##### BOOT

基于ARM芯片的硬件系统，特别是复杂的片上系统（System on Chip, SoC），确实需要一段专门的初始化代码来配置各个硬件模块，以确保系统能够正常运行。

启动文件是任何处理器在上电复位之后最先运行的一段汇编程序。在我们编写的C语言代码运行之前，需要由汇编语言为C语言的运行建立一个合适的环境，接下来才能运行我们的程序。

总的来说，启动文件的作用是:

1、初始化堆栈指针 SP。

2、初始化程序计数器指针PC。

3、栈（Stack）配置：

; 堆栈配置区域

; 分配给堆栈的内存量（字节）

; 根据应用程序需求调整此值

Stack\_Size EQU 0x00000400

AREA STACK, NOINIT, READWRITE, ALIGN=3

Stack\_Mem SPACE Stack\_Size

\_\_initial\_sp

Stack\_Size：定义了栈的大小，这里设置为 1024 字节（0x00000400）。

STACK 区域：为栈分配了内存空间，并定义了 \_\_initial\_sp，这是初始栈指针的地址。

4、堆（Heap）配置：

; 堆配置区域

; 分配给堆的内存量（字节）

Heap\_Size EQU 0x00000200

AREA HEAP, NOINIT, READWRITE, ALIGN=3

\_\_heap\_base

Heap\_Mem SPACE Heap\_Size

\_\_heap\_limit

Heap\_Size：定义了堆的大小，这里设置为 512 字节（0x00000200）。

HEAP 区域：为堆分配了内存空间，并定义了 \_\_heap\_base 和 \_\_heap\_limit，分别表示堆的起始和结束地址。

5、设置异常向量表的入口地址。向量表是中断和异常处理程序的入口点列表，它被映射到地址 0，这样在复位时可以直接访问。

; 向量表映射到复位时的地址0

AREA RESET, DATA, READONLY

EXPORT \_\_Vectors

EXPORT \_\_Vectors\_End

EXPORT \_\_Vectors\_Size

; 向量表初始化，列出所有异常和中断处理程序的入口地址

\_\_Vectors DCD \_\_initial\_sp ; 堆栈顶部

DCD Reset\_Handler ; 复位处理程序

; ... 其他中断处理程序入口地址 ...

\_\_Vectors\_End

\_\_Vectors：定义了向量表的开始，包含了栈顶地址、复位处理程序地址和其他异常处理程序地址。

; 计算向量表的大小

\_\_Vectors\_Size EQU \_\_Vectors\_End - \_\_Vectors

; 代码区域，包含复位处理程序和异常处理程序的实现

AREA |.text|, CODE, READONLY

\_\_Vectors\_End 和 \_\_Vectors\_Size：分别表示向量表的结束和大小。

6、复位处理程序（Reset\_Handler）：微控制器复位后执行的第一个程序。

; 复位处理程序

Reset\_Handler PROC

EXPORT Reset\_Handler [WEAK]

IMPORT \_\_main

IMPORT SystemInit

; 调用SystemInit来配置系统

LDR R0, =SystemInit

BLX R0

; 跳转到C库的\_\_main函数

LDR R0, =\_\_main

BX R0

ENDP

首先调用 SystemInit 来初始化系统时钟和其他必要的设置，然后跳转到 C 库的 \_\_main 函数，最终调用 main 函数开始用户程序。

7、异常处理程序（Exception Handlers）：包括 NMI（非屏蔽中断）、HardFault（硬错误）、MemoryManage（内存管理错误）、BusFault（总线错误）等异常处理程序。

; 默认的异常处理程序（无限循环）

NMI\_Handler PROC

EXPORT NMI\_Handler [WEAK]

B .

ENDP

; ... 其他默认异常处理程序 ...

8、外部中断处理程序（External Interrupt Handlers）：列出了所有可能的外部中断处理程序，如 WWDG（看门狗定时器）、PVD（可编程电压检测器）、RTC（实时时钟）等。

9、默认处理程序（Default\_Handler）：所有未单独定义的中断处理程序都在这里定义。

10、栈和堆初始化：如果使用微库，会导出 \_\_initial\_sp、\_\_heap\_base 和 \_\_heap\_limit。如果不使用微库，则定义 \_\_user\_initial\_stackheap 来初始化栈和堆。

由于这些任务对性能和时序的要求非常严格，且需要直接与硬件交互，因此通常使用汇编语言来编写启动代码，以获得最高的执行效率和对硬件的精确控制。汇编语言与机器码紧密相关，可以精确地控制硬件寄存器和内存地址。

在实际开发中，启动代码可能由芯片制造商提供，或者由系统开发者根据具体需求定制。随着技术的发展，一些高级语言和框架也被用于简化启动代码的编写，但汇编语言在这一领域仍然占据重要地位。

##### SOFTWARE

固件库就是函数的集合，固件库函数的作用是向下负责与寄存器直接打交道，向上提供用户函数调用的接口（API）。

32位的单片机内部寄存器太多了，所以为了方便使用，完成了对寄存器的封装，凝结成了固件库。

在 51 的开发中可以直接操作寄存器，比如要控制某些 IO 口的状态，我们直接操作寄存器：

P0=0x11;

在 STM32 的开发中，我们同样可以操作寄存器：

GPIOx->BRR = 0x0011;

但使用固件库，我们可以看到这样的情况：

void GPIO\_ResetBits(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin)

{

GPIOx->BRR = GPIO\_Pin;

}

本质上仍为控制寄存器，但是已经封装好了函数可以调用。只需要知道怎么使用 GPIO\_ResetBits()这个函数即可。

基本上函数名字就在说明功能。