一、位带操作

1.意义

回想以前写 51 代码

```
P0 = 0x10;      //将 P0 端口设置为 0x10
P1_0=1;         //将 P1 端口 1 号引脚设置为高电平
a = P2_2; //获取 P2 端口 2 号引脚的电平
```

根据上述的方法，我们可以发现快速定位修改某个引脚的电平还有获取引脚的状态

GPIO_SetBits、GPIO_ResetBits 操作 IO 口的**性能**没有达到极致，因为这些函数都需要进行现场保护和现场恢复的动作，**比较耗时间**，没有进行一步到位，使用位带操作则没有上述的烦恼，**简单快速**！

```
GPIO_SetBits(GPIOF,GPIO_Pin_9);
```

修改为

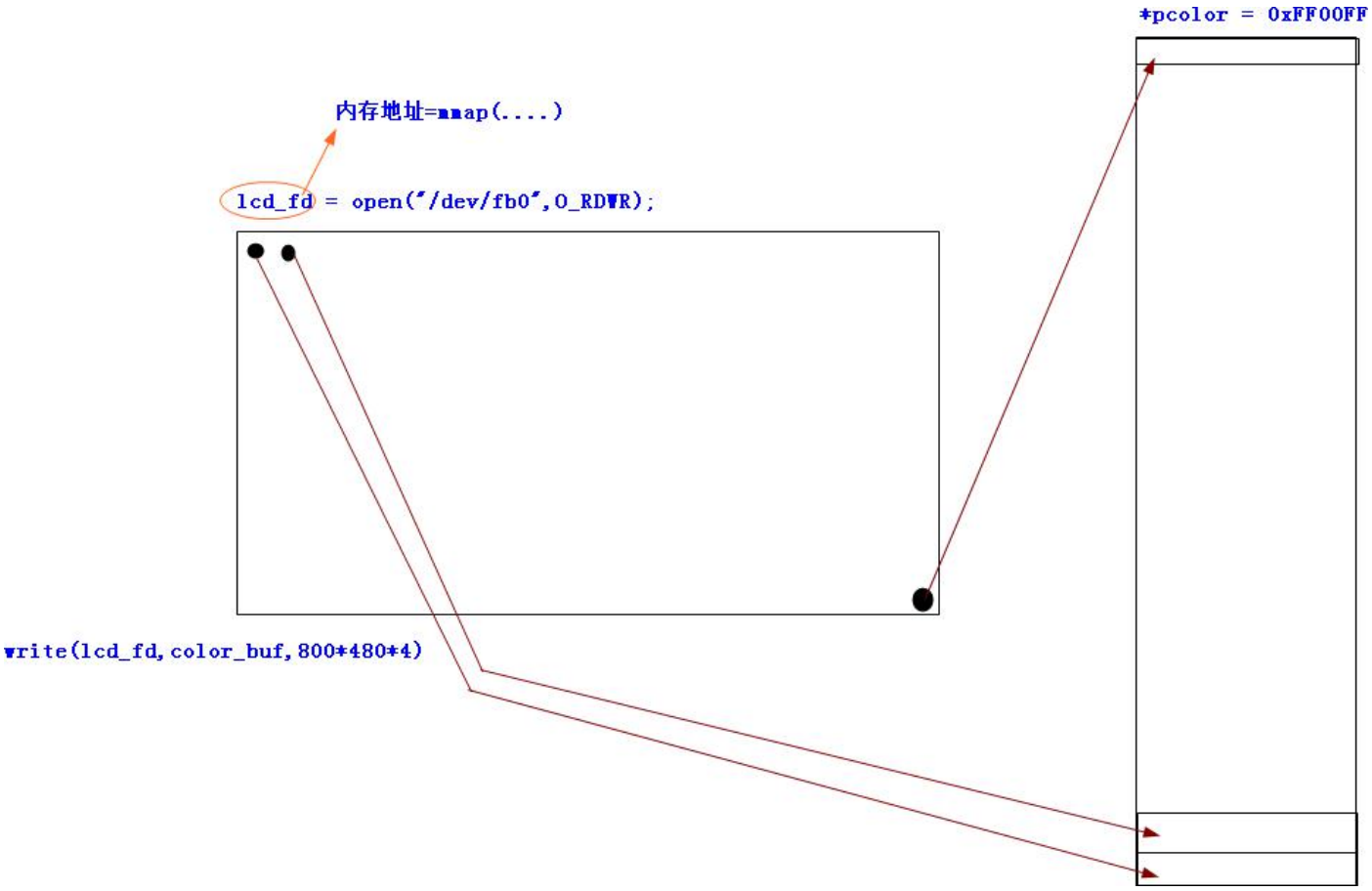
```
PFout(9)=1;
```

```
GPIO_ResetBits(GPIOF,GPIO_Pin_9);
```

修改为

```
PFout(9)=0;
```

可以理解为 LCD 编程的文件描述符映射到内存，相当于 mmap 函数，如下图。



2.参考资料

STM32F3 与 F4 系列 Cortex M4 内核编程手册.pdf 第 31 页 2.2.5 Bit-banding

3.相关寄存器的地址

a.查找 GPIOF 相关的地址

```
#define PERIPH_BASE                   ((uint32_t)0x40000000) /*!< Peripheral base address in the alias region


#define GPIOF_BASE                    (AHB1PERIPH_BASE + 0x1400)     //0x40001400
```

b.了解 GPIOF 相关寄存器

```
typedef struct
{
  __IO uint32_t MODER;       /*!< GPIO port mode register,                  Address offset: 0x00      */
  __IO uint32_t OTYPER;      /*!< GPIO port output type register,           Address offset: 0x04      */
  __IO uint32_t OSPEEDR;     /*!< GPIO port output speed register,          Address offset: 0x08      */
  __IO uint32_t PUPDR;       /*!< GPIO port pull-up/pull-down register,     Address offset: 0x0C      */
  __IO uint32_t IDR;         /*!< GPIO port input data register,            Address offset: 0x10      */
  __IO uint32_t ODR;         /*!< GPIO port output data register,           Address offset: 0x14      */
  __IO uint16_t BSRRL;       /*!< GPIO port bit set/reset low register,     Address offset: 0x18      */
  __IO uint16_t BSRRH;       /*!< GPIO port bit set/reset high register, Address offset: 0x1A        */
  __IO uint32_t LCKR;        /*!< GPIO port configuration lock register, Address offset: 0x1C        */
  __IO uint32_t AFR[2];      /*!< GPIO alternate function registers,        Address offset: 0x20-0x24 */
} GPIO_TypeDef;

#define GPIOF                         ((GPIO_TypeDef *) GPIOF_BASE)
```

c.最终操作 GPIOF 的时候，操作 GPIO_TypeDef 里的成员变量，参考例子为 GPIO_SetBits 与 GPIO_ResetBits 函数的内容。

d.如果要对 GPIOF 端口进行读写数据的时候，要对 GPIO_TypeDef->ODR 写入数据则对端口输出对应的电平；要获取端口的引脚电平，GPIO_TypeDef->IDR 进行读取。

```
GPIO_TypeDef->ODR，地址为 0x40001400+0x14          //端口输出数据地址

GPIO_TypeDef->IDR，地址为 0x40001400+0x10      //端口输入数据地址
```

公式如下：

寄存器的位带别名 = 0x42000000 + （寄存器的地址-0x40000000）*8*4 + 引脚编号*4

详细的映射流程可看图 04_位带别名的计算方法.bmp。

譬如 GPIOF 的 ODR 寄存器可以编写如下：

```
    __IO    uint32_t *pPF9_BitBand = (__IO uint32_t *)(0x42000000 + (GPIOF_BASE+0x14-0x40000000)*8*4 + 9*4);

  #define     __I     volatile                 /*!< Defines 'read only' permissions                 */
#else
  #define     __I     volatile const           /*!< Defines 'read only' permissions                 */
#endif
```

```
#define     __O     volatile             /*!< Defines 'write only' permissions     */
#define     __IO    volatile             /*!< Defines 'read / write' permissions   */
```

volatile 关键字分析，往往应用在三种场合

1）多线程编程共享全局变量的时候，该全局变量要加上 **volatile** 进行修饰，让编译器不要优化该变量。

2）裸机编程的时候，某函数与中断服务函数共享全局变量的时候，该全局变量要加上 **volatile** 进行修饰，让编译器不要优化该变量。

3）ARM 定义寄存器的时候，寄存器是指向一个地址，要加上 **volatile** 进行修饰，让编译器不要优化该变量。

编译器不要优化该变量指的是防止编译器出现优化过度，导致代码运行失效。

加上 volatile 关键字生成的汇编代码会发生明显的变化，同样调用 delay 函数，灯的速度发生变化！

-02 编译器二级代码优化

```
void delay(void)
{
    uint32_t i=0x500000;
    while(i--);
}
```

-00优化

```
11:
0x080004E0 F44F00A0  MOV     r0,#0x500000
12:         while(i--);
0x080004E4 BF00      NOP
0x080004E6 0001      MOVS    r1,r0
0x080004E8 F1A00001  SUB     r0,r0,#0x01
0x080004EC D1FB      BNE     0x080004E6
13: }
```

-02 编译器二级代码优化

```
0x0800047C F44F00A0  MOV     r0,#0x500000
0x08000480 1E40      SUBS    r0,r0,#1
12:         while(i--);
0x08000482 D2FD      BCS     0x08000480
13: }
```

-02 编译器二级代码优化

```
void delay(void)
{
    volatile  uint32_t i=0x500000;
    while(i--);
}
```

```
0x0800047E F44F00A0  MOV     r0,#0x500000
0x08000482 9000      STR     r0,[sp,#0x00]
12:         while(i--);
0x08000484 1E41      SUBS    r1,r0,#1
0x08000486 9100      STR     r1,[sp,#0x00]
0x08000488 D301      BCC     0x0800048E
0x0800048A 4608      MOV     r0,r1
0x0800048C E7FA      B       0x08000484
13: }
```

#练习 1

将 002/led_all_b 的代码修改为位带操作。

#练习 2

参考 003/参考资料/位带.pdf 文档，将位带操作的代码进行封装，接着将 002/keyn_ledn 的代码修改为位带操作。

提示：端口输入寄存器为 GPIOF->IDR 寄存器

#练习 3

阅读温老师提供的 sys.h，使用 Pxout 与 Pxin 替换之前的代码。例子如下：

```
*PF9_BitBand=0;
```

修改为

```
PFout(9)=0;
```

```
if(*PA0_InBitBand==0)
```

修改为

```
if(PAin(0) == 0)
```