## What is a bootloader in embedded system?

Like a normal operating system, the microcontroller also serves the same purpose. this is the first piece of code that run when you press the reset button if you have a bootloader. if you don’t have a bootloader, then directly an application will start running.

If you have a bootloader, then before the main application, this bootloader starts running and does some process. once it is done with the operations, the bootloader job is done. so, it gives control to the main application. now, the application does its job based on our product or project.

## What is a system bootloader?

System bootloader in STM32 is the bootloader that has been provided by the chip manufacturers. That we cannot overwrite as it has loaded into the ROM memory. We can also call this as a ROM bootloader. Using this bootloader, we can update the firmware or application from the bootloader. But we cannot do many operations like update the application wirelessly like OTA. So, writing our own bootloader in STM32 will help us to implement whatever we like.

## Why do we use bootloader?

* Firmware update
* Security purpose
* etc.

## Firmware update

Is it really useful? why do we need this? still, I am using a project which doesn’t have a bootloader. I will try to answer the question.

I agree that when you don’t have a bootloader, it’s very simple. we are nor complicating by writing an extra bootloader. only main application is enough. but when you are planning to sell your product to the customers, what will do if you want to update the application/firmware in the device that you sold already? Every time go to field and connect the JTAG/J-LINK/ST-LINK and flash the firmware or application? it is not possible, right? So, if you have your bootloader, then you don’t need to worry about that. you can update the firmware or application without connecting any debugger or flasher.

With help of bootloader you will update firmware/application using mobile or desktop application using USART or USB another batter option is OTA updating system using this you will update your firmware using Server / Bluetooth /Wi-Fi/ethernet

## Security purpose

When you have the product, which has to be secured, then what will you do when someone overwrites the application or firmware with them customize firmware to hack your product? how do you find it? in this case, we can use the bootloader to check whether the firmware valid or not. it is valid, then only we give control to the firmware or application.

## How arm cortex m boots?

* Main stack pointer
* Reset handler
* Initialize the hardware
* Copy the initialized data segment to the ram
* Copy the uninitialized the data segment to the ram and initialize it to 0
* Call the main function

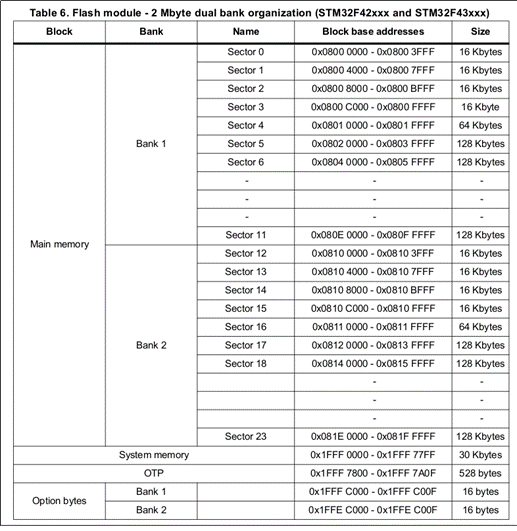
# Flash memory in STM32F401RCT6 (SECTOR 0 – SECTOR 5)

Microcontroller has internal flash memory which can store the firmware as a bin/hex format.

Before designing bootloader, we need to know how to read and write into internal flash memory. Because BIN file of firmware needs to write into internal flash memory. Here mentions some operation of flash memory.

* Flash memory program/erase operations
* Read / write protections
* Flash lock/unlock operations

In this tutorial series, we will be using flash memory as a single bank mode. The single blank is divided into multiple sectors as given below.

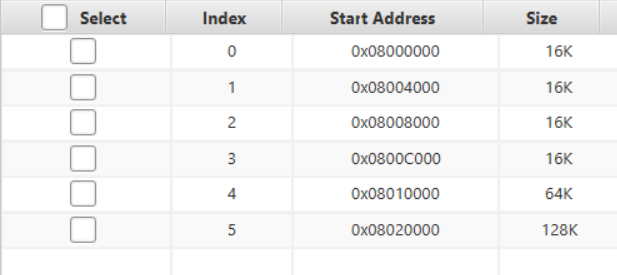


## Bootloader Design

We have discussed some important memory-related things earlier and understood that writing our own bootloader is good instead of using the system bootloader in our project. Let’s jump to the Bootloader. So, we will place the bootloader into 0x08000000 (Sector 0), and when the system power-up, it will run the bootloader. Then we have to place the application into some other memory like sector 4, sector 5. From the bootloader, we have to call the application. It’s simple right? But it is not.

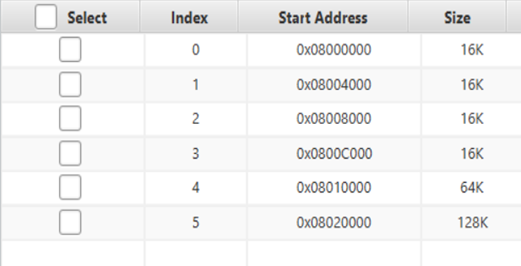
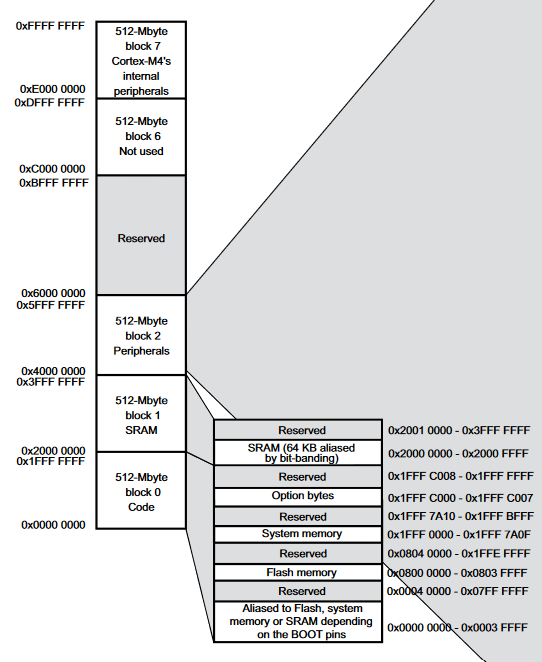
# Organizing the memory

First, we have to decide, where to keep the application and bootloader. In this series, I am also going to keep the two backup spaces for the application where I store the current version of the application (Slot 1) and the previous version of the application (Slot 2). That’s why I have decided to keep two versions of the application as a backup. Based on the memory you have; you can decide the number of slots.



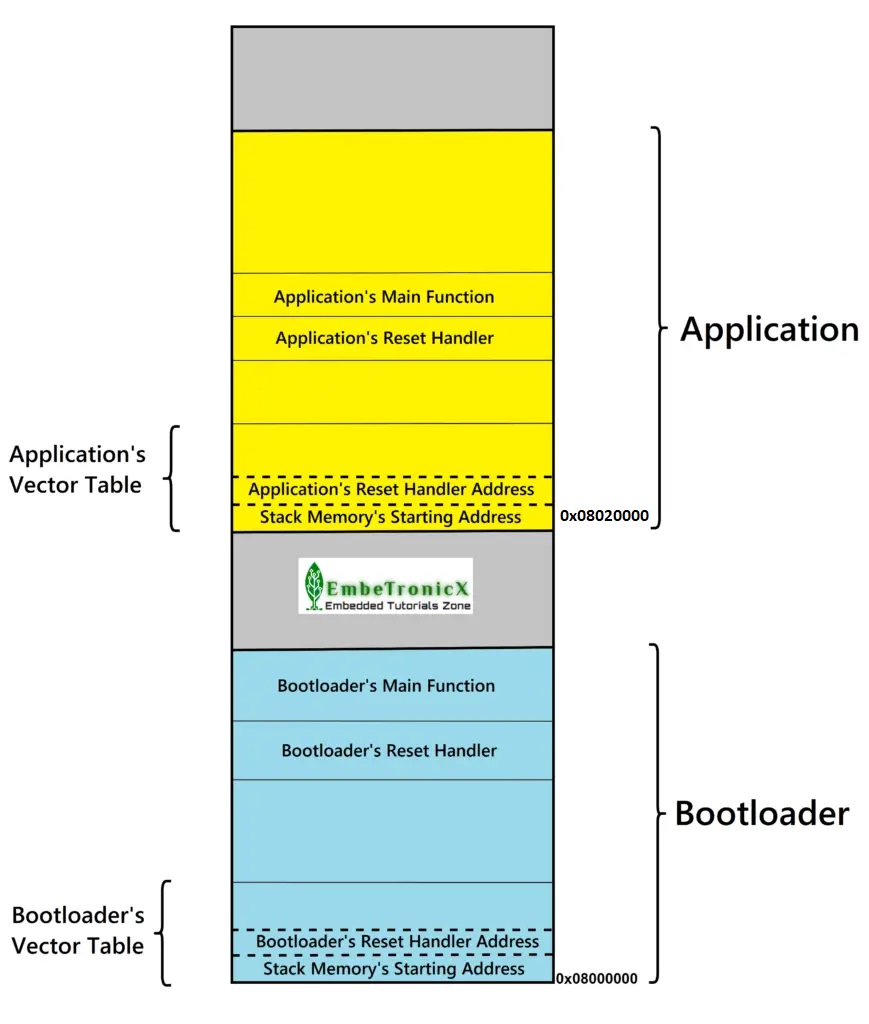
* This picture (collect from stm32 cube programmer) represents the address mapping of internal flash memory (stm32f401rct6)
* The bootloader will be placed into the sector 0 (starting addr **0x08000000**) (**16KB**).
* The actual application will be placed into the sector 5 (starting addr **0x08020000)** (**128KB**).
* Backup application will be placed into the sector 4
* Backup application 2 will be placed into the sector 3
* Configurations will be stored in sector 2

Look at the below image for a better understanding.



## Bootloader and Application operations

First of all, need to create two simple projects in “keil uvision ide”, one is for bootloader and another one is for “main application”. Both projects will create different binary files. We are going to place these binary files in our Flash memory.



* We will place the bootloader in the starting position of the flash, which is 0x08000000. And application into 0x08020000.

So, now what will happen when you press the RESET button? It reads the BOOT pin and as it is 0, then it starts from the flash memory. We have placed the bootloader in the starting position. So, it will read the stack address from 0x08000000 and store that into the MSP register. Then it takes the reset handler address from 0x08000004 and jumps to that. Then it does the all memory initialization of data segments and it call the bootloaders main function .in the bootloader main function, we will do some other operations (led off/on). once after that, we have to call the application. How the bootloader will call the application? Using the function pointer, we can directly call the application’s reset handler(0x08020004). Then the applications reset handler does some initialization and calls the applications main function. then our main application start running.

## Create bootloader application

* Create a new project in keil
* Add required library
* Write code for bootloader application
* Blink led for indicate bootloader application is running
* Create a function pointer for calling main application’s reset handler
* Jump to main application’s reset handler

Example code of simple bootloader given below:

#include "stm32f4xx.h" // Device header

#include "GPIO.h" // GPIO library header

#include "stdio.h"

#include "delay.h" // Delay library header

static void goto\_application(void);

int main()

{

SystemCoreClockUpdate();

sysTickInit();

goto\_application(); // call this function for jumping to bootloader to main application

while(1)

{

}

}

static void goto\_application(void)

{

PinName led=PinCreate(PORTC,PIN\_13,OUTPUT); // create gpio pin for blinking led

digitalWrite\_High(led);

delayms(1000);

// create function pointer

void (\*app\_reset\_handler)(void) = (void\*)(\*((volatile uint32\_t\*) (0x08020000 + 4U)));

digitalWrite\_Low(led\_boot);

app\_reset\_handler();

}

Then we will do some flash memory related operation for bootloader application :

* Go to option for target and select “Target”
* go to “Read/Only Memory areas”
* Then set the start address and size of “IROM1”
* Start address is 0x08000000
* And size will be 16kb (sector 0) or 32kb (sector 0 and sector 1)



* Compile that code and upload to device

## Create main application:

* Create a new project in keil
* Add required library
* Write code for main application which blink a led

Example code for simple application given below :

#include "stm32f4xx.h" // Device header

#include "GPIO.h"

#include "stdio.h"

#include "delay.h"

int main()

{

SystemCoreClockUpdate();

sysTickInit();

PinName led=PinCreate(PORTC,PIN\_13,OUTPUT);

while(1)

{

digitalWrite\_High(led);

delayms(1000);

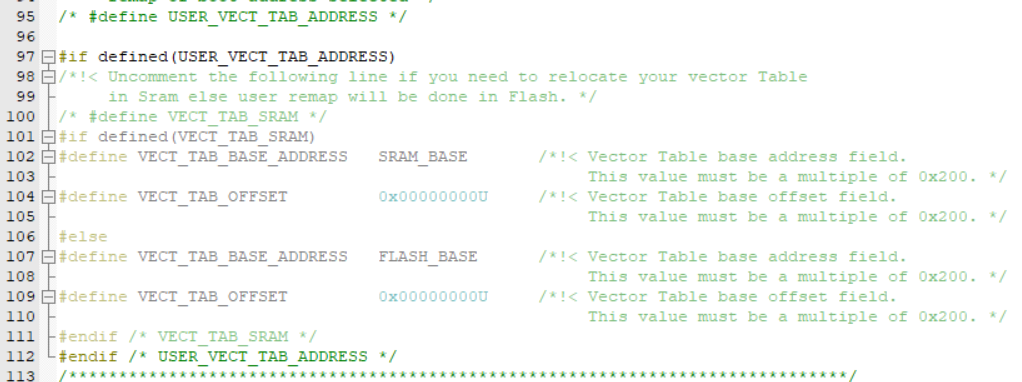
digitalWrite\_Low(led);

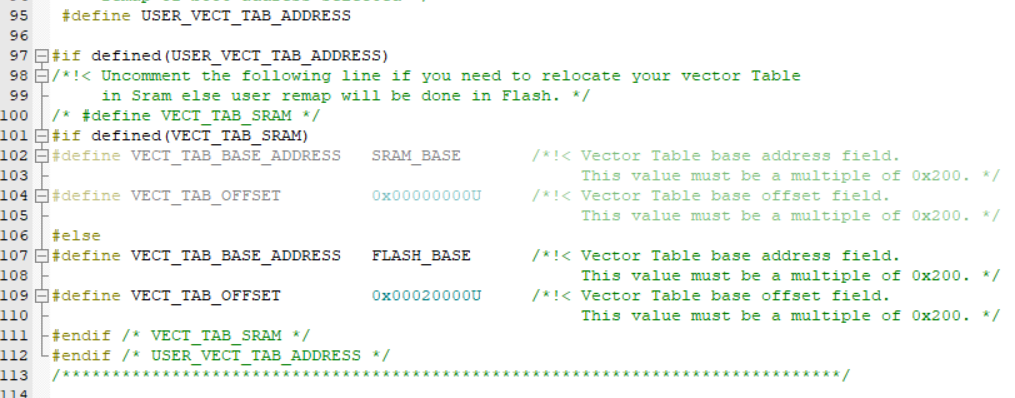
delayms(1000);

}

}

After writing code need to open startup file (system\_stmf4xx.c). uncomment line 95 and edit line 109 , here add vect\_tab\_offset . Its very important step.





Then we will do some flash memory related operation for bootloader application :

* Go to option for target and select “Target”
* go to “Read/Only Memory areas”
* Then set the start address and size of “IROM1”
* Start address is 0x08020000
* And size will be 128kb (sector 5)
* Then go to “Linker” and set R/O base address (0x08020000)
* Compile that code and upload to device

