

15. 启动文件详解 — [野火]STM32 HAL库开发实战指南-F407骄阳 文档

 doc.embedfire.com/motor/f407jiaoyang/zh/latest/doc/chapter14/chapter14.html

15. 启动文件详解

本章参考资料《STM32F4xx 中文参考手册》的第十章-中断和事件：表 45. STM32F405xx/07xx 和 STM32F415xx/17xx的向量表； MDK中的帮助手册—ARM Development Tools：用来查询ARM的汇编指令和编译器相关的指令。

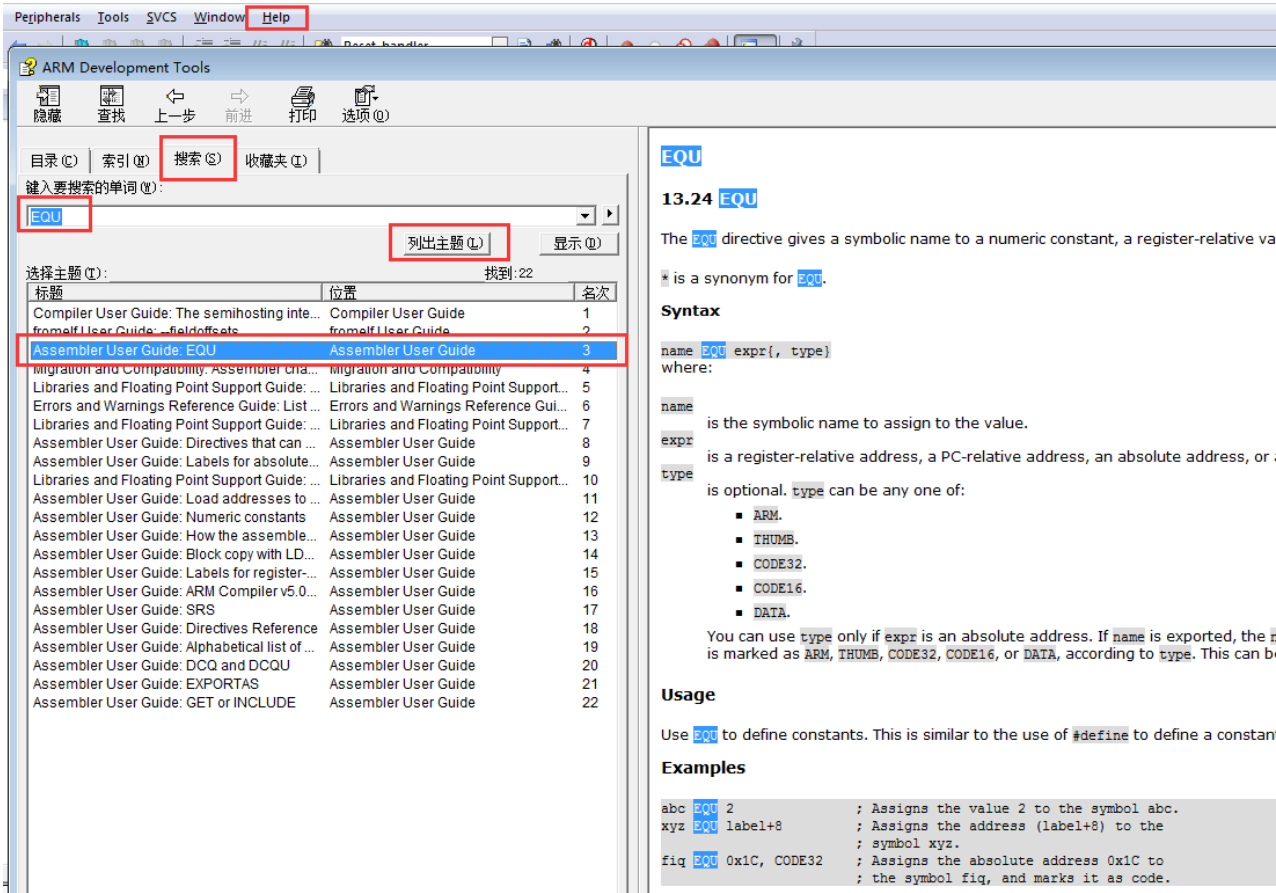
15.1. 启动文件简介

启动文件由汇编编写，是系统上电复位后第一个执行的程序。主要做了以下工作：

1. 初始化堆栈指针SP=_initial_sp
2. 初始化PC指针=Reset_Handler
3. 初始化中断向量表
4. 配置系统时钟
5. 调用C库函数_main初始化用户堆栈，从而最终调用main函数去到C的世界

15.2. 查找ARM汇编指令

在讲解启动代码的时候，会涉及到ARM的汇编指令和Cortex内核的指令，有关Cortex内核的指令我们可以参考《CM3权威指南CnR2》第四章：指令集。剩下的ARM的汇编指令我们可以在MDK（也就是Keil）的顶部菜单栏中->Help->Uvision Help中搜索到，以EQU为例，检索如下：



检索出来的结果会有很多，我们只需要看Assembler User Guide 这部分即可。下面列出了启动文件中使用到的ARM汇编指令，该列表的指令全部从ARM Development Tools这个帮助文档里面检索而来。其中编译器相关的指令WEAK和ALIGN为了方便也放在同一个表格了。

表格 14-1 启动文件使用的ARM汇编指令汇总

指令名称	作用
EQU	给数字常量取一个符号名，相当于C语言中的define
AREA	汇编一个新的代码段或者数据段
SPACE	分配内存空间
PRESERVE8	当前文件堆栈需按照8字节对齐
EXPORT	声明一个标号具有全局属性，可被外部的文件使用

DCD	以字为单位分配内存，要求4字节对齐，并要求初始化这些内存
PROC	定义子程序，与ENDP成对使用，表示子程序结束
WEAK	弱定义，如果外部文件声明了一个标号，则优先使用外部文件定义的标号，如果外部文件没有定义也不出错。要注意的是：这个不是ARM的指令，是编译器的，这里放在一起只是为了方便。
IMPORT	声明标号来自外部文件，跟C语言中的EXTERN关键字类似
B	跳转到一个标号
ALIGN	编译器对指令或者数据的存放地址进行对齐，一般需要跟一个立即数，缺省表示4字节对齐 要注意的是：这个不是ARM的指令，是编译器的，这里放在一起只是为了方便。
END	到达文件的末尾，文件结束
IF,ELSE,ENDIF	汇编条件分支语句，跟C语言的if else类似

15.3. 启动文件代码讲解

15.3.1. Stack—栈

```
Stack_Size      EQU      0x00000400

                AREA STACK, NOINIT, READWRITE, ALIGN=3

Stack_Mem       SPACE Stack_Size
__initial_sp
```

开辟栈的大小为0X00000400（1KB），名字为STACK，NOINIT即不初始化，可读可写，8（2^3）字节对齐。

栈的作用是用于局部变量，函数调用，函数形参等的开销，栈的大小不能超过内部SRAM的大小。如果编写的程序比较大，定义的局部变量很多，那么就需要修改栈的大小。如果某一天，你写的程序出现了莫名奇怪的错误，并进入了硬fault的时候，这时你就要考虑下是不是栈不够大，溢出了。

EQU：宏定义的伪指令，相当于等于，类似与C中的define。

AREA：告诉编译器汇编一个新的代码段或者数据段。STACK表示段名，这个可以任意命名；NOINIT表示不初始化；READWRITE表示可读可写，ALIGN=3，表示按照 2^3 对齐，即8字节对齐。

SPACE：用于分配一定大小的内存空间，单位为字节。这里指定大小等于Stack_Size。

标号__initial_sp紧挨着SPACE语句放置，表示栈的结束地址，即栈顶地址，栈是由高向低生长的。

15.3.2. Heap堆

```
Heap_Size      EQU 0x00000200

                AREA HEAP, NOINIT, READWRITE, ALIGN=3

__heap_base

                Heap_Mem SPACE Heap_Size

__heap_limit
```

开辟堆的大小为0X00000200（512字节），名字为HEAP，NOINIT即不初始化，可读可写，8（ 2^3 ）字节对齐。__heap_base表示堆的起始地址，__heap_limit表示堆的结束地址。堆是由低向高生长的，跟栈的生长方向相反。

堆主要用来动态内存的分配，像malloc()函数申请的内存就在堆上面。这个在STM32里面用的比较少。

1 PRESERVE8

2 THUMB

PRESERVE8：指定当前文件的堆栈按照8字节对齐。

THUMB：表示后面指令兼容THUMB指令。THUMB是ARM以前的指令集，16bit，现在Cortex-M系列的都使用THUMB-2指令集，THUMB-2是32位的，兼容16位和32位的指令，是THUMB的超集。

15.3.3. 向量表

```
AREA RESET, DATA, READONLY

EXPORT __Vectors

EXPORT __Vectors_End

EXPORT __Vectors_Size
```

定义一个数据段，名字为RESET，可读。并声明 __Vectors、__Vectors_End和 __Vectors_Size这三个标号具有全局属性，可供外部的文件调用。

EXPORT：声明一个标号可被外部的文件使用，使标号具有全局属性。如果是IAR编译器，则使用的是GLOBAL这个指令。

当内核响应了一个发生的异常后，对应的异常服务例程(ESR)就会执行。为了决定 ESR 的入口地址，内核使用了“向量表查表机制”。这里使用一张向量表。向量表其实是一个 WORD（32 位整数）数组，每个下标对应一种异常，该下标元素的值则是该 ESR 的入口地址。向量表在地址空间中的位置是可以设置的，通过 NVIC 中的一个重定位寄存器来指出向量表的地址。在复位后，该寄存器的值为 0。因此，在地址 0（即FLASH 地址0）处必须包含一张向量表，用于初始时的异常分配。要注意的是这里有个另类：0 号类型并不是什么入口地址，而是给出了复位后 MSP 的初值。

表 14-2 F407 向量表

编号	优先级	优先级类型	名称	说明	地址
	-	-	-	保留（实际存的是 MSP 地址）	0X0000 0000
	-3	固定	Reset	复位	0X0000 0004
	-2	固定	NMI	不可屏蔽中断。RCC 时钟安全系统 (CSS) 连接到 NMI 向量	0X0000 0008
	-1	固定	HardFault	所有类型的错误	0X0000 000C
	0	可编程	MemManage	存储器管理	0X0000 0010
	1	可编程	BusFault	预取指失败，存储器访问失败	0X0000 0014
	2	可编程	UsageFault	未定义的指令或非法状态	0X0000 0018
	-	-	-	保留	0X0000 001C - 0X0000 002B
	3	可编程	SVCall	通过 SWI 指令调用的系统服务	0X0000 002C
	4	可编程	Debug Monitor	调试监控器	0X0000 0030
	-	-	-	保留	0X0000 0034
	5	可编程	PendSV	可挂起的系统服务	0X0000 0038
	6	可编程	SysTick	系统嘀嗒定时器	0X0000 003C
0	7	可编程	-	窗口看门狗中断	0X0000 0040
1	8	可编程	PVD	连接 EXTI 线的可编程电压检测中断	0X0000 0044
2	9	可编程	TAMP_STAMP	连接 EXTI 线的入侵和时间戳中断	0X0000 0048
中间部分省略，详情请参考 STM32F4xx 中文参考手册》第十章-中断和事件-向量表部分					
79	86	可编程	CRYP	CRYP 加密全局中断	0X 0000 017C
80	87	可编程	HASH_RNG	哈希和随机数发生器全局中断	0X 0000 0180
81	88	可编程	FPU	FPU 全局中断	0X 0000 0184

代码 14-1 向量表

```
__Vectors DCD __initial_sp ;栈顶地址
```

```
DCD Reset_Handler ;复位程序地址
```

```
DCD NMI_Handler
```

```
DCD HardFault_Handler
```

```
DCD MemManage_Handler
```

```
DCD BusFault_Handler
```

```
DCD UsageFault_Handler
```

```
DCD 0 ; 0 表示保留
```

```
DCD 0
```

```
DCD 0
```

```
DCD 0
```

```
DCD SVC_Handler
```

```
DCD DebugMon_Handler
```

```
DCD 0
```

```
DCD PendSV_Handler
```

```
DCD SysTick_Handler
```

```
;外部中断开始
```

```
DCD WWDG_IRQHandler
```

```
DCD PVD_IRQHandler
```

```
DCD TAMPER_IRQHandler
```

```
;限于篇幅，中间代码省略
```

```
DCD DMA2_Channel1_IRQHandler
```

```
DCD DMA2_Channel2_IRQHandler
```

```
DCD DMA2_Channel3_4_IRQHandler
```

```
__Vectors_End __Vectors_Size EQU __Vectors_End - __Vectors
```

__Vectors为向量表起始地址，__Vectors_End 为向量表结束地址，两个相减即可算出向量表大小。

向量表从FLASH的0地址开始放置，以4个字节为一个单位，地址0存放的是栈顶地址，0X04存放的是复位程序的地址，以此类推。从代码上看，向量表中存放的都是中断服务函数的函数名，可我们知道C语言中的函数名就是一个地址。

DCD：分配一个或者多个以字为单位的内存，以四字节对齐，并要求初始化这些内存。在向量表中，DCD分配了一堆内存，并且以ESR的入口地址初始化它们。

15.3.4. 复位程序

```
1 AREA |.text|, CODE, READONLY
```

定义一个名称为.text的代码段，可读。

```
Reset_Handler PROC
```

```
    EXPORT Reset_Handler [WEAK]
```

```
    IMPORT SystemInit
```

```
    IMPORT __main
```

```
    LDR R0, =SystemInit
```

```
    BLX R0
```

```
    LDR R0, =__main
```

```
    BX R0
```

```
ENDP
```

复位子程序是系统上电后第一个执行的程序，调用SystemInit函数初始化系统时钟，然后调用C库函数__main，最终调用main函数去到C的世界。

WEAK：表示弱定义，如果外部文件优先定义了该标号则首先引用该标号，如果外部文件没有声明也不会出错。这里表示复位子程序可以由用户在其他文件重新实现，这里并不是唯一的。

IMPORT：表示该标号来自外部文件，跟C语言中的EXTERN关键字类似。这里表示SystemInit和__main这两个函数均来自外部的文件。

SystemInit()是一个标准的库函数，在system_stm32f103xe.c这个库文件中定义。主要作用是配置系统时钟，这里调用这个函数之后，单片机的系统时钟就被配置为72M。

__main是一个标准的C库函数，主要作用是初始化用户堆栈，最终调用main函数去到C的世界。这就是为什么我们写的程序都有一个main函数的原因。如果我们在这里不调用__main，那么程序最终就不会调用我们C文件里面的main，如果是调皮的用户就可以修改

主函数的名称， 然后在这里面IMPORT你写的主函数名称即可。

```
Reset_Handler PROC
    EXPORT Reset_Handler    [WEAK]
    IMPORT SystemInit
    IMPORT user_main

    LDR    R0, =SystemInit
    BLX    R0
    LDR    R0, =user_main
    BX     R0
ENDP
```

这个时候你在C文件里面写的主函数名称就不是main了，而是user_main了。

LDR、BLX、BX是CM4内核的指令，可在《CM3权威指南CnR2》第四章-指令集里面查询到，具体作用见下表：

指令名称	作用
LDR	从存储器中加载字到一个寄存器中
BL	跳转到由寄存器/标号给出的地址，并把跳转前的下条指令地址保存到LR
BLX	跳转到由寄存器给出的地址，并根据寄存器的LSE确定处理器的状态，还要把跳转前的下条指令地址保存到LR
BX	跳转到由寄存器/标号给出的地址，不用返回

15.3.5. 中断服务程序

在启动文件里面已经帮我们写好所有中断的中断服务函数，跟我们平时写的中断服务函数不一样的就是这些函数都是空的，真正的中断复服务程序需要我们在外部的C文件里面重新实现，这里只是提前占了一个位置而已。

如果我们在使用某个外设的时候，开启了某个中断，但是又忘记编写配套的中断服务程序或者函数名写错，那当中断来临的时，程序就会跳转到启动文件预先写好的空的中断服务程序中，并且在这个空函数中无线循环，即程序就死在这里。

NMI_Handler PROC ;系统异常

EXPORT NMI_Handler [WEAK]

B .

ENDP

;限于篇幅，中间代码省略

SysTick_Handler PROC

EXPORT SysTick_Handler [WEAK]

B .

ENDP

Default_Handler PROC ;外部中断

EXPORT WWDG_IRQHandler [WEAK]

EXPORT PVD_IRQHandler [WEAK]

EXPORT TAMP_STAMP_IRQHandler [WEAK]

;限于篇幅，中间代码省略

LTDC_IRQHandler

LTDC_ER_IRQHandler

DMA2D_IRQHandler

B .

ENDP

B：跳转到一个标号。这里跳转到一个‘.’，即表示无线循环。

15.3.6. 用户堆栈初始化

ALIGN：对指令或者数据存放的地址进行对齐，后面会跟一个立即数。缺省表示4字节对齐。

;用户栈和堆初始化,由C库函数_main来完成

IF : DEF :__MICROLIB ;这个宏在KEIL里面开启

EXPORT __initial_sp

EXPORT __heap_base

EXPORT __heap_limit

ELSE

IMPORT __use_two_region_memory ;这个函数由用户自己实现

EXPORT __user_initial_stackheap

__user_initial_stackheap

LDR R0, = Heap_Mem

LDR R1, =(Stack_Mem + Stack_Size)

LDR R2, = (Heap_Mem + Heap_Size)

LDR R3, = Stack_Mem

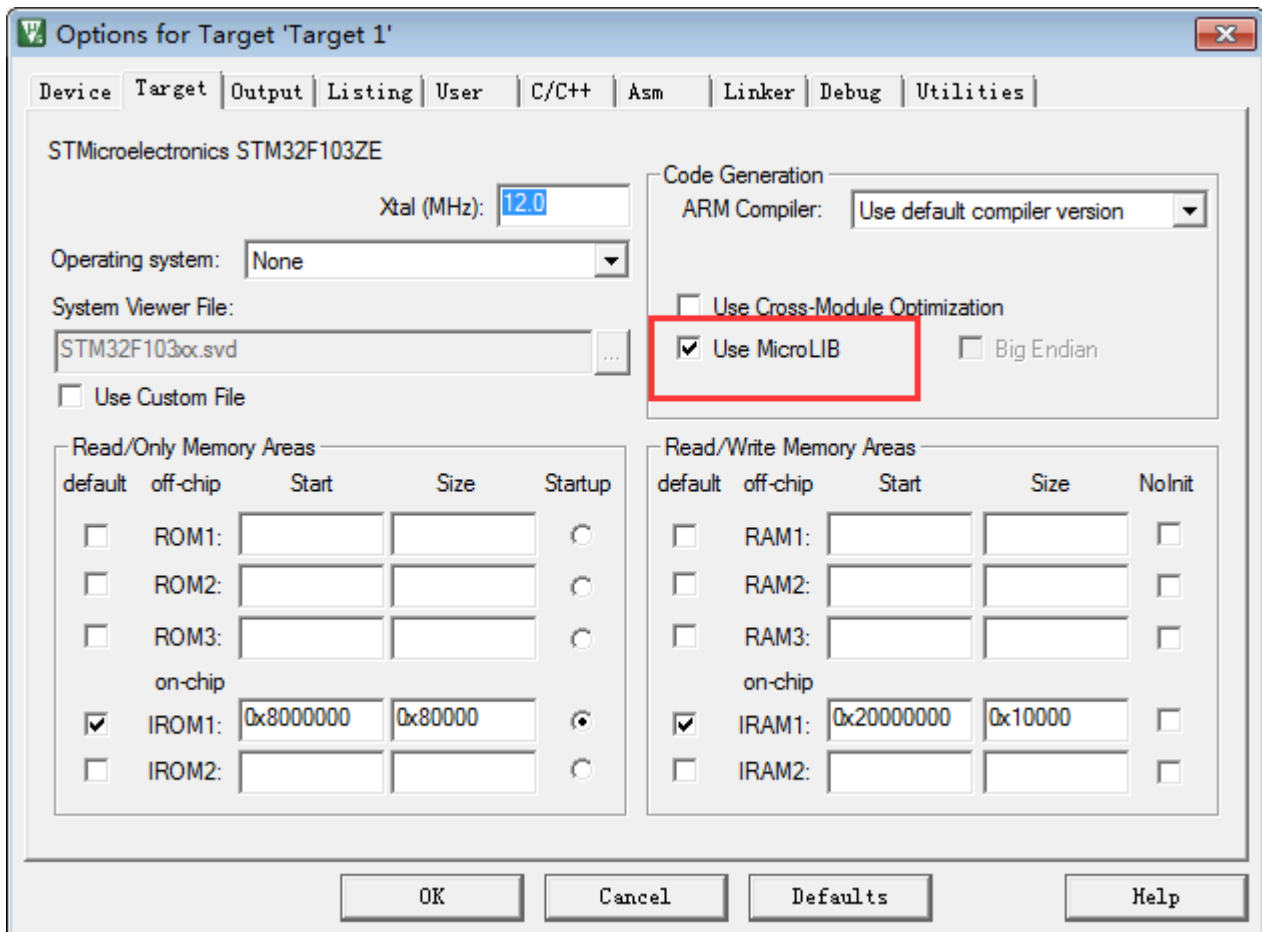
BX LR

ALIGN

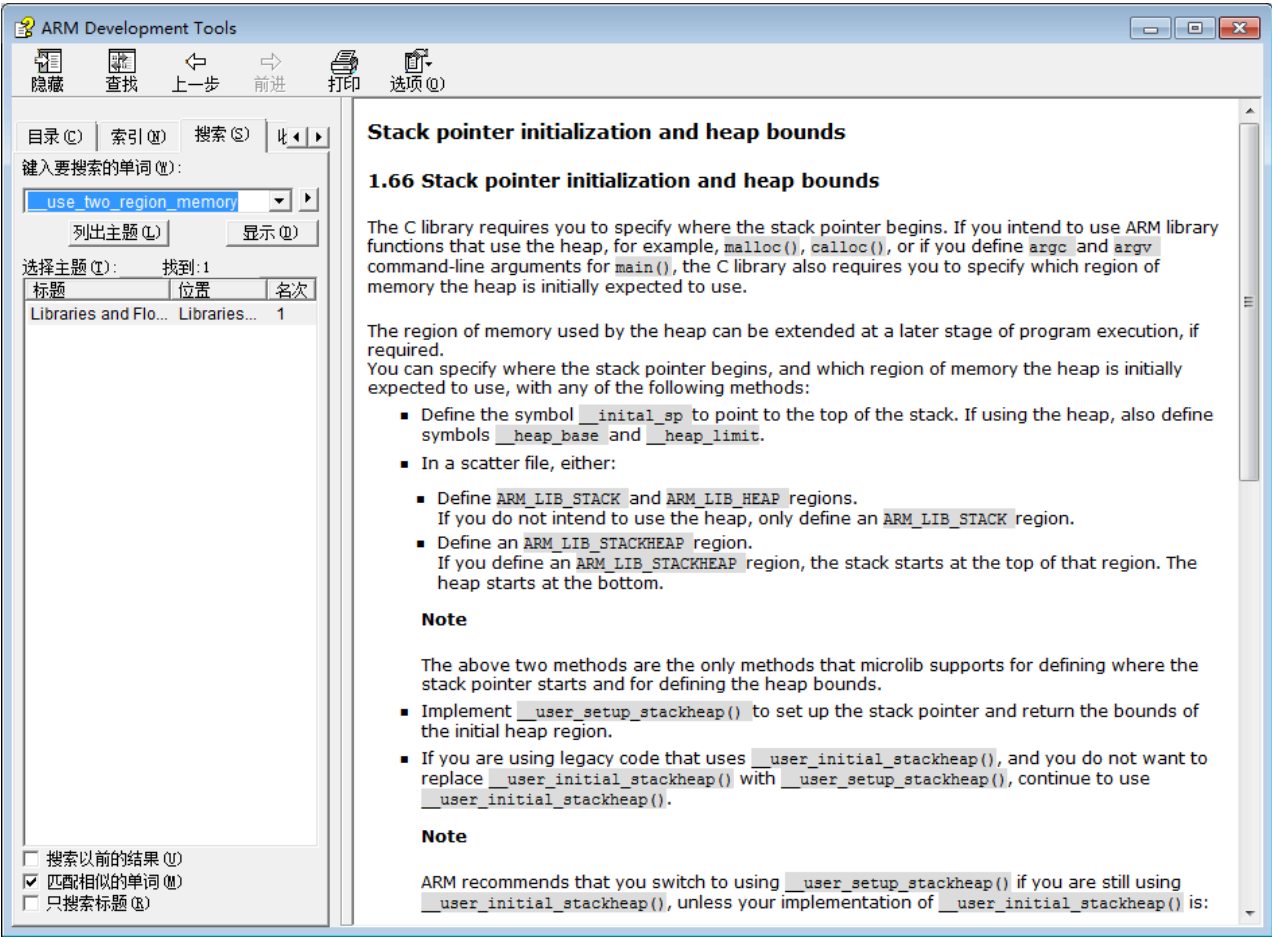
ENDIF

END

首先判断是否定义了__MICROLIB, 如果定义了这个宏则赋予标号__initial_sp (栈顶地址)、__heap_base (堆起始地址)、__heap_limit (堆结束地址) 全局属性, 可供外部文件调用。有关这个宏我们在KEIL里面配置, 具体 见 图14_2。然后堆栈的初始化就由C库函数_main来完成。



如果没有定义__MICROLIB，则插入标号__use_two_region_memory，这个函数需要用户自己实现，具体要实现成什么样，可在KEIL的帮助文档里面查询到，具体见 图14_3。



然后声明标号__user_initial_stackheap具有全局属性，可供外部文件调用，并实现这个标号的内容。

IF,ELSE,ENDIF：汇编的条件分支语句，跟C语言的if ,else类似

END：文件结束。