Boot loader

## Flow chart



## Architecture

קוד התוכנה (firmware) כתוב באזור זיכרון שנקרא flash memory, להלן "הזיכרון"

במעבד שלנו הזיכרון מחולק לשני בנקים

בהפעלה רגילה הקוד שבבנק 1 רץ. אבל יש אפשרות להריץ גם קוד מבנק 2 מיד עם הדלקת המעבד.

אפשרות זאת נועדה לכך שבזמן שכותבים קוד לאזור אחד בזיכרון, המעבד מקונפג לעלות עם קוד מהאזור האחר, וזאת כדי שבמידה ותהיה תקלה בעת הכתיבה, תמיד יהיה קוד עובד על הכוונת.

אנחנו בחרנו להשתמש בקוד עדכון תוכנה של ST (כי הוא כבר עבר debugging וכו')  
בחירה זאת מונעת מאיתנו להשתמש בחלק השני של בנק 2 לקוד המתעדכן, כי מרחב הכתובות הנגיש לקוד של ST כולל את בנק 1 ורק המחצית הראשונה של בנק 2.

בחרנו גם לקפוץ לקוד של ST מתוך הקוד שלנו (בלי להסתמך על אפילסופט)  
בחירה זאת מונעת מאיתנו להפעיל את עדכון התוכנה מתוך בנק 2, כי אם קופצים לקוד של ST כאשר התנאים לעלייה מבנק 2 מתקיימים, אז הקוד של ST קופץ לבנק 2 (במקום להיכנס לרוטינת עדכון תוכנה).

בגלל שתי המגבלות הללו   
בשילוב מבנה הקוד (חלק גדול של Reticles & Menus וחלק יותר קטן של קוד תפעול הכוונת)  
הוחלט למקם את הקוד בזיכרון כדלקמן:

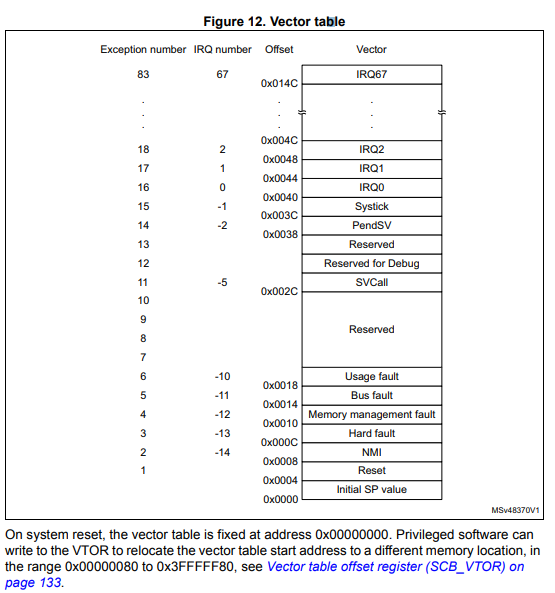
|  |  |
| --- | --- |
| Factory Code @ Bank1  (start address 0x08000000) | Updated Code @ Bank2 (start address 0x08040000) |
| Updated Reticles & Menus @ bank 1  (start address 0x08008000) | Factory Reticles & Menus @ bank2  (start address 0x08048000) |

## Startup Sequence in IAR

כשהמעבד מופעל מתקיים באופן חומרתי: התוכן של כתובת 0 בזיכרון מועתק אל הרגיסטרSP (Stack pointer). והתוכן של הבית השני בזיכרון (0x04) מועתק אל הרגיסטר PC (Program Counter).

אם nBFB2=1 (כדבעי) אז הכתובות הראשונות (כולל 0, 0x04) ממופות אל בנק1 (שכתובתו מתחילה ב-0x0800 0000). התוכן של 0x0800 0000 מכיל את הכתובת שבא נמצא הקוד של Reset\_Handler. ערכה נקבע על ידי assembler/linker/locator

אם nBFB2=0 והבית הראשון של בנק2 מכיל ערך ואלידי,   
אז הכתובות הראשונות ממופות אל system flash (ולכן צריך להיזהר לא להפנות אל כתובת אפס באופן ישיר מתוך התוכנה)  
PC, SP מקבלים את ערכם מהבתים הראשונים שבבנק 2 (ולכן הקוד עובד משם)  
יש למפות את VTOR אל בנק2 לצורך פעולה תקינה של אינרפטים (כי שם ממוקם ה-vector table).

From [ST Cortex®-M3 programming manual](http://www.st.com/content/ccc/resource/technical/document/programming_manual/5b/ca/8d/83/56/7f/40/08/CD00228163.pdf/files/CD00228163.pdf/jcr:content/translations/en.CD00228163.pdf)

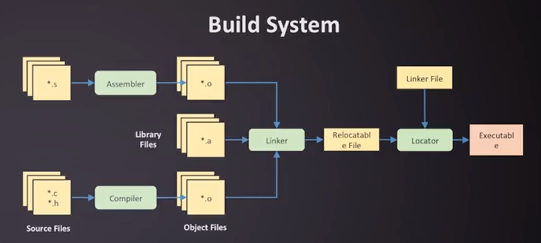
Reset\_Handler כתוב באסמלי  
דבר ראשון הוא קורא לפונקציה SystemInit (שהיא כבר כתובה ב-C)  
דבר שני קורא לפונקציות \_cmain ; \_\_low\_level\_init ; \_\_iar\_data\_init3; \_iar\_packbits\_init\_single3 ; \_call\_main שכתובות באסמבלי. למיטב הבנתי \_\_iar\_data\_init3 מאתחלת את המשתנים (BSS, Data)  
ושאר הפונקציות לא עושות משהו משמעותי.  
בסוף קוראים לאפליקציה הראשית (main)

## Jump to Boot Loader **Using** Reset

אפשר לוותר על התענוג של האיתחולים הנדרשים לפני פקודת jump, אם משתמשים בטריק של  
write to register ולאחריוreset . להלן מוטיבציה מתוך [stackoverflow](https://stackoverflow.com/questions/26891432/jump-to-bootloader-in-stm32-through-appliction-i-e-using-boot-0-and-boot-1-pins).  
בתוכנה צריך להיות כתוב, עוד לפני ביצוע ה-main ואף לפני ביצוע system init שיקפוץ לקוד של ה-Boot Loader

השיטה הזאת מודגמת [בקישור הזה](https://stackoverflow.com/questions/28288453/how-do-you-jump-to-the-bootloader-dfu-mode-in-software-on-the-stm32-f072)

## Linker



קובץ ה-Linker מורה לקומפיילר היכן למקם בזיכרון את קטעי הקוד

Linker File: C:\...\EWARM\stm32l152xe\_flash.icf

It may be editted from IAR using Right click on project\_name🡪 “options…” 🡪 “Linker” 🡪 “edit”

# Other (less relevant) references

## System memory boot mode ([AN2606](http://www.st.com/content/ccc/resource/technical/document/application_note/b9/9b/16/3a/12/1e/40/0c/CD00167594.pdf/files/CD00167594.pdf/jcr:content/translations/en.CD00167594.pdf))

### STM32L151 Specific

STM32L1xxxE is used to refer to … STM32L1xxRET6 ultralow power devices (STM32L151RET6TR)  
Bootloader ID: 0x40 ; Memory location: 0x1FF00000 ; Protocol version V3.1

8 Kbyte starting from address 0x1FF00000 contains the bootloader firmware. 0x1FF00000 - 0x1FF01FFF

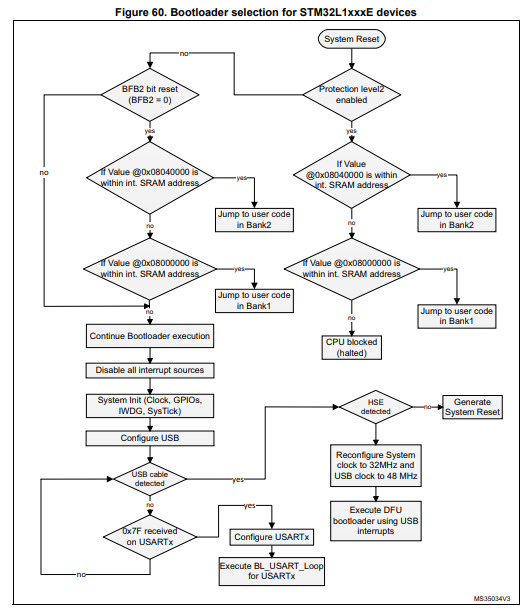
Once initialized, the USART1 configuration is: 8 bits, even parity and 1 Stop bit.

PA10 pin: USART1 in reception mode

PA9 pin: USART1 in transmission mode

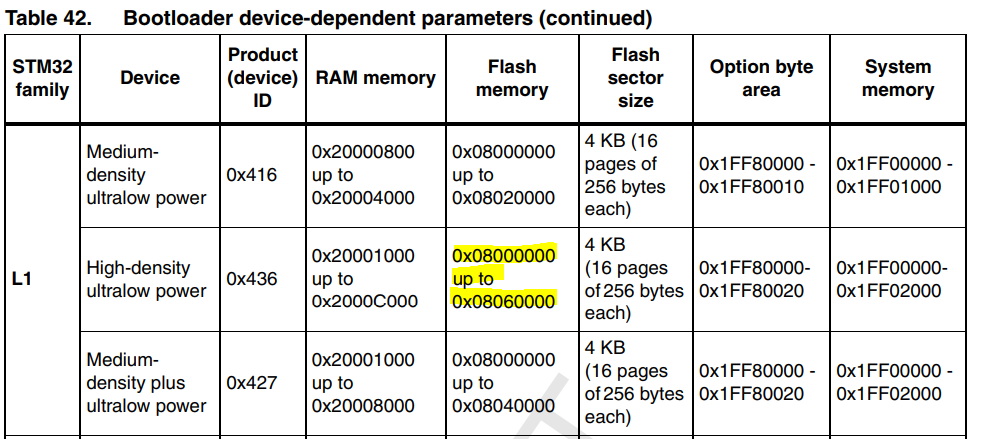
SysTick timer is Used to automatically detect the serial baud rate from the host for USARTx bootloader

The system clock is derived from the embedded internal high-speed RC for USARTx bootloader.



### Valid address space

“Valid memory addresses (RAM, Flash memory, System memory, option byte area) accepted by the bootloader when the Read Memory, Go and Write Memory commands are accepted…”



### Activation

#### Hardware

Bootloader activation pattern (Pattern 4 of table 2 is the relevant pattern):   
Boot0(pin) = 1, Boot1(pin) = 0 and BFB2(bit) = 1 ;   
OR Boot0(pin) = 0, BFB2(bit) = 0 and both banks don’t contain valid code ;   
OR Boot0(pin) = 1, Boot1(pin) = 0 and BFB2(bit) = 0

#### Software

In addition to patterns described above, user can execute bootloader by performing a jump to system memory from user code.   
Before jumping to Bootloader user must:

• Disable all peripheral clocks

• Disable used PLL

• Disable interrupts

• Clear pending interrupts

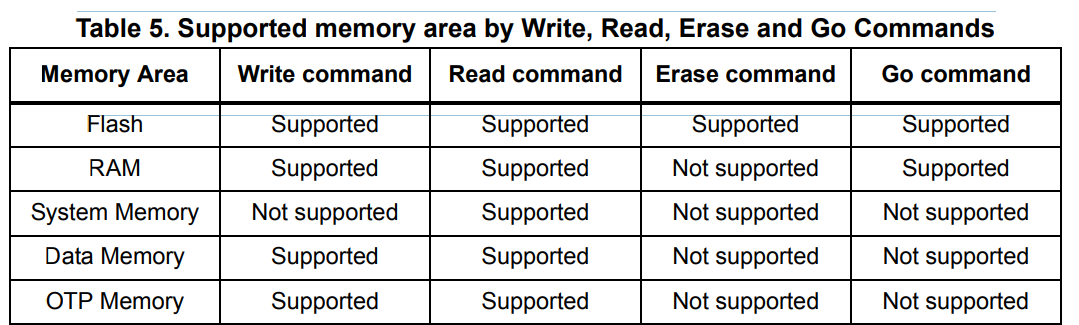
* Remap the System Memory Bootloader at address 0x00000000 using SYSCFG register

### Memory Management

4.4 Bootloader Memory Management

\* All write operations using bootloader commands must only be Word-aligned (the address should be a multiple of 4). The number of data to be written must also be a multiple of 4 (non-aligned half page write addresses are accepted).

\* Bootloader firmware of STM32 L1 and L0 series supports Data Memory in addition to standard memories (internal Flash, internal SRAM, option bytes and System memory). The start address and the size of this area depends on product, please refer to product reference manual for more information. Data memory can be read and written but cannot be erased using the Erase Command. When writing in a Data memory location, the bootloader firmware manages the erase operation of this location before any write. A write to Data memory must be Word-aligned (address to be written should be a multiple of 4) and the number of data must also be a multiple of 4. To erase a Data memory location, you can write zeros at this location.



### Timing

After bootloader (or device) reset, the host should wait until the STM32 bootloader is ready to start detection phase with the UART. This time corresponds to bootloader startup timing, during which resources used by bootloader are initialized. Minimum bootloader Startup time 0.708 ms; HSE Timeout 200 ms.

USART connection timing is the time that the host should wait for between sending the synchronization data (0x7F) and receiving the first acknowledge response (0x79). See Fig 71 for details.

One USART byte sending: 78 us; USART configuration: 8 us; USART connection: 0.164 ms

## [STM32L151xx Reference manual](http://www.st.com/content/ccc/resource/technical/document/reference_manual/cc/f9/93/b2/f0/82/42/57/CD00240193.pdf/files/CD00240193.pdf/jcr:content/translations/en.CD00240193.pdf)

### Bank selection

Memory address: 0x1FF8 0004; Bit 7: nBFB2

This bit is used to select the boot space between Flash Bank2/Bank1 and

another boot sources. It is located in the User option byte

0: If pins (BOOT1 & BOOT0)=0 system bootloader is started after reset

(boot from system memory).

Bootloader in next executes checks in following order:

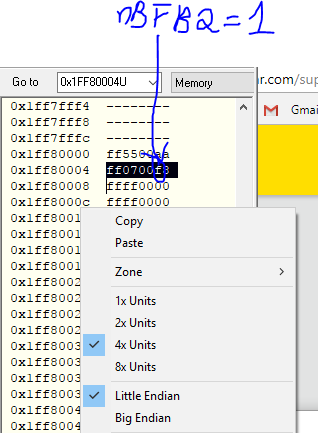
- If content of first word of Bank2 is valid SRAM address then is jumped to Bank2 (boot from Bank2).

- If content of first word of Bank1 is valid SRAM address then is jumped to Bank1 (boot from Bank1).

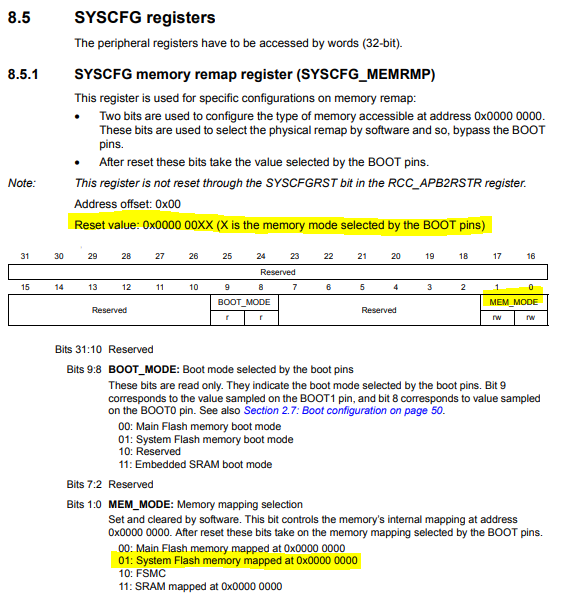
- Otherwise system bootloader is executed.

1: Boot space is selected by BOOT0 and BOOT1 pins

From IAR during debug mode:



### Remap 0x00 Address



### Memory organization

Each bank in dual bank devices is has 256 Kbytes of program memory 8 Kbytes of data

The Flash program memory block is divided into sectors of 4 Kbytes each, and each sector is further split up into 16 pages of 256 bytes each. The sector is the write protection granularity. The pages are the erase granularity for the program memory block. The Flash program memory pages can be written using a half page programming or a fast word programming operation.

Data EEPROM can be erased and written by: • Double word • Word/ Fast word • Half word / Fast half word • Byte / Fast byte During a write/erase operation to the NVM (except Half Page programming or Double-word erase/write), any attempt to read the same bank of NVM stalls the bus. The read operation is executed correctly once the programming operation is completed. This means that code or data fetches cannot be performed while a write/erase operation is ongoing in the same bank

The marking code is accessible and not accessible (see Valid address space above)

Block Name Memory addresses Size

Program memory bank 1 Sector 0 Page 0 0x0800 0000 - 0x0800 00FF 256 bytes

Program memory bank 1 Sector 32 to Sector 63 0x0802 0000 - 0x0803 FFFF 128 Kbytes

Program memory bank 1 Page 512 to 1023 0x0802 0000 - 0x0803 FFFF 128 Kbytes

Program memory bank 2 Sector 64 to Sector 95 0x0804 0000 - 0x0805 FFFF 128 Kbytes

Program memory bank 2 Page 1024 to 1535 0x0804 0000 - 0x0805 FFFF 128 Kbytes

Program memory bank 2 Sector 96 to Sector 127 0x0806 0000 - 0x0807 FFFF 128 Kbytes

Data EEPROM bank 1 0x0808 0000 - 0x0808 1FFF 8 Kbytes

Data EEPROM bank 2 0x0808 2000 - 0x0808 3FFF 8 Kbytes

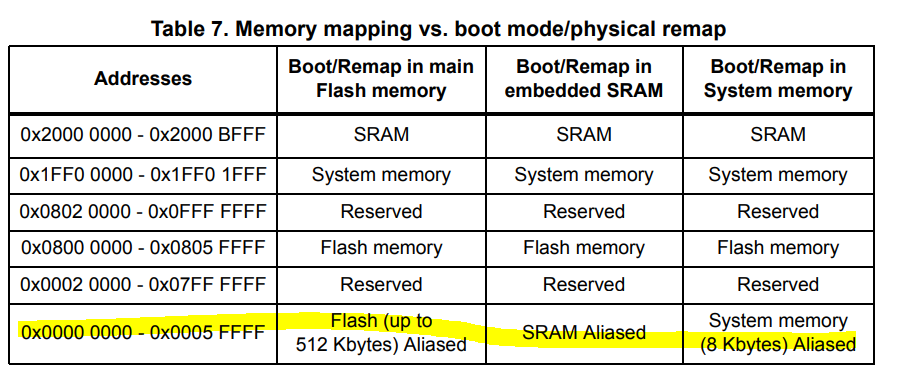
System memory bank 1 Page 0 0x1FF0 0000 - 0x1FF0 00FF 256 bytes

System memory bank 1 Page 15 0x1FF0 0F00 - 0x1FF0 0FFF 256 bytes

System memory bank 2 Page 16 to 31 0x1FF0 1000 - 0x1FF0 1FFF 4 Kbytes

Option bytes bank 1 0x1FF8 0000 - 0x1FF8 001F 32 bytes

Option bytes bank 2 0x1FF8 0080 - 0x1FF8 009F 32 bytes



### Write protection

Note: Depending on the memory protection programmed by option byte, some boot/remap configurations may not be available (refer to the readout protection section in Section 3: Flash program memory and data EEPROM (FLASH) for details).

Changing Write protection is described in page 62: 3.4.1 Unlocking/locking memory

Nevertheless, ST’s built in bootloader knows how to writes to any relevant place

## Other bank selection resources

### [Updating firmware in STM32L1xx microcontrollers through in-application programming using the USART AN3310](http://www.st.com/content/ccc/resource/technical/document/application_note/90/3a/e2/45/c7/d2/4c/09/CD00290444.pdf/files/CD00290444.pdf/jcr:content/translations/en.CD00290444.pdf)

**9. Dual bank feature for high-density devices**

For STM32L15xx high-density devices (these devices have two Flash memory banks:

Bank1 and Bank2), an additional boot mechanism is available which allows booting from

Bank2 or Bank1 (depending on the BFB2 bit status).

At startup, if BFB2 option bit is reset and the boot pins are in the boot from main Flash

memory configuration, the device boots from Flash memory Bank1 or Bank2, depending on

the activation of the bank.

The active banks are checked in the following order: Bank2, followed by Bank1.The active

bank is identified by the value programmed at the base address of the bank (corresponding

to the initial stack pointer value in the interrupt vector table).

1. When the BFB2 bit is reset, after reset, the device boots from the System memory and executes the embedded code which implements the dual bank mode:

a) First, the code checks Bank2. If it contains a valid code (see Note below), it jumps to the application located in Bank2.

b) If the Bank2 code is not valid, it checks Bank1 code. If it is valid (see Note below), it jumps to the application located in Bank1.

c) If both Bank2 and Bank1 do not contain valid code (see Note below), no jump to Flash banks is executed.

2. When BFB2 bit is set (default state), the dual bank boot mechanism is not performed.

Note: The code is considered as valid when the first data (at the bank start address, which should be the stack pointer) points to a valid address into the internal SRAM memory (stack top address). If the first address points to any other location (out of the internal SRAM), the code is considered not valid.

For more details, a dual bank Boot mode example (FLASH\Dual\_Boot) is provided within the STM32L15xx Standard Peripheral Library available on www.st.com.

To change the BFB2 status, add the following functions to your code:

/\* Reset BFB2 bit to enable boot from Flash Bank2 \*/

FLASH\_Unlock();

FLASH\_OB\_Unlock();

FLASH\_ClearFlag(FLASH\_FLAG\_EOP|FLASH\_FLAG\_WRPERR |

FLASH\_FLAG\_PGAERR |

FLASH\_FLAG\_SIZERR | FLASH\_FLAG\_OPTVERR | FLASH\_FLAG\_OPTVERRUSR);

FLASH\_OB\_UserConfig(OB\_IWDG\_SW, OB\_STOP\_NoRST, OB\_STDBY\_NoRST);

FLASH\_OB\_BORConfig(OB\_BOR\_LEVEL1);

/\* BFB2 option bit will be reset then a system (SW) reset will be

generated. After startup from reset, the device will boot from

Bank2. \*/

if (FLASH\_OB\_BootConfig(OB\_BOOT\_BANK2) == FLASH\_COMPLETE)

{

/\* generate System Reset to load the new option byte values \*/

FLASH\_OB\_Launch();

}

### STM32L151xE Datasheet

ההמלצה בדף של הרכיב היא להשתמש ב-Dual bank

"The boot from Flash usually boots at the beginning of the Flash (bank 1). An additional boot mechanism is available through user option byte, to allow booting from bank 2 when bank 2 contains valid code. This dual boot capability can be used to easily implement a secure field software update mechanism."

### [AN4767](http://www.st.com/content/ccc/resource/technical/document/application_note/group0/ab/6a/0f/b7/1a/84/40/c3/DM00230416/files/DM00230416.pdf/jcr:content/translations/en.DM00230416.pdf) usage of the dual bank

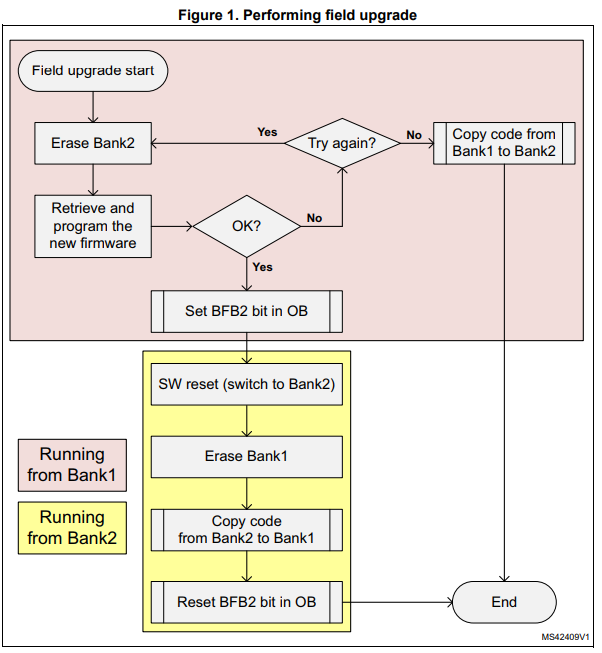
***זה מסמך מטעה ולא רלוונטי לנו***

Optimized usage of the dual bank structure of Flash memory in STM32 microcontrollers - Software expansion for STM32Cube

Although only Cat.5 devices from STM32L0 Series and access line and USB OTG devices from the STM32L4 Series are directly addressed in this document, other STM32 MCUs with two semi-independent banks of memory ***may*** share some of the described properties and be used in a similar way.

With dual bank, all the manipulation with the other bank is just another task of the main program. Thanks to internal remapping of the code address range, binaries in both banks can remain identical during normal operation.

להלן דיאגרמת זרימה של [dual bank מתוך המסמך AN4767](http://www.st.com/content/ccc/resource/technical/document/application_note/group0/ab/6a/0f/b7/1a/84/40/c3/DM00230416/files/DM00230416.pdf/jcr:content/translations/en.DM00230416.pdf)



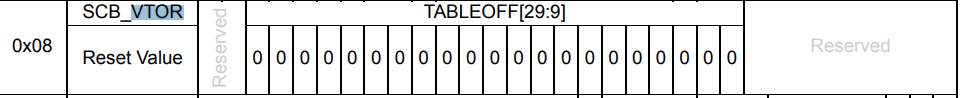
It is important to keep BFB2 flag set when there is no code in Bank1, thus being safe in case of unexpected power cut. Then, after reset, the firmware has several ways to detect that the code in the Bank1 must be replaced, and the process is running from Bank2. It is important to implement this decision, i.e. there is only one binary code, and the code is executed on each boot.

3.3 Vector table

! Beware of using the address 0x0000 0000 with dual bank automatic selection. This address range remains aliased to the system memory, despite the fact that the BOOT0 value points to main Flash memory selection.

To keep the capability of interrupt processing regardless of Flash memory mapping, the vector table must be relocated.

Side note VTOR is set to 0 upon reset



In the example provided along with this application note, it is copied to the system RAM, the safest option for field upgrade case. In this case its size (up to 192 B for the L0 Series, up to 288 B for the L4 Series, for the exact size on any given product refer to the reference manual) must be allocated in RAM and table copied there from its original location. The new interrupt vector table address must then be written to the System Control Block (SCB) register VTOR. The NVIC is not always aware of the modified memory address aliasing, the VTOR is the best guide to the vector table memory location. By default the linker places the vector table at the beginning of the program memory. When switching to Bank2, the VTOR must be changed to point to the Bank2 base address and vice versa, when switching back to Bank1 the VTOR must be reset back to Bank1 base address. Beware that the VTOR reset value is zero, in case of BFB2 option active, it will by default point to system memory.

Disabling interrupts during field upgrade is another option, however it unfortunately negates some of the big advantages of the dual bank system, such as the lack of any “limited mode”.