第6讲 使用位带操作操纵GPIO

内容提要

6.1 位操作

6.2 位带（位绑定）原理

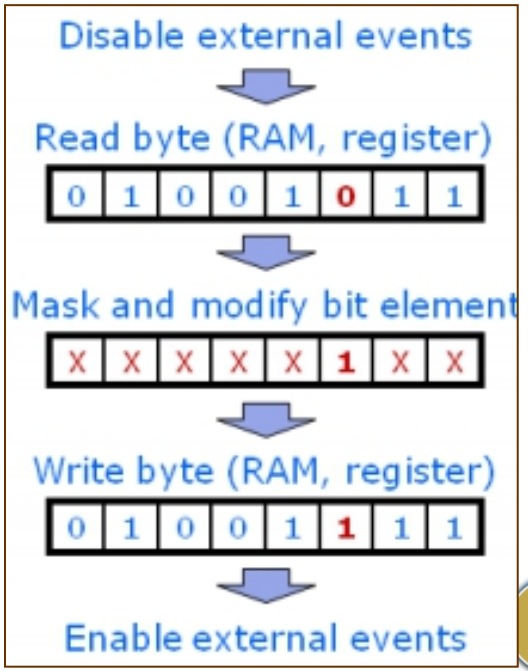
6.2 GPIO位带操作

位操作—汇编层面

- 外设控制常要针对字中某个位（Bit）操作
- 以字节编址的存储器地址空间中，需要3步骤（读出-修改-写回）
  1. （从外设）读取包含该位的字节数据
  3. 设置该位为0或1、同时屏蔽其他位（不改）
  3. 将包含该位的字节数据写入（外设）

```
LDR r0, =0x20000300
 LDR r1, [r0]          ; 读取数据
 ORR r1, r1, #0x4      ; D2位设置为1
 STR r1, [r0]          ; 写回结果
```

通常要专门设置临界区，以保证“原子操作”



位操作--C语言层面

```
a |= (1<<2);   // 位或实现置位
      //将整型变量a的D2位置位、其他位不变
a &= ~(1<<6);   // 位与实现复位
      //将整型变量a的D6位清零、其他位不变
a ^= (1<<6);   // 位异或实现求反
      //将整型变量a的D6位取反、其他位不变
```

运算符	含义	示例(假设为char类型)
&	位与	0x69 & 0x55 → 0x41
|	位或	0x69 | 0x55 → 0x7D
~	位非	~0x69 → 0x96
^	位异或	0x69 ^ 0x55 → 0x3C

51单片机C51的位定义

- 51单片机中通过关键字sbit来实现位定义

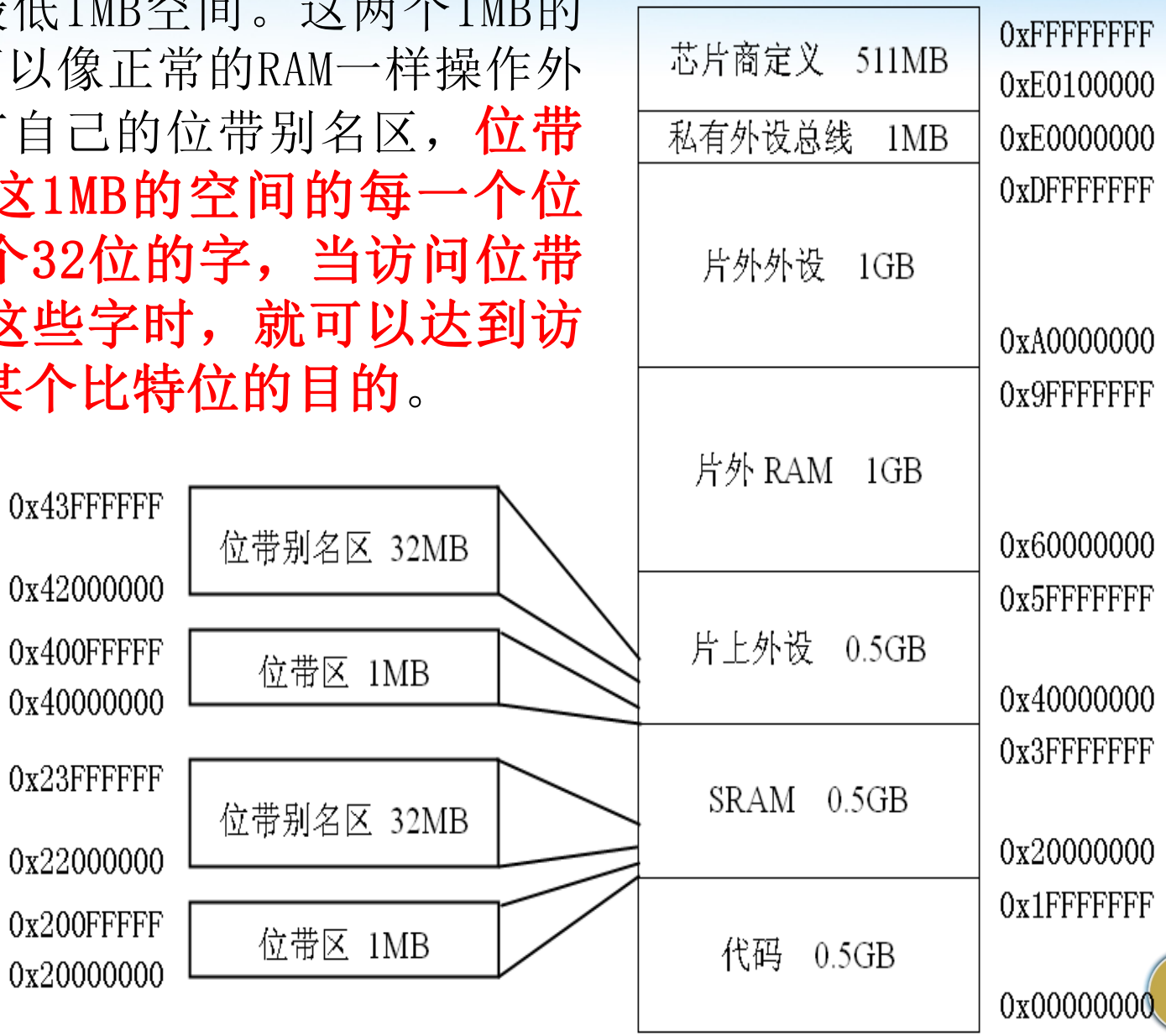
```
sfr P1 = 0x90; //定义P1 I/O 口,其地址90H

//指定位变量名所在的位置,
//当可寻址位位于特殊功能寄存器中时可采用这种方法
sbit P1_1 = P1 ^ 1;

P1_1=0;//P1口的1#引脚输出0
```

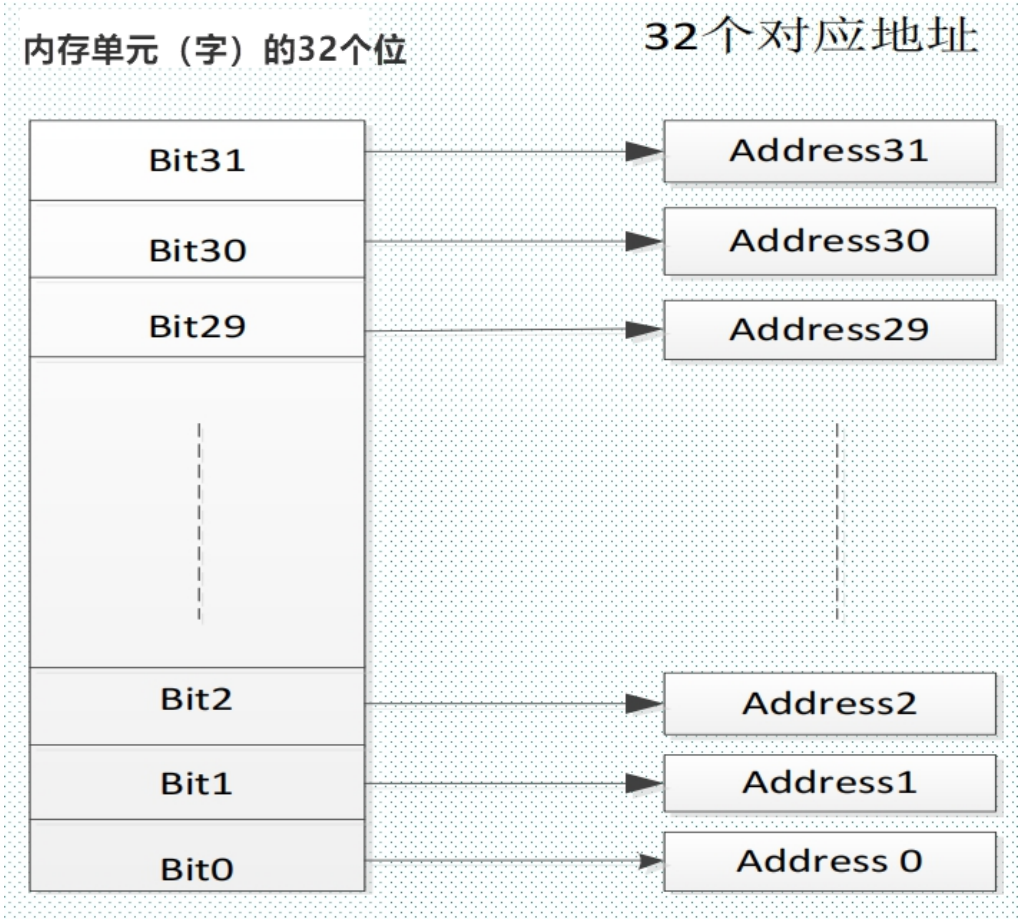
- STM32F103中没有这样的关键字，而是通过访问位带别名区来实现对位带区特定位的读写。

- 在STM32中，有两个地方实现了位带，一个是SRAM区的最低1MB空间，另一个是外设区最低1MB空间。这两个1MB的空间除了可以像正常的RAM一样操作外，他们还有自己的位带别名区，**位带别名区把这1MB的空间的每一个位膨胀成一个32位的字，当访问位带别名区的这些字时，就可以达到访问位带区某个比特位的目的。**

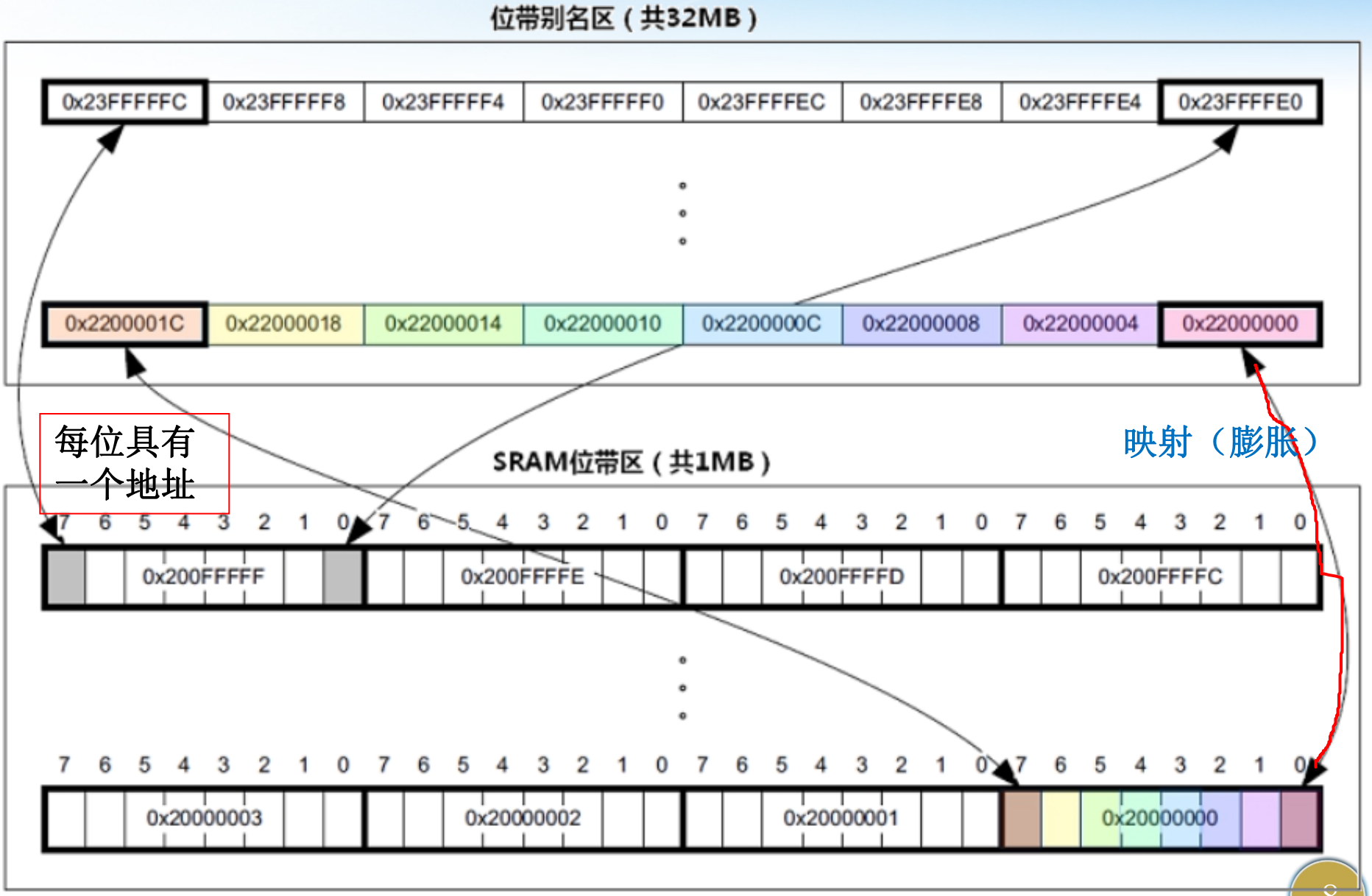


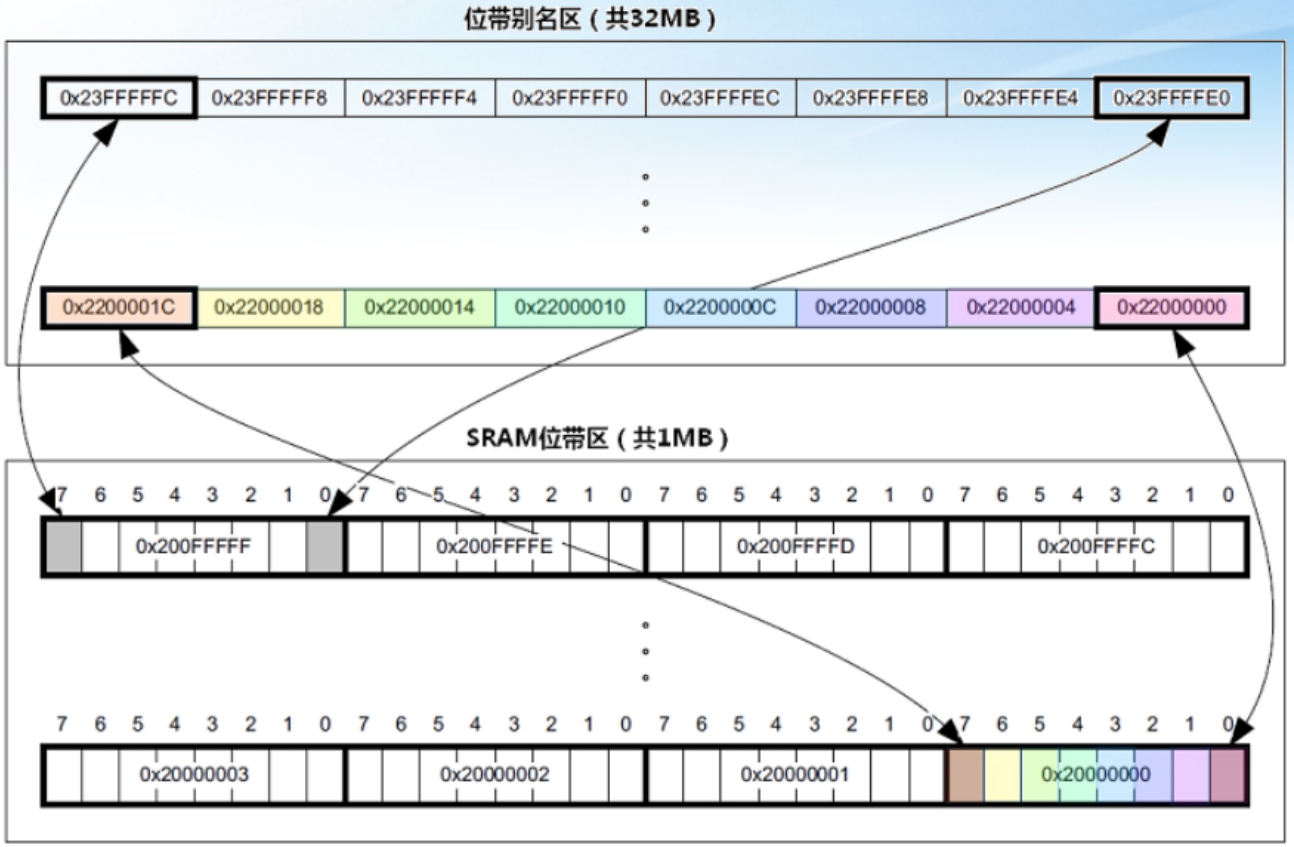
SRAM位带区

- SRAM的位带区为：0X2000_0000~0X200F_FFFF，大小为1MB，经过膨胀后的位带别名区为：0X2200_0000~0X23FF_FFFF，大小为32MB



位带区（Bit Band Region）





- W为位带区某个地址，k为改地址某个bit，A为位带别名区对应地址
- 对应关系为A=0x2200 0000+(W-0x2000 0000)×32+k×4，即位带区中地址W的第k位（记为W.k）对应着位带别名区中的地址A，对地址A（32位）的访问相当于访问W.k，即向A写入1，则W.k置1；向A写入0，则W.k清0。读出A相当于读出W.k。
- 位带别名区的每个字的内容只有第0位有效，其余的第[31:1]位保留。

SRAM位带别名区地址

- 对于SRAM位带区的某个比特，记它所在字节的地址为 A,位序号为 n(0<=n<=7)，则该比特在别名区的地址为：

AliasAddr=0x22000000+(A-0x20000000)*8*4 +n*4

或者写成：

AliasAddr=0x22000000+ (A-0x20000000)<<5 +n<<2

其中，0X22000000是SRAM位带别名区的起始地址，0x20000000是SRAM位带区的起始地址，（A-0x20000000）表示该比特前面有多少个字节，一个字节有8位，所以*8，一个位膨胀后是4个字节，所以*4，n表示该比特在A地址的序号，因为一个位经过膨胀后是四个字节，所以*4。

- 不用位带方式

```
LDR r0, =0x20000300
 LDR r1, [r0]              ; 读取数据
 ORR r1, r1, #0x4          ; D2位设置为1
 STR r1, [r0]              ; 写回结果
```

- 用位带方式

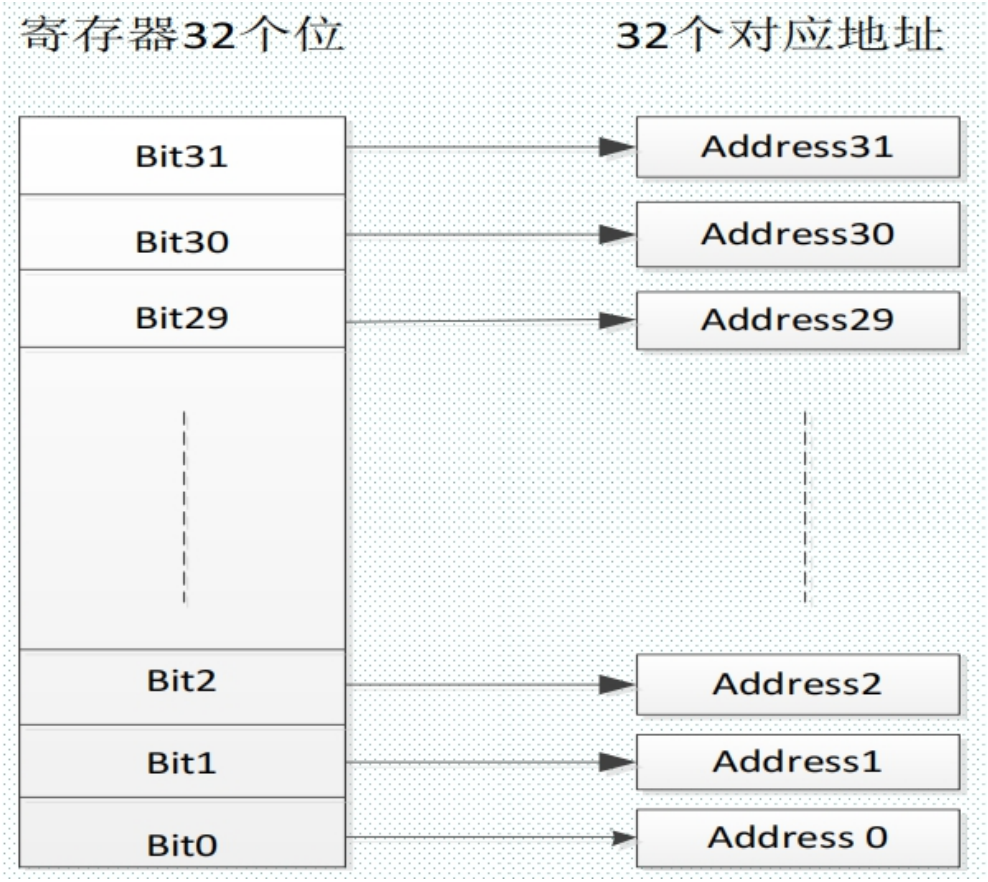
```
 LDR r0, =0x22006008
 MOV r1, #1          ; D2位设置为1
 STR r1, [r0]        ; 写入结果
```

位带(位绑定）操作的优点

- 简化操作
- 提高指令执行速度（32位cpu字对齐访问更高效）
- 保证执行过程的原子性

外设位带区

- 外设位带区的地址为：0X40000000~0X400FFFFF，大小为1MB，这1MB的大小包含了APB1、APB2和AHB上所有外设的寄存器，外设位带区经过膨胀后的位带别名区地址为：0X42000000~0X43FFFFFF，大小为32MB



外设位带别名区地址

- 对于片上外设位带区的某个比特，记它所在字节的地址为A,位序号为 n(0<=n<=7)，则该比特在别名区的地址为：

AliasAddr=0x42000000+ (A-0x40000000)*8*4 +n*4

或者写成：

AliasAddr=0x42000000+ (A-0x40000000)<<5 +n<<2

其中，0X42000000是外设位带别名区的起始地址，0x40000000是外设位带区的起始地址，（A-0x40000000）表示该比特前面有多少个字节，一个字节有8位，所以*8，一个位膨胀后是4个字节，所以*4，n表示该比特在A地址的序号，因为一个位经过膨胀后是四个字节，所以*4。

统一公式

- 为了方便操作，我们可以把这两个公式合并成一个公式，把"位带地址+位序号"转换成别名区地址统一成一个宏。

```
// 把"位带地址+位序号"转换成别名地址的宏
#define BITBAND(addr,bitnum) ((addr & 0xF0000000)+0x02000000+((addr & 0x000FFFFF)<<5)+(bitnum<<2))
```

其中：

- addr & 0xF0000000是为了区别SRAM还是外设，实际效果就是最高4位取出4或者2，
- addr & 0x000FFFFF 屏蔽了高12位，相当于减去0X200 00000或者0X400 00000，

- 最后我们就可以通过指针的形式操作这些位带别名区地址，最终实现位带区的比特位操作。

```
// 把一个地址转换成一个指针
#define MEM_ADDR(addr)  *((volatile unsigned long  *)(addr))
// 把位带别名区地址转换成指针
#define BIT_ADDR(addr, bitnum)   MEM_ADDR(BITBAND(addr, bitnum))
```

6.2 GPIO位带操作

- 外设的位带区，覆盖了全部的片上外设的寄存器，我们可以通过宏为每个寄存器的位都定义一个位带别名地址，从而实现位操作。以下以GPIO中ODR和IDR这两个寄存器的位操作举例：

从手册中我们可以知道ODR和IDR这两个寄存器对应GPIO基址的偏移是20和16，我们先实现这两个寄存器的地址映射，其中GPIOx_BASE在库函数里面有定义。

GPIO 寄存器映射

// GPIO ODR 和 IDR 寄存器地址映射

```
#define GPIOA_ODR_Addr    (GPIOA_BASE+20)
#define GPIOB_ODR_Addr    (GPIOB_BASE+20)
#define GPIOC_ODR_Addr    (GPIOC_BASE+20)
#define GPIOD_ODR_Addr    (GPIOD_BASE+20)
#define GPIOE_ODR_Addr    (GPIOE_BASE+20)
#define GPIOF_ODR_Addr    (GPIOF_BASE+20)
#define GPIOG_ODR_Addr    (GPIOG_BASE+20)
```

```
#define GPIOA_IDR_Addr    (GPIOA_BASE+16)
#define GPIOB_IDR_Addr    (GPIOB_BASE+16)
#define GPIOC_IDR_Addr    (GPIOC_BASE+16)
#define GPIOD_IDR_Addr    (GPIOD_BASE+16)
#define GPIOE_IDR_Addr    (GPIOE_BASE+16)
#define GPIOF_IDR_Addr    (GPIOF_BASE+16)
#define GPIOG_IDR_Addr    (GPIOG_BASE+16)
```

位带操作举例

```
//位带操作,实现51类似的GPIO控制功能
//具体实现思想,参考<<CM3权威指南>>第五章(87页~92页).
//IO口操作宏定义
#define BITBAND(addr, bitnum) ((addr & 0xF0000000)+0x2000000+((addr &0x000FFFFF)<<5)+(bitnum<<2))
#define MEM_ADDR(addr)  *((volatile unsigned long  *)(addr))
#define BIT_ADDR(addr, bitnum)   MEM_ADDR(BITBAND(addr, bitnum))
//IO口操作,只对单一的IO口!
//确保n的值小于16!
#define PAout(n)   BIT_ADDR(GPIOA_ODR_Addr,n)  //输出
#define PAin(n)    BIT_ADDR(GPIOA_IDR_Addr,n)  //输入
//IO口地址映射
#define GPIOA_ODR_Addr    (GPIOA_BASE+12) //0x4001080C
#define GPIOA_IDR_Addr    (GPIOA_BASE+8) //0x40010808
#define LED0 PAout(0)// PA0

LED0=0;
```

GPIO位操作

// 单独操作 GPIO的某一个IO口，
//n(0,1,2...16),n表示具体是哪一个IO口

```
#define PAout(n)    BIT_ADDR(GPIOA_ODR_Addr,n)
                                                    //输出

#define PAin(n)     BIT_ADDR(GPIOA_IDR_Addr,n)
                                                    //输入
```