GigaDevice Semiconductor Inc.

适用于 Arm Cortex-M 处理器的 GD32 Embedded Builder 分散加载说明

应用笔记

AN311

1.0 版本

（2025 年 12 月）

目录

图索引

表索引

1. 简介

ld 是 GNU binutils 工具集中的一个，是众多 Linkers（链接器）的一种。ld 把各种目标文件和库文件链接起来，并重定向它们的数据，完成符号解析。Linking 其实主要就是完成四个方面的工作：storage allocation、symbol management、libraries、relocation。因此根据工程代码的需要，我们可以通过修改该文件来实现指定代码段区的位置存储。

本应用笔记基于 GD32H75x 系列，采用 GD32H759I-EVAL 开发板，Embedded Builder 版本为 1.5.18，实现的功能主要如下：

- 实现用户定义的全局变量和数组加载到指定位置
- 实现用户定义的函数加载到指定位置
- 实现用户指定.c 文件加载到指定位置
- 实现上述功能加载到 SDRAM 指定位置

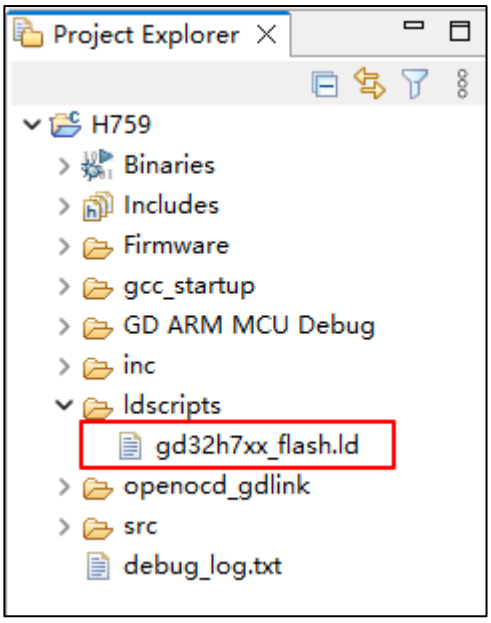
本手册同样适用于 GD32 Arm Cortex® M3/M4/M23/M33/M7 内核 MCU 产品。

2. 分散加载在 Embedded Builder 中的实现

2.1. 使用手动编写的 ld 文件

本工程直接使用手动编写的 ld 文件，在 Embedded Builder 中的“Window-> Show View-> Project Explore->ldscripts”中双击 gd32h7xx_flash.ld 文件，工程目录结构如*图 2-1 GD32 Embedded Builder 中工程目录结构*所示。同时也可到工程目录“..\ldscripts\gd32h7xx_flash.ld”下打开编辑。

图 2-1 GD32 Embedded Builder 中工程目录结构



2.2. VMA 和 LMA

在链接脚本中，VMA 和 LMA 是两个核心概念，用于控制程序段在内存中的分布。VMA (Virtual Memory Address) 即虚拟内存地址，是程序运行时段所在的内存地址，也称为运行时地址。LMA (Load Memory Address) 即加载内存地址，是程序烧录时段存储的物理地址，也称为存储地址。实现分散加载功能，需要理解 VMA 和 LMA 两个核心概念，关于 VMA 和 LMA 的具体介绍可参考*《AN200 GD32 Embedded builder 链接脚本文件说明》*。

2.3. 将全局变量和全局数组加载到指定位置

2.3.1. 自定义段

在链接脚本中添加用户自定义段，将不同类型的数据分配到指定地址，地址分配如*表 2-1. 将*

*全局变量和全局数组加载到指定位置地址分配总览*所示，链接文件代码如*表 2-2. gd32h7xx_flash.ld 中将全局变量和全局数组加载到指定位置*所示。

表 2-1. 将全局变量和全局数组加载到指定位置地址分配总览

段类型	section 名称	FLASH 地址	RAM 地址	用途
初始化变量	.data.user_vars	0x08008000	0x24008000	有初始值的变量
初始化数组	.data.user_arrays	0x08009000	0x24009000	有初始值的数组
未初始化变量	.bss.user_vars	-	0x2400A000	零初始化变量
未初始化数组	.bss.user_arrays	-	0x2400B000	零初始化数组
常量数组	.rodata.user_const_arrays	0x08007000	-	const 数组

表 2-2. gd32h7xx_flash.ld 中将全局变量和全局数组加载到指定位置

```
/* user initializes variable section */
.user_global_vars ALIGN(0x24008000, 4) : AT(ALIGN(0x08008000, 4))
{
    . = ALIGN(4);
    __user_global_vars_start__ = .;
    KEEP(*(.data.user_vars))
    . = ALIGN(4);
    __user_global_vars_end__ = .;
    __user_global_vars_size__ = __user_global_vars_end__ - __user_global_vars_start__;
}
__user_global_vars_loadaddr__ = LOADADDR(.user_global_vars);
/* user initializes array section */
.user_global_arrays ALIGN(0x24009000, 4) : AT(ALIGN(0x08009000, 4))
{
    . = ALIGN(4);
    __user_global_arrays_start__ = .;
    KEEP(*(.data.user_arrays))
    . = ALIGN(4);
    __user_global_arrays_end__ = .;
    __user_global_arrays_size__ = __user_global_arrays_end__ - __user_global_arrays_start__;
}
__user_global_arrays_loadaddr__ = LOADADDR(.user_global_arrays);
/* user uninitialized variable section */
.user_uninit_vars ALIGN(0x2400A000, 4) (NOLOAD) :
{
    . = ALIGN(4);
    __user_uninit_vars_start__ = .;
    KEEP(*(.bss.user_vars))
    . = ALIGN(4);
    __user_uninit_vars_end__ = .;
```

```
    __user_uninit_vars_size__ = __user_uninit_vars_end__ - __user_uninit_vars_start__;
}
/* user uninitialized array section */
.user_uninit_arrays ALIGN(0x2400B000, 4) (NOLOAD) :
{
    . = ALIGN(4);
    __user_uninit_arrays_start__ = .;
    KEEP(*(.bss.user_arrays))
    . = ALIGN(4);
    __user_uninit_arrays_end__ = .;
    __user_uninit_arrays_size__ = __user_uninit_arrays_end__ - __user_uninit_arrays_start__;
}
/* user constant array section */
.user_const_arrays ALIGN(0x08007000, 4) :
{
    . = ALIGN(4);
    __user_const_arrays_start__ = .;
    KEEP(*(.rodata.user_const_arrays))
    . = ALIGN(4);
    __user_const_arrays_end__ = .;
    __user_const_arrays_size__ = __user_const_arrays_end__ - __user_const_arrays_start__;
} > FLASH
```

2.3.2. C 代码中的变量定义

在头文件中定义 section 属性宏，提高代码可读性，如*表 2-3. 全局变量和全局数组加载到指定位置头文件宏定义*所示：

表 2-3. 全局变量和全局数组加载到指定位置头文件宏定义

```
/* initialized data section attributes - will be loaded from FLASH to RAM */
#define USER_INIT_VAR           __attribute__((section(".data.user_vars")))
#define USER_INIT_ARRAY         __attribute__((section(".data.user_arrays")))
/* uninitialized data section attributes - only in RAM, cleared at startup */
#define USER_UNINIT_VAR         __attribute__((section(".bss.user_vars")))
#define USER_UNINIT_ARRAY       __attribute__((section(".bss.user_arrays")))

#define USER_CONST_ARRAY        __attribute__((section(".rodata.user_const_arrays")))
```

变量定义示例如*表 2-4. 全局变量和全局数组加载到指定位置变量定义*所示。

表 2-4. 全局变量和全局数组加载到指定位置变量定义

```
/* user initialized global variables definition - will be placed in .data.user_vars section */
USER_INIT_VAR int32_t     system_status = 1;
```

```
USER_INIT_VAR uint32_t    device_serial = 0x12345678;
USER_INIT_VAR uint16_t    firmware_version = 0x0102;
USER_INIT_VAR uint8_t     system_config = 0xAB;

/* user initialized global arrays definition - will be placed in .data.user_arrays section */
USER_INIT_ARRAY uint16_t calibration_data[10] = {
    100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
};

USER_INIT_ARRAY uint8_t device_info[32] = {
    'G','D','3','2','H','7','x','x',' ','D','e','v','i','c','e',0
};

/* User uninitialized global variables definition - will be placed in .bss.user_vars section */
USER_UNINIT_VAR volatile uint32_t     tick_counter;
USER_UNINIT_VAR volatile uint16_t     error_code;
USER_UNINIT_VAR volatile uint8_t      system_state;
USER_UNINIT_VAR uint32_t                  runtime_config;

/* User uninitialized global arrays definition - will be placed in .bss.user_arrays section */
USER_UNINIT_ARRAY uint8_t     uart_tx_buffer[512];
USER_UNINIT_ARRAY uint8_t     uart_rx_buffer[512];
USER_UNINIT_ARRAY uint16_t    adc_samples[256];
USER_UNINIT_ARRAY char         log_buffer[1024];

/* Const array definition */
USER_CONST_ARRAY const uint16_t lookup_table[256] = {
    0x0000, 0x0001, 0x0004, 0x0009,
};
```

2.3.3. 自定义初始化实现

在系统启动时调用自定义段初始化函数，如*表 2-5. 全局变量和全局数组加载到指定位置数据段初始化实现*所示。

表 2-5. 全局变量和全局数组加载到指定位置数据段初始化实现

```
/* Data initialization function implementation */
void UserData_Init(void)
{
    uint32_t *src, *dst;
    uint32_t size;
    /* 1. Initialize user global variables segment - copy from FLASH to RAM */
```

```
    src = &__user_global_vars_loadaddr__;
    dst = &__user_global_vars_start__;
    size = ((uint32_t)&__user_global_vars_end__ - (uint32_t)&__user_global_vars_start__) / 4;

    for(uint32_t i = 0; i < size; i++) {
        *dst++ = *src++;
    }

    /* 2. Initialize user global arrays segment - copy from FLASH to RAM */
    src = &__user_global_arrays_loadaddr__;
    dst = &__user_global_arrays_start__;
    size = ((uint32_t)&__user_global_arrays_end__ - (uint32_t)&__user_global_arrays_start__) / 4;

    for(uint32_t i = 0; i < size; i++) {
        *dst++ = *src++;
    }
    /* 3. Clear user uninitialized variables segment */
    dst = &__user_uninit_vars_start__;
    size = ((uint32_t)&__user_uninit_vars_end__ - (uint32_t)&__user_uninit_vars_start__) / 4;

    for(uint32_t i = 0; i < size; i++) {
        *dst++ = 0;
    }
    /* 4. Clear user uninitialized arrays segment */
    dst = &__user_uninit_arrays_start__;
    size = ((uint32_t)&__user_uninit_arrays_end__ - (uint32_t)&__user_uninit_arrays_start__) / 4;

    for(uint32_t i = 0; i < size; i++) {
        *dst++ = 0;
    }
}
```

注意：

1. 初始化放在 cache 初始化函数之前。
2. 常量数组不需要初始化操作。

2.3.4. 测试验证

通过串口打印的方式，测试全局变量和全局数组加载到指定位置内存布局验证，代码如*表 2-6. 全局变量和全局数组加载到指定位置内存验证代码*所示。

表 2-6. 全局变量和全局数组加载到指定位置内存验证代码

```
void UserData_PrintMemoryInfo(void)
```

```
{
    printf("=== User-defined Memory Segments Information ===\r\n");

    printf("1. Initialized variables segment:\r\n");
    printf("    RAM address: 0x%08X - 0x%08X (%d bytes)\r\n",
           (uint32_t)&__user_global_vars_start__,
           (uint32_t)&__user_global_vars_end__,
           (uint32_t)&__user_global_vars_end__ - (uint32_t)&__user_global_vars_start__);
    printf("    FLASH source: 0x%08X\r\n", (uint32_t)&__user_global_vars_loadaddr__);

    printf("2. Initialized arrays segment:\r\n");
    printf("    RAM address: 0x%08X - 0x%08X (%d bytes)\r\n",
           (uint32_t)&__user_global_arrays_start__,
           (uint32_t)&__user_global_arrays_end__,
           (uint32_t)&__user_global_arrays_end__ - (uint32_t)&__user_global_arrays_start__);
    printf("    FLASH source: 0x%08X\r\n", (uint32_t)&__user_global_arrays_loadaddr__);

    printf("3. Uninitialized variables segment:\r\n");
    printf("    RAM address: 0x%08X - 0x%08X (%d bytes)\r\n",
           (uint32_t)&__user_uninit_vars_start__,
           (uint32_t)&__user_uninit_vars_end__,
           (uint32_t)&__user_uninit_vars_end__ - (uint32_t)&__user_uninit_vars_start__);

    printf("4. Uninitialized arrays segment:\r\n");
    printf("    RAM address: 0x%08X - 0x%08X (%d bytes)\r\n",
           (uint32_t)&__user_uninit_arrays_start__,
           (uint32_t)&__user_uninit_arrays_end__,
           (uint32_t)&__user_uninit_arrays_end__ - (uint32_t)&__user_uninit_arrays_start__);
    printf("5. Constant arrays segment:\r\n");
    printf("    FLASH address: 0x%08X - 0x%08X (%d bytes)\r\n",
           (uint32_t)&__user_const_arrays_start__,
           (uint32_t)&__user_const_arrays_end__,
           (uint32_t)&__user_const_arrays_end__ - (uint32_t)&__user_const_arrays_start__);
    printf("============================================\r\n\r\n");

}

int main(void)
{
    gd_eval_com_init(EVAL_COM);
    /*system initialization */
    printf("GD32H7xx User-defined Data Segment Example\r\n");
```

```
    printf("========================================\r\n");

    /* initialize user data segment */
    UserData_Init();
    /* enable the CPU cache */
    cache_enable();
    mpu_config();
    /* print memory information */
    UserData_PrintMemoryInfo();

    /* verify initialization data */
    printf("=== Verify Initialization Data ===\r\n");
    printf("system_status = %d (addr: 0x%08X)\r\n", system_status, (uint32_t)&system_status);
    printf("device_serial = 0x%08X (addr: 0x%08X)\r\n", device_serial, (uint32_t)&device_serial);
    printf("calibration_data[0] = %d (addr: 0x%08X)\r\n", calibration_data[0],
(uint32_t)&calibration_data[0]);
    printf("calibration_data[9] = %d (addr: 0x%08X)\r\n", calibration_data[9],
(uint32_t)&calibration_data[9]);
    printf("calibration_data array base addr: 0x%08X\r\n", (uint32_t)calibration_data);
    printf("device_info = %s (addr: 0x%08X)\r\n", (char*)device_info, (uint32_t)device_info);
    /* verify const array */
    printf("lookup_table[0] = 0x%04X (addr: 0x%08X)\r\n", lookup_table[0],
(uint32_t)&lookup_table[0]);
    printf("lookup_table[255] = 0x%04X (addr: 0x%08X)\r\n", lookup_table[255],
(uint32_t)&lookup_table[255]);
    printf("lookup_table array base addr: 0x%08X\r\n", (uint32_t)lookup_table);

    /* test uninitialized data usage */
    printf("\r\n=== Test Uninitialized Data Usage ===\r\n");

    /* buffer filling test */
    for(int i = 0; i < 10; i++) {
        uart_tx_buffer[i] = i + 0x30;    // '0' to '9'
        adc_samples[i] = i * 100;
    }
    uart_tx_buffer[10] = '\0';

    printf("uart_tx_buffer = %s (addr: 0x%08X)\r\n", (char*)uart_tx_buffer, (uint32_t)uart_tx_buffer);
    printf("adc_samples[5] = %d (addr: 0x%08X)\r\n", adc_samples[5], (uint32_t)&adc_samples[5]);
    printf("adc_samples array base addr: 0x%08X\r\n", (uint32_t)adc_samples);

    printf("\r\n=== System Ready ===\r\n");
```

```
    printf("User-defined data segments test completed successfully!\r\n");

    /* main loop - simple endless loop */
    while(1) {
    }
}
```

测试打印日志如*表 2-7. 将全局变量和全局数组加载到指定位置日志*所示，测试结果与实际分配保持一致。

表 2-7. 将全局变量和全局数组加载到指定位置日志

```
GD32H7xx User-defined Data Segment Example
==========================================
Starting initialization of user-defined data segments...
Initializing user global variables segment...
  Address range: 0x24008000 - 0x2400800C
  Data size: 12 bytes
Initializing user global arrays segment...
  Address range: 0x24009000 - 0x24009034
  Data size: 52 bytes
Clearing user uninitialized variables segment...
  Address range: 0x2400A000 - 0x2400A00C
  Data size: 12 bytes
Clearing user uninitialized arrays segment...
  Address range: 0x2400B000 - 0x2400BA00
  Data size: 2560 bytes
User data segments initialization completed!

=== User-defined Memory Segments Information ===
1. Initialized variables segment:
    RAM address: 0x24008000 - 0x2400800C (12 bytes)
    FLASH source: 0x08008000
2. Initialized arrays segment:
    RAM address: 0x24009000 - 0x24009034 (52 bytes)
    FLASH source: 0x08009000
3. Uninitialized variables segment:
    RAM address: 0x2400A000 - 0x2400A00C (12 bytes)
4. Uninitialized arrays segment:
    RAM address: 0x2400B000 - 0x2400BA00 (2560 bytes)
5. Constant arrays segment:
    FLASH address: 0x0800A000 - 0x0800A200 (512 bytes)
==========================================
```

```
=== Verify Initialization Data ===
system_status = 1 (addr: 0x24008000)
device_serial = 0x12345678 (addr: 0x24008004)
calibration_data[0] = 100 (addr: 0x24009000)
calibration_data[9] = 1000 (addr: 0x24009012)
calibration_data array base addr: 0x24009000
device_info = GD32H7xx Device (addr: 0x24009014)
lookup_table[0] = 0xA55A (addr: 0x0800A000)
lookup_table[255] = 0x0000 (addr: 0x0800A0FE)
lookup_table array base addr: 0x0800A000

=== Test Uninitialized Data Usage ===
uart_tx_buffer = 0123456789 (addr: 0x2400B000)
adc_samples[5] = 500 (addr: 0x2400B40A)
adc_samples array base addr: 0x2400B400

=== System Ready ===
User-defined data segments test completed successfully!
```

2.4. 将函数加载到指定位置

2.4.1. 自定义段

在链接脚本中添加用户自定义段，将函数分配到指定地址，地址分配如*表 2-8. 将函数加载到指定位置地址分配总览*所示，链接文件代码如*表 2-9. gd32h7xx_flash.ld 中将全局变量和全局数组加载到指定位置*所示。

表 2-8. 将函数加载到指定位置地址分配总览

段类型	section 名称	FLASH 地址	RAM 地址	用途
FLASH 函数	.text.flash_functions	0x08010000	-	直接在 FLASH 中执行
RAM 函数	.text.ram_functions	自动分配	0x2400C000	复制到 RAM 中执行
ITCM 函数	.text.itcm_functions	自动分配	0x00001000	复制到 ITCMRAM 中执行
DTCM 函数	.text.dtcm_functions	自动分配	0x20001000	复制到 DTCMRAM 中执行

表 2-9. gd32h7xx_flash.ld 中将全局变量和全局数组加载到指定位置

```
/* User custom function scatter loading */
.user_flash_functions 0x08010000 :
{
    . = ALIGN(4);
    __user_flash_functions_start__ = .;
    KEEP(*(.text.flash_functions))
```

```
    . = ALIGN(4);
    __user_flash_functions_end__ = .;
    __user_flash_functions_size__ = __user_flash_functions_end__ -
__user_flash_functions_start__;
} > FLASH

.user_ram_functions 0x2400C000 :
{
    . = ALIGN(4);
    __user_ram_functions_start__ = .;
    KEEP(*(.text.ram_functions))
    . = ALIGN(4);
    __user_ram_functions_end__ = .;
    __user_ram_functions_size__ = __user_ram_functions_end__ -
__user_ram_functions_start__;
} > AXISRAM AT> FLASH
__user_ram_functions_loadaddr__ = LOADADDR(.user_ram_functions);

.user_itcm_functions 0x00001000 :
{
    . = ALIGN(4);
    __user_itcm_functions_start__ = .;
    KEEP(*(.text.itcm_functions))
    . = ALIGN(4);
    __user_itcm_functions_end__ = .;
    __user_itcm_functions_size__ = __user_itcm_functions_end__ -
__user_itcm_functions_start__;
} > ITCMRAM AT> FLASH
__user_itcm_functions_loadaddr__ = LOADADDR(.user_itcm_functions);

.user_dtcm_functions 0x20001000 :
{
    . = ALIGN(4);
    __user_dtcm_functions_start__ = .;
    KEEP(*(.text.dtcm_functions))
    . = ALIGN(4);
    __user_dtcm_functions_end__ = .;
    __user_dtcm_functions_size__ = __user_dtcm_functions_end__ -
__user_dtcm_functions_start__;
} > DTCMRAM AT> FLASH
__user_dtcm_functions_loadaddr__ = LOADADDR(.user_dtcm_functions);
```

2.4.2. C 代码中的函数定义

在头文件中定义 section 属性宏，提高代码可读性，如*表 2-10. 将函数加载到指定位置头文件宏定义*所示：

表 2-10. 将函数加载到指定位置头文件宏定义

```
/* Function section attributes */
#define FLASH_FUNCTION        __attribute__((section(".text.flash_functions")))
#define RAM_FUNCTION          __attribute__((section(".text.ram_functions")))
#define ITCM_FUNCTION         __attribute__((section(".text.itcm_functions")))
#define DTCM_FUNCTION         __attribute__((section(".text.dtcm_functions")))
```

变量定义示例如*表 2-11. 将函数加载到指定位置函数定义与实现*所示。

表 2-11. 将函数加载到指定位置函数定义与实现

```
FLASH_FUNCTION int FlashFunction_Calculate(int a, int b)
{
    printf("FlashFunction_Calculate: a=%d, b=%d\r\n", a, b);
    int result = 0;
    for(int i = 0; i < 50; i++) {
        result += (a * b) + (i % 10);
    }
    printf("FlashFunction_Calculate result: %d\r\n", result);
    return result;
}

FLASH_FUNCTION void FlashFunction_ProcessData(uint8_t* data, uint32_t len)
{
    printf("FlashFunction_ProcessData: processing %d bytes\r\n", len);
    for(uint32_t i = 0; i < len; i++) {
        data[i] = data[i] ^ 0x5A;
    }
    printf("Data processing completed\r\n");
}

RAM_FUNCTION int RamFunction_FastProcess(int data)
{
    //printf("RamFunction_FastProcess: input=0x%08X\r\n", data);
    volatile int temp = data;
    temp = (temp << 1) ^ 0xA5A5;
    temp = (temp >> 1) | 0x5A5A0000;
    //printf("RamFunction_FastProcess result: 0x%08X\r\n", temp);
    return temp;
```

```
}

RAM_FUNCTION void RamFunction_ProcessBuffer(uint16_t* buffer, uint32_t size)
{
    printf("RamFunction_ProcessBuffer: processing %d elements\r\n", size);
    for(uint32_t i = 0; i < size; i++) {
        buffer[i] = (buffer[i] << 1) | (buffer[i] >> 15);
    }
    printf("Buffer processing completed\r\n");
}

ITCM_FUNCTION int ItcmFunction_CriticalTask(int input)
{
    printf("ItcmFunction_CriticalTask: input=%d\r\n", input);
    register int result = input;
    result = (result * 3) + 7;
    result = result ^ 0x12345678;
    result = (result << 4) | (result >> 28);
    printf("ItcmFunction_CriticalTask result: %d\r\n", result);
    return result;
}

ITCM_FUNCTION uint32_t ItcmFunction_BitOperation(uint32_t value)
{
    printf("ItcmFunction_BitOperation: input=0x%08X\r\n", value);
    uint32_t result = value;
    result = ~result;
    result = (result & 0xAAAAAAAA) | ((result & 0x55555555) << 1);
    result = (result & 0xCCCCCCCC) | ((result & 0x33333333) << 2);
    printf("ItcmFunction_BitOperation result: 0x%08X\r\n", result);
    return result;
}

DTCM_FUNCTION int DtcmFunction_DataProcess(int value)
{
    printf("DtcmFunction_DataProcess: input=%d\r\n", value);
    static int buffer[32];
    for(int i = 0; i < 32; i++) {
        buffer[i] = value + (i * 10);
    }
    int sum = 0;
    for(int i = 0; i < 32; i++) {
```

```
        sum += buffer[i];
    }
    printf("DtcmFunction_DataProcess result: %d\r\n", sum);
    return sum;
}

DTCM_FUNCTION void DtcmFunction_ArrayOperation(uint32_t* array, uint32_t count)
{
    printf("DtcmFunction_ArrayOperation: processing %d elements\r\n", count);
    for(uint32_t i = 0; i < count; i++) {
        array[i] = array[i] * 2 + 1;
    }
    printf("Array operation completed\r\n");
}
```

2.4.3. 自定义初始化实现

在系统启动时调用自定义代码段初始化函数，如*表 2-12. 函数加载到指定位置初始化实现*所示。

表 2-12. 函数加载到指定位置初始化实现

```
void UserFunctions_Init(void)
{
    uint32_t *src, *dst;
    uint32_t size;

    /* Copy RAM functions */
    src = &__user_ram_functions_loadaddr__;
    dst = &__user_ram_functions_start__;
    size = ((uint32_t)&__user_ram_functions_end__ - (uint32_t)&__user_ram_functions_start__) /
4;

    for(uint32_t i = 0; i < size; i++) {
        *dst++ = *src++;
    }
    /* Copy ITCM functions */
    src = &__user_itcm_functions_loadaddr__;
    dst = &__user_itcm_functions_start__;
    size = ((uint32_t)&__user_itcm_functions_end__ - (uint32_t)&__user_itcm_functions_start__) /
4;

    for(uint32_t i = 0; i < size; i++) {
        *dst++ = *src++;
```

```
    }

    /* Copy DTCM functions */
    printf("Copying DTCM functions...\r\n");
    src = &__user_dtcm_functions_loadaddr__;
    dst = &__user_dtcm_functions_start__;
    size = ((uint32_t)&__user_dtcm_functions_end__ - (uint32_t)&__user_dtcm_functions_start__)
/ 4;

    for(uint32_t i = 0; i < size; i++) {
        *dst++ = *src++;
    }
}
```

注意：初始化放在 cache 初始化函数之前。

2.4.4. 测试验证

通过串口打印的方式，测试将函数加载到指定位置内存布局验证，代码如*表 2-13. 函数加载到指定位置内存验证代码*所示。

表 2-13. 函数加载到指定位置内存验证代码

```
void TestUserFunctions(void)
{
    printf("=== Testing User Functions ===\r\n");

    uint8_t test_data[16] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                             0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10};

    uint16_t test_buffer[8] = {100, 200, 300, 400, 500, 600, 700, 800};
    uint32_t test_array[5] = {10, 20, 30, 40, 50};

    printf("\n--- Testing FLASH Functions ---\r\n");
    FlashFunction_Calculate(15, 25);
    FlashFunction_ProcessData(test_data, 8);

    printf("\n--- Testing RAM Functions ---\r\n");
    RamFunction_FastProcess(0x12345678);
    RamFunction_ProcessBuffer(test_buffer, 8);

    printf("\n--- Testing ITCM Functions ---\r\n");
    ItcmFunction_CriticalTask(42);
    ItcmFunction_BitOperation(0xAAAA5555);
```

```
    printf("\n--- Testing DTCM Functions ---\r\n");
    DtcmFunction_DataProcess(100);
    DtcmFunction_ArrayOperation(test_array, 5);

    printf("\n================================\r\n\r\n");
}

/*!
    \brief      main function
    \param[in]  none
    \param[out] none
    \retval     none
*/
void main(void)
{
    gd_eval_com_init(EVAL_COM);
    printf("GD32H7xx User Functions Address Test\r\n");
    printf("====================================\r\n");
    /* initialize user functions */
    UserFunctions_Init();
    /* enable the CPU cache */
    cache_enable();
    mpu_config();

    /* print all function addresses */
    PrintFunctionAddresses();
    /* test all functions */
    TestUserFunctions();

    printf("All function tests completed!\r\n");
    printf("Function address verification complete.\r\n\r\n");

    /* main application loop with periodic address display */
    uint32_t counter = 0;

    while(1) {
        counter++;

        if(counter % 5 == 1) {
            /* periodically show function addresses */
            printf("=== Periodic Address Check %d ===\r\n", counter);
```

```
            printf("FlashFunction_Calculate: 0x%08X\r\n", (uint32_t)FlashFunction_Calculate);
            printf("RamFunction_FastProcess:                    0x%08X\r\n",
(uint32_t)RamFunction_FastProcess);
            printf("ItcmFunction_CriticalTask: 0x%08X\r\n", (uint32_t)ItcmFunction_CriticalTask);
            printf("DtcmFunction_DataProcess:                   0x%08X\r\n",
(uint32_t)DtcmFunction_DataProcess);
            printf("==============================\r\n\r\n");
        }

        /* call functions to verify they work at reported addresses */
        FlashFunction_Calculate(counter, counter + 10);
        RamFunction_FastProcess(counter * 0x1000);
        ItcmFunction_CriticalTask(counter + 100);
        DtcmFunction_DataProcess(counter * 50);

        /* delay between iterations */
        for(volatile int i = 0; i < 2000000; i++);

        /* stop after several iterations */
        if(counter >= 3) {
            printf("Address verification demo completed.\r\n");
            break;
        }
    }

    while(1) {
    }
}
```

测试打印日志如*表 2-14. 将函数加载到指定位置日志*所示，测试结果与实际分配保持一致。

表 2-14. 将函数加载到指定位置日志

```
GD32H7xx User Functions Address Test
==================================
Initializing user functions...
Copying RAM functions...
  RAM functions: 0x2400C000 - 0x2400C0B0 (176 bytes)
Copying ITCM functions...
  ITCM functions: 0x00001000 - 0x000010A0 (160 bytes)
Copying DTCM functions...
  DTCM functions: 0x20001000 - 0x200010E0 (224 bytes)
User functions initialization completed!
```

```
=== Function Entry Addresses ===

--- FLASH Functions (Direct Execution) ---
Section Range: 0x08010000 - 0x080100C4
FlashFunction_Calculate        : 0x08010001
FlashFunction_ProcessData      : 0x08010071

--- RAM Functions (Copied from FLASH) ---
FLASH Source: 0x080100C4
RAM Target    : 0x2400C000 - 0x2400C0B0
RamFunction_FastProcess        : 0x2400C001
RamFunction_ProcessBuffer      : 0x2400C035

--- ITCM Functions ---
FLASH Source: 0x08010174
ITCM Target : 0x00001000 - 0x000010A0
ItcmFunction_CriticalTask      : 0x00001001
ItcmFunction_BitOperation      : 0x00001049

--- DTCM Functions ---
FLASH Source: 0x08010214
DTCM Target : 0x20001000 - 0x200010E0
DtcmFunction_DataProcess       : 0x20001001
DtcmFunction_ArrayOperation : 0x2000107D

================================

=== Testing User Functions ===

--- Testing FLASH Functions ---
FlashFunction_Calculate: a=15, b=25
FlashFunction_Calculate result: 18975
FlashFunction_ProcessData: processing 8 bytes
Data processing completed

--- Testing RAM Functions ---
RamFunction_ProcessBuffer: processing 8 elements
Buffer processing completed

--- Testing ITCM Functions ---
ItcmFunction_CriticalTask: input=42
```

```
ItcmFunction_CriticalTask result: 591753169
ItcmFunction_BitOperation: input=0xAAAA5555
ItcmFunction_BitOperation result: 0x88888888

--- Testing DTCM Functions ---
DtcmFunction_DataProcess: input=100
DtcmFunction_DataProcess result: 8160
DtcmFunction_ArrayOperation: processing 5 elements
Array operation completed

==============================

All function tests completed!
Function address verification complete.

=== Periodic Address Check 1 ===
FlashFunction_Calculate: 0x08010001
RamFunction_FastProcess: 0x2400C001
ItcmFunction_CriticalTask: 0x00001001
DtcmFunction_DataProcess: 0x20001001
==============================

FlashFunction_Calculate: a=1, b=11
FlashFunction_Calculate result: 775
ItcmFunction_CriticalTask: input=101
ItcmFunction_CriticalTask result: 591754465
DtcmFunction_DataProcess: input=50
DtcmFunction_DataProcess result: 6560
FlashFunction_Calculate: a=2, b=12
FlashFunction_Calculate result: 1425
ItcmFunction_CriticalTask: input=102
ItcmFunction_CriticalTask result: 591754257
DtcmFunction_DataProcess: input=100
DtcmFunction_DataProcess result: 8160
FlashFunction_Calculate: a=3, b=13
FlashFunction_Calculate result: 2175
ItcmFunction_CriticalTask: input=103
ItcmFunction_CriticalTask result: 591754305
DtcmFunction_DataProcess: input=150
DtcmFunction_DataProcess result: 9760
Address verification demo completed.
```

2.5. 将.c 文件代码段加载到指定位置

2.5.1. 自定义段

在链接脚本中添加用户自定义段，将.c 文件代码段分配到指定地址，地址分配如*表 2-15. 将.c 加载到指定位置地址分配总览*所示，链接文件代码如*表 2-16. gd32h7xx_flash.ld 中将全局变量和全局数组加载到指定位置*所示。

表 2-15. 将.c 加载到指定位置地址分配总览

模块文件	section 名称	FLASH 地址	RAM 地址	用途
flash_module.c	.flash_module_functions	0x08020000	-	直接在 FLASH 中执行
ram_module.c	.ram_module_functions	自动分配	0x2400D000	复制到 RAM 中执行
itcm_module.c	.itcm_module_functions	自动分配	0x00002000	复制到 ITCMRAM 中执行
dtcm_module.c	.dtcm_module_functions	自动分配	0x20002000	复制到 DTCMRAM 中执行
sram0_module.c	.sram0_module_functions	自动分配	0x3000800	复制到 SRAM0 中执行

表 2-16. gd32h7xx_flash.ld 中将全局变量和全局数组加载到指定位置

```
/* user file-based function scatter loading */
.flash_module_functions ALIGN(0x08020000, 4) :
{
    . = ALIGN(4);
    __flash_module_functions_start__ = .;
    KEEP(*flash_module.o(.text))
    KEEP(*flash_module.o(.text*))
    . = ALIGN(4);
    __flash_module_functions_end__ = .;
} > FLASH

.ram_module_functions ALIGN(0x2400D000,4):
{
    . = ALIGN(4);
    __ram_module_functions_start__ = .;
    KEEP(*ram_module.o(.text))
    KEEP(*ram_module.o(.text*))
    . = ALIGN(4);
    __ram_module_functions_end__ = .;
} > AXISRAM AT> FLASH
__ram_module_functions_loadaddr__ = LOADADDR(.ram_module_functions);

.itcm_module_functions   ALIGN(0x00002000,4):
{
    . = ALIGN(4);
```

```
    __itcm_module_functions_start__ = .;
    KEEP(*itcm_module.o(.text))
    KEEP(*itcm_module.o(.text*))
    . = ALIGN(4);
    __itcm_module_functions_end__ = .;
} > ITCMRAM AT> FLASH
__itcm_module_functions_loadaddr__ = LOADADDR(.itcm_module_functions);

.dtcm_module_functions    ALIGN(0x20002000,4):
{
    . = ALIGN(4);
    __dtcm_module_functions_start__ = .;
    KEEP(*dtcm_module.o(.text))
    KEEP(*dtcm_module.o(.text*))
    . = ALIGN(4);
    __dtcm_module_functions_end__ = .;
} > DTCMRAM AT> FLASH
__dtcm_module_functions_loadaddr__ = LOADADDR(.dtcm_module_functions);

.sram0_module_functions    ALIGN(0x30000800,4):
{
    . = ALIGN(4);
    __sram0_module_functions_start__ = .;
    KEEP(*sram0_module.o (.text))
    KEEP(*sram0_module.o(.text*))
    . = ALIGN(4);
    __sram0_module_functions_end__ = .;
} > SRAM0 AT> FLASH
__sram0_module_functions_loadaddr__ = LOADADDR(.sram0_module_functions);
```

2.5.2. C 代码中的文件定义

新建 dtcm_module.c、flash_module.c、itcm_module.c、ram_module.c 和 sram0_module.c 文件以及对应的头文件，并在每个 c 文件中定义各自的函数。主要如*表 2-17. .c 文件定义*所示。

表 2-17. .c 文件定义

```
/* dtcm_module.c */
int DtcmModule_Function1(int a, int b)
{
    /* Function with fast data access */
    printf("DtcmModule_Function1: a=%d, b=%d\r\n", a, b);
    dtcm_buffer[dtcm_counter % 64] = a + b;
```

```
    dtcm_counter++;
    return dtcm_buffer[(dtcm_counter - 1) % 64];
}
int DtcmModule_Function2(int x)
{
    /* Data-intensive function */
    printf("DtcmModule_Function2: x=%d\r\n", x);
    for(int i = 0; i < 32; i++) {
        dtcm_buffer[i] = x * i;
    }
    int sum = 0;
    for(int i = 0; i < 32; i++) {
        sum += dtcm_buffer[i];
    }
    return sum;
}
/* flash_module.c */
int FlashModule_Function1(int a, int b)
{
    /* Complex calculation function executed from FLASH */
    printf("FlashModule_Function1: a=%d, b=%d\r\n", a, b);
    int result = 0;
    for(int i = 0; i < 50; i++) {
        result += (a * b) + (i % 7);
    }
    return result;
}

int FlashModule_Function2(int x)
{
    /* Mathematical processing function */
    printf("FlashModule_Function2: x=%d\r\n", x);
    int temp = x;
    temp = temp * temp + 100;
    temp = temp % 1000;
    return temp;
}
/* itcm_module.c */
int ItcmModule_Function1(int a, int b)
{
    /* Ultra-fast critical function in ITCM */
    register int result = a * b;
```

```
    result = result + (result >> 4);
    result = result ^ 0x12345678;
    return result;
}

int ItcmModule_Function2(int x)
{
    /* Real-time processing function */
    register int temp = x;
    temp = (temp * 7) + 13;
    temp = temp & 0xFFFF;
    return temp;
}
/* ram_module.c */
int RamModule_Function1(int a, int b)
{
    /* Fast calculation function executed from RAM */
    printf("RamModule_Function1: a=%d, b=%d\r\n", a, b);
    register int result = a + b;
    result = result << 2;
    result = result ^ 0xAAAA;
    return result;
}

int RamModule_Function2(int x)
{
    /* High-speed processing function */
    printf("RamModule_Function2: x=%d\r\n", x);
    volatile int temp = x;
    for(int i = 0; i < 20; i++) {
        temp = (temp << 1) | (temp >> 31);
    }
    return temp;
}
/* sram0_module.c */
int Sram0Module_Function1(int a, int b)
{
    /* Function executed from SRAM0 */
    printf("Sram0Module_Function1: a=%d, b=%d\r\n", a, b);
    int result = a - b;
    if(result < 0) result = -result;
    return result * 2;
```

```
}
int Sram0Module_Function2(int x)
{
    /* Special processing function */
    printf("Sram0Module_Function2: x=%d\r\n", x);
    int temp = x;
    temp = temp + 0x1000;
    temp = temp & 0x7FFF;
    return temp;
}
```

2.5.3. 自定义初始化实现

在系统启动时调用自定义代码段初始化函数，如*表 2-18. .c 文件加载到指定位置初始化实现*所示。

表 2-18. .c 文件加载到指定位置初始化实现

```
void ModuleLoader_Init(void)
{
    uint32_t *src, *dst;
    uint32_t size;
    /* copy RAM module functions */
    src = &__ram_module_functions_loadaddr__;
    dst = &__ram_module_functions_start__;
    size = ((uint32_t)&__ram_module_functions_end__ -
(uint32_t)&__ram_module_functions_start__) / 4;

    for(uint32_t i = 0; i < size; i++) {
        *dst++ = *src++;
    }

    /* copy ITCM module functions */
    src = &__itcm_module_functions_loadaddr__;
    dst = &__itcm_module_functions_start__;
    size = ((uint32_t)&__itcm_module_functions_end__ -
(uint32_t)&__itcm_module_functions_start__) / 4;

    for(uint32_t i = 0; i < size; i++) {
        *dst++ = *src++;
    }

    /* copy DTCM module functions */
```

```
    src = &__dtcm_module_functions_loadaddr__;
    dst = &__dtcm_module_functions_start__;
    size = ((uint32_t)&__dtcm_module_functions_end__ -
(uint32_t)&__dtcm_module_functions_start__) / 4;

    for(uint32_t i = 0; i < size; i++) {
        *dst++ = *src++;
    }

    /* copy SRAM0 module functions */
    src = &__sram0_module_functions_loadaddr__;
    dst = &__sram0_module_functions_start__;
    size = ((uint32_t)&__sram0_module_functions_end__ -
(uint32_t)&__sram0_module_functions_start__) / 4;

    for(uint32_t i = 0; i < size; i++) {
        *dst++ = *src++;
    }
}
```

注意：初始化放在 cache 初始化函数之前。

2.5.4. 测试验证

通过串口打印的方式，测试将函数加载到指定位置内存布局验证，代码如*表 2-19. .c 文件加载到指定位置内存验证代码*所示。

表 2-19. .c 文件加载到指定位置内存验证代码

```
void TestAllModules(void)
{
    int result;
    uint8_t test_data[16] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                             0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10};

    printf("=== Testing C File-based Module Functions ===\r\n");

    /* Test FLASH module */
    printf("\n--- Testing FLASH Module (flash_module.c) ---\r\n");
    printf("Executing FlashModule_Function1 at 0x%08X\r\n", (uint32_t)FlashModule_Function1);
    result = FlashModule_Function1(15, 25);
    printf("Result: %d\r\n", result);

    printf("Executing FlashModule_Function2 at 0x%08X\r\n", (uint32_t)FlashModule_Function2);
```

```
    result = FlashModule_Function2(100);
    printf("Result: %d\r\n", result);

    printf("Executing FlashModule_ProcessData at 0x%08X\r\n",
(uint32_t)FlashModule_ProcessData);
    FlashModule_ProcessData(test_data, 8);

    /* Test RAM module */
    printf("\n--- Testing RAM Module (ram_module.c) ---\r\n");
    printf("Executing RamModule_Function1 at 0x%08X\r\n", (uint32_t)RamModule_Function1);
    result = RamModule_Function1(10, 20);
    printf("Result: %d\r\n", result);

    printf("Executing RamModule_Function2 at 0x%08X\r\n", (uint32_t)RamModule_Function2);
    result = RamModule_Function2(0x1234);
    printf("Result: 0x%08X\r\n", result);

    printf("Executing RamModule_ProcessData at 0x%08X\r\n",
(uint32_t)RamModule_ProcessData);
    RamModule_ProcessData(test_data, 8);

    /* Test ITCM module */
    printf("\n--- Testing ITCM Module (itcm_module.c) ---\r\n");
    printf("Executing ItcmModule_Function1 at 0x%08X\r\n", (uint32_t)ItcmModule_Function1);
    result = ItcmModule_Function1(7, 11);
    printf("Result: %d\r\n", result);

    printf("Executing ItcmModule_Function2 at 0x%08X\r\n", (uint32_t)ItcmModule_Function2);
    result = ItcmModule_Function2(500);
    printf("Result: %d\r\n", result);

    printf("Executing ItcmModule_ProcessData at 0x%08X\r\n",
(uint32_t)ItcmModule_ProcessData);
    ItcmModule_ProcessData(test_data, 8);

    /* Test DTCM module */
    printf("\n--- Testing DTCM Module (dtcm_module.c) ---\r\n");
    printf("Executing DtcmModule_Function1 at 0x%08X\r\n", (uint32_t)DtcmModule_Function1);
    result = DtcmModule_Function1(33, 44);
    printf("Result: %d\r\n", result);

    printf("Executing DtcmModule_Function2 at 0x%08X\r\n", (uint32_t)DtcmModule_Function2);
```

```
    result = DtcmModule_Function2(200);
    printf("Result: %d\r\n", result);

    printf("Executing DtcmModule_ProcessData at 0x%08X\r\n",
(uint32_t)DtcmModule_ProcessData);
    DtcmModule_ProcessData(test_data, 8);

    /* Test SRAM0 module */
    printf("\n--- Testing SRAM0 Module (sram0_module.c) ---\r\n");
    printf("Executing Sram0Module_Function1 at 0x%08X\r\n", (uint32_t)Sram0Module_Function1);
    result = Sram0Module_Function1(60, 40);
    printf("Result: %d\r\n", result);

    printf("Executing Sram0Module_Function2 at 0x%08X\r\n", (uint32_t)Sram0Module_Function2);
    result = Sram0Module_Function2(0x8000);
    printf("Result: 0x%04X\r\n", result);

    printf("Executing Sram0Module_ProcessData at 0x%08X\r\n",
(uint32_t)Sram0Module_ProcessData);
    Sram0Module_ProcessData(test_data, 8);

    printf("\n==========================================\r\n\r\n");
}

/*!
    \brief      main function
    \param[in]  none
    \param[out] none
    \retval     none
*/
void main(void)
{
    gd_eval_com_init(EVAL_COM);

    printf("GD32H7xx C File-based Scatter Loading Example\r\n");
    printf("==========================================\r\n");
    printf("This example demonstrates loading entire C files\r\n");
    printf("to different memory regions for optimized execution.\r\n\r\n");

    /* Initialize all modules */
    ModuleLoader_Init();
```

```
/* Enable the CPU cache */
cache_enable();
mpu_config();

/* Print all module function addresses */
PrintModuleAddresses();

/* Test all modules */
TestAllModules();

printf("All C file-based module tests completed!\r\n");
printf("Each module file is loaded to its designated memory region.\r\n");

uint32_t loop_count = 0;

while(1) {
    loop_count++;

    /* Periodic address verification */
    if(loop_count % 10 == 1) {
        printf("\n=== Periodic Address Verification (Loop %d) ===\r\n", loop_count);
        printf("flash_module.c functions in FLASH:\r\n");
        printf("    FlashModule_Function1: 0x%08X\r\n", (uint32_t)FlashModule_Function1);

        printf("ram_module.c functions in AXISRAM:\r\n");
        printf("    RamModule_Function1    : 0x%08X\r\n", (uint32_t)RamModule_Function1);

        printf("itcm_module.c functions in ITCMRAM:\r\n");
        printf("    ItcmModule_Function1 : 0x%08X\r\n", (uint32_t)ItcmModule_Function1);

        printf("dtcm_module.c functions in DTCMRAM:\r\n");
        printf("    DtcmModule_Function1 : 0x%08X\r\n", (uint32_t)DtcmModule_Function1);

        printf("sram0_module.c functions in SRAM0:\r\n");
        printf("    Sram0Module_Function1: 0x%08X\r\n", (uint32_t)Sram0Module_Function1);
        printf("============================================\r\n\r\n");
    }

    /* Simple delay */
    for(volatile int i = 0; i < 2000000; i++);

    /* Stop after a few loops for demonstration */
```

```
            if(loop_count >= 3) {
                printf("C file-based scatter loading demonstration completed.\r\n");
                break;
            }
        }

        while(1) {

        }
}
```

测试打印日志如*表 2-20. 将.c 文件加载到指定位置日志*所示，测试结果与实际分配保持一致。

表 2-20. 将.c 文件加载到指定位置日志

```
GD32H7xx C File-based Scatter Loading Example
==============================================
This example demonstrates loading entire C files
to different memory regions for optimized execution.

Initializing C file-based module functions...
Copying RAM module functions...
   RAM module: 0x2400D000 - 0x2400D158 (344 bytes)
Copying ITCM module functions...
   ITCM module: 0x00002000 - 0x000020E0 (224 bytes)
Copying DTCM module functions...
   DTCM module: 0x20002000 - 0x20002198 (408 bytes)
Copying SRAM0 module functions...
   SRAM0 module: 0x30000800 - 0x30000928 (296 bytes)
C file-based module functions initialization completed!


=== Module Function Entry Addresses ===

--- FLASH Module (flash_module.c) ---
Section Range: 0x08020000 - 0x08020130
FlashModule_Function1          : 0x08020001
FlashModule_Function2          : 0x08020065
FlashModule_ProcessData        : 0x080200B1
FlashModule_GetProcessedValue: 0x08020115

--- RAM Module (ram_module.c) ---
FLASH Source: 0x08020130
```

```
RAM Target    : 0x2400D000 - 0x2400D158
RamModule_Function1              : 0x2400D001
RamModule_Function2              : 0x2400D035
RamModule_ProcessData            : 0x2400D075
RamModule_SortArray              : 0x2400D11F

--- ITCM Module (itcm_module.c) ---
FLASH Source: 0x08020288
ITCM Target : 0x00002000 - 0x000020E0
ItcmModule_Function1             : 0x00002001
ItcmModule_Function2             : 0x0000202D
ItcmModule_ProcessData           : 0x0000204F
ItcmModule_CriticalCalculation: 0x000020B3

--- DTCM Module (dtcm_module.c) ---
FLASH Source: 0x08020368
DTCM Target : 0x20002000 - 0x20002198
DtcmModule_Function1             : 0x20002001
DtcmModule_Function2             : 0x2000206D
DtcmModule_ProcessData           : 0x200020D9
DtcmModule_GetBufferSum          : 0x20002155

--- SRAM0 Module (sram0_module.c) ---
FLASH Source: 0x08020500
SRAM0 Target: 0x30000800 - 0x30000928
Sram0Module_Function1            : 0x30000801
Sram0Module_Function2            : 0x30000839
Sram0Module_ProcessData          : 0x3000086D
Sram0Module_ValidateAndProcess: 0x300008F9

========================================

=== Testing C File-based Module Functions ===

--- Testing FLASH Module (flash_module.c) ---
Executing FlashModule_Function1 at 0x08020001
FlashModule_Function1: a=15, b=25
Result: 18897
Executing FlashModule_Function2 at 0x08020065
FlashModule_Function2: x=100
Result: 100
Executing FlashModule_ProcessData at 0x080200B1
```

```
FlashModule_ProcessData: processing 8 bytes

--- Testing RAM Module (ram_module.c) ---
Executing RamModule_Function1 at 0x2400D001
RamModule_Function1: a=10, b=20
Result: 43730
Executing RamModule_Function2 at 0x2400D035
RamModule_Function2: x=4660
Result: 0xFFFFFFFF
Executing RamModule_ProcessData at 0x2400D075
RamModule_ProcessData: processing 8 bytes

--- Testing ITCM Module (itcm_module.c) ---
Executing ItcmModule_Function1 at 0x00002001
Result: 305419817
Executing ItcmModule_Function2 at 0x0000202D
Result: 3513
Executing ItcmModule_ProcessData at 0x0000204F

--- Testing DTCM Module (dtcm_module.c) ---
Executing DtcmModule_Function1 at 0x20002001
DtcmModule_Function1: a=33, b=44
Result: 77
Executing DtcmModule_Function2 at 0x2000206D
DtcmModule_Function2: x=200
Result: 99200
Executing DtcmModule_ProcessData at 0x200020D9
DtcmModule_ProcessData: processing 8 bytes

--- Testing SRAM0 Module (sram0_module.c) ---
Executing Sram0Module_Function1 at 0x30000801
Sram0Module_Function1: a=60, b=40
Result: 40
Executing Sram0Module_Function2 at 0x30000839
Sram0Module_Function2: x=32768
Result: 0x1000
Executing Sram0Module_ProcessData at 0x3000086D
Sram0Module_ProcessData: processing 8 bytes

==========================================

All C file-based module tests completed!
```

```
Each module file is loaded to its designated memory region.

=== Periodic Address Verification (Loop 1) ===
flash_module.c functions in FLASH:
   FlashModule_Function1: 0x08020001
ram_module.c functions in AXISRAM:
   RamModule_Function1    : 0x2400D001
itcm_module.c functions in ITCMRAM:
   ItcmModule_Function1 : 0x00002001
dtcm_module.c functions in DTCMRAM:
   DtcmModule_Function1 : 0x20002001
sram0_module.c functions in SRAM0:
   Sram0Module_Function1: 0x30000801
================================================

C file-based scatter loading demonstration completed.
```

2.6. SDRAM 分散加载实现

SDRAM 分散加载的实现可参考 *2.3*、*2.4*、*2.5* 章节描述，在进行自定义初始化之前，需要实现 EXMC 初始化和 MPU 的相关配置，代码如*表 2-21. SDRAM 初始化代码*所示。

表 2-21. SDRAM 初始化代码

```
#include "exmc_sdram.h"
/*!
    \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++);
    }
```

```
}
```

注意 ：在 M7 内核的默认配置中，某些地址位于禁止指令执行的区域。因此，如果将代码加载到这些区域，在执行过程中会发生错误。GD32H7xx 的 EXMC 中的 SDRAM 地址分配范围为 0xC0000000-0xDFFFFFFF，该范围属于禁止指令执行的地址区域。需要配置 MPU（内存保护单元）寄存器，以启用 0xC0000000 地址范围内的指令执行。

3. 版本历史

表 3-1. 版本历史

版本号	说明	日期
1.0	首次发布	2025 年 12 月 31 日

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.