

Fayoum University

Engineering Faculty

Electrical Engineering Department



B. Eng. Final Year Project

Firmware Over the Air (FOTA)

By:

Kyrillos Adel Fahim,

Mark Ehab Tawfiq,

Tasneem Yaseen Mostafa,

Menna Ahmed Heshmat,

Nadine Ashraf Adeeb,

Sara Gouda Rohaim.

Supervisors:

Dr. Ahmed Mostafa

Date of examination

17/7/2022

DEDICATION

This project is wholeheartedly dedicated to our beloved parents, who have been our source of inspiration and gave us strength when we thought of giving up, who continually provide their moral, spiritual, emotional, and financial support.

To our brothers, sisters, relatives, mentor, friends, and classmates who shared their words of advice and encouragement to finish this study.

And lastly, we dedicated this book to the Almighty God, thank you for the guidance, strength, power of mind, protection and skills and for giving us a healthy life. All of these, we offer to you.

ACKNOWLEDGMENT

Above all, praise to **ALLAH**. By whose grace this work has been completed and never leaving us during this stage.

We would like to express our sincere gratitude to **Dr. Ahmed Mustafa**, Prof. of Electronics, Faculty of Engineering, Fayoum University, for his academic supervision and encouragement through the year.

We would also like to deeply thank **Eng. Ahmed Assaf**, General Manager at IMT School, for suggesting the idea of the project, mentorship, guidance and support during the year and **Eng. Omar Mekkawy** who had a crucial role mentoring us for this project to see the light.

Special thanks for our teaching assistants for helping us in our project with their great ideas and their knowledge **Eng. Demiana Emel**, **Eng. Ahmed Helmy** and finally the deceased **Eng. Amr El-Masry** who we feel grief for passing away, we wish he is in a better place. We will always remember him in our prayers.

DECLARATION

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Bachelor of Science in *Electrical Engineering* is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____

Registration No.: _____

Date: 17/7/2022.

ABSTRACT

Nowadays, any automotive vehicle has a very large number of Electronic Control Units (ECUs), it may have around 100 ECUs and approximately 100 million lines for software coding. These numbers are increasing rapidly since connected cars have been introduced.

It's suggested that by 2025, there will be 300 ECUs in a single car, to manage most of the functions and features within a car. After the vehicle's release, Original Equipment Manufacturer (OEMs) usually need to fix bugs, add new features, or update the ECUs' software. This requires the customer to visit the maintenance center periodically. Recalling the vehicles to do that will not be considered as a good option for OEMs as it negatively affects their reputation, user experience, unnecessary cost and time.

We aim to update the software of a car by flashing the code on the ECUs over the air without using any physical connection, FOTA enables cars to be upgraded remotely, without having the user to worry about a software-related recall or update.

Our solution fulfill security, proper timing and also solving real life problems which impact cost reduction for OEMs and user satisfaction and make our life easier.

Table of Contents

LIST OF FIGURES	VIII
List of Tables	X
1 INTRODUCTION	1
1.1. History of FOTA.....	1
1.2. Problem Definition.....	3
1.3. Key developments timeline.....	4
1.4. Key players and market.....	6
2 PROJECT OVERVIEW	9
2.1. Introduction.....	9
2.2. System Design (1)	9
2.2.1 Hardware components	9
2.2.2. Software Updates	10
2.3. System Design (2)	11
2.3.1. Hardware components.....	11
2.3.2. Software Updates	11
2.4. Our system	12
3 WEB.....	13
3.1. Introduction.....	13
3.2. Overview.....	15
3.3. Architecture.....	19
3.4. Screens	23
3.5. Team Members	24
4. COMMUNICATION PROTOCOLS	26
4.1. UART.....	26
4.2. CAN Bus.....	28
5. APPLICATION ECU	39
5.1. Introduction.....	39
5.2. Bootloader.....	39
5.3. Application.....	54
5.4. Some Concepts about Network	57
6. CRAPHICAL USER INTERFACE (GUI).....	59
6.1. What is GUI?	59
6.2. History of GUI:.....	59

6.3. GUI Layers:	60
6.4. Automotive GUI	61
6.5. GUI in our project	63
6.5.3.2. Disadvantages of using PyQt5	64
6.5.3.3. Advantages of using Tkinter	65
6.5.3.4. Disadvantages of using Tkinter	65
6.5.4. Controller used: Raspberry pi 3 model B	68
6.6. SPI TO CAN Module	69
6.7. Design Aspects	70
6.7.1. Our GUI system consists of	71
6.8.1. Welcome Pag	71
6.8.2. Login page	72
6.8.3. Signup page	73
6.8.4. Fill profile page	74
6.8.6. History tab	76
6.8.7. update tab	77
6.9. GUI Advantages	78
6.10. GUI Future work	78
7. SECURITY	79
7.1. Cryptography	79
7.2. HMAC	81
8. FUTURE WORK	84
8.1. Live Diagnostics	84
8.2. Seamless FOTA	84
8.3. Delta file	86
APPENDIX A	89
APPENDIX B	93
APPENDIX C	101
References	102

LIST OF FIGURES

Figure 1:FOTA mangment process.....	1
Figure 2:Automotive software recalls.....	4
Figure 3:Key developments timeline	5
Figure 4: Automotive value	6
Figure 5:HUM -device	8
Figure 6: Block Diagram 1	9
Figure 7: Block Diagram 2	11
Figure 8:MTV	15
Figure 9:Web files.....	21
Figure 10:Application files	21
Figure 11:login page	23
Figure 12:Registration page	24
Figure 13:Team members	24
Figure 14:Statistics screen	25
Figure 15:Upload screen	25
Figure 16:Connection of UART	26
Figure 17:UART Frame	27
Figure 18: CAN network	28
Figure 19:Transmit Mailbox states	31
Figure 20:Receive FIFO States	32
Figure 21>Error frame	35
Figure 22>Error frame	35
Figure 23:Data and remote frames.....	36
Figure 24:Node in loopback mode.....	38
Figure 25:Node in silent mode.....	38
Figure 26: Floating gate flash memory cell	39
Figure 27:Flash memory sections	47
Figure 28:RCC_CSR Register bits	49
<i>Figure 29:Branching Sequence flowchart</i>	<i>50</i>
<i>Figure 30:Bootloader code flowchart</i>	<i>52</i>
Figure 31:Big endian and little endian.....	54
Figure 32:IR sensor.....	54
Figure 33:LM35 Temperature sensor	55
Figure 34:First commercial GUI.....	59
Figure 35:GUI Layers	60
Figure 36:Automotive GUI.....	61
Figure 37:GUI Application	62
Figure 38:Signals and slots	66

Figure 39:QWidget interface	67
Figure 40: Raspberry pi 3 model B	68
Figure 41: VNC server	68
Figure 42:SPI to CAN module.....	69
Figure 43:The connection of Rresspberry bi to SPI to CAN module	70
Figure 44:Welcome page	71
Figure 45:login page	72
Figure 46:Signup page	73
Figure 47:Fill profile page	74
Figure 48: FOTA page	75
Figure 49:History tab	76
Figure 50:Update tab.....	77
Figure 51:GUI Advantages	78
Figure 52:Cryptography system.....	79
Figure 53:Three type of cryptography	81
Figure 54:HMAC process	83
Figure 55: Diagnostics flow	84
Figure 56:A\B swap process	85
Figure 57:Delta file algorithm.....	87
Figure 58:Generating data file	88

List of Tables

Table 1:Flash module organization.....	42
--	----

1 INTRODUCTION

1.1. History of FOTA

Firmware Over-The-Air (FOTA) is a Mobile Software Management (MSM) technology in which a mobile device's operating firmware is upgraded and updated wirelessly by its manufacturer. FOTA-capable phones directly download the updates from the service provider. Depending on link speed and file size the process typically takes three to 10 minutes.

FOTA took 4 management stages process as shown in figure 1:

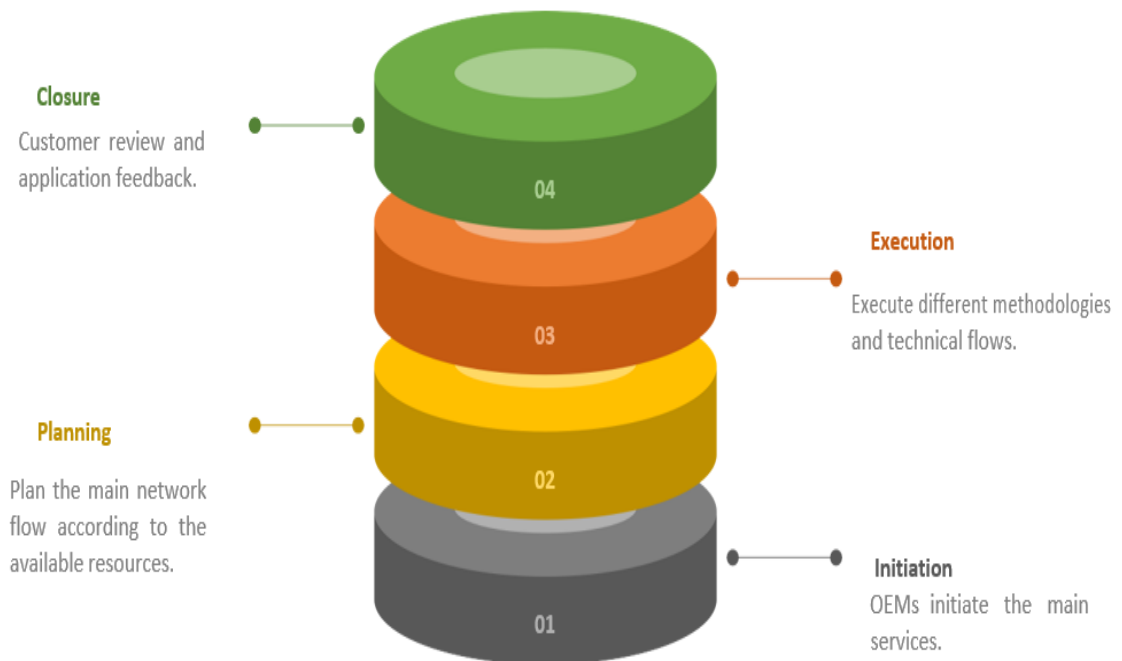


Figure 1:FOTA mangment process

Through these stages FOTA shall facilitate:

- Enables fabricators to patch glitches in new systems.
- Allows OEMs to be able to send and install new software updates and features remotely after customers have bought the cars.

FOTA represent a tremendous opportunity for automotive manufacturers, with a clear route to cost savings, particularly during vehicle warranty periods, revenue gains from the sales of new features and services, and a more engaged relationship with clients.

So, the management process took its time to bring out the whole fully experience to the customer and the OEMs, the imperative to move towards FOTA stems from the autonomous, connected, and electric vehicles trend in the automotive sector. As these vehicles are using increasingly complex hardware and operating systems, vendors would need a mechanism to repair, manage and enhance any aspect of vehicle efficiency.

In order to achieve that it was about the first steps toward any idea a good strategy to overcome the complexity of such application by dividing the process within different structures.

Next, we will summarize the historical keys of development.

1.2. Problem Definition

Today, an automotive vehicle has on average 100 **Electronic Control Units** (ECU) and about 100million lines of software code. At an average cost of \$10 per line, these electronic systems aren't cheap and recalls lead to multi-billion-dollar losses and also leave a dent on manufacturers' reputations. This situation will be worse as this number is growing rapidly such that experts suggest that there will be 300 ECUs in a car in 2025 so that, OEMs must reduce the impact of product recalls by managing the software efficiently over the life cycle of the vehicle.

So, there are several constrains to update the cars' software:

1. **Time:** upgrading the firmware could take many days.
2. **Complexity:** upgrading the firmware must be completely handled at the application layer.
3. **Cost:** to upgrade the firmware of the car, people must go to maintenance center.

1.3. Key developments timeline

In the automobile industry, in September 2012, Tesla delivered the first FOTA software update for their 'Model S' vehicles. The business published an update and use either car's integrated 3 G network link or a Wi-Fi signal from the customer's home network.

Other automotive OEMs such as General Motors, BMW, Volvo, Detroit Diesel Organization and Mercedes-Benz have begun to deliver updates to OTA from 2017-2018. Ford also plans to deliver updates to OTA by 2022. Agricultural machinery manufacturers including John Deere, AGCO Company, and CNH Industrial have begun providing simple OTA updates for their farm machinery along with automotive OEMs.

As shown in figure 2 number of automotive software recalls.

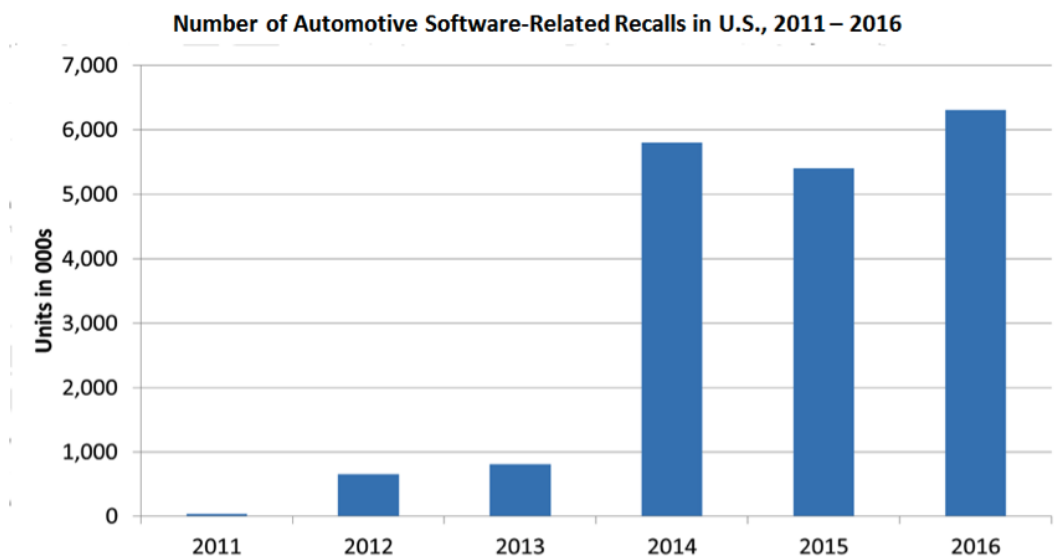


Figure 2:Automotive software recalls

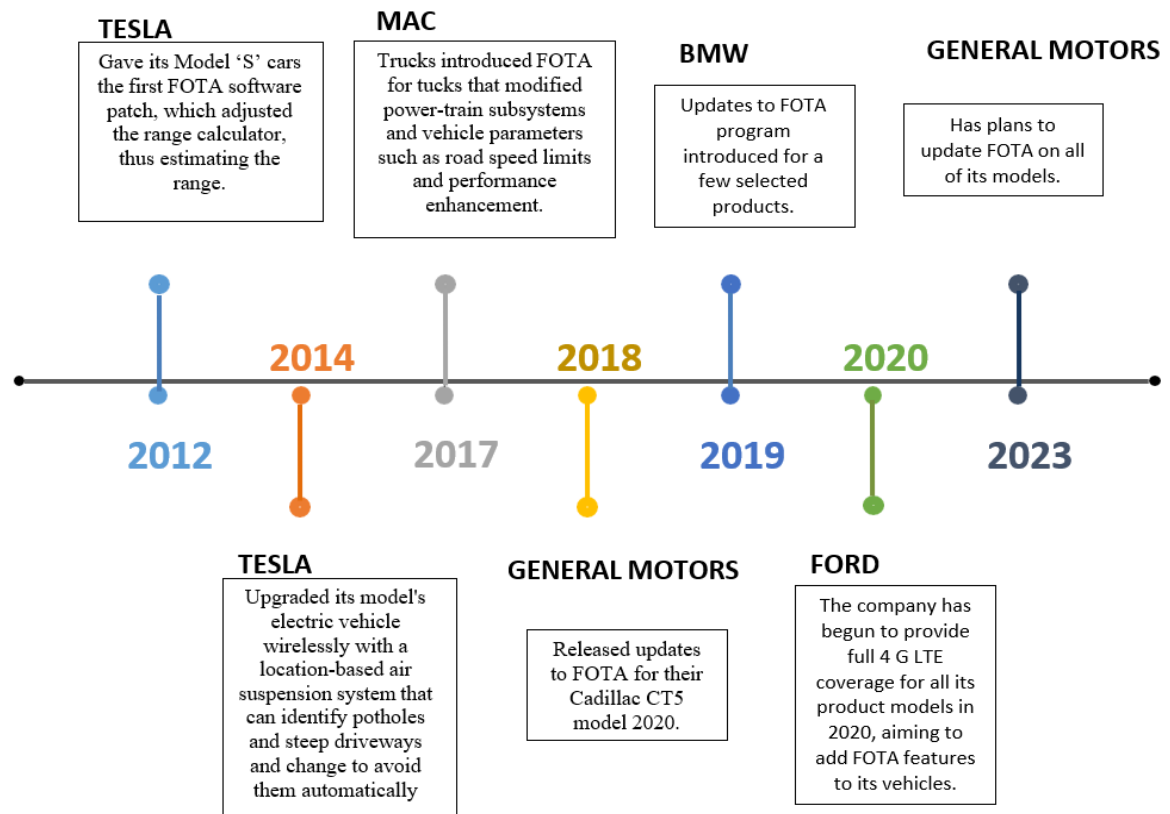


Figure 3:Key developments timeline

In the automotive industry, FOTA updates go hand in hand with self-driving and connected vehicles. IOT updates will be an important part of the near future, according to industry analysts, by 2030, 98 percent of cars sold worldwide are projected to be connected vehicles. The value produced by next-generation cars (in terms of the cost of the vehicle) will be totally different than the current ones; the software will be a key selling point in next-generation cars. To sell their vehicles in the near future, automotive OEMs will deliver robust software along with hardware systems.

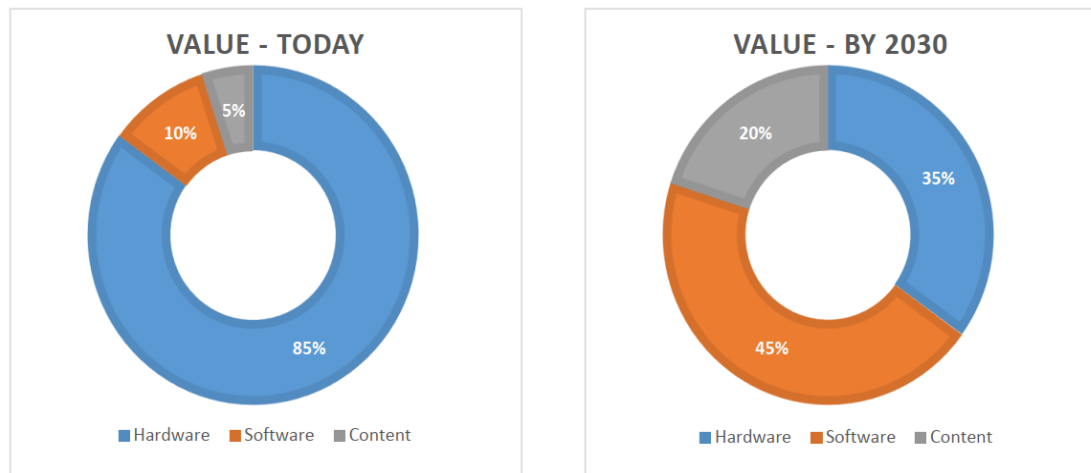


Figure 4: Automotive value

As shown in figure 4 what we expect the percentage of SW in cars by 2030.

Apparently, industry forecasts say that FOTA 's future in automotive is looking promising. This is mainly relevant to the automotive industry which is shifting its gear towards software-driven autonomous systems development.

The Research and Economies reported that the Automotive Over the Air (FOTA) updates market is expected to expand at a CAGR of 58.15 per cent over the period 2018-2022. And many of the largest automobile players are already on the path of making FOTA in automotive systems seamless and stable upgrade, a fact.

1.4. Key players and market

The over-the-air update is a means of delivering new applications, firmware, and other cloud-based upgrading facilities. The FOTA updates were only available for smartphones until recent time; however, with the significant increase in the number of connected cars around the globe and vehicles becoming more software-dependent, the need for FOTA updates has increased significantly. Software upgrades have long been achieved by local software upgrade procedures, whereby the car is sent to a vendor / mechanic who upgrades the software afterwards.

Tesla Motors has started to deliver FOTA capability for its electric cars, which has shaken the automotive industry. Other global automakers, such as BMW, Mercedes-Benz and General Motors, then started investing in providing FOTA capabilities for their connected vehicles. FOTA updates are supposed to help OEMs save a large sum they spend as a result of car recalls. It also gives car owners the ease of updating all their software without having to visit the location of the dealer.

Robert Bosch GmbH, NXP Semiconductors N.V, Verizon Communications, Inc., Continental AG, Infineon Technologies AG, Qualcomm Incorporated, Intel Corporation, HARMAN International, Airbiquity Inc, Aptiv, HERE Technologies, BlackBerry QNX Software Systems Limited, Garmin Ltd. and Intellias Ltd are some of the major players on the market.

The global FOTA automotive updates market is characterized by the presence of numerous national, regional, and local suppliers. The market is highly competitive, competing with all players to achieve full market share.

According to the report of Market Research Future (MRFR), as of 2018, HARMAN International, Aptiv, QNX Computer Systems Limited, Intel Corporation and Airbiquity Inc. are some of the major players on the global FOTA automotive update market. By growth, mergers & acquisitions, and by providing a broad product range, these businesses continue to maintain their strong global grip.

Another main announcement in the market that Qualcomm Technologies, Inc., a subsidiary of Qualcomm Incorporated, previously confirmed that its 5G test networks would be extended to include new end-to-end over-the-air (FOTA) systems for both (mmWave) and sub-6 GHz bands.

While Verizon Introduces Hum, an all-in-one connected car solution that provides users with vehicle position, diagnostics, on-road assistance, travel history and more. It's the first 4G LTE connected car solution to have Google Assistant built-in.



Figure 5: HUM -device

2 PROJECT OVERVIEW

2.1. Introduction

In this chapter, a quick overview of the project features and flow will be explained in detail, in addition to the block diagram of the system. The solution consists of Software Updates. The flow will be discussed in detail.

2.2. System Design (1)

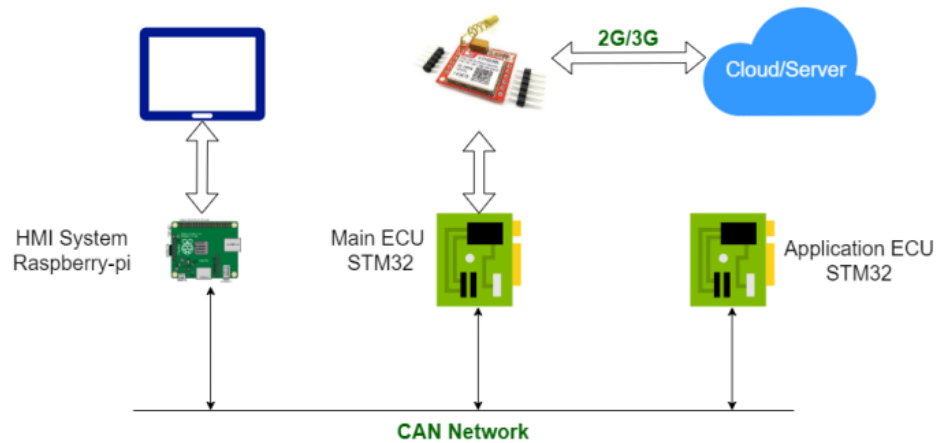


Figure 6: Block Diagram 1

2.2.1 Hardware components

This system consists of 3 main controllers as figure 6 shows:

- Main ECU: It is connected to a SIM800L GSM Module and acts as the gateway to the server.
- Application ECU: It acts as a faulty node whose software needs to update or have a feature added to it.
- HMI System: It consists of Raspberry-pi Model 3B and an HDMI Touch Screen. It acts as an interface to the user.

2.2.2. Software Updates

- 1- The OEM uploads a new software to the server.
- 2- On the next ignition cycle in the vehicle, the main ECU requests the software versions IDs from the server, it also requests the software version IDs from the application ECU.
- 3- The server replies with them in a semantic version format.
- 4- The main ECU compares the versions.
- 5- If an update is found it sends a CAN Message to the GUI requesting for the user acceptance for the update. In case no update is found, nothing happens.
- 6- Once the user accepts or rejects the update, the GUI sends a CAN Message in both cases.
- 7- In case of user acceptance, the main ECU requests the new hex file from the server and sends an update request message through CAN to the application ECU to perform a soft reset and start executing the bootloader.
- 8- The server sends the hex file to the Main ECU.
- 9- The Main ECU validates the file for errors.
- 10- The Main ECU sends the hex file through CAN Network to the application ECU, it also sends the update progress to the GUI to display it.
- 11- The bootloader receives the file and checks for any error, flashes it in the flash memory then jumps to the new application.

2.3. System Design (2)

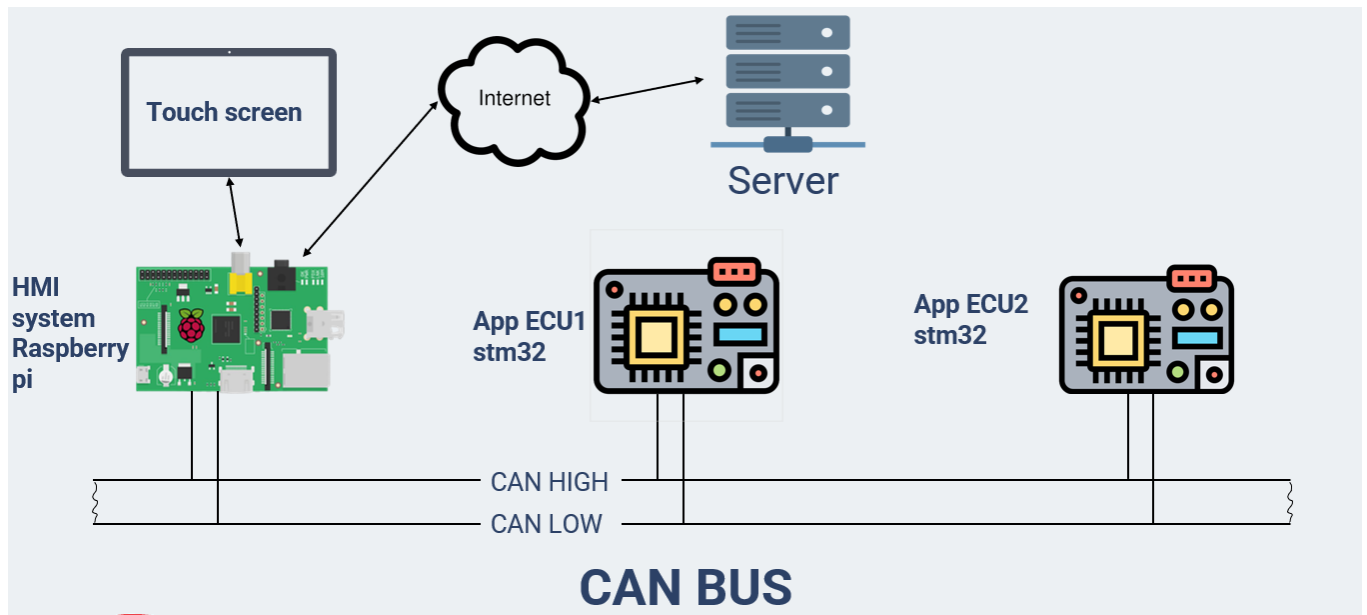


Figure 7: Block Diagram 2

2.3.1. Hardware components

This system consists of 2 main controllers as figure 7 shows:

- Raspberry pi: acts as the gateway to the server.
acts as an interface to the user by connected to HDMI touch screen.
- Application ECU: It acts as a faulty node whose software needs to update or have a feature added to it.

2.3.2. Software Updates

- 1- The OEM uploads a new software to the server.
- 2- On the next ignition cycle in the vehicle, the Raspberry pi requests the software versions IDs from the server, it also requests the software version IDs from the application ECU.
- 3- The server replies with them in a semantic version format.
- 5- The raspberry pi compares the versions.

- 6- If an update is found it sends a CAN Message to the GUI requesting for the user acceptance for the update. In case no update is found, nothing happens.
- 7- Once the user accepts or rejects the update, the GUI sends a CAN Message in both cases.
- 8- In case of user acceptance, the raspberry pi requests the new hex file from the server and sends an update request message through CAN to the application ECU to perform a soft reset and start executing the bootloader.
- 9- The server sends the hex file to the raspberry by.
- 10- The raspberry pi validates the file for errors.
- 11- The raspberry pi sends the hex file through CAN Network to the application ECU, it also sends the update progress to the GUI to display it.
- 12- The bootloader receives the file and checks for any error, flashes it in the flash memory then jumps to the new application.

2.4. Our system

We chose the second design because it has less number of ECUs and therefore less complexity.

In our system the raspberry pi connected to:

- NOIP which connected to the web backend.
- TFT touch screen to act as the GUI system to interface the user.

SPI to CAN module then connected to CAB bus to connect with the application ECU.

Chapter Three

3 WEB

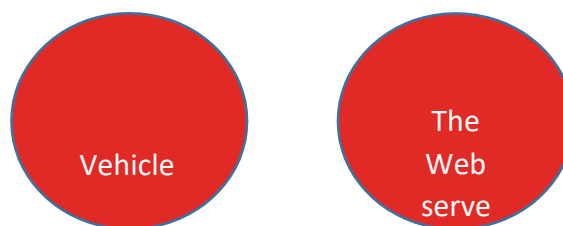
3.1. Introduction

Internet of Things (IoT) is a wide set of technologies and use cases that no clear definition would wrap it. In our case, we may define it as the use of network-connected vehicles, to communicate with the OEM's servers.

“IoT is a tool or a method. The essential quality of an IoT system lies in how much you can innovate by effectively using this tool.”

Because of outstanding opportunities IoT promises, OEMs seek for the inclusion of it into their business models. However, when it comes to reality, this brilliant idea appears too complicated to be implemented. Given the number of vehicles, conditions needed to make it work, and the security of such model. In other words, the problem of establishing a reliable architecture of Internet of Things inevitably enters the stage.

3.1.1. Top-level components



In order to establish the connection between the web server and the vehicle, some metadata must be planned first. Each vehicle can provide or consume various types of information. Each form of information might best be handled by a different backend system, and each system should be specialized around the data rate, volume, and preferred API.

For the vehicle, it will have some factory settings in its ROM. Such settings can be: vehicle identification code, component identification code, API end-points, secret keys used to exchange secure data with the server, protocols used, etc.

These settings should remain constant for as long as we could, and therefore a very strict considerations must be taken into place.

NOIP

We will get the domain name from NOIP. Then, We will download some scripts on the device and run these scripts periodically and get the IP which works dynamically and send these IP addresses to the NOIP server.

3.2. Overview

3.2.1. Technology

First component of the web is to choose the technology. We decided to work with Django framework. Django is a free and open-source, Python-based web framework that follows the model–template–views (MTV) architectural pattern. Django's primary goal is to ease the creation of complex, database-driven websites. The framework emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings, files, and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

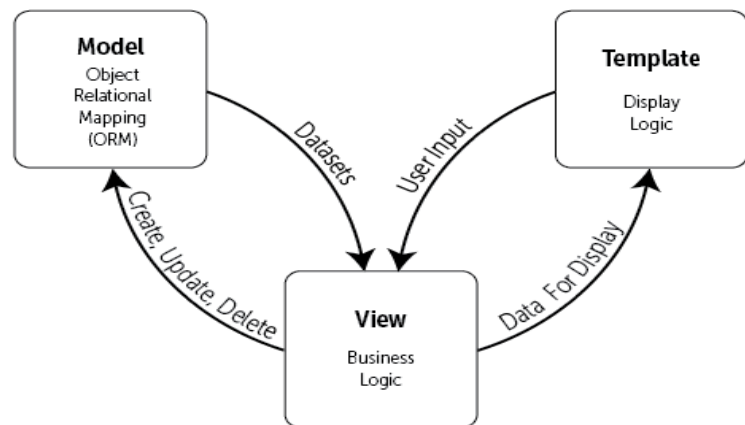


Figure 8:MTV

As shown in figure 7.

3.2.2. Why Django?

3.2.2.1. Built With Python So Easy to Learn

We all know that Python is simple, easy to read, and easy to learn. A lot of beginners choose this language as their first programming language for coding because of its simplicity and easy learning curve. Colleges and universities are using this language to teach coding to the students. Not just beginners but also tech experts are using this language for data science, machine learning, and in various other fields.

Python is a pretty much stable language and Django inherits a lot of key benefits of Python. If we look at the core stuff of Django all the exceptions files and codes are written in Python. So, learning Django is also easy if you know how to code in Python.

3.2.2.2. Cross-Platform

Django is a portable framework and you can run its code on any platform including PC, Mac, Windows, Linux, etc. The cross-platform nature of this framework allows developers to support all the development and production environment. Django has a layer called ORM (object-relational mapper) between the developer and the database. With the help of this layer, you can migrate the whole project to other major databases with few lines of change in the code.

3.2.2.3. Open Source and Huge Community Support

Django is a free and open-source framework available on Github. There are almost 2k+ contributors and many more are joining every day. It is supported by the huge community of developers and the code is always updated by the developers who use it. A lot of new libraries are also introduced by the community to solve coding related issues that developers often face while building a project.

If you search for something that how a specific thing can work in Django, chances are higher that your problem is already solved by some other developer in the community and you will get the solution easily as soon as your problem arises. Plenty of mailing lists, blogs, documentation, Slack channels, meetups, workshops, and other online resources are available for this framework.

3.2.2.4. Security

One of the best things about Django is that you can build the application at a faster speed and deliver it without compromising the security of the application. Security features are enabled by default in this framework. It has built-in protection for some common security issues such as cross-site scripting, request forgery, clickjacking, and SQL injection. Django releases new security patches quite often and it immediately responds to the security vulnerabilities and alerts other frameworks.

3.2.2.5. Built-in Admin UI

In most of the frameworks you need to create the admin panel on your own and that takes a lot of time. Django offers a fully-featured web interface that is generated automatically for every app you build. The admin panel is well structured and it allows developers to create/update/delete users and any other database objects specific to the app. As per your need, you can customize or modify the admin panel user interface and add a dashboard using the third-party applications and wrappers.

You also get permission and authentication modules out of the box. You don't need to spend weeks or days to build it from scratch. In most of the apps for building user profiles, you need various details such as username, email, address, phone number, etc. Django provides most of the necessities that you need for the user profile. This kind of huge support is not available in a lot of frameworks or libraries.

3.2.2.6. DRY (Don't Repeat Yourself)

In programming code-reusability or following the DRY principle is really important, especially when you are updating the code regularly. When you follow the DRY principle, you just don't use the existing code, you also avoid the unnecessary lines of code, bugs, or errors in the application. With the DRY principle, you can get most out of very little code and that saves a lot of time of developers when it comes to making your code work or modifies the code for any reason.

In most of the framework, you need to make efforts to make your code DRY compliant, and every time it's not possible to keep checking your code, especially when you're working in a team. Django follows the DRY principle and it is designed in such a way that you have to go out of your way to violate the DRY principle. This feature allows you to re-use the existing code and focus on the unique one.

3.2.2.7. Scalable and Reliable

Today every startup or company is concerned about the scaling of the application. What if the website hits scale and it needs to handle the heavy traffic and large volume of information? Surely you need to use the framework that can handle the huge amount of data. Well, Django is able to tackle the project of any size either it's a small scale web application or it's a high loaded web application.

Django comes with a series of wired components, ready to go by default. These components are decoupled, so as per the requirement or specific solutions in an application, development can be scaled up or scaled down by replacing or unplugging the components.

Django is a very popular and widely used web application framework across the industries. This is the reason a lot of cloud service providers are taking all measures to deploy the application fast and easily on their platforms. Take the example of Heroku. Once it is set up, you can enable the deployment and manage the application with a single command from any authorized developer. Django experts working in these areas develop a more functional, reliable, and efficient application.

3.2.2.8. Good Documentation

For quick reference, good documentation for any framework or language matters a lot especially when we get stuck somewhere during the development phase of any project. Django provides well-organized documentation with example code which is very helpful in building different kinds of real-world applications. The documentation is good for quick reference if you are building some features in your application or you are stuck with some issues in your code.

3.2.3. Apache web server

File servers, database servers, mail servers, and web servers use different kinds of server software. Each of these applications can access files stored on a physical server and use them for various purposes.

The job of a web server is to serve websites on the internet. To achieve that goal, it acts as a middleman between the server and client machines. It pulls content from the server on each user request and delivers it to the web.

The biggest challenge of a web server is to serve many different web users at the same time each of whom is requesting different pages. Web servers process files written in different programming languages such as PHP, Python, Java, and others.

They turn them to static HTML files and serve these files in the browser for web users.

Apache is a free and open-source software that allows users to deploy their websites on the internet. It is one of the oldest and most reliable web server software maintained by the Apache Software Foundation, with the first version released in 1995.

3.3. Architecture

Some components need to be designed before starting to implement the project. And figuring out those components needs some planning itself.

We saw some here:

1. The OEM's UI and API
2. The vehicle's API.
3. File system structure
4. Database model

3.3.1. Server UI Screens

Planning all the screens that should be shown on the UI. Each point will be displayed with a screenshot for the final result.

1. Login page:

Login form to login to the web site.

2. Diagnostics page:

This page shows the diagnostics data sent by vehicles to be reviewed and collected for bug fixes.

3. Upload page:

It has all the info needed for firmware upload and description for it.

4. Users or team page:

Views all users or team members working on firmwares.

5. Admin panel

Django provides a built-in admin module which can be used to perform CRUD operations on the models. It reads metadata from the model to provide a quick interface where the user can manage the content of the application.

3.3.2. Django Project Structure and File Structure

Django makes use of a directory structure to arrange different parts of the web application. It creates a project and an app folder for this.

When we create a Django project, the Django itself creates a root directory of the project with the project name you have given on it. It contains the necessary files that would provide basic functionalities to your web applications.

1. Manage.py

This file is used as a command-line utility for our projects. We will use this file for debugging, deploying, and running our web applications.

The file contains the code for running the server, make migrations or migrations, and several other commands as well, which we perform in the code editor.

2. Project files

As shown in figure 9

a. `__init__.py`

This is an empty file as you can see below in the image. The function of this file is to tell the Python interpreter that this directory is a package and involvement of this `__init__.py` file in it makes it a python project.

b. `settings.py`

It contains the Django project configuration.

c. `urls.py`

This file tells Django that if a user comes with this URL, direct them to that particular website or image whatsoever it is.

d. `wsgi.py`

WSGI stands for **Web Server Gateway Interface**, it describes the way how servers interact with the applications.

e. `asgi.py`

ASGI works similar to WSGI but comes with some additional functionality. **A** stands for **A**synchronous. It is now replacing WSGI.

3. Application files

Django uses the concept of Projects and apps for managing the codes and presents them in a readable format. A Django project contains one or more apps within it, which performs the work simultaneously to provide a smooth flow of the web application.

As shown in figure 10.

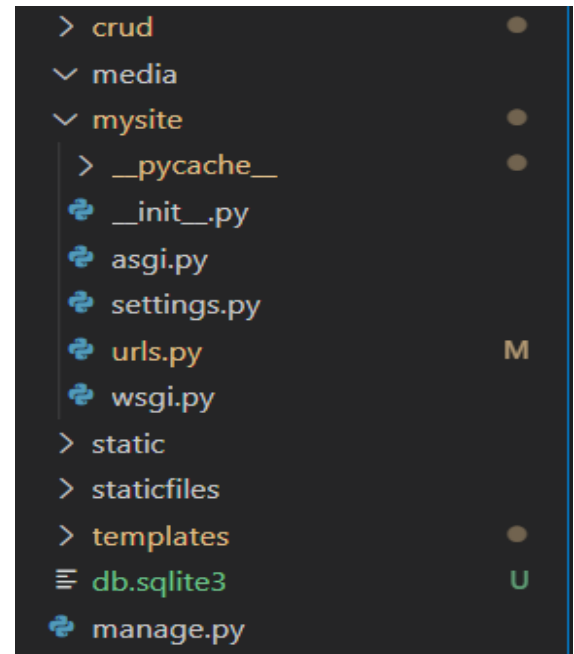


Figure 9: Web files

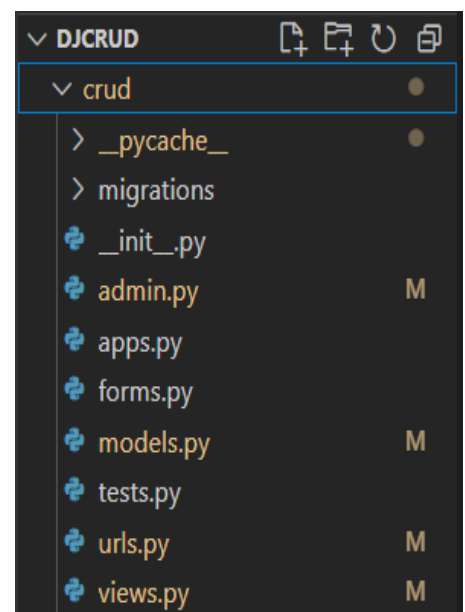


Figure 10: Application files

a. `_init_.py`

This file provides the same functionality as that in the `_init_.py` file in the Django project structure.

b. `admin.py`

`admin.py` file is used for registering the Django models into the Django administration. It is used to display the Django model in the Django admin panel. It performs three major tasks (registration model, creating a Superuser, logging in and using the web application).

c. `apps.py`

`apps.py` is a file that is used to help the user include the application configuration for their app.

d. `models.py`

`models.py` represents the models of web applications in the form of classes. It is considered the most important aspect of the App file structure.

e. `views.py`

views provide an interface through which a user interacts with a Django web application. It contains all the views in the form of classes.

f. `urls.py`

`urls.py` works the same as that of the `urls.py` in the project file structure. The primary aim being, linking the user's URL request to the corresponding pages it is pointing to.

g. `tests.py`

`tests.py` allows the user to write test code for their web applications. It is used to test the working of the app.

3.3.3. Database model

We have two main DB models that we have to take in mind:

1. Document (update file):

- a. description: description of uploaded document.
- b. document: the hex file we want to upload
- c. uploaded_at: the date and time of the uploaded file.

2. Member:

- a. firstname.
- b. lastname.
- c. dobile_number.
- d. description.
- e. location.
- f. date.
- g. created_at
- h. updated_at.

3.4. Screens

3.4.1. Login page

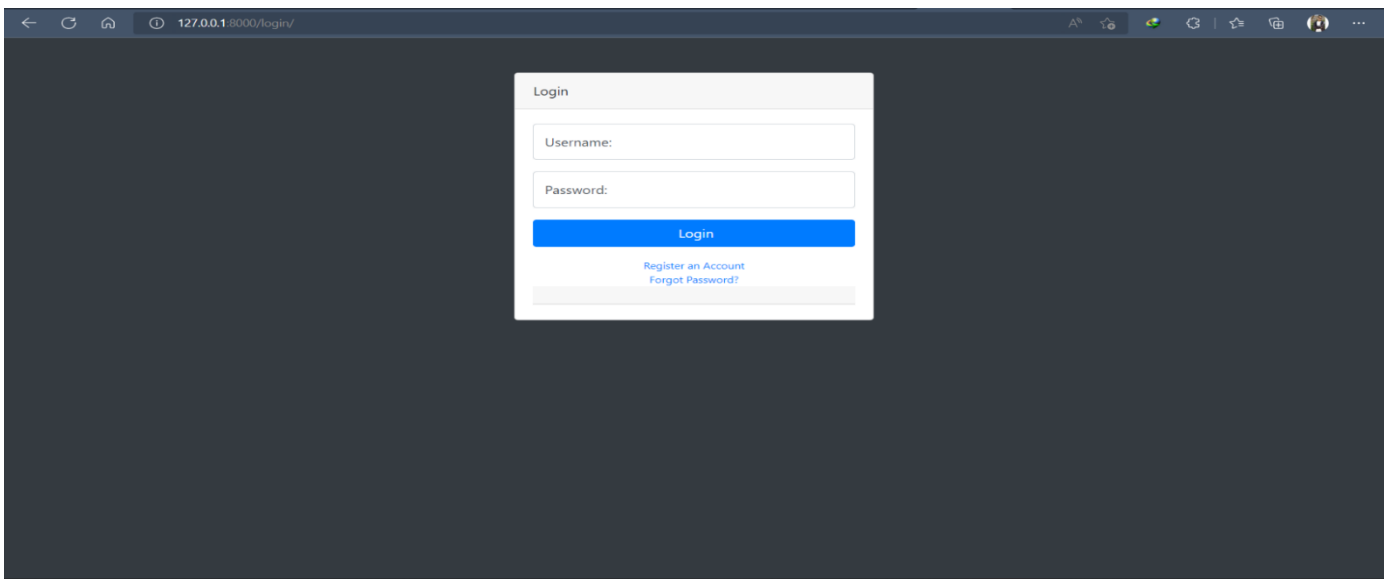


Figure 11:login page

3.4.2. Registration

Register an Account

First name: Last name:

Username: Email address:

Password: Password (again):

[Register](#)

[Login Page](#)
[Forgot Password?](#)

Figure 12:Registration page

3.5. Team Members

FOTA

Search for...

Dashboard

Team Members

File Upload

Users

More Feature

Team Members /

List

ADD

Show 10 entries

Search:

First Name	Last Name	Mobile Number	Description	Location	Created At
Sara	Gouda	+2011111111	Web developer	Egypte	10-07-2022 22:06:49

Showing 1 to 1 of 1 entries

Previous 1 Next

Figure 13:Team members

3.5.1. Diagnostics data

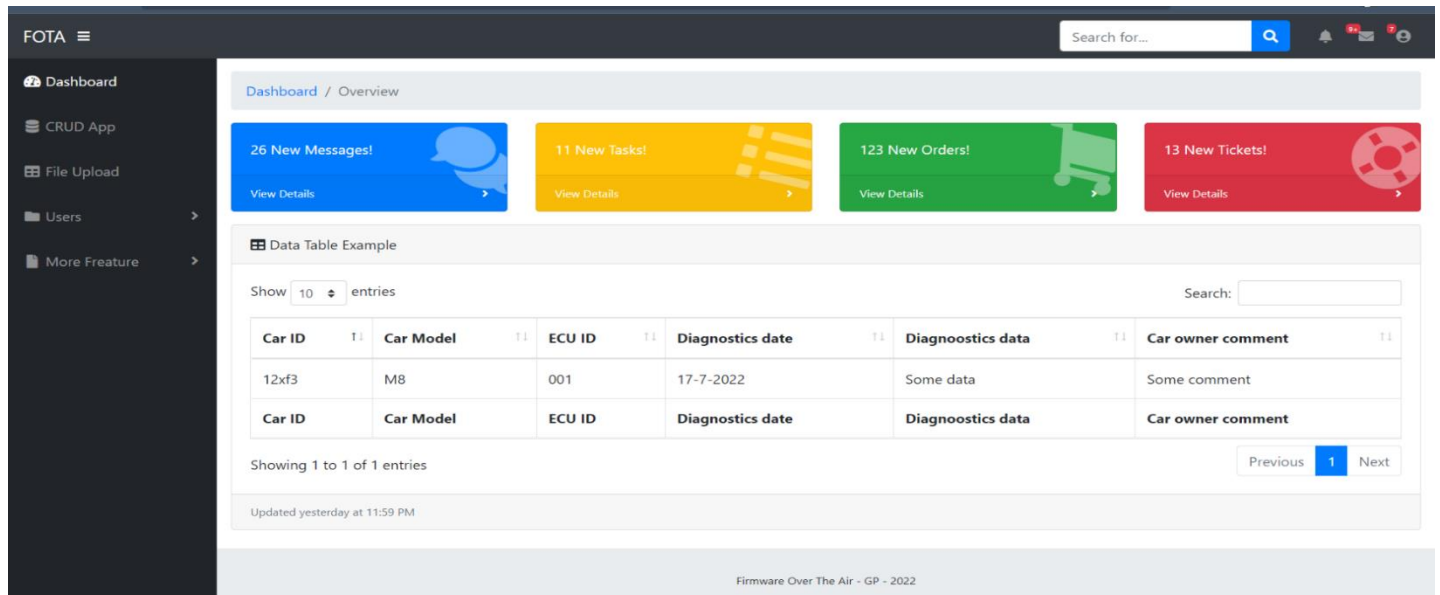


Figure 14: Statistics screen

3.5.2. Upload Screen

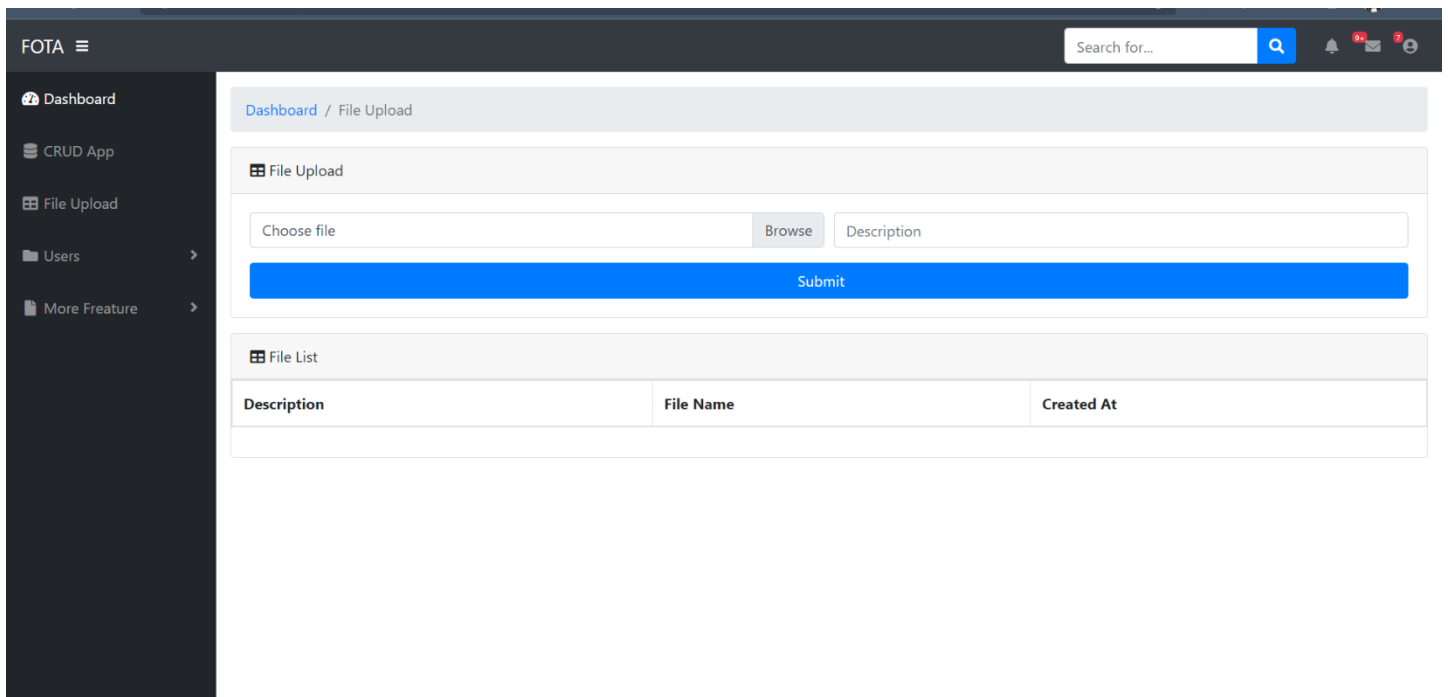


Figure 15: Upload screen

4. COMMUNICATION PROTOCOLS

4.1. UART

UART stands for Universal Asynchronous Receiver Transmitter, it's a dedicated hardware associated with serial communication. The hardware for the UART could be a dedicated IC or it could be an integrated circuit on the microcontroller. Unlike SPI and I2C, that are just communication protocols.

UART is one of the most commonly used Serial Communication techniques as it's very simple. Today, UART is being used in many applications like GSM, Modems, GPS Receivers, Wireless Communication Systems, Bluetooth Modules, and GPRS, RFID based applications etc.

Most of the microcontrollers have dedicated UART hardware built in to their architecture. The main reason for integrating the UART hardware in to microcontrollers' architecture is that it is a serial communication and requires only two wires for communication, one for transmission and the other for reception.

4.1.1. Advantages of UART

As shown in figure 16

1. Serial protocol

Serial protocols are much better than parallel protocols as parallel protocols have a lot of problems such as data skew, high cost, high complexity, and interference between wires due to magnetic field.

2. This protocol requires only two wires for full duplex data transmission (apart from the powerlines).
3. No need for clock or any other timing signal.
4. Parity bit ensures basic error checking is integrated in to the data packet frame.

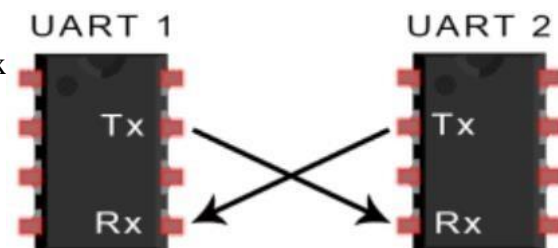


Figure 16: Connection of UART

4.1.2. UART Frame

As figure 17 shows

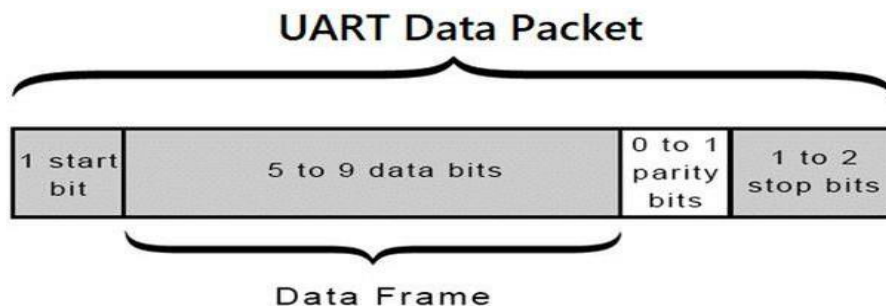


Figure 17:UART Frame

1) Start Bits

The IDLE state of the UART transmission line is high voltage when it's not transmitting data. To start the transfer of data, the transmitting UART pulls the transmission line from high to low for only one clock cycle. When the receiver detects the high to low voltage transition, it begins to read the bits in the data frame at the frequency of the baud rate.

2) Data Bits

The data bits contain the actual data that is being transferred. Its length can vary from 5 bits up to 8 bits long if a parity bit is used. If there is no parity bit, the data length can be 9 bits long. In most cases, the data is sent with the least significant bit first.

3) Parity Bits

Parity describes the evenness or oddness of a number. The parity bit is a way for the receiver of the UART to detect if any data has been changed during data transmission. Data could be changed due to mismatched baud rates, electromagnetic radiation, or long-distance data transfers. After the receiver reads the data frame, it counts the number of ones in it and checks if there is an even or odd number of ones. If the parity bit is a 0 this indicates even parity, which means that the total number of ones in the data should be an even number. If the parity bit is 1 which indicates odd parity means that the total number of ones in the data should be an odd number. If the parity bit matches the data, the UART knows that the transmission is free

of errors and has been done successfully. But if the parity bit is the frame is 0, and the total number of ones in the data is odd; or the parity bit is a 1, and the total number of ones is even, the UART knows that an error occurred during transmission and bits in the data frame have changed.

4) Stop Bits

To signal the end of the data packet, the transmitter signals the data transmission line from a low voltage to a high voltage to make it in the IDLE state again for at least two-bit durations.

4.2. CAN Bus

4.2.1. What is CAN?

CAN stands for Controller Area Network

The CAN Bus is a serial two-wire half-duplex communication that allows multiple nodes in a system to communicate efficiently with each other. Each node is capable of sending and receiving messages, not all nodes can be communicating at once. All nodes receive all broadcast data from the bus then each node decide whether or not that data is relevant according to the filter masks of each node as shown in figure 18.



Figure 18: CAN network

Any node wants to send a message it puts it on the bus with specific ID, then each node from the other nodes will check whether this ID is included in its filters or no, if yes it reads the message from the bus, otherwise it ignores it.

CAN protocol is already widely used in vehicle and vessel internal components. In recent years, it has seen adoption in data communications and control in industry field.

4.2.2. Why did we choose CAN?

- **Speed variety**

- **Low Speed**

CAN network offers low signal transfer rates that ranges between 40 kbps to 125 kbps.

- **High Speed**

CAN offers high signal transfer rates between 40 kbps and 1 Mbps. High-speed CAN networks are terminated at both ends of the bus line with a 120-ohm resistor between CAN high and CAN low lines.

The lower signaling rates allow communication to continue on the bus, even when a wiring failure takes place, while high-speed doesn't.

- **Low cost**

When the CAN protocol was created, its aim and primary goal was to make faster communication between modules and electronic devices in vehicles while decreasing the amount of wiring (and the amount of copper) necessary.

- **Built-in Error Detection**

Error handling is built in the CAN protocol, each node can check for its errors during transmission and maintain its own error counter. When errors are detected nodes transmit a special Error Flag message and will destroy the offending bus traffic to prevent the error from spreading through the whole network. Even the node which is generating the fault will detect its own error in transmission, and raise its error counter, this eventually led the device to "bus off" and cease participating in the CAN network traffic. In this way, CAN nodes can both detect errors and prevent faulty devices from creating any useless bus traffic.

- **Flexibility**

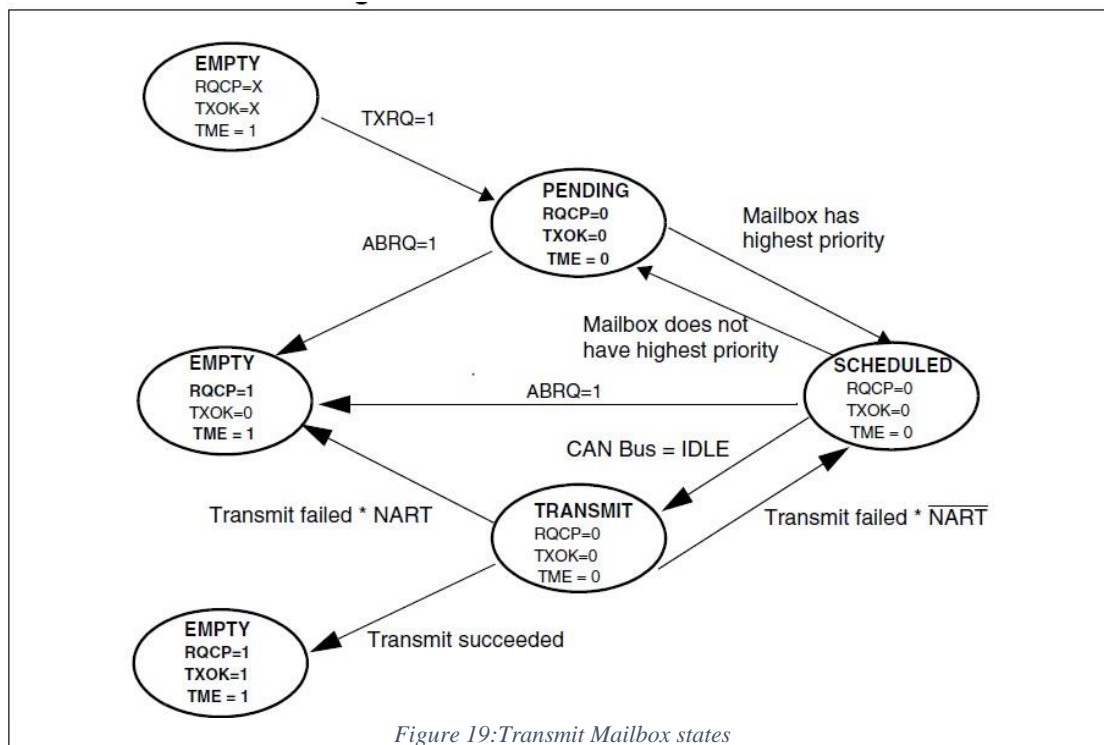
To understand the flexibility of the CAN bus protocol in communications, we need to know the difference between message-based protocols and address-based and. In address-based communication protocol, nodes communicate directly with each other by configuring themselves onto the same address.

The CAN bus protocol is a message-based communication protocol. In this type of protocol, nodes on the bus have no address to identify them. As a result, nodes can easily be added or removed without performing any software or hardware updates on the system. This feature makes it easy 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.

4.2.3. CAN specifications

- Wired
- Serial
- Asynchronous
- Multi-Master No Slave (MMNS)
- Half duplex

4.2.4. CAN Transmission Handling



As Shown in figure 19

1. In order to transmit a message, the application must select one empty transmit mailbox, setup the identifier, the data length code (DLC) and the data before requesting the transmission by setting the corresponding TXRQ bit (Transmit mailbox request) in the CAN_TiXR register ^[1].

¹ See APPENDIX B

2. When the mailbox left empty state and after the TXRQ bit has been set, the mailbox enters pending state and waits to become the highest priority mailbox (in our case the highest priority is defined by transmit request order) As soon as the mailbox has the highest priority it will be scheduled for transmission.
3. The transmission of the message will start (enter transmit state) when the CAN bus becomes idle, then once the mailbox message has been successfully transmitted by setting the RQCP (Request Complete) and TXOK (Transmission Ok) bits in the CAN_TSR register the mailbox becomes empty and we can use it again.

4.2.5. CAN Reception Handling

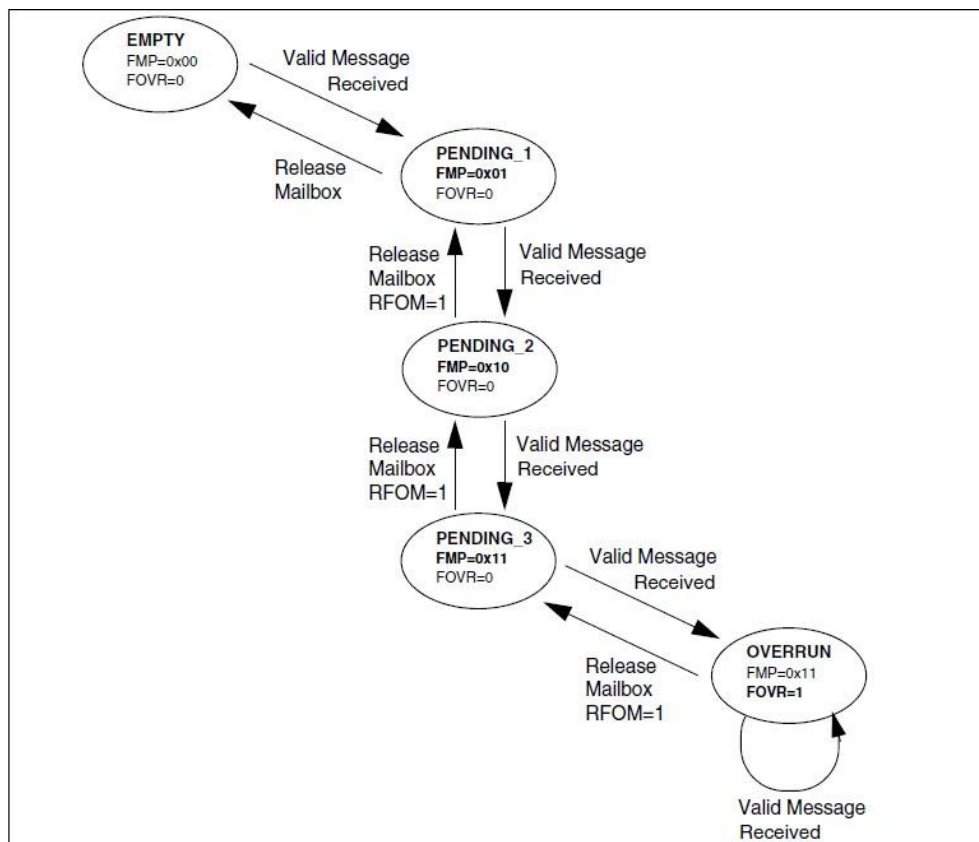


Figure 20: Receive FIFO States

As shown in figure 20

We can receive up to three messages in the FIFO mailbox then if we start from the empty state, the first valid message received is stored in the FIFO which becomes pending_1 and if we receive the next valid message, it will be stored in the FIFO and enters pending_2 until we received three messages then we must release the FIFO mailbox in order to receive other messages [2].

4.2.6. CAN Filters

We have 14 configurable and scalable filter banks (13-0) to the application in order to receive only the messages the software needs and each filter bank consists of two 32-bit registers.

- **Mask Mode:**

Filter masks are used to know which bits in the message identifier (ID) should be compared with the filter bits, as following:

The mask bit may be set to one or zero, if it is set to one the corresponding identifier bit will be compared with the filter bit, if they are matched the messages will be accepted otherwise it will be rejected. While, if a mask bit is set to zero, the corresponding identifier bit will automatically be accepted, regardless of the value of the filter bit.

² See APPENDIX B

Example 1:

We wish to accept only frames with ID of 00001657 (hexadecimal values) Set filter to 00001657

Set mask to 1FFFFFFF

When a message is received its ID should be compared with the filter and all bits must match, if only one bit does not match the ID 00001657, this frame will be rejected.

Example 2:

We wish to accept the frames that have IDs from 00001650 to 0000165F Set filter to 00001650

Set mask to 1FFFFFF0

When a message is received its ID should be compared with the filter and all bits except bits 0,1,2, and 3 must match; if any other bit does not match, this frame will be rejected.

- **Identifier list mode:**

We have a list of specific IDs, if the transmitted message carries ID that matches with anyone in the list, it will be accepted otherwise the frame is rejected.

4.2.7. CAN frame

Frame types:

- **Error Frame – Reports error condition**

Any node in the CAN network sender or receiver, may signal an error condition at any time during a data or remote frame transmission as shown in figure 21.

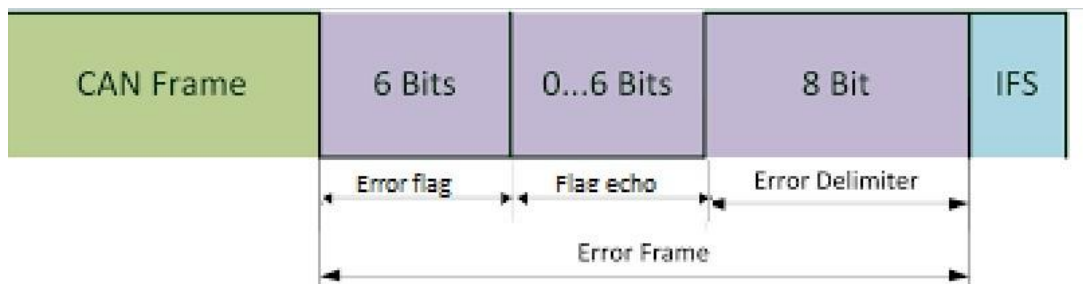


Figure 21:Error frame

- **Overload Frame – Reports node overload**

A node sends overload frame to request a delay between two remote or data frames, so the overload frame can only occur between data or remote frame transmissions as shown in figure 22.

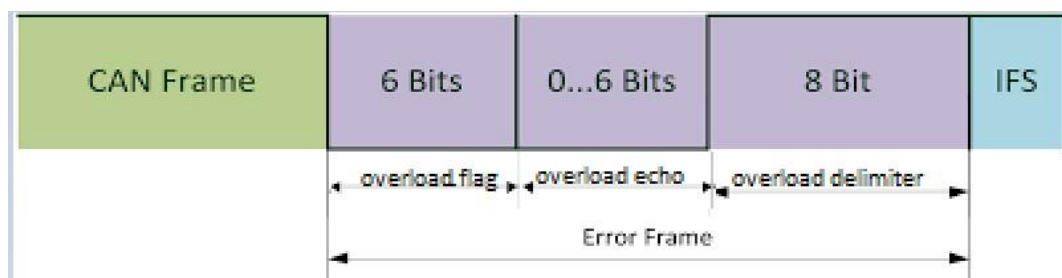


Figure 22:Error frame

- **Data Frame – Sends data**

Data transfer from node to the CAN bus and the nodes that are interested in the message can read it (This is done according to the filters of the node).

- **Remote Frame – Requests data**

Any node may request data from other nodes. The remote frame represents the request so its consequently followed by a data frame containing the requested data.

Both data and remote frame have the same frame but in remote frame data = 0

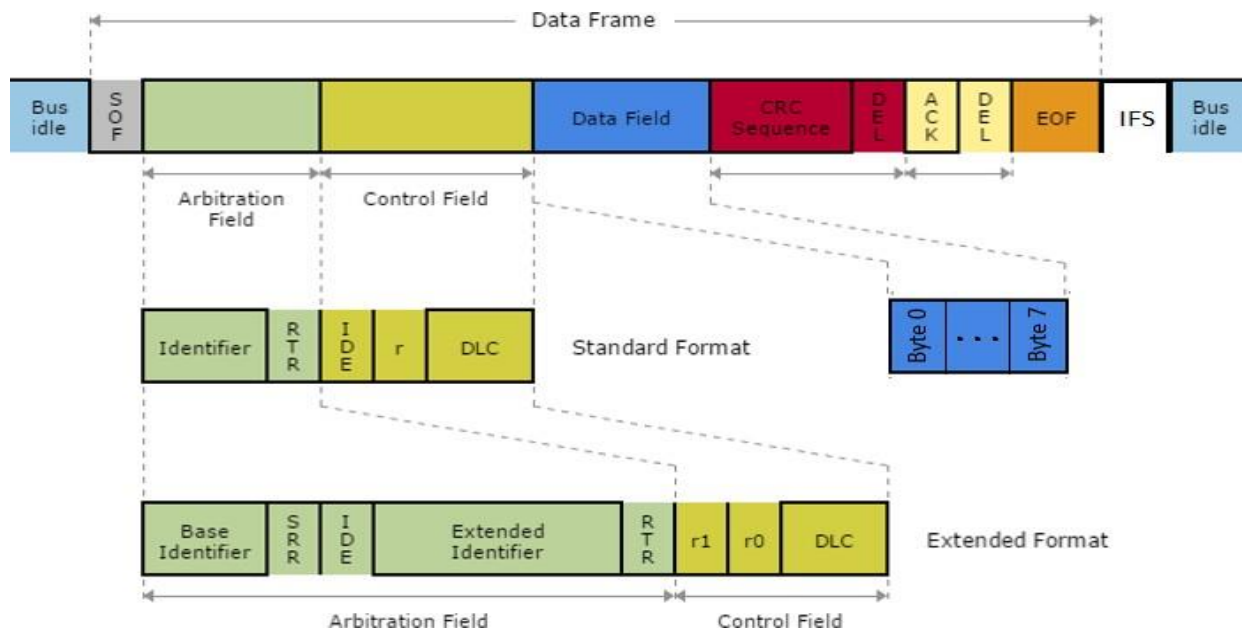


Figure 23: Data and remote frames

As shown in figure 23

- **SOF (Start of Frame)** (1 bit) – Marks the beginning of data and remote Frames
- **Arbitration Field** – Includes the message ID (11bits in standard ID and 29 bits in extended ID) and Remote Transmission Request (RTR) bit, which distinguishes data and remote frames.
- **Control Field** – DLC to determine data size (4 bits) and IDE (1 bit) to determine message ID length.

- **Data Field** – The actual data (which is zero in remote frame and contains the actual data frame)
- **CRC Field** (16 bits) – Checksum, CRC stands for Cyclic Redundancy Check, it contains Checksum and delimiter.
- **ACK Field** (2 bits) – Acknowledge and delimiter.
- **EOF** (End of Frame) (7 bits)– Marks the end of data and remote frames.
- **IFS Field** (3 bits) – Inter-frame space.

ACK delimiter: The acknowledgement is sent from the receiving node to the transmitting node and it needs some time so ACK delimiter is used.

CRC delimiter: The ECU needs some time to calculate the CRC so a delimiter bit is introduced to make some delay for the ECU.

4.2.8. CAN in our project

The specifications that are suitable with our needs are:

- Speed: 500 Kbps.
- Priority: By transmit request order.
- Using 3 mailboxes each can have 8 bytes of data

We used CAN remote frames to control our system, here is the usage of remote frames in our system :

- Ask and get response from the user whether he wants to accept the update or not.
- Give ACK that the data is transmitted from main to app ECUs.
- Get the version on the app ECU
- Wait for any update request from main to app ECU

While data frames are used to transmit hex file from main ECU to app between ECUs and also transmit update progress from main ECU to GUI.

4.2.9. Problems that faced us with CAN

1. CAN Network should be supplied with 5v.
2. The connections using wires is not stable, try to make a PCB that contains the CAN network.

4.2.10. Procedures to follow to check if your network is working or not

1. First put a list of the ID's that each node can accept in the filter registers.
2. Try with only two nodes and only one mailbox and one FIFO.
3. Work with loop back mode on the first node (this mode transmits only) and with silent mode on the second one (this mode recives only).
4. If step 2 worked try to change the first node to silent mode and the other to loopback mode.
5. If step 3 worked well , that's great , now try normal mode.
6. If step 5 worked now you should only scale the driver to use the 3 mailboxes and 2 FIFOs.

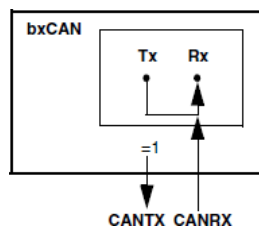


Figure 25:Node in silent mode

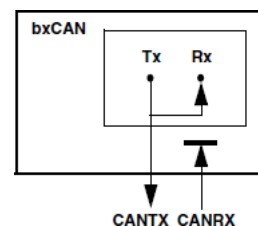


Figure 24:Node in loopback mode

This chapter was about communication protocols that we used in our project and how each of them could work. The next chapter will talk about the application ECU in our project which connect between server and all ECUs.

Chapter Five

5. APPLICATION ECU

5.1. Introduction

The Application ECU is the ECU that is running an application and going to be updated. The flash memory is typically divided into three sections: Branching Code, Bootloader Section and Application Section. The branching code can be an individual section or a part of the bootloader section, which is the case in our project. We will consider each of these sections in detail.

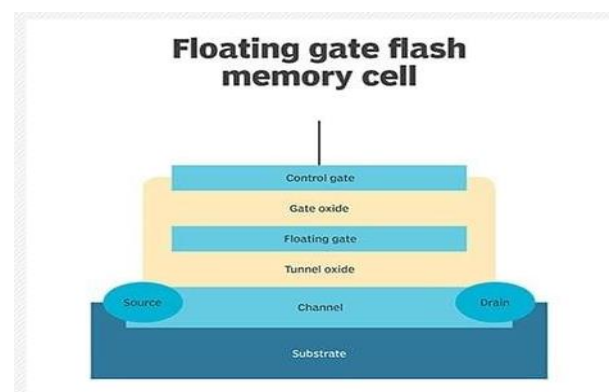
5.2. Bootloader

5.2.1. Introduction

Bootloader is a software code that runs on a microcontroller that needs to be programmed. The purpose of a bootloader is to flash or burn a new software on a microcontroller. As an example, when programming any controller, it should be connected to a PC where the PC flashes the hex file to the Flash Memory of the controller. A bootloader typically does the same job of the PC, it receives the hex file through any communication protocol as UART, SPI, CAN, etc. and stores it in the Flash memory, usually using the Flash Controller Peripheral of the microcontroller.

5.2.2. Flash Controller Overview

To understand the importance of Flash Controller, we need to understand the theory of operation of flash memory.



Flash memory, also known as flash storage, is a type of nonvolatile memory that erases data in units called blocks and rewrites data.

at the byte level. Flash memory architecture includes a memory array stacked with a large number of flash cells. A basic flash memory cell consists of a storage transistor with a control gate and a floating gate, which is insulated from the rest of the transistor by a thin dielectric material or oxide layer. The floating gate stores the electrical charge and controls the flow of the electrical current. Electrons are added to or removed from the floating gate to change the storage transistor's threshold voltage. Changing the voltage affects whether a cell is programmed as a zero or a one.

The process of moving electrons from the control gate and into the floating gate is called Fowler–Nordheim tunneling, and it fundamentally changes the characteristics of the cell by increasing the MOSFET's threshold voltage. This, in turn, changes the drain-source current that flows through the transistor for a given gate voltage, which is ultimately used to encode a binary value. The Fowler-Nordheim tunneling effect is reversible, so electrons can be added to or removed from the floating gate, processes traditionally known as writing and erasing.

Thus, the process of programming (writing or erasing) flash memory needs high voltages. The hardware flashers used to flash a chip usually have the ability to produce such high voltages, but in order to flash the microcontroller through software without the use of an external circuit, an internal circuit providing this high voltage should be present internally in the microcontroller, this circuit is controlled using the Flash Controller.

5.2.3. Flash Memory Module in STM32F103C8T6

The microcontroller used in the project is a medium-density device with a 128KB flash memory with every 1KB as a page i.e., flash memory has 128 pages of memory starting from memory address 0x08000000 to memory address 0x0801FFFF.

The Information block in the flash memory starts from memory address 0x1FFFF000. It has 2KB for System Memory, which contains the standard bootloader from STM32, followed by 2 bytes for Option Bytes. Option Bytes are 2 non-volatile data bytes in the flash memory reserved for user use. They aren't erased after reset which can be handy in storing information without the need of an external EEPROM.

Flash controller registers are present in the memory starting from address 0x40022000 to address 0x40022023

The memory organization is shown in Table 1.

Block	Name	Base Addresses	Size (bytes)
Main Memory	Page 0	0x0800 0000 - 0x0800 03FF	1 KBytes
	Page 1	0x0800 0400 - 0x0800 07FF	1 KBytes
	Page 2	0x0800 0800 - 0x0800 0BFF	1 KBytes
	Page 3	0x0800 0C00 - 0x0800 0FFF	1 KBytes
	Page 4	0x0800 1000 - 0x0800 13FF	1 KBytes
	.		
	Page 127	0x0801 FC00 - 0x0801 FFFF	1 KBytes
Information Block	System Memory	0x1FFF F000 - 0x1FFF F7FF	2 KBytes
	Option Bytes	0x1FFF F800 - 0x1FFF F80F	16
Flash Memory Interface Registers	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FLASH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4

Table 1:Flash module organization

	Reserved	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRP	0x4002 2020 - 0x4002 2023	4

The standard STM32F103C8T6 Bootloader supports several communication protocols, however it doesn't support CAN Bootloader, so a custom bootloader in the program memory is developed to flash the program in the memory.

5.2.4. Intel Hex Format

Now that we understand the use of the Flash Memory Controller, the next step is to study the hexfile format to be able to flash the program in the flash memory.

Intel hexadecimal object file format, Intel hex format or Intel Hex is a file format that conveys binary information in ASCII text form. It is commonly used for programming microcontrollers, EPROMs and other types of programmable logic devices. In a typical application, a compiler or assembler converts a program's source code (such as in C or assembly language) to machine code and outputs it into a HEX file. Common file extensions used for the resulting files are .HEX or .H86. The HEX file is then read by a programmer to write the machine code into a PROM or is transferred to the target system for loading and execution.

Intel HEX consists of lines of ASCII text that are separated by line feed or carriage return characters or both. Each text line contains hexadecimal characters that encode multiple binary numbers. The binary numbers may represent data, memory addresses, or other values, depending on their position in the line and the type and length of the line. Each text line is called a record.

The Record structure is 6 parts as follows:

- 1- Start code, one character, an ASCII colon ':'.
- 2- Byte count, two hex digits (one hex digit pair), indicating the number of bytes (hex digit pairs) in the data field. The maximum byte count is 255 (0xFF). 16 (0x10) and 32 (0x20) are commonly used byte counts. 16 is the most used in the hex files used in our project, however other byte numbers may also be found.
- 3- Address, four hex digits, representing the 16-bit beginning memory address offset of the data. The physical address of the data is computed by adding this offset to a previously established base address, thus allowing memory addressing beyond the 64 Kbyte limit of 16-bit addresses. The base address, which defaults to zero, can be changed by various types of records. Base addresses and address offsets are always expressed as big endian values.
- 4- Record type, two hex digits, 00 to 05, defining the meaning of the data field. The record types are explained in details after Record Structure.
- 5- Data, a sequence of n bytes of data, represented by 2n hex digits. Some records omit this field (n equals zero). The meaning and interpretation of data bytes depends on the application.
- 6- Checksum, two hex digits, a computed value that can be used to verify the record has no errors.

As mentioned, Record type may have a value from **00** to **05** as follows:

- 1- **00**: Data. Contains data and a 16-bit starting address for the data. The byte count specifies number of data bytes in the record. Example:

:1028000000500020212A0008272A00082B2A00084F

This line contains hex data: 0x02000500, 0x08002A21, 0x08002A27, 0x08002A2B respectively. Their memory locations are determined according to the Extended Linear Address record before them.

- 2- **01:** End of File. It must occur exactly once per file in the last line of the file. The data field is empty (thus byte count is 00) and the address field is typically 0000. Example:
:00000001FF
- 3- **02:** Extended Segment Address. The data field contains a 16-bit segment base address (thus byte count is always 02) compatible with 80x86 real mode addressing. The address field (typically 0000) is ignored. The segment address from the most recent 02 record is multiplied by 16 and added to each subsequent data record address to form the physical starting address for the data. This allows addressing up to one megabyte of address space. This type of record is not used in the hex files used in the project.
- 4- **03:** Start Segment Address. For 80x86 processors, specifies the initial content of the CS (code segment): IP (instruction pointer) registers (i.e., the starting execution address). The address field is 0000, the byte count is always 04, the first two data bytes are the CS value, the latter two are the IP value. Since the startup code manages the memory sections and code segments and the bootloader manages the execution start address of the application, this record is also not present in the hex files used in the project.
- 5- **04:** Extended Linear Address. Allows for 32 bits addressing (up to 4GiB). The record's address field is ignored (typically 0000) and its byte count is always 02. The two data bytes (big endian) specify the upper 16 bits of the 32 bit absolute address for all subsequent type 00 records; these upper address bits apply until the next 04 record.

The absolute address for a type 00 record is formed by combining the upper 16 address bits of the most recent 04 record with the low 16 address bits of the 00 record. If a type 00 record is not preceded by any type 04 records then its upper 16 address bits default to 0000. This record is usually the first record in the hex file; however, it can be repeated anywhere in the hex file if the base address of the data is changed.

Example:

:0200000040800F2

In this record, the address is 0x0800, which is concatenated to the address in the data record to get the final address. For example, in the previous data record example:

:1028000000500020212A0008272A00082B2A00084F

The data will be stored starting from address 0x08002800.

- 6- **05:** Start Linear Address. The address field is 0000 (not used) and the byte count is always

04. The four data bytes represent a 32-bit address value (big-endian). It represents the start address of the program. Example:

:0400000050800290DB9

The start address of this program is 0x0800290D.

5.2.5. Our Solution

5.2.5.1. Flash Memory Sections.

The main flash memory starting from address 0x0800000 to 0x0801FFFF should typically contain three sections: Branching Section, Bootloader Section and Application Section. The branching section is used to jump to either the application or the bootloader and it can be a part of the bootloader or has an individual section in the Flash Memory. In this project, the branching code is a part of the bootloader section, thus, the main flash memory has two sections: Bootloader section and Application Section.

Instead of having only one application section, the application section in the project contains two banks; the reason for this is that as the flash memory must be erased before writing any data to it, if only one application bank is present then if any error or interruption of communication occurs during the flashing of the file, the ECU would stop its current functionality completely. For this reason, the application section is divided into two banks with one active bank which becomes a back-up in case of error, and

another bank for programming the new software

Therefore, the 128KB flash memory is divided as follows: 10KB at the start of the main memory is reserved for the bootloader starting from address 0x08000000 to address 0x080027FF, followed by 59KB for the first Application Bank starting from address 0x08002800 to address 0x08013FFF then 59KB for the second Application Bank starting from address 0x08014000 to address 0x0801FFFF.

5.2.5.2. Flash Option Bytes

As mentioned before, the option bytes are 2 non-volatile data bytes in the flash memory for user use. Since Option Bytes are a part of the flash memory, they cannot be programmed or erased directly, but only through the flash controller module. The erase sequence is as follows:

1. Check that no Flash memory operation is ongoing by reading the BSY bit in the FLASH_SR register.
2. Unlock the OPTWRE bit in the FLASH_CR register.
3. Set the OPTER bit in the FLASH_CR register.
4. Set the STRT bit in the FLASH_CR register.
5. Wait for BSY to reset.
6. Read the erased option bytes and verify.

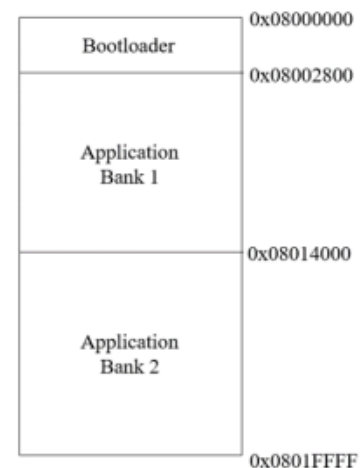


Figure 27: Flash memory sections

And the programming procedure is as follows:

- 1- Check that no Flash memory operation is ongoing by checking the BSY bit in the FLASH_SR register.
- 2- Unlock the OPTWRE bit in the FLASH_CR register.
- 3- Set the OPTPG bit in the FLASH_CR register.
- 4- Write the data (half-word) to the desired address.

- 5- Wait for the BSY bit to be reset.
- 6- Read the programmed value and verify.

The Option Bytes should also be erased before any write operation. The flash controller programs the option bytes through a register and the new programmed value is only programmed after the next system reset.

The Option Bytes Data0 Byte is used in our project to determine which application bank is currently on use, and thus determine which one is the one used for flashing the new software. Data0 can have one of three values:

- 1- **0xFF**: This means that the option bytes are not programmed and that there is no application in the application banks.
- 2- **0x00**: This means that Application Bank 0 is in use.
- 3- **0x01**: This means that Application Bank 1 is in use.

5.2.5.3. Branching Code

As mentioned previously, the branching code is the software that decides whether to jump to the application or the bootloader. In the project it is considered a part of the bootloader section in the memory.

The branching code in the project is mainly based on soft reset. As in the application, a soft reset is performed when an update request is received through CAN, so the branching decision is taken based on the soft reset. A soft reset is detected by checking the twenty eighth bit in the RCC (Reset and Clock Control) Control/Status Register (RCC_CSR).

The branching code sequence is as shown in Figure 28:

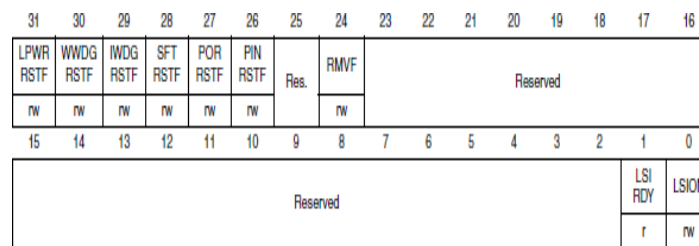


Figure 28:RCC_CSR Register bits

- 1- Get the option bytes and determine which bank is currently in use. If they aren't programmed, the new software is programmed in the first bank.
- 2- Get the soft reset flag from the RCC_CSR Register.
- 3- If a soft reset occurred, continue to the bootloader.
- 4- If no soft reset occurred, check if there is a program in the currently used bank by reading the first 4 bytes of its start address.
- 5- If a program is present, get the program reset vector address, whose address is stored in the second 4 bytes of the flash memory according to the linker script, and jump to that address by a pointer to function. If no program is present, the code continues to the bootloader.

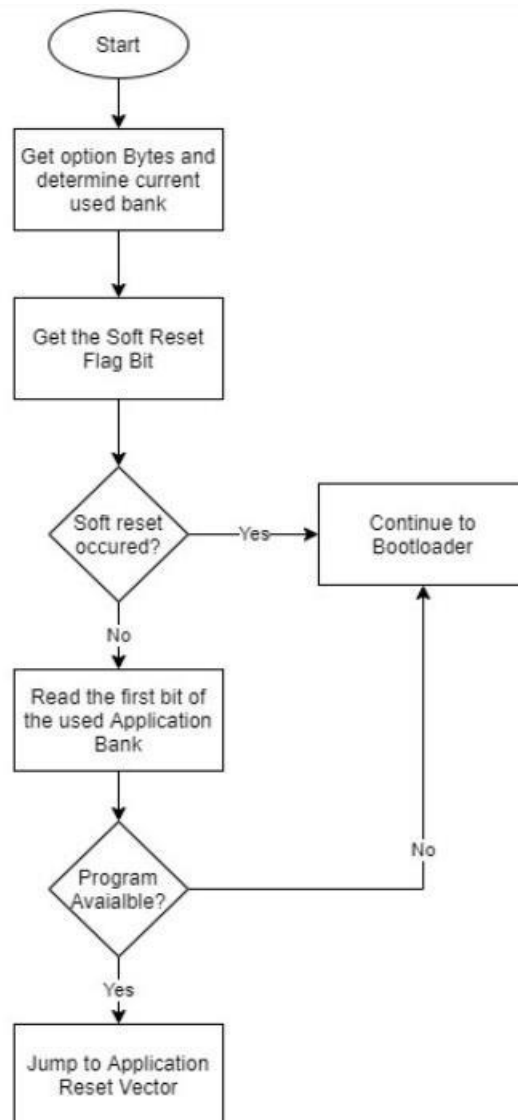


Figure 29: Branching Sequence flowchart

5.2.5.4. Bootloader Code

The bootloader sequence is always performed after the branching code, so the currently used bank and the programmed bank are known to the software ^[3]. The code has one main module: HexDataProcessor which processes the hex record, extracts the data and flashes it in the flash memory, in addition to a super loop which handles the received CAN Messages. We will first consider the super loop sequence then discuss the HexDataProcessor Module.

³ See APPENDIX A

5.2.5.4.1. Bootloader Main Sequence

- 1- Erase the 59 pages of the programming bank as no new data can be programmed unless the banks are erased.
- 2- Wait for a new CAN message.
- 3- If a can message is received. It can be one of the four messages in the CAN filter:
 - **Hex Field:** This message contains a part of the hex record. Since the CAN messages can carry a maximum of 8 bytes per message, the hex record is received as fragments of 8 bytes then re-combined at the bootloader end in a hex line array. The main ECU is ensured to not send a part of the new line in the CAN Message but terminate the CAN message with the remained number of bytes of the frame.
 - **Software Version Request:** This message is a remote frame asking for the current SW Version on the ECU. Since this message is originally meant to be received by the application code, receiving it in the bootloader sequence means that there is probably no application in the application banks, so a message with version number 0.0.0 is sent with the same CAN Message ID as a data frame, so that when compared to any version from the server, an update is requested.
 - **Used Bank Request:** This message is a remote request requesting for the currently used bank for the application. The used bank is sent with the same CAN Message ID as a data frame.
 - **Request DTCs:** This message is a remote frame asking for the DTCs array. The DTCs array is sent in a CAN Message with the same ID as a data frame.

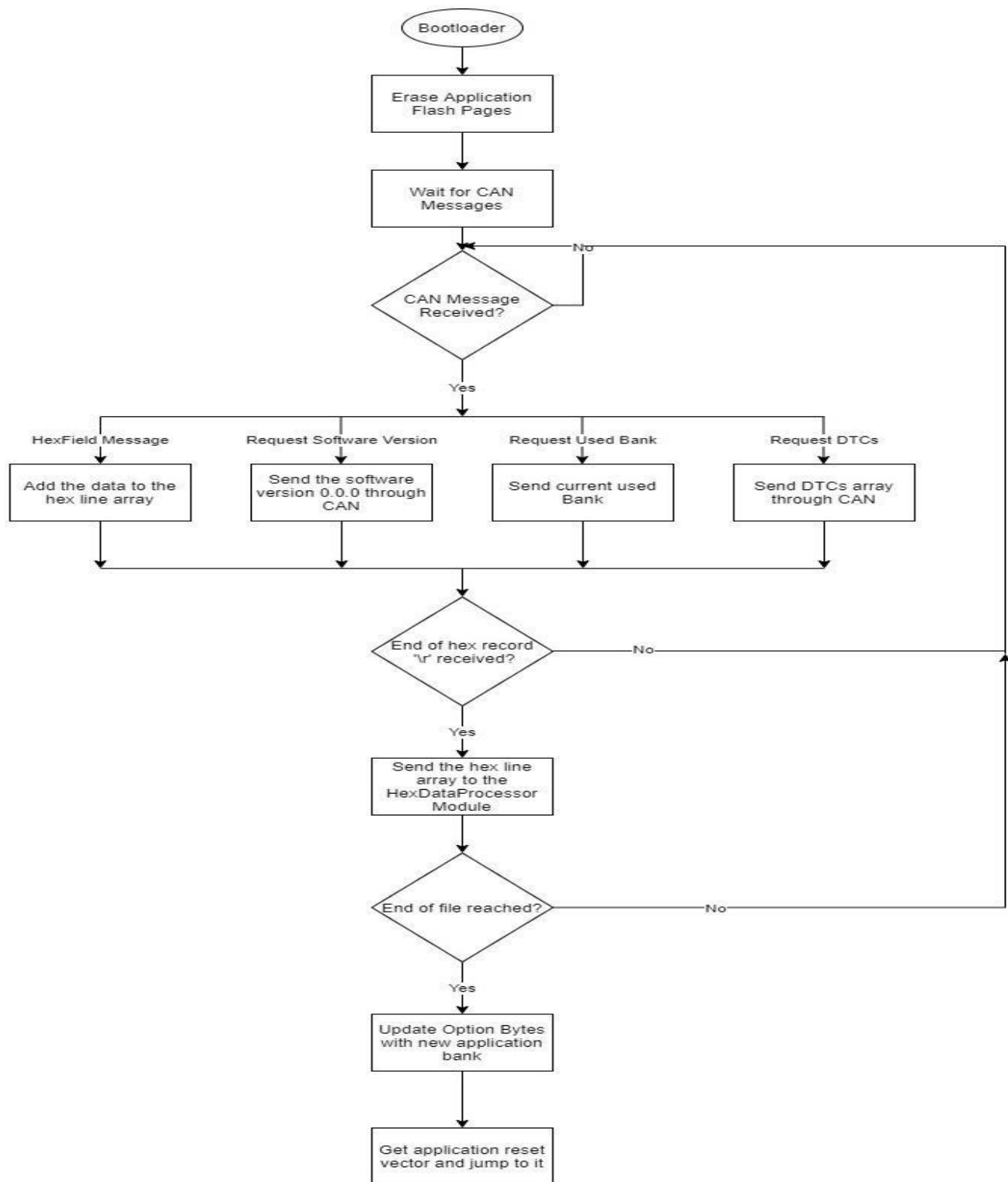


Figure 30: Bootloader code flowchart

- 4- Whether the received CAN Message was a Hex Field message or not, check if the last bit added in the Hex line array is the termination bit '\r', if so, continue to the next step, or else wait for a new CAN Message.
- 5- Send the Hex Array line to the HexDataProcessor to process the data and flash it in the flash memory and receive the feedback from the module.
- 6- Check the feedback, if the end of the file is received, continue to the next step, or else wait for a new CAN Message.
- 7- The whole file is now received, the reset vector address is retrieved using the same procedure of the branching code and the bootloader jumps to it through a pointer to function.

5.2.5.4.2. HexData Processor Module:

The Hex Data Processor Module is responsible for taking the hex record and extracting the data and addresses and flashing the data in the flash memory. The module is divided into 4 states/functions in order to decrease the code complexity:

1- Check Validation:

In this step, the software checks if the hex record is a valid hex record by checking the startcode ':' and checking that the number of the characters in the record is an even number and not exceeding the maximum record number.

2- Convert to Hex:

Since the hex file is received in ASCII format, this step converts the ASCII format into hexadecimal bytes format.

3- Parse Line:

This step parses the hex data into corresponding fields of the record structure into a structure and determine the record type. It also stores the data in a 32-bit array in the structure since the flash memory

The step of rearranging the 61 data is only done in case of a data record as the other records don't need this step. It also adds 0xFF if the data length doesn't complete a 32-bit variable, so that the next record can program the rest of the data. This step also checks the CRC Checksum and returns a NOK in case of error.

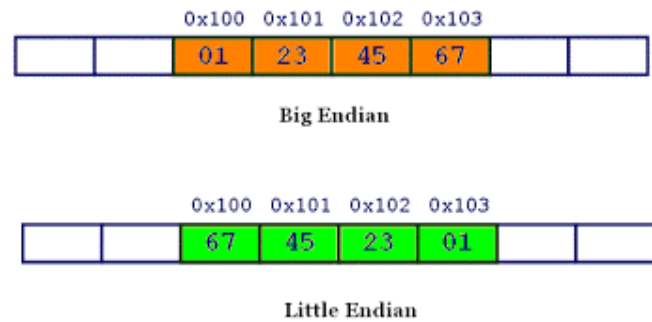


Figure 31: Big endian and little endian

4- Write Data to flash:

In this step, an action is done according to the record type: in case odd data, 32-bit chunks of data are sent one by one to the flash controller to program them according to the last extended linear address and the address in the data record. In case of extended linear address, the address is stored in a static variable to be used when storing the data.

5.3. Application

As a proof of concept, a simple application is developed using some sensors. The application consists of the following: a parking assistant, a radiator cooler which should work in case of high temperature and a CAN Message handler. The application is developed in form of a super loop. Each of these modules will be discussed in details.

5.3.1. Parking Assistant

The parking assistant consists of two main components: an IR sensor as shown in figure 32 and a piezoelectric buzzer.

IR Working Principle

- Infrared spectrum.
- Transmitter.
- Receiver.

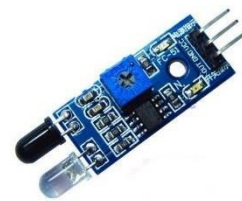


Figure 32: IR sensor

- Reflection by object.
- Voltage level.

The IR sensor is a digital sensor which sends an IR wave in front of it and senses the medium, if the wave was reflected this means a barrier is in front it, so it returns a high signal on its out pin.

The application procedure is as follows: If a high signal is received, a buzzer is turned on to alert the driver that there is a barrier during parking, otherwise the buzzer is off. As a proof of concept, the first application has this feature is turned off, after the update, the parking assistant works fine.

5.3.1. Radiator Cooler

As shown in figure 33 The cooler consists of two components: an LM35 temperature sensor and a motor with a 2N2222A transistor as a driver for it.



Some Features about LM35

- Can measure temperature -55°C to 150°C.
- Minimum and Maximum Input voltage 35V and -2V,
Typically 5V.
- $\pm 0.5^\circ\text{C}$ accuracy.
- Drain current is less than 60 μA .
- Low cost temperature sensor.
- Small and hence suitable for remote applications.

Figure 33: LM35 Temperature sensor

The temperature sensor ratings are 1mV/1°C. The controller reads the temperature sensor voltage feedback through a 12-bit ADC Peripheral and converts it into temperature through the following equation:

$$Temperature = (ADC\ Reading / 2^{12}) * 3.3 * 100$$

If the temperature is found to be above a certain limit, a DC Motor should be turned to act as a fan to the radiator. If the temperature exceeds another higher limit, a DTC of value 0x50 which corresponds to High Temperature DTC is added to the DTCs array to be sent to the server in case of diagnostics request. As a proof of concept, the threshold in the application to turn on the DC Motor is higher than the DTC Threshold and thus a DTC is recorded and the DC Motor isn't turned. After the software update, the temperature threshold is adjusted and the fan works fine.

5.3.2. CAN Messages Handling

The Application typically receives 4 CAN Messages according to its CAN Filters:

- 1- Update Software Request:** This message is sent in case a new update is found on the server and the user accepts the update. When this message is received, the RCC Reset flags are reset and a software reset is performed in order to jump to the bootloader code.
- 2- Software Version Request:** This message is a remote frame asking for the current SW Version on the ECU. A can message with the same CAN Message ID is sent as a data frame containing the current software version.
- 3- Used Bank Request:** This message is a remote request requesting for the currently used bank for the application. The used bank is read from the option bytes and sent with the same CAN Message ID as a data frame.
- 4- Request DTCs:** This message is a remote frame asking for the DTCs array. The DTCsarray is sent in a CAN Message with the same ID as a data frame.

5.4. Some Concepts about Network

A domain name:

is your website's equivalent of a physical address. It helps users find your site easily instead of using its internet protocol (IP) address. Domain names consisting of a name and an extension are a key part of the internet infrastructure.

Static IP:

A static IP address, or fixed IP address, is an IP address that never changes. Not everyone needs a static IP address.

Dynamic IP:

is an IP address that changes from time to time unlike a static IP address. Most home networks are likely to have a dynamic IP address and the reason for this is because it is cost effective for Internet Service Providers (ISP's) to allocate dynamic IP addresses to their customers.

To reach to a website it must have static Ip but in case of it's dynamic IP we use NOIP which connect the IP with the provided domain name.

Chapter Six

6. CRAPHICAL USER INTERFACE (GUI)

6.1. What is GUI?

A graphical user interface (GUI) is a type of user interface through which users interact with electronic devices via visual indicator representations.

An Interface shows objects conveying knowledge and represents actions that the user may take. When the user communicates with them, the objects change color, size, or visibility.

GUI objects include keys, icons, and cursors. Occasionally, these visual elements are enhanced with sounds, or visual effects such as transparency and falling shadows.

This breakthrough gave people a different way to communicate with systems, so they didn't need to know any coding concepts.

The big advantage of a GUI is that systems that use one are accessible to people of all knowledge levels, from a beginner to an advanced developer or other tech-savvy individual.

We make it easy for anyone to open menus, transfer files or launch programs.

Instant feedback is also provided by GUIs. For example, if you click an icon it will open, and that can be seen in real time. Using a command line GUI, once you hit return, you won't know if it's a legitimate entry; if it isn't correct, nothing will happen.

6.2. History of GUI:

In 1974, work began at PARC on Gypsy, the first bitmap What-You-See-Is-What-You-Get (WYSIWYG) cut & paste editor.

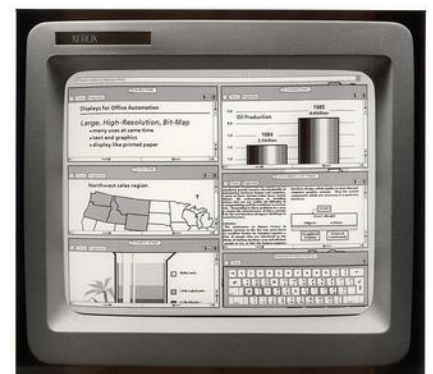
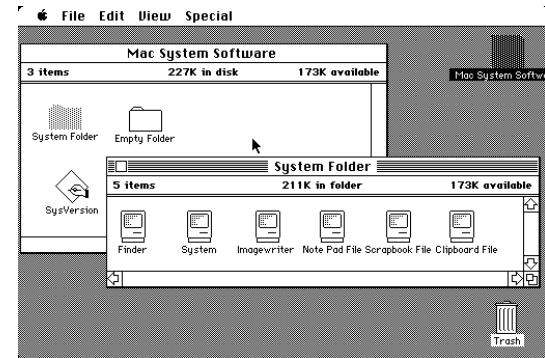


Figure 34: First commercial GUI

In 1975, Xerox engineers demonstrated a Graphical User Interface "including icons and the first use of pop-up menus".

In 1981, Xerox introduced a pioneering product, Star, a workstation incorporating many of PARC's innovations.

In 1983, as shown in figure 34 The first commercial use of a GUI occurred, Apple engineers developed Lisa, the first GUI-based computer available to the public but it hasn't commercial successful.



In 1984, the Apple Macintosh with a GUI became the most common commercial device.

In 1985, Microsoft followed suit with Windows 1.0.

In 1997, Windows 2.0 was a marked change when it was released.

6.3. GUI Layers:

From the user side he deals with the graphical interface which appears in front of him on a hardware display where icons, gestures and tools which he can choose from all appears on the display server as windows manager in (windows interfaces) which is linked to the operating system (kernel) such Linux then the hardware which is the whole container which the user use it for controlling as shown in figure 35.

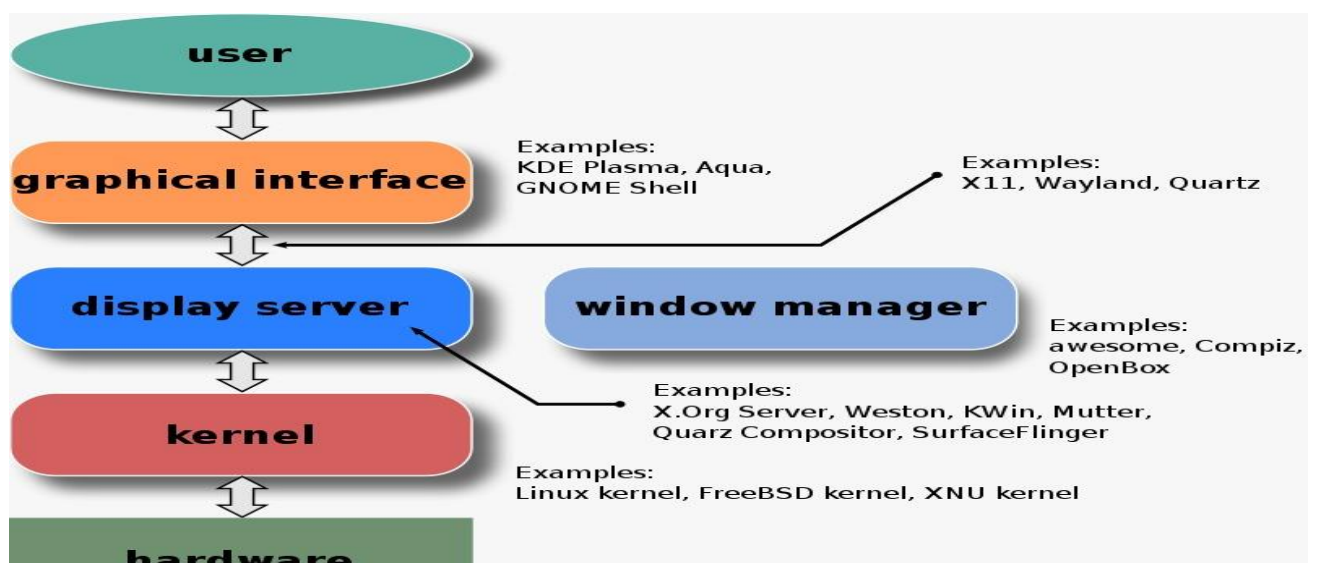


Figure 35: GUI Layers

6.4. Automotive GUI



Figure 36:Automotive GUI

Automotive OEMs and Tier 1 suppliers are faced with the challenge to deliver instrument clusters, head-up displays, radios, IVI systems and full integrated cockpits that offer functional safety, an intuitive user experience, top performance and a brand-defining look and feel.

Automotive human machine interface (HMI) allows drivers and passengers to interact with their vehicles in a far more natural way. Via in-car touch screens and buttons, push rotary controllers, swipe and gesture functions, and even speech recognition technology. These features provide safer distraction-free driving whilst also making the in-car user experience far more enjoyable.

On the other hand, to reduce driver distraction, many car makers are starting to implement multi-information displays in cars. Human machine interfaces, enabled by smart systems and embedded sensors, ensure vehicles react to driver's intent and preferences.

Different programming languages are used in developing desktop application and graphical user interfaces for multi-tasking applications, c ++ and Python are mainly used in those kinds, due to the easy interaction with those languages when dealing with heavy duty applications which needs you as developer to be easily able to deal with different kind of complexities.

GUI (Graphic User Interfaces) in cars. In the latest car models the on-screen user experience could be divided into these categories:

- Entertainment and communication: Music, television, email, internet, third-party applications UI-related. Such features are often accessed through the monitor in the center console...
- The tachometer dashboard UI, alarm, driving assist and other driving-related functions.
- Anything else, such as climate control, general settings and other vehicle settings.



Figure 37:GUI Application

So, with this kind of differences companies have challenged each other around the efficiency of a GUI and the comfort of the user who is handling it. Apple and Google are both allies and competitors of car companies in this race, and automakers are being increasingly forced to avoid providing their services because customers already have a long history with the ecosystems of those firms. Car companies have recognized that they can deliver excellent digital user experiences by investing in digital design and maximize the added benefit of deep integration with key driving experience elements.

6.5. GUI in our project

6.5.1. Flow of the GUI in our system

GUI in our project represent to the user on a TFT touch screen all the alerts and the updates needed for the car at which the OEMs send it to the user to be confirmed by the user.

TFT touch screen:

It is a combination device that includes a TFT LCD display and a touch technology overlay on the screen. The device can both display content and act as an interface device for whoever is using it.

It is directly connected to the raspberry pi.

The main scenario implemented:

FOTA update where the server sends an update to the user to appear on the touch screen waiting for the confirmation from the user by yes or no. The communication channel between the ECUs in the car gone through CAN where a SPI to can module is used to convert the signals from the ECUs to the control unit of the Touch screen which is the Raspberry pi model B (which is using SPI to communicate).

6.5.2. Why is python used?

The implementation of the GUI is done using Python 3.10 on PyCharm 2021 edition IDE. Python used in the design due to its simplicity where you can handle multi modules on the same time using 3 main modules which make it more comfortable to the developer to use it in GUI applications. Most of the python modules used in GUI are built using c/c ++ but using it in python make it easier to handle.

6.5.3. Tkinter vs PyQt5:

6.5.3.1. Advantage of using PyQt5:

1. Coding flexibility: GUI programming with Qt is designed around the concept of signals and slots for establishing communication amongst objects.
2. More than a framework: Qt uses a wide array of native platform APIs for the purpose of networking, database creation, and many more.
3. Various UI components: Qt offers several widgets, such as buttons or menus, all designed with a basic appearance across all supported platforms.
4. Various learning resources: because PyQt is one of the most used UI frameworks for Python, you can get easy access to a wide array of documentation.
5. Easy to master: PyQt comes with a user-friendly, straight forward API functionality, along with specific classes linked to Qt C++.

6.5.3.2. Disadvantages of using PyQt5

1. Lack of Python-specific documentation for classes in PyQt5.
2. It requires a lot of time for understanding all the details of PyQt, meaning it is a quite steep learning curve.

6.5.3.3. Advantages of using Tkinter

1. Available out-of-charge for commercial usage.
2. It is featured in the underlying Python library.
3. Creating executables for Tkinter apps is more accessible since Tkinter is included in Python.
4. Simple to understand and master, as Tkinter is a limited library with a simple API, being the primary choice for creating fast GUIs for Python scripts.

6.5.3.4. Disadvantages of using Tkinter

1. Tkinter does not include advanced widgets.
2. It has no similar tool as Qt Designer for Tkinter.
3. It doesn't have a native look and feel.

the PyQt5 side used in the project

the 3 main modules which we used in the implementation:

1. QtCore Module: it contains the core non-GUI classes, including the event loop and Qt's signal and slot mechanism. It also includes platform independent abstractions for Unicode, threads, mapped files, shared memory, regular expressions, and user and application settings.

- The QEventLoop class provides a means of entering and leaving an event loop.
- Signals and slots are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks. In GUI programming, when we change one widget, we often want another widget to be notified. More generally, we want objects of any kind to be able to communicate with one another. For example, if a user clicks a

Close button, we probably want the window's close() function to be called. Other toolkits achieve this kind of communication using callbacks. A callback is a pointer to a function, so if you want a processing function to notify you about some event you pass a pointer to another function (the callback) to the processing function. The processing function then calls the call back when appropriate. While successful frameworks using this method do exist, call backs can be unintuitive and may suffer from problems in ensuring the type-correctness of callback arguments.

Signals are emitted by objects when they change their state in a way that may be interesting to other objects. This is all the object does to communicate. It does not know or care whether anything is receiving the signals it emits. This is true information encapsulation and ensures that the object can be used as a software component.

As shown in figure 38.

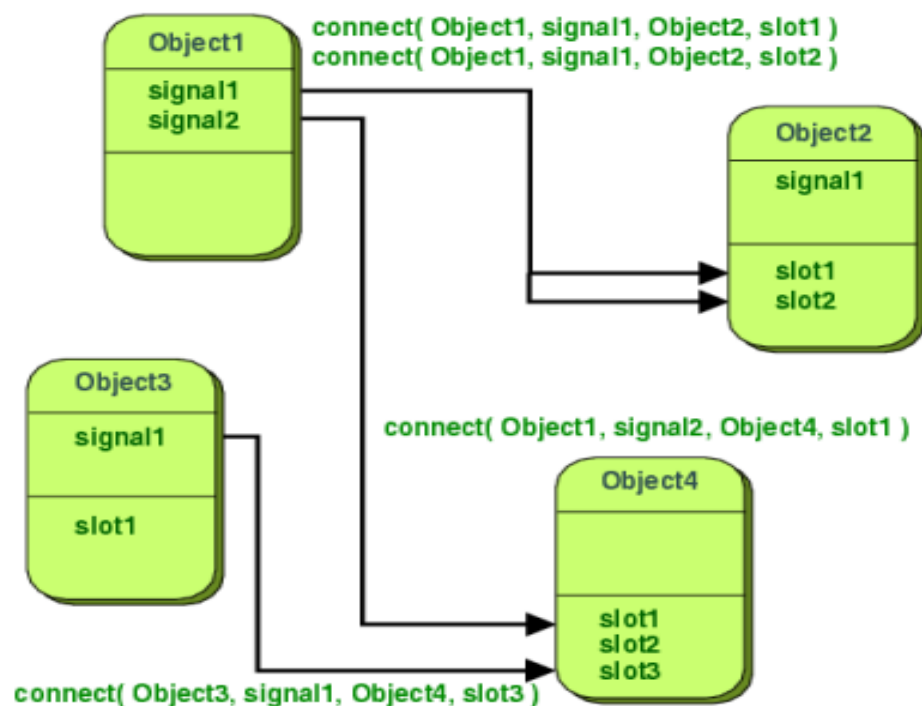


Figure 38: Signals and slots

Slots can be used for receiving signals, but they are also normal member functions. Just as an object does not know if anything receives its signals, a slot does not know if it has any signals connected to it. This ensures that truly independent components can be

created with Qt. You can connect as many signals as you want to a single slot, and a signal can be connected to as many slots as you need. It is even possible to connect a signal directly to another signal. (This will emit the second signal immediately whenever the first is emitted).

Together, signals and slots make up a powerful component programming mechanism.

2. The QtGui module: it contains the majority of the GUI classes. These include a number of table, tree and list classes based on the model–view–controller design pattern. Also provided is a sophisticated 2D canvas widget capable of storing thousands of items including ordinary widgets.

3. QtWidgets Module: it provides a set of UI elements to create classic desktop-style user interfaces, as for example:

- Widgets (QWidget) which are the primary elements for creating user interfaces in Qt.
- Layouts (QLayout) are an elegant and flexible way to automatically arrange child widgets within their container.



Figure 39: QWidget interface

4. Layouts (QLayout):

are an elegant and flexible way to automatically arrange child widgets within their container.

6.5.4. Controller used: Raspberry pi 3 model B



Figure 40: Raspberry pi 3 model B

The Raspberry Pi hardware (as shown in figure 40) has evolved through several versions that feature variations in the type of the central processing unit, amount of memory capacity, networking support, and peripheral device support, which help in developing different application programming interfaces (APIs), using python and Linux OS with its raspbian system makes it easy to install different libraries and work through different structures.

VNC (Virtual Network Computing) is used in order to remotely control the desktop interface of one computer (running VNC Server) from another computer or mobile device (running VNC Viewer), VNC Viewer transmits the keyboard and either mouse or touch events to VNC Server (as shown in figure 41), and receives updates to the screen in return. We can see the raspbian system on our windows screen on the laptop in order to virtually

control it

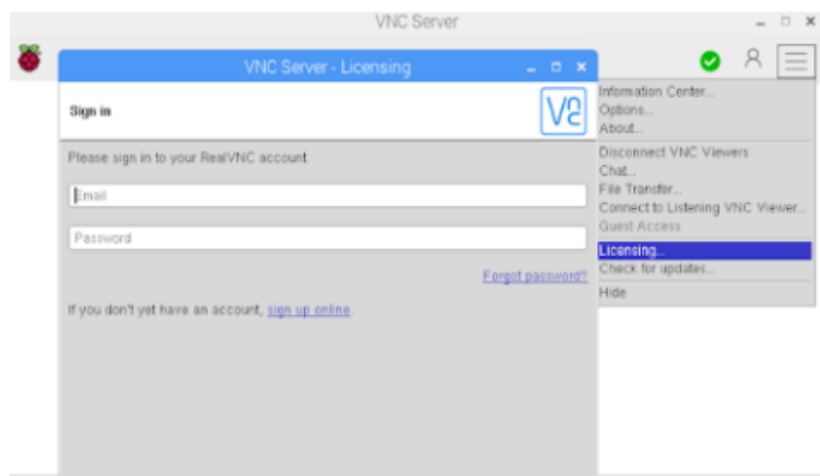


Figure 41: VNC server

6.6. SPI TO CAN Module

As shown in figure 42

In our project we built our GUI system by using touch screen connected with Raspberry pi so that it tells the user if there is any update, and the user has the right to accept it now or later and it shows the progress of the updating process if the user accepts the update request.

Because we want all ECUs in our project to be connected to the same CAN network, we use SPI TO CAN module because Raspberry Pi doesn't have a built-in CAN Bus but its GPIO includes SPI Bus, that is supported by large number of CAN controllers. So, we will use a bridge between Raspberry Pi and CAN Bus which is SPI Bus.

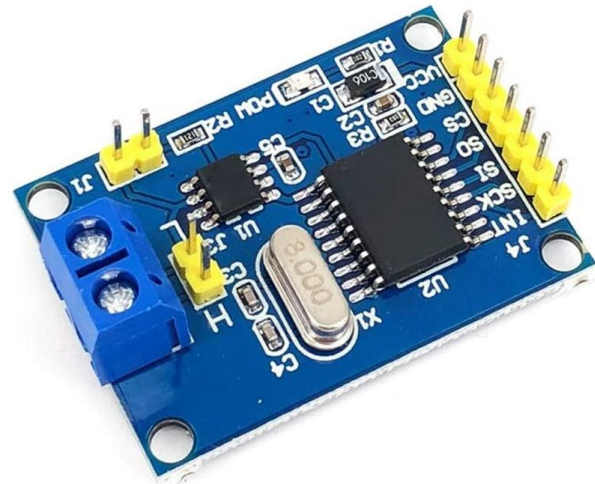


Figure 42:SPI to CAN module

Modern cars may have up to 50 to 100 sensors for control and monitoring the automobile status. With more and more features being added to cars continue to increase. any miscommunication or loss of data could lead to fatal accidents. Thus, standard simple communication protocols like UART, I2C, SPI, are not suitable because they are not as reliable as CAN communication.

Our SPI to CAN module is (mcp2515) and it consists of CAN controller because CAN Bus is multi-master protocol, each node needs a controller to manage its data. And CAN transceiver (MCP2551) because CAN controller needs a send/receive chip to adapt signals to CAN Bus levels. Controller and Transceiver are connected by two wires TxD and RxD. So, we connect Raspberry pi to SPI bus in the SPI TO CAN Module which has connections as follow:

- MOSI (Master Out Slave In).
- MISO (Master In Slave Out).
- SCLK (Serial Clock).

Adding to that, two other pinouts:

- CS or SS (Chip Select, Slave Select) to enable and disable the chip.
- INT (Interruption) to interrupt the chip when needed

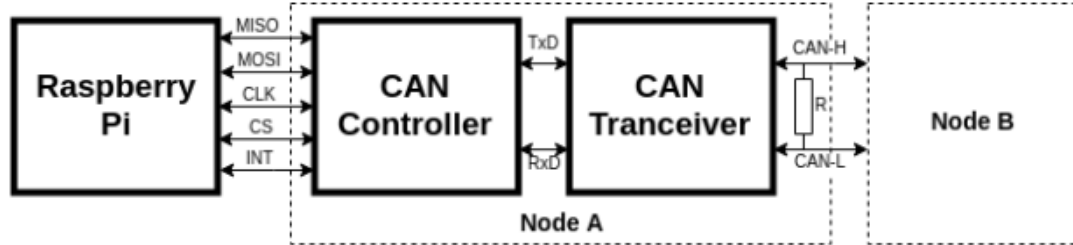


Figure 43: The connection of Raspberry Pi to SPI to CAN module

6.7. Design Aspects

Our GUI system is based on the flat design which appears in the professional user interfaces on the market. Flat design is a design style of the user interface using plain, two-dimensional elements and bright colors. It is often contrasted with the skeuomorphic style by copying real-life properties which gives the impression of three dimensions. Upon the introduction of Windows 8, Apple's iOS 7 and Google's Material Design, all of which make use of flat design, its prominence became prominent.

Flat design was originally designed for responsive design, in which the content of a website seamlessly varies depending on the screen size of the user. Use simple shapes and minimal textures, flat design ensures responsive designs work well and load quickly (especially important because mobile devices have slower internet speeds).

Flat design provides users with a simpler and more efficient user experience by reducing the amount of visual noise (in the form of textures and shadows).

So, we took advantage of that in our system using CSS (Cascading Style Sheets) language, it is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting.

6.7.1. Our GUI system consists of

1. Main.py:

to control the appearance of the GUI, widgets shapes and the organization of the design.

2. Img.qrc:

This file contains all the media used in the GUI which is transformed to be read in python.

6.8. GUI Features

6.8.1. Welcome Pag

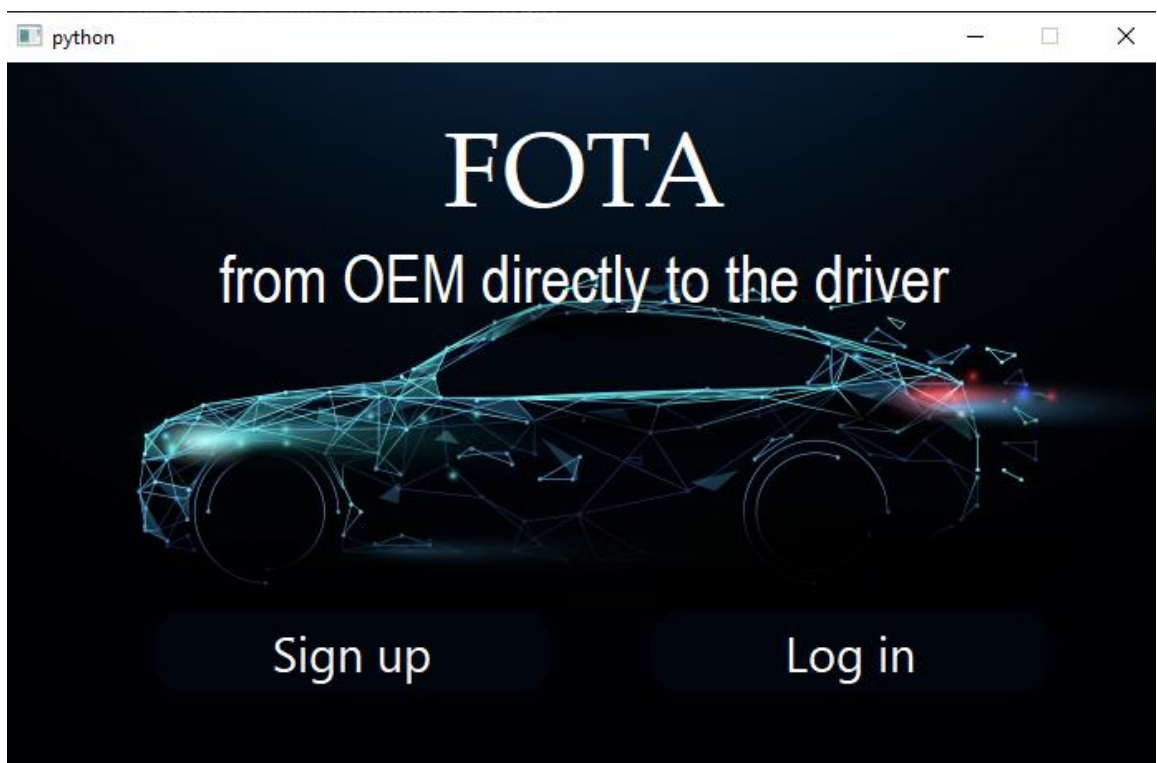


Figure 44: Welcome page

Here as we can see in figure 44, the user or driver can choose between sign up button or login button.

6.8.2. Login page

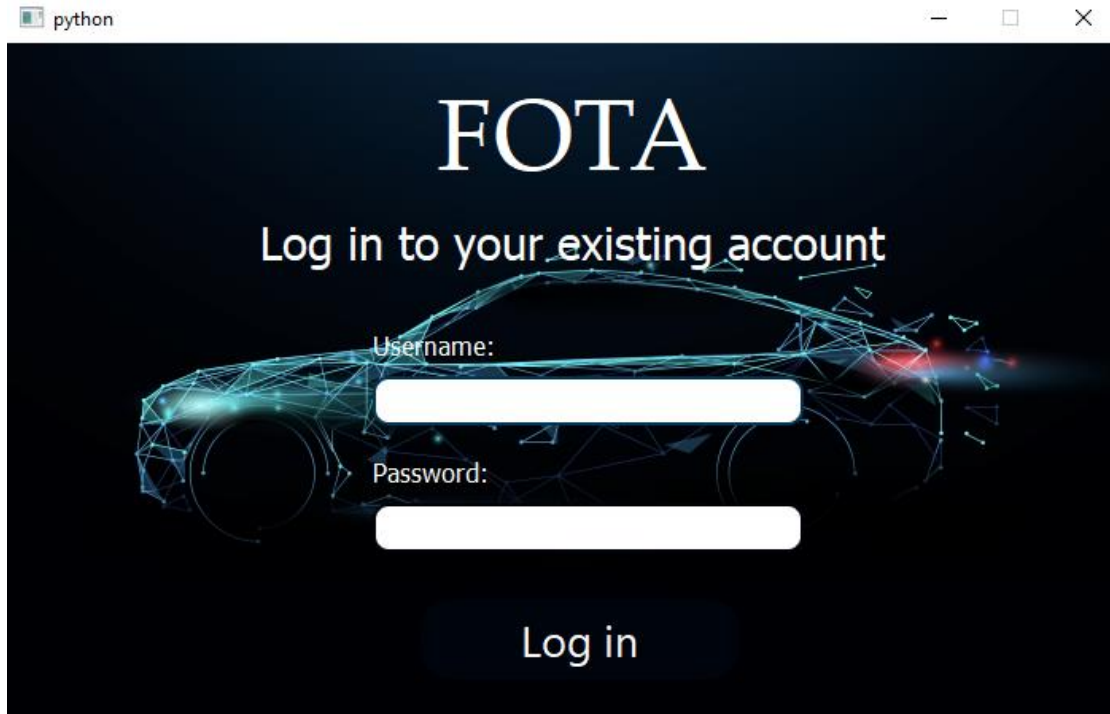


Figure 45:login page

Here as we can see in figure 45, this page where the user can login with his username and password fields with warning message to enter all fields and if the username, the password or both are not valid, there is a database consists of information of users their usernames and passwords that must the user enter to check the new updates.

6.8.3. Signup page

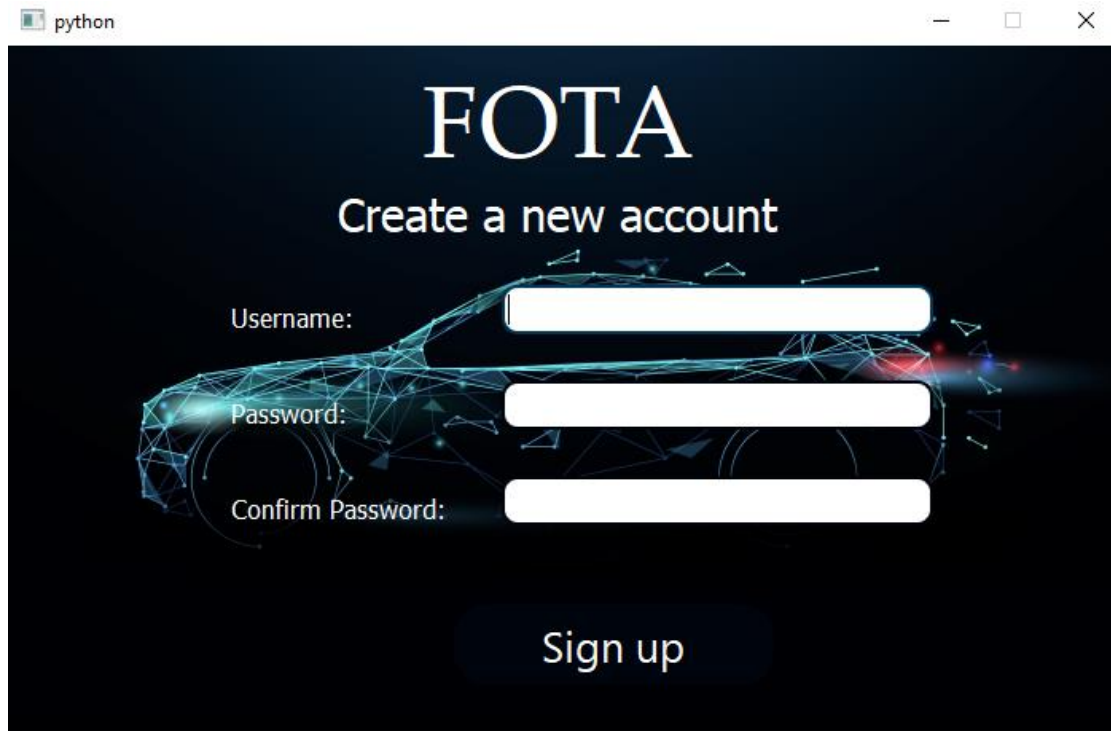
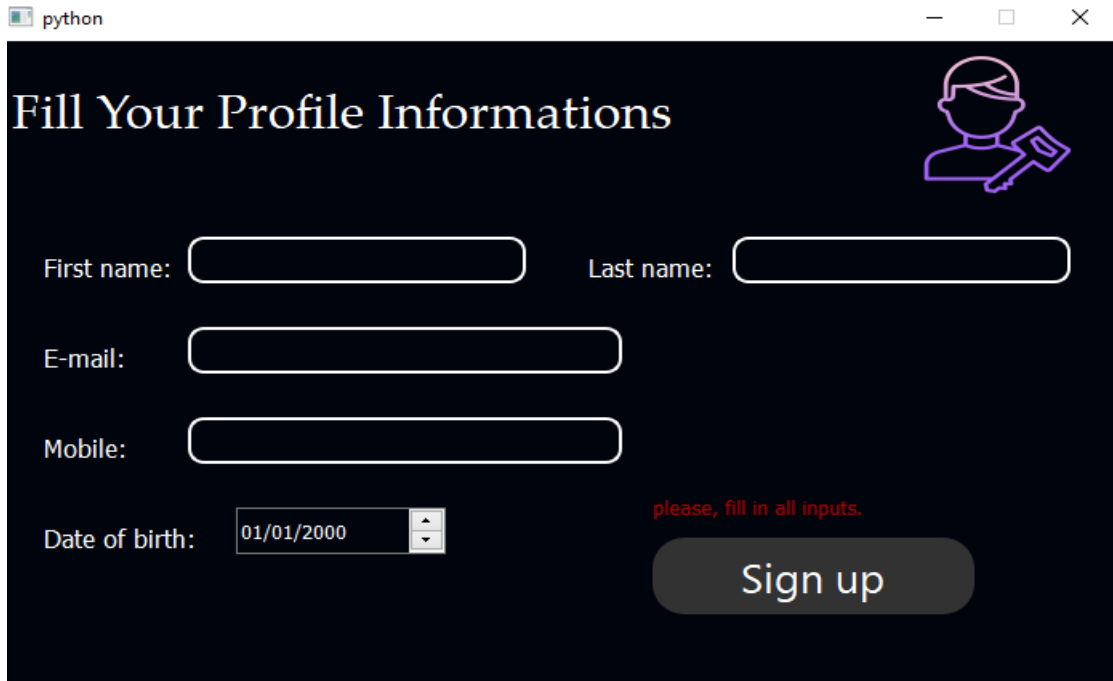
A screenshot of a web browser window titled 'python'. The page has a dark blue background with a glowing, wireframe car model in the center. At the top, the text 'FOTA' is displayed in a large, white, serif font, followed by 'Create a new account' in a smaller, white, sans-serif font. Below this, there are three white input fields stacked vertically. To the left of the first field is the label 'Username:', to the left of the second is 'Password:', and to the left of the third is 'Confirm Password:'. At the bottom center, there is a dark blue button with the text 'Sign up' in white. The browser window includes standard OS controls (minimize, maximize, close) in the top right corner.

Figure 46:Signup page

Here as we can see in figure 46, sign up page where 3 fields are existing to serve a new user with a warning message to enter all fields and make sure that password field and confirm password field are the same to add the user information to the database.

6.8.4. Fill profile page



The screenshot shows a web browser window titled 'python'. The page has a dark blue background and is titled 'Fill Your Profile Informations' in a large, white, serif font. In the top right corner, there is a white line-art icon of a person's head and shoulders with a keyhole on the chest. Below the title, there are four input fields: 'First name:' and 'Last name:' are side-by-side, followed by 'E-mail:' and 'Mobile:' stacked vertically. The 'Date of birth:' field is a date picker showing '01/01/2000'. To the right of the date picker, there is a red warning message: 'please, fill in all inputs.'. At the bottom right, there is a dark blue rounded rectangular button with the text 'Sign up' in white.

Figure 47: Fill profile page

Here as we can see in figure 47, this page to make the user enter some information with warning message to input all fields.

6.8.5. FOTA Page

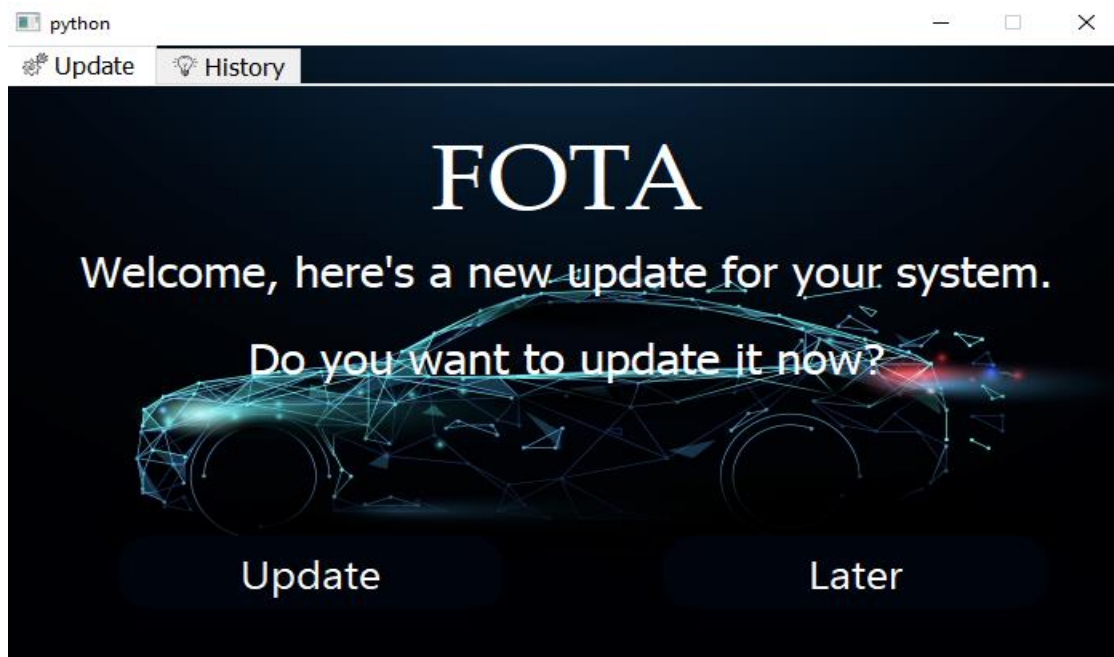


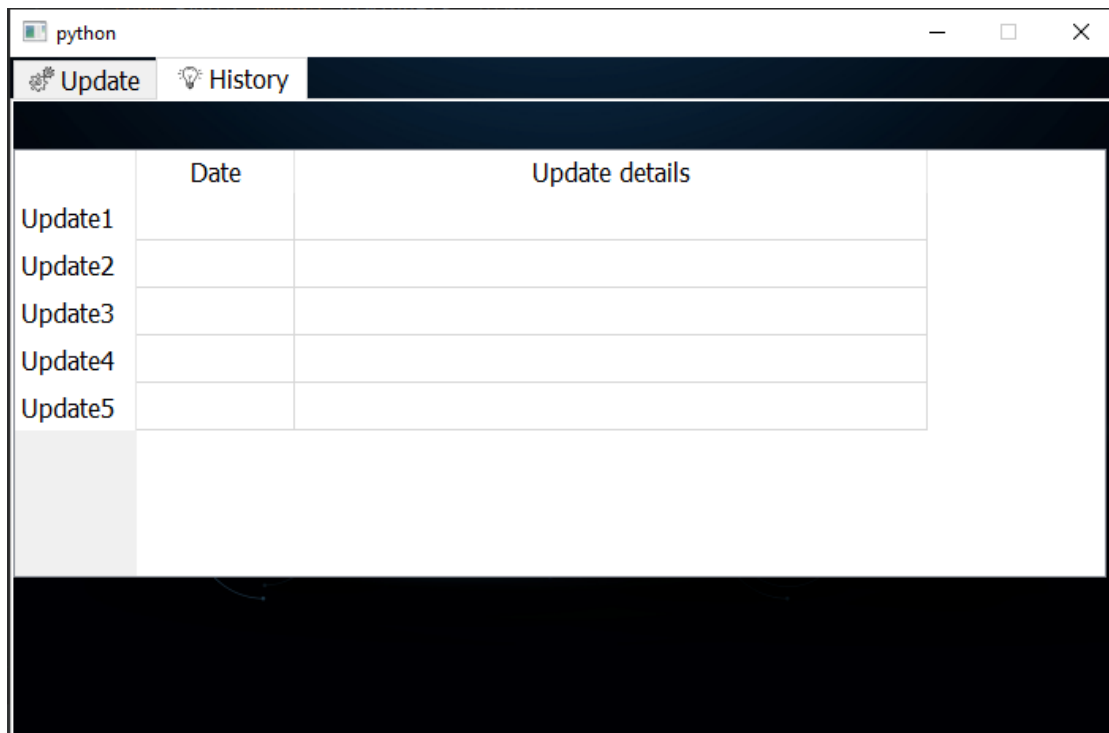
Figure 48: FOTA page

Here as we can see in figure 48, the main page that the user can choose from update now or later and the user can know if the update was updated or there's a new update to be updated.

When pressing Update button the GUI sends an HTTP request to the web server to download the file, then it parser it and sends it via the CAN BUS to the application ECU/s [4].

⁴ See APPENDIX A

6.8.6. History tab



	Date	Update details
Update1		
Update2		
Update3		
Update4		
Update5		

Figure 49:History tab

Here as we can see in figure 49, this tab will contain the date of the update and the details of information about, to see these details. (Its GUI handling is done but it needs a further work with database which is gathering the info we will talk about this point as it is considered a future work.

6.8.7. update tab



Figure 50: Update tab

Here as we can see in figure 50, this page to show the user when the update will be completed.

When pressing Update button, the GUI sends an HTTP request to the web server to download the file, then it parser it and sends it via the CAN BUS to the application ECU/s.

6.9. GUI Advantages

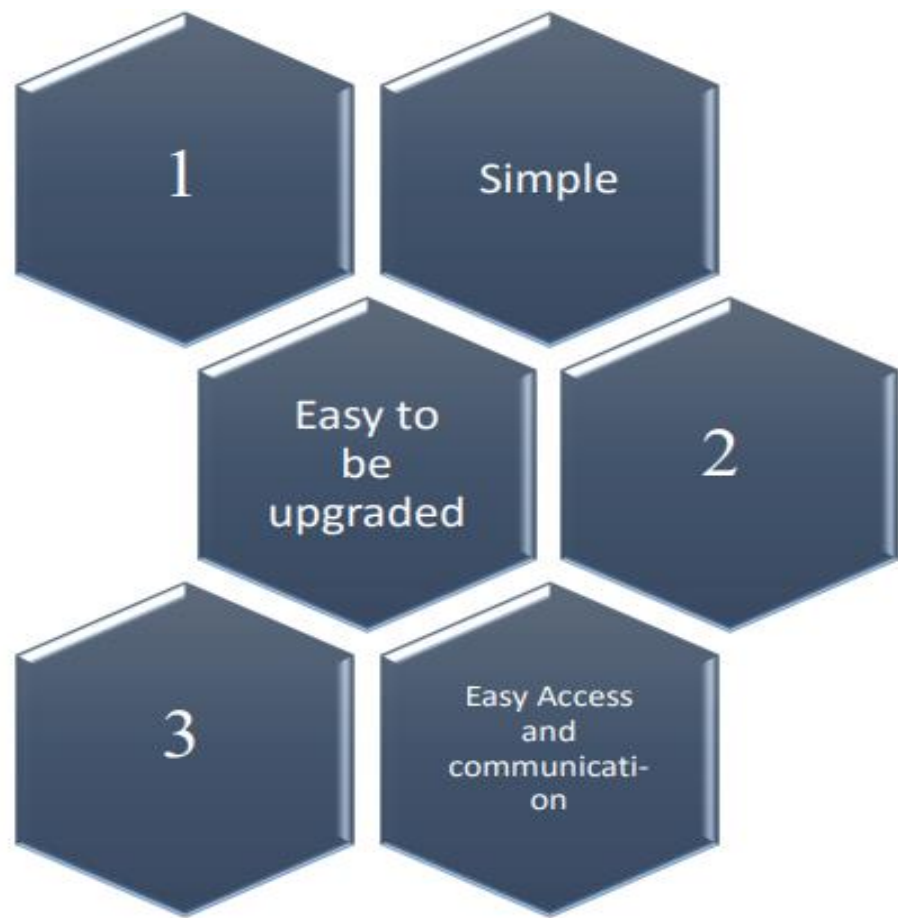


Figure 51:GUI Advantages

6.10. GUI Future work

For the Users login, sign up and updates history in the FOTA tab these features are handled by SQLite database to receive from the database the user information and check it upon the saved data also, we must do the same thing for the (updates history) tab the server must save all the updates info in a database which the GUI can access it and gather the info to view it. So, the GUI future work is to make database to the history tab and view the details of every update with its date to make the user familiar with all previous updates.

Chapter Seven

7. SECURITY

7.1. Cryptography

Cryptography is the science of encrypting and decrypting messages and text, or the study of hidden writing and it is a method of protecting communications and information through the use of codes, so that we can prevent public from reading private messages, only those for whom the information is intended can read and process it. The cryptography word consists of two prefixes, crypto stands for hidden and graphy stands for writing.

7.1.1. Cryptography techniques

Cryptography is related to the disciplines of cryptanalysis and cryptology and. It includes as shown in the figure :

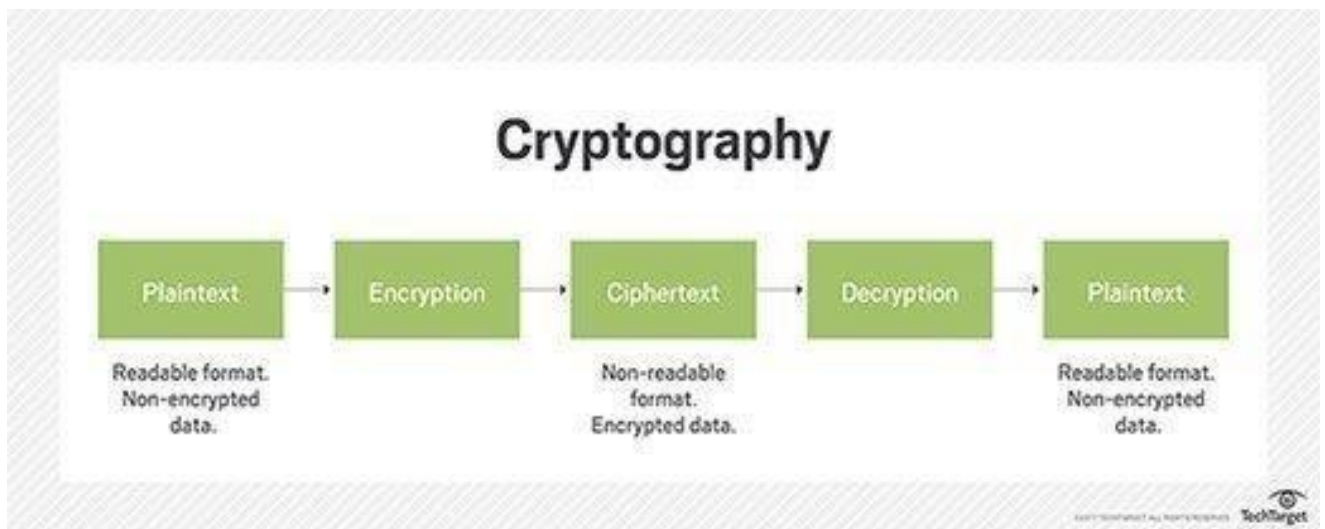


Figure 52: Cryptography system

techniques to hide information in storage or transit such as merging words with images, microdots and other ways. But, Today in computer-centric world, cryptography is about scrambling **plaintext** (Nonencrypted data) into **ciphertext** (Encrypted data) and this process is called **encryption**, then taking the ciphertext and backs it to plaintext and this process is called **decryption**. Cryptographers are the people who practice this field.

Modern cryptography should achieve the following four objectives:

- 1- **Confidentiality:** the information cannot be understood by anyone for whom it was unintended.
- 2- **Integrity:** the information cannot be altered in storage or transit between sender and intended receiver without the alteration being detected.
- 3- **Non-repudiation:** the creator/sender of the information cannot deny at a later stage his other intentions in the creation or transmission of the information.
- 4- **Authentication:** the sender and receiver can confirm each other's identity and the origin/destination of the information.

7.1.2. Cryptographic algorithm

Cryptosystems use ciphers or cryptographic algorithms, to encrypt and decrypt messages so that it can secure communications among devices such as smartphones, computer systems and applications. A cipher suite uses one algorithm for encryption, another algorithm for key exchange and another algorithm for message authentication. This process involves digital signing and verification for message authentication, and key exchange, public and private key generation for data encryption/decryption.

7.1.3. Types of cryptography

- **Single-key or symmetric-key encryption** (AES, DES): uses single key for encryption and decryption.
- **Public-key or asymmetric-key encryption** (RSA): uses key for encryption and another key for decryption.
- **Hash functions** (SHA-1, SHA-2, SHA-3): uses mathematical transformation to encrypt data.

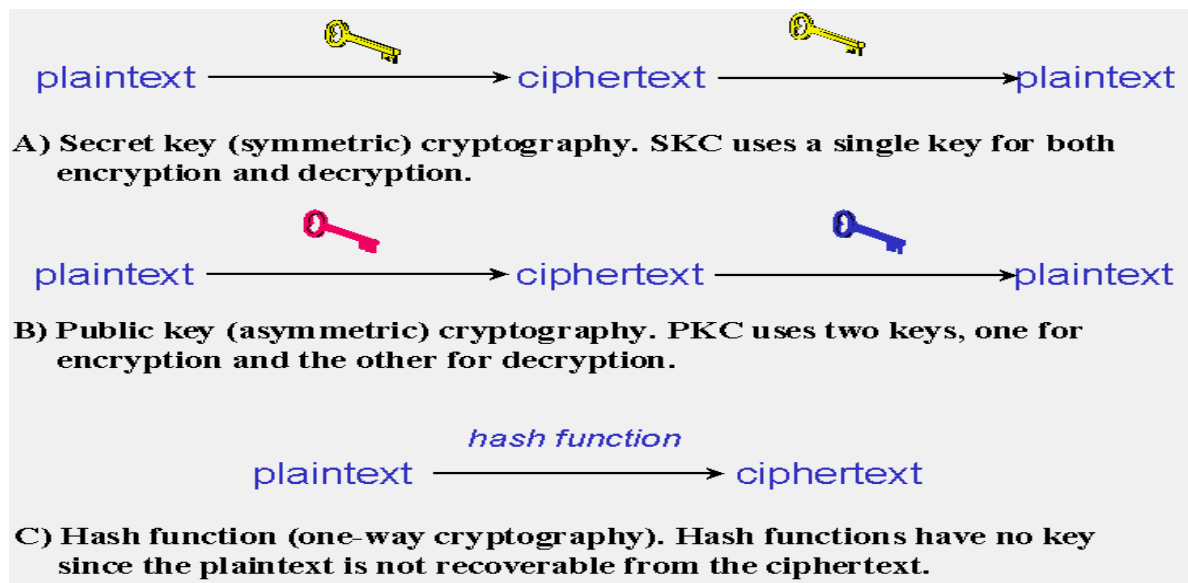


Figure 53: Three type of cryptography

7.2. HMAC

7.2.1. HMAC history

The definition and analysis of the HMAC construction was first published in 1996 in a paper by Mihir Bellare, Hugo Krawczyk, and Ran Canetti. They also wrote RFC 2104 in 1997. The 1996 paper defined a nested variant called NMAC. FIPS PUB 198 standardizes and generalizes the use of all HMACs. HMAC is used within the SSH, IPsec, JSON Web Tokens, and TLS protocols.

7.2.2. What is HMAC?

HMAC stands for Hash Message Authentication Code, it is a technique to verify the integrity of a message transmitted between two parties which agree on a shared secret key.

Any cryptographic hash function, such as SHA-3 or SHA-2, may be used to calculate HMAC function; the resulting MAC algorithm is termed HMAC-X, where X is the hash function used (e.g. HMAC-SHA256 or HMAC-SHA3-256). The cryptographic strength of the HMAC depends

upon the cryptographic strength of the underlying hash function, the size and quality of the key, and the size of its hash output.

7.2.3. HMAC Algorithm

HMAC combines the original message and the secret key to compute a message digest function. The transmitter of the message calculates the HMAC of the message and the key then transmits the HMAC with the original message. The receiver recalculates the HMAC using the message and the secret key, then compares the received HMAC with the calculated HMAC to see if they match or not. If the two HMACs are identical, then the receiver knows that the original message has not been changed because the message digest hasn't changed, and that it is authentic because the transmitter knew the shared key, which is presumed to be secret.

Any change to the data or the hash value results in a mismatch, because knowing the secret key is required to change the message and reproduce the correct hash value. Therefore, if the original and computed hash values match, the message is will be authenticated.

7.2.4. How HMAC function works?

HMAC uses two passes of hash computation as shown in figure 54. First the secret key is used to derive two keys – inner and outer. The first pass of the algorithm produces an internal hash which is the output of the HASH function when it takes the inner key and the original message as its two inputs. The second pass produces the final HMAC code derived from the same HASH function when it takes the inner hash result and the outer key as inputs. Thus, this algorithm provides better immunity against attackers.

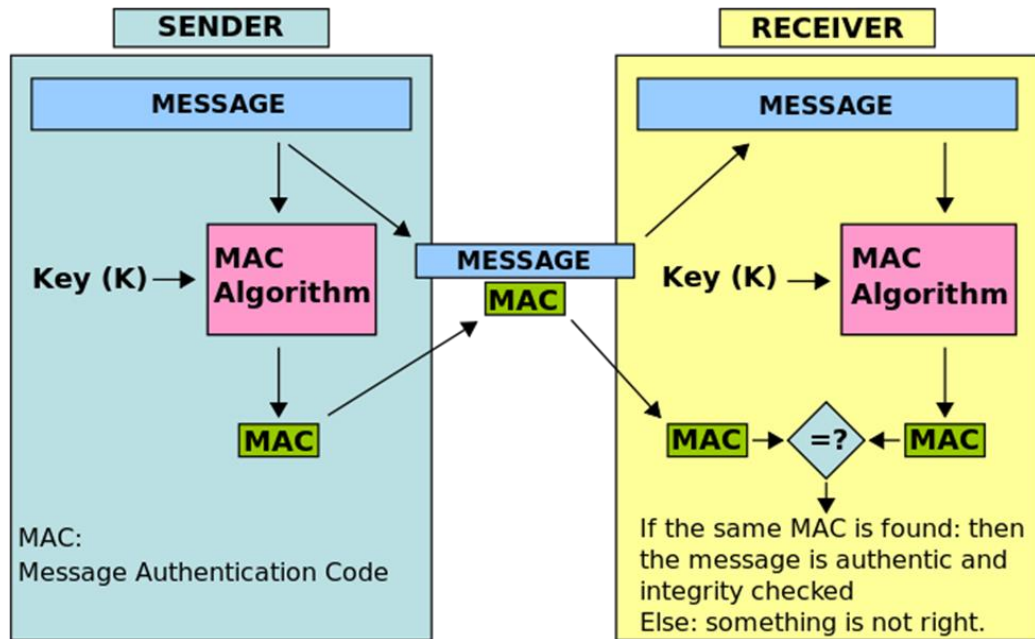


Figure 54:HMAC process

7.2.5. Properties of HMAC SHA256:

- The key can be any length. However, its recommended size is 64 bytes (512 bits).
- The output hash is of length 256 bits.
- SHA-256 operates on 512-bit blocks.
- HMACs in general are fast because they use hash functions rather than public key mathematics.

7.2.6. HMAC in our FOTA:

The client and server agree on a common hash function, which is SHA256 that has been chosen because its characteristics are suitable with our needs.

Before the server sends out the file, it first obtains a hash of that file using the HMAC-SHA256 hash function. It then sends this hash along with the file itself. Upon receiving the two items (i.e the file and the hash), the client obtains the HMAC-SHA256 hash of the downloaded file and then compares it with the downloaded hash. If the two hashes are matched, then that would mean the file was not tampered along the way. Otherwise it means that the file have been attacked and the data is changed.

Chapter Eight

8. FUTURE WORK

8.1. Live Diagnostics

Since a connectivity is already established in the system, a live diagnostics feature is implemented to inform the OEM in case of failure or error in the system. The flow is as shown in the figure.

- 1- The user asks to send the diagnostics through the GUI.
- 2- The GUI sends a CAN Message to the main and application ECUs requesting for a diagnostics session.
- 3- The application ECU sends its DTCs through the CAN Bus.
- 4- The Main ECU checks if it is currently with the GSM performing an update, if not, it sends the DTCs to the server.
- 5- The server receives the DTCs and displays them on the server terminal.

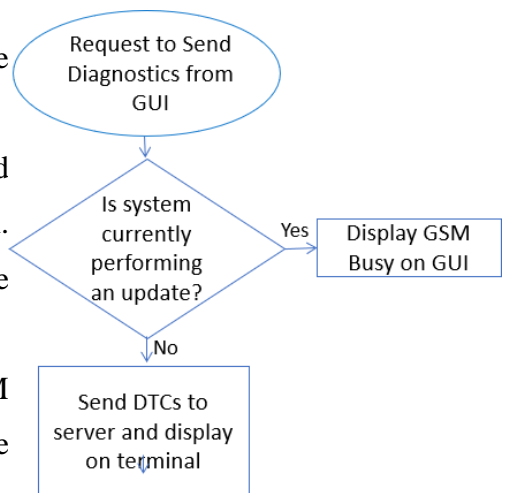


Figure 55: Diagnostics flow

8.2. Seamless FOTA

The main idea of the seamless FOTA is, to have two blocks of FLASH memory for code execution in as shown in figure 56 each microcontroller of the single ECUs. The first block (Block A) is used to execute the actual code and the second block (Block B) is a spare FLASH block. So, the new software can be programmed into block B in the background while driving the car. Then, the code execution will be swapped from block A to block B and this will happen after all ECUs finish the pre-storing process. The SWAP will be completed with a final restart of the ECUs.

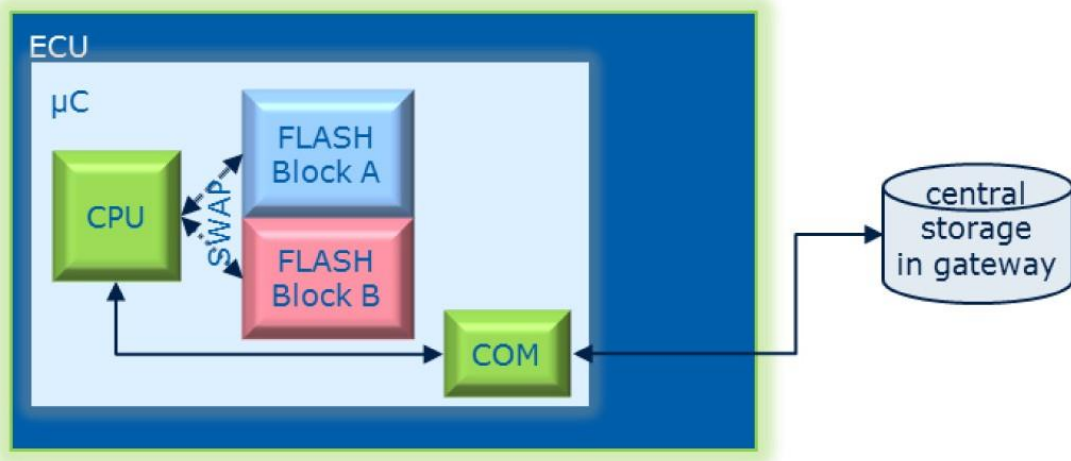


Figure 56:A\B swap process

8.2.1. Advantages and disadvantages of seamless FOTA

The advantages of seamless FOTA are:

- Almost there is zero downtime of the vehicle.
- A restart takes no more than a few milliseconds and then can be added to the power-up or -down cycle as mentioned before.
- There is a fallback solution available within block A if something went wrong in the vehicles network.
- There is always the option to switch back to block A of all ECUs of the actual service update within a few milliseconds, If the new software image of any ECU turns out to be corrupted.

The disadvantages of seamless FOTA are

1. The size of embedded program FLASH is doubling and also today, the mechanisms of swapping process are not available for most microcontrollers.
2. The demand for swapping mechanism of a seamless memory is a challenge for any microcontroller supplier. The potential impact on the entire system architecture should be carefully considered in order to come to a robust implementation of the swap mechanism apart of the necessary availability of components with the required memory sizes.
3. When the FLASH of a microcontroller is doubled from 4MB to 8MB, this obviously will increase the cost.
4. The A/B Swap challenge approach is even graver for components at the upper memory limit of the actual technology note.

8.3. Delta file

8.3.1. What is delta file

As shown in figure 57, Delta file is a very important feature that would add a very great value to the project, as it has a great effect in using large files, it saves time and storage. Delta file is mainly used to reduce the size of the sent file, it is a way of storing or transmitting data in the form of differences (deltas) between sequential data instead of complete files; this is known as data differencing. Delta encoding is also sometimes called **delta compression**.

8.3.2. Advantages of delta compression

- Reduce the potential for security flaws to be exploited before the affected software can be updated.
- Allow the Software to be updated more quickly.
- Reduce error due to wireless network.
- Reduce Transmission Time.

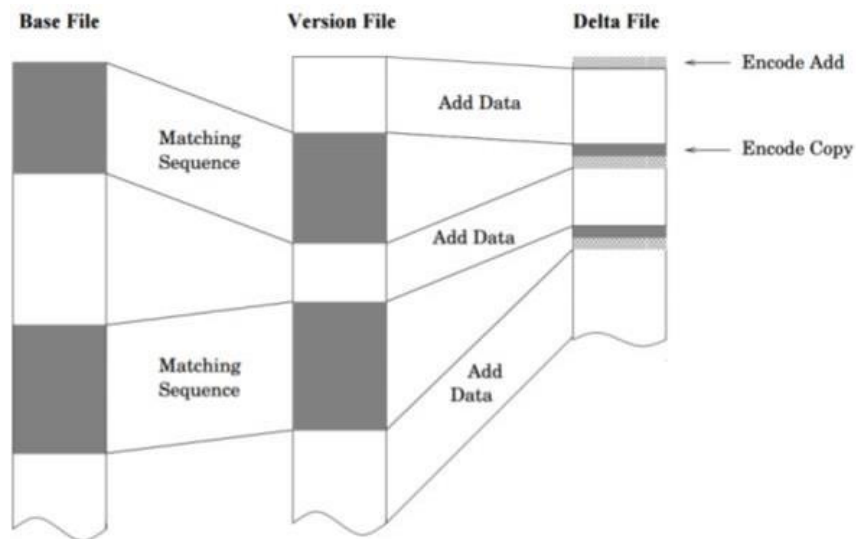


Figure 57:Delta file algorithm

8.3.3. Generating the delta (Patch) file

- **Patch File Consists of 3 files:**

1. Control File containing ADD and INSERT Instructions.
 - A. Each ADD instruction specifies an offset in the old file and a length; after that the correct number of bytes are read from the old file and added to the same number of bytes from the difference file.
 - B. INSERT instruction specify only a length; the specified number of bytes is read from the extra file.
2. Difference File.
3. Extra file.

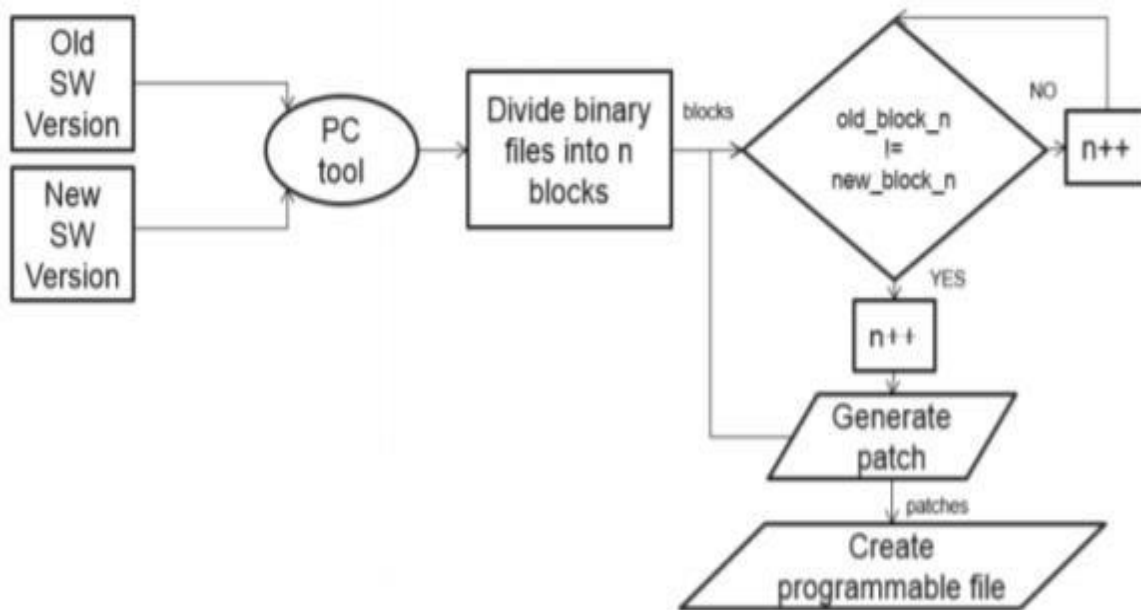


Figure 58: Generating data file

In FOTA delta file should be generated at the server using specific technique and only this file will be sent to the main ECU then after the ECU downloads the delta file it will mix both the old file and the delta file with each other and by using a specific algorithm, a new file will be generated anew file which should be burn on the microcontroller by the bootloader.

APPENDIX A

CAN Bootloader

```
/*
 * BOOTLOADER_prog.c
 *
 */

#include "STD_Types.h"
#include "util.h"
#include "RCC_int.h"
#include "DIO_int.h"
#include "NVIC_int.h"
#include "AFIO_init.h"
#include "AFIO_config.h"
#include "CAN.h"
#include "Timer_int.h"
#include "CANHANDLER_int.h"
#include "CANHANDLER_cfg.h"
#include "FLASH_int.h"
#include "SCB_int.h"
#include "HexDataProcessor_int.h"
#include "BOOTLOADER_int.h"

/*****
/*****Public Functions*****/
/*****/

/*****/
/* Description: Start Bootloader */
/* Input      : u8 u8ProgrammingBank */
/*             Description: Bank to flash software in */
/*             Range: BOOTLOADER_u8BANK0, BOOTLOADER_u8BANK1 */
/*             u8 u8OptionDataBytes */
/*             Description: Current Option Bytes Data0 */
/*             Range: 0x00 ~ 0xFF */
/* Output     : Void */
/* Scope      : Public */
/*****/
void BOOTLOADER_vidStart(u8 u8ProgrammingBank, u8 u8OptionDataBytes) {
    u8 au8DTCs[] = { 20, 30 };
    u8 HexArrayLine[261] = { 0 };
    u8 u8Counter = 0;
    Error_Status enuError = OK;
    u16 u16LineCounter = 0;
    u8 u8RxCount = 0;
    u8 u8UsedBank = 0;
    u32* pu32BaseAddress = 0;
    u8 au8version[3] = { 0 };

    filter_type filters[] = { { CANHANDLER_u8HEXFILEID, DATA_FRAME,
                                STANDARD_FORMAT }, { CANHANDLER_u8ECUSWVERSION, REMOTE_FRAME,
                                STANDARD_FORMAT }, { CANHANDLER_u8GETFLASHBANK, REMOTE_FRAME,
                                STANDARD_FORMAT }, { CANHANDLER_u8ECUDTCs, REMOTE_FRAME,
                                STANDARD_FORMAT } },

    };

    /* RCC Peripherals Initializations */
    RCC_vidEnablePeripheral(RCC_u8GPIOCLK);
    RCC_vidEnablePeripheral(RCC_u8CANCLK);
}
```

```

RCC_vidEnablePeripheral(RCC_u8AFIOCLK); // enable clock for Alternate Function
RCC_vidEnablePeripheral(RCC_u8GPIOBCLK); // enable clock for GPIO B

RCC_vidEnablePeripheral(RCC_u8TIM1CLK);
NVIC_vidEnableInterrupt(NVIC_u8TIM1_UP);
Timer1_UEV_Interrupt();

/* AFIO and DIO Initializations */
AFIO_vidinit();
DIO_vidInit();

/* Interrupt Initializations */
NVIC_vidInit();
NVIC_vidEnableInterrupt(NVIC_u8USB_HP_CAN_TX); // enable interrupt
NVIC_vidEnableInterrupt(NVIC_u8USB_LP_CAN_RX0); // enable interrupt

/* CAN Initialization */
CAN_setup(); // setup CAN interface
CAN_vid_filter_list(filters, CANHANDLER_u8MAXFILTERNUMBERS);
CAN_testmode(0); // Normal, By Salma
CAN_start(); // leave init mode
CAN_waitReady(); // wait til mbx is empty

/* Used Bank Initialization, Setting Data in programming bank to Invalid */
switch (u8ProgrammingBank) {
case BOOTLOADER_u8BANK0:
    u8UsedBank = BOOTLOADER_u8BANK1;
    pu32BaseAddress = (u32*) BOOTLOADER_pu32BANK0BASEADDRESS;

    /* Set The data in the bank as Invalid */
    CLR_BIT(u8OptionDataBytes, BOOTLOADER_u8VALIDBANK0);
    u8OptionDataBytes |= (BOOTLOADER_u8BANKUNVALID
        << BOOTLOADER_u8VALIDBANK0);
    FLASH_vidWriteOptionByteData(FLASH_u8OPTDATA0, u8OptionDataBytes);

    break;
case BOOTLOADER_u8BANK1:
    u8UsedBank = BOOTLOADER_u8BANK0;
    pu32BaseAddress = (u32*) BOOTLOADER_pu32BANK1BASEADDRESS;

    /* Set The data in the bank as Invalid */
    CLR_BIT(u8OptionDataBytes, BOOTLOADER_u8VALIDBANK1);
    u8OptionDataBytes |= (BOOTLOADER_u8BANKUNVALID
        << BOOTLOADER_u8VALIDBANK1);
    FLASH_vidWriteOptionByteData(FLASH_u8OPTDATA0, u8OptionDataBytes);

    break;
default:
    break;
}

/* Erase the flashing bank */
for (u8Counter = 0; u8Counter < 59; u8Counter++) {
    FLASH_vidErasePage(u8Counter + 10 + 59 * u8ProgrammingBank);
}

while (1) {
    // Add your code here.
    do {
        do {
            while (CAN_RxRdy == 0)
                ;
            if (CAN_RxRdy) {
                CAN_RxRdy = 0;
            }
        } while (1);
    } while (1);
}

```

```

if (CAN_RxMsg[u8RxCount].u8ActiveFlag == 1) {
    switch (CAN_RxMsg[u8RxCount].id) {
        case CANHANDLER_u8HEXFILEID:
            for (u8Counter = 0;
                 u8Counter < CAN_RxMsg[u8RxCount].len;
                 u8Counter++) {
                HexArrayLine[u16LineCounter] =
                    CAN_RxMsg[u8RxCount].data[u8Counter];
                u16LineCounter++;
            }
            CANHANDLER_vidSend(CANHANDLER_u8NEXTMSGREQUEST,
                               CAN_u8REMOTEFRAME, (void*) 0, 0);

            break;
        case CANHANDLER_u8ECUSWVERSION:
            CANHANDLER_vidSend(CANHANDLER_u8ECUSWVERSION,
                               CAN_u8DATAFRAME, au8version, 3);
            break;

        case CANHANDLER_u8GETFLASHBANK:
            CANHANDLER_vidSend(CANHANDLER_u8GETFLASHBANK,
                               CAN_u8DATAFRAME, &u8UsedBank, 1);
            break;
        case CANHANDLER_u8ECUDTCs:
            CANHANDLER_vidSend(CANHANDLER_u8ECUDTCs,
                               CAN_u8DATAFRAME, au8DTCs, 1);
            break;

    }
    CAN_RxMsg[u8RxCount].u8ActiveFlag = 0;
    u8RxCount++;
    if (u8RxCount == 3) {
        u8RxCount = 0;
    }
}
}
} while (HexArrayLine[u16LineCounter - 1] != '\r');
u16LineCounter = 0;
Timer1_vidStartCount();
enuError = HexDataProcessor_u32StoreHexInFlash(HexArrayLine);
} while (enuError != limitReached);

switch (u8ProgrammingBank) {
case BOOTLOADER_u8BANK0:
    /* Set The data in the bank as Valud and set bank to bank 0 */
    CLR_BIT(u8OptionDataBytes, BOOTLOADER_u8VALIDBANK0);
    CLR_BIT(u8OptionDataBytes, BOOTLOADER_u8USED BANK);
    u8OptionDataBytes |= ((BOOTLOADER_u8VALIDBANK0
                           << BOOTLOADER_u8VALIDBANK0)
                          | (BOOTLOADER_u8BANK0 << BOOTLOADER_u8USED BANK)
                          | (BOOTLOADER_u8OPTIONBYTESVALID
                           << BOOTLOADER_u8VALIDOPTIONBYTES));
    FLASH_vidWriteOptionByteData(FLASH_u8OPTDATA0, u8OptionDataBytes);

    break;
case BOOTLOADER_u8BANK1:
    /* Set The data in the bank as Valud and set bank to bank 0 */
    CLR_BIT(u8OptionDataBytes, BOOTLOADER_u8VALIDBANK1);
    CLR_BIT(u8OptionDataBytes, BOOTLOADER_u8USED BANK);
    u8OptionDataBytes |= ((BOOTLOADER_u8VALIDBANK1
                           << BOOTLOADER_u8VALIDBANK1)
                          | (BOOTLOADER_u8BANK1 << BOOTLOADER_u8USED BANK)
                          | (BOOTLOADER_u8OPTIONBYTESVALID
                           << BOOTLOADER_u8VALIDOPTIONBYTES));

```

```

        FLASH_vidWriteOptionByteData(FLASH_u8OPTDATA0, u8OptionDataBytes);

        break;
    default:
        break;
    }

    BOOTLOADER_vidJumpToApplication(pu32BaseAddress);

}

}

/*****
/* Description: Jump to Application */
/* Input      : u32* pu32BaseAddress */
/*            Description: Base address of application to jump to */
/* Output     : Void */
/* Scope      : Public */
*****/
void BOOTLOADER_vidJumpToApplication(u32* pu32BaseAddress) {
    void (*pfunc)(void) = 0;

    /* Get Application Reset Vector Address */
    pfunc = *(u32*) (pu32BaseAddress + 1);
    /* Jump to Application Reset Vector */
    pfunc();
}

```

APPENDIX B

CAN Driver

```
/*
 * CAN_Program.c
 *
 */

#include "STD_TYPES.h"
#include "BIT_MATH.h"

#include "CAN_Interface.h"
#include "CAN_Private.h"
#include "CAN_Config.h"

/* Some Variables Declarations */
CAN_Msg CAN_TxMsg[3];           // CAN message for sending
CAN_Msg CAN_RxMsg[3];           // CAN message for receiving
u8 CAN_TxRdy[3] = {0};          // CAN HW ready to transmit a message
u8 CAN_RxRdy = 0;                // CAN HW received a message
u32 filt_counter;

/* CAN Setup and Initialization */
void CAN_vidInit()
{
    /* APB1 clock = 36MHz (MAX Clock for APB1)*/
    u32 brp = 36000000;

    /* Disable Debug Freeze Mode */
    // CAN1->MCR |= 0x00010000;

    /* Setup CAN To Enter Initialization Mode With Automatic Retransmission Enabled */
    CAN1->MCR = CAN_INIT_MODE;

    /* Setup CAN Transmission To Be Driven By FIFO Priority
     * (Priority driven by the request order and not message ID priority)
     */
    CAN1->MCR |= CAN_MSG_FIFO_PRIORITY;

    /* Enable Transmit mailbox empty, FIFO0 message pending and FIFO1 message pending Interrupts */
    CAN1->IER |= (CAN_TMEIE | CAN_FMPIE0 | CAN_FMPIE1);
    /*
     * Setup Bit Timing For CAN Frame
     * Note: These calculations fit for PCLK1 = 36MHz
     */
    brp = (brp / 18) / 500000;           // BaudRate is set to 500k bit/s

    /* Set BTR register so that sample point is at about 72% bit time from bit start */
    /* TSEG1 = 12, TSEG2 = 5, SJW = 4 => 1 CAN bit = 18 TQ, sample at 72% */
    CAN1->BTR &= ~(((0x03) << 24) | ((0x07) << 20) | ((0x0F) << 16) |
    (0x1FF));
    CAN1->BTR |= (((4-1) & 0x03) << 24) | (((5-1) & 0x07) << 20) | (((12-1) & 0x0F) << 16) | ((brp-1)
    & 0x1FF));
}

/* Get CAN Into Operational Mode of Transmitting and
 * receiving (Leave initialization mode to normal mode)
 */
void CAN_vidStart()
{

```

```

    /* Setup CAN To Enter Operating Normal Mode */
    CAN1->MCR &= ~(CAN_INIT_MODE) ;
    /* Wait Until Normal Mode Is Confirmed By the Hardware By Clearing the INAK Bit in the CAN_MSR
Register */
    while (CAN1->MSR & CAN_INIT_ACK);
}
/* Check If TxMailboxes are Empty */
void CAN_vidWaitReady()
{
    /* Check If Transmit MailBox 0 is Empty */
    while ((CAN1->TSR & CAN_TME0) == 0);
    /* Check If Transmit MailBox 1 is Empty */
    while ((CAN1->TSR & CAN_TME1) == 0);
    /* Check If Transmit MailBox 2 is Empty */
    while ((CAN1->TSR & CAN_TME2) == 0);
    /* Confirming That TxMailboxes are Empty */
    CAN_TxRdy[0] = 1;
    CAN_TxRdy[1] = 1;
    CAN_TxRdy[2] = 1;
}
/* Write CAN Frame Contents and Features To Be Send in a Free TxMailbox If any */
void CAN_vidSendMsg(u32 Copy_u8MsgId , u8* MsgData , u8 Copy_u8MsgDataLength , u8 Copy_u8MsgIdFormat ,
u8 Copy_u8MsgType)
{
    u8 local_u8MailboxIndex;    // Index of current TxMailbox
    u8 local_u8DataBytesCounter; // A counter for bytes of CAN message data

    /* Check Which TxMailbox is Empty */
    for (local_u8MailboxIndex = 0 ; local_u8MailboxIndex < 3 ; local_u8MailboxIndex++)
    {
        if(CAN_TxRdy[local_u8MailboxIndex] == 1)
        {
            break;
        }
    }
    /* If no TxMailbox is empty ==> End the function and return the TxMailbox index to be 3 (Non is
Empty) */
    if(local_u8MailboxIndex == 3)
    {
        return ;
    }
    /* Writing of CAN message contents and features */
    /* Load Message Identifier */
    CAN_TxMsg[local_u8MailboxIndex].id = Copy_u8MsgId;
    /*
    * Check the CAN frame type :
    * In Case of CAN_DATA_FRAME ==> There is data to be loaded
    * In Case of CAN_REMOTE_FRAME ==> There is no data to be loaded
    */
    if (Copy_u8MsgType == CAN_DATA_FRAME)
    {
        for(local_u8DataBytesCounter = 0 ; local_u8DataBytesCounter < Copy_u8MsgDataLength ;
local_u8DataBytesCounter++)
        {
            /* Load Message Data */
            CAN_TxMsg[local_u8MailboxIndex].data[local_u8DataBytesCounter] =
MsgData[local_u8DataBytesCounter];
        }
    }
    /* Load Message Data Length */
    CAN_TxMsg[local_u8MailboxIndex].len = Copy_u8MsgDataLength;
    /* Load Message Identifier Format */
    CAN_TxMsg[local_u8MailboxIndex].format = Copy_u8MsgIdFormat;
    /* Load Message Type */

```



```

    CAN_TxMsg[local_u8MailboxIndex].type = Copy_u8MsgType;
    /* Current TxMailBox is Not Empty */
    CAN_TxRdy[local_u8MailboxIndex] = 0;
    /* Request To Transmit The Message */
    CAN_vidTransmitMsg(&(CAN_TxMsg[local_u8MailboxIndex]), local_u8MailboxIndex);
}
/* Set CAN Controller Test Mode */
void CAN_vidSetTestMode(u8 Copy_u32TestMode)
{
    /* Clear Silent and Loop Back bits in BTR Register */
    CAN1->BTR &= ~(CAN_SILENT_TST_MODE | CAN_LBK_TST_MODE);
    /* Check Which Test Mode To Be Applied */
    switch(Copy_u32TestMode)
    {
        case CAN_SILENT: // Silent Mode
            CAN1->BTR |= CAN_SILENT_TST_MODE;
            break;
        case CAN_LBK: // Loop Back Mode
            CAN1->BTR |= CAN_LBK_TST_MODE;
            break;
        case CAN_SILENT_LBK: // Silent Combined With Loop Back Mode
            CAN1->BTR |= CAN_SILENT_LBK_TST_MODE;
            break;
    }
}
/* Request For Transmitting A Message Through CAN Bus */
void CAN_vidTransmitMsg(CAN_Msg* msg , u8 Copy_u8MailBoxIndex)
{
    if(Copy_u8MailBoxIndex < 3)
    {
        /* Reset TIR Register */
        CAN1->sTxMailBox[Copy_u8MailBoxIndex].TIR = (u32)0 ;
        /* Check The Format of Message Identifier */
        if(msg->format == CAN_STANDARD_ID)
        {
            CAN1->sTxMailBox[Copy_u8MailBoxIndex].TIR |= (u32)(msg->id << 21) | (u32) (CAN_ID_STD);
        }
        else if(msg->format == CAN_EXTENDED_ID)
        {
            CAN1->sTxMailBox[Copy_u8MailBoxIndex].TIR |= (u32)(msg->id << 3) | (CAN_ID_EXT);
        }
        else
        {
            /* Return Error */
        }
        /* Check The Type of Message */
        if(msg->type == CAN_RTR_DATA)
        {
            CAN1->sTxMailBox[Copy_u8MailBoxIndex].TIR |= CAN_RTR_DATA;
            CAN1->sTxMailBox[Copy_u8MailBoxIndex].TDLR = (((u32)msg->data[3] << 24) |
                ((u32)msg->data[2] << 16) |
                ((u32)msg->data[1] << 8) |
                ((u32)msg->data[0]));
            CAN1->sTxMailBox[Copy_u8MailBoxIndex].TDHR = (((u32)msg->data[7] << 24) |
                ((u32)msg->data[6] << 16) |
                ((u32)msg->data[5] << 8) |
                ((u32)msg->data[4]));
        }
        else if(msg->type == CAN_RTR_REMOTE)
        {
            CAN1->sTxMailBox[Copy_u8MailBoxIndex].TIR |= CAN_RTR_REMOTE;
        }
        else
        {

```

```

        /* Return Error */
    }
    /* Reset TDTR Register */
    CAN1->TxMailBox[Copy_u8MailBoxIndex].TDTR = (u32)0 ;
    CAN1->TxMailBox[Copy_u8MailBoxIndex].TDTR |= (u32)(msg[Copy_u8MailBoxIndex].len);
    /* Enable Transmit mailbox empty Interrupt */
    CAN1->IER |= CAN_TMEIE;
    /* Request For Transmission For The Corresponding mailbox */
    CAN1->TxMailBox[Copy_u8MailBoxIndex].TIR |= CAN_TXRQ;
}
else
{
    /* Return Error */
}
}
/* Read Pending Received Message in One of RxFIFOMailboxes Then Release It */
void CAN_vidReadMsg(CAN_Msg* msg , u8 Copy_u8FIFOMailBoxIndex)
{
    if(Copy_u8FIFOMailBoxIndex == CAN_FIFO_MB_0)
    {
        /* Read Identifier Information */
        if ((CAN1->sFIFOMailBox[0].RIR & CAN_ID_EXT) == 0) { // Standard ID

            msg->format = CAN_STANDARD_ID;

            msg->id = (u32)0x000007FF & (CAN1->sFIFOMailBox[0].RIR >> 21);
        } else { // Extended ID
            msg->format = CAN_EXTENDED_ID;

            msg->id = (u32)0x0003FFFF & (CAN1->sFIFOMailBox[0].RIR >> 3);
        }
        /* Read Type Information */
        if ((CAN1->sFIFOMailBox[0].RIR & CAN_RTR_REMOTE) == 0) {

            msg->type = CAN_DATA_FRAME; // DATA FRAME

        } else {
            msg->type = CAN_REMOTE_FRAME; // REMOTE FRAME
        }
        /* Read Length (Number of received bytes) */
        msg->len = (u8)0x000000FF & CAN1->sFIFOMailBox[0].RDTR;
        /* Read Data Bytes */
        msg->data[0] = (u32)0x000000FF & (CAN1->sFIFOMailBox[0].RDLR);
        msg->data[1] = (u32)0x000000FF & (CAN1->sFIFOMailBox[0].RDLR >> 8);
        msg->data[2] = (u32)0x000000FF & (CAN1->sFIFOMailBox[0].RDLR >> 16);
        msg->data[3] = (u32)0x000000FF & (CAN1->sFIFOMailBox[0].RDLR >> 24);

        msg->data[4] = (u32)0x000000FF & (CAN1->sFIFOMailBox[0].RDHR);
        msg->data[5] = (u32)0x000000FF & (CAN1->sFIFOMailBox[0].RDHR >> 8);
        msg->data[6] = (u32)0x000000FF & (CAN1->sFIFOMailBox[0].RDHR >> 16);
        msg->data[7] = (u32)0x000000FF & (CAN1->sFIFOMailBox[0].RDHR >> 24);
        CAN_vidReleaseMessage(CAN_FIFO_MB_0);
    }
    else if(Copy_u8FIFOMailBoxIndex == CAN_FIFO_MB_1)
    {
        /* Read Identifier Information */
        if ((CAN1->sFIFOMailBox[1].RIR & CAN_ID_EXT) == 0) { // Standard ID

            msg->format = CAN_STANDARD_ID;

            msg->id = (u32)0x000007FF & (CAN1->sFIFOMailBox[1].RIR >> 21);
        } else { // Extended ID
            msg->format = CAN_EXTENDED_ID;

```

```

        msg->id      = (u32)0x0003FFFF & (CAN1->sFIFOMailBox[1].RIR >> 3);
    }
    /* Read Type Information */
    if ((CAN1->sFIFOMailBox[1].RIR & CAN_RTR_REMOTE) == 0) {

        msg->type = CAN_DATA_FRAME;                // DATA FRAME

    } else {
        msg->type = CAN_REMOTE_FRAME;              // REMOTE FRAME
    }
    /* Read Length (Number of received bytes) */
    msg->len = (u8)0x0000000F & CAN1->sFIFOMailBox[1].RDTR;
    /* Read Data Bytes */
    msg->data[0] = (u32)0x000000FF & (CAN1->sFIFOMailBox[1].RDLR);
    msg->data[1] = (u32)0x000000FF & (CAN1->sFIFOMailBox[1].RDLR >> 8);
    msg->data[2] = (u32)0x000000FF & (CAN1->sFIFOMailBox[1].RDLR >> 16);
    msg->data[3] = (u32)0x000000FF & (CAN1->sFIFOMailBox[1].RDLR >> 24);

    msg->data[4] = (u32)0x000000FF & (CAN1->sFIFOMailBox[1].RDHR);
    msg->data[5] = (u32)0x000000FF & (CAN1->sFIFOMailBox[1].RDHR >> 8);
    msg->data[6] = (u32)0x000000FF & (CAN1->sFIFOMailBox[1].RDHR >> 16);
    msg->data[7] = (u32)0x000000FF & (CAN1->sFIFOMailBox[1].RDHR >> 24);
    CAN_vidReleaseMessage(CAN_FIFO_MB_1);
}
else
{
    /* Return Error */
}
}
/* Release RxFIFOMailbox */
void CAN_vidReleaseMessage(u8 Copy_u8FIFOMailBoxIndex)
{
    switch(Copy_u8FIFOMailBoxIndex)
    {
        case CAN_FIFO_MB_0:
            /* Release RxFIFOMailbox 0 */
            CAN1->RF0R |= CAN_RFOM0;
            break;
        case CAN_FIFO_MB_1:
            /* Release RxFIFOMailbox 1 */
            CAN1->RF1R |= CAN_RFOM1;
            break;
        // default:
        //      /* Return Error */
    }
}
/* Get Number of Pending Messages of One of RxFIFOMailboxes */
u8 CAN_u8GetNumberOfPendingMessage(u8 Copy_u8FIFOMailBoxIndex)
{
    u8 local_u8Num = 0; // Number of pending messages
    if(Copy_u8FIFOMailBoxIndex == CAN_FIFO_MB_0)
    {
        local_u8Num = (u8)((CAN1->RF0R) & (CAN_FMP0));
    }
    else if(Copy_u8FIFOMailBoxIndex == CAN_FIFO_MB_1)
    {
        local_u8Num = (u8)((CAN1->RF1R) & (CAN_FMP1));
    }
    return local_u8Num;
}
/* Enable and Configure Mode and Scale of Specific
 * Filter Bank To Be Ready For Message Reception
 */

```

```

void CAN_vidWriteFilter(Filter_Type *pstrfilter)
{
    static u16 CAN_filterIdx = 0; // Filter Bank Index
    u32 CAN_msgId = 0; // Message id to be compared with received message id
    /* Check If Filter Memory is Full */
    if (CAN_filterIdx > 13) {
        return;
    }
    /* Setup Identifier Information */
    if (pstrfilter->Type == CAN_STANDARD_ID) { // Standard ID
        CAN_msgId |= (u32)(pstrfilter->Id << 21) | CAN_ID_STD;
    } else { // Extended ID
        CAN_msgId |= (u32)(pstrfilter->Id << 3) | CAN_ID_EXT;
    }
    if (pstrfilter->Frame == CAN_REMOTE_FRAME) { // Remote Frame
        CAN_msgId |= CAN_RTR_REMOTE;
    }
    else // Remote Frame
    {
        CAN_msgId |= CAN_RTR_DATA;
    }
    /* Deactivate Filter */
    CAN1->FA1R &= ~(u32)(1 << CAN_filterIdx);

    /* Initialize Filter Bank */
    /* Set 32-bit Scale Configuration */
    CAN1->FS1R |= (u32)(1 << CAN_filterIdx);
    /* Set Identifier List Mode Configuration ==> 2 32-bit Identifiers Registers (Filters)*/
    CAN1->FM1R |= (u32)(1 << CAN_filterIdx);
    /* 32-bit Identifier */
    CAN1->sFilterRegister[CAN_filterIdx].FR1 = CAN_msgId;
    CAN1->sFilterRegister[CAN_filterIdx].FR2 = CAN_msgId;
    /* Assign Filter Bank to FIFO 0 */
    CAN1->FFA1R &= ~(u32)(1 << CAN_filterIdx);
    /* Activate Filter Bank */
    CAN1->FA1R |= (u32)(1 << CAN_filterIdx);
    /* Increase Filter Bank Index */
    CAN_filterIdx += 1;
}
/* Setup List of Filters */
void CAN_vidFilterList(Filter_Type *pstrfilter,u8 Copy_u8NumOfFilters)
{
    /* Set Initialization Mode for Filter Banks */
    CAN1->FMR |= CAN_FINIT;
    for (filt_counter=0 ; filt_counter<Copy_u8NumOfFilters ; filt_counter++)
    {
        /* Enable Reception of Messages */
        CAN_vidWriteFilter(&(pstrfilter[filt_counter]));
    }
    /* Reset Initialization Mode for Filter Banks (Active Filter Mode) */
    CAN1->FMR &= ~(CAN_FINIT);
}
/* Get Filter Match Index Through it CAN Message Passed To RxFIFOMailBox0 */
u8 CAN_u8GetFilterMatchIndex()
{
    u8 local_u8Index ;
    local_u8Index = (u8)(((u32)0x000000FF) & (CAN1->sFIFOMailBox[0].RDTR >> 8));
    return local_u8Index;
}

/***** CAN Interrupt Handlers*****/

/* CAN Transmit Interrupt Handler */

```

```

void USB_HP_CAN1_TX_IRQHandler (void)
{
    /* Check If the Transmission Request of TxMailbox0 is Completed */
    if (CAN1->TSR & (CAN_RQCP0))
    {
        /* Reset Transmission Request of TxMailbox0 */
        CAN1->TSR |= (CAN_RQCP0);
        /* Disable TME Interrupt */
        CAN1->IER &= ~(CAN_TMEIE);
        CAN_TxRdy[0] = 1;
    }
    /* Check If the Transmission Request of TxMailbox1 is Completed */
    else if (CAN1->TSR & (CAN_RQCP1))
    {
        /* Reset Transmission Request of TxMailbox1 */
        CAN1->TSR |= (CAN_RQCP1);
        /* Disable TME Interrupt */
        CAN1->IER &= ~(CAN_TMEIE);
        CAN_TxRdy[1] = 1;
    }
    /* Check If the Transmission Request of TxMailbox2 is Completed */
    else if (CAN1->TSR & (CAN_RQCP2))
    {
        /* Reset Transmission Request of TxMailbox2 */
        CAN1->TSR |= (CAN_RQCP2);
        /* Disable TME Interrupt */
        CAN1->IER &= ~(CAN_TMEIE);
        CAN_TxRdy[2] = 1;
    }
    else
    {
        /* Do Nothing */
    }
}

/* CAN Receive Interrupt Handler 0 */
void USB_LP_CAN1_RX0_IRQHandler (void) {
    static u8 u8RxMsgIndex = 0;
    if (CAN1->RF0R & CAN_FMP0)
    {
        /* message pending ?
        if (CAN_RxMsg[u8RxMsgIndex].u8ActiveFlag == 0)
        {
            CAN_vidReadMsg(&(CAN_RxMsg[u8RxMsgIndex]), CAN_FIFO_MB_0); // Read the message
            CAN_RxMsg[u8RxMsgIndex].u8ActiveFlag = 1;
            CAN_RxRdy = 1; // Set receive flag
            u8RxMsgIndex++;
            if (u8RxMsgIndex == 3)
            {
                u8RxMsgIndex = 0;
            }
        }
    }
}

/* CAN Receive Interrupt Handler 1 */
void CAN_RX1_IRQHandler (void) {
    static u8 u8RxMsgIndex = 0;
    if (CAN1->RF0R & CAN_FMP1)
    {
        /* message pending ?
        if (CAN_RxMsg[u8RxMsgIndex].u8ActiveFlag == 0)
        {
            CAN_vidReadMsg(&(CAN_RxMsg[u8RxMsgIndex]), CAN_FIFO_MB_1); // Read the message
            CAN_RxMsg[u8RxMsgIndex].u8ActiveFlag = 1;
            CAN_RxRdy = 1; // Set receive flag
            u8RxMsgIndex++;
            if (u8RxMsgIndex == 3)

```

```
        {
            u8RxMsgIndex = 0;
        }
    }
}
```

APPENDIX C

Hex File Parsing and Sending Over CAN

```
import can
import requests as req
url = 'http://192.168.137.132/download/GPIO.txt'
with req.get(url) as f:
    with open('lupdate.txt', 'wb') as file:
        file.write(f.content)

file = open('lupdate.txt')
# lin_num = file.readlines()
while True:
    data = file.readline()
    if data == '':
        file.close()
        break
    else:
        print(data)
        data = str.encode(data)
        msg = can.Message(arbitration_id=0x1, data=data, is_extended_id=False)
        bus.send(msg)
        s = bus.recv(None)
        ok = str.encode('ok')
        if s != ok:
            file.close()
            break
```

References

- [1] <https://www.futurebridge.com/blog/over-the-air-software-updates-reaping-benefits-for-the-automotive-industry/>
- [2] <https://www.marketresearchfuture.com/reports/automotive-over-the-air-updates-market-7606>
- [3] <https://www.icpdas-usa.com/cancheck.html>
- [4] <https://www.totalphase.com/blog/2019/08/5-advantages-of-can-bus-protocol/>
- [5] RM0008 Reference Manual
- [6] <http://www.cse.dmu.ac.uk/~eg/tele/CanbusIDandMask.html>
- [7] <https://copperhilltech.com/blog/controller-area-network-can-bus-message-frame-architecture/>
- [8] https://en.wikipedia.org/wiki/General_Packet_Radio_Service
- [9] STM32F103x Programming Manual Data Sheet.
- [10] STM32F10x Register Description Data Sheet.
- [11] <https://searchstorage.techtarget.com/definition/flash-memory>
- [12] <https://www.hyperstone.com/en/Solid-State-bit-density-and-the-Flash-Memory-Controller-1235.12728.html>
- [13] PM0075 Programming manual, STM32F10xxx Flash memory microcontrollers
- [14] https://en.wikipedia.org/wiki/Intel_HEX
- [15] https://en.wikipedia.org/wiki/Graphical_user_interface
- [16] https://www.slant.co/versus/16724/22768/~tkinter_vs_pyqt
- [17] <https://doc.qt.io/qt-5/classes.html>
- [18] <https://docs.djangoproject.com/en/4.0/>