GigaDevice Semiconductor Inc.

GD32F5xx Dual-bank Switch

Application Note

AN195

Revision 1.0

(Apr. 2024)

Table of Contents

List of Figures

List of Tables

1. Introduction

This application note is designed for the GD32F5xx dual-bank flash product. It introduces the dual-bank code update and switch function. This application note is divided into two parts. The first part introduces the implementation principle of dual-bank switching, and the second part introduces the software implementation method.

- Evaluation board: GD32F527I-EVAL board
- Debugger: J-Link or GD-Link
- IDE: Keil 5.35

2. Dual-bank switching principle

GD32F5xx flash memory structure is divided into 4MB dual-bank, 2MB dual-bank, 1MB single bank, and 512KB single bank, each of which has bank1 extended flash (Bank1_Ex). The Bank1_Ex starts at 0x08400000 and operates in the same way as bank1. When the GD32F5xx dual block structure is adopted, bank0 is used for the first 2048KB and bank1 is for the rest capacity. No waiting time within first 2048K bytes when CPU executes instructions (in case that flash size less than 2048K, all memory is no waiting time). A long delay when CPU fetches the instructions out of the range. The 4MB dual-block product is used in this application note.

The SYSCFG_CFG0 register provides the FMC_SWP bit which can control the address mapping switching between bank0 and bank1 of the main flash memory. When the bit is set to 0, the bank0 of the main flash memory is mapped to the address 0x08000000, and the bank1 of the main flash memory is mapped to the address 0x08100000 (2M products swap 0x08000000 and 0x08100000, 4M products swap 0x08000000 and 0x08200000). When the bit is set to 1, the bank1 of the main flash memory is mapped to the address 0x0800 0000, and the bank0 of the main flash memory is mapped to the address 0x081000000 (2M products swap 0x08000000 and 0x08100000, 4M products swap 0x08000000 and 0x08200000). Please note setting FMC_SWP bit in SYSCFG will swap the bank0 and bank1 logical addresses in the bus matrix but does not affect the original erase address.

The BB bit is provided in the option bytes. When the bit is set to 0 (factory value) and configure MCU boot from main flash, the MCU will boot from bank0. When the bit is set to 1 and configure MCU boot from main flash, if there is no code in bank1 the MCU will boot from bank0. If there is code in bank1 the MCU will boot from bank1. The internal implementation principle is as follows: when MCU boots from the main flash memory, the system bootloader will detect whether the BB bit is set or not. If the BB bit is set, the system bootloader will set the FMC_SWP bit and detect whether bank1 has code. If bank1 has code, MCU will boot from bank1 (the logical address still is 0x08000000).

The NWA bit is also provided in in the option bytes. When this bit is set to 0, the no waiting time area is selected as bank1. When this bit is is set to 1 (factory value), the no waiting time area is selected as bank0. This bit is only effective after power reset and only in the 4MB dual-bank product. By configuring the bit, the no waiting time area can be configured to improve the efficiency of code execution flexibly.

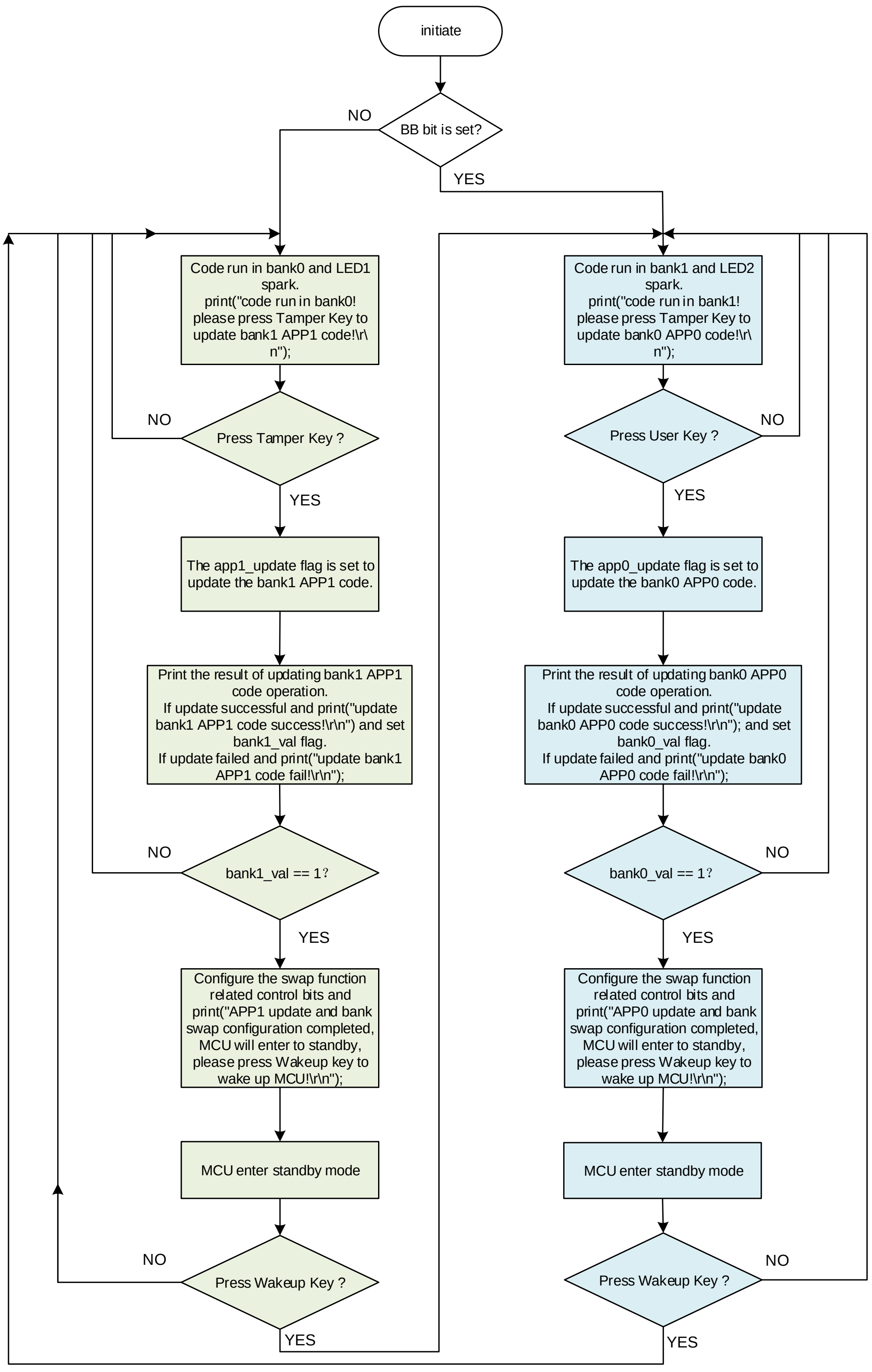
According to the above characteristics of GD32F5xx, GD32F5xx can realize the function of updating bank1 code when bank0 runs the code. After bank1 code is updated, the function of switching to bank1 code after power reset is realized by configuring the BB bit and NWA bit of the option byte. When bank1 runs the code, update the bank0 code. After the bank0 code is updated, the function of switching to bank0 code operation is realized by configuring the BB bit and NWA bit of the option byte after the power is reset.

3. Software implementation

3.1. Software process flow

In this software implementation scheme, two sets of codes for bank0 APP0 and bank1 APP1 are stored in an external Nand Flash in advance. After detecting the corresponding key press, the software programs the code in Nand Flash to bank0 or bank1. After the code programming is completed, the software configures the option byte BB bit and NWA bit, and makes the option byte BB bit and NWA bit effective by making the MCU enter standby mode and then wake up MCU. The specific implementation process is shown in ***Figure 3-1. Software flow diagram***.

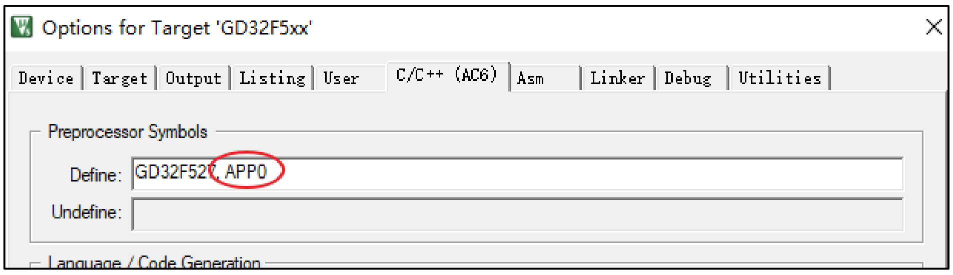
Figure 3-1. Software flow diagram


initiate
BB bit is set?
NO
YES
Code run in bank0 and LED1 spark.
print("code run in bank0! please press Tamper Key to update bank1 APP1 code!\r\n");
Press Tamper Key ?
NO
YES
The app1_update flag is set to update the bank1 APP1 code.
Print the result of updating bank1 APP1 code operation.
If update successful and print("update bank1 APP1 code success!\r\n") and set bank1_val flag.
If update failed and print("update bank1 APP1 code fail!\r\n");
bank1_val == 1?
NO
YES
Configure the swap function related control bits and print("APP1 update and bank swap configuration completed, MCU will enter to standby, please press Wakeup key to wake up MCU!\r\n");
MCU enter standby mode
Press Wakeup Key ?
NO
YES
Code run in bank1 and LED2 spark.
print("code run in bank1! please press Tamper Key to update bank0 APP0 code!\r\n");
Press User Key ?
NO
YES
The app0_update flag is set to update the bank0 APP0 code.
Print the result of updating bank0 APP0 code operation.
If update successful and print("update bank0 APP0 code success!\r\n"); and set bank0_val flag.
If update failed and print("update bank0 APP0 code fail!\r\n");
bank0_val == 1?
NO
YES
Configure the swap function related control bits and print("APP0 update and bank swap configuration completed, MCU will enter to standby, please press Wakeup key to wake up MCU!\r\n");
MCU enter standby mode
Press Wakeup Key ?
NO
YES

3.2. Test procedure and phenomena

1. Use macro APP0 and APP1 to compile the corresponding app0.bin and app1.bin. app0.bin and app1.bin correspond to the codes of bank0 and bank1 respectively. Please note when compiling the project, the Linker option uses the specified Project.sct file which is provided under the Template\Keil_project path, as shown in ***Figure 3-2. Preprocessor symbols define of Keil project***. In the User option of Keil project, configure the bin file output command after the project compilation is completed, as shown in ***Figure 3-3. Linker option of Keil project***. Please choose the local installation path of Keil.

Figure 3-2. Preprocessor symbols define of Keil project

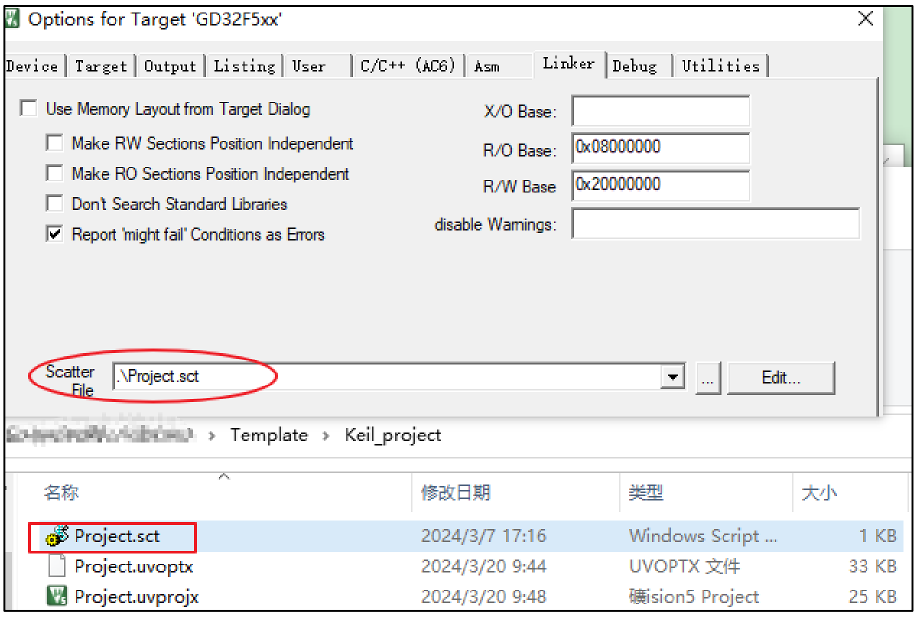

Options for Target 'GD32F5xx'

Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities

Preprocessor Symbols
Define: GD32F527, APP0
Undefine:

Language / Code Generation

Figure 3-3. Linker option of Keil project


Options for Target 'GD32F5xx'

Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities

☐ Use Memory Layout from Target Dialog
☐ Make RW Sections Position Independent
☐ Make RO Sections Position Independent
☐ Don't Search Standard Libraries
☑ Report 'might fail' Conditions as Errors

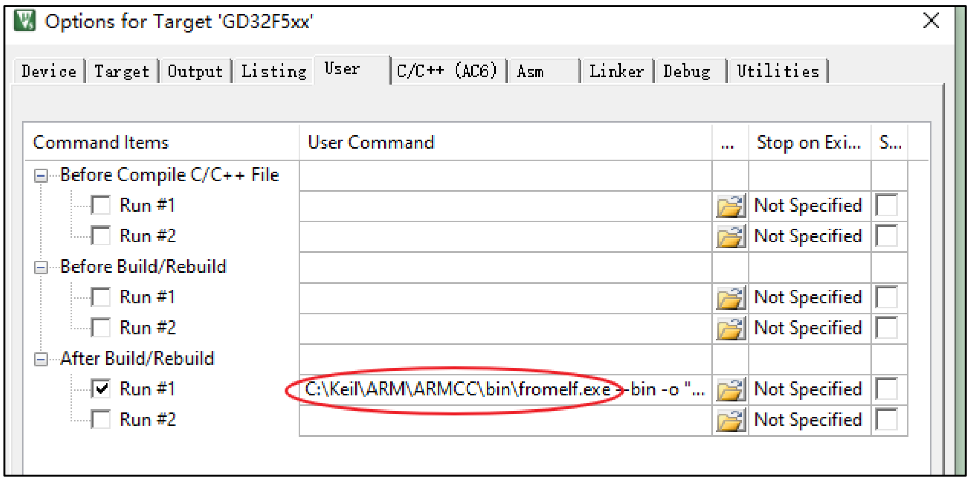
X/O Base:
R/O Base: 0x08000000
R/W Base 0x20000000
disable Warnings:

Scatter File: .\Project.sct

Template › Keil_project

名称	修改日期	类型	大小
Project.sct	2024/3/7 17:16	Windows Script ...	1 KB
Project.uvoptx	2024/3/20 9:44	UVOPTX 文件	33 KB
Project.uvprojx	2024/3/20 9:48	礦ision5 Project	25 KB

Figure 3-4. User option of Keil project

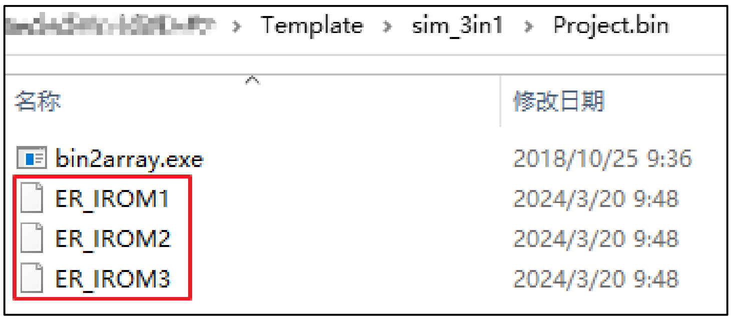

Options for Target 'GD32F5xx'

Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities

Command Items	User Command	...	Stop on Exi...	S...
Before Compile C/C++ File				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
Before Build/Rebuild				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
After Build/Rebuild				
☑ Run #1	C:\Keil\ARM\ARMCC\bin\fromelf.exe --bin -o "...		Not Specified	☐
☐ Run #2			Not Specified	☐

2. After compiling the Project, the project will generate three section bins in the Template\sim_3in1\ project.bin folder, which are ER_IROM1, ER_IROM2, and ER_IROM3.

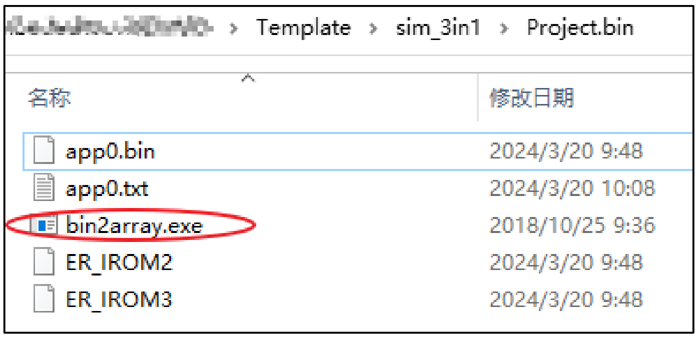
Figure 3-5. Project.bin folder after compilation


[redacted] › Template › sim_3in1 › Project.bin

名称	修改日期
bin2array.exe	2018/10/25 9:36
ER_IROM1	2024/3/20 9:48
ER_IROM2	2024/3/20 9:48
ER_IROM3	2024/3/20 9:48

When the preprocessor macro used is APP0, after compiling the project, change ER_IROM1's name to app0.bin, and use bin2array.exe software to convert app0.bin to app0.txt file, and then put the array in app0.txt into app0_code[] array in app0.h file, as shown in ***Table 3-1. Definition of app0.bin***. The generates app1.bin by same method and puts it into the app1_code[] array of app1.h file, as shown in ***Table 3-2. Definition of app1.bin***.

Figure 3-6. Output app0.bin file


[redacted] › Template › sim_3in1 › Project.bin

名称	修改日期
app0.bin	2024/3/20 9:48
app0.txt	2024/3/20 10:08
bin2array.exe	2018/10/25 9:36
ER_IROM2	2024/3/20 9:48
ER_IROM3	2024/3/20 9:48

Table 3-1. Definition of app0.bin

```
const uint8_t app0_code[]__attribute__((used))__attribute__((section ("APP0_Array")))=
{0xB0,0x0C,0x00,0x20,0xF5,0x01,0x00,0x08,0xC5,0x07,0x00,0x08,0xBD,0x07,0x00,0x08,
…};
```

Table 3-2. Definition of app1.bin

```
const uint8_t app1_code[]__attribute__((used))__attribute__((section ("APP1_Array")))=
{0xB0,0x0C,0x00,0x20,0xF5,0x01,0x00,0x08,0xC5,0x07,0x00,0x08,0xBD,0x07,0x00,0x08,
…};
```

3. Load the app0.bin and app1.bin files into Nand Flash using scatter-loading method. In this way, the main flash and Nand flash can be burned simultaneously when the project code is downloaded.

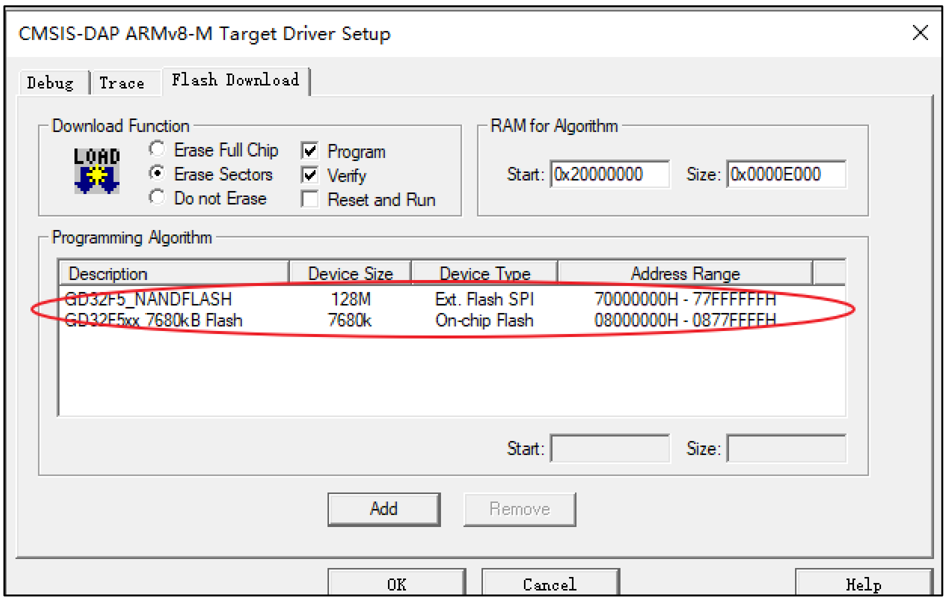
Table 3-3. Scatter-loading file

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x08000000 0x00780000  {     ; load region size_region
  ER_IROM1 0x08000000 0x00780000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
   .ANY (+XO)
  }
  RW_IRAM1 0x20000000 0x00080000  {  ; RW data
   .ANY (+RW +ZI)
  }
}

LR_IROM2 0x70000000 0x2000000  {
  ER_IROM2 0x70000000 0x2000000  {  ; load address = execution address
   *(APP0_Array)
  }
}

LR_IROM3 0x72000000 0x6000000  {
  ER_IROM3 0x72000000 0x6000000  {  ; load address = execution address
   *(APP1_Array)
  }
}
```

4. After completing the above operations, select the project preprocessor macro definition as APP0, compile project and download the code to main flash and Nand flash. User

need to make the GD32F5_NANDFLASH download algorithm in advance and place the download algorithm in the Keil installation path, in this case the path is C:\Keil_v535\ARM\Flash. Add GD32F5xx Flash download algorithm and GD32F5_NANDFLASH download algorithm in Keil "Options->Debug->Settings->Flash Download" option, as shown in ***Figure 3-7. Download algorithm configuration of Keil project***. Please note that before downloading the code, please keep the option bytes BB and NWA bits as factory default values, BB bits as 0 and NWA bits as 1. If user encounter an error while downloading the code, please erase the whole piece before downloading.

Figure 3-7. Download algorithm configuration of Keil project


CMSIS-DAP ARMv8-M Target Driver Setup

Debug | Trace | Flash Download

Download Function: Erase Full Chip [✓ Erase Sectors] Do not Erase; Program ✓; Verify ✓; Reset and Run ☐

RAM for Algorithm: Start: 0x20000000 Size: 0x0000E000

Programming Algorithm

Description	Device Size	Device Type	Address Range
GD32F5_NANDFLASH	128M	Ext. Flash SPI	70000000H - 77FFFFFFH
GD32F5xx 7680kB Flash	7680k	On-chip Flash	08000000H - 0877FFFFH

Start: Size:

5. After downloading the code, reset the MCU and the code starts to run. Serial output operation information and operation tips.

   When running bank0 code, LED1 blinks and the serial port outputs "code run in bank0! please press Tamper Key to update bank1 APP1 code!” .

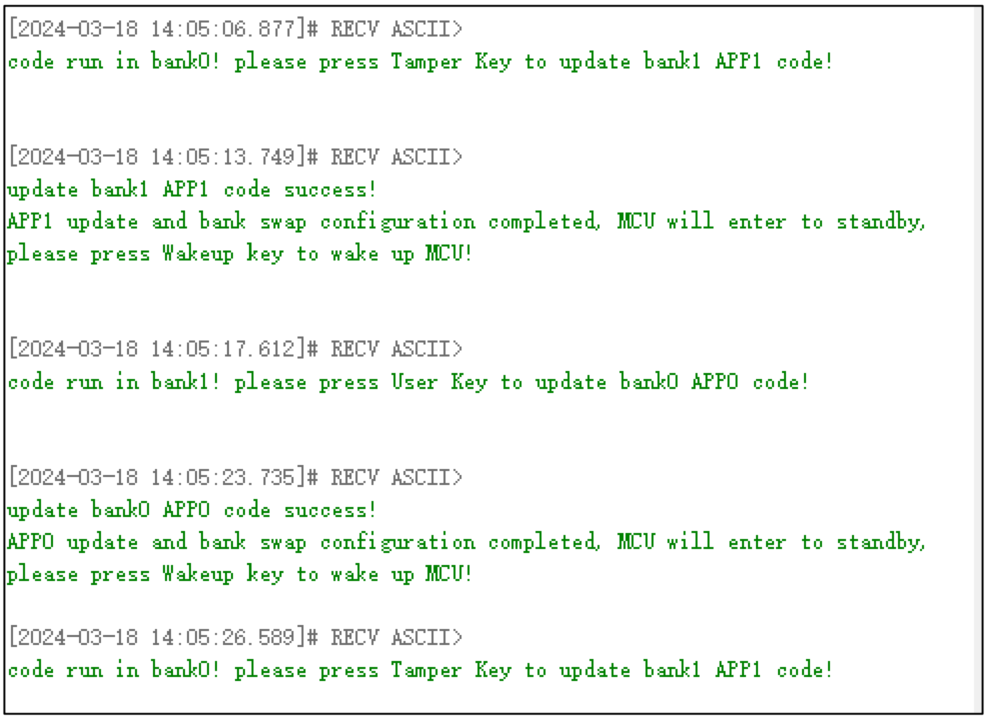
   When user presses Tamper key, the software updates the bank1 APP1 code. After the update is completed, the serial port outputs "update bank1 APP1 code success! APP1 update and bank swap configuration completed, MCU will enter to standby, please press Wakeup key to wake up MCU!”. MCU completes bank switching and enters standby mode.

   At this time, if the user presses the Wakeup key, the MCU switches to bank1 code running and the serial port output "code run in bank1! please press User Key to update bank0 APP0 code!”. LED2 blinks.

   When user presses the User key and the software updates the bank0 APP0 code. After the update is completed, the serial port outputs "update bank0 APP0 code success! APP0 update and bank swap configuration completed, MCU will enter to standby, please press Wakeup key to wake up MCU! ”. MCU completes bank switching and enters standby mode.

At this time, if the user presses the Wakeup key, the MCU switches to the bank0 code to run, and the serial port outputs "code run in bank0! please press Tamper Key to update bank1 APP1 code!". LED1 blinks.

Figure 3-8. Test results



```
[2024-03-18 14:05:06.877]# RECV ASCII>
code run in bank0! please press Tamper Key to update bank1 APP1 code!


[2024-03-18 14:05:13.749]# RECV ASCII>
update bank1 APP1 code success!
APP1 update and bank swap configuration completed, MCU will enter to standby,
please press Wakeup key to wake up MCU!


[2024-03-18 14:05:17.612]# RECV ASCII>
code run in bank1! please press User Key to update bank0 APP0 code!


[2024-03-18 14:05:23.735]# RECV ASCII>
update bank0 APP0 code success!
APP0 update and bank swap configuration completed, MCU will enter to standby,
please press Wakeup key to wake up MCU!

[2024-03-18 14:05:26.589]# RECV ASCII>
code run in bank0! please press Tamper Key to update bank1 APP1 code!
```

4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Apr.23 2024

Important Notice