解析 STM32 的启动过程

当前的嵌入式应用程序开发过程里，并且 C 语言成为了绝大部分场合的最佳选择。如此一来 main 函数似乎成为了理所当然的起点——因为 C 程序往往从 main 函数开始执行。但一个经常会被忽略的问题是：微控制器（单片机）上电后，是如何寻找到并执行 main 函数的呢？很显然微控制器无法从硬件上定位 main 函数的入口地址，因为使用 C 语言作为开发语言后，变量/函数的地址便由编译器在编译时自行分配，这样一来 main 函数的入口地址在微控制器的内部存储空间中不再是绝对不变的。相信读者都可以回答这个问题，答案也许大同小异，但肯定都有个关键词，叫“启动文件”，用英文单词来描述是“Bootloader”。

无论性能高下，结构简繁，价格贵贱，每一种微控制器（处理器）都必须有启动文件，启动文件的作用便是负责执行微控制器从“复位”到“开始执行 main 函数”中间这段时间（称为启动过程）所必须进行的工作。最为常见的 51，AVR 或 MSP430 等微控制器当然也有对应启动文件，但开发环境往往自动完整地提供了这个启动文件，不需要开发人员再行干预启动过程，只需要从 main 函数开始进行应用程序的设计即可。

话题转到 STM32 微控制器，无论是 keil uvision4 还是 IAR EWARM 开发环境，ST 公司都提供了现成的直接可用的启动文件，程序开发人员可以直接引用启动文件后直接进行 C 应用程序的开发。这样能大大减小开发人员从其它微控制器平台跳转至 STM32 平台，也降低了适应 STM32 微控制器的难度（对于上一代 ARM 的当家花旦 ARM9，启动文件往往是第一道难啃却又无法逾越的坎）。

相对于 ARM 上一代的主流 ARM7/ARM9 内核架构，新一代 Cortex 内核架构的启动方式有了比较大的变化。ARM7/ARM9 内核的控制器在复位后，CPU 会从存储空间的绝对地址 0x000000 取出第一条指令执行复位中断服务程序的方式启动，即固定了复位后的起始地址为 0x000000（PC = 0x000000）同时中断向量表的位置并不是固定的。而 Cortex-M3 内核则正好相反，有 3 种情况:

1、 通过 boot 引脚设置可以将中断向量表定位于 SRAM 区，即起始地址为 0x2000000，同时复位后 PC 指针位于 0x2000000 处；
2、 通过 boot 引脚设置可以将中断向量表定位于 FLASH 区，即起始地址为 0x8000000，同时复位后 PC 指针位于 0x8000000 处；
3、 通过 boot 引脚设置可以将中断向量表定位于内置 Bootloader 区，本文不对这种情况做论述；

而 Cortex-M3 内核规定，起始地址必须存放堆顶指针，而第二个地址则必须存放复位中断入口向量地址，这样在 Cortex-M3 内核复位后，会自动从起始地址的下一个 32 位空间取出复位中断入口向量，跳转执行复位中断服务程序。对比 ARM7/ARM9 内核，Cortex-M3 内核则是固定了中断向量表的位置而起始地址是可变化的。

有了上述准备只是后，下面以 STM32 的 2.02 固件库提供的启动文件“stm32f10x_vector.s”为模板，对 STM32 的启动过程做一个简要而全面的解析。

程序清单一：

```
；文件“stm32f10x_vector.s”，其中注释为行号
DATA_IN_ExtSRAM   EQU       0                    ；1
Stack_Size        EQU       0x00000400           ；2
AREA      STACK, NOINIT, READWRITE, ALIGN = 3    ；3
Stack_Mem         SPACE     Stack_Size           ；4
__initial_sp                                     ；5
Heap_Size         EQU       0x00000400           ；6
AREA      HEAP, NOINIT, READWRITE, ALIGN = 3     ；7
```

```
__heap_base                                    ; 8
Heap_Mem            SPACE     Heap_Size        ; 9
__heap_limit                                   ; 10
THUMB                                          ; 11
PRESERVE8                                      ; 12
IMPORT  NMIException                           ; 13
IMPORT  HardFaultException                     ; 14
IMPORT  MemManageException                     ; 15
IMPORT  BusFaultException                      ; 16
IMPORT  UsageFaultException                    ; 17
IMPORT  SVCHandler                             ; 18
IMPORT  DebugMonitor                           ; 19
IMPORT  PendSVC                                ; 20
IMPORT  SysTickHandler                         ; 21
IMPORT  WWDG_IRQHandler                        ; 22
IMPORT  PVD_IRQHandler                         ; 23
IMPORT  TAMPER_IRQHandler                      ; 24
IMPORT  RTC_IRQHandler                         ; 25
IMPORT  FLASH_IRQHandler                       ; 26
IMPORT  RCC_IRQHandler                         ; 27
IMPORT  EXTI0_IRQHandler                       ; 28
IMPORT  EXTI1_IRQHandler                       ; 29
IMPORT  EXTI2_IRQHandler                       ; 30
IMPORT  EXTI3_IRQHandler                       ; 31
IMPORT  EXTI4_IRQHandler                       ; 32
IMPORT  DMA1_Channel1_IRQHandler               ; 33
IMPORT  DMA1_Channel2_IRQHandler               ; 34
IMPORT  DMA1_Channel3_IRQHandler               ; 35
IMPORT  DMA1_Channel4_IRQHandler               ; 36
IMPORT  DMA1_Channel5_IRQHandler               ; 37
IMPORT  DMA1_Channel6_IRQHandler               ; 38
IMPORT  DMA1_Channel7_IRQHandler               ; 39
IMPORT  ADC1_2_IRQHandler                      ; 40
IMPORT  USB_HP_CAN_TX_IRQHandler               ; 41
IMPORT  USB_LP_CAN_RX0_IRQHandler              ; 42
IMPORT  CAN_RX1_IRQHandler                     ; 43
IMPORT  CAN_SCE_IRQHandler                     ; 44
IMPORT  EXTI9_5_IRQHandler                     ; 45
IMPORT  TIM1_BRK_IRQHandler                    ; 46
IMPORT  TIM1_UP_IRQHandler                     ; 47
IMPORT  TIM1_TRG_COM_IRQHandler                ; 48
IMPORT  TIM1_CC_IRQHandler                     ; 49
IMPORT  TIM2_IRQHandler                        ; 50
IMPORT  TIM3_IRQHandler                        ; 51
```

```
IMPORT  TIM4_IRQHandler                 ; 52
IMPORT  I2C1_EV_IRQHandler              ; 53
IMPORT  I2C1_ER_IRQHandler              ; 54
IMPORT  I2C2_EV_IRQHandler              ; 55
IMPORT  I2C2_ER_IRQHandler              ; 56
IMPORT  SPI1_IRQHandler                 ; 57
IMPORT  SPI2_IRQHandler                 ; 58
IMPORT  USART1_IRQHandler               ; 59
IMPORT  USART2_IRQHandler               ; 60
IMPORT  USART3_IRQHandler               ; 61
IMPORT  EXTI15_10_IRQHandler            ; 62
IMPORT  RTCAlarm_IRQHandler             ; 63
IMPORT  USBWakeUp_IRQHandler            ; 64
IMPORT  TIM8_BRK_IRQHandler             ; 65
IMPORT  TIM8_UP_IRQHandler              ; 66
IMPORT  TIM8_TRG_COM_IRQHandler         ; 67
IMPORT  TIM8_CC_IRQHandler              ; 68
IMPORT  ADC3_IRQHandler                 ; 69
IMPORT  FSMC_IRQHandler                 ; 70
IMPORT  SDIO_IRQHandler                 ; 71
IMPORT  TIM5_IRQHandler                 ; 72
IMPORT  SPI3_IRQHandler                 ; 73
IMPORT  UART4_IRQHandler                ; 74
IMPORT  UART5_IRQHandler                ; 75
IMPORT  TIM6_IRQHandler                 ; 76
IMPORT  TIM7_IRQHandler                 ; 77
IMPORT  DMA2_Channel1_IRQHandler        ; 78
IMPORT  DMA2_Channel2_IRQHandler        ; 79
IMPORT  DMA2_Channel3_IRQHandler        ; 80
IMPORT  DMA2_Channel4_5_IRQHandler      ; 81
AREA    RESET, DATA, READONLY           ; 82
EXPORT  __Vectors                       ; 83
__Vectors                               ; 84
DCD  __initial_sp                       ; 85
DCD  Reset_Handler                      ; 86
DCD  NMIException                       ; 87
DCD  HardFaultException                 ; 88
DCD  MemManageException                 ; 89
DCD  BusFaultException                  ; 90
DCD  UsageFaultException                ; 91
DCD  0                                  ; 92
DCD  0                                  ; 93
DCD  0                                  ; 94
DCD  0                                  ; 95
```

```
DCD    SVCHandler                    ; 96
DCD    DebugMonitor                  ; 97
DCD    0                             ; 98
DCD    PendSVC                       ; 99
DCD    SysTickHandler                ; 100
DCD    WWDG_IRQHandler               ; 101
DCD    PVD_IRQHandler                ; 102
DCD    TAMPER_IRQHandler             ; 103
DCD    RTC_IRQHandler                ; 104
DCD    FLASH_IRQHandler              ; 105
DCD    RCC_IRQHandler                ; 106
DCD    EXTI0_IRQHandler              ; 107
DCD    EXTI1_IRQHandler              ; 108
DCD    EXTI2_IRQHandler              ; 109
DCD    EXTI3_IRQHandler              ; 110
DCD    EXTI4_IRQHandler              ; 111
DCD    DMA1_Channel1_IRQHandler      ; 112
DCD    DMA1_Channel2_IRQHandler      ; 113
DCD    DMA1_Channel3_IRQHandler      ; 114
DCD    DMA1_Channel4_IRQHandler      ; 115
DCD    DMA1_Channel5_IRQHandler      ; 116
DCD    DMA1_Channel6_IRQHandler      ; 117
DCD    DMA1_Channel7_IRQHandler      ; 118
DCD    ADC1_2_IRQHandler             ; 119
DCD    USB_HP_CAN_TX_IRQHandler      ; 120
DCD    USB_LP_CAN_RX0_IRQHandler     ; 121
DCD    CAN_RX1_IRQHandler            ; 122
DCD    CAN_SCE_IRQHandler            ; 123
DCD    EXTI9_5_IRQHandler            ; 124
DCD    TIM1_BRK_IRQHandler           ; 125
DCD    TIM1_UP_IRQHandler            ; 126
DCD    TIM1_TRG_COM_IRQHandler       ; 127
DCD    TIM1_CC_IRQHandler            ; 128
DCD    TIM2_IRQHandler               ; 129
DCD    TIM3_IRQHandler               ; 130
DCD    TIM4_IRQHandler               ; 131
DCD    I2C1_EV_IRQHandler            ; 132
DCD    I2C1_ER_IRQHandler            ; 133
DCD    I2C2_EV_IRQHandler            ; 134
DCD    I2C2_ER_IRQHandler            ; 135
DCD    SPI1_IRQHandler               ; 136
DCD    SPI2_IRQHandler               ; 137
DCD    USART1_IRQHandler             ; 138
DCD    USART2_IRQHandler             ; 139
```

```
DCD    USART3_IRQHandler                ; 140
DCD    EXTI15_10_IRQHandler             ; 141
DCD    RTCAlarm_IRQHandler              ; 142
DCD    USBWakeUp_IRQHandler             ; 143
DCD    TIM8_BRK_IRQHandler              ; 144
DCD    TIM8_UP_IRQHandler               ; 145
DCD    TIM8_TRG_COM_IRQHandler          ; 146
DCD    TIM8_CC_IRQHandler               ; 147
DCD    ADC3_IRQHandler                  ; 148
DCD    FSMC_IRQHandler                  ; 149
DCD    SDIO_IRQHandler                  ; 150
DCD    TIM5_IRQHandler                  ; 151
DCD    SPI3_IRQHandler                  ; 152
DCD    UART4_IRQHandler                 ; 153
DCD    UART5_IRQHandler                 ; 154
DCD    TIM6_IRQHandler                  ; 155
DCD    TIM7_IRQHandler                  ; 156
DCD    DMA2_Channel1_IRQHandler         ; 157
DCD    DMA2_Channel2_IRQHandler         ; 158
DCD    DMA2_Channel3_IRQHandler         ; 159
DCD    DMA2_Channel4_5_IRQHandler       ; 160
AREA       |.text|, CODE, READONLY      ; 161
Reset_Handler       PROC                ; 162
EXPORT    Reset_Handler                ; 163
IF          DATA_IN_ExtSRAM == 1        ; 164
LDR R0,= 0x00000114                     ; 165
LDR R1,= 0x40021014                     ; 166
STR R0,[R1]                             ; 167
LDR R0,= 0x000001E0                     ; 168
LDR R1,= 0x40021018                     ; 169
STR R0,[R1]                             ; 170
LDR R0,= 0x44BB44BB                     ; 171
LDR R1,= 0x40011400                     ; 172
STR R0,[R1]                             ; 173
LDR R0,= 0xBBBBBBBB                     ; 174
LDR R1,= 0x40011404                     ; 175
STR R0,[R1]                             ; 176
LDR R0,= 0xB44444BB                     ; 177
LDR R1,= 0x40011800                     ; 178
STR R0,[R1]                             ; 179
LDR R0,= 0xBBBBBBBB                     ; 180
LDR R1,= 0x40011804                     ; 181
STR R0,[R1]                             ; 182
LDR R0,= 0x44BBBBBB                     ; 183
```

```
LDR R1,= 0x40011C00                                    ; 184
STR R0,[R1]                                            ; 185
LDR R0,= 0xBBBB4444                                    ; 186
LDR R1,= 0x40011C04                                    ; 187
STR R0,[R1]                                            ; 188
LDR R0,= 0x44BBBBBB                                    ; 189
LDR R1,= 0x40012000                                    ; 190
STR R0,[R1]                                            ; 191
LDR R0,= 0x44444B44                                    ; 192
LDR R1,= 0x40012004                                    ; 193
STR R0,[R1]                                            ; 194
LDR R0,= 0x00001011                                    ; 195
LDR R1,= 0xA0000010                                    ; 196
STR R0,[R1]                                            ; 197
LDR R0,= 0x00000200                                    ; 198
LDR R1,= 0xA0000014                                    ; 199
STR R0,[R1]                                            ; 200
ENDIF                                                  ; 201
IMPORT   __main                                        ; 202
LDR      R0, =__main                                   ; 203
BX       R0                                            ; 204
ENDP                                                   ; 205
ALIGN                                                  ; 206
IF       :DEF:__MICROLIB                               ; 207
EXPORT   __initial_sp                                  ; 208
EXPORT   __heap_base                                   ; 209
EXPORT   __heap_limit                                  ; 210
ELSE                                                   ; 211
IMPORT   __use_two_region_memory                       ; 212
EXPORT   __user_initial_stackheap                      ; 213
__user_initial_stackheap                               ; 214
LDR      R0, = Heap_Mem                                ; 215
LDR      R1, = (Stack_Mem + Stack_Size)                ; 216
LDR      R2, = (Heap_Mem +    Heap_Size)               ; 217
LDR      R3, = Stack_Mem                               ; 218
BX       LR                                            ; 219
ALIGN                                                  ; 220
ENDIF                                                  ; 221
END                                                    ; 222
ENDIF                                                  ; 223
END                                                    ; 224
```

如程序清单一，STM32 的启动代码一共 224 行，使用了汇编语言编写，这其中的主要原因下文将会给出交代。现在从第一行开始分析：

- 第 1 行：定义是否使用外部 SRAM，为 1 则使用，为 0 则表示不使用。此语行若用 C 语

言表达则等价于：

#define DATA_IN_ExtSRAM 0

- 第 2 行：定义栈空间大小为 0x00000400 个字节，即 1Kbyte。此语行亦等价于：

  #define Stack_Size 0x00000400
- 第 3 行：伪指令 AREA，表示
- 第 4 行：开辟一段大小为 Stack_Size 的内存空间作为栈。
- 第 5 行：标号__initial_sp，表示栈空间顶地址。
- 第 6 行：定义堆空间大小为 0x00000400 个字节，也为 1Kbyte。
- 第 7 行：伪指令 AREA，表示
- 第 8 行：标号__heap_base，表示堆空间起始地址。
- 第 9 行：开辟一段大小为 Heap_Size 的内存空间作为堆。
- 第 10 行：标号__heap_limit，表示堆空间结束地址。
- 第 11 行：告诉编译器使用 THUMB 指令集。
- 第 12 行：告诉编译器以 8 字节对齐。
- 第 13—81 行：IMPORT 指令，指示后续符号是在外部文件定义的（类似 C 语言中的全局变量声明），而下文可能会使用到这些符号。
- 第 82 行：定义只读数据段，实际上是在 CODE 区（假设 STM32 从 FLASH 启动，则此中断向量表起始地址即为 0x8000000）
- 第 83 行：将标号__Vectors 声明为全局标号，这样外部文件就可以使用这个标号。
- 第 84 行：标号__Vectors，表示中断向量表入口地址。
- 第 85—160 行：建立中断向量表。
- 第 161 行：
- 第 162 行：复位中断服务程序，PROC…ENDP 结构表示程序的开始和结束。
- 第 163 行：声明复位中断向量 Reset_Handler 为全局属性，这样外部文件就可以调用此复位中断服务。
- 第 164 行：IF…ENDIF 为预编译结构，判断是否使用外部 SRAM，在第 1 行中已定义为“不使用”。
- 第 165—201 行：此部分代码的作用是设置 FSMC 总线以支持 SRAM，因不使用外部 SRAM 因此此部分代码不会被编译。
- 第 202 行：声明__main 标号。
- 第 203—204 行：跳转__main 地址执行。
- 第 207 行：IF…ELSE…ENDIF 结构，判断是否使用 DEF:__MICROLIB（此处为不使用）。
- 第 208—210 行：若使用 DEF:__MICROLIB，则将__initial_sp，__heap_base，__heap_limit 亦即栈顶地址，堆始末地址赋予全局属性，使外部程序可以使用。
- 第 212 行：定义全局标号__use_two_region_memory。
- 第 213 行：声明全局标号__user_initial_stackheap，这样外程序也可调用此标号。
- 第 214 行：标号__user_initial_stackheap，表示用户堆栈初始化程序入口。
- 第 215—218 行：分别保存栈顶指针和栈大小，堆始地址和堆大小至 R0，R1，R2，R3 寄存器。
- 第 224 行：程序完毕。

以上便是 STM32 的启动代码的完整解析，接下来对几个小地方做解释：

1、 AREA 指令：伪指令，用于定义代码段或数据段，后跟属性标号。其中比较重要的一个标号为“READONLY”或者“READWRITE”，其中“READONLY”表示该段为只读属性，联系到 STM32 的内部存储介质，可知具有只读属性的段保存于 FLASH 区，即 0x8000000

地址后。而“READONLY”表示该段为“可读写”属性，可知“可读写”段保存于 SRAM 区，即 0x2000000 地址后。由此可以从第 3、7 行代码知道，堆栈段位于 SRAM 空间。从第 82 行可知，中断向量表放置与 FLASH 区，而这也是整片启动代码中最先被放进 FLASH 区的数据。因此可以得到一条重要的信息：**0x8000000 地址存放的是栈顶地址__initial_sp，0x8000004 地址存放的是复位中断向量 Reset_Handler（STM32 使用 32 位总线，因此存储空间为 4 字节对齐）。**

2、 DCD 指令：作用是开辟一段空间，其意义等价于 C 语言中的地址符“&”。因此从第 84 行开始建立的中断向量表则类似于使用 C 语言定义了一个指针数组，其每一个成员都是一个函数指针，分别指向各个中断服务函数。

3、 标号：前文多处使用了“标号”一词。标号主要用于表示一片内存空间的某个位置，等价于 C 语言中的“地址”概念。地址仅仅表示存储空间的一个位置，从 C 语言的角度来看，变量的地址，数组的地址或是函数的入口地址在本质上并无区别。

4、 第 202 行中的__main 标号并不表示 C 程序中的 main 函数入口地址，因此第 204 行也并不是跳转至 main 函数开始执行 C 程序。__main 标号表示 C/C++标准实时库函数里的一个初始化子程序__main 的入口地址。该程序的一个主要作用是初始化堆栈（对于程序清单一来说则是跳转__user_initial_stackheap 标号进行初始化堆栈的），并初始化映像文件，最后跳转 C 程序中的 main 函数。这就解释了为何所有的 C 程序必须有一个 main 函数作为程序的起点——因为这是由 C/C++标准实时库所规定的——并且不能更改，因为 C/C++标准实时库并不对外界开发源代码。因此，实际上在用户可见的前提下，程序在第 204 行后就跳转至.c 文件中的 main 函数，开始执行 C 程序了。

至此可以总结一下 STM32 的启动文件和启动过程。首先对栈和堆的大小进行定义，并在代码区的起始处建立中断向量表，其第一个表项是栈顶地址，第二个表项是复位中断服务入口地址。然后在复位中断服务程序中跳转 C/C++标准实时库的__main 函数，完成用户堆栈等的初始化后，跳转.c 文件中的 main 函数开始执行 C 程序。假设 STM32 被设置为从内部 FLASH 启动（这也是最常见的一种情况），中断向量表起始地位为 0x8000000，则栈顶地址存放于 0x8000000 处，而复位中断服务入口地址存放于 0x8000004 处。当 STM32 遇到复位信号后，则从 0x80000004 处取出复位中断服务入口地址，继而执行复位中断服务程序，然后跳转__main 函数，最后进入 mian 函数，来到 C 的世界。