



جامعة المنوفية
كلية الهندسة الإلكترونية
قسم هندسة وعلوم الحاسبات



Secure FOTA for Automotive

By:

- *Mohamed Hussien Arafa Eldeen*
- *Mostafa Ahmed Nassar*
- *Mohamed El-Sayed Abd El-Moneam*
- *Elhosseni Gamal Nour*
- *Yahya Mohamed Dorgham*
- *Mahmoud Samy Badawy*
- *Mohamed Afify Ragab*

Supervised By:

Dr. Salah Shaban

Dr. Mohamed Badawy

2023

Head of the Department

Dean

Abstract

Recently, the world around us has become linked in one way or another to embedded systems and IoT, as we are rapidly moving towards everything new in the world of technology, which aims at development in all fields, to make it easier for the user to use the devices that he may be exposed to in his life daily.

According to what is known, the greater the characteristics, features, and requirements, the greater the complexity of the system, it was expected that the matter would be very complicated, but this is not in the presence of the Embedded System, especially the Embedded Arm, which was used in our project.

Since the development was very rapid, it was necessary to have a system that could keep pace with these developments, which would make the available devices adaptable and evolving.

Here comes the role of firmware over the Air (FOTA), which is one of its most important features that it connects all devices through the cloud to the mother company to always be connected to them to be able to provide them with the update of their devices permanently using OTA, so that their devices keep pace with the development of technology. Also, FOTA can help users keep their devices free from software defects and alert them to hardware defects and potential dangers to their device components.

Those are the features that made maintenance and updating the software easy, simple, and cost less after it used to take a long time and huge costs.

The automotive field is one of the areas most interested in FOTA, so our team was interested in participating in this field and contributing to the development cycle by adding some security features as well as facilitating the use of these features for the user.

Table of Contents

Abstract	2
Table of Contents	3
List of Figures	8
List of Abbreviations	11
Chapter 1: Introduction	13
1.1. Introduction to FOTA	13
1.1.1. What is FOTA?	13
1.1.2. Why do we need FOTA?	13
1.1.3. Examples of FOTA	14
1.2. Flashing Techniques for Microcontrollers	15
1.2.1. Off-Circuit Programming	15
1.2.2. In-Circuit Programming	16
1.2.3. In-Circuit Programming with Bootloader	17
1.3. Software Development Lifecycle	17
1.3.1. What is Software Development Lifecycle?	17
1.3.2. How the Software Development Life Cycle Works	18
1.3.3. The Seven Phases of the SDLC	18
Chapter 2: Problem Definition	22
2.1. Problem Definition	22
2.2. Implementing FOTA	23

2.3. Major Challenges	23
2.4. FOTA Benefits	24
2.6. Market Research	25
Chapter 3: System Design	29
3.1. System Diagram	29
3.1.1. Abstract View	29
3.1.2. Implemented View	30
3.2. System Components	31
3.2.1. Main ECU	31
3.2.2. App ECU	36
3.2.3. User ECU	40
3.2.4. Server	41
3.3. Software Design	42
3.3.1. Software Design Process	42
3.3.2. Design Considerations:	43
3.3.3. Software Design Approaches	44
3.3.4. Software Design Types	45
Chapter 4: Implementation	49
4.1. Server Connection	49
4.1.1. Firebase	49
4.1.2 ESP with Firebase:	50
4.1.3. Website	50
4.1.3.1 Server-Side Components	50
4.1.3.2 Client-Side Technologies:	51
4.1.3.3 Interaction Between Components:	51
4.1.3.4 Features:	52
4.1.3.5 Layout and Navigation	53
4.1.3.6 Key UI Components	54
4.1.3.7 Database Integration	55
4.1.3.8 Deployment Process	56
4.1.3.9 Continuous Deployment	57

4.1.3.10 Testing	57
4.1.3.10 User Documentation	59
4.1.3.12 Troubleshooting	61
4.1.3.13 Implementation Details	62
4.1.3.14 Architectural Overview	64
4.2. Advanced Encryption Standard (AES)	66
4.2.1. Introduction	66
4.2.2. Encryption Algorithms	67
4.2.3 Algorithm Characteristics:	67
4.2.4. Encryption process	68
4.2.5. Decryption process	70
4.6.2 Code flow chart	72
4.3. CAN Network	75
4.3.1. Introduction	75
4.3.2. What is CAN?	75
4.3.3. Features of CAN protocol	76
4.3.4. Can Network Message Format	77
4.3.5. Message frame	78
4.3.6. CAN Protocol Working Principle	80
4.3.7. Operation of the CAN Network	81
4.3.8. Interface between Nodes and Communication Network	81
4.3.9. CAN in Action	82

4.4. USART protocol	82
4.4.1. Introduction	82
4.4.2. Interface	83
4.4.3. Data Transmission	84
4.4.4. USART Transmission	86
4.5. Implemented Communication Protocol	88
4.5.1. Parallel Communication Protocols	88
4.5.2. Asynchronous Octal Communication (AOC)	89
4.6. Bootloader	92
4.6.1. Introduction	92
4.6.2. Memory Architecture	93
4.6.3. Bootloader Design	94
4.6.4 Bootloader Structure	95
4.7. Real Time Operating System (RTOS)	98
4.7.1. Introduction	98
4.7.2. Free RTOS Overview	99
4.7.3. Free RTOS Implementation	100
4.7.4. RTOS Implementation	100
4.8. Diagnostics Handling	101
4.8.1. Introduction	101
4.8.2. Diagnostics in our system	102
4.8.3. Diagnostics for different application ECUs	103
4.8.4. Future Developments in Vehicle Diagnostics	103
4.9. User Interface	104
4.9.1. The dashboard	104
4.10. Hardware Implementation	106
4.10.1. Altium PCB Designer	106
4.10.2. Implemented PCB	107
Chapter 5: Conclusion	109
5.1. Achievements	109
5.2. Future Improvements	110

5.2.1. Adaptive AUTOSAR for diagnostics	110
5.2.2. Delta file	110
Tools	110
References	111
ملخص المشروع	114

List of Figures

Figure 1.1.11 FOTA/OTA update for Remote Device ManagementFOTA/OTA update for Remote Device Management.....	15
Figure 2 ST-Link Programmer.....	17
Figure 3 The Seven Phases of the SDLC	18
Figure 4 Global Automotive OTA Updates Mark	26
Figure 5 Market Size	27
Figure 6 Impact on Automotive Updates Market.....	28
Figure 7 System Design.....	29
Figure 8 Implemented View System.....	30
Figure 9 ESP32 IC.....	31
Figure 10 STM32 IC	33
Figure 11 HC- SR04 IC.....	36
Figure 12 DC Motor with Encoder.....	37
Figure 13 LM35 Temperature Sensor	37
Figure 14 Nextion HMI Display	38
Figure 15 Asynchronous Octal Communication Pins	40
Figure 16 Detailed View of System.....	41
Figure 17 Software Development Cycle	42
Figure 18 Top-down Approach vs Bottom-up Approach	45
Figure 19 Layered Architecture	47
Figure 20 Layered Architecture	48
Figure 21 Firebase BaaS	49
Figure 22 ESP with Firebase	50
Figure 23 Encryption Life Cycle.....	67
Figure 24 Encryption process	68
Figure 25 SubByte	69
Figure 26 MixColumns	70
Figure 27 Main Function.....	72
Figure 28 Inverse Cipher Function.....	73
Figure 29 Cipher Function.....	73
Figure 30 AES_CBC_encrypt_buffer Function.....	73
Figure 31 AES_CBC_decrypted_buffer Function.....	73
Figure 32 XorWithIv Function.....	74
Figure 33 AES_init_ctx_iv Function.....	74

Figure 34 decryptFile Function	74
Figure 35 encryptFile Function	74
Figure 36 CAN networks significantly reduce wiring.	75
Figure 37 The standard CAN frame format	77
Figure 38 The c CAN frame format.	78
Figure 39 CAN Data frames	79
Figure 40 CAN Error Frames	79
Figure 41 CAN Overload frame.	80
Figure 42 Two USARTs directly communicate with each other.	83
Figure 43 USART with data bus	83
Figure 44 USART packet	85
Figure 45 Start bit.	85
Figure 46 Data Frame	85
Figure 47 Parity bits	86
Figure 48 Stop bits.	86
Figure 49 Data bus to the transmitting USART	86
Figure 50 The USART data frame at the Rx side	87
Figure 51 USART transmission	87
Figure 52 The USART data frame at the Rx side	88
Figure 53 Receiving USART to data bus.	88
Figure 54 AOC Diagram	89
Figure 55 Diagnostics Request	90
Figure 56 Diagnostics Data	90
Figure 57 Update Notification and Data	91
Figure 58 Update Response	91
Figure 59 SWD and JTAG protocol	92
Figure 60 Load CAN Bootloader code through JTAG	93
Figure 61 Memory Hierarchy	94
Figure 62 Bootloader Flow Chart	95
Figure 63 Main Function	96
Figure 64 Static Functions	96
Figure 65 CAN Initialization	96
Figure 66 Start Command	97
Figure 67 Flash Memory Erasure	97
Figure 68 Firmware Update Process	98
Figure 69 Jumping to Application	98

Figure 63 RTOS Time Slicing	100
Figure 64 Remote Diagnostics - Mode 1	102
Figure 65 Remote Diagnostics - Mode 2	102
Figure 66 Second page	105
Figure 67 Gauges	105
Figure 68 Music page	106
Figure 69 Altium Designer System	106
Figure 70 3D Model of our PCB	107
Figure 71 Real view of our PCB	108
Figure 72 Detailed Diagram with all system components	108

List of Abbreviations

IoT	Internet of things
ARM	Advanced RISC Machines
Wi-Fi	Wireless Fidelity
RAM	Random Access Memory
ROM	Read-Only Memory
FOTA	Firmware over the air
OTA	Over-the-air
MSM	Mobile Software Management
ECU	Electronic control unit
OEM	Original Equipment Manufacturer
CAN	Controller Area Network
USART	Universal Synchronous Asynchronous Receiver Transmitter
GPIO	General Purpose Input/Output
SPI	Serial Peripheral Interface
AES	Advanced Encryption Standard
CAGR	Compound Annual Growth Rate
HLD	High-Level Design
SDLC	Software Development Life Cycle
RPI	Raspberry Pi
iOS	iPhone Operating System
JSON	JavaScript Object Notation
CDN	Content Delivery Network
SDK	Software Development Toolkit
GUI	Graphical User Interface
NoSQL	Not Only Structured Query Language

BaaS	Backend-as-a-Service
PC	Personal Computer
ICP	In-circuit Programming
JTAG	Joint Test Action Group
SWD	Serial Wired Debugger
MCU	Micro-Controller Unit
OEM	Original Equipment Manufacture
SRS	Software Requirements Specifications
CRC	Cyclic Redundancy Check
FIFO	First In First Out
RTR	Remote Transmission Request
FMP	FIFO Message Pending
PWM	Pulse Width Modulation
AUTOSAR	Automotive Open System Architecture
RTOS	Real Time Operating System
FCFS	First Come First Serve

Chapter 1: Introduction

1.1. Introduction to FOTA

1.1.1. What is FOTA?

Firmware Over-The-Air (FOTA) is a Mobile Software Management (MSM) technology in which the operating firmware of a device is wirelessly upgraded and updated by its manufacturer.

Firmware runs in the background without any input from the user and is there to ensure the device's hardware runs properly. The process of downloading these updates wirelessly usually takes little time, depending on the connection speed and the size of the update. This saves businesses the time and money spent sending a technician to have each one of their cellular devices physically upgraded or updated.

Bugfix allows manufacturers to repair faulty units and remotely update software updates. This method typically involves the consumer downloading and updating mobile device firmware through a manufacturer's website or server. FOTA updates are generally accessible through the device menu, software, or firmware update.

Security updates are constantly being released by the manufacturers, which includes a list of all known vulnerabilities during the Data transfer process. All devices must be updated regularly to avoid potential exploitation.

1.1.2. Why do we need FOTA?

Cost-Efficient and better-managed firmware update: Most of the vehicle development process is spread across numerous stakeholders and geographical locations. In such a case, performing any firmware update could be a challenging task that involves multiple revisions and modifications. Thus, an over-the-air update through a wireless network eases the task while lowering any additional cost and time consumed for multiple software firmware updates during the entire lifecycle of a car. IHS automotive had estimated that the total OEM cost savings from OTA (firmware and software) update process will grow to more than \$35 billion in 2022 from \$2.7 billion in 2015.

Upgrade the firmware safely, anytime & anywhere: One of the critical uses of a FOTA update is to send an update to the car, post-decommissioning. FOTA enables the cars to be upgraded remotely, without having the end-user worry about a software-related recall or update. Many OEMs have relied on over-the-air upgrades to fix major functional faults in their automotive software. With FOTA, often most of such firmware bugs can be fixed without the need for a vehicle recall by either sending a new firmware or sending a small patch to update the existing firmware with the bug fixing module. For example, some time back, Tesla's Model S's caught fire in a collision. Tesla dealt with this by sending out an update that changed the suspension settings. Since the update, no further fires have been reported, (although it's not very clear if this was due to the update).

Lowers the Time to market: As already discussed above (point 1) FOTA has been helping the OEMs with reduced time-to-market by updating firmware while the vehicle is still on the production line.

1.1.3. Examples of FOTA

In a well-known example from 2016, Tesla used FOTA to update each of their cars with the ability to self-park. Without FOTA, each car would have had to either be recalled or visited by a Tesla technician for these updates to be installed.

FOTA is particularly useful when it comes to IoT systems, especially those with large numbers of connected devices that require frequent updates. For example, updating hundreds of sensors measuring soil moisture levels across a large farm would be a near-impossible task using the traditional method, in which each sensor would have to be retrieved, connected to a computer or handheld device, reprogrammed with the new update, and placed back in the field, creating unnecessary costs and performance disruption.

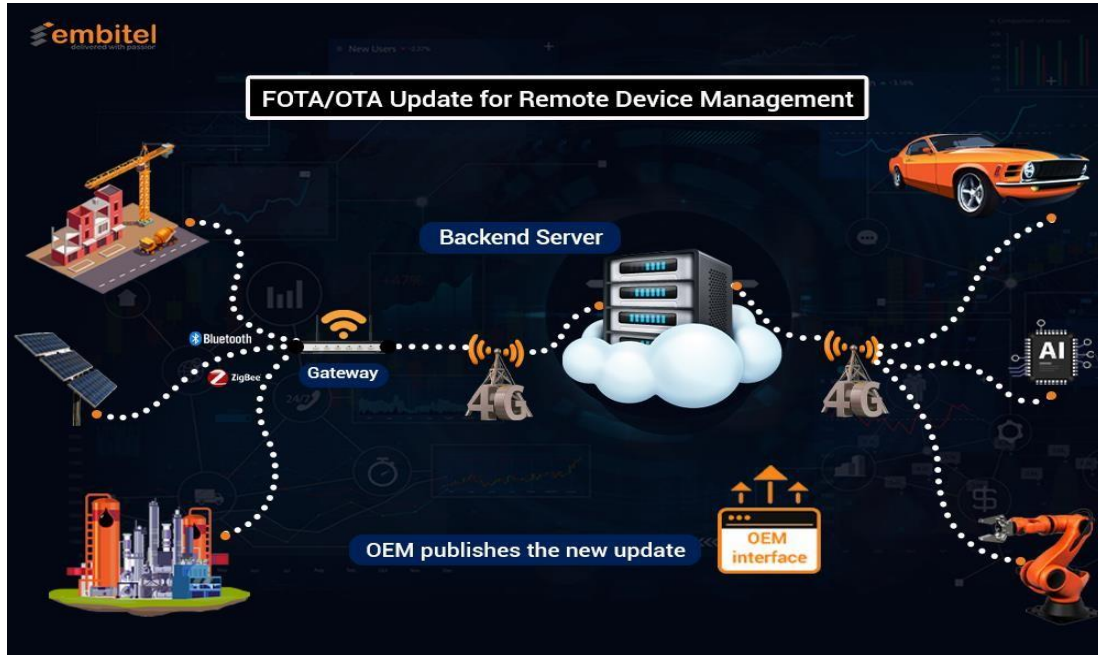


Figure 1.1.11 FOTA/OTA update for Remote Device ManagementFOTA/OTA update for Remote Device Management.

1.2. Flashing Techniques for Microcontrollers

Without FOTA, we use the traditional methods to connect directly to the device through a defined interface and there are different types of connections.

1.2.1. Off-Circuit Programming

In this technique, to program the microcontroller, the microcontroller shall be removed from its application circuit, and then connected to a burner, which is a hardware kit that sometimes has a socket on which the microcontroller can be placed, or simply connected to the required pins with jumpers. It uploads the program to the flash memory, and when it finishes, the microcontroller can now go back to its application circuit and start working, hence called off circuit programming as it requires removing the microcontroller from its application circuit.

The flash interface is external outside the microcontroller and the flash memory programming pins are connected to some microcontroller pins to be programmed through.

When the file is uploaded to the flash using the burner, the microcontroller can be connected again to its application circuit.

1.2.2. In-Circuit Programming

In-system programming (ISP), also called in-circuit serial programming (ICSP), is the ability of some programmable logic devices, microcontrollers, and other embedded devices to be programmed while installed in a complete system, rather than requiring the chip to be programmed before installing it into the system. It also allows firmware updates to be delivered to the on-chip memory of microcontrollers and related processors without requiring specialist programming circuitry on the circuit board and simplifies design work.

There is no standard for in-system programming protocols for programming microcontroller devices. Almost all manufacturers of microcontrollers support this feature, but all have implemented their protocols, which often differ even for different devices from the same manufacturer.

The primary advantage of in-system programming is that it allows manufacturers of electronic devices to integrate programming and testing into a single production phase, and save money, rather than requiring a separate programming stage before assembling the system. This may allow manufacturers to program the chips in their own system's production line instead of buying pre-programmed chips from a manufacturer or distributor, making it feasible to apply code or design changes in the middle of a production run. The other advantage is that production can always use the latest firmware, and new features, as well as bug fixes, can be implemented and put into production without the delay occurring when using pre-programmed microcontrollers.

A flash driver is needed to communicate with the external world to get the required application code, this is done by serial communication, so the flash driver manufacturer defines one or more communication protocols through which the flash driver can interface, for example, stm32f103 has 2 protocols that are provided to communicate with the in-circuit flash programmer: JTAG and SWD.

We used the ST-Link programmer and debugger, It works as a translator from the USB protocol to the SWD protocol, this device is target-specific which means that changing the microcontroller to another one with a different flash interface communication protocol will result in changing the translator device, this will be a big headache if it is required to reprogram several different microcontrollers in the same system, That's one of the reasons why we use a bootloader.

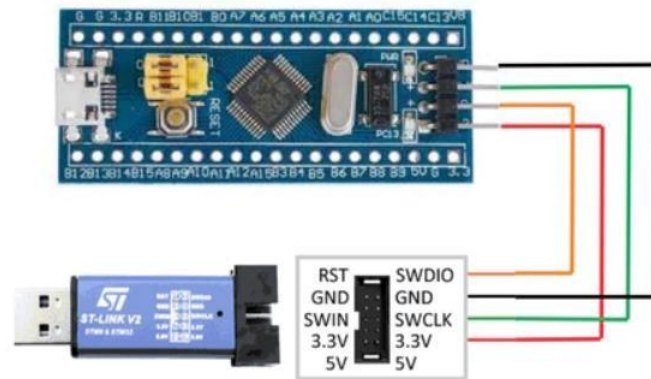


Figure 2 ST-Link Programmer.

1.2.3. In-Circuit Programming with Bootloader

A Bootloader is a program that allows you to load other programs via a more convenient interface like a standard USB cable. When you power up or reset your microcontroller board, the bootloader checks to see if there is an upload request. If there is, it will upload the new program and burn it into Flash memory. If not, it will start running the last program that you loaded. So, instead of different interfaces with different microcontrollers, we use a bootloader to create a unified interface with the system.

1.3. Software Development Lifecycle

SDLC, or Software Development Life Cycle, is a set of steps used to create software applications. These steps divide the development process into tasks that can then be assigned, completed, and measured.

1.3.1. What is Software Development Lifecycle?

Software Development Life Cycle is the application of standard business practices to building software applications. It's typically divided into six to **eight steps**: Planning, Requirements, Design, Build, Document, Test, Deploy, and Maintain. Some project managers will combine, split, or omit steps, depending on the project's scope. These are the core components recommended for all software development projects.

SDLC is a way to measure and improve the development process. It allows a fine-grain analysis of each step of the process. This, in turn, helps companies

maximize efficiency at each stage. As computing power increases, it places a higher demand on software and developers. Companies must reduce costs, deliver software faster, and meet or exceed their customers' needs. SDLC helps achieve these goals by identifying inefficiencies and higher costs and fixing them to run smoothly.

1.3.2. How the Software Development Life Cycle Works

The Software Development Life Cycle simply outlines each task required to put together a software application. This helps to reduce waste and increase the efficiency of the development process. Monitoring also ensures the project stays on track and continues to be a feasible investment for the company.

Many companies will subdivide these steps into smaller units. Planning might be broken into technology research, marketing research, and a cost-benefit analysis. Other steps can merge. The Testing phase can run concurrently with the Development phase since developers need to fix errors that occur during testing.

1.3.3. The Seven Phases of the SDLC

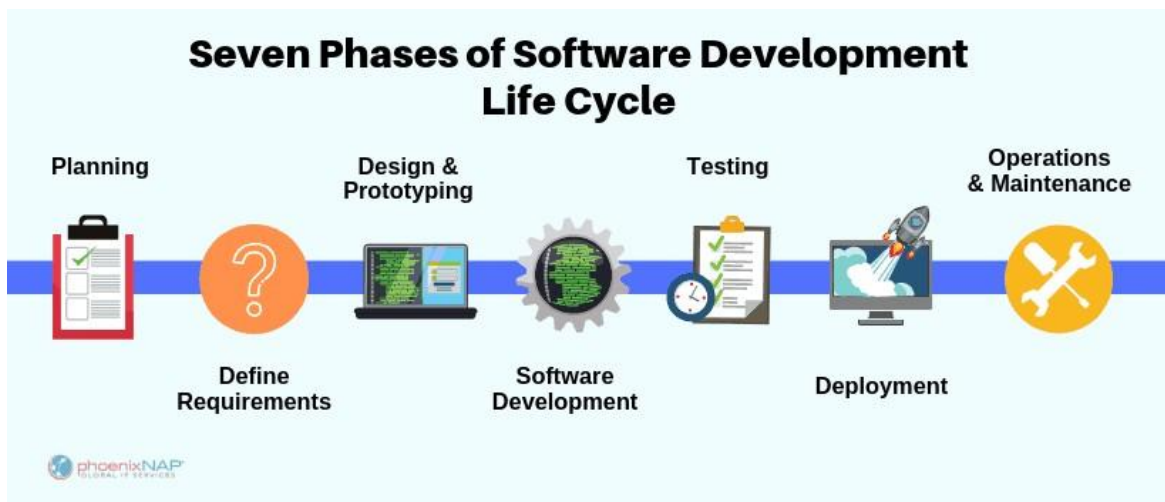


Figure 3 The Seven Phases of the SDLC

1) Planning

In the Planning phase, project leaders evaluate the terms of the project. This includes calculating labor and material costs, creating a timetable with target goals, and creating the project's teams and leadership structure.

Planning can also include feedback from stakeholders. Stakeholders are anyone who stands to benefit from the application. Try to get feedback from

potential customers, developers, subject matter experts, and sales reps. Planning should clearly define the scope and purpose of the application. It plots the course and provisions the team to effectively create the software. It also sets boundaries to help keep the project from expanding or shifting from its original purpose.

2) Define Requirements

Defining requirements is considered part of planning to determine what the application is supposed to do and its requirements. For example, a social media application would require the ability to connect with a friend. An inventory program might require a search feature.

Requirements also include defining the resources needed to build the project. For example, a team might develop software to control a custom manufacturing machine. The machine is a requirement in the process.

3) Design and Prototyping

The Design phase models the way a software application will work. Some aspects of the design include:

- **Architecture** – Specifies programming language, industry practices, overall design, and use of any templates or boilerplate.
- **User Interface** – Defines the ways customers interact with the software, and how the software responds to input.
- **Platforms** – Defines the platforms on which the software will run, such as Apple, Android, Windows version, Linux, or even gaming consoles.
- **Programming** – Not just the programming language but including methods of solving problems and performing tasks in the application.
- **Communications** – Defines the methods by which the application can communicate with other assets, such as a central server or other instances of the application.
- **Security** – Defines the measures taken to secure the application, and may include SSL traffic encryption, password protection, and secure storage of user credentials.

Prototyping can be a part of the Design phase. A prototype is like one of the early versions of software in the Iterative software development model. It

demonstrates a basic idea of how the application looks and works. This “Handson” design can be shown to stakeholders. Use feedback to improve the application. It’s less expensive to change the Prototype phase than to rewrite code to make a change in the Development phase.

4) Software Development

This is the actual writing of the program. A small project might be written by a single developer, while a large project might be broken up and worked by several teams. Use an Access Control or Source Code Management application in this phase. These systems help developers track changes to the code. They also help ensure compatibility between different team projects and to make sure target goals are being met.

The coding process includes many other tasks. Many developers need to brush up on skills or work as a team. Finding and fixing errors and glitches is critical. Tasks often hold up the development process, such as waiting for test results or compiling code so an application can run. SDLC can anticipate these delays so that developers can be tasked with other duties.

Software developers appreciate instructions and explanations. Documentation can be a formal process, including wiring a user guide for the application. It can also be informal, like comments in the source code that explain why a developer used a certain procedure. Even companies that strive to create software that’s easy and intuitive benefit from the documentation.

Documentation can be a quick guided tour of the application’s basic features that are displayed on the first launch. It can be video tutorials for complex tasks. Written documentation like user guides, troubleshooting guides, and FAQs help users solve problems or technical questions.

5) Testing

It’s critical to test an application before making it available to users. Much of the testing can be automated, like security testing. Another testing can only be done in a specific environment – consider creating a simulated production environment for complex deployments. Testing should ensure that each function works correctly. Different parts of the application should also be tested to work seamlessly together—performance test, to reduce any hangs or lags in

processing. The testing phase helps reduce the number of bugs and glitches that users encounter. This leads to higher user satisfaction and a better usage rate.

6) Development

In the deployment phase, the application is made available to users. Many companies prefer to automate the deployment phase. This can be as simple as a payment portal and download link on the company website. It could also be downloading an application on a smartphone.

Deployment can also be complex. Upgrading a company-wide database to a newly developed application is one example. Because there are several other systems used by the database, integrating the upgrade can take more time and effort.

7) Operations and Maintenance

At this point, the development cycle is almost finished. The application is done and being used in the field. The Operation and Maintenance phase is still important, though. In this phase, users discover bugs that weren't found during testing. These errors need to be resolved, which can spawn new development cycles.

In addition to bug fixes, models like Iterative development plan additional features in future releases. For each new release, a new Development Cycle can be launched.

Chapter 2: Problem Definition

2.1. Problem Definition

The technological evolution of the car has been in sync with rapid advances in communication, information processing, and electronic hardware systems. Starting with isolated Electronic Control Units (ECUs) for Engine Management and AntiLock Braking, typical cars now use from 25 to 100 microcontrollers for providing many features to ensure comfort and safety to the driver and passengers. Vehicles were mostly seen as mechanical systems. But that has changed a lot with the introduction of electronics in vehicles.

In 1970 the Electronic Control Unit (ECU) was introduced to the automotive industry. Modern-day vehicles are full of electronic devices. The ECUs used in the car, for handling its overall function, are controlled by complex software code called firmware. With a large number of ECUs present in a car, the size of code in a modern luxury car can exceed 100 million lines of code as compared to 5 to 6 million lines of code required for a supersonic fighter aircraft. Maintaining and updating such a large volume of code for several thousand vehicles in production and millions in the field is a mammoth task.

Recalling vehicles in the field to correct field problems, introduce new features, and upgrade vehicle performance can cause a loss of reputation for the OEM and incurrence of huge costs. Hence a methodology to update the firmware of automotive ECUs automatically in an error-free manner and at a faster rate is necessary to support the programming and updating of the firmware of a modern car during its life cycle.

FOTA (Firmware over the Air) is emerging as a new flexible solution for these problems. FOTA is already an established technology for mobile phones and its application in the automotive domain helps to tackle the rising complexity of automotive systems. FOTA is based on the wireless communication between the firmware server located in Cloud and the Telematics Control Unit used in a car as a client to download the new firmware.

FOTA can update the firmware of a car at any location whether it is an assembly shop, a dealer location, a service workshop, or the parking space of the owner. Besides, FOTA allows the download of firmware in the background when the

vehicle is running and informs the driver, owner, or service person when the installation of the updated firmware can be started.

2.2. Implementing FOTA

FOTA implementation is a three-step process:

- a) Generating and storing required new versions of firmware and its updates in the databases located on cloud-based servers accessible to all stakeholders.
- b) Downloading required firmware in the local storage unit attached to the Telematics Control Unit in the vehicle using wireless networks.
- c) Installation of the new firmware and updating of ECUs in the vehicle.

FOTA is a collaborative process and requires participation from different stakeholders involved in providing life cycle support to the vehicle.

2.3. Major Challenges

Based on this high-level description of the OTA update process, three major challenges arise that the OTA update solution must address. The first challenge relates to **memory**. The software solution must organize the new software application into volatile or nonvolatile memory of the client device so that it can be executed when the update process is complete.

The solution must ensure that a previous version of the software is kept as a fallback application in case the new software has problems. Also, we must retain the state of the client device between resets and power cycles, such as the version of the software we are currently running, and where it is in memory. The second major challenge is **communication**.

The new software must be sent from the server to the client in discrete packets, each targeting a specific address in the client's memory. The scheme for packetizing, the packet structure, and the protocol used to transfer the data must all be accounted for in the software design. The final major challenge is **security**.

With the new software being sent wirelessly from the server to the client, we must ensure that the server is a trusted party. This security challenge is known as authentication. We also must ensure that the new software is obfuscated by many observers, since it may contain sensitive information. This security challenge is known as confidentiality. The final element of security is integrity, ensuring that the new software is not corrupted when it is sent over the air.

2.4. FOTA Benefits

Continuous improvement

FOTA updates allow manufacturers to continuously amend their connected systems, fix bugs, and enhance product performance, even when they are on the production line or in the hands of a consumer. This approach eliminates recalls and in-person maintenance and provides a competitive edge: this way, there is no need to wait to incorporate new features into a new batch of devices.

Reduces risk.

With FOTA giving the IoT device distributor or manufacturer the ability to continually configure devices, and post-distribution, they also gain the ability to remain compliant with evolving industry standards. This expands product lifetime and offers greater flexibility to manage the devices on the edge.

Cost-effective

The ability to manage FOTA allows the manufacturer remotely and conveniently to cut down on customer care and other operational costs. The efficiency, flexibility, and convenience offered by this feature save techs the hassle of traveling to fix bugs and similar problems on-site; hence it's time efficient. The time saved from these trips can be invested in other valuable projects.

Gets to the market quicker.

IoT devices need to be affordably constructed but reliably built. Nevertheless, with FOTA capabilities, the manufacturer gains the opportunity to iterate; engineers can spend less time determining how they will physically upgrade each device or how to render updates unnecessary. This allows the IoT provider to continue the work to minimize the size of the software solution to optimize processing on the edge, shorten download time and lessen network congestion, even after going to market.

Sustainability and adaptability

Better sustainability and adaptability of devices with the help of FOTA. As small updates are possible in the FOTA system, software/firmware development teams

have the leeway to carry on with their tasks of development even in the post-sales scenario of a product.

Additional revenue streams.

Original Equipment Manufacturers (OEMs) can implement add-ons through OTA updates after the release, without physical access to the firmware that needs an upgrade.

2.6. Market Research

Automotive Firmware over the Air (FOTA) Market Statistics 2030

With a CAGR of 18.1% from 2021 to 2030, the global automobile over-the-air (OTA) market, which was valued at \$2.59 billion in 2020, is anticipated to grow to \$13.71 billion by 2030.

The COVID-19 outbreak harshly impacted the automotive sector on a global level, which in turn resulted in a substantial drop in vehicle sales and the insufficiency of raw materials. Many industry players in the automotive sector have witnessed issues such as a halt of production activities and mandated plant closures by the government. However, the pandemic resulted in an indirect effect on the automotive over-the-air (OTA) market.

At the mid-pandemic industry leaders are to formulate different ways to integrate critical information such as contact information, a list of nearest hospitals/medical centers, and quick reference information for primary symptoms with official local governmental guidance along with enhanced user experience. Adoption of such methods is foreseen to reinforce the demand for automotive over the air (OTA) in the near future.

The need for automotive over-the-air updates is anticipated to increase due to factors like the rise in connected vehicle adoption, the rise in demand for electric vehicles, government regulations regarding vehicle safety and cyber security, and government initiatives for implementing connected car technology. However, high over-the-air update costs and a lack of infrastructure in developing nations may limit growth potential.



Figure 4 Global Automotive OTA Updates Mark

Based on type, the automotive over-the-air updates market had a greater revenue share for software in 2020. The OTA software update technology is a more effective approach for OEMs to update the software and repair issues than the traditional method. Easy connectivity between embedded systems and connected devices, which offers vital information like the location of fuel and charging stations and the status of the battery and charging, in addition to facilitating remote diagnosis of all types of connected vehicles, is the main factor causing this category to dominate the market.

In terms of market share for automobile over-the-air upgrades in 2020, North America dominated. The region's growing use of connected automobiles is largely responsible for this leadership position. With mobile devices becoming increasingly compatible with these vehicles, there is a huge demand for networking, navigation, communication, and entertainment services. Additionally, regional OEMs and technology providers continually invest sizable sums in the development of sturdy technologies, which is drawing in buyers for vehicles. For instance, Tesla Inc. offers a module for troubleshooting as well as routinely downloading OTA software upgrades that improve and add to the current capabilities over Wi-Fi.

Major automakers are being encouraged to turn their focus to linked vehicles and either develop new variants or include cutting-edge connection capabilities into

the existing models as a result of the growing acceptance of internet of things (IoT) technology. This is a response to people's rising desire for cutting-edge navigation, telematics, and infotainment systems. For proper operation, these systems depend on the software that is placed on the embedded hardware of the vehicle. Thus, key development in the market for automotive over-the-air updates is the increasing adoption of smart, connected technologies.

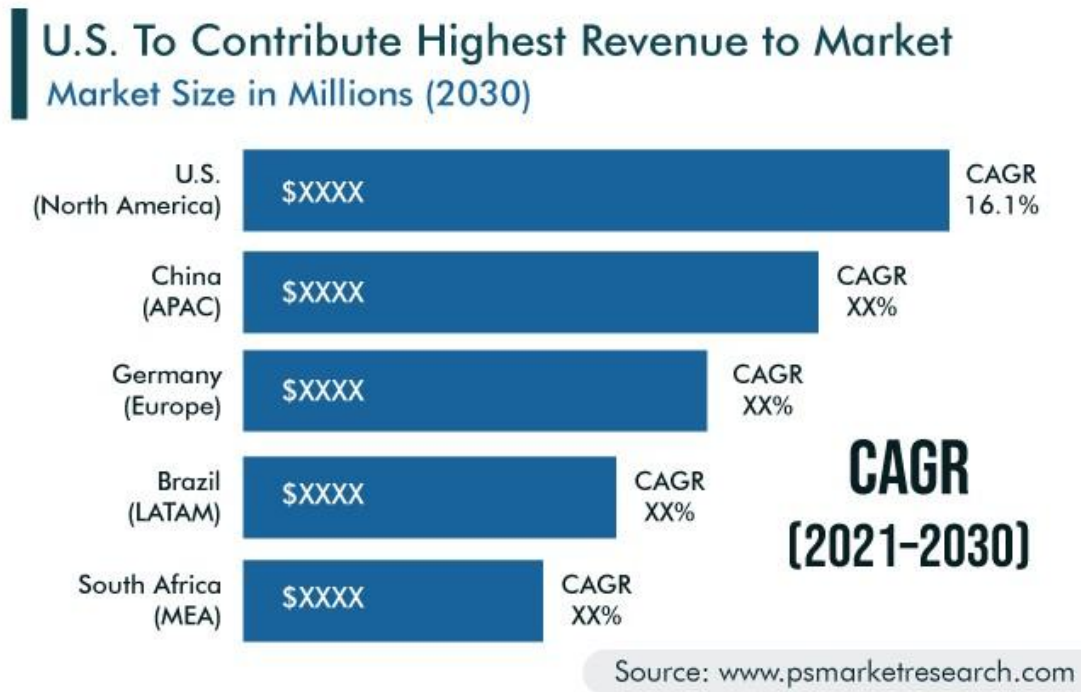


Figure 5 Market Size

Because of the speed at which the automotive industry is going digital, it follows that to stay up with the most recent developments in connectivity technology, vehicles must regularly receive software upgrades, including cybersecurity patches. Installing these updates wirelessly or OTA is faster and more cost-effective than repeatedly calling the sold vehicles back to a garage. Additionally, OTA updates support IoT environments by improving functionality, simplifying the introduction of new features, and securing devices from hacking, data breaches, and other forms of cyber-attacks. These additional advantages of OTA updates serve as a key industry driver.

The global automotive OTA updates market is anticipated to be driven by tight standards relating to vehicle and passenger safety. The primary priority for

automotive manufacturers worldwide has always been vehicle safety. It has become crucial for manufacturers to adapt their products to the current vehicle standards as a result of the implementation of greater safety laws and monitoring standards. Additionally, automakers are concentrating quickly on upgrading their vehicles to ensure compliance with the updated norms and regulations due to the rising sophistication of automotive technology and the increasing degree of programmability.

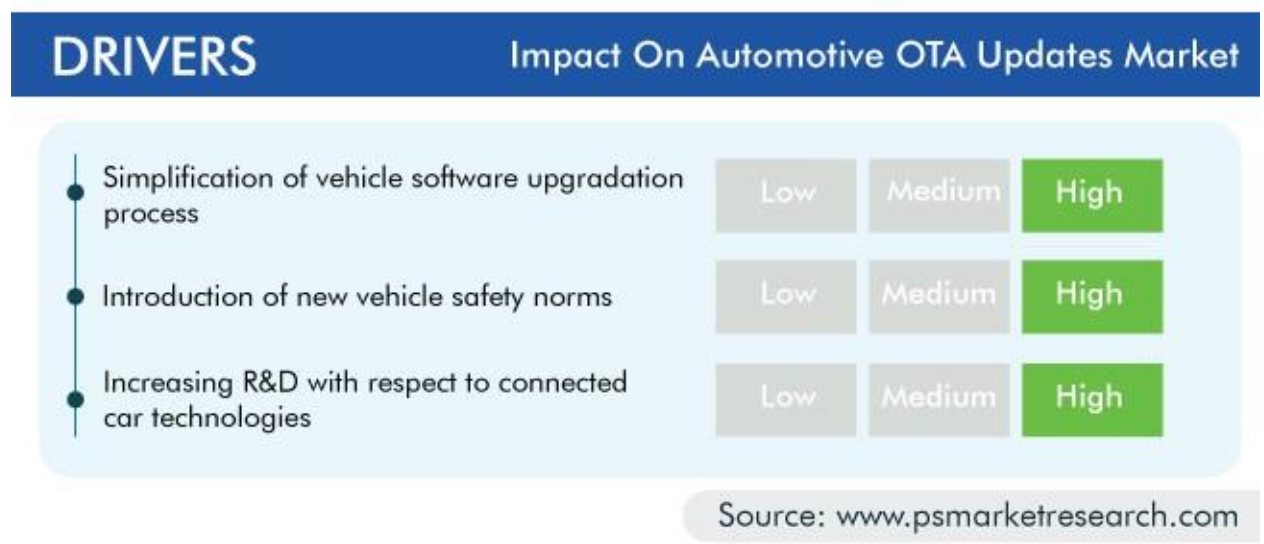


Figure 6 Impact on Automotive Updates Market.

Chapter 3: System Design

3.1. System Diagram

3.1.1. Abstract View

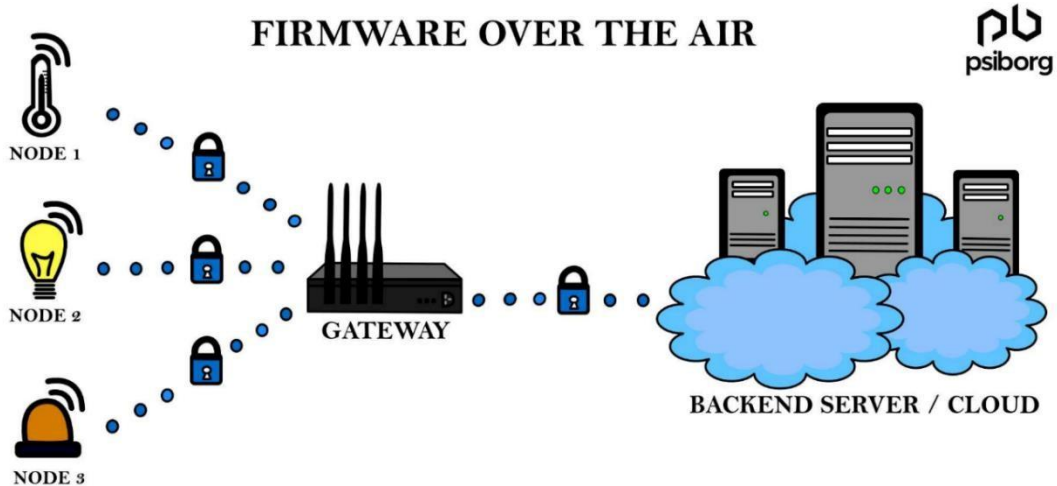


Figure 7 System Design.

The previous diagram represents an overview of our FOTA system which includes:

1- Backend Server/Cloud

Responsible for the management of vehicle software release, and optionally to customize updates for every vehicle client based on OEM policies. It's where the users can get new updates for their systems or send diagnostics to get problems solved.

2- Telematics Unit

It's the bridge connecting the server and the whole system allowing you to send and receive multiple data through it. It can interface with other systems to provide Wi-Fi and Bluetooth functionality through its SPI / SDIO or I2C / UART interfaces.

3- Gateway (STM32)

The FOTA update will be sent to a Gateway through the telematics units after getting permission from the user, it's literally a gateway connecting between the telematics unit and the target ECU. It's also used to get important data from the received update before sending it to the target ECU, like the CRC and target ECU ID.

4- Target ECU

It's where the update is directed to after going through the gateway and where the diagnostics are carried out. It's connected to the gateway through a communication bus like the can bus.

There is also the bootloader which it is designed to download the firmware in MCU's internal or external memory. It supports FOTA updates and must be installed in the ECU before shipping the product. And a graphical user interface which will make it easier for the user to handle the product system and make it more reliable by making it easier to accept a new update or send diagnostics to the OEM server.

3.1.2. Implemented View

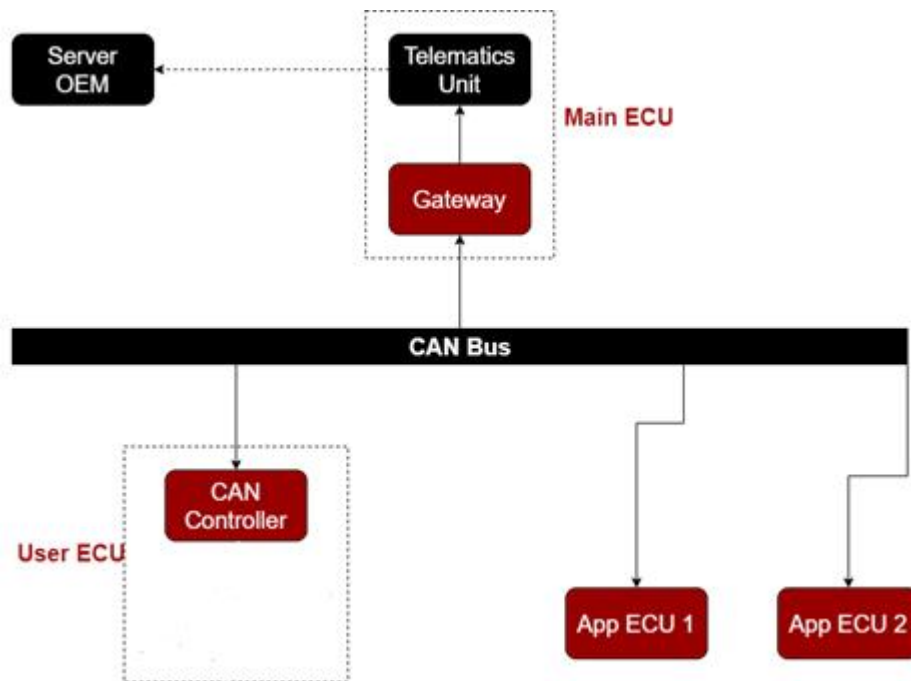


Figure 8 Implemented View System.

As an overview of our implemented system, we used:

- Google Firebase for the OEM server as it provides real-time database and free large storage.
- ESP32 as the telematics unit, as it has a Wi-Fi module which will make it easy to connect to the server and will be connected to gateway through a UART bus.

- Gateway is implemented using STM32F103C8T6 ARM-based microcontroller, as it has the required communication protocols.
- CAN transceiver MCP 2551 which serves as the interface between a CAN protocol controller and the physical bus.
- And the same microcontroller is used which is the applications of the target ECUs and the user interface.

3.2. System Components

We can divide our system into a bunch of subsystems connected through communication protocols like UART or CAN protocols, and we're going to talk in more detail about them in the following section.

3.2.1. Main ECU

Telematics Unit

As we said before, the Telematics unit is the bridge connecting the server and the whole system allowing us to send and receive multiple data through it.

We use the Telematics unit to communicate mainly with two other units, the server, and the gateway (STM32).

We used ESP32 module as it has a WI-FI module which is easy to establish using a few lines of code through Arduino IDE which gives us access to multiple libraries and functions in the ESP32.



Figure 9 ESP32 IC.

The main function of the ESP32 is:

1- Communicate to the server:

- Download the updated file.
 - Decrypt downloaded update files.
 - Send a request to download the update file.
 - Send diagnostics to the server.
- 2- Communicate to the Gateway:
- Send the update file to the Gateway.
 - Receive download request from the user.

ESP32 Specs:

- Single or Dual-Core 32-bit Microprocessor with clock frequency up to 240 MHz
- 520 KB of SRAM, 448 KB of ROM and 16 KB of RTC SRAM.
- Supports 802.11 b/g/n Wi-Fi connectivity with speeds up to 150 Mbps.
- Support for both Classic Bluetooth v4.2 and BLE specifications.
- 34 Programmable GPIOs.
- Up to 18 channels of 12-bit SAR ADC and 2 channels of 8-bit DAC
- Serial Connectivity includes 4 x SPI, 2 x I²C, 2 x I²S, 3 x UART.
- Ethernet MAC for physical LAN Communication (requires external PHY).
- 1 Host controller for SD/SDIO/MMC and 1 Slave controller for SDIO/SPI.
- Motor PWM and up to 16-channels of LED PWM.
- Secure Boot and Flash Encryption.
- Cryptographic Hardware Acceleration for AES, Hash (SHA-2), RSA, ECC and RNG.

Gateway

Gateway acts as a bridge between the ESP32 module and the target ECUs, it receives the update code from the ESP through UART protocol and forward it to the target ECU through a CAN bus is very important especially when we're having more than one target ECU, and this is surely the case in the cars industry.

We used STM32F103C8T6 ARM-based microcontroller as our gateway, and its main functions are:

- Receive the downloaded update from the Telematics unit before sending it to the target ECU.
- Get important data from the downloaded update like the CRC and target ECU ID.

- Save information about connected ECUs, including ECUs IDs, software version of every ECU and any important data related to the connected ECUs.
- Determine which ECU this code is for, and that is one of the main tasks to make sure the code is delivered to the correct ECU.
- Notify the node when the update is complete so the node can notify the server too that the code update is done.

STM32f103c8t6 microcontroller

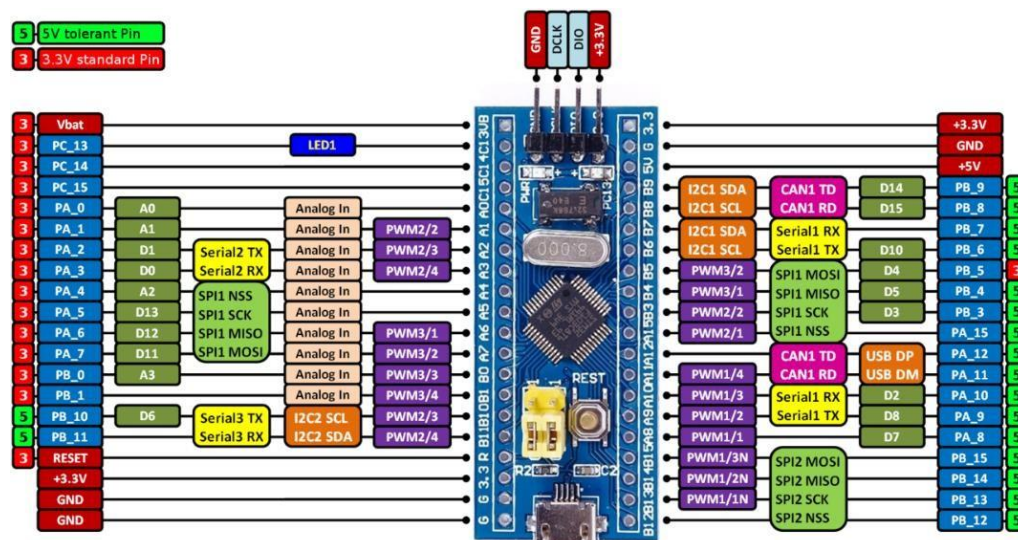


Figure 10 STM32 IC

STM32f103c8t6 Specs:

Pinout Configuration

Category	Pin Name	Details
----------	----------	---------

Power	3.3V, 5V, GND	<ul style="list-style-type: none"> • 3.3V – Regulated output voltage from the onboard regulator (drawing current is not recommended), CAN also be used to supply the chip. • 5V from USB or onboard regulator can be used to supply the onboard 3.3V regulator. • GND – Ground pins
Analog Pins	PA0 – PA7 PB0 – PB1	Pins act as ADCs with 12-bit resolution
Input/output pins	PA0 – PA15 PB0 – PB15 PC13 – PC15	37 General-purpose I/O pins.
Serial	TX1, RX1 TX2, RX2 TX3, RX3	UART with RTS and CTS pins
External interrupts	PA0 – PA15 PB0 – PB15 PC13 – PC15	All digital pins have interrupt capability
PWM	PA0 – PA3	15 PWM pins total
	PA6 – PA10 PB0 - PB1 PB6 – PB9	

SPI	MISO0, MOSI0, SCK0, CS0 MISO1, MOSI1, SCK1, CS0	2 SPI
Inbuilt LED	PC13	LED to act as a general-purpose GPIO indicator
I²C	SCL1, SDA1 SCL2, SDA2	Inter-Integrated Circuit communication ports
CAN	CAN0TX, CAN0RX	CAN bus ports

Technical Specifications

Microcontroller	STM32F103C8T6
Operating voltage	3.3V
Analog inputs	10
Digital I/O pins	37
DC source/sink from I/O pins	6mA
Flash memory (KB)	64/128
SRAM	20KB
Frequency (clock speed)	72MHz max.

Communication	I ² C, SPI, UART, CAN, USB
----------------------	---------------------------------------

3.2.2. App ECU

It's where the updates are heading after going through the Gateway.

- 1- The first target ECU is implemented to act as a collision system, and it's a simple system that gives a warning whenever the driver is about to collide with another object. Also, there is an engine system that simulates the motion of the vehicle with feedback for the motor to monitor its response.

It's composed mainly of:

1. **Ultrasonic sensor:** We used an HC-SR04 sensor to provide the distance, so that it can give feedback if the driver is about to collide with another object, as the Ultrasonic module can detect existence of an object by sending ultrasound signals and monitoring its reflection. Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach 3mm.



Figure 11 HC- SR04 IC.

2. **Buzzer:** When the ultrasonic sensor detects an object, the buzzer will fire an alarm to warn the driver that when the object is out of range it will stop.
3. **DC Servo Motor with Encoder:** The Motor rotates in both directions using the L298 motor driver according to inputs from push buttons and the encoder returns feedback for the direction of rotation.



Figure 12 DC Motor with Encoder

4. The second target ECU is a temperature monitoring system in which we used LM35 which is a precision Integrated circuit Temperature sensor, whose output voltage varies, based on the temperature around it. It can be used to measure temperature anywhere between -55°C to 150°C and It can easily be interfaced with any Microcontroller that has an ADC function or any development platform like STM32, which we've already used with that system.

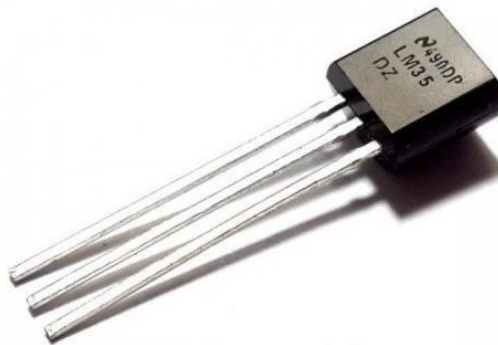


Figure 13 LM35 Temperature Sensor



Figure 14 Nextion HMI Display

5. Nextion HMI Display:

We used Nextion HMI as its display has a high resolution and a large size, making it easy to read and interact with even while driving. The display can be customized to show relevant data, such as speed, fuel level, and engine diagnostics. The touchscreen interface allows for a user-friendly experience and can be programmed to respond to various gestures. We used the NX8048T070 version, and these are the features.

- 800 x 480 Resolution.
- RGB 65K true-to-life colors.
- TFT Screen with integrated 4-wire Resistive Touch Panel
- 4-pin TTL serial interface.
- 16M Flash memory for User Application Code and Data.
- Onboard micro-SD card slot for firmware upgradation.
- Visual Area: 154.08mm(L)×85.92mm(W).
- Adjustable Brightness: 0~230 nit, the interval of adjustment is 1%.
- Recommended Power Supply: 5V, 2A, DC.

6. Mp3 Player:

Certainly! Let's walk through how the application would work in a step-by-step manner:

1. **Initialization:** Upon power-up or reset, the STM32 microcontroller initializes its UART communication module, configures GPIO pins for communication with the DFPlayer module, and sets up the audio output pins to connect to the speaker.
2. **DFPlayer Initialization:** The STM32 sends the required initialization commands to the DFPlayer module via UART. This step ensures that the

DFPlayer is ready to receive further commands and initializes any settings, such as volume level and playback mode.

3. **Sending Commands:** Based on the user input or predefined actions, the STM32 sends the appropriate UART commands to the DFPlayer module. These commands instruct the DFPlayer to perform various functions, such as starting playback, stopping playback, adjusting volume, selecting a specific track, and more.
4. **Audio Playback:** When the DFPlayer receives the playback command, it reads the audio file from the microSD card or USB drive (whichever storage medium is being used), decodes the audio data, and sends the audio signal to the STM32.
5. **Audio Output:** The STM32 receives the audio signal from the DFPlayer and routes it to the speaker through the configured audio output pins. The speaker then reproduces the audio, allowing the user to hear the sound.
6. **User Interaction and Control:** Throughout the playback process, the STM32 remains responsive to user input. For example, the user might press a "Pause" button to pause the audio, "Next" button to play the next track, or adjust volume using dedicated buttons.
7. **End of Playback:** When the audio reaches its end or the user explicitly stops playback, the STM32 can send a command to the DFPlayer to stop playback. This ensures that the DFPlayer is in a ready state for the next command.
8. **Application Loop:** The application continues to run in a loop, waiting for user input, processing commands, and controlling the audio playback until the user decides to power off the device or perform other specific actions.

This basic flow illustrates how the application works to control the DFPlayer module via UART and produce audio output through the connected speaker. Depending on your specific project requirements, you can expand the application to include more features, like playlists, equalizer settings, file management, and other functionalities.

The Process:

The instructions come from the website or the Nextion screen to control the main functions of the mp3 player by using specific command over UART to the STM32. For example:

- Play: "PLAY"
- Pause: "PAUSE"
- Mute: "MUTE"

- Unmute: "UNMUTE"
- Next: "NEXT"
- Previous: "PREVIOUS"
- Volume Up: "VOLUME_UP"
- Volume Down: "VOLUME_DOWN"

By using ADC, the speaker is connected to an ADC (Analog-to-Digital Converter) to indicate the volume level, and accordingly, an LED's intensity increases when the track layers get higher.

3.2.3. User ECU Can Controller

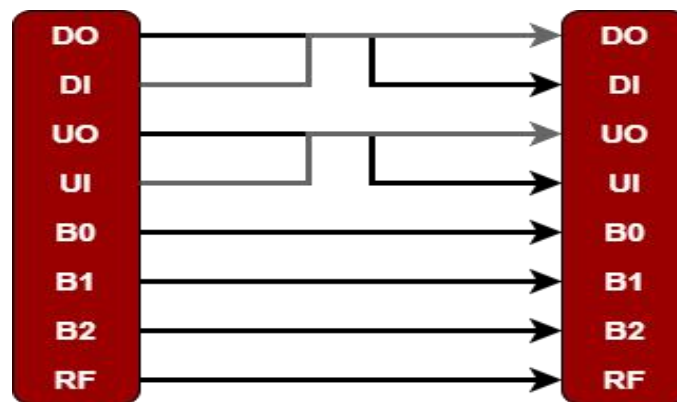


Figure 15 Asynchronous Octal Communication Pins

Since raspberry pi doesn't have a CAN peripheral, we needed CAN controller to convert the data provided by the application into a CAN message frame fit to be transmitted across the bus.

Our first approach was to use SPI to CAN Bus Converter (MCP 2515), but the RPI pins have a maximum voltage input of 3.3V and our CAN Bus has a voltage level of 5V. Also, the module didn't have clear documentation.

So, we added another STM32 to interpret the messages on the CAN Bus and communicate with the Raspberry Pi.

At first, we tried USART, but the Serial Communication of the Raspberry Pi wasn't very reliable, so we defined our own protocol to communicate between RPI and STM.

The protocol is called Asynchronous Octal Communication (AOC) which is a simple parallel communication to indicate updates and diagnostics requests and data.

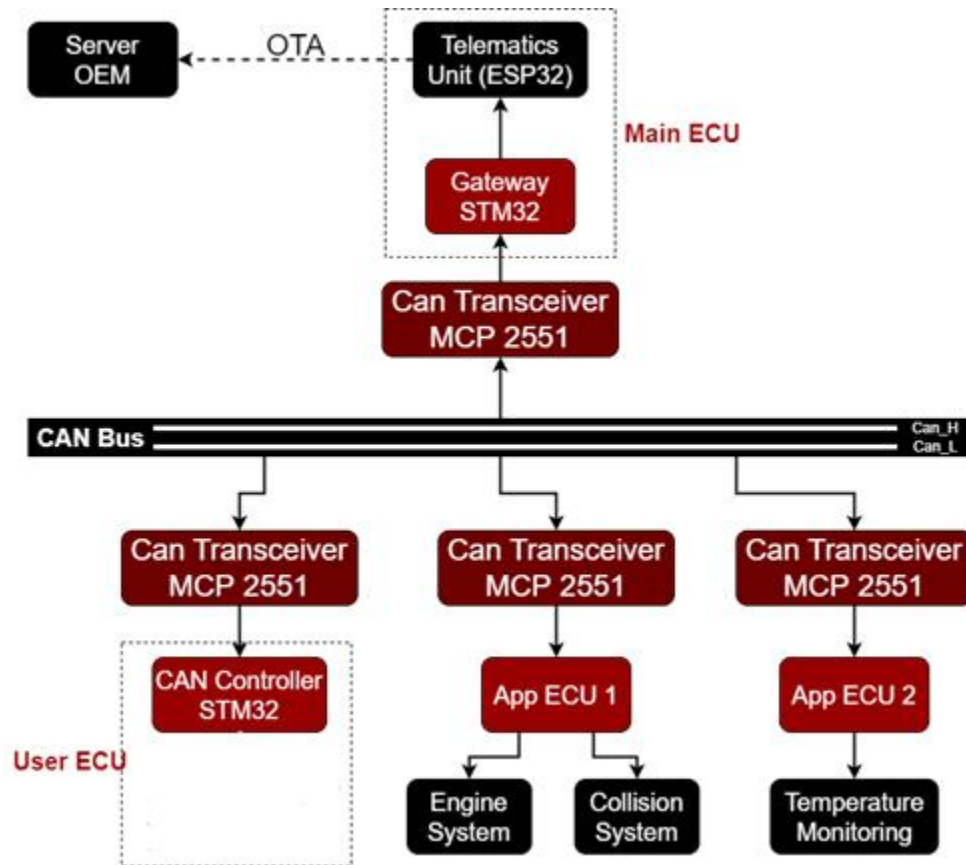


Figure 16 Detailed View of System

3.2.4. Server

For the server implementation, we used Google Firebase which is a Google backed application development software that enables developers to develop iOS, Android, and Web apps. Firebase provides tools for tracking analytics, reporting, and fixing app crashes, creating marketing and product experiment. It's free and easy to use, and it has a real-time database.

Google Firebase advantages:

- **Real-time Database**

A real-time Database is a cloud-hosted database. Data is stored as JSON and synchronized continuously to each associated client.

- **Hosting**

Hosting is production-grade web content that facilitates the developers. With Hosting, you can rapidly and effectively send web applications and static content to a Content Delivery Network (CDN) with a single command.

- **Authentication**

Firebase Authentication gives back-end development services, simple-to-use SDKs, and instant UI libraries to confirm clients over your application. It supports authentication using passwords, email ID, or username.

- **Storage**

It is built for application developers who need to store and serve user-generated content, for example, photos or videos. It gives secure document transfers and downloads for Firebase applications, regardless of network quality.

3.3. Software Design

3.3.1. Software Design Process

software Design is the process by which an agent creates a specification of a software artifact intended to accomplish goals, using a set of primitive components and subject to constraints. It's the process of envisioning and defining software solutions to one or more sets of problems.



Figure 17 Software Development Cycle

3.3.2. Design Considerations:

There are many aspects to consider in the design of a piece of software. The importance of each consideration should reflect the goals and expectations that the software is being created to meet. Some of these aspects are:

- **Compatibility** - The software can operate with other products that are designed for interoperability with another product. For example, a piece of software may be backward-compatible with an older version of itself.
- **Extensibility** - New capabilities can be added to the software without major changes to the underlying architecture.
- **Modularity** - the resulting software comprises well-defined, independent components which leads to better maintainability. The components could be then implemented and tested in isolation before being integrated to form a desired software system. This allows division of work in a software development project.
- **Fault-tolerance** - The software is resistant to and able to recover from component failure.
- **Maintainability** - A measure of how easily bug fixes or functional modifications can be accomplished. High maintainability can be the product of modularity and extensibility.
- **Reliability** (Software durability) - The software can perform a required function under stated conditions for a specified period of time.
- **Reusability** - The ability to use some or all the aspects of the preexisting software in other projects with little to no modification.
- **Robustness** - The software can operate under stress or tolerate unpredictable or invalid input. For example, it can be designed with resilience to low memory conditions.
- **Security** - The software can withstand and resist hostile acts and influences.
- **Usability** - The software user interface must be usable for its target user/audience. Default values for the parameters must be chosen so that they are a good choice for most of the users.^[6]

- **Performance** - The software performs its tasks within a timeframe that is acceptable for the user and does not require too much memory.
- **Portability** - The software should be usable across several different conditions and environments.
 - **Scalability** - The software adapts well to increasing data or added features or number of users.

3.3.3. Software Design Approaches

1- Top-Down Design Model:

In the top-down model, an overview of the system is formulated without going into detail for any part of it. Each part of it, then refined into more details, defining it in yet more details until the entire specification is detailed enough to validate the model. If we glance at a haul as a full, it's going to appear not possible as a result of it being so complicated for example: Writing a university system program, writing a word processor. Complicated issues may be resolved victimization high down style, conjointly referred to as Stepwise refinement where:

1. We break the problem into parts,
2. Then break the parts into parts soon and now each part will be easy to do.

Advantages:

- Breaking problems into parts helps us to identify what needs to be done.
- At each step of refinement, new parts will become less complex and therefore easier to solve.
- Parts of the solution may turn out to be reusable.
 - Breaking problems into parts allows more than one person to solve the problem.

Bottom-Up Design Model:

In this design, individual parts of the system are specified in detail. The parts are linked to form larger components, which are in turn linked until a complete system is formed. Object-oriented languages such as C++ or java use a bottom-up approach where each object is identified first.

Advantages:

- Make decisions about reusable low-level utilities then decide how they will be put together to create high-level construct.
- The contrast between Top-down design and bottom-up design.

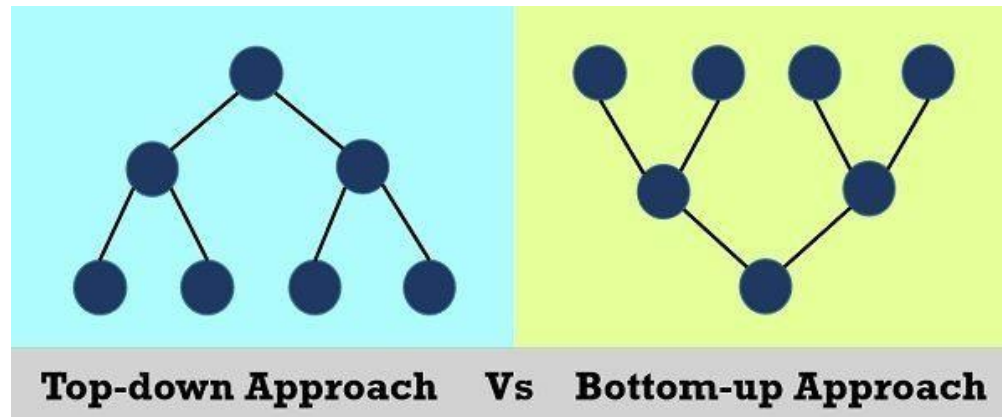


Figure 18 Top-down Approach vs Bottom-up Approach

3.3.4. Software Design Types

1- High-Level Design

High-Level Design (HLD) provides a comprehensive overview of the software development process along with the system architecture, applications, database management, and complete flowchart of the system and navigation. It's a blueprint that consolidates the various steps and modules, their objectives, variable components, results, architecture, and timeline to develop the software. HLD translates a business plan into a software product or service.

The first step of that process is that the system engineer analyzes customer requirements and extracts the software specifications and writes it in a document called SRS. The architect takes the SRS and puts his imagination about the system software from the point view of static design, physical design, and dynamic design.

2- Low-level design

Low-Level Design (LLD) deals with the planning, coding, and execution of the various components, modules, and steps in the HLD, at an individual level. Each module in an HLD has a unique LLD document that provides comprehensive details about how the module will be coded, executed, tested for quality, and integrated into the larger program. LLD provides actionable plans by deconstructing HLD components into working solutions.

S.NO.	Comparison Basis	HLD	LLD
1.	Stands for	It stands for High-Level Design.	It stands for Low-Level Design.
2.	Definition	It is the general system design, which means it signifies the overall system design.	It is like describing high-level design, which means it signifies the procedure of the component-level design.
3.	Purpose	The HLD states the concise functionality of each component.	LLD states the particular efficient logic of the component.
4.	Also, known as	HLD is also called a System or macro-level design.	LLD is also called details or micro-level design.
5.	Developed by	Solution Architect prepares the High-Level Design.	Designer and developer prepare the Low-Level Design.
6.	Sequential order in the design phase	It is developed first in sequential order, which implies that the HLD is created before the LLD.	It is developed after High-level design.
7.	Target Audience	It is used by management, program, and solution teams.	It is used by designers, operation teams, and implementers.

8.	Conversion	The HLD changes the client or business requirement into a high-level solution.	The LLD changes the high-level solution to a comprehensive solution.
9.	Probable output	The high-level design is necessary to understand the flow across several system objects.	A low-level design is required for creating the configurations and troubleshooting inputs.
10.	Input Criteria	The input measure in high-level design is. SRS (Software Requirement Specification).	The input measure in low-level design is the reviewed HLD (High-Level Design).
11.	Output Criteria	The output measures in the HLD are functional design, database design, and review record.	The output bases in the low-level design are the unit test plan and program specification.

3.4. Software design of dashboard

3.4.1. Layered architecture

Layered architectures are said to be the most common and widely used architectural framework in software development. It is also known as an n-tier architecture and describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software. A layer is a logical separation of components or code:

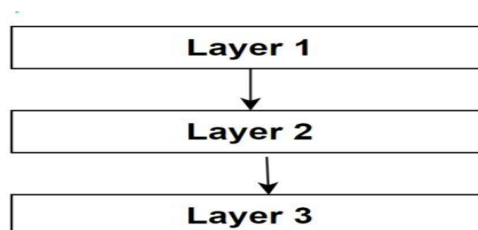


Figure 19 Layered Architecture

In these frameworks, components that are related or that are similar are usually placed on the same layers. However, each layer is different and contributes to a different part of the overall system.

Characteristics

A major characteristic of this framework is that layers are only connected to the layers directly below them. In the illustration given previously, layer 1 only connects to layer 2, layer 2 connects to layer 3, and layer 1 is connected to layer 3 only through layer 2.

Another characteristic is the concept of layers of isolation. This means that layers can be modified, and the change won't affect other layers. In short, changes are isolated to the specific layer that is altered.

Separation of concerns is another notable feature that speaks to how the modules on a single layer together perform a single function.

We followed the Layered architecture method, and our system is displayed in the following figure.

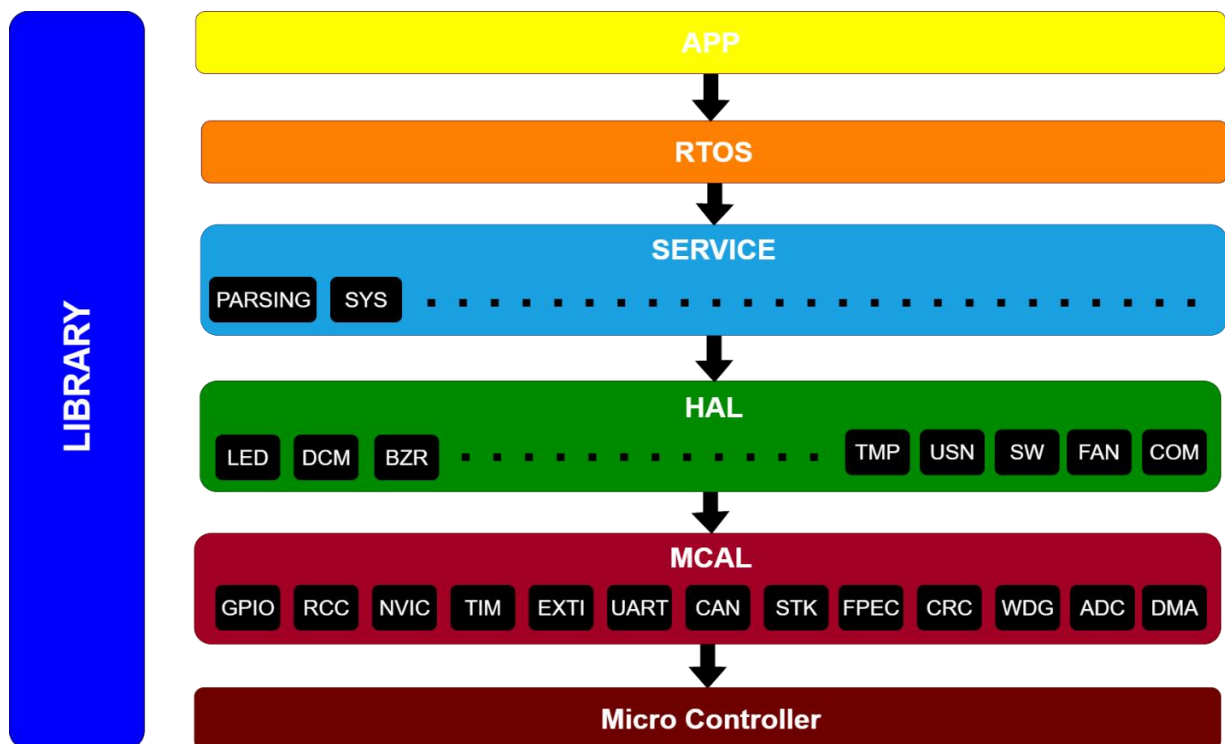


Figure 20 Layered Architecture

Chapter 4: Implementation

4.1. Server Connection

4.1.1. Firebase



Figure 21 Firebase BaaS

Firebase is a Backend-as-a-Service (BaaS). It provides developers with a variety of tools and services to help them develop quality apps, grow their user base, and earn profit. It is built on Google's infrastructure as well as easy to use.

Firebase is categorized as a NoSQL database program, that stores data in JSON-like documents.

(a)Authentication: - Firebase Authentication makes it easy for developers to build secure authentication systems and enhances the sign-in and onboarding experience for users. This feature offers a complete identity solution, supporting email and password accounts, phone authentications, as well as Google, Facebook, GitHub, Twitter login and more.

(b)Real-time database: The Firebase Real-time Database is a cloud-hosted NoSQL database that enables data to be stored and synced between users in real-time. The data is synced across all clients in real-time and is still available when an app goes offline.

(c)Performance: Firebase Performance Monitoring service gives developers insight into the performance characteristics of their iOS and Android apps to help them determine where and when the performance of their apps can be improved.

(d)Storage: **Firebase** allows us to store files like the software update to access them later on and download it.

4.1.2 ESP with Firebase:

One of Firebase's key features is Firebase Cloud Storage, which lets you store arbitrary binary blobs which can be retrieved over the Internet using a simple HTTP request.

A manual way to upload a new binary to Firebase is to visit the Firebase console, click on "Storage" in the left nav bar, and then click on the "Upload file" button, but we upload the file from our website.

Our website is linked with firebase and makes it easy to upload the file without a lot steps, with use ESP we can download the file from Firebase.



Figure 22 ESP with Firebase

We created a Firebase project with a real-time database, and we used the ESP to store and read data from the database. The ESP can interact with the database from anywhere in the world as long as it is connected to the internet.

4.1.3. Website

The FOTA website follows a client-server architecture, where the client-side code runs in the user's web browser, and the server-side code handles the business logic and data management. This section will delve into the different components and technologies used in architecture.

4.1.3.1 Server-Side Components

The server-side of the FOTA website is built using a combination of technologies to handle data storage, server logic, and API interactions. Here are the key server-side components:

- **Backend Framework:** The website utilizes a popular backend framework, such as Express.js, to handle HTTP requests, routing, and

API endpoints. The chosen framework provides a structured and organized approach to building server-side logic.

- **Database:** The FOTA website integrates with a database system, such as MongoDB or MySQL, to store and manage data. The database is used to store user information, firmware versions, update metadata, and other relevant data for the application's functionality.
- **APIs:** The server exposes APIs to enable communication between the client-side and server-side components. These APIs handle user authentication, firmware management, update requests, and other functionalities required by the website.
- **Authentication:** The server implements authentication mechanisms, such as JWT (JSON Web Tokens) or session-based authentication, to ensure secure user access and protect sensitive data. Authentication APIs handle user registration, login, and session management

4.1.3.2 Client-Side Technologies:

The client-side of the FOTA website comprises HTML, CSS, and JavaScript code that runs in the user's web browser. It provides the user interface and handles user interactions. Here are the key client-side technologies:

- **HTML:** The website's structure and layout are defined using HTML (Hypertext Markup Language), which allows the browser to interpret and render the content.
- **CSS:** CSS (Cascading Style Sheets) is used to apply visual styles and formatting to the HTML elements. It ensures a consistent and visually appealing user interface.
- **JavaScript:** JavaScript is responsible for the dynamic behavior of the website. It handles user interactions, makes API calls to the server, updates the UI, and manages client-side validations.
- **Frontend Framework/Library:** The FOTA website may utilize a frontend framework or library, such as React, Angular, or Vue.js, to facilitate efficient UI rendering, component reusability, and state management.
- **Charting Library:** To display temperature and other graphical data, the website may incorporate a charting library like Chart.js or D3.js. These libraries provide easy-to-use APIs for creating interactive and visually appealing charts.

4.1.3.3 Interaction Between Components:

The server-side components and client-side technologies interact to provide the functionality of the FOTA website. Here's an overview of the typical interaction flow:

- **User Interaction:** The user interacts with the website's UI, such as submitting forms, clicking buttons, or navigating between pages.
- **Client-Side Code Execution:** The client-side JavaScript code handles user interactions, performs validations, and prepares data to be sent to the server.
- **API Calls:** The client-side code makes API calls to the server to perform actions such as user authentication, firmware management, update requests, or retrieving data from the database.
- **Server-Side Processing:** The server receives API requests, validates them, performs necessary operations, and interacts with the database to store or retrieve data.
- **Response Handling:** The server sends back a response to the client-side code, which can be success messages, updated data, or error notifications.
- **UI Updates:** The client-side code updates the UI based on the response received from the server, ensuring a seamless user experience.

The architecture of the FOTA website ensures a separation of concerns, scalability, and maintainability.

The server-side components handle data storage, business logic, and API interactions, while the client-side technologies focus on providing a responsive and interactive user interface.

Next, we'll dive into the features of the FOTA website. Let's continue with the next section or inform me if you have any specific points to include in the architecture section.

4.1.3.4 Features:

The FOTA website is designed to provide users with a seamless and convenient way to manage firmware updates for their cars. Here are the key features offered by the website:

- **Firmware Version Management:** Users can easily view and manage the available firmware versions for their cars. The website allows them to add new firmware versions, update existing ones, and delete obsolete versions.
- **Node Selection:** Users can select the target node (ECU) for which they want to perform a firmware update. The website provides a user-friendly interface to choose the specific node among the available options.
- **Version Update:** Once the target node is selected, users can update the firmware version for that particular node. They can specify the desired version number and initiate the update process.

- **Real-Time Temperature Graph:** The website displays a visually appealing and interactive temperature graph. Users can monitor the temperature of various components in real-time, enabling them to detect any anomalies or changes.
- **Audio Player:** The website incorporates an audio player that allows users to play audio files associated with specific firmware versions. This feature enables users to listen to audio instructions or explanations related to the firmware updates.
- **User Authentication:** The FOTA website implements a secure user authentication system. Users can create an account, log in, and access their personalized dashboard to manage firmware updates for their cars.
- **Responsive Design:** The website is built with a responsive design approach, ensuring compatibility and optimal user experience across different devices and screen sizes. Users can access the website seamlessly from desktops, laptops, tablets, or smartphones.
- **User-Friendly Interface:** The FOTA website prioritizes user experience by providing an intuitive and user-friendly interface. Clear navigation, well-organized information, and interactive elements enhance usability and make it easy for users to perform desired actions.

These features work together to empower users in managing firmware updates for their cars. The intuitive interface, real-time temperature monitoring, and audio instructions contribute to a seamless and efficient user experience.

4.1.3.5 Layout and Navigation

The FOTA website follows a responsive layout design, ensuring a seamless experience across different devices and screen sizes. The layout is organized and structured to enable easy navigation and access to the website's features. Here are some key aspects of the UI layout and navigation:

- **Header:** The header section typically contains the website logo, navigation menu, and user authentication options. It remains consistent across different pages, providing easy access to essential functionalities.
- **Sidebar or Menu:** The sidebar or menu provides navigation options for different sections of the website. It may include links to dashboard, firmware management, user profile, and other relevant pages.
- **Content Area:** The content area displays the main content of each page, including features, firmware information, version updates, and other related data. It adapts to the specific page's requirements and may contain tables, forms, charts, or other interactive components.

- Footer: The footer section usually contains additional links, copyright information, and any relevant disclaimers.

4.1.3.6 Key UI Components

The FOTA website utilizes various UI components to enhance the user experience and provide a visually appealing interface. Here are some notable UI components used on the website:

- Forms: Forms are employed for user registration, login, firmware updates, and other data entry tasks. They include input fields, dropdowns, checkboxes, and validation mechanisms to ensure accurate data submission.
- Tables: Tables are utilized to display firmware metadata, version history, and other tabular data. They may include sorting, filtering, and pagination options for easy data exploration.
- Charts and Graphs: Charts and graphs are used to visually represent firmware analytics, temperature trends, and other statistical information. They provide a quick overview and facilitate data-driven decision-making.
- Modals and Dialogs: Modals and dialogs are employed for displaying additional information, confirmation prompts, or success/error messages. They enhance the user experience by providing context-specific interactions without navigating away from the current page.

Visual Design:

The visual design of the FOTA website adheres to a specific style and branding guidelines to create a consistent and visually appealing user interface. Here are some aspects of the visual design:

- Color Scheme: The website incorporates a carefully chosen color scheme that aligns with the brand identity. The colors are used consistently throughout the UI to create a cohesive visual experience.
- Typography: Typography selection ensures readability and legibility of the website's content. Fonts and font sizes are chosen to provide a comfortable reading experience.
- Icons and Imagery: Icons and imagery are utilized to convey information, represent actions, and enhance the visual appeal of the website. They are used strategically to assist with user comprehension and navigation.
- Whitespace and Layout: The use of whitespace and layout principles ensures content clarity and avoids visual clutter. Proper spacing, alignment, and grid systems contribute to an organized and balanced UI.

The user interface of the FOTA website aims to provide a seamless and engaging experience for its users. By incorporating intuitive navigation, key UI components, and visually appealing design, the website strives to enhance user interactions and facilitate efficient usage.

4.1.3.7 Database Integration

The FOTA website relies on a database system to store and manage critical data related to firmware updates, version history, user profiles, and other relevant information. This section provides an overview of the database integration on the website, including the database technology used, data structure, and key functionalities.

Database Technology

The FOTA website utilizes Firebase Realtime Database as database technology. Firebase Realtime Database is a cloud-hosted NoSQL database that allows real-time synchronization of data between clients and servers. It provides a scalable and reliable solution for storing and retrieving data in real-time.

Data Structure

The data in the Firebase Realtime Database is organized in a hierarchical structure, consisting of key-value pairs. Here is an overview of the data structure employed in the FOTA website:

- **Nodes:** Each ECU (Electronic Control Unit) is represented as a node in the database. Nodes store information such as the ECU ID, firmware version, and other relevant metadata.
- **Firmware Updates:** Firmware update data includes details about the firmware package, version number, release date, and associated ECUs.
- **User Profiles:** User profiles contain information about registered users, including username, email, and other relevant user-specific data.

Database Security

Security measures are implemented to protect the data stored in the Firebase Realtime Database. These measures include:

- **Authentication and Authorization:** User authentication and authorization mechanisms are employed to ensure that only authorized users can access and modify the database.
- **Data Validation and Sanitization:** Input data is validated and sanitized to prevent unauthorized access, data corruption, or injection attacks.

- **Access Control Rules:** Firebase Realtime Database allows the definition of access control rules to restrict reading and write access to specific data based on user roles and permissions.

The database integration in the FOTA website plays a crucial role in storing, managing, and retrieving firmware-related information, user profiles, and version control data. By leveraging Firebase Realtime Database, the website ensures real-time synchronization, data consistency, and secure storage of critical information.

Next, we will cover the Deployment section or let me know if there's anything specific, you'd like to include in this section.

4.1.3.8 Deployment Process

The deployment process involves the following steps:

- **Build:** Before deploying the website, the source code is typically built to generate optimized and minified production-ready assets. This step may involve running build scripts, bundling the JavaScript files, optimizing assets, and performing other necessary build tasks.
- **Firebase Project Setup:** To deploy the website using Firebase Hosting, a Firebase project needs to be set up. This involves creating a new project on the Firebase console, configuring project settings, and linking the project with the website's source code.
- **Firebase CLI:** The Firebase command-line interface (CLI) is used to deploy the website to Firebase Hosting. The CLI provides a set of commands to interact with Firebase services and manage deployments.
- **Deployment Configuration:** The deployment configuration file, such as `firebase.json`, is updated to specify the settings and configurations for Firebase Hosting. This file includes details such as the target hosting project, public directory, routing rules, and other relevant settings.
- **Deploy:** With the Firebase CLI and the updated deployment configuration, the website is deployed to Firebase Hosting using the command `firebase deploy`. This command triggers the deployment process, uploading the website's assets to Firebase Hosting servers.
- **Domain Configuration:** After the deployment, the website can be accessed using the default domain provided by Firebase Hosting (e.g., `your-projectid.firebaseio.com`). Optionally, a custom domain can be configured to map the website to a specific domain name.

4.1.3.9 Continuous Deployment

To streamline the deployment process and ensure that the website stays up to date, a continuous deployment approach can be adopted. Continuous deployment automates the deployment process by triggering deployments whenever new code changes are pushed to the source code repository.

This can be achieved by integrating a continuous integration and deployment (CI/CD) system such as GitHub Actions, Travis CI, or CircleCI. These systems can be configured to monitor the source code repository, run build processes, and automatically deploy the updated website to Firebase Hosting.

By implementing continuous deployment, the FOTA website can benefit from automated deployments, faster release cycles, and improved efficiency in delivering updates to users.

Monitoring and Maintenance

Once the website is deployed, it is crucial to monitor its performance and address any issues that arise. Monitoring tools, such as Firebase Performance Monitoring or third-party solutions, can be employed to track website metrics, identify performance bottlenecks, and ensure optimal user experience.

Regular maintenance tasks, such as updating dependencies, applying security patches, and monitoring server health, should be performed to keep the website secure and up to date. Additionally, logging and error tracking systems can be integrated to capture and analyze any errors or exceptions occurring within the website, enabling quick identification and resolution of issues.

By properly monitoring and maintaining the deployed FOTA website, it can provide a seamless and reliable experience to users, ensuring that the firmware update process is efficient and effective.

4.1.3.10 Testing

The testing section focuses on the different types of testing performed on the FOTA website to ensure its functionality, reliability, and performance. Testing plays a crucial role in identifying and resolving issues before the website is deployed to production.

Test Strategy

The test strategy for the FOTA website encompasses various levels and types of testing. The goal is to cover all critical components and functionalities of the website to ensure a high-quality user experience. The following types of testing are included:

- **Unit Testing:** Unit tests are performed on individual units or components of the website to verify their functionality in isolation. This includes testing functions, modules, and components using testing frameworks such as Jest or React Testing Library. Unit tests help identify and fix bugs at an early stage and ensure that individual components work as expected.
- **Integration Testing:** Integration tests focus on testing the interaction between different components or modules of the website. This ensures that the integrated system functions correctly and that the components work together as intended. Integration testing is performed using frameworks such as Jest, Cypress, or Selenium.
- **End-to-End (E2E) Testing:** E2E testing simulates real user scenarios and tests the entire website from start to finish. It covers multiple components, user interactions, and data flows. E2E testing tools like Cypress, Puppeteer, or Selenium WebDriver are used to automate the tests and ensure that the website behaves as expected in different browsers and environments.
- **Performance Testing:** Performance testing evaluates the website's performance under different conditions, such as high user load or heavy data processing. Tools like Lighthouse, Webpage Test, or JMeter can be used to measure key performance metrics such as page load time, response time, and resource utilization. Performance testing helps identify performance bottlenecks and optimize the website for better user experience.

Test Coverage

The test coverage aims to ensure that critical functionalities and scenarios are thoroughly tested. The following areas are covered in the testing process:

- **Authentication and Authorization:** Testing user authentication and authorization workflows, ensuring secure access to the website and its features. This includes verifying login, registration, password reset, and role-based access control.
- **Firmware Update:** Testing the firmware update process, including uploading firmware files, validating compatibility, and ensuring successful installation on target nodes. This involves simulating different scenarios, such as valid and invalid firmware files, network disruptions, and error handling.

- **User Interface (UI) Testing:** Validating the user interface components, layouts, and interactions. UI testing ensures that the website is visually appealing, responsive, and user-friendly across different devices and browsers. It involves checking the alignment of elements, button functionalities, form validations, and overall user experience.
- **Error Handling and Exception Testing:** Verifying that the website handles errors and exceptions gracefully. This includes testing error messages, error logging, and appropriate handling of unexpected situations. It ensures that users receive meaningful error messages, and the website maintains stability even in error scenarios.

Test Automation

To streamline the testing process and improve efficiency, test automation is employed. Automation frameworks and tools, such as Jest, Cypress, or Selenium, are used to automate repetitive test cases, reducing manual effort, and increasing test coverage. Automated tests can be integrated into the CI/CD pipeline to run automatically with each code change.

Test automation allows for quicker feedback on code changes, faster regression testing, and better overall test coverage. It helps ensure that critical functionalities are consistently validated and reduces the likelihood of human errors.

Test Data and Environment

Test data and environments play a crucial role in ensuring comprehensive testing. The following considerations are considered:

- **Test Data:** Test data is carefully prepared to cover various scenarios, including valid and invalid inputs, edge cases, and boundary conditions. It ensures that the website behaves correctly in different data scenarios and handles all possible data variations.
- **Test Environments:** Multiple test environments are set up to replicate the production environment as closely as possible. This includes creating staging environments, simulating different network conditions, and provisioning test databases. Test environments help identify and fix issues specific to different environments and configurations.

4.1.3.10 User Documentation

The User Documentation section provides instructions and guidelines for users to effectively use and navigate the FOTA website. It aims to empower users with the knowledge and

understanding they need to make the most out of the website's features and functionalities. The user

documentation covers the following areas:

1. Getting Started

This section provides an overview of the FOTA website and its purpose. It includes information on how to access the website, whether it is through a web browser or a dedicated application. Users will find instructions on creating an account, logging in, and accessing their personalized dashboard.

2. Dashboard

The Dashboard section explains the main interface of the website. Users will learn about the various components and widgets available on the dashboard, such as the temperature graph, clock, and volume player. Instructions on how to interpret and interact with these components will be provided.

3. Firmware Update

The Firmware Update section guides users through the process of updating their car's firmware using the FOTA website. It includes step-by-step instructions on how to upload a firmware file, select the target ECU, and initiate the update process. Users will also find information on monitoring the update progress and handling any potential issues that may arise.

4. Personalization Options

This section covers the personalization options available to users. It explains how to customize settings, such as the selected ECU, version number, and volume level. Users will learn how to modify these settings to suit their preferences and requirements.

5. Troubleshooting

The Troubleshooting section provides users with troubleshooting tips and solutions for common issues they may encounter while using the FOTA website. It covers topics such as troubleshooting firmware update failures, resolving connectivity issues, and handling errors or unexpected behaviors.

6. Frequently Asked Questions (FAQ)

The FAQ section addresses frequently asked questions and provides answers to common queries from users. It covers topics such as compatibility requirements, security measures, data privacy, and general usage guidelines. The FAQ section aims to address common concerns and provide users with quick answers to their questions.

7. Support and Contact Information

The Support and Contact Information section provides users with details on how to seek further assistance or support if they encounter any issues or need additional help. Users will find information on how to contact the support team, including email addresses, phone numbers, and support hours. The User Documentation aims to provide users with a comprehensive guide to navigate and utilize the FOTA website effectively. It empowers users to make informed decisions and ensures a smooth and hassle-free user experience.

4.1.3.12 Troubleshooting

The Troubleshooting section provides users with guidance on resolving common issues they may encounter while using the FOTA website. If you experience any difficulties or unexpected behaviors, refer to the following troubleshooting tips and solutions:

1. Firmware Update Failure

If a firmware update fails, follow these steps to troubleshoot the issue:

- Check the firmware file: Ensure that the firmware file you uploaded is compatible with your car's ECU and meets the required format and specifications.
- Verify the internet connection: Make sure you have a stable and reliable internet connection during the firmware update process. Unstable connections or network disruptions can lead to update failures.
- Retry the update: In some cases, the initial failure may be due to a temporary glitch. Retry the firmware update process to see if it succeeds on subsequent attempts.
- Contact support: If the issue persists, reach out to the support team for further assistance. Provide details about the specific error message or behavior you encountered to help them diagnose the problem.

2. Connectivity Issues

If you experience connectivity issues while using the FOTA website, try the following troubleshooting steps:

- Check your internet connection: Verify that you have a stable and active internet connection. Test your connection by visiting other websites or using other online services to ensure it is not a general network issue.
- Clear browser cache and cookies: Sometimes, cached data or cookies can interfere with website functionality. Clear your browser cache and cookies and then reload the FOTA website to see if the issue is resolved.

- Try a different browser or device: If the issue persists, try accessing the website from a different browser or device. This can help determine if the problem is specific to your current setup.
- Disable browser extensions: Some browser extensions or plugins may conflict with the FOTA website. Disable any extensions temporarily and check if the issue is resolved.
- Contact your internet service provider (ISP): If the problem persists and you have ruled out other potential causes, contact your ISP to ensure there are no connectivity issues on their end.

3. Error Messages and Unexpected Behaviors

If you encounter error messages or unexpected behaviors on the FOTA website, consider the following troubleshooting steps:

- Read the error message: Carefully read and understand the error message displayed on the screen. Error messages often provide useful information about the issue at hand.
- Refresh the page: Sometimes, a simple page refresh can resolve minor glitches or temporary issues. Try refreshing the page and see if the error or unexpected behavior persists.
- Log out and log in again: If the issue seems related to your user session, log out of the website, and then log in again. This can help reset your session and resolve any session related issues.
- Contact support: If the error message persists or the unexpected behavior continues, contact the support team. Provide them with as much detail as possible, including the specific error message, steps to reproduce the issue, and any relevant screenshots or logs.

Remember, if you encounter any issues or need further assistance, do not hesitate to reach out to the FOTA website support team. They are there to help and provide you with the necessary guidance to resolve any problems you may encounter while using the website.

4.1.3.13 Implementation Details

The FOTA website is built using a combination of front-end and back-end technologies to create a robust and user-friendly web application. Here are the key implementation details:

Front-end Technologies:

- HTML5: The structure and content of the web pages are created using HTML5, the latest version of the Hypertext Markup Language.

- **CSS3:** CSS3 is used for styling and customizing the visual appearance of the website, including layout, colors, typography, and animations.
- **JavaScript:** JavaScript is used to add interactivity and dynamic functionality to the website. It enables features such as real-time temperature graph updates, audio player control, and form validation.
- **React:** The FOTA website is developed using the React JavaScript library, which allows for building reusable UI components and managing the application's state efficiently.
- **React Router:** React Router is used for client-side routing, enabling navigation between different pages of the website without a full page reload.
- **Chart.js:** Chart.js is utilized to create visually appealing and interactive temperature graphs. It provides the necessary tools to render and update the graph based on real-time temperature data.

Back-end Technologies:

- **Firebase:** Firebase is a comprehensive platform that offers a range of cloud-based services. The FOTA website leverages Firebase for features such as user authentication, real-time database management, and cloud storage for audio files.
- **Firebase Authentication:** Firebase Authentication is used to handle user registration, login, and authentication. It provides secure user management and authentication functionality out of the box.
- **Firebase Realtime Database:** Firebase Realtime Database is utilized for storing and retrieving firmware versions, temperature data, and user-related information. It enables real-time data synchronization between the client and the server.
- **Firebase Cloud Storage:** Firebase Cloud Storage is employed to store and serve audio files associated with firmware versions. It provides secure and scalable storage for audio assets.

Additional Tools and Libraries:

- **React Router:** React Router is used for client-side routing, enabling navigation between different pages of the website without a full page reload.
- **React Router DOM:** React Router DOM is a library that provides routing functionality specific to web applications built with React.
- **react-chartjs-2:** The react-chartjs-2 library is used as a React wrapper for Chart.js. It simplifies the integration of Chart.js with React components.
- **axios:** Axios is utilized for making HTTP requests from the client-side code to interact with the server and retrieve or update data.

These technologies and tools work together to create a robust and efficient web application for firmware updates. The front-end technologies provide a seamless user interface, while the back-end technologies handle data storage, user authentication, and real-time updates.

4.1.3.14 Architectural Overview

The FOTA website follows a client-server architecture, where the client-side code is responsible for rendering the user interface and handling user interactions, while the server-side code manages data storage, authentication, and other server-side functionalities. Here's an overview of the different components and how they interact with each other:

- Client-Side Components.
- React Components: The FOTA website is built using React components, which are reusable UI elements responsible for rendering different parts of the user interface.
- React components are organized in a component hierarchy and are dynamically updated based on user interactions and data changes.
- Routing: The React Router library is used for client-side routing, allowing users to navigate between different pages without a full page reload. The routing configuration maps specific URLs to corresponding React components to render the appropriate content.
- State Management: React's built-in state management and hooks system is utilized to manage and update the application's state. State variables are used to store and manipulate data, such as selected ECU, version numbers, and form inputs.
- Server-Side Components.
- Firebase Authentication: Firebase Authentication handles user registration, login, and authentication. It provides secure user management and authentication functionality, ensuring that only authorized users can access the FOTA website.
- Firebase Realtime Database: Firebase Realtime Database is used to store and retrieve data related to firmware versions, temperature records, and user information. It provides real-time synchronization, allowing users to see updates in real-time without the need for manual refresh.
- Firebase Cloud Storage: Firebase Cloud Storage is utilized to store and serve audio files associated with firmware versions. It provides secure and scalable storage for audio assets, ensuring they can be accessed by authorized users.
- API Endpoints: The server-side code exposes API endpoints to handle requests from the client-side code. These endpoints are responsible for updating firmware versions, retrieving temperature data, and other operations related to the FOTA functionality.
- Communication and Data Flow.

- The client-side code communicates with the server-side components through HTTP requests. Axios is used as the HTTP client library to make requests to the server and retrieve or update data.
- User interactions, such as selecting an ECU, entering version numbers, and submitting forms, trigger events that update the application's state and initiate HTTP requests to the server.
- The server-side components handle these requests, perform the necessary operations, and update the database accordingly. Real-time updates are propagated back to the client-side code, triggering re-rendering of the UI components with the updated data.

The client-server architecture ensures separation of concerns, with the client-side code responsible for providing a smooth user experience and the server-side code handling data storage, authentication, and other server-side operations. This architecture enables scalability, maintainability, and extensibility of the FOTA website.

Key Features and Functionalities

The FOTA website offers several key features and functionalities that enhance the user experience and facilitate the firmware over-the-air update process. Here are the main features of the website:

User Registration and Authentication:

- Users can create an account and register on the FOTA website using their email and password. Firebase Authentication handles the registration and login process, ensuring secure access to the website.
- Authentication features include email verification and password reset functionality to enhance security and account recovery.
- **Dashboard and Firmware Updates**
- The website provides a user-friendly dashboard that displays information about different Electronic Control Units (ECUs) in the car and their corresponding firmware versions.
- Users can select a target ECU and enter a new firmware version number to initiate a firmware update process. The updated firmware version is stored in the Firebase Realtime Database for real-time synchronization.
- Firmware updates trigger events that are propagated to the relevant ECUs, allowing them to fetch and install the latest firmware version over-the-air.

Temperature Monitoring:

The FOTA website includes a temperature monitoring feature that displays temperature records for different ECUs in a graphical format. Temperature data is fetched from the Firebase Realtime Database and visualized using interactive charts. Users can view historical temperature trends and analyze the data to identify patterns or anomalies.

Audio Player Integration:

The website incorporates an audio player component that allows users to upload and play audio files associated with firmware versions.

Users can choose an audio file from their local system, and the selected file is stored in Firebase Cloud Storage. The audio player component fetches the audio file from the storage and provides playback controls for users to listen to the associated audio.

Responsive Design and User Experience:

- The FOTA website is designed to be responsive, ensuring a seamless user experience across different devices and screen sizes.
- The user interface is intuitive and user-friendly, with clear navigation, form validations, and informative feedback to guide users throughout the firmware update process. Styling and theming are applied to maintain a consistent and visually appealing look that aligns with the website's personality.

The combination of these features enables users to securely manage firmware updates, monitor temperature data, and access associated audio files. The website provides an efficient and user-friendly interface for managing FOTA processes in the context of car firmware updates.

4.2. Advanced Encryption Standard (AES)

4.2.1. Introduction

At FOTA, data is transmitted over air, network, and always susceptible to theft and exploration of what is inside. Did you remember our application? We transmit code files, very sensitive data. It can be stolen or edited and retransmitted. In case of edit may cause crisis. Imagine the scenario of code editing and retransmission at industrial environment, same thing and may kill people. Therefore, we are going to create our own encryption code using AES Algorithm in C.

4.2.2. Encryption Algorithms

Is a security technique used to protect data and communications by converting plaintext information into ciphertext using mathematical algorithms. Nobody can read or reuse ciphertext until convert it into plaintext. The receiver can easily convert this into plaintext by doing decryption process when anyone other does attack for conversion and it isn't easy depending on the strength of algorithm. Is used to ensure data confidentiality, integrity, and privacy.

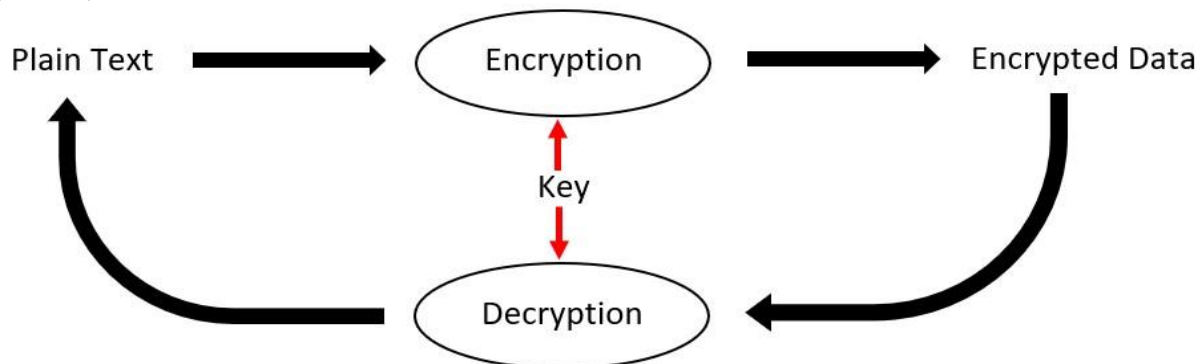


Figure 23 Encryption Life Cycle

4.2.3 Algorithm Characteristics:

- **Symmetric-Key Encryption:** is a symmetric-key encryption algorithm, which means it uses the same secret key for both encryption and decryption of data. This simplicity allows for faster processing and is suitable for applications that require high-speed encryption and decryption.
- **Block Cipher:** is a block cipher, which means it processes data in fixed-size blocks. The block size for is 128 bits.
- **Key Length Options/Fixable key length** supports three key lengths: 128 bits, 192 bits, and 256 bits. The key length directly affects the strength of the encryption. This key length contributes against brute force attack.
- **Security:** is designed to be highly secure and resistant to various cryptographic attacks, including brute-force attacks, differential cryptanalysis, and linear cryptanalysis.
- **Substitution-Permutation Network:** employs a Substitution-Permutation Network (SPN) structure, which involves multiple rounds of substitution and permutation operations on the data. These rounds contribute to the confusion and diffusion properties, increasing its resistance against attacks.
- **Performance:** is designed to be efficient and fast on a wide range of devices, including both software and hardware implementations.

You know, we have a problem:

We must design an algorithm which is suitable for our application. We have small memory and small storage for decryption on the node, we decrypt using STM32 Microcontroller.

4.2.4. Encryption process

Here, we describe the encryption process and decryption will be in the reverse direction.

Here we have a box called a Round and is repeated box based on the key length and because we have 128-bit original key length, the number of rounds is 10. Each round has its own key derived from Original Key. It's easy to develop code to receive 256 original key length. In the last round we skip MixColumns () and do only AddRoundKey () then generate block of output. Back again to file and generate another block of plain. You must know that we take only a block of plain and decrypt it separated than file, then out it to new file, then back and take another block.

The black draw tells you how encryption applied in our application, we encrypt at PC and decrypt at node, at Microcontroller.

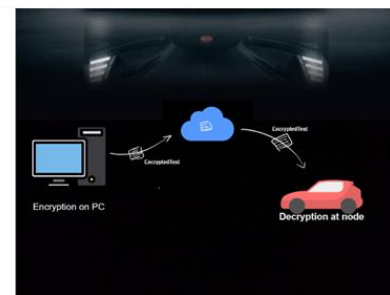
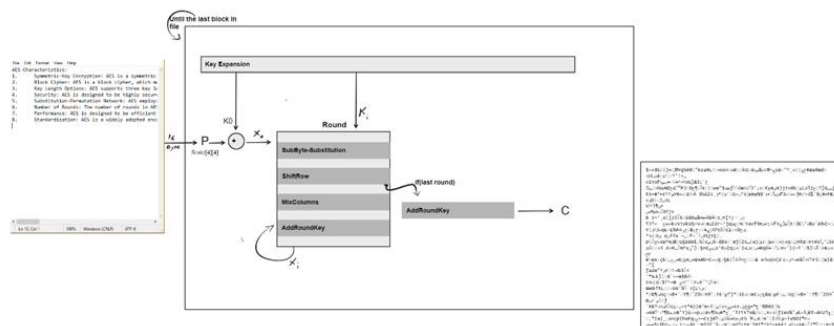


Figure 24 Encryption process

- **Key Expansion**

It takes the original encryption key and generates a set of round keys, which are used in subsequent rounds of the AES encryption algorithm.

1. SubByte (Byte substitution): an S-box substitution step.

- Defines a 16 * 16 matrix of byte values, called an S-box that contains a permutation of all possible 256 input values. (Hint) As we know from ASCII, we have 8 bits to represent the keyboard input, so we have 2 power 8 available values.
- Each individual byte of State, plain, is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value to select a unique 8-bit output value. Easy process, TRYIT.



Figure 25 SubByte

2. ShiftRows: a Permutation step.

The bytes in each row of the state matrix are shifted to the left. The number of positions each byte is shifted depends on its row number in the matrix. First row (Row0) isn't shifted.

Row1 shift by 1, Row2 shift by 2, and Row3 shift by 3. Easy process, TRYIT.

3. MixColumns: a Matrix multiplication step.

- It is designed to provide further diffusion and confusion, enhancing the overall security of the cipher. Each column of the state array is processed separately to produce a new column. The new column replaces the old one. The process is called Matrix Multiplication.
- In the MixColumns process, each column of the State matrix is treated as a polynomial, and a mathematical transformation called "polynomial multiplication" is performed on each column. Polynomial multiplication is done modulo a fixed irreducible polynomial in the Galois Field.

- This process helps to prevent patterns and correlations in the data from persisting through multiple rounds of encryption. It increases the resistance of the cipher against various cryptanalytic attacks.

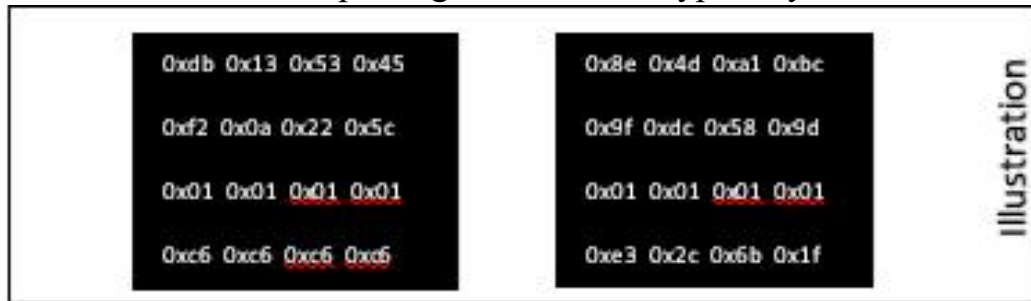


Figure 26 MixColumns

4. AddRoundKey: an XOR step with a round key, K_i .
The 128 bits of State are bitwise XORed with the 128 bits of the round key.

4.2.5. Decryption process

In the previous section at encryption, the input is plaintext, clear text. At decryption the input is ciphertext and we go in reverse operation. The process aims to recover the original plaintext from the ciphertext using the same encryption key that was used during the encryption process. S-box be RS-box, ShiftRow be InverseShiftRow, and MixColumns be InverseMixColumns.

1. KeyExpansion: The decryption process starts by expanding the original encryption key into a set of round keys. These round keys are used in reverse order during decryption, beginning with the last round key used during encryption.
2. AddRoundKey(Last Round): The first decryption step is to apply the AddRoundKey operation using the last round key used during encryption. This step is identical to the AddRoundKey step in encryption.
3. Inverse ShiftRows: The Inverse ShiftRows step is applied to the State matrix (cipher), reversing the shifting of rows performed during the encryption process. This step moves the bytes in each row to the right instead of left.
4. Inverse SubBytes: The Inverse SubBytes step involves replacing each byte in the State matrix with its original value from the Substitution Box (S-box). This step reverses the substitution applied during encryption.
5. Rounds: In each round, the following steps are performed:

- Inverse MixColumns: The Inverse MixColumns step is applied to the State matrix, undoing the mixing done during encryption. It involves polynomial multiplication in the Galois Field.
 - Inverse ShiftRows: The Inverse ShiftRows step is applied again to undo the shifting of rows performed in the corresponding encryption round.
 - Inverse SubBytes: The Inverse SubBytes step is performed once more to revert the substitution of bytes from the Substitution Box.
 - AddRoundKey: In each round, the State matrix is XORed with the corresponding round key in reverse order to undo the key mixing done during encryption.
6. Final Round (Inverse AddRoundKey): The last decryption round involves applying the AddRoundKey operation using the first-round key used during encryption. This undoes the initial AddRoundKey operation in the first encryption round.

4.6.2 Code flow chart

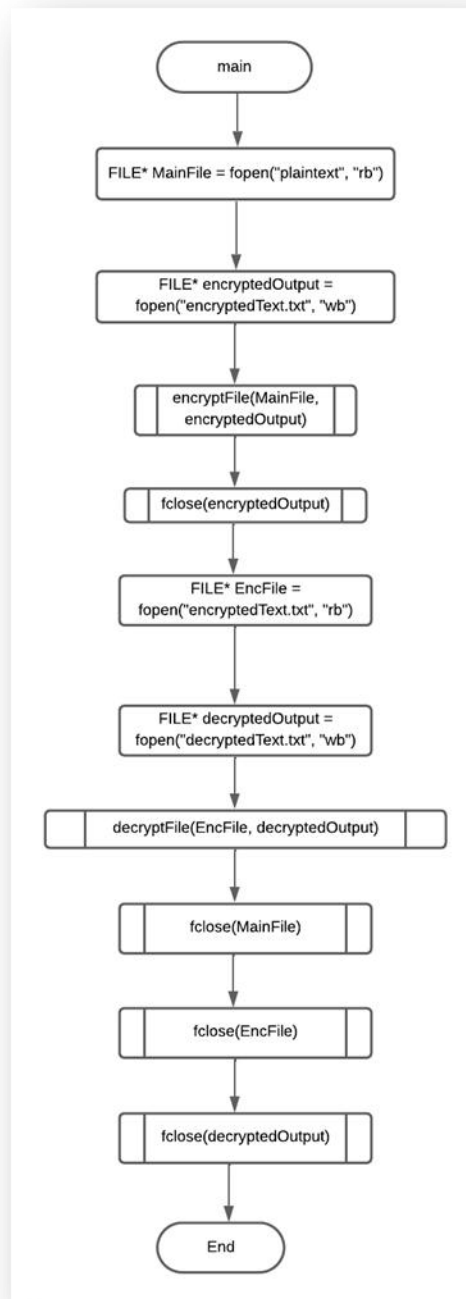


Figure 27 Main Function

Chapter4: Implementation

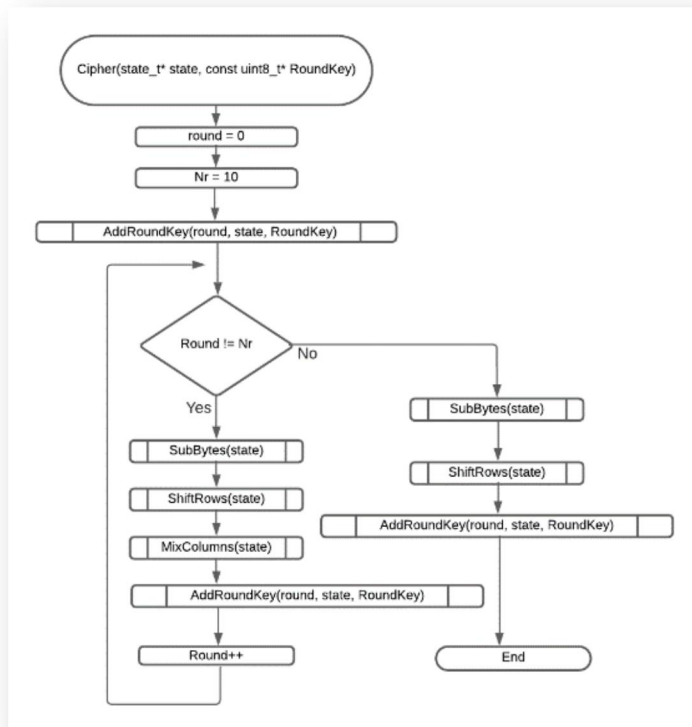


Figure 29 Cipher Function

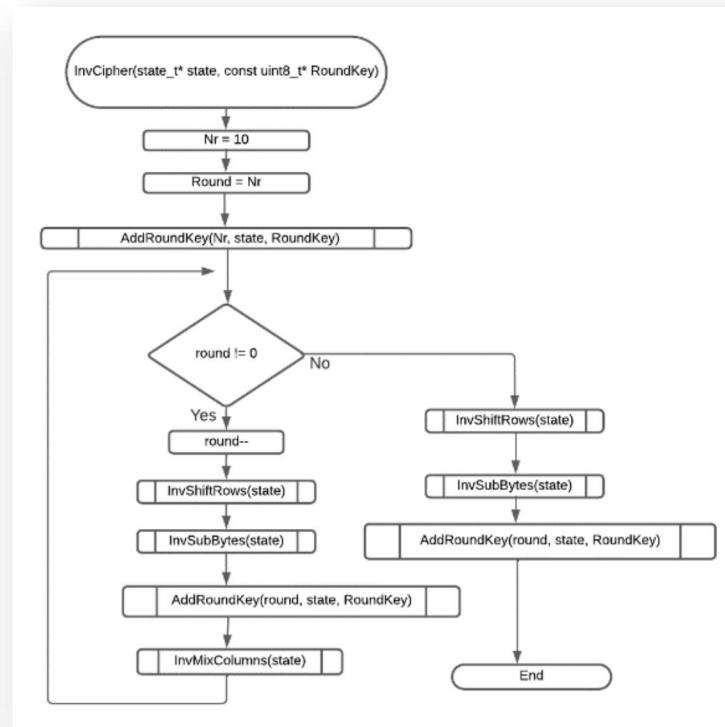


Figure 28 Inverse Cipher Function

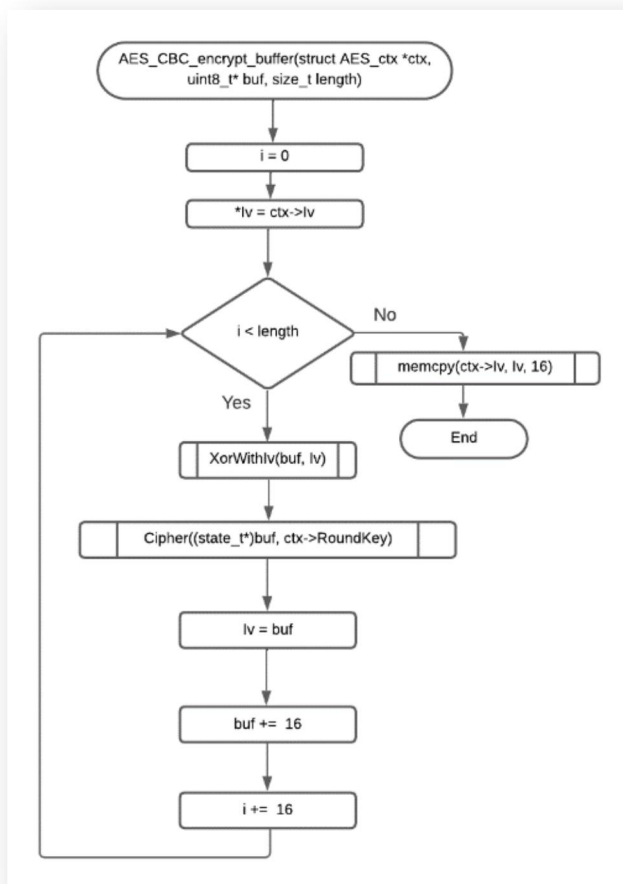


Figure 30 AES_CBC_encrypt_buffer Function

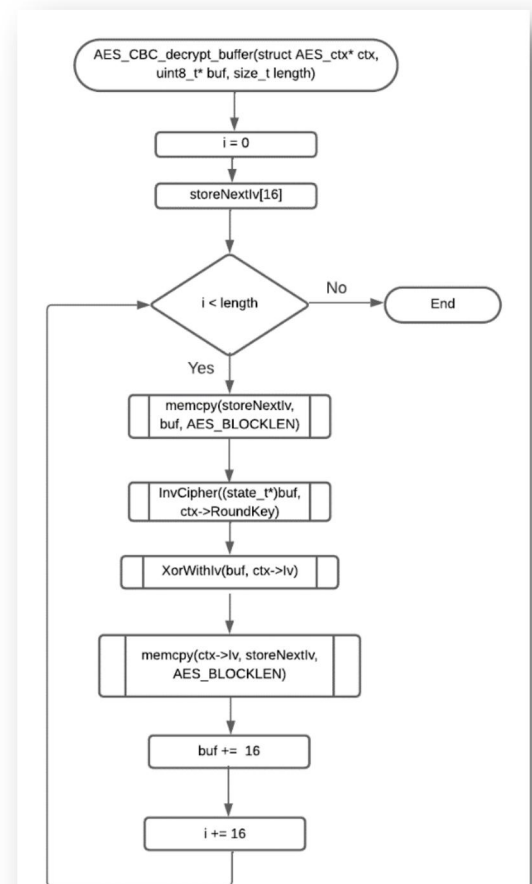


Figure 31 AES_CBC_decrypt_buffer Function

Chapter4: Implementation

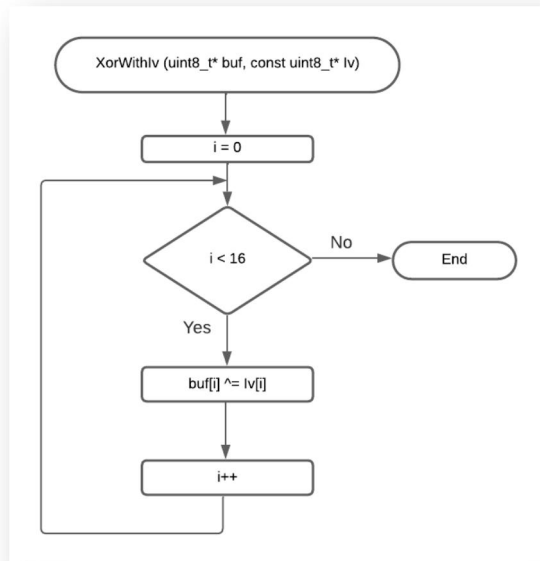


Figure 32 XorWithIv Function

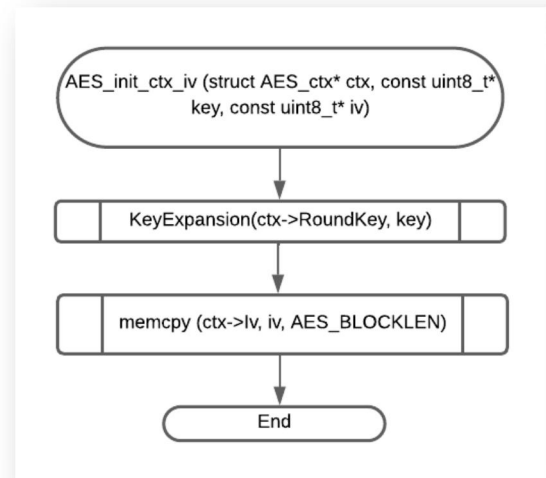


Figure 33 AES_init_ctx_iv Function

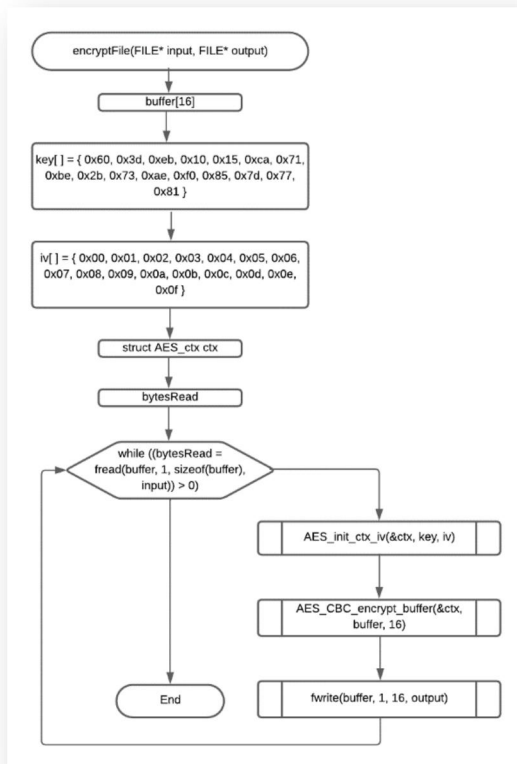


Figure 35 encryptFile Function

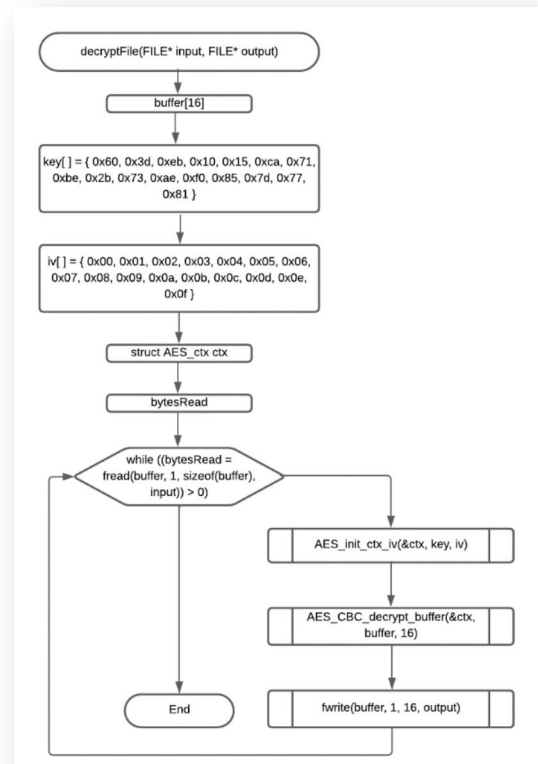


Figure 34 decryptFile Function

4.3. CAN Network

4.3.1. Introduction

Before the year 1985, the wiring of the automotive system was very hard, complicated, and expensive due to the high number of ECUs in the vehicle. So, the need for a system more simply was necessary, so Bosch originally developed CAN at that time, a high-integrity serial bus system for networking intelligent devices, which replaced automotive point-to-point wiring systems.

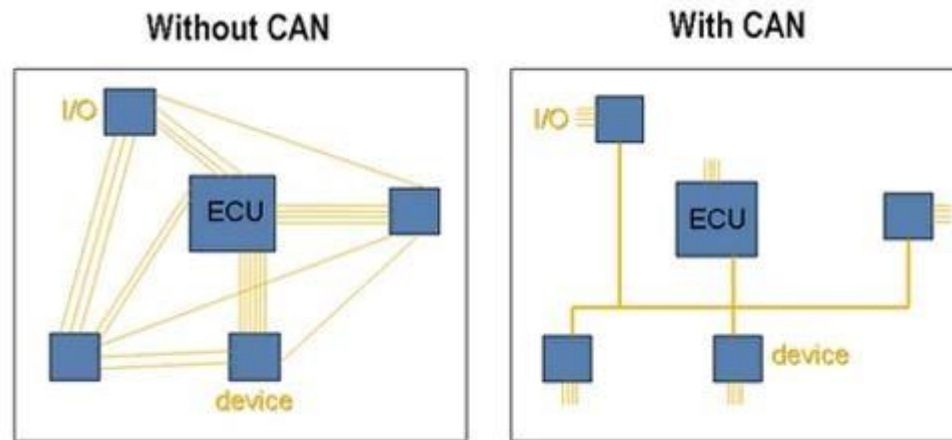


Figure 36 CAN networks significantly reduce wiring.

In 1993, the CAN emerged as the standard in-vehicle network due to its energetic features for safety and it became the international standard known as ISO11898.

In 1996 the On-Board Diagnostics OBD-II standard which incorporates CAN becomes mandatory for all cars and light trucks sold in the United States.

4.3.2. What is CAN?

CAN is short for 'controller area network'. A controller area network is an electronic communication bus defined by the ISO 11898 standards. Those standards define how communication happens, how the wiring is configured, and how messages are constructed, among other things. Collectively, this system is referred to as a CAN bus.

4.3.3. Features of CAN protocol

- Low cost: Since a CAN serial bus uses two wires (with high-volume and low-cost production), it offers a good price-to-performance ratio.
- Bitrate: the bit rate is uniform and fixed for all the nodes and the speed of the CAN may be different for different networks available in a system.
- System flexibility: it is easy for engineers to integrate new electronic devices into the CAN bus network without significant programming overhead and supports a modular system that is easily modified to suit your specs or requirements.
- Message routing: every message is identified by a special unique identifier that does not indicate the destination of the message but only describes the meaning of the data available in the message. So that all the nodes connected in the network can decide by message filtering technology using this message ID whether to receive the data or not.
- Multi-master communication: Any node can access the bus.
- Multicast: more than one node/ECU in the network is able to receive the same transmitted message.
- Arbitration: If two or more nodes start transmitting messages at the same time, the bus access conflict is resolved by bit-wise arbitration using the Identifier. The mechanism of arbitration guarantees that neither information nor time is lost. If a data frame and a Remote frame with the same Identifier are initiated at the same time, the Data frame prevails over the Remote frame.
- Priorities: every message has a priority, so if two nodes try to send messages simultaneously, the one with the higher priority gets transmitted and the one with the lower priority gets postponed. This arbitration is non-destructive and results in the non-interrupted transmission of the highest priority message.
- Error Capabilities: the CAN specification includes a Cyclic Redundancy Code (CRC) to perform error checking on each frame's contents. Frames with errors are disregarded by all nodes, and an error frame can be transmitted to signal the error to the network. Global and local errors are differentiated by the controller, and if too many errors are detected, individual nodes can stop transmitting errors or disconnect themselves from the network completely.

4.3.4. Can Network Message Format

CAN devices send data across the CAN network in packets called frames. These frames can be differentiated based on identifier fields. A CAN frame with 11bit identifier fields is called Standard CAN and with a 29-bit identifier, fields called the extended frame.

1. Standard frame:

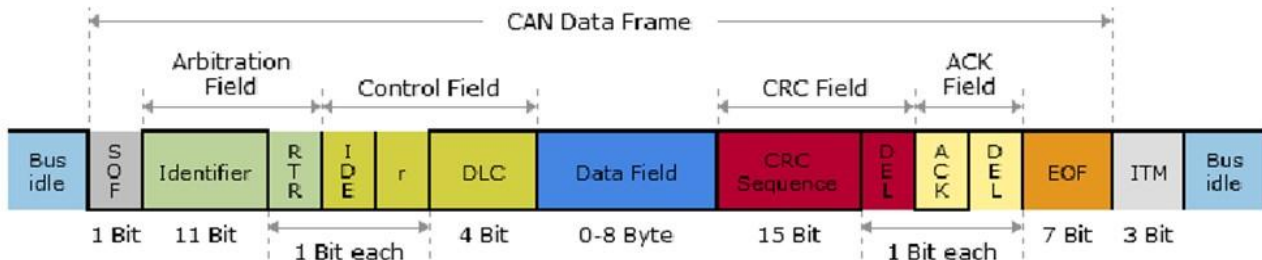


Figure 37 The standard CAN frame format

SOF (start-of-frame) bit – indicates the beginning of a message with a dominant (logic 0) bit.

- **Arbitration ID** – identifies the message and indicates the message's priority. Frames come in two formats -- standard, which uses an 11-bit arbitration ID, and extended, which uses a 29-bit arbitration ID.
- **RTR (remote transmission request) bit** – serves to differentiate a remote frame from a data frame. A dominant (logic 0) RTR bit indicates a data frame. A recessive (logic 1) RTR bit indicates a remote frame.
- **IDE (identifier extension) bit** – allows differentiation between standard and extended frames.
- **R0** – Reversed bit. Not used currently and kept for future use.
- **DLC (data length code)** – indicates the number of bytes the data field contains.
- **Data Field** – contains 0 to 8 bytes of data.
- **CRC (cyclic redundancy check)** – contains a 15-bit cyclic redundancy check code and a recessive delimiter bit. The CRC field is used for error detection.
- **ACK (Acknowledgement) slot** – It comprises the ACK slot and the ACK delimiter. When the data is received correctly the recessive bit in the ACK slot is overwritten as a dominant bit by the receiver.

- **EOF (End of Frame)** – the 7-bit field marks the end of a CAN frame (message) and disables.

2. Extended frame:

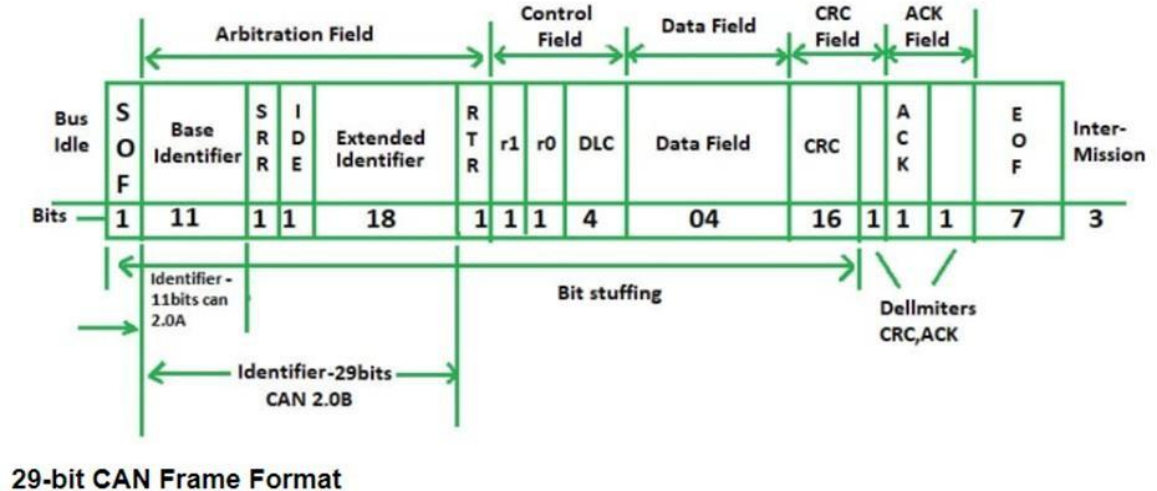


Figure 38 The c CAN frame format.

It is the same as an 11-bit identifier with some added bits.

SRR (Substitute Reverse Request) – The SRR bit is always transmitted as a recessive bit to ensure that, in the case of arbitration between a Standard Data Frame and an Extended Data Frame, the Standard Data Frame will always have priority if both messages have the same base (11 bit) identifier.

R1– It is another bit not used currently and kept for future use.

4.3.5. Message frame

Four different frames can be used on the bus.

- **Data frames:** These are the most commonly used frames and are used when a node transmits information to any or all other nodes in the system. Data Frames consist of fields that provide additional information about the message as defined by the CAN specification. Embedded in the Data Frames are Arbitration Fields, Control Fields, Data Fields, CRC Fields, a 2-bit Acknowledge Field, and an End of Frame.

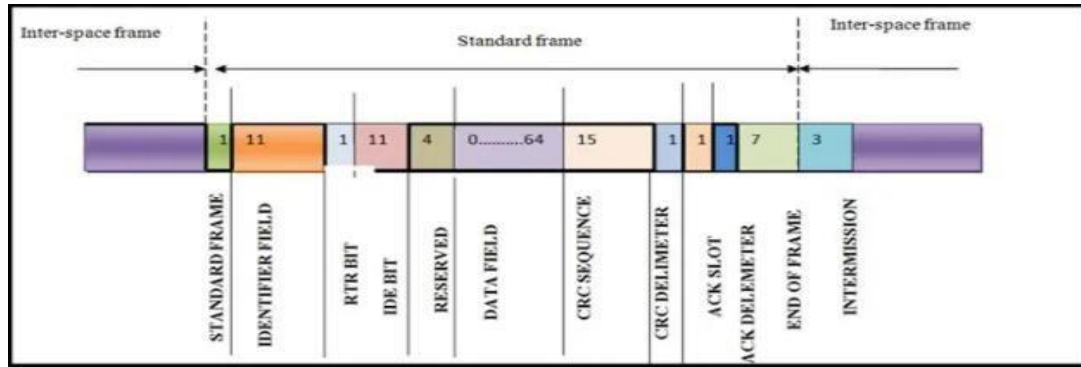


Figure 39 CAN Data frames

- **Extended frame:** The purpose of the remote frame is to seek permission for the transmission of data from another node. This is similar to the data frame without the data field and the RTR bit is recessive.
- **Error frames:** If the transmitting or receiving node detects an error, it will immediately abort transmission and send an error frame consisting of an error flag made up of six dominant bits and an error flag delimiter made up of eight recessive bits. The CAN controller ensures that a node cannot tie up a bus by repeatedly transmitting the error frame.

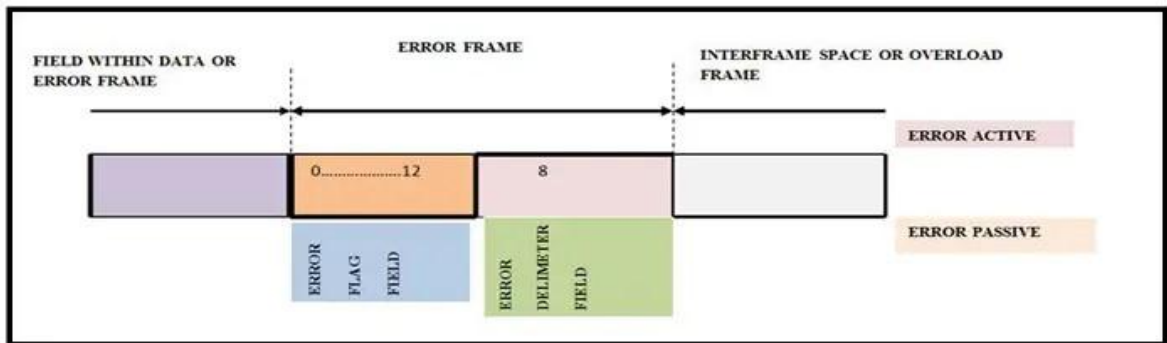


Figure 40 CAN Error Frames

- **Overload frame:** It is similar to the error frame but used for providing an extra delay between the messages. An Overload frame is generated by a node when it becomes too busy and is not ready to receive.

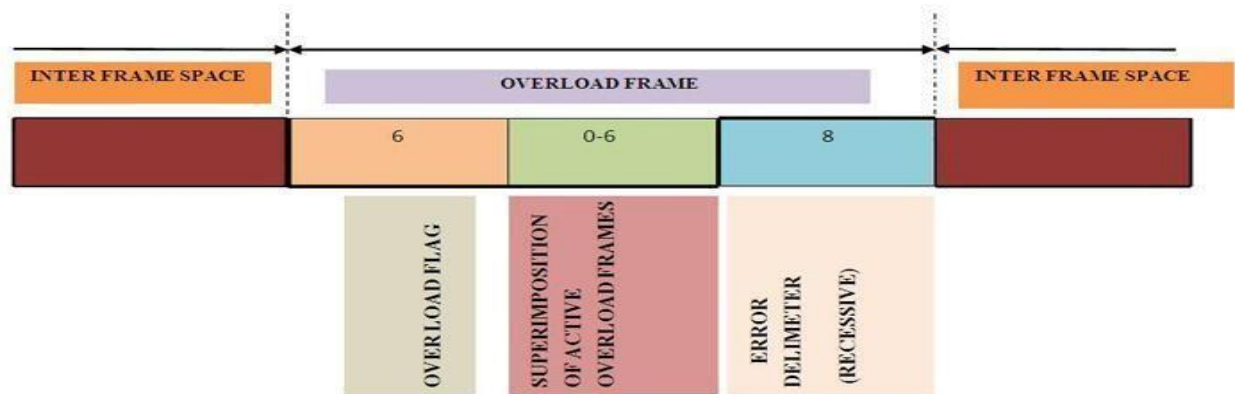


Figure 41 CAN Overload frame.

4.3.6. CAN Protocol Working Principle

Each node in the CAN bus requires the below modules to work together in a CAN network.

1. **Microcontroller (Host):** It decides what the received messages mean and what messages it wants to transmit.
2. **CAN controller:** They are often an integral part of the microcontroller that handles framing, CRC, etc. like the Data Frame, Remote Frame, Error Frame, Overload Frame, and Inter-Frame Space generation.
3. **CAN Transceiver:** It converts the data from the CAN controller to the CAN bus levels and also converts the data from the CAN bus levels to a suitable level that the CAN controller uses.

The transceiver drives or detects the dominant and recessive bits by the voltage difference between the CAN_H and CAN_L lines. The nominal dominant differential voltage is between 1.5V to 3V and the recessive differential voltage is always 0V.

The CAN transceiver actively drives to the logical 0 (dominant bits) voltage level and the logical 1 (recessive bits) are passively returned to 0V by the termination resistor. The idle state will always be at the recessive level (logical 1).

Individually, CAN_H will always be driven towards supply voltage (VCC) and CAN_L towards 0V when transmitting to the dominant (0). But in a practical case, supply voltage (VCC) or 0V cannot be reached due to the transceiver's internal diode drop. CAN H/L will not be driven when transmitting a recessive (1) where the voltage will be maintained at $VCC/2$.

4.3.7. Operation of the CAN Network

Each node is then assigned a unique identification number called the Physical Address of the ECU.

All the nodes are interesting to transmit and compete for the channel by transmitting a binary signal based on their identification value.

A node will drop out of the competition if it detects a dominant state while transmitting a passive state.

Thus, the node which has the lowest identification value will win the bus network and start the transmission of the message.

If any error detects either the transmitter or receiver, it stops the sending of the Data Frame and starts sending the Error Frame. The error is having an error state to handle the error in CAN Protocol called Error Handling in CAN Protocol. There are 5 types of errors in CAN protocol that can occur.

4.3.8. Interface between Nodes and Communication Network

System Nodes are stm32f103 boards, which support CAN units. The system contains four nodes, every one of them must have its unique message id so that all other nodes in the system can communicate with it. The messages must contain this id so the corresponding node can ack. We should know that all messages in the systems will be sent through the same network bus, but the node will response and save only the messages that hold its id. There are techniques in the CAN unit itself that can handle transmitting and receiving messages. **a) CAN Filtering:**

The filtering is done by arbitration identifier of the CAN frame. This the technique is also used when monitoring a CAN bus, to focus messages of importance using an identifier and mask. These allow a range of IDs to be accepted with a single filter rule. When a CAN frame is received, the mask is applied. This determines which bits of the identifier will be used to determine if the frame matches the filter.

b) CAN FIFO Buffer:

First in First Out concept using in CAN Store multiple Messages Received in RAM as Receiving each message Individually keeps interrupting the CPU for every single Message receives. It will make the system overhead, especially in multitasking systems.

4.3.9. CAN in Action

For the next couple of flow chart, we will illustrate how to transmit and receive a message based on our blue pill board and our CAN unit Driver.

a) Transmitting a CAN message:

Transmitting a message typically requires loading the identifier, RTR, ID, Data length, and the Data into Transmit structure. b)

Receiving a CAN message:

Receiving a message typically requires loading filtering id to set filter structure and monitoring the FMP bit with interrupt or polling to notify user application when a CAN message has been received.

4.4. USART protocol

4.4.1. Introduction

USART, or universal asynchronous receiver-transmitter, is one of the most used device-to-device communication protocols. This article shows how to use USART as a hardware communication protocol by following the standard procedure.

When properly configured, USART can work with many different types of serial protocols that involve transmitting and receiving serial data. In serial communication, data is transferred bit by bit using a single line or wire. In two-way communication, we use two wires for successful serial data transfer. Depending on the application and system requirements, serial communications need less circuitry and wires, which reduces the cost of implementation.

Communication protocol plays a big role in organizing communication between devices. It is designed in different ways based on system requirements, and these protocols have a specific rule agreed upon between devices to achieve successful communication.

Embedded systems, microcontrollers, and computers mostly use USART as a form of device-to-device hardware communication protocol. Among the available communication protocols, USART uses only two wires for its transmitting and receiving ends.

Despite being a widely used method of hardware communication protocol, it is not fully optimized all the time. Proper implementation of frame protocol is commonly disregarded when using the USART module inside the microcontroller.

By definition, USART is a hardware communication protocol that uses asynchronous serial communication with configurable speed. Asynchronous means there is no clock signal to synchronize the output bits from the transmitting device going to the receiving end.

4.4.2. Interface

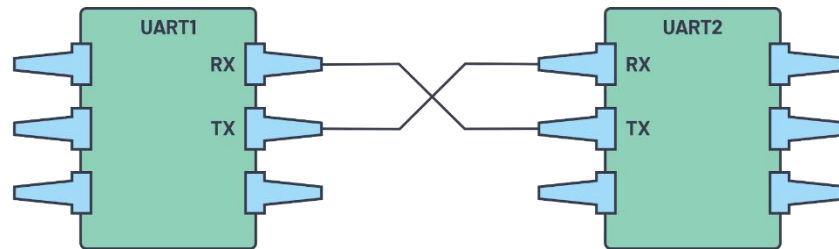


Figure 42 Two USARTs directly communicate with each other.

The two signals of each USART device are named:

- Transmitter (Tx)
- Receiver (Rx)

The main purpose of a transmitter and receiver line for each device is to transmit and receive serial data intended for serial communication.

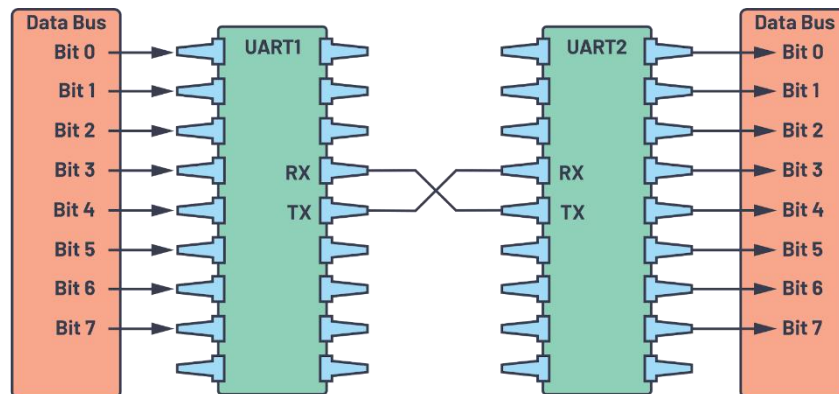


Figure 43 USART with data bus

The transmitting USART is connected to a controlling data bus that sends data in a parallel form. From this, the data will now be transmitted on the transmission line (wire) serially, bit by bit, to the receiving USART. This, in turn, will convert the serial data into parallel for the receiving device.

The USART lines serve as the communication medium to transmit and receive one data to another. Take note that a USART device has a transmit and receive pin dedicated for either transmitting or receiving.

For USART and most serial communications, the baud rate needs to be set the same on both the transmitting and receiving device. The baud rate is the rate at which information is transferred to a communication channel. In the serial port context, the set baud rate will serve as the maximum number of bits per second to be transferred.

Wires	2
Speed	9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600, 1000000, 1500000
Methods of Transmission	Asynchronous
Maximum Number of Masters	1
Maximum Number of Slaves	1

The USART interface does not use a clock signal to synchronize the transmitter and receiver devices; it transmits data asynchronously. Instead of a clock signal, the transmitter generates a bitstream based on its clock signal while the receiver is using its internal clock signal to sample the incoming data. The point of synchronization is managed by having the same baud rate on both devices. Failure to do so may affect the timing of sending and receiving data which can cause discrepancies during data handling. The allowable difference of the baud rate is up to 10% before the timing of bits gets too far off.

4.4.3. Data Transmission

In USART, the mode of transmission is in the form of a packet. The piece that connects the transmitter and receiver includes the creation of serial packets and controls those physical hardware lines. A packet consists of a start bit, a data frame, a parity bit, and stop bits.

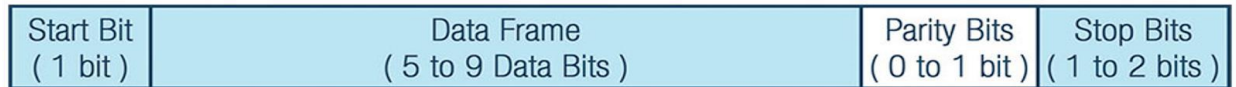


Figure 44 USART packet

• Start Bit

The USART data transmission line is normally held at a high voltage level when it's not transmitting data. To start the transfer of data, the transmitting USART pulls the transmission line from high to low for one clock cycle. When the receiving USART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate.

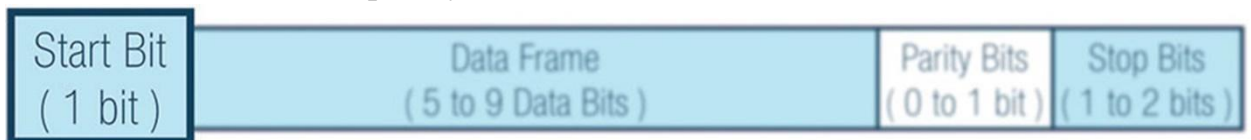


Figure 45 Start bit.

• Data Frame

The data frame contains the actual data being transferred. It can be five bits up to eight bits long if a parity bit is used. If no parity bit is used, the data frame can be nine bits long. In most cases, the data is sent with the least significant bit first.



Figure 46 Data Frame

• Parity

Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving USART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long-distance data transfers.

After the receiving USART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number. If the parity bit is a 0 (even parity), the 1 or logic-high bit in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1 bit or logic highs in the data frame should total to an odd number.

When the parity bit matches the data, the USART knows that the transmission was free of errors. But if the parity bit is a 0, and the total is odd, or the parity bit is a 1, and the total is even, the USART knows that bits in the data frame have changed.

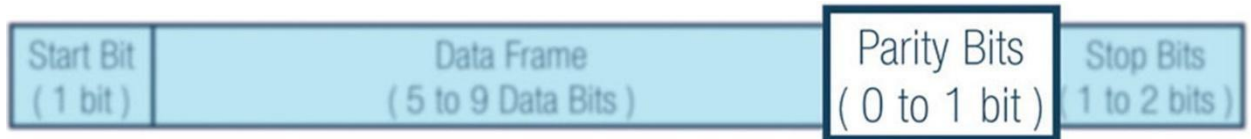


Figure 47 Parity bits

• Stop Bits

To signal the end of the data packet, the sending USART drives the data transmission line from a low voltage to a high voltage for one- or two-bit(s) duration.



Figure 48 Stop bits.

4.4.4. USART Transmission

First: The transmitting USART receives data in parallel from the data bus.

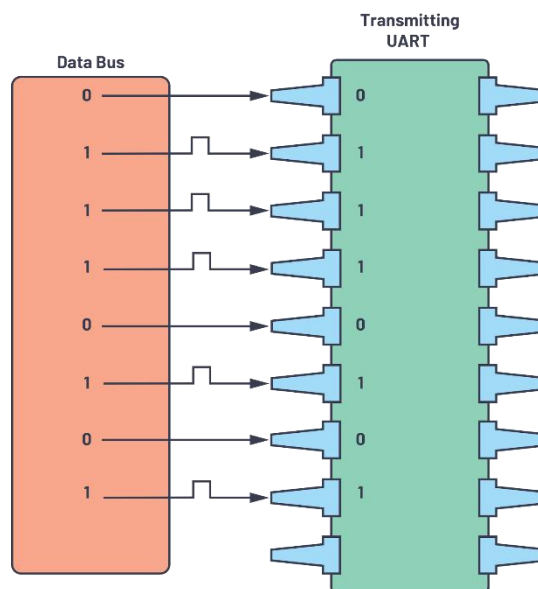


Figure 49 Data bus to the transmitting USART

Second: The transmitting USART adds the start bit, parity bit, and the stop bit(s) to the data frame.

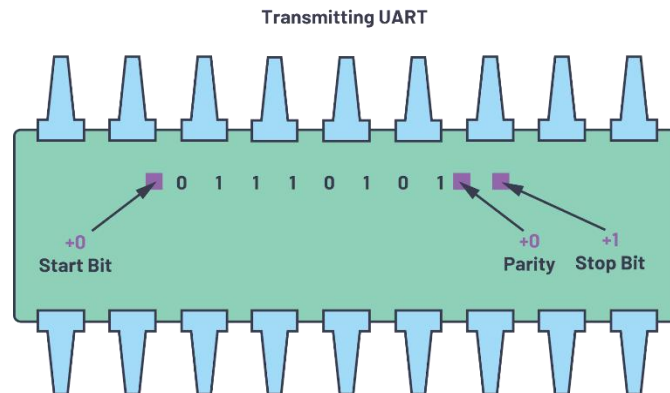


Figure 50 The USART data frame at the Rx side

Third: The entire packet is sent serially starting from the start bit to the stop bit from the transmitting USART to the receiving USART. The receiving USART samples the data line at the preconfigured baud rate.

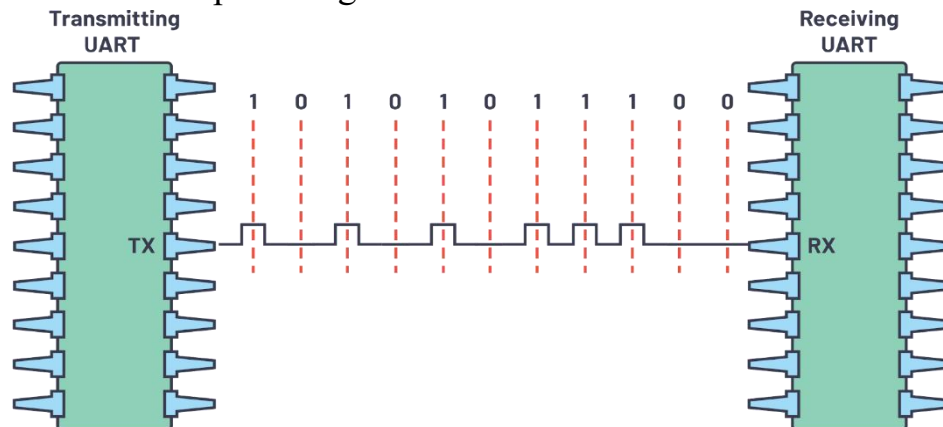


Figure 51 USART transmission

Fourth: The receiving USART discards start bit, parity bit, and stop bit from the data frame.

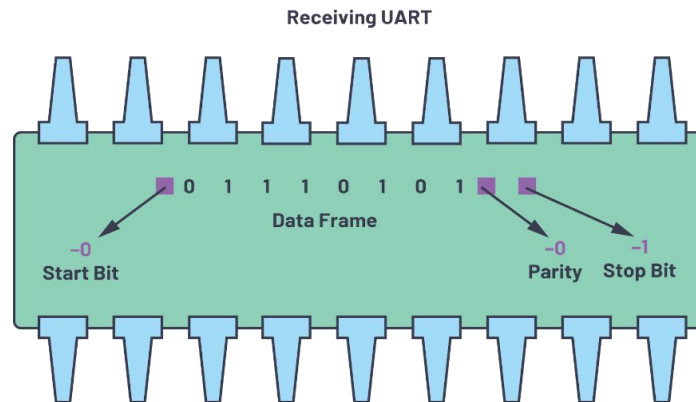


Figure 52 The USART data frame at the Rx side

Fifth: The receiving USART converts the serial data back into parallel and transfers it to the data bus on the receiving end.

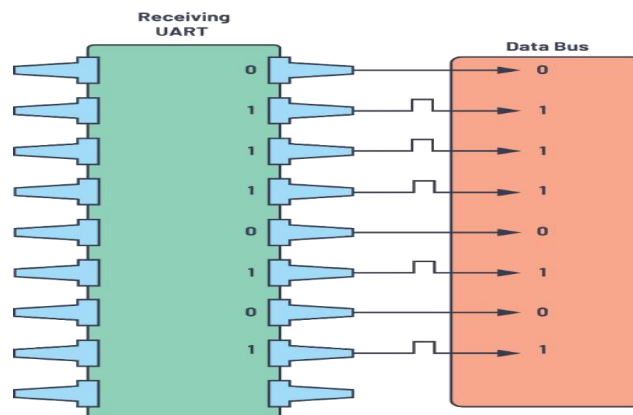


Figure 53 Receiving USART to data bus.

4.5. Implemented Communication Protocol

4.5.1. Parallel Communication Protocols

Parallel communication is a method of conveying multiple bits simultaneously. It contrasts with serial communication, which conveys only a single bit at a time; this distinction is one way of characterizing a communications link.

The basic difference between a parallel and a serial communication channel is the number of electrical conductors used at the physical layer to convey bits. Parallel communication implies more than one such conductor. For example, an 8-bit parallel channel will convey eight bits (or a byte) simultaneously, whereas a serial channel would convey those same bits sequentially, one at a time. If both channels operated at the same clock speed, the parallel channel would be eight

times faster. A parallel channel may have additional conductors for other signals, such as a clock signal to pace the flow of data, a signal to control the direction of data flow, and handshaking signals.

Parallel communication is and always has been widely used within integrated circuits, in peripheral buses, and in memory devices such as RAM. Computer system buses, on the other hand, have evolved over time: parallel communication was commonly used in earlier system buses, whereas serial communications are prevalent in modern computers.

4.5.2. Asynchronous Octal Communication (AOC)

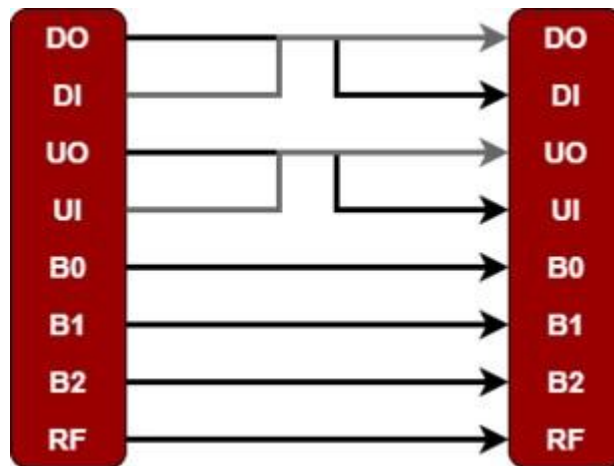


Figure 54 AOC Diagram

The protocol is called Asynchronous Octal Communication (AOC) which is a simple parallel communication to indicate updates and diagnostics requests and data.

The protocol is customized for our project where every controller can send and receive data for updates and diagnostics. It is based on interruptions to increase responsiveness.

There are 4 states for communication:

1. Diagnostics Request from Raspberry Pi to STM32:



Figure 55 Diagnostics Request

The raspberry pi sends pulse from low to high on its output diagnostics pin to be received by STM32.

2. Diagnostics Data from STM32 to Raspberry Pi:

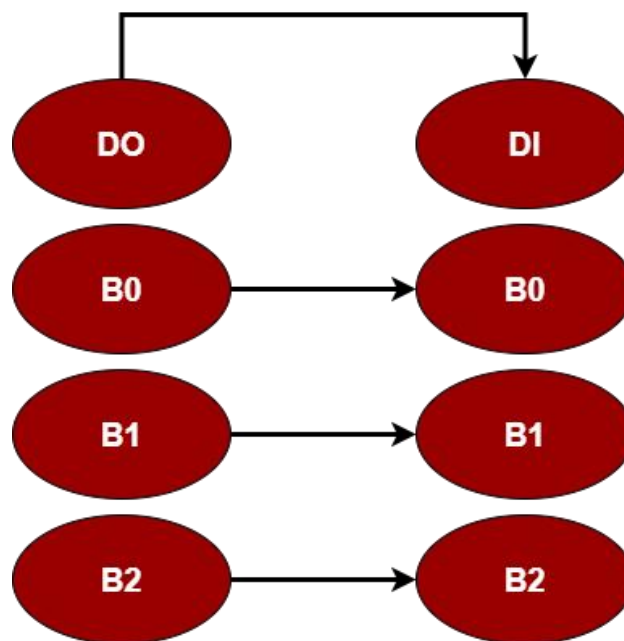


Figure 56 Diagnostics Data

STM32 sends diagnostics data on the 3 data bits, then sends pulse from low to high on its output diagnostics pin to be received by Raspberry Pi. 3. Update Notification and Progress from STM32 to Raspberry Pi:

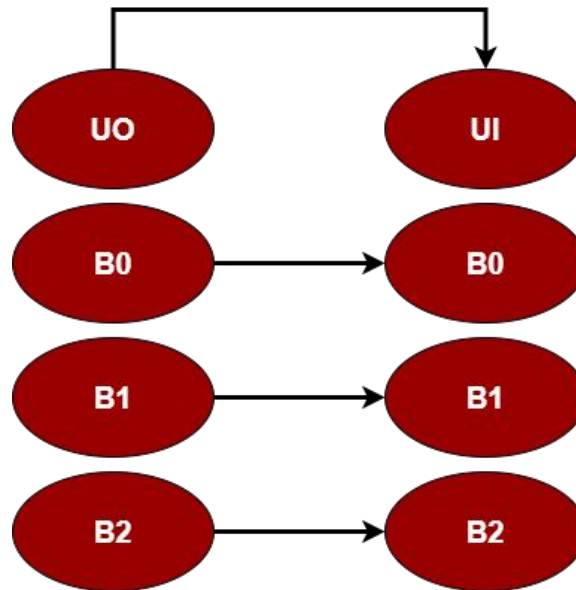


Figure 57 Update Notification and Data

STM32 sends to Raspberry Pi on the data lines:

- a. 000 for update notification.
- b. 100 to indicate update is in progress.
- c. 010 to indicate update completion.
- d. 001 to indicate update failure.

Then a pulse is sent from low to high on its output update pin to be received by Raspberry Pi.

4. Update Response from Raspberry Pi to STM32:

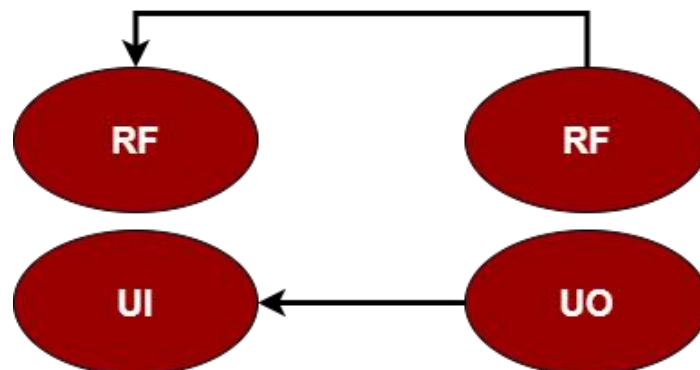


Figure 58 Update Response

The response is sent from Raspberry Pi to STM32 on its output update pin, then a pulse is sent from low to high on the response flag to be received by STM32.

4.6. Bootloader

4.6.1. Introduction

In automotive, before a bootloader, to solve the problem in software or update the application using ICP (in-circuit programming) method, Using JTAG or SWD protocol to load the user application into MCU.

JTAG (Joint Test Action Group) was designed largely for chip and board testing. It is used for boundary scans, checking faults in chips /boards in productions debugging and flashing micros was an evolution in its applications over the time SWD is an ARM-specific protocol designed specifically for micro debugging.

Protocol	SWD	JTAG
Supported CPU	ARM only	Independent group
Topology	Star	Daisy chained
Special feature	Printing debugging information	Not supported



Figure 59 SWD and JTAG protocol

Then, using in application programming Method, implement the small application used to flash the main application in flash memory called bootloader.

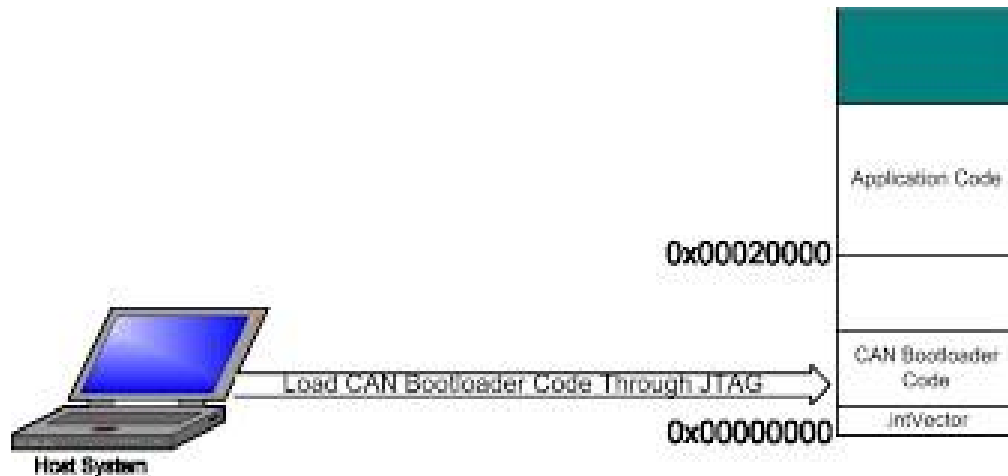


Figure 60 Load CAN Bootloader code through JTAG

A bootloader is a small OS or Application, it is designed to download the firmware in MCU's internal or external memory.

It gives the possibility to upgrade or change the application in the MCU.

4.6.2. Memory Architecture

Memory is split into 4 partitions:

- a) Bootloader: It is a small application that is used to flash application in flash memory.
- b) Application 1: It is the version one of application, these space from the flash, the bootloader loads the application hex file in it.
- c) Application 2: It is the second version of the application 1, it run if the data corruption when the bootloader flash update or another file in the space of applicatin1.
- d) Request flag: It is the indication that indicates there is an update or no update.

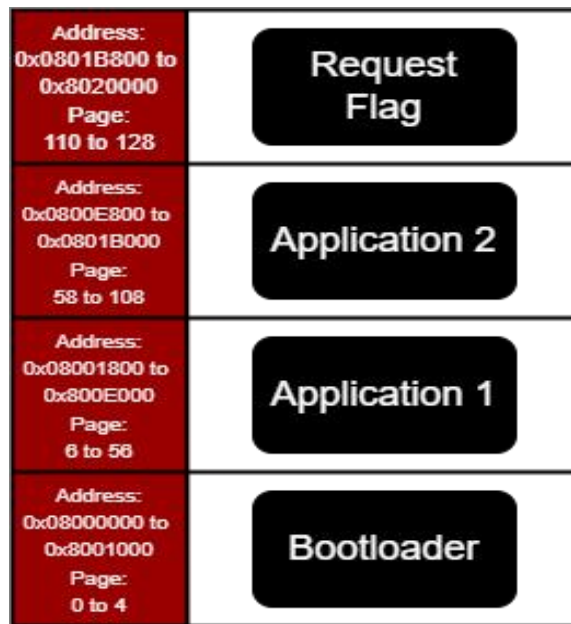


Figure 61 Memory Hierarchy

Memory architecture is selected based on safety and the application is not stopped when data corruption occurs.

4.6.3. Bootloader Design

Bootloader design has 4 blocks:

a) Preload:

- i) This block is used to check there is update or not by using the request flag, if the request flag is zero then there is update and run the bootloader and write the application hex file on the space of application1 but if the request flag is one then, no update and run the application1, if the request flag any number except zero or one then no update and run application2.

ii) The benefits of the preload:

- (1) It prevents the bootloader discovery. (Is there an update or not?)
- (2) Manage between Two applications and bootloader.

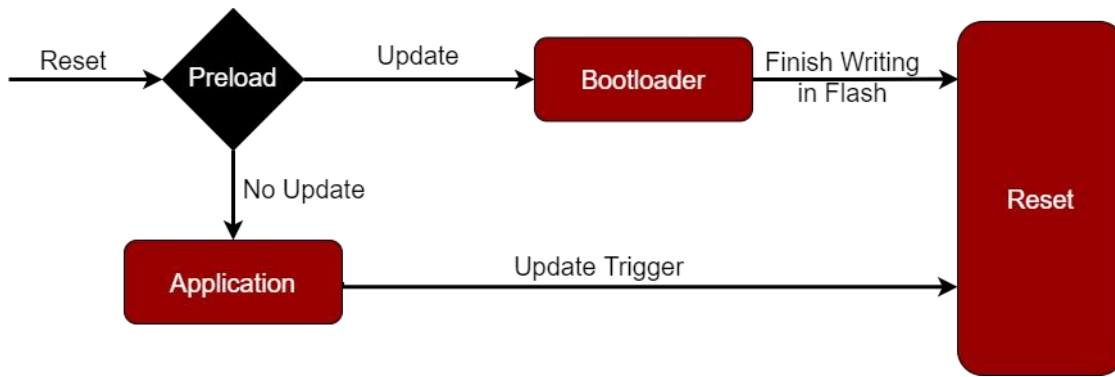


Figure 62 Bootloader Flow Chart

b) Application:

There are two applications, the preload is selected between them using the request flag or the state of operation of the writing in flash memory. **c) Bootloader:**

Bootloader implementation:

- (1) Erase the space of application1.
- (2) Receive one record in iteration.
- (3) Make a checksum to verify the data is received correctly.
- (4) Then write the record in flash.
- (5) With each iteration repeat this step from receive record.

d) Reset:

There are two state to go to reset:

- 1) The bootloader End the operation, if it success to update the application then change the request flag into the no update then, reset the microcontroller then, start the preload then run the application one, if it failed to update the application because of data corruption then change the request flag into Data corruption then reset micro-controller then start the preload and run the version 2 from the previous application (Space of Application2).
- 2) Application receives the update notification from the user to update the software of application then change the request flag into update flag the start the preload and run the bootloader perform update operation.

4.6.4 Bootloader Structure

Describe the main components of the bootloader, such as:

1. **Main Function:** The main function initializes the required peripheral and continuously calls the bootloader's start function.

```
c Copy code  
  
int main(void)  
{  
    // ... Initialization code ...  
  
    for (;;)   
    {  
        /* Start Bootloader */  
        Bootloader_Start();  
    }  
}
```

Figure 63 Main Function

2. **Static Functions:** These functions perform essential tasks such as flash memory erasure, memory write, and sending ACK messages.

```
c Copy code  
  
static void Bootloader_Send_ACK();  
static ReturnStatus Bootloader_Preform_Flash_Erase(u8 pageNum, u8 NumOfPage)  
static void Bootloader_Jumb_To_Address(u32 Address);  
static ReturnStatus Bootloader_Erase_Flash(u8 *Host_Buffer);  
static ReturnStatus Bootloader_Memory_Write(u8 *Host_Buffer);
```

Figure 64 Static Functions

3. **CAN Initialization:** The CAN Module is initialized to enable communication between the bootloader and the host.

```
c Copy code  
  
CAN_VoidInit(CAN1, &CAN_InitStruct);  
CAN_VoidFilterSet(&CAN_FilterInitStruct_1);
```

Figure 65 CAN Initialization

- 4. Start Command:** The Bootloader sends a start command to the host to initiate the firmware update process.

```
c Copy code  
  
void Bootloader_Send_Start_Command()  
{  
    CAN_VoidTransmit(CAN1, &CAN_START_MSG_NODE2);  
}
```

Figure 66 Start Command

Flash

Memory Erasure:

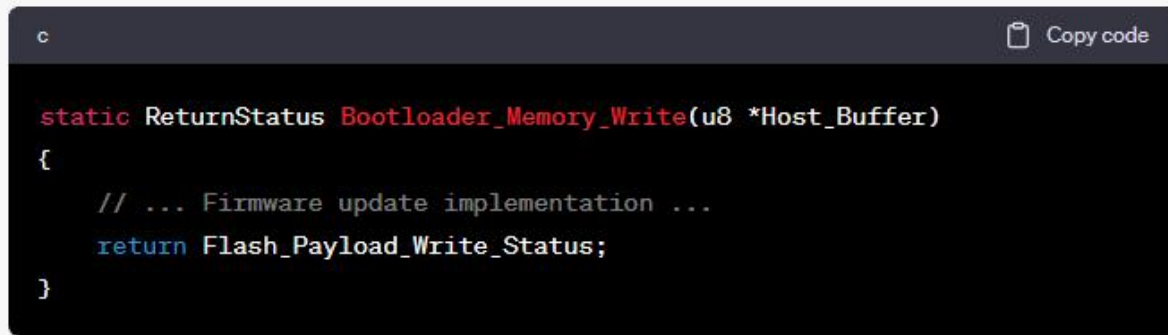
Explain how the bootloader performs flash memory erasure. Use the provided code snippet for the flash memory erase function and illustrate how it prepares the memory for the new firmware.

```
c Copy code  
  
static ReturnStatus Bootloader_Preform_Flash_Erase(u8 pageNum, u8 NumOfPage)  
{  
    // ... Flash erase implementation ...  
    return Erase_Status;  
}
```

Figure 67 Flash Memory Erasure

Firmware Update Process:

Describe the steps involved in the firmware update process. Start with the host sending new firmware data, the bootloader receiving it, and finally writing it to the appropriate memory location.

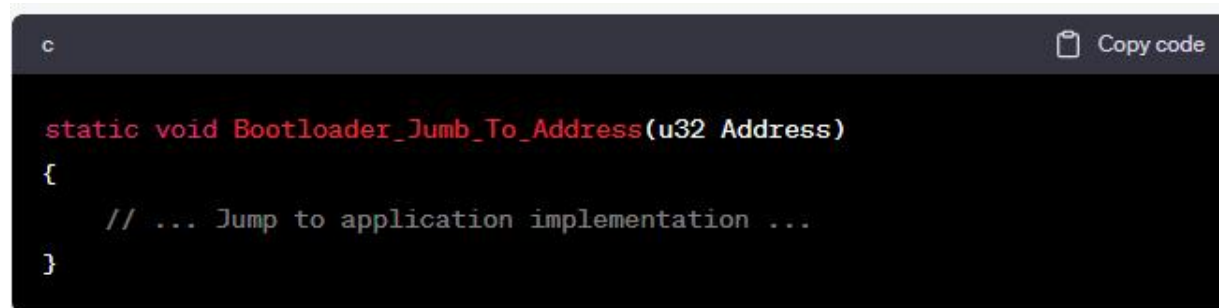
A code editor window with a dark background. The title bar shows a 'c' icon and a 'Copy code' button. The code is in C and defines a static function Bootloader_Memory_Write that takes a pointer to a u8 array and returns a ReturnStatus. The function body contains a comment and a return statement.

```
c Copy code  
  
static ReturnStatus Bootloader_Memory_Write(u8 *Host_Buffer)  
{  
    // ... Firmware update implementation ...  
    return Flash_Payload_Write_Status;  
}
```

Figure 68 Firmware Update Process

Jumping to Application:

Explain how the bootloader initiates the main application after successfully updating the firmware. Provide insights into how the bootloader jumps to the application start address and begins the execution of the new firmware.

A code editor window with a dark background. The title bar shows a 'c' icon and a 'Copy code' button. The code is in C and defines a static void function Bootloader_Jumb_To_Address that takes a u32 Address. The function body contains a comment.

```
c Copy code  
  
static void Bootloader_Jumb_To_Address(u32 Address)  
{  
    // ... Jump to application implementation ...  
}
```

Figure 69 Jumping to Application

4.7. Real Time Operating System (RTOS)

4.7.1. Introduction

An Operating System (OS) is software that acts as an interface between computer hardware components and the user. Every computer system must have at least one operating system to run other programs. Applications like Browsers, MS Office, Notepad Games, etc., need some environment to run and perform their tasks.

A real-time operating system (RTOS) is an operating system with two key features: predictability and determinism. In an RTOS, repeated tasks are performed within a tight time boundary, while in a general-purpose operating system, this is not necessarily so. Predictability and determinism, in this case, go hand in hand: We know how long a task will take, and that it will always produce the same result.

RTOSes are subdivided into “soft” real-time and “hard” real-time systems. Soft real-time systems operate within a few hundred milliseconds, at the scale of a human reaction. Hard real-time systems, however, provide responses that are predictable within tens of milliseconds or less.

Characteristics of an RTOS:

1. **Determinism:** Repeating an input will result in the same output.
2. **High performance:** RTOS systems are fast and responsive, often executing actions within a small fraction of the time needed by a general OS.
3. **Safety and security:** RTOSes are frequently used in critical systems when failures can have catastrophic consequences, such as robotics or flight controllers. To protect those around them, they must have higher security standards and more reliable safety features.
4. **Priority-based scheduling:** Priority scheduling means that actions assigned a high priority are executed first, and those with lower priority come after. This means that an RTOS will always execute the most important task.
5. **Small footprint:** Versus their hefty general OS counterparts, RTOSes weigh in at just a fraction of the size. For example, Windows 10, with post-install updates, takes up approximately 20 GB. VxWorks®, on the other hand, is approximately 20,000 times smaller, measured in the low single-digit megabytes.

4.7.2. Free RTOS Overview

Free RTOS developed in partnership with the world’s leading chip companies over an 18-year period, and now downloaded every 170 seconds, Free RTOS is a market-leading real-time operating system (RTOS) for microcontrollers and small microprocessors.

It’s distributed freely under the MIT open-source license, Free RTOS includes a kernel and a growing set of IoT libraries suitable for use across all industry sectors. Free RTOS is built with an emphasis on reliability and ease of use.

Characteristics of free RTOS:

1. **Trusted kernel:** With proven robustness, tiny footprint, and wide device support, the Free RTOS kernel is trusted by world-leading companies as the de facto standard for microcontrollers and small microprocessors.

2. **Accelerate time to market:** With detailed pre-configured demos and Internet of Things (IoT) reference integrations, there is no need to determine how to set up a project. Quickly download, compile, and get to the market faster.
3. **Broad ecosystem support:** Our partner ecosystem provides a breadth of options including community contributions, professional support, as well as integrated IDE and productivity tools.
4. **Predictability of long-term support:** Free RTOS offers feature stability with long term support (LTS) releases. Free RTOS LTS libraries come with security updates and critical bug fixes for two years. Maintained by AWS for the benefit of the Free RTOS community.

4.7.3. Free RTOS Implementation

- We downloaded the free RTOS libraries and tested the needed files.
- We divided the application ECU code into several tasks each abstracted from one another.
- We used queues for inter-task communication as the tasks must be abstracted from each other.

After understanding free RTOS concepts we implemented our own Real Time Operating System.

4.7.4. RTOS Implementation

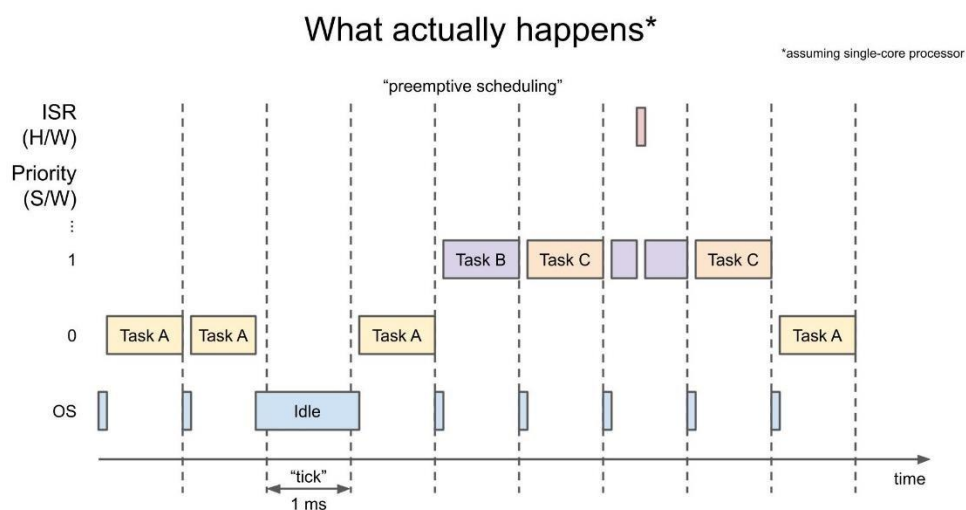


Figure 63 RTOS Time Slicing

We divided the RTOS functions into

a. Scheduler Related Functions.

The scheduler is an operating system module that selects the next jobs to be admitted into the system and the next process to run.

Our scheduler is designed in First Come First Serve (FCFS) algorithm which is a non-preemptive scheduling algorithm.

First Come First Serve (FCFS) algorithm is easy to understand and implement which tends to serve the highest priority tasks first b. Tasks Related Functions.

The most important function is `RTOS_createTask` which creates each task with specific priority and periodicity (period time of task) in addition to first delay (Delay after which the task will begin).

There are other functions such as `RTOS_resumeTask`, `RTOS_suspendTask`, `RTOS_deleteTask` which changes the task states whether to be executed by scheduler (Resumed), temporarily suspended, or completely deleted.

4.8. Diagnostics Handling

4.8.1. Introduction

When it comes to OTA technology in the automotive industry, a common perception is the addition of new features to the vehicle, in the same way as Tesla does. However, the range of possibilities is actually much wider. Remote live diagnostics is one of the main features for smart vehicles, and it can timely deliver a new driving experience and be used to diagnose and repair the vehicle's faults, to ensure the vehicle's safety through its life cycle.

In non-FOTA capable ECUs, the update-relevant diagnostic services are usually supported in reprogramming or boot mode only, but as the installation on FOTA Target ECUs shall now happen during the normal operating mode, i.e., while driving, these services shall be supported in the normal operating mode too.

Remote diagnostics use vehicle data to determine the cause of a problem. A recommendation can be given if it is sensible to continue the drive to the next workshop or not.

4.8.2. Diagnostics in our system

The status of the vehicle is determined through various sensors, and in case of any issues, a unique code is sent to the user and uploaded to the OEM server to indicate the issue.

There are two different modes implemented for running diagnostics to achieve higher safety and better user experience.

Mode One

In Mode one, the user chooses to run diagnostics on the car which will return to the user any problem, even if it is not critical, in any application. In this mode, the user will also be informed if there are no problems at all.

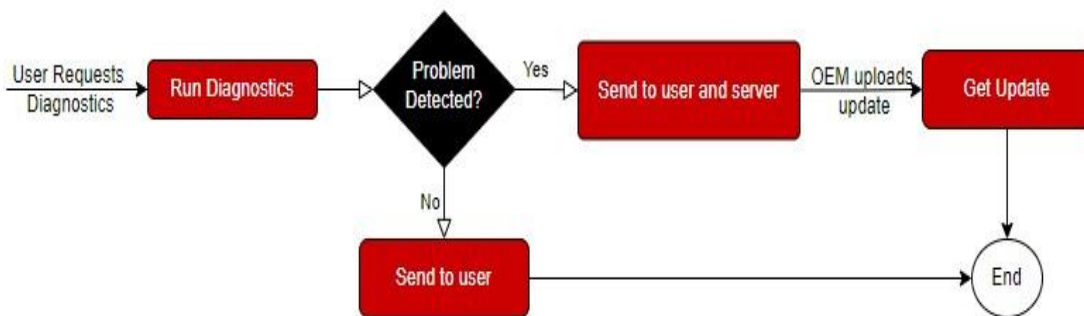


Figure 64 Remote Diagnostics - Mode 1

Mode Two

In mode two, diagnostics are run periodically on the application ECUs and the user is automatically alerted only in case a serious problem occurs.

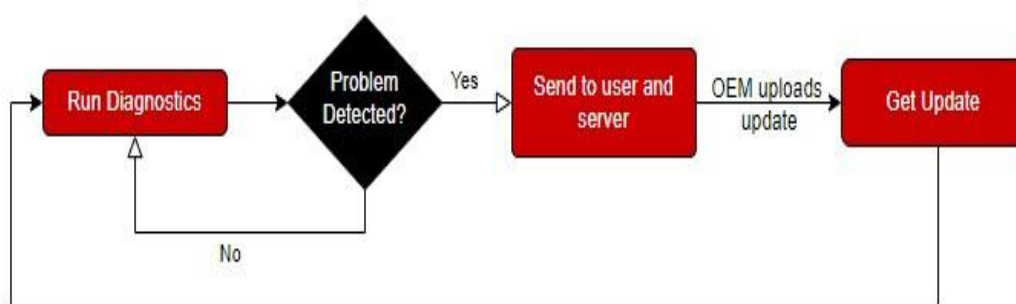


Figure 65 Remote Diagnostics - Mode 2

When an issue is discovered, in either mode, the user is alerted, and the issue is also sent to the OEM server. Some car issues can be resolved, temporarily or permanently, by some tweaks in the ECU firmware, in which case the OEM

uploads a firmware version that solves this problem on its server for the car experiencing the issue, and a notification of the update is sent to the user who can then choose whether to download the update.

In case the issue cannot be resolved by a firmware update, the user will still be alerted so that they can head to the nearest repair center to solve the issue as soon as possible.

The OEM can also collect the data of the issues uploaded by different users to figure out any firmware/hardware problems and modify them in future systems.

4.8.3. Diagnostics for different application ECUs

Application ECU 1

- Engine System

In the engine system, motor issues are monitored in both modes, where the feedback of the encoder is compared to the desired direction of movement and gives an error instantly when there is any direction error.

- Collision System

The collision system is only tested in mode 1, where the user keeps a certain distance between the vehicle and a barrier then the reading of the ultrasonic sensor is compared to that distance and the user is alerted if there is any error in the reading.

Application ECU 2

- Temperature Monitoring

Temperature Sensor reading is monitored in both modes. However, the range of error is different in the two modes, where, in mode one (requested by the user), the user is alerted of any reading even slightly outside of the normal range. On the other hand, in mode two (carried out periodically) the user is only alerted when the temperature sensor reading is significantly outside the normal range.

4.8.4. Future Developments in Vehicle Diagnostics

Collecting dynamically shared data from a fleet of vehicles helps OEMs gain new insights as needed. Additional value can be created by applying statistics methods, machine learning or big data analytics on such collected data. For example, early feedback loops can be implemented which makes predictive

maintenance possible by collecting data indicating an imminent damage and new services can be offered based on vehicle data.

4.9. User Interface

4.9.1. The dashboard

First page: Home page which has an image that expresses the idea of the project, and a button to go to the menu page.



Second page: Menu page has four buttons and a slider.

- Home button: to go to Home Page.
- Sleep mode button: to make the screen enter the sleep mode to save battery, and touch will auto awake switch during sleep mode.
- Gauges button: to go to Gauges Page.
- Music button: to go to Music Page.
- Brightness slider: to control the brightness of the screen.

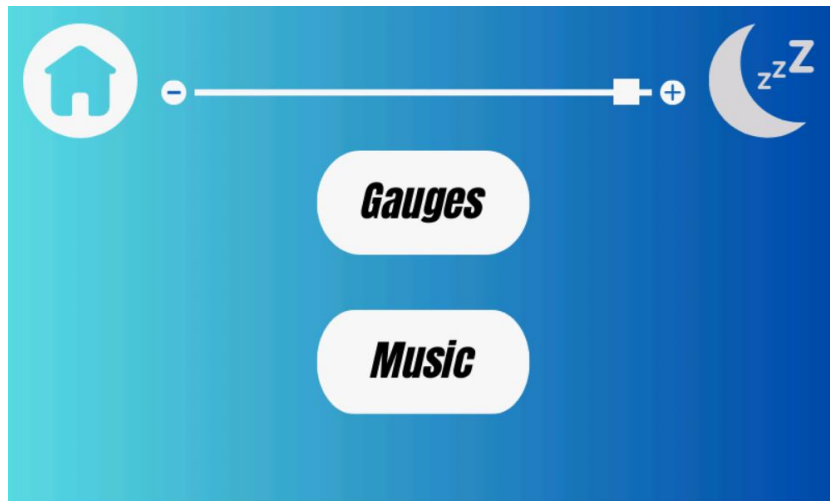


Figure 66 Second page

Third page: Gauges page has two gauges one for Speed and the other for RPM, two buttons one to go menu page and the other to go home page and check the engine alarm which flashlight when there is a problem with the engine.



Figure 67 Gauges

Fourth page: Music page which controls the mp3 player that connects to the other ECU. It has eight buttons and a text bar.

- Home button: to go to Home Page.
- Sound name bar: it shows the name of the soundtrack.
- Menu button: to go to Menu Page.
- Next button: to Play the next soundtrack.
- Pause button: to Pause the soundtrack.
- Volume button: to play the sound.
- Previous button: to Play the previous soundtrack.
- Play button: to Play the soundtrack.

- Mute button: to mute the sound.

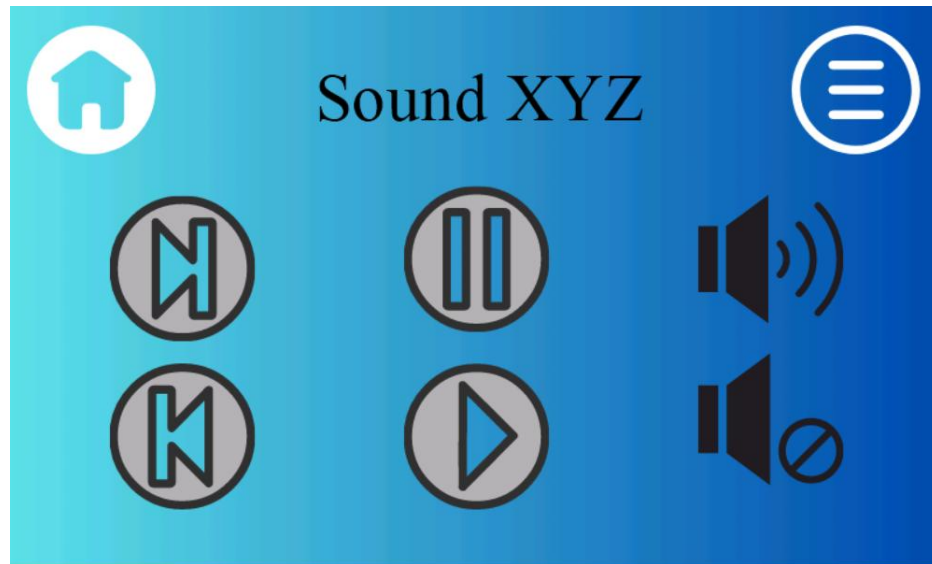


Figure 68 Music page

4.10. Hardware Implementation

4.10.1. Altium PCB Designer

Altium Designer System Engineering (SE) is a fully featured editor for schematics that includes powerful collaboration capabilities and a rich set of schematic capture tools to quickly create, edit, simulate, and document schematics.



Figure 69 Altium Designer System

Altium Designer's suite encompasses four main functional areas, including schematic capture, 3D PCB design, field-programmable gate array (FPGA)

development and release/data management. It integrates with several component distributors for access to manufacturer's data. It also has interactive 3D editing of the board and MCAD export to STEP.

4.10.2. Implemented PCB

We designed the PCB to contain all four vehicle ECUs, in addition to CAN Bus.

Each ECU has its own LED indications to indicate successful transmitting and receiving data from other ECUs.

The Final PCB Dimensions is 30cm x 15cm.

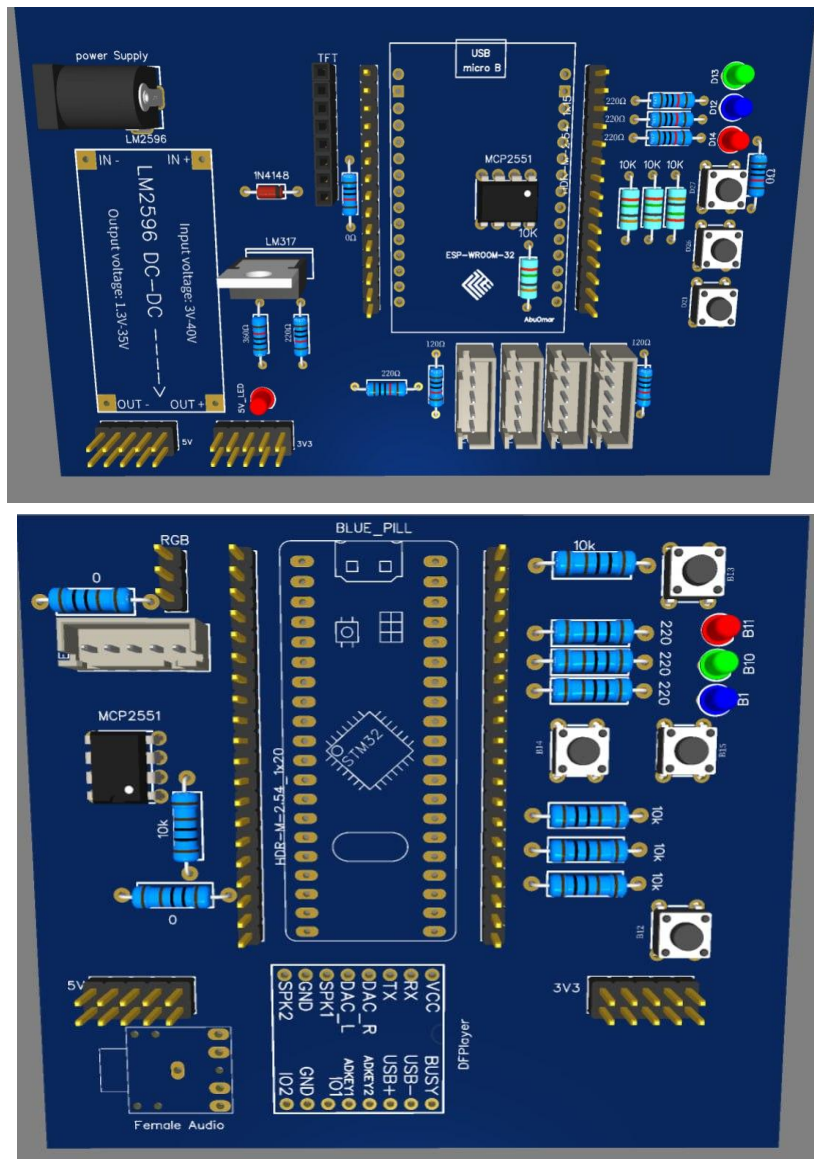


Figure 70 3D Model of our PCB

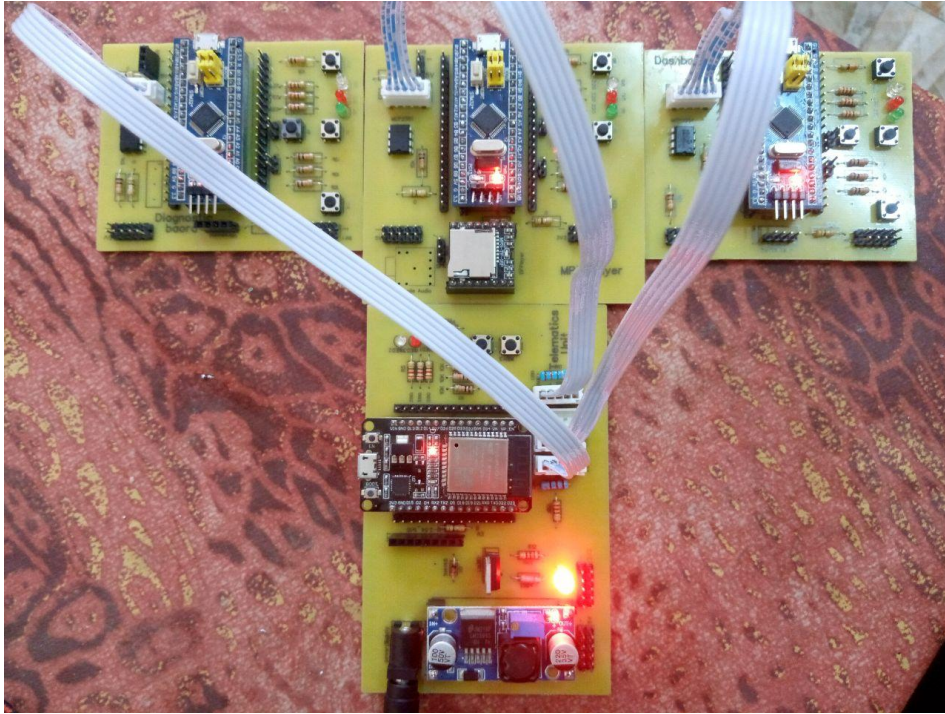


Figure 71 Real view of our PCB

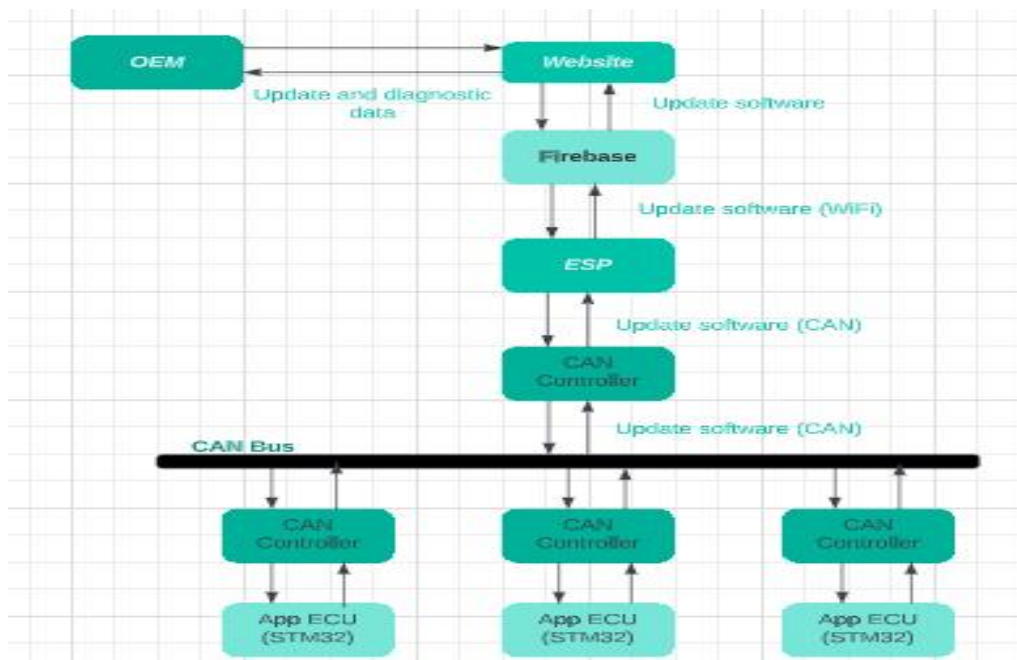


Figure 72 Detailed Diagram with all system components

Chapter 5: Conclusion

5.1. Achievements

1. Implemented all the drivers dealing with hardware from scratch.
2. Made the software portable as possible to support many platforms.
3. Designed a robust bootloader with many features, the most important one is a rollback feature.
4. Designed a robust bootloader with many features, the most important one is a rollback feature.
5. To ensure security of the code we applied encryption and decryption algorithms.
6. Implemented a fully functional GUI to provide the best user experience and interface.
7. Fabricated PCB to organize all hardware components and test all software using hardware.
8. Implemented our own Real Time operating system (RTOS) to be used in application ECUs.
9. Implemented our own communication protocol to communicate between Raspberry Pi and STM32.

GitHub Repository Link:

https://github.com/YehiaEhab16/FOTA_Graduation-Project-2022 Final

Project Video Link:

https://drive.google.com/drive/folders/1yehVSdoG_k_3O2G6XFhWJKm2tc4vHAyb?usp=sharing

5.2. Future Improvements

5.2.1. Adaptive AUTOSAR for diagnostics

Our software uses only software test cases implementation in the diagnostic check modes. Our future work is to use adaptive AUTOSAR to make it more reliable.

5.2.2. Delta file

Our system is to upload the whole file during the update, our future work is to use delta file which means upload only the part of file that need to be updated instead of downloading a big file.

Tools

- 1) Eclipse IDE.
- 2) STM32CubeIDE.
- 3) Visual Studio Code.
- 4) Arduino IDE.
- 5) Putty: Serial Communication.
- 6) VNC viewer: Raspberry Pi Simulation.
- 7) QT designer: GUI design.
- 8) Altium PCB designer: hardware design. 9) Proteus simulation.
- 10) Draw.io: to draw software diagrams.
- 11) STM32 ST-LINK Utility: used to burn code to microcontroller. 12)
Git & GitHub: for version control.

References

- [1] Firmware over the Air Introduction <https://www.soracom.io/iot-definitions/what-is-firmware-over-the-air-fota/>
- [2] Software Development Cycle <https://phoenixnap.com/blog/software-development-life-cycle>
- [3] FOTA Architecture <https://mirror-medium.com/?m=https%3A%2F%2Fmedium.com%2F%40shrimantshubham%2Fwhat-is-fota-how-does-it-work-502c34a06d60>
- [4] OTA System Design <https://www.analog.com/ru/analog-dialogue/articles/over-the-air-ota-updatesin-embedded-microcontroller-applications.html>
- [5] Role of OTA Updates in IoT Device Management <https://thefutureofthings.com/14610-the-role-of-ota-updates-in-iot-devicemanagement/>
- [6] CAN Protocol <https://www.javatpoint.com/can-protocol>
- [7] CAN Protocol Overview <https://www.ni.com/en-lb/innovations/white-papers/06/controller-areanetwork--can--overview.html>
- [8] Bootloader <https://www.sciencedirect.com/topics/engineering/bootloader>

- [9] Bootloader Overview
<https://docs.oracle.com/en/industries/communications/session-bordercontroller/8.1.0/installation/boot-loader-overview.html>

| References

- [10] UART Introduction
https://wiki.st.com/stm32mcu/wiki/STM32StepByStep:Step3_Introduction_to_the_UART
- [11] RTOS Overview
[https://www.windriver.com/solutions/learning/rtos#:~:text=A%20real%2Dtime%20operating%20system%20\(RTOS\)%20is%20an%20operating,this%20is%20not%20necessarily%20so.](https://www.windriver.com/solutions/learning/rtos#:~:text=A%20real%2Dtime%20operating%20system%20(RTOS)%20is%20an%20operating,this%20is%20not%20necessarily%20so.)
- [12] Free RTOS Documentation
<https://www.freertos.org/>
- [13] UART Tutorial <https://deepbluembedded.com/stm32-usart-uart-tutorial/>
- [14] Client Server Overview https://developer.mozilla.org/en-US/docs/Learn/Serverside/First_steps/Client-Server_overview
- [15] Firebase Overview <https://firebase.google.com/docs/extensions/overview-use-extensions>
- [16] Vehicle Diagnostics Over Internet Protocol and Over-the-Air Updates by M. Kathires, R. Neelaveni, M. Adwin Benny & B. Jeffrin Samuel Moses.
- [17] STM32F103 datasheet.
- [18] Raspberry Pi Documentation.
<https://www.raspberrypi.com/>

ملخص المشروع

مؤخرًا، أصبح العالم من حولنا مرتبطًا بطريقة أو بأخرى بالأنظمة المدمجة والإنترنت الأشياء (IoT) ، حيث نتجه بسرعة نحو كل جديد في عالم التكنولوجيا، الذي يهدف إلى التطور في جميع المجالات لتسهيل استخدام المستخدم للأجهزة التي قد يتعرض لها في حياته اليومية.

ووفقًا لما هو معروف، فإن كلما زادت الخصائص والميزات والمتطلبات، زاد تعقيد النظام، كان من المتوقع أن يكون الأمر معقدًا للغاية، ولكن هذا ليس الوضع عند وجود النظام المدمج، خاصة نظام ARM المدمج، الذي تم استخدامه في مشروعنا.

نظرًا للتطور السريع جدًا، كان من الضروري أن يكون هناك نظام يمكنه مواكبة هذه التطورات، مما يجعل الأجهزة المتاحة قابلة للتكيف والتطور. وهنا يأتي دور التحديث عبر الهواء (FOTA) ، حيث أنه من بين أهم ميزاته أنه يربط جميع الأجهزة من خلال السحابة بالشركة الأم لتكون دائمًا متصلة بها لتمكينها من توفير تحديثات لأجهزتها باستمرار باستخدام تحديث البرامج عبر الهواء (OTA) ، حتى تبقى أجهزتها على موعد مع تطور التكنولوجيا. بالإضافة إلى ذلك، يمكن لـ FOTA مساعدة المستخدمين في الحفاظ على أجهزتهم خالية من عيوب البرمجيات وتنبيههم بوجود عيوب في الأجهزة والمخاطر المحتملة لمكونات الجهاز.

هذه هي الميزات التي جعلت صيانة وتحديث البرامج سهلاً وبسيطاً وأقل تكلفة بعد أن كان يستغرق وقتاً طويلاً وتكاليف ضخمة. ومجال السيارات هو واحد من المجالات التي تهتم بشدة بـ FOTA، لذلك كان فريقنا مهتمًا بالمشاركة في هذا المجال والمساهمة في دورة التطوير من خلال إضافة بعض ميزات الأمان وتسهيل استخدام هذه الميزات للمستخدم.