零死角玩转STM32

读写内部FLASH

淘宝：firestm32.taobao.com

论坛： www.firebbs.cn

扫描进入淘宝店铺

主讲内容

01 STM32的内部FLASH简介

02 对内部FLASH的写入过程

03 查看工程的空间分布

04 操作内部FLASH的库函数介绍

05 实验：读写内部FLASH

参考资料:《零死角玩转STM32》

“读写内部FLASH”章节

读写内部FLASH

对内部FLASH的写入过程

1. 解锁

由于内部FLASH空间主要存储的是应用程序，是非常关键的数据，为了防止误操作修改了这些内容，芯片复位后默认会结FLASH上锁，这个时候不允许设置FLASH的控制寄存器，并且不能对修改FLASH中的内容。

所以对FLASH写入数据前，需要先给它解锁。解锁的操作步骤如下：

- 往Flash 密钥寄存器 FLASH_KEYR中写入 KEY1 = 0x45670123
- 再往Flash 密钥寄存器 FLASH_KEYR中写入 KEY2 = 0xCDEF89AB

读写内部FLASH

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：

- 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；
- 在 FLASH_CR 寄存器中，将“激活页擦除寄存器位PER ”置 1，
- 用FLASH_AR寄存器选择要擦除的页；
- 将 FLASH_CR 寄存器中的“开始擦除寄存器位 STRT ”置 1，开始擦除；
- 等待 BSY 位被清零时，表示擦除完成

读写内部FLASH

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：

- 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；
- 将 FLASH_CR 寄存器中的 “激活编程寄存器位PG” 置 1；
- 向指定的FLASH存储器地址执行数据写入操作，每次只能以16位的方式写入；
- 等待 BSY 位被清零时，表示写入完成

读写内部FLASH

查看工程的空间分布

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，可以了解程序存储到了哪些区域。

读写内部FLASH

查看工程的空间分布

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：

代码清单 43-1 map 文件中的存储映像分布说明

```
1  ============================================================
2  Memory Map of the image //存储分布映像
3
4  Image Entry point : 0x08000131
5  /*程序 ROM 加载空间*/
6  Load Region LR_IROM1 (Base: 0x08000000, Size: 0x0000178c, Max: 0x00040000, ABSOLUTE)
7  /*程序 ROM 执行空间*/
8  Execution Region ER_IROM1 (Base: 0x08000000, Size: 0x00001760, Max: 0x00040000, ABSOLUTE)
9  /*地址分布列表*/
10 Base Addr    Size         Type   Attr      Idx    E Section Name        Object
11
12 0x08000000   0x00000130   Data   RO          3    RESET               startup_stm32f10x_hd.o
13 0x08000130   0x00000000   Code   RO        479  * .ARM.Collect$$$$00000000  mc_w.l(entry.o)
14 0x08000130   0x00000004   Code   RO        742    .ARM.Collect$$$$00000001  mc_w.l(entry2.o)
15 0x08000134   0x00000004   Code   RO        745    .ARM.Collect$$$$00000004  mc_w.l(entry5.o)
16 /*...此处省略大部分内容*/
17 0x080016cc   0x00000024   Code   RO        772    .text               mc_w.l(init.o)
18 0x080016f0   0x00000010   Code   RO        483    i.__0printf$bare    mc_w.l(printfb.o)
19 0x08001700   0x0000000e   Code   RO        784    i.__scatterload_copy  mc_w.l(handlers.o)
20 0x0800170e   0x00000002   Code   RO        785    i.__scatterload_null  mc_w.l(handlers.o)
21 0x08001710   0x0000000e   Code   RO        786    i.__scatterload_zeroinit  mc_w.l(handlers.o)
22 0x0800171e   0x00000022   Code   RO        490    i._printf_core      mc_w.l(printfb.o)
23 0x08001740   0x00000020   Data   RO        782    Region$$Table       anon$$obj.o
24
```

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。

读写内部FLASH

1.程序ROM的加载与执行空间

上述说明中有两段分别以“Load Region LR_ROM1”及“Execution Region ER_IROM1”开头的内容，它们分别描述程序的加载及执行空间。在芯片刚上电运行时，会加载程序及数据，例如它会从程序的存储区域加载到程序的执行区域，还把一些已初始化的全局变量从ROM复制到RAM空间，以便程序运行时可以修改变量的内容。加载完成后，程序开始从执行区域开始执行。

读写内部FLASH

1.程序ROM的加载与执行空间

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0x0000178c及0x00001760，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0x00040000，即256K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+0x0000178c)地址的空间区域。

读写内部FLASH

2. ROM空间分布表

在加载及执行空间总体描述之后，紧接着一个ROM详细地址分布表，它列出了工程中的各个段(如函数、常量数据)所在的地址Base Addr及占用的空间Size，列表中的Type说明了该段的类型，CODE表示代码，DATA表示数据，而PAD表示段之间的填充区域，它是无效的内容，PAD区域往往是为了解决地址对齐的问题。

观察表中的最后一项，它的基地址是0x08001740，大小为0x00000020，可知它占用的最高的地址空间为0x08001760，跟执行区域的最高地址0x00001760一样，但它们比加载区域说明中的最高地址0x0000178c要小，所以我们以加载区域的大小为准。对比表 43-1的内部FLASH页地址分布表，可知仅使用页0至页2就可以完全存储本应用程序，所以从页3(地址0x08001800)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。

THANKS

论坛： www.firebbs.cn

淘宝：firestm32.taobao.com

扫描进入淘宝店铺