第五天笔记

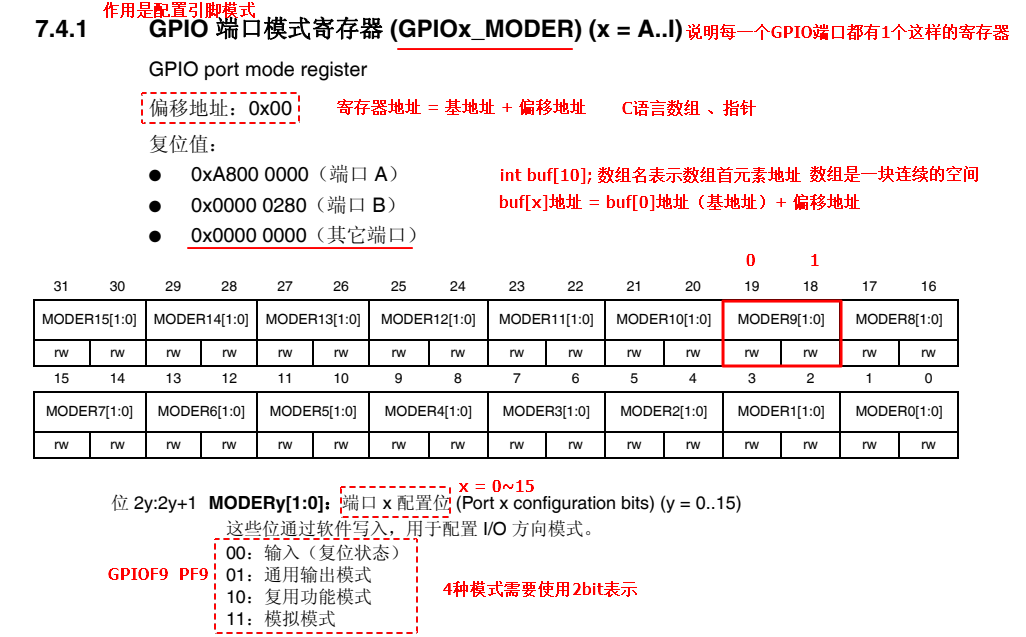
1. GPIO寄存器开发流程

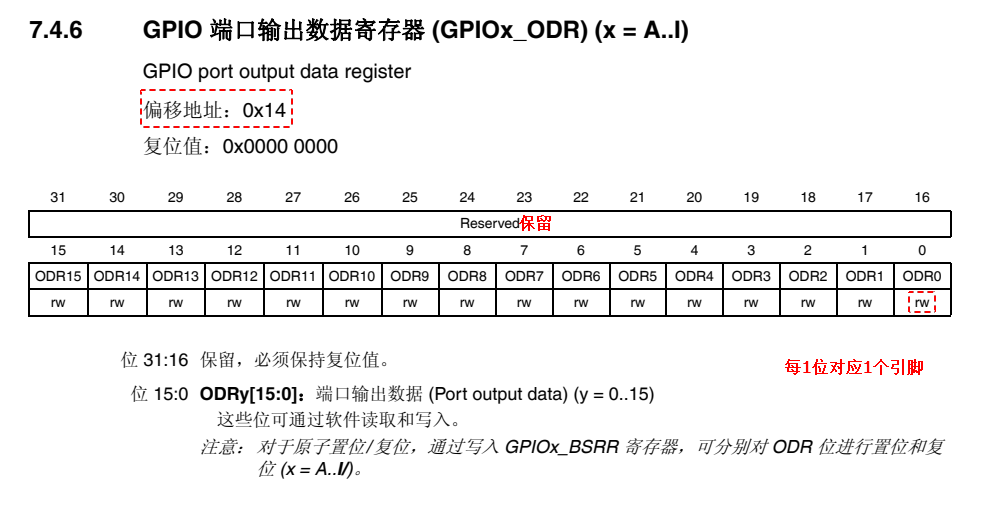
不管是图形界面还是函数库开发，本质都是对底层寄存器的使用，一般在实时性要求不高的情况，就使用前两种方式，但是**对实时性要求比较高的情况下，就需要使用寄存器开发**。所以就说明一下使用流程。

1. 分析原理图，找到外设连接的芯片的引脚 PF9
2. 分析原理图，理解硬件的控制原理 高电平灭 低电平亮
3. 分析中文参考手册或者芯片数据手册，了解需要使用的寄存器都有哪些

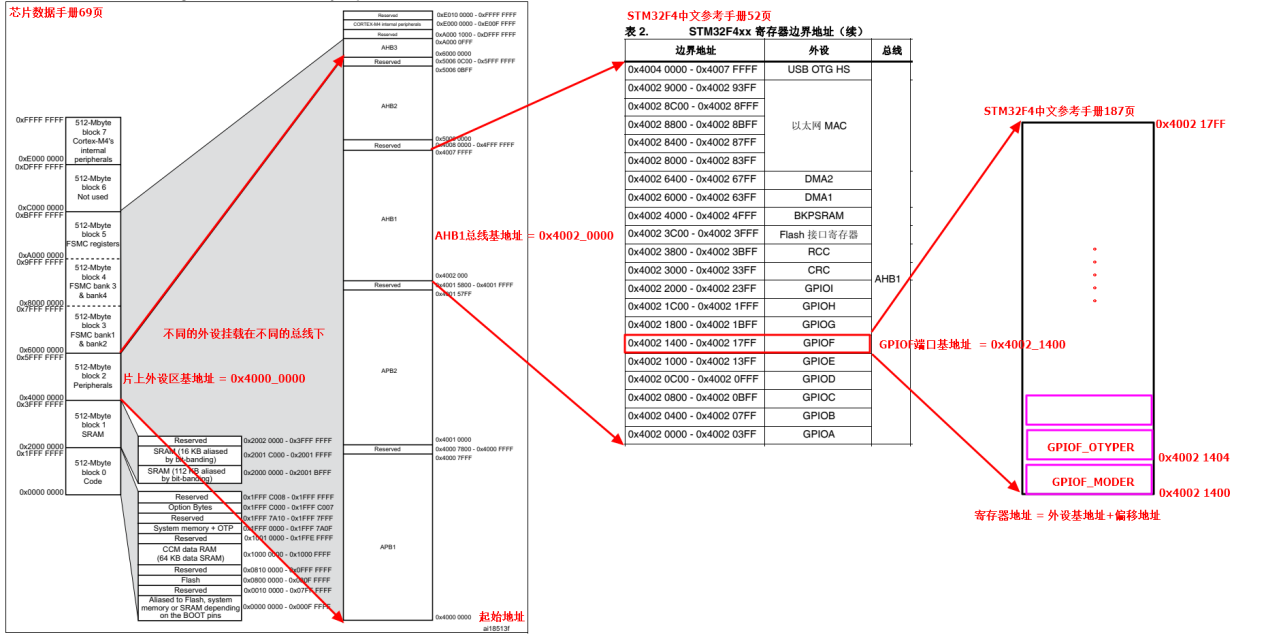


1. 理解寄存器的工作原理 参考中文参考手册 STM32的寄存器是32bit





1. 找到寄存器的地址，根据地址写入数据 （数据手册+中文参考手册）



1. 根据寄存器地址写入数据

想要对地址进行操作，需要利用C语言的位操作符，位运算符有6种 ~ | & ^ << >>

~ 按位取反 0变1 1变0 0110\_1101 --- 1001\_0010

& 按位与 两位同时为1，则结果为1 0 & x = 0

1001 0110

1110 0010 ------ 1000 0010

| 按位或 两位中有1位为1，则结果为1 1 | x = 1

1001 0110

1110 0010 ------ 1111 0110

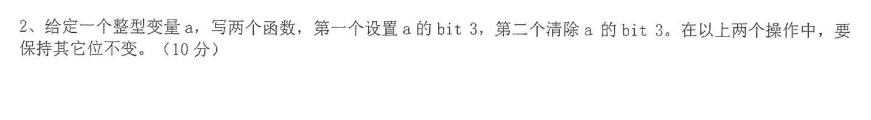
^ 按位异或 两位相同，结果为0，不同为1

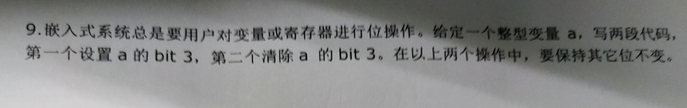
1001 0110

1110 0010 ------ 0111 0100

<< 左移运算符 高位舍弃，低位补0 0110 1101 << 3 ----- 0110 1000

>> 右移运算符 原理同左移

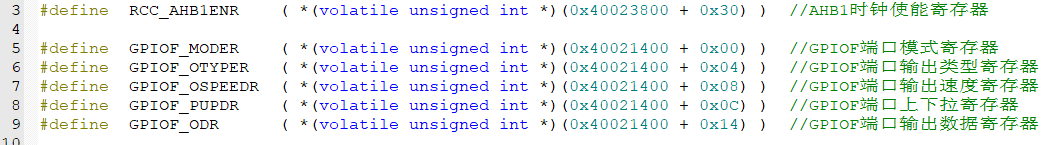




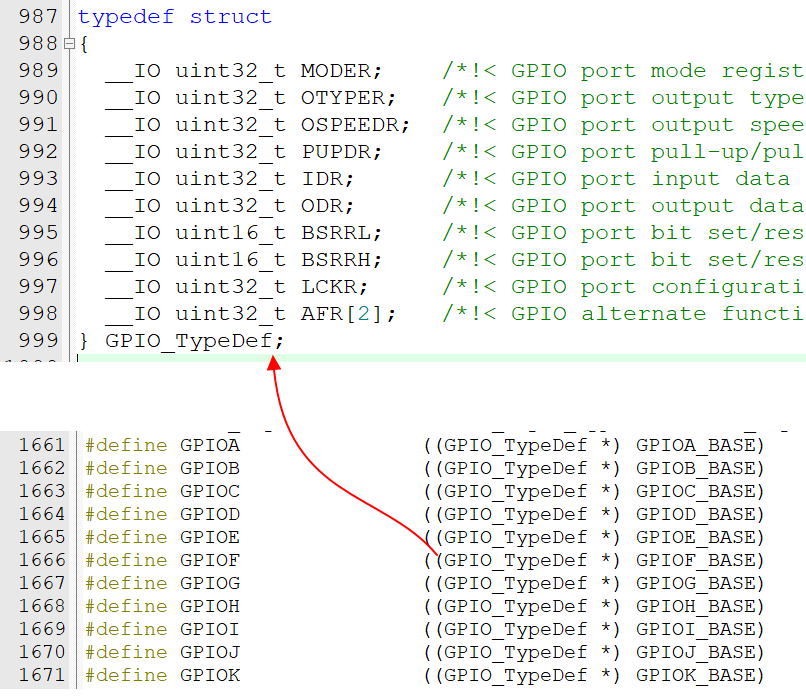
答案不唯一（提供一个最优的 可读性最好、代码精简、容易维护） 提示：使用位运算符

**把某一位清零 a &= ~(1<<3);**   **把某一位置1 a |= (1<<3);**

需要把地址先转换为指针，然后再进行操作 \*(volatile unsigned int \*)0x12345678 = 1234;



其实ST公司的开发人员把所有外设的寄存器的地址都定义在**stm32f4xx.h这个头文件中**，大家就**不需要自己定义寄存器的地址**



举例：利用寄存器的方式点亮LD0 代码如下



**练习：编写代码，利用寄存器的方式实现KEY0控制LED0 （如何检测引脚电平？？？）**

1. STM32时钟体系
2. 什么是晶振

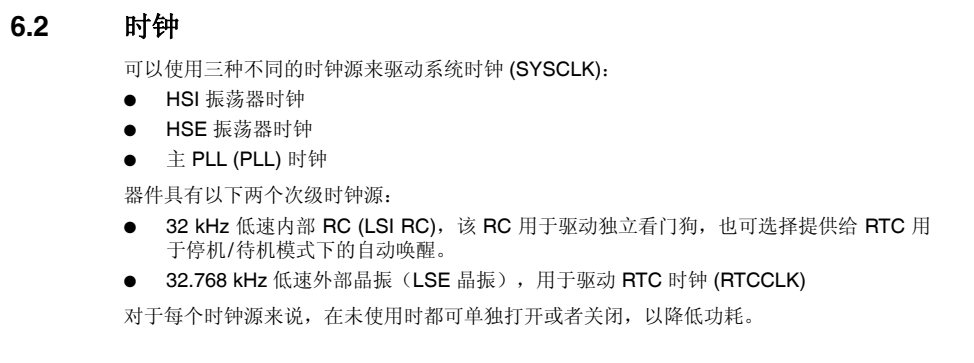
晶振的全称叫做晶体振荡器，是晶体（石英）和电子元件组成，晶振有一个非常重要的特性：机电效应（压电效应），一般晶振会提供高度稳定的频率（振荡频率是固定的），一般晶振的频率有8MHZ、12MHZ、25MHZ、11.0592MHZ...... 晶振的单位是HZ 频率（单位时间振荡的次数）



1. 什么是时钟

时钟相当于处理器的“**心脏**”，外部晶振经过振荡会产生高度稳定的信号，由芯片的引脚输入芯片内部，再经过芯片内部的频率放大器进行放大或者缩小，最终变成各种总线的时钟频率，处理器的外设都是由时钟控制的，举个例子：交响乐团 指挥

思考：时钟信号是由什么产生的？时钟信号是由**时钟源**产生的，比如外部晶振产生的时钟信号就是时钟源的一种，但是有的外设不需要太高的时钟频率，所以STM32提供了多种时钟源给用户使用。 参考STM32中文参考手册第6章 **一共有5种时钟源**



HSE ：High Speed External 高速外部时钟，特点是精度高，一般可以作为系统时钟

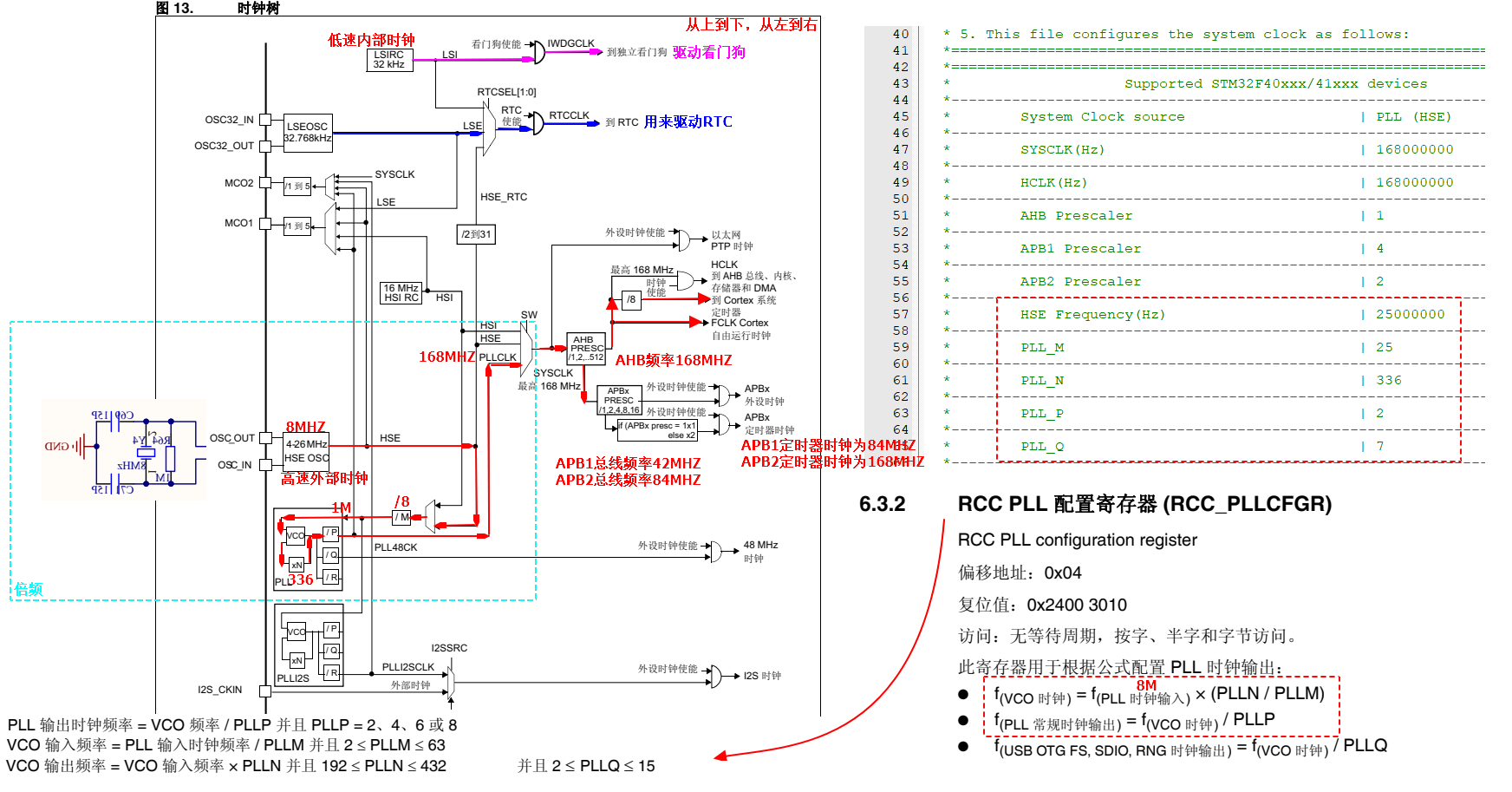
HSI ：High Speed Internal 高速内部时钟，频率为16MHZ的RC振荡器，优点是成本低，缺 点是精度低，容易受温度影响，适合小家电使用。

LSE ：Low Speed External 低速外部时钟，频率为32.768KHZ的晶振，可以驱动RTC实时时 钟，优点是功耗低、精度高。

LSI ：Low Speed Internal 低速内部时钟，频率为32KHZ的RC振荡器，可以驱动独立看门狗 和低功耗唤醒功能

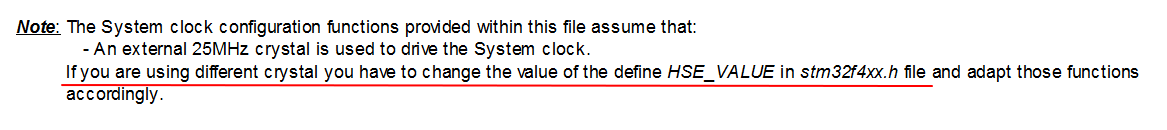
PLL ：倍频锁相环， 由 HSE 或 HSI 振荡器提供时钟信号，一般可以作为系统时钟使用。

1. STM32时钟树

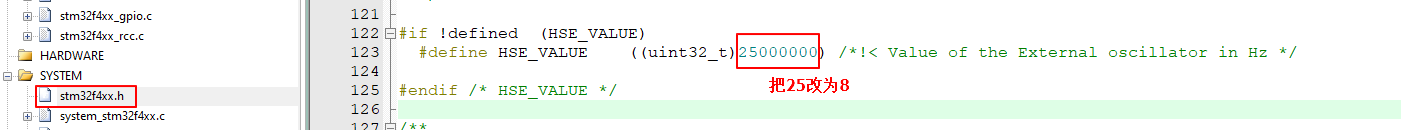


1. PLL参数的修改

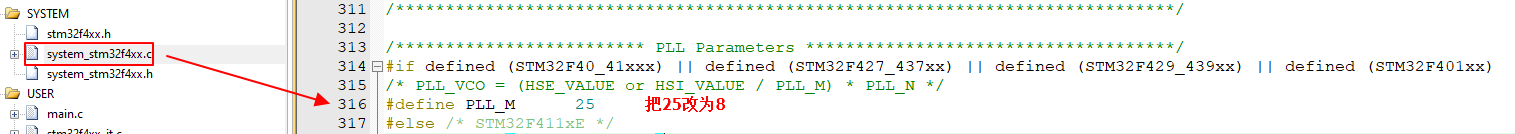
大家可以看到标准库中使用的外部晶振是25MHZ，而开发平台实际使用是8MHZ，所以如果不去修改工程的PLL参数，会导致芯片的运行频率是错误的，所以需要修改PLL参数，可以去system\_stm32f4xx.c和stm32f4xx.h两个文件中修改。



* 修改stm32f4xx.h的123行，把25改为8 修改外部晶振的频率（根据开发板）

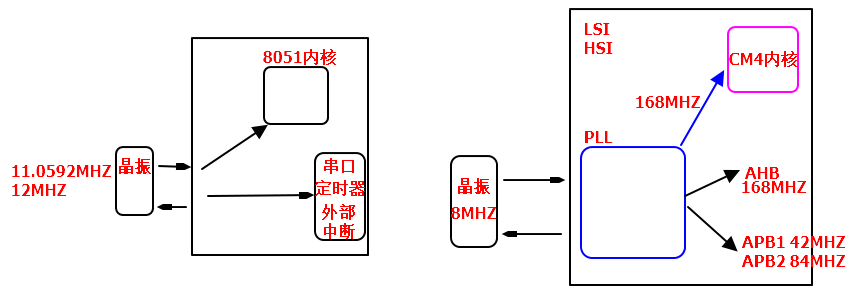


* 修改system\_stm32f4xx.c的316行的PLL参数 修改PLL\_M的值 把 25改为8



**练习：根据时钟树的参数，修改自己的工程的外部晶振频率以及PLL参数**

1. STM32与51单片机的时钟对比



但是需要注意的是，不是所有的产品都需要PLL，因为使用PLL会增大功耗，减少产品的续航能力（智能手表、手机锁屏）

项目经验：根据实际的情况来修改PLL的参数，这样就可以降低功耗或者提供性能，一般修改PLL\_N的值（192~432），针对高性能模式、平衡模式、低功耗模式（以手机为例）

高性能模式 ：如果修改PLL\_N的值为432，则主频会达到216MHZ 最高为多少MHZ？

平衡模式 ：如果修改PLL\_N的值为336，则主频会达到168MHZ

低功耗模式 ：主频可以降到84MHZ 最低能降到多少MHZ？

注意：如果芯片主频超过168MHZ，就是超频工作，会增加芯片功耗以及导致芯片发烫，谨慎使用。

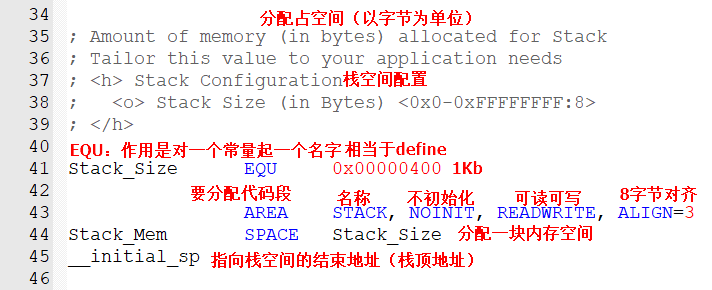
1. STM32的启动过程

一般大家都知道程序的入口是main函数，但是在主程序运行之前，是需要先运行一个启动文件xxx.s，对于STM32F407ZET6来说，汇编文件为startup\_stm32f40\_41xxx.s，当然不同的芯片型号所使用的汇编文件是不一样的。

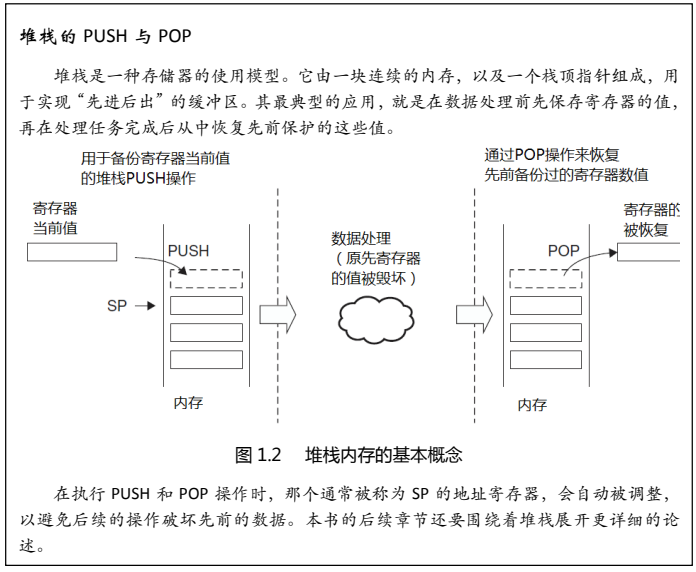
启动文件的功能如下



1. 配置系统的栈空间



栈是一种特殊的线性表，特殊在用户只能在栈的一端进行操作（栈顶），另一端操作不了，如果没有数据进入栈空间，称为空栈，如果写入数据称为入栈，如果读取数据称为出栈。

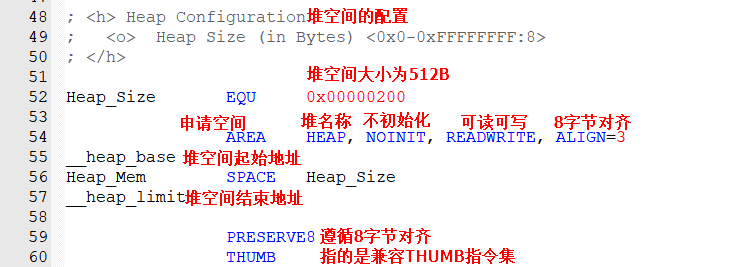


栈空间是由编译器来自动分配和释放的，栈空间一般存储局部变量、存储函数调用过程中的传递的参数、保护现场，对于启动文件中的栈空间，默认是1KB，但是如果大家写的程序比较大，用到的局部变量比较多，就必须修改栈空间的大小。

如果栈空间的大小不修改，会导致**栈溢出**，让程序出现未知问题（相当于Linux开发中的段错误）。

**注意：栈空间可以由用户手动调整，但是不能超过内部SRAM的大小（192KB）**

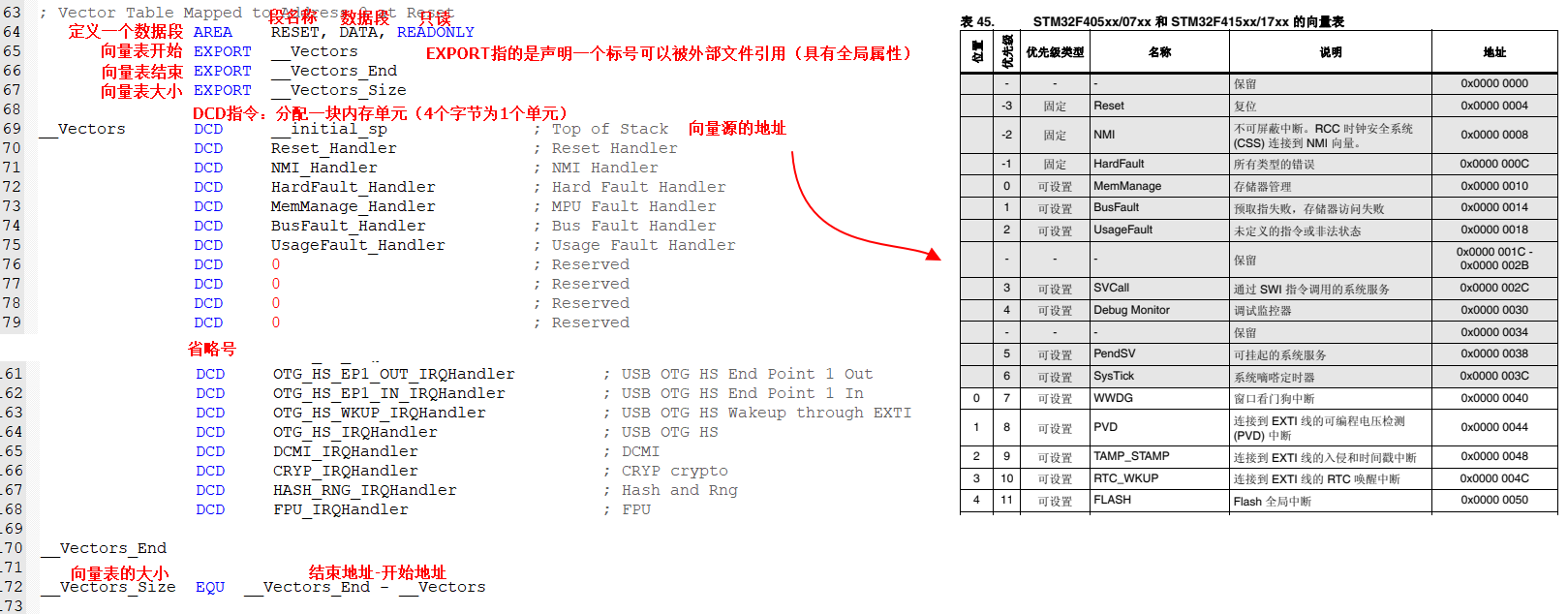
1. 配置系统的堆空间



THUMB指的是THUMB指令集，是ARM以前的指令集，THUMB指令集是16bit，ARM现在使用的是THUMB-2指令集是32bit的，也就意味着THUMB-2指令集包含THUMB指令集。

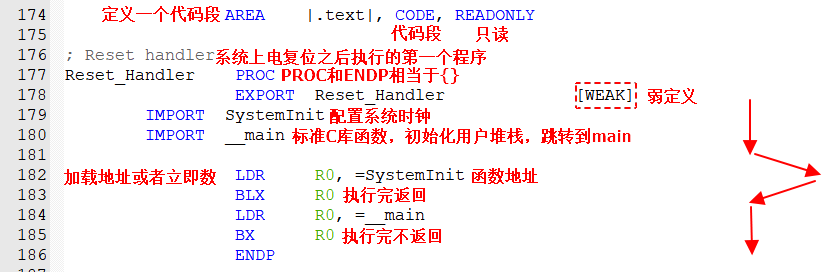
对于堆空间一般是动态分配的，由用户来分配，如Linux系统学习的malloc函数，但是堆空间在STM32中使用较少。

1. 配置系统的向量表



向量表指的是中断源表，向量其实就是内核发生的异常的源头，**启动文件中的向量表的地址都是从Flash空间的首地址开始**，0x00000000存储的是栈顶指针，0x00000004存储的是复位程序地址........向量表可以参考STM32F4中文参考手册的第10章

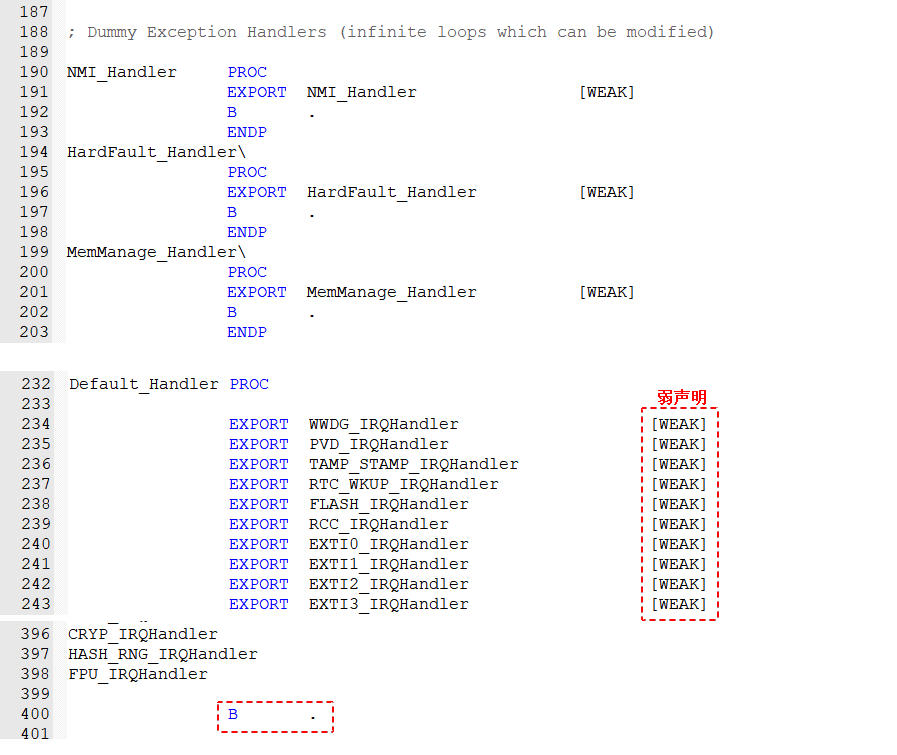
1. 复位程序分析



**【WEAK】**指的是**弱声明**，如果外部文件中重新定义了该函数，则优先使用外部文件中的，如果外部文件中没重新定义，则默认使用启动文件中。也就是说该函数不是唯一的。

思考：是否可以改变函数的入口？ 答案是可以的 可以修改为自己指定的函数接口中

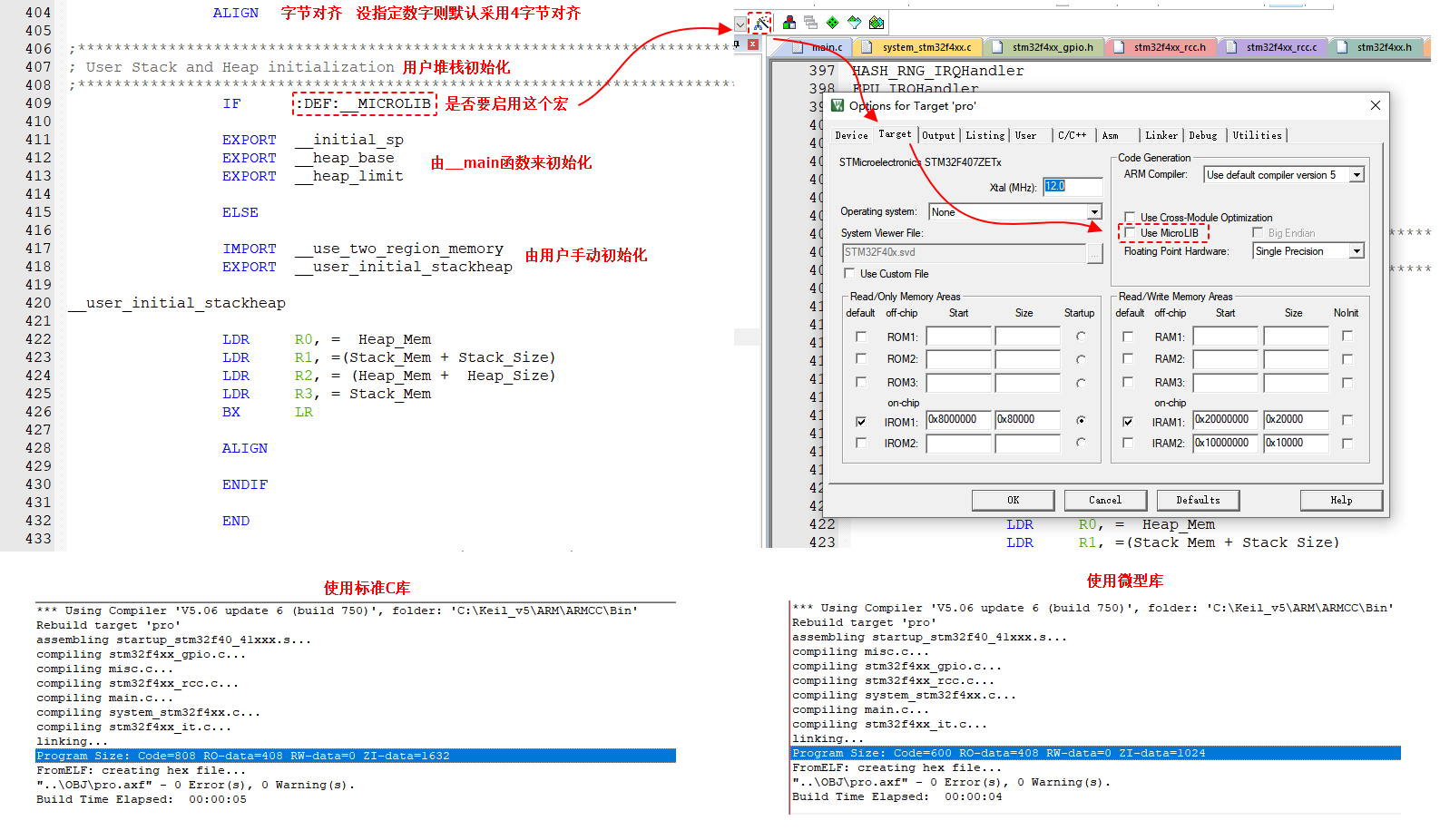
1. 中断服务函数分析



一旦发生了异常，内核就需要去响应和处理异常，需要跳转到对应的中断服务函数中执行功能，中断服务函数是由用户在外部文件中来重新定义。如果用户开启某个外设的中断，但是没有编写对应的中断服务函数或者中断服务函数名字写错了，就会导致程序一直运行汇编文件中定义的中断服务函数，但是预定义的中断服务函数内部是空的，所以就会跳转到无限循环，导致程序卡死。

**注意：用户在使用外设中断（外部中断、定时器中断、串口中断、看门狗中断.....）的时候，中断服务函数的名字必须在汇编文件中查找，不能随意定义。**

1. 用户堆栈初始化分析



如果启动MicroLIB库，这个库属于KEIL公司自带的库，会替换标准C库，这个微型库会对代码进行高度优化，所以生成的可执行文件的体积会减小，但是缺点是微型库的功能没有标准C库多。

预习：中文参考手册的第十章 中断和事件

预习：Cortex M3与M4权威指南.pdf的9.5章节 Systick定时器

预习：中文参考手册的第14、15、16、17章 定时器