昵称： 竹风清
园龄： 5年2个月
粉丝： 20
关注： 4
+加关注

< 2019年11月 >

日	一	二	三	四	五	六
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

搜索
找找看
谷歌搜索

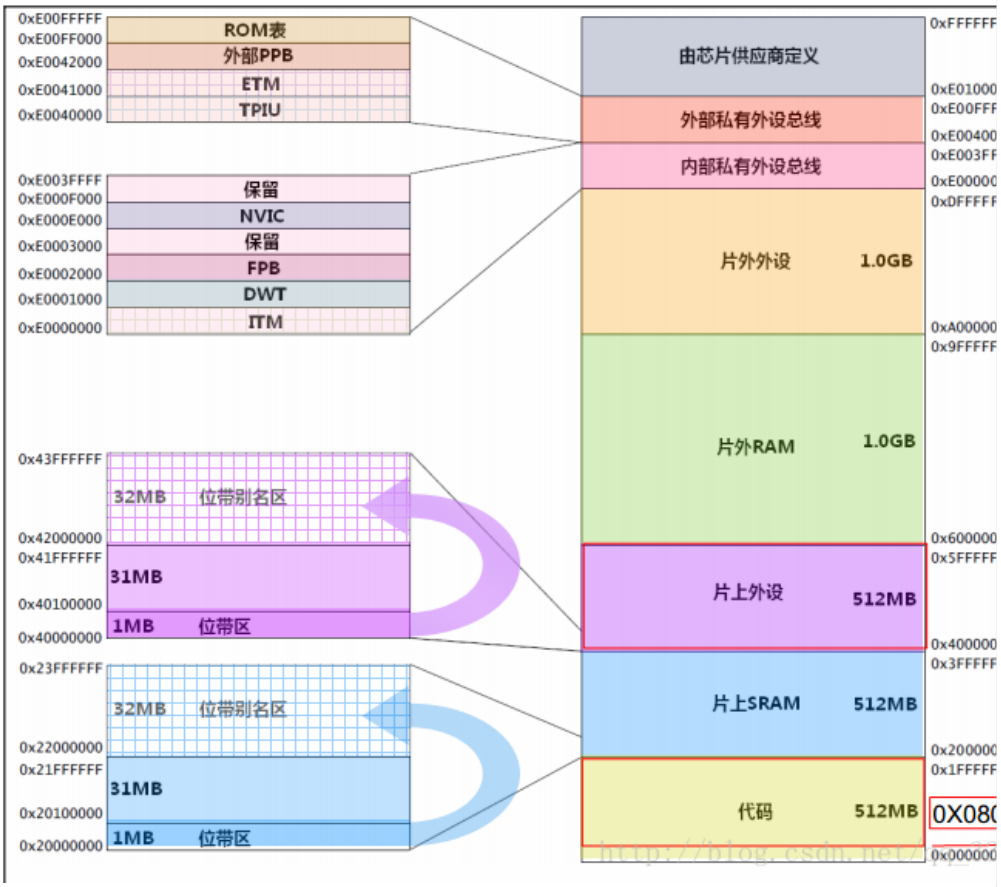
常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签
Linux(3)
ubuntu(3)
python(3)
caffe(2)
Caffe框架搭建(1)
CUDA、NVIDIA、Linux(1)
IWDG(1)
jupyter(1)

STM32学习笔记：读写内部Flash（介绍+附代码）

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：

levelDB(1)

LmDb(1)

更多

随笔分类

C++(2)

C语言(5)

FPGA

Linux笔记(7)

电路(2)

电子单片机嵌入式(13)

机器学习笔记

模电

通信(2)

问题解决(5)

随笔档案

2019年10月(2)

2018年12月(1)

2018年11月(1)

2018年9月(2)

2018年8月(8)

2018年6月(1)

2018年5月(2)

2018年4月(12)

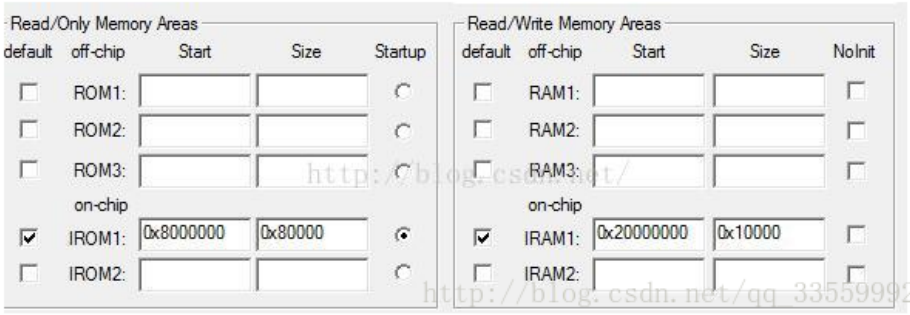
2017年9月(1)

2017年8月(2)

2017年4月(1)

2017年3月(2)

2017年2月(3)



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：

主存储器	块 1	扇区 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
		扇区 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
		扇区 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
		扇区 3	0x0800 C000 - 0x0800 FFFF	16 Kbyte
		扇区 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
		扇区 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
		扇区 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
		扇区 7	0x0806 0000 - 0x0807 FFFF	128 Kbytes
		扇区 8	0x0808 0000 - 0x0809 FFFF	128 Kbytes
		扇区 9	0x080A 0000 - 0x080B FFFF	128 Kbytes
		扇区 10	0x080C 0000 - 0x080D FFFF	128 Kbytes
		扇区 11	0x080E 0000 - 0x080F FFFF	128 Kbytes
	块 2	扇区 12	0x0810 0000 - 0x0810 3FFF	16 Kbytes
		扇区 13	0x0810 4000 - 0x0810 7FFF	16 Kbytes
		扇区 14	0x0810 8000 - 0x0810 BFFF	16 Kbytes
		扇区 15	0x0810 C000 - 0x0810 FFFF	16 Kbyte
		扇区 16	0x0811 0000 - 0x0811 FFFF	64 Kbytes
		扇区 17	0x0812 0000 - 0x0813 FFFF	128 Kbytes
		扇区 18	0x0814 0000 - 0x0815 FFFF	128 Kbytes
		扇区 19	0x0816 0000 - 0x0817 FFFF	128 Kbytes
		扇区 20	0x0818 0000 - 0x0819 FFFF	128 Kbytes
		扇区 21	0x081A 0000 - 0x081B FFFF	128 Kbytes
		扇区 22	0x081C 0000 - 0x081D FFFF	128 Kbytes
		扇区 23	0x081E 0000 - 0x081F FFFF	128 Kbytes
系统存储区			0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP 区域			0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
选项字节	块 1		0x1FFF C000 - 0x1FFF C00F	16 bytes
	块 2		0x1FFE C000 - 0x1FFE C00F	16 bytes

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：

相册

photo1(1)

最新评论

1. Re:STM32定时器配置（TIM1、TIM2、TIM3、TIM4、TIM5、TIM8）高级定时器+普通定时器，定时计数模式下总结

我手头没有板子，只是keil仿真调试，即使tim5部分都改过来，仿真的时候，定时器的cnt不会发生变化。我不知道是仿真环境的问题还是现实上单片机运行就是这样。

--凉初捕影

2. Re:STM32定时器配置（TIM1、TIM2、TIM3、TIM4、TIM5、TIM8）高级定时器+普通定时器，定时计数模式下总结

@ 凉初捕影现在改了下，把定时器五的中断使能，我写成TIM3,你改成TIM5就好了，由于，这些是我从我两年前写的个人开发库函数中摘抄的，复制出来为了省事，没搞严谨。你看看改了能用不，我文章也做修改了...

--竹风清

3. Re:STM32定时器配置（TIM1、TIM2、TIM3、TIM4、TIM5、TIM8）高级定时器+普通定时器，定时计数模式下总结

@ 凉初捕影多谢提醒，以后会注意了...

--竹风清

4. Re:STM32定时器配置（TIM1、TIM2、TIM3、TIM4、TIM5、TIM8）高级定时器+普通定时器，定时计数模式下总结

你的定时器5没有启动。

--凉初捕影

5. Re:STM32定时器配置（TIM1、TIM2、TIM3、TIM4、TIM5、TIM8）高级定时器+普通定时器，定时计数模式下总结

表3　闪存模块的组织(中容量产品)

模块	名称	地址	大小(字
主存储块	页0	0x0800 0000 - 0x0800 03FF	1K
	页1	0x0800 0400 - 0x0800 07FF	1K
	页2	0x0800 0800 - 0x0800 0BFF	1K
	页3	0x0800 0C00 - 0x0800 0FFF	1K
	页4	0x0800 1000 - 0x0800 13FF	1K
	… …	… …	… …
	页127	0x0801 FC00 - 0x0801 FFFF	1K
信息块	系统存储器	0x1FFF F000 - 0x1FFF F7FF	2K
	选择字节	0x1FFF F800 - 0x1FFF F80F	16
闪存存储器接口寄存器	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FALSH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4
	保留	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRPR	0x4002 2020 - 0x4002 2023	4

表5　闪存模块的组织(互联型产品)

模块	名称	地址	大小(字节
主存储块	页0	0x0800 0000 - 0x0800 07FF	2K
	页1	0x0800 0800 - 0x0800 0FFF	2K
	页2	0x0800 1000 - 0x0800 17FF	2K
	页3	0x0800 1800 - 0x0800 1FFF	2K
	… …	… …	… …
	页127	0x0803 F800 - 0x0803 FFFF	2K
信息块	系统存储器	0x1FFF B000 - 0x1FFF F7FF	18K
	选择字节	0x1FFF F800 - 0x1FFF F80F	16
闪存存储器接口寄存器	FLASH_ACR	0x4002 2000 - 0x4002 2003	4
	FALSH_KEYR	0x4002 2004 - 0x4002 2007	4
	FLASH_OPTKEYR	0x4002 2008 - 0x4002 200B	4
	FLASH_SR	0x4002 200C - 0x4002 200F	4
	FLASH_CR	0x4002 2010 - 0x4002 2013	4
	FLASH_AR	0x4002 2014 - 0x4002 2017	4
	保留	0x4002 2018 - 0x4002 201B	4
	FLASH_OBR	0x4002 201C - 0x4002 201F	4
	FLASH_WRPR	0x4002 2020 - 0x4002 2023	4

有关闪存寄存器的详细信息，请参考《STM32F10xxx闪存编程手册》

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。

表2　STM32F105xx和STM32F107xx互联型产品功能和外设配置

外设(1)	STM32F105Rx			STM32F107Rx		STM32F105Vx			STM32F1
闪存(K字节)	64	128	256	128	256	64	128	256	128
SRAM(K字节)	64								

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)，

```
================================================================================
Memory Map of the image      //存储分布映像

Image Entry point : 0x080001ad

/*程序 ROM 加载空间*/
Load Region LR_IROM1 (Base: 0x08000000, Size: 0x00000b50, Max: 0x00100000, ABSOLUTE)

/*程序 ROM 执行空间*/
Execution Region ER_IROM1 (Base: 0x08000000, Size: 0x00000b3c, Max: 0x00100000, ABSOLUTE)

/*地址分布列表*/
Base Addr    Size         Type   Attr      Idx    E Section Name        Object

0x08000000   0x000001ac   Data   RO            3    RESET               startup_stm32f429_439xx
0x080001ac   0x00000000   Code   RO         5359  * .ARM.Collect$$$$00000000  mc_w.l(entry.o)
0x080001ac   0x00000004   Code   RO         5622    .ARM.Collect$$$$00000001  mc_w.l(entry2.o)
0x080001b0   0x00000004   Code   RO         5625    .ARM.Collect$$$$00000004  mc_w.l(entry5.o)
0x080001b4   0x00000000   Code   RO         5627    .ARM.Collect$$$$00000008  mc_w.l(entry7b.o)
0x080001b4   0x00000000   Code   RO         5629    .ARM.Collect$$$$0000000A  mc_w.l(entry8b.o)
/*...此处省略大部分内容*/
0x08000948   0x0000000e   Code   RO         4910    i.USART_GetFlagStatus  stm32f4xx_usart.o
0x08000956   0x00000002   PAD
0x08000958   0x000000bc   Code   RO         4914    i.USART_Init           stm32f4xx_usart.o
0x08000a14   0x00000008   Code   RO         4924    i.USART_SendData       stm32f4xx_usart.o
0x08000a1c   0x00000002   Code   RO         5206    i.UsageFault_Handler   stm32f4xx_it.o
0x08000a1e   0x00000002   PAD
0x08000a20   0x00000010   Code   RO         5363    i.__0printf$bare       mc_w.l(printfb.o)
0x08000a30   0x0000000e   Code   RO         5664    i.__scatterload_copy   mc_w.l(handlers.o)
0x08000a3e   0x00000002   Code   RO         5665    i.__scatterload_null   mc_w.l(handlers.o)
0x08000a40   0x0000000e   Code   RO         5666    i.__scatterload_zeroinit  mc_w.l(handlers.o
0x08000a4e   0x00000022   Code   RO         5370    i._printf_core         mc_w.l(printfb.o)
0x08000a70   0x00000024   Code   RO         5275    i.fputc                bsp_debug_usart.o
0x08000a94   0x00000088   Code   RO         5161    i.main                 main.o
0x08000b1c   0x00000020   Data   RO         5662    Region$$Table          anon$$obj.o
```


【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。

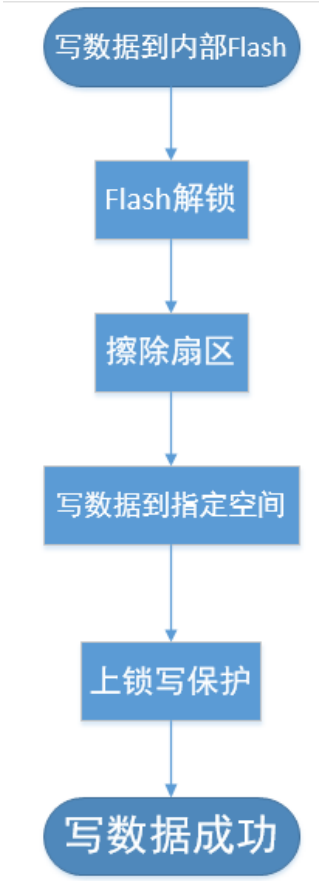
计算程序占用的空间时，需要使用加载区域的大小进行计算，本例子中应用程序使用
的内部 FLASH 是从 0x08000000 至(0x08000000+0x00000b50)地址的空间区域。
所以从扇区 1(地址 0x08004000)后的存储空间都可以作其它用途，使用这些存储空间时不会篡改应用程序空间的数据。
具体可参考原子的例程：实验四十一：FLASH 模拟 EEPROM 实验

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：

```
/**
  * @brief  Unlocks the FLASH Program Erase Controller.
  * @note   This function can be used for all STM32F10x devices.
  *         - For STM32F10X_XL devices this function unlocks Bank1 and Bank2.
  *         - For all other devices it unlocks Bank1 and it is equivalent
  *           to FLASH_UnlockBank1 function..
  * @param  None
  * @retval None
  */
void FLASH_Unlock(void)
{
  /* Authorize the FPEC of Bank1 Access */
  FLASH->KEYR = FLASH_KEY1;
  FLASH->KEYR = FLASH_KEY2;

#ifdef STM32F10X_XL
  /* Authorize the FPEC of Bank2 Access */
  FLASH->KEYR2 = FLASH_KEY1;
  FLASH->KEYR2 = FLASH_KEY2;
#endif /* STM32F10X_XL */
}
```

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：

```c
/**
  * @brief  Erases a specified FLASH page.
  * @note   This function can be used for all STM32F10x devices.
  * @param  Page_Address: The page address to be erased.
  * @retval FLASH Status: The returned value can be: FLASH_BUSY, FLASH_ERROR_PG,
  *         FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.
  */
FLASH_Status FLASH_ErasePage(uint32_t Page_Address)
{
  FLASH_Status status = FLASH_COMPLETE;
  /* Check the parameters */
  assert_param(IS_FLASH_ADDRESS(Page_Address));

#ifdef STM32F10X_XL
  if(Page_Address < FLASH_BANK1_END_ADDRESS)
  {
    /* Wait for last operation to be completed */
    status = FLASH_WaitForLastBank1Operation(EraseTimeout);
    if(status == FLASH_COMPLETE)
    {
      /* if the previous operation is completed, proceed to erase the page */
      FLASH->CR|= CR_PER_Set;
      FLASH->AR = Page_Address;
      FLASH->CR|= CR_STRT_Set;

      /* Wait for last operation to be completed */
      status = FLASH_WaitForLastBank1Operation(EraseTimeout);

      /* Disable the PER Bit */
      FLASH->CR &= CR_PER_Reset;
    }
  }
  else
  {
    /* Wait for last operation to be completed */
    status = FLASH_WaitForLastBank2Operation(EraseTimeout);
    if(status == FLASH_COMPLETE)
    {
      /* if the previous operation is completed, proceed to erase the page */
      FLASH->CR2|= CR_PER_Set;
      FLASH->AR2 = Page_Address;
      FLASH->CR2|= CR_STRT_Set;

      /* Wait for last operation to be completed */
      status = FLASH_WaitForLastBank2Operation(EraseTimeout);

      /* Disable the PER Bit */
      FLASH->CR2 &= CR_PER_Reset;
    }
  }
#else
  /* Wait for last operation to be completed */
  status = FLASH_WaitForLastOperation(EraseTimeout);

  if(status == FLASH_COMPLETE)
  {
    /* if the previous operation is completed, proceed to erase the page */
    FLASH->CR|= CR_PER_Set;
    FLASH->AR = Page_Address;
    FLASH->CR|= CR_STRT_Set;

    /* Wait for last operation to be completed */
    status = FLASH_WaitForLastOperation(EraseTimeout);

    /* Disable the PER Bit */
    FLASH->CR &= CR_PER_Reset;
  }
```

```
#endif /* STM32F10X_XL */

  /* Return the Erase Status */
  return status;
}
```

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，咸少是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：

```
/**
  * @brief  Programs a half word at a specified address.
  * @note   This function can be used for all STM32F10x devices.
  * @param  Address: specifies the address to be programmed.
  * @param  Data: specifies the data to be programmed.
  * @retval FLASH Status: The returned value can be: FLASH_ERROR_PG,
  *         FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.
  */
FLASH_Status FLASH_ProgramHalfWord(uint32_t Address, uint16_t Data)
{
  FLASH_Status status = FLASH_COMPLETE;
  /* Check the parameters */
  assert_param(IS_FLASH_ADDRESS(Address));

#ifdef STM32F10X_XL
  /* Wait for last operation to be completed */
  status = FLASH_WaitForLastOperation(ProgramTimeout);

  if(Address < FLASH_BANK1_END_ADDRESS)
  {
    if(status == FLASH_COMPLETE)
    {
      /* if the previous operation is completed, proceed to program the new data */
      FLASH->CR |= CR_PG_Set;

      *(__IO uint16_t*)Address = Data;
      /* Wait for last operation to be completed */
      status = FLASH_WaitForLastBank1Operation(ProgramTimeout);

      /* Disable the PG Bit */
      FLASH->CR &= CR_PG_Reset;
    }
  }
  else
  {
    if(status == FLASH_COMPLETE)
    {
```

```
      /* if the previous operation is completed, proceed to program the new data */
      FLASH->CR2 |= CR_PG_Set;

      *(__IO uint16_t*)Address = Data;
      /* Wait for last operation to be completed */
      status = FLASH_WaitForLastBank2Operation(ProgramTimeout);

      /* Disable the PG Bit */
      FLASH->CR2 &= CR_PG_Reset;
    }
  }
#else
  /* Wait for last operation to be completed */
  status = FLASH_WaitForLastOperation(ProgramTimeout);

  if(status == FLASH_COMPLETE)
  {
    /* if the previous operation is completed, proceed to program the new data */
    FLASH->CR |= CR_PG_Set;

    *(__IO uint16_t*)Address = Data;
    /* Wait for last operation to be completed */
    status = FLASH_WaitForLastOperation(ProgramTimeout);

    /* Disable the PG Bit */
    FLASH->CR &= CR_PG_Reset;
  }
#endif  /* STM32F10X_XL */

  /* Return the Program Status */
  return status;
}
```

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

```
//读取指定地址的半字(16位数据)
//也是按照半字读出，即每次读2个字节数据返回
uint16_t FLASH_ReadHalfWord(uint32_t address)
{
  return *(__IO uint16_t*)address;
}
```

如果要连续都区多个地址数据，可以进行如下代码操作

```
//从指定地址开始读取多个数据
void FLASH_ReadMoreData(uint32_t startAddress,uint16_t *readData,uint16_t countToRead)
{
  uint16_t dataIndex;
  for(dataIndex=0;dataIndex<countToRead;dataIndex++)
  {
```

```
7     readData[dataIndex]=FLASH_ReadHalfWord(startAddress+dataIndex*2);
8   }
9 }
```

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

```
 1 /**
 2   * @brief  Locks the FLASH Program Erase Controller.
 3   * @note   This function can be used for all STM32F10x devices.
 4   *         - For STM32F10X_XL devices this function Locks Bank1 and Bank2.
 5   *         - For all other devices it Locks Bank1 and it is equivalent
 6   *           to FLASH_LockBank1 function.
 7   * @param  None
 8   * @retval None
 9   */
10 void FLASH_Lock(void)
11 {
12   /* Set the Lock Bit to lock the FPEC and the CR of  Bank1 */
13   FLASH->CR |= CR_LOCK_Set;
14
15 #ifdef STM32F10X_XL
16   /* Set the Lock Bit to lock the FPEC and the CR of  Bank2 */
17   FLASH->CR2 |= CR_LOCK_Set;
18 #endif /* STM32F10X_XL */
19 }
```

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

```
 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0~255
10   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)))
```

```
  {
    return;//非法地址
  }
  FLASH_Unlock();          //解锁写保护

  FLASH_ErasePage(sectorStartAddress);//擦除这个扇区

  for(dataIndex=0;dataIndex<countToWrite;dataIndex++)
  {
    FLASH_ProgramHalfWord(startAddress+dataIndex*2,writeData[dataIndex]);
  }

  FLASH_Lock();//上锁写保护
}

//读取指定地址的半字(16位数据)
uint16_t FLASH_ReadHalfWord(uint32_t address)
{
  return *(__IO uint16_t*)address;
}

//从指定地址开始读取多个数据
void FLASH_ReadMoreData(uint32_t startAddress,uint16_t *readData,uint16_t
countToRead)
{
  uint16_t dataIndex;
  for(dataIndex=0;dataIndex<countToRead;dataIndex++)
  {
    readData[dataIndex]=FLASH_ReadHalfWord(startAddress+dataIndex*2);
  }
}

void write_to_flash(void)
{
    u16 buff[1200];
    u16 count_len = 2272 / 2;
    FLASH_WriteMoreData(StartServerManageFlashAddress,buff,count_len);
}

void read_from_flash(void)
{
    u16 buff[1200];
    u16 count_len = 2272 / 2;
    FLASH_WriteMoreData(StartServerManageFlashAddress,buff,count_len);

 }
```

```
void mian(void)
{
    .........//初始化其他外设
    while(1)
    {
        ...........//其他外设执行函数
        if(满足条件真)//写数据操作
        {
            write_to_flash();
        }
        else //读数据操作
        {
            read_from_flash();
        }

    }
 }
```

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）

分类: 电子单片机嵌入式

标签: 片上Flash, STM32F105/STM32F103VE代码适用, 512KB空间代码适用, 其他代码稍作调整即可

竹风清
关注 - 4
粉丝 - 20
+加关注

2 0

« 上一篇: Keil-MDK编译完成后代码大小
» 下一篇: STM32串口通信配置（USART1+USART2+USART3+UART4）

posted @ 2018-08-16 15:36 竹风清 阅读(27871) 评论(0) 编辑 收藏

相关博文:
· STM32对内部Flash的保护措施
· stm32——Flash读写
· MSP430程序库<十五>Flash控制器
· Nand Flash原理分析与编程
· STM32F103使用内部Flash保存参数
» 更多推荐...

最新 IT 新闻:
· Google Chrome 浏览器重新设计"隐私和安全设置"页面
· Gartner：全球公有云收入明年将增长17%达2664亿美元
· 华为云新一代云操作系统"瑶光"智慧云脑正式商用
· 媒体：阿里巴巴香港上市募股获得"数倍"认购
· 谷歌取消每周一次全员大会：将改为每月一次 重点讨论业务和战略
» 更多新闻...

历史上的今天:
2018-08-16 STM32串口通信配置（USART1+USART2+USART3+UART4）
2018-08-16 STM32学习笔记：读写内部Flash（介绍+附代码）
