零死角玩转STM32

设置FLASH的读写保护及解除

淘宝：firestm32.taobao.com

论坛： www.firebbs.cn

扫描进入淘宝店铺

主讲内容

01 选项字节与读写保护

02 修改选项字节的过程

03 操作选项字节的库函数

04 实验：设置读写保护及解除

参考资料:《零死角玩转STM32》

“设置FLASH的读写保护及解除”章节

设置FLASH的读写保护及解除

修改选项字节的过程

根据前面的说明，修改选项字节的内容可修改读写保护配置，不过选项字节复位后的默认状态是始终可以读但被写保护的，因此它具有类似前面《读写内部FLASH》章节提到的FLASH_CR寄存器的访问限制，要想修改，需要先对FLASH_OPTKEYR寄存器写入解锁编码。由于修改选项字节时也需要访问FLASH_CR寄存器，所以同样也要对FLASH_KEYR写入解锁编码。

设置FLASH的读写保护及解除

修改选项字节的过程

修改选项字节的配置步骤如下：

- 解除FLASH_CR寄存器的访问限制
- 往FPEC键寄存器 FLASH_KEYR中写入 KEY1 = 0x45670123
- 再往FPEC键寄存器 FLASH_KEYR中写入 KEY2 = 0xCDEF89AB
- 解除对选项字节的访问限制
- 往FLASH_OPTKEYR中写入 KEY1 = 0x45670123

设置FLASH的读写保护及解除

修改选项字节的过程

- 再往FLASH_OPTKEYR中写入 KEY2 = 0xCDEF89AB
- 配置FLASH_CR的OPTPG位，准备修改选项字节
- 直接使用指针操作修改选项字节的内容，根据需要修改RDP、WRP等内容
- 对于读保护的解除，由于它会擦除FLASH的内容，所以需要检测状态寄存器标志位以确认FLASH擦除操作完成。
- 若是设置读保护及其解除，需要给芯片重新上电复位，以使新配置的选项字节生效；对于设置写保护及其解除，需要给芯片进行系统复位，以使新配置的选项字节生效。

设置FLASH的读写保护及解除

1.选项字结构体定义

```
2   * @brief 选项字节结构体
3   */
4  typedef struct {
5      __IO uint16_t RDP;  /*RDP 及 nRDP*/
6      __IO uint16_t USER; /*USER 及 nUSER,下面类似*/
7      __IO uint16_t Data0;
8      __IO uint16_t Data1;
9      __IO uint16_t WRP0;
10     __IO uint16_t WRP1;
11     __IO uint16_t WRP2;
12     __IO uint16_t WRP3;
13 } OB_TypeDef;
14
15 /*强制转换为选项字节结构体指针*/
16 #define OB               ((OB_TypeDef *) OB_BASE)
17 /*选项字节基地址 */
18 #define OB_BASE          ((uint32_t)0x1FFFF800)
```

标准库中定义的选项字节结构体，包含了RDP、USER、DATA0/1及WRP0/1/2/3这些内容，每个结构体成员指向选项字节对应选项的原始配置码及反码。不过，根据手册中的说明可了解到，当向选项字节的这些地址写入配置时，它会自动取低位字节计算出高位字节的值再存储，即自动取反码，非常方便。例如程序中执行操作给结构体成员WRP0赋值为0x0011时，最终它会自动写入0xEE11（0xEE是0x11的反码）。最后，从OB_BASE宏的定义可以确认它所指向的正是前面介绍的选项字节基地址，说明若在程序中使用该结构体赋值，会直接把内容写入到选项字节地址对应的空间中。

设置FLASH的读写保护及解除

2.设置写保护及解除

库文件提供了FLASH_EnableWriteProtection函数，可用于设置写保护及解除:

```
/* 掩码 */
#define RDPRT_Mask                ((uint32_t)0x00000002)
#define WRP0_Mask                 ((uint32_t)0x000000FF)
#define WRP1_Mask                 ((uint32_t)0x0000FF00)
#define WRP2_Mask                 ((uint32_t)0x00FF0000)
#define WRP3_Mask                 ((uint32_t)0xFF000000)

/* 大容量产品页保护的宏定义，每位控制 4K 字节（2 页） */
#define FLASH_WRProt_Pages0to1          ((uint32_t)0x00000001)
#define FLASH_WRProt_Pages2to3          ((uint32_t)0x00000002)
#define FLASH_WRProt_Pages4to5          ((uint32_t)0x00000004)
#define FLASH_WRProt_Pages6to7          ((uint32_t)0x00000008)
/*..部分省略*/
/*!< 特殊位，最后一位，页 62 至 511 */
#define FLASH_WRProt_Pages62to511       ((uint32_t)0x80000000)
/*保护所有页*/
#define FLASH WRProt AllPages           ((uint32 t)0xFFFFFFFF)
/**
* @brief  对指定的页设置写保护
* @param  FLASH_Pages: 指定要设置写保护的页.
*   可输入参数:
*    STM32 大容量产品: FLASH_WRProt_Pages0to1 至 FLASH_WRProt_Pages60to61
*     或 FLASH_WRProt_Pages62to255 和 FLASH_WRProt_AllPages

* @retval FLASH Status:
*       可能的返回值: FLASH_ERROR_PG, FLASH_ERROR_WRP,
*      FLASH_COMPLETE or FLASH_TIMEOUT.
*/
```

```
FLASH_Status FLASH_EnableWriteProtection(uint32_t FLASH_Pages)
{
    uint16_t WRP0_Data = 0xFFFF, WRP1_Data = 0xFFFF,
             WRP2_Data = 0xFFFF, WRP3_Data = 0xFFFF;

    FLASH_Status status = FLASH_COMPLETE;

    /* 检查参数 */
    assert_param(IS_FLASH_WRPROT_PAGE(FLASH_Pages));
    /*根据输入计算要设置的值*/
    FLASH_Pages = (uint32_t)(~FLASH_Pages);
    WRP0_Data = (uint16_t)(FLASH_Pages & WRP0_Mask);
    WRP1_Data = (uint16_t)((FLASH_Pages & WRP1_Mask) >> 8);
    WRP2_Data = (uint16_t)((FLASH_Pages & WRP2_Mask) >> 16);
    WRP3_Data = (uint16_t)((FLASH_Pages & WRP3_Mask) >> 24);

    /* 等待上一次操作完毕 */
    status = FLASH_WaitForLastOperation(ProgramTimeout);

    if (status == FLASH_COMPLETE) {
        /* 对选项字节进行解锁 */
        FLASH->OPTKEYR = FLASH_KEY1;
        FLASH->OPTKEYR = FLASH_KEY2;
        FLASH->CR |= CR_OPTPG_Set; //准备写入选项字节
        if (WRP0_Data != 0xFF) {
            OB->WRP0 = WRP0_Data;
            /* 等待上一次操作完毕 */
            status = FLASH_WaitForLastOperation(ProgramTimeout);
        }
        if ((status == FLASH_COMPLETE) && (WRP1_Data != 0xFF)) {
            OB->WRP1 = WRP1_Data;
            status = FLASH_WaitForLastOperation(ProgramTimeout);
        }
        if ((status == FLASH_COMPLETE) && (WRP2_Data != 0xFF)) {
            OB->WRP2 = WRP2_Data;
            status = FLASH_WaitForLastOperation(ProgramTimeout);
        }
        if ((status == FLASH_COMPLETE)&& (WRP3_Data != 0xFF)) {
            OB->WRP3 = WRP3_Data;
            status = FLASH_WaitForLastOperation(ProgramTimeout);
        }
        if (status != FLASH_TIMEOUT) {
            /* 若写入完成，对选项字节重新上锁 */
            FLASH->CR &= CR_OPTPG_Reset;
        }
    }
    /* 返回设置结果 */
    return status;
}
```

设置FLASH的读写保护及解除

2.设置写保护及解除

该函数的输入参数可选FLASH_WRProt_Pages0to1至FLASH_WRProt_Pages62to511等宏，该参数用于指定要对哪些页进行写保护。

从该宏的定义方式可了解到，它用一个32位的数值表示WRP0/1/2/3，而宏名中的页码使用数据位1来在WRP0/1/2/3中对应的位作掩码指示。如控制页0至页1的宏FLASH_WRProt_Pages0to1，它由WRP0最低位控制，所以其宏值为0x00000001（bit0为1）；类似地，控制页2至页3的宏FLASH_WRProt_Pages2to3，由WRP0的bit1控制，所以其宏值为0x00000002（bit1为1）。

设置FLASH的读写保护及解除

2.设置写保护及解除

理解了输入参数宏的结构后，即可分析函数中的具体代码。其中最核心要理解的是对输入参数的运算，输入参数FLASH_Pages自身会进行取反操作，从而用于指示要保护页的宏对应的数据位会被置0，而在选项字节WRP中，被写0的数据位对应的页会被保护。FLASH_Pages取反后的值被分解成WRP0/1/2/3_Data四个部分，所以在后面的代码中，可以直接把WRP0/1/2/3_Data变量的值写入到选项字节中。关于这部分运算，您可以亲自代入几个宏进行运算，加深理解。

设置FLASH的读写保护及解除

2.设置写保护及解除

得到数据后，函数开始对FLASH_OPTKEYR寄存器写入解锁码，然后操作FLASH_CR寄存器的OPTPG位准备写入，写入的时候它直接往指向选项字节的结构体OB赋值，如OB->WRP0 = WRP0_Data，注意在这部分写入的时候，根据前面的运算，可知WRP0_Data中只包含了WRP0的内容，而nWRP0的值为0，这个nWRP0的值最终会由芯片自动产生。代码后面的WRP1/2/3操作类似。

仔细研究了这个库函数后，可知它内部并没有对FLASH_CR的访问作解锁操作，所以在调用本函数前，需要先调用FLASH_Unlock解锁。另外，库文件中并没有直接的函数用于解除保护，但实际上解除保护也可以使用这个函数来处理，例如使用输入参数0来调用函数FLASH_EnableWriteProtection(0)，根据代码的处理，它最终会向WRP0/1/2/3选项字节全写入1，从而达到整片FLASH解除写保护的目的。

设置FLASH的读写保护及解除

2.设置读保护及解除

类似地，库文件中提供了函数FLASH_ReadOutProtection 来设置FLASH的读保护及解除：

```
1
2
3  #define RDP_Key                  ((uint16_t)0x00A5)
4
5
6  /**
7    * @brief  使能或关闭读保护
8    * @note   若芯片本身有对选项字节进行其它操作，
9              请先读出然后再重新写入，因为本函数会擦除所有选项字节的内容
10
11   * @param  Newstate: 使能（ENABLE）或关闭（DISABLE）
12   * @retval FLASH Status: 可能的返回值: FLASH_ERROR_PG,
13   *         FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.
14   */
15 FLASH_Status FLASH_ReadOutProtection(FunctionalState NewState)
16 {
17     FLASH_Status status = FLASH_COMPLETE;
18     /* 检查参数 */
19     assert_param(IS_FUNCTIONAL_STATE(NewState));
20     status = FLASH_WaitForLastOperation(EraseTimeout);
21     if (status == FLASH_COMPLETE) {
22         /* 写入选项字节解锁码 */
23         FLASH->OPTKEYR = FLASH_KEY1;
24         FLASH->OPTKEYR = FLASH_KEY2;
25         FLASH->CR |= CR_OPTER_Set; //擦除选项字节
26         FLASH->CR |= CR_STRT_Set;  //开始擦除
27         /* 等待上一次操作完毕 */
28         status = FLASH_WaitForLastOperation(EraseTimeout);
29         if (status == FLASH_COMPLETE) {
30             /* 若擦除操作完成，复位 OPTER 位 */
31             FLASH->CR &= CR_OPTER_Reset;
32             /* 准备写入选项字节 */
33             FLASH->CR |= CR_OPTPG_Set;
34             if (NewState != DISABLE) {
35                 OB->RDP = 0x00;//写入非 0xA5 值，进行读保护
36             } else {
37                 OB->RDP = RDP_Key;  //写入 0xA5，解除读保护
38             }
39             /* 等待上一次操作完毕 */
40             status = FLASH_WaitForLastOperation(EraseTimeout);
41
42             if (status != FLASH_TIMEOUT) {
43                 /* 若操作完毕，复位 OPTPG 位 */
44                 FLASH->CR &= CR_OPTPG_Reset;
45             }
46         } else {
47             if (status != FLASH_TIMEOUT) {
48                 /* 复位 OPTER 位 */
49                 FLASH->CR &= CR_OPTER_Reset;
50             }
51         }
52     }
53     /* 返回设置结果 */
54     return status;
55 }
56
```

设置FLASH的读写保护及解除

2.设置读保护及解除

由于读保护都是针对整个芯片的，所以读保护的配置函数相对简单，它通过输入参数ENABLE或DISABL参数来进行保护或解除。它的内部处理与前面介绍的修改选项字节过程完全一致，当要进行读保护时，往选项字节结构体OB->RDP写入0x00（实际上写入非0xA5的值均可达到目的），而要解除读保护时，则写入0xA5。要注意的是，本函数同样有对FLASH_CR寄存器的访问，但并没有进行解锁操作，所以调用本函数前，同样需要先使用FLASH_Unlock函数解锁。

THANKS

论坛： www.firebbs.cn

淘宝：firestm32.taobao.com

扫描进入淘宝店铺