ABSTRACT

Bootloader is a small program executed when the microcontroller boots and contains the basic functionalities. It is a special operating system software that loads into working memory of a computer after start-up, this is the first piece of code that runs when you press the reset button . The bootloader is also used to update the firmware.

In this project , we are making the custom bootloader which would update the firmware over the air. We are using STM32F407 board as target and ESP8266 as host which will be connected to AWS(Amazon Web Services) server and will download the binary file. This downloaded file will be sent to target device that is STM32F407 via the UART transmission using the XMODEM protocol and will be downloaded in the download area of target .Further, the target will check the downloaded update in the download area and thereby update the user code.

Contents

List Of Figures

CHAPTER 1

INTRODUCTION

In this project we propose to develop an experimental setup for updating/flashing the firmware of STM32F407VG via Over The Air (OTA) Boot-Loader. The cloud server has the firmware/application binaries, this binaries are sent to ESP8266. The system(ESP8266) automatically detects the updates and notifies about the update available on the server and sends the updated firmware/application to STM32F407VG, UART based communication is used in this model which is internally using XMODEM(FTP) protocol.

1.1 Overview

Modern electronic devices are complex hence improved diagnosis and reprogramming methods are currently researched and tested, in this context a flash Boot-Loader is the core of such approaches because it verifies the firmware's/software's validity at the time of reset and waits for an external reprogramming request. One such reprogramming method is OTA Over The Air, this method is being used for pushing application/software(OS) updates for mobile devices like (Android/Ios), so if we can implement this for updating the firmware/application for microcontroller then this would be an efficient way for updating/flashing the Boot-Loader firmware/application. The partial fulfilment of above method is described in this report OTA Boot-Loader flasher/updater using cloud server for pushing the updates.

1.2 Block Diagram

The proposed system consists of a STM32F407VG which will be the host which has the Boot-Loader in its FLASH Memory, the work of updating this boot loader firmware will be done by the ESP8266 which has the updated firmware/application downloaded in it via a cloud server.

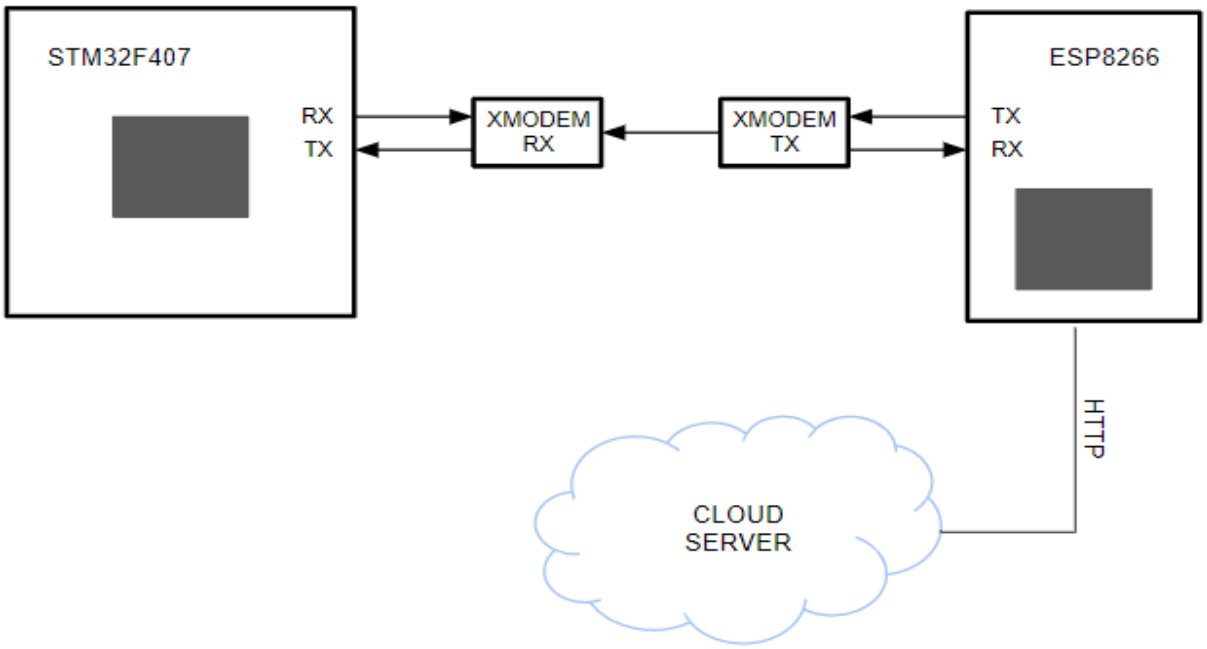

Figure 1.1 Block Diagram of Proposed System

1.3 Hardware Setup

The system is based on CortexM3 based STM32F407VG microcontroller board. This system is interfaced with ESP8266 via UART. In this context the ESP8266 is the HOST device which will send the downloaded files present in its flash memory to the TARGET device which is STM32F407VG. This transaction of files is done in the form of packets using the XMODEM protocol.

1.3.1 UART

Universal Asynchronous Receiver Transmitter (UART) is a serial transmission protocol which is used to communicate between two or more devices, UART 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.

1.4 Working Principle

The working of this model is given in the below flow diagram :-

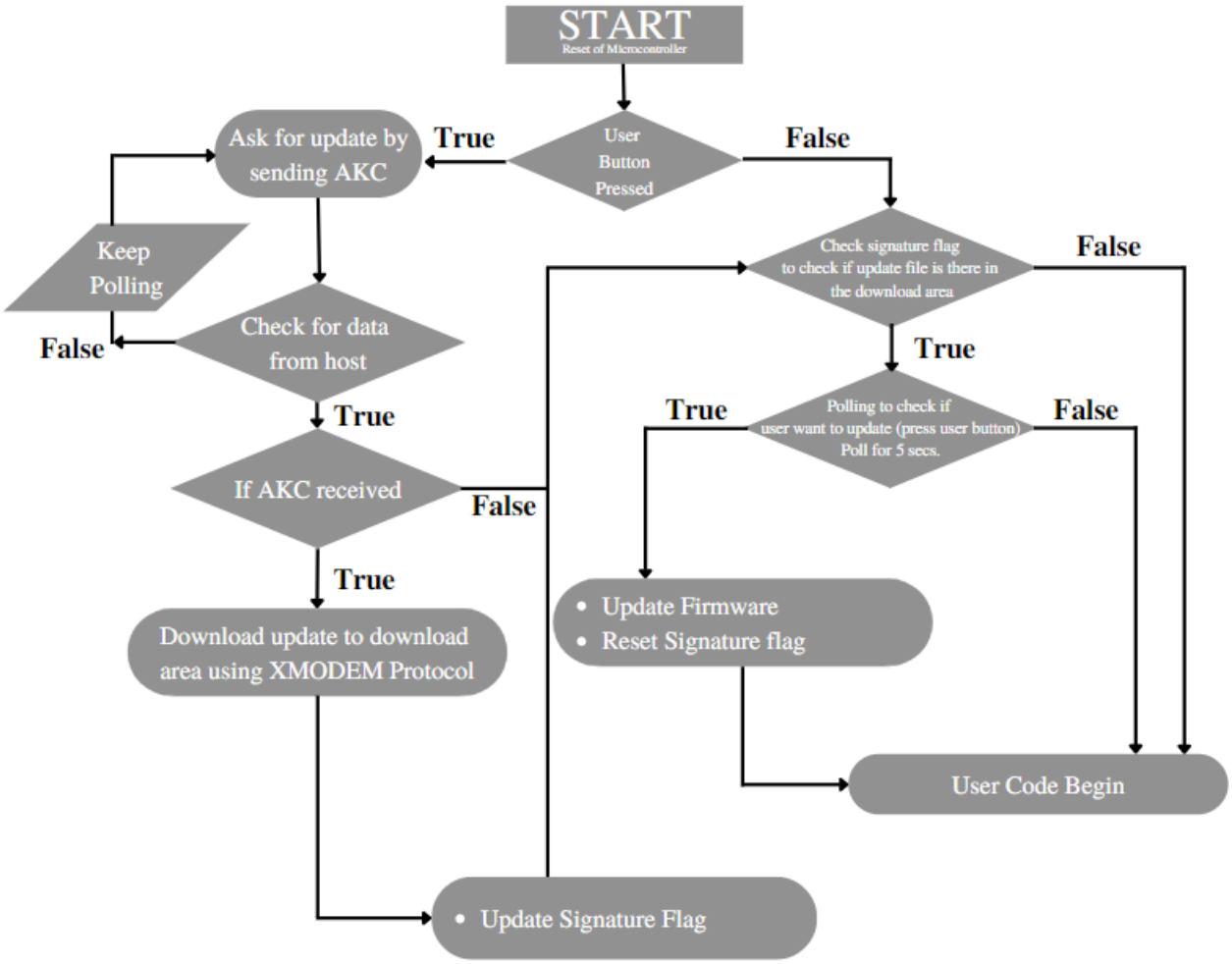

Figure 1.2 STM32F407VG Working Flow

The working flow of ESP8266 is as described :-

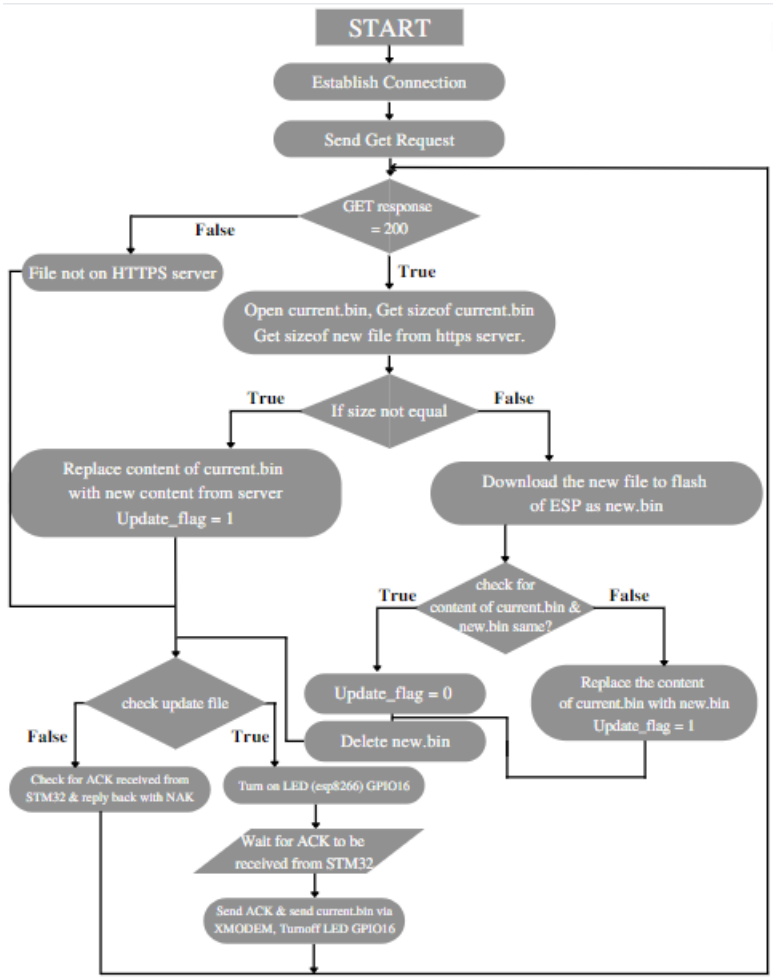

Figure 1.3 ESP8266 Working Flow

CHAPTER 2

LITERATURE SURVEY

2.1 General

A bootloader is a small program executed when the microcontroller boots and contains the basic functionalities.

The STM32 internal bootloader have the functionality of UART programming, but not USB programming. To solve this problem, there are another bootloaders that are stored on program memory (remember that bootloader is Read Only). Those bootloaders have the functionality of use the internal USB, converting it into a programmer, similar to UART programmer.

2.2 STM Standard Boot-Loader

The standard ('factory', 'native', 'STM') bootloader is always available -- being stored in read-only memory -- and cannot be modified or deleted. You can access it by configuring the boot pins high or low, and then powering (or resetting) the MCU. Different STM32 device families offer different capabilities in their factory bootloaders. For example, some support uploading firmware over USB while others do not. An example of the latter is the popular F103 series.

2.3 Maple Boot-Loader

Maple is based off the STM32 (ARM cortex M3) series chips, which do have embedded USB support. Thus, Maple doesn’t need the extra FTDI chip. Firmware is uploaded via the standard DFU protocol (also used by iPhone and openMoko). Since DFU is a standard, there is no need for custom software running on the host to upload the firmware. Any DFU compliant program will work. The Maple IDE is based around **dfu-util**, openMoko’s DFU utility.

CHAPTER 3

SYSTEM DESIGN

This chapter explains in detail about hardware and software that are being used and their features and application.

3.1 STM32F407VG Microcontroller

The STMF407VG ARM based microcontroller, which is a high-performance microcontroller. This board allows users to develop and design applications. It has multiple modules within itself which allows the user to communicate and design the interface of different kinds without relying on any third device. The board has all the modern system modules peripherals like DAC, ADC, audio port, UART, etc which makes it one of the best-developing devices. The device may be for developing modern applications but some protocols will need to be followed to use the device, like the compiler, voltage potential, etc.

The STM32F407xx family is based on the high-performance ARM® Cortex®-M4 32-bit RISC core with FPU operating at a frequency of up to 72 MHz, and embedding a floating-point unit (FPU), a memory protection unit (MPU) and an embedded trace macro cell (ETM). The family incorporates high-speed embedded memories (up to 1 Mbytes of Flash memory, up to 192 Kbytes of RAM) and an extensive range of enhanced I/O ’s and peripherals connected to two APB buses. They also feature standard and advanced communication interfaces: up to two I2Cs, up to three SPIs (two SPIs are with multiplexed full-duplex I2Ss), three USARTs, up to two UARTs, CAN and USB. To achieve audio class accuracy, the I2S peripherals can be clocked via an external PLL. The STM32F303xB/STM32F303xC family operates in the -40 to +85°C and -40 to +105 °C temperature ranges from a 2.0 to 3.6 V power supply. A comprehensive set of power-saving mode allows the design of low-power applications.

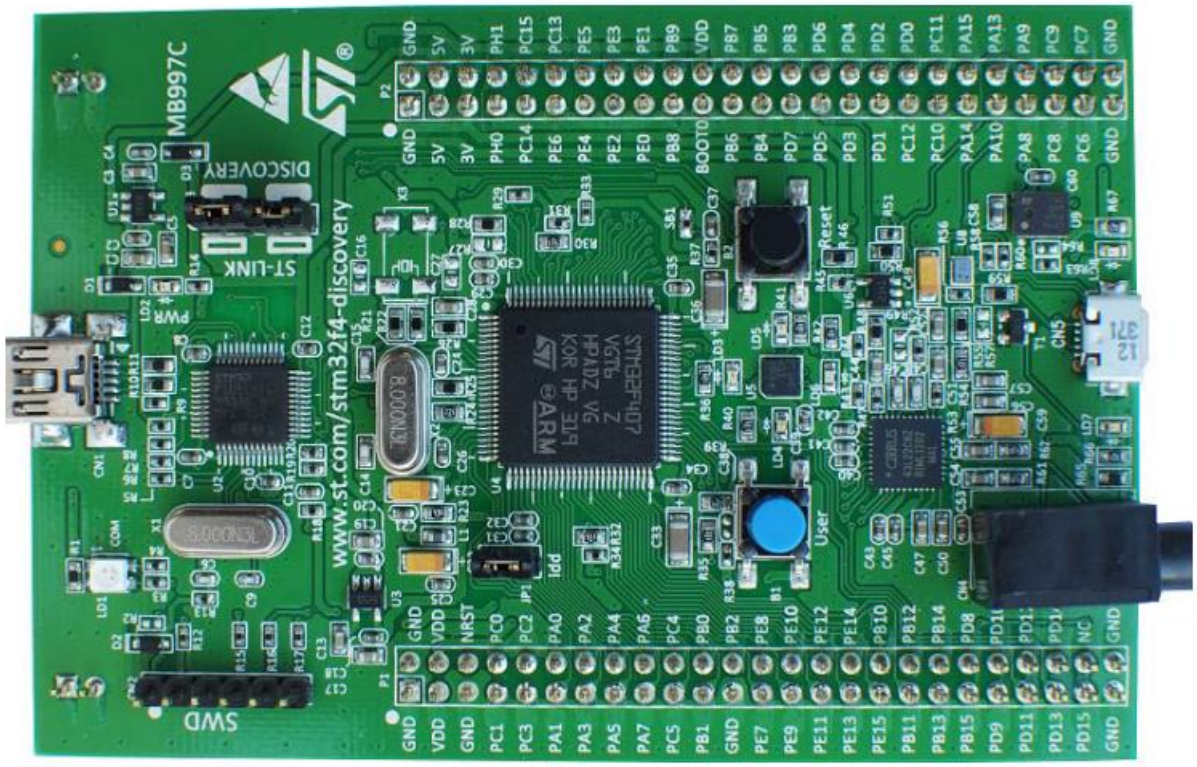

Figure 3.1 STM32F407VG Microcontroller Board

3.1.2 STM32F4 Oscillator Pins

STM32F4 may be larger and smarter but it doesn't have any internal crystal for clock pulse. It has four external clock pulse pins two they are used for 32KHz crystal and the other two are used for High-frequency crystal. The crystal can be used up to 50MHz on that pin but after 25MHz it becomes tricky to handle the crystal. These crystal pins can be used as GPIO, so it should be instructed within the program about the use of an oscillator. Both oscillators can't be used at the same time and both are in the same header P2.

- OSC (IN) – GPIO7
- OSC (OUT) – GPIO8
- OSC RTC (IN) – GPIO9
- OSC RTC (OUT) – GPIO10

3.1.3 STM32F4 UART

This application note describes how to implement an emulated universal asynchronous receiver transmitter (UART) on microcontrollers of STM32F4 Series. Such an emulation is needed in applications that require more UARTs than the ones available on STM32F4 microcontrollers. The emulated UART is full-duplex, supports up to 9 data bits and baud rates up to 115200 bps. It also offers high flexibility, as any I/O pin can be used as TX or RX line. In addition, this UART emulation uses DMA to minimize CPU usage.

The main features of the UART emulator are the following

- Full-duplex asynchronous communications up to 115200 bps
- Programmable data word length: from 5 to 9 bits
- Flexible GPIO usage: all GPIOs can be configured as UART_TX/RX
- Configurable number of stop bits: 1 or 2 stop bits
- Reception:
  - Frames are received at once
  - To receive new frames the UART has to be reinitialized
- Parity control
  - Transmission of parity bit
  - Parity check of received data frame
- Transfer detection flags
  - Receive complete
  - Transmit complete
- Error detection flags
  - Frame error
  - Parity error

The below given diagram describes the UART communication frame :-

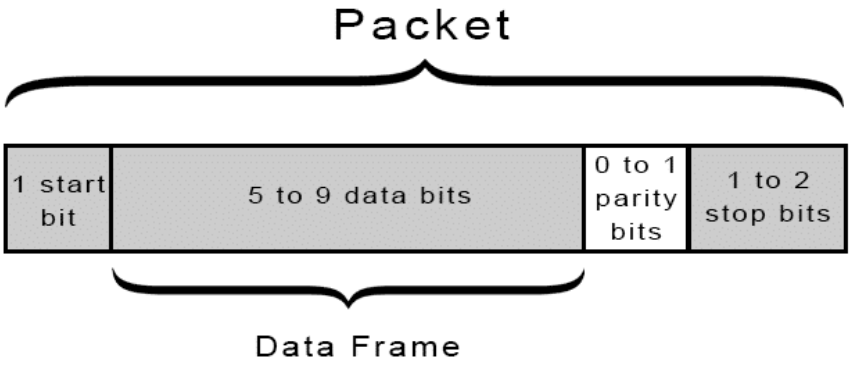

Figure 3.2 UART Communication Frame

In this model we have used the UART communication to communicate between the Target device (STM32F407VG) and the Host device (ESP8266) using the XMODEM protocol. In this model we are using the UART5 as the main communication port which is using XMODEM protocol internally and UART4 for debug messages the host device sends the updated firmware/application files to the target using aforementioned method. The pins used for UART config are as follows :-

UART5 :- UART5_TX : PC12, UART5_RX : PD2 and for UART4 the pins used are UART4 :- UART4_TX : PC10, UART4_RX : PC1.

The diagram given below gives a brief idea about this :-

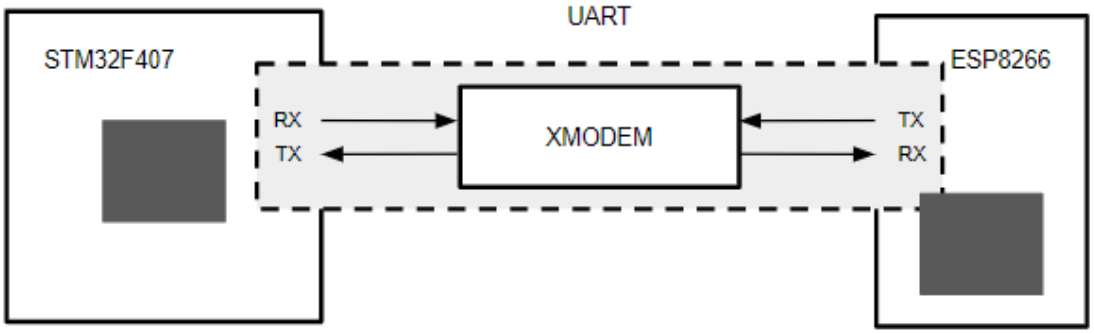

Figure 3.3 UART Communication using XMODEM

3.1.4 STM32F407VG Board GPIO Pins

There are a total of six ports (A, B, C, D, E, H) in the device and all of them come with an internal pull-up resistor and can be used for input/output function. Those pins also support some other functions and it can be controlled using programming. All I/O pins are given below.

In P1 Header: GPIO7 – GPIO22, GPIO24 – GPIO47

In P2 Header: GPIO7 – GPIO21, GPIO23 – GPIO48

From the above mentioned GPIO port and pins we are using

GPIO Port D pins :- PD12, PD13, PD14, PD15. GPIO_OUTPUT mode.

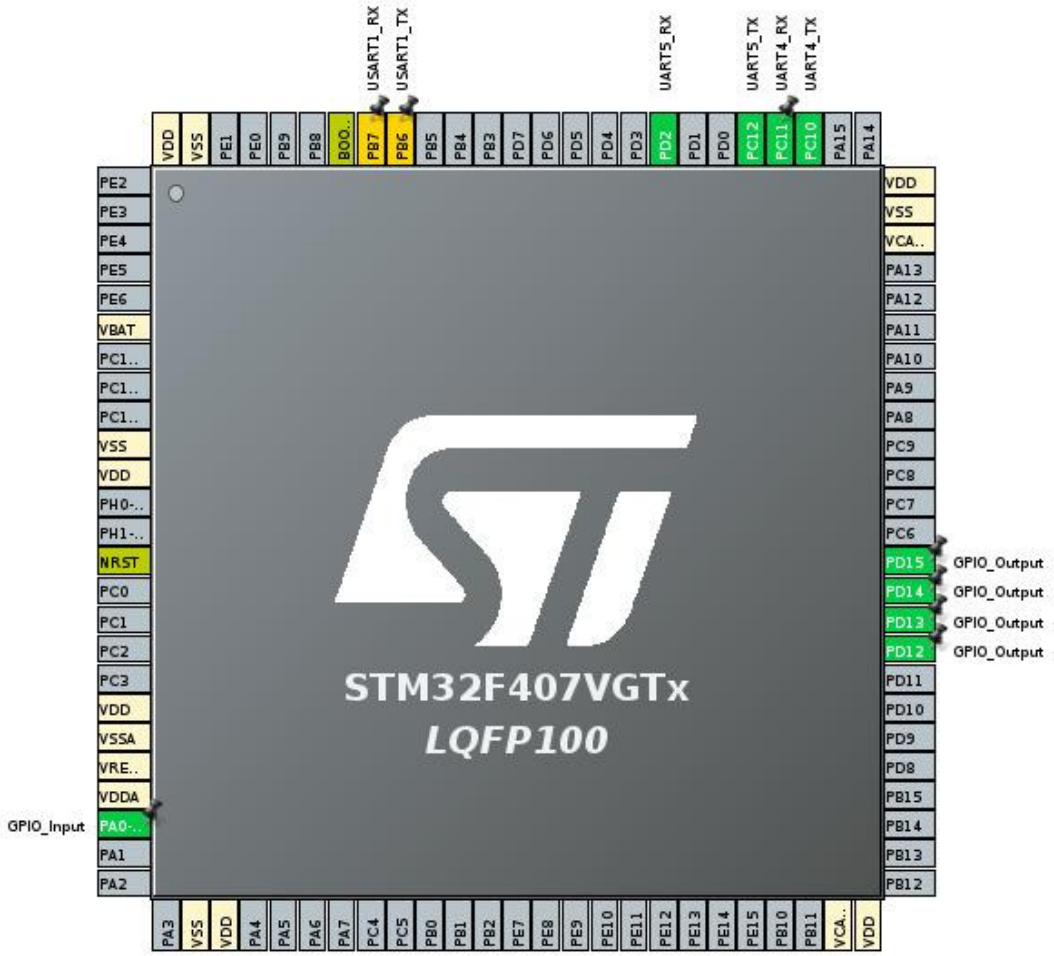

Figure 3.4 STM32F407VG Pin Config

3.1.5 STM32F407VG Memory Aliasing

Reset Sequence

When you reset the microcontroller, the Program Counter (PC) of the processor is loaded with the value 0x00000000, The processor reads the value @ memory location 0x00000000 into the Main Stack Pointer (MSP) so the MSP has the value at the address 0x00000000, after that processor reads the value @ memory location 0x00000004 into the PC (This value is actually the address of RESET HANDLER), then PC jumps to rest handler, then from reset handler the main function is called.

All ARM Cortex M based microcontroller's after reset does

- Load value @memory address 0x00000000 into MPS
- Load value @memory address 0x00000004 into PC (value = &reset_handler)

In STM32F4xx microcontroller

- MPS value stored @memory address 0x08000000
- Vector table starts from memory address 0x08000004 (address of reset handler found at this location).

The below diagram shows the reallocation and memory aliasing :-

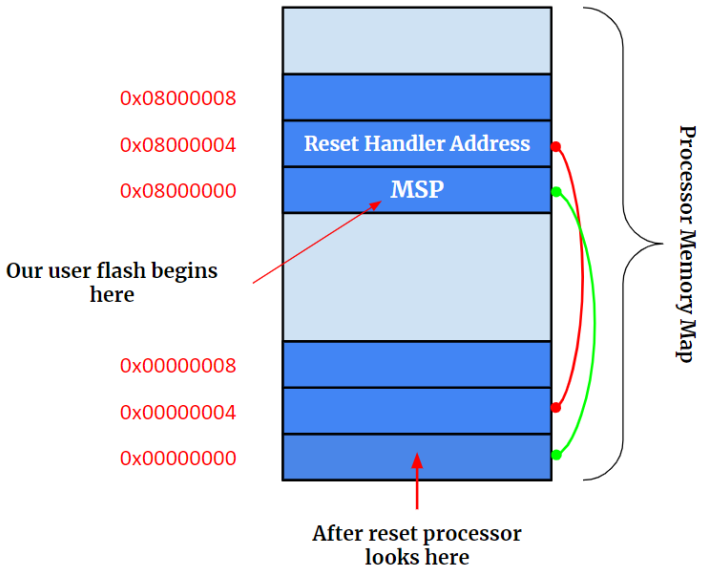

Figure 3.5 Memory Aliasing

3.2 Embedded Flash Memory Interface

3.2.1 Introduction

The Flash memory interface manages CPU AHB I-Code and D-Code accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms. The Flash memory interface accelerates code execution with a system of instruction prefetch and cache lines.

3.2.2 Main Features

- Flash Memory read operation.
- Flash memory program/erase operation.
- Read/Write protections.
- Prefetch on I-Code.
- 64 cache line of 128-bits on I-Code.
- 8 cache lines of 128-bits on D-Code.

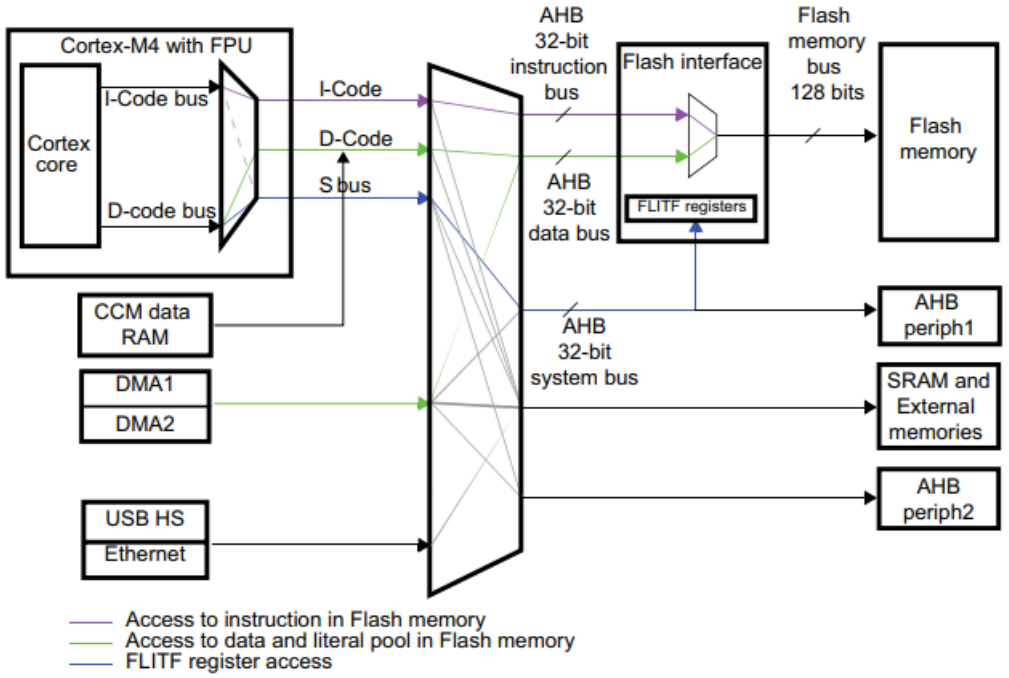

Figure 3.6 Flash Memory Interface Connection Inside System Architecture.

3.2.3 Embedded Flash memory in STM32F407xx

Flash Memory has the following main features :

- Capacity upto 1-Mbyte.
- 128-bits wide data read.
- Byte, half-word, word, double-word write.
- Sector and Mass erase.
- Memory organization
  - The Flash Memory is organized as follows :
    - A main memory block divided into 4 sectors of 16 Kbytes, 1 sector of 64 Kbytes, and 7 sectors of 128 Kbytes.
    - System Memory from which the device boots in System memory boot mode.
    - 512 (OTP) one-time-programmable bytes for user data. The OTP area contains 16 additional bytes used to lock the corresponding OTP data block.
    - Option bytes to configure read and write protection, BOR level, watchdog software/hardware and reset when the device is in standby or Stop mode.
- Low-power modes.

Block	Name	Block base addresses	Size
Main memory	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
	Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
	Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
	Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbytes
	Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
	Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
	Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
	. . .	. . .	. . .
	Sector 11	0x080E 0000 - 0x080F FFFF	128 Kbytes
System memory		0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP area		0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
Option bytes		0x1FFF C000 - 0x1FFF C00F	16 bytes

Figure 3.7 Flash Module Organization (STM32F4xx)

3.2.4 Flash Interface Registers

Flash Key Register (FLASH_KEYR)

The flash key register is used to allow access to the Flash Control Register and so, to allow programs and erase operations.

To unlock Flash Control Register (FLASH_CR) write

Key1 = 0x45670123 in the FLASH_KEYR

Key2 = 0xCDEF89AB in the FLASH_KEYR

Flash Status Register (FLASH_SR)

The Flash Status Register gives information about ongoing program and erase operations.

Flash Control Register (FLASH_CR)

The Flash Control Register is used to configure and start Flash Memory operations.

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	FLASH_ACR	Reserved																			DCRST	ICRST	DCEN	ICEN	PRFTEN	Reserved						LATENCY [2:0]		
	Reset value																				0	0	0	0	0							0	0	0
0x04	FLASH_ KEYR	KEY[31:16]																KEY[15:0]																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	FLASH_OPT KEYR	OPTKEYR[31:16]																OPTKEYR[15:0]																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	FLASH_SR	Reserved															BSY	Reserved								PGSERR	PGPERR	PGAERR	WRPERR	Reserved		OPERR	EOP
	Reset value																0									0	0	0	0			0	0
0x10	FLASH_CR	LOCK	Reserved							EOPIE	Reserved						STRT	Reserved					PSIZE[1:0]		Reserved	SNB[3:0]				MER	SER	PG	
	Reset value	1								0							0						0	0		0	0	0	0		0	0	
0x14	FLASH_ OPTCR	Reserved				nWRP[11:0]												RDP[7:0]								nRST_STDBY	nRST_STOP	WDG_SW	Reserved	BOR_LEV[1:0]		OPTSTRT	OPTLOCK
	Reset value					1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	1	1		1	1	0	1

Figure 3.8 Flash Register Map.

3.2.5 HAL FLASH Generic/Extension Driver

Programming operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations. This section contains the following APIs :

- HAL_FLASH_Program()
- HAL_FLASH_Program_IT()
- HAL_FLASH_IRQHandler()
- HAL_FLASH_EndOfOperationCallback()
- HAL_FLASH_OperationErrorCallback()

Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations. This section contains the following APIs :

- HAL_FLASH_Unlock()
- HAL_FLASH_Lock()
- HAL_FLASH_OB_Unlock()
- HAL_FLASH_OB_Lock()
- HAL_FLASH_OB_Launch()

Extended programming operation functions

This subsection provides a set of functions allowing to manage the Extension FLASH programming operations. This section contains the following APIs:

- HAL_FLASHEx_Erase()
- HAL_FLASHEx_Erase_IT()
- HAL_FLASHEx_OBProgram()
- HAL_FLASHEx_OBGetConfig()
- HAL_FLASHEx_AdvOBProgram()
- HAL_FLASHEx_AdvOBGetConfig()
- HAL_FLASHEx_OB_SelectPCROP()
- HAL_FLASHEx_OB_DeSelectPCROP()
- HAL_FLASHEx_OB_GetBank2WRP()

3.3 XMODEM

3.3.1 Introduction

The Xmodem protocol was created years ago as a simple means of having two computers talk to each other. With its half-duplex mode of operation, 128- byte packets, ACK/NACK responses and CRC data checking, the Xmodem protocol has found its way into many applications. In fact most communication packages found on the PC today have a Xmodem protocol available to the user.

3.3.2 Theory Of Operation

Xmodem is a half-duplex communication protocol. The receiver, after receiving a packet, will either acknowledge (ACK) or not acknowledge (NAK) the packet. The CRC extension to the original protocol uses a more robust 16-bit CRC to validate the data block and is used here. Xmodem can be considered to be receiver driven. That is, the receiver sends an initial character “C” to the sender indicating that it’s ready to receive data in CRC mode. The sender then sends a 133-byte packet, the receiver validates it and responds with an ACK or a NAK at which time the sender will either send the next packet or re-send the last packet. This process is continued until an EOT is received at the receiver side and is properly ACKed to the sender. After the initial handshake the receiver controls the flow of data through ACKing and NAKing the sender.

Byte 1	Byte 2	Byte 3	Byte 4-131	Byte 132-133
SOH	Packet No.	Inverse of Packet No.	DATA	16-bit CRC

Figure 3.9 XMODEM CRC Packet Format

3.3.3 Definitions

The following defines are used for protocol flow control.

Symbol	Description	Value
SOH	Start of Header	0x01
EOT	End of Transmission	0x04
ACK	Acknowledge	0x06
NAK	Not Acknowledge	0x15
ETB	End of Transmission Block (Return to Amulet OS mode)	0x17
CAN	Cancel (Force receiver to start sending C's)	0x18
C	ASCII “C”	0x43

Byte 1 of the Xmodem CRC packet can only have a value of SOH, EOT, CAN or ETB anything else is an error. Bytes 2 and 3 form a packet number with checksum, add the two bytes together and they should always equal 0xff. Please note that the packet number starts out at 1 and rolls over to 0 if there are more than 255 packets to be received. Bytes 4 - 131 form the data packet and can be anything. Bytes 132 and 133 form the 16-bit CRC. The high byte of the CRC is located in byte 132. The CRC is calculated only on the data packet bytes (4 - 131).

3.3.4 Synchronization

The receiver starts by sending an ASCII “C” (0x43) character to the sender indicating it wishes to use the CRC method of block validating. After sending the initial “C” the receiver waits for either a 3 second time out or until a buffer full flag is set. If the receiver is timed out then another “C” is sent to the sender and the 3 second time out starts again. This process continues until the receiver receives a complete 133-byte packet.

3.3.5 Receiver Considerations

This protocol NAKs the following conditions: 1. Framing error on any byte 2. Overrun error on any byte 3. Duplicate packet 4. CRC error 5. Receiver timed out (didn't receive packet within 1 second) On any NAK, the sender will re-transmit the last packet. Items 1 and 2 should be considered serious hardware failures. Verify that sender and receiver are using the same baud rate, start bits and stop bits. Item 3 is usually the sender getting an ACK garbled and re-transmitting the packet. Item 4 is found in noisy environments. And the last issue should be self-correcting after the receiver NAKs the sender.

Sender						Receiver
					<----	"C"
						Times Out after 3 Seconds
					<----	"C"
SOH	0x01	0xFE	Data	CRC	---->	Packet OK
					<----	ACK
SOH	0x02	0xFD	Data	CRC	---->	(Line Hit during Data Transmission)
					<----	NACK
SOH	0x02	0xFD	Data	CRC	---->	Packet OK
					<----	ACK
SOH	0x03	0xFC	Data	CRC	---->	Packet OK
(ACK Gets Garbled)					<----	ACK
					<----	ACK
SOH	0x04	0xFB	Data	CRC	---->	(UART Framing Error on Any Byte)
					<----	NACK
SOH	0x04	0xFB	Data	CRC	---->	Packet OK
					<----	ACK
SOH	0x05	0xFA	Data	CRC	---->	(UART Overrun Error on Any Byte)
					<----	NACK
SOH	0x05	0xFA	Data	CRC	---->	Packet OK
					<----	ACK
EOT					---->	Packet OK
					<----	ACK
ETB					---->	Finished
Finished					<----	ACK

Figure 3.10 XMODEM transaction

3.4 HTTPS Cloud Server

AWS S3 bucket service is being used as https cloud server for storage of updated firmware's/application's binary file. A bucket is created named as application.binary with Read_Only permission with public access. Hence the need for a finger print is avoided. The binary file uploaded by the user is stored as an object in the Bucket.

The URL for the same is generated and the host will fetch the file using this URL.

A library named WiFiClientSecureBearSSL.h services are used in ESP8266. This library facilitate the use of SSL certificates, for secure server.

Chapter 4

Results

Whenever user button is press then the STM checks for update from ESP8266 and if host (ESP8266) does not have update STM will check in its download area and if download area does not have any update then the user code will be executed.

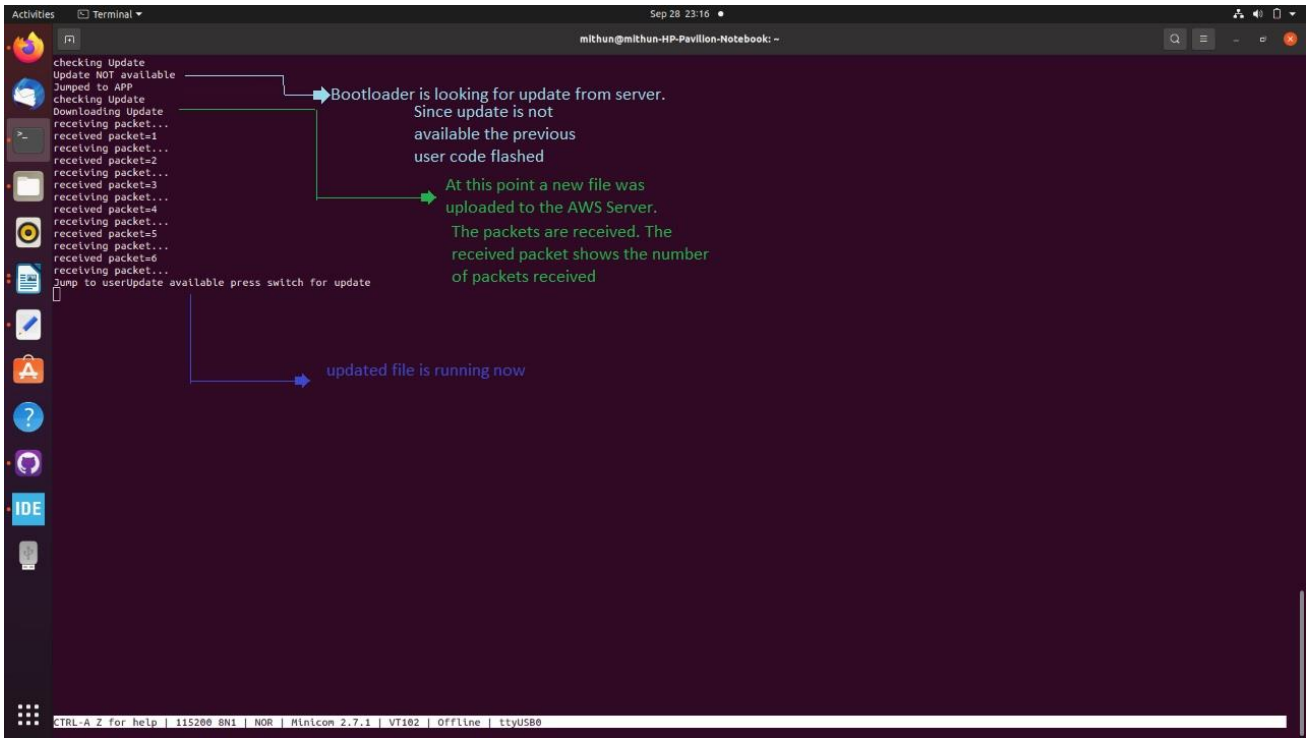

Figure 4.1 Custom Boot-Loader.

If ESP8266 has downloaded the update from the server as described in the following diagram :-

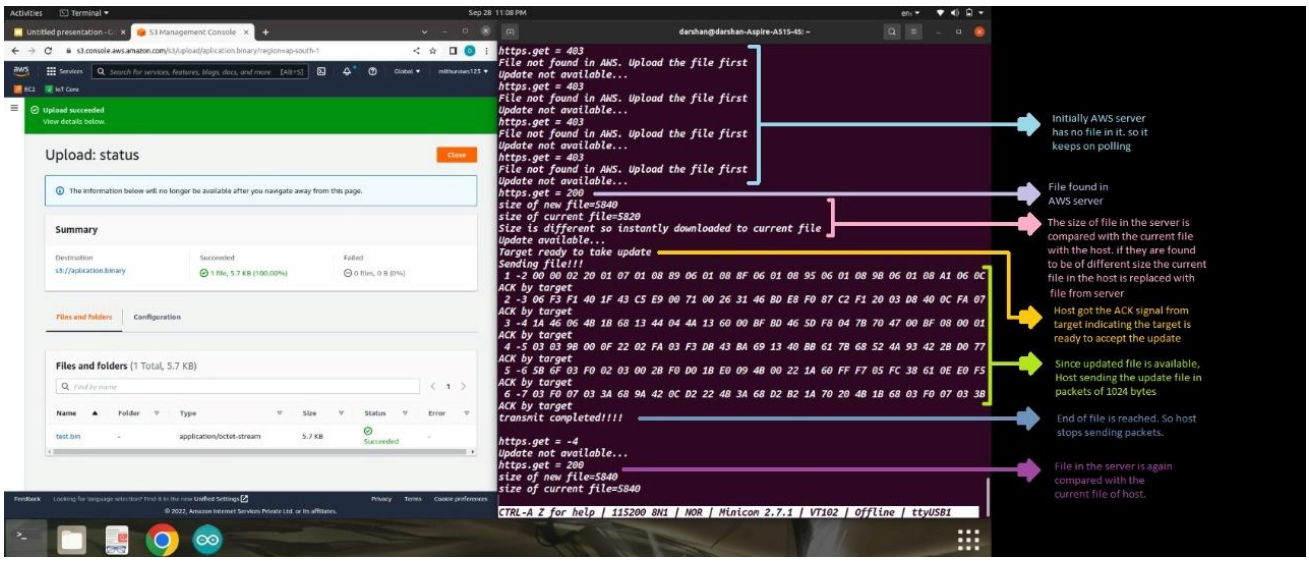

Figure 4.2 Update is available with ESP8266 and send it to STM32F407VG.

And in case when the update is not available

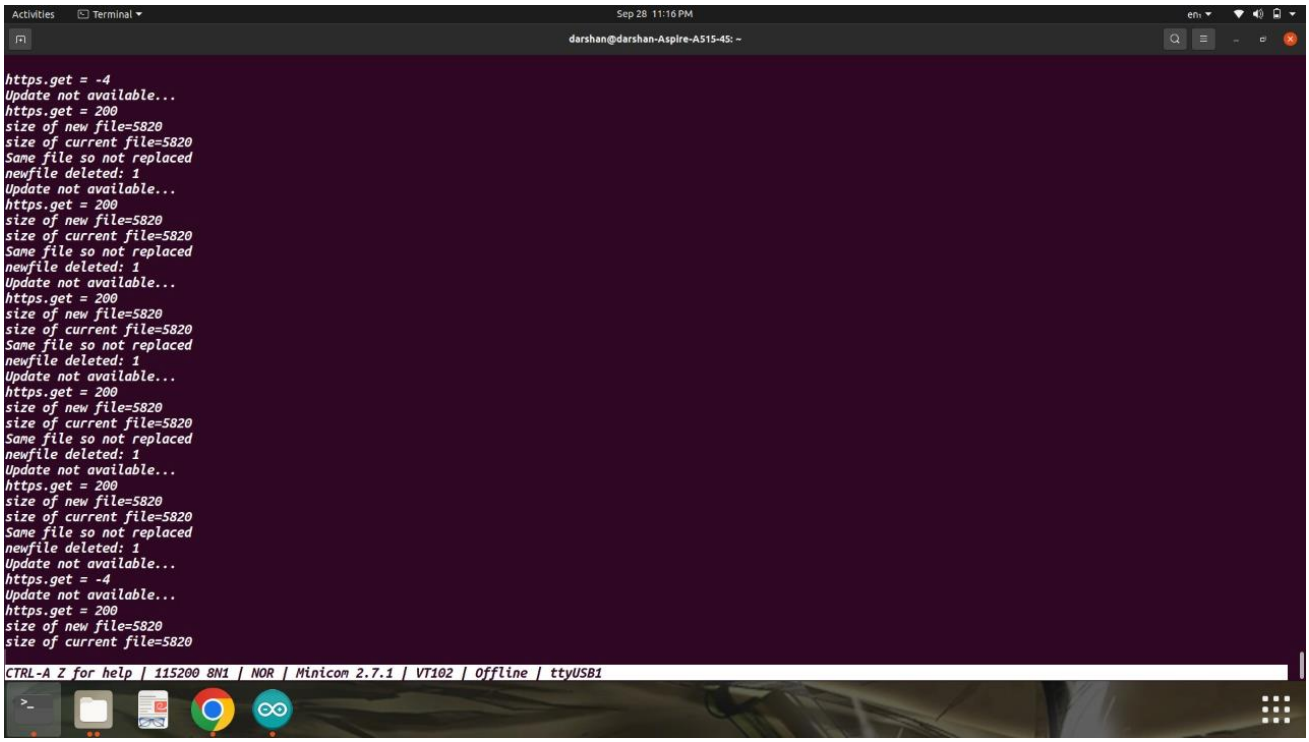

Figure 4.3 If update is not available.

CONCLUSION : -

The project focused mainly on the implementation of portable and customisable bootloader that facilitated over the air upgrades for microcontroller based application. For the demonstration ESP8266 was used as host which checked for the updates on AWS server and STM32F407 board was used as target where the update was download using XMODEM protocol. The project was successful in its implementation.

References

[1] Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors, The - Yiu, Joseph

[2] STM32F4xxMCU_ReferenceManual

[3] STM32Fxx_HAL_LowLevel_GenericGuide

[4] ESP8266ex_datasheet

[5] ESP8266-technical_reference