Project Report On

Design and Implementation of Firmware Over-The-Air (FOTA) for STM32 Microcontroller

Submitted in Partial Fulfilment for the award of

Post Graduate Diploma in Embedded Systems Design

(PG-DESD)

From

CDAC, ACTS (Pune)

Guided by

Mr. Rhugved Rane

Presented by

Mr. Aditya Bachal	**250840130002**
Mr. Akash Shelke	**250840130006**
Mr. Yash Pandey	**250840130034**
Mr. Poduri Sankar	**250840130038**
Mr. Devesh Suroshi	**250840130053**

Centre for Development of Advanced Computing

(C-DAC), Pune

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to **C-DAC (Centre for Development of Advanced Computing)** for providing me the opportunity to work on this project as a part of the **Post Graduate Diploma in Embedded Systems Design (PG-DESD)** curriculum.

I am extremely thankful to my project guide **Mr. Rhugved Rane** for their valuable guidance, continuous encouragement, and technical support throughout the development of this project. Their insights and suggestions played a crucial role in shaping the design and implementation of the **Firmware Over-The-Air (FOTA) system for STM32**.

I would also like to thank the **Mrs. Jubera Khan (Course Coordinator, PG-DESD) and faculty members** for their constant support, motivation, and for providing the necessary infrastructure, lab facilities, and learning resources required to successfully complete this project.

Finally, I extend my sincere thanks to my friends and family for their encouragement and support throughout the course and during the completion of this project.

ABSTRACT

In modern embedded systems, firmware plays a critical role in defining device functionality, performance, and reliability. As embedded devices are increasingly deployed in remote, inaccessible, or large-scale environments, the ability to update firmware after deployment has become a fundamental requirement. Traditional firmware update methods that rely on physical access using interfaces such as JTAG, USB, or serial communication are inefficient, costly, and impractical for deployed systems. This has led to the widespread adoption of **Firmware Over-The-Air (FOTA)** mechanisms, which enable remote firmware updates while maintaining system integrity and minimizing downtime.

This project presents the **design and implementation of a robust Firmware Over-The-Air (FOTA) system for the STM32F407VG microcontroller**, implemented on the STM32F407 Discovery board. The proposed system uses a **custom bootloader**, a **dedicated flash metadata section**, and a **dual-application firmware architecture** to ensure safe, reliable, and fault-tolerant firmware updates. The firmware update data is transferred using **SPI communication**, with an **ESP32 microcontroller acting as an external firmware controller and data source**. A host system running a Python-based server application manages firmware images and initiates the update process.

The STM32 bootloader is the first software component executed after system reset and resides in a protected region of internal flash memory. Its responsibilities include initializing essential hardware, reading firmware metadata from flash, deciding which application firmware slot to execute, and managing firmware updates when requested. A dedicated **metadata section** stored in flash memory is used to track critical firmware state information such as the active application slot, firmware validity, and update-in-progress status. This metadata-driven approach allows the bootloader to make deterministic decisions at every system startup and ensures safe recovery in case of power failure or incomplete updates.

To improve system reliability, a **dual-application (dual-slot) architecture** is implemented. Two independent application firmware slots are maintained in flash memory. During a firmware update, the new firmware image is written to the inactive slot while the currently running application remains untouched. Only after the new firmware is completely received, programmed, and validated does the bootloader update the metadata and switch execution to the new application. If any error occurs during the update process, the system automatically falls back to the previously verified firmware, thereby preventing device bricking.

The firmware update process follows a structured flow in which firmware data is transmitted from a host system to the ESP32, and subsequently from the ESP32 to the STM32 bootloader over SPI. The STM32 bootloader receives the firmware in fixed-size packets, performs flash erase and write operations using **STM32 HAL APIs**, and updates metadata flags at each critical stage of the update process. This ensures compliance with STM32 flash memory constraints and guarantees data integrity.

The complete system was implemented and tested on actual hardware. Various test scenarios were performed, including normal firmware updates, interrupted updates due to power loss, corrupted firmware transfers, and rollback to a previous firmware version. Experimental results demonstrate that the implemented FOTA system reliably updates firmware without physical access while maintaining system stability and recoverability.

The proposed architecture closely reflects industry-standard embedded firmware update practices and provides a scalable foundation for future enhancements such as secure firmware authentication, encryption, wireless communication support, and RTOS integration. This project highlights the importance of metadata-driven decision logic and dual-slot firmware management in building reliable FOTA systems for modern embedded applications.

Keywords

STM32F407, Firmware Over-The-Air (FOTA), Custom Bootloader, Flash Metadata, Dual Application, SPI Communication, ESP32, Embedded Systems

---

TABLE OF CONTENTS

1. INTRODUCTION

Embedded systems are increasingly deployed in environments where physical access to the device is either difficult or completely unavailable. Applications such as industrial automation, automotive electronics, smart devices, and Internet of Things (IoT) systems require frequent firmware updates to fix bugs, improve performance, add new features, and address security vulnerabilities after deployment. Traditional firmware update techniques based on physical interfaces such as JTAG, UART, USB, or SWD are impractical for such systems due to increased maintenance cost, downtime, and manual intervention.

Firmware Over-The-Air (FOTA) is a modern and efficient solution that enables remote firmware updates without physical access to the target device. In a FOTA system, firmware images are transferred through a communication interface and programmed into the internal flash memory of the microcontroller while ensuring system reliability and fault tolerance.

This project presents the design and implementation of a reliable FOTA system for the **STM32F407VG** microcontroller using a **custom bootloader** and a **dual-application (Slot A / Slot B) architecture**. To enable network-based firmware delivery, an **ESP32** module is used as an external communication controller. The ESP32 handles cloud connectivity, downloads firmware binaries from an **AWS S3** bucket using HTTP, and transfers the firmware to the STM32 via SPI communication.

SPI is selected as the firmware transfer interface between ESP32 and STM32 due to its high data rate, simplicity, and reliability. The STM32 bootloader manages flash memory operations, firmware validation, and safe application switching. By maintaining two separate application slots in internal flash memory, the system ensures that a valid firmware is always available, preventing device failure during interrupted or corrupted updates.

This project demonstrates a complete end-to-end FOTA solution, from cloud-based firmware storage to safe execution of updated firmware on the target embedded device.

2. LITERATURE SURVEY

Firmware update mechanisms have been an essential part of embedded system development, particularly as devices evolve from standalone systems to connected and remotely deployed platforms. Early embedded systems relied on physical interfaces such as JTAG, UART, or USB for firmware updates. While these methods are reliable during development and testing, they are not suitable for deployed systems due to the requirement of physical access, increased maintenance cost, and system downtime.

With the growth of connected embedded systems and IoT applications, **Firmware Over-The-Air (FOTA)** has emerged as a standard solution for remote firmware management. Existing literature highlights FOTA as a critical component in maintaining system functionality, deploying security patches, and extending the operational lifetime of embedded devices. Many commercial and open-source implementations adopt bootloader-based approaches to safely manage firmware updates.

Bootloader-centric FOTA designs are widely discussed in research and industry documentation. In such systems, a minimal and reliable bootloader is stored in a protected flash memory region and executed immediately after reset. The bootloader handles firmware reception, validation, and installation before transferring control to the application. Studies emphasize that keeping the bootloader small and independent of application logic significantly improves system reliability.

Several researchers propose **dual-application or dual-bank firmware architectures** to enhance fault tolerance. In this approach, two separate application slots are maintained in flash memory. One slot contains the currently running application, while the other is used to store the new firmware during the update process. This method ensures that a valid application is always available, even if the update process is interrupted due to power loss or communication failure. Dual-application architectures are particularly suitable for microcontrollers like the STM32F4 series, which provide sufficient internal flash memory.

Communication protocols play a vital role in FOTA performance and reliability. Literature explores various interfaces such as UART, SPI, I2C, CAN, Ethernet, and wireless technologies for firmware transfer. Among these, **SPI-based firmware transfer** is recognized for its high data rate, simplicity, and robustness. SPI is commonly used in embedded systems where firmware is fetched from an external controller, memory device, or communication module.

Flash memory management is another major focus area in FOTA research. Since flash memory supports sector-based erase and limited write cycles, careful handling of erase and write operations is essential. Many studies recommend using hardware abstraction layers, such as the **STM32 HAL API**, to ensure correct timing, alignment, and protection during flash operations. Proper flash partitioning and validation techniques such as CRC checks are also emphasized to maintain data integrity.

In summary, existing literature indicates that a reliable FOTA system must integrate a custom bootloader, robust communication interface, fault-tolerant memory architecture, and careful flash management. The design choices made in this project—custom bootloader, SPI communication, dual-application architecture, and HAL-based implementation—are strongly supported by established research and industry practices.

3. AIM OF THE PROJECT

The primary aim of this project is to **design and implement a reliable Firmware Over-The-Air (FOTA) update system for the STM32F407VG microcontroller** using a **custom bootloader and dual-application architecture**. The project focuses on enabling firmware updates without physical access to the device while ensuring system stability, data integrity, and recovery from update failures.

The project aims to develop a **custom bootloader** that executes immediately after reset and manages the complete firmware update lifecycle. This includes receiving the new firmware image over **SPI communication**, storing it safely in flash memory, verifying its validity, and transferring execution to the appropriate application firmware. The bootloader is designed to operate independently of the application firmware and remains protected from accidental overwriting during update operations.

Another key objective of the project is to implement a **dual-application firmware structure** within the internal flash memory of the STM32F407 microcontroller. In this approach, two separate application regions are maintained: one for the currently running firmware and another for the newly received firmware image. This architecture ensures that the system can safely fall back to a known working firmware if the update process fails due to power interruption, communication error, or firmware corruption.

The project also aims to efficiently manage **flash memory erase and write operations** using the **STM32 HAL API**. Proper handling of flash sectors, alignment constraints, and write protection mechanisms is critical to avoid memory corruption and ensure long-term reliability of the system. The implementation focuses on complying with STM32 flash programming guidelines to maintain data integrity during firmware updates.

Additionally, the project aims to validate the implemented FOTA system through extensive testing on the **STM32F407 Discovery board**. This includes verifying successful firmware updates, correct application switching, safe handling of invalid firmware images, and system recovery under abnormal conditions. The project ultimately aims to provide a practical and scalable FOTA solution suitable for real-world embedded applications.

The primary aim of this project is to design and implement a robust and fault-tolerant Firmware Over-The-Air (FOTA) update system for the STM32F407VG microcontroller using a custom bootloader and dual-application architecture.

The project aims to:

- Enable remote firmware updates without physical access
- Ensure system reliability during firmware updates
- Prevent device bricking due to power failure or corrupted firmware
- Integrate cloud-based firmware storage and transfer mechanisms

4. SCOPE AND OBJECTIVES

The scope of this project encompasses the complete design, development, and validation of a **Firmware Over-The-Air (FOTA) update system for the STM32F407VG microcontroller** using a custom bootloader approach. The project is focused on implementing a reliable firmware update mechanism that operates without physical access to the target device and ensures safe recovery in case of update failure.

The scope includes the development of a **custom bootloader** that resides in a protected region of the internal flash memory. The bootloader is responsible for system initialization, firmware update decision-making, firmware reception over SPI, flash memory programming, firmware validation, and controlled transfer of execution to the application firmware. The bootloader is designed to be independent of application-level functionality to maximize reliability.

A significant scope of the project involves implementing a **dual-application firmware architecture**. The internal flash memory of the STM32F407 is partitioned into multiple regions, including the bootloader region, primary application region, and secondary application (backup) region. This architecture ensures that a valid application firmware is always available for execution, even if the firmware update process is interrupted or fails.

The project scope also includes the use of **SPI communication** as the firmware transfer interface. The firmware image is received in chunks over SPI, allowing controlled and efficient transfer of large firmware binaries. The SPI-based update mechanism is designed to handle communication errors and ensure proper synchronization between the transmitting and receiving ends.

The scope of this project includes the complete design, development, and validation of a FOTA system based on internal flash memory of the STM32F407VG microcontroller.

Objectives:

- Design and implement a custom STM32 bootloader
- Implement dual-slot firmware architecture (Slot A and Slot B)
- Use ESP32 as an external communication controller
- Download firmware from AWS S3 using HTTP
- Transfer firmware from ESP32 to STM32 using SPI
- Safely manage flash memory using STM32 HAL APIs
- Validate system reliability through failure testing

Advanced features such as encrypted firmware images and authentication are outside the current scope and are considered future enhancements.

5. THEORETICAL DESCRIPTION OF THE PROJECT

This chapter explains the theoretical concepts that form the foundation of the implemented **Firmware Over-The-Air (FOTA) system for the STM32F407 microcontroller**. The discussion is limited to concepts directly relevant to the project, including the STM32 boot process, flash memory organization, bootloader operation, and dual-application firmware update methodology.

5.1 STM32 Boot Process

The STM32F407 microcontroller follows a predefined boot sequence after a reset. On power-up or system reset, the microcontroller fetches the initial stack pointer value from the first word of the flash memory, and the reset handler address from the second word. Execution begins at the reset handler, which is typically located at the start of the flash memory.

In this project, the reset vector is configured to point to a **custom bootloader** located at the beginning of the internal flash memory. The bootloader is therefore the first software component to execute after reset. Its role is to initialize minimal system resources and decide whether to remain in bootloader mode or transfer control to one of the application firmware images.

5.2 Flash Memory Organization in STM32F407

The STM32F407VG microcontroller provides internal flash memory organized into sectors of varying sizes. Flash memory operations are sector-based for erase and word-based for programming. Due to these constraints, firmware update mechanisms must carefully manage erase and write operations to avoid data corruption.

For this project, the internal flash memory is logically partitioned into the following regions:

- **Bootloader Region:** Contains the custom bootloader and is protected from overwrite.
- **Primary Application Region (App Slot 1):** Holds the currently active application firmware.
- **Secondary Application Region (App Slot 2):** Used to store the newly received firmware during the update process.

This partitioning enables safe firmware updates while maintaining a fallback option.

5.3 Bootloader-Based Firmware Update Concept

A bootloader is a small program responsible for managing firmware execution and updates. In a FOTA system, the bootloader performs tasks such as firmware reception, integrity checking, and application switching.

In this project, the bootloader checks for the presence of a new firmware image in the secondary application region. If a valid update is detected, the bootloader programs the

firmware into the designated application slot and updates the execution pointer. If the update is invalid or incomplete, the bootloader transfers control to the previously verified application firmware.

5.4 Dual-Application Firmware Architecture

The dual-application architecture is a fault-tolerant approach commonly used in reliable FOTA systems. Instead of overwriting the currently running firmware, the new firmware is written to a separate flash region. This approach prevents device failure due to interrupted updates.

The bootloader maintains metadata to identify which application slot contains a valid firmware image. Based on this information, the bootloader jumps to the appropriate application entry point after reset. This theoretical model ensures safe firmware upgrades and is directly applied in the implemented system.

This theoretical framework supports the practical implementation of the FOTA system described in subsequent chapters.

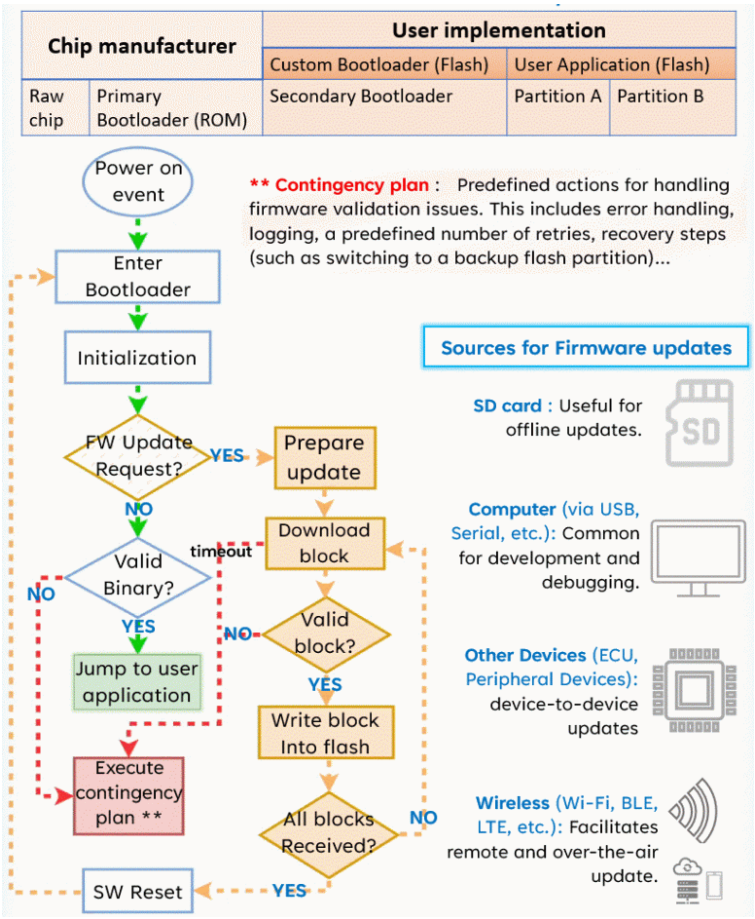
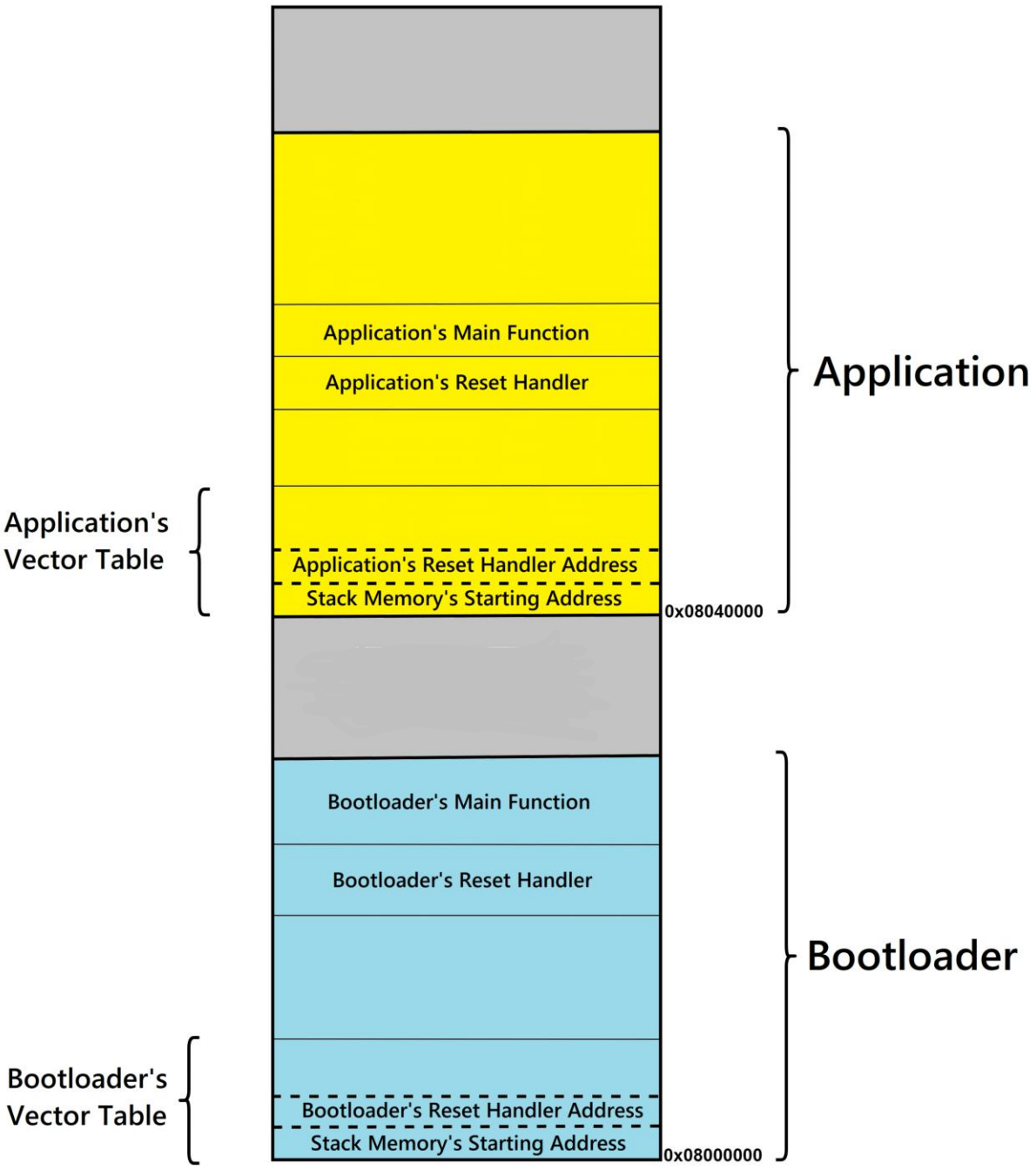


Figure 1. How Bootloaders Work?

6. SYSTEM DESIGN AND IMPLEMENTATION



The system design of the proposed Firmware Over-The-Air (FOTA) solution is centred around a **custom bootloader** and a **dual-application firmware architecture** implemented on the **STM32F407VG Discovery board**. The design ensures reliable firmware updates while preventing system failure due to interrupted or invalid updates.

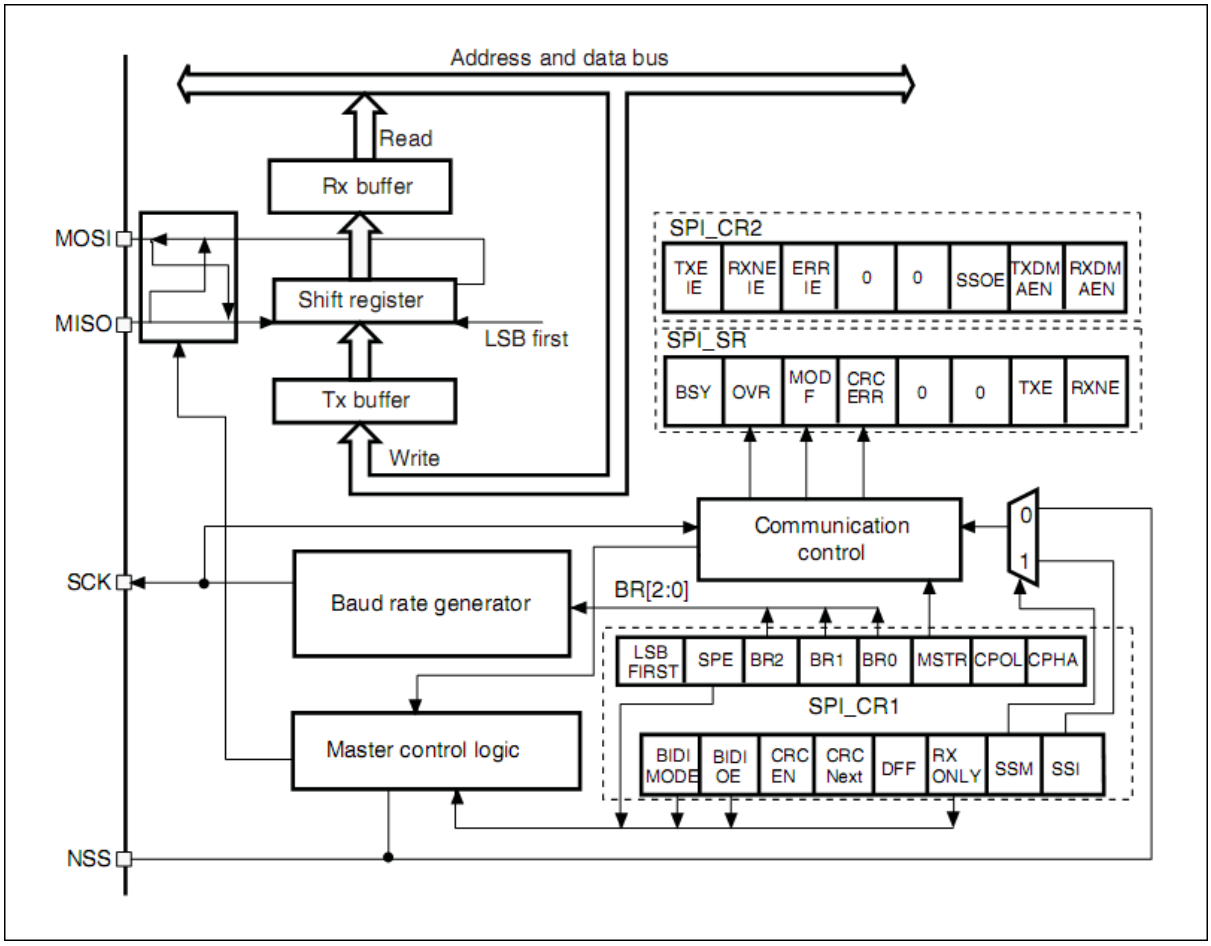
6.1 Overall System Architecture

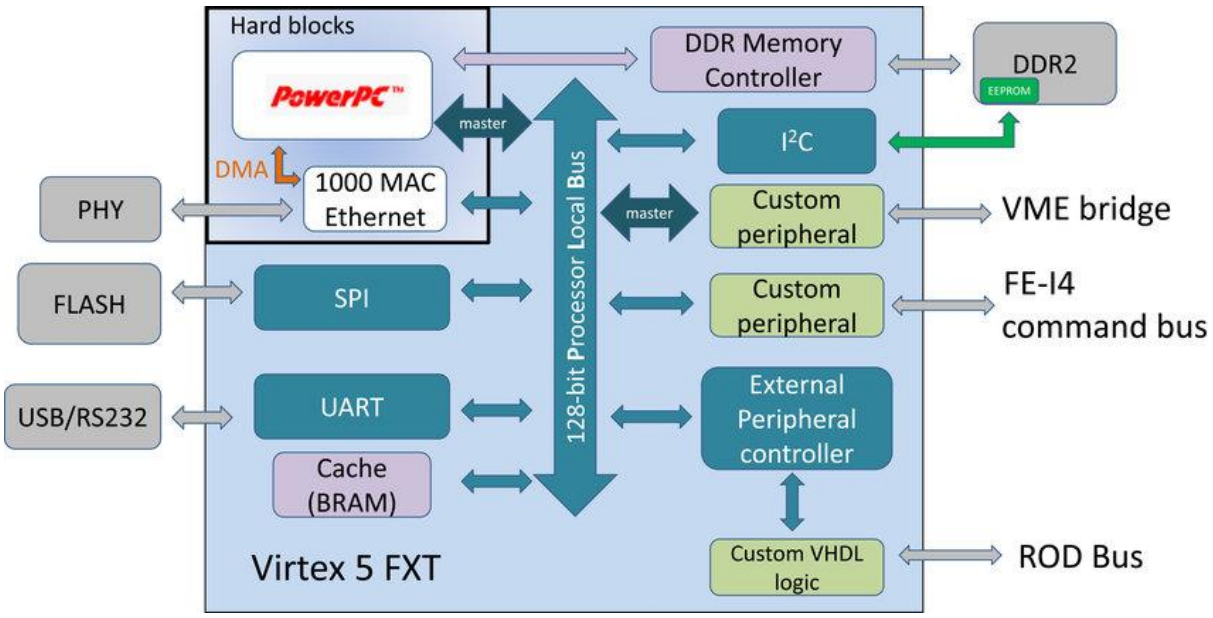
The overall architecture consists of three major components:

1. **Custom Bootloader**
2. **Primary Application Firmware (App Slot 1)**
3. **Secondary Application Firmware (App Slot 2)**

The custom bootloader resides at the beginning of the internal flash memory and executes immediately after reset. It is responsible for system initialization, firmware update management, validation, and application switching. The two application slots are placed in separate flash regions to support safe firmware upgrades.

6.2 Block Diagram of FOTA System



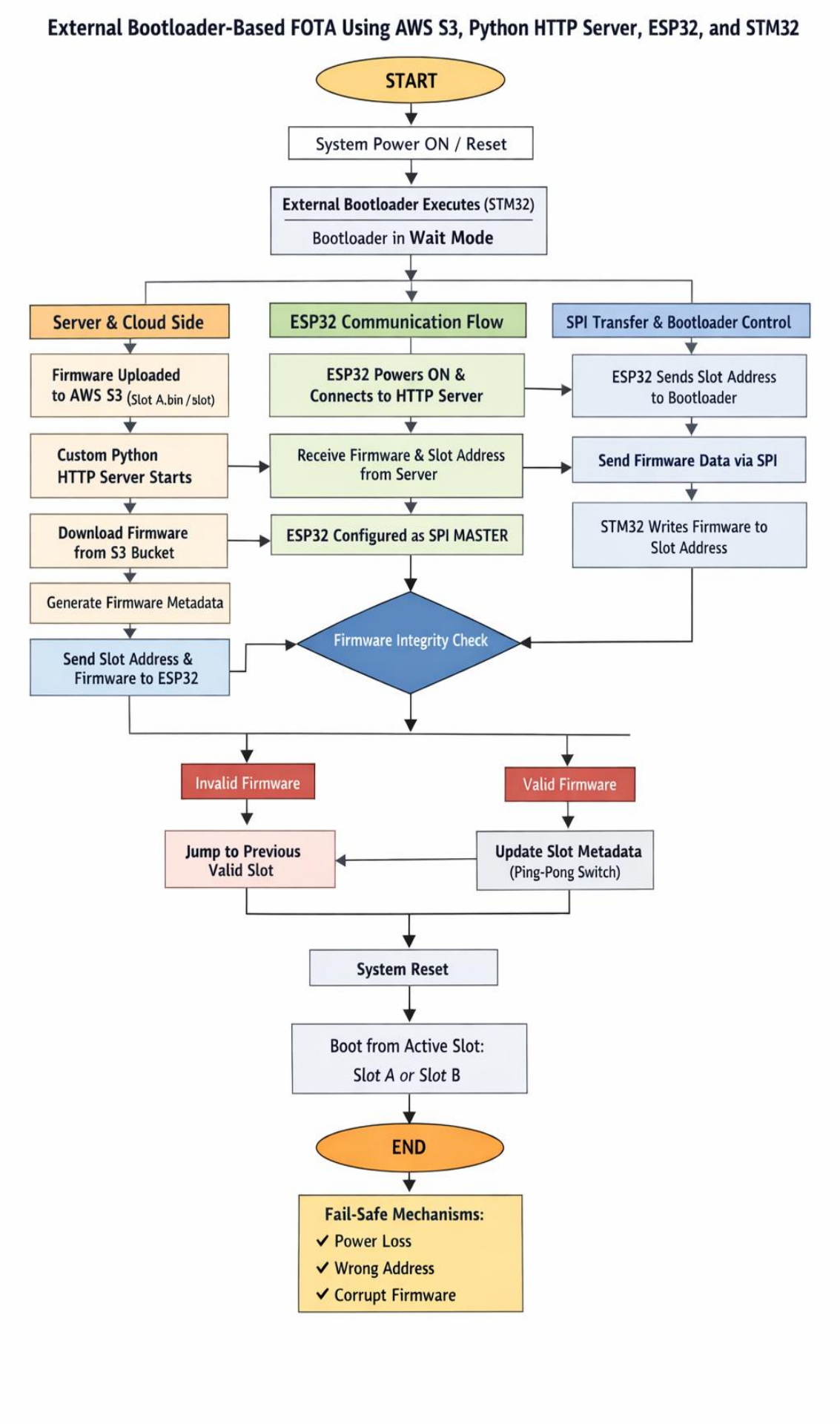


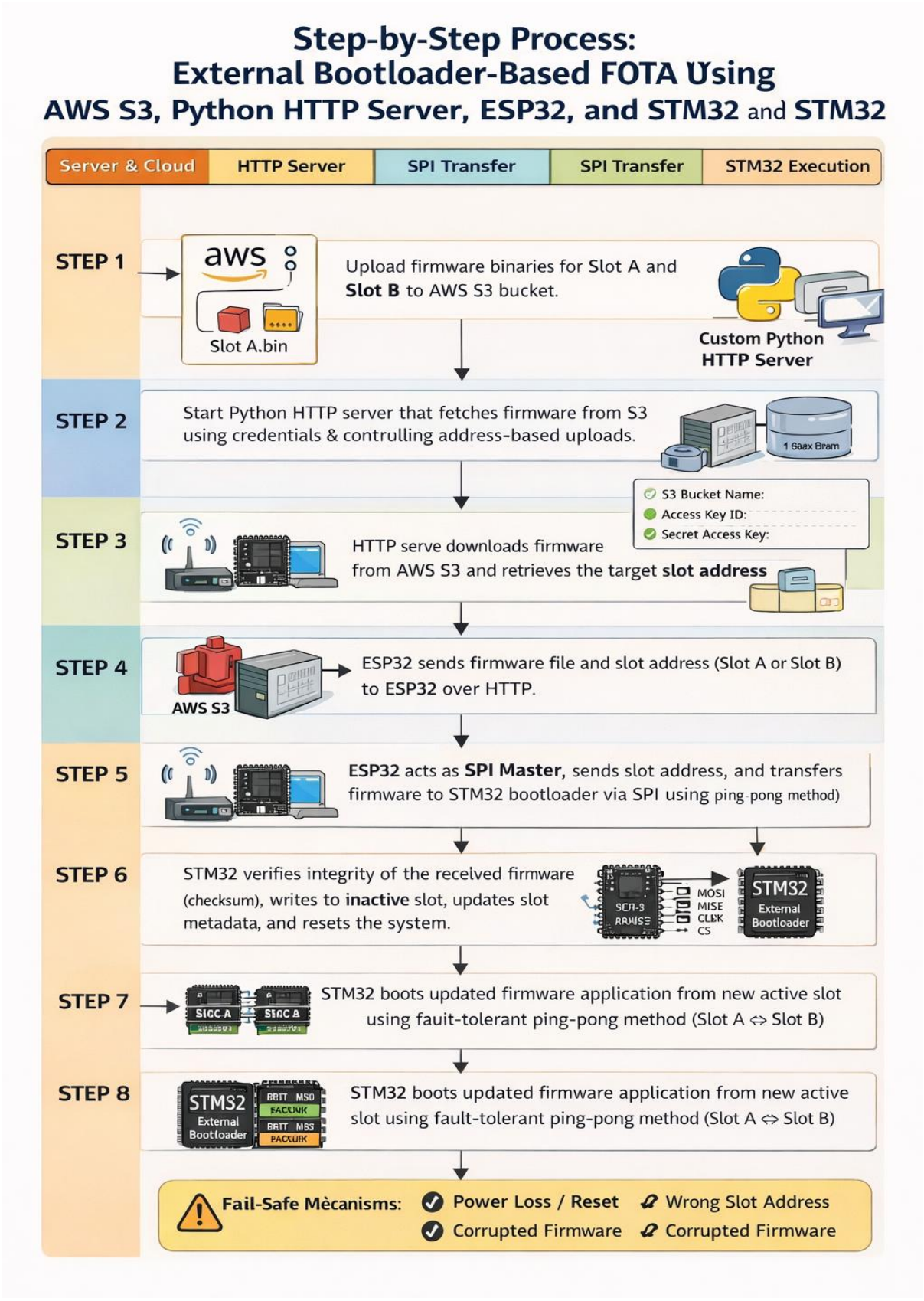
The block diagram illustrates the interaction between the bootloader, application firmware, flash memory, and the SPI communication interface. The firmware image is transferred to the target device using SPI and stored in the secondary application region. After successful validation, the bootloader updates the execution flow to the new application.

6.3 Firmware Update Flow

The firmware update process follows a well-defined sequence:

1. System reset triggers execution of the bootloader.
2. Bootloader initializes essential peripherals and flash interface.
3. Bootloader checks for a new firmware update request.
4. New firmware is received over SPI in fixed-size chunks.
5. Firmware is written to the secondary application flash region.
6. Integrity checks are performed to validate the firmware image.
7. On successful validation, the bootloader switches execution to the updated application.
8. In case of failure, the bootloader falls back to the previous valid application.


External Bootloader-Based FOTA Using AWS S3, Python HTTP Server, ESP32, and STM32
START
System Power ON / Reset
External Bootloader Executes (STM32)
Bootloader in Wait Mode
Server & Cloud Side
ESP32 Communication Flow
SPI Transfer & Bootloader Control
Firmware Uploaded to AWS S3 (Slot A.bin / slot)
ESP32 Powers ON & Connects to HTTP Server
ESP32 Sends Slot Address to Bootloader
Custom Python HTTP Server Starts
Receive Firmware & Slot Address from Server
Send Firmware Data via SPI
Download Firmware from S3 Bucket
ESP32 Configured as SPI MASTER
STM32 Writes Firmware to Slot Address
Generate Firmware Metadata
Firmware Integrity Check
Send Slot Address & Firmware to ESP32
Invalid Firmware
Valid Firmware
Jump to Previous Valid Slot
Update Slot Metadata (Ping-Pong Switch)
System Reset
Boot from Active Slot: Slot A or Slot B
END
Fail-Safe Mechanisms:
✓ Power Loss
✓ Wrong Address
✓ Corrupt Firmware


Step-by-Step Process:
External Bootloader-Based FOTA Using
AWS S3, Python HTTP Server, ESP32, and STM32 and STM32
Server & Cloud
HTTP Server
SPI Transfer
SPI Transfer
STM32 Execution
STEP 1
aws
Slot A.bin
Upload firmware binaries for Slot A and Slot B to AWS S3 bucket.
Custom Python HTTP Server
STEP 2
Start Python HTTP server that fetches firmware from S3 using credentials & contrulling address-based uploads.
S3 Bucket Name:
Access Key ID:
Secret Access Key:
STEP 3
HTTP serve downloads firmware from AWS S3 and retrieves the target slot address
STEP 4
AWS S3
ESP32 sends firmware file and slot address (Slot A or Slot B) to ESP32 over HTTP.
STEP 5
ESP32 acts as SPI Master, sends slot address, and transfers firmware to STM32 bootloader via SPI using ping-pong method)
STEP 6
STM32 verifies integrity of the received firmware (checksum), writes to inactive slot, updates slot metadata, and resets the system.
MOSI
MISE
CLEK
CS
STM32
External Bootloader
STEP 7
STM32 boots updated firmware application from new active slot using fault-tolerant ping-pong method (Slot A ⇔ Slot B)
STEP 8
STM32
External Bootloader
STM32 boots updated firmware application from new active slot using fault-tolerant ping-pong method (Slot A ⇔ Slot B)
Fail-Safe Mèchanisms:
Power Loss / Reset
Corrupted Firmware
Wrong Slot Address
Corrupted Firmware

6.4 Flash Memory Partitioning

The internal Flash memory of the STM32F407 microcontroller is logically partitioned to support a reliable and fail-safe Firmware-Over-The-Air (FOTA) update mechanism. The partitioning is organized as follows:

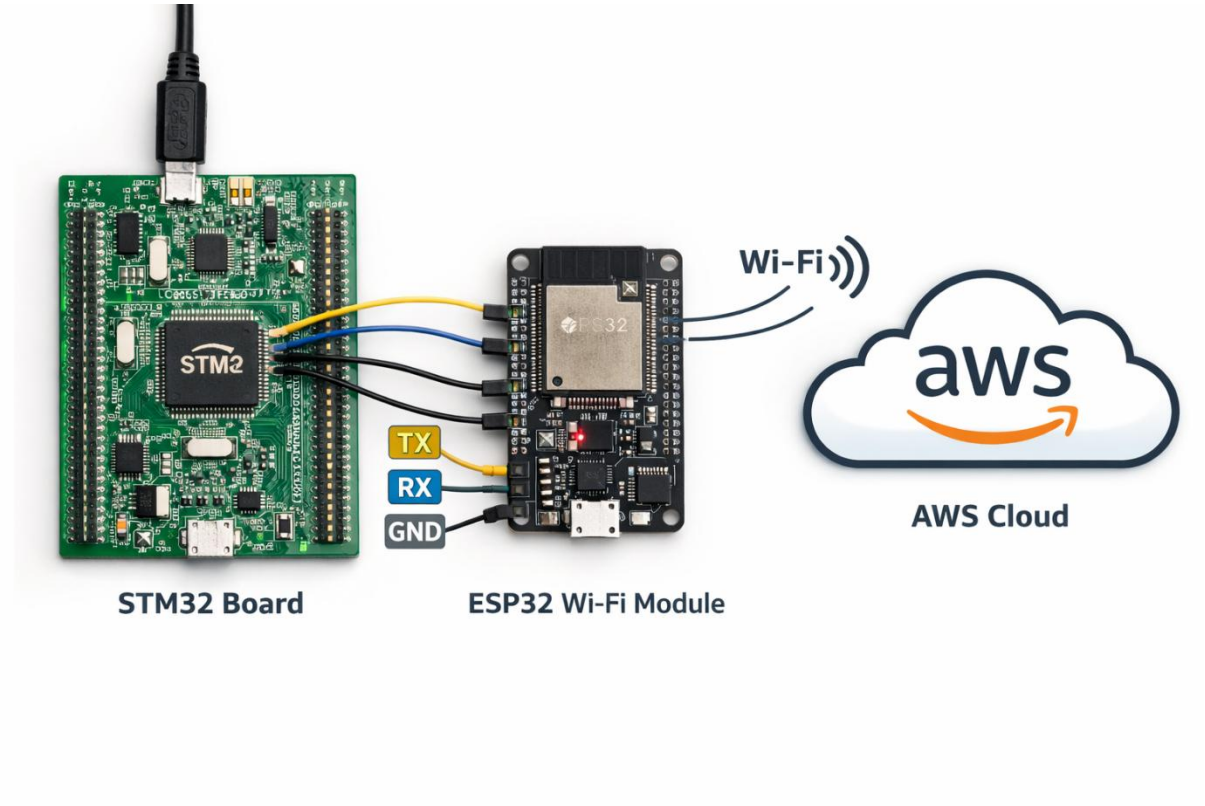
- Bootloader Region
  A protected flash region that contains the bootloader code. This region is non-overwritable during normal firmware updates and is responsible for system initialization, firmware validation, and application selection during startup.
- Application Slot 1 (Slot A)
  Stores the currently active and verified application firmware. The microcontroller executes this firmware under normal operating conditions.
- Application Slot 2 (Slot B)
  Serves as the backup or update slot where the newly downloaded firmware is stored. After successful validation, the bootloader switches execution from Slot A to Slot B using a ping-pong mechanism.

This dual-slot flash architecture ensures that at least one valid application firmware is always present, allowing the system to safely recover from power loss, incomplete updates, or communication failures during the firmware upgrade process.

6.5 Implementation Considerations

- **SPI Communication**
  SPI is utilized for reliable and high-speed firmware data transfer between the external communication module and the STM32F407, ensuring minimal latency and data integrity during updates.
- **HAL API Usage**
  The STM32 HAL (Hardware Abstraction Layer) APIs are used for flash erase and write operations, providing hardware-safe access, portability, and improved reliability.
- **Fail-Safe Design**
  The bootloader validates firmware integrity (e.g., size, checksum, or CRC) before execution. Corrupted or incomplete firmware is rejected, and the system continues running the last known valid application.
- **Scalability**
  The modular bootloader and memory architecture support future enhancements such as encrypted firmware images, secure key-based authentication, and cloud-based secure FOTA integration.

7. TOOLS USED AND RESULTS



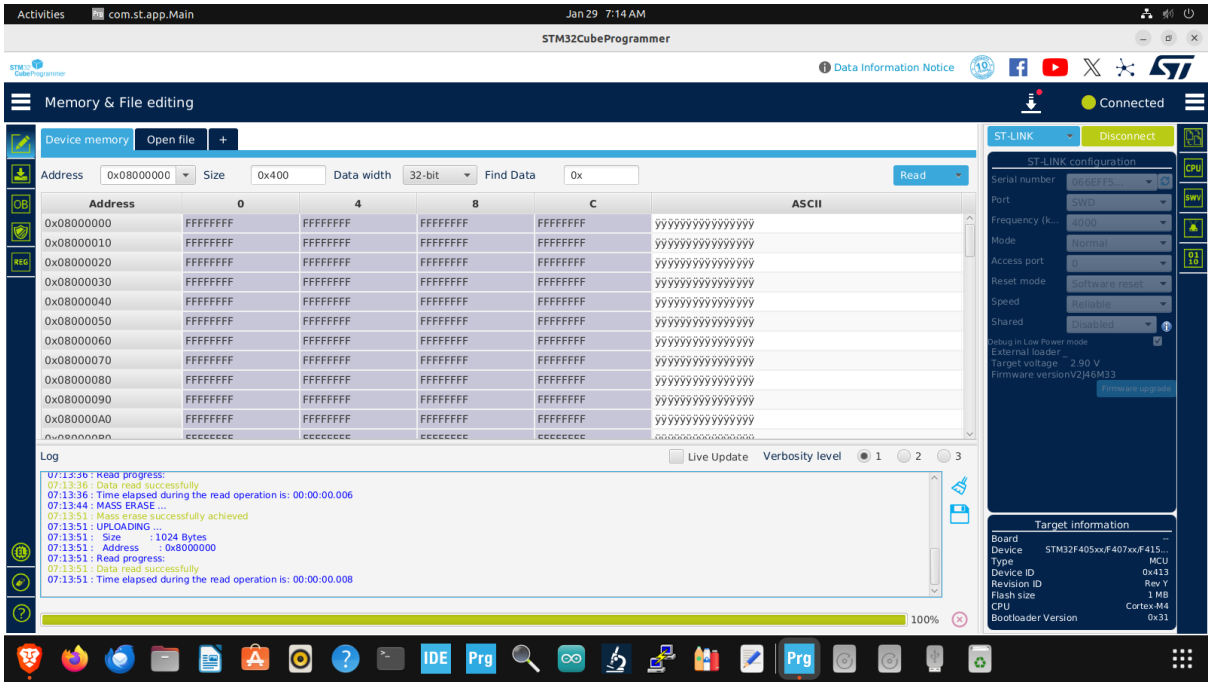
Step 1: Flash Memory Verification and Preparation

In this step, **STM32CubeProgrammer** is used to connect to the STM32F407 microcontroller via **ST-LINK (SWD interface)** and verify the internal Flash memory status before programming.

After establishing a successful connection, the **device memory starting at address 0x08000000** is read. The displayed values (0xFFFFFFFF) confirm that the Flash memory is **fully erased and in a clean state**, which is the expected condition prior to firmware programming.

A **mass erase operation** is performed to ensure that no residual or corrupted data remains in Flash. The successful erase and read-back verification indicate that the device is ready for bootloader or application firmware flashing.

This step is critical to guarantee reliable firmware deployment and to avoid conflicts caused by leftover memory contents.



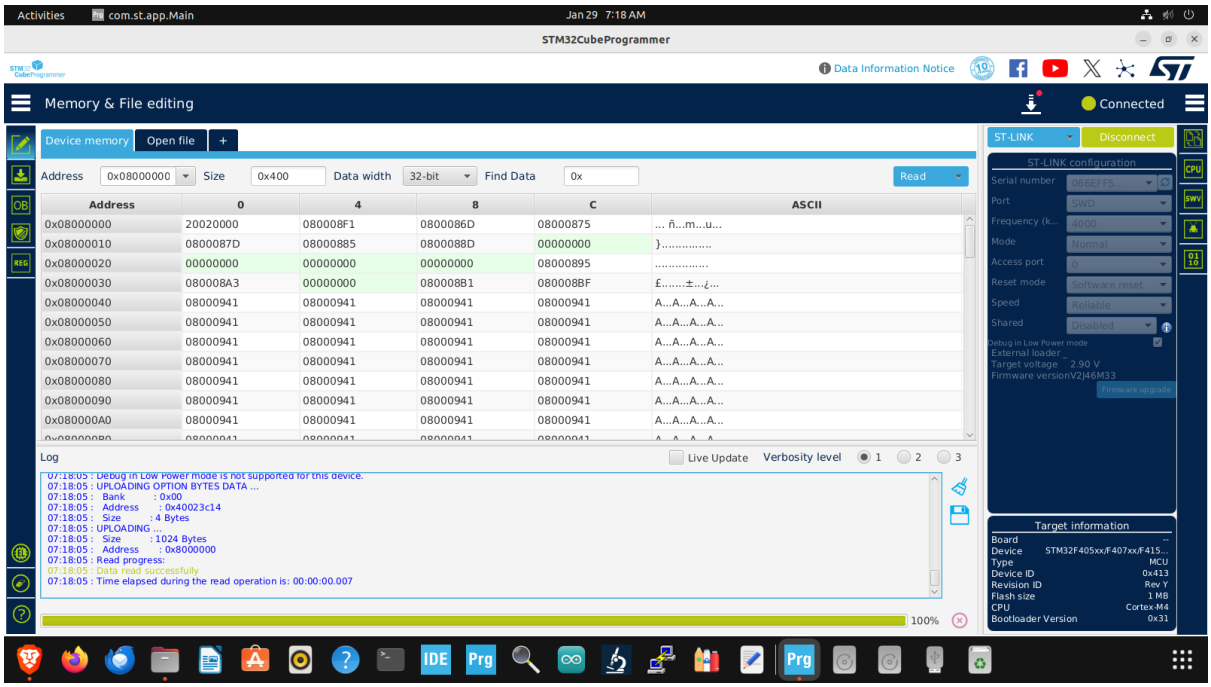
Step 2: Bootloader Programming and Validation

In this step, the **custom bootloader firmware** is successfully programmed into the STM32F407 internal Flash memory starting at the base address **0x08000000** using **STM32CubeProgrammer** via the **ST-LINK (SWD interface)**.

After the upload process, the Flash memory is read back to verify correct programming. The presence of valid vector table entries—such as the **initial stack pointer (0x20020000)** and **reset handler address (0x080008F1)**—confirms that the bootloader has been correctly written and linked.

The subsequent memory values indicate valid executable code rather than erased (0xFFFFFFFF) memory, verifying that the Flash now contains a functional bootloader image. This bootloader is responsible for startup initialization, firmware integrity checks, and application slot selection (Slot A / Slot B) during system reset.

Successful completion of this step ensures that the device can safely manage firmware updates and recover from failed or interrupted application programming.



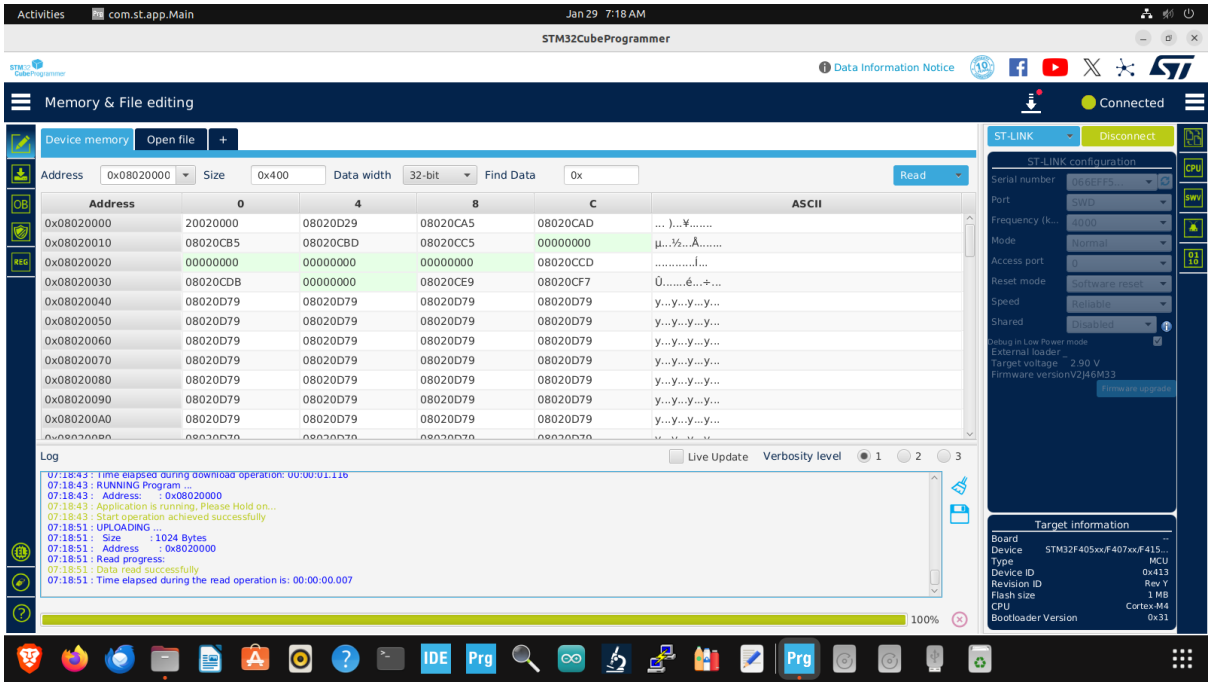
Step 3: Application Firmware Programming in Slot A

In this step, the **primary application firmware (Slot A)** is programmed into the STM32F407 internal Flash memory starting at address **0x08020000**, which is reserved for the active application region.

Using **STM32CubeProgrammer**, the application binary is uploaded while the bootloader remains intact at the lower flash region. After programming, a read-back verification is performed to confirm successful flashing. The memory contents show valid vector table entries, including the **initial stack pointer (0x20020000)** and the **reset handler address (0x08020D29)**, indicating that the application firmware has been correctly placed and linked.

The log confirms that the application is running successfully after programming, demonstrating proper bootloader-to-application handover. This step establishes Slot A as the **currently active firmware**, which will be executed during normal system operation.

Successful completion of this step ensures that the system has a stable application image available before initiating any firmware update to the secondary slot.



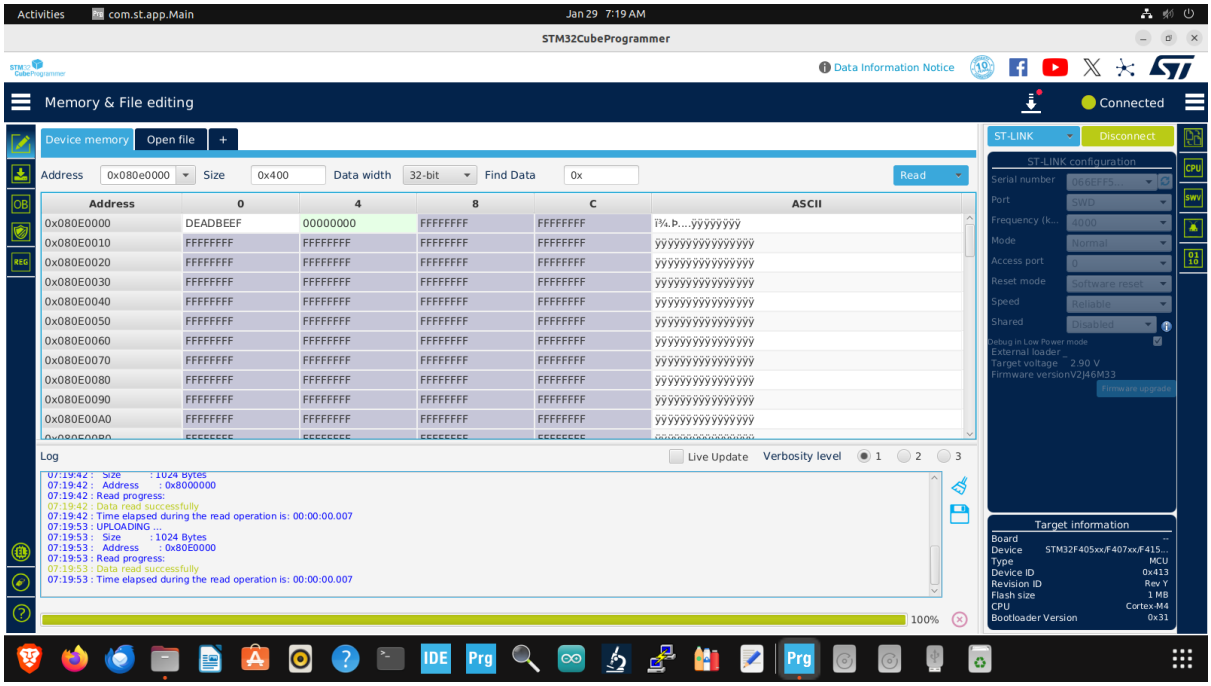
Step 4: Secondary Application Slot (Slot B) Initialization and Validation

In this step, the **secondary application region (Slot B)** located at Flash address **0x080E0000** is inspected and prepared for firmware update. Using **STM32CubeProgrammer**, the memory contents of Slot B are read to verify its initial state.

The Flash memory region shows erased values (0xFFFFFFFF), indicating that **Slot B is empty and ready to receive a new firmware image**. A predefined marker value (0xDEADBEEF) is used at the start of the slot as a **slot identification or status flag**, allowing the bootloader to differentiate between valid and invalid firmware images during startup.

This step ensures that Slot B is safely isolated from the currently active firmware (Slot A) and can be used as a temporary storage area for new firmware received via the ESP32 module during the FOTA process.

Successful completion of this step confirms that the system is ready to download and store updated firmware without affecting the running application.



Step 5: Firmware Hosting and Version Management Using AWS S3

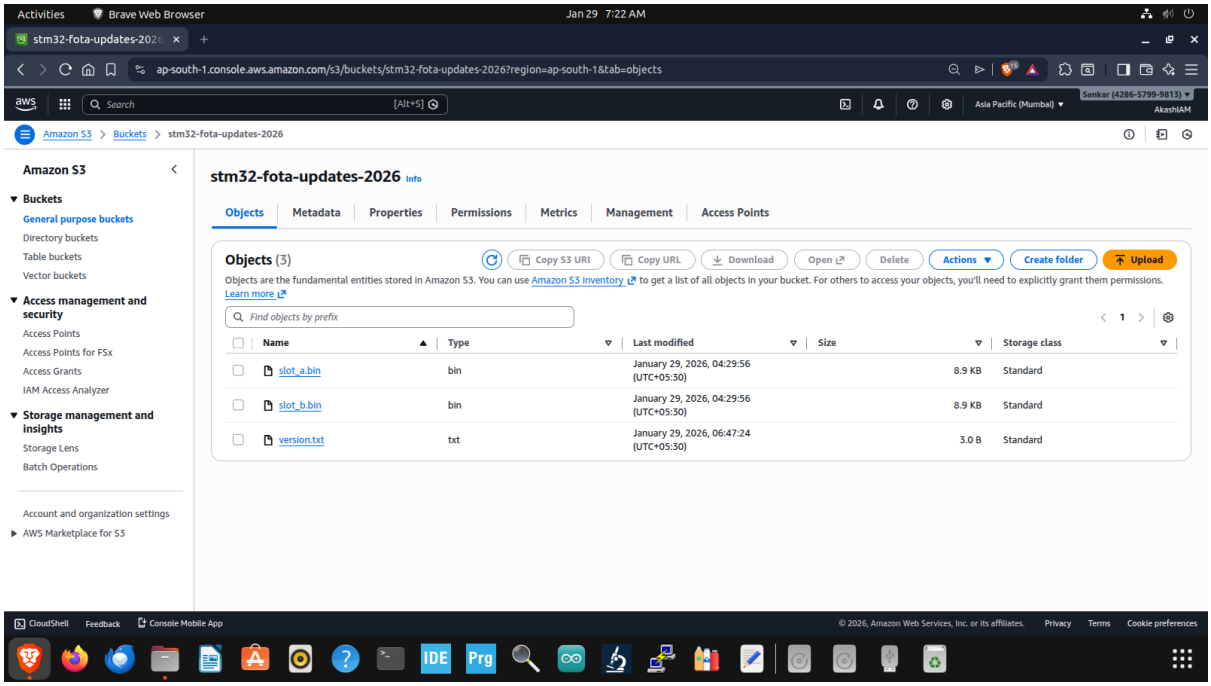
In this step, **Amazon S3** is used as the **cloud-based firmware repository** for the FOTA system. A public S3 bucket (stm32-fota-updates-2026) is configured to store firmware binaries and version information required for update management.

The bucket contains the following key objects:

- **slot_a.bin** – Firmware image intended for Application Slot A
- **slot_b.bin** – Firmware image intended for Application Slot B
- **version.txt** – A lightweight version control file used to indicate the latest available firmware version

When a firmware update is required, the version.txt file is updated with a new version number. This change is detected by the backend update server, which triggers the firmware synchronization process. The ESP32 module subsequently retrieves the firmware images from the server, which are originally sourced from the AWS S3 bucket.

Using AWS S3 provides **high availability, scalability, and reliable global access** to firmware files, enabling seamless cloud-based firmware updates without direct device intervention.



Step 6: Firmware Synchronization from AWS S3 via ESP32

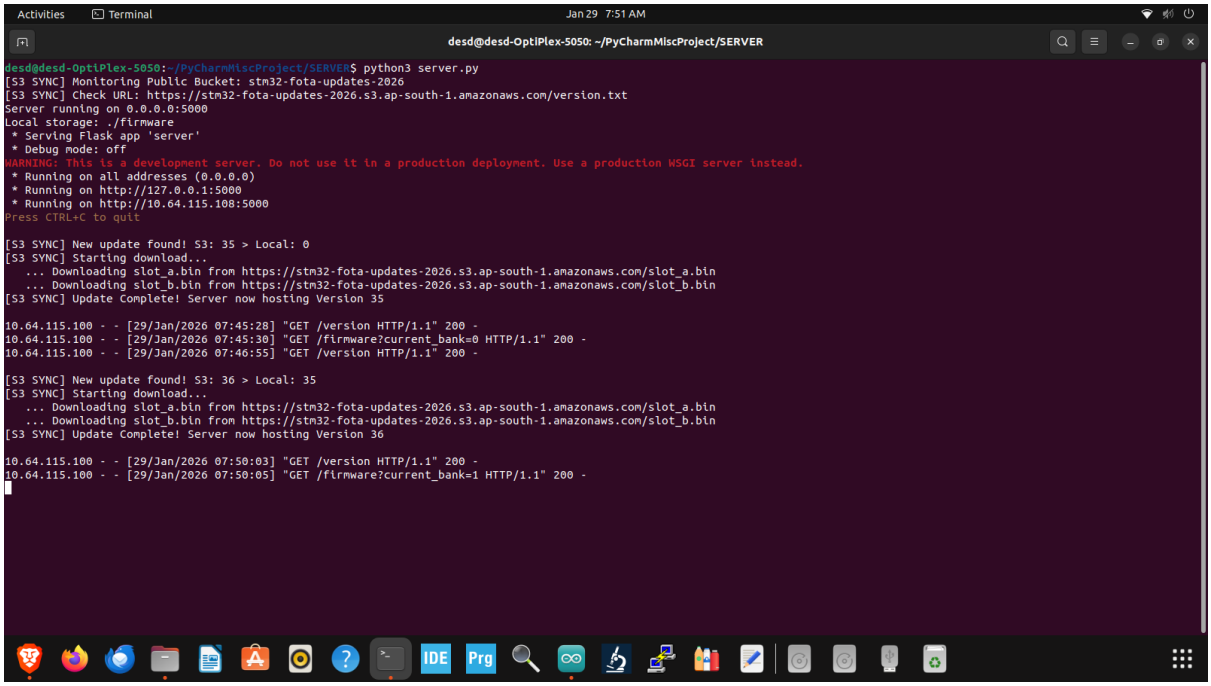
In this step, the **firmware update process is initiated from the cloud** using an **AWS S3 bucket** as the firmware repository. A Python-based **Flask server** is executed to continuously monitor the public S3 bucket for firmware updates by periodically checking the version.txt file.

When a new firmware version is detected, the server automatically downloads the corresponding firmware binaries (slot_a.bin and slot_b.bin) from the AWS S3 bucket and stores them locally. The server then exposes REST API endpoints over HTTP, allowing the **ESP32 module** to query the latest firmware version and request the appropriate firmware slot based on the current active bank.

The terminal logs confirm:

- Successful detection of new firmware versions (Version 35 and Version 36)
- Correct download of firmware binaries from AWS S3
- Proper handling of ESP32 HTTP GET requests for version and firmware data

This step enables **cloud-based Firmware Over-The-Air (FOTA)** capability, where the ESP32 acts as a communication bridge between the AWS cloud and the STM32 bootloader, ensuring reliable and scalable firmware delivery without requiring physical access to the device.



Step 7: Firmware Installation Confirmation and Ping-Pong Slot Switching

In this step, the **Firmware Over-The-Air (FOTA) update process is completed and verified**, confirming successful firmware installation, slot switching, and system recovery.

Once the ESP32 completes downloading the firmware from the update server, it transfers the firmware to the STM32 using the **SPI interface**. The serial logs show real-time update progress from **0% to 100%**, confirming reliable data transmission without interruption. After the entire firmware image is transferred, an **END_OTA command** is issued to notify the STM32 bootloader that the update is complete.

The STM32 bootloader then:

- Stores update metadata (firmware version and slot status)
- Marks the newly written slot as **valid**
- Automatically reboots the system

After reboot, the **ping-pong mechanism** becomes active. The bootloader switches execution from the previously active slot to the newly updated slot (Slot A ↔ Slot B). This ensures that firmware updates are applied **only after successful validation**, maintaining system reliability.

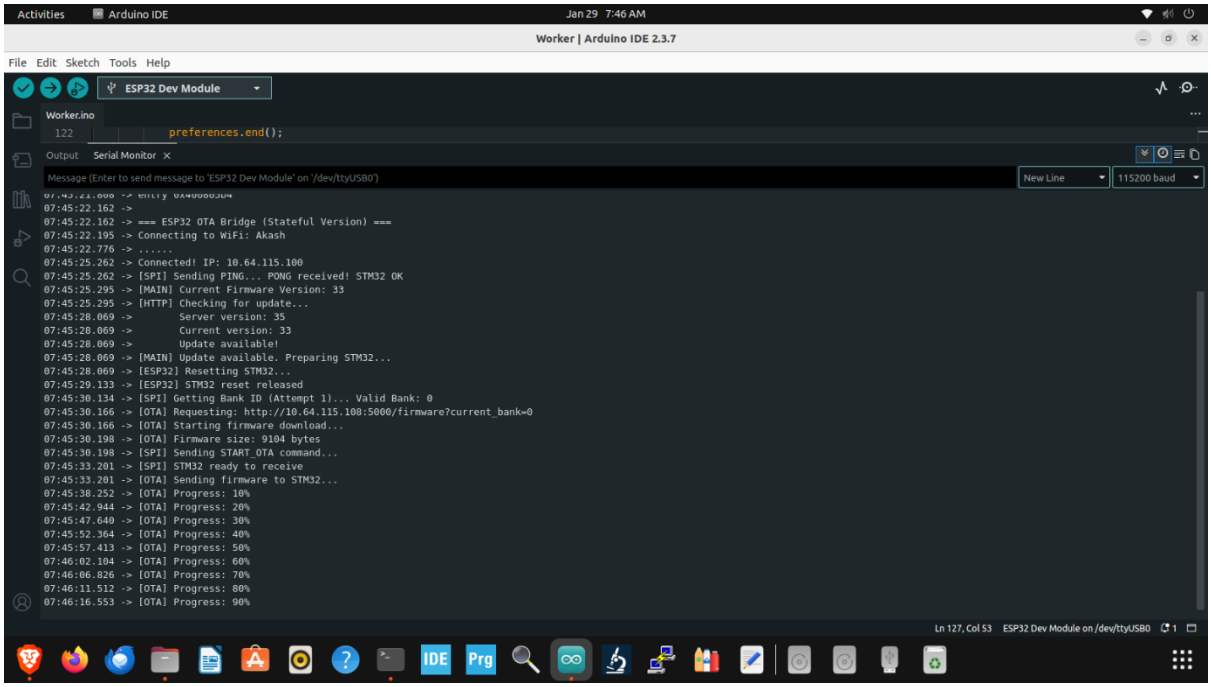
Post-update verification using **STM32CubeProgrammer** confirms:

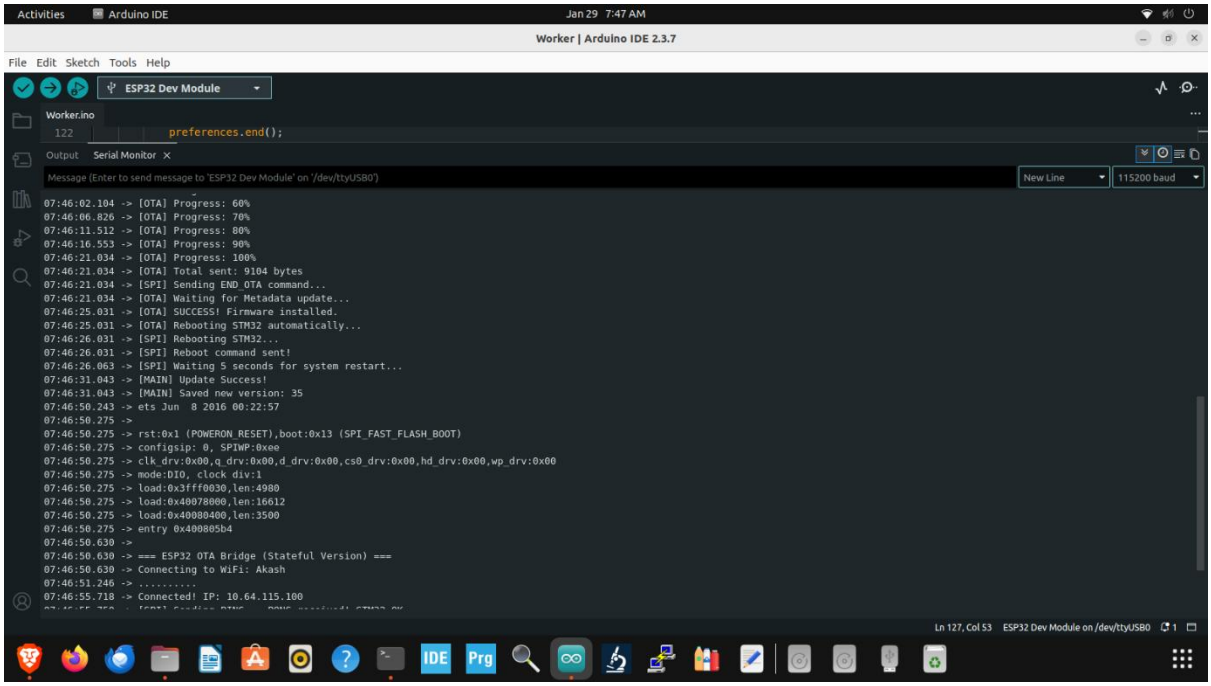
- The Slot B memory region (0x080E0000) contains a valid marker (0xDEADBEEF)
- The slot status flag changes from 0x00000000 to 0x00000001, indicating a **successful firmware commit**
- The previously active slot remains intact as a fallback

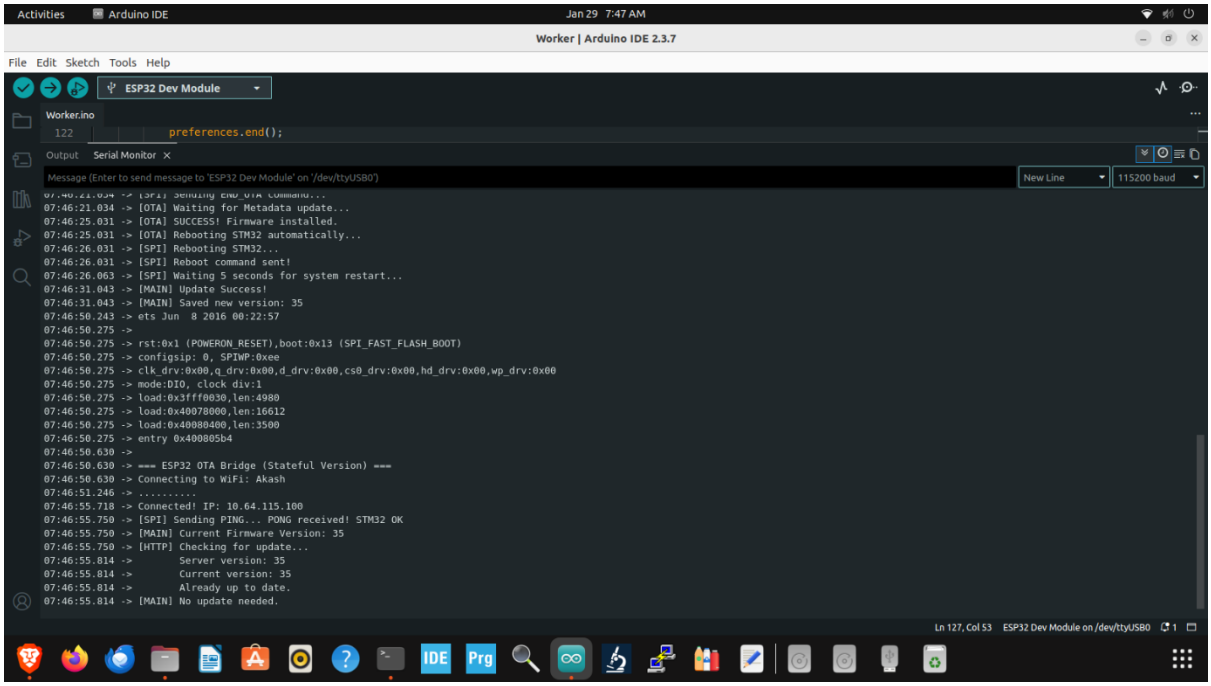
Finally, ESP32 logs confirm:

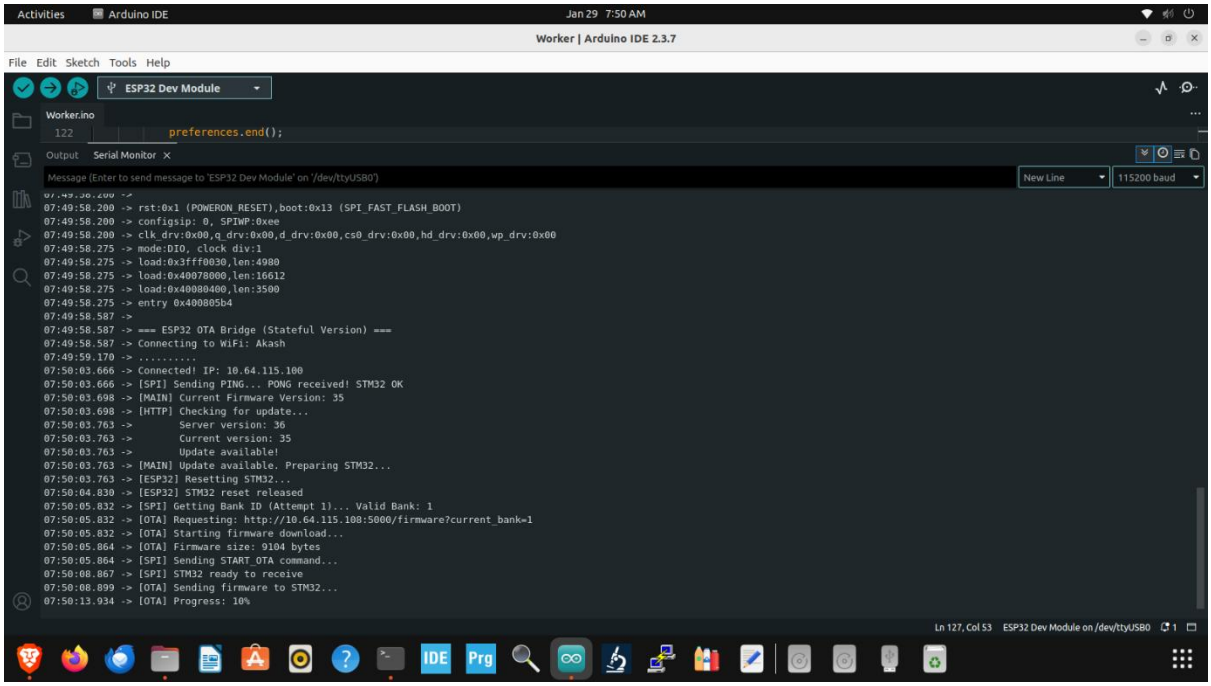
- Updated firmware version saved successfully (Version 35 → Version 36)
- No further updates required when the device rechecks the server

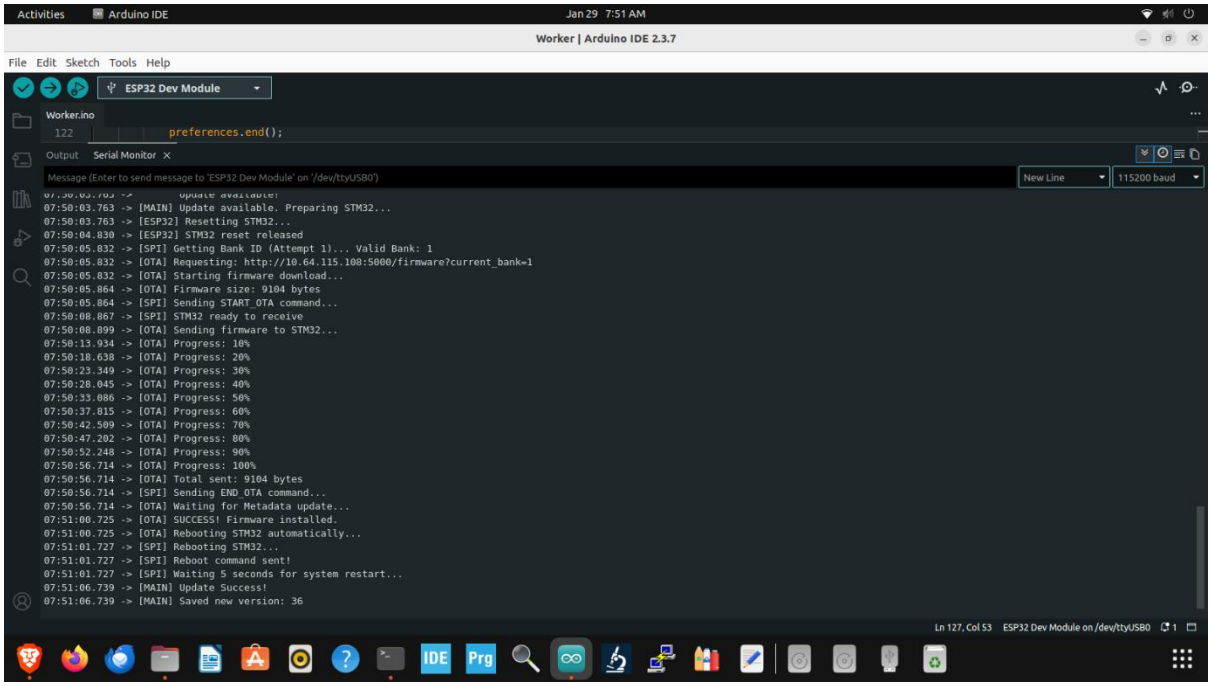
This step validates the **fail-safe FOTA design**, ensuring uninterrupted operation, safe rollback capability, and reliable firmware upgrades even in the event of power loss or communication failure.






Activities
Arduino IDE
Jan 29 7:47 AM
Worker | Arduino IDE 2.3.7
File Edit Sketch Tools Help
ESP32 Dev Module
Worker.ino
122 preferences.end();
Output Serial Monitor
Message (Enter to send message to 'ESP32 Dev Module' on '/dev/ttyUSB0')
New Line
115200 baud
07:46:21.034 -> [OTA] Waiting for Metadata update...
07:46:25.031 -> [OTA] SUCCESS! Firmware installed.
07:46:25.031 -> [OTA] Rebooting STM32 automatically...
07:46:26.031 -> [SPI] Rebooting STM32...
07:46:26.031 -> [SPI] Reboot command sent!
07:46:26.063 -> [SPI] Waiting 5 seconds for system restart...
07:46:31.043 -> [MAIN] Update Success!
07:46:31.043 -> [MAIN] Saved new version: 35
07:46:50.243 -> ets Jun 8 2016 00:22:57
07:46:50.275 ->
07:46:50.275 -> rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
07:46:50.275 -> configsip: 0, SPIWP:0xee
07:46:50.275 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
07:46:50.275 -> mode:DIO, clock div:1
07:46:50.275 -> load:0x3fff0030,len:4980
07:46:50.275 -> load:0x40078000,len:16612
07:46:50.275 -> load:0x40080400,len:3500
07:46:50.275 -> entry 0x400805b4
07:46:50.630 ->
07:46:50.630 -> === ESP32 OTA Bridge (Stateful Version) ===
07:46:50.630 -> Connecting to WiFi: Akash
07:46:51.246 -> .....
07:46:55.718 -> Connected! IP: 10.64.115.100
07:46:55.750 -> [SPI] Sending PING... PONG received! STM32 OK
07:46:55.750 -> [MAIN] Current Firmware Version: 35
07:46:55.750 -> [HTTP] Checking for update...
07:46:55.814 -> Server version: 35
07:46:55.814 -> Current version: 35
07:46:55.814 -> Already up to date.
07:46:55.814 -> [MAIN] No update needed.
Ln 127, Col 53 ESP32 Dev Module on /dev/ttyUSB0


Activities
Arduino IDE
Jan 29 7:50 AM
Worker | Arduino IDE 2.3.7
File Edit Sketch Tools Help
ESP32 Dev Module
Worker.ino
122 preferences.end();
Output Serial Monitor
Message (Enter to send message to 'ESP32 Dev Module' on '/dev/ttyUSB0')
New Line
115200 baud
07:49:58.200 -> rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
07:49:58.200 -> configsip: 0, SPIWP:0xee
07:49:58.200 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
07:49:58.275 -> mode:DIO, clock div:1
07:49:58.275 -> load:0x3fff0030,len:4980
07:49:58.275 -> load:0x40078000,len:16612
07:49:58.275 -> load:0x40080400,len:3500
07:49:58.275 -> entry 0x400805b4
07:49:58.587 ->
07:49:58.587 -> === ESP32 OTA Bridge (Stateful Version) ===
07:49:58.587 -> Connecting to WiFi: Akash
07:49:59.170 -> .....
07:50:03.666 -> Connected! IP: 10.64.115.100
07:50:03.666 -> [SPI] Sending PING... PONG received! STM32 OK
07:50:03.698 -> [MAIN] Current Firmware Version: 35
07:50:03.698 -> [HTTP] Checking for update...
07:50:03.763 -> Server version: 36
07:50:03.763 -> Current version: 35
07:50:03.763 -> Update available!
07:50:03.763 -> [MAIN] Update available. Preparing STM32...
07:50:03.763 -> [ESP32] Resetting STM32...
07:50:04.830 -> [ESP32] STM32 reset released
07:50:05.832 -> [SPI] Getting Bank ID (Attempt 1)... Valid Bank: 1
07:50:05.832 -> [OTA] Requesting: http://10.64.115.100:5000/firmware?current_bank=1
07:50:05.832 -> [OTA] Starting firmware download...
07:50:05.864 -> [OTA] Firmware size: 9104 bytes
07:50:05.864 -> [SPI] Sending START_OTA command...
07:50:08.867 -> [SPI] STM32 ready to receive
07:50:08.899 -> [OTA] Sending firmware to STM32...
07:50:13.934 -> [OTA] Progress: 10%
Ln 127, Col 53 ESP32 Dev Module on /dev/ttyUSB0



7.1 Hardware Platform

The Firmware Over-The-Air (FOTA) system was implemented and experimentally validated using the **STM32F407VG Discovery board**, which is powered by an **ARM Cortex-M4 processor** operating at a maximum frequency of **168 MHz**. This development board provides sufficient computational capability, memory resources, and peripheral support required for implementing a custom bootloader and a dual-application firmware architecture.

The STM32F407VG microcontroller offers adequate **internal Flash memory** for storing the bootloader and two independent application images, along with **SRAM** for runtime operation. Additionally, the board includes **SPI peripherals** used for firmware data transfer and an **on-board ST-LINK debugger**, enabling efficient programming, debugging, and flash memory inspection during development.

Key Hardware Components Used

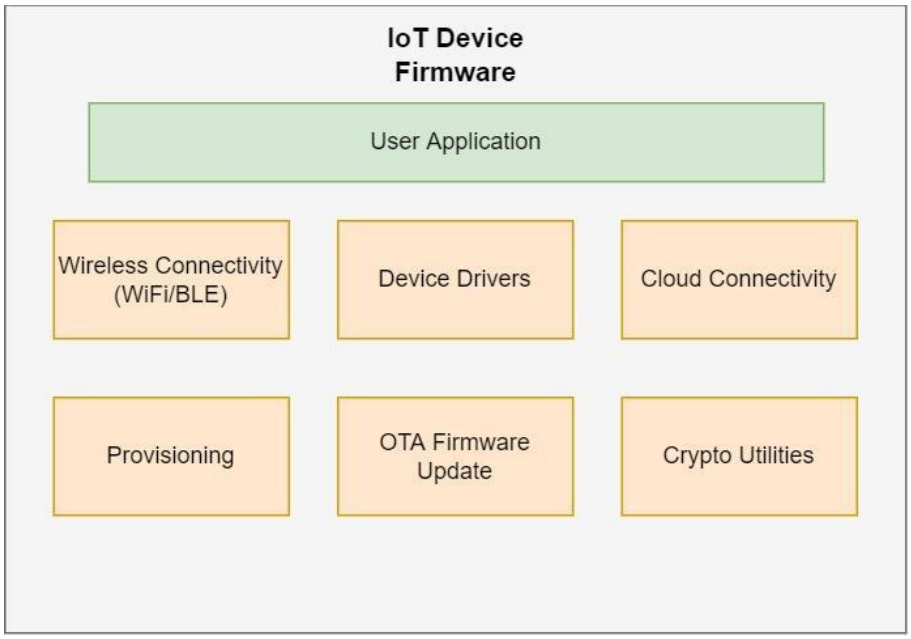
- STM32F407VG microcontroller
- Internal Flash memory for bootloader and dual application storage
- SPI peripheral for firmware transmission
- On-board ST-LINK debugger for programming and debugging

7.2 Software Tools

The following software tools were used during the development and testing of the project:

- **STM32CubeIDE:**
  Used for writing, building, and debugging the bootloader and application firmware. It provides an integrated development environment with compiler, linker, debugger, and HAL configuration support.
- **STM32CubeProgrammer:**
  Used for initial flashing of the bootloader and application binaries into the microcontroller's flash memory. It is also used to verify flash memory contents during testing.
- **STM32 HAL Library:**
  HAL APIs are used for SPI communication, flash erase/program operations, clock configuration, and peripheral initialization. HAL abstraction simplifies development and improves portability.

7.3 Experimental Results



The implemented FOTA system was tested through multiple firmware update cycles. During testing, a new firmware image was transmitted over SPI and written into the secondary application flash region. After successful reception and validation, the bootloader correctly transferred execution to the updated application firmware.

The following results were observed:

- Successful firmware reception over SPI in fixed-size packets

- Correct flash erase and write operations using HAL APIs
- Reliable switching between application slots
- Safe fallback to the previous application in case of invalid firmware

The system consistently avoided execution of incomplete or corrupted firmware, demonstrating the robustness of the dual-application bootloader design.

8. TESTING AND VALIDATION

Testing and validation are critical phases in the development of a Firmware Over-The-Air (FOTA) system, as any failure during the firmware update process can lead to permanent device malfunction. The implemented FOTA system was thoroughly tested on the **STM32F407 Discovery board** to verify correct functionality, robustness, and fault tolerance.

8.1 Functional Testing

Functional testing was performed to verify the correct operation of the bootloader, SPI communication, flash memory programming, and application switching mechanism. The following test cases were executed:

- **Bootloader Execution Test:**
  On system reset, the custom bootloader was verified to execute first and initialize the required peripherals correctly.
- **SPI Firmware Reception Test:**
  Firmware data was transmitted over SPI in predefined packet sizes. The bootloader successfully received the data without loss or corruption.
- **Flash Write and Erase Test:**
  Flash sectors allocated for the secondary application were erased and programmed using STM32 HAL APIs. Written data was verified by reading back flash contents.
- **Application Switching Test:**
  After successful firmware validation, control was correctly transferred from the bootloader to the updated application firmware.

8.2 Failure Handling and Recovery Testing

To validate the reliability of the dual-application architecture, multiple failure scenarios were intentionally introduced:

- **Power Failure During Update:**
  Power was interrupted during firmware reception and flash programming. Upon restart, the bootloader detected the incomplete update and safely reverted to the previously valid application.
- **Invalid Firmware Image Test:**
  Corrupted firmware data was intentionally transmitted. The bootloader correctly identified the invalid image and prevented execution of the corrupted firmware.

- **SPI Communication Error Test:**
  SPI transmission errors were simulated to observe system behaviour. The bootloader handled communication failures gracefully without affecting the existing application firmware.

8.3 Validation Results

The validation results confirm that the implemented FOTA system meets the reliability requirements of embedded firmware update mechanisms. The dual-application architecture ensured that at least one valid firmware image was always available for execution. Flash memory operations were performed safely, and the system consistently recovered from abnormal update conditions.

The successful execution of all test cases demonstrates that the proposed FOTA solution is robust, fault-tolerant, and suitable for deployment in real-world embedded systems.

9. CONCLUSION

This project successfully demonstrated the **design and implementation of a reliable Firmware Over-The-Air (FOTA) update system for the STM32F407VG microcontroller** using a **custom bootloader and dual-application architecture**. The primary objective of enabling firmware updates without physical access while maintaining system stability and fault tolerance was effectively achieved.

A custom bootloader was developed and placed in a protected region of the internal flash memory. The bootloader was designed to execute immediately after system reset and manage all firmware update-related operations, including firmware reception over **SPI communication**, flash memory programming, firmware validation, and controlled application switching. The use of **STM32 HAL APIs** ensured safe and portable implementation of peripheral initialization and flash operations.

The dual-application firmware architecture played a crucial role in enhancing system reliability. By maintaining two separate application slots, the system ensured that a valid firmware image was always available for execution. This approach effectively prevented device bricking in scenarios such as power failure, communication interruption, or corrupted firmware updates. The bootloader consistently reverted to the previously verified application when update failures were detected.

Extensive testing and validation were carried out on the **STM32F407 Discovery board**. Functional tests verified correct bootloader execution, SPI-based firmware reception, flash erase/write operations, and application switching. Failure scenario testing further confirmed the robustness of the system by demonstrating safe recovery from abnormal conditions.

Overall, the project provides a practical and scalable FOTA solution suitable for STM32-based embedded systems. The implemented architecture and methodologies can be extended to support additional features such as enhanced security, wireless communication, and advanced update strategies, making it relevant for modern embedded and IoT applications.