GigaDevice Semiconductor Inc.

Arm® Cortex®-M3/4 32-bit MCU

Application Note
AN024

Table of Contents

List of Figures

List of Tables

1. Introduction to scatter loading in KEIL

In the project generated by KEIL's default configuration, MDK will get the FLASH and RAM size information of the chip according to the chip model we configured in the option, and will automatically generate a scatter-loading named after the project name with the suffix * .sct File (Linker Control File, scatter loading), the linker determines the allocation address of each section on the memory according to the configuration of the generated scatter-loading file. Therefore, we can modify the file to store the specified code section in different locations.

This application note is based on the GD32F4xx series, using the GD32F450i-EVAL board, the keil version is 4.74.0.22, and the compiler version is V5.03.0.76, which describes how to implement the following functions:

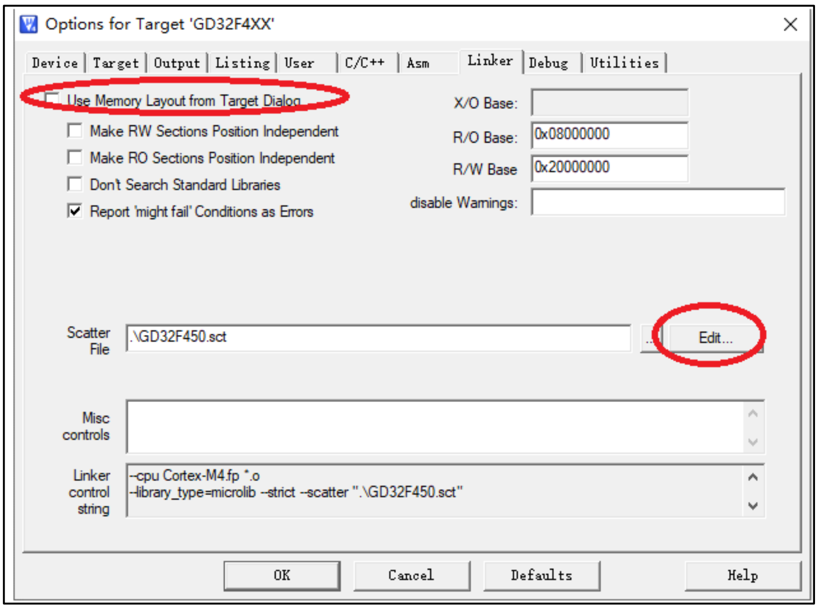
- Load global variables to the specified location.
- Load function to the specified location.
- Load array to the specified position.
- Load .c file to the specified location.
- The above function is loaded to the designated location of SDRAM.

2. Implementation of scatter-loading in KEIL

2.1. Use manually written sct files

This project directly uses the manually-written sct file. Uncheck the “Options for Target-> Linker-> Use Memory Layout from Target Dialog” option in MDK. After unchecking, you can directly click the “Edit” button to edit the sct file of the project The related configuration is shown in ***Figure 2-1. Use manually written sct file***.

Figure 2-1. Use manually written sct file



Similarly, you can also go to the project directory "GD32F4xx_ScatterLoading_v1.0.0 \ Project \ Keil_project \ MDK-ARM \ GD32F450.sct" to edit it, the file opening code is shown in ***Table 2-1. GD32F450.sct code***.

Table 2-1. GD32F450.sct code

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

LR_IROM1 0x08000000 0x0001ffff   {     ; load region size_region
  ER_IROM1 0x08000000 0x0001ffff   {   ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
  }
  RW_IRAM1 0x20000000 0x00001000   {   ; RW data
   .ANY (+RW +ZI)
  }
  ER_ISDRAM_FUNC 0xc0000000 0x00001000   {
```

```
        *(SDRAM_FUNC)
  }
  ER_ISDRAM_ARRAY 0xc0001000 0x00001000   {
        *(SDRAM_ARRAY)
  }
  ER_ISDRAM_OBJ 0xc0002000 0x00001000   {
         test.o (+RO)
  }
}

/*** Array scatter loading ***/
LR_IROM2 0x08020000 0x0001ffff {
  RW_IRAM_Array 0x20001000 0x00000020 {
   main.o(RAM_Array)
  }
}

/**** File scatter loading ***/
LR_IROM3 0x08040000 0x0001ffff {
  ER_IROM_Object 0x08040000 0x0001ffff {
    gd32f4xx_it.o (+RO)
  }
  RW_IRAM_Object 0x20001100 0x00000100 {
    hw_config.o (+RO)
  }
}
/****   Function scatter loading   ***/
LR_IROM4 0x08060000 0x0001ffff {
  ER_IROM_FUNC 0x08060000 0x0001ffff {
    main.o(ROM_FUNC)
  }
  ER_IRAM_FUNC 0x20001200 0x00000100 {
    main.o(SRAM_FUCN)
  }
}

/****   Variable scatter loading   ***/
LR_IROM5 0x08080000 0x0001ffff {
  ER_IROM_VARIABLE 0x08080000 0x0001ffff {
    main.o(ROM_VARIABLE)
  }
}
```

```
LR_IROM6 0x080a0000 0x0025ffff {
  ER_IROM4 0x080a0000 0x0025ffff {
   .ANY (+RO)
   .ANY (+XO)
  }
}
```

The red part is the main part of the code added to achieve the scattered loading function, which will be analyzed in detail below.

2.2. Load global variables to the specified location

Method 1: Add the following code to the GD32F450.sct file, as shown in *Table 2-2. GD32F450.sct loads the global variable to the specified location code*.

Table 2-2. GD32F450.sct loads the global variable to the specified location code

```
/****   Variable scatter loading   ***/
LR_IROM5 0x08080000 0x0001ffff {
  ER_IROM_VARIABLE 0x08080000 0x0001ffff {
    main.o(ROM_VARIABLE)
  }
}
```

The above code loads the ROM_VARIABLE section in the main.o module to the starting position of 0x08080000. The global variables defined in the main.c file are shown in *Table 2-3. Load the global variable to the specified location code in main.c 1*.

Table 2-3. Load the global variable to the specified location code in main.c 1

```
/* load the variable testValue_ROM to flash address 0x08080000 */
uint32_t testValue_ROM __attribute__((section("ROM_VARIABLE")))=5;
```

Method 2: Add __attribute __ ((at (xxx))) after the global variable. This routine defines the variable uint32_t testValue_RAM in main.c.

Table 2-4. Main.c loads the global variable to the specified location code 2

```
/* load the variable testValue_RAM to ram address 0x20003000 */
uint32_t testValue_RAM __attribute__((at(0x20003000)))=6;
```

Table 2-5. Load the global variable to the specified location and print the result

```
variable testValue_ROM address is 0x8080000
variable testValue_RAM address is 0x20003000
```

2.3. Load the function to the specified location

Add the following code to the GD32F450.sct file, as shown in ***Table 2-6. Load the function to the specified location code in GD32F450.sct***.

Table 2-6. Load the function to the specified location code in GD32F450.sct

```
/**** Function scatter loading ***/
LR_IROM4 0x08060000 0x0001ffff {
  ER_IROM_FUNC 0x08060000 0x0001ffff {
    main.o(ROM_FUNC)
  }
  ER_IRAM_FUNC 0x20001200 0x00000100 {
    main.o(SRAM_FUCN)
  }
}
```

The above code will load the ROM_FUNC section and SRAM_FUCN section in the main.o module to the starting position of 0x08060000 and 0x20001200, respectively. In the main.c file, allocate the delay function and the fill_TX_Data function to ROM_FUNC and SRAM_FUCN respectively, and the code is shown in ***Table 2-7. Load the function to the specified location code in main.c***.

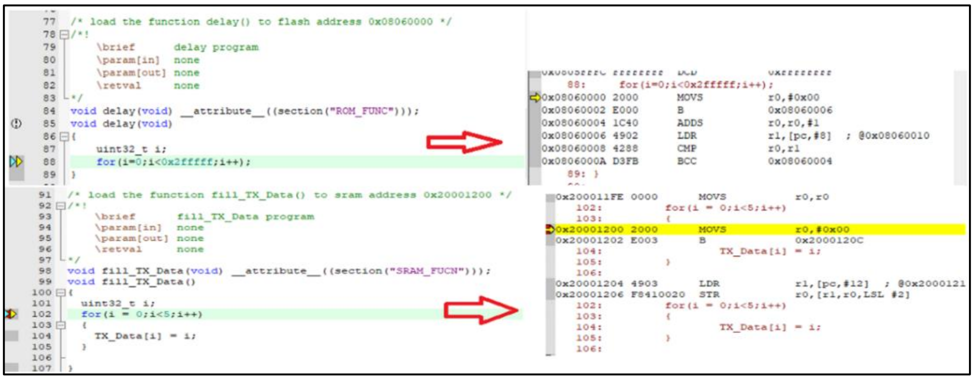
Table 2-7. Load the function to the specified location code in main.c

```
/* load the function delay() to flash address 0x08060000*/
/*
    \brief          delay program
    \param[in]      none
    \param[out]     none
    \retval         none
*/
void delay(void) __attribute__((section("ROM_FUNC")));
void delay(void)
{
    uint32_t i;
    for(i=0;i<0x2fffff;i++);
}
/* load the function fill_TX_Data() to sram address 0x20001200 */
/*
    \brief          fill_TX_Data program
    \param[in]      none
    \param[out]     none
    \retval         none
*/
void fill_TX_Data(void) __attribute__((section("SRAM_FUCN")));
```

```
void fill_TX_Data()
{
    uint32_t i;
    for(i = 0;i<5;i++)
    {
        TX_Data[i] = i;
    }
}
```

The program debugging results are shown in ***Figure 2-2. Use manually written sct file***.

Figure 2-2. Use manually written sct file



2.4. Load the array to the specified location

Method 1: Add the following code to the GD32F450.sct file, as shown in ***Table 2-8. Load the function to the specified location code in GD32F450.sct***.

Table 2-8. Load the function to the specified location code in GD32F450.sct

```
/*** Array scatter loading    ***/
LR_IROM2 0x08020000 0x0001ffff {
  RW_IRAM_Array 0x20001000 0x00000020 {
    main.o(RAM_Array)
  }
}
```

The above code loads the RAM_Array section in the main.o module to the starting position of 0x20001000, and defines the array TX_Data [] in main.c. The codes are shown in ***Table 2-9. Code to load the array to the specified location in main.c 1***.

Table 2-9. Code to load the array to the specified location in main.c 1

```
/* load the array TX_Data[5] to sram address 0x20001000 */
uint32_t TX_Data[5] __attribute__((section(".bss.RAM_Array")))={0};
```

Method 2: Add __attribute __ ((at (xxx))) after the array. This routine defines the array test in main.c and the const char constdata [] code in the const-data.c file as shown in ***Table 2-10. Code to load the array to the specified position in data.c.*** As shown, define test_sram [] in main.c, the code is shown in ***Table 2-11. Code to load the array to the specified location in main.c 2***.

Table 2-10. Code to load the array to the specified position in data.c

```
/* Load const array constdata to address 0x08001000 */
const char constdata[] __attribute__((at(0x08001000))) ={
      0x52,0x49,0x46,0x46,0xB4,0x5C,0x03,0x00,
      0x57,0x41,0x56,0x45,0x66,0x6D,0x74,0x20,
      0x10,0x00,0x00,0x00,0x01,0x00,0x02,0x00,
      0x80,0x3E,0x00,0x00,0x00,0xFA,0x00,0x00,
      0x04,0x00,0x10,0x00,0x64,0x61,0x74,0x61,
      0x90,0x5C,0x03,0x00,0x00,0x00,0x00,0x00,
       0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
       …
}
```

Table 2-11. Code to load the array to the specified location in main.c 2

```
/* load the array test_sram[5] to sram address 0x20007000*/
uint32_t test_sram[5] __attribute__((at(0x20007000)))={1,2,3,4,5};
```

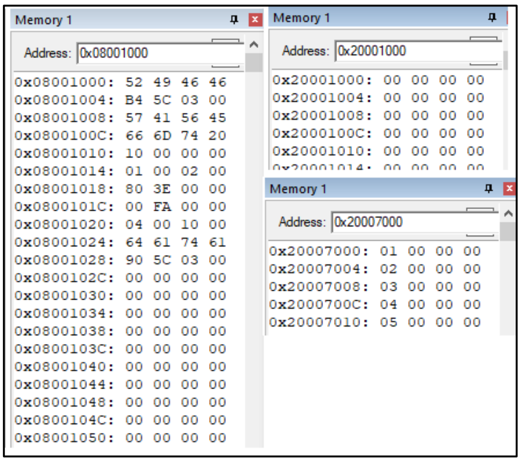
Print the array address through the printf function, the results are shown in ***Table 2-12. Load the array to the specified position and print the result***.

Table 2-12. Load the array to the specified position and print the result

```
constdata address is 0x8001000
TX_Data address is 0x20001000
test_sram address is 0x20007000
```

The program debugging results are shown in ***Figure 2-3. Debugging result of the array loaded to the specified position***.

Figure 2-3. Debugging result of the array loaded to the specified position



2.5. Load the .c file to the specified location

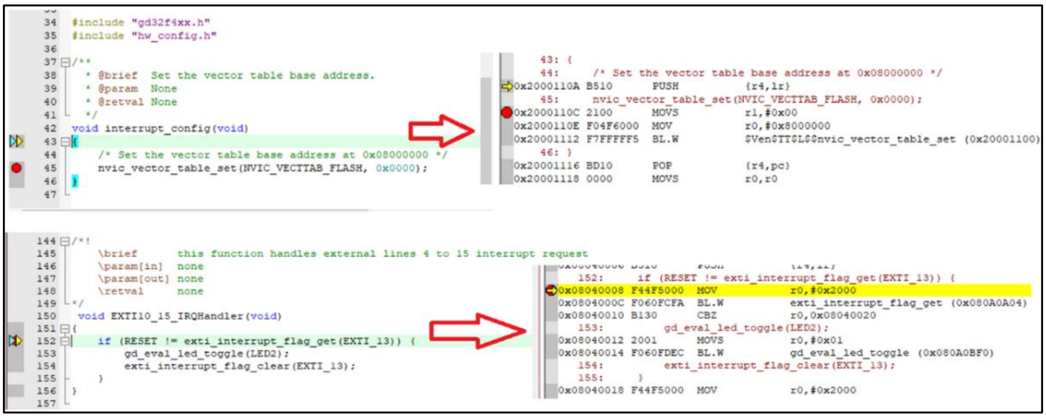
Add the following code to the GD32F450.sct file, as shown in ***Table 2-13. Code to load the file to the specified location in GD32F450.sct***.

Table 2-13. Code to load the file to the specified location in GD32F450.sct

```
/****   File scatter loading   ***/
LR_IROM3 0x08040000 0x0001ffff {
  ER_IROM_Object   0x08040000 0x0001ffff {
    gd32f4xx_it.o (+RO)
  }
  RW_IRAM_Object 0x20001100 0x00000100 {
     hw_config.o (+RO)
  }
}
```

The above code will load the gd32e230_it.o file to the starting position of 0x08040000 and the hw_config.o file to the starting position of 0x20001100. The debugging results of the program as shown in ***Figure 2-4. Debugging result of the .c file load to the specified location***.

Figure 2-4. Debugging result of the .c file load to the specified location



3. Scattered loading of SDRAM

3.1. The basic principle of scatter loading of SDRAM

In Cortex-M4 core, we can access the addresses above 0x2000 0000 and read data and instructions through the system bus, but in the default configuration of the kernel, some addresses are in the address segment that prohibits execution of instructions, so the code is loaded onto this segment, and an error occurs during execution. The address allocation of SDRAM in EXMC of GD32F450 is 0xC00000000-0xDFFFFFF located in this address segment.

In response to the above problems, there are two solutions to achieve scatter loading in SDRAM:

1. Configure the MPU (Memory Protect Unit) register to make the 0xC0000000 address segment executable (this example will use this implementation).

2. Adopt memory mapping method (map SDRAM address segment to executable area by configuring SYSCFG register).

3.2. Implementation of SDRAM distributed loading

Add the following red font codes to the GD32F450.sct file. The codes are shown in ***Table 3-1. SDRAM scatter-loading implementation code in GD32F450.sct***.

Table 3-1. SDRAM scatter-loading implementation code in GD32F450.sct

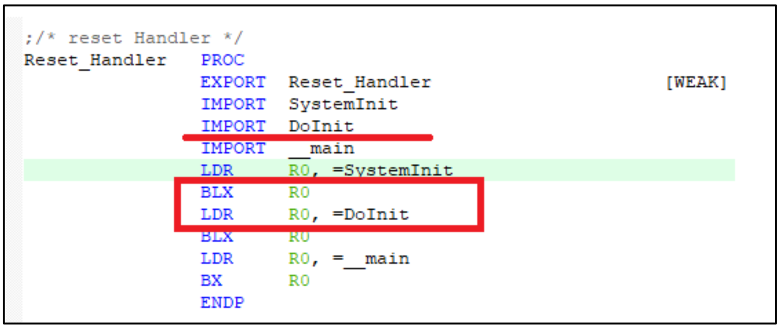
```
LR_IROM1 0x08000000 0x0001ffff {          ; load region size_region
  ER_IROM1 0x08000000 0x0001ffff {        ; load address = execution address
    *.o (RESET, +First)
    *(InRoot$$Sections)
  }
  RW_IRAM1 0x20000000 0x00001000 {   ; RW data
   .ANY (+RW +ZI)
  }
  ER_ISDRAM_FUNC 0xc0000000 0x00001000 {
    *(SDRAM_FUNC)
  }
  ER_ISDRAM_ARRAY 0xc0001000 0x00001000 {
    *(SDRAM_ARRAY)
  }
  ER_ISDRAM_OBJ 0xc0002000 0x00001000 {
    test.o (+RO)
  }
```

```
}
```

The above code will load the SDRAM_FUNC segment, SDRAM_ARRAY segment and test.o file to the starting addresses of 0xc0001000, 0xc0000000 and 0xc0002000 respectively.

Add the following code to startup_gd32f450.s, as shown in ***Figure 3-1. Add code to startup_gd32f450.s***.

Figure 3-1. Add code to startup_gd32f450.s



```
;/* reset Handler */
Reset_Handler    PROC
                 EXPORT  Reset_Handler                    [WEAK]
                 IMPORT  SystemInit
                 IMPORT  DoInit
                 IMPORT  __main
                 LDR     R0, =SystemInit
                 BLX     R0
                 LDR     R0, =DoInit
                 BLX     R0
                 LDR     R0, =__main
                 BX      R0
                 ENDP
```

The DoInt function is defined in main.c, which mainly implements EXMC initialization and MPU related configuration. The function codes are shown in ***Table 3-2. DoInit function implementation code***.

Table 3-2. DoInit function implementation code

```
/*!
    \brief      initialize the sdram, setup the MPU
    \param[in]  none
    \param[out] none
    \retval     none
*/
void DoInit(void)
{
    /* sdram peripheral initialize */
    exmc_synchronous_dynamic_ram_init(EXMC_SDRAM_DEVICE0);
    /* Configures the MPU regions */
    mpu_setup();
}
```

Define the variable uint32_t testValue_SDRAM in main.c, the array int test_sdram [5], the function testFuncInSDRAM, and add the file test.c. The main codes are shown in ***Table 3-3. Scatter-loading into the specified location code of SDRAM***.

Table 3-3. Scatter-loading into the specified location code of SDRAM

```
/* load the variable testValue_RAM to sdram address 0xC0003000 */
uint32_t testValue_SDRAM __attribute__((at(0xC0003000)));
/* load the array test_sdram[5] to sdram address 0xc0001000 */
uint32_t test_sdram[5] __attribute__((section("SDRAM_ARRAY")))={0};
/* load the function testFuncInSDRAM to sdram address 0xc0000000 */
```

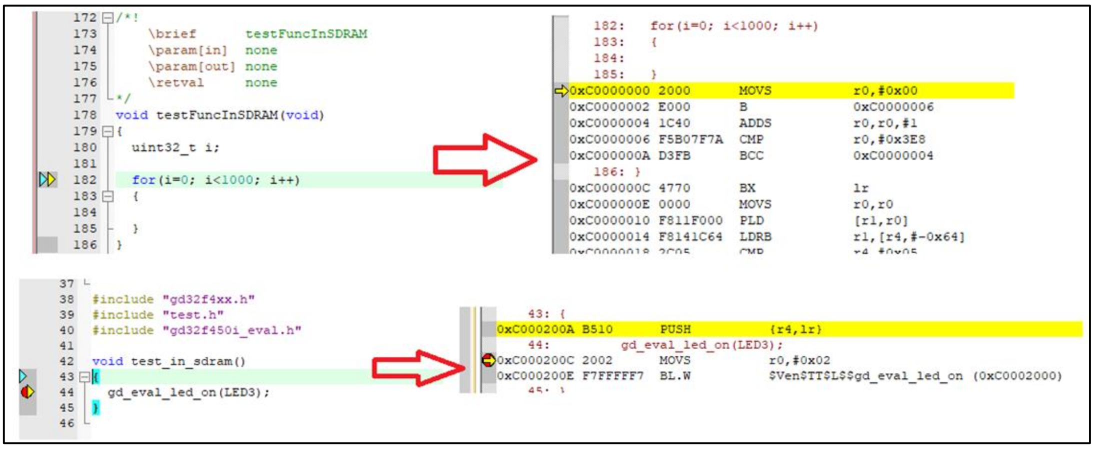
```
void testFuncInSDRAM(void) __attribute__((section("SDRAM_FUNC")));
/* test.c */
void test_in_sdram()
{
    gd_eval_led_on(LED3);
}
```

Table 3-4. Load variables and arrays to the specified location of SDRAM and the result and *Figure 3-2. Debugging result of loading the function and .c file to the designated location of SDRAM* show the results of program operation and debugging:

Table 3-4. Load variables and arrays to the specified location of SDRAM and the result

variable testValue_SDRAM address is 0xc0003000
test_sdram address is 0xc0001000

Figure 3-2. Debugging result of loading the function and .c file to the designated location of SDRAM



4. Results

View the "GD32F4XX_ScatterLoading_v1.0.0\Project\Keil\MDK-ARM\ Listings \ Project.map" file and open it as shown in ***Figure 4-1. Scatter loading project to compile Project.map file***.

Figure 4-1. Scatter loading project to compile Project.map file

```
Memory Map of the image
  Image Entry point : 0x080001ad
  Load Region LR_IROM1 (Base: 0x08000000, Size: 0x00009590, Max: 0x0001ffff, ABSOLUTE)
    Execution Region ER_ISDRAM_FUNC (Base: 0xc0000000, Size: 0x00000010, Max: 0x00001000, ABSOLUTE)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
    0xc0000000   0x0000000e   Code   RO          343    SDRAM_FUNC          main.o
    Execution Region ER_ISDRAM_ARRAY (Base: 0xc0001000, Size: 0x00000014, Max: 0x00001000, ABSOLUTE)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
    0xc0001000   0x00000014   Data   RW          354    SDRAM_ARRAY         main.o
    Execution Region ER_ISDRAM_OBJ (Base: 0xc0002000, Size: 0x00000014, Max: 0x00001000, ABSOLUTE)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
    0xc0002000   0x0000000a   Ven    RO         6342    Veneer$$Code        anon$$obj.o
    0xc000200a   0x0000000a   Code   RO          444    i.test_in_sdram     test.o

  Load Region LR_IROM2 (Base: 0x08020000, Size: 0x00000014, Max: 0x0001ffff, ABSOLUTE)
    Execution Region RW_IRAM_Array (Base: 0x20001000, Size: 0x00000014, Max: 0x00000020, ABSOLUTE)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
    0x20001000   0x00000014   Data   RW          352    .bss.RAM_Array      main.o

  Load Region LR_IROM3 (Base: 0x08040000, Size: 0x00000050, Max: 0x0001ffff, ABSOLUTE)
    Execution Region ER_IROM_Object (Base: 0x08040000, Size: 0x00000038, Max: 0x0001ffff, ABSOLUTE)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
    0x08040000   0x00000004   Code   RO          262    i.BusFault_Handler  gd32f4xx_it.o
    0x08040004   0x00000002   Code   RO          263    i.DebugMon_Handler  gd32f4xx_it.o
    0x08040006   0x0000001c   Code   RO          264    i.EXTI10_15_IRQHandler  gd32f4xx_it.o
    0x08040022   0x00000004   Code   RO          265    i.HardFault_Handler  gd32f4xx_it.o
    0x08040026   0x00000004   Code   RO          266    i.MemManage_Handler  gd32f4xx_it.o
    0x0804002a   0x00000002   Code   RO          267    i.NMI_Handler       gd32f4xx_it.o
    0x0804002c   0x00000002   Code   RO          268    i.PendSV_Handler    gd32f4xx_it.o
    0x0804002e   0x00000002   Code   RO          269    i.SVC_Handler       gd32f4xx_it.o
    0x08040030   0x00000002   Code   RO          270    i.SysTick_Handler   gd32f4xx_it.o
    0x08040032   0x00000004   Code   RO          271    i.UsageFault_Handler  gd32f4xx_it.o
    Execution Region RW_IRAM_Object (Base: 0x20001100, Size: 0x00000018, Max: 0x00000100, ABSOLUTE)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
    0x20001100   0x0000000a   Ven    RO         6343    Veneer$$Code        anon$$obj.o
    0x2000110a   0x0000000e   Code   RO          424    i.interrupt_config  hw_config.o

  Load Region LR_IROM4 (Base: 0x08060000, Size: 0x0000002c, Max: 0x0001ffff, ABSOLUTE)
    Execution Region ER_IROM_FUNC (Base: 0x08060000, Size: 0x00000014, Max: 0x0001ffff, ABSOLUTE)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
    0x08060000   0x00000014   Code   RO          342    ROM_FUNC            main.o
    Execution Region ER_IRAM_FUNC (Base: 0x20001200, Size: 0x00000018, Max: 0x00000100, ABSOLUTE)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
    0x20001200   0x00000018   Code   RO          344    SRAM_FUCN           main.o
  Load Region LR_IROM5 (Base: 0x08080000, Size: 0x00000004, Max: 0x0001ffff, ABSOLUTE)
    Execution Region ER_IROM_VARIABLE (Base: 0x08080000, Size: 0x00000004, Max: 0x0001ffff, ABSOLUTE)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
    0x08080000   0x00000004   Data   RW          353    ROM_VARIABLE        main.o
  Load Region LR$$.ARM.__AT_0x20002000 (Base: 0x20002000, Size: 0x00000000, Max: 0x00000020, ABSOLUTE)
    Execution Region ER$$.ARM.__AT_0x20002000 (Base: 0x20002000, Size: 0x00000020, Max: 0x00000020, ABSOLUTE, UNINIT)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
   0x20002000   0x00000020   Zero   RW            5   .ARM.__AT_0x20002000  gd32f4xx_mpu.o
  Load Region LR$$.ARM.__AT_0x20003000 (Base: 0x20003000, Size: 0x00000004, Max: 0x00000004, ABSOLUTE)
    Execution Region ER$$.ARM.__AT_0x20003000 (Base: 0x20003000, Size: 0x00000004, Max: 0x00000004, ABSOLUTE, UNINIT)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
    0x20003000   0x00000004   Data   RW          349    .ARM.__AT_0x20003000  main.o
  Load Region LR$$.ARM.__AT_0x20007000 (Base: 0x20007000, Size: 0x00000014, Max: 0x00000014, ABSOLUTE)
    Execution Region ER$$.ARM.__AT_0x20007000 (Base: 0x20007000, Size: 0x00000014, Max: 0x00000014, ABSOLUTE, UNINIT)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
    0x20007000   0x00000014   Data   RW          350    .ARM.__AT_0x20007000  main.o
  Load Region LR$$.ARM.__AT_0xC0003000 (Base: 0xc0003000, Size: 0x00000004, Max: 0x00000004, ABSOLUTE)
    Execution Region ER$$.ARM.__AT_0xC0003000 (Base: 0xc0003000, Size: 0x00000004, Max: 0x00000004, ABSOLUTE, UNINIT)
    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
    0xc0003000   0x00000004   Data   RW          351    .ARM.__AT_0xC0003000  main.o
```

From the map file, it can be seen that the load address and execution address of each segment conform to the specified scattered load area.

5. Revision history

Table 5-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Apr.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.