GigaDevice Semiconductor Inc.

Introduction to GD32G5x3 Flash read-while-write (RWW) feature

Application Notes

AN249

Revision 1.1

(Sep. 2025)

Table of Contents

List of Figure

List of Table

1. Foreword

The dual-bank flash of GD32G5x3 features RWW capabilities, allowing the CPU to execute code in flash while erasing and programming another area of flash. This means that while one partition is being erased or written, the other can still be read. This capability is crucial for implementing imperceptible software updates, such as OTA updates. Thanks to the RWW feature, IAP and applications can be combined for operation without the need for separate handling. This dual-bank mode greatly simplifies the implementation of IAP, enhances the speed of IAP updates, and allows developers more flexibility in program upgrades and management.

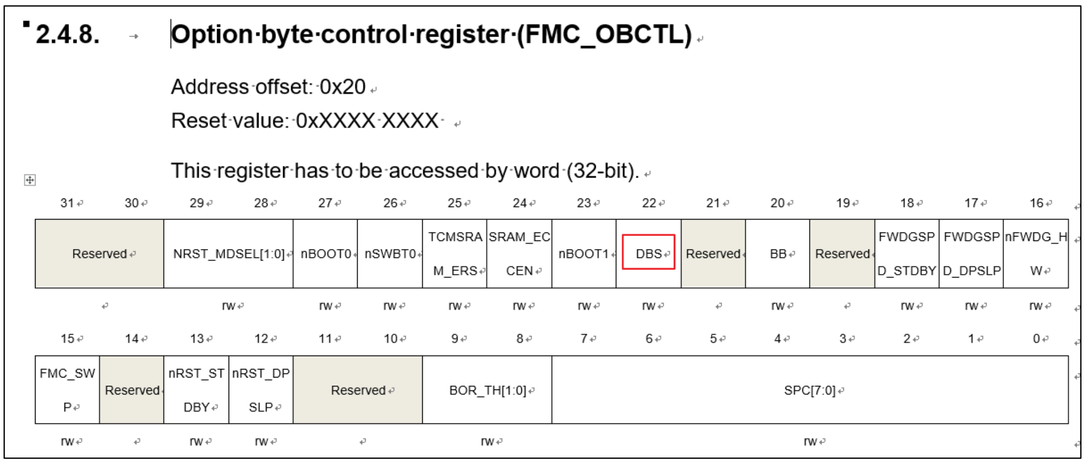
2. GD32G5x3 flash configuration

The main storage flash memory of GD32G5x3 can reach up to 512KB, composed of 2x256 pages in a dual-bank structure, with each page being 1KB, and also includes 2x13KB of information blocks for boot loader. Each page of the main storage flash memory can be erased individually; in a single-bank structure, it is composed of 256 pages, with each page being 2KB. In the following context, the flash memory size is taken as 512KB.

2.1. Single and double bank switching configuration

The method to switch from single-bank to dual-bank is to set the DBS bit in the Option Byte Control Register (FMC_OBCTL) to 1, at which point the read width is 64 bits. As shown in ***Figure 2-1. Single and double bank switching register bit***.

Figure 2-1. Single and double bank switching register bit



2.4.8. Option byte control register (FMC_OBCTL)

Address offset: 0x20

Reset value: 0xXXXX XXXX

This register has to be accessed by word (32-bit).

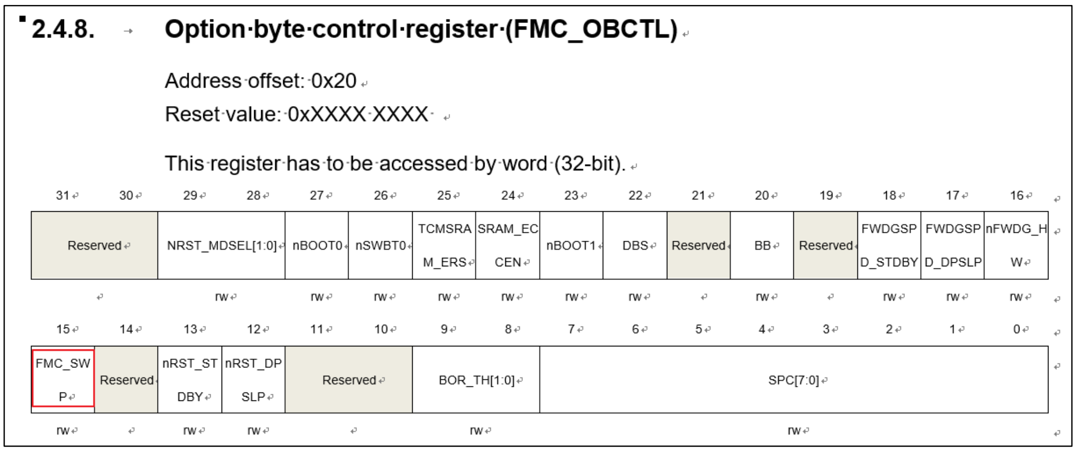
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		NRST_MDSEL[1:0]		nBOOT0	nSWBT0	TCMSRAM_ERS	SRAM_ECCEN	nBOOT1	DBS	Reserved	BB	Reserved	FWDGSPD_STDBY	FWDGSPD_DPSLP	nFWDG_HW
		rw		rw	rw	rw	rw	rw	rw		rw		rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FMC_SWP	Reserved	nRST_STDBY	nRST_DPSLP	Reserved		BOR_TH[1:0]		SPC[7:0]							
rw		rw	rw			rw		rw							

2.2. Dual-bank mapping switch

The bank0 and bank1 of main memory can be mapped to address 0x08000000 and 0x08040000 respectively, as shown in ***Figure 2-2. Bank switch register bit***.

Figure 2-2. Bank switch register bit



2.4.8. Option byte control register (FMC_OBCTL)

Address offset: 0x20
Reset value: 0xXXXX XXXX

This register has to be accessed by word (32-bit).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		NRST_MDSEL[1:0]		nBOOT0	nSWBT0	TCMSRAM_ERS	SRAM_ECCEN	nBOOT1	DBS	Reserved	BB	Reserved	FWDGSPD_STDBY	FWDGSPD_DPSLP	nFWDG_HW
		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
FMC_SWP	Reserved	nRST_STDBY	nRST_DPSLP	Reserved		BOR_TH[1:0]		SPC[7:0]							
rw		rw	rw			rw		rw							

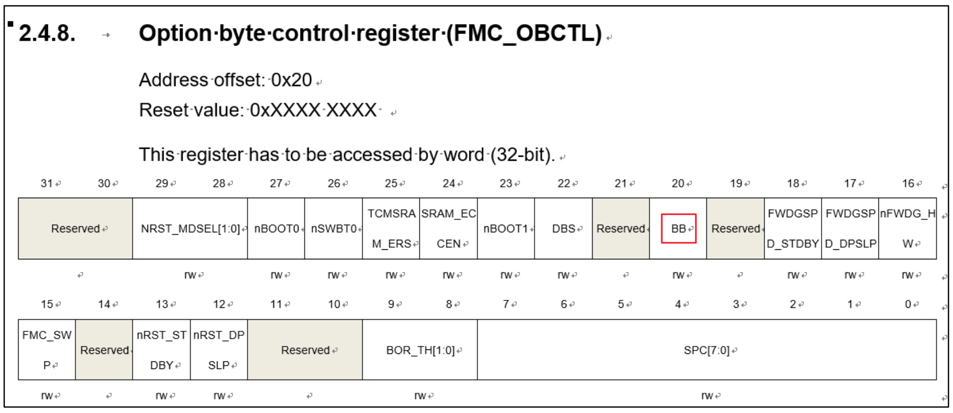
Note: After modifying the option bytes through option byte programming operations, the changes will only take effect after a system reset or by setting the OBRLD bit in the FMC_CTL register to 1. If you need the changes to take effect immediately without saving to the option byte register, you can switch the memory mapping in real-time by operating the FMC_SWP bit in the SYSCFG_CFG0 register.

2.3. Bank boot configuration

The GD32G5x3 series of MCUs supports dual-bank mode for the main memory, and when configured in dual-bank mode, users can configure registers to determine which bank the application starts from.

As shown in ***Figure 2-3. Bank boot selection register bit***, by configuring nSWBT0, nBOOT0, nBOOT1, and the BB bit, it is possible to boot from a specific bank. If the boot mode is to boot from the main flash memory, the system will boot from bank0 when the BB bit is 0, and from bank1 when the BB bit is 1. While, if bank1 is empty, the system will fall back to boot from bank0. If the boot mode needs to be configured to boot from the system memory, nBOOT1 can be set to 1 and set nBOOT0 value in conjunction with nSWBT0, and the BB bit is reset to 0, so as to boot from the system memory and execute the bootloader.

Figure 2-3. Bank boot selection register bit



2.4.8. Option byte control register (FMC_OBCTL)

Address offset: 0x20
Reset value: 0xXXXX XXXX

This register has to be accessed by word (32-bit).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		NRST_MDSEL[1:0]		nBOOT0	nSWBT0	TCMSRAM_ERS	SRAM_ECCEN	nBOOT1	DBS	Reserved	BB	Reserved	FWDGSPD_STDBY	FWDGSPD_DPSLP	nFWDG_HW
		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
FMC_SWP	Reserved	nRST_STDBY	nRST_DPSLP	Reserved		BOR_TH[1:0]		SPC[7:0]							
rw		rw	rw			rw		rw							

3. IAP upgrade

3.1. IAP introduction

IAP (In-Application Programming) upgrade is a technology that allows firmware to be upgraded during the application's operation. Its advantage is that it allows the firmware to be updated through the application code without stopping the application. For systems that need to run continuously, IAP can reduce downtime caused by firmware updates. Of course, it also has certain defects. One is that implementing IAP may require a complex software architecture and additional storage space. The other is that if the firmware verification is not sufficient, it may introduce security risks, so if the new firmware has defects, there must be a reliable recovery mechanism to switch back to the old firmware.

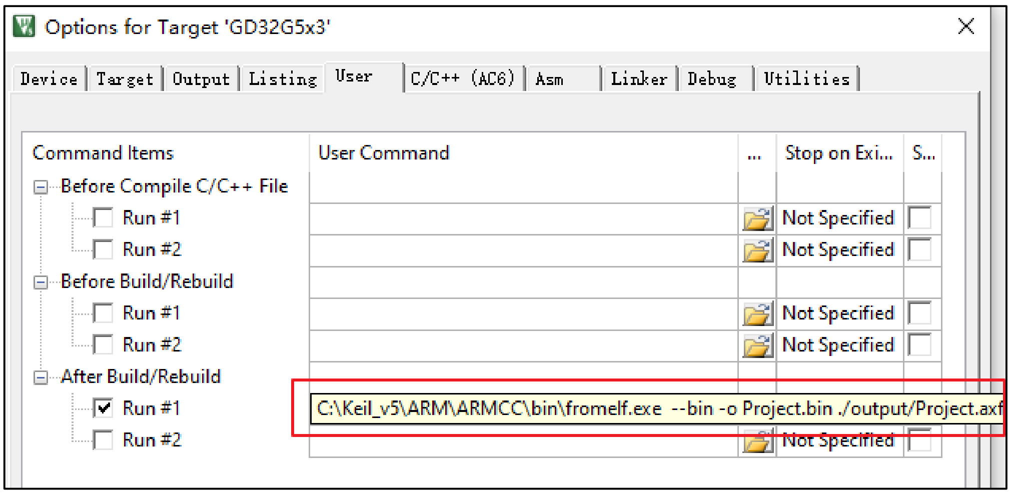
The RWW feature of GD32G5x3 allows for updating the code in bank1 while the code in bank0 is running, followed by a bank switch to implement firmware updates. When the FMC_SWP bit is set, the code in bank0 can be updated while the code in bank1 is running, followed by a bank switch to implement firmware updates.

3.2. IAP upgrade steps

Based on the above characteristics, assuming the current BB position is 0, the simplified IAP upgrade process for GD32G5x3 dual-bank is as follows.

Add the instruction to generate bin file during compilation in the Keil for the upgrade project to be written.

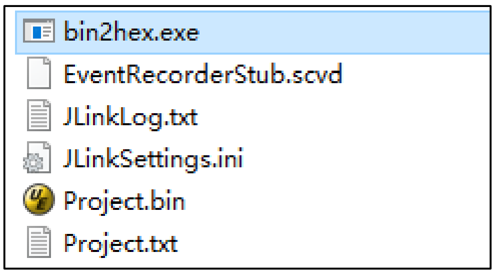
Figure 3-1. Instruction for generating bin file



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_v5\ARM\ARMCC\bin\fromelf.exe --bin -o Project.bin ./output/Project.axf			
☐ Run #2		📂	Not Specified	☐

After compilation, use the bin2hex software to convert the generated bin file into a txt file.

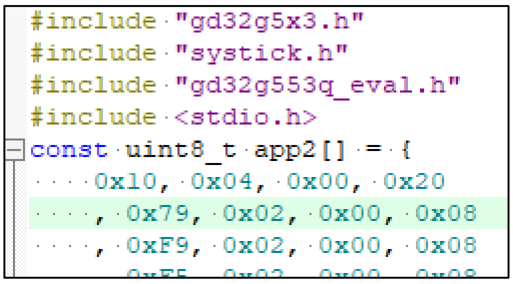
Figure 3-2. generate a txt file



bin2hex.exe
EventRecorderStub.scvd
JLinkLog.txt
JLinkSettings.ini
Project.bin
Project.txt

Open the txt file and write the data from the txt into a const array.

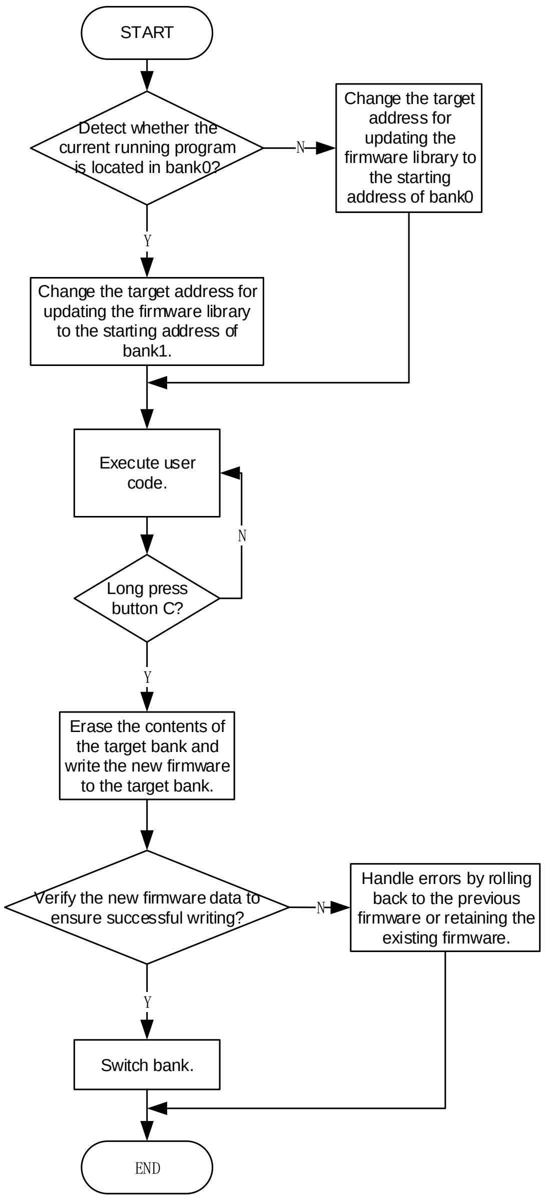
Figure 3-3. Store data



```
#include "gd32g5x3.h"
#include "systick.h"
#include "gd32g553q_eval.h"
#include <stdio.h>
const uint8_t app2[] = {
    0x10, 0x04, 0x00, 0x20
    , 0x79, 0x02, 0x00, 0x08
    , 0xF9, 0x02, 0x00, 0x08
```

Note: These steps are simplified, and in engineering practice, one can write code to transmit the code to be upgraded via methods such as UART.

Figure 3-4. Simple IAP upgrade plan for dual-bank


START
Detect whether the current running program is located in bank0?
N
Change the target address for updating the firmware library to the starting address of bank0
Y
Change the target address for updating the firmware library to the starting address of bank1.
Execute user code.
N
Long press button C?
Y
Erase the contents of the target bank and write the new firmware to the target bank.
Verify the new firmware data to ensure successful writing?
N
Handle errors by rolling back to the previous firmware or retaining the existing firmware.
Y
Switch bank.
END

3.3. IAP upgrade code

The IAP upgrade simple code is shown as in ***Table 3-1. IAP Upgrade Code***. This code is only used to demonstrate the simple IAP upgrade using the RWW feature, which is quite basic and can be used as an idea for code upgrading.

Table 3-1. IAP Upgrade Code

```
#include <stdint.h>
#include <stdbool.h>
#include "gd32g5x3.h"
/* Added by the user */
const uint8_t app2[] = { ……}
```

```
#define destinaion_add0 0x08000000
#define destinaion_add1 0x08040000
uint32_t destinaion_add = 0;
uint32_t NUM = sizeof(app2);
void switch_to_new_firmware(uint64_t add)
{
    for(uint16_t i = 0; i < NUM; i += 8) {
        if(*(uint64_t *)(add + i) != (uint64_t)((uint64_t)app2[i] | ((uint64_t)app2[i + 1] << 8) |
((uint64_t)app2[i + 2] << 16) | ((
                uint64_t)app2[i + 3] << 24) | ((uint64_t)app2[i + 4] << 32) | ((uint64_t)app2[i + 5] <<
40) | ((uint64_t)app2[i + 6] << 48) | ((
                    uint64_t)app2[i + 7] << 56))) {
            return;
        }
    }
    /* The implementation needs to be modified according to the actual situation */
    __disable_irq();

    /* disable pre-fetch */
    FMC_WS &= ~FMC_WS_PFEN;
    /* disable IBUS cache */
    FMC_WS &= ~FMC_WS_ICEN;
    /* disable DBUS cache */
    FMC_WS &= ~FMC_WS_DCEN;

    /* reset IBUS cache   */
    FMC_WS |= FMC_WS_ICRST;
    /* reset DBUS cache */
    FMC_WS |= FMC_WS_DCRST;
    /* bank mapping switch */
    ob_bank_memory_swap_config(OB_BANK_MAPPING_SWAP);

    /* enable pre-fetch */
    FMC_WS |= FMC_WS_PFEN;
    /* enable IBUS cache */
    FMC_WS |= FMC_WS_ICEN;
    /* enable DBUS cache */
    FMC_WS |= FMC_WS_DCEN;

    /* Recovery interrupt */
    __enable_irq();
    /* Pay attention to ensure the safe state during the restart or switchover proces */
}
```

```
void confirm_to_switch(void)
{

    if(RESET == gpio_input_bit_get(KEY_D_GPIO_PORT, KEY_D_PIN)) {
        delay_1ms(1000);
        /* check whether the key is pressed */
        if(RESET == gpio_input_bit_get(KEY_D_GPIO_PORT, KEY_D_PIN)) {
            fmc_unlock();
            ob_unlock();
            switch_to_new_firmware(destinaion_add);
            ob_reload();
        }
    }
}

/*!
    \brief      main function
    \param[in]  none
    \param[out] none
    \retval     none
*/
int main(void)
{
    /* configure systick */
    systick_config();

    if(FMC_OBCTL & FMC_OBCTL_FMC_SWP) {
        destinaion_add = destinaion_add1;
    } else {
        destinaion_add = destinaion_add0;
    }
    /* enable the LED2 GPIO clock */
    rcu_periph_clock_enable(RCU_GPIOE);
    /* configure LED2 GPIO pin */
    gpio_mode_set(GPIOE, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, GPIO_PIN_4);
    gpio_output_options_set(GPIOE, GPIO_OTYPE_PP, GPIO_OSPEED_60MHZ, GPIO_PIN_4);
    /* reset LED2 GPIO pin */
    gpio_bit_reset(GPIOE, GPIO_PIN_4);

    /* enable the key GPIO clock */
    rcu_periph_clock_enable(RCU_GPIOA);
    rcu_periph_clock_enable(RCU_GPIOC);
    rcu_periph_clock_enable(RCU_GPIOF);
    /* configure key pin as input */
    gpio_mode_set(KEY_B_GPIO_PORT,       GPIO_MODE_INPUT,       GPIO_PUPD_NONE,
KEY_B_PIN);
    gpio_mode_set(KEY_C_GPIO_PORT,       GPIO_MODE_INPUT,       GPIO_PUPD_NONE,
KEY_C_PIN);
    gpio_mode_set(KEY_D_GPIO_PORT,       GPIO_MODE_INPUT,       GPIO_PUPD_NONE,
KEY_D_PIN);
    while(1) {
        /* user code */
        /* check whether the key is pressed */
        if(RESET == gpio_input_bit_get(KEY_B_GPIO_PORT, KEY_B_PIN)) {
            delay_1ms(100);

            /* check whether the key is pressed */
            if(RESET == gpio_input_bit_get(KEY_B_GPIO_PORT, KEY_B_PIN)) {
                /* toggle LED2 GPIO pin */
                gpio_bit_toggle(GPIOE, GPIO_PIN_4);
            }
        }
        /*   upgrade code */
        if(RESET == gpio_input_bit_get(KEY_C_GPIO_PORT, KEY_C_PIN)) {
            delay_1ms(1000);
            /* check whether the key is pressed */
            if(RESET == gpio_input_bit_get(KEY_C_GPIO_PORT, KEY_C_PIN)) {
                fmc_unlock();
                fmc_bank1_erase();
                /* Write app2 code to another bank, where the upgrade code can be imported by
other means such as UART */
                for(uint16_t i = 0; i < NUM; i += 8) {
                    fmc_doubleword_program(destinaion_add + i, (uint64_t)((uint64_t)app2[i] |
((uint64_t)app2[i + 1] << 8) | ((uint64_t)app2[i + 2] << 16) | ((
                                                   uint64_t)app2[i + 3] << 24) | ((uint64_t)app2[i +
4] << 32) | ((uint64_t)app2[i + 5] << 40) | ((uint64_t)app2[i + 6] << 48) | ((
                                                             uint64_t)app2[i + 7] << 56)));
                }
            }
        }
        confirm_to_switch();
    }
}
```

4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial release	Dec.30, 2024
1.1	Modify bank boot configuration	Sep.17, 2025

Important Notice