

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.firebbs.cn



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SRAM

06

**实验：优先使用内部SRAM并
分配堆到外部SRAM**

MDK的编译过程及文件类型全解



Listing目录下的文件

在Listing目录下包含了*.map及*.lst文件，它们都是文本格式的，可使用Windows的记事本软件打开。其中lst文件仅包含了一些汇编符号的链接信息，我们重点分析map文件。

1.map文件说明

map文件是由链接器生成的，它主要包含交叉链接信息，查看该文件可以了解工程中各种符号之间的引用以及整个工程的Code、RO-data、RW-data以及ZI-data的详细及汇总信息。它的内容中主要包含了“节区的跨文件引用”、“删除无用节区”、“符号映像表”、“存储器映像索引”以及“映像组件大小”。

MDK的编译过程及文件类型全解



节区的跨文件引用

打开“多彩流水灯.map”文件，可看到它的第一部分——节区的跨文件引用(Section Cross References):

代码清单 42-10 节区的跨文件引用(部分，流水灯.map 文件)

```
1 =====
2
3 Section Cross References
4
5 startup_stm32f10x_hd.o(RESET) refers to startup_stm32f10x_hd.o(STACK) for __initial_sp
6 startup_stm32f10x_hd.o(RESET) refers to startup_stm32f10x_hd.o(.text) for Reset_Handler
7 startup_stm32f10x_hd.o(RESET) refers to stm32f10x_it.o(i.SysTick_Handler) for SysTick_Handler
8 /**...以下部分省略****/
9 main.o(i.main) refers to bsp_led.o(i.LED_GPIO_Config) for LED_GPIO_Config
10 main.o(i.main) refers to main.o(i.Delay) for Delay
11 bsp_led.o(i.LED_GPIO_Config) refers to stm32f10x_rcc.o(i.RCC_APB2PeriphClockCmd) for
                                RCC_APB2PeriphClockCmd
12 bsp_led.o(i.LED_GPIO_Config) refers to stm32f10x_gpio.o(i.GPIO_Init) for GPIO_Init
13 bsp_led.o(i.LED_GPIO_Config) refers to stm32f10x_gpio.o(i.GPIO_SetBits) for GPIO_SetBits
14 /**...以下部分省略****/
```

MDK的编译过程及文件类型全解



节区的跨文件引用

在这部分中，详细列出了各个*.o文件之间的符号引用。由于*.o文件是由asm或c/c++源文件编译后生成的，各个文件及文件内的节区间互相独立，链接器根据它们之间的互相引用链接起来，链接的详细信息在这个“Section Cross References”一一列出。

例如，开头部分说明的是startup_stm32f429_439xx.o文件中的“RESET”节区分为它使用的“__initial_sp”符号引用了同文件“STACK”节区。也许我们对启动文件不熟悉，不清楚这究竟是什么，那我们继续浏览，可看到main.o文件的引用说明，如说明main.o文件的i.main节区为它使用的LED_GPIO_Config符号引用了bsp_led.o文件的i.LED_GPIO_Config节区。同样地，下面还有bsp_led.o文件的引用说明，如说明了bsp_led.o文件的i.LED_GPIO_Config节区为它使用的GPIO_Init符号引用了stm32f4xx_gpio.o文件的i.GPIO_Init节区。

MDK的编译过程及文件类型全解



节区的跨文件引用

可以了解到，这些跨文件引用的符号其实就是源文件中的函数名、变量名。有时在构建工程的时候，编译器会输出 “Undefined symbol xxx (referred from xxx.o)” 这样的提示，该提示的原因就是在链接过程中，某个文件无法在外部找到它引用的标号，因而产生链接错误。

例如：把bsp_led.c文件中定义的函数LED_GPIO_Config改名为LED_GPIO_ConfigABCD，而不修改main.c文件中的调用，就会出现main文件无法找到LED_GPIO_Config符号的提示。

MDK的编译过程及文件类型全解



节区的跨文件引用

```
main.c
16  /*
17  |
18  | #include "stm32f10x.h"
19  | #include "bsp_led.h"
20  |
21  | #define SOFT_DELAY Delay(0x0FFFFFFF);
22  |
23  | void Delay(__IO u32 nCount);
24  |
25  | /**
26  |  * @brief 主函数
27  |  * @param 无
28  |  * @retval 无
29  |  */
30  | int main(void)
31  | {
32  |     /* LED 端口初始化 */
33  |     LED_GPIO_Config();
34  |
35  |     while (1)
```

```
bsp_led.c
16  /*
17  |
18  | #include "bsp_led.h"
19  |
20  | /**
21  |  * @brief 初始化控制LED的IO
22  |  * @param 无
23  |  * @retval 无
24  |  */
25  | void LED_GPIO_Config(ABCD)(void)
26  | {
27  |     /*定义一个GPIO_InitTypeDef类型
28  |     GPIO_InitTypeDef GPIO_InitS
29  |
30  |     /*开启LED相关的GPIO外设时钟*/
31  |     RCC_APB2PeriphClockCmd( LED
32  |     /*选择要控制的GPIO引脚*/
33  |     GPIO_InitStructure.GPIO_Pin
34  |
35  |     /*设置引脚模式为通用推挽输出*/
```

```
Build Output
*** Using Compiler 'V5.05 update 2 (build 169)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Led'
compiling bsp_led.c...
linking...
..\..\Output\流水灯.axf: Error: L6218E: Undefined symbol LED_GPIO_Config (referred from main.o).
Not enough information to list image symbols.
Finished: 1 information, 0 warning and 1 error messages.
"..\..\Output\流水灯.axf" - 1 Error(s), 0 Warning(s).
Target not created.
Build Time Elapsed: 00:00:01
```

MDK的编译过程及文件类型全解



删除无用节区

map文件的第二部分是删除无用节区的说明(Removing Unused input sections from the image.):

```
1 =====
2
3 Removing Unused input sections from the image.
4
5 Removing startup_stm32f10x_hd.o(HEAP), (512 bytes).
6 Removing core_cm3.o(.emb_text), (32 bytes).
7 Removing system_stm32f10x.o(i.SystemCoreClockUpdate), (164 bytes).
8 Removing system_stm32f10x.o(.data), (20 bytes).
9 Removing misc.o(i.NVIC_Init), (112 bytes).
10 Removing misc.o(i.NVIC_PriorityGroupConfig), (20 bytes).
11 Removing misc.o(i.NVIC_SetVectorTable), (20 bytes).
12 Removing misc.o(i.NVIC_SystemLPConfig), (32 bytes).
13 Removing misc.o(i.SysTick_CLKSourceConfig), (40 bytes).
14 Removing stm32f10x_adc.o(i.ADC_AnalogWatchdogCmd), (20 bytes).
15 Removing stm32f10x_adc.o(i.ADC_AnalogWatchdogSingleChannelConfig), (16 bytes).
16 Removing stm32f10x_adc.o(i.ADC_AnalogWatchdogThresholdsConfig), (6 bytes).
17 Removing stm32f10x_adc.o(i.ADC_AutoInjectedConvCmd), (22 bytes).
18 Removing stm32f10x_adc.o(i.ADC_ClearFlag), (6 bytes).
19 Removing stm32f10x_adc.o(i.ADC_ClearITPendingBit), (10 bytes).
20 Removing stm32f10x_adc.o(i.ADC_Cmd), (22 bytes).
21 Removing stm32f10x_adc.o(i.ADC_DMACmd), (22 bytes).
22 Removing stm32f10x_adc.o(i.ADC_DeInit), (92 bytes).
23 Removing stm32f10x_adc.o(i.ADC_DiscModeChannelCountConfig), (24
bytes).
24 /*...以下部分省略*/
```


MDK的编译过程及文件类型全解



删除无用节区

这部分列出了在链接过程它发现工程中未被引用的节区，这些未被引用的节区将会被删除(指不加入到*.axf文件，不是指在*.o文件删除)，这样可以防止这些无用数据占用程序空间。

例如，上面的信息中说明startup_stm32f10x.o中的HEAP(在启动文件中定义的用于动态分配的“堆”区)以及 stm32f10x_adc.o的各个节区都被删除了，因为在我们这个工程中没有使用动态内存分配，也没有引用任何 stm32f10x_adc.c中的内容。由此也可以知道，虽然我们把STM32标准库的各个外设对应的c库文件都添加到了工程，但不必担心这会使工程变得臃肿，因为未被引用的节区内容不会被加入到最终的机器码文件中

MDK的编译过程及文件类型全解



符号映像表

map文件的第三部分是符号映像表(Image Symbol Table):

流水灯.map 文件)

```
1
2 Image Symbol Table
3
4 Local Symbols
5
6 Symbol Name Value Ov Type Size Object(Section)
7 /**...省略部分****/
8 ../clib/microlib/init/entry.s 0x00000000 Number 0 entry7b.o ABSOLUTE
9 ../clib/microlib/init/entry.s 0x00000000 Number 0 entry11b.o ABSOLUTE
10 ../clib/microlib/init/entry.s 0x00000000 Number 0 entry11a.o ABSOLUTE
11 ../clib/microlib/init/entry.s 0x00000000 Number 0 entry10b.o ABSOLUTE
12
13 i.DebugMon_Handler 0x08000190 Section 0 stm32f10x_it.o(i.DebugMon_Handler)
14 i.Delay 0x08000192 Section 0 main.o(i.Delay)
15 i.GPIO_Init 0x080001a4 Section 0 stm32f10x_gpio.o(i.GPIO_Init)
16 i.GPIO_SetBits 0x080002ba Section 0 stm32f10x_gpio.o(i.GPIO_SetBits)
17 i.HardFault_Handler 0x080002be Section 0 stm32f10x_it.o(i.HardFault_Handler)
18 i.LED_GPIO_Config 0x080002c4 Section 0 bsp_led.o(i.LED_GPIO_Config)
19 i.RCC_APB2PeriphClockCmd 0x0800032c Section 0 stm32f10x_rcc.o(i.RCC_APB2PeriphClockCmd)
20 i.main 0x080004c0 Section 0 main.o(i.main)
21 STACK 0x20000000 Section 1024 startup_stm32f10x_hd.o(STACK)
22
23 Global Symbols
24
25 Symbol Name Value Ov Type Size Object(Section)
26 /**...省略部分****/
27 LED_GPIO_Config 0x080002c5 Thumb Code 90 bsp_led.o(i.LED_GPIO_Config)
28 RCC_APB2PeriphClockCmd 0x0800032d Thumb Code 26 stm32f10x_rcc.o(i.RCC_APB2PeriphClockCmd)
29 SVC_Handler 0x0800034d Thumb Code 2 stm32f10x_it.o(i.SVC_Handler)
30 SysTick_Handler 0x08000439 Thumb Code 2 stm32f10x_it.o(i.SysTick_Handler)
31 SystemInit 0x0800043d Thumb Code 78 system_stm32f10x.o(i.SystemInit)
32 main 0x080004c1 Thumb Code 252 main.o(i.main)
33 Region$$Table$$Base 0x080005c4 Number 0 anon$$obj.o(Region$$Table)
34 Region$$Table$$Limit 0x080005d4 Number 0 anon$$obj.o(Region$$Table)
35 __initial_sp 0x20000400 Data 0 startup_stm32f10x_hd.o(STACK)
36 /**...以下部分省略****/
```

MDK的编译过程及文件类型全解



符号映像表

这个表列出了被引用的各个符号在存储器中的具体地址、占据的空间大小等信息。如我们可以查到LED_GPIO_Config符号存储在0x080002c4地址，它属于Thumb Code类型，大小为90字节，它所在的节区为bsp_led.o文件的i.LED_GPIO_Config节区。

MDK的编译过程及文件类型全解



存储器映像索引

map文件的第四部分是存储器映像索引(Memory Map of the image):

代码清单 42-13 存储器映像索引(部分, 流水灯.map 文件)

```
1
2 Memory Map of the image
3
4 Image Entry point : 0x08000131
5
6 Load Region LR_IROM1 (Base: 0x08000000, Size: 0x000005d4, Max: 0x00080000, ABSOLUTE)
7
8 Execution Region ER_IROM1 (Base: 0x08000000, Size: 0x000005d4, Max: 0x00080000, ABSOLUTE)
9
10 Base Addr      Size      Type  Attr      Idx      E Section Name      Object
11
12
13 0x08000190      0x00000002  Code  RO        3130     i.DebugMon_Handler stm32f10x_it.o
14 0x08000192      0x00000012  Code  RO        3108     i.Delay             main.o
15 0x080001a4      0x00000116  Code  RO        1292     i.GPIO_Init         stm32f10x_gpio.o
16 0x080002ba      0x00000004  Code  RO        1300     i.GPIO_SetBits      stm32f10x_gpio.o
17 0x080002be      0x00000004  Code  RO        3131     i.HardFault_Handler stm32f10x_it.o
18 0x080002c2      0x00000002  PAD
19 0x080002c4      0x00000060  Code  RO        3192     i.LED_GPIO_Config   bsp_led.o
20 0x08000324      0x00000004  Code  RO        3132     i.MemManage_Handler stm32f10x_it.o
21 0x08000328      0x00000002  Code  RO        3133     i.NMI_Handler       stm32f10x_it.o
22 0x0800032a      0x00000002  Code  RO        3134     i.PendSV_Handler    stm32f10x_it.o
23 0x0800032c      0x00000020  Code  RO        1710     i.RCC_APB2PeriphClockCmd stm32f10x_rcc.o
24
25 0x080004be      0x00000002  PAD
26 0x080004c0      0x00000104  Code  RO        3109     i.main              main.o
27 0x080005c4      0x00000010  Data  RO        3226     Region$$Table       anon$$obj.o
28
29
30 Execution Region RW_IRAM1 (Base: 0x20000000, Size: 0x00000400, Max: 0x00010000, ABSOLUTE)
31
32 Base Addr      Size      Type  Attr      Idx      E Section Name      Object
33
34 0x20000000      0x00000400  Zero  RW        1        STACK             startup_stm32f10x_hd.o
```

MDK的编译过程及文件类型全解



存储器映像索引

该工程的存储器映像索引分为ER_IROM1及RW_IRAM1部分，它们分别对应STM32内部FLASH及SRAM的空间。相对于符号映像表，这个索引表描述的单位是节区，而且它描述的主要信息中包含了节区的类型及属性，由此可以区分Code、RO-data、RW-data及ZI-data。

例如，从上面的表中我们可以看到i.LED_GPIO_Config节区存储在内部FLASH的0x080002c4地址，大小为0x00000060，类型为Code，属性为RO。而程序的STACK节区(栈空间)存储在SRAM的0x20000000地址，大小为0x00000400，类型为Zero，属性为RW（即RW-data）。

MDK的编译过程及文件类型全解



映像组件大小

map文件的最后一部分是包含映像组件大小的信息(Image component sizes)，这也是最常查询的内容：

代码清单 42-14 映像组件大小(部分，多彩流水灯.map 文件)

```
1
2 Image component sizes
3
4 Code (inc. data)  RO Data  RW Data  ZI Data  Debug  Object Name
5
6 96 6 0 0 0 622 bsp_led.o
7 0 0 0 0 0 4504 core_cm3.o
8 278 8 0 0 0 1675 main.o
9 36 8 304 0 1024 932 startup_stm32f10x_hd.o
10 282 0 0 0 0 2771 stm32f10x_gpio.o
11 26 0 0 0 0 4726 stm32f10x_it.o
12 32 6 0 0 0 665 stm32f10x_rcc.o
13 328 28 0 0 0 214041 system_stm32f10x.o
14 -----
15 1084 56 320 0 1024 229936 Object Totals
16 0 0 16 0 0 0 (incl. Generated)
17 6 0 0 0 0 0 (incl. Padding)
18 -----
19 /*...省略部分*/
20 =====
21 Code (inc. data)  RO Data  RW Data  ZI Data  Debug
22
23 1172 72 320 0 1024 229428 Grand Totals
24 1172 72 320 0 1024 229428 ELF Image Totals
25 1172 72 320 0 0 0 ROM Totals
26 =====
27 Total RO Size (Code + RO Data) 1492 ( 1.46kB)
28 Total RW Size (RW Data + ZI Data) 1024 ( 1.00kB)
29 Total ROM Size (Code + RO Data + RW Data) 1492 ( 1.46kB)
30 =====
```

MDK的编译过程及文件类型全解



映像组件大小

这部分包含了各个使用到的*.o文件的空间汇总信息、整个工程的空间汇总信息以及占用不同类型存储器的空间汇总信息，它们分类描述了具体占据的Code、RO-data、RW-data及ZI-data的大小，并根据这些大小统计出占据的ROM总空间。

此处最后两部分信息，如Grand Totals一项，它表示整个代码占据的所有空间信息，其中Code类型的数据大小为1172字节，这部分包含了72字节的指令数据(inc .data)已算在内，另外RO-data占320字节，RW-data占0字节，ZI-data占1024字节。在它的下面两行有一项ROM Totals信息，它列出了各个段所占据的ROM空间，除了ZI-data不占ROM空间外，其余项都与Grand Totals中相等(RW-data也占据ROM空间，只是本工程中没有RW-data类型的数据而已)。

MDK的编译过程及文件类型全解



映像组件大小

最后一部分列出了只读数据(RO)、可读写数据(RW)及占据的ROM大小。其中只读数据大小为1492字节，它包含Code段及RO-data段；可读写数据大小为1024字节，它包含RW-data及ZI-data段；占据的ROM大小为1492字节，它除了Code段和RO-data段，还包含了运行时需要从ROM加载到RAM的RW-data数据（本工程中RW-data数据为0字节）。

综合整个map文件的信息，可以分析出，当程序下载到STM32的内部FLASH时，需要使用的内部FLASH是从0x0800 0000地址开始的大小为1492字节的空间；当程序运行时，需要使用的内部SRAM是从0x20000000地址开始的大小为1024字节的空间。

MDK的编译过程及文件类型全解



映像组件大小

粗略一看，发现这个小程序竟然需要1024字节的SRAM，实在说不过去，但仔细分析map文件后，可了解到这1024字节都是STACK节区的空间(即栈空间)，栈空间大小是在启动文件中定义的，这1024字节是默认值(0x00000400)。它是提供给C语言程序局部变量申请使用的空间，若我们确认自己的应用程序不需要这么大的栈，完全可以修改启动文件，把它改小一点，查看前面讲解的htm静态调用图文件可了解静态的栈调用情况，可以用它作为参考。

零死角玩转STM32



THANKS

论坛：www.firebbs.cn

淘宝：firestm32.taobao.com



扫描进入淘宝店铺