GigaDevice Semiconductor Inc.

Arm® Cortex®-M3/4 32-bit MCU

应用笔记

AN032

目录

表索引

图索引

1. IAR 中分散加载简介

在 IAR 默认配置生成的工程中，IAR 会根据我们在 General option 中所选择的芯片型号，得到芯片 FLASH 和 SRAM 大小等信息，选择相应的*.icf 的分散加载文件(Linker Control File，scatter loading)，链接器根据该文件的配置分配各个节区地址，生成分散加载代码，因此我们可以通过修改该文件来实现指定代码节区在不同位置的存储。

本应用笔记基于 GD32F4xx 系列，采用 GD32F450i-EVAL 开发板，IAR 版本为 7.40.2，分别介绍如何实现以下功能：

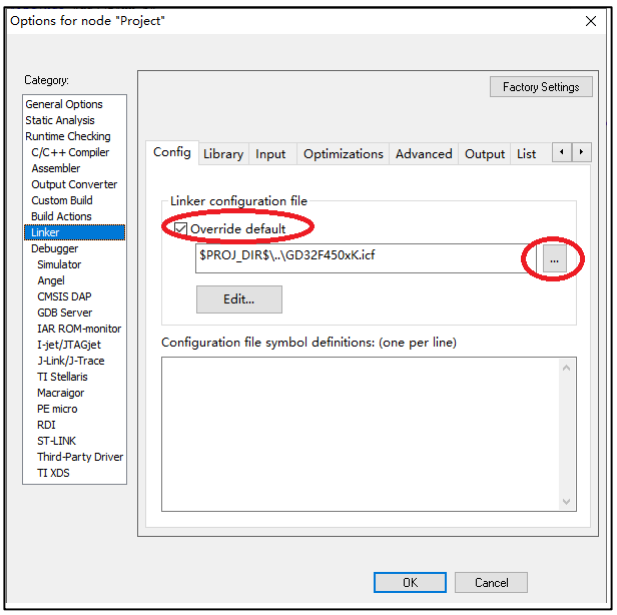
- 实现全局变量加载到指定位置
- 实现函数加载到指定位置
- 实现数组加载到指定位置
- 实现.c 文件加载到指定位置
- 实现上述功能加载到 SDRAM 指定位置

2. 分散加载在 IAR 中的实现

2.1. 使用手动编写的 icf 文件

本工程直接使用手动编写的 icf 文件，在 IAR 的“Project->Option->Linker->Config->Linker configuration file”选项勾选 override default，勾选后点击“...”按钮，选择工程目录“GD32F4xx_ScatterLoading_v1.0.0\Project\IAR_project\GD32F450xK.icf”，相关配置如*图 2-1. 使用手动编写的 icf 文件*所示：

图 2-1. 使用手动编写的 icf 文件



打开 GD32F450xK.icf 进行编辑，文件打开代码如表 2-1：

表 2-1. GD32F450xK.icf 代码

```
/*###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};
```

红色部分为实现分散加载主要添加的部分，下面进行详细分析。

2.2. 将全局变量加载到指定位置

本例程中在 main.c 文件中定义全局变量 uint32_t testValue_ROM 和 uint32_t testValue_RAM。

方式一：通过定义 section variable,在 GD32F450xK.icf 文件中添加如下代码,如*表 2-2. GD32F450.icf 中将全局变量加载到指定位置代码*所示:

表 2-2. GD32F450.icf 中将全局变量加载到指定位置代码

```
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}
```

在 main.c 中定义全局变量 uint32_t testValue_RAM，代码如*表 2-3. main.c 中将全局变量加载到指定位置代码 1* 所示：

表 2-3. main.c 中将全局变量加载到指定位置代码 1

```
/* load the variable testValue_RAM to ram address 0x20001200 */
uint32_t testValue_RAM @"variable"=6;
```

方式二：通过添加“@”操作符直接将变量加载到指定位置，代码如下：

表 2-4. main.c 中将全局变量加载到指定位置代码 2

```
/* load the variable testValue_ROM to flash address 0x08080000 */
uint32_t testValue_ROM @0x08080000=5;
```

通过 printf 函数打印变量地址，结果如*表 2-5. 将全局变量加载到指定位置打印结果*所示：

表 2-5. 将全局变量加载到指定位置打印结果

```
variable testValue_ROM address is 0x8080000
variable testValue_RAM address is 0x20001200
```

2.3. 将函数加载到指定位置

在 GD32F450xK.icf 文件中加入如下代码，代码如*表 2-6. GD32F450xK.icf 中将函数加载到指定位置代码*所示：

表 2-6. GD32F450xK.icf 中将函数加载到指定位置代码

```
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};
```

上述代码将通过定义不同的 region，将 section .funflash 放在 0x0800F000 定义的地址空间，将 section funram 放在 RAM1_region 定义的地址空间。

在 main.c 文件中通过加入“@”或者“#pragma location =”将 delay 函数和 fill_TX_Data 函数分别分配到 section .funflash 和 section funram，代码如*表 2-7. main.c 中将函数加载到指定位置代码*所示：

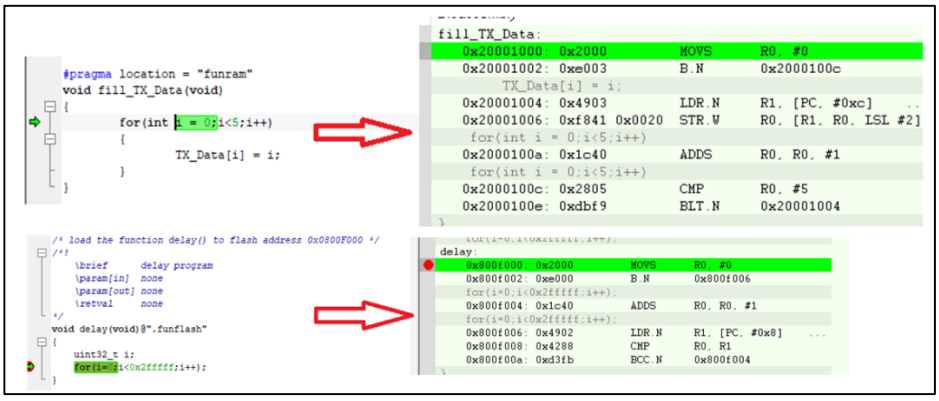
表 2-7. main.c 中将函数加载到指定位置代码

```
/* 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<0x2fffff;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;
    }
}
```

程序调试结果如*图 2-2. 函数加载到指定位置程序调试结果*所示：

图 2-2. 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

本例程中在 const-data.c 文件中定义数组 constdata[],在 main.c 中定义数组 TX_Data[]，方式一：通过添加上述通过定义 section array,在 GD32F450xK.icf 文件中添加如下代码，代码如*表 2-8. GD32F450.sct 中将函数加载到指定位置代码*所示：

表 2-8. GD32F450.sct 中将函数加载到指定位置代码

```
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};
```

在 main.c 中定义数组 TX_Data[]，代码如*表 2-9. main.c 中将数组加载到指定位置代码 1* 所示：

表 2-9. main.c 中将数组加载到指定位置代码 1

```
/* load the array TX_Data[5] to sram address 0x20001300 */
uint32_t TX_Data[5]@"array"={0};
```

方式二：通过添加“@”操作符直接将数组加载到指定位置，代码如*表 2-10. data.c 中将数组加载到指定位置代码*所示：

表 2-10. data.c 中将数组加载到指定位置代码

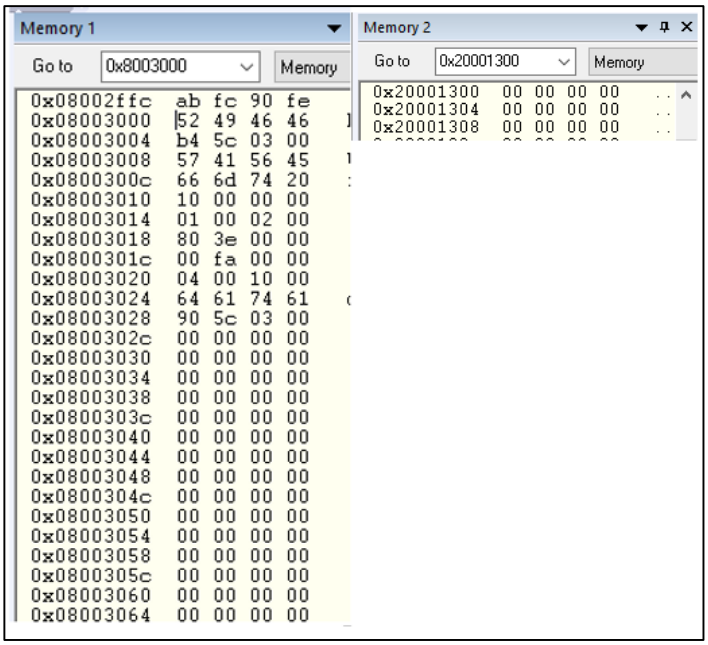
```
/* Load const array constdata to address 0x08003000 */
const char constdata[]@0x8003000={
    0x52,0x49,0x46,0x46,0xB4,0x5C,0x03,0x00,
    0x57,0x41,0x56,0x45,0x66,0x6D,0x74,0x20,
    0x10,0x00,0x00,0x00,0x01,0x00,0x02,0x00,
    0x80,0x3E,0x00,0x00,0x00,0xFA,0x00,0x00,
    0x04,0x00,0x10,0x00,0x64,0x61,0x74,0x61,
     0x90,0x5C,0x03,0x00,0x00,0x00,0x00,0x00,
     …
     }
```

通过 printf 函数打印数组地址，结果如*表 2-11. 将数组加载到指定位置打印结果*所示：

表 2-11. 将数组加载到指定位置打印结果

```
constdata address is 0x8003000
TX_Data address is 0x20001300
```

图 2-3. 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 GD32F450xK.icf 文件中代码如*表2-12. GD32F450.icf 中将文件加载到指定位置代码*所示：

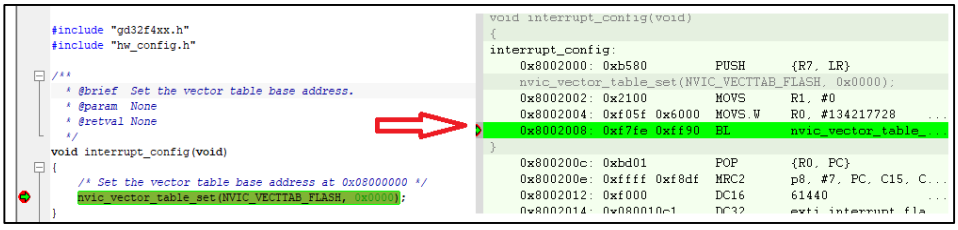
表 2-12. GD32F450.icf 中将文件加载到指定位置代码

```
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};
```

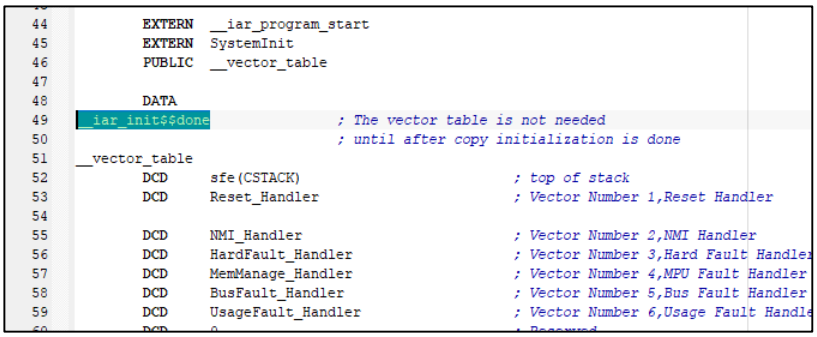
通过将 hw_config.o 文件的加载到地址 0x08002000 处，程序调试结果如*图 2-4. 将.c 文件加载到指定位置程序调试结果*所示：

图 2-4. 将.c 文件加载到指定位置程序调试结果



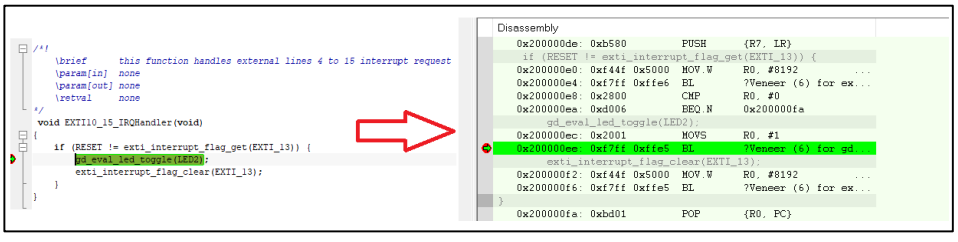
将文件 gd32f4xx_it.c 由 ROM1_regoion 加载到 sram 中（注意此处将 readwrite 放在定义的 RAM_Region），本例程通过上面 initialize by copy 加入 gd32f4xx_it.o 文件的同时，需要在启动代码 startup_gd32f4xx.s 添加__iar_init$$done,如*图 2-5. 将 startup_gd32f4xx.s 文件添加__iar_init$$done* 所示：

图 2-5. 将 startup_gd32f4xx.s 文件添加__iar_init$$done



程序调试结果*图 2-6. 将.c 文件加载到 SRAM 程序调试结果*所示：

图 2-6. 将.c 文件加载到 SRAM 程序调试结果



注意：此方法可以实现将.c 文件加载到 SRAM 起始位置处，需要加载到指定位置，可以参考章节 SDRAM 的分散加载的方法。

3. SDRAM 分散加载实现

3.1. 实现 SDRAM 的分散加载的基本原理

在 M4 内核中，我们可以通过系统总线对位于 0x2000 0000 以上的地址进行访问的并对数据和指令进行读取，但是在内核的默认配置中，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。GD32F450 的 EXMC 中 SDRAM 的地址分配为 0xC00000000-0xDFFFFFF 位于该地址段。

针对以上问题，为了在 SDRAM 中实现分散加载，具有两种解决方法：

1. 通过配置 MPU (Memory Protect Unit) 寄存器，让 0xC0000000 地址段可执行指令（本例程将采用这种实现方式）。

2. 采用内存映射的方法（通过配置 SYSCFG 寄存器将 SDRAM 的地址段映射到可执行区）。

3.2. SDRAM 分散加载的实现

在 startup_gd32f450.s 中加入下图中红色标注代码，如*图 3-1. SDRAM 分散加载实现中 startup_gd32f450.s 添加代码*所示：

图 3-1. SDRAM 分散加载实现中 startup_gd32f450.s 添加代码

```
        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
```

其中 DoIint 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，以及完成在 SDRAM 上的函数或.c 文件的拷贝，代码如*表 3-1. DoInit 函数代码*所示：

表 3-1. DoInit 函数代码

```
/*!
    \brief        initialize the sdram, setup the MPU，block data copy
    \param[in]    none
    \param[out] none
    \retval       none
*/
#pragma section = "MYBLOCK_init"
```

```
#pragma section = "MYBLOCK"
void DoInit(void)
{
    /* sdram peripheral initialize */
    exmc_synchronous_dynamic_ram_init(EXMC_SDRAM_DEVICE0);
    /* Configures the main MPU regions */
    mpu_setup();
    /* Data copy */
    char * from = __section_begin("MYBLOCK_init");
    char * to = __section_begin("MYBLOCK");
    memcpy(to, from, __section_size("MYBLOCK"));
}
```

在 GD32F450xK.icf 文件中加入如下代码，代码如*表 3-2. GD32F450xK.icf 中 SDRAM 分散加载代码*所示：

表 3-2. GD32F450xK.icf 中 SDRAM 分散加载代码

```
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 };//无 readwrite
place in SDRAM_region   { readwrite};//将 readwrite 定义在 SDRAM_region,则 IAR 中__ramfunc 指定的函数将加载到 SDRAM 中
place in SDRAM1_region   { section sdram_array};
```

上述代码将 sdram_array 段加载到 0xC0002000 起始地址，采用手动拷贝的方式，将 test.o 文件加载到 0xc0004000 起始地址，将__ramfunc 指定的函数和 gd32f4xx_it.o 加载到 0xc0000000 起始位置(这里注意与上一节.c 文件分散加载到 RAM 的区别，此处 readwrite 放在 SDRAM_region)。

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 3-3. 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 3-3. 将变量、数组、函数和文件分散加载到 SDRAM 指定位置代码

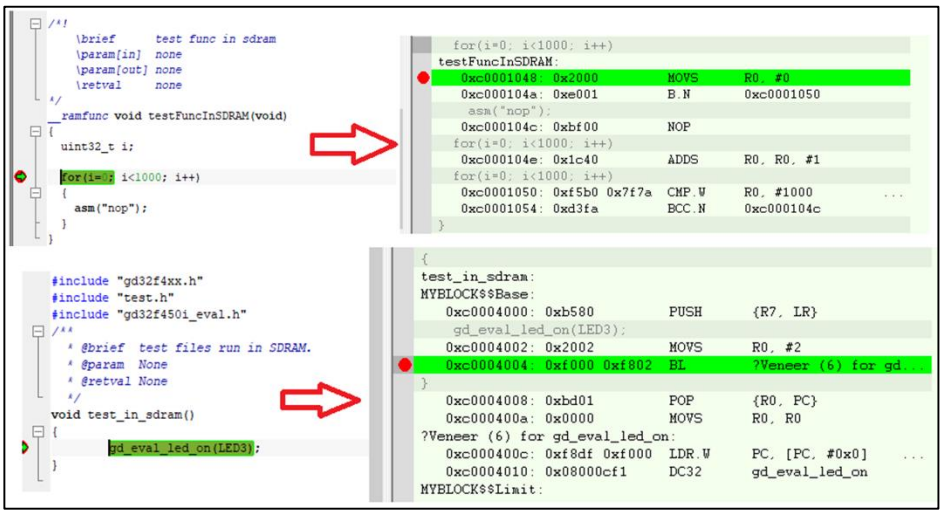
```
/* load the variable testValue_SDRAM to ram address 0xC0003000 */
uint32_t testValue_SDRAM @0xC0003000;
/* load the array test_sdram[5] to sdram address 0xc0001000 */
#pragma location =    "sdram_array"
uint32_t test_sdram[5] ={0};
/* load the function testFuncInSDRAM to sdram address 0xc0000000 */
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);
}
```

程序运行和调试结果如*表 3-4. 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 3-4. 将变量和数组加载到 SDRAM 指定位置打印结果

```
variable testValue_SDRAM address is 0xc0003000
test_sdram address is 0xc0002000
```

图 3-2. 将函数和.c 文件加载到 SDRAM 指定位置程序调试结果


/*!
\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");
}
}
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
}
#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);
}
{
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:

4. 结果

查看“GD32F4xx_ScatterLoading_v1.0.0\Project\IAR\EWARM\Debug\List\Project.map”结果如*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

```
  Section                Kind        Address    Size  Object
"A3":                                           0xe
  .text                  ro code  0x08002000     0xe  hw_config.o [1]
                                - 0x0800200e     0xe
"P4":                                          0x48
  Initializer bytes      const    0x08002010    0x48  <for P7 s4>
                                - 0x08002058    0x48
Absolute sections, part 1 of 4:               0x84f0
  .rodata                const    0x08003000  0x84f0  const-data.o [1]
                                - 0x0800b4f0  0x84f0
"A2":                                          0x14
  .funflash              ro code  0x0800f000    0x14  main.o [1]
                                - 0x0800f014    0x14
"A4":                                          0x14
  MYBLOCK_init                   0x08010000    0x14  <Block>
    Initializer bytes    const    0x08010000    0x14  <for MYBLOCK-1>
                                - 0x08010014    0x14
Absolute sections, part 2 of 4:                 0x4
  Absolute sections 2-1          0x08080000     0x4  <Init block>
    .data                inited   0x08080000     0x4  main.o [1]
                                - 0x08080004     0x4

"P3":                                          0x18
  P3 s2                          0x20001000    0x18  <Init block>
    funram               inited   0x20001000    0x18  main.o [1]
                                - 0x20001018    0x18
"P5":                                           0x4
  P5 s3                          0x20001200     0x4  <Init block>
    variable             inited   0x20001200     0x4  main.o [1]
                                - 0x20001204     0x4
"P6":                                          0x14
  array                  zero     0x20001300    0x14  main.o [1]
                                - 0x20001314    0x14
Absolute sections, part 3 of 4:                0x14
  Absolute sections 3-1          0x20007000    0x14  <Init block>
    .data                inited   0x20007000    0x14  main.o [1]
                                - 0x20007014    0x14
"P7":                                          0x58
  P7 s4                          0xc0001000    0x48  <Init block>
    Veneer               inited   0xc0001000     0x8  - Linker created -
    Veneer               inited   0xc0001008     0x8  - Linker created -
    Veneer               inited   0xc0001010     0x8  - Linker created -
    .text                inited   0xc0001018    0x30  gd32f4xx_it.o [1]
  P7 s1                          0xc0001048    0x10  <Init block>
    .textrw              inited   0xc0001048    0x10  main.o [1]
                                - 0xc0001058    0x58
"P8":                                          0x14
  sdram_array            zero     0xc0002000    0x14  main.o [1]
                                - 0xc0002014    0x14
Absolute sections, part 4 of 4:                 0x4
  .bss                   zero     0xc0003000     0x4  main.o [1]
                                - 0xc0003004     0x4
"A5":                                          0x14
  MYBLOCK                        0xc0004000    0x14  <Block>
    MYBLOCK-1                    0xc0004000    0x14  <Init block>
      .text              inited   0xc0004000     0xa  test.o [1]
      Veneer             inited   0xc000400c     0x8  - Linker created -
                                - 0xc0004014    0x14
*******************************************************************************
```

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	2021 年 04 月 30 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 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 it’s 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 it’s 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.