STM32开发启示录

# 1 简介

以STM32为例，深入了解计算机原理。

# 2 STM32内存分配深入分析

以STM32F103RCT6芯片，keil4.7编译平台进行分析。

## 2.1 CotexM3与STM32存储器映射

存储器映射是指把芯片中或芯片外的FLASH，RAM，外设，BOOT,BLOCK等进行**统一编址。**即用地址来表示对象。这个地址绝大多数是由厂家规定好的，用户只能用而不能改。用户只能在挂外部RAM或FLASH的情况下可进行自定义。

### 2.1.1 Cortex-M3的存储器映射

Cortex-M3支持4GB的存储空间，它的存储系统采用统一编址的方式，程序存储器、数据存储器、寄存器被组织在4GB的线性地址空间内，以小端格式(little-endian)存放。由于Cortex-M3是32位的内核，因此其PC指针可以指向2^32=4G的地址空间，也就是0x0000\_0000——0xFFFF\_FFFF这一大块空间。见图1：

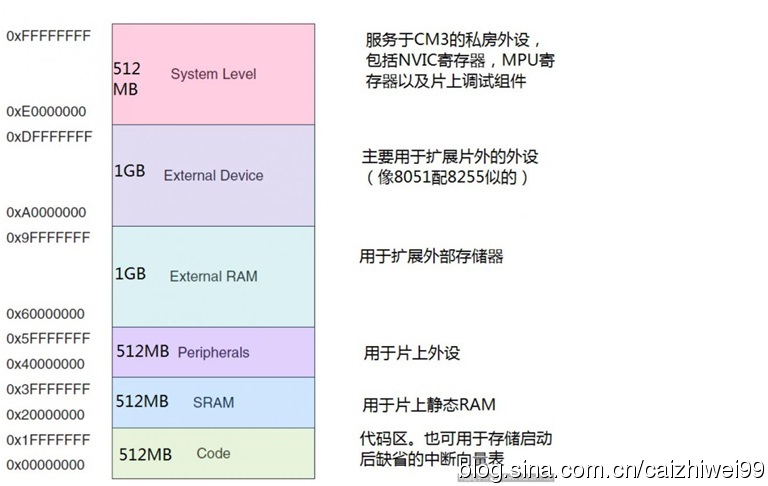


图1-1 Cortex-M3的存储器映射

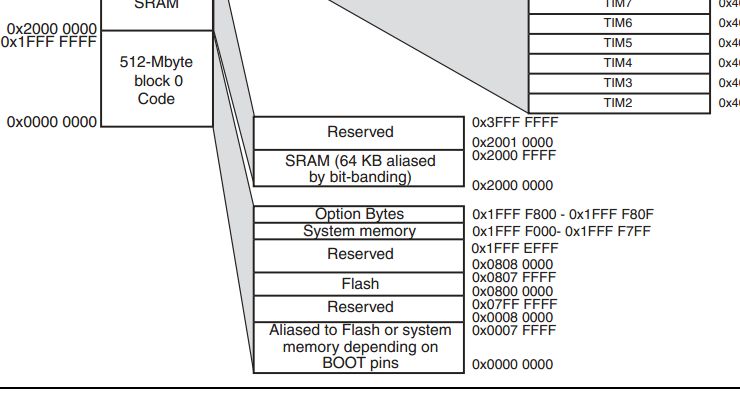


图1-2存储器映射

Cortex-M3内核将0x0000\_0000——0xFFFF\_FFFF这块4G大小的空间分成8大块：代码、SRAM、外设、外部RAM、外部设备、专用外设总线-内部、专用外设总线-外部、特定厂商（见图1-1）。这就导致了，使用该内核的芯片厂家必须按照这个进行各自芯片的存储器结构设计。

### 2.1.2 Cortex-M3与中密度stm32的存储器映射对比

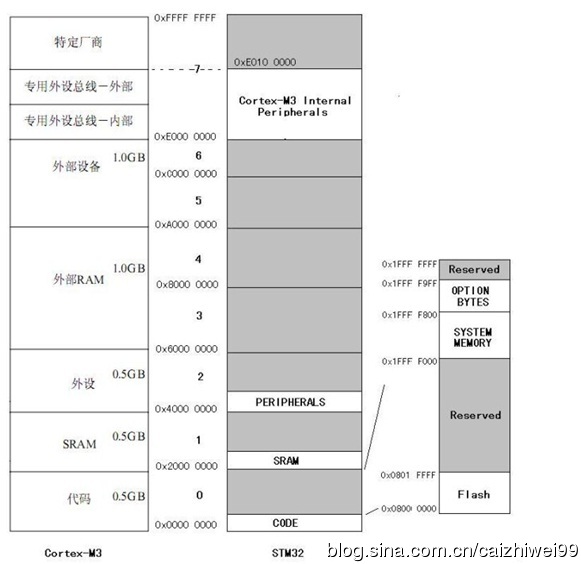


图2 Cortex-M3与中密度stm32的存储器映射对比

图2可以很清晰的看到，STM32的存储器结构和Cortex-M3的很相似，不同的是，STM32加入了很多实际的东西，如：Flash、SRAM等。只有加入了这些东西，才能成为一个拥有实际意义的、可以工作的处理芯片——STM32。**STM32的存储器地址空间被划分为大小相等的8块区域**，每块区域大小为512MB(如：0x20000000~0x40000000)。对STM32存储器知识的掌握，实际上就是对Flash和SRAM这两个区域知识的掌握。

不同类型的STM32单片机的SRAM大小是不一样的，但是他们的起始地址都是0x2000 0000，终止地址都是0x2000 0000+其固定的容量大小。**SRAM的理解比较简单，其作用是用来存取各种动态的输入输出数据、中间计算结果以及与外部存储器交换的数据和暂存数据。设备断电后，SRAM中存储的数据就会丢失。**

## 2.2 STM32F103RCT6存储资源

### 2.2.1 嵌入式SRAM

STM32F10xxx内置64K字节的静态SRAM。它可以以字节、半字(16位)或全字(32位)访问。SRAM的起始地址是0x2000 0000。

### 2.2.2 STM32 FLASH

STM32的Flash，严格说，应该是Flash模块。三个分区的称呼与datasheet保持一致。该Flash模块包括：

#### （1）FLASH主存储块（Main memory）

**存放代码的地方**，如图2中的FLASH区域：128KB（0x08000000~0x0801ffff）（不同容量的Flash终止地址不同）；

解读：每页1KB，故FLASH擦除时是以1KB为单位，同时起始地址边界对齐。

注：STM32内部flash为norflash（ram like），区分了地址总线和数据总线，可直接寻址， 而nandflash没有区分地址总线和数据总线，通过总线IO口+控制线的方式。

参考资料：<http://blog.csdn.net/changqiang08/article/details/8102861>

#### （2）FLASH信息块（Information block）

该区域又可以分为Option Bytes和System Memory区域；System Memory：STM32在出厂时，已经固化了一段程序在System memory（medium-density devices的地址为：0x1FFF\_F000，大小为2KB）存储器中。这段程序就是一个固定好的，并且没法修改的Boot Loader（见编程手册PM0042这种描述）。Option Bytes：可以按照用户的需要进行配置（如配置看门狗为硬件实现还是软件实现）；该区域除了互联型所用型号地址都一样：（0x1fff\_f000~0x1fff\_f80f）图中终止地址有误：应为0x1fff\_f80f,正好16个字节。

信息块又分成两部分：系统存储器(2K)、选项字节(16bytes)。 总容量为：258×64位 = 2048bytes + 16bit = 2K+16bit

1）系统存储器

存储用于存放在系统存储器自举模式下的启动程序（BootLoader），当使用ISP方式加载程序时，就是由这个程序执行。这个区域由芯片厂写入BootLoader，然后锁死，用户是无法改变这个区域的。

**解读：ISP、ICP、IAP**

1、ISP：In System Programing 在系统编程

2、IAP：In applicating Programing 在应用编程

3、ICP：In Circuit Programing 在电路编程

@ ISP下载，例如：串口1和4、CAN等方式下载，官方下载软件Flash\_Loader\_Demonstrator\_V1.3\_Setup.exe。芯片厂家将代码固化好了在Flash中。

@ IAP，实际工程常用的程序升级方式

@ ICP，JTAG、JLINK等烧写工具烧写，调试代码。

2）选项字节

存储芯片的配置信息及对主存储块的保护信息。



#### （3）FLASH存储接口寄存器区（Flash memory interface）

Flash memory interface，用于片上外设。是图2中从0x40000000开始的PERIPHERALS区域。也称作外设存储器映射，对该区域操作，就是对相应的外设进行操作。

## 2.2 STM3210x启动配置

### 2.3.1 启动模式

在STM32F10xxx里，可以通过BOOT[1:0]引脚选择三种不同启动模式。

|  |  |  |  |
| --- | --- | --- | --- |
| **启动模式选择引脚** | | **启动模式** | **说明** |
| **BOOT1** | **BOOT0** |
| X | 0 | 主闪存存储器 | 主闪存存储器被选为启动区域 |
| 0 | 1 | 系统存储器 | 系统存储器被选为启动区域 |
| 1 | 1 | 内置SRAM | 内置SRAM被选为启动区域 |

* 在系统复位后， SYSCLK的第4个上升沿， BOOT引脚的值将被锁存。用户可以通过设置BOOT1和BOOT0引脚的状态，来选择在复位后的启动模式。
* 在从待机模式退出时， BOOT引脚的值将被被重新锁存；因此，在待机模式下BOOT引脚应保持为需要的启动配置。
* 在启动延迟之后， CPU从地址0x0000 0000获取堆栈顶的地址，并从启动存储器的0x0000 0004指示的地址开始执行代码。

**代码区：**因为固定的存储器映像，代码区始终从地址0x0000 0000开始(通过ICode和

DCode总线访问)。

**数据区：**数据区(SRAM)始终从地址0x2000 0000开始(通过系统总线访问)。

### 2.3.2 访问方式

根据选定的启动模式，主闪存存储器、系统存储器或SRAM可以按照以下方式访问：

#### （1）从主闪存存储器启动

主闪存存储器被映射到启动空间(0x0000 0000)，**但仍然能够在它原有的地址(0x0800 0000)访问它**，即闪存存储器的内容可以在两个地址区域访问， 0x0000 0000或0x0800 0000。

#### （2）从系统存储器启动

系统存储器被映射到启动空间(0x0000 0000)，但仍然能够在它原有的地址(互联型产品原有地址为0x1FFF B000，其它产品原有地址为0x1FFF F000)访问它。

#### （3）从内置SRAM启动

只能在0x2000 0000开始的地址区访问SRAM。

解读：

主存储块起始地址为0x0800 0000，如果BOOT模式是从主闪存存储器(内部FLASH)启动，启动时，主闪存存储器被映射到启动空间(0x0000 0000)，这样就可以访问0x0800 0000开始的FLASH代码了。

流程：

系统复位 -> SYSCLK第4个上升沿，锁定BOOT[1:0]引脚电平 -> 启动模式确定，从主闪存存储器启动 -> CPU从地址0x0000 0000获取堆栈顶的地址(0x0800 0000被映射到0x0000 0000) -> 并从启动存储器的0x0000 0004(0x0800 0004被映射到0x0000 0004)指示的地址开始执行代码

注：本实验基于MCU从主闪存存储器启动

### 2.3.3 疑问

**Q:** 理论上，CM3中规定上电后CPU是从0地址开始执行，但是这里中断向量表却被烧写在0x0800 0000地址里，那启动时不就找不到中断向量表了？

**A:** 既然CM3定下的规矩是从0地址启动，SMT32当然不能破坏ARM定下的“规矩”，所以它做了一个启动映射的过程，就是和芯片上总能见到的BOOT0和BOOT1有关了，当选择从主Flash启动模式后，芯片一上电，Flash的0x0800 0000地址**被映射到0地址处**，不影响CM3内核的读取，所以这时的CM3既可以在0地址处访问中断向量表，也可以在0x0800 0000地址处访问中断向量表，而代码还是在0x0800 0000地址处存储的。

其实，这是基本上所有ARM芯片采用的启动映射方法。ARM7，ARM9D内部Flash的通常都是这样做的。

还要注意，这个中断向量表是可以在程序中再次被映射的。控制它的就是CM3已经规定的NVIC寄存器SCB->VTOR。在STM32库中给出的启动代码里，startup\_stm32f10x\_hd.s文件里，第146行，是上电后读取中断向量表中的复位中断位置，并执行复位中断处理代码，代码如下：

; Reset handler

Reset\_Handler PROC

EXPORT Reset\_Handler [WEAK]

IMPORT \_\_main

IMPORT SystemInit

LDR R0, =SystemInit

BLX R0

LDR R0, =\_\_main

BX R0

ENDP

注意复位后第一个被执行的是SystemInit代码，这个代码在库目录下的system\_stm32f10x.c文件里，它初始化了时钟，NVIC等一系列操作，这里摘要与中断向量有关的代码：

void SystemInit (void)

{

......

#ifdef VECT\_TAB\_SRAM

SCB->VTOR = SRAM\_BASE | VECT\_TAB\_OFFSET; /\* Vector Table Relocation in Internal SRAM. \*/

#else

SCB->VTOR = FLASH\_BASE | VECT\_TAB\_OFFSET; /\* Vector Table Relocation in Internal FLASH. \*/

// VECT\_TAB\_OFFSET = 0

#endif

}

可以看出中断向量重映射是一个选择性编译，通常宏定义VECT\_TAB\_SRAM都没有被定义，所以这里执行结束后，SCB->VTOR就是FLASH\_BASE了，值为0x0800 0000。以后CM3再取中断向量里，就会根据SCB->VTOR的设置，从这里取向量执行了。中断向量自此终于转正。

注意这时连\_\_main函数都还没进。

# 3 STM32启动分析

见印象笔记《STM32启动分析》

# 4 程序运行与内存分配

## 4.1 程序运行

**问题1：STM32代码是在flash中运行还是在ram中运行？**

（1）对于x86的pc机和单片机等嵌入式开发系统程序的存储是截然相反的。

x86的pc机cpu在**运行的时候**程序是存储在RAM中的；而单片机等嵌入式系统则

是存于flash中。

pc机在运行程序的时候将程序从外存（硬盘）中，调入到RAM中运行，cpu从RAM中读取程序和数据； 而单片机的程序则是固化在flash中，cpu运行时直接从flash中读取程序，从RAM中读取数据。

（2）造成这种现象的原因是什么？

x86构架的cpu是基于冯.诺依曼体系的，即**数据和程序存储在一起**，而且pc机的RAM资源相当丰富，从几十M到几百M甚至是几个G，客观上能够承受大量的程序数据。

单片机的构架大多是哈弗体系的，即**程序和数据分开存储**，而且单片的片内RAM资源是相当有限的，内部的RAM过大会带来成本的大幅度提高。

通过以上分析可知，单片机的程序（比如.bin文件，这里的程序包括代码和数据）存储在flash中，CPU（*解释计算机指令以及处理计算机软件中的数据*）上电后将flash中的数据加载到RAM中，代码仍在Flash中，整个代码是在flash中运行，这套机制是基于体系架构和RAM资源的大小考虑的。

（3）单片机程序不能在RAM中运行吗？

以STM32为例，STM32代码是可以在RAM中运行的，主要有两种方式。一，硬件修改BOOT[1:0]脚启动方式为RAM启动；二，在keil中选中.c文件，将其设置为在RAM中运行。

参考资料：<http://blog.csdn.net/denghuanhuandeng/article/details/6904728> 很好的资料

## 4.2 内存分配

### （1）固化

代码都是烧写到flash中，运行的话，不同体系结构有不同的运行机制，见4.1。

### （2）程序运行

# 3 从keil编译信息xxx.map分析STM32中的内存分配