GigaDevice Semiconductor Inc.

Instructions for Scatter Loading in Keil MDK5 for GD32MCU

Application note
AN206

Revision 1.1

(Feb. 2026)

Table of Contents

List of Figures

List of Tables

1. Introduction to scatter loading in Keil

In the project generated with the default configuration in Keil, MDK automatically obtains information such as the FLASH and RAM sizes of the chip based on the chip model selected in the options, and generates a scatter loading file (*.sct, Linker Control File) named after the project. The linker uses the configuration of the scatter-loading file to determine the allocation addresses of various sections in memory. Therefore, we can modify this file to specify the placement of code sections at different memory locations.

This application note is based on the GD32H737_757 series, using the GD32H759i-EVAL development board, Keil version 5.30.0.0, and compiler version V6.14. It describes how to implement the following functionalities:

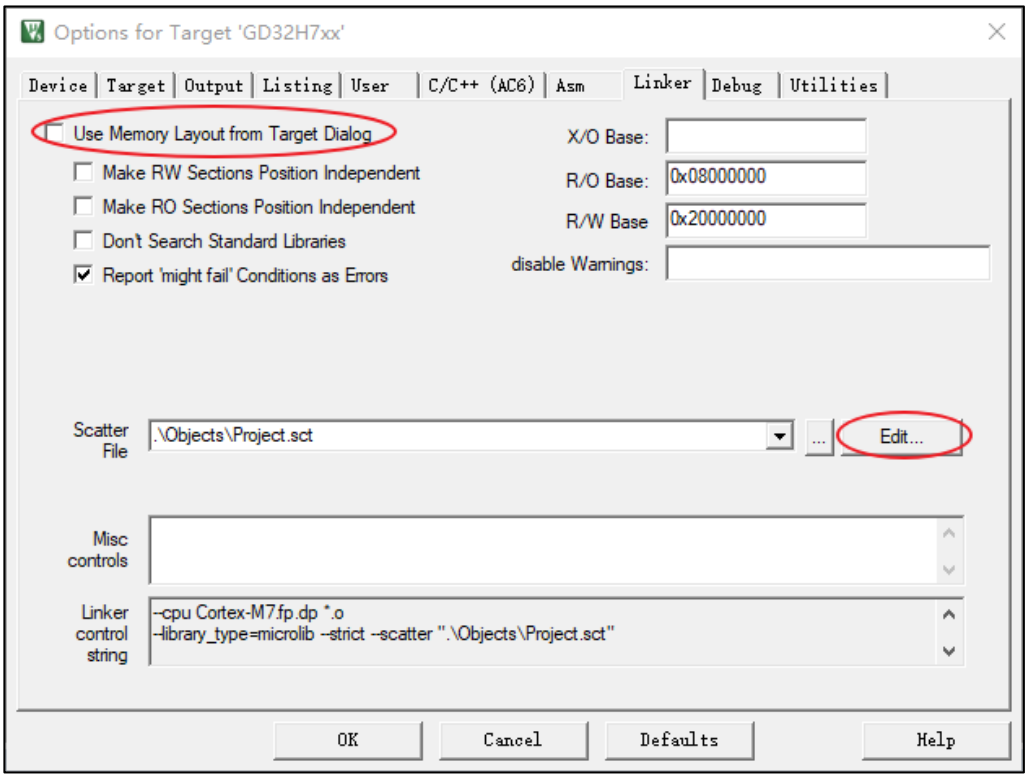
- Implement global variable loading to specified locations
- Implement function loading to specified locations
- Implement array loading to specified locations
- Implement .c file loading to the specified location
- Implement loading the above function to the specified location in SDRAM

2. Implementation of scatter-loading in Keil

2.1. Use manually written .sct file

This project directly uses a 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 project's .sct file. Relevant configuration is shown in *Figure 2-1. Use manually written .sct file*.

Figure 2-1. Use manually written .sct file



User can also open and edit the file in the project directory 'GD32H7xx_ScatterLoading_v1.0.0\Project\Keil_project\Objects\Project.sct'. The file opening code is shown in *Table 2-1. Project.sct code*.

Table 2-1. Project.sct code

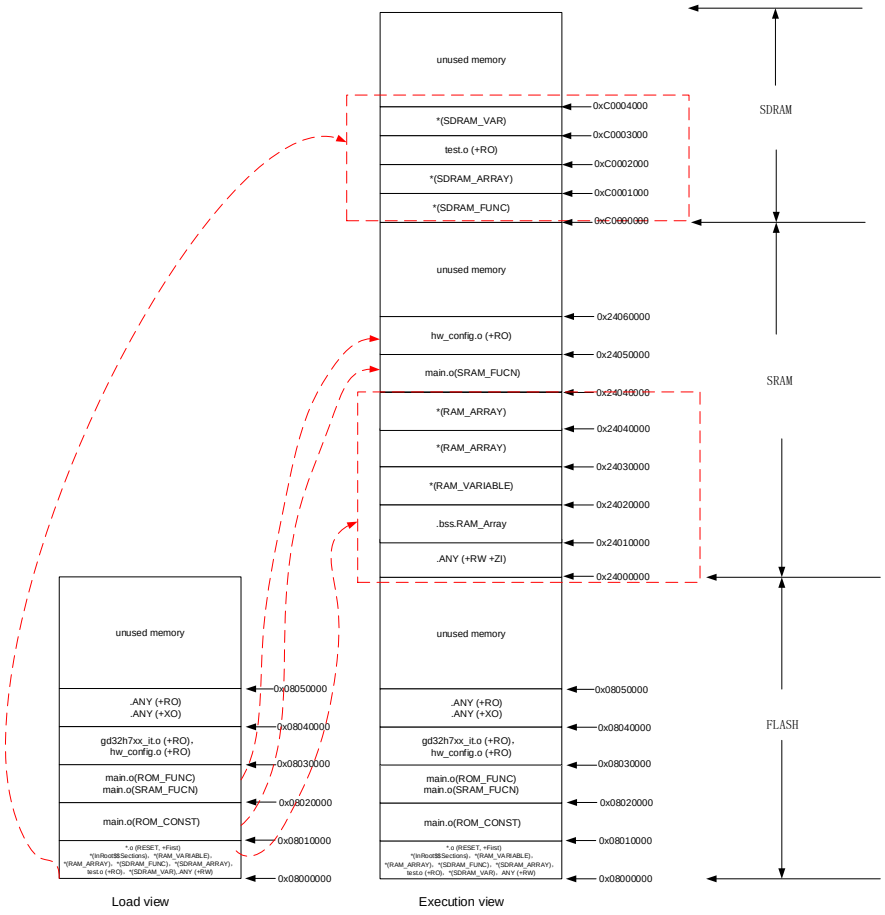
```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x08000000 0x00020000  {    ; load region size_region
ER_IROM1 0x08000000 0x00010000  {
*.o (RESET, +First)
*(InRoot$$Sections)
```

```
}
/****    constant scatter loading    ***/
ER_IROM_CONSTANT 0x08010000 0x00010000 {
main.o(ROM_CONST)
}
RW_IRAM1 0x24000000 0x00010000    {    ; RW data
.ANY (+RW +ZI)
}
RW_IRAM_Array 0x24010000 0x00010000 {
main.o(.bss.RAM_Array)
}
RW_IRAM_VAR 0x24020000 0x00010000 {
*(RAM_VARIABLE)
}
ER_IRAM_ARRAY 0x24030000 0x0010000 {
*(RAM_ARRAY)
}
RW_DTCMRAM_VAR 0x20000000 0x00010000 {
*(DTCMRAM_VARIABLE)
}
ER_ISDRAM_FUNC 0xC0000000 0x00001000 {
*(SDRAM_FUNC)
}
ER_ISDRAM_ARRAY 0xC0001000 UNINIT 0x00001000 {
*(SDRAM_ARRAY)
}
ER_ISDRAM_OBJ 0xC0002000 0x00001000 {
test.o (+RO)
}
ER_ISDRAM_VAR 0xC0003000 0x00001000 {
*(SDRAM_VAR)
}
}

/****    Function scatter loading    ***/
LR_IROM2 0x08020000 0x00010000 {
ER_IROM_FUNC 0x08020000 0x00010000 {
main.o(ROM_FUNC)
}
ER_IRAM_FUNC 0x24040000 0x0010000 {
main.o(SRAM_FUCN)
}
}
```

```
/****    File scatter loading    ***/
LR_IROM3 0x08030000 0x00010000 {
ER_IROM_Object    0x08030000 0x00010000 {
gd32h7xx_it.o (+RO)
}
RW_IRAM_Object 0x24050000 0x00010000 {
hw_config.o (+RO)
}
}

LR_IROM4 0x08040000 0x00010000 {
ER_IROM4 0x08040000 0x00010000 {
.ANY (+RO)
.ANY (+XO)
}
}
```

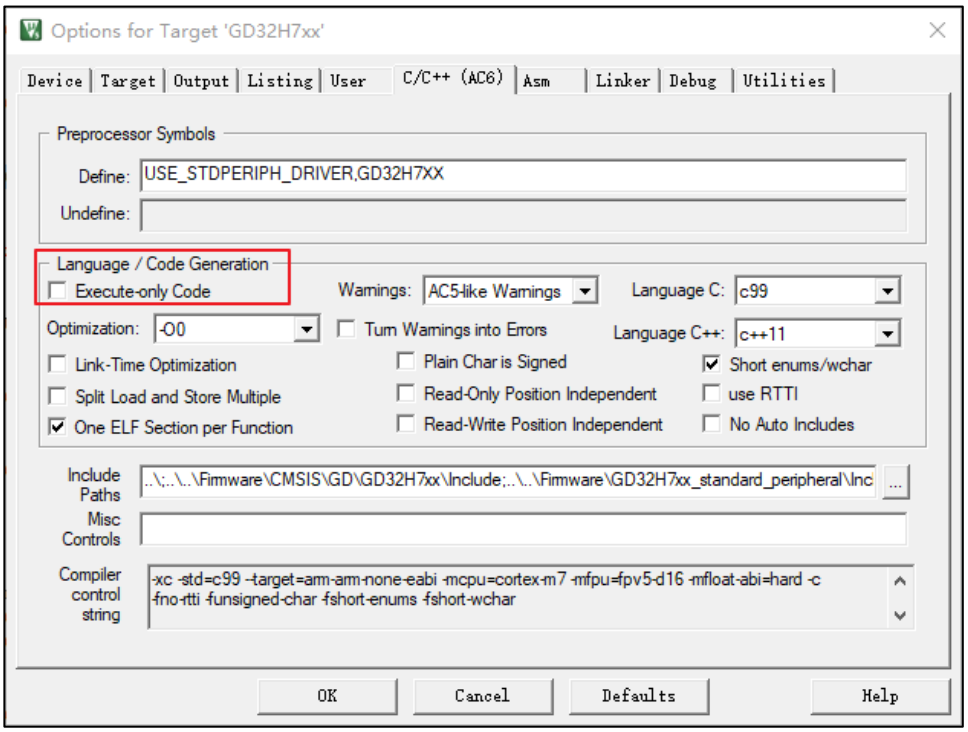
The red part is the main addition for implementing scatter loading. The memory distribution diagram is shown in ***Figure 2-2. Memory distribution diagram in the .sct file***. A detailed analysis is provided below.

Figure 2-2. Memory distribution diagram in the .sct file



Note: In Keil, the Execute-only Code compile option needs to be unchecked. See ***Figure 2-3. Execute-only Code compile option***.

Figure 2-3. Execute-only Code compile option



2.2. Load global variables to the specified location

Add the following code to the Project.sct file. The code is shown in ***Table 2-2. Code to load global variables to the specified location in Project.sct***.

Table 2-2. Code to load global variables to the specified location in Project.sct

```
/****   Variable scatter loading   ***/
RW_IRAM_VAR 0x24020000 0x00010000 {
*(RAM_VARIABLE)
}
```

The above code loads the specified RAM_VARIABLE section to the starting address 0x24020000. Define global variables in the main.c file as shown in ***Table 2-3. Code in main.c for loading global variables to the specified location***.

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

```
/* load the variable testValue_RAM to SRAM address 0x24020000 */
int testValue_RAM __attribute__((section('RAM_VARIABLE'))) = 0xCC;
```

Print variable addresses using the printf function, results are shown in ***Table 2-4. Printed results for loading global variables to the specified location***:

Table 2-4. Printed results for loading global variables to the specified location

```
testValue_RAM address is 0x24020000, value is 0xCC
```

2.3. Load the function to the specified location

Add the following code to the Project.sct file, as shown in ***Table 2-5. Code in Project.sct for loading the function to the specified location***:

Table 2-5. Code in Project.sct for loading the function to the specified location

```
/****    Function scatter loading    ***/
LR_IROM2 0x08020000 0x00010000 {
ER_IROM_FUNC 0x08020000 0x00010000 {
main.o(ROM_FUNC)
}
ER_IRAM_FUNC 0x24040000 0x0010000 {
main.o(SRAM_FUCN)
}
}
```

The above code loads the ROM_FUNC section and SRAM_FUNC section of the specified main.o module to the starting addresses 0x08020000 and 0x24040000, respectively. In the main.c file, the delay function and fill_TX_Data function are assigned to ROM_FUNC and SRAM_FUNC, respectively, as shown in ***Table 2-6. Code in main.c for loading functions to the specified location*** is as follows:

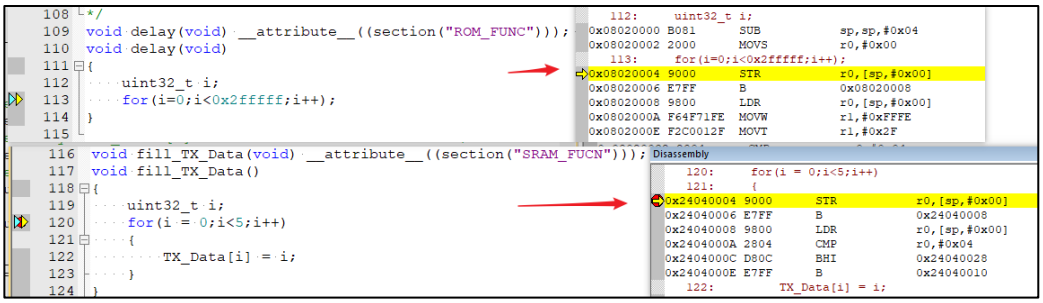
Table 2-6. Code in main.c for loading functions to the specified location

```
/* 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_FUNC')));
```

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

The debugging results of the program are shown in ***Figure 2-4. Debugging results for loading functions to the specified location*** are as follows:

Figure 2-4. Debugging results for loading functions to the specified location



2.4. Load the array to the specified location

2.4.1. Uninitialized array loaded to the specified location

Add the following code to the Project.sct file, as shown in ***Table 2-7. Code in Project.sct for loading uninitialized arrays to the specified location*** is as follows:

Table 2-7. Code in Project.sct for loading uninitialized arrays to the specified location

```
/*** Array scatter loading ***/
RW_IRAM_Array 0x24010000 0x00010000 {
main.o(.bss.RAM_Array)
}
```

The above code loads the .bss.RAM_Array section of the main.o module to the starting address 0x24010000. The array TX_Data[] is defined in main.c, as shown in ***Table 2-8. Code for loading an uninitialized array to a specified location in main.c*** as shown.

Table 2-8. Code for loading an uninitialized array to a specified location in main.c

```
/* load the array TX_Data[5] to SRAM address 0x24010000 */
uint32_t TX_Data[5] __attribute__((section('.bss.RAM_Array')));
```

2.4.2. Constant array loaded to a specified location

Add __attribute__((section('.ARM.__at_xxxxxxxx'))) after the array. In this example, the constant array `const char constdata[]` is defined in main.c, as shown in ***Table 2-9. Code for loading a constant array to a specified location in main.c*** is shown.

Table 2-9. Code for loading a constant array to a specified location in main.c

```
/* Load const array constdata to address 0x08050000 */
const char constdata[] __attribute__((section('.ARM.__at_0x08050000'))) = {
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,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
    };
```

2.4.3. Global initialized array loaded to a specified location

Add the following code to the Project.sct file, as shown in ***Table 2-10. Code in Project.sct for loading the global initialized array to a specified location***. The array `test_sram[]` is defined in main.c, as shown in ***Table 2-11. Code in main.c for loading the global initialized array to a specified location***.

Table 2-10. Code in Project.sct for loading the global initialized array to a specified location

```
/*** Array scatter loading ***/
ER_IRAM_ARRAY 0x24030000 0x0010000 {
*(RAM_ARRAY)
}
```

Table 2-11. Code in main.c for loading the global initialized array to a specified location

```
/* load the array test_sram[5] to SRAM address 0x24030000 */
uint32_t test_sram[5] __attribute__((section('RAM_ARRAY'))) = {1, 2, 3, 4, 5};
```

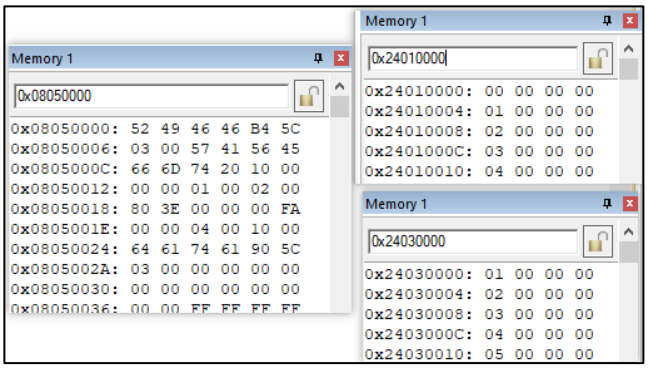
Using the printf function to print the array address, the results are shown in ***Table 2-12. Printed results of loading the array to a specified location***.

Table 2-12. Printed results of loading the array to a specified location

```
constdata address is 0x08050000
TX_Data address is 0x24010000
test_sram address is 0x24030000
```

Debugging results of the program are shown in ***Figure 2-5. Debugging results of loading the array to a specified location***.

Figure 2-5. Debugging results of loading the array to a specified location



2.5. Load the .c file to a specified location

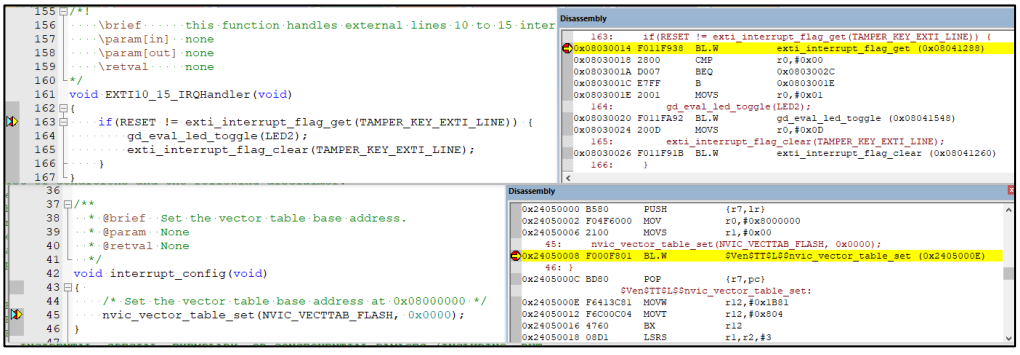
Add the following code to the Project.sct file, as shown in ***Table 2-13. Code for loading files to the specified location in Project.sct***.

Table 2-13. Code for loading files to the specified location in Project.sct

```
/****    File scatter loading    ***/
LR_IROM3 0x08030000 0x00010000 {
ER_IROM_Object    0x08030000 0x00010000 {
gd32h7xx_it.o (+RO)
}
RW_IRAM_Object 0x24050000 0x00010000 {
hw_config.o (+RO)
}
}
```

The above code loads the specified gd32h7xx_it.o file to the starting address 0x08030000 and the hw_config.o file to the starting address 0x24050000. The program debugging results are as follows:

Figure 2-6. Program debugging results for loading .c files to the specified location



Note: The GD32H7xx series MCU includes ITCM SRAM and DTCM SRAM. Based on the method provided above, you can modify the scatter-loading file according to project requirements to load variables, arrays, functions, and files to specified regions.

3. Implementation of SDRAM scatter loading

3.1. Implementation of SDRAM scatter loading

Add the following code in red to the Project.sct file, as shown in ***Table 3-1. SDRAM scatter-loading implementation code in Project.sct***:

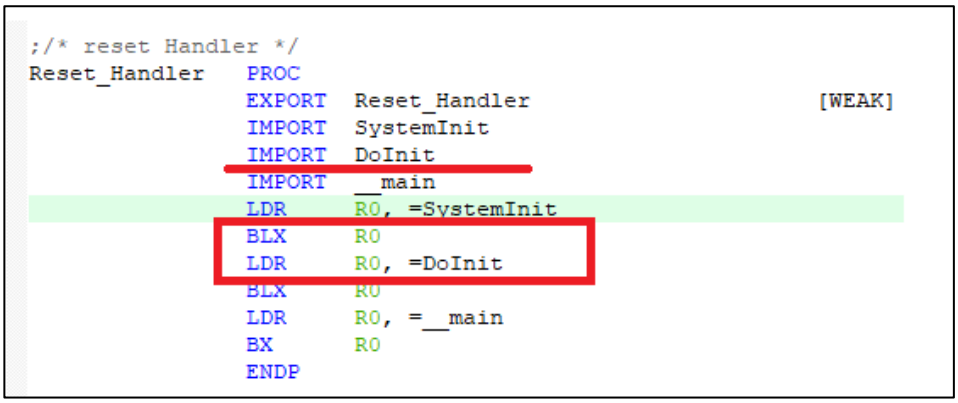
Table 3-1. SDRAM scatter-loading implementation code in Project.sct

```
LR_IROM1 0x08000000 0x00020000  {    ; load region size_region
ER_IROM1 0x08000000 0x00010000  {
*.o (RESET, +First)
*(InRoot$$Sections)
}
/****   constant scatter loading   ***/
ER_IROM_CONSTANT 0x08010000 0x00010000 {
main.o(ROM_CONST)
}
RW_IRAM1 0x24000000 0x00010000  {  ; RW data
.ANY (+RW +ZI)
}
RW_IRAM_Array 0x24010000 0x00010000 {
main.o(.bss.RAM_Array)
}
RW_IRAM_VAR 0x24020000 0x00010000 {
*(RAM_VARIABLE)
}
ER_IRAM_ARRAY 0x24030000 0x0010000 {
*(RAM_ARRAY)
}
ER_ISDRAM_FUNC 0xC0000000 0x00001000 {
*(SDRAM_FUNC)
}
ER_ISDRAM_ARRAY 0xC0001000 0x00001000 {
*(SDRAM_ARRAY)
}
ER_ISDRAM_OBJ 0xC0002000 0x00001000 {
test.o (+RO)
}
ER_ISDRAM_VAR 0xC0003000 0x00001000 {
*(SDRAM_VAR)
}
}
```

The above code loads the SDRAM_FUNC section, SDRAM_ARRAY section, and test.o file to the starting addresses 0xc0000000, 0xc0001000, and 0xc0002000, respectively.

Add the following code to `startup_gd32h7xx.s`, as shown in ***Figure 3-1. Code added in `startup_gd32h7xx.s`***:

Figure 3-1. Code added in `startup_gd32h7xx.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 `DoInit` function is defined in `main.c`, which primarily implements EXMC initialization and MPU configuration. The function code is shown in ***Table 3-2. `DoInit` function implementation code***.

Table 3-2. `DoInit` function implementation code

```
/*!
\brief        Initialize the SDRAM, set up the MPU
\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++);
}
}
```

Note: In the default configuration of the M7 core, certain address ranges 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 GD32H7xx EXMC is 0xC0000000 – 0xDFFFFFF, which falls within this restricted region. By configuring the MPU (Memory Protection Unit) registers, the 0xC0000000 address range can be made executable. The MPU configuration code is shown in ***Table 3-3. MPU configuration code***.

Table 3-3. 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);
}
```

Define the variable `uint32_t testValue_SDRAM`, the array `int test_sdram[5]`, the function `testFuncInSDRAM` in `main.c`, and include the file `test.c`. The main code is shown in ***Table 3-4. Code for distributing variables, arrays, functions, and files to specified locations in SDRAM***

Table 3-4. Code for distributing variables, arrays, functions, and files to specified

locations in SDRAM

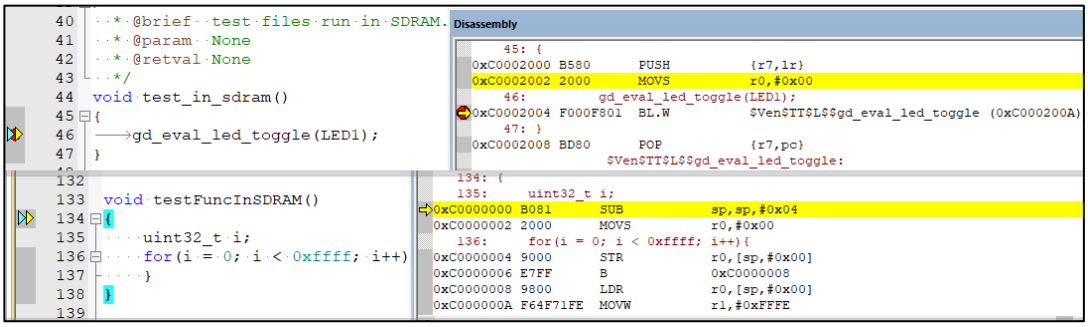
```
/* Load the variable testValue_RAM to SDRAM address 0xC0003000 */
uint32_t testValue_SDRAM __attribute__((section('SDRAM_VAR'))) = 5;
/* Load the array test_sdram[5] to SDRAM address 0xC0001000 */
uint32_t test_sdram[5] __attribute__((section('SDRAM_ARRAY'))) = {0};
/* Load the function testFuncInSDRAM to SDRAM address 0xC0000000 */
void testFuncInSDRAM(void) __attribute__((section('SDRAM_FUNC')));
/* test.c */
void test_in_sdram()
{
gd_eval_led_on(LED1);
}
```

The program execution and debugging results are shown in ***Table 3-5. Print results of loading variables and arrays to specified locations in SDRAM*** are shown, along with ***Figure 3-2. Debugging results of loading functions and `.c` file to specified locations in SDRAM*** are as shown.

Table 3-5. Print results of loading variables and arrays to specified locations in SDRAM

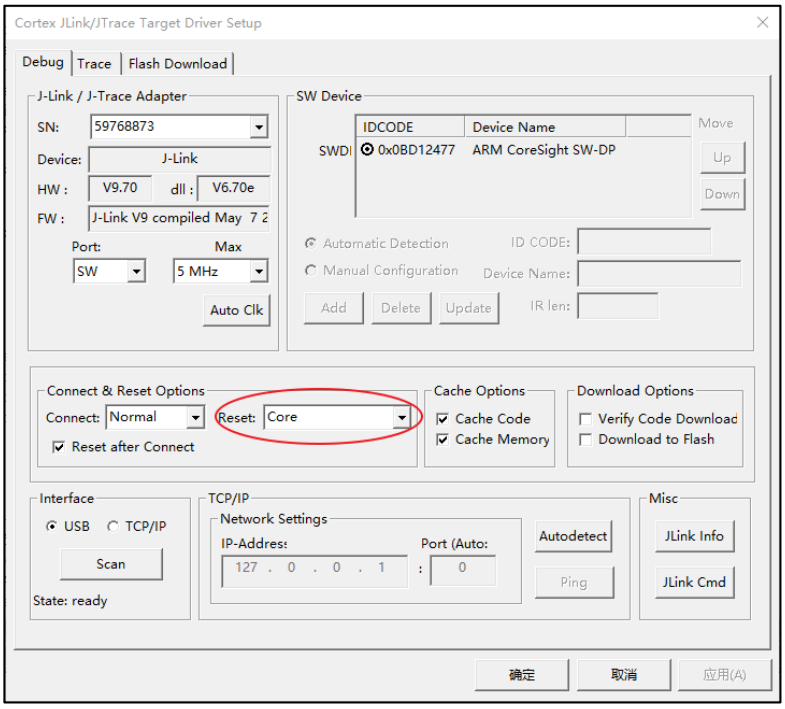
```
testValue_SDRAM is 0xC0003000, value is 0x5
test_sdram is 0xc0001000
```

Figure 3-2. Debugging results of loading functions and `.c` file to specified locations in SDRAM



Note: When using J-Link for debugging, use the Reset configuration option and select CORE to avoid SDRAM being reset when the MCU enters debugging, which may cause debugging failure. As shown in ***Figure 3-3. J-Link Reset option configuration***.

Figure 3-3. J-Link Reset option configuration



4. Result

User can check the `GD32H7XX_ScatterLoading_v1.0.0\Project\Keil\MDK-ARM\Listings\Project.map` file to further analyze the memory distribution. Open it as shown in ***Figure 4-1. Scatter Loading Project Compilation `Project.map` File***.

Figure 4-1. Scatter Loading Project Compilation `Project.map` File

```
==============================================================================

Memory Map of the image

  Image Entry point : 0x080003a5

  Load Region LR_IROM1 (Base: 0x08000000, Size: 0x00000530, Max: 0x00020000, ABSOLUTE)

    Execution Region ER_IROM1 (Exec base: 0x08000000, Load base: 0x08000000, Size: 0x000004bc, Max: 0x00010000, ABSOLUTE)

    Exec Addr    Load Addr    Size         Type   Attr      Idx    E Section Name        Object

    0x08000000   0x08000000   0x000003a4   Data   RO         3680    RESET               startup_gd32h7xx.o
    0x080003a4   0x080003a4   0x00000000   Code   RO         3721  * .ARM.Collect$$$$00000000  mc_w.l(entry.o)
    0x080003a4   0x080003a4   0x00000004   Code   RO         3756    .ARM.Collect$$$$00000001  mc_w.l(entry2.o)
    0x080003a8   0x080003a8   0x00000004   Code   RO         3759    .ARM.Collect$$$$00000004  mc_w.l(entry5.o)
    0x080003ac   0x080003ac   0x00000000   Code   RO         3761    .ARM.Collect$$$$00000008  mc_w.l(entry7b.o)
    0x080003ac   0x080003ac   0x00000000   Code   RO         3763    .ARM.Collect$$$$0000000A  mc_w.l(entry8b.o)
    0x080003ac   0x080003ac   0x00000008   Code   RO         3764    .ARM.Collect$$$$0000000B  mc_w.l(entry9a.o)
    0x080003b4   0x080003b4   0x00000000   Code   RO         3766    .ARM.Collect$$$$0000000D  mc_w.l(entry10a.o)
    0x080003b4   0x080003b4   0x00000000   Code   RO         3768    .ARM.Collect$$$$0000000F  mc_w.l(entry11a.o)
    0x080003b4   0x080003b4   0x00000004   Code   RO         3757    .ARM.Collect$$$$00002712  mc_w.l(entry2.o)
    0x080003b8   0x080003b8   0x00000024   Code   RO         3786    .text               mc_w.l(init.o)
    0x080003dc   0x080003dc   0x0000000e   Code   RO         3798    i.__scatterload_copy  mc_w.l(handlers.o)
    0x080003ea   0x080003ea   0x00000002   Code   RO         3799    i.__scatterload_null  mc_w.l(handlers.o)
    0x080003ec   0x080003ec   0x0000000e   Code   RO         3800    i.__scatterload_zeroinit  mc_w.l(handlers.o)
    0x080003fa   0x080003fa   0x00000002   PAD
    0x080003fc   0x080003fc   0x000000c0   Data   RO         3797    Region$$Table       anon$$obj.o


    Execution Region ER_IROM_CONSTANT (Exec base: 0x08010000, Load base: 0x080004bc, Size: 0x00000004, Max: 0x00010000, ABSOLUTE)

    Exec Addr    Load Addr    Size         Type   Attr      Idx    E Section Name        Object

    0x08010000   0x080004bc   0x00000004   Data   RO           18    ROM_CONST           main.o


    Execution Region RW_IRAM1 (Exec base: 0x24000000, Load base: 0x080004c0, Size: 0x00001010, Max: 0x00010000, ABSOLUTE)

    Exec Addr    Load Addr    Size         Type   Attr      Idx    E Section Name        Object

    0x24000000   0x080004c0   0x00000004   Data   RW         3770    .data               mc_w.l(stdout.o)
    0x24000004   0x080004c4   0x00000004   Data   RW          130    .data.SystemCoreClock  system_gd32h7xx.o
    0x24000008   -            0x00000004   Zero   RW           69    .bss.delay          systick.o
    0x2400000c   0x080004c8   0x00000004   PAD
    0x24000010   -            0x00001000   Zero   RW         3678    STACK               startup_gd32h7xx.o


    Execution Region RW_IRAM_Array (Exec base: 0x24010000, Load base: 0x080004c8, Size: 0x00000014, Max: 0x00010000, ABSOLUTE)

    Exec Addr    Load Addr    Size         Type   Attr      Idx    E Section Name        Object

    0x24010000   -            0x00000014   Zero   RW           19    .bss.RAM_Array      main.o
```

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

5. Revision History

Table 5-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	May.30, 2025
1.1	Update AN name to “Instructions for scatter loading in Keil MDK5 for GD32MCU”.	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.