# Table of Contents

Table of Contents ........................................................................................................................................... 2  
List of Figures .................................................................................................................................................. 3  
List of Tables .................................................................................................................................................. 4  
1. Introduction to scatter loading in IAR ..................................................................................................... 5  
2. Implementation of scatter-loading in IAR ................................................................................................. 6  
   2.1. Use manually written icf files ............................................................................................................. 6  
   2.2. Load global variables to the specified location ................................................................................. 8  
   2.3. Load the function to the specified location ......................................................................................... 9  
   2.4. Load the array to the specified location ............................................................................................ 10  
   2.5. Load the .c file to the specified location .......................................................................................... 12  
3. Scattered loading of SDRAM .................................................................................................................... 14  
   3.1. The basic principle of scatter loading of SDRAM ........................................................................... 14  
   3.2. Implementation of SDRAM distributed loading ............................................................................. 14  
4. Results ....................................................................................................................................................... 18  
5. Revision history ......................................................................................................................................... 19
List of Figures

Figure 2-1. Use manually written icf file ................................................................. 6
Figure 2-2. Debugging results of the program loaded into the specified location .......... 10
Figure 2-3. Debugging result of the array loaded to the specified position .................. 12
Figure 2-4. Debugging result of the .c file load to the flash specified location .......... 13
Figure 2-5. Add the startup_gd32f4xx.s file to __iar_init $$ done .................................................. 13
Figure 2-6. Debugging result of the .c file load to the SRAM .................................. 13
Figure 3-1. Add code to startup_gd32f450.s in SDRAM scatter loading ..................... 14
Figure 3-2. Add code to startup_gd32f450.s .................................................................. 15
Figure 3-3. Debugging result of loading the function and .c file to the designated location of SDRAM ........................................................................................................ 17
Figure 4-1. Scatter loading project to compile Project.map file .................................. 18
List of Tables

Table 2-1. GD32F450.icf code .................................................................................................................................................. 6
Table 2-2. GD32F450.icf loads the global variable to the specified location code ................................................................. 8
Table 2-3. Load the global variable to the specified location code in main.c 1 ................................................................. 8
Table 2-4. Main.c loads the global variable to the specified location code 2 ................................................................. 9
Table 2-5. Load the global variable to the specified location and print the result ................................................................. 9
Table 2-6. Load the function to the specified location code in GD32F450.icf ................................................................. 9
Table 2-7. Load the function to the specified location code in main.c ......................................................................................... 9
Table 2-8. Load the function to the specified location code in GD32F450.icf ................................................................. 10
Table 2-9. Code to load the array to the specified location in main.c 1 ........................................................................ 11
Table 2-10. Code to load the array to the specified position in data.c .................................................................................. 11
Table 2-11. Code to load the array to the specified location in main.c 2 ........................................................................ 11
Table 2-12. Load the array to the specified position and print the result ........................................................................ 11
Table 2-13. Code to load the file to the specified location in GD32F450.icf ..................................................................... 12
Table 3-1. SDRAM scatter-loading implementation code in GD32F450.icf .................................................................... 14
Table 3-2. Dolnit function implementation code .................................................................................................................. 15
Table 3-3. Scatter-loading into the specified location code of SDRAM ................................................................................ 16
Table 3-4. Load variables and arrays to the specified location of SDRAM and the result .................................................. 17
Table 5-1. Revision history ................................................................................................................................................. 19
1. **Introduction to scatter loading in IAR**

In the project generated by the IAR default configuration, IAR will obtain the chip FLASH and SRAM size and other information according to the chip model we selected in the General option, and select the corresponding *.icf distributed loading file (Linker Control File, scatter loading). The linker allocates the address of each section according to the configuration of the file, and generates scattered loading code, so 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, IAR version is 7.40.2, respectively introduces how to achieve 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 IAR

2.1. Use manually written icf files

This project directly uses the manually-written icf file, select the override default in the "Project-> Option-> Linker-> Config-> Linker configuration file" option of IAR, click the "..." button after the selection, and select the project directory "GD32F4xx_ScatterLoading_v1.0.0\Project\IAR_project\GD32F450xK.icf ", the relevant configuration is shown in Figure 2-1. Use manually written icf file.

Figure 2-1. Use manually written icf file

Open GD32F450xK.icf for editing, the file opening code is shown in Table 2-1. GD32F450.icf code.

Table 2-1. GD32F450.icf code

```c
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x08000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x08000000;
define symbol __ICFEDIT_region_ROM_end__ = 0x08001fff;
define symbol __ICFEDIT_region_ROM1_start__ = 0x08002000;
define symbol __ICFEDIT_region_ROM1_end__ = 0x08003fff;
define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;
```
define symbol __ICFEDIT_region_RAM_end__ = 0x2002ffff;
define symbol __ICFEDIT_region_RAM1_start__ = 0x20001000;
define symbol __ICFEDIT_region_RAM1_end__ = 0x200011ff;
define symbol __ICFEDIT_region_RAM2_start__ = 0x20001200;
define symbol __ICFEDIT_region_RAM2_end__ = 0x200012ff;
define symbol __ICFEDIT_region_RAM3_start__ = 0x20001300;
define symbol __ICFEDIT_region_RAM3_end__ = 0x200013ff;
define symbol __ICFEDIT_region_SDRAM_start__ = 0xC0001000;
define symbol __ICFEDIT_region_SDRAM_end__ = 0xC0001fff;
define symbol __ICFEDIT_region_SDRAM1_start__ = 0xC0002000;
define symbol __ICFEDIT_region_SDRAM1_end__ = 0xC0002fff;
#define __ICFEDIT_size_cstack__ = 0x400;
#define __ICFEDIT_size_heap__ = 0x400;
define memory mem with size = 4G;
define region ROM_region = mem:[from __ICFEDIT_region_ROM_start__ to __ICFEDIT_region_ROM_end__];
define region ROM1_region = mem:[from __ICFEDIT_region_ROM1_start__ to __ICFEDIT_region_ROM1_end__];
define region RAM_region = mem:[from __ICFEDIT_region_RAM_start__ to __ICFEDIT_region_RAM_end__];
define region RAM1_region = mem:[from __ICFEDIT_region_RAM1_start__ to __ICFEDIT_region_RAM1_end__];
define region RAM2_region = mem:[from __ICFEDIT_region_RAM2_start__ to __ICFEDIT_region_RAM2_end__];
define region RAM3_region = mem:[from __ICFEDIT_region_RAM3_start__ to __ICFEDIT_region_RAM3_end__];
define region SDRAM_region = mem:[from __ICFEDIT_region_SDRAM_start__ to __ICFEDIT_region_SDRAM_end__];
define region SDRAM1_region = mem:[from __ICFEDIT_region_SDRAM1_start__ to __ICFEDIT_region_SDRAM1_end__];
#define block CSTACK with alignment = 8, size = __ICFEDIT_size_cstack__ { };
#define block HEAP with alignment = 8, size = __ICFEDIT_size_heap__ { }; initialize by copy { readwrite,section funram,object gd32f4xx_it.o }; do not initialize { section .noinit };
initialize manually {object test.o};
define block MYBLOCK { object test.o};
define block MYBLOCK_init {readonly object test.o};

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };
place at address mem:0x0800f000 { readonly section .funflash};
place at address mem:0x08002000 { section .text object hw_config.o };
place at address mem:0x08010000 { block MYBLOCK_init};
place at address mem:0xc0002000 { block MYBLOCK };
place in RAM_region   {  block CSTACK, block HEAP ,section .data,section .bss,
                             section sram };  
place in ROM_region   { readonly};
place in RAM1_region   { section funram};
place in ROM1_region   { readonly object gd32f4xx_it.o );
place in RAM2_region   { section variable};
place in RAM3_region   { section array};
place in SDRAM_region  { readwrite};
place in SDRAM1_region  { section sdram_array};

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: By defining the section variable, add the following code in the GD32F450xK.icf file, as shown in Table 2-2. GD32F450.icf loads the global variable to the specified location code.

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

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>define symbol <strong>ICFEDIT_region_RAM2_start</strong> = 0x20001200;</td>
</tr>
<tr>
<td>define symbol <strong>ICFEDIT_region_RAM2_end</strong> = 0x200012ff;</td>
</tr>
<tr>
<td>define region RAM2_region = mem:[from <strong>ICFEDIT_region_RAM2_start</strong> to <strong>ICFEDIT_region_RAM2_end</strong>];</td>
</tr>
<tr>
<td>place in RAM2_region { section variable};</td>
</tr>
</tbody>
</table>

Define the global variable uint32_t testValue_RAM in main.c, the code is 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_RAM to ram address 0x20001200 */
uint32_t testValue_RAM @$"variable"$=6;
```

Method 2: By adding the "@" operator to directly load the variable to the specified location, the code is as follows:
**Table 2-4. Main.c loads the global variable to the specified location code 2**

```c
/* load the variable testValue_ROM to flash address 0x08080000 */
uint32_t testValue_ROM @0x08080000 = 5;
```

Print the variable address through the printf function, the results are shown in **Table 2-5. Load the global variable to the specified location and print the result**.

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>testValue_ROM</td>
<td>0x08080000</td>
</tr>
<tr>
<td>testValue_RAM</td>
<td>0x20003000</td>
</tr>
</tbody>
</table>

**2.3. Load the function to the specified location**

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

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

```c
define symbol __ICFEDIT_region_RAM1_start__ = 0x20001000;
define symbol __ICFEDIT_region_RAM1_end__ = 0x20001100;
define region RAM1_region = mem:[from __ICFEDIT_region_RAM1_start__ to __ICFEDIT_region_RAM1_end__];
initialize by copy { readwrite, section funram, object gd32f4xx_it.o};
place at address mem:0x0800F000 { readonly section .funflash};
place in RAM1_region { section funram};
```

The above code will place section.funflash in the address space defined by 0x0800F000 and place section funram in the address space defined by RAM1_region by defining different regions.

In the main.c file, add the "@" or "#pragma location =" to allocate the delay function and the fill_TX_Data function to section .funflash and section funram respectively. 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**

```c
/* load the function delay() to flash address 0x0800F000 */

/*! 
    \brief delay program
    \param[in] none
    \param[out] none
    \retval none
*/
void delay(void) @".funflash"
{
    uint32_t i;
    for(i=0;i<0x2fffff;i++);
}
```
/* load the function fill_TX_Data() to sram address 0x20001000 */

/*
 \brief fill_TX_Data program
 \param[in]  none
 \param[out] none
 \retval none
*/

#pragma location = "funram"

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

The program debugging results are shown in Figure 2-2. Debugging results of the program loaded into the specified location.

Figure 2-2. Debugging results of the program loaded into the specified location

2.4. Load the array to the specified location

Method 1: By adding the above by defining the section array, add the following code in the GD32F450xK.icf file, as shown in Table 2-8. Load the function to the specified location code in GD32F450.icf.

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

<table>
<thead>
<tr>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>define symbol <strong>ICFEDIT_region_RAM3_start</strong> = 0x20001300;</td>
</tr>
<tr>
<td>define symbol <strong>ICFEDIT_region_RAM3_end</strong>   = 0x200013FF;</td>
</tr>
<tr>
<td>define region RAM3_region = mem:[from <strong>ICFEDIT_region_RAM3_start</strong> to</td>
</tr>
<tr>
<td>0x200013FF]</td>
</tr>
<tr>
<td>define symbol <strong>ICFEDIT_region_RAM4_start</strong> = 0x20001400;</td>
</tr>
<tr>
<td>define symbol <strong>ICFEDIT_region_RAM4_end</strong>   = 0x200017FF;</td>
</tr>
<tr>
<td>define region RAM4_region = mem:[from <strong>ICFEDIT_region_RAM4_start</strong> to</td>
</tr>
<tr>
<td>0x200017FF]</td>
</tr>
</tbody>
</table>
Define the array TX_Data[] in main.c, the code is 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 0x20001300 */
uint32_t TX_Data[5]@"array"={0};
```

**Method 2:** By adding the "@" operator to directly load the array to the specified location, the code is shown in Table 2-10. Code to load the array to the specified position in data.c.

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

```
/* Load const array constdata to address 0x8003000 */
const char constdata[@0x8003000]={
    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,
    ...
};
```

Print the array address through the printf function, the results are shown in Table 2-11. Code to load the array to the specified location in main.c 2.

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 0x8003000
TX_Data address is 0x20001300
```

The program debugging results are shown in 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 GD32F450xK.icf file, as shown in Table 2-13. Code to load the file to the specified location in GD32F450.icf.

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

```c
#define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;
#define symbol __ICFEDIT_region_RAM_end__ = 0x2002ffff;
#define symbol __ICFEDIT_region_ROM1_start__ = 0x08002000;
#define symbol __ICFEDIT_region_ROM1_end__ = 0x0800FFFF;
#define region RAM_region = mem:[from __ICFEDIT_region_RAM_start__ to __ICFEDIT_region_RAM_end__];
#define region ROM1_region = mem:[from __ICFEDIT_region_ROM1_start__ to __ICFEDIT_region_ROM1_end__];
initialize by copy { readwrite, section funram, object gd32f4xx_it.o};
place in RAM_region { readwrite, block CSTACK, block HEAP, section .data, section .bss, section sram};
place at address mem:0x08002000 { section .text object hw_config.o};
place in ROM1_region { readonly object gd32f4xx_it.o};
```

By loading the hw_config.o file to the address 0x08002000, the program debugging results are shown in Figure 2-4. Debugging result of the .c file load to the flash specified location.
Load the file gd32f4xx_it.c into the sram from ROM1_region (note that readwrite is placed in the defined RAM_Region). This routine is added to the gd32f4xx_it.c file by initialize by copy above, and __iar_init needs to be added to the startup code startup_gd32f4xx.s. As shown in Figure 2-5, Add the startup_gd32f4xx.s file to __iar_init $$ done.

The program debugging results are shown in Figure 2-6. Debugging result of the .c file load to the SRAM.

Note: This method can be used to load the .c file to the starting position of SRAM, and it needs to be loaded to the specified position. You can refer to the chapter SDRAM distributed loading method.
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 0xC0000000-0xDFFFFFF 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 red code in the following figure to startup_gd32f450.s, as shown in Figure 3-1. Add code to startup_gd32f450.s in SDRAM scatter loading.

Figure 3-1. Add code to startup_gd32f450.s in SDRAM scatter loading

The DoIint function is defined in main.c, which mainly implements EXMC initialization and MPU related configuration, and completes the copy of the function or .c file on SDRAM. The code is shown in Table 3-1. SDRAM scatter-loading implementation code in GD32F450xK.icf.

Table 3-1. SDRAM scatter-loading implementation code in GD32F450xK.icf

<table>
<thead>
<tr>
<th>Symbol Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ICFEDIT_region_SDRAM_start</strong></td>
</tr>
<tr>
<td><strong>ICFEDIT_region_SDRAM_end</strong></td>
</tr>
</tbody>
</table>
define symbol __ICFEDIT_region_SDRAM1_start__ = 0xC0002000;
define symbol __ICFEDIT_region_SDRAM1_end__ = 0xC0002fff;
define region SDRAM_region = mem:[from __ICFEDIT_region_SDRAM1_start__ to __ICFEDIT_region_SDRAM1_end__];
define region SDRAM1_region = mem:[from __ICFEDIT_region_SDRAM1_start__ to __ICFEDIT_region_SDRAM1_end__];
initialize by copy { readwrite,section funram,object gd32f4xx_it.o};
initialize manually {object test.o};
define block MYBLOCK { object test.o};
define block MYBLOCK_init {readonly object test.o};
place at address mem:0x08010000 { block MYBLOCK_init};
place at address mem:0xc0004000 { block MYBLOCK };
place in RAM_region { block CSTACK, block HEAP, section .data, section .bss, section sram }; // no readwrite
place in SDRAM_region { readwrite}; // Define readwrite in SDRAM_region, then the function specified by __ramfunc in IAR will be loaded into SDRAM
place in SDRAM1_region { section sdram_array};
The above code loads the sdram_array segment to the starting address of 0xC0002000, and manually copies the test.o file to the starting address of 0xc0004000, and loads the function specified by __ramfunc and gd32f4xx_it.o to the starting position of 0xc0000000 (note here that the difference between the .c file in the previous section and the scattered loading into RAM, here readwrite is placed in SDRAM_region).
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.

Figure 3-2. Add code to startup_gd32f450.s

```
/* reset Handler */
Reset_Handler PROC
IMPORT Reset_Handler
IMPORT SystemInit
IMPORT DoInit
IMPORT main
LDR R0, =SystemInit
BLX R0
LDR R0, =DoInit
BLX R0
LDR R0, =main
BR R0
ENDP
```

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

Table 3-2. Dolint function implementation code

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

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

/* load the variable testValue_SDRAM to ram address 0xC0003000 */
uint32_t testValue_SDRAM @0xC0003000;
/* load the array test_sdram[5] to sdram address 0xc0001000 */
#pragma location = "sdram_array"
uint32_t test_sdram[5] = {0};
/* load the function testFuncInSDRAM to sdram address 0xc0000000 */
void testFuncInSDRAM(void) __attribute__((section("SDRAM_FUNC")))
{ /*
    \brief test func in sdram
    \param[in] none
    \param[out] none
    \retval none
*/
    __ramfunc void testFuncInSDRAM(void)
    {
        uint32_t i;

        for(i=0; i<1000; i++)
        {
            asm("nop");
        }
    }
}
Table 3-4. Load variables and arrays to the specified location of SDRAM and the result

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>testValue_SDRAM</td>
<td>0xc0003000</td>
</tr>
<tr>
<td>test_sdram</td>
<td>0xc0002000</td>
</tr>
</tbody>
</table>

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

```c
gd_eval_led_on(LED3);
```

and Figure 3-2. Add code to startup_gd32f450.s show the results of program operation and debugging:
4. Results

View the "GD32F4xx_ScatterLoading_v1.0.0\Project\IAR\EWARM\Debug\List\Project.map" results 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**

<table>
<thead>
<tr>
<th>Section</th>
<th>Kind</th>
<th>Address</th>
<th>Size</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;A&quot;:</td>
<td>text</td>
<td>0x00002000</td>
<td>0x0</td>
<td>hiw_config.o [1]</td>
</tr>
<tr>
<td>&quot;D&quot;:</td>
<td>const</td>
<td>0x00002010</td>
<td>0x0</td>
<td>&lt;for D7 s4&gt;</td>
</tr>
<tr>
<td>&quot;D&quot;:</td>
<td>ro code</td>
<td>0x00002020</td>
<td>0x0</td>
<td></td>
</tr>
<tr>
<td>Absolute</td>
<td>sections, 1 of 4:</td>
<td>0x00002030</td>
<td>0x0</td>
<td></td>
</tr>
</tbody>
</table>
| .rodata     | const      | 0x00003000         | 0x0  | <const-data.o [1]
| Absolute    | sections, 2 of 4: | 0x00004000      | 0x0  |                 |
| .rodata     | const      | 0x00005000         | 0x0  |                 |
| Absolute    | sections 3 of 4: | 0x00006000      | 0x0  |                 |
| .rodata     | const      | 0x00007000         | 0x0  |                 |
| Absolute    | sections 4 of 4: | 0x00008000     | 0x0  |                 |
| .rodata     | const      | 0x00009000         | 0x0  |                 |
| Absolute    | sections 3 of 4: | 0x0000a000      | 0x0  |                 |
| .rodata     | const      | 0x0000b000         | 0x0  |                 |
| Absolute    | sections 4 of 4: | 0x0000c000      | 0x0  |                 |
| .rodata     | const      | 0x0000d000         | 0x0  |                 |
| Absolute    | sections 5 of 4: | 0x0000e000      | 0x0  |                 |
| .rodata     | const      | 0x0000f000         | 0x0  |                 |
| Absolute    | sections 6 of 4: | 0x00010000     | 0x0  |                 |
| .rodata     | const      | 0x00011000         | 0x0  |                 |
| Absolute    | sections 7 of 4: | 0x00012000     | 0x0  |                 |
| .rodata     | const      | 0x00013000         | 0x0  |                 |
| Absolute    | sections 8 of 4: | 0x00014000     | 0x0  |                 |
| .rodata     | const      | 0x00015000         | 0x0  |                 |
| Absolute    | sections 9 of 4: | 0x00016000     | 0x0  |                 |
| .rodata     | const      | 0x00017000         | 0x0  |                 |
| Absolute    | sections 10 of 4: | 0x00018000    | 0x0  |                 |
| .rodata     | const      | 0x00019000         | 0x0  |                 |
| Absolute    | sections 11 of 4: | 0x0001a000    | 0x0  |                 |
| .rodata     | const      | 0x0001b000         | 0x0  |                 |
| Absolute    | sections 12 of 4: | 0x0001c000    | 0x0  |                 |
| .rodata     | const      | 0x0001d000         | 0x0  |                 |
| Absolute    | sections 13 of 4: | 0x0001e000    | 0x0  |                 |
| .rodata     | const      | 0x0001f000         | 0x0  |                 |
| Absolute    | sections 14 of 4: | 0x00020000    | 0x0  |                 |
| .rodata     | const      | 0x00021000         | 0x0  |                 |
| Absolute    | sections 15 of 4: | 0x00022000    | 0x0  |                 |
| .rodata     | const      | 0x00023000         | 0x0  |                 |
| Absolute    | sections 16 of 4: | 0x00024000    | 0x0  |                 |
| .rodata     | const      | 0x00025000         | 0x0  |                 |
| Absolute    | sections 17 of 4: | 0x00026000    | 0x0  |                 |
| .rodata     | const      | 0x00027000         | 0x0  |                 |
| Absolute    | sections 18 of 4: | 0x00028000    | 0x0  |                 |
| .rodata     | const      | 0x00029000         | 0x0  |                 |
| Absolute    | sections 19 of 4: | 0x0002a000    | 0x0  |                 |
| .rodata     | const      | 0x0002b000         | 0x0  |                 |
| Absolute    | sections 20 of 4: | 0x0002c000    | 0x0  |                 |
| .rodata     | const      | 0x0002d000         | 0x0  |                 |
| Absolute    | sections 21 of 4: | 0x0002e000    | 0x0  |                 |
| .rodata     | const      | 0x0002f000         | 0x0  |                 |
| Absolute    | sections 22 of 4: | 0x00030000    | 0x0  |                 |
| .rodata     | const      | 0x00031000         | 0x0  |                 |
| Absolute    | sections 23 of 4: | 0x00032000    | 0x0  |                 |
| .rodata     | const      | 0x00033000         | 0x0  |                 |
| Absolute    | sections 24 of 4: | 0x00034000    | 0x0  |                 |
| .rodata     | const      | 0x00035000         | 0x0  |                 |
| Absolute    | sections 25 of 4: | 0x00036000    | 0x0  |                 |
| .rodata     | const      | 0x00037000         | 0x0  |                 |
| Absolute    | sections 26 of 4: | 0x00038000    | 0x0  |                 |
| .rodata     | const      | 0x00039000         | 0x0  |                 |
| Absolute    | sections 27 of 4: | 0x0003a000    | 0x0  |                 |
| .rodata     | const      | 0x0003b000         | 0x0  |                 |
| Absolute    | sections 28 of 4: | 0x0003c000    | 0x0  |                 |
| .rodata     | const      | 0x0003d000         | 0x0  |                 |
| Absolute    | sections 29 of 4: | 0x0003e000    | 0x0  |                 |
| .rodata     | const      | 0x0003f000         | 0x0  |                 |

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>
<thead>
<tr>
<th>Revision No.</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Initial Release</td>
<td>Apr.30, 2021</td>
</tr>
</tbody>
</table>
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 its 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 its 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.