# AN4826
# Application note

STM32F7 Series Flash memory dual bank mode

## Introduction

With the growing demand of applications such as mobile communications and automotive systems which have strict real-time needs, it is necessary to access critical information about the system functionality before the completion of a Flash memory erase/program operation.

Other applications need a firmware upgrade, which can be risky, especially when the system power loss occurs during the update process. This can result in many problems such as a transmission error or an information loss.

For these reasons, ST offers STM32 MCUs that embed dual bank Flash memories designed to respond to the above needs.

The dual bank Flash memory allows a code to be executed in one bank, while another bank is being erased or programmed. It avoids a CPU stalling during programming operations and protects the system from power failures or other errors.

This application note gives an overview of the STM32F7 Series Flash memory dual bank capabilities, such as the Read-While-Write (RWW) and the dual boot features.

This application note is provided with the X-CUBE-DBANK-F7 embedded software package that contains three examples with all the embedded software modules required to run the examples.

The examples describe the main features of the Flash dual bank mode:

- **Read-while-write example**: explains through oscilloscope waveforms how the read-while-write feature allows a code to be executed from the Flash bank1 while writing in the Flash bank2 without stalling the execution.
- **Dual boot example**: describes the dual boot capability either by booting in the Flash bank1 and toggling the LED1 or by booting in the Flash bank2 and toggling the LED2.
- **Performance and consumption example**: runs a **CMSIS Arm® graphic equalizer** algorithm and measures the STM32F7 Series device performance and consumption comparing the dual bank mode to the single bank mode.

## Related documents

Available from STMicroelectronics web site *www.st.com*:

- *STM32F76xxx and STM32F77xxx advanced Arm®-based 32-bit MCUs* (RM0410)
- *STM32 microcontroller system memory boot mode* (AN2606).

# Contents

# List of tables

# List of figures

# 1 General information

This document applies to Arm®[a]-based devices.

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

# 2 Flash single bank and dual bank configurations

The STM32F7 Series devices offer a Flash memory with 1 Mbyte and 2 Mbyte memory sizes.

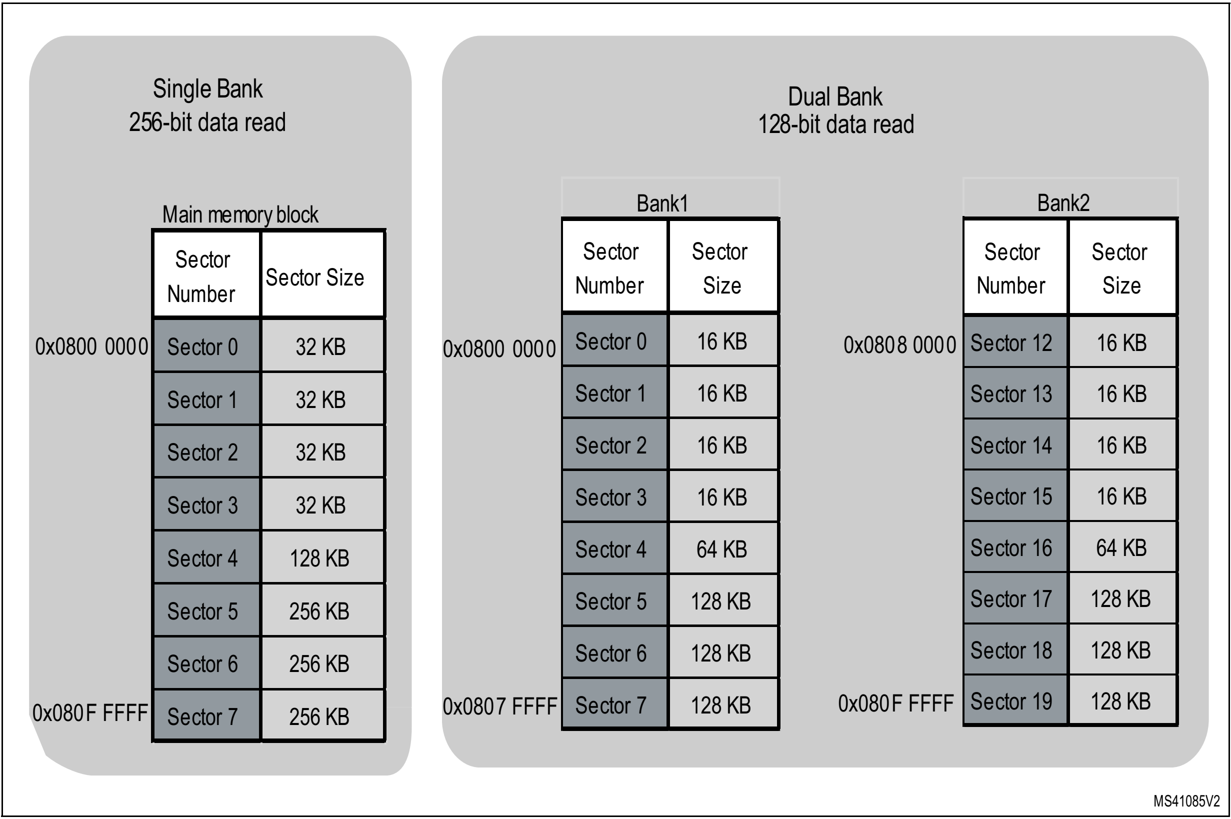
This Flash memory can be configured as a single bank or as a dual bank.

## 2.1 1 Mbyte Flash memory organization

*Figure 1* presents the 1 Mbyte Flash memory main block organization for both configurations: single bank and dual bank.

- **Single bank configuration**: the main memory block is divided into 4 sectors of 32 Kbytes, 1 sector of 128 Kbytes and 3 sectors of 256 Kbytes.
- **Dual bank configuration:** in each 512 Kbytes bank, the main memory block is divided into 4 sectors of 16 Kbytes, 1 sector of 64 Kbytes and 3 sectors of 128 Kbytes.

**Figure 1. 1 Mbyte Flash memory organization**



*Note: The sector numbering of the dual bank memory organization is different from the single bank memory organization:*

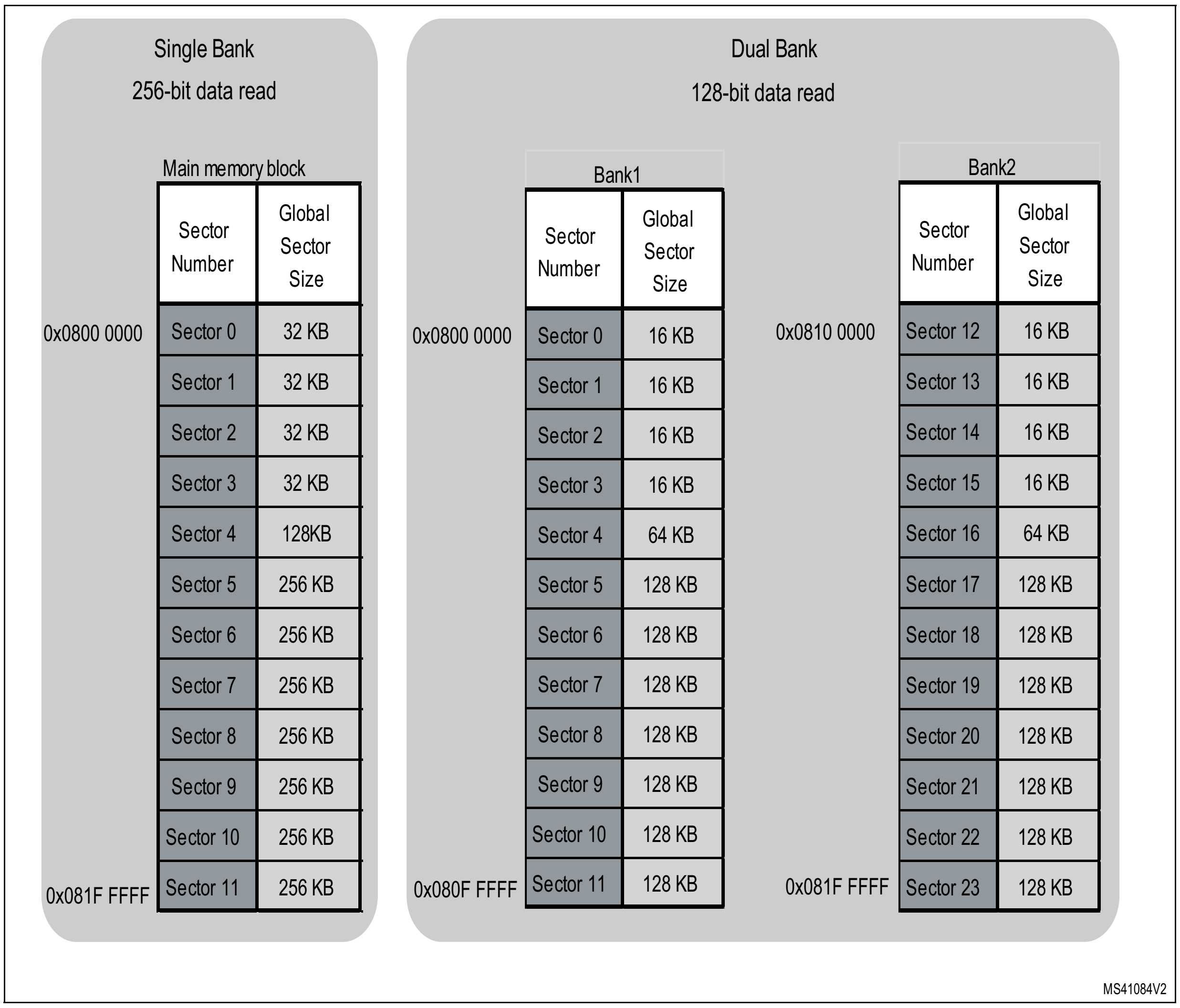
*The single bank memory contains 8 continuous sector numbers whereas the dual bank memory contains 16 sectors with a discontinuity on the sector numbering.*

## 2.2 2 Mbyte Flash memory organization

*Figure 2* presents the 2 Mbyte Flash memory main block organization for both configurations: single bank and dual bank.

- **Single bank configuration:**
  The main memory block is divided into 4 sectors of 32 Kbytes, 1 sector of 128 Kbytes and 7 sectors of 256 Kbytes.
- **Dual bank configuration**:
  In each 1 Mbyte bank, the main memory block is divided into 4 sectors of 16 Kbytes, 1 sector of 64 Kbytes and 7 sectors of 128 Kbytes.

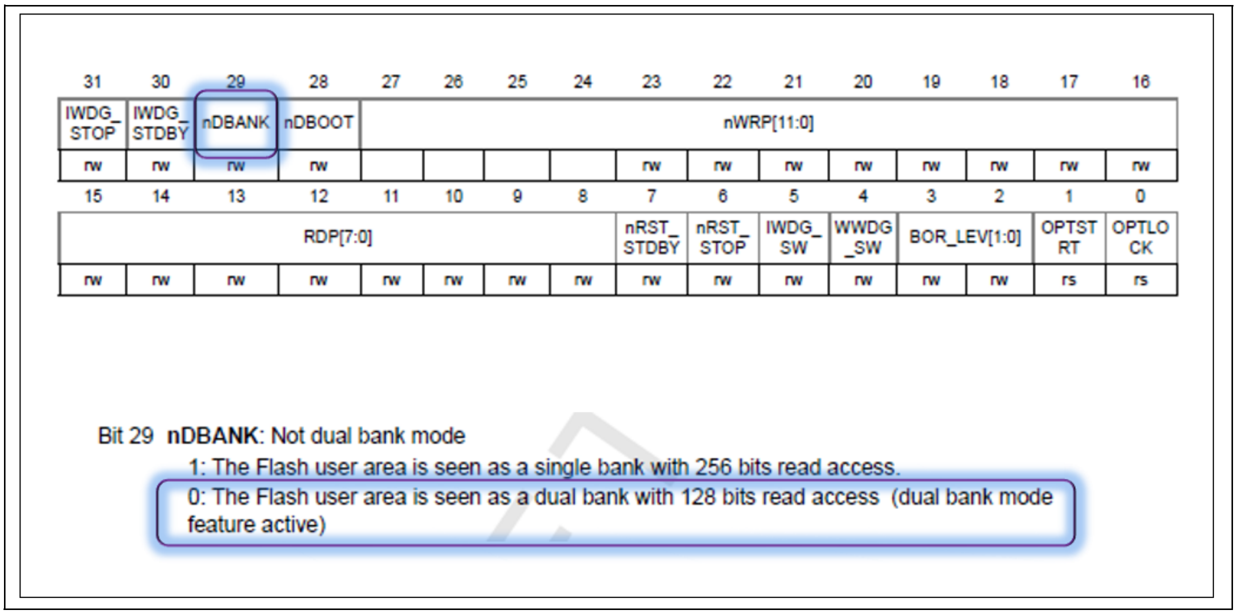
**Figure 2. 2 Mbyte Flash memory organization**



## 2.3 How to activate the dual bank mode

The dual bank Flash memory mode is activated by setting nDBANK= 0 in user option bytes via the FLASH_OPTCR register.

**Figure 3. How to activate the dual bank mode**



| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IWDG_ STOP | IWDG_ STDBY | nDBANK | nDBOOT | nWRP[11:0] | | | | | | | | | | | |
| rw | rw | rw | rw | | | | | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RDP[7:0] | | | | | | | | nRST_ STDBY | nRST_ STOP | IWDG_ SW | WWDG _SW | BOR_LEV[1:0] | | OPTST RT | OPTLO CK |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rs | rs |

Bit 29 **nDBANK**: Not dual bank mode

1: The Flash user area is seen as a single bank with 256 bits read access.

0: The Flash user area is seen as a dual bank with 128 bits read access (dual bank mode feature active)

# 3 Read-while-write (RWW)

The dual bank Flash memory allows a read-while-write capability so as to program the systems while continuing to operate.

However, it is not possible to execute an erase or program operation on one bank while erasing or programming the other bank (except for a mass erase that erases both banks at the same time). This is mentioned in *Figure 4*, where the output of the prog/erase FSM must be either a program or an erase operation.

**Figure 4. Read-while-write diagram**

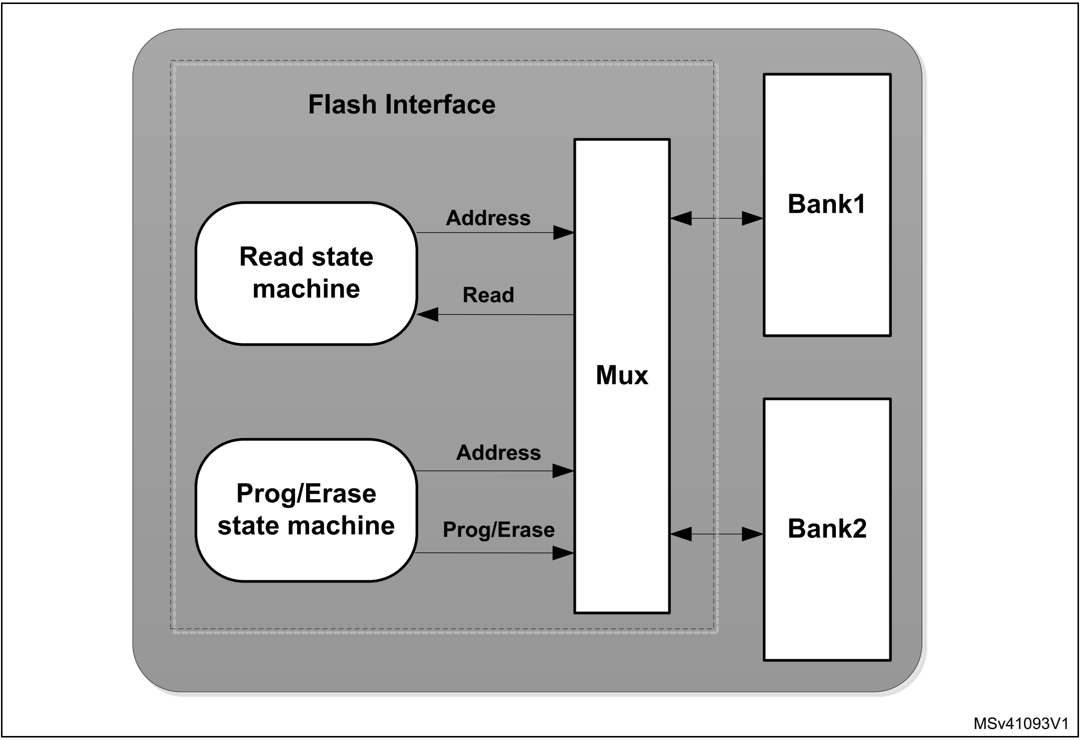


*Table 1* summarizes the read-while-write operation possibilities:

The legend for *Table 1* is as follows:

A = Allowed,

NA and grayed = Not allowed.

**Table 1. RWW operation possibilities**

| | | Bank2 | | |
|---|---|---|---|---|
| | | **Read** | **Prog** | **Erase** |
| **Bank1** | **Read** | NA | A | A |
| | **Prog** | A | NA | NA |
| | **Erase** | A | NA | A |

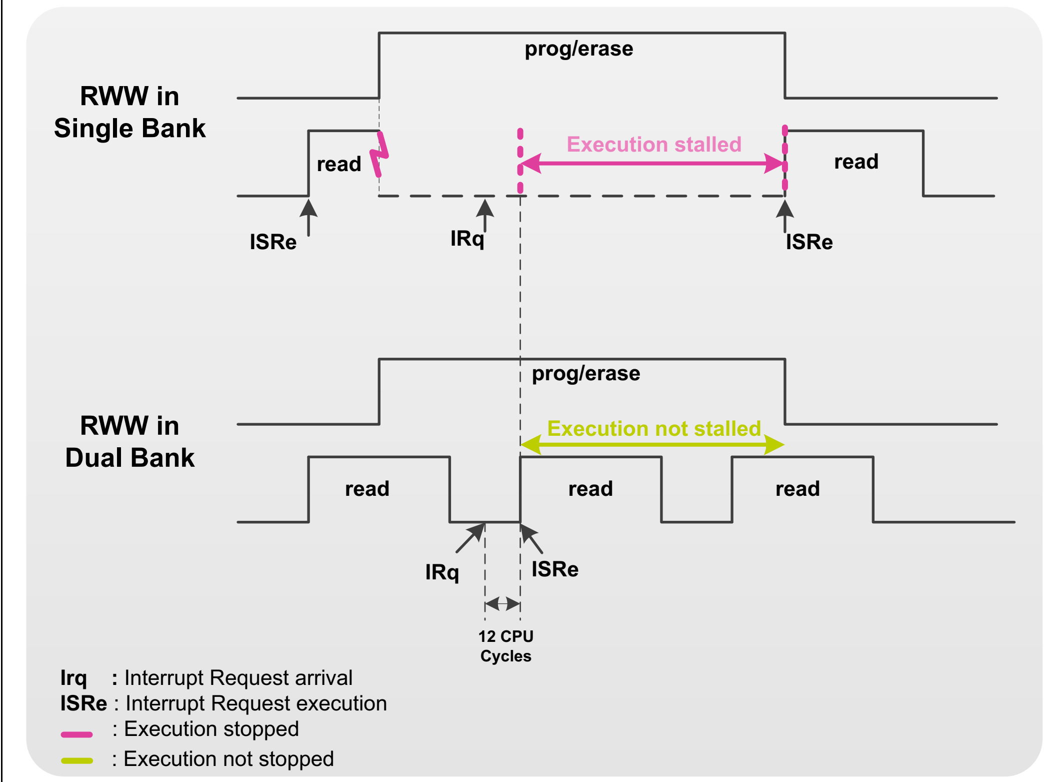
## 3.1 RWW in the Flash memory single bank versus dual bank

*Figure 5* describes how in the Flash memory single bank the CPU execution is stalled, while in Flash memory dual bank the execution continues.

In the single bank mode, when a read interrupt request arrives while a prog/erase operation is ongoing, the CPU cannot execute this interruption until the prog/erase operation is completed.

On the contrary in the dual bank mode even during a prog/erase operation (example in bank1) the CPU can simultaneously execute an interrupt from the bank2 without stalling.

**Figure 5. RWW in Flash memory single bank versus dual bank**

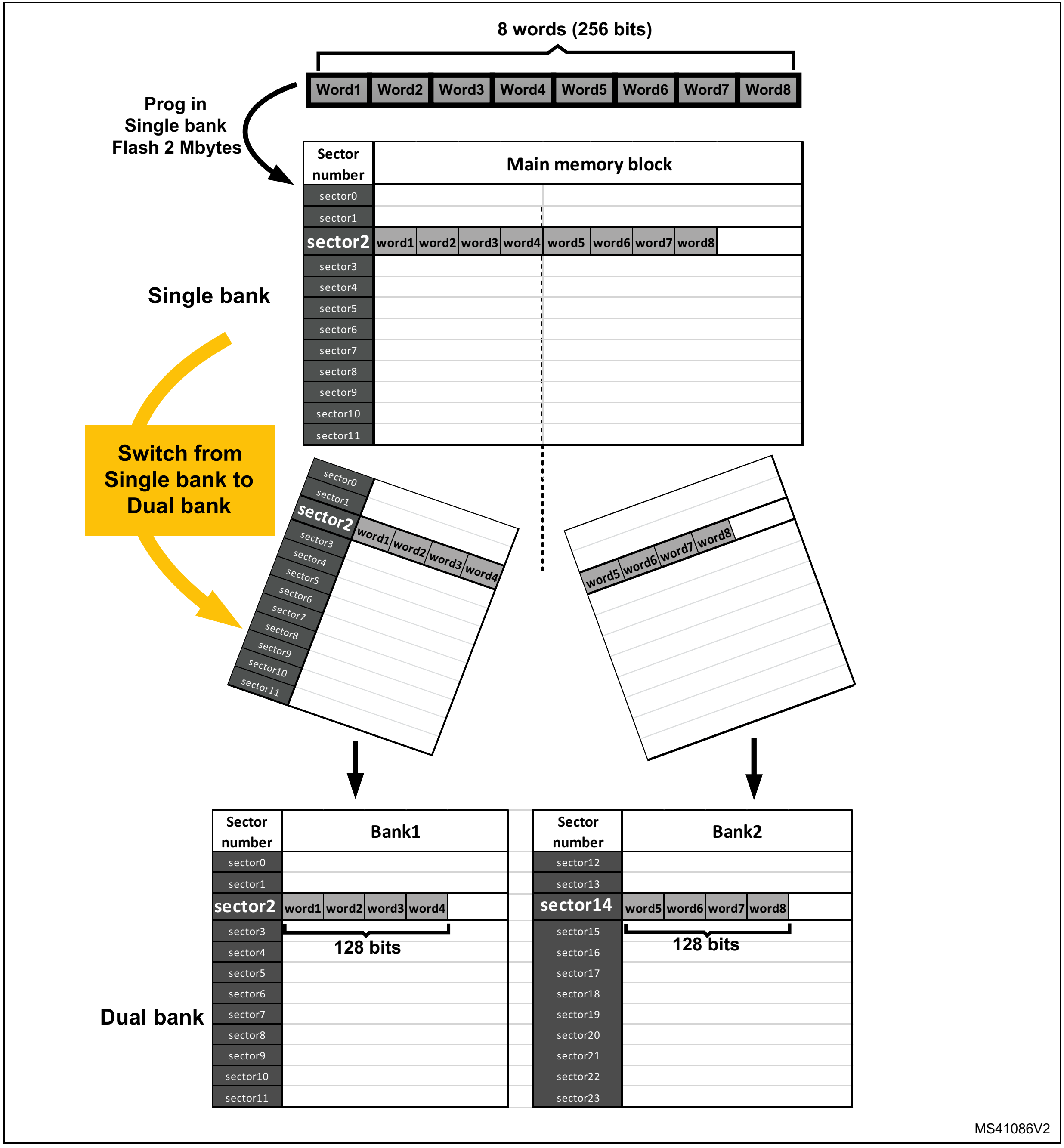


# 4 Switching from the single bank to the dual bank mode (or inversely)

## 4.1 Memory data organization

*Figure 6* describes the memory data organization when switching from the single bank to the dual bank mode:

Figure 6. Switch from the single bank to the dual bank mode

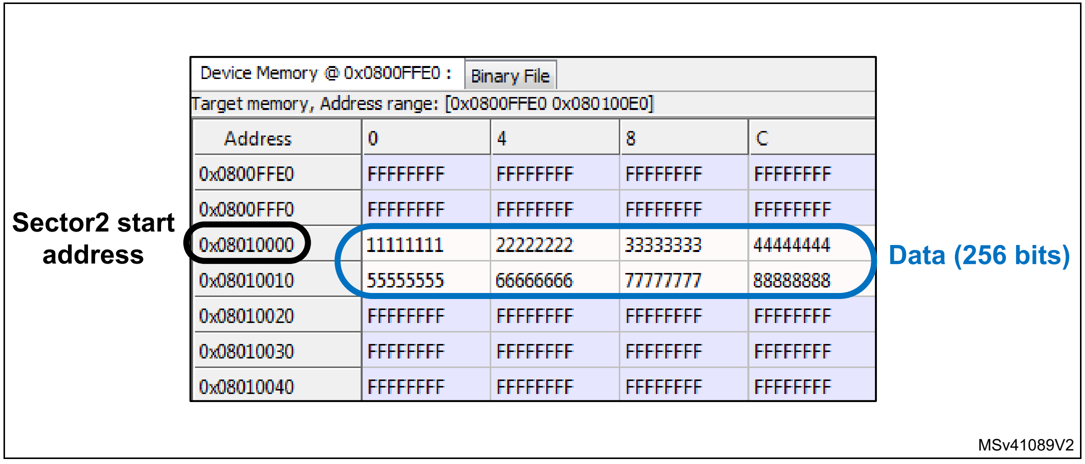

8 words (256 bits)
Word1 Word2 Word3 Word4 Word5 Word6 Word7 Word8
Prog in Single bank Flash 2 Mbytes
Sector number
Main memory block
sector0 … sector11
sector2 word1 word2 word3 word4 word5 word6 word7 word8
Single bank
Switch from Single bank to Dual bank
Dual bank
Bank1
Bank2
sector12 … sector23
sector14 word5 word6 word7 word8
128 bits
MS41086V2

## 4.2 Switching example

*Figure 7* and *Figure 8* show an example of switching from the single bank to the dual bank mode and reading the Flash memory content using the SLINK-Utility tool.

First the user programs 256 bits of constants in the sector 2 (0x08010000) of the 2 Mbyte Flash memory in the **single bank** mode (nDBANK=1).

**Figure 7. Single bank data organization example**

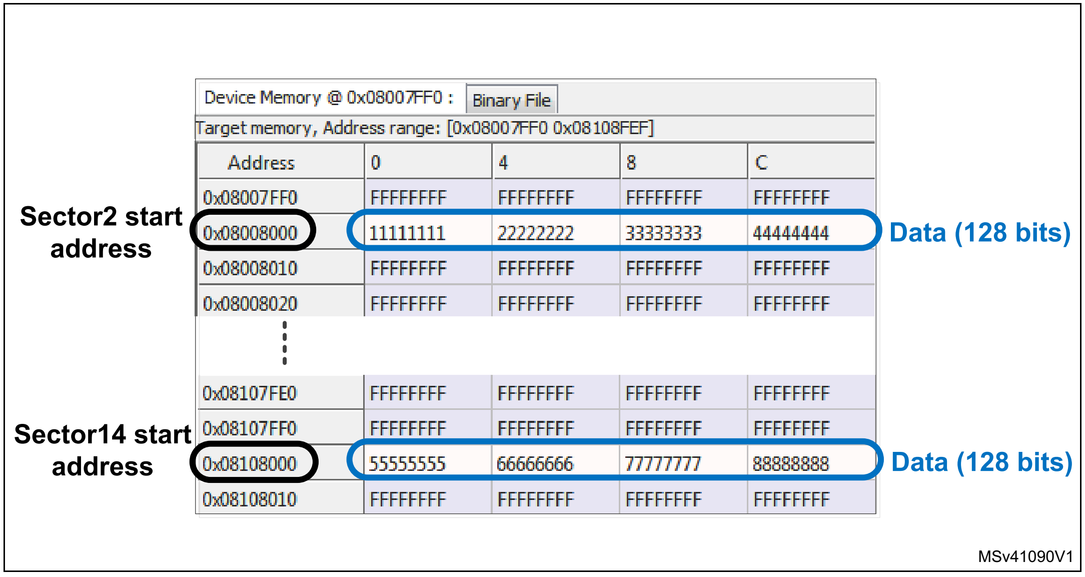


Target memory, Address range: [0x0800FFE0 0x080100E0]

| Address | 0 | 4 | 8 | C |
|---|---|---|---|---|
| 0x0800FFE0 | FFFFFFFF | FFFFFFFF | FFFFFFFF | FFFFFFFF |
| 0x0800FFF0 | FFFFFFFF | FFFFFFFF | FFFFFFFF | FFFFFFFF |
| 0x08010000 | 11111111 | 22222222 | 33333333 | 44444444 |
| 0x08010010 | 55555555 | 66666666 | 77777777 | 88888888 |
| 0x08010020 | FFFFFFFF | FFFFFFFF | FFFFFFFF | FFFFFFFF |
| 0x08010030 | FFFFFFFF | FFFFFFFF | FFFFFFFF | FFFFFFFF |
| 0x08010040 | FFFFFFFF | FFFFFFFF | FFFFFFFF | FFFFFFFF |

Then the user switches to the **dual bank** mode (nDBANK=0) and gets the programmed data in the Flash memory structured as follows: the 256 bits are split between bank1 and bank2, the first 128 bits being located at sector2 (0x0800 8000) and the next 128 bits at sector14 (0x0810 8000).

**Figure 8. Dual bank data organization example**



Target memory, Address range: [0x08007FF0 0x08108FEF]

| Address | 0 | 4 | 8 | C |
|---|---|---|---|---|
| 0x08007FF0 | FFFFFFFF | FFFFFFFF | FFFFFFFF | FFFFFFFF |
| 0x08008000 | 11111111 | 22222222 | 33333333 | 44444444 |
| 0x08008010 | FFFFFFFF | FFFFFFFF | FFFFFFFF | FFFFFFFF |
| 0x08008020 | FFFFFFFF | FFFFFFFF | FFFFFFFF | FFFFFFFF |
| ⋮ | | | | |
| 0x08107FE0 | FFFFFFFF | FFFFFFFF | FFFFFFFF | FFFFFFFF |
| 0x08107FF0 | FFFFFFFF | FFFFFFFF | FFFFFFFF | FFFFFFFF |
| 0x08108000 | 55555555 | 66666666 | 77777777 | 88888888 |
| 0x08108010 | FFFFFFFF | FFFFFFFF | FFFFFFFF | FFFFFFFF |

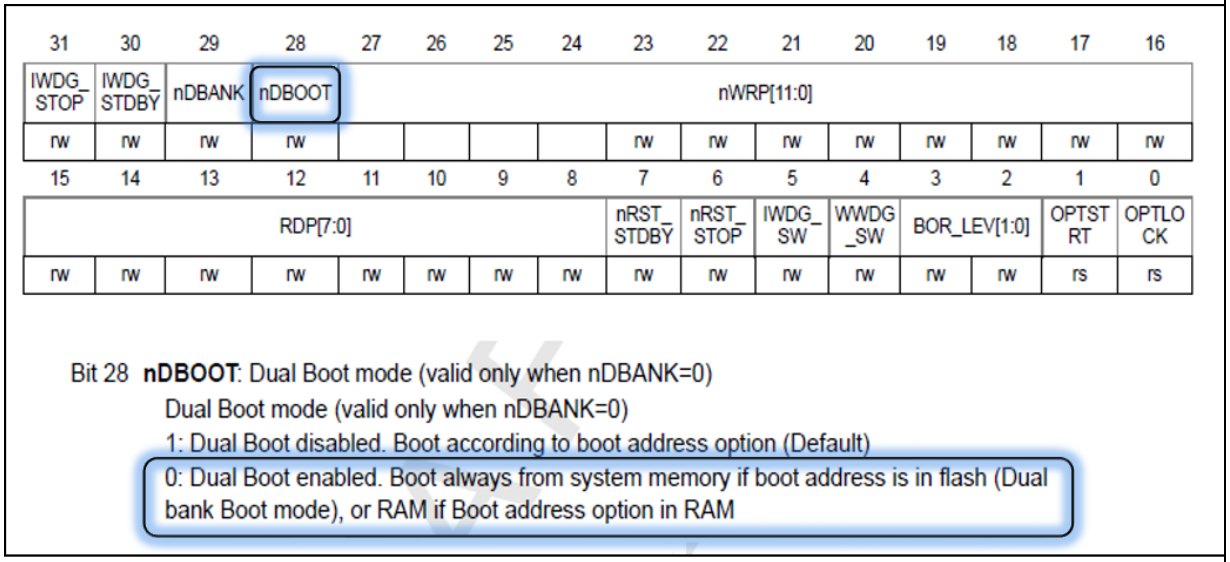
# 5 Dual boot

When the STM32F7 Series device is in the dual bank mode (nDBANK =0) the application software can either boot from bank 1 or from bank 2.

The dual boot Flash memory mode is activated by setting nDBOOT = 0 in user option bytes via the FLASH_OPTCR register.

*Figure 9* describes how the dual boot mode is activated at register level.

**Figure 9. How to activate the dual boot mode**



| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IWDG_ STOP | IWDG_ STDBY | nDBANK | nDBOOT | nWRP[11:0] | | | | | | | | | | | |
| rw | rw | rw | rw | | | | | rw | rw | rw | rw | rw | rw | rw | rw |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RDP[7:0] | | | | | | | | nRST_ STDBY | nRST_ STOP | IWDG_ SW | WWDG _SW | BOR_LEV[1:0] | | OPTST RT | OPTLO CK |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rs | rs |

Bit 28 **nDBOOT**: Dual Boot mode (valid only when nDBANK=0)

Dual Boot mode (valid only when nDBANK=0)

1: Dual Boot disabled. Boot according to boot address option (Default)

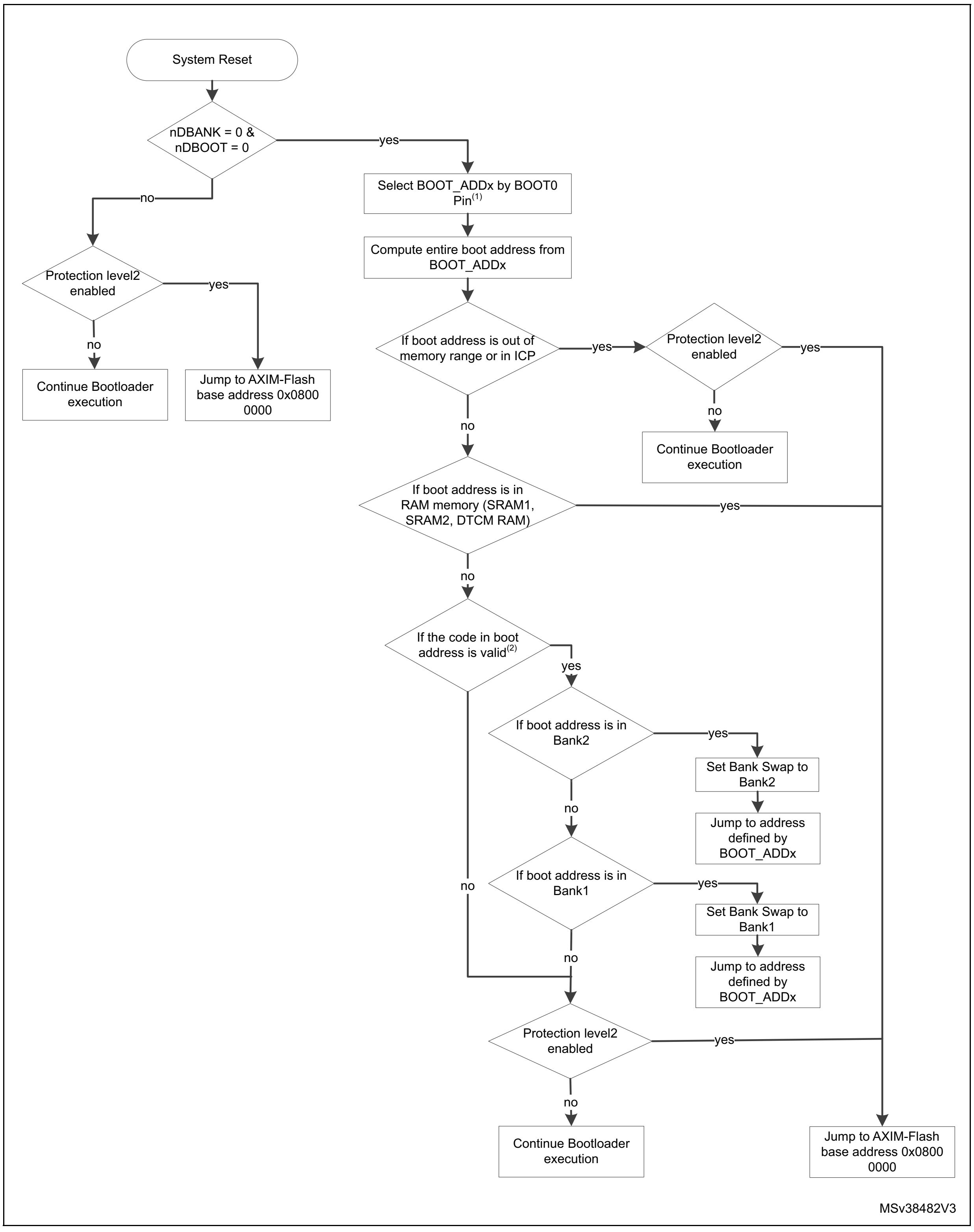
0: Dual Boot enabled. Boot always from system memory if boot address is in flash (Dual bank Boot mode), or RAM if Boot address option in RAM

## 5.1 Dual boot flowchart

When the STM32F7 Series device is in the dual bank mode (nDBANK =0) and nDBOOT =0 and the BOOT pin selects an address in the Flash bank1 or bank2 memory range, the device boots from the system memory, and the bootloader jumps to execute the user application programmed in the Flask memory bank1 or bank2.

The flowchart in *Figure 5* explains how the dual boot is implemented.

**Figure 10. Dual boot flow chart**


System Reset
nDBANK = 0 & nDBOOT = 0
yes
no
Select BOOT_ADDx by BOOT0 Pin(1)
Compute entire boot address from BOOT_ADDx
Protection level2 enabled
yes
no
Continue Bootloader execution
Jump to AXIM-Flash base address 0x0800 0000
If boot address is out of memory range or in ICP
yes
Protection level2 enabled
yes
no
Continue Bootloader execution
no
If boot address is in RAM memory (SRAM1, SRAM2, DTCM RAM)
yes
no
If the code in boot address is valid(2)
yes
If boot address is in Bank2
yes
Set Bank Swap to Bank2
Jump to address defined by BOOT_ADDx
no
If boot address is in Bank1
yes
Set Bank Swap to Bank1
Jump to address defined by BOOT_ADDx
no
no
Protection level2 enabled
yes
no
Continue Bootloader execution
Jump to AXIM-Flash base address 0x0800 0000
MSv38482V3

For further details, refer to the application note (AN2606).

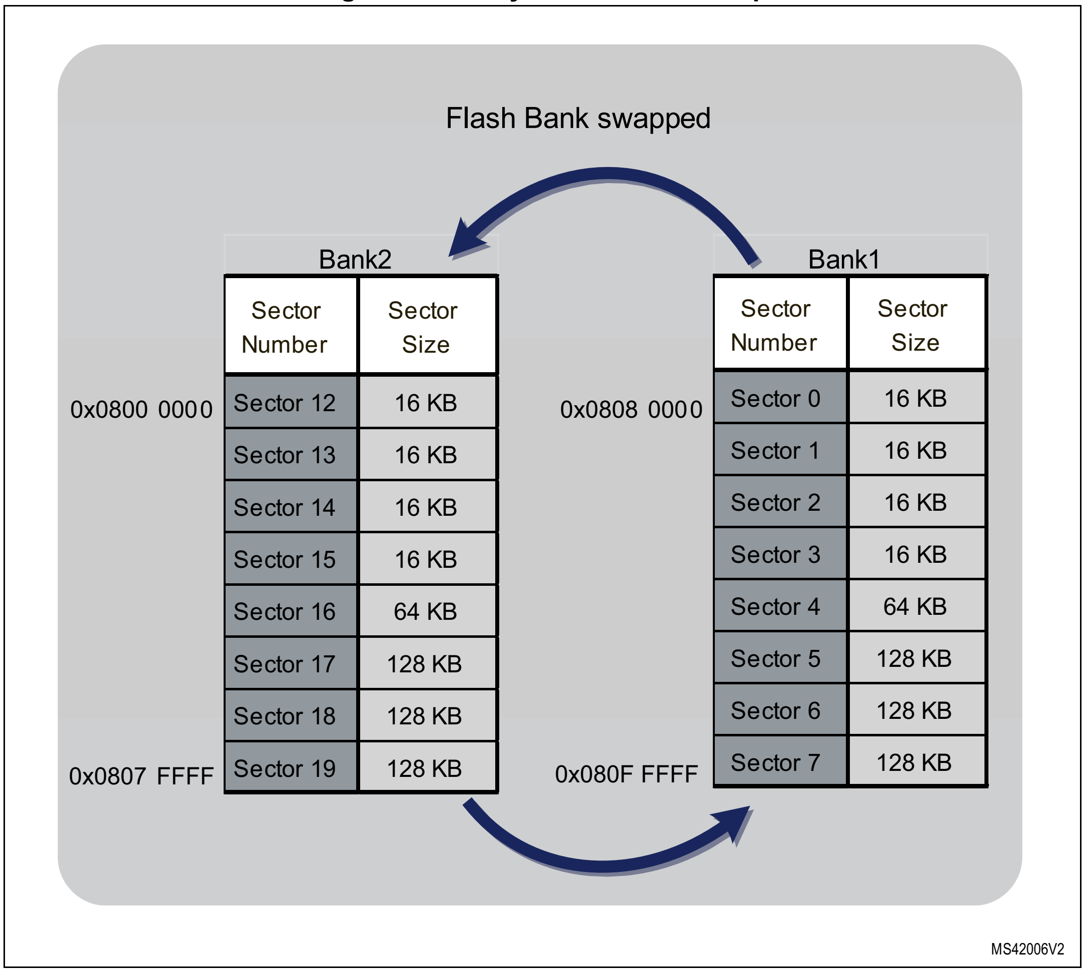
## 5.2 Flash bank swap:

Once the dual boot mode is activated and boot address is valid, the bootloader sets the Flash bank swap bit (SWP_FB =1) and it generates the following mappings.

### 5.2.1 1 Mbyte Flash bank swap

– Flash **bank2** base address mapped at **0x08000000** (AXI)
– Flash **bank1** base address mapped at **0x08080000** (AXI)

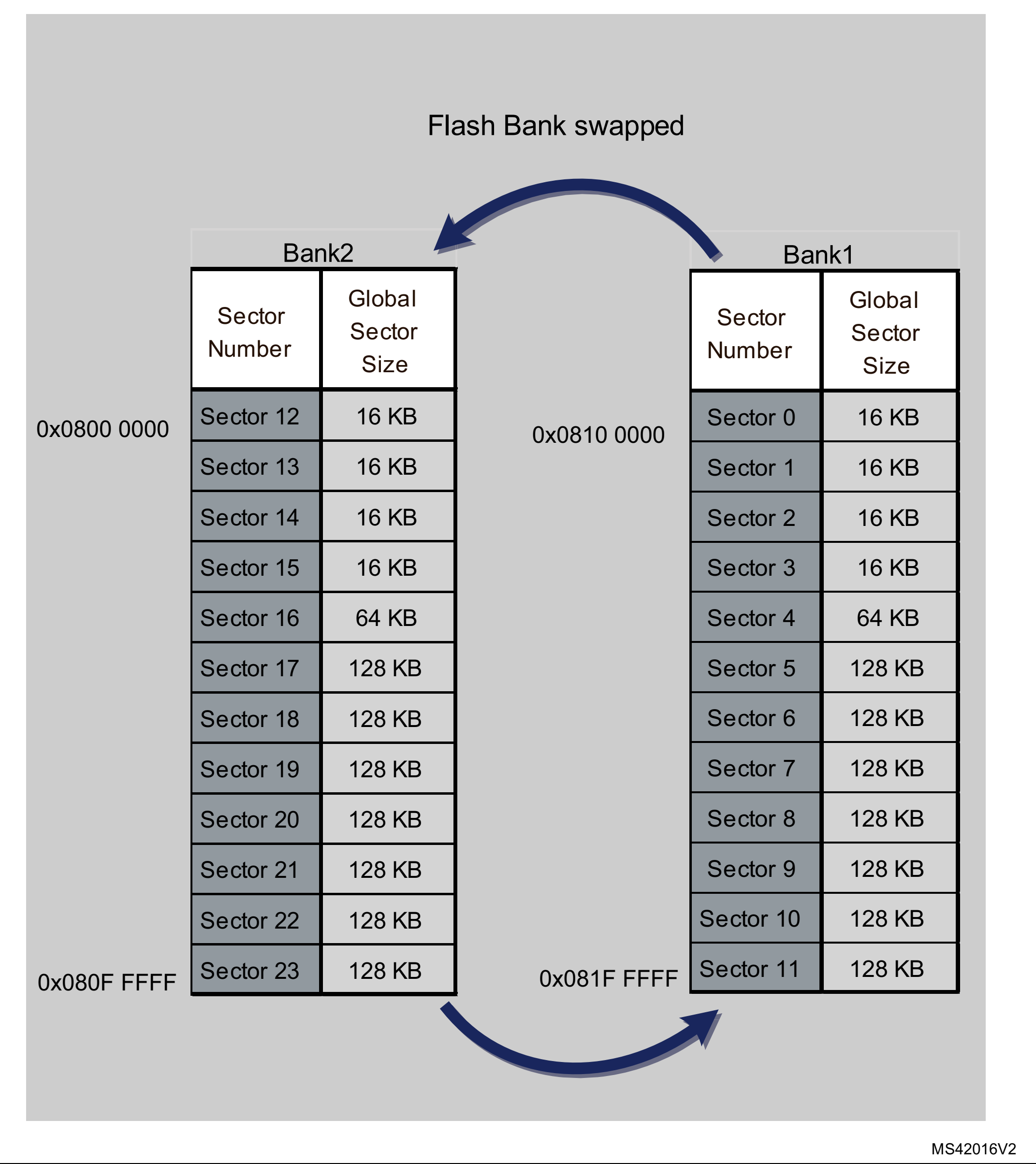
**Figure 11. 1 Mbyte Flash bank swap**



### 5.2.2 2 Mbyte Flash bank swap

– Flash **bank2** base address mapped at **0x08000000** (AXI)
– Flash **bank1** base address mapped at **0x08100000** (AXI)

**Figure 12. 2 Mbyte Flash bank swap**



Bank2

| Sector Number | Global Sector Size |
|---|---|
| Sector 12 | 16 KB |
| Sector 13 | 16 KB |
| Sector 14 | 16 KB |
| Sector 15 | 16 KB |
| Sector 16 | 64 KB |
| Sector 17 | 128 KB |
| Sector 18 | 128 KB |
| Sector 19 | 128 KB |
| Sector 20 | 128 KB |
| Sector 21 | 128 KB |
| Sector 22 | 128 KB |
| Sector 23 | 128 KB |

Bank1

| Sector Number | Global Sector Size |
|---|---|
| Sector 0 | 16 KB |
| Sector 1 | 16 KB |
| Sector 2 | 16 KB |
| Sector 3 | 16 KB |
| Sector 4 | 64 KB |
| Sector 5 | 128 KB |
| Sector 6 | 128 KB |
| Sector 7 | 128 KB |
| Sector 8 | 128 KB |
| Sector 9 | 128 KB |
| Sector 10 | 128 KB |
| Sector 11 | 128 KB |

## 5.3 Safety firmware upgrade using CRCs:

Several systems need to upgrade their firmware. This is done mainly using a bootloader. However, during the firmware upgrade some problems might happen, for instance a power loss during the upgrade process. Therefore, the new firmware is not installed properly and it generates an issue in the behavior of the system.

Among the techniques for enforcing safety, the CRCs are used to verify the data transmission or the storage integrity. They can be used to verify the Flash memory integrity.

The CRC calculation unit helps compute a signature over the data to send. This signature is then transmitted along with the original data to the Flash Bank2 for example, while the original firmware is executing from bank1.

When the target receives the data, the new signature is computed and compared to the one received. If both are equal the bootloader can continue to upgrade the process safely and then to swap to bank2 to execute the new firmware.

## 5.4 Write protections:

- **Single bank configuration:**

  The user sectors (sector 0 to sector 11) can be protected with the following scheme:

  nWRP[i] bit is write protection bit for sector(i).

  When the Not Write Protection is active for one of the sectors, the sectors cannot be neither erased nor programmed. Consequently, a mass erase cannot be performed.
- **Dual bank configuration:**

  The user sectors of bank 1 (sector 0 to sector 11) and bank 2 (sector 12 to sector 23) can be protected with the following scheme:

  nWRP[i] bit is write protection bit for sector(2*i) and sector (2*i+1).

  When the Not Write Protection is active for one of the sector pairs, the pairs of sectors canot be neither erased nor programmed. Consequently a mass erase, or bank erase, cannot be performed.

## 5.5 Software setting tips:

The user must keep in mind some tips for the programming of the Flash single bank or the Flash dual bank.

For example: about the programming of the Flash memory with 20 Kbytes of code:

- **First configuration: Project configured in the single bank mode and nDBANK = 0**

  As the project is configured in the single bank mode, the user must erase 1 sector (32 Kbytes), but with **nDBANK = 0** the Flash interface is configured in the dual bank mode, therefore 1 sector = 16 Kbytes. The result is that only 16 Kbytes are erased and 4 Kbytes are missed.

Consequently, the programming of 20 Kbytes cannot be performed and a programming error appears.

- **Second configuration: Project configured in the dual bank mode and nDBANK= 1**

  As the project is configured in the dual bank mode, the user must erase 2 sectors (2 x16K bytes), but with **nDBANK = 1** the Flash interface is configured in the single bank mode, therefore 1 sector = 32 Kbytes. The result is that 2 sectors (2 x 32 Kbytes) are erased while only 20 Kbytes are needed. Consequently, extra sectors (2 x 16 Kbytes) are erased.

When programming the Flash memory the user must be careful that the project configuration and the Flash option bytes are aligned.

# 6 Examples

This section describes the X-CUBE-DBANK-F7 embedded software examples, highlighting the Flash dual bank mode capabilities.

## 6.1 Read-while-write (RWW) example

### 6.1.1 Description

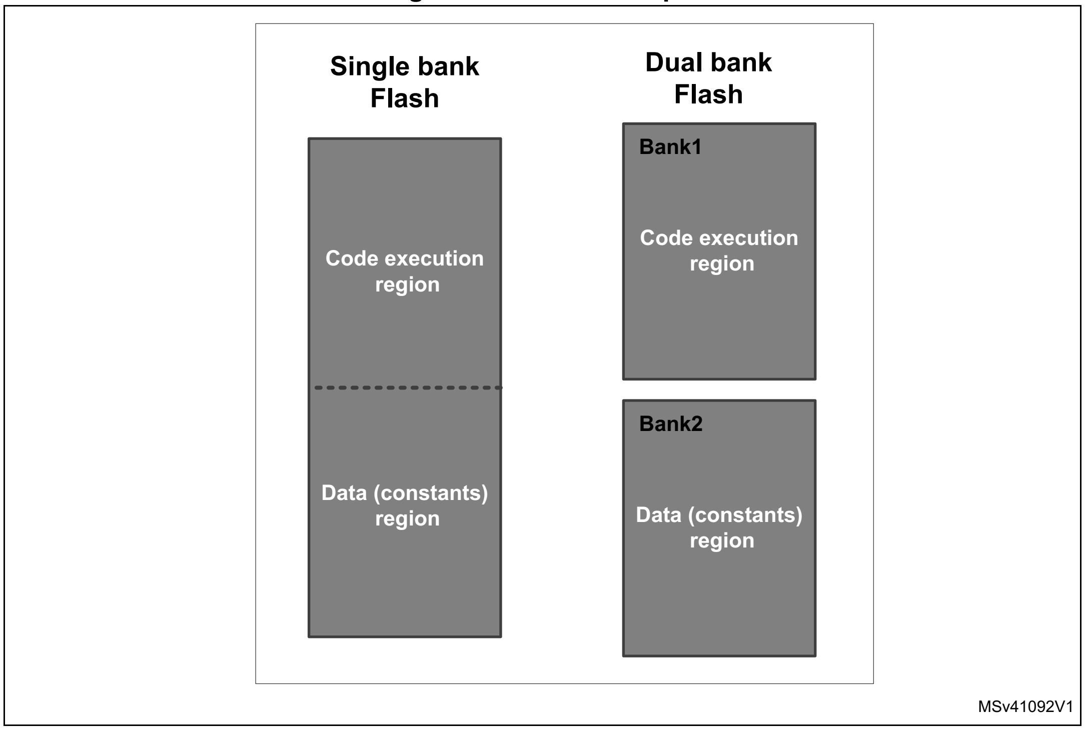
This example demonstrates how the **read-while-write** feature works correctly with the dual bank Flash configuration. This feature does not apply to the single bank Flash configuration.

In this example the user programs the Flash memory with data (constants) while executing a LED toggling.

The first case shows how to use the Flash memory in the single bank configuration, in order for the programmed constants and the executed code to be within the same Flash bank.

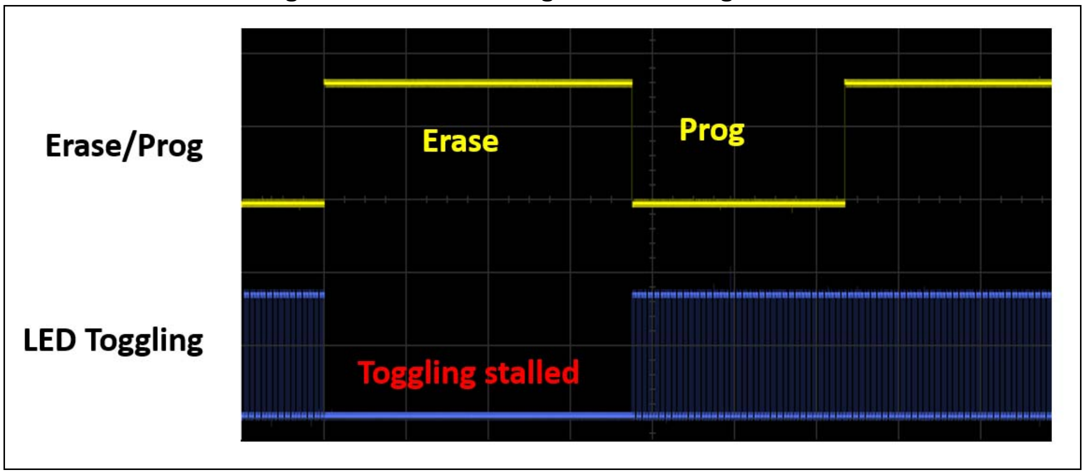
The second case shows how to configure the Flash memory in the dual bank configuration and enables the user to storage the constants and execute the code in separate banks.

Figure 13. RWW example



### 6.1.2 Single bank configuration

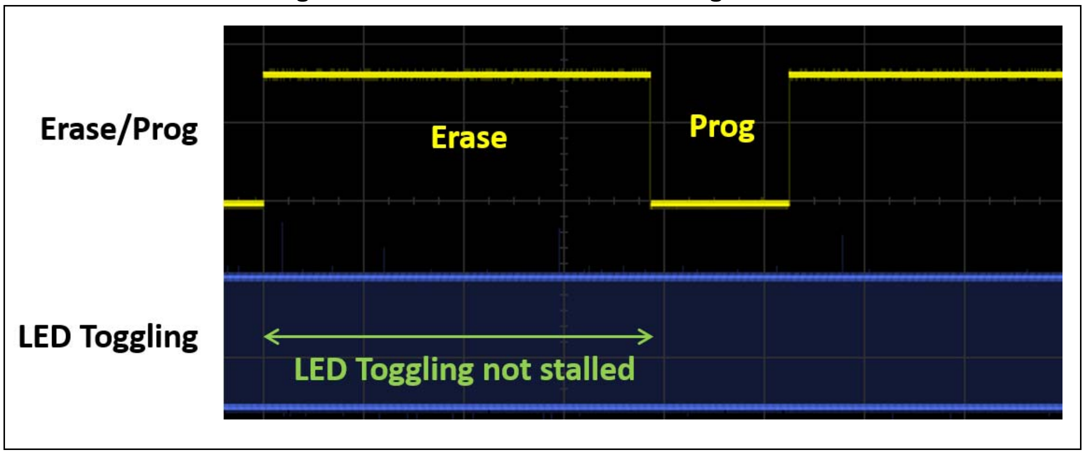
Figure 14. RWW in single bank configuration



The example waveforms above demonstrate how the execution (LED toggling) is stopped when an erase operation starts while the CPU is executing a code from the Flash memory. As described in *Section 3: Read-while-write (RWW)* this example proves that executing and programming/erasing from the Flash memory in the single bank mode is not allowed.

### 6.1.3 Dual bank configuration

Figure 15. RWW in dual bank configuration



In the Flash dual bank mode the LED toggling is executed from bank1 while constants are being programmed in bank2. Therefore, the CPU execution is not stalled, as shown in *Figure 15*.

### 6.1.4 EEPROM emulation

The EEPROM emulation consists of using a STM32F7 Series device with two banks of Flash memory: one accessed as a Flash memory, the other accessed as an EEPROM. For most devices, this avoids the need of the emulation software and also allows read-while-write (RWW) operations, thus reducing the access latency.

An application example of the EEPROM emulation is developed and can be loaded from the STM32CubeF7 MCU Package.

## 6.2 Dual boot example

### 6.2.1 Description

The dual boot example describes how the STM32F7 Series device boots either in bank1 or bank2 depending on a valid address.

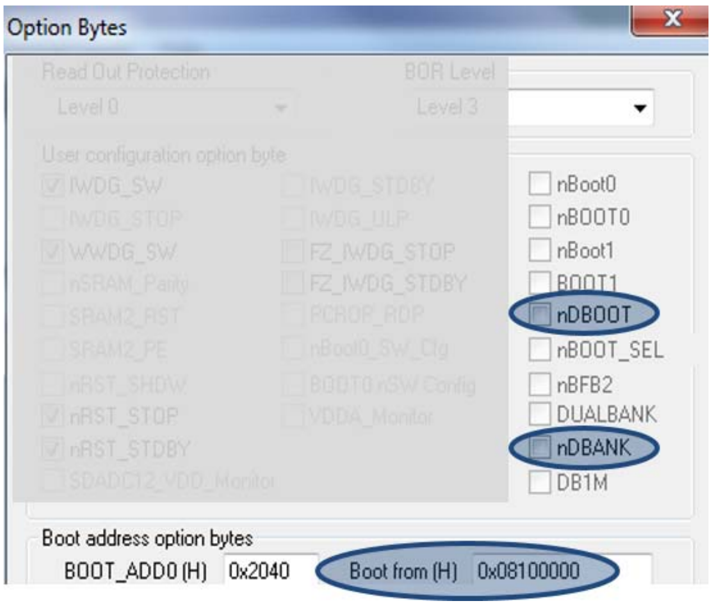
The code in bank1 toggles the LED1 and the code in bank2 toggles the LED2.

To run this example the user must follow the following steps:

- Configure the Flash memory in the dual boot mode (nDBANK=nDBOOT=0) using the STM32 ST-LINK Utility.
- Build the **FLASH_DualBoot_Bank2** project and load the binary file **FLASH_DualBoot_Bank2.bin** at the address 0x08100000 using the STM32 ST-LINK Utility.
- Build the **FLASH_DualBoot_Bank1** project and load it to the address 0x08000000.

– If the boot address is 0x081000000, the Flash bank is swapped and the bootloader jumps to bank2 to run the LED2 toggling.

**Figure 16. Boot from bank2**



Option Bytes

Read Out Protection: Level 0
BOR Level: Level 3

User configuration option byte

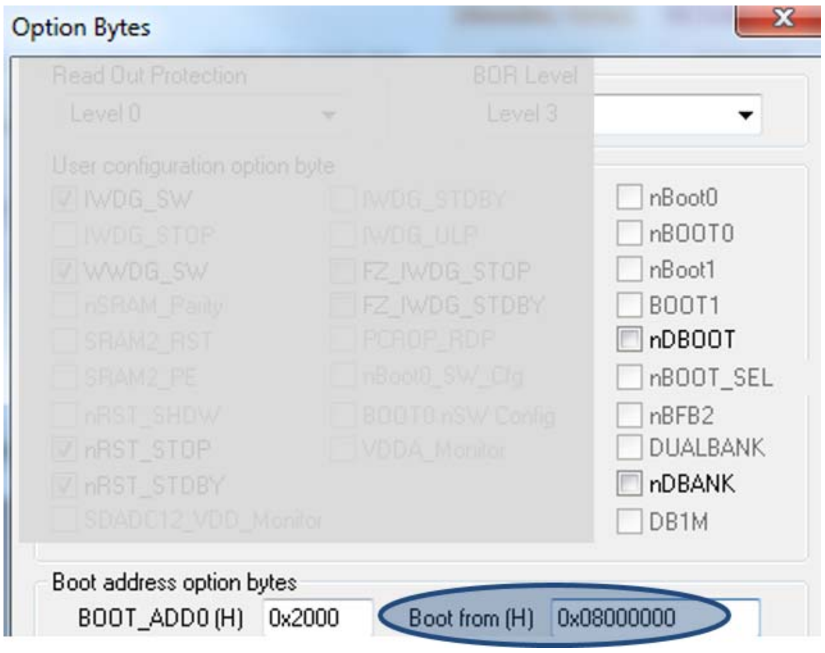
| | | |
|---|---|---|
| ✓ IWDG_SW | IWDG_STDBY | nBoot0 |
| IWDG_STOP | IWDG_ULP | nBOOT0 |
| ✓ WWDG_SW | FZ_IWDG_STOP | nBoot1 |
| nSRAM_Parity | FZ_IWDG_STDBY | BOOT1 |
| SRAM2_RST | PCROP_RDP | nDBOOT |
| SRAM2_PE | nBoot0_SW_Cfg | nBOOT_SEL |
| nRST_SHDW | BOOT0 nSW Config | nBFB2 |
| ✓ nRST_STOP | VDDA_Monitor | DUALBANK |
| ✓ nRST_STDBY | | nDBANK |
| SDADC12_VDD_Monitor | | DB1M |

Boot address option bytes

BOOT_ADD0 (H) 0x2040 Boot from (H) 0x08100000

– If the boot address is 0x080000000, the bootloader jumps to the boot address to run the LED1 toggling.

**Figure 17. Boot from bank1**



Option Bytes

Read Out Protection: Level 0
BOR Level: Level 3

User configuration option byte

| | | |
|---|---|---|
| ✓ IWDG_SW | IWDG_STDBY | nBoot0 |
| IWDG_STOP | IWDG_ULP | nBOOT0 |
| ✓ WWDG_SW | FZ_IWDG_STOP | nBoot1 |
| nSRAM_Parity | FZ_IWDG_STDBY | BOOT1 |
| SRAM2_RST | PCROP_RDP | nDBOOT |
| SRAM2_PE | nBoot0_SW_Cfg | nBOOT_SEL |
| nRST_SHDW | BOOT0 nSW Config | nBFB2 |
| ✓ nRST_STOP | VDDA_Monitor | DUALBANK |
| ✓ nRST_STDBY | | nDBANK |
| SDADC12_VDD_Monitor | | DB1M |

Boot address option bytes

BOOT_ADD0 (H) 0x2000 Boot from (H) 0x08000000

## 6.3 Performance and consumption example

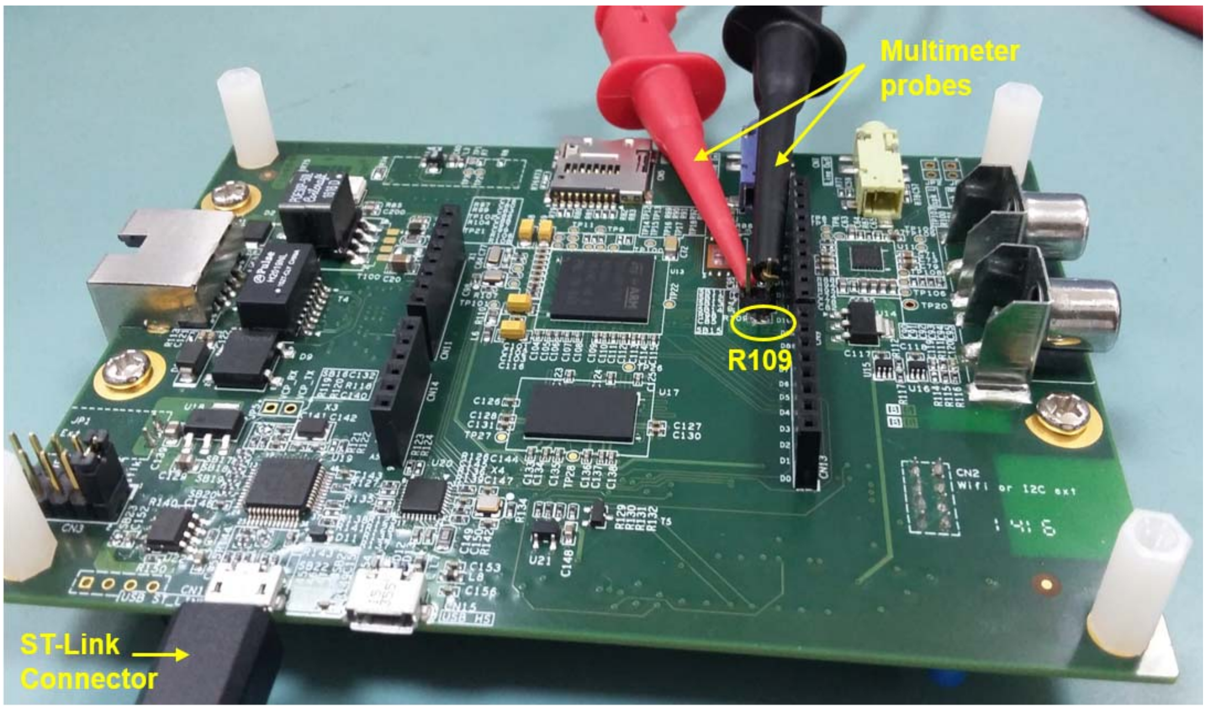
### 6.3.1 Hardware requirement

To run the examples the user needs:

- The 32F769IDISCOVERY board.
- To connect the board to PC through ST-LINK with a micro USB for programming and debugging.
- A multimeter to measure the current consumption.

To measure the current consumption of the microcontroller the user needs to remove the resistance R109 from the Discovery board, as illustrated in *Figure 18*:

**Figure 18. Hardware connection**



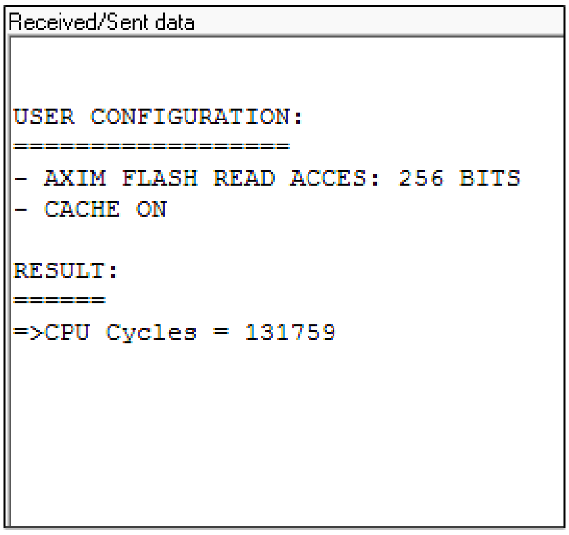
### 6.3.2 Measure results

The performance and consumption of the STM32F7 Series device with the Flash memory in the single and dual bank mode are measured running the “CMSIS Arm$^{®}$ graphic equalizer” algorithm. The toolchain used is the MDK-RAM.

- Performance measurement results:

The performance values are displayed on the terminal, as shown in *Figure 19*:

Figure 19. Performance display



```
USER CONFIGURATION:
==================
- AXIM FLASH READ ACCES: 256 BITS
- CACHE ON

RESULT:
======
=>CPU Cycles = 131759
```

*Table 2* summarizes the STM32F7 performance measures for the CMSIS Arm® graphic equalizer:

Table 2. Performance measures

| Flash configuration | | Performance measures | | |
|---|---|---|---|---|
| | | **Single bank (CPU cycles)** | **Dual bank (CPU cycles)** | **Gain** |
| Flash_AXI | Cache ON | 131759 | 131760 | 0% |
| | Cache OFF | 365656 | 442481 | - 21% |
| Flash_ITCM | ART ON | 137989 | 154146 | - 12% |
| | ART OFF | 236302 | 320732 | - 36% |

- Consumption measurement results

*Table 3* summarizes the STM32F7 consumption measures:

Table 3. Consumption measures

| Flash configuration | | Consumption measures | | |
|---|---|---|---|---|
| | | **Single bank (mA)** | **Dual bank (mA)** | **Gain** |
| Flash_AXI | Cache ON | 111 | 111 | 0% |
| | Cache OFF | 96 | 85 | 11% |
| Flash_ITCM | ART ON | 117 | 114 | 3% |
| | ART OFF | 118 | 95 | 19% |

Based on the measures in *Table 3* and *Table 4*, the switching from the single to the dual bank mode shows that:

- If the cache and ART are OFF, the performance is degraded, and there is a power consumption gain in the dual bank mode.
- If the cache and ART are ON, the degradation in performance is almost masked (bit degradation with ART ON) and there is still a power consumption gain in the dual bank mode.

In the dual bank mode a power consumption can be saved depending on the user application and thanks to the STM32F7 ART and cache, the same performance is maintained approximately for both single and dual bank modes.

# 7 Conclusion

This application note demonstrates that the Flash dual bank mode responds to the strict real-time needs of a wide range of applications and resolves many issues, such as reducing the access latency thanks to the read-while-write feature.

The use of the dual bank memory makes it possible to have at least one working version of the firmware in the STM32F7 Series device at any time. This is useful to avoid a firmware corruption if a problem occurs during an upgrade, such as a power loss or a connection loss.

# 8 Revision history

Table 4. Document revision history

| Date | Revision | Changes |
|---|---|---|
| 13-Jun-2016 | 1 | Initial release. |
| 14-Oct-2019 | 2 | Added *Section 1: General information*.<br>Updated *Section 6.3.2: Measure results*. |

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to *www.st.com/trademarks*. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2019 STMicroelectronics – All rights reserved