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

2.数据操作位数

在内部FLASH进行擦除及写入操作时，电源电压会影响数据的最大操作位数，该电源电压可通过配置FLASH_CR 寄存器中的 PSIZE位改变，配置表如下：

电压范围	2.7 - 3.6 V (使用外部Vpp)	2.7 - 3.6 V	2.1 – 2.7 V	1.8 – 2.1 V
位数	64	32	16	8
PSIZE(1:0)配置	11b	10b	01b	00b

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，且其供电时间不得超过一小时，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。

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；
- 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；
- 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；
- 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：

```
1  ==============================================================================
2  Memory Map of the image      //存储分布映像
3
4  Image Entry point : 0x080001ad
5
6  /*程序 ROM 加载空间*/
7  Load Region LR_IROM1 (Base: 0x08000000, Size: 0x00000b50, Max: 0x00100000, ABSOLUTE)
8
9  /*程序 ROM 执行空间*/
10 Execution Region ER_IROM1 (Base: 0x08000000, Size: 0x00000b3c, Max: 0x00100000, ABSOLUTE)
11
12 /*地址分布列表*/
13 Base Addr    Size         Type   Attr      Idx    E Section Name        Object
14
15 0x08000000   0x000001ac   Data   RO            3    RESET               startup_stm32f429_439xx.o
16 0x080001ac   0x00000000   Code   RO         5359  * .ARM.Collect$$$$00000000  mc_w.l(entry.o)
17 0x080001ac   0x00000004   Code   RO         5622    .ARM.Collect$$$$00000001  mc_w.l(entry2.o)
18 0x080001b0   0x00000004   Code   RO         5625    .ARM.Collect$$$$00000004  mc_w.l(entry5.o)
19 0x080001b4   0x00000000   Code   RO         5627    .ARM.Collect$$$$00000008  mc_w.l(entry7b.o)
20 0x080001b4   0x00000000   Code   RO         5629    .ARM.Collect$$$$0000000A  mc_w.l(entry8b.o)
21 /*...此处省略大部分内容*/
22 0x08000948   0x0000000e   Code   RO         4910    i.USART_GetFlagStatus  stm32f4xx_usart.o
23 0x08000956   0x00000002   PAD
24 0x08000958   0x000000bc   Code   RO         4914    i.USART_Init           stm32f4xx_usart.o
25 0x08000a14   0x00000008   Code   RO         4924    i.USART_SendData       stm32f4xx_usart.o
26 0x08000a1c   0x00000002   Code   RO         5206    i.UsageFault_Handler   stm32f4xx_it.o
27 0x08000a1e   0x00000002   PAD
28 0x08000a20   0x00000010   Code   RO         5363    i.__0printf$bare       mc_w.l(printfb.o)
29 0x08000a30   0x0000000e   Code   RO         5664    i.__scatterload_copy   mc_w.l(handlers.o)
30 0x08000a3e   0x00000002   Code   RO         5665    i.__scatterload_null   mc_w.l(handlers.o)
31 0x08000a40   0x0000000e   Code   RO         5666    i.__scatterload_zeroinit  mc_w.l(handlers.o)
32 0x08000a4e   0x00000022   Code   RO         5370    i._printf_core         mc_w.l(printfb.o)
33 0x08000a70   0x00000024   Code   RO         5275    i.fputc                bsp_debug_usart.o
34 0x08000a94   0x00000088   Code   RO         5161    i.main                 main.o
35 0x08000b1c   0x00000020   Data   RO         5662    Region$$Table          anon$$obj.o
36
```

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，可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。

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0x08000b1c，大小为0x00000020，可知它占用的最高的地址空间为0x08000b3c，跟执行区域的最高地址0x00000b3c一样，但它们比加载区域说明中的最高地址0x8000b50要小，所以我们以加载区域的大小为准。对比内部FLASH扇区地址分布表，可知仅使用扇区0就可以完全存储本应用程序，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。

THANKS

论坛： www.firebbs.cn

淘宝：firestm32.taobao.com

扫描进入淘宝店铺