13.1 位带简介

位操作就是可以单独的对一个比特位读和写，这个在 51 单片机中非常常见。51 单片机中通过关键字 sbit 来实现位定义， M4 中没有这样的关键字，而是通过访问位带别名区来实现。

在 M4 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。

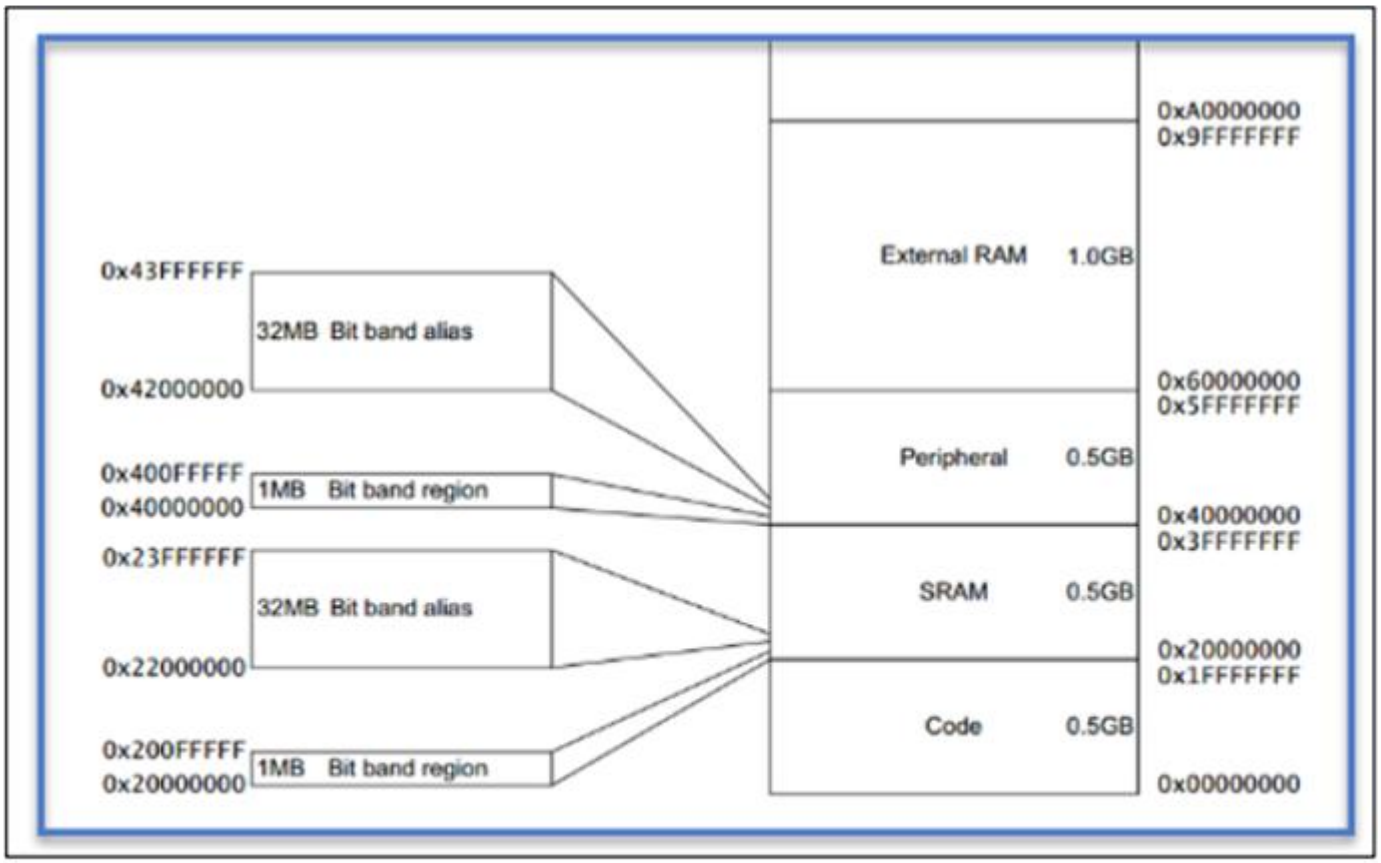

图 3 M4 位带地址

13.1.1 外设位带区

外设位带区的地址为：0X40000000~0X400F0000，大小为 1MB，这 1MB 的大小包含了 APB1/2 和 AHB1 上所以外设的寄存器，AHB2/3 总线上的寄存器没有包括。AHB2 总线上的外设地址范围为：0X50000000~0X50060BFF，AHB3 总线上的外设地址范围为：0XA0000000~0XA0000FFF。外设位带区经过膨胀后的位带别名区地址为：0X42000000~0X43FFFFFF，这部分地址空间为保留地址，没有跟任何的外设地址重合。

13.1.2 SRAM 位带区

SRAM 的位带区的地址为：0X2000 0000~X200F 0000，大小为 1MB，经过膨胀后的位带别名区地址为：0X2200 0000~0X23FF FFFF，大小为 32MB。操作 SRAM 的比特位这个用得很少。

13.1.3 位带区和位带别名区地址转换

位带区的一个比特位经过膨胀之后，虽然变大到 4 个字节，但是还是 LSB 才有效。有人会问这不是浪费空间吗，要知道 M4 的系统总线是 32 位的，按照 4 个字节访问的时候是最快的，所以膨胀成 4 个字节来访问是最高效的。

我们可以通过指针的形式访问位带别名区地址从而达到操作位带区比特位的效果。那这两个地址直接如何转换，我们简单介绍一下。

1. 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

该比特在别名区的地址为：

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

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，所以也*4。

2. 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：

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

公式分析同上。

3. 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。

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

addr & 0xF0000000 是为了区别 SRAM 还是外设，实际效果就是取出 4 或者 2，如果是外设，则取出的是 4，+0X02000000 之后就等于 0X42000000，0X42000000 是外设别名区的起始地址。如果是 SRAM，则取出的是 2，+0X02000000 之后就等于 0X22000000，0X22000000 是 SRAM 别名区的起始地址。

addr & 0x00FFFFFF 屏蔽了高三位，相当于减去 0X20000000 或者 0X40000000，但是为什么是屏蔽高三位？因为外设的最高地址是：0X2010 0000，跟起始地址 0X20000000 相减的时候，总是低 5 位才有效，所以干脆就把高三位屏蔽掉来达到减去起始地址的效果，具体屏蔽掉多少位跟最高地址有关。SRAM 同理分析即可。<<5 相当于*8*4，<<2 相当于*4，这两个我们在上面分析过。

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。

```
1 // 把一个地址转换成一个指针
2 #define MEM_ADDR(addr)  *((volatile unsigned long  *)(addr))
3
4 // 把位带别名区地址转换成指针
5 #define BIT_ADDR(addr, bitnum)   MEM_ADDR(BITBAND(addr, bitnum))
```

13.2 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。但这个在实际项目中不是很现实，也很少人会这么做，我们在这里仅仅演示下 GPIO 中 ODR 和 IDR 这两个寄存器的位操作。

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。

1. GPIO 寄存器映射

代码 9 GPIO ODR 和 IDR 寄存器映射

```
1  // GPIO ODR 和 IDR 寄存器地址映射
2  #define GPIOA_ODR_Addr    (GPIOA_BASE+20)
3  #define GPIOB_ODR_Addr    (GPIOB_BASE+20)
4  #define GPIOC_ODR_Addr    (GPIOC_BASE+20)
5  #define GPIOD_ODR_Addr    (GPIOD_BASE+20)
6  #define GPIOE_ODR_Addr    (GPIOE_BASE+20)
7  #define GPIOF_ODR_Addr    (GPIOF_BASE+20)
8  #define GPIOG_ODR_Addr    (GPIOG_BASE+20)
9  #define GPIOH_ODR_Addr    (GPIOH_BASE+20)
10 #define GPIOI_ODR_Addr    (GPIOI_BASE+20)
11 #define GPIOJ_ODR_Addr    (GPIOJ_BASE+20)
12 #define GPIOK_ODR_Addr    (GPIOK_BASE+20)
13
14 #define GPIOA_IDR_Addr    (GPIOA_BASE+16)
15 #define GPIOB_IDR_Addr    (GPIOB_BASE+16)
16 #define GPIOC_IDR_Addr    (GPIOC_BASE+16)
17 #define GPIOD_IDR_Addr    (GPIOD_BASE+16)
18 #define GPIOE_IDR_Addr    (GPIOE_BASE+16)
19 #define GPIOF_IDR_Addr    (GPIOF_BASE+16)
20 #define GPIOG_IDR_Addr    (GPIOG_BASE+16)
21 #define GPIOH_IDR_Addr    (GPIOH_BASE+16)

22 #define GPIOI_IDR_Addr    (GPIOI_BASE+16)
23 #define GPIOJ_IDR_Addr    (GPIOJ_BASE+16)
24 #define GPIOK_IDR_Addr    (GPIOK_BASE+16)
```

现在我们就可以用位操作的方法来控制 GPIO 的输入和输出了，其中宏参数 n 表示具体是哪一个 IO 口，n(0,1,2...16)。这里面包含了端口 A~K，并不是每个单片机型号都有这么多端口，使用这部分代码时，要查看你的单片机型号，如果是 176pin 的则最多只能使用到 I 端口。

2. GPIO 位操作

代码 10 GPIO 输入输出位操作

```
1  // 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
2  #define PAout(n)    BIT_ADDR(GPIOA_ODR_Addr,n)  //输出
3  #define PAin(n)     BIT_ADDR(GPIOA_IDR_Addr,n)  //输入
4
5  #define PBout(n)    BIT_ADDR(GPIOB_ODR_Addr,n)  //输出
6  #define PBin(n)     BIT_ADDR(GPIOB_IDR_Addr,n)  //输入
7
8  #define PCout(n)    BIT_ADDR(GPIOC_ODR_Addr,n)  //输出
9  #define PCin(n)     BIT_ADDR(GPIOC_IDR_Addr,n)  //输入
10
11 #define PDout(n)    BIT_ADDR(GPIOD_ODR_Addr,n)  //输出
12 #define PDin(n)     BIT_ADDR(GPIOD_IDR_Addr,n)  //输入
13
14 #define PEout(n)    BIT_ADDR(GPIOE_ODR_Addr,n)  //输出
15 #define PEin(n)     BIT_ADDR(GPIOE_IDR_Addr,n)  //输入
16
17 #define PFout(n)    BIT_ADDR(GPIOF_ODR_Addr,n)  //输出
18 #define PFin(n)     BIT_ADDR(GPIOF_IDR_Addr,n)  //输入
19
20 #define PGout(n)    BIT_ADDR(GPIOG_ODR_Addr,n)  //输出
21 #define PGin(n)     BIT_ADDR(GPIOG_IDR_Addr,n)  //输入
22
23 #define PHout(n)    BIT_ADDR(GPIOH_ODR_Addr,n)  //输出
24 #define PHin(n)     BIT_ADDR(GPIOH_IDR_Addr,n)  //输入
25
26 #define PIout(n)    BIT_ADDR(GPIOI_ODR_Addr,n)  //输出
27 #define PIin(n)     BIT_ADDR(GPIOI_IDR_Addr,n)  //输入
28
29 #define PJout(n)    BIT_ADDR(GPIOJ_ODR_Addr,n)  //输出
30 #define PJin(n)     BIT_ADDR(GPIOJ_IDR_Addr,n)  //输入
31
32 #define PKout(n)    BIT_ADDR(GPIOK_ODR_Addr,n)  //输出
33 #define PKin(n)     BIT_ADDR(GPIOK_IDR_Addr,n)  //输入
```