AN4767
Application note

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

Introduction

Dual bank functionality is a feature common to several STM32 microcontrollers. The goal of this document is to describe how to take advantage of this feature in customer applications.

The main topic treated in this application note is the Field Upgrade, covered also by the provided code example (X-CUBE-DBFU).

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.

The following documents, all available on *www.st.com*, are considered as reference:

- Reference manual RM0367: “Ultra-low-power STM32L0x3 advanced ARM®-based 32-bit MCUs”
- Reference manual RM0376: “Ultra-low-power STM32L0x2 advanced ARM®-based 32-bit MCUs”
- Reference manual RM0377: “Ultra-low-power STM32L0x1 advanced ARM®-based 32-bit MCUs”
- Application note AN2606: “STM32 microcontroller system memory boot mode”
- Application note AN4024: “STM32 secure firmware upgrade (SFU)”
- Application note AN4657: “STM32L0 in-application programming using UART”

Contents

List of tables

List of figures

1 Definitions

Table 1. List of acronyms

Term	Description
CPU	Central processing unit (part of the MCU)
EEPROM	Electrically erasable programmable read only memory
IAP	In-application programming
MCU	Microcontroller
NVIC	Nested vector interrupt controller
NVM	Non-volatile memory (EEPROM or Flash based)
RM	Reference manual
UART	Universal asynchronous receiver transmitter
USART	Universal synchronous and asynchronous receiver transmitter
VTOR	Vector table offset register

2 Dual bank use cases

When designing an application that uses a dual bank device, there are several choices to make on how to utilize the second half of the program memory.

2.1 Big code

A first case is an application with a code that needs both banks for a single instance of the program.

Even in this case the dual bank approach may be beneficial. It is still possible to think of two programs with different set of functions loaded separately in the two banks, implementing two operation modes (among them, master/slave, recorder/player, transmitter/receiver).

2.2 Data storage

The second bank may be advantageous used for data storage (example: data logging).

In this case the dual bank can be used to enable the “Read while Write” feature, thus avoiding the CPU stall when the data are programmed into the other bank. This approach also makes mass erase of data stored in the second bank easier and faster.

2.3 Code backup

If firmware upgrade scenario is not considered (i.e. device is not accessible), then the second bank may be used for code backup. Bank2 may store identical code and serve as a backup in case of minor Flash memory damage (for example by radiation and heat).

It is relatively easy to implement bank switch with subsequent code integrity checks as a reaction to a non-handled exception.

As a preventive measure, integrity of both copies may be periodically checked, and, in case of failure, the faulty copy can be rewritten with the correct one.

2.4 Dual bank field upgrade

Usage for Field upgrade purposes is the main topic of this application note.

There are several advantages on top of the standard in-application programming, described in AN4657. With dual bank approach, all the code is kept in a single project, resulting in a single binary. This eliminates the need of implementing an application loader with limited functionality, to whom the device must rely on during application programming.

Typical drawbacks of traditional IAP are:

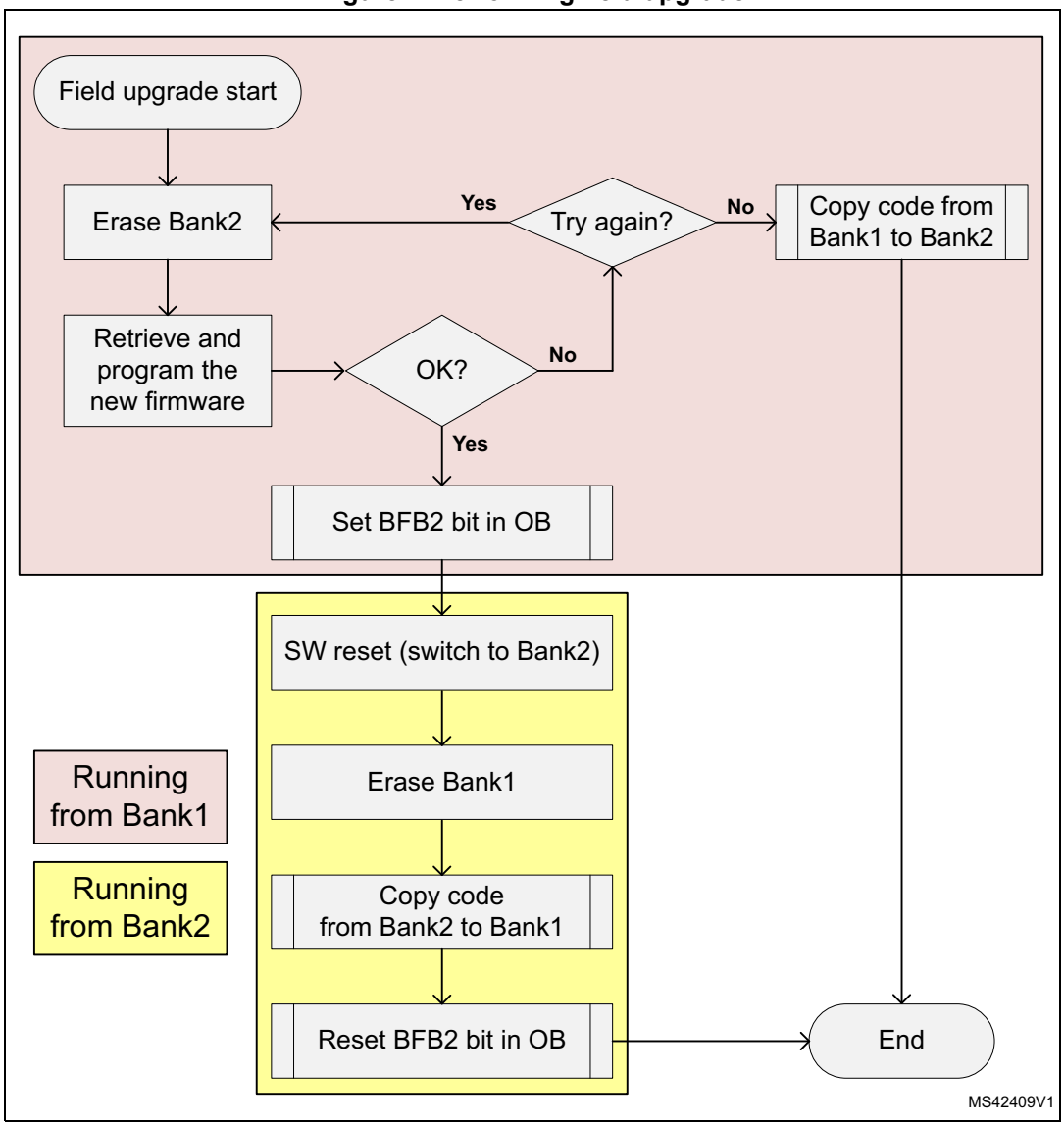
1. the loader is not capable of performing all the background tasks
2. the loader cannot be upgraded or fixed
3. in case of failure during the upgrade process, the device may remain in loader mode
4. no option to rollback to previous version in case of problems.

All these problems are addressed and solved by correct use of dual bank approach.

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.

The typical scenario to perform the field upgrade is shown in *Figure 1*.

Figure 1. Performing field upgrade



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.

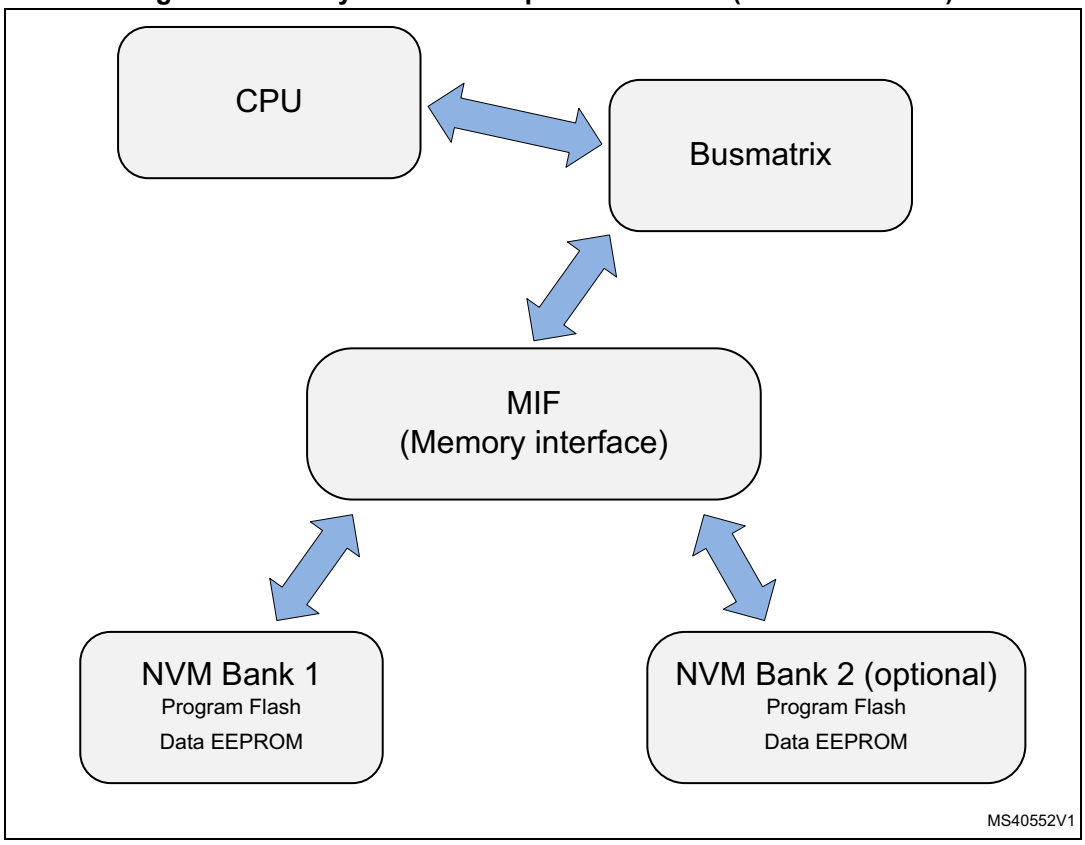
3 Memory implementation summary

The NVM in the STM32L0 and STM32L4 Series is split in several areas:

- User code memory (Program Flash)
- System code memory
- Option bytes
- User data memory (Data EEPROM), only for the STM32L0 Series

The memories to which the dual bank feature is applicable (i.e. those that are found in two instances on Dual bank capable devices) are User code memory and User data memory.

Figure 2. Memory interface simplified overview (STM32L0 Series)



The memory interface is capable of reading both banks in parallel, or can read one bank while writing the other. No such parallelism is possible within one bank, hence it is more efficient for application to store the data in the opposite bank.

The most efficient EEPROM storage solution (for STM32L0) is when the code in Bank1 uses data EEPROM from Bank2 and code from Bank2 uses data EEPROM from Bank1 (same is true when program memory of the other bank is written).

3.1 Manual bank selection

When the device boots to code in the main Flash memory, bank mapping and aliasing can be dynamically changed using the SYSCFG registers.

By flipping the bank swap bit (UFB flag in the SYSCFG_CFGR1 register for L0) or bank mode (FB_MODE flag in the SYSCFG_MEMRMP register for L4) it is possible to pass the execution to the other bank. Value of the flag also indicates which bank is currently being used.

If the code in both banks is identical, the PC and stack values will remain valid and the program execution can continue after the mapping change occurs.

If the code in both banks is not identical (or at least the section prior to the bank switch point is not identical) then the execution is most likely to crash.

It is also important to relocate the vector table before doing the switch (see *Section 3.3*).

3.2 Automatic bank selection

On reset, the BFB2 flag in option bytes is evaluated, along with the state of BOOT0 pin and BOOT1 flag. When boot into main Flash memory is selected by the BOOT0 value and the BFB2 bit is set, the system memory bootloader is called to evaluate the contents of the Bank2. If the first word of the Bank2 memory contains valid stack pointer address, the system bootloader assumes the bank is loaded with meaningful code and Bank2 is then aliased to the address range starting 0x0800 0000, instead of Bank1.

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.

The automatic bank selection is a process free of side-effects and adds no more than a few microseconds to the startup time.

3.3 Vector table

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

4 Example project

An example code is described in the following, demonstrating firmware upgrade use case. It is closely related to an IAP application, with only one difference, there is only one firmware, identical for both banks, serving as both the loader and the application.

4.1 HW setup

Connect USART2 port to a PC using a RS-232 serial cable. Use a terminal application that supports YMODEM protocol. Tera Term is offering a solid, cost free alternative to Windows HyperTerminal. Example functionality has been tested with Tera Term version 4.84. Configure communication speed of 115200 Bd, 8 data bits, no parity, 1 stop bit and then power the board.

4.2 Encryption option

It is understood that IP contained in the firmware upgrade file may be precious to the owner and there may be concerns regarding the code confidentiality. Combined with integrity protection, even symmetric encryption provides a basic defense against introducing counterfeit or fraudulent firmware.

Several ST microcontrollers include optional AES HW accelerator peripheral option. In this case, it is possible to use STM32L486 instead of STM32L476 and STM32L083 instead of STM32L073 in the Eval board to use the encryption functionality.

4.2.1 Encrypting the binary

The encryption scheme used in this example is a plain AES used in counter (CTR) mode.

The advantage of this method is that no padding is necessary, only the payload is encrypted. While it is not relevant to field upgrade case, the counter mode is also known for not propagating the communication errors (only the bytes that get corrupted during communication are corrupted in the decrypted message).

Furthermore, there is no need to develop dedicated encryption utility, any publicly available encryption library is capable of generating counter mode scratchpad and make XOR between the scratchpad and the data.

In case the user is not sure on how to prepare the inputs on a computer, the steps indicated below must be followed:

1. Download the open source library Crypto++® (www.cryptopp.com)
2. Install GCC free tools preferably Cygwin since MinGW is not supported.
3. Open Cygwin.bat and use the command *make* to build Cryptopp library.
4. Test the library using the*./cryptest.exe v* (v option: the Validation Suite)
5. If all tests are passed, the message “All tests passed!” will be displayed:
6. Generate a binary file and place it under Cryptopp library
7. Call the cryptest.exe with parameters
   cryptest.exe ae <HexKey> <HexIV> Project.bin Firmware.aes
   A batch file is included along with the project to remove any doubts about key format.

The *cryptest.exe* will generate an *.aes* file that will be used on devices with AES peripheral (STM32L083/STM32L486).

4.2.2 Configuring for decryption

Add or uncomment *#define ENCRYPT* in the project. It will only work on devices with AES peripheral (STM32L083/STM32L486 in this case).

Replace the placeholder key and initialization vector with any other value in *main.c* to personalize the project.

4.2.3 Limits of the simple solution

The basic symmetric encryption solution used in this example has some limitations.

It uses the same key for all the devices in the field. Widespread use of single key makes the cryptographic system more vulnerable. Not only it is giving more opportunities to the attacker to obtain the secret key, but once the key is exposed, all the devices are open. This effect can be partially reduced by deriving the key using some property of the code, for example version.

Also, the authentication relies only on the fact of knowing the secret key, not completely mitigating the possibility to provide not genuine code.

The fact that first several bytes of the code (consisting of initial stack pointer and vector table) are mostly predictable (the “known plaintext attack”) can lead to weakening of the cryptographic properties. This weakness can be easily addressed by adding a random sequence (“salt”) to the “plaintext” before encryption. Then the “salt” is ignored after decryption.

4.2.4 Other cryptography options

Protection of the code in not only matter of the encryption algorithm, but rather a complex cryptographic system, encompassing security requirements implemented in all product life phases, from design and development, through manufacturing and servicing to deactivation and recycling.

Authentication roles and integrity protection must also be seriously considered.

STMicroelectronics is capable of providing top-tier embedded security solutions.

For more details refer to AN4024 or contact your nearest ST sales office.

4.3 Example operation

With user interface provided by the terminal SW the operation is easy. The example firmware first displays a header and a menu. Pressing numbers on PC keyboard selects choices.

1. Download a file to the other bank

   First menu choice initiates download of a binary file into the other bank. The former content of the other bank is erased and overwritten. The file size is constrained by the memory size. Select the file to download, it will be decrypted and written. Communication proceeds using the YMODEM protocol.

2. Erase the contents of the other bank

   This option invalidates the content of the other bank.

3. Rewrite the content of the other bank

   Code from the active bank is copied to the opposite memory bank.

4. Check the other bank integrity

   This option initiates check of the other bank content.

   With L0 Series the example FW stores the code CRC integrity value in the EEPROM Data memory. In case of L4 devices the check is limited to the simple presence.

5. Switch bank

   If there appears to be a functional code in the other bank, the vector table is rewritten and the banks are switched (as described in *Section 3.1: Manual bank selection*).

6. Toggle the system bank selection

   Programming the option bytes to modify the value of the BFB2 bit (boot from Bank2, as described in *Section 3.2: Automatic bank selection*).

5 Conclusion

The dual bank feature, when used properly, holds number of advantages for a broad range of applications.

The user can exploit these advantages by selecting devices featuring a double bank memory, this may require use of products with higher cost and larger package.

6 Revision history

Table 2. Revision history

Date	Revision	Description of changes
29-Sep-2016	1	Initial release.
31-Oct-2016	2	Updated document title.