GigaDevice Semiconductor Inc.

FLASH emulate EEPROM for GD32E23x series

Application Note
AN196

Revision 1.0

(Mar. 2024)

Table of Contents

List of Figures

List of Tables

1. Introduction

Both FLASH and EEPROM are non-volatile storage devices that can retain data after a power reset. The distinguish between FLASH and EEPROM is that the erase mode is different. EEPROM can be erased by bytes, but the minimum erase unit of FLASH is page. A page usually contains several bytes or even several K bytes. The erase attributes of FLASH and EEPROM determine that EEPROM has a small capacity but a high erasure life, while FLASH has a very large capacity but a short erasure life.

As the MCU frequency is high, FLASH can be used to simulate EEPROM to reduce the cost. In this paper, a method of simulating EEPROM with FLASH is introduced, which realizes the EEPROM data modification by byte, and can prevent the data lost by software reset or power reset. The larger the FLASH storage space, the better the performance of EEPROM.

2. Structure of EEPROM backup

In this paper, 63 pages of FLASH simulate 2K bytes EEPROM are used to introduce the method of FLASH simulation EEPROM.

2.1. GD32E23x FLASH introduction

The GD32E23x consists up to 128KB on-chip FLASH. ***Table 2-1. Base address and size for 128 KB flash memory*** shows the base address and size.

Table 2-1. Base address and size for 128 KB flash memory

Block	Name	Address	size(bytes)
Main Flash Block	Page 0	0x0800 0000 - 0x0800 03FF	1KB
	Page 1	0x0800 0400 - 0x0800 07FF	1KB
	Page 2	0x0800 0800 - 0x0800 0BFF	1KB
	. . .	. . .	. . .
	Page 127	0x0801 FC00 - 0x0801 FFFF	1KB
Information Block	Boot Loader	0x1FFF EC00 - 0x1FFF F7FF	3KB
Option byte Block	Option byte	0x1FFF F800 - 0x1FFF F80F	16B
One-time program Block	OTP bytes	0x1FFF_7000~0x1FFF_73FF	1KB

2.2. Data structure of EEPROM data

Table 2-2. Data structure of flash page

Function		size(bytes)
EEPROM page 0	FLASH page use flag	8
	EEPROM page start flag	8
	EEPROM page end flag	8
	EEPROM data	2048
EEPROM page 1	FLASH page use flag	8
	EEPROM page start flag	8
	EEPROM page end flag	8
	EEPROM data	2048
…	…	…
EEPROM page n	FLASH page use flag	8
	EEPROM page start flag	8
	EEPROM page end flag	8
	EEPROM data	2048

3. FLASH emulate EEPROM solution

3.1. Algorithm implementation

3.1.1. Parameter macro

Table 3-1. Parameter macro

Name	function
EEPROM_DATA_SIZE	Size of emulated EEPROM
EEPROM_FLASH_PAGE_NUM	FLASH pages
FLASH_PAGE_SIZE	FLASH page size
EEPROM_PAGE_SIZE	EEPROM page size
EEPROM_PAGE_DW_NUM	Number of double words in EEPROM page
EEPROM_PAGE_NUM	EEPROM pages in a FLASH page
EEPROM_BACKUP_SIZE	Size of EEPROM backup
EEPROM_BACKUP_END_ADDR	End address of EEPROM backup
EEPROM_BACKUP_START_ADDR	Start address of EEPROM backup
EEPROM_PAGE_HEAD_FLAG	EEPROM start flag
EEPROM_PAGE_END_FLAG	EEPROM end flag
EEPROM_WORK_PAGE_FLAG	FLASH page use flag
FLASH_PAGES_PER_EEPROM_PAGE	FLASH page number per EEPROM page

3.1.2. API function

Function eeprom_init

The eeprom_init function is used to initialize the EEPROM backup area and obtain the relative number of the FLASH page currently being used for EEPROM backup.

Table 3-2. EEPROM initialization

```
void eeprom_init(void)
{
    uint16_t i = 0;
    uint8_t flag_mark_num = 0;
    uint64_t flag_work_page = 0;
    uint8_t check_ff_flag = 0;
    for(i = 0; i < EEPROM_FLASH_PAGE_NUM; i += FLASH_PAGES_PER_EEPROM_PAGE) {
        flag_work_page = REG64(EEPROM_BACKUP_START_ADDR + i * FLASH_PAGE_SIZE);
        check_ff_flag = check_ff(EEPROM_BACKUP_START_ADDR + i * FLASH_PAGE_SIZE,
FLASH_PAGE_SIZE * FLASH_PAGES_PER_EEPROM_PAGE);
        /* if the flash is without EEPROM_WORK_PAGE_FLAG but the page is not empty, erase
the flash page */
```

```
        if(((0xffffffffffffffff == flag_work_page) && (0x01 != check_ff_flag)) || ((0xffffffffffffffff !=
flag_work_page) && (EEPROM_WORK_PAGE_FLAG != flag_work_page))) {
            eeprom_block_erase(EEPROM_BACKUP_START_ADDR + i * FLASH_PAGE_SIZE);
        } else if(REG64(EEPROM_BACKUP_START_ADDR + i * FLASH_PAGE_SIZE) ==
EEPROM_WORK_PAGE_FLAG) {
            /* find the flash page with EEPROM_WORK_PAGE_FLAG marked */
            current_page = i;
            flag_mark_num++;
        }
    }
    /* no EEPROM_WORK_PAGE_FLAG is found */
    if(flag_mark_num == 0) {
        current_page = 0;
    }
    if(flag_mark_num > 1) {
        /* the first block is not the marked page */
        if(REG64(EEPROM_BACKUP_START_ADDR) == 0xffffffffffffffff) {
            if(REG64(EEPROM_BACKUP_START_ADDR + current_page * FLASH_PAGE_SIZE
+ 8 * 2) == EEPROM_PAGE_END_FLAG) {
                /* erase the page whose index is current_page-
FLASH_PAGES_PER_EEPROM_PAGE(the forward EEPROM block) */
                eeprom_block_erase(EEPROM_BACKUP_START_ADDR + (current_page -
FLASH_PAGES_PER_EEPROM_PAGE)*FLASH_PAGE_SIZE);
            } else {
                /* erase the page whose index is current_page, because
EEPROM_PAGE_END_FLAG is not found, and the data is incomplete, discard the data */
                eeprom_block_erase(EEPROM_BACKUP_START_ADDR + current_page *
FLASH_PAGE_SIZE);
                current_page -= FLASH_PAGES_PER_EEPROM_PAGE;
            }
        /* the first block is the marked block */
        } else {
            /* the marked block is the first block and the last block */
            if(FLASH_PAGES_PER_EEPROM_PAGE != current_page) {
                if(REG64(EEPROM_BACKUP_START_ADDR + 8 * 2) ==
EEPROM_PAGE_END_FLAG) {
                    eeprom_block_erase(EEPROM_BACKUP_START_ADDR + current_page *
FLASH_PAGE_SIZE);
                    current_page = 0;
                } else {
                    eeprom_block_erase(EEPROM_BACKUP_START_ADDR);
                }
            /* the marked block is the first block and the second block */
            } else {
```

```
                if(REG64(EEPROM_BACKUP_START_ADDR + current_page * FLASH_PAGE_SIZE + 8 * 2) == EEPROM_PAGE_END_FLAG) {
                    eeprom_block_erase(EEPROM_BACKUP_START_ADDR);
                } else {
                    eeprom_block_erase(EEPROM_BACKUP_START_ADDR + current_page * FLASH_PAGE_SIZE);
                    current_page = 0;
                }
            }
        }
    }
}
```

Function eeprom_write

The eeprom_write function is used to index the current writable address and write data to the corresponding FLASH address. Note that the parameter ee_addr of this function is an emulated EEPROM address, ranging from 0 to 2047.

Table 3-3. EEPROM write function

```
uint8_t eeprom_write(uint16_t ee_addr, uint8_t *data, uint16_t size)
{
    uint8_t ee_state = 0x01, i = 0;
    uint32_t block_addr = 0, ee_data_addr = 0;
    uint64_t temp_flag = 0;
    uint16_t tmp_size = 0, addr_tmp = 0;
    uint8_t *p_tmp = data;
    if(ee_addr + size > EEPROM_DATA_SIZE) {
        ee_state = 0x00;
        size = EEPROM_DATA_SIZE - ee_addr;
    }
    eeprom_read(0, (uint8_t *)record_buf, EEPROM_DATA_SIZE);
    tmp_size = size;
    addr_tmp = ee_addr;
    /* refresh the data in EEPROM */
    while(tmp_size--) {
        ((uint8_t *)record_buf)[addr_tmp++] = *p_tmp++;
    }
    /* find the block start address to write data */
    if(0xffffffffffffffff != REG64(EEPROM_BACKUP_START_ADDR + current_page * FLASH_PAGE_SIZE + 8)){
        if((EEPROM_FLASH_PAGE_NUM - FLASH_PAGES_PER_EEPROM_PAGE) == current_page){
            current_page = 0;
        }else{
```

```
            current_page = current_page + FLASH_PAGES_PER_EEPROM_PAGE;
        }
    }
    block_addr = EEPROM_BACKUP_START_ADDR + current_page * FLASH_PAGE_SIZE + 8;
    ee_data_addr = block_addr + 8 * 2;
    temp_flag = EEPROM_WORK_PAGE_FLAG;
    if(0 == flash_program(block_addr - 8, &temp_flag, 1)) {
        ee_state = 0x00;
    }
    /* write the EEPROM_PAGE_HEAD_FLAG */
    temp_flag = EEPROM_PAGE_HEAD_FLAG;
    if(0 == flash_program(ee_data_addr - 8 * 2, &temp_flag, 1)) {
        ee_state = 0x00;
    }
    /* write the data */
    if(0 == flash_program(ee_data_addr, record_buf, EEPROM_PAGE_DW_NUM)) {
        ee_state = 0x00;
    }
    /* read back data */
    flash_read_word(ee_data_addr, (uint8_t *)record_buf, EEPROM_DATA_SIZE);
    tmp_size = size;
    addr_tmp = ee_addr;
    while(tmp_size--) {
        /* check the data */
        if(((uint8_t *)record_buf)[addr_tmp++] != *data++) {
            ee_state = 0x00;
        }
    }
    /* write the EEPROM_PAGE_END_FLAG */
    if(ee_state == 0x01) {
        temp_flag = EEPROM_PAGE_END_FLAG;
        if(0 == flash_program(ee_data_addr - 8, &temp_flag, 1)) {
            ee_state = 0x00;
        }
    }
    if(ee_data_addr == block_addr + 8 * 2) {
        if(block_addr == EEPROM_BACKUP_START_ADDR + 8) {
            /* the current page is the last flash page */
            eeprom_block_erase(EEPROM_BACKUP_START_ADDR +
(EEPROM_FLASH_PAGE_NUM - FLASH_PAGES_PER_EEPROM_PAGE)*FLASH_PAGE_SIZE);
        } else {
            /* the current page is not the last flash page */
            eeprom_block_erase(EEPROM_BACKUP_START_ADDR + (current_page -
FLASH_PAGES_PER_EEPROM_PAGE)*FLASH_PAGE_SIZE);
```

```
        }
    }
    return ee_state;
}
```

Function eeprom_read

The eeprom_read function is used to read the latest data in the EEPROM backup area. Note that the entry ee_addr to this function is an EEPROM address, ranging from 0 to 135.

Table 3-4. EEPROM read function

```
uint8_t eeprom_read(uint16_t ee_addr, uint8_t *data, uint16_t size)
{
    uint8_t ee_state = 1, i = 0;
    uint32_t page_addr, ee_data_addr;
    /* find the page start address to write data */
    page_addr = EEPROM_BACKUP_START_ADDR + current_page * FLASH_PAGE_SIZE + 8;
    /* locate at the address to read data */
    ee_data_addr = page_addr + 8 * 2;
    if(ee_addr + size > EEPROM_DATA_SIZE) {
        ee_state = 0x00;
        size = EEPROM_DATA_SIZE - ee_addr;
    }
    /* read data */
    flash_read_word(ee_data_addr + ee_addr, data, size);
    return(ee_state);
}
```

3.1.3. Test result

Overwrites the first byte in EEPROM 16 times. The data to write is shown in ***Figure 3-1. Read and write data***.

Figure 3-1. Read and write data

```
uint8_t data[2048] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
                      0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F
                     };
```

The code is shown in ***Table 3-5. Test demo***.

Table 3-5. Test demo

```
int main(void)
{
    gd_eval_led_init(LED1);
    gd_eval_led_init(LED2);
    eeprom_init();
    eeprom_read(0, data_read, 2048);
```

```
        for(int i=0; i<16; i++){
            data[0] = i;
            eeprom_write(0, data, 2048);
            eeprom_read(0, data_read, 2048);
            if(SUCCESS != byte_memory_compare(data, data_read, 2048)) {
                gd_eval_led_on(LED2);
                return ERROR;
            }
        }
        gd_eval_led_on(LED1);
        while(1) {
        }
}
```

The test result is shown in ***Figure 3-2. Data in EEPROM backup***.

Figure 3-2. Data in EEPROM backup

```
0x08010400: F0F0F0F0 A55AA55A F0F0F0F0 AAAA5555 F0F0F0F0 5555AAAA 03020100 07060504 0B0A0908 0F0E0D0C 13121110 17161514 1B1A1918
0x08010434: 1F1E1D1C 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010468: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x0801049C: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x080104D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010504: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010538: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x0801056C: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x080105A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x080105D4: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010608: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x0801063C: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010670: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x080106A4: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x080106D8: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x0801070C: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010740: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010774: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x080107A8: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x080107DC: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010810: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010844: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010878: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x080108AC: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x080108E0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010914: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010948: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x0801097C: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x080109B0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x080109E4: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010A18: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010A4C: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010A80: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010AB4: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010AE8: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010B1C: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010B50: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010B84: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010BB8: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x08010BEC: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 FFFFFFFF FFFFFFFF
0x08010C20: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

4. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Mar.28, 2024