GigaDevice Semiconductor Inc.

GD32G5x3_Flash read-while-write (RWW)功能介绍

应用笔记

AN249

1.0 版本

（2025 年 1 月）

目录

目录 2
图索引 3
表索引 4
1. 简介 5
2. GD32G5x3 flash 配置 6
2.1. 单双 bank 切换配置 6
2.2. 双 bank 映射切换 6
2.3. bank 启动配置 7
3. IAP 升级 8
3.1. IAP 介绍 8
3.2. IAP 升级步骤 8
3.3. IAP 升级代码 11
4. 版本历史 15

图索引

图 2-1. 单双 bank 切换寄存器位 6
图 2-2. bank 切换寄存器位 6
图 2-3. bank 启动选择寄存器位 7
图 3-1. 生成 bin 文件指令 8
图 3-2. 生成 txt 文件 9
图 3-3. 存放数据 9
图 3-4. 双 bank 的 IAP 升级简易方案 10

表索引

表 3-1. IAP 升级代码 11
表 4-1. 版本历史 15

1. 简介

GD32G5x3 的双 bank Flash 具有 RWW 特性，它允许 CPU 在 Flash 中执行代码的同时，对另一个区域的 Flash 进行擦除和编程操作。这意味着，当一个分区正在被擦除或写入时，另一个分区仍然可以被读取。这种能力对于实现无感知的软件更新（如 OTA 更新）至关重要，由于支持 RWW 特性，IAP 和应用程序可以合并到一起进行操作，不需要分开处理。这种双 Bank 模式大大简化了 IAP 的实现过程，提升了 IAP 更新速度，使得开发者可以更灵活地进行程序升级和管理。

2. GD32G5x3 flash 配置

GD32G5x3 主存储闪存高达 512KB，双 bank 结构下由 2x256 页组成，每页 1KB，还包含 2x13KB 的用于引导装载程序的信息块。主存储闪存的每页都可以单独擦除；单 bank 结构下由 256 页组成，每页 2KB。在下文中，均以闪存大小为 512KB 为例。

2.1. 单双 bank 切换配置

将单 bank 切换为双 bank 的方法是将选项字节控制寄存器(FMC_OBCTL)中的 DBS 位置为 1，此时读取位宽为 64 位。如*图 2-1. 单双 bank 切换寄存器位*所示。

图 2-1. 单双 bank 切换寄存器位

2.4.8. 选项字节控制寄存器(FMC_OBCTL)

地址偏移：0x20

复位值：0xXXXX XXXX

该寄存器只能按字(32 位)访问。

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

2.2. 双 bank 映射切换

主存储器的 bank0 和 bank1 可以分别被映射到地址 0x0800 0000 和地址 0x0804 0000。如*图 2-2. bank 切换寄存器位*所示。

图 2-2. bank 切换寄存器位

2.4.8. 选项字节控制寄存器(FMC_OBCTL)

地址偏移：0x20

复位值：0xXXXX XXXX

该寄存器只能按字(32 位)访问。

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

注意：通过选项字节编程操作修改选项字节后，需要通过系统复位或者将 FMC_CTL 寄存器中 OBRLD 位置 1 后，方可生效。如需立即生效且不保存至选项字节寄存器，可以通过操作 SYSCFG_CFG0 寄存器的 FMC_SWP 位，来实时切换存储器映射。

2.3. bank 启动配置

GD32G5x3 系列 MCU 主存储器支持双 bank 模式，当配置为双 bank 模式时，用户可以配置寄存器来决定应用程序从任意一个 bank 启动。

当 BB 位为 0 时，若当前为从主闪存区启动，则从 bank0 启动；当 BB 位为 1 时，若当前为从主闪存区启动，则从 bank1 启动，若 bank1 为空则从 bank0 启动。如*图 2-3. bank 启动选择寄存器位*所示。

图 2-3. bank 启动选择寄存器位

2.4.8. 选项字节控制寄存器(FMC_OBCTL)

地址偏移：0x20

复位值：0xXXXX XXXX

该寄存器只能按字(32 位)访问。

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

3. IAP 升级

3.1. IAP 介绍

IAP（In-Application Programming）升级是一种在应用程序运行期间对固件进行升级的技术。其优点在于它允许在不停止应用程序的情况下，通过应用程序代码来更新固件的功能。对于需要持续运行的系统来说，IAP 可以减少因固件更新导致的停机时间。当然，它也存在一定缺陷。其一是实现 IAP 可能需要复杂的软件架构和额外的存储空间。其二是如果固件验证不充分，可能会引入安全风险，因此如果新固件有缺陷，需要有一个可靠的恢复机制来切换回旧固件。

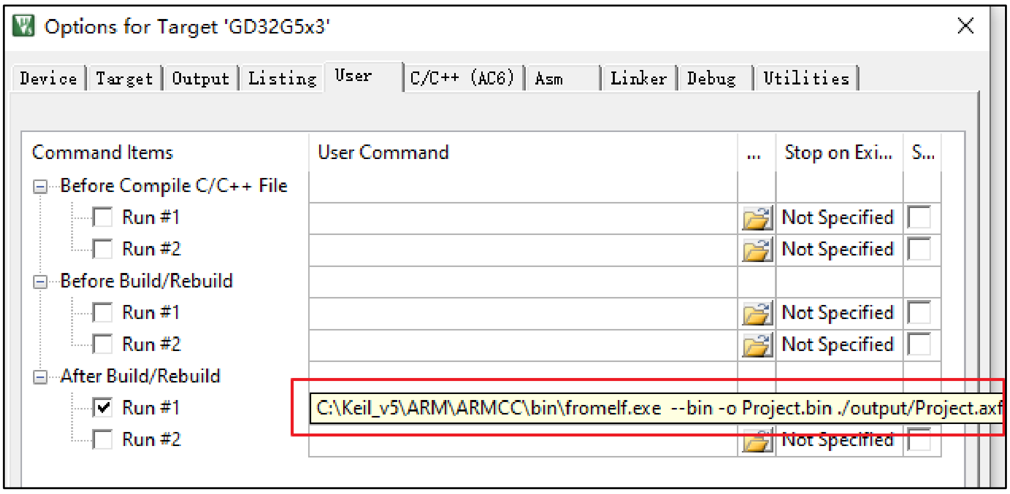
GD32G5x3 具备的 RWW 特性，可以在 bank0 代码运行时，更新 bank1 代码，然后执行 bank 切换，实现固件更新。当 FMC_SWP 位已置位，bank1 代码运行时，更新 bank0 代码，然后执行 bank 切换，实现固件更新。

3.2. IAP 升级步骤

基于以上特性，假定当前 BB 位为 0，GD32G5x3 双 bank 的 IAP 升级简易方案如以下流程所示。

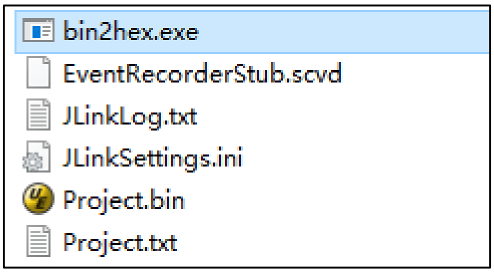
如图所示，在要写入的升级工程的 keil 中添加编译生成 bin 文件指令。

图 3-1. 生成 bin 文件指令



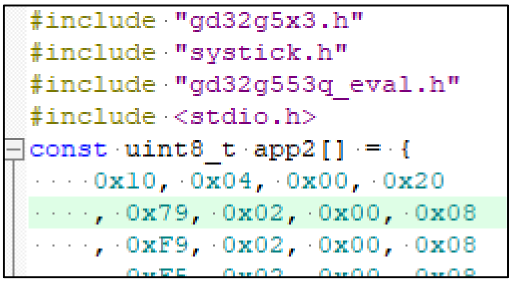
编译后，将生成的 bin 文件通过 bin2hex 软件生成 txt 文件。

图 3-2. 生成 txt 文件



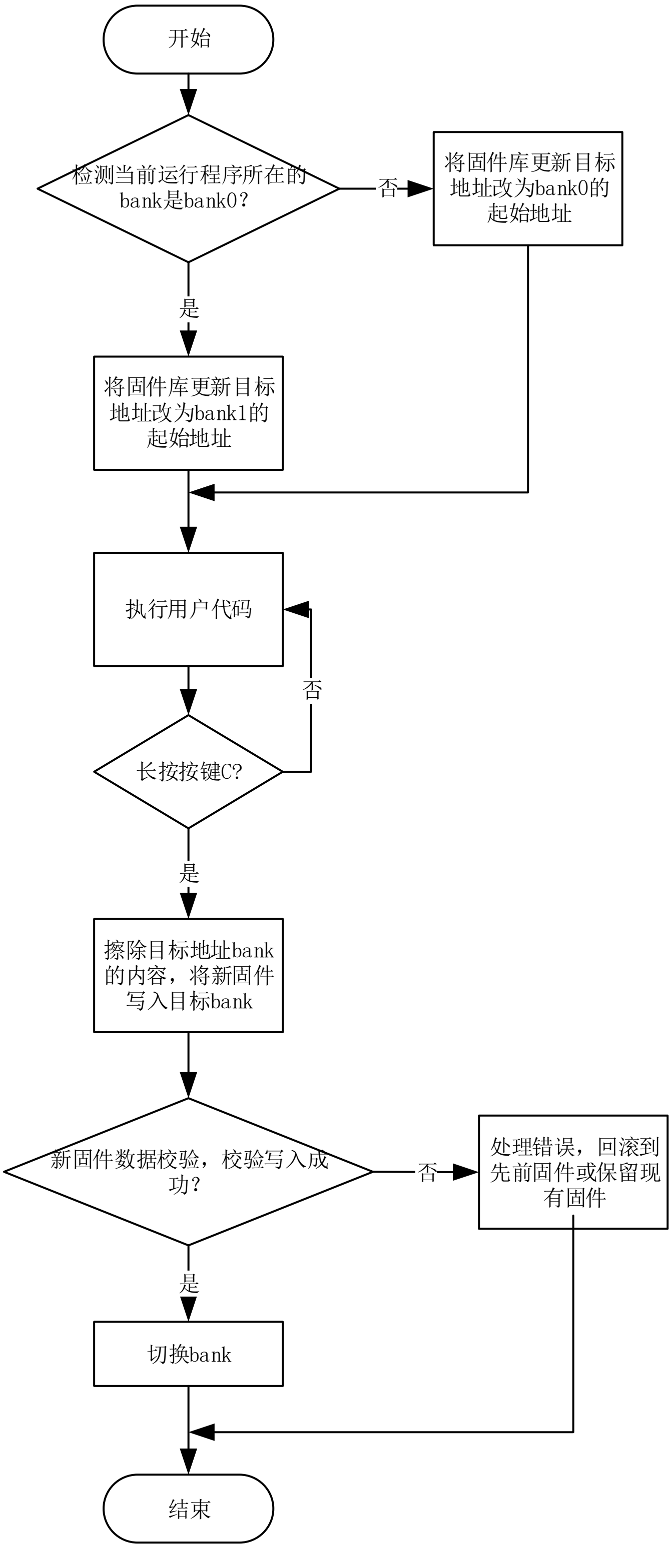
打开 txt 文件，将 txt 中的数据写入 const 数组中。

图 3-3. 存放数据



注意：这里简化了步骤，实际上可编写代码通过 UART 等方法传输要升级的代码。

图 3-4. 双 bank 的 IAP 升级简易方案



3.3. IAP 升级代码

IAP 升级简易代码如*表 3-1. IAP 升级代码*所示。此处代码仅用以展示利用 RWW 特性进行简易 IAP 升级，比较基础，可作为代码升级的一个思路。

表 3-1. IAP 升级代码

```
#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. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	首次发布	2025 年 1 月 14 日

Important Notice