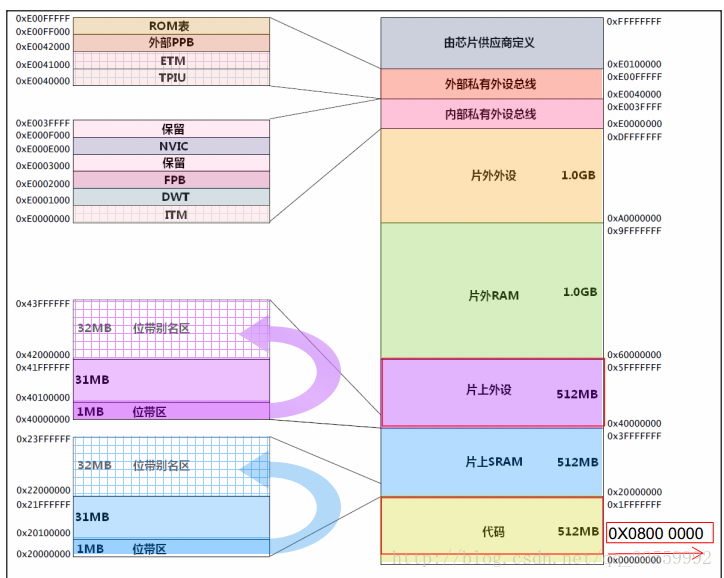
# **[STM32学习笔记：读写内部Flash（介绍+附代码）](https://www.cnblogs.com/pertor/p/9484663.html" \o "发布于 2018-08-16 15:36)**

**一、介绍**

首先我们需要了解一个内存映射：



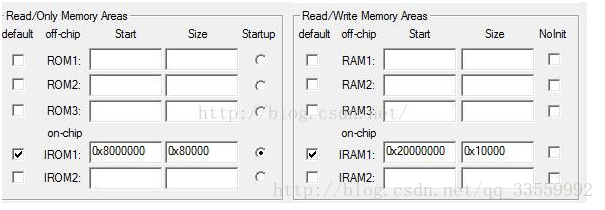
stm32的flash地址起始于0x0800 0000，结束地址是0x0800 0000加上芯片实际的flash大小，不同的芯片flash大小不同。

RAM起始地址是0x2000 0000，结束地址是0x2000 0000加上芯片的RAM大小。不同的芯片RAM也不同。

Flash中的内容一般用来存储代码和一些定义为const的数据，断电不丢失，   
RAM可以理解为内存，用来存储代码运行时的数据，变量等等。掉电数据丢失。

STM32将外设等都映射为地址的形式，对地址的操作就是对外设的操作。   
stm32的外设地址从0x4000 0000开始，可以看到在库文件中，是通过基于0x4000 0000地址的偏移量来操作寄存器以及外设的。

一般情况下，程序文件是从 0x0800 0000 地址写入，这个是STM32开始执行的地方，0x0800 0004是STM32的中断向量表的起始地址。   
在使用keil进行编写程序时，其编程地址的设置一般是这样的：

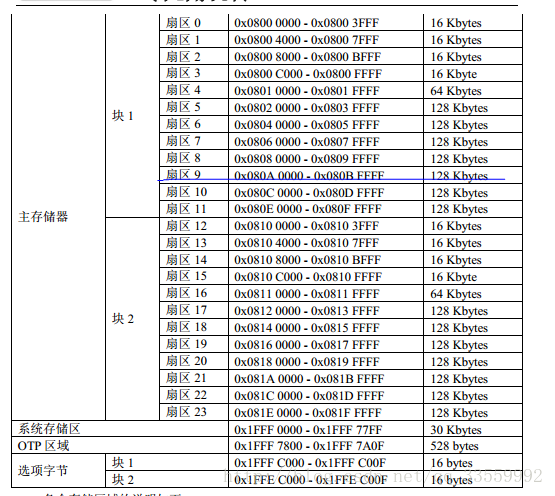


程序的写入地址从0x08000000（数好零的个数）开始的，其大小为0x80000也就是512K的空间，换句话说就是告诉编译器flash的空间是从0x08000000-0x08080000，RAM的地址从0x20000000开始，大小为0x10000也就是64K的RAM。这与STM32的内存地址映射关系是对应的。

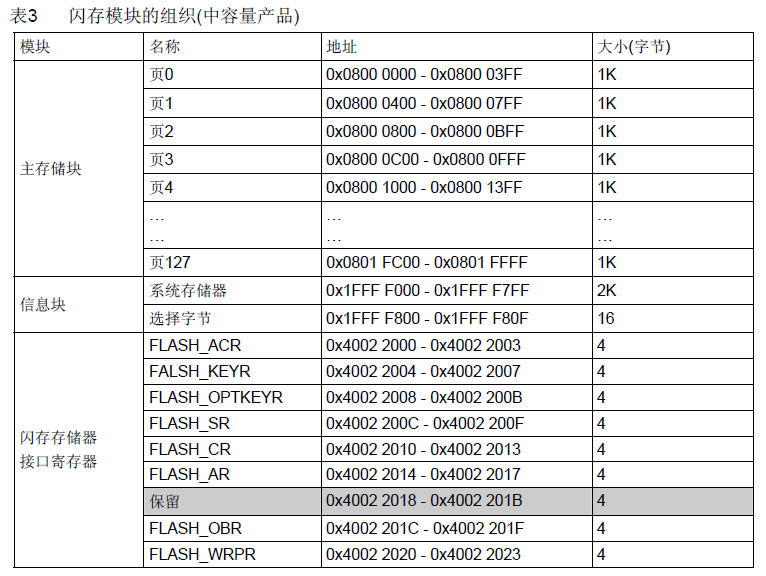
M3复位后，从0x08000004取出复位中断的地址，并且跳转到复位中断程序，中断执行完之后会跳到我们的main函数，main函数里边一般是一个死循环，进去后就不会再退出，当有中断发生的时候，M3将PC指针强制跳转回中断向量表，然后根据中断源进入对应的中断函数，执行完中断函数之后，再次返回main函数中。大致的流程就是这样。

#### **1.1、内部Flash的构成：**

STM32F429 的内部 FLASH 包含主存储器、系统存储器、 OTP 区域以及选项字节区域，它们的地址分布及大小如下：

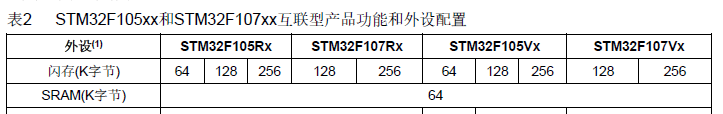


STM32F103的中容量内部 FLASH 包含主存储器、系统存储器、 OTP 区域以及选项字节区域，它们的地址分布及大小如下：





注意STM32F105VC的是有64K或128页x2K=256k字节的内置闪存存储器，用于存放程序和数据。



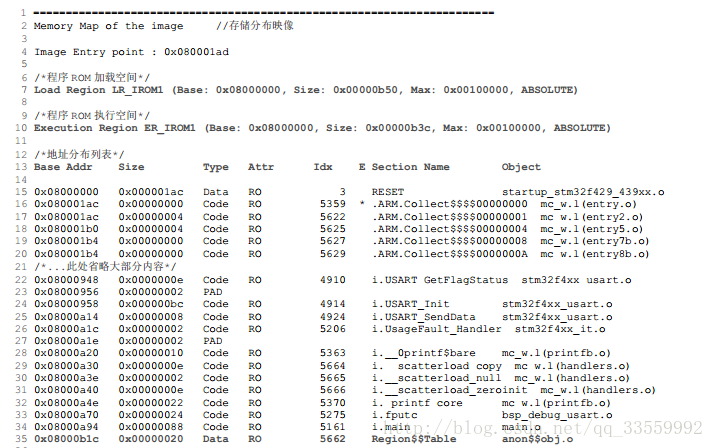
1. 主存储器：一般我们说 STM32 内部 FLASH 的时候，都是指这个主存储器区域它是存储用户应用程序的空间，芯片型号说明中的 1M FLASH、 2M FLASH 都是指这个区域的大小。与其它 FLASH 一样，在写入数据前，要先按扇区擦除，
2. 系统存储区：系统存储区是用户不能访问的区域，它在芯片出厂时已经固化了启动代码，它负责实现串口、 USB 以及 CAN 等 ISP 烧录功能。
3. OTP 区域：OTP(One Time Program)，指的是只能写入一次的存储区域，容量为 512 字节，写入后数据就无法再更改， OTP 常用于存储应用程序的加密密钥。
4. 选项字节：选项字节用于配置 FLASH 的读写保护、电源管理中的 BOR 级别、软件/硬件看门狗等功能，这部分共 32 字节。可以通过修改 FLASH 的选项控制寄存器修改。

#### **1.2、对内部Flash的写入过程：**

1. 解锁 （固定的KEY值）  
(1) 往 Flash 密钥寄存器 FLASH\_KEYR 中写入 KEY1 = 0x45670123   
(2) 再往 Flash 密钥寄存器 FLASH\_KEYR 中写入 KEY2 = 0xCDEF89AB   
2. 数据操作位数   
最大操作位数会影响擦除和写入的速度，其中 64 位宽度的操作除了配置寄存器位外，还需要在 Vpp 引脚外加一个 8-9V 的电压源，且其供电间不得超过一小时，否则 FLASH可能损坏，所以 64 位宽度的操作一般是在量产时对 FLASH 写入应用程序时才使用，大部分应用场合都是用 32 位的宽度。   
3. 擦除扇区   
在写入新的数据前，需要先擦除存储区域， STM32 提供了扇区擦除指令和整个FLASH 擦除(批量擦除)的指令，批量擦除指令仅针对主存储区。   
扇区擦除的过程如下：   
(1) 检查 FLASH\_SR 寄存器中的“忙碌寄存器位 BSY”，以确认当前未执行任何   
Flash 操作；   
(2) 在 FLASH\_CR 寄存器中，将“激活扇区擦除寄存器位 SER ”置 1，并设置“扇   
区编号寄存器位 SNB”，选择要擦除的扇区；   
(3) 将 FLASH\_CR 寄存器中的“开始擦除寄存器位 STRT ”置 1，开始擦除；   
(4) 等待 BSY 位被清零时，表示擦除完成。   
4. 写入数据   
擦除完毕后即可写入数据，写入数据的过程并不是仅仅使用指针向地址赋值，赋值前还还需要配置一系列的寄存器，步骤如下：   
(1) 检查 FLASH\_SR 中的 BSY 位，以确认当前未执行任何其它的内部 Flash 操作；   
(2) 将 FLASH\_CR 寄存器中的 “激活编程寄存器位 PG” 置 1；   
(3) 针对所需存储器地址（主存储器块或 OTP 区域内）执行数据写入操作；   
(4) 等待 BSY 位被清零时，表示写入完成。

#### **1.3、查看工程内存的分布：**

由于内部 FLASH 本身存储有程序数据，若不是有意删除某段程序代码，一般不应修改程序空间的内容，所以在使用内部 FLASH 存储其它数据前需要了解哪一些空间已经写入了程序代码，存储了程序代码的扇区都不应作任何修改。通过查询应用程序编译时产生的“ \*.map”后缀文件，   
打开 map 文件后，查看文件最后部分的区域，可以看到一段以“ Memory Map of the   
image”开头的记录(若找不到可用查找功能定位)，



【注】ROM加载空间

这一段是某工程的 ROM 存储器分布映像，在 STM32 芯片中， ROM 区域的内容就是 指存储到内部 FLASH 的代码。   
在上面 map 文件的描述中，我们了解到加载及执行空间的基地址(Base)都是0x08000000，它正好是 STM32 内部 FLASH 的首地址，即 STM32 的程序存储空间就直接是执行空间；它们的大小(Size)分别为 0x00000b50 及 0x00000b3c，执行空间的 ROM 比较小的原因就是因为部分 RW-data 类型的变量被拷贝到 RAM 空间了；它们的最大空间(Max)均为 0x00100000，即 1M 字节，它指的是内部 FLASH 的最大空间。

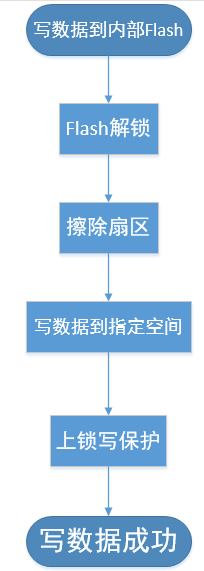
计算程序占用的空间时，需要使用加载区域的大小进行计算，本例子中应用程序使用   
的内部 FLASH 是从 0x08000000 至(0x08000000+0x00000b50)地址的空间区域。   
所以从扇区 1(地址 0x08004000)后的存储空间都可以作其它用途，使用这些存储空间时不会篡改应用程序空间的数据。   
具体可参考原子的例程：实验四十一：FLASH 模拟 EEPROM 实验

文章引用地址：https://blog.csdn.net/qq\_33559992/article/details/77676716

感谢原文作者

**二、代码拆分介绍**（以STM32F105系列为例，如上图表5所示）

**2.1 读/写入数据流程**



                                      写数据流程

**2.1.1、Flash 解锁**，直接调用#include "stm32f10x\_flash.h"中的void FLASH\_Unlock(void)函数，这个函数是官方提供的，其内部代码如下：

IMG_264 View Code

**2.1.2、擦除扇区**，也是直接调用固件库官方的函数FLASH\_Status FLASH\_ErasePage(uint32\_t Page\_Address)，这个官方函数代码也贴出来看看，代码如下：

IMG_265 View Code

 注意这个擦除扇区函数是你提供一个STM32f105系列扇区的开始地址即可，擦除是按照页擦除（每页2KB=1024Byte）或者整个擦除（见STM32参考手册的第二章2.3.3嵌入式闪存部分介绍）

 比如我们要擦除互联网型的127页，我们只需要FLASH\_ErasePage(0x0803f800);执行后，第127页的0x0803f800-0x0803FFFF数据都将被擦除。

当然官方提供的也不知一个擦除函数，而是三个，具体如下，对于32位系统：**一个是字节=4byte=32bite；一个是半字=2byte=16bite；一个是字节=1byte=8bite；**进行擦除。

FLASH\_Status FLASH\_ErasePage(uint32\_t Page\_Address);

FLASH\_Status FLASH\_EraseAllPages(void);

FLASH\_Status FLASH\_EraseOptionBytes(void);

**2.1.3、接下来是写/读数据函数**，该函数也是官方给出的，我们只需要用就好了。**但要注意，这个是个半字的写操作，威少是uint16\_t 的数据算半字呢，因为单片机是32的，对于32位单片机系统来说，一个字是4个字节的，8位的比如51单片机系统一个字就是2位的，64位单片机系统一个字就是8个字节，脱离单片机系统说字是多少个字节是没意义的。**所以这里写入/读出半字也就是一次写入2个字节，写完/读出一次地址会加2。

写数据操作：

IMG_266 View Code

 当然官方给的不止是这一个函数写数据，官方提供了3个

FLASH\_Status FLASH\_ProgramWord(uint32\_t Address, uint32\_t Data);//一次写一个字，对于32系统，一次写的是4个字节，uint32\_t 变量大小，32bit

FLASH\_Status FLASH\_ProgramHalfWord(uint32\_t Address, uint16\_t Data);//一次写一个半字，对于32系统，一次写的是2个字节，uint16\_t 变量大小，16bit

FLASH\_Status FLASH\_ProgramOptionByteData(uint32\_t Address, uint8\_t Data);//一次写一个字节，对于32系统，一次写的是1个字节，uint8\_t 变量大小，8bit

读数据操作：

读数据的函数，官方并没有给出：下面我们自己给出，具体的读法代码如下

[IMG_267](https://www.cnblogs.com/pertor/p/javascript:void(0);)

1 //读取指定地址的半字(16位数据)2 //也是按照半字读出，即每次读2个字节数据返回3 uint16\_t FLASH\_ReadHalfWord(uint32\_t address)4 {5 return \*(\_\_IO uint16\_t\*)address; 6 }

[IMG_268](https://www.cnblogs.com/pertor/p/javascript:void(0);)

如果要连续都区多个地址数据，可以进行如下代码操作

[IMG_269](https://www.cnblogs.com/pertor/p/javascript:void(0);)

1 //从指定地址开始读取多个数据2 void FLASH\_ReadMoreData(uint32\_t startAddress,uint16\_t \*readData,uint16\_t countToRead)3 {4 uint16\_t dataIndex;5 for(dataIndex=0;dataIndex<countToRead;dataIndex++)6 {7 readData[dataIndex]=FLASH\_ReadHalfWord(startAddress+dataIndex\*2);8 }9 }

[IMG_270](https://www.cnblogs.com/pertor/p/javascript:void(0);)

**2.1.4、这步骤应该就是再次上锁**，保护存储区不被重写覆盖了，直接使用官方的函数即可：FLASH\_Lock();//上锁写保护

具体官方代码贴出如下

IMG_271 View Code

**三、简单的小例程代码实现**

例子功能：

       1、将数据存储在stm32F105单片机的主存储区0x08036000地址开始的扇区，（0x08036000应该是该单片机大约108个扇区的开始地址位置即页108起始地址）。

       2、将该单片机的页108（page108=0x08036000）处的数据再读出来;

具体实现代码如下，作为例子，只进行了半字的读写操作，我们写的数据buff为空，内容默认值为0

[IMG_272](https://www.cnblogs.com/pertor/p/javascript:void(0);)

1 #include "stm32f10x\_flash.h" 2 3 #define StartServerManageFlashAddress ((u32)0x08036000)//读写起始地址（内部flash的主存储块地址从0x08036000开始） 4 5 //从指定地址开始写入多个数据 6 void FLASH\_WriteMoreData(uint32\_t startAddress,uint16\_t \*writeData,uint16\_t countToWrite) 7 { 8 uint32\_t offsetAddress=startAddress - FLASH\_BASE; //计算去掉0X08000000后的实际偏移地址 9 uint32\_t sectorPosition=offsetAddress/SECTOR\_SIZE; //计算扇区地址，对于STM32F103VET6为0~25510 uint32\_t sectorStartAddress=sectorPosition\*SECTOR\_SIZE+FLASH\_BASE; //对应扇区的首地址11 uint16\_t dataIndex;12 13 if(startAddress<FLASH\_BASE||((startAddress+countToWrite\*2)>=(FLASH\_BASE + SECTOR\_SIZE \* FLASH\_SIZE)))14 {15 return;//非法地址16 }17 FLASH\_Unlock(); //解锁写保护18 19 FLASH\_ErasePage(sectorStartAddress);//擦除这个扇区20 21 for(dataIndex=0;dataIndex<countToWrite;dataIndex++)22 {23 FLASH\_ProgramHalfWord(startAddress+dataIndex\*2,writeData[dataIndex]);24 }25 26 FLASH\_Lock();//上锁写保护27 }28 29 //读取指定地址的半字(16位数据)30 uint16\_t FLASH\_ReadHalfWord(uint32\_t address)31 {32 return \*(\_\_IO uint16\_t\*)address; 33 }34 35 //从指定地址开始读取多个数据36 void FLASH\_ReadMoreData(uint32\_t startAddress,uint16\_t \*readData,uint16\_t countToRead)37 {38 uint16\_t dataIndex;39 for(dataIndex=0;dataIndex<countToRead;dataIndex++)40 {41 readData[dataIndex]=FLASH\_ReadHalfWord(startAddress+dataIndex\*2);42 }43 }44 45 void write\_to\_flash(void)46 {47 u16 buff[1200];48 u16 count\_len = 2272 / 2;50 FLASH\_WriteMoreData(StartServerManageFlashAddress,buff,count\_len);55 }56 57 void read\_from\_flash(void)58 {59 u16 buff[1200];60 u16 count\_len = 2272 / 2;61 FLASH\_WriteMoreData(StartServerManageFlashAddress,buff,count\_len);66 67 }

[IMG_273](https://www.cnblogs.com/pertor/p/javascript:void(0);)

[IMG_274](https://www.cnblogs.com/pertor/p/javascript:void(0);)

1 void mian(void) 2 { 3 .........//初始化其他外设 4 while(1) 5 { 6 ...........//其他外设执行函数 7 if(满足条件真)//写数据操作 8 ｛ 9 write\_to\_flash();10 ｝11 else //读数据操作12 {13 read\_from\_flash();14 }15 16 }17 }

[IMG_275](https://www.cnblogs.com/pertor/p/javascript:void(0);)

四、附言

值得的注意的是，我们读写的地址是0x08036000，读写方式是半字，这里地址空间对于stm32f105芯片来说是第108扇区，每个扇区2KB，stm32F105VC总共是256KB空间,128页。所以地址能取到0x08036000，像小中容量stm32f103单片机，64KB和128KB的主存储区地址都是到不了0x08036000，除非是stm32f103VE的256KB芯片的主存储快，0x08036000才是有效的存储地址，中小型这个地址都不是有效的主存储开地址（超出了）