GigaDevice Semiconductor Inc.

Instructions for Scatter Loading in IAR for GD32 Arm Cortex-M Processors

Application Note
AN221

Version 1.1

(Feb 2026)

Table of Contents

List of Figures

List of Table

1. Introduction to scatter loading in IAR

In the project generated with the default configuration in IAR, IAR retrieves information such as the FLASH and SRAM sizes of the MCU based on the chip model selected in the General Options. It selects the corresponding *.icf scatter loading file (Linker Control File, scatter loading), and the linker allocates the addresses of each section according to the configuration in this file, generating scatter loading code. Therefore, we can modify this file to specify the storage locations of code sections in different areas.

This application note is based on the GD32H737_757 series, using the GD32H759i-EVAL_V1.1 development board and IAR version 8.50.9. It explains how to achieve the following functionalities:

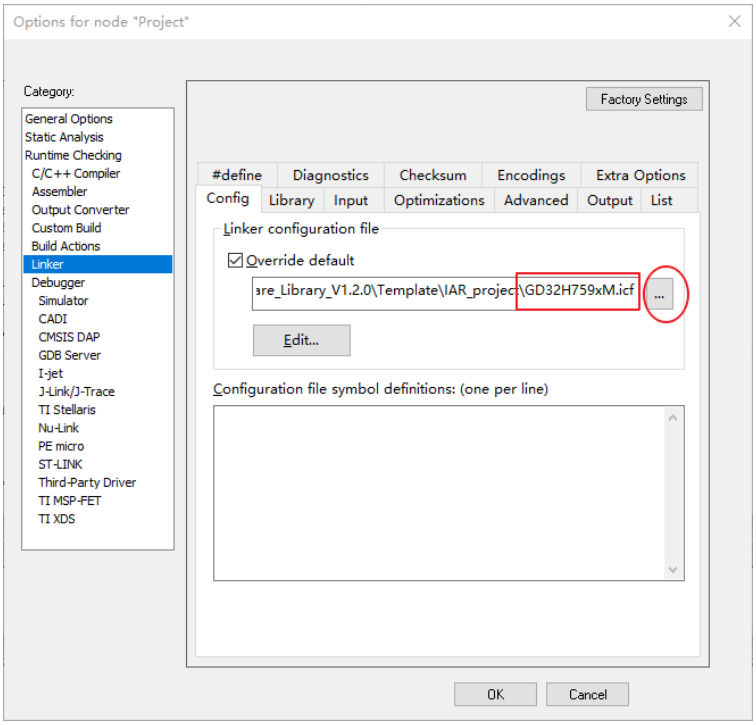
- Load global variables to a specified location
- Load functions to a specified location
- Load arrays to a specified location
- Implement .c file loading to a specified location
- Implement the above function loading to the specified SDRAM location.

2. Implementation of scatter-loading in IAR

2.1. Use manually written .icf file

This project directly uses a manually written .icf file. In IAR, under “Project->Option->Linker->Config->Linker configuration file,” check the override default option. After checking, click the “ ··· ” button and select the project directory “GD32H7xx_ScatterLoading_v1.0.0\Project\IAR_project\GD32H759xM.icf”. The relevant configuration is shown in *Figure 2-1. Use manually written icf file*.

Figure 2-1. Use manually written icf file



Open GD32H759xM.icf for editing, the file opening code is shown in *Table 2-1. GD32H759xM.icf code*.

Table 2-1. GD32H759xM.icf code

```
/*###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_ROM1_start__ = 0x08000000;
define symbol __ICFEDIT_region_ROM1_end__ = 0x0801FFFF;
```

```
define symbol __ICFEDIT_region_ROM2_start__ = 0x08020000;
define symbol __ICFEDIT_region_ROM2_end__   = 0x0802FFFF;
define symbol __ICFEDIT_region_ROM3_start__ = 0x08030000;
define symbol __ICFEDIT_region_ROM3_end__ = 0x0803FFFF;
define symbol __ICFEDIT_region_ROM4_start__ = 0x08040000;
define symbol __ICFEDIT_region_ROM4_end__ = 0x0804FFFF;
define symbol __ICFEDIT_region_ROM5_start__ = 0x08050000;
define symbol __ICFEDIT_region_ROM5_end__ = 0x0805FFFF;

define symbol __ICFEDIT_region_RAM1_start__ = 0x24000000;
define symbol __ICFEDIT_region_RAM1_end__ = 0x2400FFFF;
define symbol __ICFEDIT_region_RAM2_start__ = 0x24010000;
define symbol __ICFEDIT_region_RAM2_end__ = 0x2401FFFF;
define symbol __ICFEDIT_region_RAM3_start__ = 0x24020000;
define symbol __ICFEDIT_region_RAM3_end__   = 0x2402FFFF;
define symbol __ICFEDIT_region_RAM4_start__ = 0x24030000;
define symbol __ICFEDIT_region_RAM4_end__ = 0x2403FFFF;
define symbol __ICFEDIT_region_RAM5_start__ = 0x24040000;
define symbol __ICFEDIT_region_RAM5_end__   = 0x2404FFFF;
define symbol __ICFEDIT_region_RAM6_start__ = 0x24050000;
define symbol __ICFEDIT_region_RAM6_end__   = 0x2405FFFF;

define symbol __ICFEDIT_region_SDRAM1_start__ = 0xC0000000;
define symbol __ICFEDIT_region_SDRAM1_end__   = 0xC0000FFF;
define symbol __ICFEDIT_region_SDRAM2_start__ = 0xC0001000;
define symbol __ICFEDIT_region_SDRAM2_end__   = 0xC0001FFF;
define symbol __ICFEDIT_region_SDRAM3_start__ = 0xC0002000;
define symbol __ICFEDIT_region_SDRAM3_end__   = 0xC0002FFF;
define symbol __ICFEDIT_region_SDRAM4_start__ = 0xC0003000;
define symbol __ICFEDIT_region_SDRAM4_end__   = 0xC0003FFF;

/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x1000;
define symbol __ICFEDIT_size_heap__   = 0x1000;

define memory mem with size = 4G;
define region ROM1_region = mem:[from __ICFEDIT_region_ROM1_start__ to
__ICFEDIT_region_ROM1_end__];
define region ROM2_region = mem:[from __ICFEDIT_region_ROM2_start__ to
__ICFEDIT_region_ROM2_end__];
define region ROM3_region = mem:[from __ICFEDIT_region_ROM3_start__ to
__ICFEDIT_region_ROM3_end__];
```

```
define region ROM4_region = mem:[from __ICFEDIT_region_ROM4_start__ to __ICFEDIT_region_ROM4_end__];
define region ROM5_region = mem:[from __ICFEDIT_region_ROM5_start__ to __ICFEDIT_region_ROM5_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 RAM4_region = mem:[from __ICFEDIT_region_RAM4_start__ to __ICFEDIT_region_RAM4_end__];
define region RAM5_region = mem:[from __ICFEDIT_region_RAM5_start__ to __ICFEDIT_region_RAM5_end__];
define region RAM6_region = mem:[from __ICFEDIT_region_RAM6_start__ to __ICFEDIT_region_RAM6_end__];

define region SDRAM1_region = mem:[from __ICFEDIT_region_SDRAM1_start__ to __ICFEDIT_region_SDRAM1_end__];
define region SDRAM2_region = mem:[from __ICFEDIT_region_SDRAM2_start__ to __ICFEDIT_region_SDRAM2_end__];
define region SDRAM3_region = mem:[from __ICFEDIT_region_SDRAM3_start__ to __ICFEDIT_region_SDRAM3_end__];
define region SDRAM4_region = mem:[from __ICFEDIT_region_SDRAM4_start__ to __ICFEDIT_region_SDRAM4_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 SRAM_FUCN, section SDRAM_FUNC};
initialize manually {object hw_config.o};
initialize manually {object test.o};
define block FILE {object hw_config.o};
define block FILE_init {readonly object hw_config.o};
define block SDRAM_FILE {object test.o};
define block SDRAM_FILE_init {readonly object test.o};
do not initialize { section .noinit };

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in ROM1_region { readonly };
```

```
place in ROM2_region     { section ROM_FUNC };
place in ROM3_region { readonly object gd32h7xx_it.o };
place in ROM5_region     { block FILE_init, block SDRAM_FILE_init};
place in RAM1_region     { readwrite,
block CSTACK, block HEAP };
place in RAM2_region    { section .bss.RAM_Array};
place in RAM3_region    { section RAM_VARIABLE };
place in RAM4_region    { readwrite section RAM_Array};
place in RAM5_region    { section SRAM_FUCN };
place in RAM6_region    { block FILE};

place in SDRAM1_region    { section SDRAM_FUNC};
place in SDRAM2_region    { section SDRAM_ARRAY};
place in SDRAM3_region    { block SDRAM_FILE};
place in SDRAM4_region    { section SDRAM_VAR};
```

The red-highlighted section indicates the key additions for implementing scatter loading, which will be analyzed in detail below.

2.2. Load global variables to the specified location

In this example, the global variable 'uint32_t testValue_RAM' is defined in the 'main.c' file. By defining the section 'RAM_VARIABLE', the following code is added to the 'GD32H759xM.icf' file, as shown in ***Table 2-2. Code for loading global variables to the specified location in GD32H759xM.icf***.

Table 2-2. Code for loading global variables to the specified location in GD32H759xM.icf

```
define symbol __ICFEDIT_region_RAM3_start__ = 0x24020000;
define symbol __ICFEDIT_region_RAM3_end__     = 0x2402FFFF;
define region RAM3_region = mem:[from __ICFEDIT_region_RAM3_start__ to __ICFEDIT_region_RAM3_end__];
place in RAM3_region    { section RAM_VARIABLE };
```

In 'main.c', the global variable 'uint32_t testValue_RAM' is defined. The code is shown in ***Table 2-3. Code for loading global variables to the specified location in 'main.c'***.

Table 2-3. Code for loading global variables to the specified location in 'main.c'

```
/* load the variable testValue_RAM to RAM_VARIABLE section */
int testValue_RAM @"RAM_VARIABLE" = 0xCC;
```

Using the 'printf' function to print the variable address, the result is shown in ***Table 2-4. Load global variables to specified location and print results***.

Table 2-4. Load global variables to specified location and print results

testValue_RAM address is 0x24020000, value is 0xCC

2.3. Load functions to specified location

2.3.1. Load functions to the specified location in FLASH

Add the following code to the GD32H759xM.icf file, as shown in ***Table 2-5. Code for loading functions to the specified location in FLASH in GD32H759xM.icf***.

Table 2-5. Code for loading functions to the specified location in FLASH in GD32H759xM.icf

```
define symbol __ICFEDIT_region_ROM2_start__ = 0x08020000;
define symbol __ICFEDIT_region_ROM2_end__   = 0x0802FFFF;
define region ROM2_region = mem:[from __ICFEDIT_region_ROM2_start__ to __ICFEDIT_region_ROM2_end__];
place in ROM2_region { section ROM_FUNC };
```

The above code defines a region to place the section ROM_FUNC in the address space defined by ROM2_region. In the main.c file, the delay function is assigned to section ROM_FUNC using “@” as shown in ***Table 2-6. Code for loading functions to the specified location in main.c***.

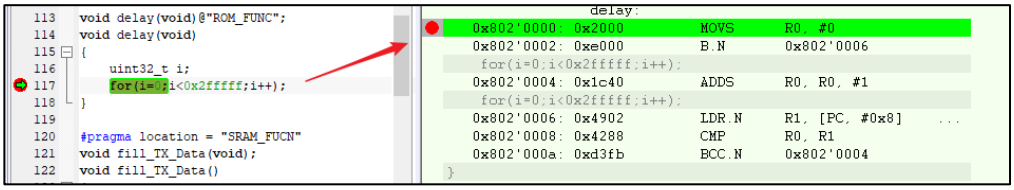
Table 2-6. Code for loading functions to the specified location in main.c

```
/*
\brief        delay program
\param[in]    None
\param[out] None
\retval       None
*/
void delay(void)@”ROM_FUNC”;
void delay(void)
{
uint32_t I;
for(i=0;i<0x2fffff;i++);
}
```

Debugging results of the program as shown in ***Figure 2-2. Debugging results for loading functions to the specified location in FLASH***.

Figure 2-2. Debugging results for loading functions to the specified location in

FLASH



2.3.2. Load functions to the specified location in SRAM

Add the following code to the GD32H759xM.icf file, as shown in ***Table 2-7. Code for loading functions to the specified location in SRAM in GD32H759xM.icf***.

Table 2-7. Code for loading functions to the specified location in SRAM in GD32H759xM.icf

```
define symbol __ICFEDIT_region_RAM5_start__ = 0x24040000;
define symbol __ICFEDIT_region_RAM5_end__ = 0x2404FFFF;
define symbol __ICFEDIT_region_ROM2_end__ = 0x0802FFFF;
define region RAM5_region = mem:[from __ICFEDIT_region_RAM5_start__ to __ICFEDIT_region_RAM5_end__];
initialize by copy { readwrite, section SRAM_FUCN };
place in RAM5_region { section SRAM_FUCN };
```

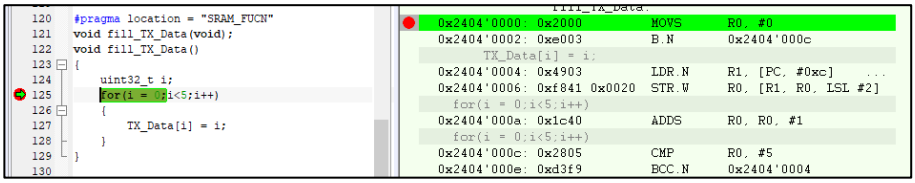
The above code defines a region to place the section SRAM_FUNC in the address space defined by RAM5_region. In the main.c file, the fill_TX_Data function is assigned to section SRAM_FUNC using “#pragma location =”, as shown in ***Table 2-8. Code for loading functions to the specified location in SRAM in main.c***.

Table 2-8. Code for loading functions to the specified location in SRAM in main.c

```
#pragma location = “SRAM_FUCN”
void fill_TX_Data(void);
void fill_TX_Data()
{
uint32_t I;
for(I = 0; I < 5; i++)
{
TX_Data[i] = I;
}
}
```

Debugging results of the program as shown in ***Figure 2-3. Function loaded to SRAM at a specific location program debugging result***.

Figure 2-3. Function loaded to SRAM at a specific location program debugging result



2.4. Load the array to the specified location

2.4.1. Load the global array to the specified SRAM location

By defining section RAM_Array and adding the following code in the GD32H759xM.icf file, as shown in ***Table 2-9. Code in GD32H759xM.icf for loading the global array to the specified SRAM location***:

Table 2-9. Code in GD32H759xM.icf for loading the global array to the specified SRAM location

```
initialize by copy   { readwrite, section SRAM_FUCN, section SDRAM_FUNC};
define symbol __ICFEDIT_region_RAM4_start__ = 0x24030000;
define symbol __ICFEDIT_region_RAM4_end__ = 0x2403FFFF;
define region RAM4_region = mem:[from __ICFEDIT_region_RAM4_start__ to __ICFEDIT_region_RAM4_end__];
place in RAM4_region { section RAM_Array };
```

Define the array test_sram[5] in main.c, as shown in ***Table 2-10. Code in main.c for loading the global array to the specified SRAM location***.

Table 2-10. Code in main.c for loading the global array to the specified SRAM location

```
#pragma location = “RAM_Array”
uint32_t test_sram[5] = {1,2,3,4,5};
```

Print the variable address using the printf function, as shown in ***Table 2-11. Print results of loading the global array to the specified SRAM location***.

Table 2-11. Print results of loading the global array to the specified SRAM location

```
test_sram address is 0x24030000
```

2.4.2. Load the constant array to the specified FLASH location

By adding the “@” operator, directly load the array to the specified location, as shown in ***Table 2-12. Code for loading the constdata constant array to the specified FLASH location***.

Table 2-12. Code for loading the constdata constant array to the specified FLASH

location

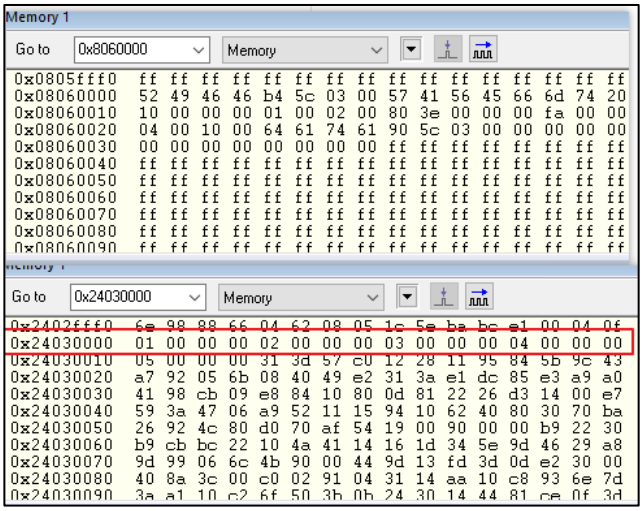
```
const char constdata[]@0x08060000 = {
0x52,0x49,0x46,0x46,0xB4,0x5C,0x03,0x00,
0x57,0x41,0x56,0x45,0x66,0x6D,0x74,0x20,
0x10,0x00,0x00,0x00,0x01,0x00,0x02,0x00,
0x80,0x3E,0x00,0x00,0x00,0xFA,0x00,0x00,
0x04,0x00,0x10,0x00,0x64,0x61,0x74,0x61,
0x90,0x5C,0x03,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
    };
```

Print the array address using the printf function, as shown in *Table 2-13. Print results of loading the constant array to the specified FLASH location*.

Table 2-13. Print results of loading the constant array to the specified FLASH location

```
constdata address is 0x08060000
```

Figure 2-4. Program debugging result of loading the array to the specified location



2.5. Load the .c file code segment to the specified location

2.5.1. Load the .c file code segment to the specified location in FLASH

In the GD32H759xM.icf file, the code is shown in *Table 2-14. Code for loading the .c file code segment to the specified location in FLASH in GD32H759xM.icf*.

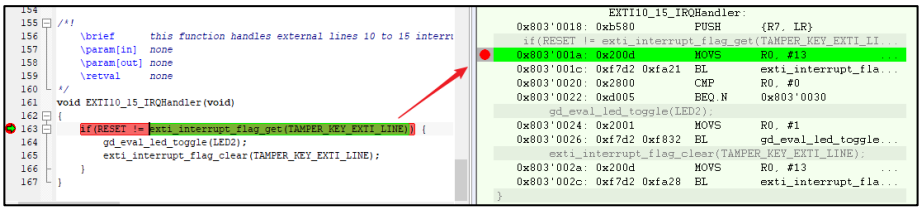
Table 2-14. Code for loading the .c file code segment to the specified location in FLASH in GD32H759xM.icf

```
define symbol __ICFEDIT_region_ROM3_start__ = 0x08030000;
define symbol __ICFEDIT_region_ROM3_end__ = 0x0803FFFF;
```

```
define region ROM3_region = mem:[from __ICFEDIT_region_ROM3_start__ to __ICFEDIT_region_ROM3_end__];
place in ROM3_region { readonly object gd32h7xx_it.o };
```

By loading the ro section of the gd32h7xx_it.o file to address 0x08030000, the program debugging results are shown in ***Figure 2-5. Loading the .c file code segment to the specified location in FLASH***.

Figure 2-5. Loading the .c file code segment to the specified location in FLASH



2.5.2. Load the .c file code segment to the specified location in SRAM

In the GD32H759xM.icf file, the code is shown in ***Table 2-15. Code for loading the .c file code segment to the specified location in SRAM in GD32H759xM.icf***.

Table 2-15. Code for loading the .c file code segment to the specified location in SRAM in GD32H759xM.icf

```
define symbol __ICFEDIT_region_ROM5_start__ = 0x08050000;
define symbol __ICFEDIT_region_ROM5_end__ = 0x0805FFFF;
define symbol __ICFEDIT_region_RAM6_start__ = 0x24050000;
define symbol __ICFEDIT_region_RAM6_end__ = 0x2405FFFF;
define region ROM5_region = mem:[from __ICFEDIT_region_ROM5_start__ to __ICFEDIT_region_ROM5_end__];
define region RAM6_region = mem:[from __ICFEDIT_region_RAM6_start__ to __ICFEDIT_region_RAM6_end__];
initialize manually {object hw_config.o};
define block FILE {object hw_config.o};
define block FILE_init {readonly object hw_config.o};
place in ROM5_region { block FILE_init };
place in RAM6_region { block FILE};
```

By manually transferring the ro section of the hw_config.o file defined in the ROM5_region on FLASH, load it to RAM6_region. Add the transfer code function file_init in main.c, and execute the file_init function before calling functions in hw_config.c to complete loading the .c file ro section to SRAM, as shown in ***Table 2-16. Transfer code added to Main.c***. The program debugging results are shown in ***Figure 2-6. Loading the .c file code segment to the specified location in SRAM***.

Table 2-16. Transfer code added to Main.c

```
#pragma section = “FILE_init”
```

```
#pragma section = “FILE”
void flie_init(void)
{
/* Data Copy */
char * from = __section_begin(“FILE_init”);
char * to = __section_begin(“FILE”);
‘memcpy(to, from, __section_size(“FILE”));’
}
```

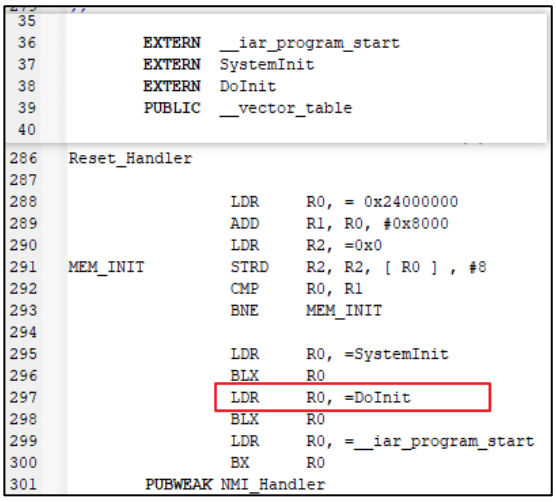
Figure 2-6. Loading the .c file code segment to the specified location in SRAM

3. Implementation of SDRAM scatter loading

3.1. Implementation of SDRAM scatter loading

Add the code marked in red in startup_gd32h7xx.s as shown in ***Figure 3-1. Add code to 'startup_gd32h7xx.s' for SDRAM scatter loading implementation***.

Figure 3-1. Add code to 'startup_gd32h7xx.s' for SDRAM scatter loading implementation



The 'DoInit' function is defined in 'main.c'. This function mainly implements EXMC initialization, MPU-related configuration, and completes the copying of functions or '.c' files to SDRAM. The code is shown in ***Table 3-1. 'DoInit' function code***.

Table 3-1. 'DoInit' function code

```
'#pragma section = "SDRAM_FILE_init"'
'#pragma section = "SDRAM_FILE"'
/*!
\brief        Initialize the SDRAM
\param[in]    None
\param[out] None
\retval       None
*/
'void DoInit(void)'
{
/* Configure the clock of EXMC */
rcu_exmc_config();

/* Configure the MPU */
mpu_config();
```

```
/* Configure the EXMC access mode */
exmc_synchronous_dynamic_ram_init(EXMC_SDRAM_DEVICE0);
‘__IO int I, j;’
for(i=0;i<500;i++){
for(j=0;j<5000;j++);
}
/* Data Copy */
‘char * from = __section_begin(“SDRAM_FILE_init”);’
‘char * to = __section_begin(“SDRAM_FILE”);’
memcpy(to, from, __section_size(“SDRAM_FILE”));
}
```

Note: In the default configuration of the M7 core, certain addresses are in regions where instruction execution is prohibited. Therefore, if code is loaded into these regions, errors will occur during execution. The SDRAM address allocation in the EXMC of GD32H7xx is ‘0xC0000000-0xDFFFFFFF’, which is located in such a region. By configuring the MPU (Memory Protection Unit) registers, the ‘0xC0000000’ region can be made executable. The MPU configuration code is shown in ***Table 3-2. MPU configuration code***.

Table 3-2. MPU configuration code

```
/*!
\brief        configure MPU
\param[in]    none
\param[out]   none
\retval       none
*/
void mpu_config(void)
{
mpu_region_init_struct mpu_init_struct;
mpu_region_struct_para_init(&mpu_init_struct);

/* disable the MPU */
ARM_MPU_Disable();
ARM_MPU_SetRegion(0, 0);

/* configure the MPU attributes for SDRAM */
mpu_init_struct.region_base_address = SDRAM_DEVICE0_ADDR;
mpu_init_struct.region_size             = MPU_REGION_SIZE_32MB;
mpu_init_struct.access_permission     = MPU_AP_FULL_ACCESS;
mpu_init_struct.access_bufferable     = MPU_ACCESS_NON_BUFFERABLE;
mpu_init_struct.access_cacheable      = MPU_ACCESS_CACHEABLE;
mpu_init_struct.access_shareable      = MPU_ACCESS_NON_SHAREABLE;
mpu_init_struct.region_number          = MPU_REGION_NUMBER0;
```

```
mpu_init_struct.subregion_disable     = MPU_SUBREGION_ENABLE;
mpu_init_struct.instruction_exec      = MPU_INSTRUCTION_EXEC_PERMIT;
mpu_init_struct.tex_type              = MPU_TEX_TYPE0;
mpu_region_config(&mpu_init_struct);
mpu_region_enable();
/* Enable the MPU */
ARM_MPU_Enable(MPU_MODE_PRIV_DEFAULT);
}
```

Add the following code to the 'GD32H759xM.icf' file. The code is shown in ***Table 3-3. SDRAM scatter loading code in 'GD32H759xM.icf'***.

Table 3-3. SDRAM scatter loading code in 'GD32H759xM.icf'

```
define symbol __ICFEDIT_region_SDRAM1_start__ = 0xC0000000;
define symbol __ICFEDIT_region_SDRAM1_end__   = 0xC0000FFF;
define symbol __ICFEDIT_region_SDRAM2_start__ = 0xC0001000;
define symbol __ICFEDIT_region_SDRAM2_end__   = 0xC0001FFF;
define symbol __ICFEDIT_region_SDRAM3_start__ = 0xC0002000;
define symbol __ICFEDIT_region_SDRAM3_end__   = 0xC0002FFF;
define symbol __ICFEDIT_region_SDRAM4_start__ = 0xC0003000;
define symbol __ICFEDIT_region_SDRAM4_end__   = 0xC0003FFF;
define region SDRAM1_region = mem:[from __ICFEDIT_region_SDRAM1_start__ to __ICFEDIT_region_SDRAM1_end__];
define region SDRAM2_region = mem:[from __ICFEDIT_region_SDRAM2_start__ to __ICFEDIT_region_SDRAM2_end__];
define region SDRAM3_region = mem:[from __ICFEDIT_region_SDRAM3_start__ to __ICFEDIT_region_SDRAM3_end__];
define region SDRAM4_region = mem:[from __ICFEDIT_region_SDRAM4_start__ to __ICFEDIT_region_SDRAM4_end__];
initialize by copy  { readwrite, section SRAM_FUCN, section SDRAM_FUNC};
initialize manually {object test.o };
define block SDRAM_FILE   {object test.o};
define block SDRAM_FILE_init {readonly object test.o};
place in ROM5_region    { block FILE_init, block SDRAM_FILE_init};
place in SDRAM1_region   { section SDRAM_FUNC};
place in SDRAM2_region   { section SDRAM_ARRAY};
place in SDRAM3_region   { block SDRAM_FILE};
place in SDRAM4_region   { section SDRAM_VAR};
```

The above code loads the 'SDRAM_ARRAY' section into 'SDRAM2_region' and uses manual copying to load the 'test.o' file into 'SDRAM3_region'.

In 'main.c', define the global variable 'uint32_t testValue_SDRAM', the uninitialized global array 'uint32_t test_sdram[5]', and the function 'testFuncInSDRAM'. Additionally, include the file 'test.c'. The main code is shown in ***Table 3-4. Code for scatter loading variables, arrays,***

functions, and files to specific SDRAM locations, *Table 3-5. Printed Results of Loading Variables and Arrays to Specific SDRAM Locations*, and *Figure 3-3. CMSIS-DAP reset option configuration*.

Table 3-4. Code for scatter loading variables, arrays, functions, and files to specific SDRAM locations

```
uint32_t testValue_SDRAM@"SDRAM_VAR" = 5;
uint32_t test_sdram[5]@"SDRAM_ARRAY";
void testFuncInSDRAM(void)@"SDRAM_FUNC";

void testFuncInSDRAM()
{
uint32_t I;
for(I = 0; I < 0xffff; i++){
}
}

test.c
#include "gd32h7xx.h"
#include "test.h"
#include "gd32h759i_eval.h"
#include <stdio.h>

void test_in_sdram()
{
gd_eval_led_toggle(LED1);
}
```

Table 3-5. Printed Results of Loading Variables and Arrays to Specific SDRAM Locations

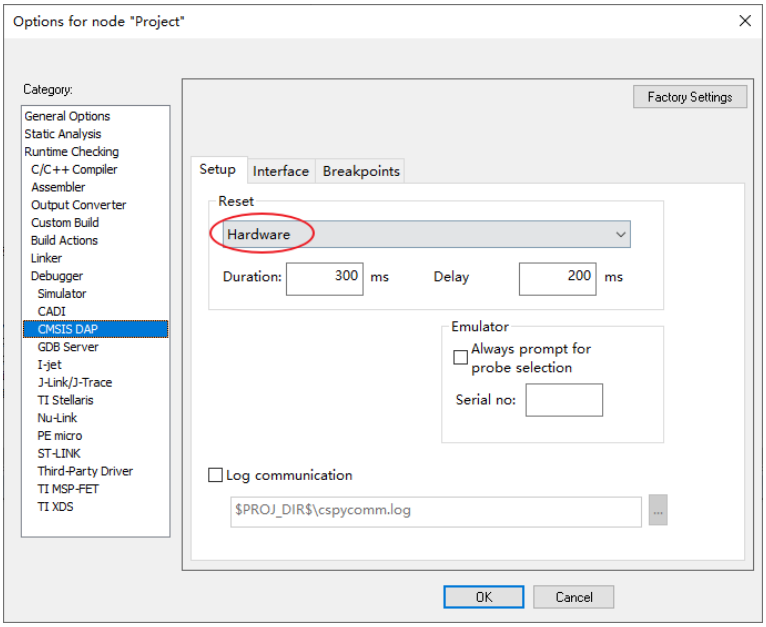
```
testValue_SDRAM address is 0xc0003000, value is 0x5
test_sdram address is 0xc0001000
```

Figure 3-2. Debugging results of loading functions and '.c' files to specific SDRAM

locations

Note: When using CMSIS-DAP for debugging, select the Reset configuration option as "Hardware"; otherwise, debugging will fail. See ***Figure 3-3. CMSIS-DAP reset option configuration***.

Figure 3-3. CMSIS-DAP reset option configuration



4. Result

Check the “GD32H7XX_ScatterLoading_v1.0.0\Project\IAR_project\GD32H7xx\List\Project.map” results as shown in *Figure 4-1. Project.map file compiled for scatter-loading project* is displayed.

Figure 4-1. Project.map file compiled for scatter-loading project

```
  Section                  Kind         Address    Size  Object
  -------                  ----         -------    ----  ------
"A0":                                              0x3a4
  .intvec                  ro code   0x800'0000   0x3a4  startup_gd32h7xx.o [1]
                                   - 0x800'03a4   0x3a4

"P1":                                             0x290c
  .text                    ro code   0x800'03a4   0xb6c  gd32h7xx_rcu.o [1]
  .text                    ro code   0x800'0f10    0xa6  ABImemcpy.o [4]
  .text                    ro code   0x800'0fb6    0x3a  zero_init3.o [4]
  .text                    ro code   0x800'0ff0   0x47c  exmc_sdram.o [1]
  .text                    ro code   0x800'146c   0x19c  gd32h7xx_exmc.o [1]
  .text                    ro code   0x800'1608   0x136  gd32h7xx_gpio.o [1]
  .text                    ro code   0x800'173e    0x2e  copy_init3.o [4]
  .text                    ro code   0x800'176c   0x148  gd32h7xx_misc.o [1]
  .text                    ro code   0x800'18b4   0x298  system_gd32h7xx.o [1]
  .text                    ro code   0x800'1b4c   0x220  gd32h7xx_usart.o [1]
  Veneer                   ro code   0x800'1d6c     0x8  - Linker created -
  Veneer                   ro code   0x800'1d74     0x8  - Linker created -
  Veneer                   ro code   0x800'1d7c     0x8  - Linker created -
  Veneer                   ro code   0x800'1d84     0x8  - Linker created -
  .text                    ro code   0x800'1d8c   0x1f8  main.o [1]
  .text                    ro code   0x800'1f84    0xb0  systick.o [1]
"P2":                                              0x14
  ROM_FUNC                 ro code   0x802'0000    0x14  main.o [1]
                                   - 0x802'0014    0x14

"P3":                                              0x32
  .text                    ro code   0x803'0000    0x32  gd32h7xx_it.o [1]
                                   - 0x803'0032    0x32

"P4":                                              0x2c
  FILE_init                          0x805'0000    0x18  <Block>
    Initializer bytes      const     0x805'0000    0x18  <for FILE-1>
  SDRAM_FILE_init                    0x805'0018    0x14  <Block>
    Initializer bytes      const     0x805'0018    0x14  <for SDRAM_FILE-1>
                                   - 0x805'002c    0x2c

Absolute sections, part 2 of 2:                    0x38
  .rodata                  const     0x806'0000    0x38  main.o [1]
                                   - 0x806'0038    0x38

"P5", part 1 of 3:                                 0x50
  P5-1                               0x2400'0000    0x50  <Init block>
    .data                  inited    0x2400'0000     0x4  system_gd32h7xx.o [1]
    .data                  inited    0x2400'0004    0x48  xfiles.o [2]
    DTCMRAM_VARIABLE       inited    0x2400'004c     0x4  main.o [1]
                                   - 0x2400'0050    0x50

"P5", part 2 of 3:                                 0x54
  .bss                     zero      0x2400'0050    0x50  xfiles.o [2]
  .bss                     zero      0x2400'00a0     0x4  systick.o [1]
                                   - 0x2400'00a4    0x54

"P5", part 3 of 3:                                0x1000
  CSTACK                             0x2400'00a8  0x1000  <Block>
    CSTACK                 uninit    0x2400'00a8  0x1000  <Block tail>
                                   - 0x2400'10a8  0x1000

"P6":                                              0x14
  .bss.RAM_Array           zero      0x2401'0000    0x14  main.o [1]
                                   - 0x2401'0014    0x14
```

```
"P7":                                      0x4
  P7-1                         0x2402'0000    0x4  <Init block>
    RAM_VARIABLE       inited  0x2402'0000    0x4  main.o [1]
                             - 0x2402'0004    0x4

"P8":                                     0x14
  P8-1                         0x2403'0000   0x14  <Init block>
    RAM_Array          inited  0x2403'0000   0x14  main.o [1]
                             - 0x2403'0014   0x14

"P9":                                     0x18
  P9-1                         0x2404'0000   0x18  <Init block>
    SRAM_FUCN          inited  0x2404'0000   0x18  main.o [1]
                             - 0x2404'0018   0x18

"P10":                                    0x18
  FILE                         0x2405'0000   0x18  <Block>
    FILE-1                     0x2405'0000   0x16  <Init block>
      Veneer           inited  0x2405'0000    0x8  - Linker created -
      .text            inited  0x2405'0008    0xe  hw_config.o [1]
                             - 0x2405'0018   0x18

"P11":                                    0x10
  P11-1                        0xc000'0000   0x10  <Init block>
    SDRAM_FUNC         inited  0xc000'0000   0x10  main.o [1]
                             - 0xc000'0010   0x10

"P12":                                    0x14
  SDRAM_ARRAY          zero    0xc000'1000   0x14  main.o [1]
                             - 0xc000'1014   0x14

"P13":                                    0x14
  SDRAM_FILE                   0xc000'2000   0x14  <Block>
    SDRAM_FILE-1               0xc000'2000   0x12  <Init block>
      Veneer           inited  0xc000'2000    0x8  - Linker created -
      .text            inited  0xc000'2008    0xa  test.o [1]
                             - 0xc000'2014   0x14

"P14":                                     0x4
  P14-1                        0xc000'3000    0x4  <Init block>
    SDRAM_VAR          inited  0xc000'3000    0x4  main.o [1]
                             - 0xc000'3004    0x4
```

From the map file, the load addresses and execution addresses of each section can be observed, matching the specified scatter-loading regions.

5. Revision History

Table 5-1. Revision History

Version number	Description	Date
1.0	Initial Release	May.30, 2025
1.1	Update AN name to “Instructions for scatter loading in IAR for GD32 Arm Cortex-M processors”	Feb.03, 2026

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 according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. 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.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively 'Unintended Uses'). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby 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 Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates 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 Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.