

**GigaDevice Semiconductor Inc.**

**Arm<sup>®</sup> Cortex<sup>®</sup>-M3/4 32-bit MCU**

**Application Note**

**AN032**

## 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 GD32F450xK.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:

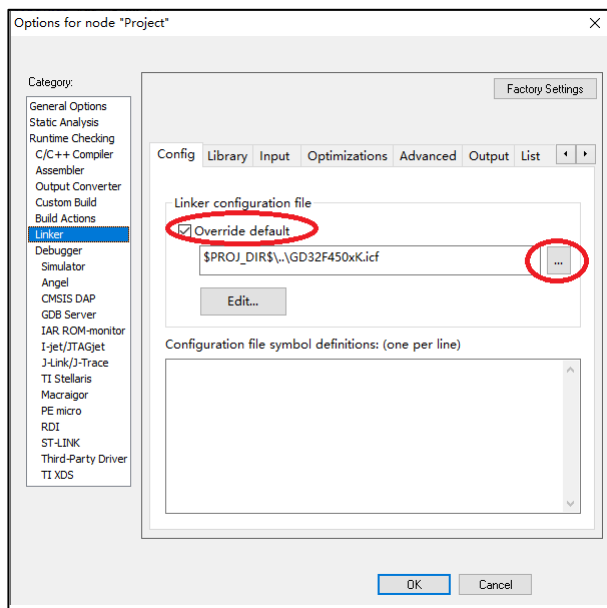
- Load global variables to the specified location
- Load function to the specified location
- Load array to the specified position
- Load .c file to the specified location
- The above function is loaded to the designated location of SDRAM.

## 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**

```

/*###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;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __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-*/

```

```

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**

```

define symbol __ICFEDIT_region_RAM2_start__ = 0x20001200;
define symbol __ICFEDIT_region_RAM2_end__ = 0x200012ff;
define region RAM2_region = mem:[from __ICFEDIT_region_RAM2_start__ to
__ICFEDIT_region_RAM2_end__];
place in RAM2_region { section variable};

```

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**

```
/* 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**

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

## 2.3. Load the function to the specified location

Add the following code to the GD32F450.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**

```
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**

```
/* 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<0x2ffff;i++);
}
```

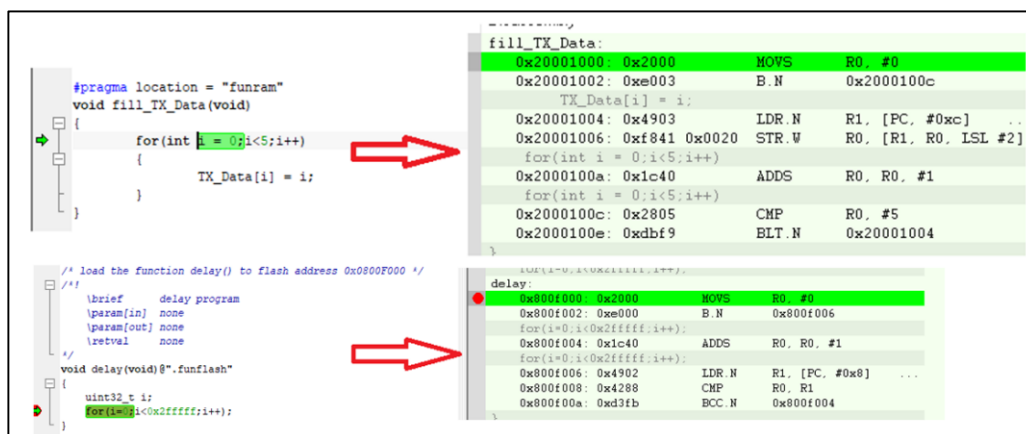
```

/* 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**

```

define symbol __ICFEDIT_region_RAM3_start__ = 0x20001300;
define symbol __ICFEDIT_region_RAM3_end__   = 0x200013FF;
define region RAM3_region = mem:[from __ICFEDIT_region_RAM3_start__ to

```

```
__ICFEDIT_region_RAM3_end__];
place in RAM3_region { section array};
```

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 0x08003000 */
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.](#)

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

Memory 1		Memory 2	
Go to	Memory	Go to	Memory
0x08002ffc	ab fc 90 fe	0x20001300	00 00 00 00
0x08003000	52 49 46 46	0x20001304	00 00 00 00
0x08003004	b4 5c 03 00	0x20001308	00 00 00 00
0x08003008	57 41 56 45		
0x0800300c	66 6d 74 20		
0x08003010	10 00 00 00		
0x08003014	01 00 02 00		
0x08003018	80 3e 00 00		
0x0800301c	00 fa 00 00		
0x08003020	04 00 10 00		
0x08003024	64 61 74 61		
0x08003028	90 5c 03 00		
0x0800302c	00 00 00 00		
0x08003030	00 00 00 00		
0x08003034	00 00 00 00		
0x08003038	00 00 00 00		
0x0800303c	00 00 00 00		
0x08003040	00 00 00 00		
0x08003044	00 00 00 00		
0x08003048	00 00 00 00		
0x0800304c	00 00 00 00		
0x08003050	00 00 00 00		
0x08003054	00 00 00 00		
0x08003058	00 00 00 00		
0x0800305c	00 00 00 00		
0x08003060	00 00 00 00		
0x08003064	00 00 00 00		

## 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**

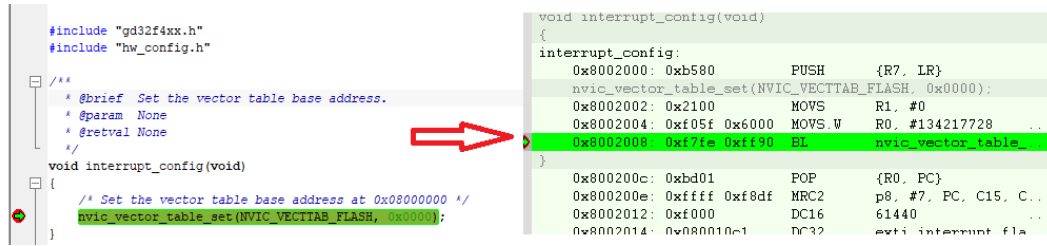
```

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](#).

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



```

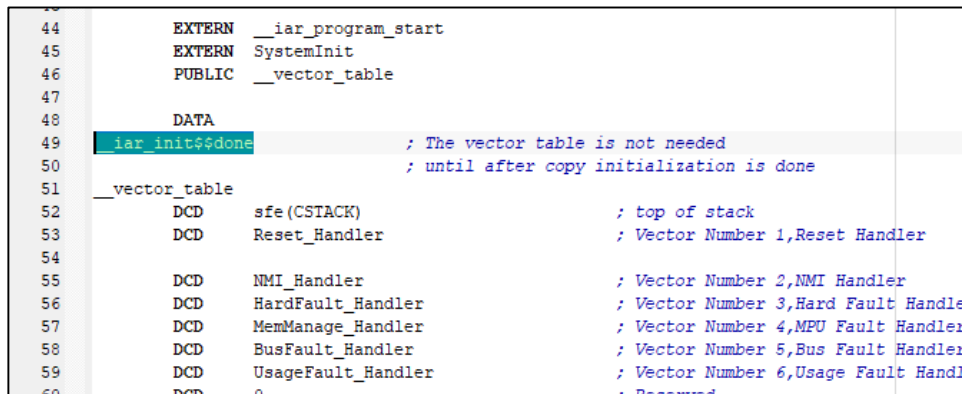
#include "gd32f4xx.h"
#include "hw_config.h"

/**
 * @brief Set the vector table base address.
 * @param None
 * @retval None
 */
void interrupt_config(void)
{
    /* Set the vector table base address at 0x08000000 */
    nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x0000);
}

void interrupt_config(void)
{
    interrupt_config:
    0x8002000: 0xb580      PUSH    {R7, LR}
    nvic_vector_table_set(NVIC_VECTTAB_FLASH, 0x0000);
    0x8002002: 0x2100      MOVS   R1, #0
    0x8002004: 0xf05f 0x6000 MOVS.W R0, #134217728
    0x8002008: 0xf7fe 0xff90 EL    nvic_vector_table...
}
    0x800200c: 0xbd01      POP    {R0, PC}
    0x800200e: 0xffff 0xf8df MRC2   p8, #7, PC, C15, C...
    0x8002012: 0xf000      DC16  61440
    0x8002014: 0x080010c1 DC32
    exti_interrupt fla
    
```

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.o file by initialize by copy above, and \_\_iar\_init needs to be added to the startup code startup\_gd32f4xx.s \$\$ done, as shown in [Figure 2-5. Add the startup\\_gd32f4xx.s file to \\_\\_iar\\_init \\$\\$ done](#).

**Figure 2-5. Add the startup\_gd32f4xx.s file to \_\_iar\_init \$\$ done**

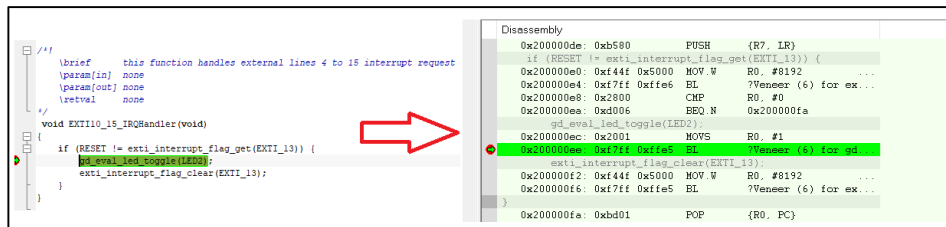


```

44     EXTERN __iar_program_start
45     EXTERN SystemInit
46     PUBLIC __vector_table
47
48     DATA
49     iar_init$$done          ; The vector table is not needed
50                             ; until after copy initialization is done
51     __vector_table
52     DCD     sfe (CSTACK)          ; top of stack
53     DCD     Reset_Handler        ; Vector Number 1,Reset Handler
54
55     DCD     NMI_Handler          ; Vector Number 2,NMI Handler
56     DCD     HardFault_Handler    ; Vector Number 3,Hard Fault Handle
57     DCD     MemManage_Handler    ; Vector Number 4,MPU Fault Handler
58     DCD     BusFault_Handler     ; Vector Number 5,Bus Fault Handler
59     DCD     UsageFault_Handler   ; Vector Number 6,Usage Fault Handle
60     DCD     0
    
```

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

**Figure 2-6. Debugging result of the .c file load to the SRAM**



```

/**
 * @brief this function handles external lines 4 to 15 interrupt request
 * @param[in] none
 * @param[out] none
 * @retval none
 */
void EXTI10_15_IRQHandler(void)
{
    if (RESET != exti_interrupt_flag_get(EXTI_13)) {
        exti_interrupt_flag_clear(EXTI_13);
    }
}

Disassembly
0x200000de: 0xb580      PUSH    {R7, LR}
if (RESET != exti_interrupt_flag_get(EXTI_13)) {
0x200000e0: 0xf44f 0x5000 MOVS.W R0, #8192
0x200000e4: 0xf7ff 0xffe6 EL    ?Veneer (6) for ex...
0x200000e8: 0x2800      CMP    R0, #0
0x200000ea: 0xd006      BEQ.N 0x200000fa
    gd_eval_led_toggle(LED2);
0x200000ec: 0x2001      MOVS   R0, #1
0x200000e8: 0xf7ff 0xffe5 EL    ?Veneer (6) for gd...
    exti_interrupt_flag_clear(EXTI_13);
0x200000f2: 0xf44f 0x5000 MOVS.W R0, #8192
0x200000f6: 0xf7ff 0xffe5 EL    ?Veneer (6) for ex...
}
0x200000fa: 0xbd01      POP    {R0, PC}
    
```

**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 0xC0000000-0xDFFFFFFF 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**

```

EXTERN __iar_program_start
EXTERN SystemInit
EXTERN DoInit
PUBLIC __vector_table

//
THUMB

PUBWEAK Reset_Handler
SECTION .text:CODE:NOROOT:REORDER(2)
Reset_Handler
LDR    R0, =SystemInit
BLX   R0
LDR    R0, =DoInit
BLX   R0
LDR    R0, =__iar_program_start
BX    R0

```

The DoInit 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**

```

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 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__];
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 sdrum_array};

```

The above code loads the sdrum\_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\_sdrum [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
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 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 sdrum, setup the MPU
\param[in] none
\param[out] none

```

```

    \retval    none
*/
void DoInit(void)
{
    /* sdram peripheral initialize */
    exmc_synchronous_dynamic_ram_init(EXMC_SDRAM_DEVICE0);
    /* Configures the MPU regions */
    mpu_setup();
}

```

**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");
    }
}
test.c:
#include "gd32f4xx.h"
#include "test.h"
#include "gd32f450i_eval.h"
/**
 * @brief  test files run in SDRAM.
 * @param  None
 * @retval None
 */
void test_in_sdram()
{

```



```
gd_eval_led_on(LED3);
}
```

[Table 3-4. Load variables and arrays to the specified location of SDRAM and the result](#) and [Figure 3-2. Add code to startup\\_gd32f450.s](#) show the results of program operation and debugging:

**Table 3-4. Load variables and arrays to the specified location of SDRAM and the result**  
 variable testValue\_SDRAM address is 0xc0003000  
 test\_sDRAM address is 0xc0002000

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



The screenshot displays the IAR IDE interface with source code on the left and assembly code on the right. Red arrows indicate the mapping between source code and assembly instructions.

**Source Code (Left):**

```

/**
 * @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");
  }
}

#include "gd32f4xx.h"
#include "test.h"
#include "gd32f450i_eval.h"
/**
 * @brief test files run in SDRAM.
 * @param None
 * @retval None
 */
void test_in_sdram()
{
  gd_eval_led_on(LED3);
}
  
```

**Assembly Code (Right):**

```

for(i=0; i<1000; i++)
testFuncInSDRAM:
0xc0001048: 0x2000      MOVS    R0, #0
0xc000104a: 0xe001      B.N     0xc0001050
asm("nop");
0xc000104c: 0xbf00      NOP
for(i=0; i<1000; i++)
0xc000104e: 0x1c40      ADDS   R0, R0, #1
for(i=0; i<1000; i++)
0xc0001050: 0xf5b0 0x7f7a  CMP.W  R0, #1000
0xc0001054: 0xd3fa      BCC.N  0xc000104c
}

{
test_in_sdram:
MYBLOCK$$Base:
0xc0004000: 0xb580      PUSH   {R7, LR}
gd_eval_led_on(LED3);
0xc0004002: 0x2002      MOVS   R0, #2
0xc0004004: 0xf000 0xf802  BL     ?Veneer (6) for gd
}
0xc0004008: 0xbd01      POP    {R0, PC}
0xc000400a: 0x0000      MOVS   R0, R0
?Veneer (6) for gd_eval_led_on:
0xc000400c: 0xf8df 0xf000  LDR.W  PC, [PC, #0x0]
0xc0004010: 0x08000cf1  DC32   gd_eval_led_on
MYBLOCK$$Limit:
  
```

Red arrows point from the source code to the assembly code: one from the `for(i=0; i<1000; i++)` loop in `testFuncInSDRAM` to the corresponding assembly instructions, and another from the `gd_eval_led_on(LED3);` call in `test_in_sdram` to its assembly instruction.

## 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**

1	Section	Kind	Address	Size	Object
2	"A3":			0xe	
3	.text	ro code	0x08002000	0xe	hw_config.o [1]
4			- 0x0800200e	0xe	
5	"P4":			0x48	
6	Initializer bytes	const	0x08002010	0x48	<for P7 s4>
7			- 0x08002058	0x48	
8	Absolute sections, part 1 of 4:			0x84f0	
9	.rodata	const	0x08003000	0x84f0	const-data.o [1]
10			- 0x0800b4f0	0x84f0	
11	"A2":			0x14	
12	.funflash	ro code	0x0800f000	0x14	main.o [1]
13			- 0x0800f014	0x14	
14	"A4":			0x14	
15	MYBLOCK_init		0x08010000	0x14	<Block>
16	Initializer bytes	const	0x08010000	0x14	<for MYBLOCK-1>
17			- 0x08010014	0x14	
18	Absolute sections, part 2 of 4:			0x4	
19	Absolute sections 2-1			0x4	<Init block>
20	.data	inited	0x08080000	0x4	main.o [1]
21			- 0x08080004	0x4	
22					
23	"P3":			0x18	
24	P3 s2		0x20001000	0x18	<Init block>
25	funram	inited	0x20001000	0x18	main.o [1]
26			- 0x20001018	0x18	
27	"P5":			0x4	
28	P5 s3		0x20001200	0x4	<Init block>
29	variable	inited	0x20001200	0x4	main.o [1]
30			- 0x20001204	0x4	
31	"P6":			0x14	
32	array	zero	0x20001300	0x14	main.o [1]
33			- 0x20001314	0x14	
34	Absolute sections, part 3 of 4:			0x14	
35	Absolute sections 3-1			0x14	<Init block>
36	.data	inited	0x20007000	0x14	main.o [1]
37			- 0x20007014	0x14	
38	"P7":			0x58	
39	P7 s4		0xc0001000	0x48	<Init block>
40	Veneer	inited	0xc0001000	0x8	- Linker created -
41	Veneer	inited	0xc0001008	0x8	- Linker created -
42	Veneer	inited	0xc0001010	0x8	- Linker created -
43	.text	inited	0xc0001018	0x30	gd32f4xx_it.o [1]
44	P7 s1		0xc0001048	0x10	<Init block>
45	.textrw	inited	0xc0001048	0x10	main.o [1]
46			- 0xc0001058	0x58	
47	"P8":			0x14	
48	s dram_array	zero	0xc0002000	0x14	main.o [1]
49			- 0xc0002014	0x14	
50	Absolute sections, part 4 of 4:			0x4	
51	.bss	zero	0xc0003000	0x4	main.o [1]
52			- 0xc0003004	0x4	
53	"A5":			0x14	
54	MYBLOCK		0xc0004000	0x14	<Block>
55	MYBLOCK-1		0xc0004000	0x14	<Init block>
56	.text	inited	0xc0004000	0xa	test.o [1]
57	Veneer	inited	0xc000400c	0x8	- Linker created -
58			- 0xc0004014	0x14	
59	*****				

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

## 5. Revision history

Table 5-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Apr.30, 2021

## Important Notice

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

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