应用笔记

Application Note

文档编号：**AN1130**

G32R501 Keil 链接文件应用笔记

版本：**V1.1**

1 引言

在 MDK-ARM 开发环境中，链接脚本文件（Linker Script）用于指示链接器如何将代码和数据映射到目标微控制器的内存中。它们在 G32R5xx 系列 MCU 的开发中至关重要，因为它们定义了程序和数据在不同内存区域中的布局，从而确保代码能够正确运行并有效利用内存资源。

本文档就 G32R5xx 系列 MCU 开发中较为通用的几个链接脚本文件的文件内容进行说明和使用方法描述，供用户参考。

注意：以上文件位于 G32R5xx_SDK 的以下位置：

G32R5xx_SDK_VERSION#/device_support/DEVICE_GPN/common/sct

目录

2 G32R501 的内存分配情况

在正式使用链接脚本文件前，需要了解链接脚本文件的对应芯片的内存分配情况。G32R501 系列 MCU 部分产品为双核配置，部分产品为单核配置，不同配置下的芯片的内存分配情况并不一致。但他们都一样拥有以下内存区域：

- ITCM（Instruction Tightly Coupled Memory）：用于存放指令，提供高速指令执行。在单核模式下，CPU0 拥有更大的 ITCM（64KB），而在双核模式下，ITCM 大小为 48KB（CPU0）和 8KB（CPU1）。
- DTCM（Data Tightly Coupled Memory）：用于存放数据，提供高速数据访问。在单核模式下，CPU0 拥有 16KB 的 DTCM，而在双核模式下，DTCM 大小为 16KB（CPU0）和 8KB（CPU1）。
- Flash：非易失性存储器，用于存放程序代码和静态数据。无论是单核模式还是双核模式，Flash 的大小都是 640KB。
- SRAM：静态随机存储器，可用于一般数据存储。在单核模式和双核模式下，SRAM 的基地址和大小保持不变，但在双核模式下需要确保不同核心使用的区域不重叠。

在选择合适的链接脚本文件进行开发工作的前提是：了解目标芯片的内存分配情况。

2.1 G32R501 单核 MCU 的内存分配

单核模式下，CPU0 使用以下内存分配：

表格 1 G32R501 单核 MCU 的内存分配

内存区域	基地址	大小
ITCM RAM	0x00000000	64KB
DTCM RAM	0x20000000	16KB
Flash	0x08000000（ITCM：0x00100000）	640KB
SRAM1	0x20100000	8KB
SRAM2	0x20200000	8KB
SRAM3	0x20300000	32KB

2.2 G32R501 双核 MCU 的内存分配

双核模式下，CPU0 和 CPU1 共享一些内存区域，但它们也有各自的专用内存区域：

表格 2 G32R501 双核 MCU 的内存分配

内存区域	基地址	大小	备注
ITCM RAM（CPU0）	0x00000000	48KB	CPU0 的 ITCM
ITCM RAM（CPU1）	0x00000000	8KB	CPU1 的 ITCM
DTCM RAM（CPU0）	0x20000000	16KB	CPU0 的 DTCM
DTCM RAM（CPU1）	0x20000000	8KB	CPU1 的 DTCM
Flash	0x08000000（ITCM：0x00100000）	640KB	共享 Flash
SRAM1	0x20100000	8KB	共享 SRAM1
SRAM2	0x20200000	8KB	共享 SRAM2
SRAM3	0x20300000	32KB	共享 SRAM3

3 各个链接脚本文件（Linker Script）的内存分配情况

在章节 2 的描述中，G32R501 系列 MCU 的内存结构有所不同，且其单核和双核模式下的内存分配也各有差异。因此，需要根据具体的应用场景和性能需求创建不同的链接脚本文件，以优化系统性能和内存利用效率。

本章节就对各个链接脚本文件的内存分配情况及其目的进行简要描述。

下表选择了部分重要的.sct 文件进行说明，SDK 中其他没有举例说明的.sct 文件，参考自下表的说明。

表格 3 SDK 中的.sct 文件清单（节选）

序号	文件名	说明
1	g32r501dxy_cpu0_cbus_flash.sct	代码运行及下载位置：0x08000000，使用 CBUS 接口
2	g32r501dxy_cpu0_itcm_flash.sct	代码运行及下载位置：0x00100000，使用 ITCM 接口
3	g32r501dxy_cpu1_cbus_flash.sct	代码运行及下载位置：0x08040000，使用 CBUS 接口
4	g32r501dxy_cpu1_itcm_flash.sct	代码运行及下载位置：0x00140000，使用 ITCM 接口
5	g32r501dxy_cpu0_cbus_flash_secure.sct	双核 CPU0 安全启动.sct 示例文件
6	g32r501dxy_cpu1_cbus_flash_secure.sct	双核 CPU1 安全启动.sct 示例文件
7	g32r501dxy_cpu0_itcm_ram.sct	代码运行及下载位置：0x00000000，使用 ITCM 接口
8	g32r501xc_cbus_flash.sct	代码运行及下载位置：0x08000000，使用 CBUS 接口
9	g32r501xc_itcm_flash.sct	代码运行及下载位置：0x08000000，使用 CBUS 接口
10	g32r501xy_cbus_flash.sct	代码运行及下载位置：0x08000000，使用 CBUS 接口
11	g32r501xy_cbus_flash_secure.sct	单核安全启动.sct 示例文件
12	g32r501xy_itcm_flash.sct	代码运行及下载位置：0x00100000，使用 ITCM 接口
13	g32r501xy_itcm_ram.sct	代码运行及下载位置：0x00000000，使用 ITCM 接口

4 在 MDK 中使用链接脚本文件（.sct）

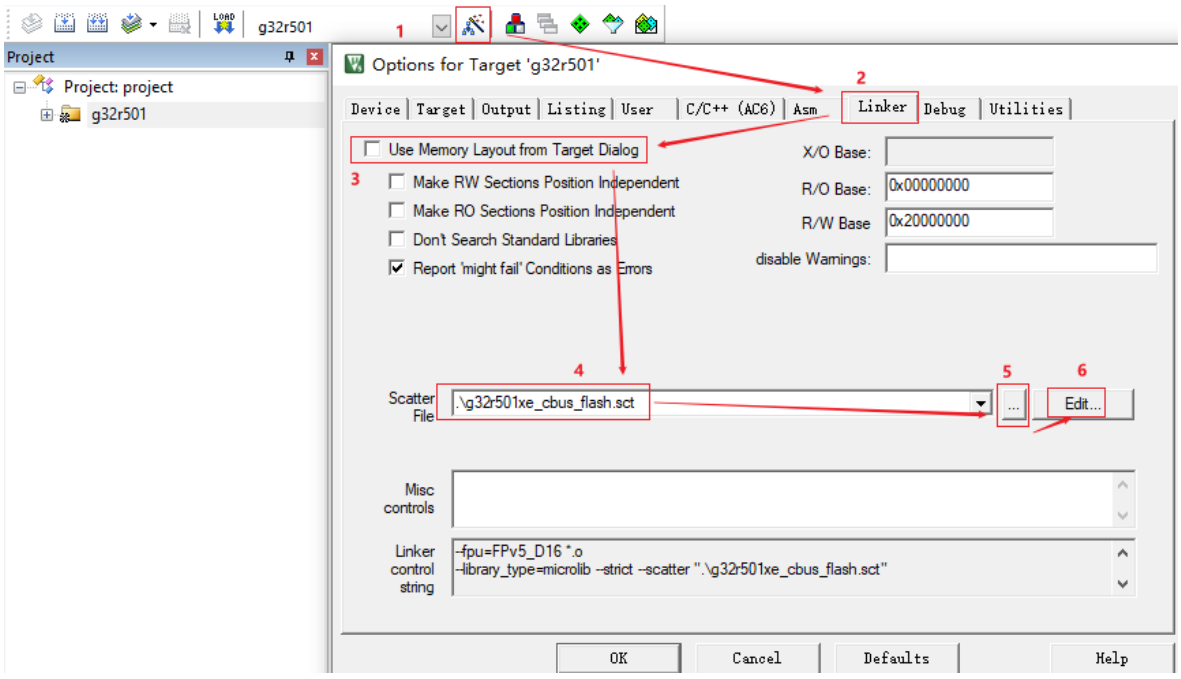
在 MDK 开发环境中，使用链接脚本文件（.sct）可以有效地管理和配置微控制器的内存布局。下面是如何在 MDK 中选择和自定义链接脚本文件的步骤和详细说明。

4.1 选择链接脚本文件（.sct）

将需要选择的“*.sct”文件复制至相应的 MDK 项目后，可按照如下步骤选择在工程中使用该文件。

1. 打开 MDK 项目：进入 MDK 项目。
2. 进入项目设置窗口：点击菜单栏中的“Project”或在工具栏直接选择“Options for Target”。
3. 选择“Linker”选项卡：在打开的项目设置窗口中，切换到“Linker”选项卡。
4. 选择“…”选项：在“…”弹出的文件选择框中，点击浏览按钮选择相应的“*.sct”文件。
5. 选择“Edit”选项：可编辑选择好的“*.sct”文件。
6. 保存配置：点击“OK”按钮。

图 1 如何选择或编辑“*.sct”文件



4.2 自定义配置

选择“*.sct”文件后，可以在 Configuration Wizard 窗口中进行以下自定义配置，以满足具体的应用需求。

图 2 选择 Configuration Wizard 编辑窗口

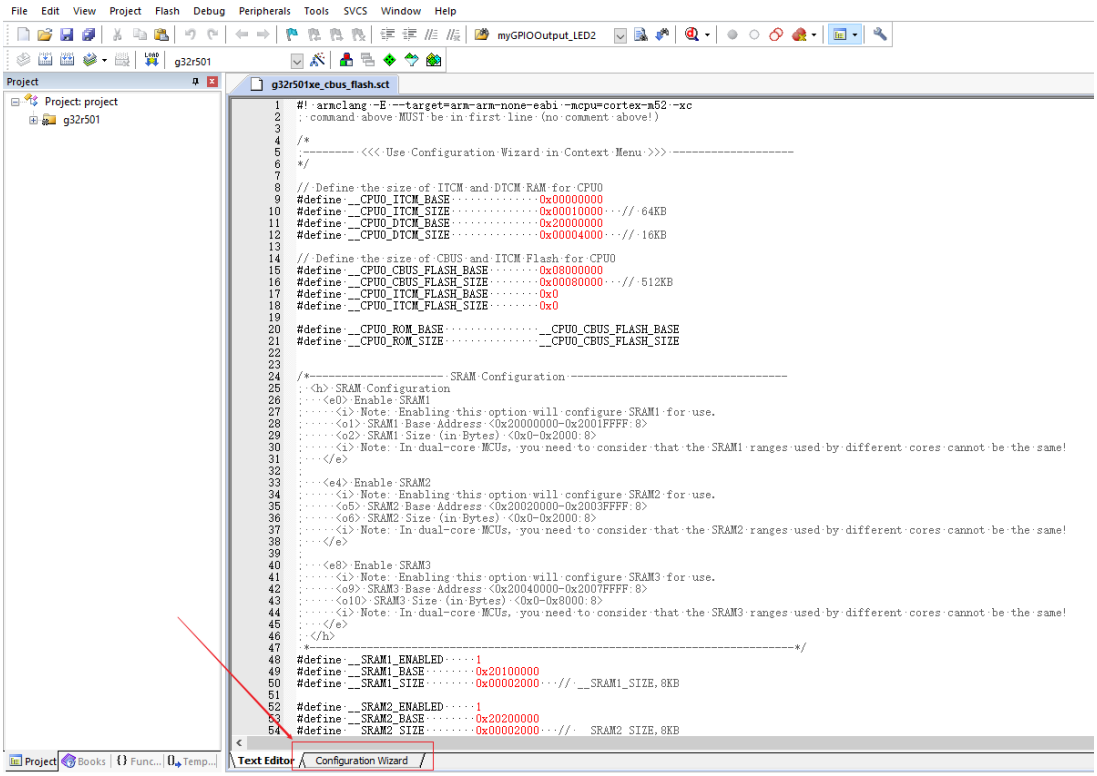
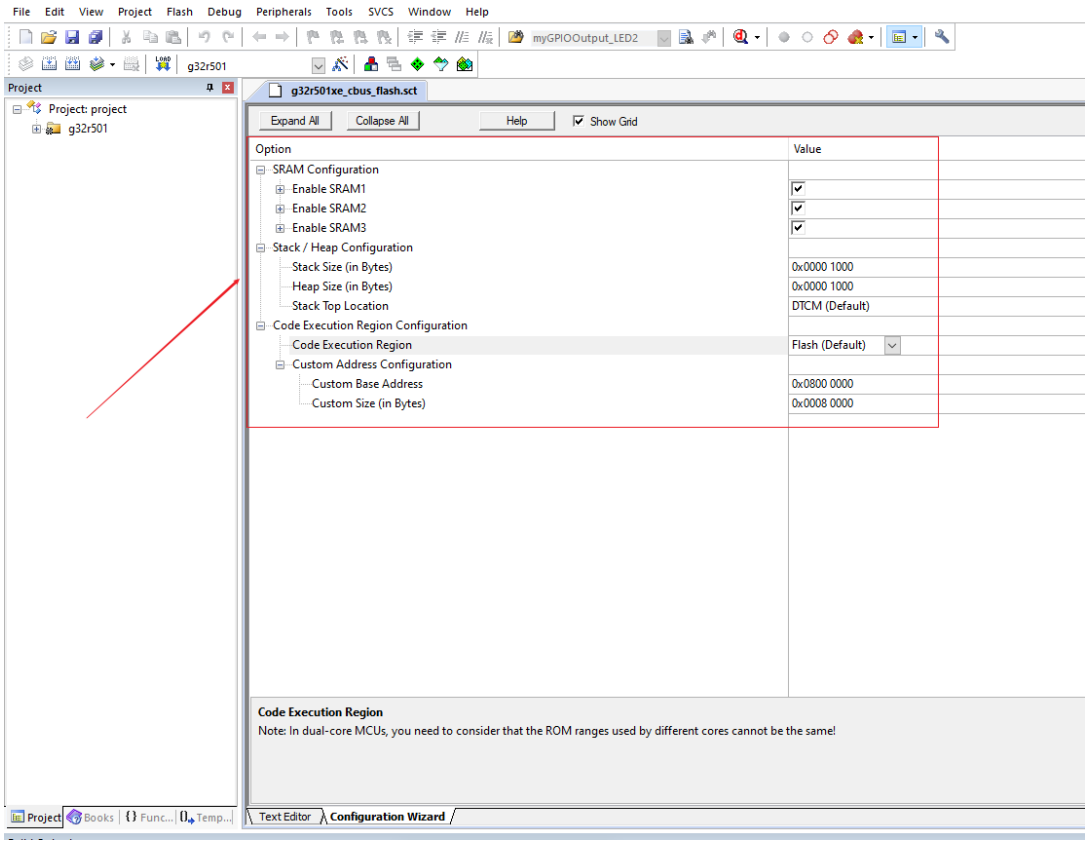


图 3 Configuration Wizard 编辑窗口



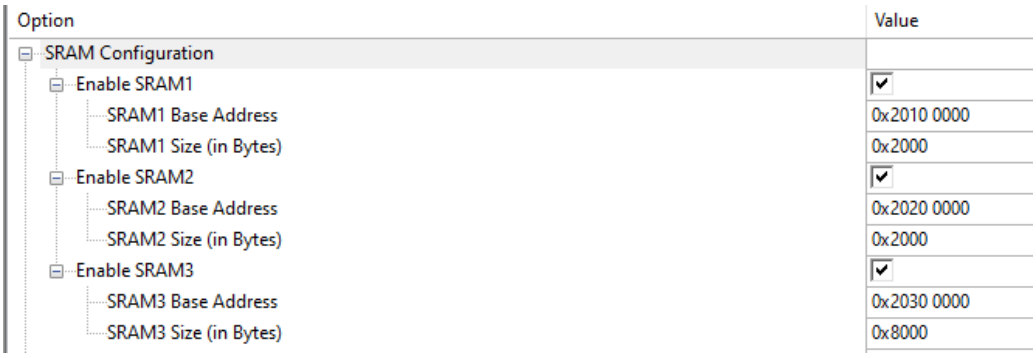
4.2.1 SRAM 配置

可在该配置中启用或禁用不同的 SRAM 区域。

- SRAM1：配置基地址和大小，典型大小为 8KB。
- SRAM2：配置基地址和大小，典型大小为 8KB。
- SRAM3：配置基地址和大小，典型大小为 32KB。

注意：为避免内存冲突，用户需要确保在双核模式下，不同核心使用的 SRAM 区域不重叠。

图 4 启用或禁用不同的 SRAM 区域



4.2.2 栈/堆配置

可在该配置中设置栈和堆的大小和位置（DTCM、SRAM1、SRAM2 或 SRAM3）。

- 默认栈位置：默认情况下，栈存放在芯片的 DTCM 中。

图 5 设置栈和堆的大小和位置

注意：DMA 访问的数据不应放在 DTCM 中，应该放置在单独的区域如 dma_data 段中，并存放在其他位置（SRAM1、SRAM2、SRAM3）。

4.2.3 代码执行区域配置

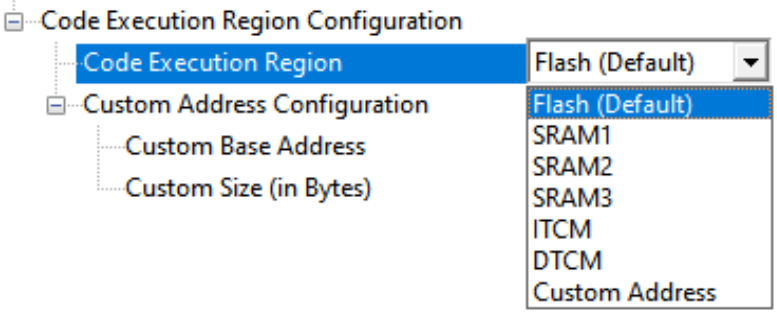
可在该配置中选择代码的执行区域（Flash、SRAM、ITCM、DTCM 或自定义地址）。如果选择自定义地址，需自行确认设置的区域内容是否有效。

代码执行区域选项：

- Flash：默认的代码存储和执行区域。
- SRAM：可选择 SRAM1、SRAM2 或 SRAM3 作为代码执行区域，以提高执行速度。

- ITCM：用于存放关键代码，提供更快的执行速度。
- DTCM：用于存放数据，提供快速数据访问。
- 自定义地址：用户可以手动设置代码执行区域。

图 6 选择代码的执行区域



注意：以上配置在修改后点击保存，在下一次编译程序时生效。

5 自行编写链接脚本文件（.sct）

用户可以基于 MDK-ARM 的链接脚本文件（.sct）格式自行编写适合项目的链接脚本文件（.sct）。相关信息，请参见 MDK-ARM 的链接脚本文件（.sct）格式要求和章节 2 中的 G32R501 的内存分配情况。

5.1 散列文件语法规则

一个散列文件包含一个或多个加载区域，每个加载区域可以包含一个或多个执行区域。示例如下：

```
;加载区域 1
LOAD_ROM_1 0x0000
{
    ;执行区域 1
    EXEC_ROM_1 0x0000
    {
        ;输入节区 1
        program1.o (+RO)
    }
    ;执行区域 2
    DRAM 0x18000 0x8000
    {
        ;输入节区 2
        program1.o (+RW,+ZI)
    }
}
```

- 加载区域：

  加载区域的标准形式为：

```
//方括号中的为选填内容
加载域名 (基地址 | ("+"地址偏移(+<offset>))) [属性列表] [最大容量]
```

```
{
    执行区域
}
```

- 加载域名：用于链接器识别不同的加载域。
- 基地址：指定加载域中代码和数据的起始地址，地址偏移为可选内容，偏移地址必须满足对齐要求。<offset>表示地址位于前一个加载区域的末尾之后<offset>字节的位置。如果这是第一个加载区域，加载区域的起始地址不依赖于任何先前的加载区域，<offset>表示基地址距离零点<offset>字节。
- 属性：指定加载域的属性，常见的属性如下：

  ABSOLUTE----------绝对地址(默认)

  ALIGN <alignment>----------地址对齐

  NOCOMPRESS----------指定某个加载区域内容在镜像文件中不被压缩
- 最大容量：指定加载域的最大大小。如果指定了可选的 max_size 值，则如果该区域分配给它的字节数超过 max_size，armlink 将生成错误。

- 执行区域：

  执行区域的标准形式为：

```
//方括号中的为选择内容
执行域名 (基地址 | ("+"地址偏移(+<offset>))) [属性] [最大容量 | <length>]
{
    输入节区
}
```

- 执行域名：链接器用于标识不同的执行区。
- 基地址：与加载域类似。
- 属性：指定加载域的属性，常见的属性如下：

  ABSOLUTE----------绝对地址(默认)

  ALIGN <alignment>----------地址对齐

  NOCOMPRESS----------指定某个加载区域内容在镜像文件中不被压缩

  EMPTY [-]<length>----------在执行区域中保留一个给定大小的空内存块，通常由堆栈使用。在具有 EMPTY 属性的区域中不能放置节选择器。 <length> 表示在内存中

向下增长的堆栈长度。如果长度为负值，则 <base_address> 被视为区域的结束地址。

- 最大容量：与加载域类似。

- 输入节区：

  输入节区的标准形式为：

输入节区描述
常见形式为：
模块选择模式(+输入节区属性)
模块选择模式(输入符号样式)
模块选择模式(输入节区样式)

- 模块选择模式：
  - 可以使用通配符字符” * ”与.ANY，使用.ANY 可以选择所有的.o 文件与.lib 文件，其中.ANY 选择的文件是优先级最低的。
  - 使用”.o”可以选择所有.o 文件。
  - 使用”.lib”可以选择所有.lib 文件。
- 输入节区属性：可以在模块选择模式后加入输入节区属性，每个节区属性描述符需添加”+”，使用空格或”,”分开，如：.ANY(+RO +ZI)。

表格 4 属性段描述

属性描述符	说明
RO-CODE/CODE	只读代码段
RO-DATA/CONST	只读数据段
RO/TEXT	RO-CODE+ RO-DATA
RW-CODE	可读写代码段
RW-DATA	可读写数据段
RW/DATA	RW-CODE+ RW-DATA
XO	只可执行的区域
ZI/BSS	初始化为 0 的可读写数据段
ENTRY	节区的入口

此外，还有以下伪属性被识别：

- FIRST.
- LAST.

- 输入符号样式：通过符号来选择输入节区，符号样式需要使用”:gdef:”前缀。

- 输入节区样式：通过输入节区样式可以选择控制的节区。
  - 例如：InRoot$$Sections：用于定义特定段的符号，用于将某些节放置在特定的内存区域。
  - RESET：通常定义程序的复位向量，当系统上电或复位时，处理器会跳转到该地址执行启动代码。

5.2 应用场景修改示例

5.2.1 使用自定义链接宏

在文件 g32r501.h 中，定义了一些宏，来控制 sdk 中代码的内存布局，如图：

图 7 自定义链接宏

```
// Define macros to place functions or variables in specific memory sections
// Usage:
// - Use these macros before function declarations or variable definitions to place them in specific sections.
// - Ensure that the linker script (MDK) or ICF file (IAR), or LD script (GCC) includes the corresponding sections.

#if defined (__CC_ARM) || defined (__ICCARM__) || defined (__GNUC__) || defined (__ARMCC_VERSION)
  #define SECTION_ITCM_INSTRUCTION  __attribute__((section("itcm.instruction")))
  #define SECTION_ITCM_RAMFUNC      __attribute__((section("itcm.ramfunc")))
  #define SECTION_DTCM_DATA         __attribute__((section("dtcm.data")))
  #define SECTION_DTCM_BSS          __attribute__((section("dtcm.bss")))
  #define SECTION_SRAM1_SHARE_DATA  __attribute__((section("sram1.share_data")))
  #define SECTION_SRAM2_SHARE_DATA  __attribute__((section("sram2.share_data")))
  #define SECTION_SRAM3_SHARE_DATA  __attribute__((section("sram3.share_data")))
#else
  #error "Unsupported compiler: no section macros can be defined."
#endif
```

表格 5 自定义链接段名基地址

特定段名	链接基地址
itcm.instruction	ITCM(0x00000000)
itcm.ramfunc	ITCM(0x00000000)
dtcm.data	DTCM(0x20000000)
dtcm.bss	DTCM(0x20000000)
sram1.share_data	SRAM1(0x20000000)
sram2.share_data	SRAM2(0x20200000)
sram3.share_data	SRAM3(0x20300000)

用户可以用 g32r501.h 文件中的宏来指定函数和变量存放在 ITCM、DTCM、SRAM 中的位置。若该文件中的宏无法满足用户使用，可参考以下示例自行编写链接脚本文件(.sct)，添加自定义的链接段。

5.2.2 指定函数、变量链接区域

示例 1：将变量 Data_Ex 放在 DTCM 中

- 确定函数、变量将要存放的内存区域
  - Data_Ex-------0x20000000 – 0x2000C000(DTCM)
- 在指定的内存区域添加新输入段
  - 在属于 DTCM 的执行域添加新输入段，自定义段名为 data.ex

```
RW_RAM_CPU0_DTCM __CPU0_DTCM_BASE __CPU0_DTCM_SIZE {
        ;新添加的输入段
        .ANY (data.ex)
        .ANY (+RW +ZI)
}
```

- 在定义实现时，通过__attribute__((section("xxx")))来控制

```
__attribute__((section("data.ex ")))
uint32_t Data_Ex;
```

5.2.3 指定函数、变量链接地址

用户在应用开发过程中，有时需要把某个函数或变量存放到指定的地址中。Keil 一般有两种方法，以下是两种方法的详细说明。

示例 2：将变量 Data_Ex 变量存放到指定的 0x20300000 地址中

- 方法 1：在定义实现时，通过__attribute__((__used__, section(".ARM.__at_0x20300000")))来控制。

```
__attribute__((__used__, section(".ARM.__at_0x20300000")))
const uint32_t Data_Ex;
```

注意：该方法只能应用于由 const 关键字限定的只读变量，如果应用于可读可写的变量，将会产生警告。

- 方法 2：修改.sct 文件，通过.sct 文件指定存储的地址。
  - 首先修改.sct 文件，自定义一个执行域，起始地址就是用户想要将该变量链接到的指定地址，然后再自定义一个输入段名称。

```
RW_RAM 0x20001000 0x400 {
        ;新添加的输入段
        .ANY (data.ex)
}
```

- 在定义实现时，可以使用__attribute__((section("xxx")))来控制变量链接到在链接脚本.sct文件中上面新增的执行区域。

```
__attribute__((section("data.ex ")))
uint32_t Data_Ex;
```

这样就可以把定义的变量链接到指定的地址。

5.2.4 应用程序划分区域

为了实现更好的内存管理、性能优化，使用分散加载文件将应用程序划分区域是一种重要策略。

示例：

```
#define __RO_BASE    __CPU0_ROM_BASE
#define __RO_SIZE    __CPU0_ROM_SIZE

#define __CPU0_ROM_BASE              __CPU0_CBUS_FLASH_BASE
#define __CPU0_ROM_SIZE              __CPU0_CBUS_FLASH_SIZE

#define __CPU0_CBUS_FLASH_BASE       0x08000000
#define __CPU0_CBUS_FLASH_SIZE       0x00050000   // 320KB

#define __SRAM3_BASE                 0x20300000
#define __SRAM3_SIZE                 0x00008000   // __SRAM3_SIZE,32KB

#define __SRAM1_BASE                 0x20100000
#define __SRAM1_SIZE                 0x00002000   // __SRAM1_SIZE,8KB

#define __CPU0_DTCM_BASE             0x20000000
#define __CPU0_DTCM_SIZE             0x00004000   // 16KB

LR_ROM __RO_BASE __RO_SIZE {
  ER_ROM __RO_BASE __RO_SIZE {
   *.o (RESET, +First)
```

```
   *(InRoot$$Sections)
   .ANY (+RO)
   .ANY (+XO)
  }

  RW_RAM_CPU0_DTCM __CPU0_DTCM_BASE __CPU0_DTCM_SIZE {
    .ANY (dtcm.data)
    .ANY (dtcm.bss)
    .ANY (+RW +ZI)
  }

  RW_RAM_SRAM1 __SRAM1_BASE __SRAM1_SIZE {
    .ANY (sram1.share_data)
    .ANY (+RW +ZI)
  }
}

LR_ROM_SRAM3 __SRAM3_BASE __SRAM3_SIZE {
  ER_ROM_SRAM3 __SRAM3_BASE __SRAM3_SIZE {
    .ANY (+CONST)
    .ANY (share.data)
  }
}
```

这个链接器脚本的结构清晰地定义了两个加载区域，分别用于存放常量数据和只读代码。

- LR_ROM_SRAM3 的内存区域，起始地址为 __SRAM3_BASE，大小为 __SRAM3_SIZE。这个区域通常用于存放特定的只读数据。
- LR_ROM __RO_BASE __RO_SIZE 定义了一个名为 LR_ROM 的只读内存区域，起始地址为 __RO_BASE，大小为 __RO_SIZE。

5.2.4.1 双核例程链接文件

双核例程的.sct 示例文件位于 SDK\device_support\g32r501\common\sct\中的 g32r501dxy_cpu0_cbus_flash.sct 与 g32r501dxy_cpu1_cbus_flash.sct。

在双核例程中，Flash 一分为 2，若用户想修改两个核的 Flash 大小分配，只需修改示例文件中的宏定义即可。

示例：修改 CPU0 的 Flash 大小为：384KB，CPU1 的 flash 大小为：256KB

```
// Define the size of CBUS and ITCM Flash for CPU0/CPU1
#define __CPU0_CBUS_FLASH_BASE          0x08000000
#define __CPU0_CBUS_FLASH_SIZE          0x00060000    // 修改部分，384KB
#define __CPU1_CBUS_FLASH_BASE          0x08060000    // 修改部分
#define __CPU1_CBUS_FLASH_SIZE          0x00040000    // 修改部分，256KB
#define __CPU0_ITCM_FLASH_BASE          0x0
#define __CPU0_ITCM_FLASH_SIZE          0x0
#define __CPU1_ITCM_FLASH_BASE          0x0
#define __CPU1_ITCM_FLASH_SIZE          0x0

#define __CPU0_ROM_BASE                 __CPU0_CBUS_FLASH_BASE
#define __CPU0_ROM_SIZE                 __CPU0_CBUS_FLASH_SIZE
#define __CPU1_ROM_BASE                 __CPU1_CBUS_FLASH_BASE
#define __CPU1_ROM_SIZE                 __CPU1_CBUS_FLASH_SIZE
```

5.2.4.2 安全启动例程链接文件

对于需要 CPU 在 Flash 读取的数据是不允许加密的，在设计安全启动例程的.sct 文件时，需要把只读数据分离开，这样用户可以明确知道只读数据所存放的 Flash 区域，然后确保该区域不被设置为加密权限。

双核例程的.sct 示例文件位于 SDK\device_support\g32r501\common\sct\中的 g32r501dxy_cpu0_cbus_flash_secure.sct。有关如何将 CPU 在 Flash 中读取的数据进行分隔的具体设计，用户可以参考该文件。

对于该文件，不允许加密的区域，默认是存放到 0x08050000 的起始地址处。若用户想修改该区域，只需要修改 g32r501dxy_cpu0_cbus_flash_secure.sct 文件的起始地址宏定义。

例如，将不允许加密的区域的起始地址和大小进行修改，由 SDK 提供的.sct 文件默认的起始地址 0x08050000 和大小 64KB，起始地址和大小分别修改为 0x08020000 和 128KB。

```
// Define the size of ITCM and DTCM RAM for CPU0
#define __CPU0_ITCM_BASE                    0x00000000
#define __CPU0_ITCM_SIZE                    0x0000C000      // 48KB
#define __CPU0_DTCM_BASE                    0x20000000
#define __CPU0_DTCM_SIZE                    0x00004000      // 16KB

// Define the size of CBUS and ITCM secure Flash for CPU0,
// This area allows users to be set with encryption permissions.
#define __CPU0_CBUS_FLASH_SECURE_BASE       0x08000000
#define __CPU0_CBUS_FLASH_SECURE_SIZE       0x00020000      // 128KB
#define __CPU0_ITCM_FLASH_SECURE_BASE       0x0
#define __CPU0_ITCM_FLASH_SECURE_SIZE       0x0

// Define the size of CBUS and ITCM unsecure Flash for CPU0,
// This area does not allow users to be set with encryption permissions.
#define __CPU0_CBUS_FLASH_UNSECURE_BASE     0x08020000      // 不允许加密区域修改到 0x08020000
#define __CPU0_CBUS_FLASH_UNSECURE_SIZE     0x00020000      // 128KB
#define __CPU0_ITCM_FLASH_UNSECURE_BASE     0x0
#define __CPU0_ITCM_FLASH_UNSECURE_SIZE     0x0

// Define the size of CBUS Flash for CPU1
#define __CPU1_CBUS_FLASH_BASE              0x08060000
#define __CPU1_CBUS_FLASH_SIZE              0x00040000      // 256KB
```

6 版本历史

表格 6 文件版本历史

日期	版本	变更历史
2025.01	1.0	新建
2025.04	1.1	新增散列文件语法规则说明

声明

本手册由珠海极海半导体有限公司（以下简称“极海”）制订并发布，所列内容均受商标、著作权、软件著作权相关法律法规保护，极海保留随时更正、修改本手册的权利。使用极海产品前请仔细阅读本手册，一旦使用产品则表明您（以下称“用户”）已知悉并接受本手册的所有内容。用户必须按照相关法律法规和本手册的要求使用极海产品。

1、权利所有

本手册仅应当被用于与极海所提供的对应型号的芯片产品、软件产品搭配使用，未经极海许可，任何单位或个人均不得以任何理由或方式对本手册的全部或部分内容进行复制、抄录、修改、编辑或传播。

本手册中所列带有“®”或“™”的“极海”或“Geehy”字样或图形均为极海的商标，其他在极海产品上显示的产品或服务名称均为其各自所有者的财产。

2、无知识产权许可

极海拥有本手册所涉及的全部权利、所有权及知识产权。

极海不应因销售、分发极海产品及本手册而被视为将任何知识产权的许可或权利明示或默示地授予用户。

如果本手册中涉及任何第三方的产品、服务或知识产权，不应被视为极海授权用户使用前述第三方产品、服务或知识产权，也不应被视为极海对第三方产品、服务或知识产权提供任何形式的保证，包括但不限于任何第三方知识产权的非侵权保证，除非极海在销售订单或销售合同中另有约定。

3、版本更新

用户在下单购买极海产品时可获取相应产品的最新版的手册。

如果本手册中所述的内容与极海产品不一致的，应以极海销售订单或销售合同中的约定为准。

4、信息可靠性

本手册相关数据经极海实验室或合作的第三方测试机构批量测试获得，但本手册相关数据难免会出现校正笔误或因测试环境差异所导致的误差，因此用户应当理解，极海对本手册中可能出现的该等错误无需承担任何责任。本手册相关数据仅用于指导用户作为性能参数参照，不构成极海对任何产品性能方面的保证。

用户应根据自身需求选择合适的极海产品，并对极海产品的应用适用性进行有效验证和测试，以确认极海产品满足用户自身的需求、相应标准、安全或其它可靠性要求；若因用户未充分对极海产品进行有效验证和测试而致使用户损失的，极海不承担任何责任。

5、合规要求

用户在使用本手册及所搭配的极海产品时，应遵守当地所适用的所有法律法规。用户应了解产品可能受到产品供应商、极海、极海经销商及用户所在地等各国有关出口、再出口或其它法律的限制，用户（代表其本身、子公司及关联企业）应同意并保证遵守所有关于取得极海产品及/或技术与直接产品的出口和再出口适用法律与法规。

6、免责声明

本手册由极海“按原样”（as is）提供，在适用法律所允许的范围内，极海不提供任何形式的明示或暗示担保，包括但不限于对产品适销性和特定用途适用性的担保。

极海产品并非设计、授权或担保适合用于军事、生命保障系统、污染控制或有害物质管理系统中的关键部件，亦非设计、授权或担保适合用于在产品失效或故障时可导致人员受伤、死亡、财产或环境损害的应用。

如果产品未标明“汽车级”，则表示不适用于汽车应用。如果用户对产品的应用超出极海提供的规格、应用领域、规范，极海不承担任何责任。

用户应该确保对产品的应用符合相应标准以及功能安全、信息安全、环境标准等要求。用户对极海产品的选择和使用负全部的责任。对于用户后续在针对极海产品进行设计、使用的过程中所引起的任何纠纷，极海概不承担责任。

7、责任限制

在任何情况下，除非适用法律要求或书面同意，否则极海和/或以“按原样”形式提供本手册及产品的任何第三方均不承担损害赔偿责任，包括任何一般、特殊因使用或无法使用本手册及产品而产生的直接、间接或附带损害（包括但不限于数据丢失或数据不准确，或用户或第三方遭受的损失），这涵盖了可能导致的人身安全、财产或环境损害等情况，对于这些损害极海概不承担责任。

8、适用范围

本手册的信息用以取代本手册所有早期版本所提供的信息。