Alexandria University
Faculty of Engineering
Communications and Electronics Department

Firmware over the Air

(FOTA)

Prepared By

Alaa Hisham Mostafa Morsy.

Esraa Khamis Abdel-Hamid El-Bakly.

Magdy Salah El-Din Abo-Khedr.

Mohamed Ahmed Ali Hassan.

Mohamed Mahmoud Mohamed Ibrahim.

Mayar Saber Fahmy Saber.

Yehia Ahmed Ibrahim Shahin.

Yehia Ehab Orfy Mohamed.

Supervised By

Prof. Dr. Hesham Tolba

2021 - 2022

Abstract

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

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

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

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

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

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

Table of Contents

List of Figures

List of Abbreviations

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

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

Chapter 1: Introduction

1.1. Introduction to FOTA

1.1.1. What is FOTA?

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

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

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

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

1.1.2. Why do we need FOTA?

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

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

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

1.1.3. Examples of FOTA

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

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

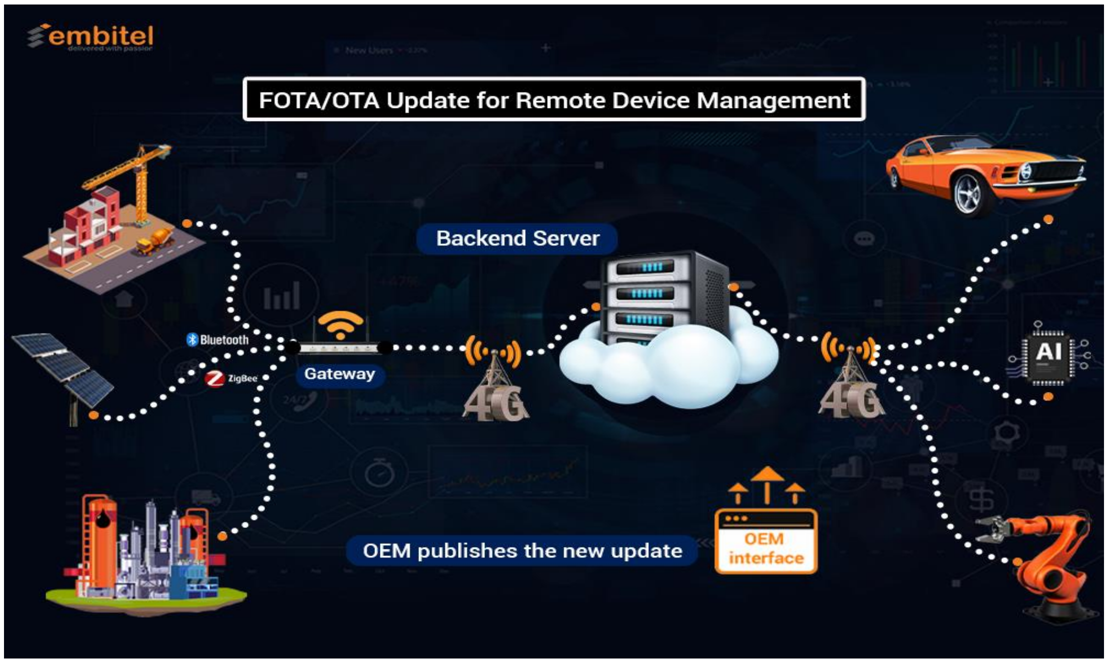


Figure 1.1 FOTA/OTA update for Remote Device Management.

1.2. Flashing Techniques for Microcontrollers

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

1.2.1. Off-Circuit Programming

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

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

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

1.2.2. In-Circuit Programming

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

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

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

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

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

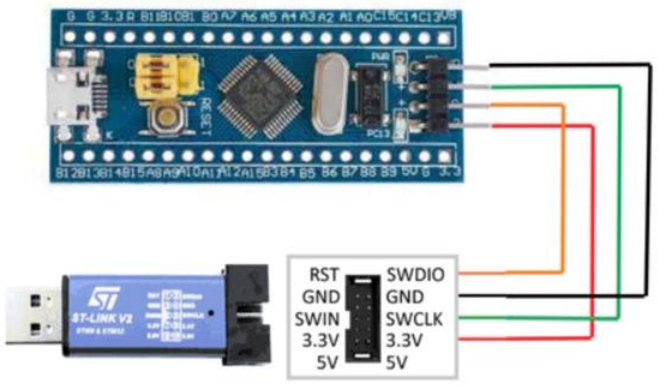


Figure 1.2 ST-Link Programmer.

1.2.3. In-Circuit Programming with Bootloader

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

1.3. Software Development Lifecycle

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

1.3.1. What is Software Development Lifecycle?

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

SDLC is a way to measure and improve the development process. It allows a fine-grain analysis of each step of the process. This, in turn, helps companies maximize efficiency at each stage. As computing power increases, it places a higher demand on software and developers. Companies must reduce costs, deliver software faster, and meet or exceed their customers' needs. SDLC helps achieve these goals by identifying inefficiencies and higher costs and fixing them to run smoothly.

1.3.2. How the Software Development Life Cycle Works

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

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

1.3.3. The Seven Phases of the SDLC

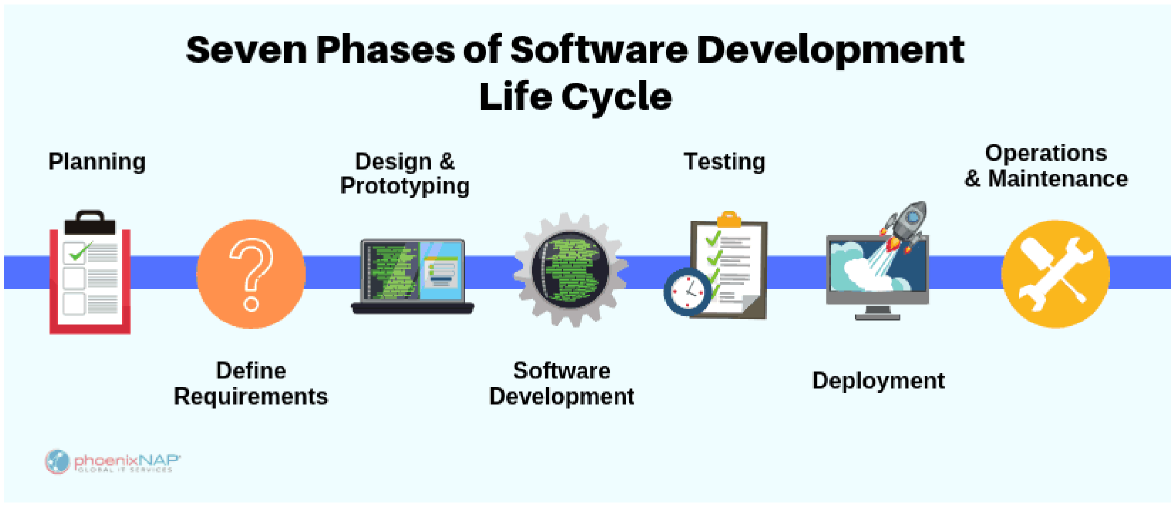


Figure 1.3 The Seven Phases of the SDLC

1) Planning

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

Planning can also include feedback from stakeholders. Stakeholders are anyone who stands to benefit from the application. Try to get feedback from potential customers, developers, subject matter experts, and sales reps.
Planning should clearly define the scope and purpose of the application. It plots the course and provisions the team to effectively create the software. It also sets boundaries to help keep the project from expanding or shifting from its original purpose.

2) Define Requirements

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

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

3) Design and Prototyping

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

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

Prototyping can be a part of the Design phase. A prototype is like one of the early versions of software in the Iterative software development model. It demonstrates a basic idea of how the application looks and works. This “hands-on” design can be shown to stakeholders. Use feedback o improve the application. It’s less expensive to change the Prototype phase than to rewrite code to make a change in the Development phase.

4) Software Development

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

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

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

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

5) Testing

It's critical to test an application before making it available to users. Much of the testing can be automated, like security testing. Another testing can only be done in a specific environment – consider creating a simulated production environment for complex deployments. Testing should ensure that each function works correctly. Different parts of the application should also be tested to work seamlessly together—performance test, to reduce any hangs or lags in processing. The testing phase helps reduce the number of bugs and glitches that users encounter. This leads to higher user satisfaction and a better usage rate.

6) Development

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

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

7) Operations and Maintenance

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

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

Chapter 2: Problem Definition

2.1. Problem Definition

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

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

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

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

FOTA can update the firmware of a car at any location whether it is an assembly shop, a dealer location, a service workshop, or the parking space of the owner. Besides, FOTA allows the download of firmware in the background when the vehicle is running and informs the driver, owner, or service person when the installation of the updated firmware can be started.

2.2. Implementing FOTA

FOTA implementation is a three-step process:

a) Generating and storing required new versions of firmware and its updates in the databases located on cloud-based servers accessible to all stakeholders.

b) Downloading required firmware in the local storage unit attached to the Telematics Control Unit in the vehicle using wireless networks.

c) Installation of the new firmware and updating of ECUs in the vehicle.

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

2.3. Major Challenges

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

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

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

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

2.4. FOTA Benefits

Continuous improvement

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

Reduces risk

With FOTA giving the IoT device distributor or manufacturer the ability to continually configure devices, and post-distribution, they also gain the ability to remain compliant with evolving industry standards. This expands product lifetime and offers greater flexibi5lity to manage the devices on the edge.

Cost-effective

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

Gets to market quicker

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

Sustainability and adaptability

Better sustainability and adaptability of devices with the help of FOTA. As small updates are possible in the FOTA system, software/firmware development teams have the leeway to carry on with their tasks of development even in the post-sales scenario of a product.

Additional revenue streams.

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

2.6. Market Research

Automotive Firmware over the Air (FOTA) Market Statistics 2030

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

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

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

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

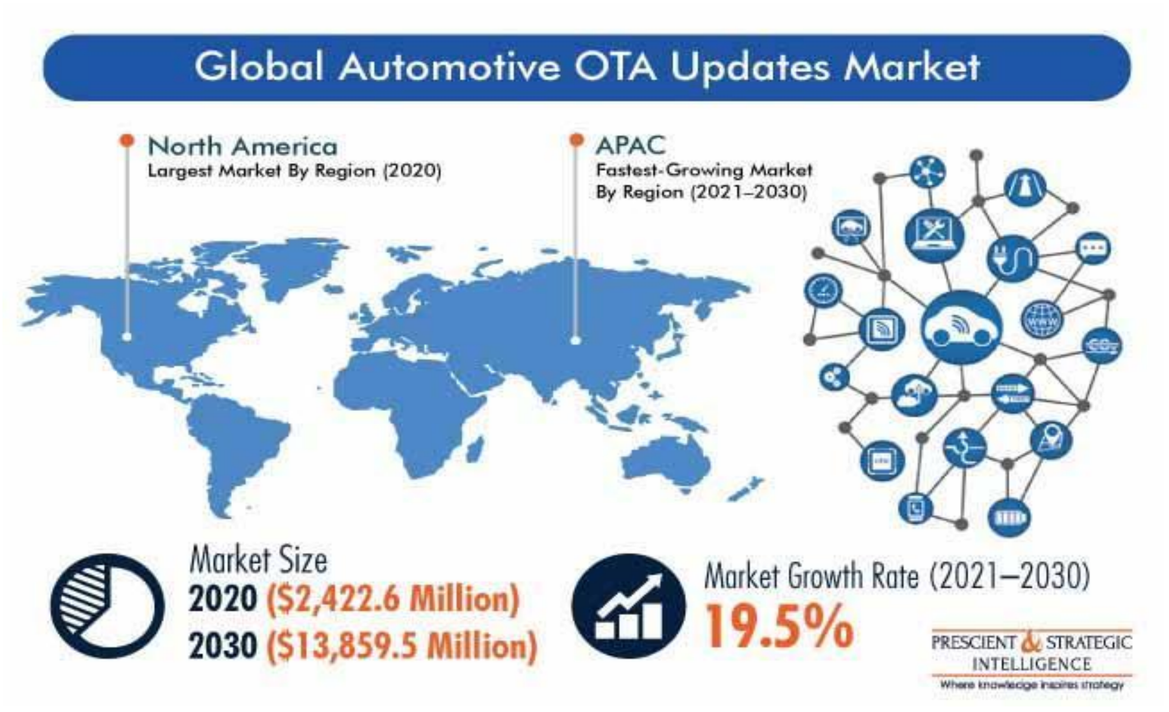


Figure 2.1 Global Automotive OTA Updates Mark

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

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

Major automakers are being encouraged to turn their focus to linked vehicles and either develop new variants or include cutting-edge connection capabilities into the existing models as a result of the growing acceptance of internet of things (IoT) technology. This is a response to people's rising desire for cutting-edge navigation, telematics, and infotainment systems. For proper operation, these systems depend on the software that is placed on the embedded hardware of the vehicle. Thus, key development in the market for automotive over-the-air updates is the increasing adoption of smart, connected technologies.

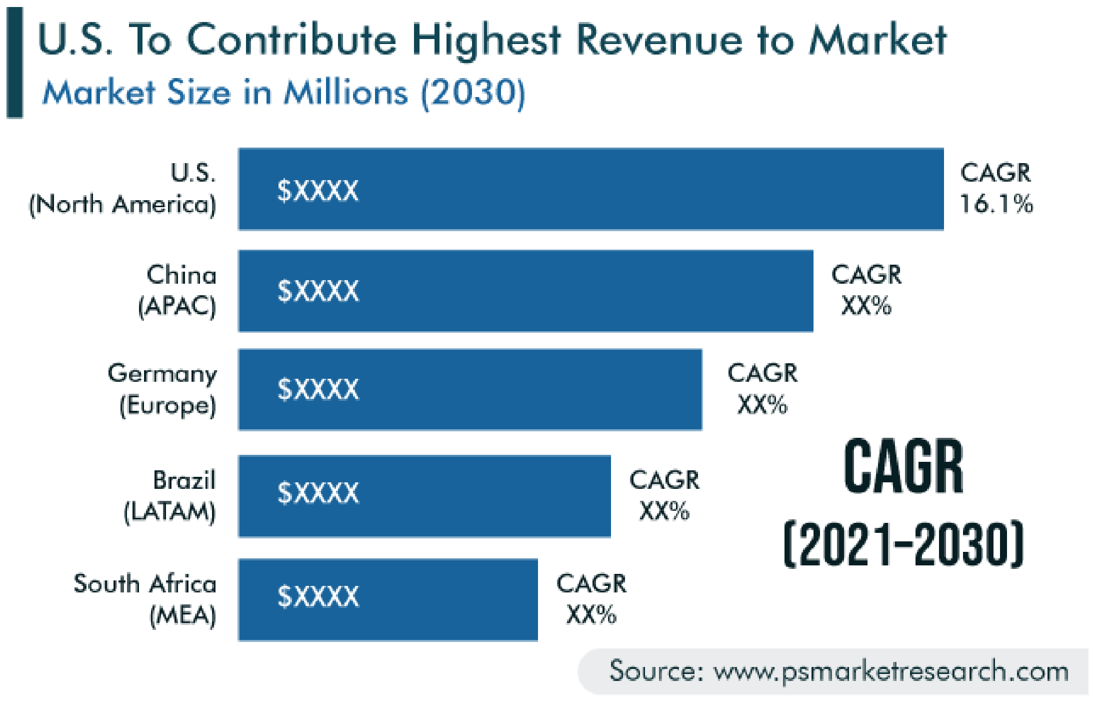
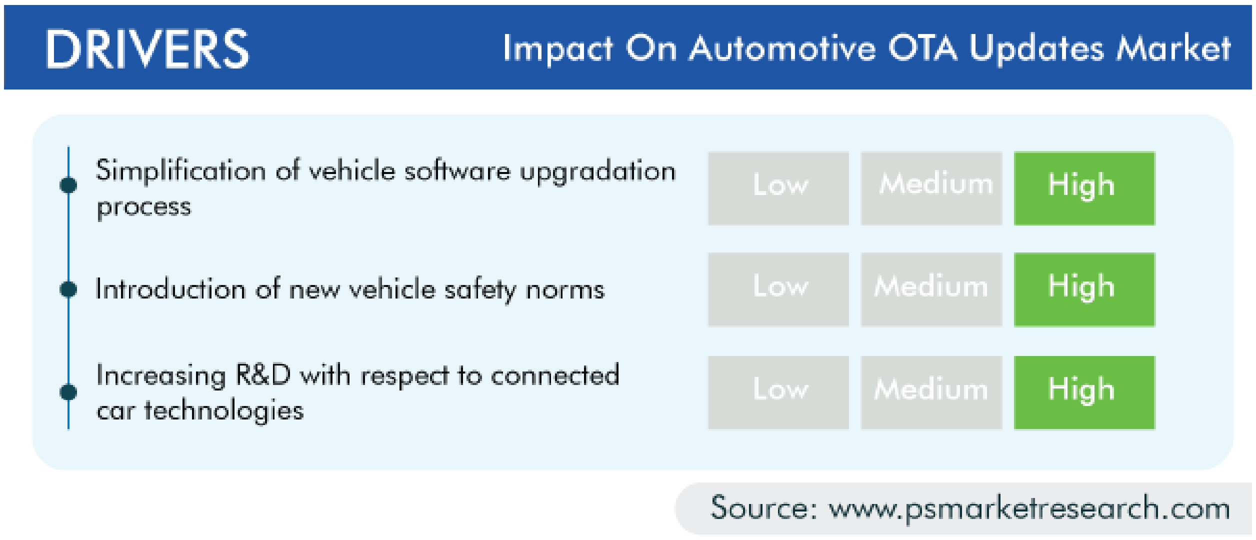


Figure 2.2 Market Size

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

The global automotive OTA updates market is anticipated to be driven by the tight standards relating to vehicle and passenger safety. The primary priority for automotive manufacturers worldwide has always been vehicle safety. It has become crucial for manufacturers to adapt their products to the current vehicle standards as a result of the implementation of greater safety laws and monitoring standards. Additionally, automakers are concentrating quickly on upgrading their vehicles to ensure compliance with the updated norms and regulations due to the rising sophistication of automotive technology and the increasing degree of programmability.



DRIVERS	Impact On Automotive OTA Updates Market		
Simplification of vehicle software upgradation process	Low	Medium	High 🟩
Introduction of new vehicle safety norms	Low	Medium	High 🟩
Increasing R&D with respect to connected car technologies	Low	Medium	High 🟩

Source: www.psmarketresearch.com

Figure 2.3 Impact on Automotive Updates Market.

Chapter 3 : System Design

3.1. System Diagram

3.1.1. Abstract View

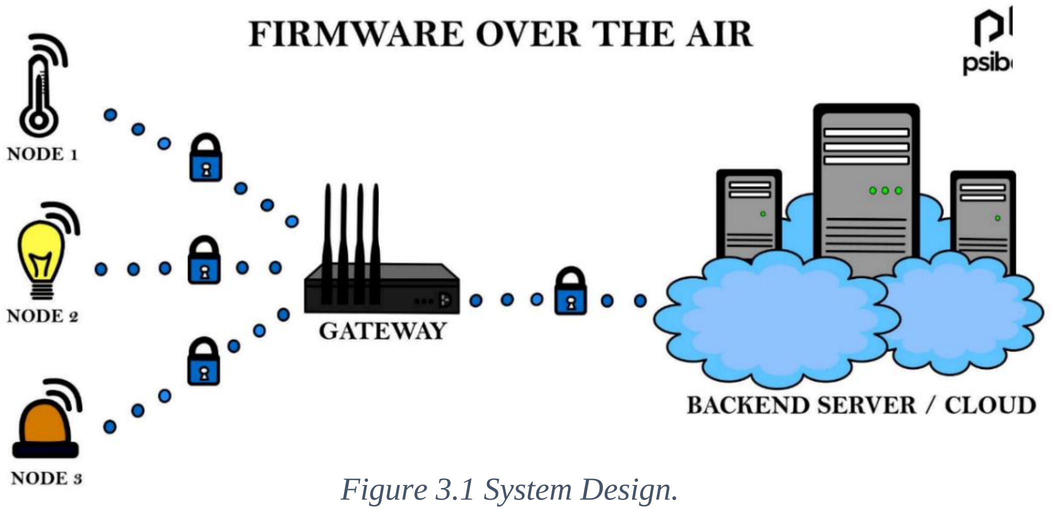


Figure 3.1 System Design.

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

1- Backend Server/Cloud

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

2- Telematics Unit

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

3- Gateway (STM32)

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

4- Target ECU

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

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

3.1.2. Implemented View

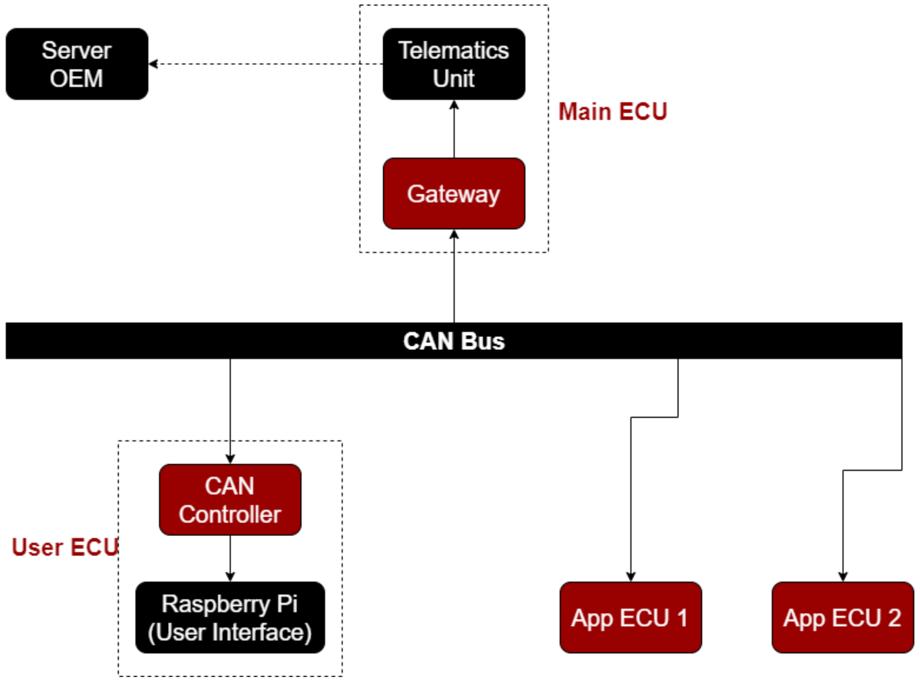


Figure 3.2 Implemented View System.

As an overview of our implemented system, we used:

- Google Firebase for the OEM server as it provides real-time database and free large storage.
- ESP32 as the telematics unit, as it has a Wi-Fi module which will make it easy to connect to the server and will be connected to gateway through a UART bus.
- Gateway is implemented using STM32F103C8T6 ARM-based microcontroller, as it has the required communication protocols.
- CAN transceiver MCP 2551 which serves as the interface between a CAN protocol controller and the physical bus.
- And the same microcontroller is used which is the applications of the target ECUs and the user interface.

3.2. System Components

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

3.2.1. Main ECU

Telematics Unit

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

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

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

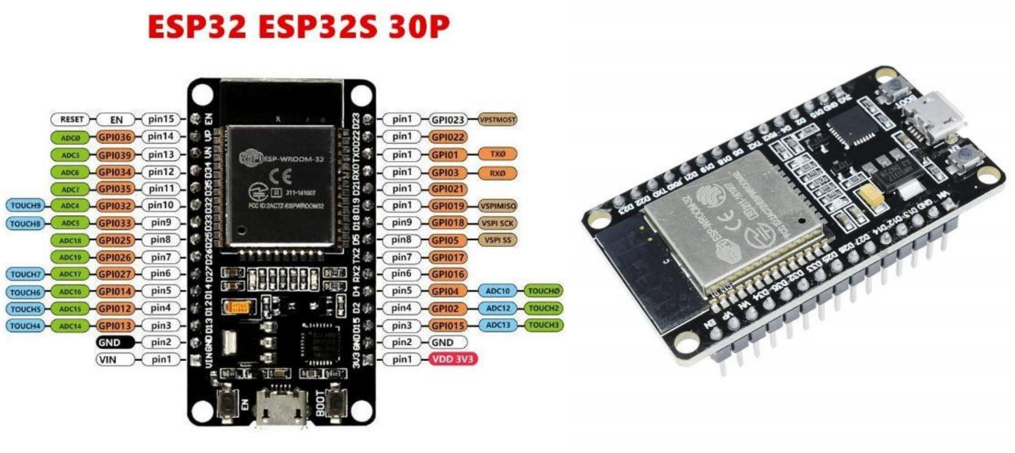


Figure 3.3 ESP32 IC.

The main function of the ESP32 is:

1- Communicate to the server:
   - Download the updated file.
   - Decrypt downloaded update files.
   - Send a request to download update file.
   - Send diagnostics to the server.

2- Communicate to the Gateway:
   - Send update file to the Gateway.
   - Receive download request from the user.

<u>ESP32 Specs:</u>

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

Gateway

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

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

- Receive the downloaded update from the Telematics unit before sending it to the target ECU
- Get important data from the downloaded update like the CRC and target ECU ID.
- Save information about connected ECUs, including ECUs IDs, software version of every ECU and any important data related to the connected ECUs.
- Determine which ECU this code is for, and that is one of the main tasks to make sure the code is delivered to the correct ECU
- Notify the node when the update is complete so the node can notify the server too that the code update is done.

<u>STM32f103c8t6 microcontroller</u>

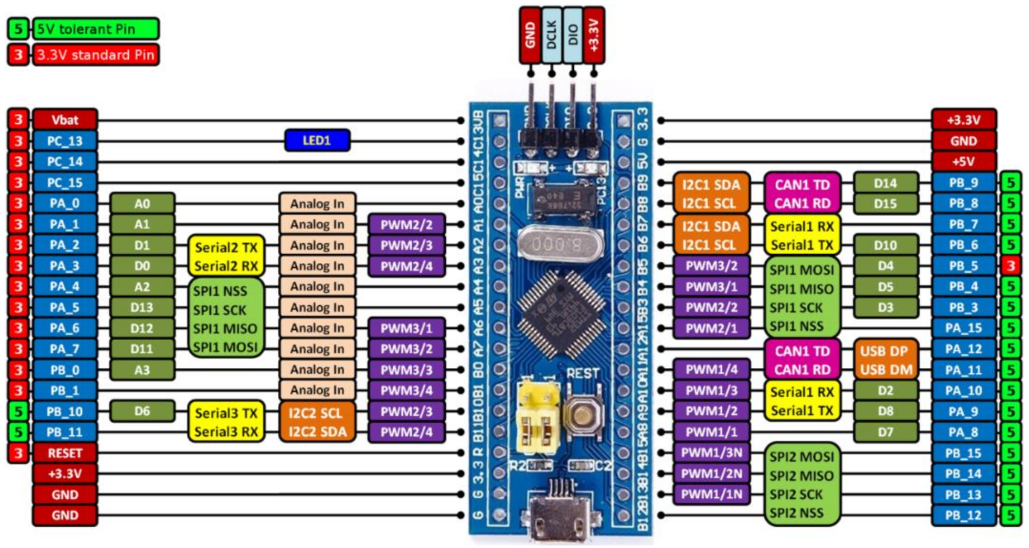


Figure 3.4 STM32 IC.

STM32f103c8t6 Specs:

Pinout Configuration

Category	Pin Name	Details
Power	3.3V, 5V, GND	• 3.3V – Regulated output voltage from the onboard regulator (drawing current is not recommended), CAN also be used to supply the chip. • 5V from USB or onboard regulator can be used to supply the onboard 3.3V regulator. • GND – Ground pins
Analog Pins	PA0 – PA7 PB0 – PB1	Pins act as ADCs with 12-bit resolution
Input/output pins	PA0 – PA15 PB0 – PB15 PC13 – PC15	37 General-purpose I/O pins.
Serial	TX1, RX1 TX2, RX2 TX3, RX3	UART with RTS and CTS pins
External interrupts	PA0 – PA15 PB0 – PB15 PC13 – PC15	All digital pins have interrupt capability
PWM	PA0 – PA3	15 PWM pins total

	PA6 – PA10 PB0 - PB1 PB6 – PB9	
SPI	MISO0, MOSI0, SCK0, CS0 MISO1, MOSI1, SCK1, CS0	2 SPI
Inbuilt LED	PC13	LED to act as a general-purpose GPIO indicator
I^{2}C	SCL1, SDA1 SCL2, SDA2	Inter-Integrated Circuit communication ports
CAN	CAN0TX, CAN0RX	CAN bus ports

Technical Specifications

Microcontroller	STM32F103C8T6
Operating voltage	3.3V
Analog inputs	10
Digital I/O pins	37
DC source/sink from I/O pins	6mA
Flash memory (KB)	64/128
SRAM	20KB
Frequency (clock speed)	72MHz max.
Communication	I^2C, SPI, UART, CAN, USB

3.2.2. App ECU

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

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

It's composed mainly of:

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

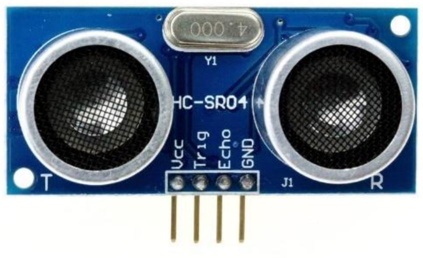


Figure 3.5 HC- SR04 IC.

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

Figure 3.6 DC Motor with Encoder.

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

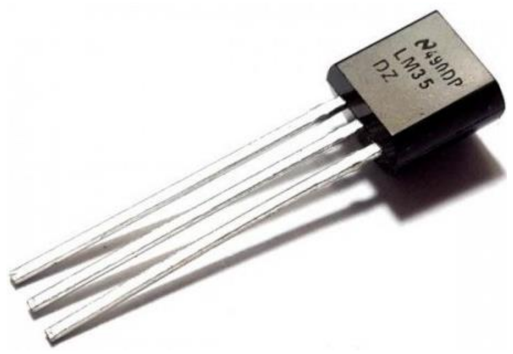


Figure 3.7 LM35 Temperature Sensor.

3.2.3. User ECU

Raspberry Pi

The Raspberry Pi is used for user interface, it's connected with a 5-inch touch screen that displays a full functioning GUI that enables the user to interact with all vehicle systems.

The GUI consists of Splash Screen (loading screen) Main Window which displays time and navigates through 8 separate apps.

- Phone.
- Guide.
- Calendar.
- Settings.
- Music.
- Video.
- Radio.
- Weather.

The most important app is the settings app which enables the user to check for new firmware for the vehicle, track his update history, run a full system diagnostic check on the vehicle to detect the occurrence of any error and send this data to the server to be resolved by the company.

Can Controller

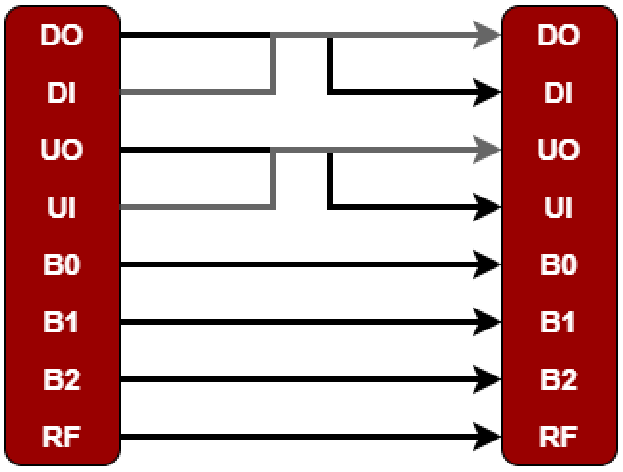


Figure 3.8 Asynchronous Octal Communication Pins.

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

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

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

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

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

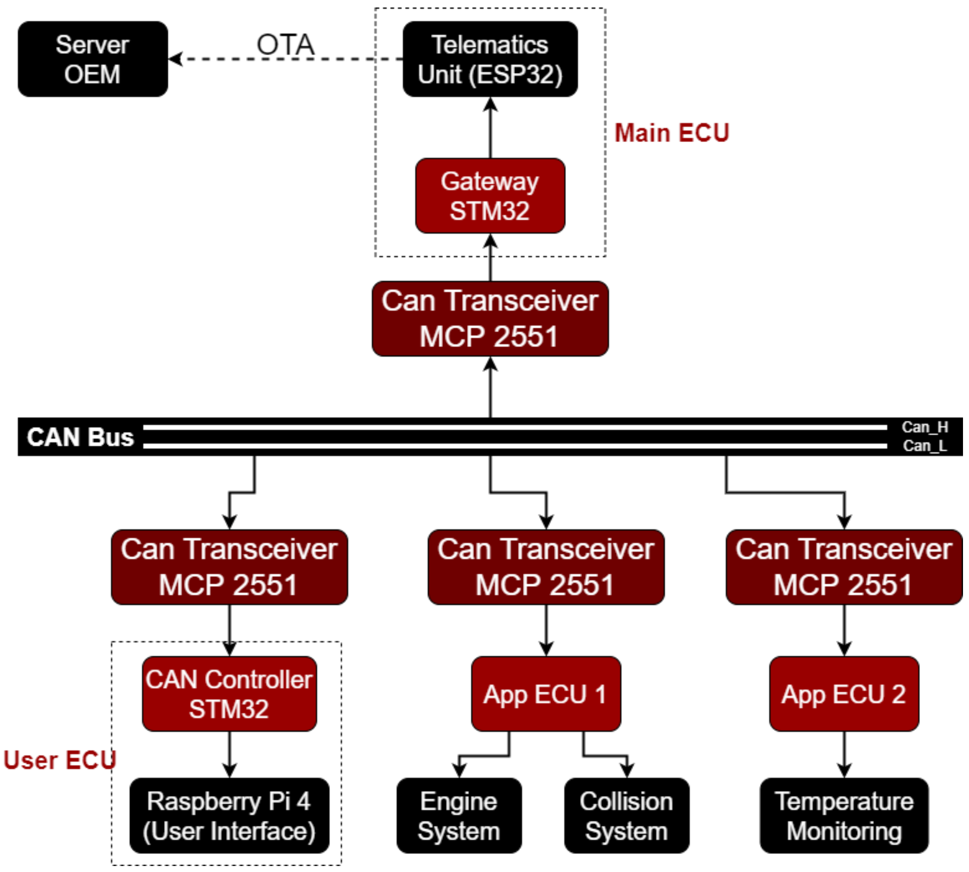


Figure 3.9 Detailed View of System.

3.2.4. Server

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

<u>Google Firebase advantages:</u>

- **Real-time Database**

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

- **Hosting**

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

- **Authentication**

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

- **Storage**

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

3.3. Software Design

3.3.1. Software Design Process

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

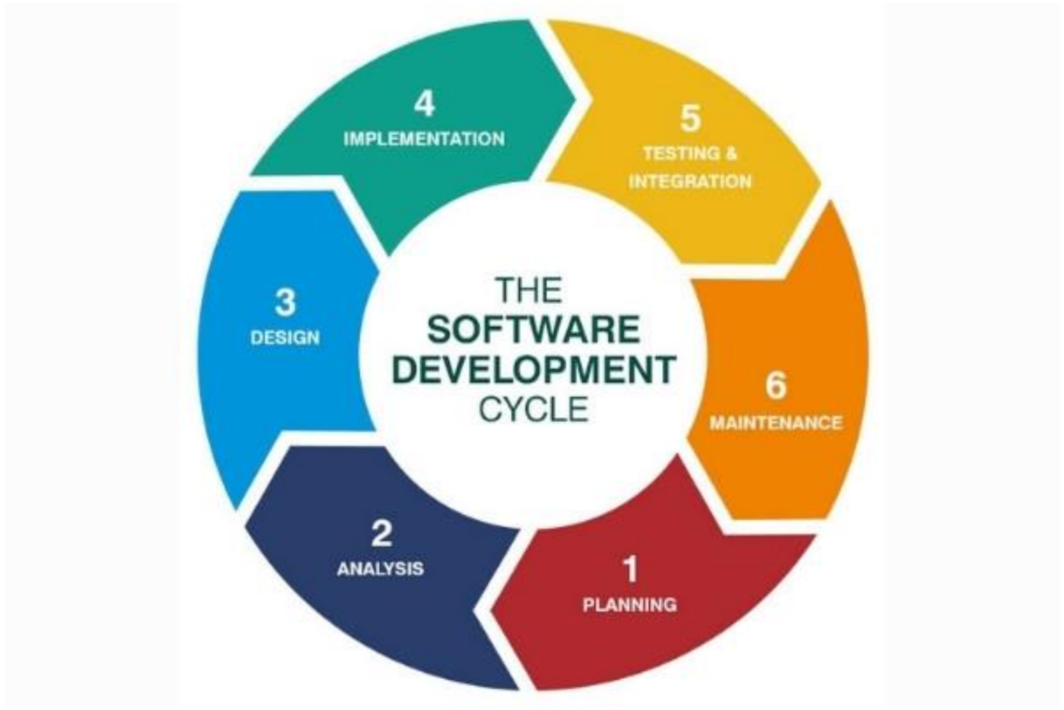


Figure 3.10 Software Development Cycle.

3.3.2. Design Considerations:

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

- **Compatibility** - The software can operate with other products that are designed for interoperability with another product. For example, a piece of software may be backward-compatible with an older version of itself.
- **Extensibility** - New capabilities can be added to the software without major changes to the underlying architecture.
- **Modularity** - the resulting software comprises well-defined, independent components which leads to better maintainability. The components could be then implemented and tested in isolation before being integrated to form a desired software system. This allows division of work in a software development project.
- **Fault-tolerance** - The software is resistant to and able to recover from component failure.

- **Maintainability** - A measure of how easily bug fixes or functional modifications can be accomplished. High maintainability can be the product of modularity and extensibility.
- **Reliability** (Software durability) - The software can perform a required function under stated conditions for a specified period of time.
- **Reusability** - The ability to use some or all the aspects of the preexisting software in other projects with little to no modification.
- **Robustness** - The software can operate under stress or tolerate unpredictable or invalid input. For example, it can be designed with resilience to low memory conditions.
- **Security** - The software can withstand and resist hostile acts and influences.
- **Usability** - The software user interface must be usable for its target user/audience. Default values for the parameters must be chosen so that they are a good choice for most of the users.[6]
- **Performance** - The software performs its tasks within a timeframe that is acceptable for the user and does not require too much memory.
- **Portability** - The software should be usable across several different conditions and environments.
- **Scalability** - The software adapts well to increasing data or added features or number of users.

3.3.3. Software Design Approaches

1- Top-Down Design Model:

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

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

Advantages:

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

Bottom-Up Design Model:

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

Advantages:

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

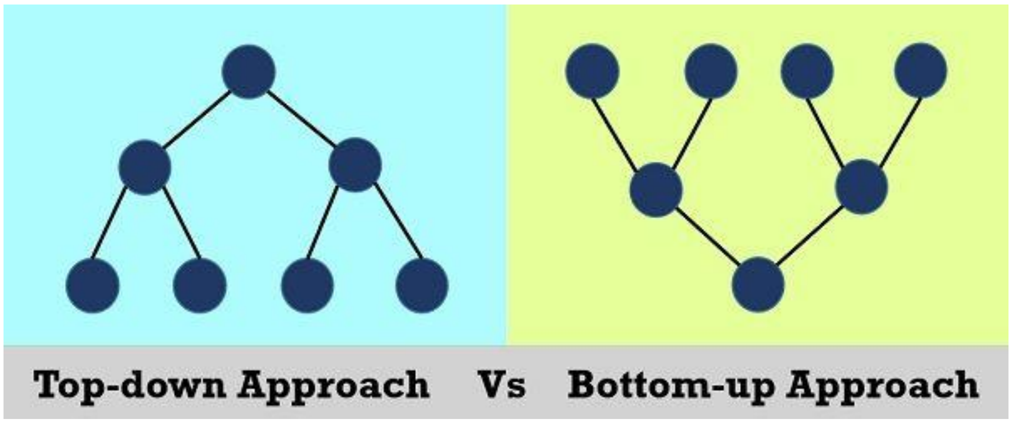


Figure 3.11 Top-down Approach vs Bottom-up Approach

3.3.4. Software Design Types

1- High-Level Design

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

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

2- Low-level design

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

S.NO.	Comparison Basis	HLD	LLD
1.	**Stands for**	It stands for High-Level Design.	It stands for Low-Level Design.
2.	**Definition**	It is the general system design, which means it signifies the overall system design.	It is like describing high-level design, which means it signifies the procedure of the component-level design.
3.	**Purpose**	The HLD states the concise functionality of each component.	LLD states the particular efficient logic of the component.

4.	**Also, known as**	HLD is also called a System or macro-level design.	LLD is also called details or micro-level design.
5.	**Developed by**	Solution Architect prepares the High-Level Design.	Designer and developer prepare the Low-Level Design.
6.	**Sequential order in the design phase**	It is developed first in sequential order, which implies that the HLD is created before the LLD.	It is developed after High-level design.
7.	**Target Audience**	It is used by management, program, and solution teams.	It is used by designers, operation teams, and implementers.
8.	**Conversion**	The HLD changes the client or business requirement into a high-level solution.	The LLD changes the high-level solution to a comprehensive solution.
9.	**Probable output**	The high-level design is necessary to understand the flow across several system objects.	A low-level design is required for creating the configurations and troubleshooting inputs.
10.	**Input Criteria**	The input measure in high-level design is SRS (Software Requirement Specification).	The input measure in low-level design is the reviewed HLD (High-Level Design).
11.	**Output Criteria**	The output measures in the HLD are functional design, database design, and review record.	The output bases in the low-level design are the unit test plan and program specification.

3.4. Software design of dashboard

3.4.1. Layered architecture

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

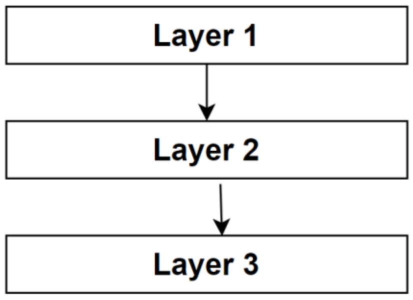


Figure 3.12 Layered Architecture.

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

Characteristics

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

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

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

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

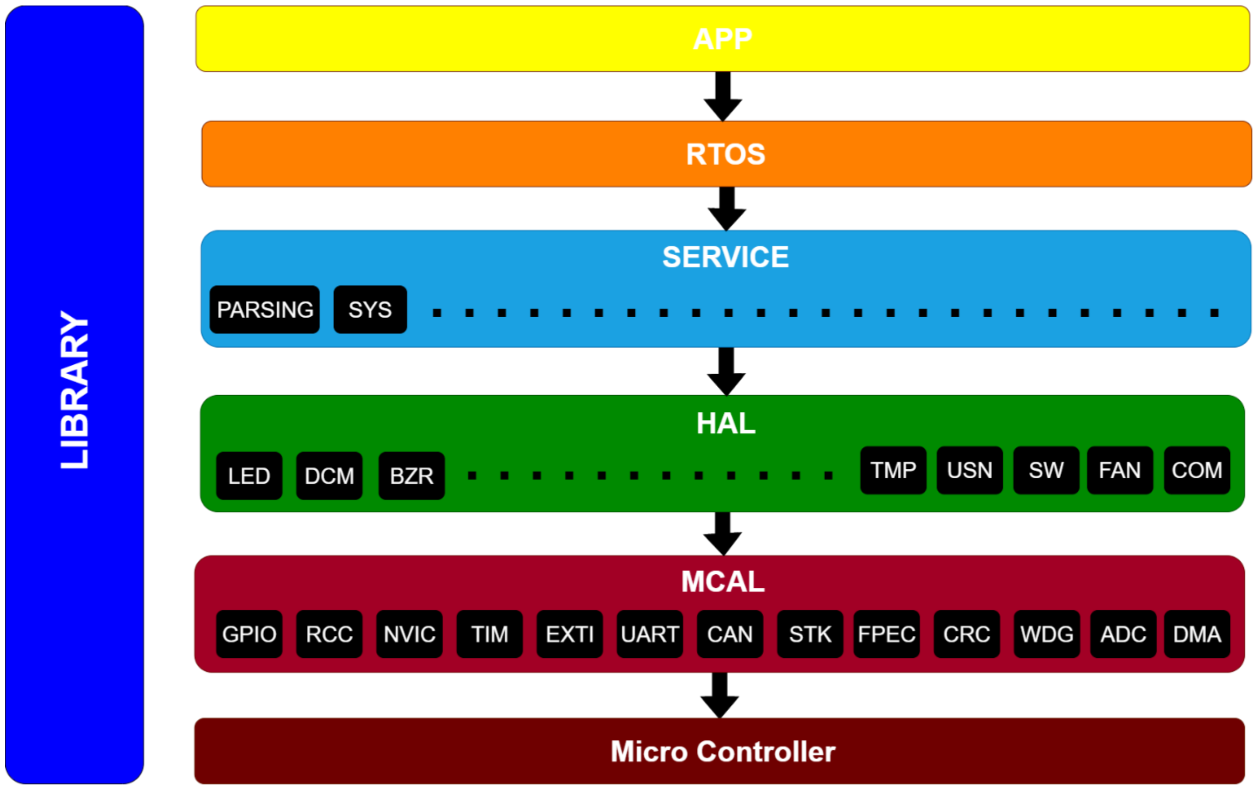


Figure 3.13 Layered Architecture

Chapter 4: Implementation

4.1. Server Connection

4.1.1. Firebase

Figure 4.1 Firebase BaaS.

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

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

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

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

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

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

4.1.2. GUI

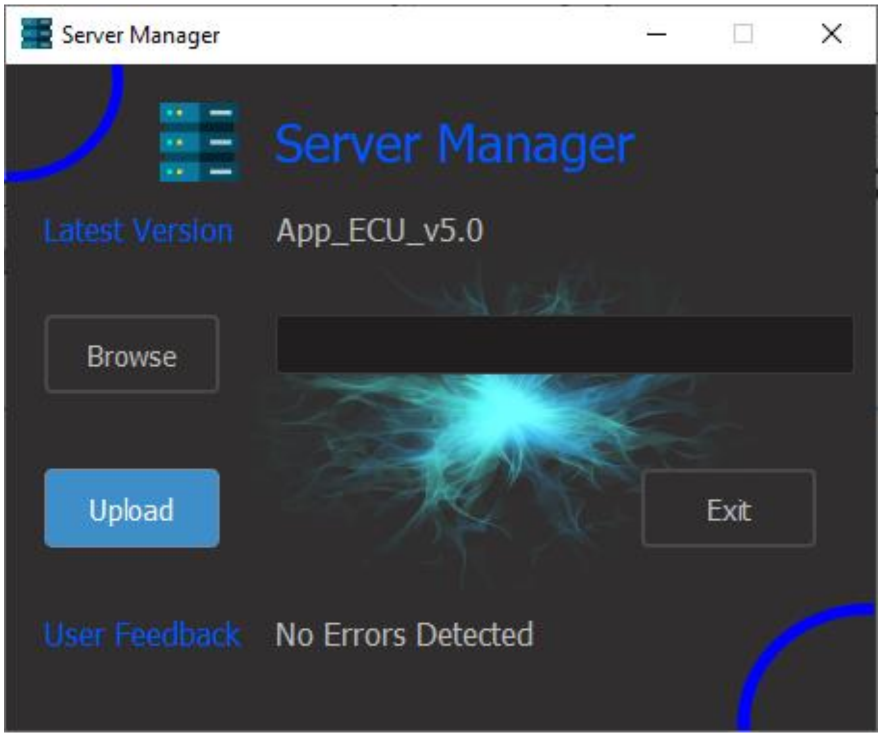


Figure 4.2 Server GUI.

Used to abstract the backend implementations with firebase and create a standard interface to deal with the update, GUI is implemented using Python via a library called PyQt5.

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and binding, makes it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python is a simple easy-to-learn syntax that emphasizes readability and therefore, reduces the cost of program maintenance.

We fetch the current factory firmware version and display it next to the Latest Version label, we can browse a new update file over the PC and upload it to our firebase storage.

If there is any kind of errors while uploading, it is displayed next to the User Feedback label.

4.2. Advanced Encryption Standard (AES)

4.2.1. Introduction

The more popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES).
Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S National Institute of Standards and Technology (NIST) in 2001. AES is widely used today as it is much stronger than DES and triple DES despite being harder to implement.

The features of AES are as follows:

- Symmetric key symmetric block cipher.
- 128-bit data, 128/192/256-bit keys.
- Stronger and faster than Triple-DES.
- Provide full specification and design details.
- Software implementable in C and Java.

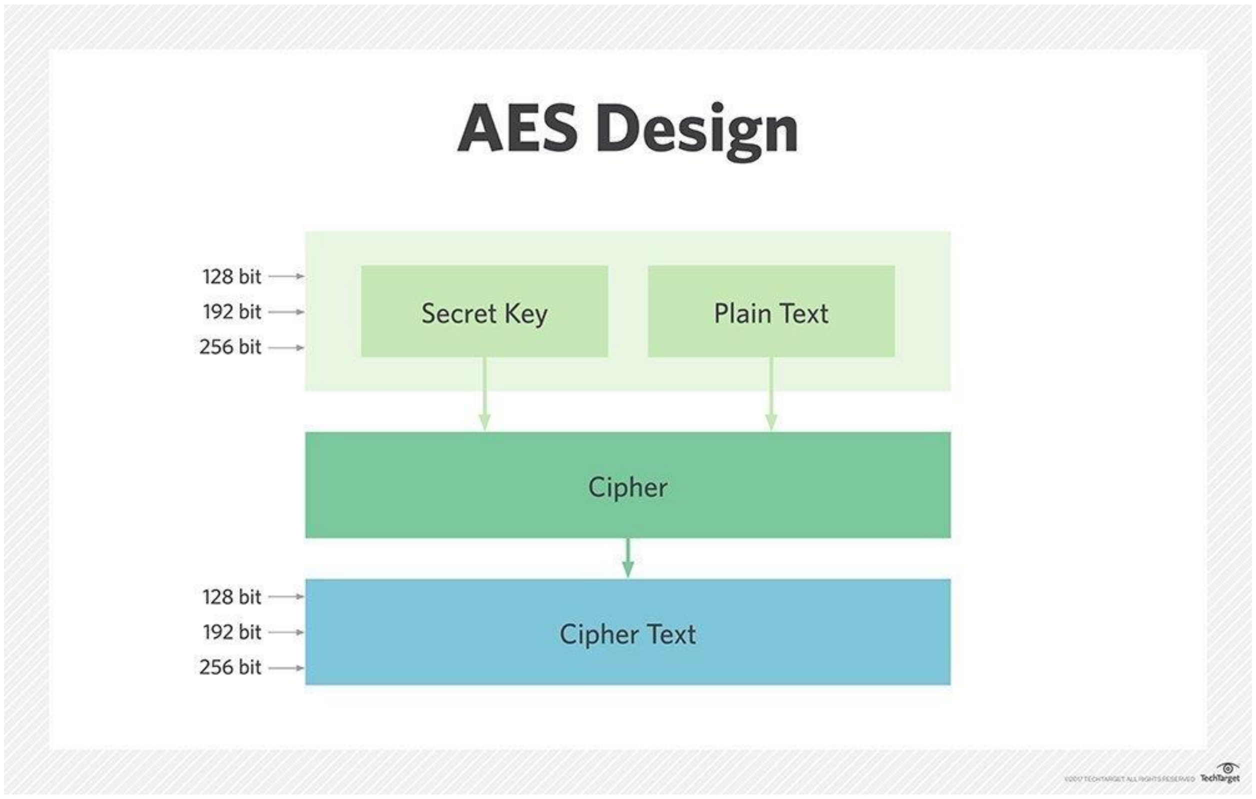


Figure 4.3 AES Design.

4.2.2. Encryption Algorithms

In computing, encryption is the method by which plain text or any other type of data is converted from a readable form to an encoded version that can only be decoded by another entity if they have access to a decryption key. Encryption is one of the most important methods for providing data security, especially for end-to-end protection of data transmitted across networks.

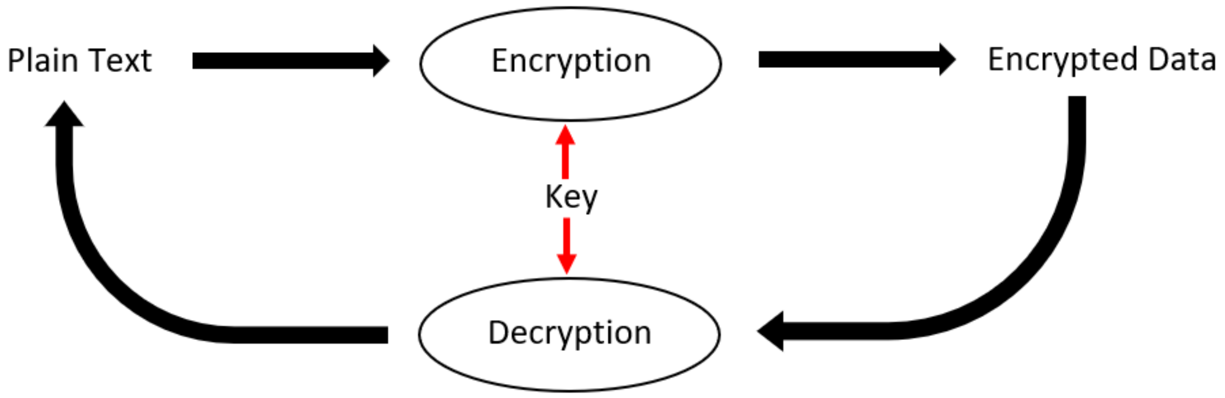


Figure 4.4 Encryption Life Cycle.

Plain text is encrypted using an encryption algorithm and an encryption key. This generates an unreadable text which is called as ciphertext (encrypted data). Decryption is the inverse of encryption; original form of data can only be viewed by decrypting encrypted data with the correct key.

There are two main types of data encryption methods:

- Symmetric Encryption (private-key encryption).
- Asymmetric Encryption (public-key encryption).

Cryptographically strength is similar in both of these methods but asymmetric encryption requires heavier mathematics and more computational power compared to symmetric encryption. So, asymmetric encryption is less efficient than symmetric encryption.

4.2.3. AES Encryption Algorithm

For our algorithm we use the Symmetric Encryption called AES, which means there is a single key, use the same key for both encryption and decryption and must share the key with entity intends to communicate with.

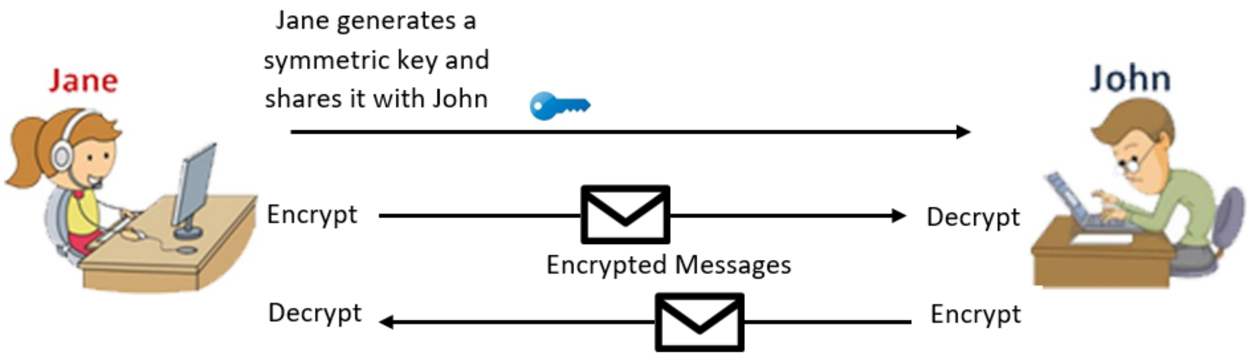


Figure 4.5 AES Encryption Example.

Imagine *Jane* has an important message that wants to send to *John* (over a network), So *Jane* generates a symmetric key (K1) and shares it with *John* prior to the communication. Both of them will come to an agreement that they will use K1 key for both encryption and decryption for the messages they pass between them. So, *Jane* encrypts the message and sends it to *John* over the network. *John* will decrypt the message on the receiving end and will get the original message. Since only *Jane* and *John* have the key, others cannot read the message they share even though they have access to it.

4.2.4. Encryption Process

Here, we restrict to the description of a typical round of AES encryption. Each round comprises four sub-processes.

The first-round process is depicted below:

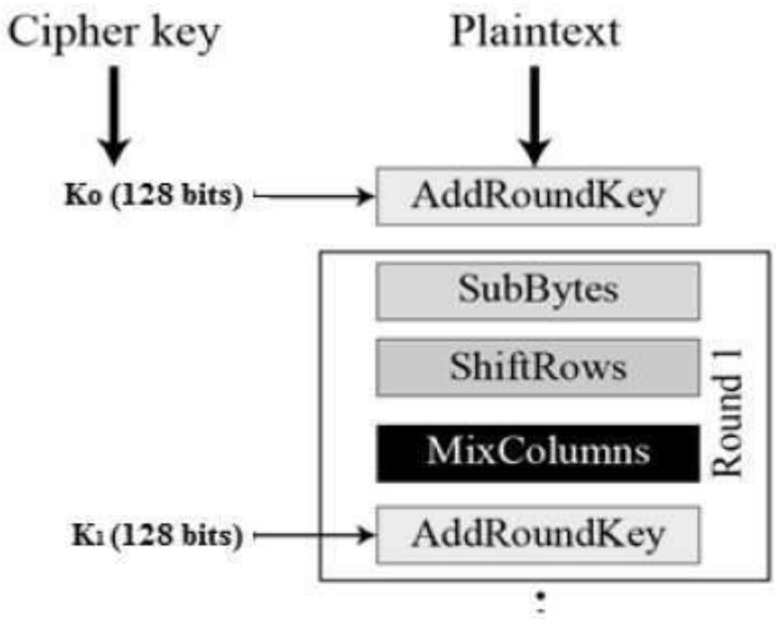


Figure 4.6 AES Lifecycle.

1- Byte Substitution (Sub-Bytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given in the design. The result is in a matrix of four rows and four columns.

2- Shift rows

Each of the four rows of the matrix is shifted to the left. Any entries that 'fall off' are re-inserted on the right side of the row. Shift is carried out as follows:

a) The First row is not shifted.
b) The Second row is shifted one (byte) position to the left.
c) The Third row is shifted two positions to the left.
d) The Fourth row is shifted three positions to the left.
e) The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

3- Mix Columns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

4- Add round key

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round, then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes, and we begin another similar round.

4.2.5. Decryption Process

The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order:

- Add round key
- Mix columns
- Shift rows
- Byte substitution

Since sub-processes in each round are in a reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms need to be separately implemented, although they are very closely related.

4.3. CAN Network

4.3.1. Introduction

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

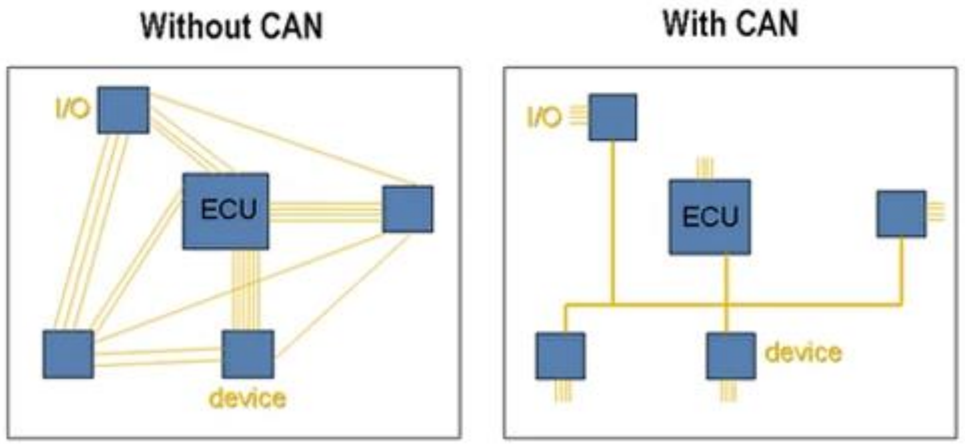


Figure 4.7 CAN networks significantly reduce wiring

In 1993, the CAN emerged as the standard in-vehicle network due to its energetic features for safety and it became the international standard known as ISO-11898.

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

4.3.2. What is CAN?

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

4.3.3. Features of CAN protocol

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

4.3.4. Can Network Message Format

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

1. **Standard frame:**

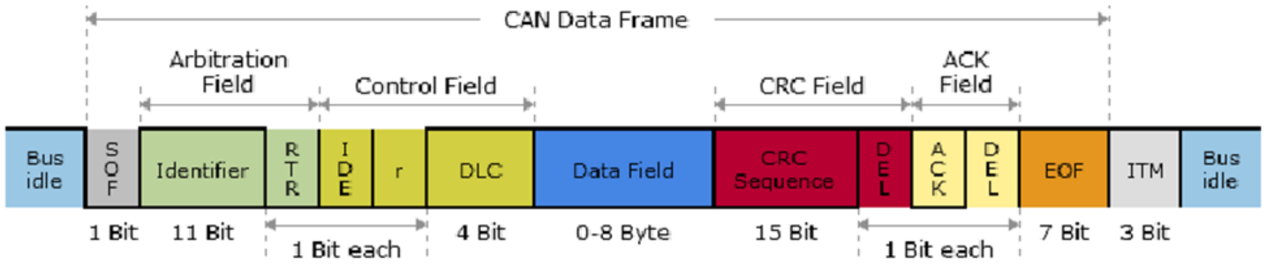


Figure 4.8 The standard CAN frame format

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

- **Arbitration ID** – identifies the message and indicates the message's priority. Frames come in two formats -- standard, which uses an 11-bit arbitration ID, and extended, which uses a 29-bit arbitration ID.

- **RTR (remote transmission request) bit** – serves to differentiate a remote frame from a data frame. A dominant (logic 0) RTR bit indicates a data frame. A recessive (logic 1) RTR bit indicates a remote frame.
- **IDE (identifier extension) bit** – allows differentiation between standard and extended frames.
- **R0** – Reversed bit. Not used currently and kept for future use.
- **DLC (data length code)** – indicates the number of bytes the data field contains.
- **Data Field** – contains 0 to 8 bytes of data.
- **CRC (cyclic redundancy check)** – contains a 15-bit cyclic redundancy check code and a recessive delimiter bit. The CRC field is used for error detection.

- **ACK (Acknowledgement) slot** – It compromises of the ACK slot and the ACK delimiter. When the data is received correctly the recessive bit in the ACK slot is overwritten as a dominant bit by the receiver.
- **EOF (End of Frame) –** the 7-bit field marks the end of a CAN frame (message) and disables.

2. Extended frame:

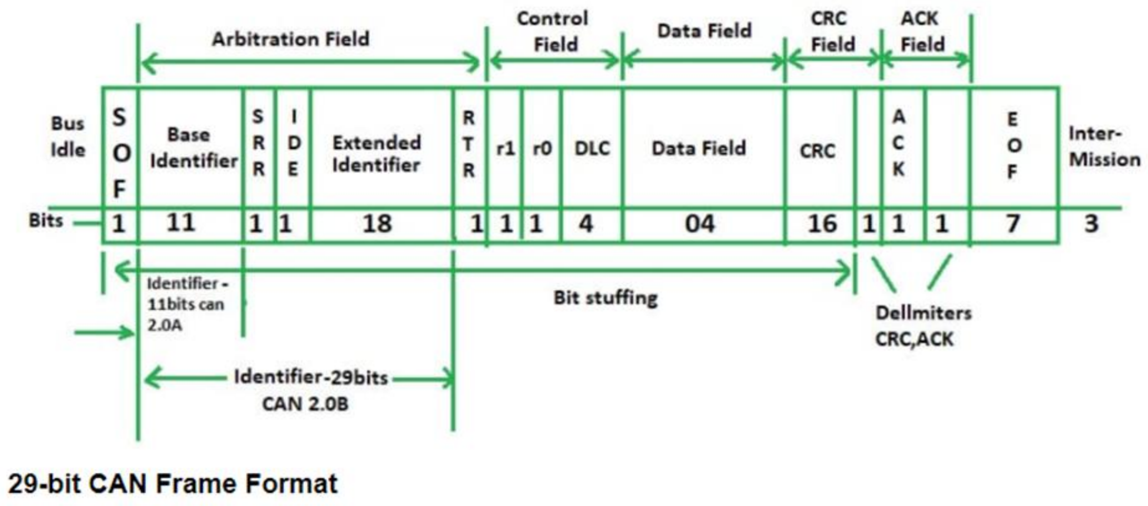


Figure 4.9 The c CAN frame format.

It is the same as an 11-bit identifier with some added bits.
SRR (Substitute Reverse Request) – The SRR bit is always transmitted as a recessive bit to ensure that, in the case of arbitration between a Standard Data Frame and an Extended Data Frame, the Standard Data Frame will always have priority if both messages have the same base (11 bit) identifier.
R1– It is another bit not used currently and kept for future use.

4.3.5. Message frame

Four different frames can be used on the bus.

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

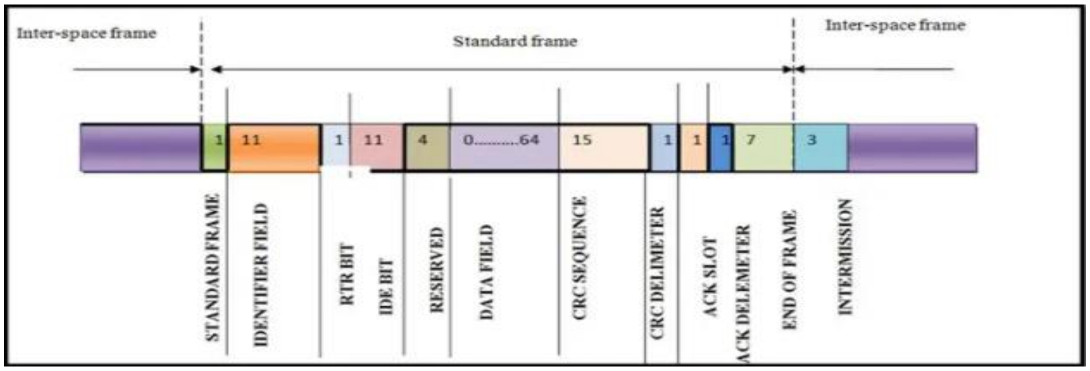


Figure 4.10 CAN Data frames.

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

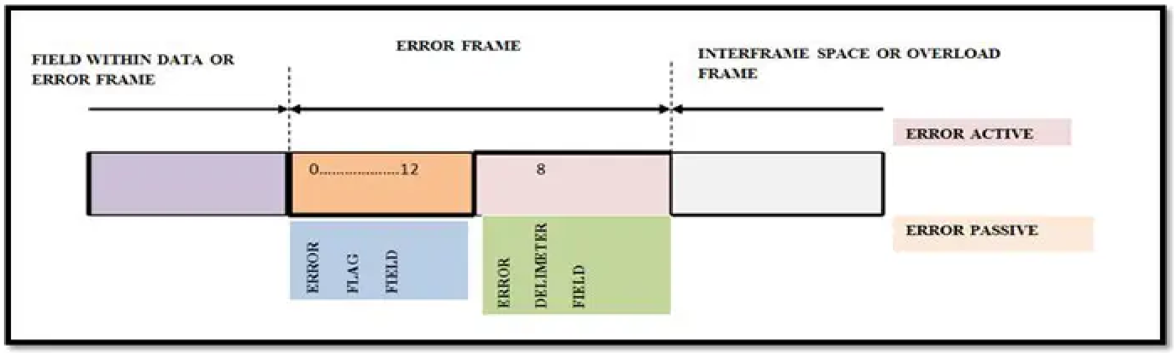


Figure 4.11 CAN Error Frames.

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

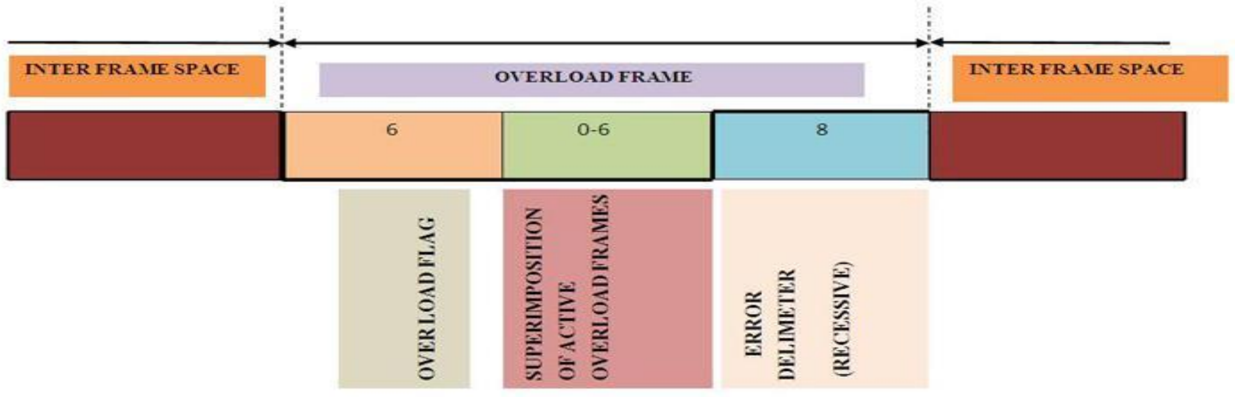

INTER FRAME SPACE
OVERLOAD FRAME
INTER FRAME SPACE
6
0-6
8
OVERLOAD FLAG
SUPERIMPOSITION OF ACTIVE OVERLOAD FRAMES
ERROR DELIMETER (RECESSIVE)

Figure 4.12 CAN Overload frame.

4.3.6. CAN Protocol Working Principle

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

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

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

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

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

4.3.7. Operation of the CAN Network

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

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

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

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

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

4.3.8. Interface between Nodes and Communication Network

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

a) CAN Filtering:

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

b) CAN FIFO Buffer:

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

4.3.9. CAN in Action

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

a) Transmitting a CAN message:

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

b) Receiving a CAN message:

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

4.4. USART protocol

4.4.1. Introduction

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

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

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

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

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

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

4.4.2. Interface

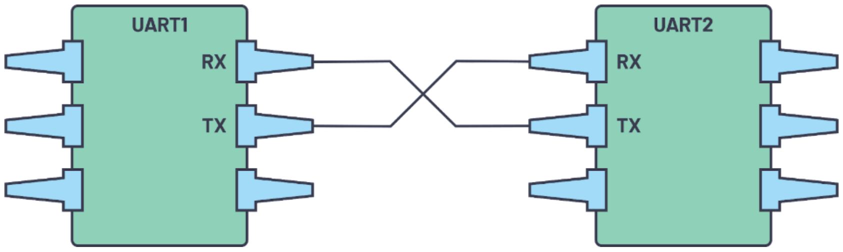


Figure 4.13 Two USARTs directly communicate with each other.

The two signals of each USART device are named:

- Transmitter (Tx)
- Receiver (Rx)

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

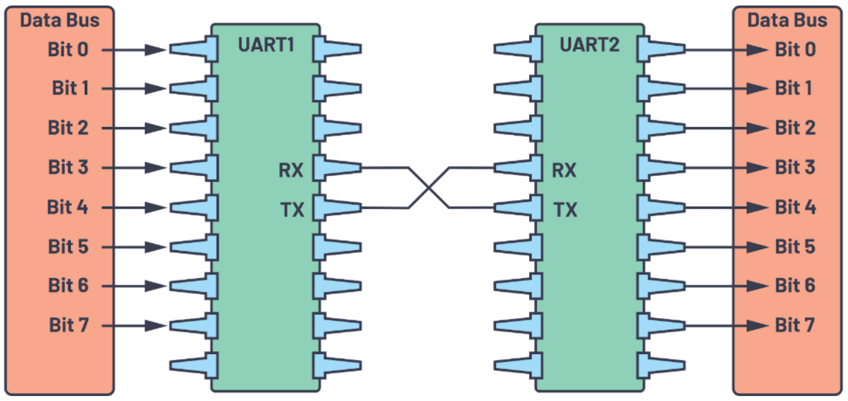


Figure 4.14 USART with data bus.

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

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

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

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

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

4.4.3. Data Transmission

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

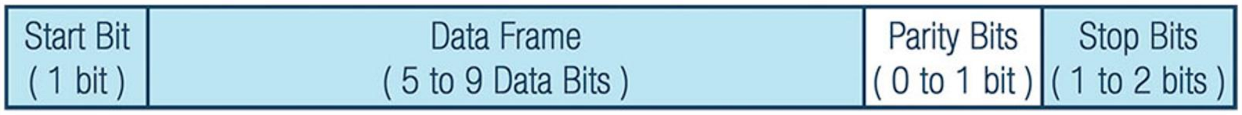


Figure 4.15 USART packet.

- **Start Bit**

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

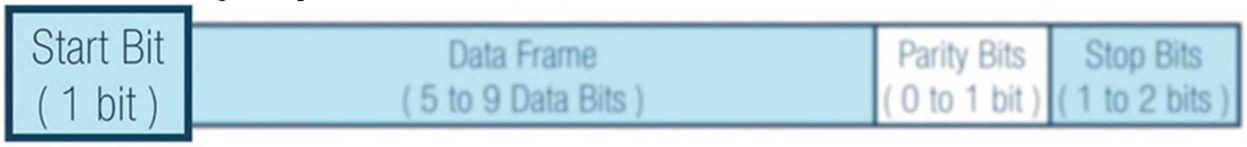


Figure 4.16 Start bit.

- **Data Frame**

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

Start Bit
(1 bit)
Data Frame
(5 to 9 Data Bits)
arity Bits
to 1 bit)
Stop Bits
(1 to 2 bits)

Figure 4.17 Data Frame.

- **Parity**

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

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

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

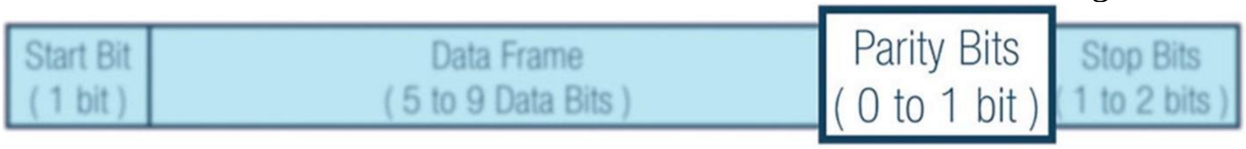


Figure 4.18 Parity bits.

- **Stop Bits**

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

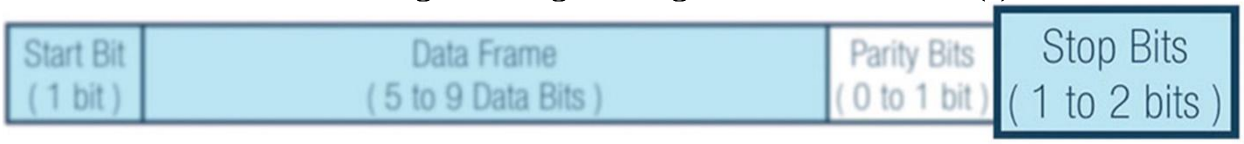


Figure 4.19 Stop bits.

4.4.4. USART Transmission

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

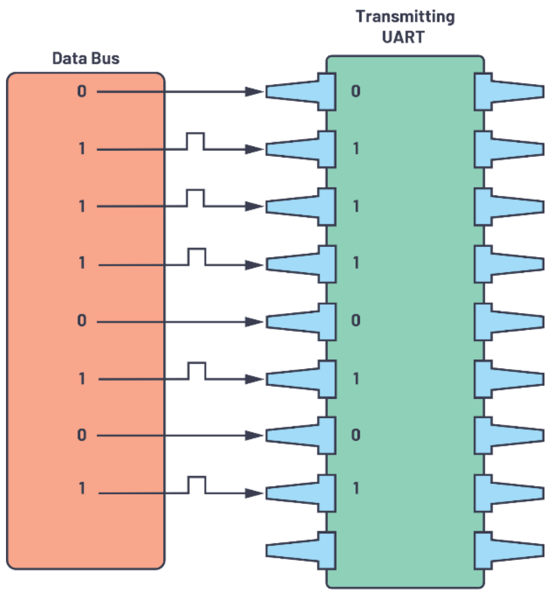


Figure 4.20 Data bus to the transmitting USART.

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

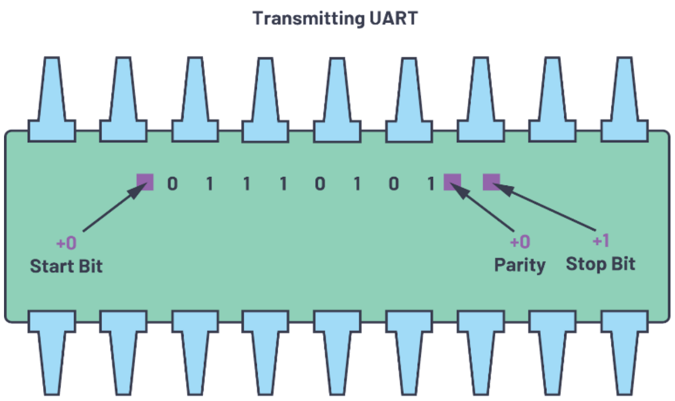


Figure 4.21 The USART data frame at the Rx side.

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

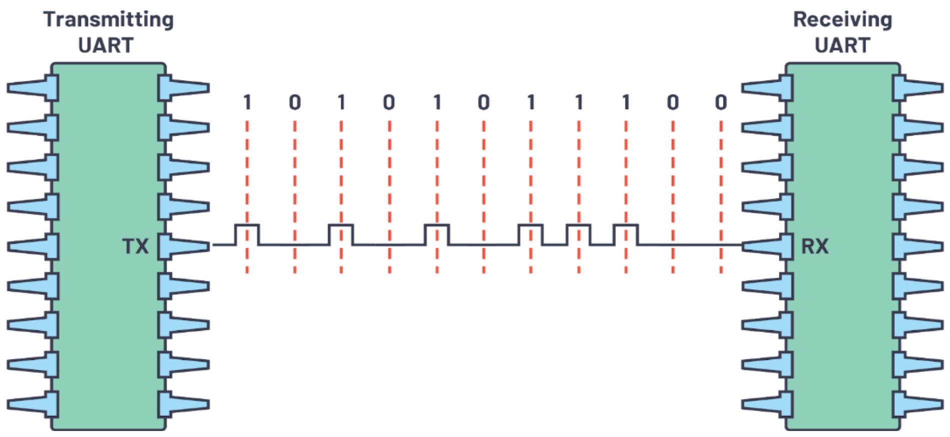


Figure 4.22 USART transmission.

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

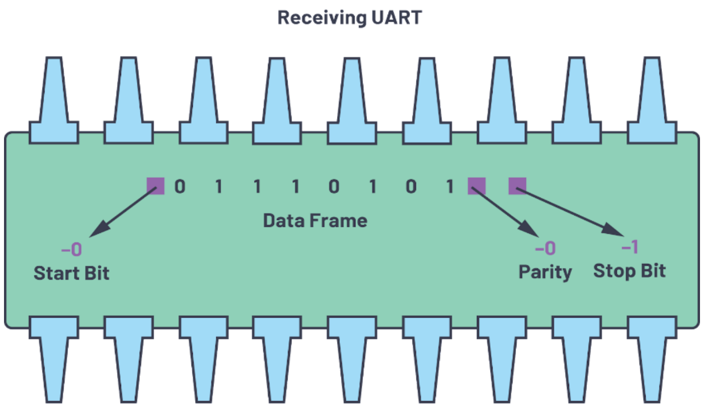


Figure 4.23 The USART data frame at the Rx side.

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

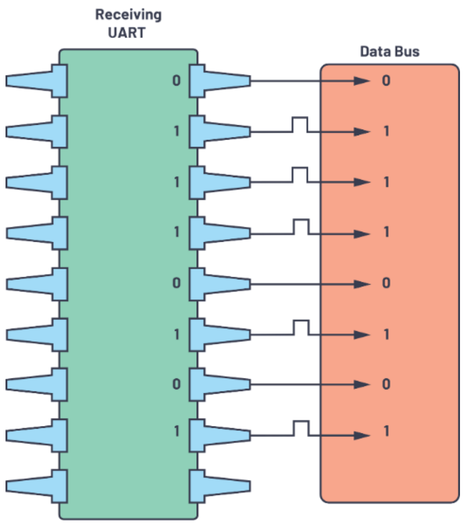


Figure 4.24 Receiving USART to data bus.

4.5. Implemented Communication Protocol

4.5.1. Parallel Communication Protocols

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

The basic difference between a parallel and a serial communication channel is the number of electrical conductors used at the physical layer to convey bits. Parallel communication implies more than one such conductor. For example, an 8-bit parallel channel will convey eight bits (or a byte) simultaneously, whereas a serial channel would convey those same bits sequentially, one at a time. If both channels operated at the same clock speed, the parallel channel would be eight times faster. A parallel channel may have additional conductors for other signals, such as a clock signal to pace the flow of data, a signal to control the direction of data flow, and handshaking signals.

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

4.5.2. Asynchronous Octal Communication (AOC)

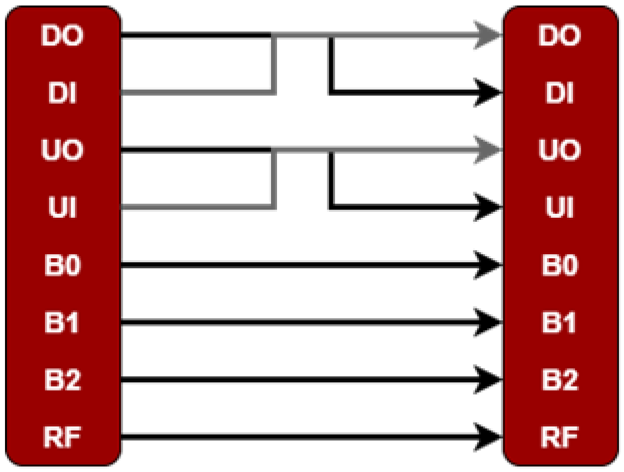


Figure 4.25 AOC Diagram.

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

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

The are 4 states four states for communication:

1. Diagnostics Request from Raspberry Pi to STM32:

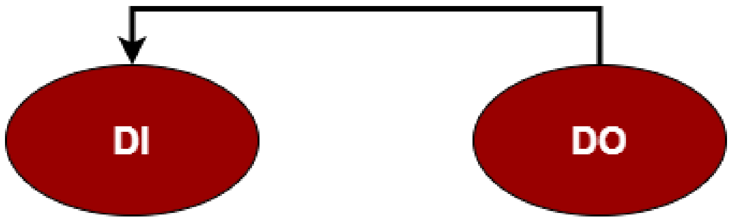


Figure 4.26 Diagnostics Request.

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

2. Diagnostics Data from STM32 to Raspberry Pi:

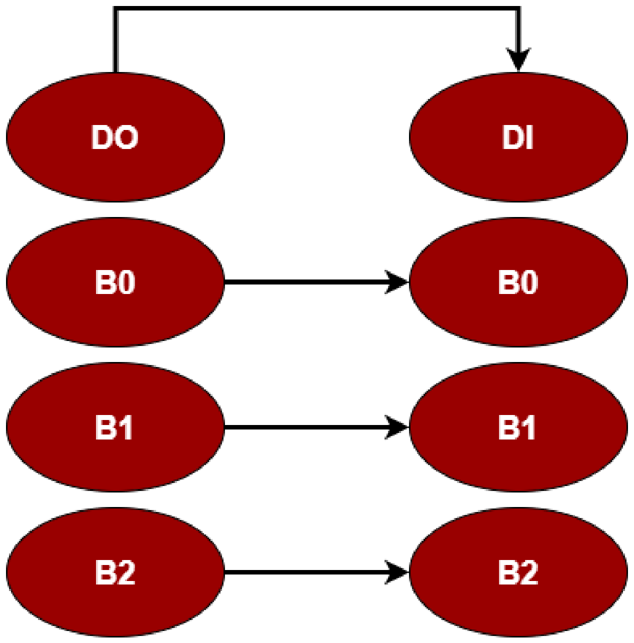


Figure 4.27 Diagnostics Data.

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

3. Update Notification and Progress from STM32 to Raspberry Pi:

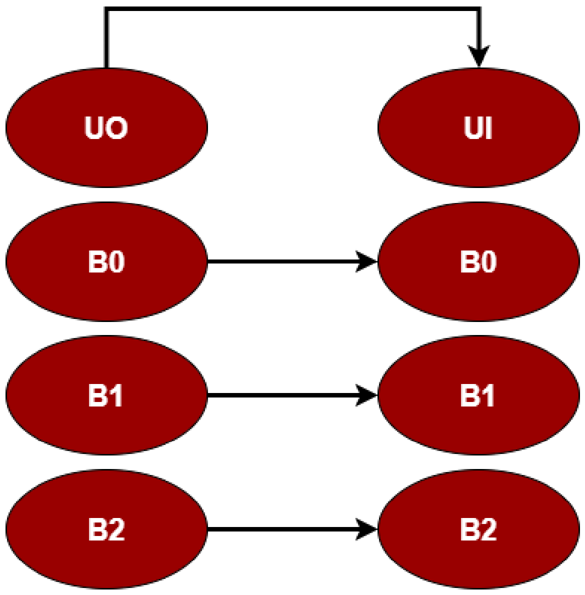


Figure 4.28 Update Notification and Data.

STM32 sends to Raspberry Pi on the data lines:

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

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

4. Update Response from Raspberry Pi to STM32:

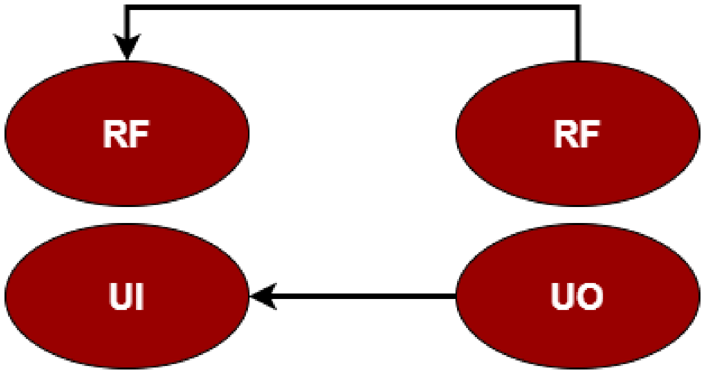


Figure 4.29 Update Response.

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

4.6. Bootloader

4.6.1. Introduction

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

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

SWD is an ARM-specific protocol designed specifically for micro debugging

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

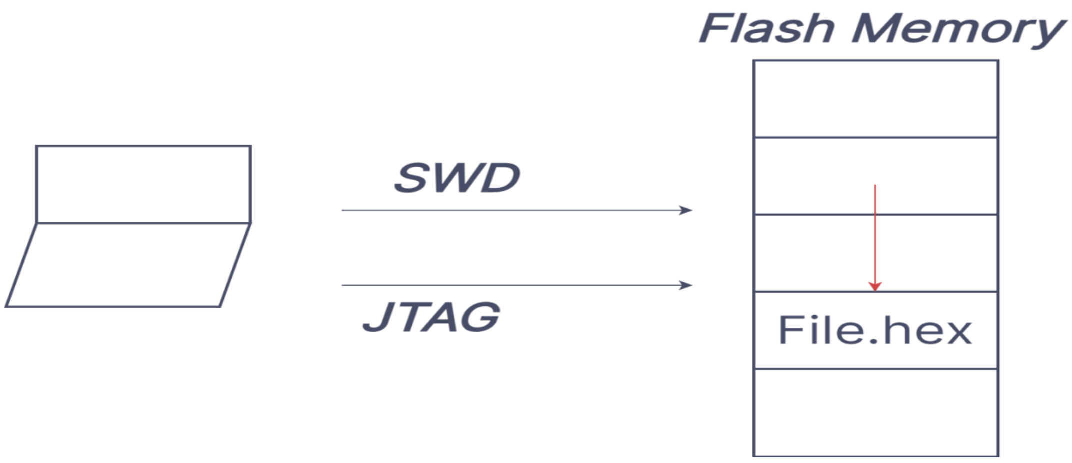


Figure 4.30 SWD and JTAG protocol.

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

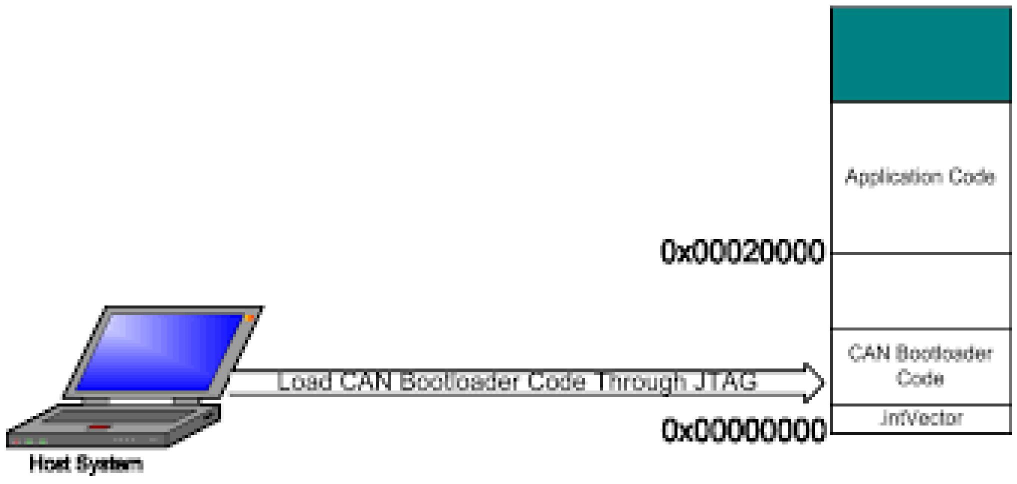


Figure 4.31 Load CAN Bootloader code through JTAG

A bootloader is a small OS or Application, it is designed to download the firmware in MCU’s internal or external memory.
It gives the possibility to upgrade or change the application in the MCU.

4.6.2. Memory Architecture

Memory is split into 4 partitions:

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

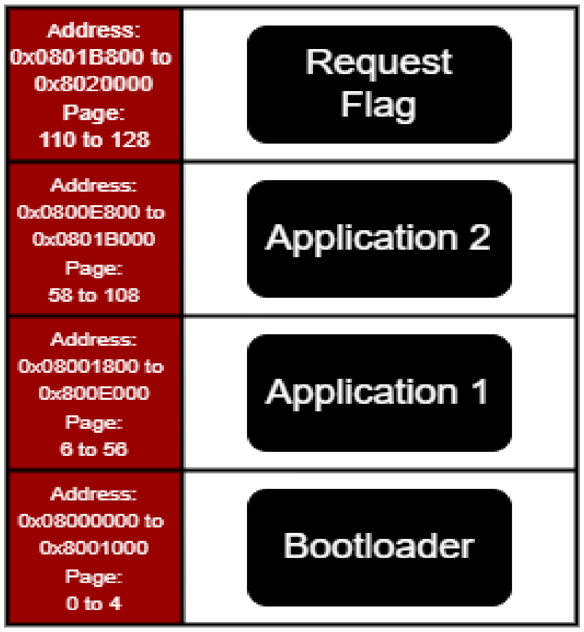

Address: 0x0801B800 to 0x8020000 Page: 110 to 128
Request Flag
Address: 0x0800E800 to 0x0801B000 Page: 58 to 108
Application 2
Address: 0x08001800 to 0x800E000 Page: 6 to 56
Application 1
Address: 0x08000000 to 0x8001000 Page: 0 to 4
Bootloader

Figure 4.32 Memory Hierarchy.

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

4.6.3. Bootloader Design

Bootloader design has 4 blocks:

a) Preload:

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

ii) **The benefits of the preload:**

(1) It prevents the bootloader discover. (Is there an update or Not?)

(2) Manage between Two application and bootloader.

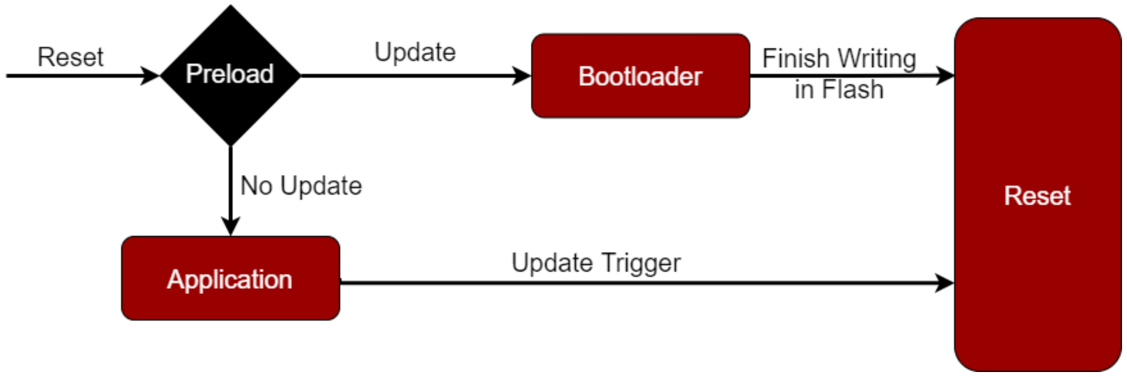


Figure 4.33 Bootloader Flow Chart.

b) Application:

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

c) Bootloader:

Bootloader implementation:

(1) Erase the space of application1.

(2) Receive one record in iteration.

(3) Make checksum to verify the data is received correctly.

(4) Then write the record in flash.

(5) With each iteration repeat this step from receive record.

d) Reset:

There are two state to go to reset:

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

4.7. Real Time Operating System (RTOS)

4.7.1. Introduction

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

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

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

Characteristics of an RTOS:

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

4.7.2. Free RTOS Overview

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

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

Characteristics of free RTOS:

1. **Trusted kernel:** With proven robustness, tiny footprint, and wide device support, the Free RTOS kernel is trusted by world-leading companies as the de facto standard for microcontrollers and small microprocessors.
2. **Accelerate time to market:** With detailed pre-configured demos and Internet of Things (IoT) reference integrations, there is no need to determine how to setup a project. Quickly download, compile, and get to market faster.
3. **Broad ecosystem support:** Our partner ecosystem provides a breadth of options including community contributions, professional support, as well as integrated IDE and productivity tools.
4. **Predictability of long-term support:** Free RTOS offers feature stability with long term support (LTS) releases. Free RTOS LTS libraries come with security updates and critical bug fixes for two years. Maintained by AWS for the benefit of the Free RTOS community.

4.7.3. Free RTOS Implementation

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

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

4.7.4. RTOS Implementation

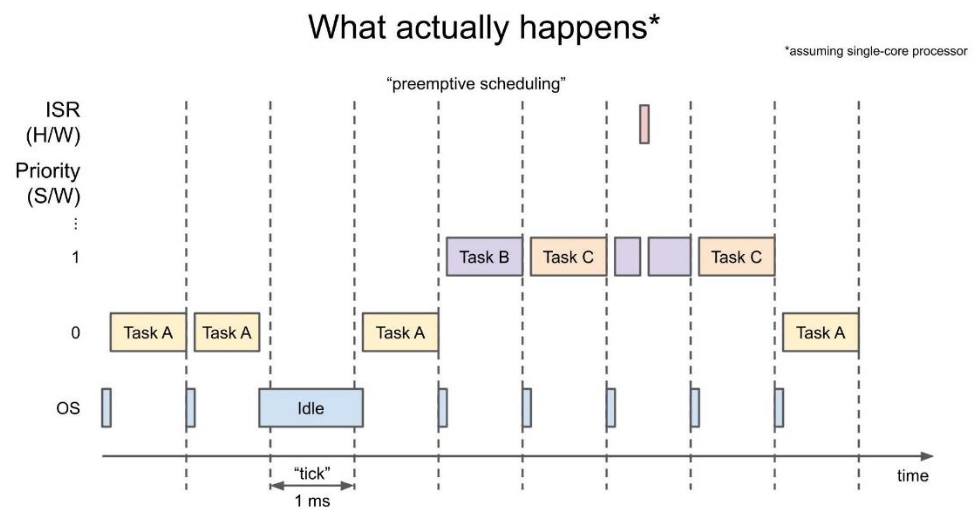


Figure 4.34 RTOS Time Slicing.

We divided the RTOS functions to:

a. Scheduler Related Functions.

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

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

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

b. Tasks Related Functions.

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

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

4.8. Diagnostics Handling

4.8.1. Introduction

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

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

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

4.8.2. Diagnostics in our system

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

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

Mode One

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

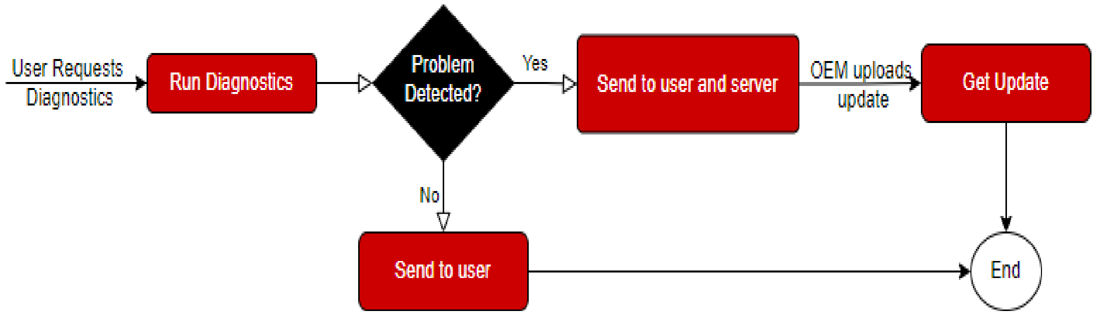


Figure 4.29 Remote Diagnostics - Mode 1

Mode Two

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

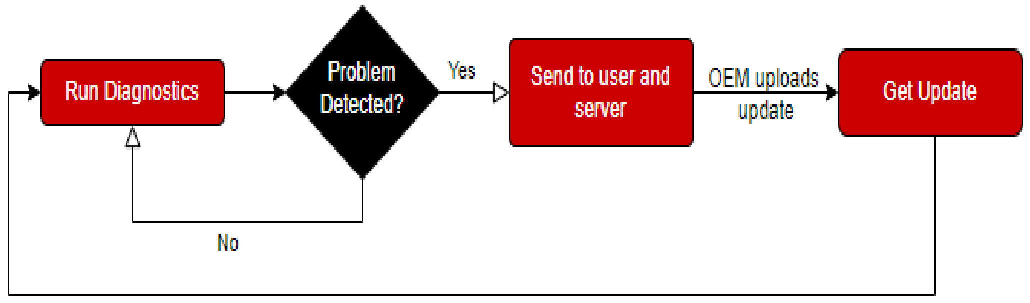


Figure 4.30 Remote Diagnostics - Mode 2

When an issue is discovered, in either mode, the user is alerted, and the issue is also sent to the OEM server. Some car issues can be resolved, temporarily or permanently, by some tweaks in the ECU firmware, in which case the OEM uploads a firmware version that solves this problem on its server for the car experiencing the issue, and a notification of the update is sent to the user who can then choose whether to download the update.

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

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

4.8.3. Diagnostics for different application ECUs

Application ECU 1

- Engine System

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

- Collision System

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

Application ECU 2

- Temperature Monitoring

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

4.8.4. Future Developments in Vehicle Diagnostics

Collecting dynamically shared data from a fleet of vehicles helps OEMs gain new insights as needed. Additional value can be created by applying statistics methods, machine learning or big data analytics on such collected data. For example, early feedback loops can be implemented which makes predictive maintenance possible by collecting data indicating an imminent damage and new services can be offered based on vehicle data.

4.9. User Interface

4.9.1. PyQt5 Introduction

PyQt is a GUI widget toolkit. It is a Python interface for Qt, one of the most powerful, and popular cross-platform GUI libraries. PyQt was developed by Riverbank Computing Ltd. Its API is a set of modules containing a large number of classes and functions. QtCore module contains the non-GUI functionality, QtGui module contains all the graphical controls and QtMultimedia module contains the low-level multimedia programming.

What made us choose PyQt5 is:

- Code flexibility
- Various UI components
- Various learning resources

As the other option was Tkinter, which doesn’t include any advanced widgets and features, it doesn’t have a tool that is similar to Qt Designer, and it doesn’t have the modern look for an application.

4.9.2. PyQt5 Usage

We have 10 different applications each for a specific task or function.

Splash Screen: is used as a loading screen while the whole application is initialized.

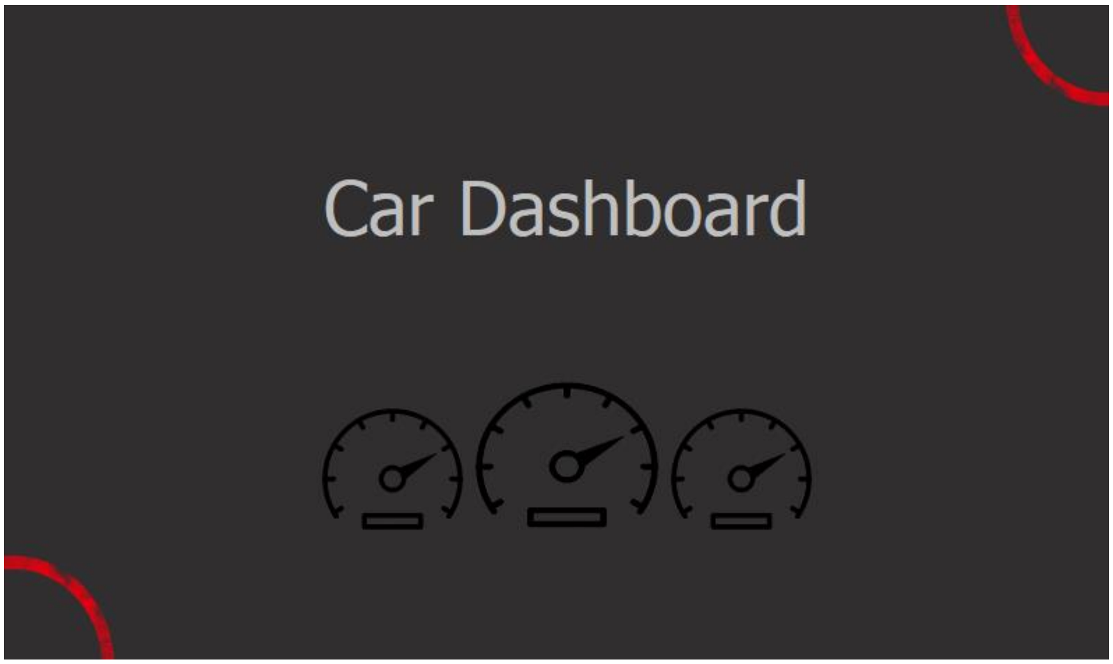


Figure 4.35 GUI Splash Screen.

Main Window: is used for displaying the whole application along with dynamic Time and Date.

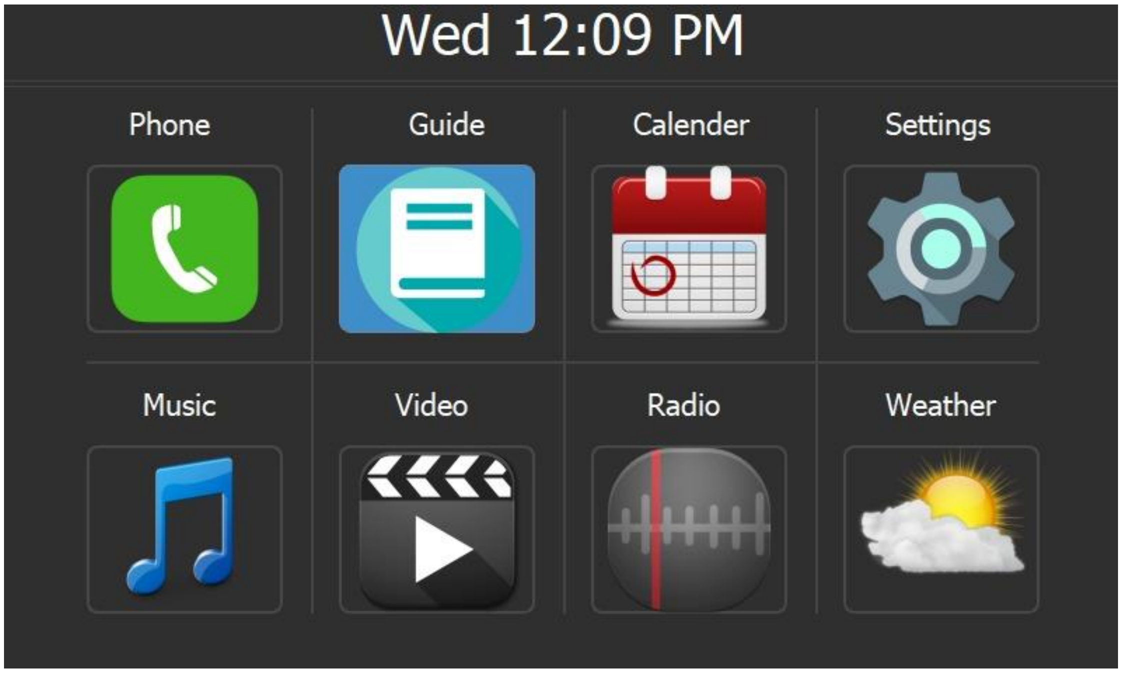


Figure 4.36 GUI Main Window.

Phone: is used for calling a phone number, viewing recently dialed numbers and the user's contacts.

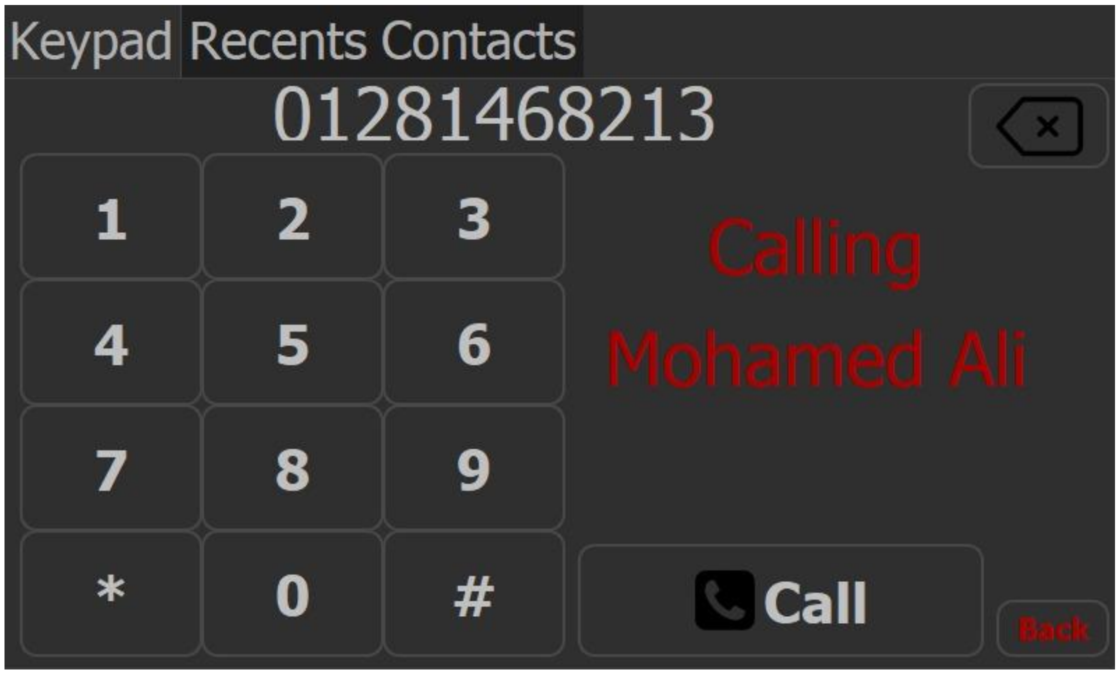
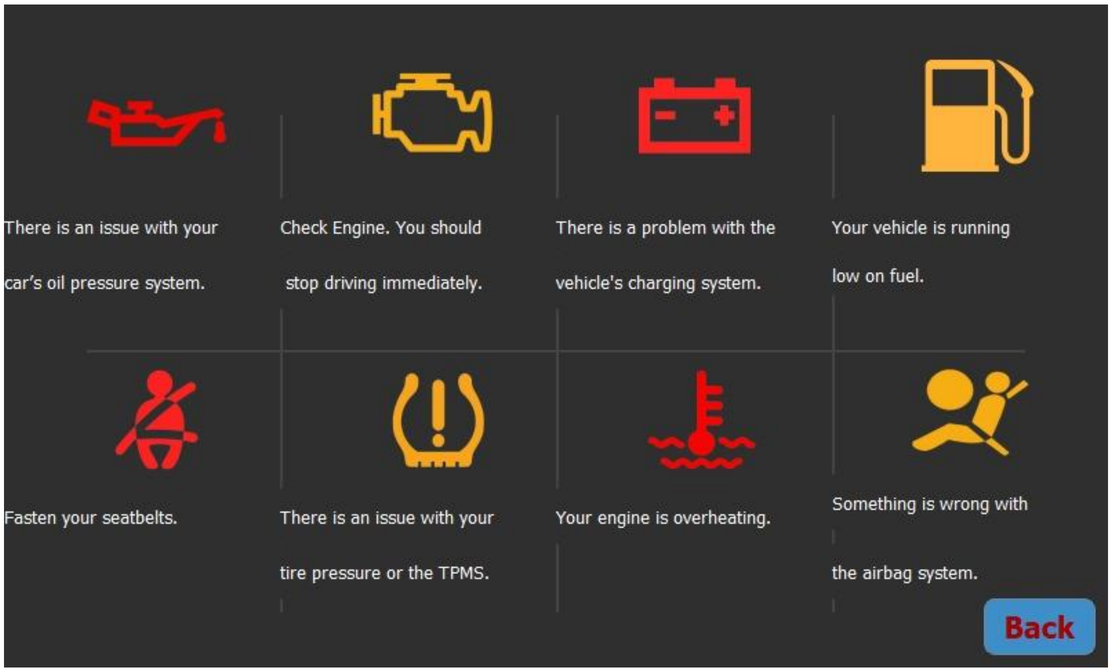


Figure 4.37 GUI Phone Window.

Guide: is used for displaying the car dashboard warnings and briefly describing the problem



There is an issue with your car's oil pressure system.

Check Engine. You should stop driving immediately.

There is a problem with the vehicle's charging system.

Your vehicle is running low on fuel.

Fasten your seatbelts.

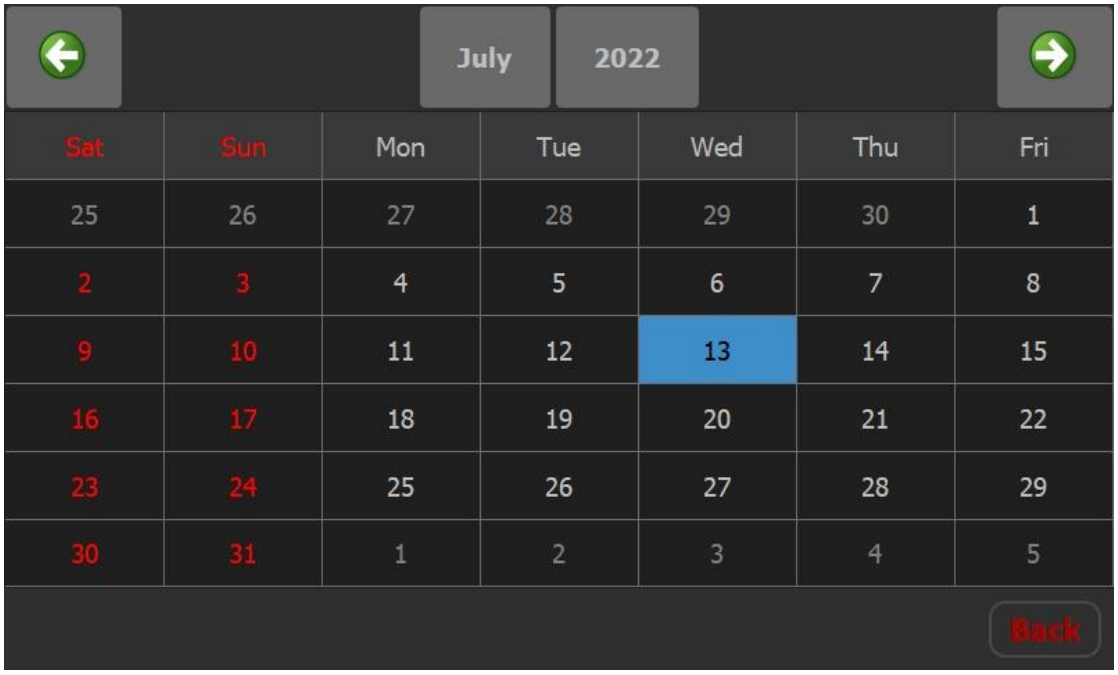
There is an issue with your tire pressure or the TPMS.

Your engine is overheating.

Something is wrong with the airbag system.

Figure 4.38 GUI Guide Window.

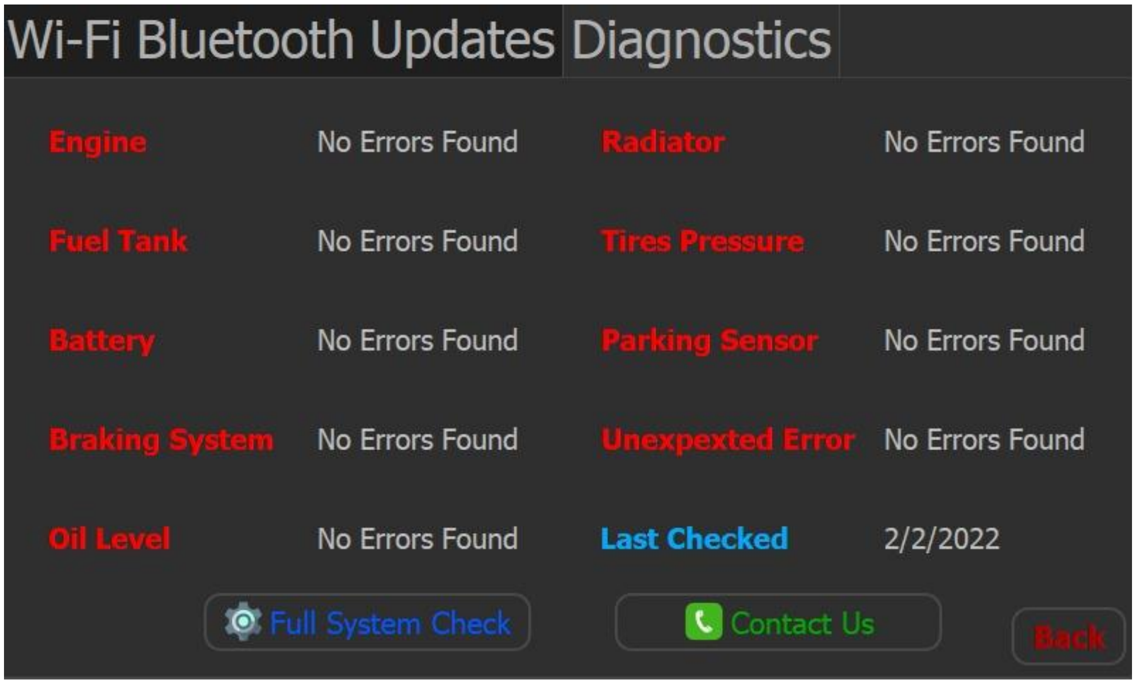
Calendar: is used for displaying dynamic Date and navigating the Day, Month and Year.



Sat	Sun	Mon	Tue	Wed	Thu	Fri
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Figure 4.39 GUI Calendar Window.

Settings: is used for dynamically scanning for Wi-Fi connections and Bluetooth devices, checking and receiving an update, sending and receiving and diagnostics system check.



Engine	No Errors Found	Radiator	No Errors Found
Fuel Tank	No Errors Found	Tires Pressure	No Errors Found
Battery	No Errors Found	Parking Sensor	No Errors Found
Braking System	No Errors Found	Unexpexted Error	No Errors Found
Oil Level	No Errors Found	Last Checked	2/2/2022

Figure 4.40 GUI Settings Window.

Music: is used for dynamically playing and pausing music files that are on the vehicle.

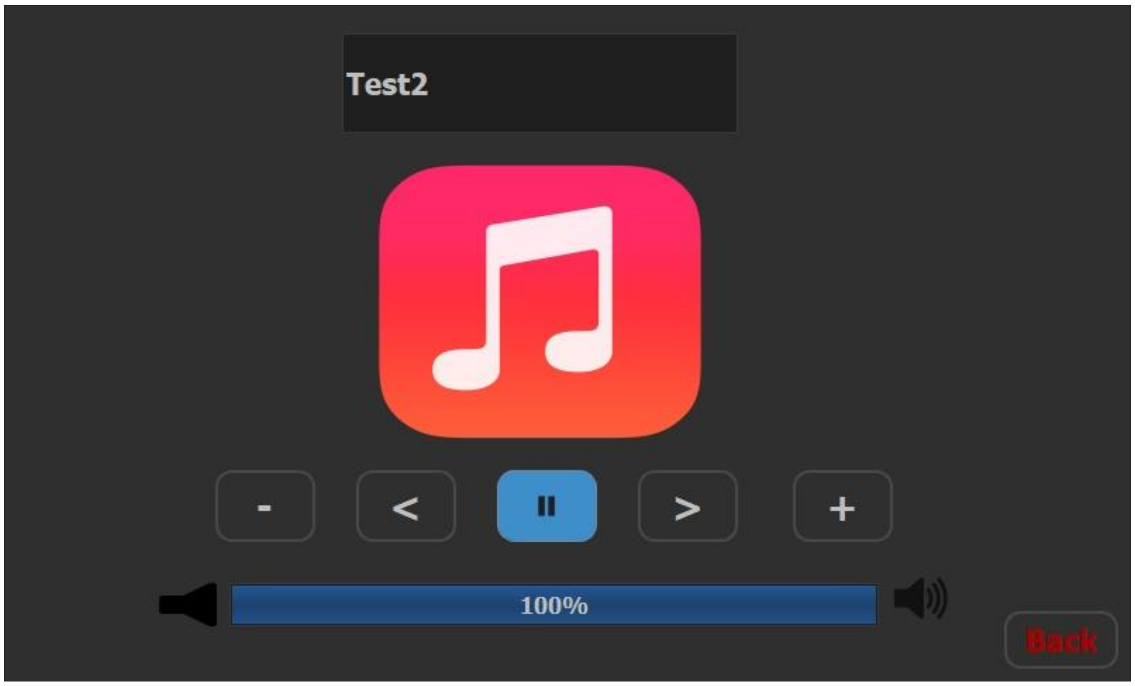


Figure 4.41 GUI Music Window.

Video: is used for dynamically playing and pausing video files that are on the vehicle.

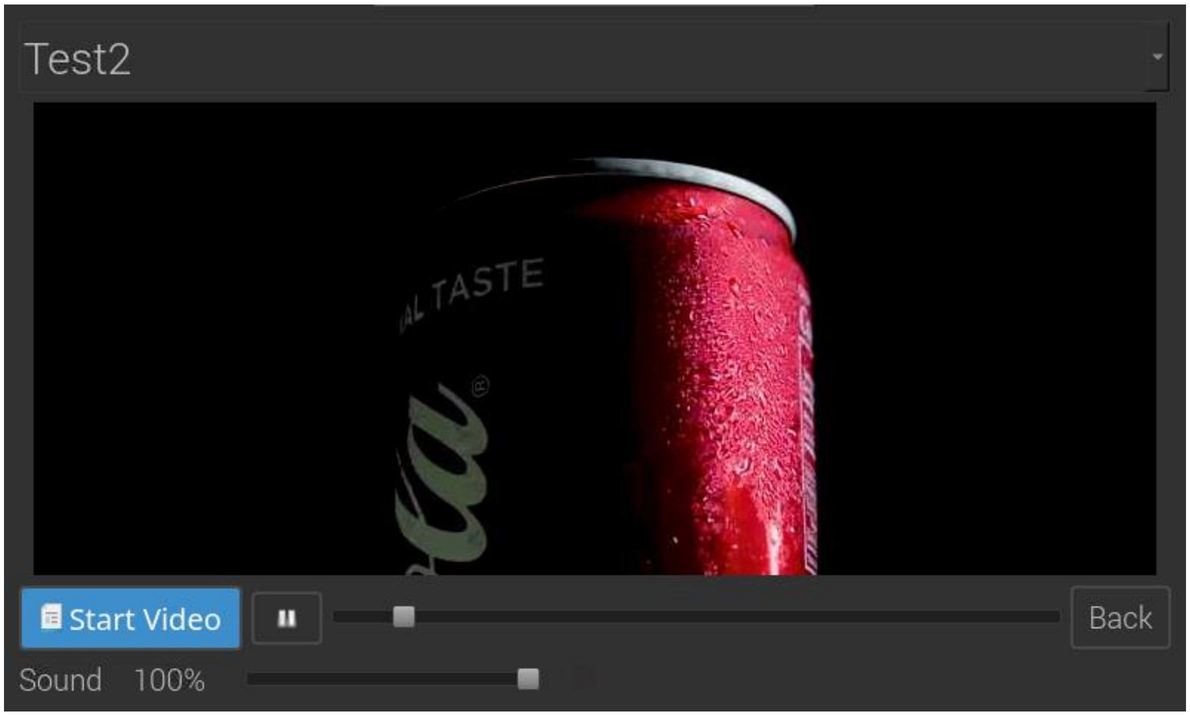


Figure 4.42 GUI Video Window.

Radio: is used for playing the radio that is scanned by the vehicle’s antenna.

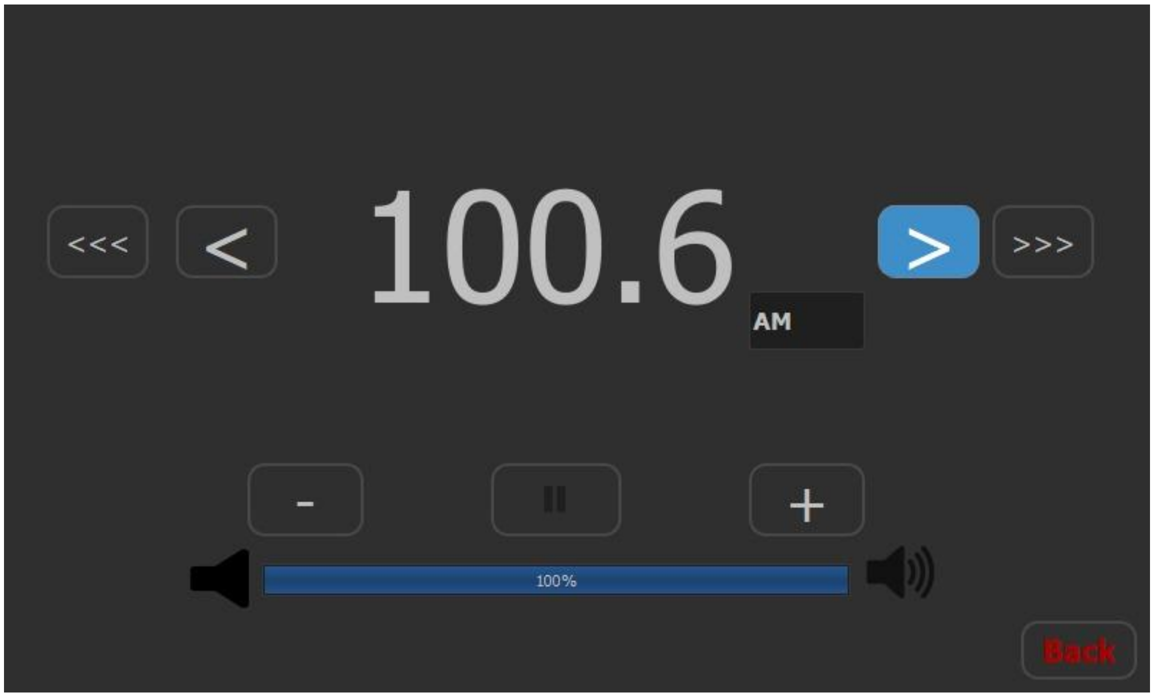


Figure 4.43 GUI Radio Window

Weather: is used for dynamically displaying the weather details with a whole day forecast integrating with Weather API free to use public API for the data.

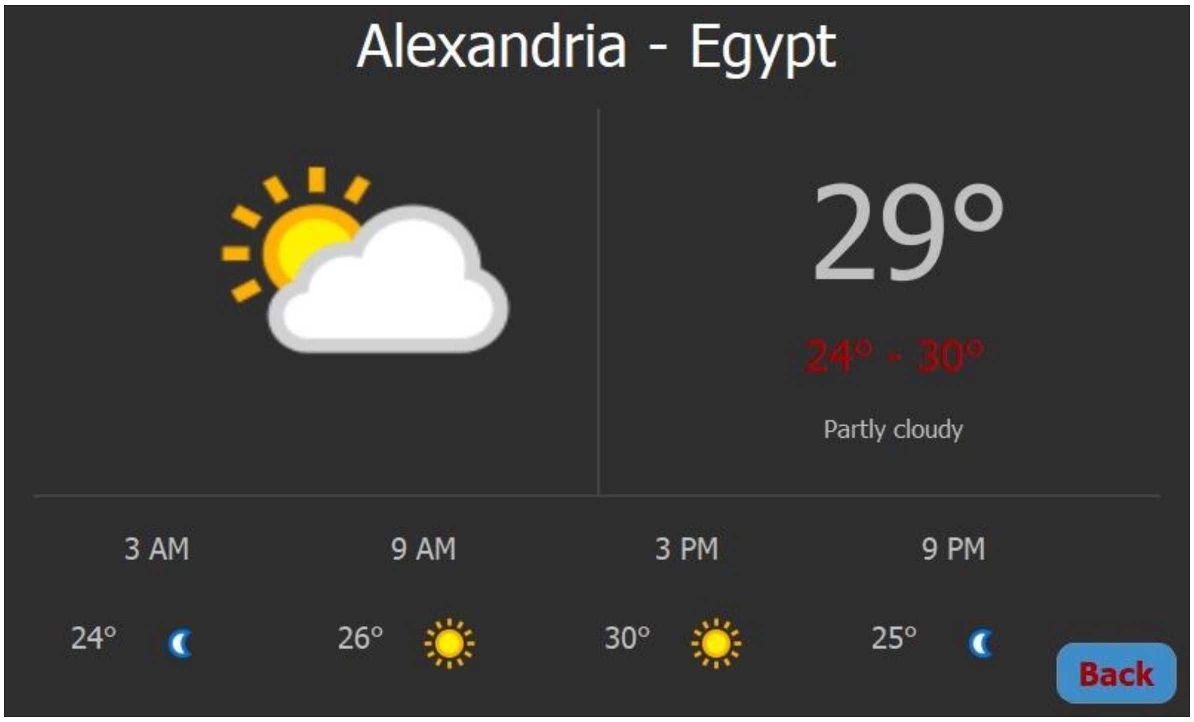


Figure 4.44 GUI Weather GUI.

4.10. Hardware Implementation

4.10.1. Altium PCB Designer

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

Figure 4.45 Altium Designer System.

Altium Designer's suite encompasses four main functional areas, including schematic capture, 3D PCB design, field-programmable gate array (FPGA) development and release/data management. It integrates with several component distributors for access to manufacturer's data. It also has interactive 3D editing of the board and MCAD export to STEP.

4.10.2. Implemented PCB

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

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

The Final PCB Dimensions is 30cm x 15cm.

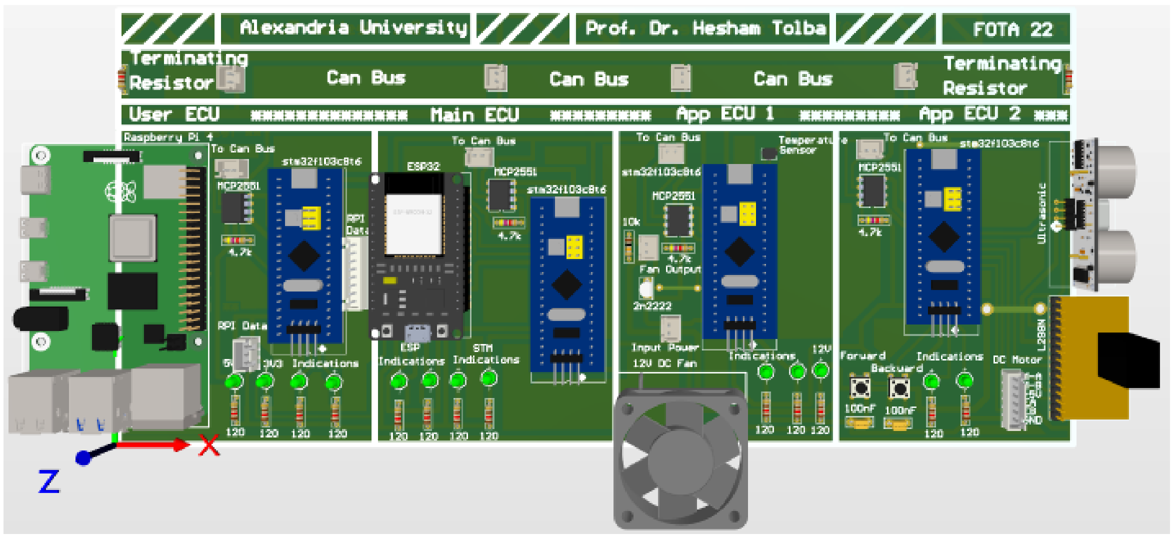


Figure 4.46 3D Model of our PCB.

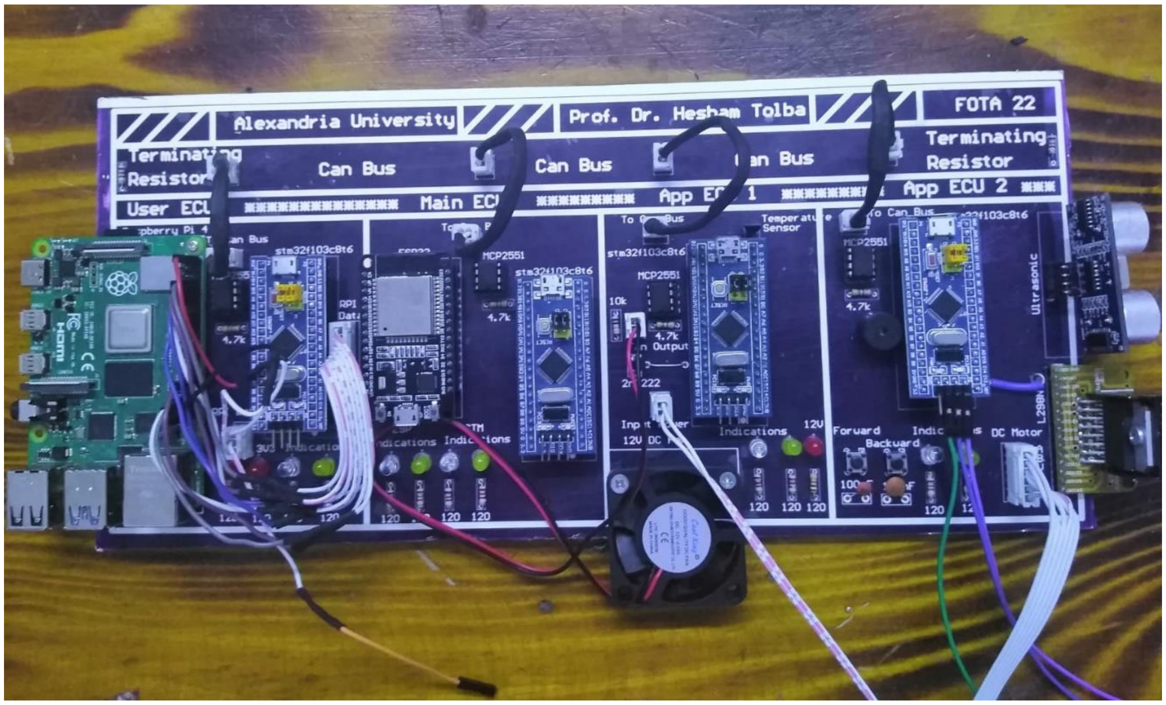


Figure 4.47 Real view of our PCB.

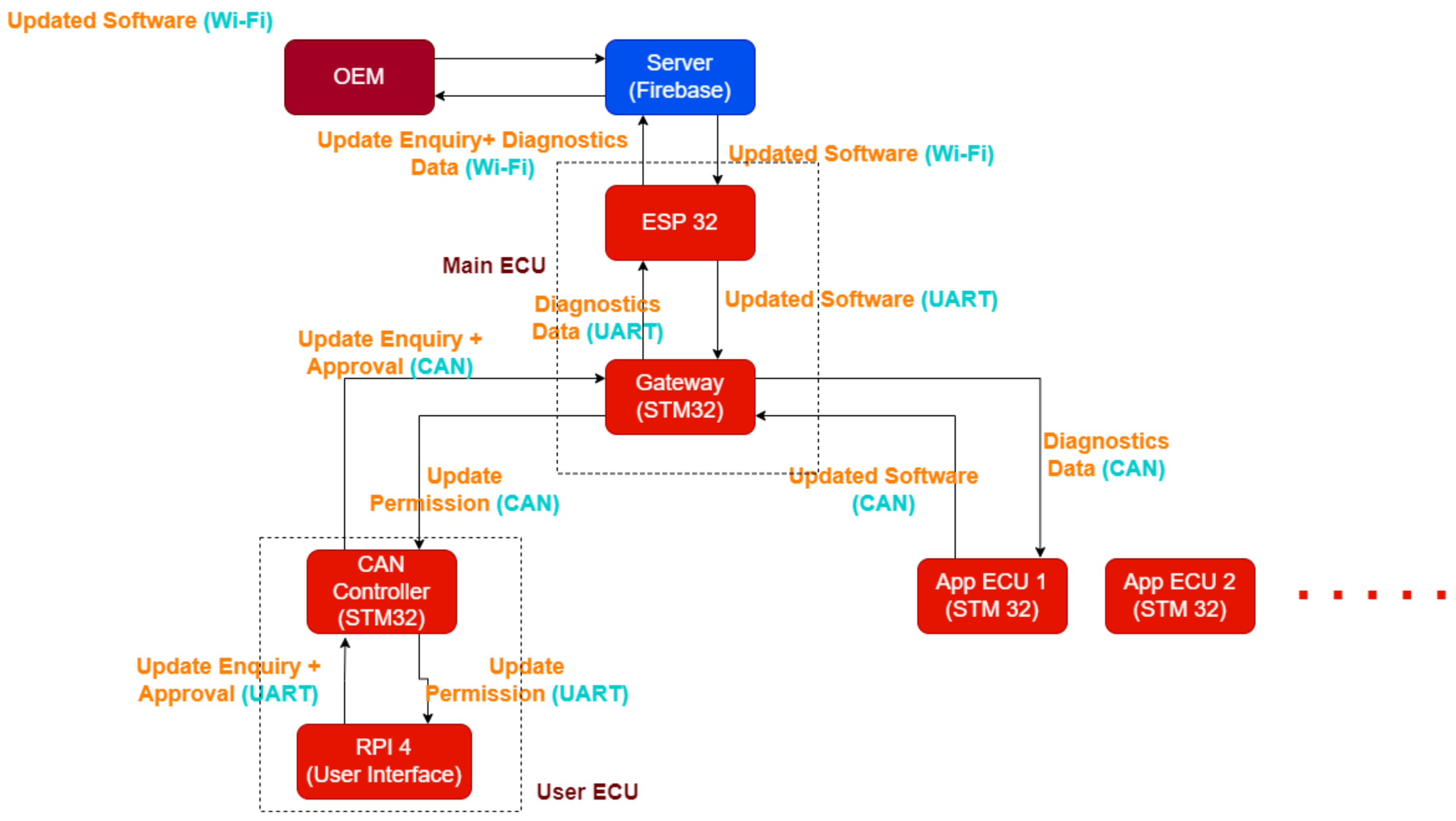


Figure 4.48 Detailed Diagram with all system components.

Chapter 5: Conclusion

5.1. Achievements

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

GitHub Repository Link:

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

Final Project Video Link:

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

5.2. Future Improvements

5.2.1. Adaptive AUTOSAR for diagnostics

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

5.2.2. Delta file

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

5.3. Total Cost

Here is a table that comprise hardware components and their price:

Component	Number	Price
Raspberry Pi 4	1	1,900
Touch Screen	1	750
SPI to CAN Module	1	180
Raspberry Pi Case + Fan	1	100
STM32F103C8	4	175
ESP 32	1	220
SD Card Module	1	50
SD Card	2	100
FM Radio Receiver Module	1	310
HDMI + Type C Cables	1	50
Reader	2	10
MCP 2551	7	45
Logic Level Converter	2	15
Screw Driver	1	12
Adaptor 5v	3	30
Solder Iron + Solder Wire	1	125
St Link	1	120
L298	2	35
PCB	1	780
Total	6022	

Tools

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

References

[1] Firmware over the Air Introduction
https://www.soracom.io/iot-definitions/what-is-firmware-over-the-air-fota/

[2] Software Development Cycle
https://phoenixnap.com/blog/software-development-life-cycle

[3] FOTA Architecture
https://mirror-medium.com/?m=https%3A%2F%2Fmedium.com%2F%40shrimantshubham%2Fwhat-is-fota-how-does-it-work-502c34a06d60

[4] OTA System Design
https://www.analog.com/ru/analog-dialogue/articles/over-the-air-ota-updates-in-embedded-microcontroller-applications.html

[5] Role of OTA Updates in IoT Device Management
https://thefutureofthings.com/14610-the-role-of-ota-updates-in-iot-device-management/

[6] CAN Protocol
https://www.javatpoint.com/can-protocol

[7] CAN Protocol Overview
https://www.ni.com/en-lb/innovations/white-papers/06/controller-area-network--can--overview.html

[8] Bootloader
https://www.sciencedirect.com/topics/engineering/bootloader

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

[10] UART Introduction
https://wiki.st.com/stm32mcu/wiki/STM32StepByStep:Step3_Introduction_to_the_UART

[11] RTOS Overview
https://www.windriver.com/solutions/learning/rtos#:~:text=A%20real%2Dtime%20operating%20system%20(RTOS)%20is%20an%20operating,this%20is%20not%20necessarily%20so.

[12] Free RTOS Documentation
https://www.freertos.org/

[13] UART Tutorial
https://deepbluembedded.com/stm32-usart-uart-tutorial/

[14] Client Server Overview
https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview

[15] Firebase Overview
https://firebase.google.com/docs/extensions/overview-use-extensions

[16] Vehicle Diagnostics Over Internet Protocol and Over-the-Air Updates by M. Kathiresh, R. Neelaveni, M. Adwin Benny & B. Jeffrin Samuel Moses.

[17] STM32F103 datasheet.

[18] Raspberry Pi Documentation.
https://www.raspberrypi.com/