零死角玩转STM32

在SRAM中调试代码

淘宝：firestm32.taobao.com

论坛： www.chuxue123.com

扫描进入淘宝店铺

主讲内容

01 在RAM中调试代码-简介

02 STM32的启动方式

03 内部FLASH的启动过程

04 实验：在内部SRAM中调试代码

参考资料:《零死角玩转STM32》

“在SRAM中调试代码”章节

在RAM中调试代码

在RAM中调试代码

一般情况下，我们在MDK中编写工程应用后，调试时都是把程序下载到芯片的内部FLASH运行测试的，代码的CODE及RW-data的内容被写入到内部FLASH中存储。但在某些应用场合下却不希望或不能修改内部FLASH的内容，这时就可以使用RAM调试功能了，它的本质是把原来存储在内部FLASH的代码(CODE及RW-data的内容)改为存储到SRAM中(内部SRAM或外部SDRAM均可)，芯片复位后从SRAM中加载代码并运行。

在RAM中调试代码

在RAM中调试代码

把代码下载到RAM中调试有如下优点：

- 下载程序非常快。RAM存储器的写入速度比在内部FLASH中要快得多，且没有擦除过程，因此在RAM上调试程序时程序几乎是秒下的，对于需要频繁改动代码的调试过程，能节约很多时间，省去了烦人的擦除与写入FLASH过程。另外，STM32的内部FLASH可擦除次数为1万次，虽然一般的调试过程都不会擦除这么多次导致FLASH失效，但这确实也是一个考虑使用RAM的因素。
- 不改写内部FLASH的原有程序。
- 对于内部FLASH被锁定的芯片，可以把解锁程序下载到RAM上，进行解锁。

在RAM中调试代码

相对地，把代码下载到RAM中调试有如下缺点：

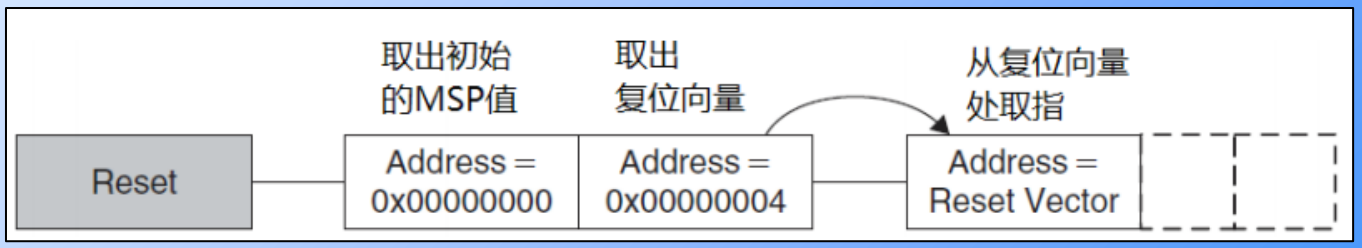
- 存储在RAM上的程序掉电后会丢失，不能像FLASH那样保存。
- 若使用STM32的内部SRAM存储程序，程序的执行速度与在FLASH上执行速度无异，但SRAM空间较小。
- 若使用外部扩展的SDRAM存储程序，程序空间非常大，但STM32读取SDRAM的速度比读取内部FLASH慢，这会导致程序总执行时间增加，因此在SDRAM中调试的程序无法完美仿真在内部FLASH运行时的环境。另外，由于STM32无法直接从SDRAM中启动且应用程序复制到SDRAM的过程比较复杂(下载程序前需要使STM32能正常控制SDRAM)，所以在很少会在STM32的SDRAM中调试程序。

在RAM中调试代码

STM32的启动方式

在前面讲解的STM32启动代码章节了解到CM-4内核在离开复位状态后的工作过程如下：

- 从地址0x00000000处取出栈指针MSP的初始值，该值就是栈顶的地址。
- 从地址0x00000004处取出程序指针PC的初始值，该值指向复位后应执行的第一条指令。



上述过程由内核自动设置运行环境并执行主体程序，因此它被称为自举过程。

在RAM中调试代码

STM32的启动方式

虽然内核是固定访问0x00000000和0x00000004地址的，但实际上这两个地址可以被重映射到其它地址空间。以STM32F429为例，根据芯片引出的BOOT0及BOOT1引脚的电平情况，这两个地址可以被映射到内部FLASH、内部SRAM以及系统存储器中，不同的映射配置如下：

BOOT1	BOOT0	映射到的存储器	0x00000000 地址映射到	0x00000004 地址映射到
x	0	内部FLASH	0x08000000	0x08000004
1	1	内部SRAM	0x20000000	0x20000004
0	1	系统存储器	0x1FFF0000	0x1FFF0004

在RAM中调试代码

STM32的启动方式

内核在离开复位状态后会从映射的地址中取值给栈指针MSP及程序指针PC，然后执行指令，一般以存储器的类型来区分自举过程，例如内部FLASH启动方式、内部SRAM启动方式以及系统存储器启动方式。

- **内部FLASH启动方式**

当芯片上电后采样到BOOT0引脚为低电平时， 0x00000000和0x00000004地址被映射到内部FLASH的首地址0x08000000和0x08000004。因此，内核离开复位状态后，读取内部FLASH的0x08000000地址空间存储的内容，赋值给栈指针MSP，作为栈顶地址，再读取内部FLASH的0x08000004地址空间存储的内容，赋值给程序指针PC，作为将要执行的第一条指令所在的地址。具备这两个条件后，内核就可以开始从PC指向的地址中读取指令执行了。

在RAM中调试代码

STM32的启动方式

- **内部SRAM启动方式**

类似地，当芯片上电后采样到BOOT0和BOOT1引脚均为高电平时，0x00000000和0x00000004地址被映射到内部SRAM的首地址0x20000000和0x20000004，内核从SRAM空间获取内容进行自举。

在实际应用中，由启动文件starttup_stm32f429_439xx.s决定了0x00000000和0x00000004地址存储什么内容，链接时，由分散加载文件(sct)决定这些内容的绝对地址，即分配到内部FLASH还是内部SRAM。（下一小节将以实例讲解）

在RAM中调试代码

STM32的启动方式

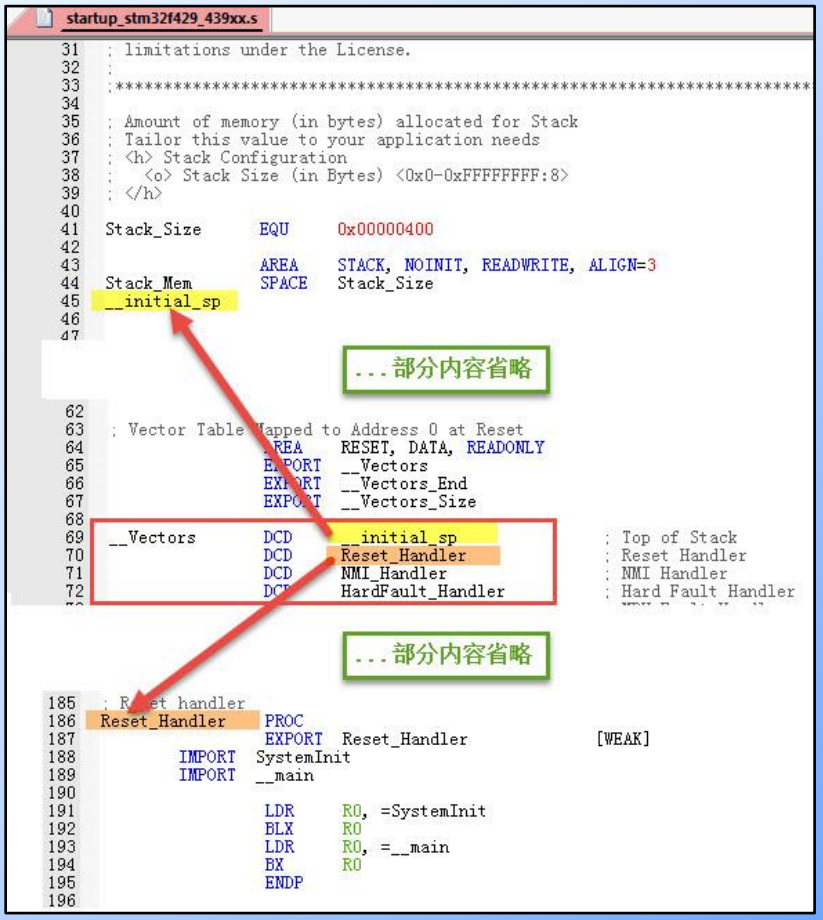
- 系统存储器启动方式

当芯片上电后采样到BOOT0引脚为高电平，BOOT1为低电平时，内核将从系统存储器的0x1FFF0000及0x1FFF0004获取MSP及PC值进行自举。系统存储器是一段特殊的空间，用户不能访问，ST公司在芯片出厂前就在系统存储器中固化了一段代码。因而使用系统存储器启动方式时，内核会执行该代码，该代码运行时，会为ISP提供支持(In System Program)，如检测USART1/3、CAN2及USB通讯接口传输过来的信息，并根据这些信息更新自己内部FLASH的内容，达到升级产品应用程序的目的，因此这种启动方式也称为ISP启动方式。

在RAM中调试代码

内部FLASH的启动过程

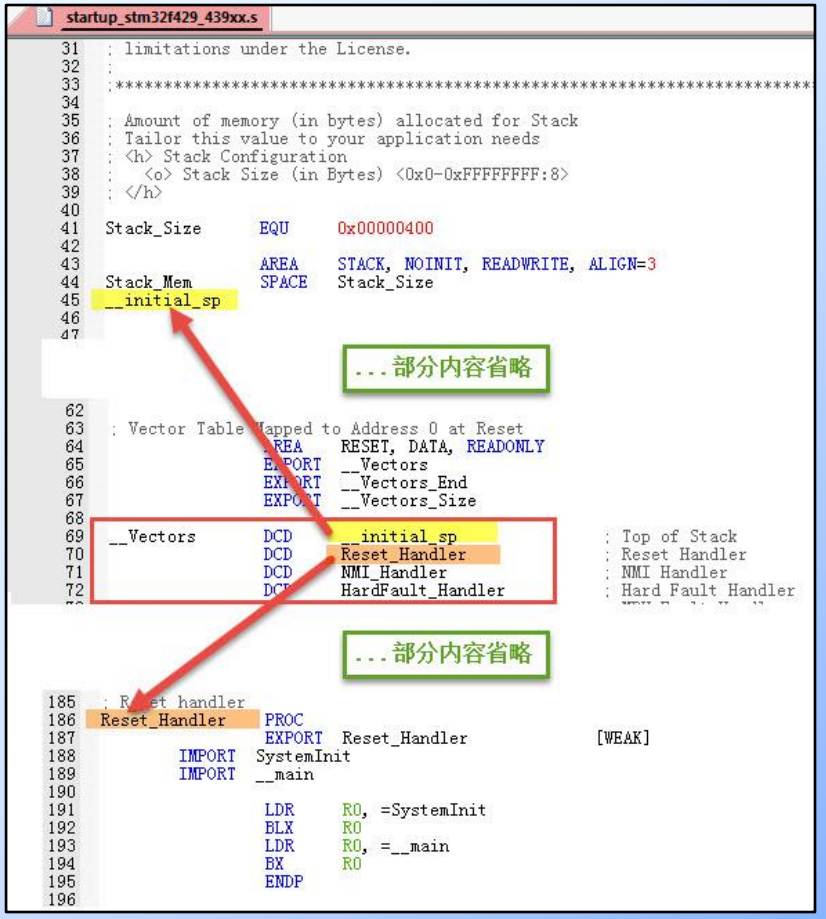
下面以最常规的内部FLASH启动方式来分析自举过程，主要理解MSP和PC内容是怎样被存储到0x08000000和0x08000004这两个地址的。



STM32F4默认的启动文件的代码

在RAM中调试代码

内部FLASH的启动过程



- 启动文件的开头定义了一个大小为0x400的栈空间，且栈顶的地址使用标号“__initial_sp”来表示；
- 在图下方定义了一个名为“Reset_Handler”的子程序，它就是芯片启动后第一个执行的代码。在汇编语法中，程序的名字和标号都包含它所在的地址，因此，它的目标是把“__initial_sp”和“Reset_Handler”赋值到0x08000000和0x08000004地址空间存储，这样内核自举的时候就可以获得栈顶地址以及第一条要执行的指令了。
- 在启动代码的中间部分，使用了汇编关键字“DCD”把“__initial_sp”和“Reset_Handler”定义到了最前面的地址空间。

在RAM中调试代码

内部FLASH的启动过程

在启动文件中把设置栈顶及首条指令地址到了最前面的地址空间，但这并没有指定绝对地址，各种内容的绝对地址是由链接器根据分散加载文件(*.sct)分配的，STM32F429IGT6型号的默认分散加载文件配置如下：

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x08000000 0x00100000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00100000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW IRAM1 0x20000000 UNINIT 0x00030000  {  ; RW data
   .ANY (+RW +ZI)
  }
}
```

在RAM中调试代码

内部FLASH的启动过程

```
1  ; *************************************************************
2  ; *** Scatter-Loading Description File generated by uVision ***
3  ; *************************************************************
4
5  LR_IROM1 0x08000000 0x00100000  {    ; load region size_region
6    ER_IROM1 0x08000000 0x00100000  {  ; load address = execution address
7     *.o (RESET, +First)
8     *(InRoot$$Sections)
9     .ANY (+RO)
10   }
11   RW IRAM1 0x20000000 UNINIT 0x00030000  {  ; RW data
12    .ANY (+RW +ZI)
13   }
14 }
15
```

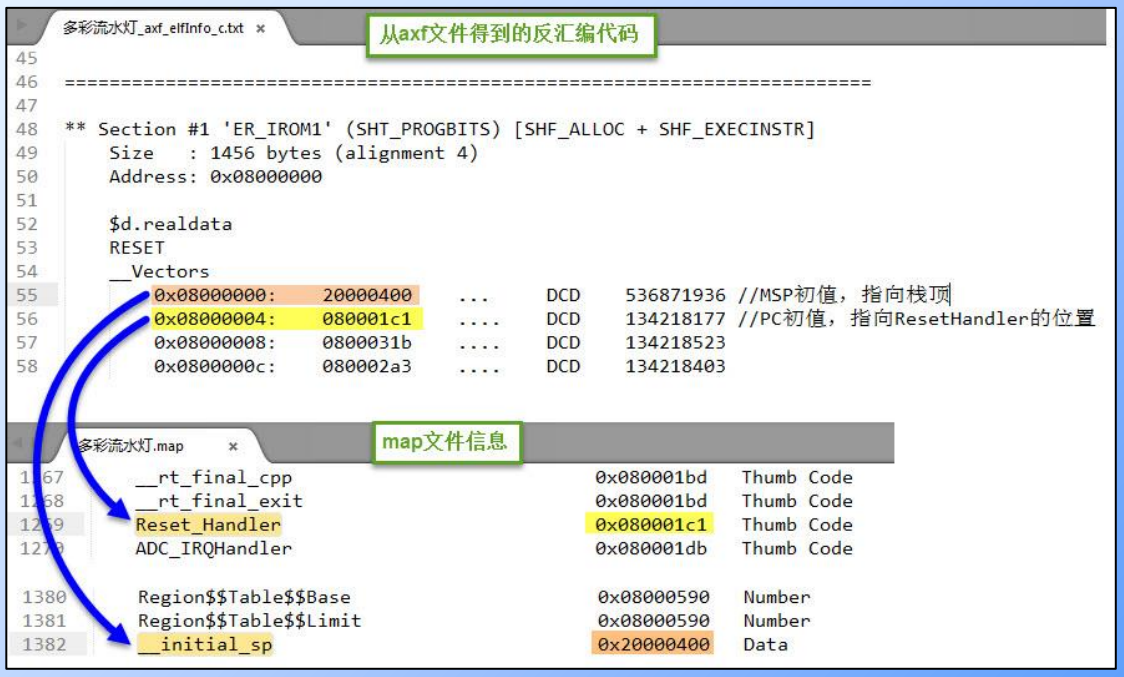
分散加载文件把加载区和执行区的首地址都设置为0x08000000，正好是内部FLASH的首地址，因此汇编文件中定义的栈顶及首条指令地址会被存储到0x08000000和0x08000004的地址空间。

类似地，如果修改分散加载文件，把加载区和执行区的首地址设置为内部SRAM的首地址0x20000000，那么栈顶和首条指令地址将会被存储到0x20000000和0x20000004的地址空间了。

在RAM中调试代码

内部FLASH的启动过程

可以查看反汇编代码及map文件信息来了解各个地址空间存储的内容：



这是多彩流水灯工程编译后的信息，它的启动文件及分散加载文件都按默认配置。其中反汇编代码是使用fromelf工具从axf文件生成的。

在RAM中调试代码

内部FLASH的启动过程

- 可了解到，这个工程的0x08000000地址存储的值为0x20000400，0x08000004地址存储的值为0x080001C1，查看map文件，这两个值正好是栈顶地址__initial_sp以及首条指令Reset_Handler的地址。下载器会根据axf文件(bin、hex类似)存储相应的内容到内部FLASH中。
- 由此可知，BOOT0为低电平时，内核复位后，从0x08000000读取到栈顶地址为0x20000400，了解到子程序的栈空间范围，再从0x08000004读取到第一条指令的存储地址为0x080001C1，于是跳转到该地址执行代码，即从ResetHandler开始运行，运行SystemInit、__main(包含分散加载代码)，最后跳转到C语言的main函数。
- 对比在内部FLASH中运行代码的过程，可了解到若希望在内部SRAM中调试代码，需要设置启动方式为从内部SRAM启动，修改分散加载文件控制代码空间到内部SRAM地址以及把生成程序下载到芯片的内部SRAM中。

THANKS

论坛： www.chuxue123.com

淘宝：firestm32.taobao.com

扫描进入淘宝店铺