# 嵌入式系统实验报告



|  |  |
| --- | --- |
| 实验名称： | Lab3 嵌入式C程序生命周期观察与分析 |
| 姓 名： | 江姝潼 |
| 学 号： | 2019211653 |
| 学 院(系)： | 计算机学院 |
| 专 业： | 网络工程 |
| 指导教师： | 戴志涛、刘健培 |

2021年 11 月 10 日

# 实验目的

在上一个实验Lab1中，使用的是预先配置好的 example 工程，概览了 MDK下程序的编译、调试，本实验将使用MDK的调试功能，按照步骤观察程序的编译、链接、加载、运行过程，观察了解嵌入式C语言程序的生命周期。

# 实验环境

* FS-STM32F407开发平台
* ST-Link 仿真器
* KeiluVision MDK集成开发软件
* 串口调试工具
* PC机Window10 (64bit)

# 实验要求

1.阅读相关材料，按照步骤观察程序的编译、链接、加载、运行过程，并练习使用MDK配置/控制该过程的方法。

2.完成实验步骤 2.3，将程序的启动与运行的流程图贴在作业答卷里。

# 实验原理

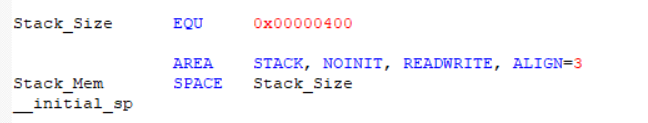
本实验旨在了解当使用MDK编写好源代码，再把代码下载到模拟器上运行的这个过程究竟做了哪些工作，已经处理器从上电开始至找到第一条指令并正确执行过程所经历的操作。

在MDK编译系统里对源程序的编译与链接和C源程序的执行大体流程相似，包括编译、链接和格式转换三个步骤。在编译的过程中，MDK软件使用 armcc 和 armasm编译器根据每个 c/c++ 和汇编源文件编译成对应的以".o"为后缀名的对象文件；在链接的步骤中，链接器 armlink 把各个.o 文件及库文件链接成一个可执行的映像文件，是以".axf"或".elf"为后缀的二进制格式文件；最后再进行格式转换，由于需要把文件的内容加载到芯片上，故需要对链接器生成的 elf映像文件利用格式转换器 fromelf转换成".bin"或".hex"文件，交给下载器下载到芯片的 FLASH 或 ROM中，在上电后由处理器启动运行。

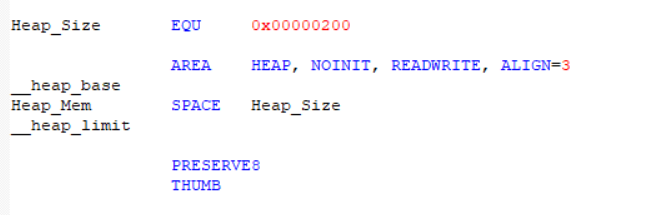
# 预备知识

要了解STM32F103的启动流程，先需要阅读了解startup\_stm32f10x\_md.s 文件。STM32E103的启动代码由4个段构成，分别为STACK段、HEAP段、RESET段（数据段）和|text|段（代码段）。其中，|.text|段是一个通过某种方式与C库关联的代码段。STM32F103的启动代码主要完成以下工作：定义栈空间（STACK段），定义堆空间（HEAP段），定义异常向量表（RESET段），定义异常服务程序（|.text|段），以及初始化堆栈（|.text|段）。

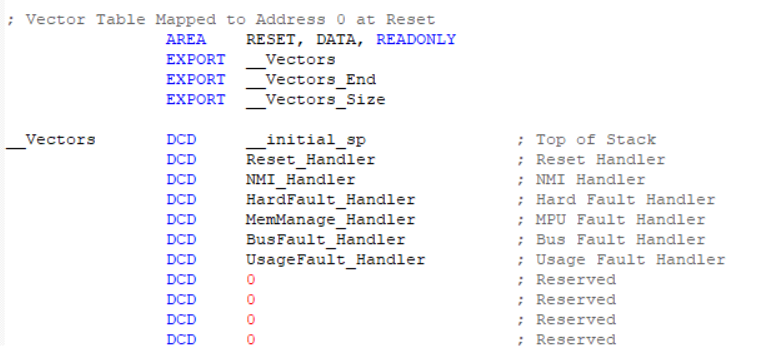
在stack段中，STM32F103启动代码仅定义栈内存空间而不进行初始化，栈的大小为Stack\_Size，即 0x00000400，1KB，其中，标号Stack\_Mem表示栈底地址，标号\_\_initial\_sp表示栈顶指针。



在HEAP段中，STM32F103启动代码仅定义堆内存空间而不进行初始化，堆的大小为Heap\_Size，即0x00000200，512B。其中，标号heap\_base表示堆空间起始地址，标号heap\_limit表示堆空间结束地址。

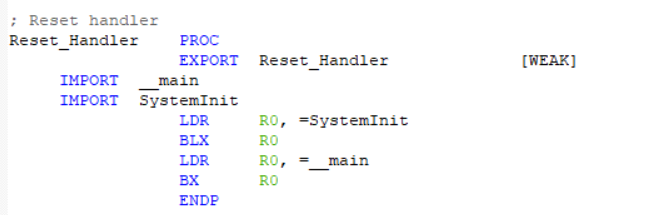


STM32F103启动代码还定义了STM32F103的异常向量表。异常向量表的起始地址为Vectors，结束地址为Vectors\_End，大小为Vectore\_Size。

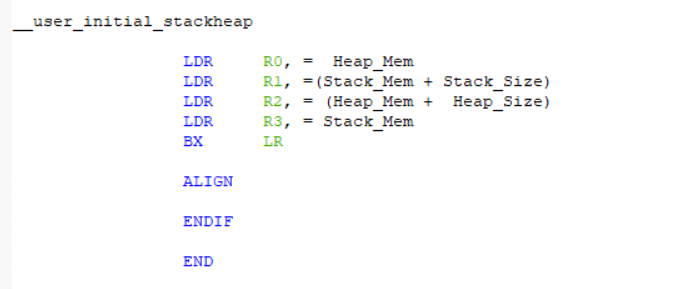


定义异常服务程序（|.text|段）为启动代码的核心部分，

异常服务程序在STM32F103启动代码的定义，包括复位异常服务程序、默认异常服务程序、其他异常服务程序三个部分。其中，复位异常服务程序Reset\_Handler是最重要的一个异常服务程序，它执行SystemInit()和\_\_main() 函数，其中，SystemInit()完成硬件初始化工作，包括初始化始终系统和使能Flash预取缓冲区，而\_\_main() 函数则负责完成应用程序运行环境的初始化工作。包括把RW/RO 输出端从装载域地址复制到运行域地址、ZI段运行域初始化、初始化堆栈、完成库函数的初始化，并最终跳到应用程序的main()函数。

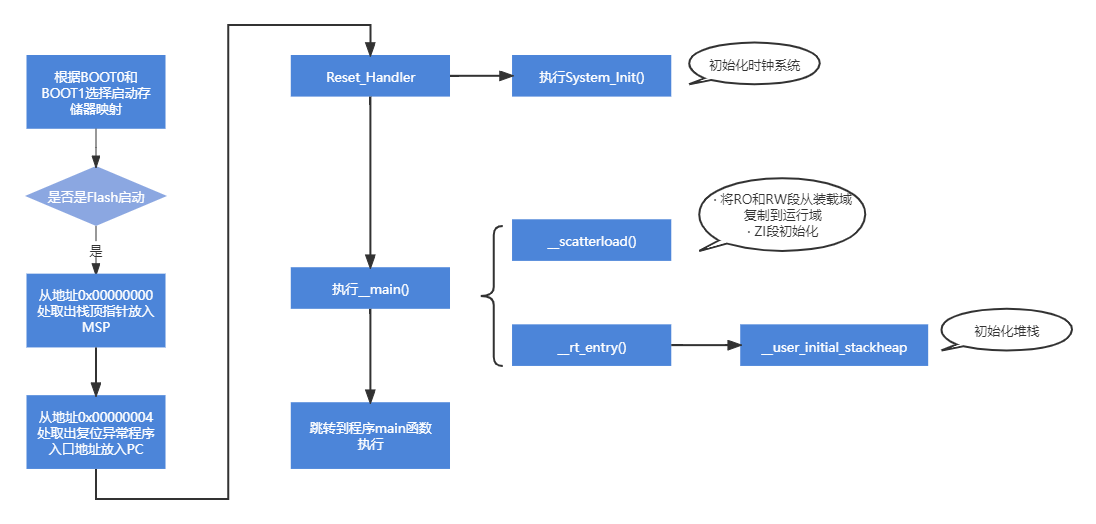


初始化堆栈（|.text|段）用于堆栈初始化程序，在\_\_main()函数执行过程中被调用，为编译器的初始化C库函数设置用户程序的堆栈提供所需要的信息。



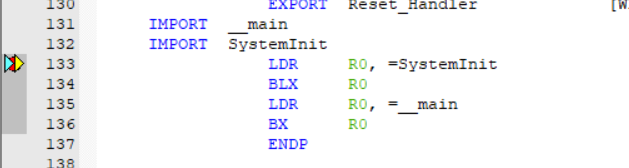
# 实验步骤

* 分析 lab2 的程序从处理器上电执行第一行指令到所做的工作用流程图的方式的流程图如下所示：



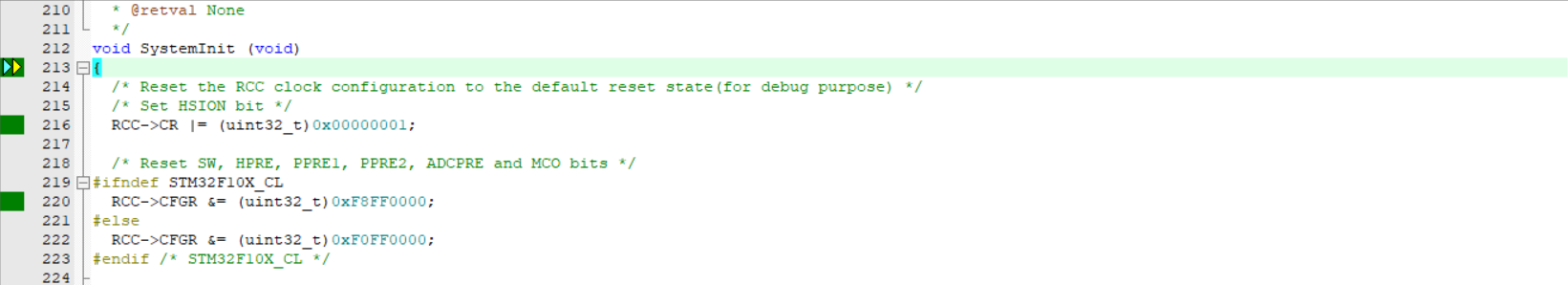
* 下面进入启动程序部分验证阶段：

1. 首先在SystemInit处加上断点，开始调试程序。

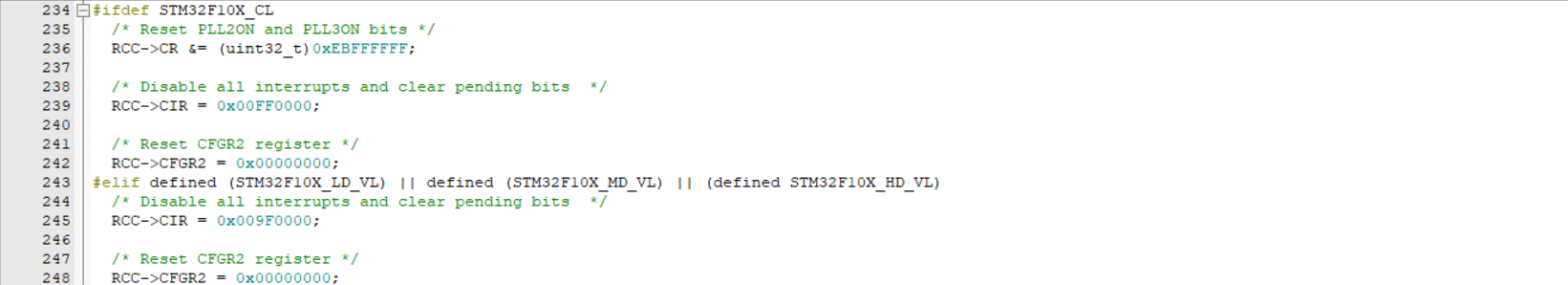


1. 程序进入了SystemInit函数，开始初始化时间系统，对应流程图部分如下：

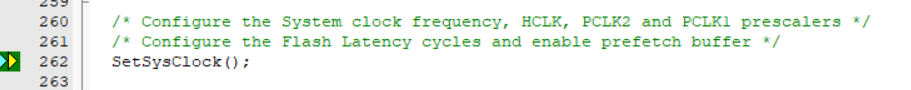


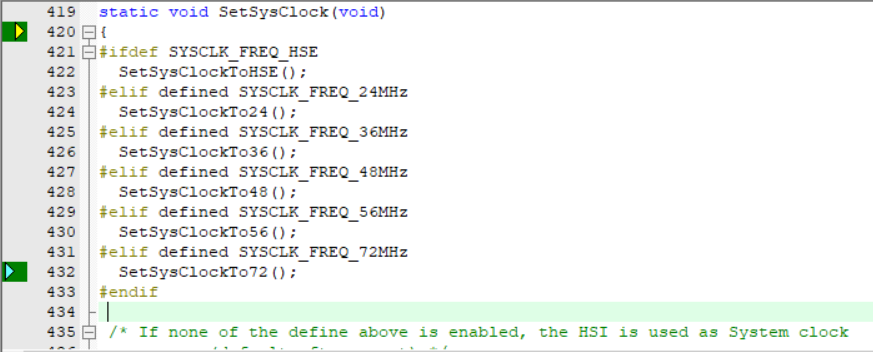


单步执行，可以看出其初始化了一些硬件，修改部分内存为特定值：

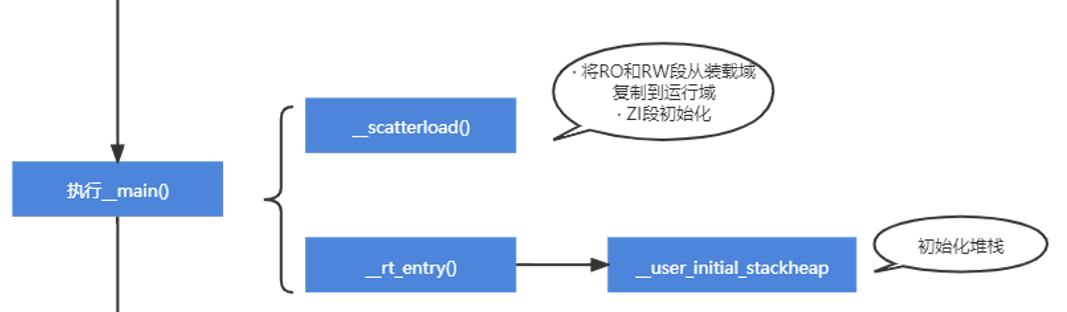


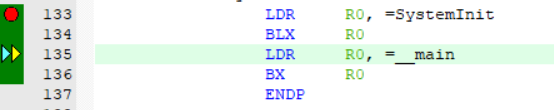
然后进入SetSysClock函数，初始化时钟系统：



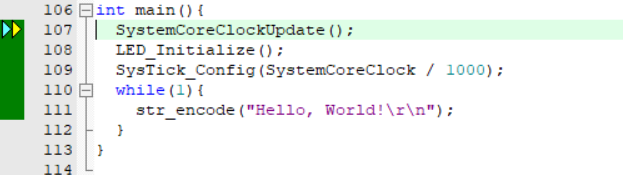


1. 程序调用\_\_main()函数，完成应用程序运行环境的初始化工作，完成了流程图的以下部分：



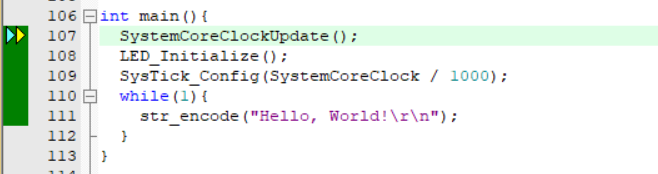


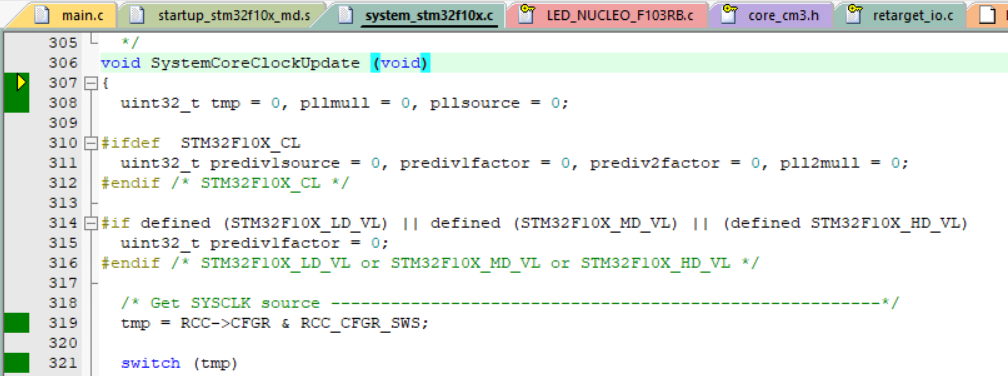
1. 继续执行，跳转至自己的程序代码的main()函数中执行。



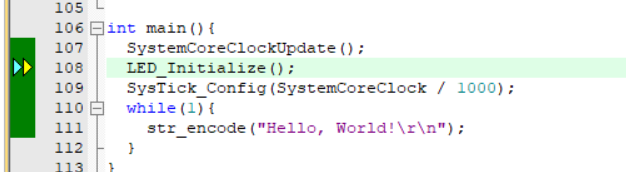
* 下面一步步调试观察执行main函数的过程：

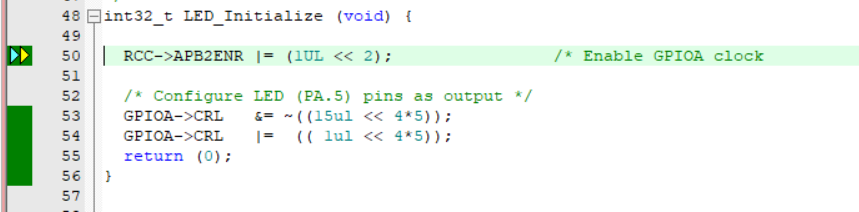
1. 先更新系统核时间，可以看到其进入system\_stm32f10x.c文件去执行该函数。



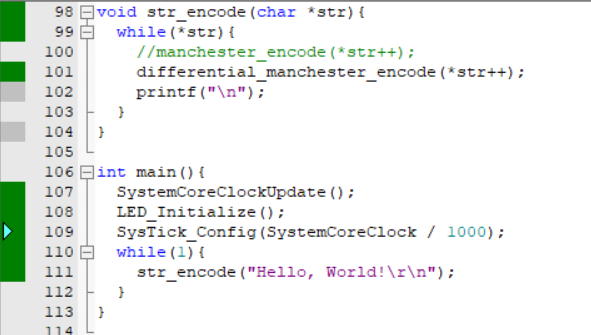


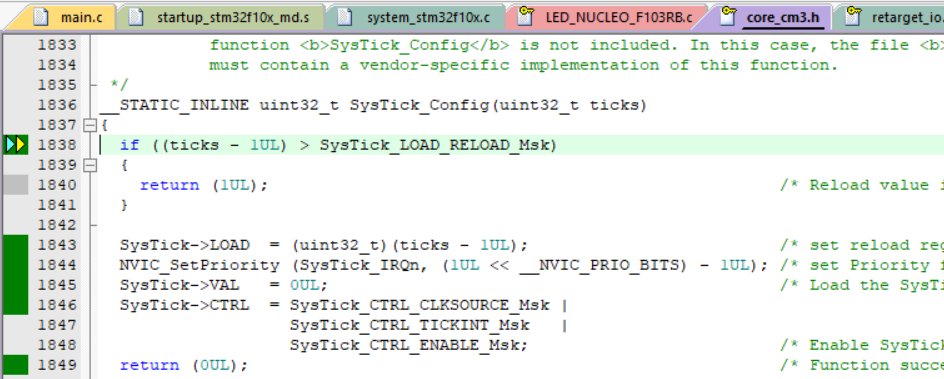
1. 然后初始化LED灯，可以看到其进入LED\_NUCLEO\_F103RB.c文件执行函数。



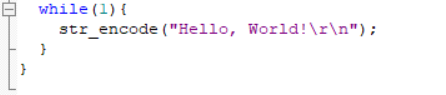


1. 下面进入SysTick\_Config函数，进入core\_cm3.h执行





1. 后面循环进入while（1）执行自己所编写的代码。



* 该程序不会结束，因为main函数里是个while，会一直循环执行，循环输出"Hello, World!\r\n"的差分曼彻斯特编码。

# 实验总结

本次实验了解了嵌入式C程序生命周期观察与分析的流程，与我们平时编写编译运行C语言不同的是，嵌入式实验是和硬件紧密相关的，实验创建的工程都是“裸板”程序，处理器从上电开始，需要完成内部硬件的初始化。处理器启动后一般无法直接执行 C 代码，原因是C 语言的执行对运行环境有一定的要求，需要先将环境初始化，例如堆栈、时钟、内存RW、ZI部分等，初始化完成后才继续执行用户所编写的C源代码。

本次实验通过自己单步调试执行startup文件并阅读了相关资料，帮助我认识了解了嵌入式C程序生命周期，了解了从MDK编译系统源程序的编译与链接文件的过程，到程序启动的软硬件初始化流程。主要分为软硬件两个部分；就软件部分，细化了解了各种文件格式，包括axf与 sct 文件、ELF文件格式等，以及在编译链接的过程中各个文件的变化情况，在后续的实验中可以更加透彻地理解执行过程；就硬件部分，本次实验也帮助我了解了各个节区和其对应的作用，包括CODE、RO、RW、ZI Data 段及堆栈空间，并了解了存储和加载的过程操作，其中提到了一个问题，即目标程序的存储与加载时，RW-data 和 ZI-data仅初值不同，但编译器需要将它们区分开，因为这涉及到是否要掉电保存的问题，和硬件性质是息息相关的。

本次实验较前两次实验更突出硬件的性质和编译环境的结合，如果说前两次是为了了解一些MDK的基本操作，此次实验就是了解那些操作的具体实现细节和背后涉及的原理，和arm芯片紧密地联系起来，为后面的实验打下了很好的基础。