GigaDevice Semiconductor Inc.

Arm® Cortex®- M3/M4/M23/M33 32-bit MCU

Application Note

AN022

Table of Contents

List of Figures

List of Tables

1. Introduction

SDRAM is short for Synchronous Dynamic Random Access Memory. Synchronization means that memory work needs to synchronize clock, and internal command sending and data transmission are based on it; Dynamic means that the storage array needs constant refresh to ensure that the stored data is not lost. Because the data stored in SDRAM works through the capacitor, because the capacitor will discharge in the natural state, if the discharge is finished, it means that the data in SDRAM is lost, so SDRAM needs to refresh before the discharge of the capacitor is finished. Random means that the data is not stored in linear order, but read and write the data freely at the specified address.

Since SDRAM has some of the same characteristics as SRAM, if you need to run code on SDRAM, in addition to using MPU, you can also map SDRAM Device0 address (0xC0000000) to 0x00000000 to run the code stored in SDRAM. This article introduces how to map SDRAM Device0 address (0xC0000000) to 0x00000000 to run the code on GD32F450. Among them, APP_GPIO_Running_LED is the APP program programmed to SDRAM Device0 address (0xC0000000). BOOT_EXMC_SDRAM is the BOOT program programmed to address 0x08000000.

2. FLM file

2.1. Custom SDRAM_256Mb.FLM file

This AN is based on GD32F450Z-EVAL evaluation board, equipped with a MT48LZ16M16 A2P-6AIT SDRAM with a capacity of 256Mb.

When making FLM files, the starting address is 0x00000000. The details are shown in ***Figure 2-1. Address range of FLM file***.

Figure 2-1. Address range of FLM file

Description	Device Size	Device Type	Address Range
SDRAM_256Mb	256M	Unknown Device	00000000H - 0FFFFFFFH

2.2. Copy the FLM file to the Keil 5 installation path

Copy the SDRAM_256Mb.FLM file to the Keil 5 installation path, generally "C:\Keil_v5\ARM\Flash".

3. APP_GPIO_Running_LED project

3.1. Modify the project scatter-loading file

Since the APP_GPIO_Running_LED code is to run at address 0x00000000, it is necessary to modify the scatter-loading file of the project and load it to address 0x00000000. The details are shown in ***Table 3-1. Scatter-loading file of APP_GPIO_Running_LED***.

Table 3-1. Scatter-loading file of APP_GPIO_Running_LED

```
LR_IROM1 0x00000000 0x00100000   {      ; load region size_region
  ER_IROM1 0x00000000 0x00100000   {   ; load address = execution address
    *.o (RESET, +First)
    *(InRoot$$Sections)
    .ANY (+RO)
    .ANY (+XO)
  }
  RW_IRAM1 0x20000000 0x00030000   {   ; RW data
    .ANY (+RW +ZI)
  }
}
```

3.2. Modify the entry address of the interrupt vector table

Since the code itself runs at address 0x00000000, it is necessary to modify the entry address of the interrupt vector table of the APP project. You need to call nvic_vector_table_set at the appropriate location and set the entry address of the interrupt vector table to 0x00000000 and the offset to 0, as shown in ***Table 3-2. Setting the entry address of interrupt vector table***.

Table 3-2. Setting the entry address of interrupt vector table

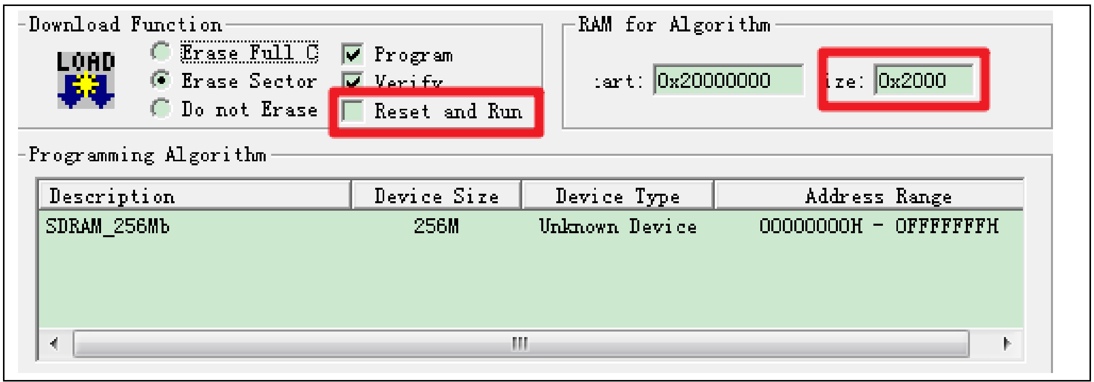
```
/*!
    \brief        main function
    \param[in]    none
    \param[out]   none
    \retval       none
*/
int main(void)
{
    nvic_vector_table_set(0, 0);
    /* configure systick */
    systick_config();

    ……
}
```

3.3. Add the SDRAM_256Mb.FLM file to the Keil project

Open the Keil project, in the Keil Flash download interface, add the SDRAM_256Mb.FLM file, and change the RAM for Algorithm Size to 0x2000, as shown in ***Figure 3-1. Configration of Flash Algorithm in Keil project***. Then download the program to SDRAM.

Figure 3-1. Configration of Flash Algorithm in Keil project



4. BOOT_EXMC_SDRAM project

In the BOOT_EXMC_SDRAM project, you need to change the BOOT_MODE of the SYSCFG_CFG0 register to 0b`100. The specific implementation is shown in ***Table 4-1. Set boot mode to EXMC SDRAM***.

Table 4-1. Set boot mode to EXMC SDRAM

```
……
rcu_periph_clock_enable(RCU_SYSCFG);
syscfg_bootmode_config(SYSCFG_BOOTMODE_EXMC_SDRAM);
……
```

If EXMC clock is enabled and EXMC SDRAM has been configured in BOOT_EXMC_SDRAM, you can observe the value of SDRAM through debugging to see if it is the APP_GPIO_Running_LED code. The details are shown in ***Figure 4-1. The value of 0xC0000000 in debug mode*** and ***Figure 4-2. The bin file compiled by the APP_GPIO_Running_LED project***.

Figure 4-1. The value of 0xC0000000 in debug mode

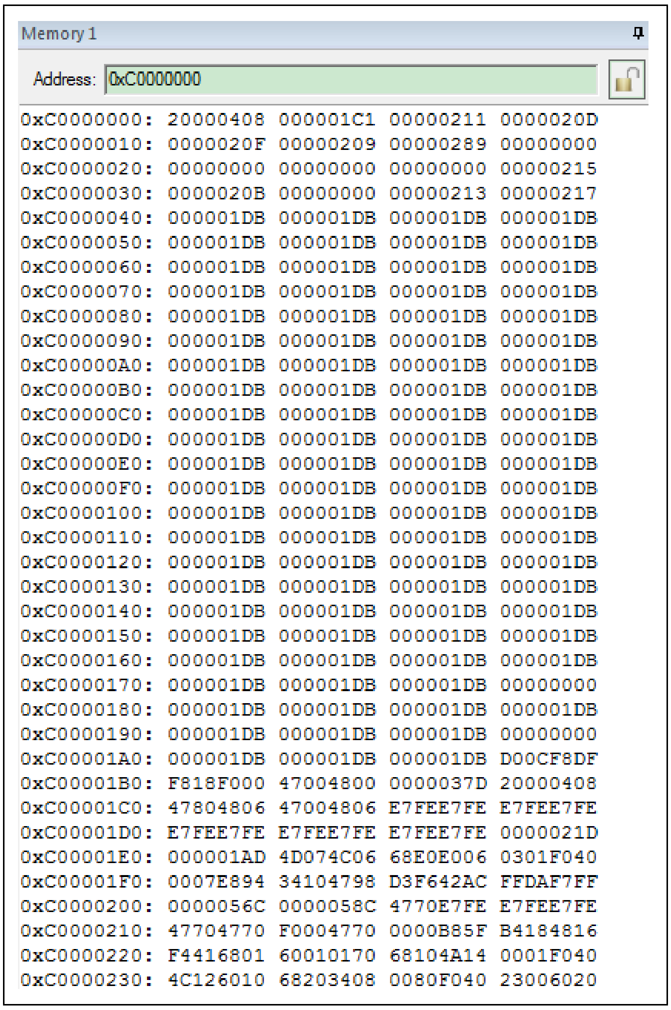


Figure 4-2. The bin file compiled by the APP_GPIO_Running_LED project

Address: 0x0 x1 x2 x4

Address	0	4	8	C	ASCII
0000	20000408	000001C1	00000211	0000020D	....
0010	0000020F	00000209	00000289	00000000	....
0020	00000000	00000000	00000000	00000215	....
0030	0000020B	00000000	00000213	00000217	....
0040	000001DB	000001DB	000001DB	000001DB	....
0050	000001DB	000001DB	000001DB	000001DB	....
0060	000001DB	000001DB	000001DB	000001DB	....
0070	000001DB	000001DB	000001DB	000001DB	....
0080	000001DB	000001DB	000001DB	000001DB	....
0090	000001DB	000001DB	000001DB	000001DB	....
00A0	000001DB	000001DB	000001DB	000001DB	....
00B0	000001DB	000001DB	000001DB	000001DB	....
00C0	000001DB	000001DB	000001DB	000001DB	....
00D0	000001DB	000001DB	000001DB	000001DB	....
00E0	000001DB	000001DB	000001DB	000001DB	....
00F0	000001DB	000001DB	000001DB	000001DB	....
0100	000001DB	000001DB	000001DB	000001DB	....
0110	000001DB	000001DB	000001DB	000001DB	....
0120	000001DB	000001DB	000001DB	000001DB	....
0130	000001DB	000001DB	000001DB	000001DB	....
0140	000001DB	000001DB	000001DB	000001DB	....
0150	000001DB	000001DB	000001DB	000001DB	....
0160	000001DB	000001DB	000001DB	000001DB	....

After configuring the BOOT_MODE of the SYSCFG_CFG0 register to 0b`100, you need to jump the program to the address where the APP_GPIO_Running_LED project is located to run the code. The specific configuration is shown in ***Table 4-2. Jump to the address where the APP project is located to run the code***.

Table 4-2. Jump to the address where the APP project is located to run the code

```
typedef    void (*pFunction)(void);
#define ApplicationAddress                  0xC0000000

pFunction Jump_To_Application;
uint32_t JumpAddress = 0;

......
    if (((*(__IO uint32_t*)ApplicationAddress) & 0x2FFE0000 ) == 0x20000000) {
        JumpAddress = *(__IO uint32_t*) (ApplicationAddress + 4);
        Jump_To_Application = (pFunction) JumpAddress;
        __set_MSP(*(__IO uint32_t*) ApplicationAddress);
        Jump_To_Application();
    }
......
```

After resetting the chip, it can be observed that the APP code is already running normally.

5. Revision history

Table 5-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Nov.30 2021

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.

© 2021 GigaDevice – All rights reserved