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

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 48-10 节区的跨文件引用(部分，多彩流水灯.map 文件)

```
1  ==============================================================================
2  Section Cross References
3
4      startup_stm32f429_439xx.o(RESET) refers to startup_stm32f429_439xx.o(STACK) for __initial_sp
5      startup_stm32f429_439xx.o(RESET) refers to startup_stm32f429_439xx.o(.text) for Reset_Handler
6      startup_stm32f429_439xx.o(RESET) refers to stm32f4xx_it.o(i.NMI_Handler) for NMI_Handler
7      startup_stm32f429_439xx.o(RESET) refers to stm32f4xx_it.o(i.HardFault_Handler) for HardFault_Handler
8      /**...以下部分省略****/
9
10     main.o(i.main) refers to bsp_led.o(i.LED_GPIO_Config) for LED_GPIO_Config
11     main.o(i.main) refers to stm32f4xx_gpio.o(i.GPIO_ResetBits) for GPIO_ResetBits
12     main.o(i.main) refers to main.o(i.Delay) for Delay
13     main.o(i.main) refers to stm32f4xx_gpio.o(i.GPIO_SetBits) for GPIO_SetBits
14     bsp_led.o(i.LED_GPIO_Config) refers to stm32f4xx_rcc.o(i.RCC_AHB1PeriphClockCmd) for RCC_AHB1PeriphClockCmd
15     bsp_led.o(i.LED_GPIO_Config) refers to stm32f4xx_gpio.o(i.GPIO_Init) for GPIO_Init
16     bsp_led.o(i.LED_GPIO_Config) refers to stm32f4xx_gpio.o(i.GPIO_ResetBits) for GPIO_ResetBits
17     /**...以下部分省略****/
18 ==============================================================================
19
```

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

```
17  #include "stm32f4xx.h"
18  #include "./led/bsp_led.h"
19
20
21
22  void Delay(__IO u32 nCount);
23
24  /**
25    * @brief  主函数
26    * @param  无
27    * @retval 无
28    */
29  int main(void)
30  {
31    /* LED 端口初始化 */
32    LED_GPIO_Config();
33
34
35    /* 方法2，使用固件库控制IO */
36    while (1)
37    {
```

bsp_led.c

```
8     ********************************************************************
9     * @attention
10    *
11    * 实验平台:秉火  STM32 F429 开发板
12    * 论坛    :http://www.chuxue123.com
13    * 淘宝    :http://firestm32.taobao.com
14    *
15    ********************************************************************
16    */
17
18  #include "./led/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_ConfigABCD(void)
26  {
27    /*定义一个GPIO_InitTypeDef类型的结构体*/
28    GPIO_InitTypeDef GPIO_InitStructure;
29
30    /*开启LED相关的GPIO外设时钟*/
31    RCC_AHB1PeriphClockCmd ( LED1_GPIO_CLK|LED2_GPIO_CLK
32
```

Build Output

```
Build target '多彩流水灯'
compiling bsp_led.c...
compiling main.c...
linking...
..\..\Output\多彩流水灯.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\多彩流水灯.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  Removing Unused input sections from the image.
3
4      Removing startup_stm32f429_439xx.o(HEAP), (512 bytes).
5      Removing system stm32f4xx.o(.rev16 text), (4 bytes).
6      Removing system_stm32f4xx.o(.revsh_text), (4 bytes).
7      Removing system_stm32f4xx.o(.rrx_text), (6 bytes).
8      Removing system_stm32f4xx.o(i.SystemCoreClockUpdate), (136 bytes).
9      Removing system_stm32f4xx.o(.data), (20 bytes).
10     Removing misc.o(.rev16_text), (4 bytes).
11     Removing misc.o(.revsh_text), (4 bytes).
12     Removing misc.o(.rrx_text), (6 bytes).
13     Removing misc.o(i.NVIC_Init), (104 bytes).
14     Removing misc.o(i.NVIC_PriorityGroupConfig), (20 bytes).
15     Removing misc.o(i.NVIC_SetVectorTable), (20 bytes).
16     Removing misc.o(i.NVIC_SystemLPConfig), (28 bytes).
17     Removing misc.o(i.SysTick_CLKSourceConfig), (28 bytes).
18     Removing stm32f4xx_adc.o(.rev16_text), (4 bytes).
19     Removing stm32f4xx_adc.o(.revsh_text), (4 bytes).
20     Removing stm32f4xx_adc.o(.rrx_text), (6 bytes).
21     Removing stm32f4xx_adc.o(i.ADC_AnalogWatchdogCmd), (16 bytes).
22     Removing stm32f4xx_adc.o(i.ADC_AnalogWatchdogSingleChannelConfig), (12 bytes).
23     Removing stm32f4xx_adc.o(i.ADC_AnalogWatchdogThresholdsConfig), (6 bytes).
24     Removing stm32f4xx_adc.o(i.ADC_AutoInjectedConvCmd), (24 bytes).
25     /**...以下部分省略****/
26 ==============================================================================
```

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_stm32f429_439xx.o中的HEAP(在启动文件中定义的用于动态分配的“堆”区)以及 stm32f4xx_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stm32f4xx_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)：

```
1  ==============================================================================
2  Image Symbol Table
3
4      Local Symbols
5
6      Symbol Name                             Value     Ov Type        Size  Object(Section)
7      ../clib/microlib/init/entry.s           0x00000000   Number         0  entry.o ABSOLUTE
8      ../clib/microlib/init/entry.s           0x00000000   Number         0  entry9a.o ABSOLUTE
9      ../clib/microlib/init/entry.s           0x00000000   Number         0  entry9b.o ABSOLUTE
10     /*...省略部分*/
11     LED_GPIO_Config                         0x080002a5   Thumb Code   106  bsp_led.o(i.LED_GPIO_Config)
12     MemManage_Handler                       0x08000319   Thumb Code     2  stm32f4xx_it.o(i.MemManage_Handler)
13     NMI_Handler                             0x0800031b   Thumb Code     2  stm32f4xx_it.o(i.NMI_Handler)
14     PendSV_Handler                          0x0800031d   Thumb Code     2  stm32f4xx_it.o(i.PendSV_Handler)
15     RCC_AHB1PeriphClockCmd          0x08000321   Thumb Code    22  stm32f4xx_rcc.o(i.RCC_AHB1PeriphClockCmd)
16     SVC_Handler                             0x0800033d   Thumb Code     2  stm32f4xx_it.o(i.SVC_Handler)
17     SysTick_Handler                         0x08000415   Thumb Code     2  stm32f4xx_it.o(i.SysTick_Handler)
18     SystemInit                              0x08000419   Thumb Code    62  system_stm32f4xx.o(i.SystemInit)
19     UsageFault_Handler                      0x08000469   Thumb Code     2  stm32f4xx_it.o(i.UsageFault_Handler)
20     __scatterload_copy                      0x0800046b   Thumb Code    14  handlers.o(i.__scatterload_copy)
21     __scatterload_null                      0x08000479   Thumb Code     2  handlers.o(i.__scatterload_null)
22     __scatterload_zeroinit                  0x0800047b   Thumb Code    14  handlers.o(i.__scatterload_zeroinit)
23     main                                    0x08000489   Thumb Code   270  main.o(i.main)
24     /*...省略部分*/
25 ==============================================================================
```

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0x080002a5地址，它属于Thumb Code类型，大小为106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)：

```
1  ==============================================================================
2  Memory Map of the image
3
4    Image Entry point : 0x080001ad
5    Load Region LR_IROM1 (Base: 0x08000000, Size: 0x000005b0, Max: 0x00100000, ABSOLUTE)
6
7      Execution Region ER_IROM1 (Base: 0x08000000, Size: 0x000005b0, Max: 0x00100000, ABSOLUTE)
8
9      Base Addr    Size         Type   Attr      Idx    E Section Name        Object
10
11     0x08000000   0x000001ac   Data   RO          3    RESET               startup_stm32f429_439xx.o
12     /*..省略部分*/
13     0x0800020c   0x00000012   Code   RO       5161    i.Delay             main.o
14     0x0800021e   0x0000007c   Code   RO       2046    i.GPIO_Init         stm32f4xx_gpio.o
15     0x0800029a   0x00000004   Code   RO       2053    i.GPIO_ResetBits    stm32f4xx_gpio.o
16     0x0800029e   0x00000004   Code   RO       2054    i.GPIO_SetBits      stm32f4xx_gpio.o
17     0x080002a2   0x00000002   Code   RO       5196    i.HardFault_Handler  stm32f4xx_it.o
18     0x080002a4   0x00000074   Code   RO       5269    i.LED_GPIO_Config   bsp_led.o
19     0x08000318   0x00000002   Code   RO       5197    i.MemManage_Handler  stm32f4xx_it.o
20     /*..省略部分*/
21     0x08000488   0x00000118   Code   RO       5162    i.main              main.o
22     0x080005a0   0x00000010   Data   RO       5309    Region$$Table       anon$$obj.o
23
24     Execution Region RW_IRAM1 (Base: 0x20000000, Size: 0x00000400, Max: 0x00030000, ABSOLUTE)
25
26     Base Addr    Size         Type   Attr      Idx    E Section Name        Object
27
28     0x20000000   0x00000400   Zero   RW          1     STACK               startup_stm32f429_439xx.o
29 ==============================================================================
```

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0x080002a4地址，大小为0x00000074，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：

```
==============================================================================
Image component sizes

      Code (inc. data)   RO Data    RW Data    ZI Data      Debug   Object Name

       116          10          0          0          0        578   bsp_led.o
       298          10          0          0          0       1459   main.o
        36           8        428          0       1024        932   startup_stm32f429_439xx.o
       132           0          0          0          0       2432   stm32f4xx_gpio.o
        18           0          0          0          0       3946   stm32f4xx_it.o
        28           6          0          0          0        645   stm32f4xx_rcc.o
       292          34          0          0          0     253101   system_stm32f4xx.o

    ------------------------------------------------------------------------------
       926          68        444          0       1024     263093   Object Totals
         0           0         16          0          0          0   (incl. Generated)
         6           0          0          0          0          0   (incl. Padding)

    /*...省略部分*/
==============================================================================
      Code (inc. data)   RO Data    RW Data    ZI Data      Debug

      1012          84        444          0       1024     262637   Grand Totals
      1012          84        444          0       1024     262637   ELF Image Totals
      1012          84        444          0          0          0   ROM Totals
==============================================================================
    Total RO  Size (Code + RO Data)                 1456 (   1.42kB)
    Total RW  Size (RW Data + ZI Data)              1024 (   1.00kB)
    Total ROM Size (Code + RO Data + RW Data)       1456 (   1.42kB)
==============================================================================
```

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1012字节，这部分包含了84字节的指令数据(inc .data)已算在内，另外RO-data占444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1456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1456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。

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1456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。

THANKS

论坛： www.chuxue123.com

淘宝：firestm32.taobao.com

扫描进入淘宝店铺