## 零死角玩转STM32



# MDK的编译过程及文 件类型全解

淘宝: firestm32.taobao.com

论坛: www.chuxue123.com



扫描进入淘宝店铺

## 主讲内容



01 编译过程

02 程序的组成、存储与运行

03 编译工具链

04 MDK工程的文件类型

05 实验:自动分配变量到外部SDRAM

96 实验:优先使用内部SRAM并

分配堆到SDRAM



### sct分散加载文件的格式与应用

### 1.sct分散加载文件简介

当工程按默认配置构建时,MDK会根据我们选择的芯片型号,获知芯片的内部FLASH及内部SRAM存储器概况,生成一个以工程名命名的后缀为\*.sct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,因此我们通过修改该文件可以定制具体节区的存储位置。



### 1.sct分散加载文件简介

例如可以设置源文件中定义的所有变量自动按地址分配到外部SDRAM,这样就不需要再使用关键字"\_\_attribute\_\_"按具体地址来指定了;

利用它还可以控制代码的加载区与执行区的位置,例如可以把程序代码存储到单位容量价格便宜的NAND-FLASH中,但在NAND-FLASH中的代码是不能像内部FLASH的代码那样直接提供给内核运行的,这时可通过修改分散加载文件,把代码加载区设定为NAND-FLASH的程序位置,而程序的执行区设定为SDRAM中的位置,这样链接器就会生成一个配套的分散加载代码,该代码会把NAND-FLASH中的代码加载到SDRAM中,内核再从SDRAM中运行主体代码,大部分运行Linux系统的代码都是这样加载的。



### 2.分散加载文件的格式

打开MDK默认使用的sct文件,在Output目录下可找到"多彩流水灯.sct",该文件记录的内容:

```
*******************************
 2 ; *** Scatter-Loading Description File generated by uVision ***
 3 • ******************
 5 LR_IROM1 0x08000000 0x00100000 { ; 注释:加载域,基地址 空间大小
    ER IROM1 0x08000000 0x00100000 { ; 注释:加载地址 = 执行地址
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
10
   RW IRAM1 0x20000000 0x00030000 { ; 注释:可读写数据
11
12
    .ANY (+RW + ZI)
13
14 }
15
```



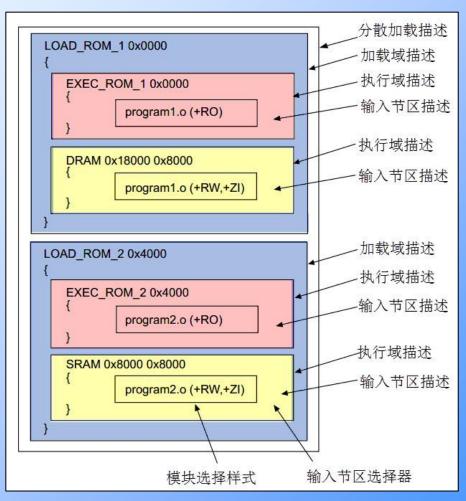
### 2.分散加载文件的格式

在默认的sct文件配置中仅分配了Code、RO-data、RW-data及ZI-data 这些大区域的地址,链接时各个节区(函数、变量等)直接根据属性排列到具体 的地址空间。

sct文件中主要包含描述加载域及执行域的部分,一个文件中可包含有多个加载域,而一个加载域可由多个部分的执行域组成。同等级的域之间使用花括号"{}"分隔开,最外层的是加载域,第二层"{}"内的是执行域,其整体结构如下:



### 2.分散加载文件的格式



分散加载文件的整体结构



### 加载域

sct文件的加载域格式如下:

```
1 //方括号中的为选填内容
2 加载域名 (基地址 | ("+" 地址偏移)) [属性列表] [最大容量]
3 "{"
4 执行区域描述+
5 "}"
```

- 加载域名: 名称,在map文件中的描述会使用该名称来标识空间。如本例中只有 一个加载域,该域名为LR IROM1。
- 基地址+地址偏移: 这部分说明了本加载域的基地址,可以使用+号连接一个地址偏移,算进基地址中,整个加载域以它们的结果为基地址。如本例中的加载域基地址为0x08000000,刚好是STM32内部FLASH的基地址。



### 2.分散加载文件的格式

sct文件的加载域格式如下:

```
1 //方括号中的为选填内容
2 加载域名 (基地址 | ("+" 地址偏移)) [属性列表] [最大容量]
3 "{"
4 执行区域描述+
5 "}"
```

- **属性列表:** 属性列表说明了加载域的是否为绝对地址、**N**字节对齐等属性,该配置是可选的。本例中没有描述加载域的属性。
- 最大容量:最大容量说明了这个加载域可使用的最大空间,该配置也是可选的,如果加上这个配置后,当链接器发现工程要分配到该区域的空间比容量还大,它会在工程构建过程给出提示。本例中的加载域最大容量为0x00100000,即1MB,正是本型号STM32内部FLASH的空间大小。



### 输入节区描述

配合加载域及执行域的配置,在相应的域配置"输入节区描述"即可控制该节区存储到域中,其格式如下:

- 1 //除模块选择样式部分外,其余部分都可选选填
- 2 模块选择样式"("输入节区样式",""+"输入节区属性")"
- 3 模块选择样式"("输入节区样式",""+"节区特性")"
- 4
- 5 模块选择样式"("输入符号样式",""+"节区特性")"
- 6 模块选择样式"("输入符号样式",""+"输入节区属性")"
- 模块选择样式:模块选择样式可用于选择o及lib目标文件作为输入节区,它可以直接使用目标文件名或"\*"通配符,也可以使用".ANY"。例如,使用语句"bsp\_led.o"可以选择bsp\_led.o文件,使用语句"\*.o"可以选择所有o文件,使用"\*.lib"可以选择所有lib文件,使用"\*"或".ANY"可以选择所有的o文件及lib文件。其中".ANY"选择语句的优先级是最低的,所有其它选择语句选择完剩下的数据才会被".ANY"语句选中。



### 输入节区描述

输入节区样式:在目标文件中会包含多个节区或符号,通过输入节区样式可以 选择要控制的节区。

示例文件中"(RESET, +First)"语句的RESET就是输入节区样式,它选择了名为RESET的节区,并使用后面介绍的节区特性控制字"+First"表示它要存储到本区域的第一个地址。示例文件中的"\*(InRoot\$\$Sections)"是一个链接器支持的特殊选择符号,它可以选择所有标准库里要求存储到root区域的节区,如\_\_\_main.o、\_\_scatter\*.o等内容。

• 输入符号样式: 同样地,使用输入符号样式可以选择要控制的符号,符号样式需要使用":gdef:"来修饰。例如可以使用"\*(:gdef:Value\_Test)"来控制选择符号"Value\_Test"。



### 输入节区描述

• **输入节区属性**:通过在模块选择样式后面加入输入节区属性,可以选择样式中不同的内容,每个节区属性描述符前要写一个"+"号,使用空格或","号分隔开,可以使用的节区属性描述符:

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



### 输入节区描述

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

例如,示例文件中使用".ANY(+RO)"选择剩余所有节区RO属性的内容都分配到执行域ER\_IROM1中,使用".ANY(+RW +ZI)"选择剩余所有节区RW及ZI属性的内容都分配到执行域RW\_IRAM1中。



### 输入节区描述

• 节区特性: 节区特性可以使用"+FIRST"或"+LAST"选项配置它要存储到的位置,FIRST存储到区域的头部,LAST存储到尾部。通常重要的节区会放在头部,而CheckSum(校验和)之类的数据会放在尾部。

例如示例文件中使用"(RESET,+First)"选择了RESET节区,并要求 把它放置到本区域第一个位置,而RESET是工程启动代码中定义的向量表,该 向量表中定义的堆栈顶和复位向量指针必须要存储在内部FLASH的前两个地址, 这样STM32才能正常启动,所以必须使用FIRST控制它们存储到首地址。

```
; Vector Table Mapped to Address 0 at Reset
                AREA
                        RESET, DATA, READONLY
                          Vectors
                EXPORT
                        Vectors End
                EXPORT
                          Vectors Size
                EXPORT
              DCD
                        initial sp
                                                  ; Top of Stack
 Vectors
                        Reset Handler
                                                    : Reset Handler
                DCD
                        NMI Handler
                                                    ; NMI Handler
                DCD
```



总的来说,前面的sct示例文件配置如下:程序的加载域为内部FLASH的0x08000000,最大空间为0x00100000;程序的执行基地址与加载基地址相同,其中RESET节区定义的向量表要存储在内部FLASH的首地址,且所有o文件及lib文件的RO属性内容都存储在内部FLASH中,程序执行时RW及ZI区域都存储在以0x20000000为基地址,大小为0x00030000的空间(192KB),这部分正好是STM32内部主SRAM的大小。

链接器根据sct文件链接,链接后各个节区、符号的具体地址信息可以 在map文件中查看。



### 3.通过MDK配置选项来修改sct文件

了解sct文件的格式后,可以手动编辑该文件控制整个工程的分散加载 配置,但sct文件格式比较复杂,所以MDK提供了相应的配置选项可以方便地修 改该文件,这些选项配置能满足基本的使用需求,

#### 选择sct文件的产生方式





#### 3.通过MDK配置选项来修改sct文件

了解sct文件的格式后,可以手动编辑该文件控制整个工程的分散加载 配置,但sct文件格式比较复杂,所以MDK提供了相应的配置选项可以方便地修 改该文件,这些选项配置能满足基本的使用需求,

#### 选择sct文件的产生方式

使用MDK生成还是使用用户自定义的sct文件。在MDK的"Options for Target->Linker->Use Memory Layout from Target Dialog"选项即可配置该选择:



#### 选择sct文件的产生方式

Use Men	nory Layout from Target Dialog	X/O Base:		
☐ Make	e RW Sections Position Independent	R/O Base:	0x08000000	
	e RO Sections Position Independent	R/W Base	0x20000000	
	: Search Standard Libraries ort 'might fail' Conditions as Errors	disable Wamings:		11
Scatter File	\.\Output\多彩流水灯.sct			Edit
	\.\Output\多彩流水灯.sct			Edit

vice   Tar	get   Output   Listing   Vser	C/C++ Asm L	inker.	Debug   Utilities	1
Use Men	nory Layout from Target Dialog	X/0	) Base:		
☐ Make	RW Sections Position Independent	R/0	Base:	0x08000000	
	RO Sections Position Independent	R/V	N Base	0x20000000	
	Search Standard Libraries ort 'might fail' Conditions as Errors	disable Wa	minas:		
Scatter File	VOutput\多彩流水灯.sct				Edit
Scatter File					Edit
Scatter					Edit

选择使用MDK生成的sct文件

使用指定的sct文件构建工程



#### 通过Target对话框控制存储器分配

Device Target   Output   Li	isting   User   C	/L++   Asm	Linker   Debug   Uti	lities
Software Packs		¥		
Vendor: STMicroelectronics Device: STM32F429IGTx Toolset: ARM Search:		2000	e Pack k: Kell.STM32F4xx_DF L: http://www.keil.	
	332F429IGHx 132F429IGTx 429II 429NE 429NG 429NI 429VE	and an extensive ra to two APB buses, i matrix.  - 64-Kbyte of CCN - LCD parallel inter - Timer with quadr - 5 V-tolerant I/Os - Parallel camera ir - True random nur	three AHB buses and a 1 (core coupled memory) fface, 8080/6800 modes ature (incremental) enco-	ind peripherals connected 32-bit multi-AHB bus data RAM der input
1				

× ₩ Options for Target '多彩流水灯' Device Target Output Listing User | C/C++ | Asm | Linker | Debug | Utilities | STMicroelectronics STM32F429IGTx Code Generation Xtal (MHz): 12.0 ARM Compiler: Use default compiler version Operating system: None System Viewer File ☐ Use Cross-Module Optimization Use MicroLIB STM32F429x.svd ☐ Big Endian Floating Point Hardware: Use Single Precision ▼ Use Custom File Read/Only Memory Areas Read/Write Memory Areas default off-chip Startup default off-chip NoInit RAM1: ROM1: RAM2: ROM2: RAM3: ROM3: 0x8000000 0x100000 0x20000000 0x30000 IROM1: IRAM1: IRAM2: 0x10000000 0x10000 IROM2: OK Cancel Defaults Help

选择芯片类型

Target对话框中的存储器分配



#### MDK修改存储器分配示例

		ory Areas		M 1 0			nory Areas	(APPOINT)	
erauit	off-chip	Start	Size	No Init	defaul	off-chip	Start	Size	NoInit
Г	RAM1:			Г	Г	RAM1:			
Г	RAM2:					RAM2:			
	RAM3:					RAM3:			
	on-chip	~				on-chip			
~	IRAM1:	0x20001000	0x30000		₽ IV	IRAM1:	0x20000000	0x30000	Г
	IRAM2:	0x10000000	0x10000		V	IRAM2:	0x10000000	0x10000	



#### MDK修改存储器分配示例

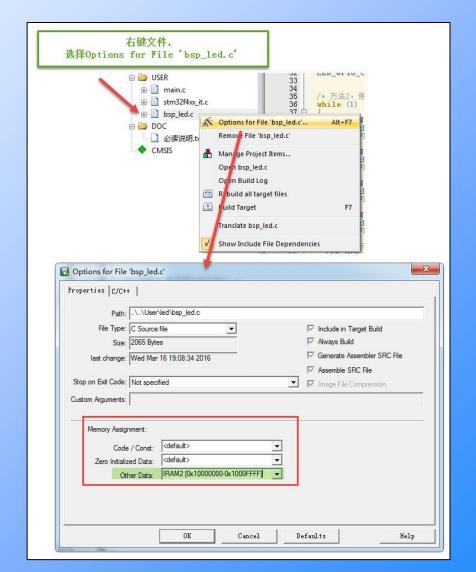
#### 代码清单 48-21 仅使用 IRAM2 时的 sct 文件内容

```
1 LR_IROM1 0x08000000 0x00100000 {
2 ER_IROM1 0x08000000 0x00100000 {
3 *.o (RESET, +First)
4 *(InRoot$$Sections)
5 .ANY (+RO)
6 }
7 RW_IRAM1 0x20000000 0x00030000 {
8 .ANY (+RW +ZI)
9 }
10 RW_IRAM2 0x10000000 0x00010000 {
11 .ANY (+RW +ZI)
12 }
13 }
```



#### 控制文件分配到指定的存储空间

设定好存储器的信息后,可以控制各个源文件定制到哪个部分存储器,在MDK的工程文件栏中,选中要配置的文件,右键,并在弹出的菜单中选择"Options for File xxxx"即可弹出一个文件配置对话框,在该对话框中进行存储器定制。





#### 控制文件分配到指定的存储空间

修改bsp\_led.c配置后的sct文件:

```
LR IROM1 0x08000000 0x00100000 { ; load region size region
   ER IROM1 0x08000000 0x00100000 { ; load address = execution address
     *.o (RESET, +First)
    *(InRoot$$Sections)
   .ANY (+RO)
   RW IRAM1 0x20000000 0x00030000 { ; RW data
8
     .ANY (+RW + ZI)
9
10
     RW IRAM2 0x10000000 0x00010000
       bsp led.o (+RW)
11
12
      .ANY (+RW + ZI)
13
```

类似地,还可以设置某些文件的代码段被存储到特定的ROM中,或者设置某些文件使用的ZI-data或RW-data存储到外部SDRAM中(控制ZI-data到SDRAM时注意还需要修改启动文件设置堆栈对应的地址,原启动文件中的地址是指向内部SRAM的)。



#### 控制文件分配到指定的存储空间

虽然MDK的这些存储器配置选项很方便,但有很多高级的配置还是需要手动编写sct文件实现的,例如MDK选项中的内部ROM选项最多只可以填充两个选项位置,若想把内部ROM分成多片地址管理就无法实现了;另外MDK配置可控的最小粒度为文件,若想控制特定的节区也需要直接编辑sct文件。

# 零死角玩转STM32





论坛: www.chuxue123.com

淘宝: firestm32.taobao.com



扫描进入淘宝店铺