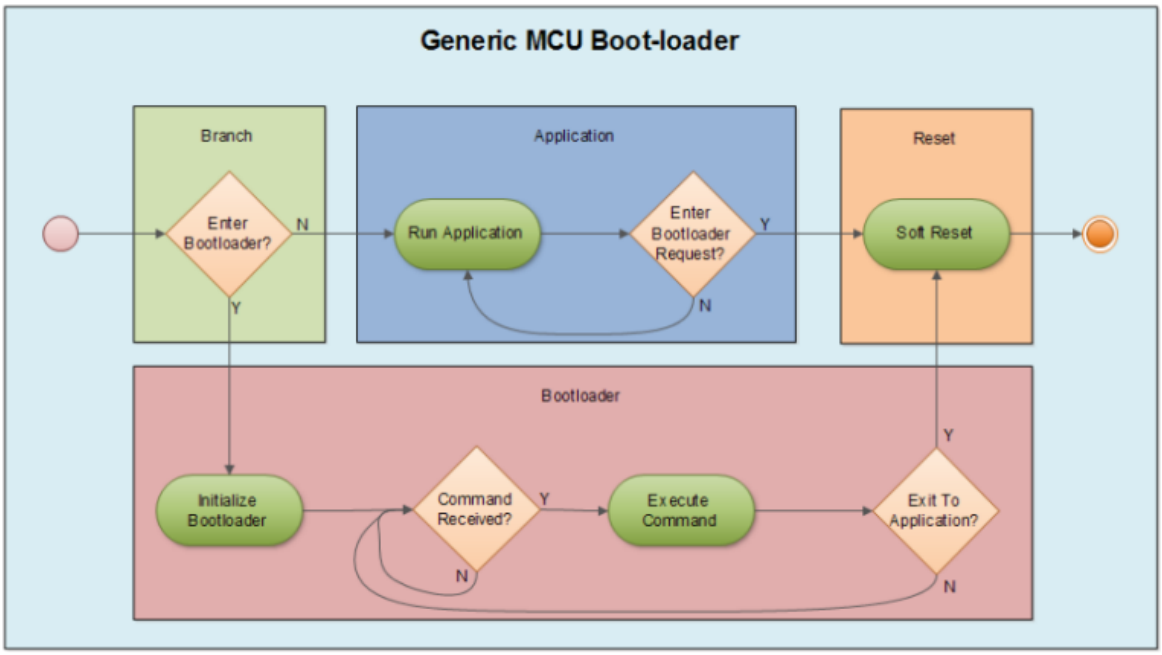


Figure 2-6 : General Bootloader Operation [7]

The branching code (green) decides whether the bootloader enters bootloader mode or application mode.

The application code (blue) is executed only after the branching code has decided that there is no update requirement and the app code is safe to perform.

It is essential that an application can accept commands to enter the bootloader while loading and performing its task. When the application receives the requested

update command, it will restart the MCU. (orange).

The bootloader code (red) performs all the bootloader's necessary functions. Once the bootloader is started, it will begin setting up the basic peripherals needed for the bootloader to perform all its functions, such as timers, interrupt services, and communication peripherals. This allows the bootloader to communicate with external devices and execute flashing commands. When it receives a reprogramming command, it will begin backing up the main firmware memory to the backup memory used for recovery if needed. Then, the bootloader clears the main firmware memory and receives firmware via a predefined protocol (such as UART, SPI, I2C, CAN, ...). When reprogramming is finished, it will reset and enter the main firmware.

2.5. UDS STANDARD FOR CAN BUS

Unified Diagnostic Service (UDS) is a diagnostic communication protocol used in electronic control units (ECUs) in automotive electronics, specified in ISO 14229-3. "Unified" refers to an international standard rather than a standard specific to a certain company or organization. This makes the UDS standard extremely important in ECU development in the automotive industry, as it makes it easy to design diagnostic tools and communicate with the ECU to obtain information, maintain and update new firmware, … [8]**.**

Overall, the UDS standard is essential for engineers developing ECUs and diagnostic tools in the automotive industry. Its use ensures efficient and secure communication between ECUs and diagnostic tools, making vehicle maintenance and repair more accessible and safer.

UDS uses bytes to define services, called Service Identifiers (SID). Tables 2-4 introduce a few UDS codes used in this thesis:

Table 2-4 : UDS Code Table

Request SID	Response SID	Service	Description
0x10	0x50	Diagnostic Session Control	Used to start operating FOTA sessions
0x34	0x74	Request Download	Used to request downloading new software
0x36	0x76	Transfer Data	This service is used for both uploading and downloading
0x37	0x77	Request Transfer Exit	Used to request finished download progress

2.6. AUTOSAR

2.6.1. Overview AUTOSAR

AUTOSAR, which stands for Automotive Open System Architecture, is a collaboration between vehicle manufacturers, suppliers, service providers, and companies from the global automotive, semiconductor, and software industries. [9]

The main goal of AUTOSAR is to standardize the software architecture of ECUs (electronic control units), an essential component in modern vehicles. By doing so, AUTOSAR enables the development of innovative electronic systems that improve vehicle performance, safety, and security.

We will not discuss AUTOSAR in detail; instead, we will introduce its layers and discuss the RTE layer, which is extremely important in AUTOSAR.

2.6.2. AUTOSAR Layer

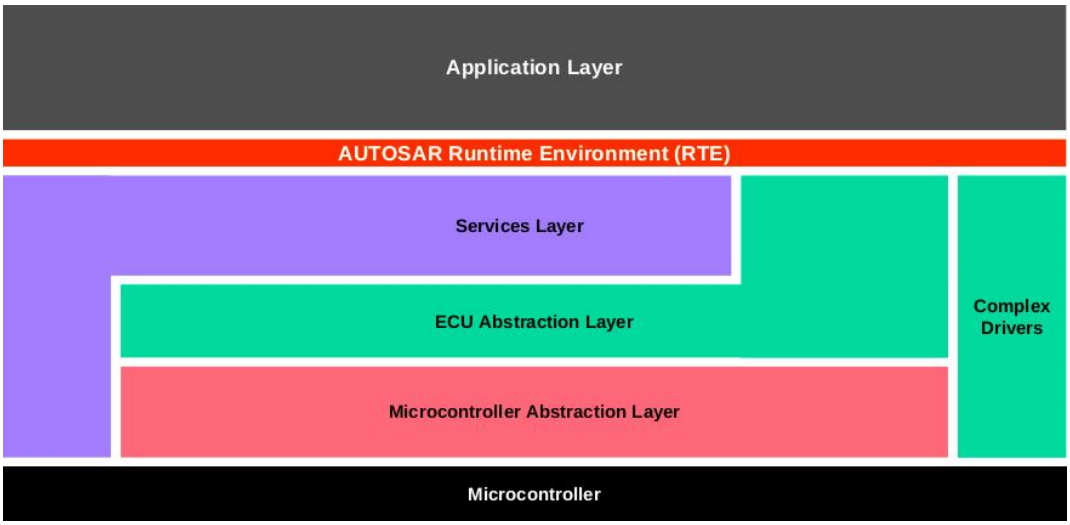


Figure 2-7 : AUTOSAR layer [24]

The AUTOSAR software architecture in Figure 2-7 consists of 3 primary layers:

- **AUTOSAR Basic Software Layer:** Includes three sub-layers.
  - **Microcontroller Abstraction Layer** (MCAL): MCAL is the hardware abstraction layer that implements the interface for specific microcontrollers. It also has software layers that are integrated with microcontrollers through registers and provide peripheral drivers.
  - **ECU abstraction layer**: The main task of the ECU abstraction layer is to provide services dedicated to ECUs, provide access to all peripherals, and support functions such as memory, I / O, etc.
  - **Service Layer**: The service Layer is mainly responsible for helping BSW provide services on ASW thanks to API.
- **AUTOSAR Runtime Environment (RTE Layer)**: The middleware layer between the Basic Software (BSW) and the Application Layer. It provides communication services to application software.
- **Application layer:** This level represents the highest layer in the AUTOSAR architecture and is the most accessible to users. It includes components that software developers can interact with and modify directly, such as indicators,

brakes, exhaust, air conditioning, fuel gauge, tachometer, or touch screen.

- **Complex Driver:** Used for systems that respond quickly, such as airbag systems, emergency braking systems.

2.6.3. RTE Layer

The runtime environment (RTE) is the layer between the Application and Basic Software of the AUTOSAR architecture. It is essential in operating and communicating the modules included through the Virtual Function Bus (VFB) mechanism. RTE creates an environment where there are ports for modules to communicate and exchange data through ports.

The modules can read and write data through RTE. In addition, it also controls the scheduling of tasks in the ECU. For example, the ECU has a task that the lights inside the car will turn on when the user presses the touch screen placed inside the car.

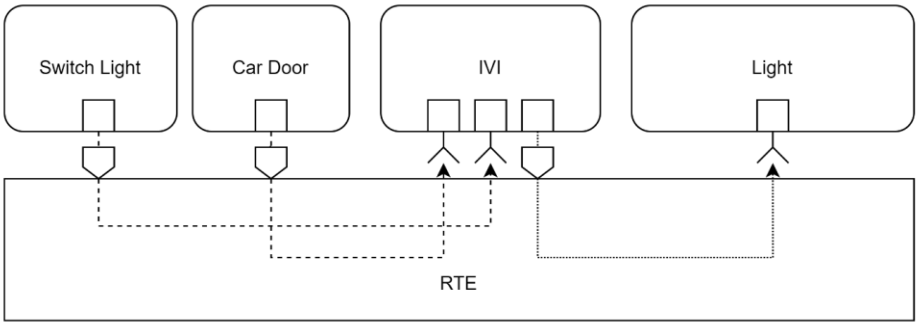


Figure 2-8 : RTE Example

For example, in Figure 2-8, when the software component reads the car door sensor data, if it detects that the door is open, it will send a signal to the RTE indicating that the car door is open. The IVI (In-Vehicle Infotainment) system will receive this signal and display a message on the screen indicating that the car door is open and turn on the light inside the car. The Light module will read the signal from the RTE and then perform the necessary actions to turn on the car light. Similarly, when the switch is turned on, the light will turn on, and when the switch is turned off, the light will turn off.

2.7. CAN COMMUNICATIONS PROTOCOL

2.7.1. CAN Introduction

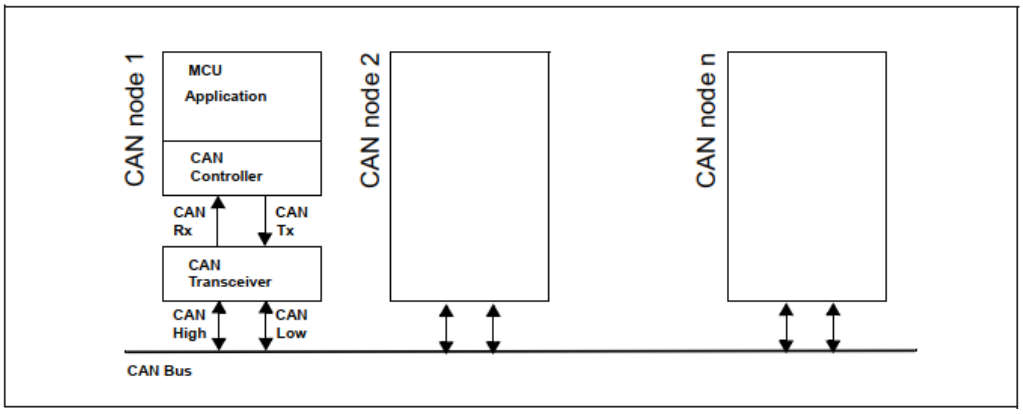


Figure 2-9 : CAN Topology [23, Fig 222]

The CAN bus uses a two-wire interface consisting of a twisted pair of wires: CAN High (CANH) and CAN Low (CANL), as shown in Figure 2-9. It employs a differential signaling method, where the voltage difference between the two wires represents the transmitted data. This differential signaling helps reduce the impact of electromagnetic interference (EMI) and noise.

CAN bus is known for its reliability, real-time capabilities, scalability, and cost-effectiveness. It is widely used in automotive systems, industrial automation, aerospace, medical devices, and other applications that require robust and efficient communication between devices.

2.7.2. CAN Network Message Format

CAN frames, or CAN messages, are the data transmission units in the Controller Area Network (CAN) protocol. A CAN frame consisting of several components is presented in Figure 2-10 [10]**:**

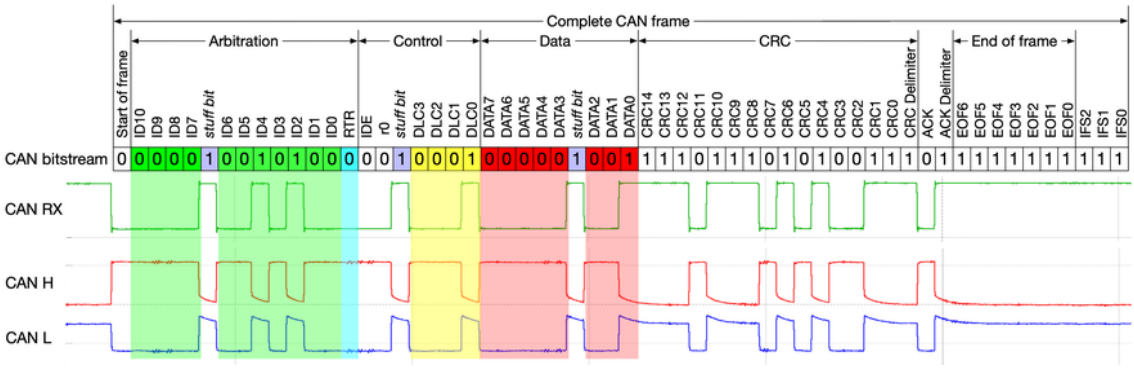


Figure 2-10 : CAN Message Structure [10]

- **Identifier (ID):** The ID field (green) is a numeric value that identifies the priority and content of the message. It determines the message's position in arbitration and helps nodes filter relevant messages.
- **Data Length Code (DLC):** The DLC field (yellow) specifies the number of data bytes contained in the message. It ranges from 0 to 8, indicating the size of the data payload.
- **Data Field:** The data field (red) carries the actual information being transmitted. It can contain up to 8 bytes of data, depending on the DLC value.
- **Control Bits:** The control bits (cyan) include various flags and control information related to the frame, such as the Remote Transmission Request (RTR) flag. The RTR flag distinguishes between data frames and remote frames used for requesting data.
- **Cyclic Redundancy Check (CRC):** The CRC field contains a checksum that helps detect errors during transmission. It allows the receiving nodes to verify the integrity of the received data.
- **Acknowledge (ACK) Slot:** The ACK slot, located after CRC, is used to acknowledge successful message reception. After transmitting a message, the receiving nodes send an ACK bit to indicate successful reception. If no ACK bit is received, it indicates an error or a non-responsive node.

CAN frames are transmitted serially over the CAN bus, with each node on the network receiving the frames. However, only the nodes with matching IDs or relevant filtering criteria process the message, while others ignore it. This filtering mechanism allows for efficient communication within the network.

2.8. I2C COMMUNICATIONS PROTOCOL

2.8.1. I2C Introduction

The Inter-Integrated Circuit (I2C) protocol is a widely used serial communication protocol that enables communication between multiple devices using a two-wire interface. Figure 2-11 illustrates this model. It was developed by Philips (now NXP Semiconductors) and is commonly used in various electronic systems. [11]

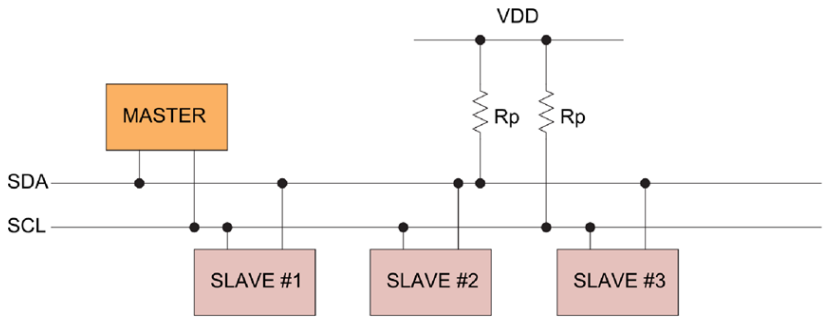


Figure 2-11 : I2C Topology [11, Fig 1]

I2C is a synchronous, multi-controller / multi-target (or master-slave) serial communication bus. Each slave device has its unique address, allowing the master to differentiate slaves from another.

It uses only two bidirectional open-drain lines, SDA and SCL, which are pulled high for data communication.

- Serial Data (SDA) – Data transfer takes place through this pin.
- Serial Clock (SCL) – It carries the clock signal.

2.8.2. I2C transfer operation

In I2C, data transfer occurs between a master device and one or more slave

devices connected on the same bus. The sequence to transfer data is shown in Figure 2-12 [11]**:**

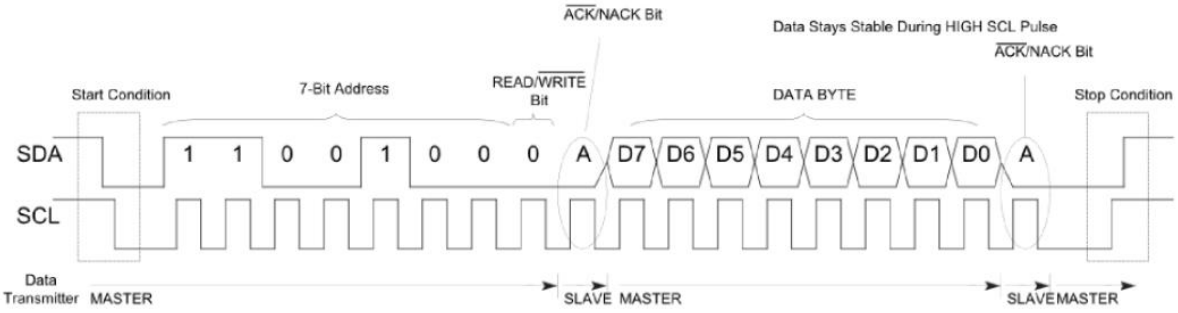


Figure 2-12 : I2C Data Frame [11, Fig 6]

- Start Condition: The master begins communicating by sending a start condition. It is a specific sequence in which the SDA line transitions from high to low while the SCL line remains high.
- Address: The master sends the address of the slave device it wants to communicate with. Depending on the addressing scheme used, the address is a 7-bit or 10-bit value. The most significant bit of the address byte indicates whether it is a read or write operation.
- Acknowledgment: After sending the address, the master releases the SDA line and waits for an acknowledgment from the addressed slave. The slave that matches the address pulls the SDA line low to acknowledge the address.
- Data Transfer: Once the slave acknowledges, the master and slave can transfer data. The master sends or receives data in 8-bit bytes, and each byte is followed by an acknowledgment bit from the receiver.
- Repeated Start: If the master wants to perform multiple read or write operations without releasing control of the bus, it can use a repeated start condition. Instead of sending a stop condition, the master sends a start condition again to initiate a new communication sequence.
- Stop Condition: When the master has finished the data transfer, it sends a stop condition. It is a specific sequence where the SDA line transitions from low to

high while the SCL line remains high. The stop condition indicates the end of the communication sequence.

Throughout the data transfer, the SCL line provides clock synchronization for both the master and slave devices. The master generates clock pulses, and data is sampled on the SDA line during the high period of the clock.

2.9. UART COMMUNICATIONS PROTOCOL

UART (Universal Asynchronous Receiver-Transmitter) is a widely used communication protocol that enables serial communication point-to-point between devices, as shown in Figure 2-13. It is a simple protocol commonly found in microcontrollers, embedded systems, and various electronic devices [12].

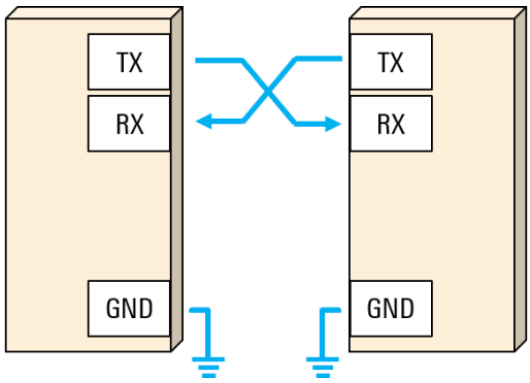


Figure 2-13 : UART Topology [12, Fig 1]

- **Data Format:** UART transfers data sequentially, typically using 8 data bits. However, it can support different data sizes, including 5, 6, 7, or 9 bits. It also includes a start bit at the beginning of each data frame and one or more stop bits at the end to indicate the frame boundaries.
- **Baud Rate:** The baud rate determines the data transmission speed in bits per second (bps). The transmitting and receiving devices must be configured with the same baud rate to ensure proper communication.
- **Asynchronous Communication:** UART is an asynchronous protocol, meaning that the transmitting and receiving devices do not share a common clock signal. Instead, the receiver synchronizes with the incoming data by

detecting the start bit and sampling the subsequent bits at the appropriate intervals.

- **Simplex or Half-Duplex:** UART communication can be simplex or half-duplex. In simplex mode, data flows in only one direction, from the transmitter to the receiver. In half-duplex mode, data transmission can occur in both directions but not simultaneously.
- **Error Detection:** UART does not include built-in error detection or correction mechanisms. However, it is common to use additional protocols, such as parity bits, to detect and handle transmission errors.
- **Hardware Implementation:** UART is often implemented using dedicated hardware, such as UART chips or modules, which handle low-level communication tasks. These hardware components handle the timing, buffering, and framing of data, making UART communication easier to implement in software.

UART frame format consists of Start Bit, Data Bits, Parity Bit (optional), and Stop Bit(s) as in Figure 2-14:

Figure 2-14 : UART Data Frame [12, Fig 2]

- **Start Bit:** The frame starts with a start bit, always a logic low (0). The start bit indicates the beginning of the data transmission and allows the receiver to synchronize with the incoming data.
- **Data Bits:** The actual data bits are transmitted following the start bit. The

number of data bits can vary but is typically 8 bits. However, depending on the configuration, UART can support different data sizes, including 5, 6, 7, or 9 bits.

- **Parity Bit (Optional):** A parity bit is an optional component used for error detection. It can be configured as even parity or odd parity. The parity bit is set to ensure that the total number of logic high bits (1s) in the data and parity bits is either even or odd. The receiver checks the parity bit to detect transmission errors.
- **Stop Bit(s):** The frame concludes with one or more stop bits. The stop bit(s) is always a logic high (1) and indicates the end of the data transmission. The stop bit(s) provide a brief idle period before the start of the next frame.

2.10. FIREBASE DATABASE

Firebase is a comprehensive mobile and web application development platform provided by Google. It offers various tools and services that help developers build, deploy, and manage applications more efficiently. Firebase provides backend infrastructure, APIs, and ready-to-use services, allowing developers to focus on building the front end of their applications as shown in Figure 2-15 [13].

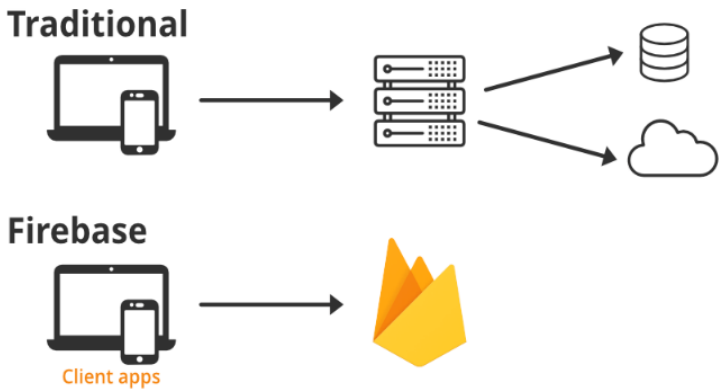


Figure 2-15 : Firebase [13, Fig 1]

- **Authentication:** Provides user authentication and authorization services, supporting various authentication methods such as

email/password, social media logins, and more.

- **Cloud Firestore:** A flexible and scalable NoSQL document database that allows for real-time syncing and offline capabilities.
- **Cloud Functions:** Enables serverless computing, allowing developers to run custom backend code in response to events triggered by Firebase or HTTP requests.
- **Cloud Storage:** Offers secure cloud storage for user-generated content like images, videos, and files.
- **Analytics:** Offers detailed insights into user behavior and app usage, helping developers make data-driven decisions.
- **Cloud Messaging:** Allows sending push notifications and targeted messages to users across multiple platforms

Firebase supports various platforms, including iOS, Android, and web applications, making it a versatile and popular choice for developers looking to build feature-rich applications with minimal backend infrastructure setup.

2.11. SECURITY WITH AES ALGORITHM

2.11.1. Encryption Algorithms.

Symmetric and asymmetric encryption algorithms are distinct cryptographic methods with strengths, weaknesses, and use cases.

Asymmetric encryption is when the encryption key is published for anyone to use to encrypt messages. Only the receiving party can access the decryption key to read the message.

On the other hand, Symmetric Encryption is where the encryption key and decryption key are the same. The parties must have the same key to achieve secure communication, as Figure 2-16 illustrates. In encrypting the firmware, we have used the AES Symmetric Algorithm with Cipher block chaining (CBC) mode.

Figure 2-16 : Symmetric Encryption [25, Fig 1]

2.11.2. AES Algorithm

AES encryption is a type of symmetric block cipher, which is an encryption algorithm created by the National Institute of Standards and Technology (NIST) in 2001. The purpose of this algorithm is to make government data less vulnerable to brute-force attacks [14].

2.11.3. AES Encryption

AES (Advanced Encryption Standard) is a symmetric block cipher, which uses the same key to encrypt and decrypt data. However, the AES encryption algorithm is quite different from standard symmetric encryption. Instead of encrypting the entire message simultaneously, it encrypts it in smaller blocks. Additionally, successive rounds of encryption make the message even more challenging to decode.

The number of rounds required in the AES algorithm depends on the size of the key used. A 128-bit key needs 10 rounds, a 192-bit key needs 12 rounds, and a 256-bit key needs 14 rounds. Each round of the AES algorithm comprises four phases:

a. Substitution: Figure 2-17 illustrates the Substitution step. In this step, the algorithm replaces the plain text with the encrypted text based on a predefined cipher via transform function $S_{(a)} = b$.

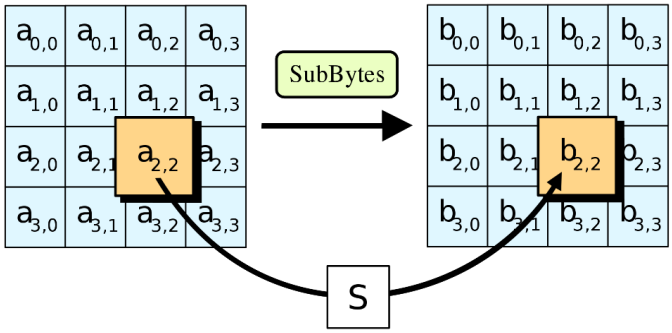


Figure 2-17 : AES Sub Bytes Step [14]

b. Shifting: Figure 2-18 illustrates Shifting step. Except for the first row, all subsequent rows are translated left by the formula N-1 times where N is the row order. For example, row 4 is translated to the left 4-1 =3 times.

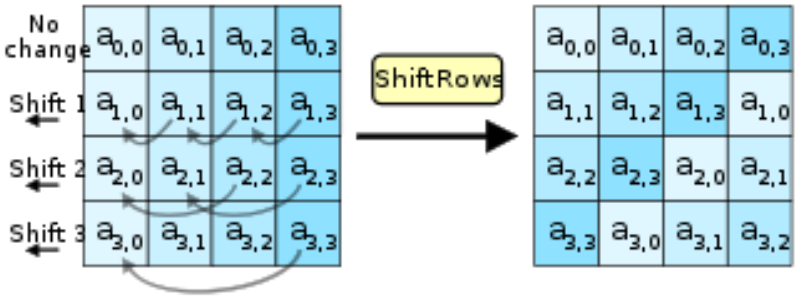


Figure 2-18 : AES Shift Row Step [14]

c. Mixing: Another cipher, the Hill cipher, is used to mix columns using XOR with a C matrix to prevent someone from merely shifting the rows back to begin decrypting the data, as shown in Figure 2-19.

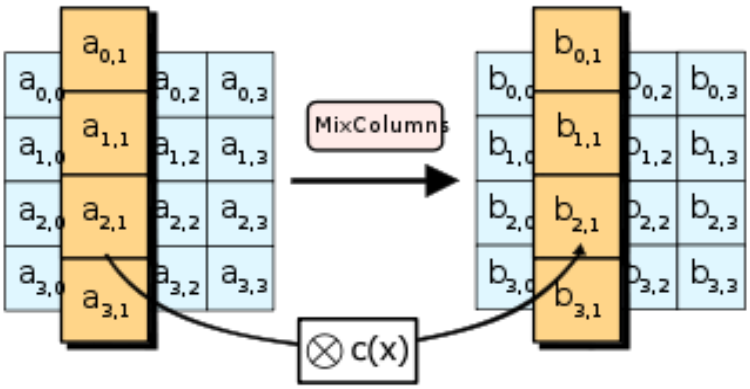


Figure 2-19 : AES Mixing Step [14]

d. Add Round Key: The encryption subkey encrypts that data block using

bitwise XOR. For each round, a subkey of the same size is derived from the main key using Rijndael's key schedule, as shown in Figure 2-20.

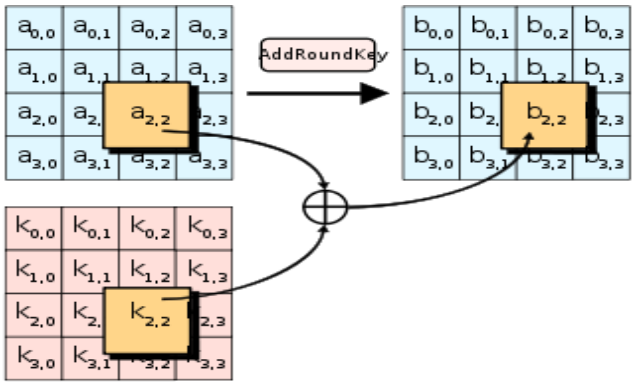


Figure 2-20 : AES Add Round Key Step [14]

As the size of the key increases, the difficulty of cracking the encryption also increases. AES must be secure enough to withstand brute force attacks, which are one of the most common methods used by hackers.

2.11.4. AES Cipher block chaining (CBC) mode

The Cipher Block Chaining (CBC) encryption mode uses the same key to encrypt plaintext blocks and combines them with the previous ones using an XOR operation, as shown in Figure 2-21. This ensures that even if the same plaintext block occurs multiple times in the message, it will be encrypted to a different ciphertext block each time.

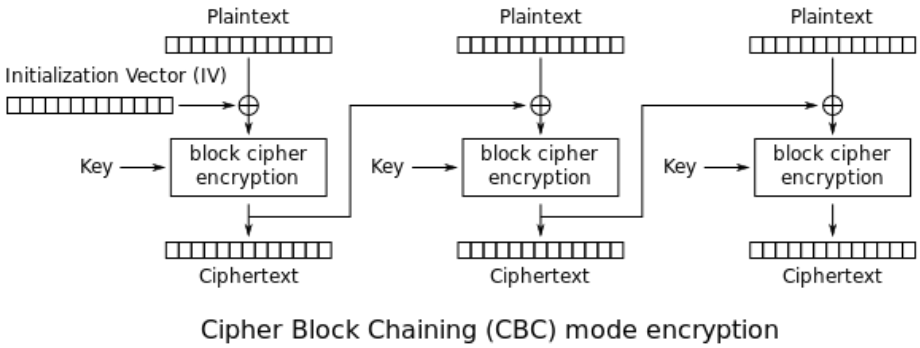


Figure 2-21 : AES Cipher Block Chaining Mode [26]

To implement CBC mode, an initialization vector (IV) is required. The IV is a

random value added to the first plaintext block before encryption. The IV is shared with the receiver and the encrypted message so they can use it to decrypt it.

2.11.5. AES Decryption

Decryption is the process of obtaining the original encrypted data. This process is based on the key received from the sender of the data. AES decryption procedures are similar to encryption but in reverse order. The sender's key is required to decrypt the data. Figure 2-22 illustrates the decryption phase, which consists of four stages: Add Round Key, Inverse Mix Columns, Inverse Shift Rows, and Inverse Sub Bytes.

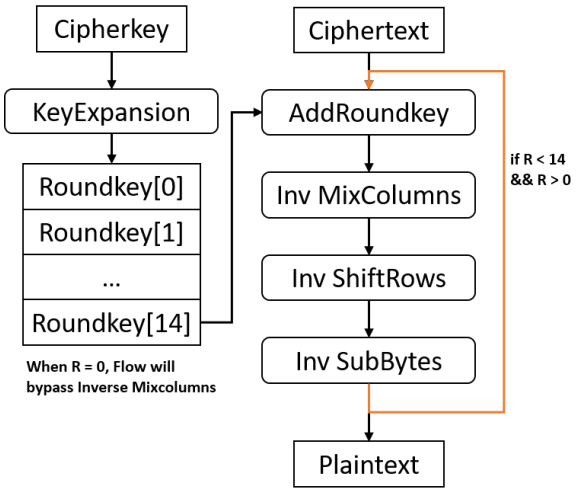


Figure 2-22 : AES Decryption [27]

The receiver receives the key through the Diffie-Hellman key exchange process because it is complex. In this thesis topic, we assume that the key exchange has taken place successfully and that the two parties have obtained the symmetric key.

CHAPTER 3: SYSTEM DESIGN

This chapter presents system specifications and requirements, the design for both hardware and software like the System diagram, Bootloader design, Telematic Unit, Gateway system, Collision system and Lighting system, explains the software flowchart. Then, we discuss the implementation.

3.1. SYSTEM REQUIREMENTS

Functionality:

- FOTA function: Execute update MCU process when Cloud has an updated firmware.
- Progress monitoring: Have a User Interface that the user can recognize, accept, or reject to update new firmware for the specified MCU.
- Application Firmware: Firmware works properly after flashing.

Non-Functionality:

- Performance:
  - Update speed: The update process should be completed as fast as possible, estimate how long firmware will be updated in actual conditions depends on the size of the firmware and Wi-Fi bandwidth, ….
- Availability:
  - User notifications: Provide visual feedback to the user on the progress of the update to indicate which MCU is unavailable in the estimated time.
- Reliability:
  - Safe update process: The update process should not interrupt critical functions and should have mechanisms for rollback in case of issues.
  - Connectivity: The system should have reliable connectivity for downloading CAN, UART, and Wi-Fi.
- Scalability:
  - Both hardware and software can be designed to support many embedded

devices by providing multicast and broadcast capabilities in addition to unicast.

- Security:
  - Secure boot: The system should only boot with a valid firmware, if not valid, MCU will erase that firmware and wait for an update from the developer.
  - Encryption: Communication between Gateway and MCU should be encrypted to protect against hackers.

3.2. SYSTEM BLOCK DIAGRAM AND SYSTEM FLOWCHART

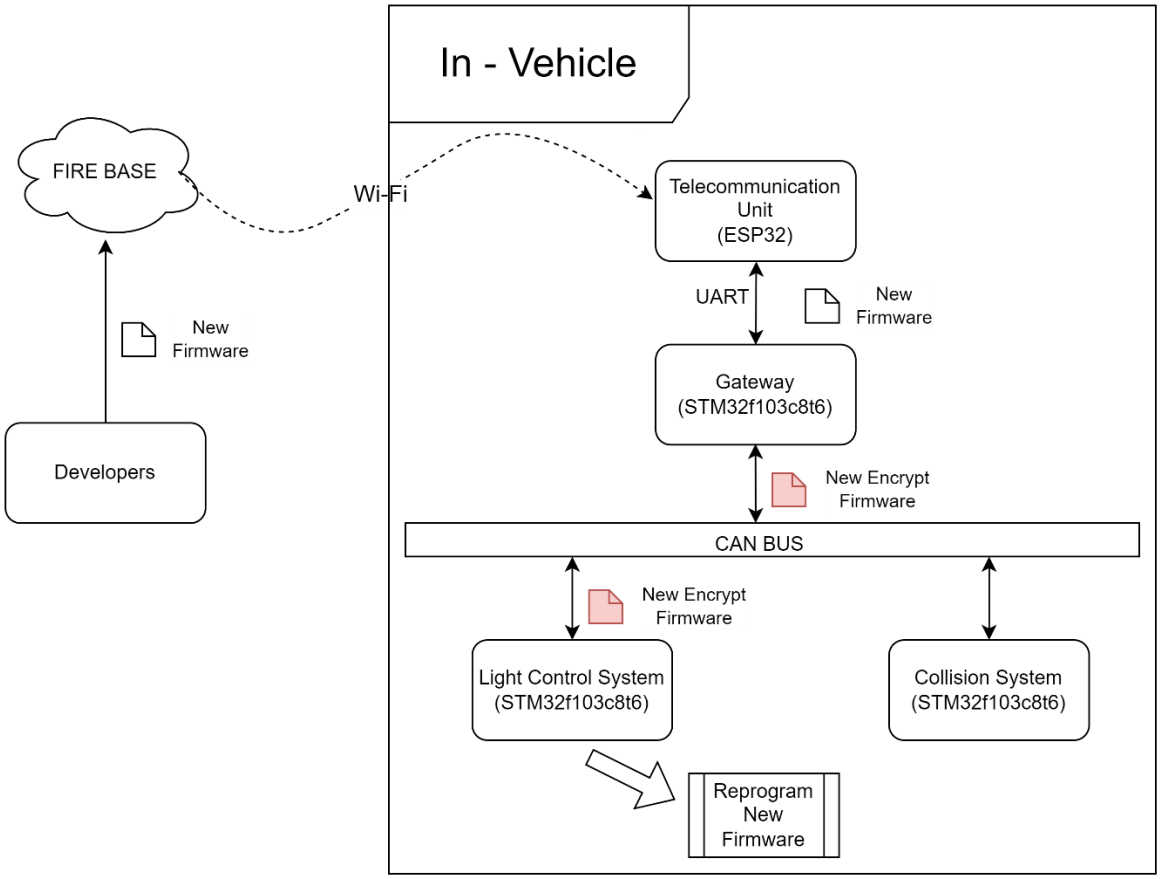


Figure 3-1 : System Block Diagram

Figure 3-1 depicts the overall model of the system, which is designed to include four main blocks: Database Server (Firebase), Telecommunication Unit, Gateway, and target MCU blocks: Lighting System and Collision Avoidance System.

- **Database Server**: We will upload the new update using the Graphic User Interface (GUI) and store information about the products and the latest update. For example, it will contain details such as firmware size and node ID in CAN Bus.
- **Telecommunication Unit**: This unit connects to the server, checks for new firmware, and downloads the firmware if it is connected to Wi-Fi.
- **Gateway**: The Gateway receives the update from the telecommunication unit via UART and also receives important data about the update, such as the size of the firmware. It encrypts the code before sending it to the target ECU.
- **The target MCU:** These MCUs are the destination of the update and are connected to the Gateway via CAN Bus. It performs tasks such as turning on and off car lights and signaling when a collision is imminent.

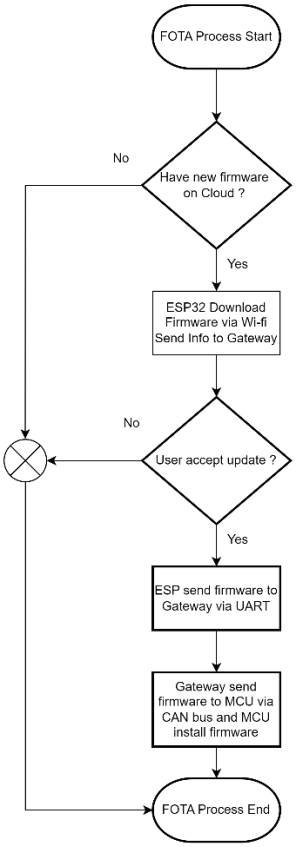


Figure 3-2 : FOTA process flowchart

With the flowchart Figure 3-2, the Telecommunication Unit will check if the firmware is available in the database and download it. Then, it will send an update request signal to the Gateway. The Gateway will then allow the user to accept or decline the update. If the update is declined, the FOTA process will end. If the user accepts, the firmware will be transmitted from the Telecommunication Unit to the Gateway. Then, the Gateway will redirect the firmware to the MCU, which requires the update. Upon receiving the firmware, the MCU will begin the reprogramming process.

3.3. HARDWARE DESIGN AND IMPLEMENT

3.3.1. Telecommunication Unit Hardware Design

The telecommunication Unit (or Telematics Unit in the Automotive Industry) is the part that helps exchange information on vehicles with the outside world via the Internet.

It typically offers a combination of IoT vehicle telematics solutions, cloud solutions, hardware, and software, including GPS tracking, cloud-based platforms with easy integration for multiple partners, telematics sensors, management software, streamlined and accessible dashboards with visuals, automated and scalable reporting, notification and alerting capabilities configurable, compliance parameters, and AI integration for early warning and instant insights [15].

Vehicle telematics applications include GPS tracking, collecting in-vehicle data components and sending them to databases, helping monitor the condition of vehicle components, alerting parts for maintenance, and updating software for vehicle modules ...

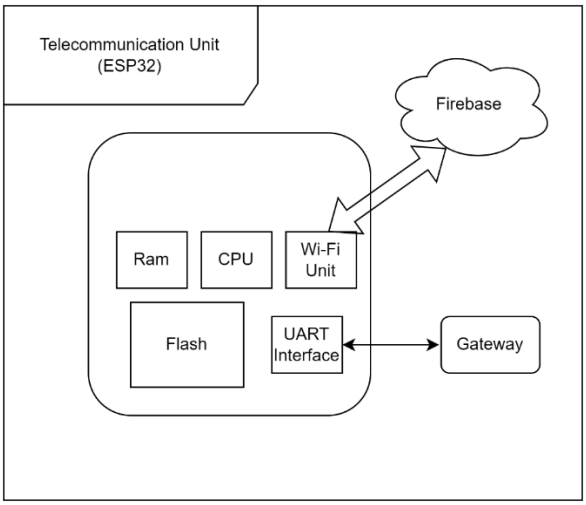


Figure 3-3 : Telecommunication Unit Diagram

In this project, we design a telecommunication unit that can announce firmware updates in the Cloud, download them, and then send them to the Gateway via UART.

The hardware design requirements for the Gateway (Figure 3-3) are as follows:

- Wi-Fi connectivity: it has built-in Wi-Fi capabilities, making it easy to connect to the internet and download firmware updates. This allows devices to detect and download new updates from Firebase remotely without requiring physical access.
- Sufficient memory: The ESP32 typically comes with a significant amount of Flash memory, essential for storing firmware updates. It provides enough space to accommodate the new firmware version, ensuring smooth update processes.

To achieve these features listed above, if we use STM32 Blue Pill, we need a Wi-Fi module for connection with the Cloud because STM32 Blue Pill does not support a Wi-Fi module. Moreover, we need large RAM to store the firmware when downloaded and waiting for transfer. On the other hand, ESP32 has a large amount of RAM for firmware storage, has UART support to connect to the Gateway, and has intense processing speed. We decided to use the ESP32 dev kit as a

Telecommunication Unit. Table 3-1 indicates the connection between STM32 and ESP32.

Table 3-1 : The connection table Telecommunication Unit

Peripheral	Peripheral pin	ESP32 pin	Label Name
STM32	PA10	G17	UART_RX
	PA9	G16	UART_TX

3.3.2. Gateway Unit Hardware Design

One of the most important systems in FOTA has many functions, such as encoding the code, forwarding the update to the correct ECU, and ensuring that the update has been completed successfully to notify the server again.

Gateway will also inform the user of new updates to give the user a choice to accept or reject the update request, which will be achieved through a graphical interface; if the user rejects the update, the server will drop that update, and if the user accepts, the updated firmware will be downloaded and processed.

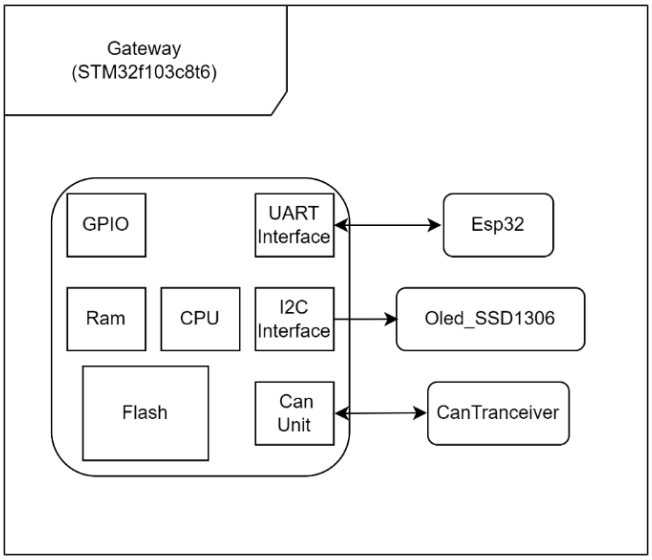


Figure 3-4 : Gateway Unit Diagram

The hardware design requirements for the Gateway Unit (Figure 3-4) are as follows:

- Communication with the Telecommunication Unit via UART for receiving

updates and firmware.

- Create a user interface to get responses from users who accept updates and display the update process.
- Communication with Target MCU via CAN Bus for transmitting updated firmware.

To achieve these features listed above, we decided to use four main components:

1. STM32F103C8T6 Micro-controller.
2. Button
3. Oled-SSD1306
4. Can Transceiver SN65HVD

a) STM32F103C8T6 Micro-controller.

Why did we choose STM32F103C8T6 as the Gateway core?

Table 3-2 : Comparison table between STM32f103x and stm32f104x [24]

	STM32F103	STM32F401	STM32F411
Clock Freq	72MHz	84MHz	100MHz
Flash	64KB	256KB	256KB
SRAM	20KB	64KB	128KB
DIO	30	32	32
UART Port	3	3	3
I2C Port	2	3	3
CAN Port	1	No	No
Timers	4	8	8

Based on Table 3-2 and Table 2-1 in Chapter 2, The STM32F103C8T6 features

ADC, timers, standard and advanced communication interfaces such as UART, I2C, and CAN, which are needed for communication with another module in the system. A higher model like STM32F4x has higher speed, more RAM, flash size, and peripherals port, … so it costs more than STM32F103C8T6 but is not necessary; STM32F103C8T6 provides enough peripherals for this project. Moreover, it has a large community to discuss difficulties when implementing and fixing bugs. That is why the project team uses STM32F103C8T6 as the Gateway.

b) Button.

A button (Figure 3-5) is a switch that controls a process. Its surface is flat or shaped and easily pressed or released.

It is the user's way to interact with the gateway to accept or reject the update.

Figure 3-5 : Button Image

Table 3-3 : Push button specifications

Dimension	6x6x6 mm
Color	Black
Current / Voltage	50 mA @ 12 V dc
Operating Temperature	-35 → +85°C
Signal output	Digital

c) Oled-SSD1306

SSD1306 (Figure 3-6) is an Integrated Circuit with a controller for a dot-matrix graphic display system made of organic or polymer light-emitting diodes. It comprises 128 segments and 64 commands and is designed for a joint cathode-type

OLED panel [16].

Figure 3-6 : Oled SSD1306 [16]

Table 3-4 : Oled SSD1306 specifications

Resolution	128 x 64 dot matrix panel
Color	Black and White
Operating Voltage	1.65V to 3.3V
Operating Temperature	-40°C to 85°C
Connect Type	I2C

Features of Oled-SSD1306 displays:

- Deliver sharp images and content.
- Stable and quicker response times.
- Wide viewing angles.
- Support variable of MCU.

d) Can Transceiver SN65HVD230.

Tzo transmit and receive CAN messages with an STM32, a component that can convert the signal into Can High and Can Low is necessary.

The SN65HVD230 CAN Board (Figure 3-7) is a helpful accessory with an onboard CAN transceiver SN65HVD230. Based on Table 3-5, this board operates on 3.3V and has ESD protection, and signal speed up to 1Mbps making it the perfect choice for connecting microcontrollers to the CAN network [17].

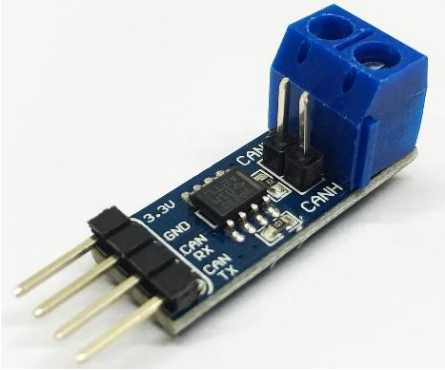


Figure 3-7 : SN65HVD230 CAN Board [17]

Table 3-5 : SN65HVD230 CAN Board specifications

Operating Voltage	3.3V
ESD Protection	Yes
Thermal Shutdown Protection	Yes
Signaling Rates	Up to 1 Mbps

e) The connection table

Tables 3-6 indicates the connection of the pins in Gateway.

Table 3-6 : The connection table of Gateway

Peripheral	Peripheral pin	STM32 pin	Label Name
Button	P1	PA1	SWITCH_BNT
	P2	PA2	OK_BNT
SN65HVD230 CAN Board	CAN_RX	PA11	CAN_RX
	CAN_TX	PA12	CAN_TX
Oled_SSD1306	SDA	PB7	I2C1_SDA
	SCL	PB6	I2C1_SCL
ESP32	TX0	PA10	USART1_RX
	RX0	PA9	USART1_TX

3.3.3. Lighting System Hardware Design

It is MCU1 which will be implemented and used as a test case for FOTA. Car lighting systems are very important in automotive technology, particularly where road safety is concerned. If headlights were suddenly to fail at night and high speed, the result could be catastrophic. Many techniques have been used, ranging from automatic changeover circuits to thermal circuit breakers, … [18].

However, in this project, we focus on the FOTA feature, so we decided to design a basic lighting system, with three LEDs: a headlight, a backlight, a brake light, a button, and a switch.

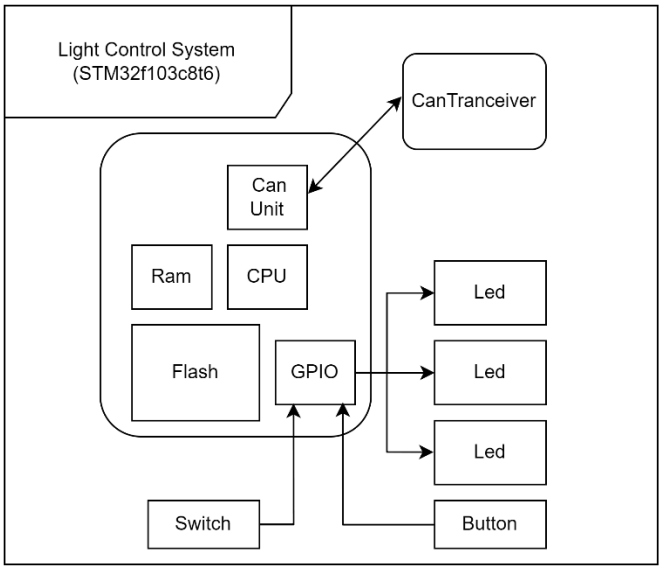


Figure 3-8 : Lighting System Diagram

The hardware design requirements for the Lighting System (Figure 3-8) are as follows:

- Communication with Gateway Unit via CAN Bus for receiving updated firmware.
- Connect to 3 LEDs to simulate headlight, backlight, and brake light.
- Connect the switch to control the headlight and backlight.
- Connect the button to simulate the brake.

To achieve these features listed above, we decided to use five main components:

1. STM32F103C8T6 Micro-controller.
2. Button.
3. DIP Switch.
4. Can Transceiver SN65HVD.
5. LED 5mm.

For any of the components that we introduce above, we will not mention it again to avoid duplication.

a) DIP Switch

DIP switches (Figure 3-9) are manual electrical switches packaged with other switches in the group in a standard dual inline (DIP) package. This term can refer to individual switches or the entire device. This type of switch is designed for use on printed circuit boards and other electronic components and is often used to customize the operation of electronic devices in specific situations.

DIP switches are still used in some remote controls to avoid interference; for example, to control a ceiling fan (and its lighting fixture) that has been retrofitted into a single-circuit junction box. DIP switches set a different frequency or radio address for each transmitter/receiver pair so that multiple devices can be installed without accidentally controlling each other [19].

In this project, we use the DIP Switch to control the LED and turn on/off headlight and backlight..

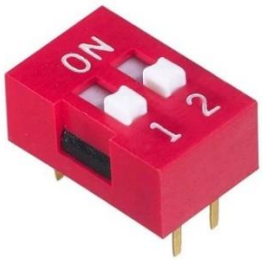


Figure 3-9 : DIP Switch [19]

Table 3-7 indicates the specifications of the DIP.

Table 3-7 : DIP Switch specifications

Contact Rating	100mA, 50VDC
Contact Resistance	500 mOhm max
Operating Temperature	-25C~+85C
Mechanical Life	>= 100,000 Cycles
Electrical Life	2,000 Cycles per Position

b) LED 5mm

LEDs (Figure 3-10) are two-wire semiconductor light sources that emit light when activated. When an appropriate voltage is applied to the LED terminal, the electrons can recombine with the electron holes inside the device and release energy as photons. This effect is called electroluminescence. The color of an LED is determined by the energy band gap of the semiconductor [20].

Figure 3-10 : Led 5mm [20]

Table 3-8 : Led 5mm specifications

Operating Voltage	1.8V to 2V
Operating Current	10 to 20mA
Operating Temperature	-30℃ to +85℃
Luminous Intensity	2,000 – 4000 millicandela

Table 3-8 indicates the specifications of the Led. The operation voltage required to turn the LED on depends on the color of the LED. We can connect the LED directly to the source if we provide the correct value for the forward voltage. If the voltage is higher than using a resistor in series with the Led, to calculate the value

of the resistor R, use the Formula 3.1:

$$R = \frac{V_S - V_{Led}}{I_{Led}} \times X \qquad (3.1)\ [20]$$

With:

- V_S is the supply voltage
- V_{LED} is the forward voltage of Led
- X represents the number of Leds connected in series
- I_{LED} is LED current

According to the formula above, we used 120 Ohm with Yellow Led as a car lighting bulb.

c) The connection table

Tables 3-9 indicates the connection of the pins in Lighting System.

Table 3-9 : The connection table of Lighting System

Peripheral	Peripheral pin	STM32 pin	Label Name
Button		PA6	Brake
Led		PA5	Brake_Light
DIP Switch		PA4	Back_Light
		PA3	Head_Light
SN65HVD230 CAN Board	CAN_RX	PA11	CAN_RX
	CAN_TX	PA12	CAN_TX

3.3.4. Collision Avoidance System Hardware Design

It is MCU2 which will be implemented and used as a test case for FOTA.

Collision avoidance systems are an indispensable part of the driver assistance system in automotive technology. Their primary goal is to prevent and minimize collisions. They harness the power of advanced sensors, such as radar, lidar, and cameras, to meticulously detect potential obstacles on the road ahead. These advanced technologies work harmoniously to provide drivers with real-time data,

allowing them to make informed decisions and take the necessary actions to avoid accidents [21].

In practical applications, collision systems detect parking and obstacles. Therefore, for this project, we decided to design a basic collision system in which the buzzer is activated when we encounter an obstacle, and the LED glows.

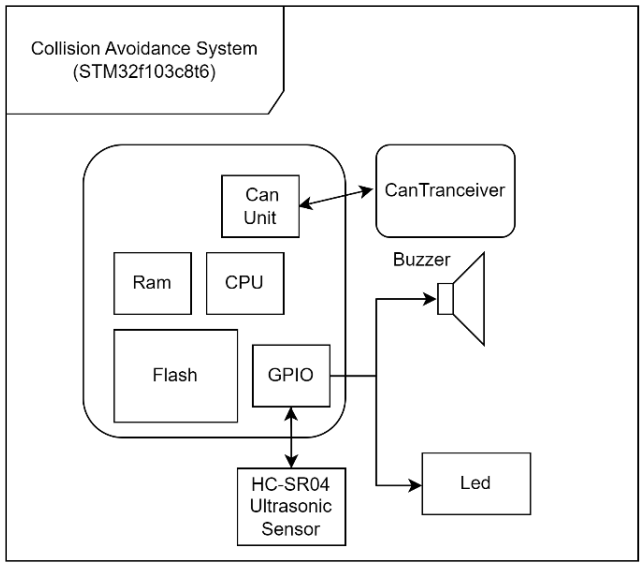


Figure 3-11 : Collision Avoidance System Diagram

The hardware design requirements for the Collision System (Figure 3-11) are as follows:

- Communication with Gateway Unit via CAN Bus for receiving updated firmware.
- Connect to Buzzer, Led to notify the user
- Connecting an ultrasonic sensor for collision notification

To achieve these features listed above, we decided to use five main components:

1. STM32F103C8T6 Micro-controller.
2. Led.
3. Buzzer.
4. HC-SR04 Ultrasonic Sensor.
5. Can Transceiver SN65HVD

For any of the components that we introduce above, we will not mention it again to avoid duplication.

a) HC-SR04 Ultrasonic Sensor

The HC-SR04 ultrasonic sensor (Figure 3-12) uses ultrasonic energy to measure the distance to objects in the 2-400 cm range with an accuracy of 0.3 cm [22].

Figure 3-12 : HC-SR04 Image [22, Fig 1]

he ultrasonic sensor HC-SR04 has an operation mechanism, as shown in Figure 3-13. The transmitter circuit input electrical signal is a 10μs pulse input to the activation pin of the HCSR04M sensor. The transmitter circuit converts the electrical signal into a 40KHz burst of 8 pulses of sonar waves. On the other hand, the ultrasonic receiver circuit listens for these ultrasonic waves generated by the transmitter circuit. Then, the distance is calculated, which we will discuss in more detail in software design.

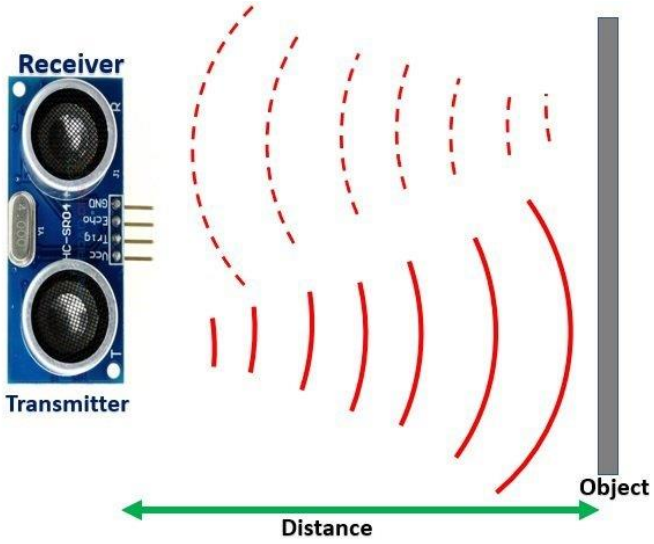


Figure 3-13 : Mechanism of HC-SR04 [22, Fig 3]

In this project, we use HC-SR04 to detect obstacles to support parking, if an obstacle is detected near 4cm, it will output a signal for the processing MCU to notify the user. In addition, we created the specification Table 3-10 for HC-SR04.

Table 3-10 : HC-SR04 specifications

Operating Voltage	DC 5 V
Operating Current	15mA
Working Frequency	40kHz
Max Range	4m
Min Range	2cm

b) Buzzer

Piezo, a buzzer (Figure 3-14), is a component used to create sound. It is a digital component that can be connected to a digital output and emits a tone when the output is HIGH. Additionally, it can be connected to an analog pulse width modulation output to create a variety of tones and effects. This module can provide audio feedback to our application, such as the sound of a button pressed on a digital watch.

In this project, we use this to make sound for collision warnings. In addition, we created the specification Table 3-11 for the buzzer.

Figure 3-14 : Buzzer Image

Table 3-11 : Buzzer Specification

Operating Voltage	DC 5 V – 24V
Operating Current	≤30mA
Operating Temperature	– 20° C to +60°C
Sound Output at 10cm	≥85dB
The frequency range	3,300Hz

c) The connection table

Tables 3-12 indicates the connection of the pins in Collision Avoidance System

Table 3-12 : The connection table of Collision Avoidance System

Peripheral	Peripheral pin	STM32 pin	Label Name
Buzzer		PA3	BUZZER_PIN
Led		PA4	LED_PIN
HC-SR04	TRIG	PA1	TRIG_PIN
	ECHO	PA2	ECHO_PIN
SN65HVD230 CAN Board	CAN_RX	PA11	CAN_RX
	CAN_TX	PA12	CAN_TX

3.4. SOFTWARE DESIGN AND IMPLEMENT

3.4.1. Telecommunication Unit Software Design

3.4.1.1. Telecommunication Unit Tasks

The telecommunication unit is responsible for detecting new updates, sending requests to Gateway, downloading, and sending the new update to Gateway.

3.4.1.2. Telecommunication Unit Flowchart

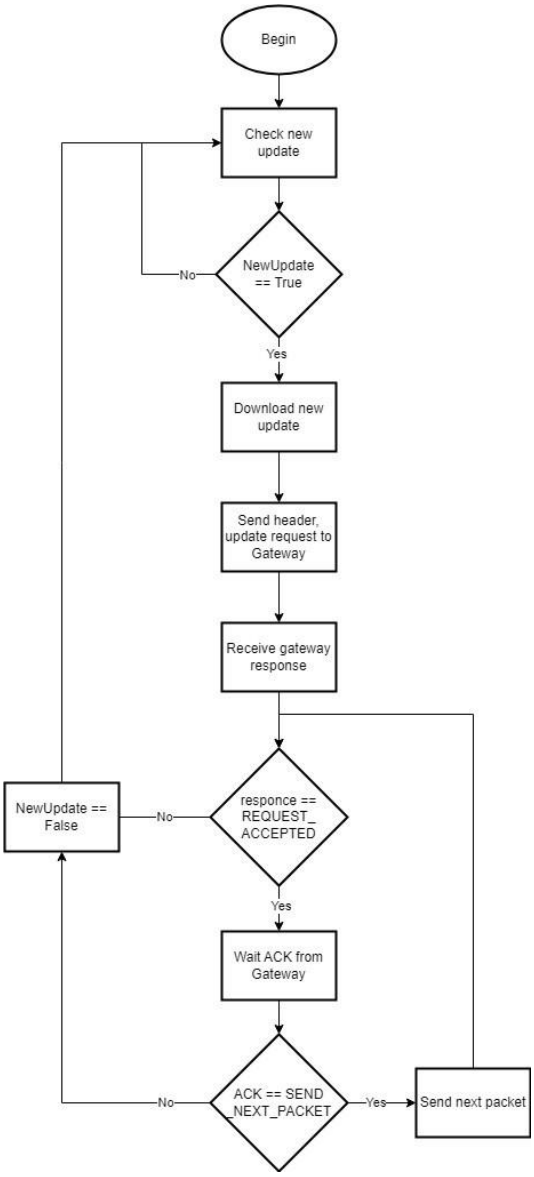


Figure 3-15 : Telecommunication Unit Flowchart

According to the flowchart Figure 3-15, The Telecommunication Unit will infinitely loop to detect new updates, until the new update is detected on Firebase. First, it will download the new update through Wi-Fi. Next, it will send the file header to the Gateway Unit, including file size and Node ID, and send the update request to the Gateway. If the Gateway unit responds to “Reject update”, the update will be canceled, and the Telecommunication Unit will go back to its loop to detect a new update. If the Gateway responds “Accept Update”, the Telecommunication unit will send the new update to Gateway through UART, and when it is done, the

Telecommunication Unit will go back to detect the new update loop.

3.4.1.3. Telecommunication Unit State Machine

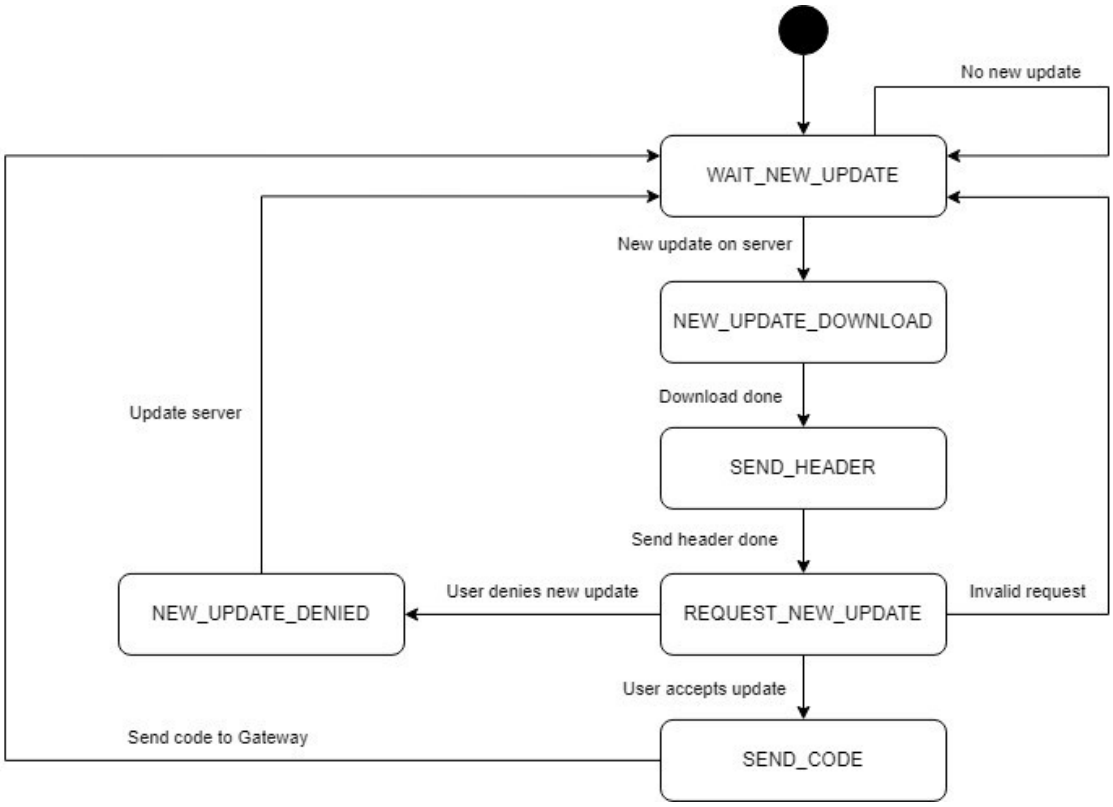


Figure 3-16 : Telecommunication Unit State Machine

In Figure 3-16, we see the state machine of the entire Telecommunication Unit system. The system state is a variable declared static, which will persist and hold assigned values throughout the operation. It has six operating states:

- WAIT_NEW_UPDATE: Detect when the new update is uploaded.
- NEW_UPDATE_DOWNLOAD: Download the update from Firebase.
- SEND_HEADER: Send updated file information to Gateway (File size, Node ID).
- REQUEST_NEW_UPDATE: Send a new update request to Gateway and wait for Gateway's response.
- NEW_UPDATE_DENIED: The Gateway denies the new update. The telecommunication unit will update the real-time database.

- SEND_CODE: The Gateway accepts the new update. Firmware is sent to the Gateway and, when it is done, updated in the real-time database.

3.4.1.4. Firebase Connection

We utilize Firebase to manage and distribute firmware updates to the user. It provides real-time updates, guaranteeing that the user's devices will promptly receive the latest firmware updates. Additionally, it offers excellent built-in scalability and security to efficiently and securely manage many updates.

In Figure 3-17, we will store critical information for updating new firmware. This will include the file size and the ID of the node we intend to update. It will also have a "New_update" flag to notify the telecommunication unit of a new update on the server. Lastly, the link to the new update file will direct to the new firmware files stored in Firebase Storage (Figure 3-18).

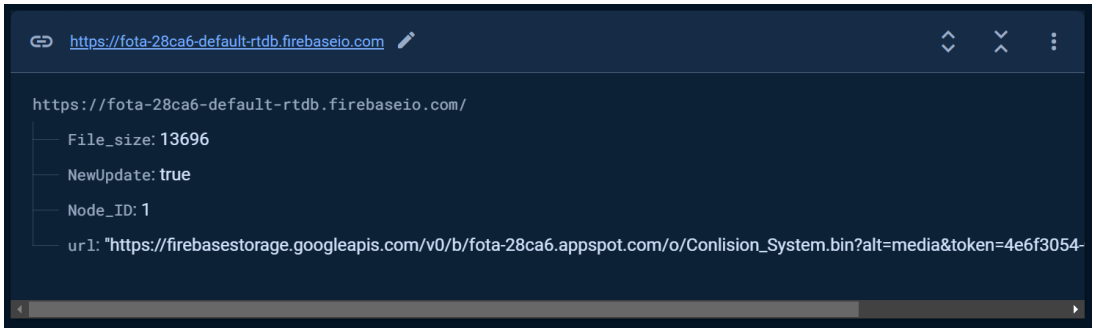


Figure 3-17 : Firebase Realtime Database

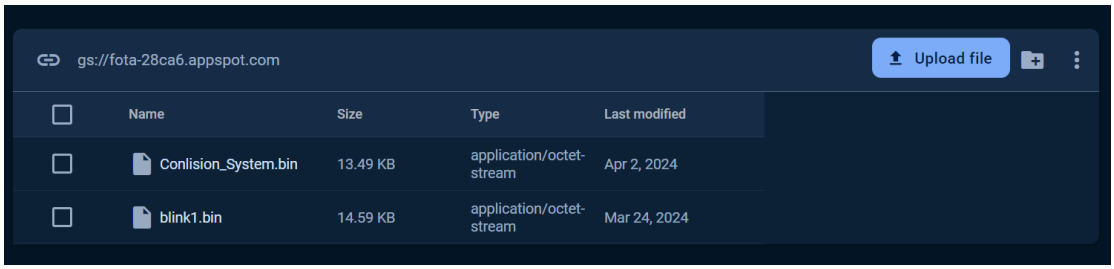


Figure 3-18 : Firebase Storage

3.4.2. Graphic User Interface (GUI)

A GUI can provide a central location to manage updates for all projects. Developers can easily see available updates, compare versions, and initiate updates

with a few clicks.

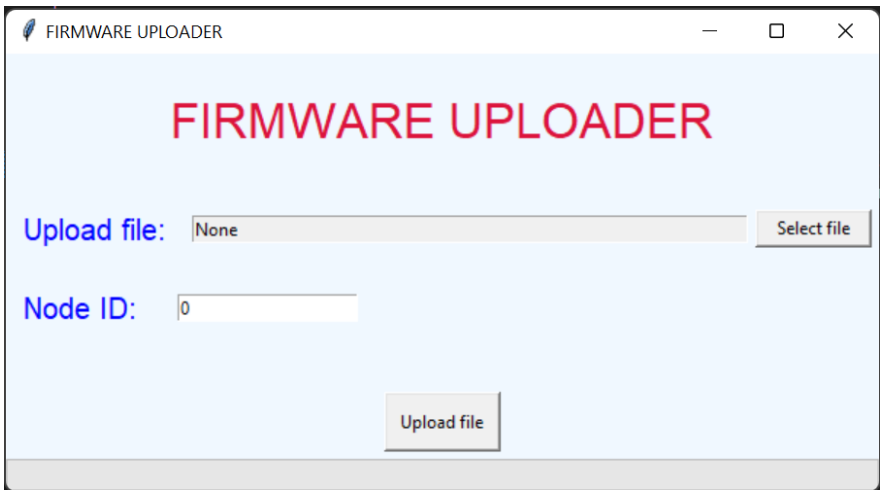


Figure 3-19 : GUI

The developer can select the update file to upload, choose the MCU that needs to be updated, and press upload file (Figure 3-19), which will automatically upload the file to the Firebase server and update the Node ID and NewUpdate flag.

3.4.3. Gateway Software Design

The Gateway is the bridge between the telecommunications unit and the target MCUs. It is very important, especially for systems with many MCUs, such as cars. The Gateway receives code from the Telecommunication Unit via UART protocol and forwards the code to the target MCU via CAN bus.

3.4.3.1. Gateway Tasks

Gateway has the following main tasks:

- Stores information about connected MCUs. This information includes the MCU ID, a software version of every MCU, and any critical data regarding the connected MCUs.
- Calculate CRC to handle errors if firmware sends incorrectly.
- Determine which MCU this firmware is for and make sure the firmware is sent to the correct MCU. As stated, the port has

information about all MCUs, including MCU IDs, to determine which MCUs the update targets.

- Notify the node when the update is complete so the node can notify the same goes for the server when the code update is complete.
- Encrypt data firmware before sending it to MCU.

3.4.3.2. Gateway Flowchart

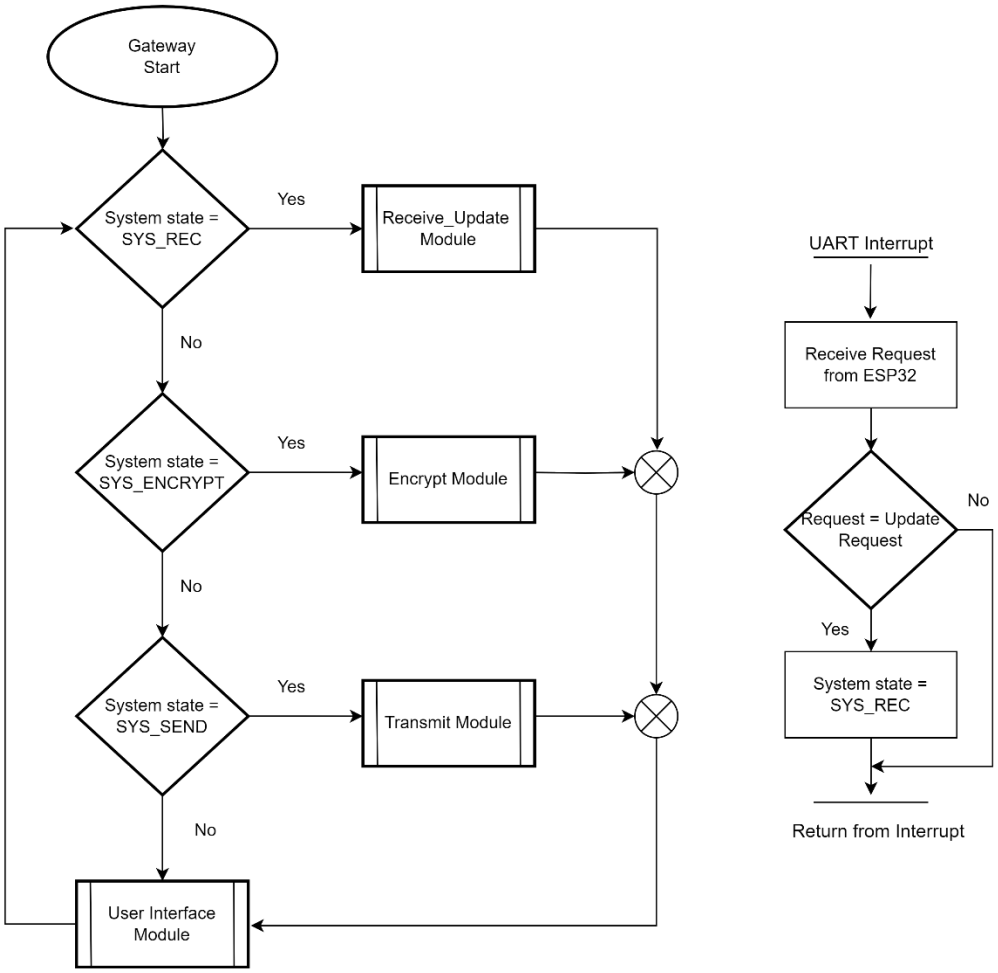


Figure 3-20 : Gateway Flowchart

According to the flowchart Figure 3-20, the gateway will perform an infinite loop and execute a module depending on the system state. Initialize the system state as IDLE. This status will remain until there is one interrupt signal from the Telecommunication Unit. It will then proceed to the FOTA process. We will discuss

the details of each module below.

3.4.3.3. RTE in Gateway

As we discussed in Chapter 2 about RTE, we're now going to implement that mechanism in the Gateway, as shown in Figure 3-21.

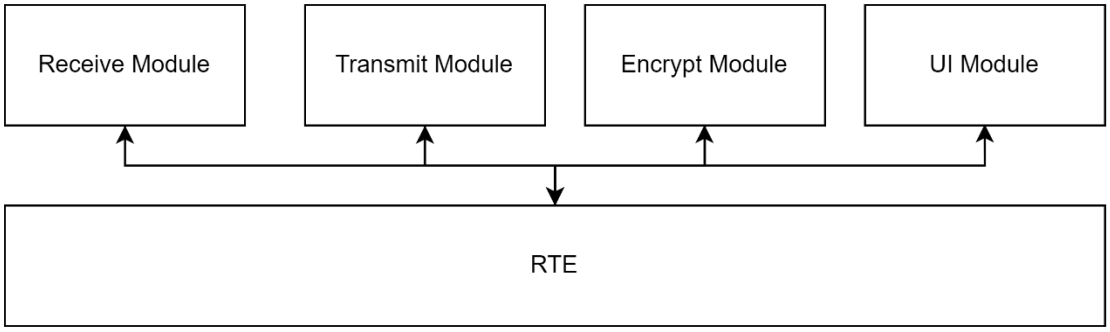


Figure 3-21 : RTE in Gateway

RTE is an environment where modules in the system can communicate and exchange information with each other, and it also arranges the sequence of operation of modules in the system. In this project, we create several ports to read and write variables and buffers, which will help modules get each other's information. Specifically, how will be clearly outlined below:

- Global_CodeSizeValue: contains firmware size.
- Global_NodeId: Contains the ID of the MCU.
- Global_HeaderAckFlag: "1" if the header has been received, "0" if not
- Global_BufferPtr: contains the pointer of the Encrypt buffer.
- Global_BufferFlag: "0" if Encrypt buffer was transmitted, "1" if not.
- Global_UserResponse: "1" if the user accepts, "0" if the user rejects.
- Global_UpdateProgress: Contains percent of process value.

3.4.3.4. Gateway State Machine

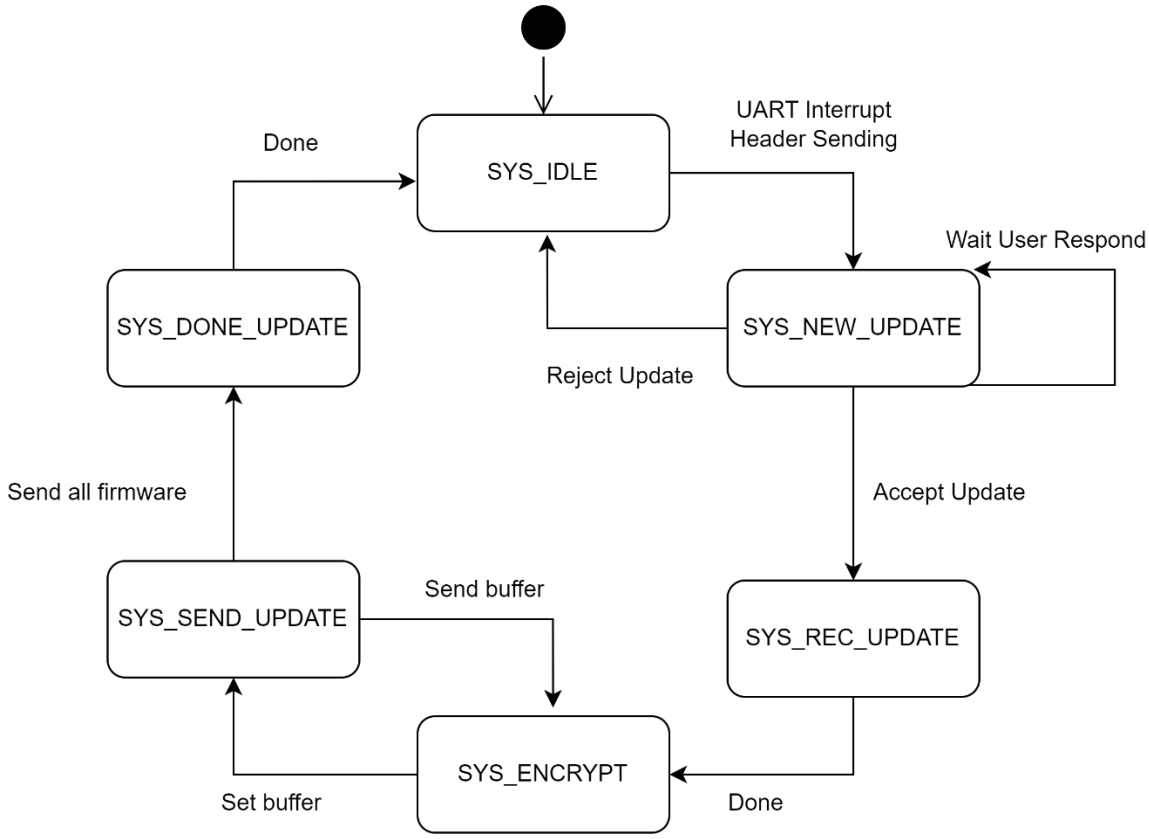


Figure 3-22 : Gateway System States

In Figure 3-22, we see the state machine of the entire gateway system. The system state is a variable declared as static, which will persist and hold assigned values throughout the operation. It has six operating states:

- SYS_IDLE: When there are no active modules other than the UI.
- SYS_NEW_UPDATE: Let the UI get feedback from users.
- SYS_REC_UPDATE: When we need the Receive module to work.
- SYS_ENCRYPT: When we need the Encrypt module to work.
- SYS_SEND_UPDATE: When we need the Transmit module to work.
- SYS_DONE_UPDATE: Re-initialize modules.

3.4.3.5. Communication Sequence

We also design a data transmission sequence that waits for a response before sending the next data, ensuring that the data is received correctly and sufficiently.

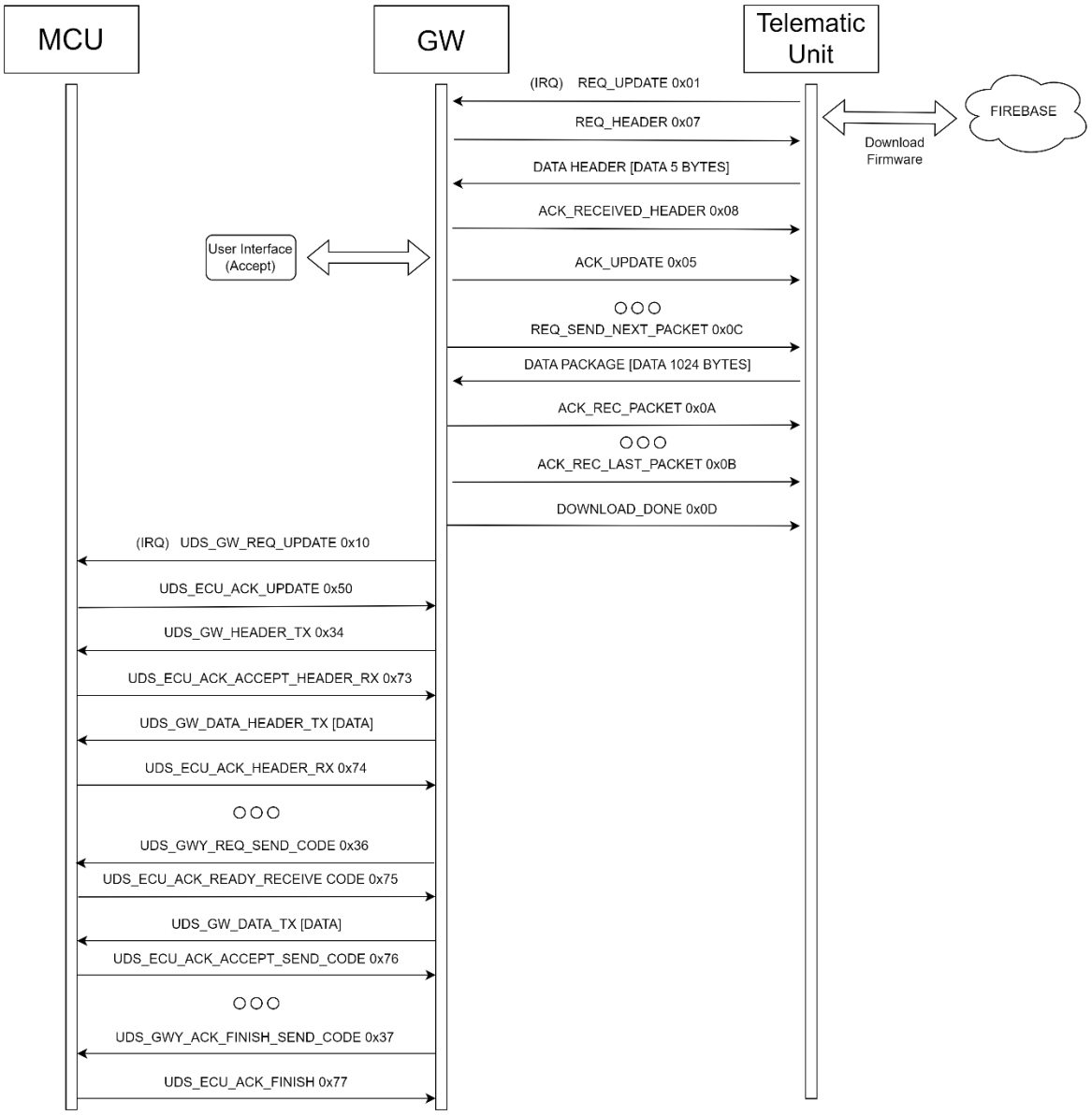


Figure 3-23 : Gateway Communication Sequence

The communication sequence of the Gateway is shown in Figure 3-23:

1. First, the Telecommunication Unit downloads firmware from Firebase, then sends an interrupt UART signal (0x01) to start the FOTA process.
2. Gateway requests send header (0x07) have information about nodeID which MCU receives this update, size of firmware…
3. After receiving the header, Gateway will wait for the user to respond whether to accept or reject this update. If rejected, Gateway sends the reject code and

finishes the process. If accepted, Gateway sends a request package of firmware (0x0C) and sends ACK (0x0A) if it receives the packet until it receives all firmware, then ends the UART protocol (0x0D).

4. Next, Gateway will send a UDS request (0x10) to reprogram the MCU process.
5. After the MCU accepts the process by sending an ACK (0x40), the Gateway sends a header including the size of firmware and CRC code and, in turn, sends data through to the MCU, only transferring the next packet if it receives an ACK code from the MCU (0x75).
6. Upon receiving the final packet, the MCU returns the ACK finish code (0x76) and finishes the FOTA process (0x77).

3.4.3.6. Receive Module

Based on the Flow Chart Figure 3-24 below, the receiver module has the following internal state operation:

- RX_IDLE: The module waits for a signal to start the process of receiving firmware data. If there is a signal, switch the status to RECEIVE HEADER.
- RX_RECEIVE HEADER: Receive and verify the header of the firmware data, including code size and node ID. It returns to the IDLE state to wait for a response signal from the User Interface.
- RX_ACCEPT: Receives acceptance from the User Interface, the module transitions to RX_RECEIVE_DATA state.
- RX_REJECT: Returns to the IDLE state.
- RX_RECEIVE_PACKET: In this state, the module receives and stores the firmware data from the source. It also checks for errors during the reception process to ensure data integrity.
- RX_END: The module transitions to the END state when the firmware data

reception process is complete. It signals that the process has finished and returned to the IDLE state.

In progress, it also calculates the percentage of progress for the UI to show up on the screen via the following Formula 3.1:

$$\text{Percent} = \frac{\text{Received Packet}}{\text{Total Packet}} \times 100 \qquad (3.1)$$

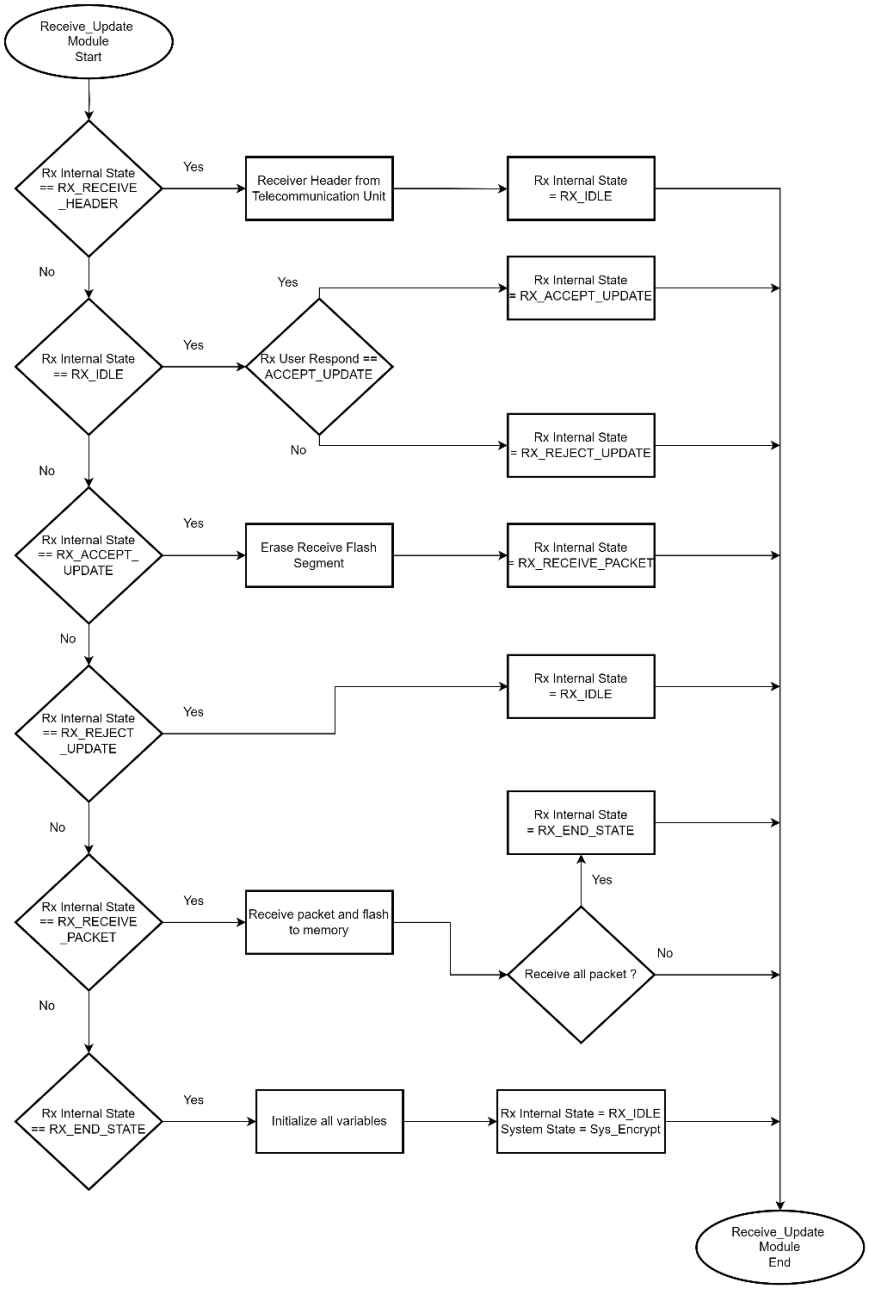


Figure 3-24 : Receive Module Flowchart

3.4.3.7. Encrypt Module

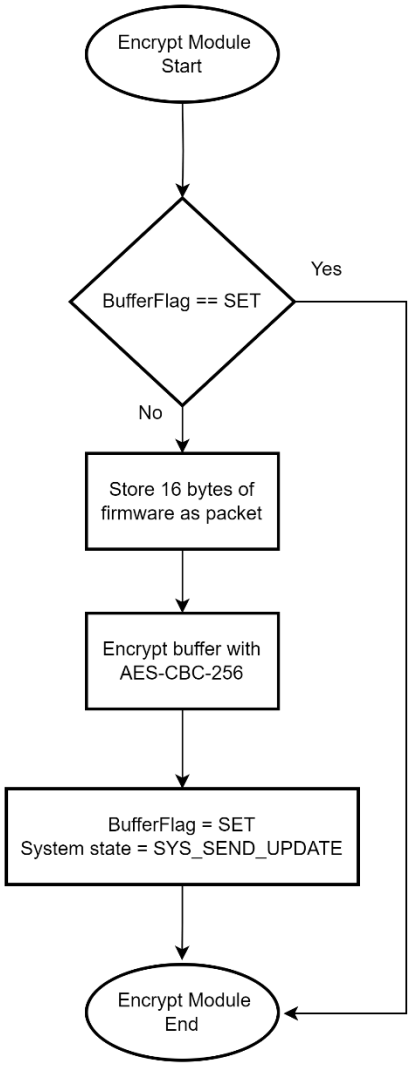


Figure 3-25 : Encrypt Module Flowchart

Based on the flowchart in Figure 3-25, this module performs the following steps: It saves 16 bytes of firmware to a buffer, encrypts the buffer using the AES-CBC-128 standard, and then changes the system state to SYS_SEND_UPDATE to transfer the encrypted buffer. It checks the BufferFlag in RTE - if the data has not been transferred, it sets the flag to "1" to prevent overwriting the buffer. Once the data has been transferred, the flag is set to "0" to allow the process to continue.

3.4.3.8. Transmit Module

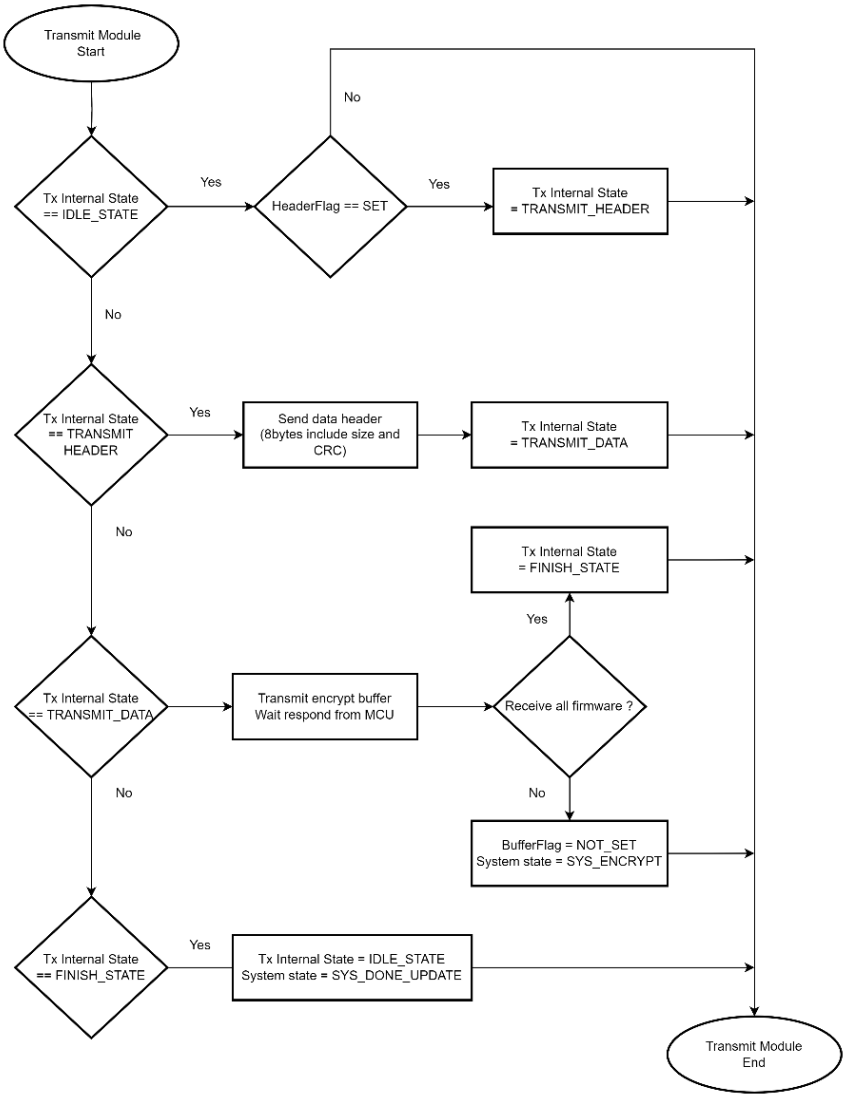


Figure 3-26 : Transmit Module Flowchart

Figure 3-26, the Transmit module has the following internal state operation:

- TX_IDLE: The module is in its default state, waiting for the Header flag to start the transmission process.
- TX_TRANSMIT HEADER: Upon receiving the activation signal from the TX_IDLE state, the module transitions to the TX_TRANSMIT HEADER state. Here, it prepares to transmit the data header to be sent.
- TX_TRANSMIT DATA: The module moves to the TX_TRANSMIT DATA state after successfully transmitting the header. In this state, it sends

the encrypted data buffer from the Encrypt module to the MCU.

- TX_END: The module transitions to the TX_END state once the data is transmitted. It signals that the transmission process has finished and returned to the TX_IDLE state.

3.4.3.9. User Interface Module

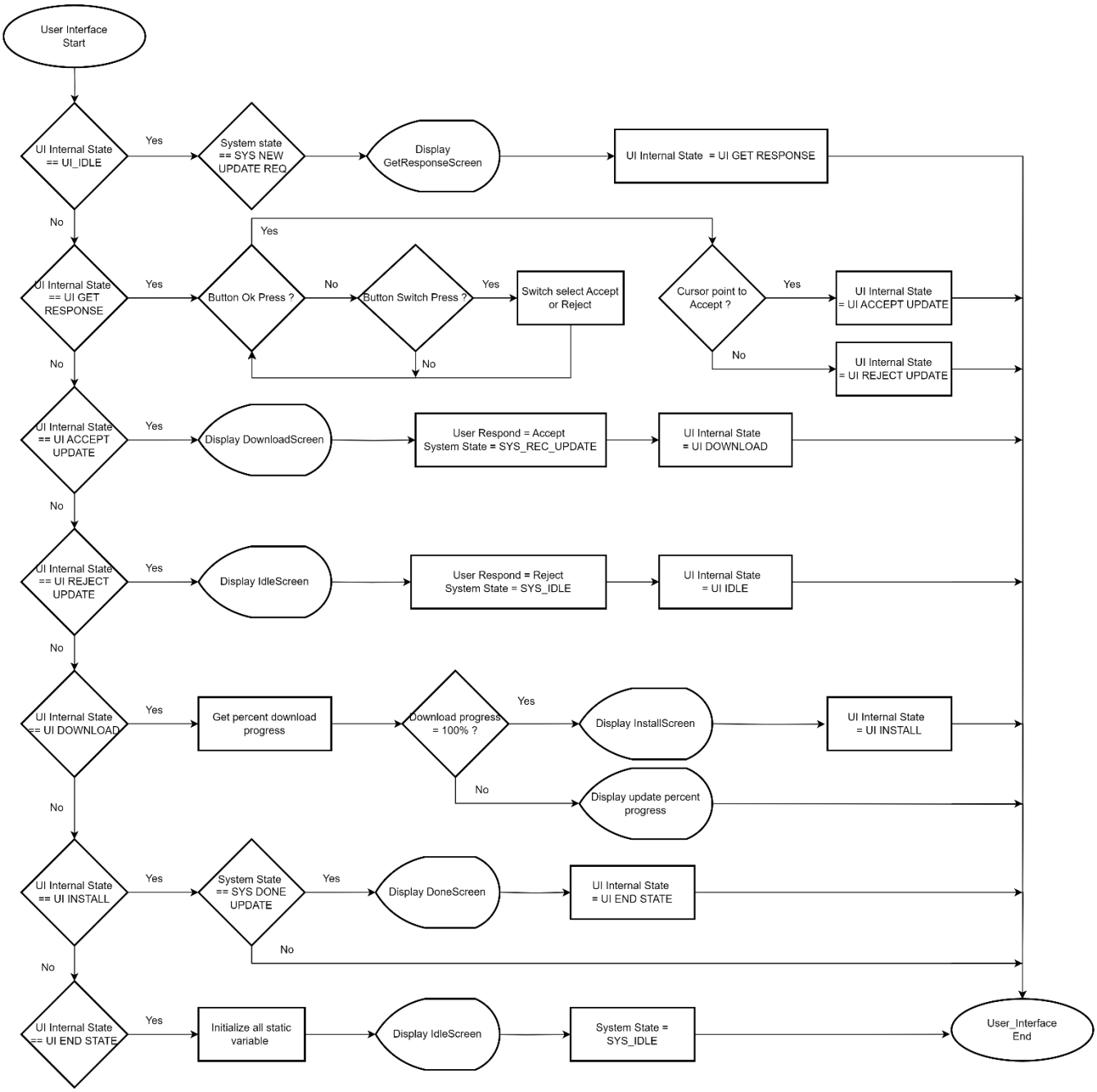


Figure 3-27 : User Interface Module Flowchart

Based on flowchart Figure 3-27, The User Interface module has the following internal state operation:

- UI_IDLE: Display default idle screen.
- UI_GET_RESPONSE: Display information about the update and the estimated time to update and give two choices: accept or reject this update. If rejected, display the IDLE screen. If accepted, the module transitions to the UI ACCEPT.
- UI_ACCEPT: Display the download progress, the module transitions to the UI_DOWNLOAD
- UI_DOWNLOAD: This progress will continuously update the value until it reaches 100%, then switch to UI_INSTALL state.
- UI_INSTALL: Display the install screen until the MCU is successfully updated, then switch the status to UI_DONE.
- UI_DONE: Display the done screen and complete the FOTA process.

When in the GET_RESPONSE state, OLED displays on the screen the estimated time for the update, the Gateway will calculate based on the size of the update, network speed, and the buffer time for transferring firmware from the Gateway to the MCU, according to the following Formula 3.2:

$$\text{Estimate time} = \frac{\text{Size of firmware}}{\text{Bandwidth average}} + \text{Buffer time} \quad (3.2)$$

With :

- Bandwidth average: 1000 (kB/s)
- Buffer time: 10 (s)

It also displays which MCU for this firmware by checking the node ID; we set up that if the node ID is equal to “1,” it is a Collision Avoidance System, and if it is equal to “2,” it is a Lighting System.

3.4.4. Bootloader

There is a hidden component that is crucial for the whole FOTA process, and that is the Bootloader. A bootloader that supports updating FOTA via a predefined protocol must be installed in the MCU before shipping the product, the definition and mechanism of the Bootloader will be discussed below. In this project, the predefined protocol is the CAN protocol.

3.4.4.1. Bootloader Requirement

Each Bootloader will have its own requirements based on the type of application. However, there are still a few requirements standard to all bootloaders. In this project, we have some requirements for our Bootloader:

- Ability to switch or select the operating mode (application or bootloader).
- Communication protocol requirements (CAN).
- Flash Programming requirements (Erase, Read, Write).
- Error detection mechanism (CRC).
- Security (Decrypt data firmware from Gateway)

3.4.4.2. Bootloader Flash Setup

Each microcontroller has some form of non-volatile memory used to store programs. The most commonly used type of memory is called flash. Flash is divided into divisible parts. The smallest part of the flash is often referred to as a page. Each processor differs in how these flash parts can be manipulated. Most will allow we to write a single byte (half-word, one word equal to 2 bytes) to flash at a time. It is crucial that we read the microcontroller datasheet and read through the memory and flash organization to know how it works.

Block	Name	Base addresses	Size (bytes)
Main memory	Page 0	0x0800 0000 - 0x0800 03FF	1 K
	Page 1	0x0800 0400 - 0x0800 07FF	1 K
	Page 2	0x0800 0800 - 0x0800 0BFF	1 K
	Page 3	0x0800 0C00 - 0x0800 0FFF	1 K
	Page 4	0x0800 1000 - 0x0800 13FF	1 K
	. . .	. . .	. . .
	Page 127	0x0801 FC00 - 0x0801 FFFF	1 K
Information block	System memory	0x1FFF F000 - 0x1FFF F7FF	2 K
	Option bytes	0x1FFF F800 - 0x1FFF F80F	16
Flash memory interface registers	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FLASH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4
	Reserved	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRPR	0x4002 2020 - 0x4002 2023	4

Figure 3-28 : Flash module organization of stm32f103c8t6 [23, Tab 5]

In Figure 3-28, Flash has 127 pages, each containing 1kB, equivalent to 128kB flash memory, Cortex-M3 only allows half-word (one-byte) flash and deletes at least one page so it can be flashed again. Attached are registers that control the write, read, and write to flash memory.

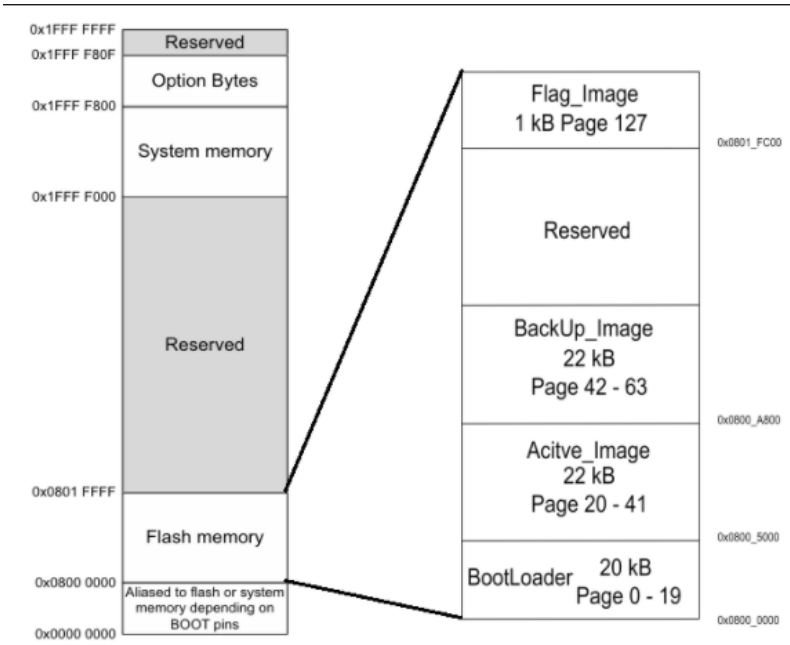


Figure 3-29 : Flash memory arrangement, adapted from [1]

We designed the Bootloader at the start of the flash memory, as shown in Figure

3-29, ensuring that the MCU always runs the first Bootloader at startup. The Bootloader will have 20kB of flash memory, it does not spend it all but leaves space for further improvements, like Active Image (Start at 0x0800_5000), Backup Image (Start at 0x0800_A800), which has 22kB. The last page of flash memory (Figure 3-30) contains flags that allow viewing the status of images for bootloader operation.

We use STM32 STLINK Utility, the software provided by ST to view the internal memory.

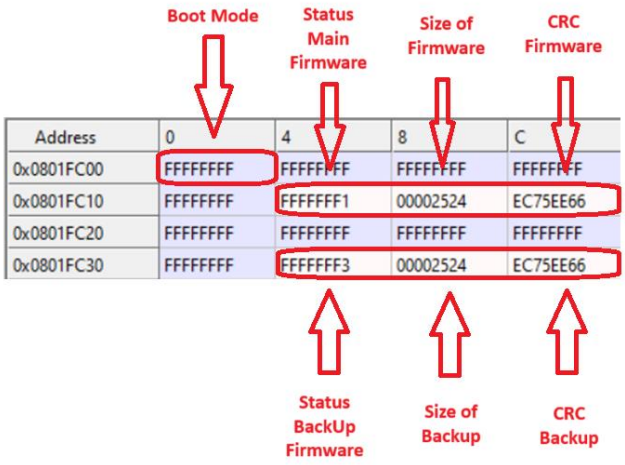


Figure 3-30 : Flag page design

At address 0x0801_FC00 :

- Offset 0, we define Flag_Branch, which decides whether the MCU should jump into bootloader mode (0x0000_0000) or application mode (0xFFFF_FFFF).
- Offset 14, we define Active_Image_Status, which represents the state of the Image (Active = 0xFFFF_FFF1, Backup = 0xFFFF_FFF3, Corrupt = 0xFFFF FFF4).
- Offset 18, we define Active_Image_SizeCode, which represents the size of firmware in the image.
- Offset 1C, we define Active_Image_CRC, which represents the CRC code of firmware in the image.

- It is the same with offset 34,38,3C but for the Backup Image.

3.4.4.3. Bootloader Behavior

The bootloader itself is no different from a standard application. The standard embedded application will perform a task repeatedly, and the more advanced one has an RTOS system to perform many different tasks, but in the end, it will perform some loop. What makes the bootloader special is that it contains programs that only run once when executed, share flash space with another application, and can delete and program a new application in its place.

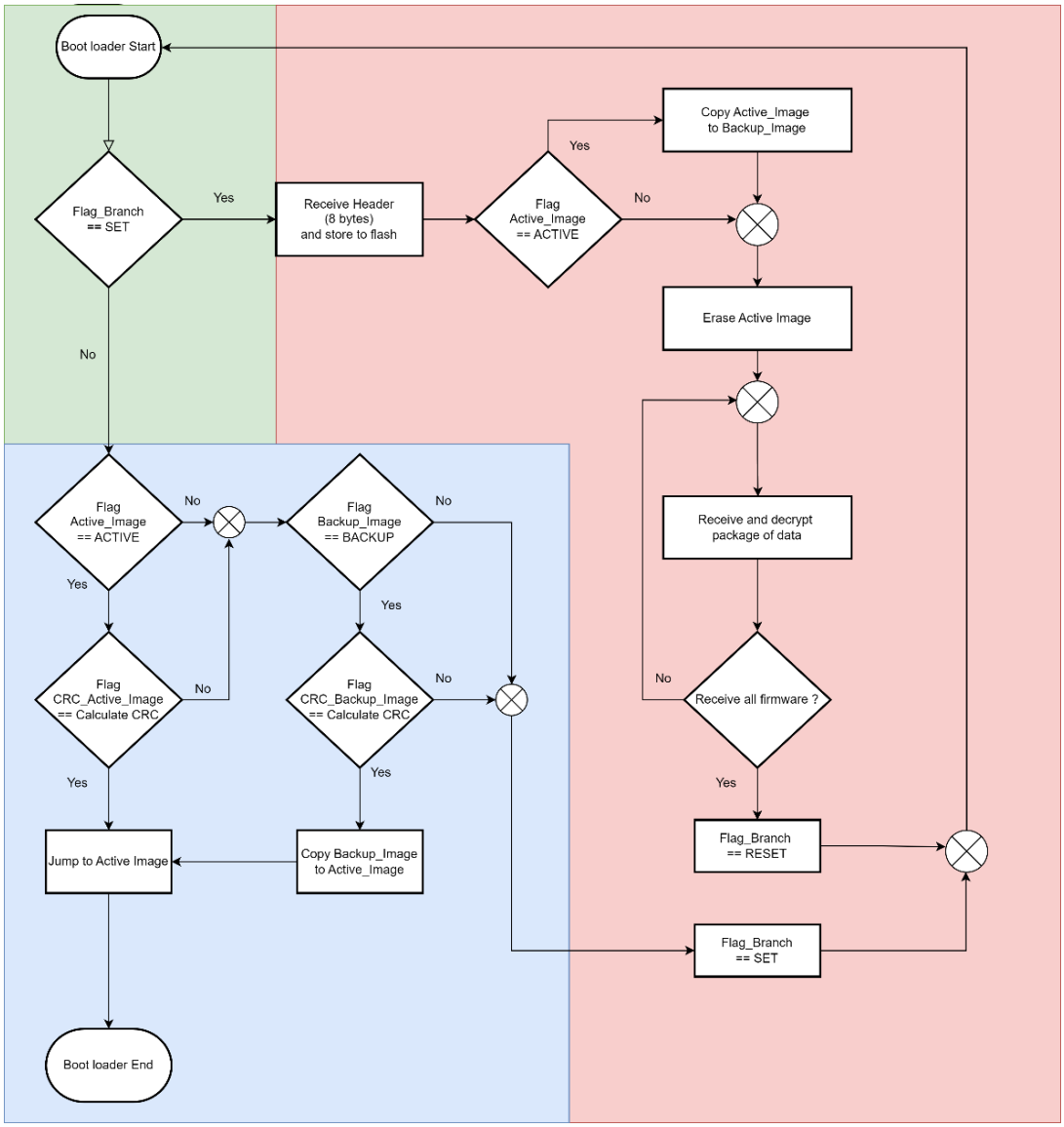


Figure 3-31 : Bootloader Flowchart

Figure 3-31 illustrate how the bootloader works in our project:

- Start the branch code (Green area) to determine which mode should be performed based on Flag_branch in flash memory, as shown in Figure 3-30.
- If Flag_branch is SET(0000_0000), it enters bootloader mode (Red area):
  - Move the active image to the backup image if it is available.
  - Receive the encrypted code packet using the package from the CAN protocol, decrypt it, and then store it in the active area in flash memory.
  - Reset the branching flag and reset the software.
- If Flag_branch is RESET(FFFF_FFFF), it enters an application mode (Blue area):
  - Calculate the CRC of the entire main firmware.
  - If the calculated CRC matches the original CRC, it will jump to the active area to be executed.
  - If the calculated CRC is incorrect, it will check the backup image and move the backup image to the active area to be executed.
  - If both images are corrupted, Set the branching flag, reset the MCU, and enter the bootloader.

3.4.4.4. Application Behavior

When Bootloader jump to Active Image and it works, that's great. But there will also come a time when the program has bugs, and we must update that firmware to be safe and provide a better experience for users. In the firmware application, there will always be a part of the bootloader that receives UDS signals from the Gateway while running the task of that MCU to put the MCU into the bootloader to conduct updates.

That is why in each firmware, we always program an interrupt function that allows signals from the Gateway to be received through the CAN protocol.

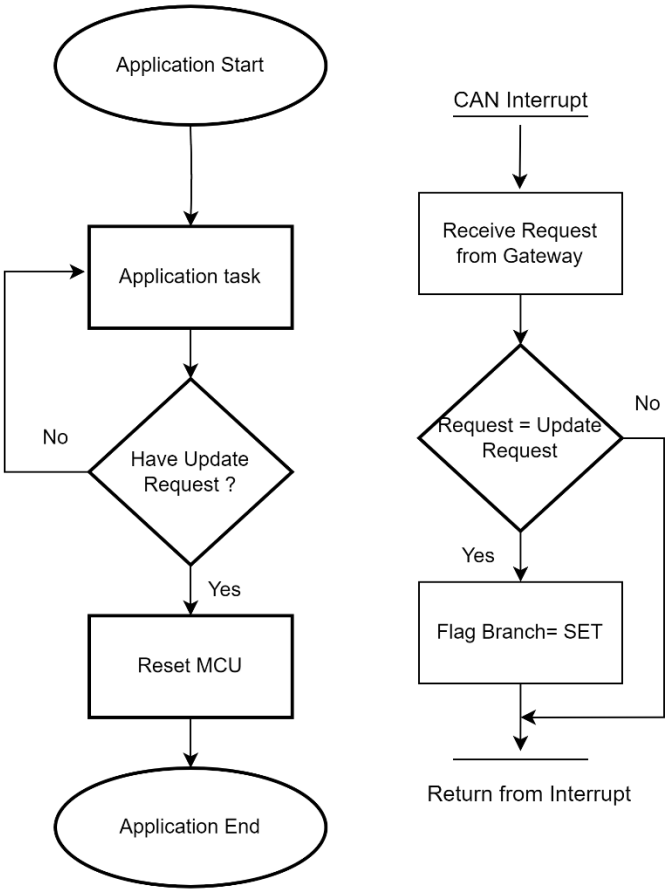


Figure 3-32 : Bootloader Application Flowchart

Figures 3-32 illustrate how the bootloader in the application works:

When running the application task, once it receives a Request Update via CAN bus, set Flag_Branch = SET, then reset MCU, and make it jump to boot mode.

3.4.5. Lighting System Software Design

The lighting system is one of FOTA's examples for this project. The basic lighting system controls the headlight, backlight, and brake light.

3.4.5.1. Lighting System Tasks

Lighting System has the following main tasks:

- Turn on/off the headlight and backlight by turning the switch on/off.
- Turn on the brake light when pressing the button.

3.4.5.2. Lighting System Flowchart

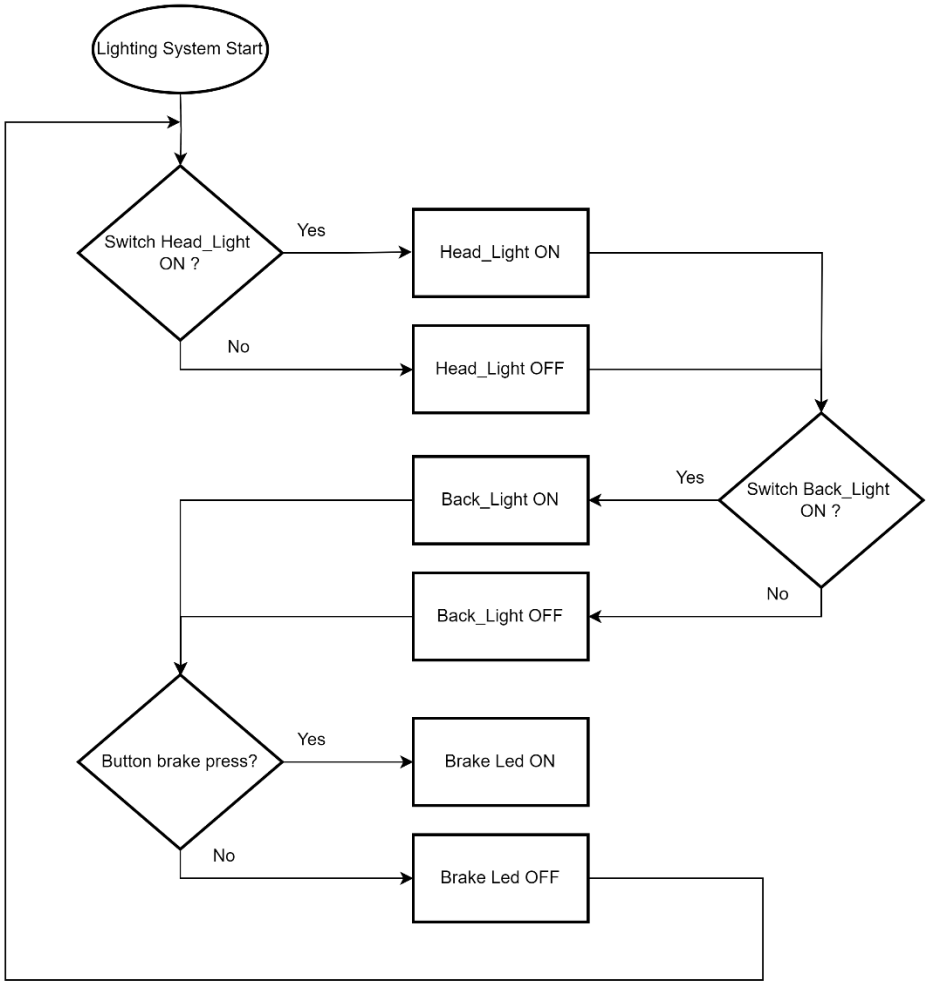


Figure 3-33 : Lighting System Flowchart

Based on the flowchart (Figure 3-33), the lighting system only needs to make a loop to see if any switch is turned on to open the front or back lights and detect whether the brake pedal is on or not to flash the brake lights.

3.4.6. Collision Avoidance System Software Design

The collision system is also one of the FOTA's examples for this project. A basic collision system alerts obstacles with a buzzer and blinking blinking LED.

3.4.6.1. Collision Avoidance System Tasks

The Collision System has the following main tasks:

- Use the HC_SR04 to detect the distance from the obstacle.

- If closer than 4cm, activate the buzzer and flash the LED.

3.4.6.2. Collision Avoidance System Flowchart

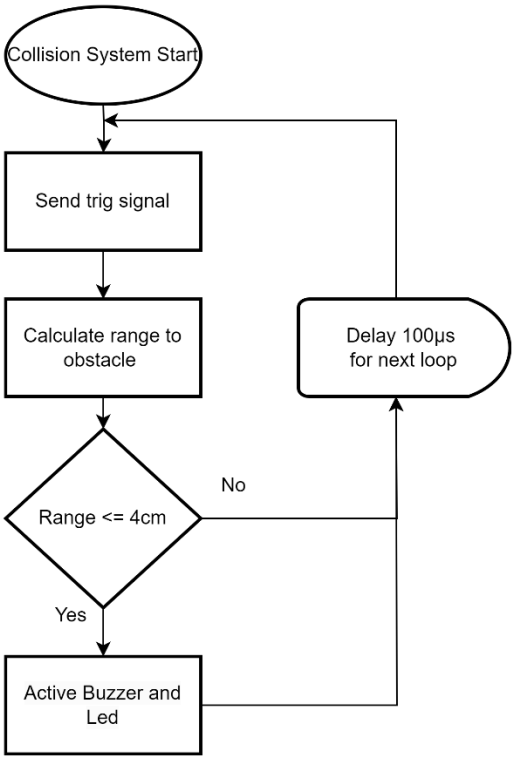


Figure 3-34 : Collision Avoidance System Flowchart

Based on the flowchart (Figure 3-34), we send a signal at a high level of at least 10us to trigger HC_SR04, send eight 40 kHz, and detect whether there is a pulse signal back. We can calculate the range through the time interval. The echo pin rises to high until it goes low.

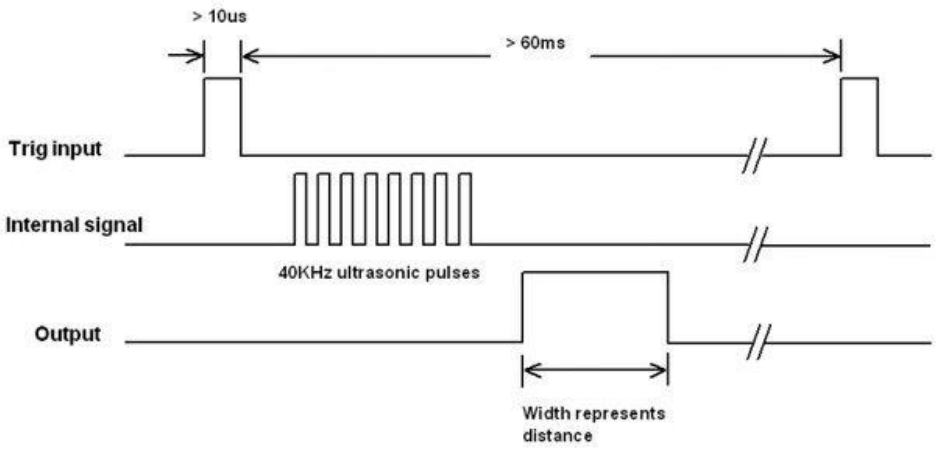


Figure 3-35 : Time Diagram of HC_SR04 [22, Fig 4]

Figure 3-35 simulating the operation of HC-SR04. The distance calculation will be based on the datasheet of the HC_SR04, precisely according to the following Formula 3.3:

$$\text{Distance} = \frac{\text{High level time} \times \text{Velocity of sound}}{2} \text{ (cm)} \quad (3.3)$$

With :

- High-level time: Time interval Echo pin High (μs)
- Velocity of sound: 0.034 (cm/ μs) = 340 (m/s)

CHAPTER 4: RESULTS AND EVALUATIONS

This chapter presents the results of the research topic and product. We also present the system's operation and performance through test cases and evaluation.

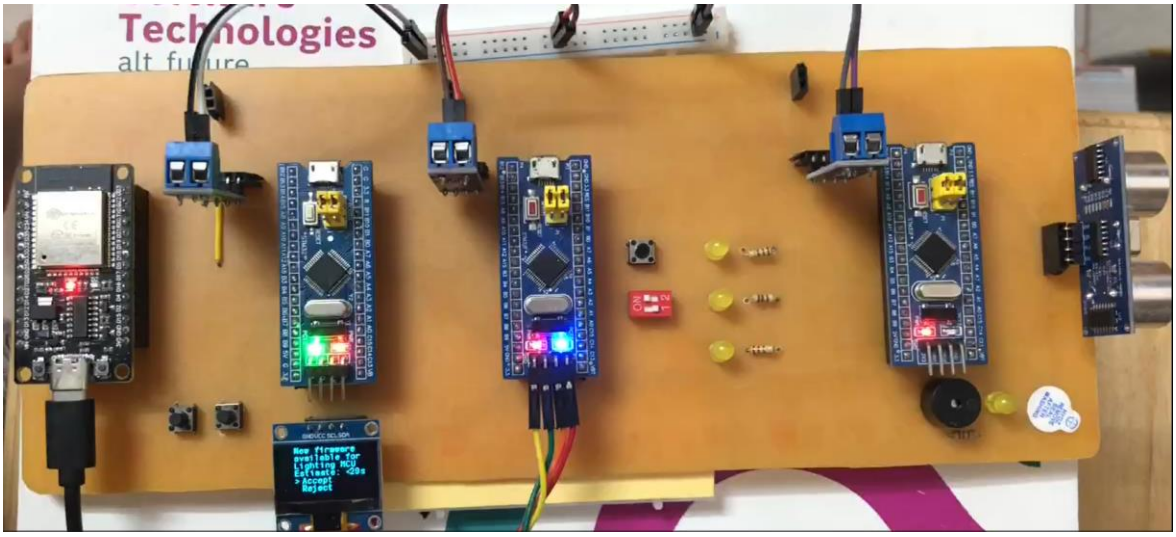


Figure 4-1 : Completed model FOTA System in vehicle

4.1. REQUIREMENTS OF RESULT

After the completion of the system, as Figure 4-1, the software and hardware of the FOTA system have been thoroughly checked to meet the following requirements based on the FOTA process flowchart in Chapter 3:

- The telecommunication unit must be able to detect if new firmware has been uploaded to Firebase.
- The gateway should display which MCU the firmware is for and how long the update will take.
- FOTA performance should be fast or close to the expected time displayed to the user.
- MCU functions should work properly without any issues.
- If an update error or firmware is invalid, the MCU must automatically roll back to the previous firmware.

4.2. GATEWAY UI SEQUENCE

Our system relies on OLED to communicate update notifications effectively and provide real-time FOTA progress updates.

While idle, the Gateway awaits notification from the Telecommunication Unit. The oled screen displays an idle state, as Figure 4-2.

Figure 4-2 : UI Idle State

When the Telecommunication Unit sends a UART interrupt to the gateway indicating the availability of a new update, the gateway will notify the user. The user can then accept or reject the update as Figure 4-3.

Figure 4-3 : UI NewUpdate State

When a software update is available, the user will be prompted to accept or reject it. If the user accepts the update, the updated code will be downloaded and sent to the gateway with the progress percentage displayed on OLED as Figure 4-4. However, if the user rejects the update, the gateway will not proceed with the installation and will remain idle.

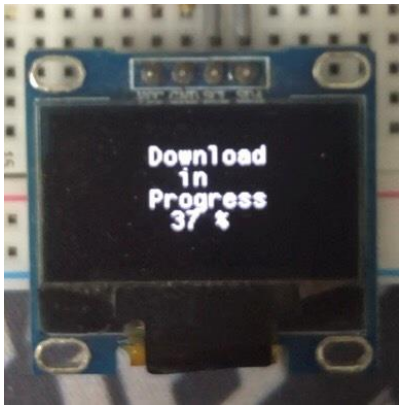


Figure 4-4 : UI Download State

After the download, the update will be encrypted and forwarded to the target ECU for installation, as shown in Figures 4-5.

Figure 4-5 : UI Install State

Once the installation process is complete, the gateway will notify the user that the update has been successfully installed, as shown in Figures 4-6. Upon receiving the notification, the gateway will return to its idle state.

Figure 4-6 : UI Finish State

4.3. COMMUNICATION SEQUENCE

Our MCUs use advanced protocols to ensure reliable transmission and reception of data packets without overloading transmitters and receivers as Figure 3-23. We use HyperTerminal and AduinoIDE to view UART transmissions and the Linux terminal to view transmissions on the CAN Bus.

4.3.1. UART

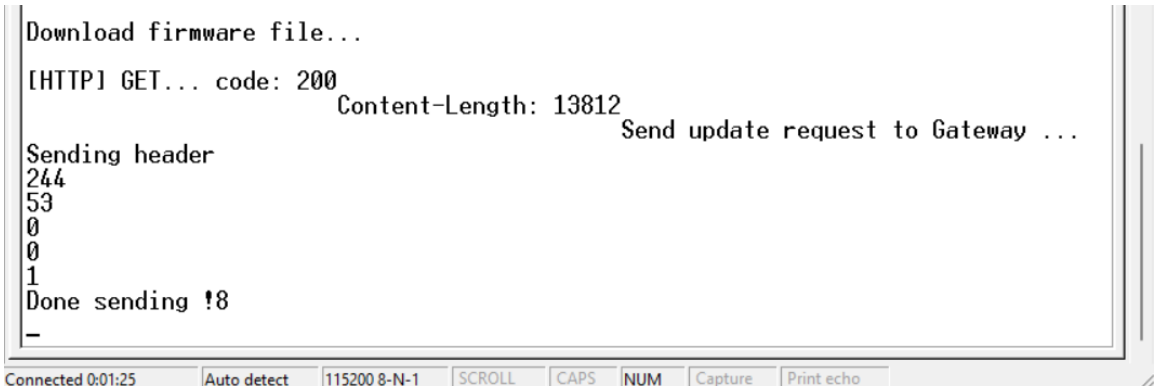


Figure 4-7 : UART Send Header

When the Telecommunications Unit detects new firmware, it sends a signal to the Gateway known as a "Header Send Request". This request serves as a confirmation that the firmware has been detected and is ready to be updated. Once the request has been received, the Telecommunications Unit initiates the transfer process by sending a 5-byte Header, as shown in Figure 4-7. This Header contains crucial information about the firmware, including the size of the firmware (indicated by the first 4 bytes) and the MCU ID (indicated by the last 1 byte). This information is essential for the Gateway to receive and process the firmware update accurately.

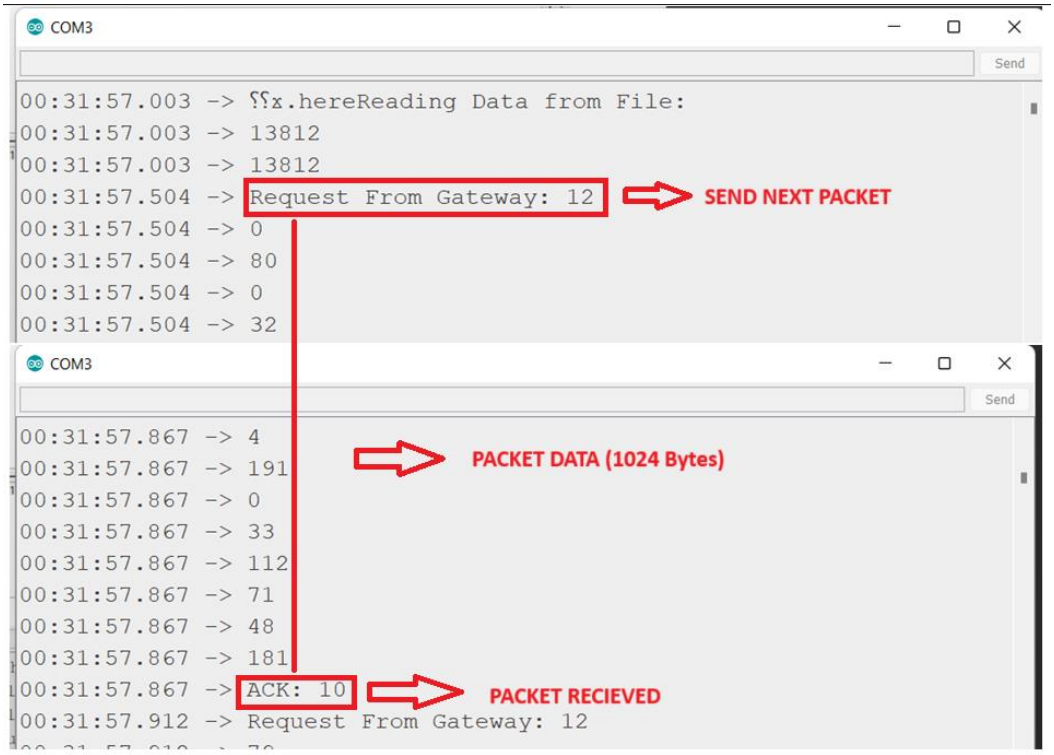


Figure 4-8 : UART Send Data Packet

Figures 4-8 and 4-9 illustrate the process of firmware file transfer from the Telecommunications Unit to the Gateway, which sends 1024-byte packets. After the Gateway signals its acceptance and readiness for the next packet, the Telecommunications Unit sends the next packet from the firmware file. As soon as the Gateway receives the packet, an ACK "Packet Received" signal is sent back to confirm that the packet has been processed. This process repeats until the Gateway receives the last packet.

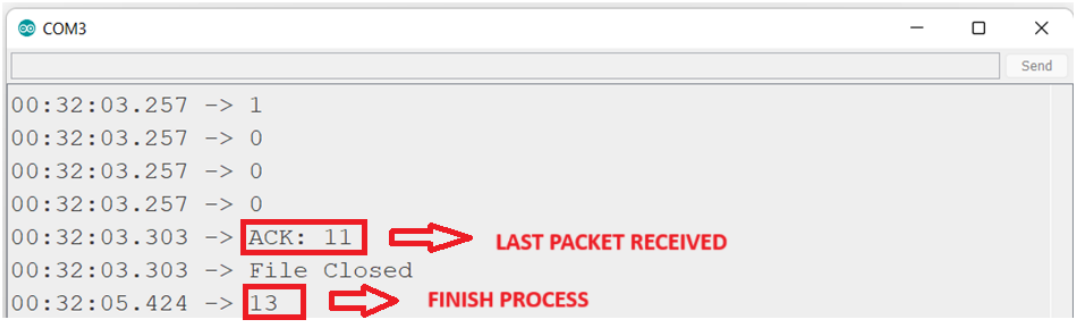


Figure 4-9 : UART Finish Transfer Process

When the Gateway receives the final packet of a firmware update, it must respond with the message "Last Packet Received" instead of "Packet Received". This helps to ensure that all packets have been received and that the update process can be completed successfully. Once the last packet has been processed, it is necessary to send an ACK "Finish" message to signal the end of the firmware receiving process. This will allow the device to transition to the next stage of the update process.

4.3.2. CAN

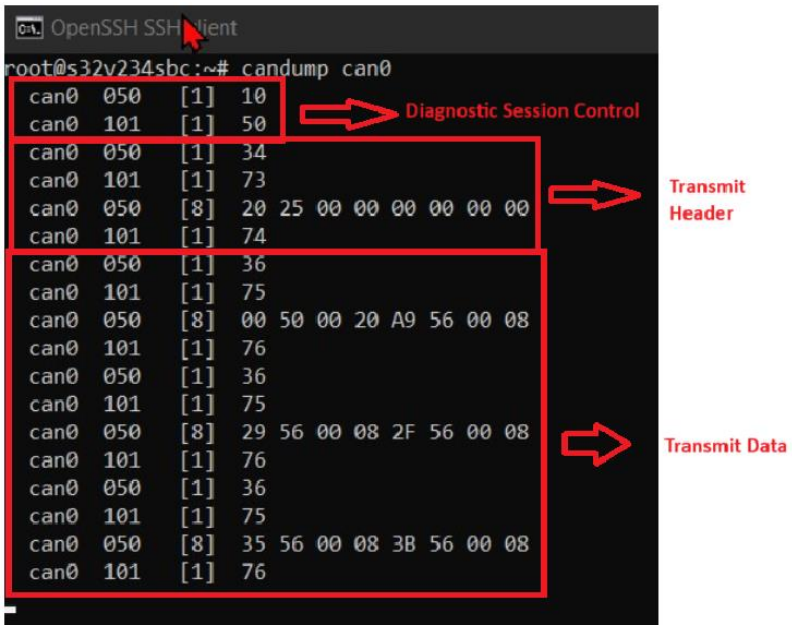


Figure 4-10 : CAN Transmit Data

Figures 4-10 and 4-11 illustrate the process of Gateway transferring data to MCU. After finishing the UART data transmission, the Gateway sends a CAN signal called "Diagnostic Session Control" coded 0x10 to the CAN bus. For each different MCU, the Gateway uses a unique CAN ID. For instance, with the Collision System MCU, the Gateway sends data with an ID of 0x50, while with the Lighting System, it is 0x60. This ensures that only the MCU we want to make FOTA responds, while the other MCU ignores irrelevant signals on the CAN bus. For example, the Gateway (ID 050) communicates with the Collision System MCU (ID 101). If ID is 102, it is Lighting System.

Once the MCU (0x50) sends an acceptance signal, the Gateway will transmit a "Request Download" signal with code 0x34. If the response is 0x73, a header containing the firmware size (4 bytes) and a CRC code (4 bytes) is sent to the MCU to check for errors.

When an ACK of 0x74 is received, data transmissions begin. The data is encoded according to the AES_CBC_128 standard and sent in 8-byte chunks. After each transmission, the Gateway sends a "Transfer Data" signal with code 0x36. When the MCU accepts it (0x75), the Gateway transmits 8 bytes of data and waits for an ACK 0x76. This process continues until the last 8 bytes of firmware are sent.

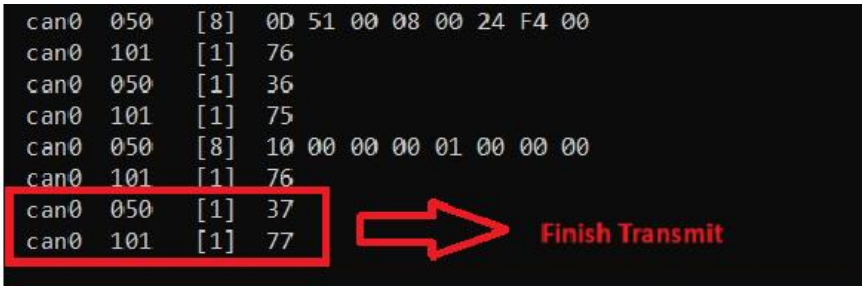


Figure 4-11 : CAN Finish Transmit

Once all firmware has been transferred to the MCU, the gateway will send a signal indicating the end of the transmission process, called "Request Transfer Exit" (0x37). The FOTA process will be considered complete once the MCU returns an ACK 0x77 as Figure 4-11. At this point, the MCU will restart itself and begin performing the update.

4.4. TEST CASE

Test case with Wi-Fi speed of 10 kBps, UART speed of 11,520 kbps, and CAN Bus speed of 500 kbps. We use STM32 STLINK Utility, the software provided by ST, to view the internal memory of the ST series of microcontrollers and allow direct memory modification to display STM32 Flash memory and to view the bootloader's behavior in the following cases.

We create the test case summary Table 4-1:

Table 4-1 : Test case summary table

Test Case Name	Test Case Type	Description	Expected Result
Lighting System test case	Functionality and Performance	Perform the FOTA process for the Lighting System	Lighting System work properly
Collision Avoidance System test case		Perform the FOTA process for Collision Avoidance System	Collision System work properly
Main Firmware was modified	Security and Safety	Modify the firmware to observe the bootloader behavior	Backup firmware will be used for MCU to work properly
FOTA process interruption		Disconnect the power or reset the MCU when perform FOTA process	
Main and backup firmware were broken		Both firmware are broken, observe the bootloader behavior	The MCU will stay in bootloader, ensuring only valid firmware works

4.4.1. Lighting System Test Case

The firmware of Lighting System has:

- Size: 0x2598 bytes (9,624 bytes in decimal)
- CRC code: 0x1671CADA (to check firmware validation)
- Estimate update time: <29s (Figure 4-12)
- Feature: Blink LED 1 when pressing the button and open LED 2 and 3 when opening the switch.

Figure 4-12 : Lighting System Estimate Update Time

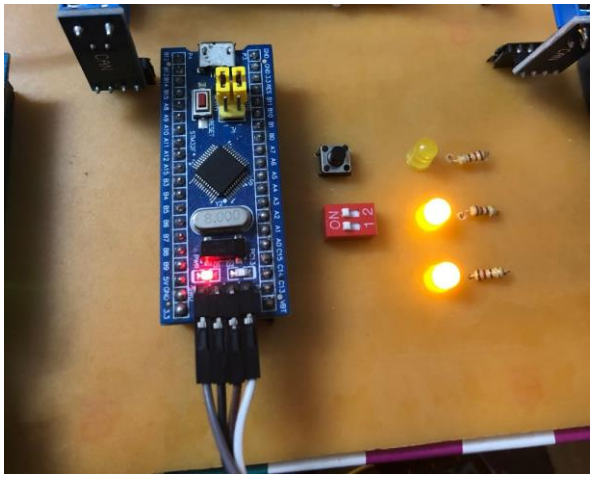


Figure 4-13 : Lighting System

Figure 4-13 shows the Lighting System after the FOTA process, it takes just 15 seconds for the installation to be completed. We calculate the estimated time in the worst case when Wi-Fi is low speed and the max delay when transferring firmware to MCU. Therefore, it takes less time than expected.

Device Memory @ 0x0801FC00 : Binary File

Target memory, Address range: [0x0801FC00 0x08022124]

Address	0	4	8	C
0x0801FC00	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x0801FC10	FFFFFFFF	FFFFFFF1	00002598	1671CADA
0x0801FC20	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF

Figure 4-14 : Active Flag in Lighting System

When accessing the Flash memory in the Flag area (Figure 4-14) to check the code size (Address 0x0801FC10, offset 8) and CRC values (Address 0x0801FC10, offset C), it matches with the calculated values. The memory area 0x0801_FC14 is the current operating state of this firmware. The firmware with the value

0xFFFF_FFF1 is the one that works properly.

4.4.2. Collision Avoidance System Test Case

The firmware of Collision Avoidance System has:

- Size: 0x35f4 bytes (13,812 bytes in decimal)
- CRC code: 0xE326C2AB
- Estimate update time: <33s (Figure 4-15)
- Feature: If the MCU detects a distance from the obstacle < 4cm, open the LED and active buzzer.

Figure 4-15 : Collision Avoidance System Estimate Update Time

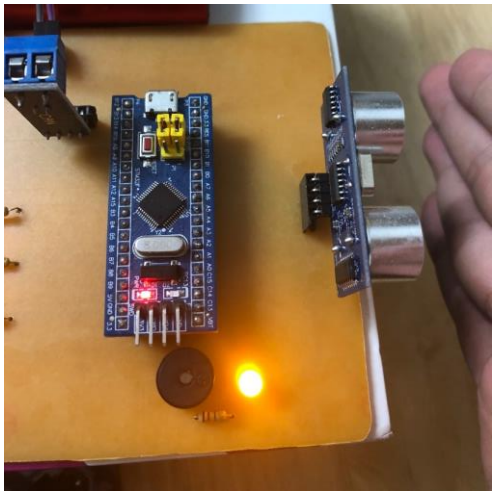


Figure 4-16 : Collision Avoidance System

Figure 4-16 shows the Collision System after the FOTA process. Once we have initiated the FOTA process and have selected the new firmware to be installed on the MCU, it takes only 20 seconds for the installation to be completed after pressing

the "Accept" button. This is a relatively shorter time than the expected duration of 33 seconds for the FOTA process. The efficient installation process ensures that the new firmware is installed quickly and without any interruptions, minimizing downtime and ensuring that the MCU is up and running with the latest firmware.

Device Memory @ 0x0801FC00 : Binary File

Target memory, Address range: [0x0801FC00 0x08022124]

Address	0	4	8	C
0x0801FC00	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x0801FC10	FFFFFFFF	FFFFFFF1	000035F4	E326C2AB
0x0801FC20	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF

Figure 4-17 : Active Flag in Collision Avoidance System

When accessing the Flash memory in the Flag area (Figure 4-17) to check the code size and CRC values, it matches with the calculated values. The memory area 0x0801_FC14 is the current operating state of this firmware. The firmware with the value 0xFFFF_FFF1 is the one that works properly.

4.4.3. Update Fail Handle Test Case

This test case is divided into three cases: Firmware invalid, FOTA process interruption, both main and backup firmware are corrupted.

4.4.3.1. Firmware Invalid

Target memory, Address range: [0x08005000 0x08007524]

Address	0	4	8	C
0x08005000	20005000	080061DD	0800615D	08006163
0x08005010	08006169	0800616F	08006175	00000000
0x08005020	00000000	00000000	00000000	0800617B
0x08005030	08006187	00000000	08006193	0800619F

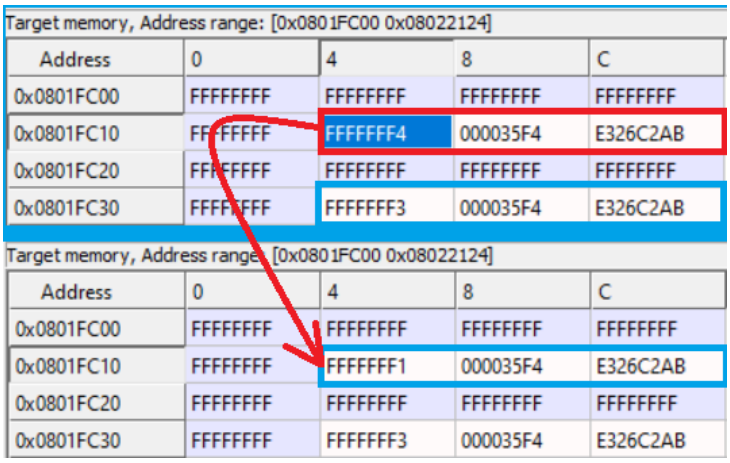
Device Memory @ 0x08005000 : Binary File

Target memory, Address range: [0x08005000 0x08007524]

Address	0	4	8	C
0x08005000	20005432	080061DD	0800615D	08006163
0x08005010	08006169	0800616F	08006175	00000000
0x08005020	00000000	00000000	00000000	0800617B
0x08005030	08006187	00000000	08006193	0800619F

Figure 4-18 :The Manipulated Firmware

In this case, we will change some bytes in the firmware to see what the bootloader does when it checks the validity of the firmware. In Figure 4-18, we change 2 bytes at address 0x0800_5000 from 5000 to 5432.



Target memory, Address range: [0x0801FC00 0x08022124]

Address	0	4	8	C
0x0801FC00	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x0801FC10	FFFFFFFF	FFFFFFF4	000035F4	E326C2AB
0x0801FC20	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x0801FC30	FFFFFFFF	FFFFFFF3	000035F4	E326C2AB

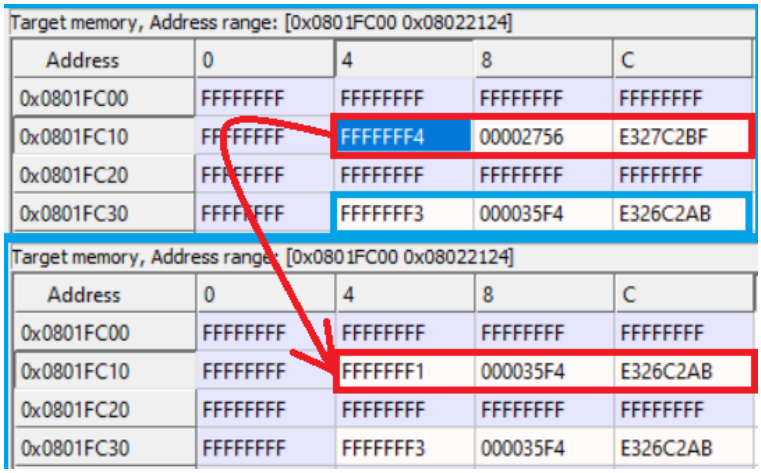
Target memory, Address range: [0x0801FC00 0x08022124]

Address	0	4	8	C
0x0801FC00	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x0801FC10	FFFFFFFF	FFFFFFF1	000035F4	E326C2AB
0x0801FC20	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x0801FC30	FFFFFFFF	FFFFFFF3	000035F4	E326C2AB

Figure 4-19 : Flag region in case MCU's firmware was manipulated

In the Flag area (Figure 4-19), after the bootloader calculates the CRC code of the entire firmware and compares it with the correct CRC of the firmware in the Active area (Start at 0x0801_FC10), it detects that the firmware has changed, so it does not boot that firmware and changes the flag to (0xFFFF_FFF4), it will check the Backup area (Start at 0x0801_FC30). If it checks that the Backup is working, copy the Backup firmware to the Active area and run normally.

4.4.3.2. FOTA process interruption



Target memory, Address range: [0x0801FC00 0x08022124]

Address	0	4	8	C
0x0801FC00	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x0801FC10	FFFFFFFF	FFFFFFF4	00002756	E327C2BF
0x0801FC20	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x0801FC30	FFFFFFFF	FFFFFFF3	000035F4	E326C2AB

Target memory, Address range: [0x0801FC00 0x08022124]

Address	0	4	8	C
0x0801FC00	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x0801FC10	FFFFFFFF	FFFFFFF1	000035F4	E326C2AB
0x0801FC20	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x0801FC30	FFFFFFFF	FFFFFFF3	000035F4	E326C2AB

Figure 4-20 : Flag region in case FOTA process interruption

In case the FOTA process is interrupted due to power failure, interference, or physical impact on the circuit. In Figure 4-20, the MCU being updated has a firmware of size 0x2756 bytes and a CRC code is 0xE326C2AB. But we pressed the reset button on the board and the FOTA process was interrupted while receiving firmware data. When running the bootloader, it will be detected that the Active area is corrupted because the firmware has not been completely transferred. It immediately checks the Backup area and copies back to the Active area when the Backup firmware is valid.

4.4.3.3. Main Firmware and Backup Firmware are corrupted

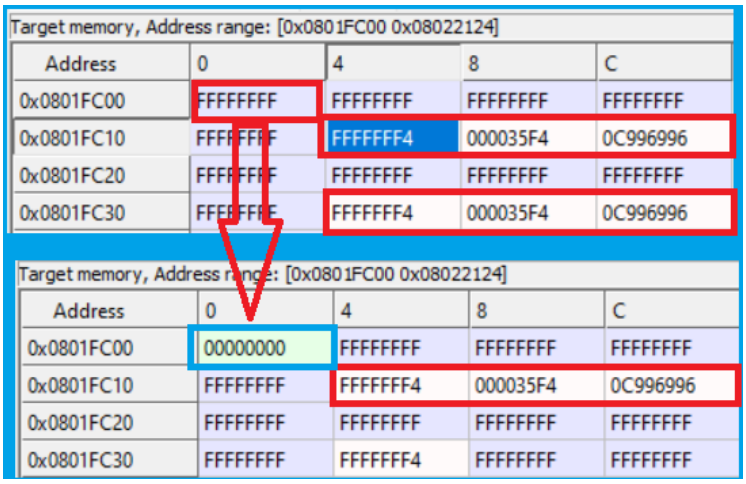


Figure 4-21 : Flag region in case both firmware are invalid

In the worst-case scenario, both the Active and Backup areas are corrupted. In Figure 4-21, when the bootloader calculates the CRC and detects that both firmware are invalid, it does not boot any firmware that the MCU runs. Instead, it stays in the bootloader by passing the Bootloader flag to 0x0000_0000 (at 0x0801FC00 address). The MCU will remain in the bootloader until the firmware is completely updated, ensuring safe booting for the MCU. However, in this project, the MCU will stay in the bootloader indefinitely if we do not fix it manually. The bootloader feature to fix this problem is not currently developed but will be considered in future improvements.

4.5. TEST CASE EVALUATIONS

After successfully implementing the FOTA process for 2 MCUs, Lighting System and Collision System, with the same Wi-Fi speed, UART transfer speed, speed on CAN bus.

We have made a comparison and evaluation of Table 4-2:

Table 4-2 : FOTA Process Evaluation Table

	Lighting System	Collision System
Code size	9,624 bytes	13,812 bytes
Code CRC	0x1671CADA	0xE326C2AB
Estimate Time	29s	33s
Flashing Time 1	16s	21s
Flashing Time 2	15s	20s
Flashing Time 2	15s	20s
Average Flashing Time	~15s	~20s

We perform the FOTA procedure three times for both MCUs. The updated speed results show that with the MCU Lighting, it is 15s and with the MCU Collision Avoidance it is 21s. Thereby, we found that the speed of the FOTA process depends on the size of the firmware. If the firmware is larger, the FOTA process will be slower due to the need to download more data. In addition, the speed of Wi-Fi, UART, and CAN also affects the speed of the FOTA process. Therefore, it is essential to consider the speed of these components when designing the FOTA process to ensure optimal performance.

CHAPTER 5: CONCLUSION

This chapter summarizes the main findings, concludes, and suggests directions for future improvement.

5.1 CONCLUSION

The research thesis achieved the set goals, we implemented the FOTA model for two test case MCUs, proving the system's feasibility. As we said, FOTA technology is important for developing and integrating embedded system solutions. We gained much knowledge through the process of working on the dissertation process and achieved several achievements:

- Read the user manual and datasheet of MCU.
- The bootloader design supports the CAN protocol, which supports rollback if an error occurs.
- Learn about AUTOSAR and implement the RTE layer into the Gateway.
- Implement security between the Gateway and the MCU using the AES_CBC_128 algorithm.
- Connect to Firebase via Wifi.
- Use Git to manage the version of all source code.

However, due to time constraints, the system may be designed to be unstable, not optimal, and still has many limitations:

- The connection to Firebase is unstable.
- Because the OLED screen is too small and cannot display information about firmware changes updated with the current firmware.
- There is no security method between the Firebase and the Telecommunication Unit.
- It is not possible to detect multiple firmware for multiple MCUs at the same time.
- Users cannot actively roll back the MCU to the previous version because no

database contains firmware versions.

- UART is not anti-interference.
- No measures have been taken against channel interference.
- PCB circuits have not been aesthetically pleasing.

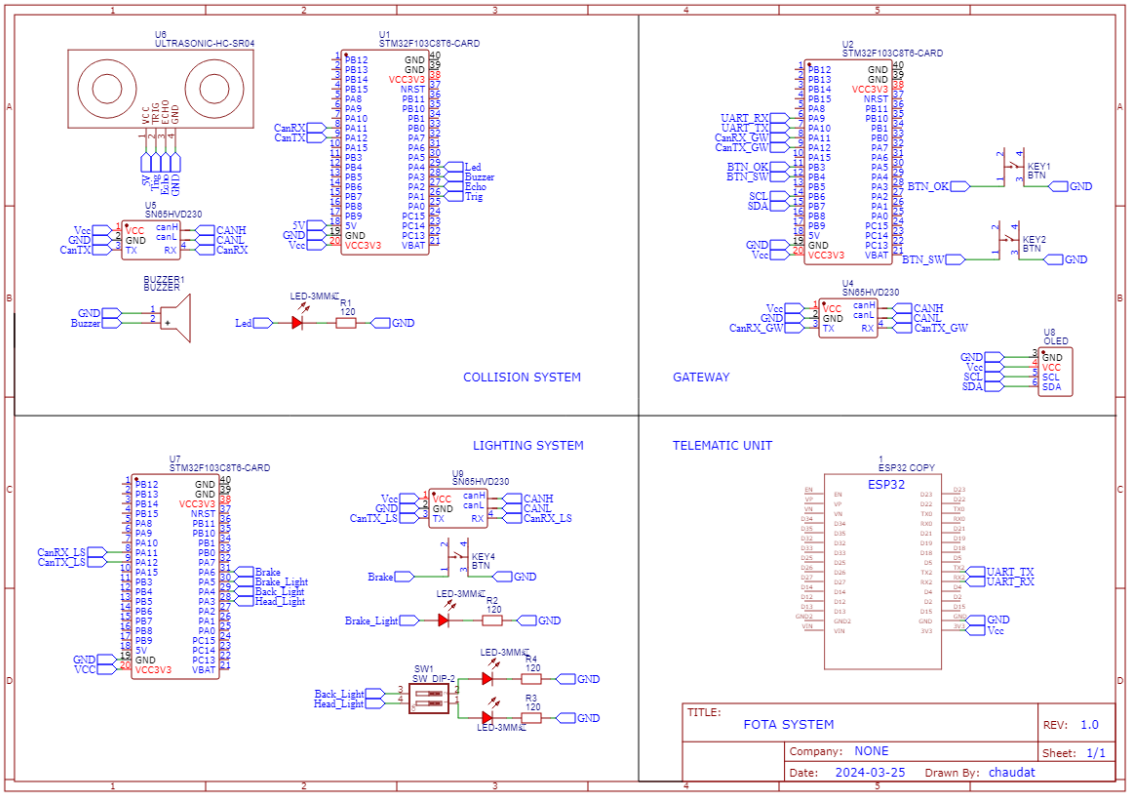
5.2 PROJECT IMPROVEMENT

After completing the financial goals and achieving the targeted goals, the project sets out future directions and development steps for the system to reach a complete system in all aspects of practice:

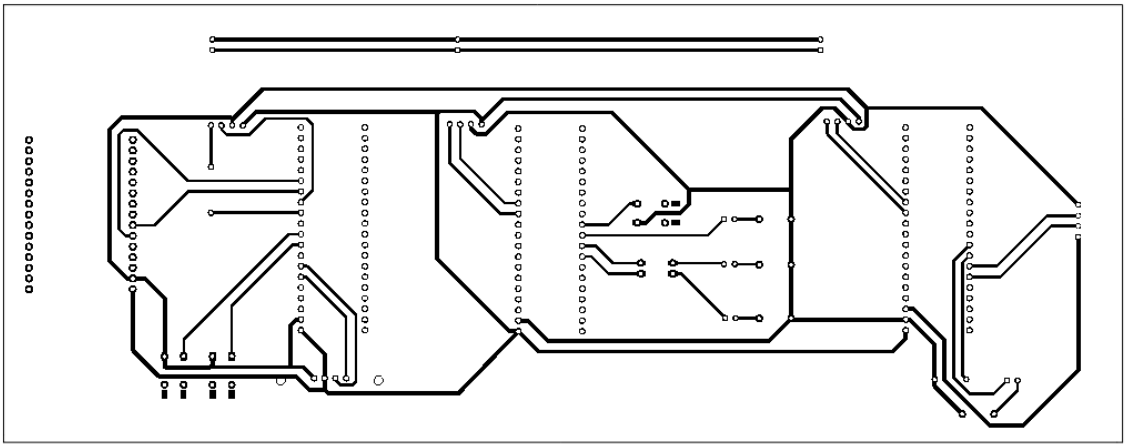
- Develop additional firmware version control functionality on the Gateway so users know the current firmware version, allowing owners to choose which updates to install.
- Shows the difference between the current version and the version to be updated.
- Allows users to restore to an earlier version of any MCU when necessary.
- Implement firmware security when posted to the database, using more robust encryption and authentication methods to protect vehicle data.
- Integrating the FOTA system with other systems is considered limited such as navigation systems and entertainment systems.
- Provides new, limited services to vehicle owners such as remote vehicle retrieval and technical support.
- The optimization feature is used when updating, improving update speed, and reducing estimation time errors.
- Sensors need to improve accuracy.
- Create an application for the user's mobile device to notify the user about updates, allowing remote updates through the app.

APPENDIX

1. Schematic FOTA System



2. PCB FOTA System



3. Source Code:

https://github.com/DatChauThanh/FoTA.git

REFERENCES

[1] STMicroelectronics, "Datasheet stm32f103x," DS5319 Datasheet, July 2007 [Revised Sept. 2023].

[2] Espressif Systems "ESP32 Series Datasheet," ESP32 Datasheet, Feb 2007 [Revised Sept. 2023].

[3] A. M. Abdelhady, M. H. Mohamed. A. M. Abdelhady, "FOTA Project Handbook," Thesis, Mansoura University, Egypt, 2021.

[4] NXP editor, "Firmware Over-the-Air (FOTA)," [Online]. Available: https://www.nxp.com [Accessed 10 March 2024].

[5] Vaibhav, "Understanding FOTA in the Times of 'Connected Cars'," embitel.com, 10 July 2018. [Online].
Available: https://www.embitel.com [Accessed 29 March 2024].

[6] N. Carmine, "Interrupts Management" and "Booting Process," in *Mastering STM32, Second Edition*, Italy, leanpub, 2022, pp. 144-147, 558-562.

[7] J. Beningo, "Bootloader Design for MCUs in Embedded Systems," 26 June 2015. [Online]. Available: https://www.beningo.com. [Accessed 01 April 2024].

[8] Wikipedia, "Unified Diagnostic Services," [Online]. Available: https://en.wikipedia.org/wiki/Unified_Diagnostic_Services [Accessed 1 April 2024].

[9] AUTOSAR, "AUTOSAR Introduction," [Online]. Available: https://www.autosar.org. [Accessed 5 April 2024].

[10] Wikipedia, "CAN Bus," [Online]. Available: https://en.wikipedia.org/wiki/CAN_bus [Accessed 5 April 2024].

[11] S. Afzal, "I2C Primer: What is I2C?," Analog Devices, 2 Sep 2016. [Online]. Available: https://www.analog.com [Accessed 6 April 2024].

[12] Rohde-Schwarz, "Understanding UART," [Online]. Available: www.rohde-schwarz.com [Accessed 6 April 2024].

[13] GeeksForGeeks, "Firebase – Introduction," 15 July 2021. [Online]. Available: https://www.geeksforgeeks.org/firebase-introduction/ [Accessed 6 April 2024].

[14] Wikipedia, "Advanced Encryption Standard," [Online]. Available: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard. [Accessed 18 April 2024].

[15] HEAVY AI Editor, "Vehicle Telematics Definition," HEAVY AI, [Online]. Available: https://www.heavy.ai. [Accessed 7 April 2024].

[16] Solomon Systech, "SSD1306 Advance Information," SSD1306 Datasheet, March 2001 [Revised Aug. 2010]

[17] Texas Instruments, "3.3v CAN Transceivers," SLOS346G Datasheet 2001, Oct 2007 [Revised June. 2002]

[18] A J Bhosale, Class Lecture, Topic: "Automotive Accessories & Lighting Systems", Unit3, Dept. of Automobile Engineering, Government College of Engineering and Research, Maharastra, India [Online]. Available: https://www.gcoeara.ac.in. [Accessed 9 April 2024].

[19] Wikipedia , "DIP switch", 14 November 2023 . [Online]. Available: https://en.wikipedia.org/wiki/DIP_switch. [Accessed 10 April 2024].

[20] Components101, "5mm Round LED," 29 July 2018 . [Online]. Available: https://components101.com/diodes/5mm-round-led. [Accessed 10 April 2024].

[21] Wikipedia, "Collision avoidance system," [Online]. Available: https://en.wikipedia.org/wiki/Collision_avoidance_system [Accessed 10 April 2024].

[22] Microcontrollerslab, "HC-SR04 Ultrasonic Sensor with STM32 Blue Pill", [Online]. Available: https://microcontrollerslab.com. [Accessed 11 April 2024].

[23] STMicroelectronics, "Reference manual," RM0008 Manual, Sep 2011 [Revised Feb. 2024].

[24] Researchgate editors, “The AUTOSAR layered architecture”, [Online] Available: The AUTOSAR layered architecture,[Accessed 11 April 2024]

[25] Wikipedia editors, “Symmetric-key algorithm”, Wikipedia.com [Online] . Available: Symmetric-key algorithm, [Accessed 15 April 2024]

[26] Wikipedia editors, “Block cipher mode of operation”, Wikipedia.com [Online]. Available: Block cipher mode of operation, [Accessed 15 April 2024]

[27] Xilinx editors, “AES Decryption Algorithms”, xilinx.github.io, [Online] .Available: AES Decryption Algorithms, [Accessed 15 April 2024]

[28] “STM32F411 Black Pill Development Board”, [Online]. Available: STM32F411 Black Pill, [Accessed 15 April 2024]

[29] Smishad Thomas, “Automotive ECU: An Inevitable Part of the Automobiles,” January 30, 2020 [Online], Available: https://www.einfochips.com/blog/automotive-ecu-an-inevitable-part-of-the-automobiles/ [Accessed: 7 June 2024].