

GigaDevice Semiconductor Inc.

适用于 GD32MCU Keil MDK5 分散加载说明

应用笔记

AN206

1.1 版本

(2026 年 2 月)

目录

目录	2
图索引	3
表索引	4
1. Keil 中分散加载简介	5
2. 分散加载在 Keil 中的实现	6
2.1. 使用手动编写的 sct 文件	6
2.2. 将全局变量加载到指定位置	9
2.3. 将函数加载到指定位置	10
2.4. 将数组加载到指定位置	11
2.4.1. 未初始化数组加载到指定位置	11
2.4.2. 常量数组加载到指定位置	11
2.4.3. 全局初始化数组加载到指定位置	12
2.5. 将.c 文件加载到指定位置	13
3. SDRAM 分散加载实现	14
3.1. SDRAM 分散加载的实现	14
4. 结果	18
5. 历史版本	19

图索引

图 2-1. 使用手动编写的 sct 文件.....	6
图 2-2. sct 文件内存分布简图.....	8
图 2-3. Execute-only Code 编译选项.....	9
图 2-4. 函数加载到指定位置程序调试结果.....	11
图 2-5. 数组加载到指定位置程序调试结果.....	13
图 2-6. 将.c 文件加载到指定位置程序调试结果.....	13
图 3-1. 在 startup_gd32h7xx.s 中加入代码.....	15
图 3-2. 将函数和.c 文件加载到 SDRAM 指定位置程序调试结果.....	17
图 3-3. J-link 复位选项配置.....	17
图 4-1. 分散加载工程编译 Project.map 文件.....	18

表索引

表 2-1. Project.sct 代码.....	6
表 2-2. Project.sct 中将全局变量加载到指定位置代码.....	9
表 2-3. main.c 中将全局变量加载到指定位置代码.....	9
表 2-4. 将全局变量加载到指定位置打印结果.....	9
表 2-5. Project.sct 中将函数加载到指定位置代码.....	10
表 2-6. main.c 中将函数加载到指定位置代码.....	10
表 2-7. Project.sct 中将未初始化数组加载到指定位置代码.....	11
表 2-8. main.c 中将未初始化数组加载到指定位置代码.....	11
表 2-9. main.c 中将常量数组加载到指定位置代码.....	12
表 2-10. Project.sct 中将全局初始化数组加载到指定位置代码.....	12
表 2-11. main.c 中将全局初始化数组加载到指定位置代码.....	12
表 2-12. 将数组加载到指定位置打印结果.....	12
表 2-13. Project.sct 中将文件加载到指定位置代码.....	13
表 3-1. Project.sct 中 SDRAM 分散加载实现代码.....	14
表 3-2. Dolnit 函数实现代码.....	15
表 3-3. MPU 配置代码.....	16
表 3-4. 将变量、数组、函数和文件分散加载到 SDRAM 指定位置代码.....	16
表 3-5. 将变量和数组加载到 SDRAM 指定位置打印结果.....	17
表 5-1. 历史版本.....	19

1. Keil 中分散加载简介

在 Keil 默认配置生成的工程中，MDK 会根据我们在 option 选项中所选择的芯片型号，得到芯片的 FLASH 和 RAM 大小等信息，并且会自动生成一个以工程名命名后缀为*.sct 的分散加载文件(Linker Control File, scatter loading)，链接器根据所生成分散加载文件的配置来决定各个段在存储器上的分配地址。因此我们可以通过修改该文件来实现指定代码段在不同位置的存储。

本应用笔记基于 GD32H73x_75x 系列，采用 GD32H759i-EVAL 开发板，Keil 版本为 5.30.0.0，编译器版本为 V6.14，分别介绍如何实现以下功能：

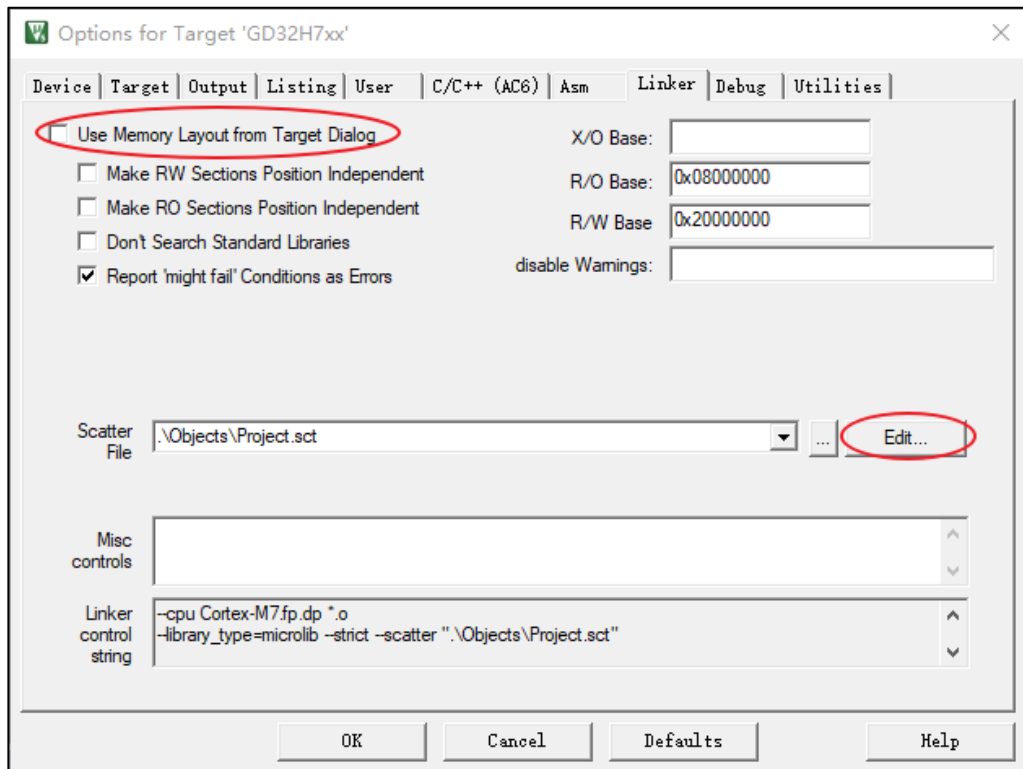
- 实现全局变量加载到指定位置
- 实现函数加载到指定位置
- 实现数组加载到指定位置
- 实现.c 文件加载到指定位置
- 实现上述功能加载到 SDRAM 指定位置

2. 分散加载在 Keil 中的实现

2.1. 使用手动编写的 sct 文件

本工程直接使用手动编写的 sct 文件，在 MDK 的“Options for Target->Linker->Use Memory Layout from Target Dialog”选项取消勾选，取消勾选后可直接点击“Edit”按钮编辑工程的 sct 文件，相关配置如 [图 2-1. 使用手动编写的 sct 文件](#) 所示。

图 2-1. 使用手动编写的 sct 文件



同时也可到工程目录“GD32H7xx_ScatterLoading_v1.0.0\Project\Keil_project\Objects\Project.sct”下打开编辑，文件打开代码如 [表 2-1. Project.sct 代码](#) 所示：

表 2-1. Project.sct 代码

```

; *****
;
; *** 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)
}
}

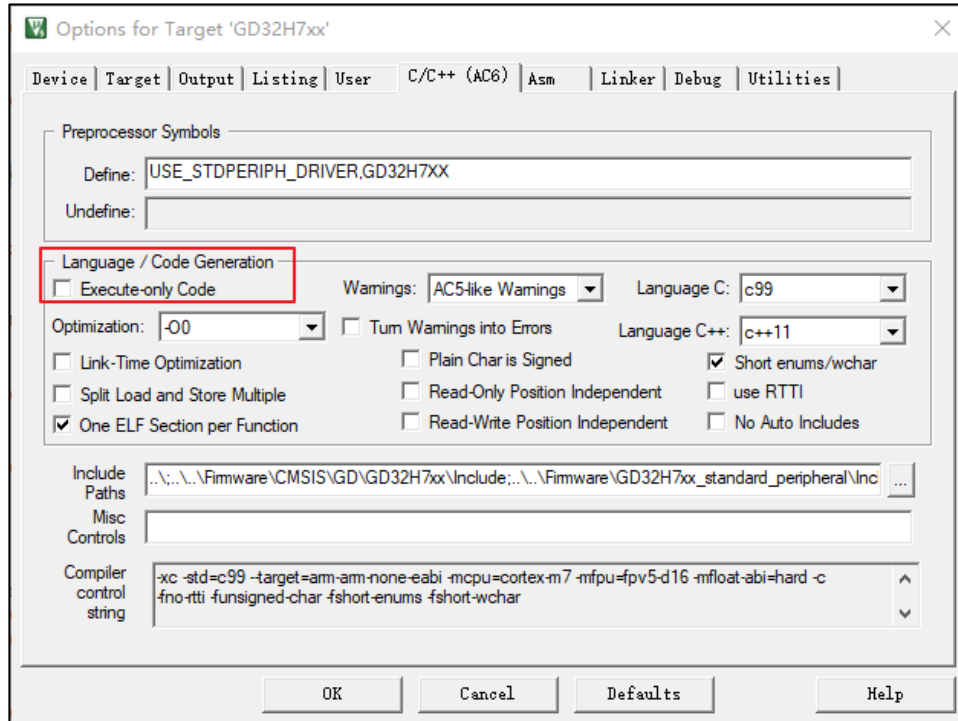
/**** Function scatter loading ****/
LR_IROM2 0x08020000 0x00010000 {
    ER_IROM_FUNC 0x08020000 0x00010000 {
        main.o(ROM_FUNC)
    }
    ER_IRAM_FUNC 0x24040000 0x0010000 {
        main.o(SRAM_FUCN)
    }
}
}

/**** File scatter loading ****/

```


注意：在 Keil 编译时需要将 Execute-only Code 编译选项勾选去掉。如[图 2-3. Execute-only Code 编译选项](#)所示。

图 2-3. Execute-only Code 编译选项



2.2. 将全局变量加载到指定位置

在 Project.sct 文件中加入如下代码，代码如[表 2-2. Project.sct 中将全局变量加载到指定位置代码](#)所示。

表 2-2. Project.sct 中将全局变量加载到指定位置代码

```

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

上述代码将指定的 RAM_VARIABLE 段加载到 0x24020000 起始位置，在 main.c 文件中定义全局变量如[表 2-3. main.c 中将全局变量加载到指定位置代码](#)所示：

表 2-3. main.c 中将全局变量加载到指定位置代码

```

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

通过 printf 函数打印变量地址，结果如[表 2-4. 将全局变量加载到指定位置打印结果](#)所示：

表 2-4. 将全局变量加载到指定位置打印结果

```

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

2.3. 将函数加载到指定位置

在 Project.sct 文件中加入如下代码，代码如[表 2-5. Project.sct 中将函数加载到指定位置代码](#)所示：

表 2-5. Project.sct 中将函数加载到指定位置代码

```

/**** 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)
  }
}

```

上述代码将指定 main.o 模块中的 ROM_FUNC 段和 SRAM_FUCN 段分别加载到 0x08020000 起始位置和 0x24040000 起始位置。在 main.c 文件中将 delay 函数和 fill_TX_Data 函数分别分配到 ROM_FUNC 和 SRAM_FUCN，代码如[表 2-6. main.c 中将函数加载到指定位置代码](#)所示：

表 2-6. main.c 中将函数加载到指定位置代码

```

/* 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<0x2ffff;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_FUCN")));
void fill_TX_Data()

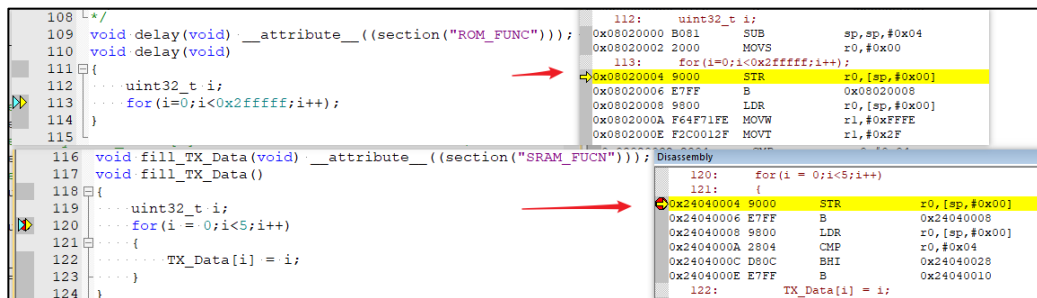
```

```

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

程序调试结果如 [图 2-4. 函数加载到指定位置程序调试结果](#)所示：

图 2-4. 函数加载到指定位置程序调试结果



2.4. 将数组加载到指定位置

2.4.1. 未初始化数组加载到指定位置

在 Project.sct 文件中加入如下代码，代码如 [表 2-7. Project.sct 中将未初始化数组加载到指定位置代码](#)所示：

表 2-7. Project.sct 中将未初始化数组加载到指定位置代码

```

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

上述代码将 main.o 模块中.bss.RAM_Array 段加载到 0x24010000 起始位置，在 main.c 中定义数组 TX_Data[]，代码如 [表 2-8. main.c 中将未初始化数组加载到指定位置代码](#)所示：

表 2-8. main.c 中将未初始化数组加载到指定位置代码

```

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

2.4.2. 常量数组加载到指定位置

在数组后加入 __attribute__((section(".ARM.__at_XXXXXXX")))，本例程在 main.c 中定义数组 const char constdata[]代码如 [表 2-9. main.c 中将常量数组加载到指定位置代码](#)所示。

表 2-9. main.c 中将常量数组加载到指定位置代码

```
/* 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
};
```

2.4.3. 全局初始化数组加载到指定位置

在 Project.sct 文件中加入如下代码，代码如[表 2-10. Project.sct 中将全局初始化数组加载到指定位置代码](#)所示。在 main.c 中定义 test_sram[],代码如[表 2-11. main.c 中将全局初始化数组加载到指定位置代码](#)所示。

表 2-10. Project.sct 中将全局初始化数组加载到指定位置代码

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

表 2-11. main.c 中将全局初始化数组加载到指定位置代码

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

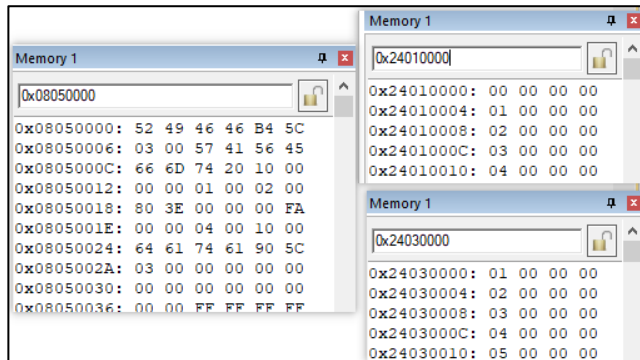
通过 printf 函数打印数组地址，结果如[表 2-12. 将数组加载到指定位置打印结果](#)所示：

表 2-12. 将数组加载到指定位置打印结果

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

程序调试结果如[图 2-5. 数组加载到指定位置程序调试结果](#)所示：

图 2-5. 数组加载到指定位置程序调试结果



2.5. 将.c 文件加载到指定位置

在 Project.sct 文件中加入如下代码，代码如 [表 2-13. Project.sct 中将文件加载到指定位置代码](#) 所示：

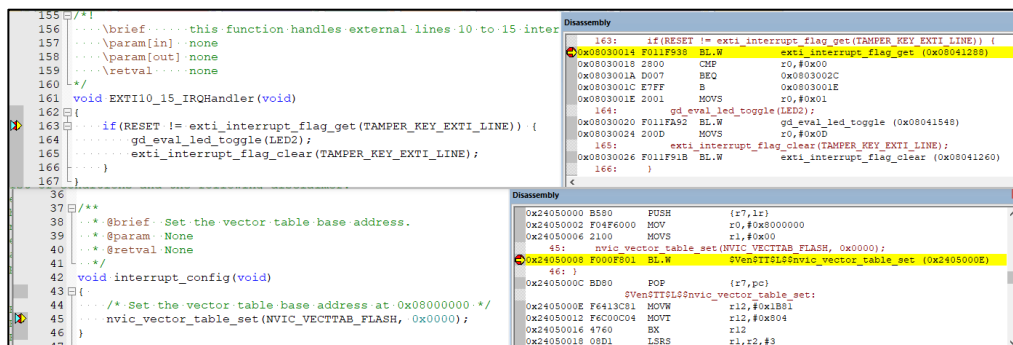
表 2-13. Project.sct 中将文件加载到指定位置代码

```

/**** 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)
  }
}
  
```

上述代码将指定 gd32h7xx_it.o 文件加载到 0x08030000 起始位置，将 hw_config.o 文件加载到 0x24050000 起始位置，程序调试结果如下：

图 2-6. 将.c 文件加载到指定位置程序调试结果



注意：GD32H7xx 系列 MCU 包含 ITCM SRAM 和 DTCM SRAM，可以根据上面提供的方法，按照项目需求，修改分散加载文件，实现变量、数组、函数和文件加载到指定区域。

3. SDRAM 分散加载实现

3.1. SDRAM 分散加载的实现

在 Project.sct 文件中加入如下红色字体代码，代码如[表 3-1. Project.sct 中 SDRAM 分散加载实现代码](#)所示：

表 3-1. Project.sct 中 SDRAM 分散加载实现代码

```

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 0x00010000 {
    *(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)
  }
}

```

上述代码将 SDRAM_FUNC 段、SDRAM_ARRAY 段和 test.o 文件将分别加载到 0xc0000000、0xc0001000 和 0xc0002000 起始地址。

在 startup_gd32h7xx.s 中加入如下代码，代码如 [图 3-1. 在 startup_gd32h7xx.s 中加入代码](#) 所示：

图 3-1. 在 startup_gd32h7xx.s 中加入代码

```

; /* 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

```

其中 DoInit 函数定义在 main.c 中，该函数主要实现 EXMC 初始化和 MPU 的相关配置，函数代码如 [表 3-2. DoInit 函数实现代码](#) 所示：

表 3-2. DoInit 函数实现代码

```

/*!
 \brief initialize the sdram, setup 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++);
    }
}

```

注意：在 M7 内核的默认配置中，部分地址处于禁止执行指令的地址段，因此若当代码加载到该段上，在执行时会发生错误。GD32H7xx 的 EXMC 中 SDRAM 的地址分配为 0xC0000000-0xDFFFFFFF 位于该地址段。通过配置 MPU (Memory Protect Unit) 寄存器，让 0xC0000000

地址段可执行指令。MPU 配置代码如[表 3-3. MPU 配置代码](#)所示。

表 3-3. MPU 配置代码

```

/*!
 *brief      configure the 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);
}

```

在 main.c 中定义变量 uint32_t testValue_SDRAM，数组 int test_sdram[5]，函数 testFuncInSDRAM，以及加入文件 test.c，主要代码如[表 3-4. 将变量、数组、函数和文件分散加载到 SDRAM 指定位置代码](#)所示：

表 3-4. 将变量、数组、函数和文件分散加载到 SDRAM 指定位置代码

```

/* 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);
}
    
```

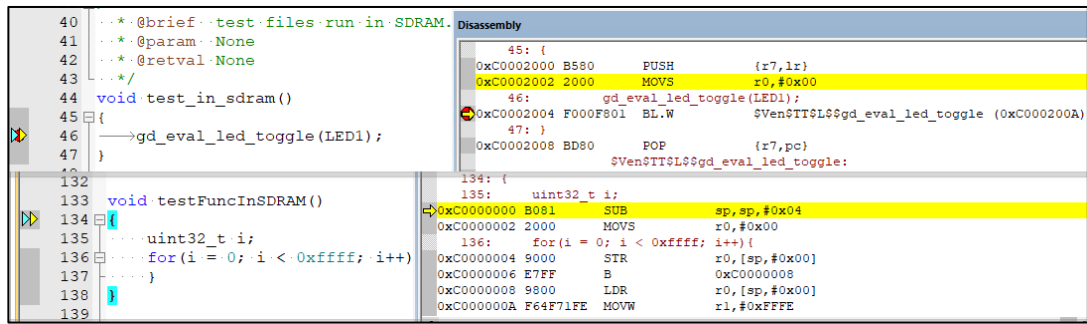
程序运行和调试结果如 [表 3-5. 将变量和数组加载到 SDRAM 指定位置打印结果](#)和 [图 3-2. 将函数和.c 文件加载到 SDRAM 指定位置程序调试结果](#)所示:

表 3-5. 将变量和数组加载到 SDRAM 指定位置打印结果

```

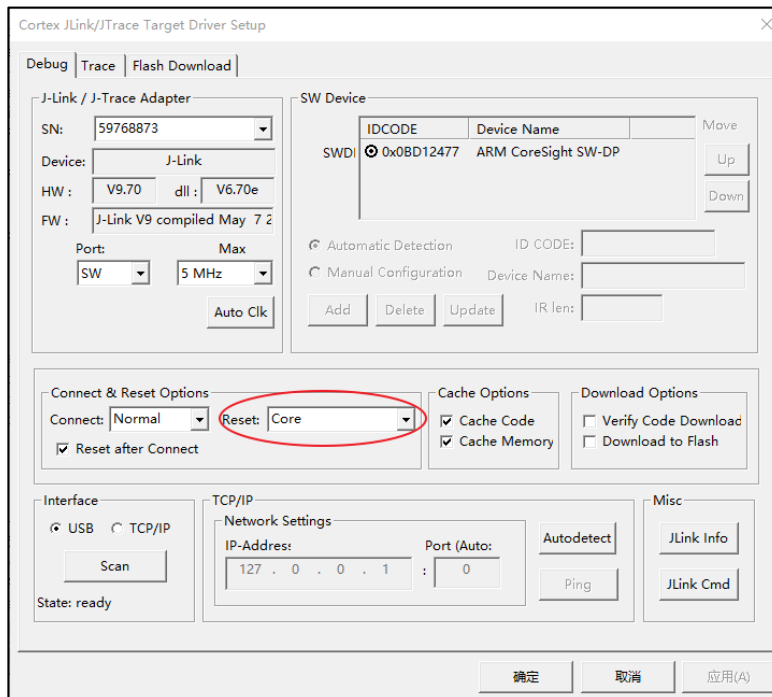
testValue_SDRAM is 0xc0003000, value is 0x5
test_sdram is 0xc0001000
    
```

图 3-2. 将函数和.c 文件加载到 SDRAM 指定位置程序调试结果



注意: 在使用 J-Link 进入调试时, 使用 Reset 配置选项选择 CORE, 避免 MCU 进入调试时, SDRAM 被复位, 导致调试失败问题。如 [图 3-3. J-link 复位选项配置](#)所示。

图 3-3. J-link 复位选项配置



4. 结果

可以查看“GD32H7XX_ScatterLoading_v1.0.0\Project\Keil\MDK-ARM>Listings\Project.map”文件，进一步分析内存分布情况，打开如[图 4-1. 分散加载工程编译 Project.map 文件](#)所示：

图 4-1. 分散加载工程编译 Project.map 文件

```

=====
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$4$00000000--mc_w_l(entry.o)
--0x080003a4--0x080003a4--0x00000004--Code--RO--3756--*.ARM.Collect$4$00000001--mc_w_l(entry2.o)
--0x080003a8--0x080003a8--0x00000004--Code--RO--3759--*.ARM.Collect$4$00000004--mc_w_l(entry5.o)
--0x080003ac--0x080003ac--0x00000000--Code--RO--3761--*.ARM.Collect$4$00000008--mc_w_l(entry7b.o)
--0x080003ac--0x080003ac--0x00000000--Code--RO--3763--*.ARM.Collect$4$0000000A--mc_w_l(entry8b.o)
--0x080003ac--0x080003ac--0x00000008--Code--RO--3764--*.ARM.Collect$4$0000000B--mc_w_l(entry9a.o)
--0x080003b4--0x080003b4--0x00000000--Code--RO--3766--*.ARM.Collect$4$0000000D--mc_w_l(entry10a.o)
--0x080003b4--0x080003b4--0x00000000--Code--RO--3768--*.ARM.Collect$4$0000000F--mc_w_l(entry11a.o)
--0x080003b4--0x080003b4--0x00000004--Code--RO--3757--*.ARM.Collect$4$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--Regions$4Table--anon$4obj.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--0x080004c8--0x00000004--Zero--RW--69--.bss.delay--systick.o
--0x2400000c--0x080004cc--0x00000004--PAD
--0x24000010--0x080004d0--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--0x080004c8--0x00000014--Zero--RW--19--.bss.RAM_Array--main.o

```

从 map 文件可以看出各段的加载地址和执行地址，符合指定的分散加载区域。

5. 历史版本

表 5-1. 历史版本

版本号.	描述	日期
1.0	首次发布	2024 年 05 月 09 日
1.1	更新 AN 名称为“适用于 GD32MCU Keil MDK5 分散加载说明”。	2026 年 02 月 03 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.