

STM32地址映射、位带操作

目录

- 1：前言
- 2：地址（门牌号）
- 3：STM32地址映射
- 4：位带操作原理
- 5：位操作代码实现

1：前言

以前学51的时候，没有注重地址的这个概念，因为51 **寄存器** 少，一个reg52.h里面涵盖了你要用到的所有寄存器地址，你只需要去写几个字母调用一下就可以了。我甚至一度以为TMOD就是TMOD，单片机生产出来，程序就是这样写，就好像用手机指纹解锁，你只是把手指放上去，手机就解锁了。你以为理所应当，但是手机却做了很多工作。直到学习32后，由于庞大的寄存器数量，让我不得不去注意到为什么我写几个字母，就可以操作到我想要操作的寄存器上去。SO，地址概念也就被我重视起来。下面只是简单的一些我理解的32地址知识。如有不足，咋~~评论区过两招!!!

2：地址（门牌号）

“地址”：第一印象，就是一张地图。一个你家的门牌号。就是一条可以让你找的你想到位置的指引。这是在广义上我的理解在计算机狭义上：内存中每个用于**数据存取**的基本单位，都被赋予一个唯一的**序号**，称为地址，也叫做内存地址。bala~~bala~~bala~~bala~~地址就是“唯一”的，不管是在现实生活中，还是在单片机，嵌入式里面。都是为了方便你去快速准确找到你想要的。

3：STM32地址映射

不多说，让我们康康，32里面的地址是怎么操作的以GPIOA为例

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE); // 使能GPIOA时钟
```

可以看到，GPIOA是挂载在AHB1下



AHB1地址段如上，那么GPIOA肯定地址区间就在这个范围下面，挖掘GPIOA的映射

```
#define GPIOA ((GPIO_TypeDef *) GPIOA_BASE)

#define GPIOA_BASE (AHB1PERIPH_BASE + 0x0000)

#define AHB1PERIPH_BASE (PERIPH_BASE + 0x00020000)

#define PERIPH_BASE ((uint32_t)0x40000000)

// 最终得到GPIOA的首地址为0x40020000

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;
```

GPIO port mode register	GPIO port output type register
偏移地址：0x00	偏移地址：0x04

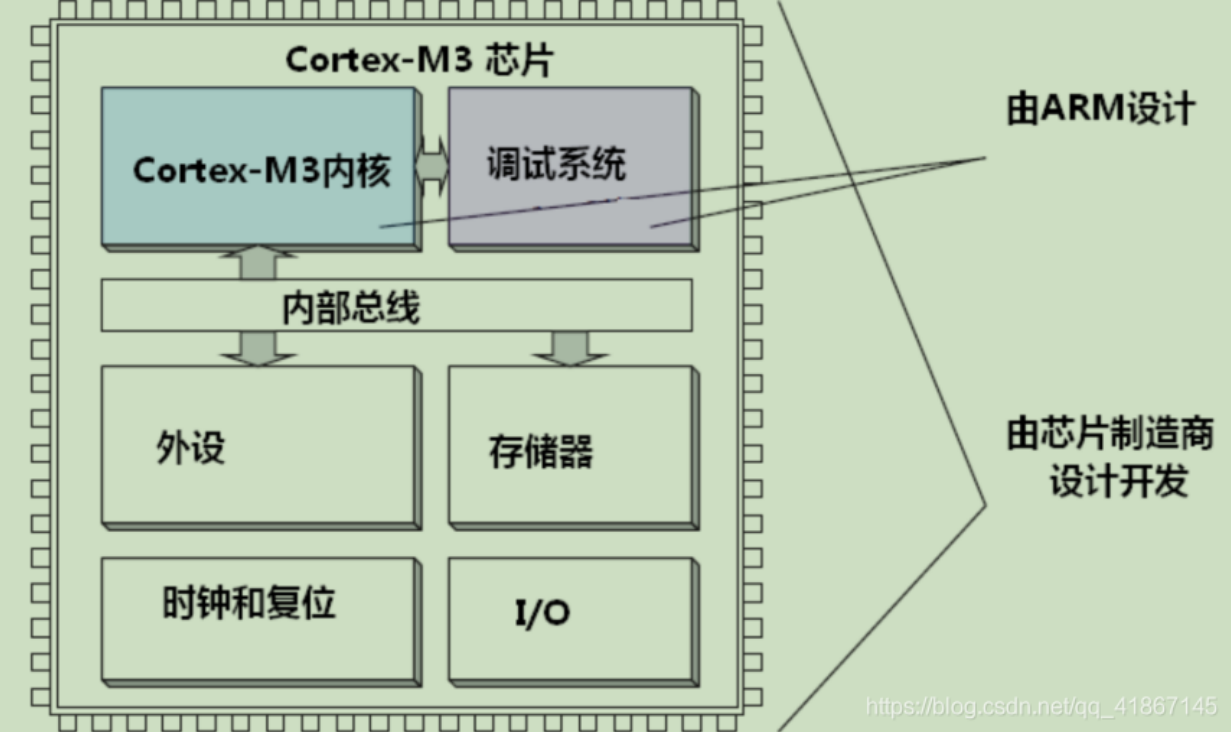
巧了，就是AHB1的首地址，这不就好起来了又由于GPIO下各寄存器已经在GPIO_TypeDef的结构体中都有声明，并且按照相对应的地址偏移顺序排列，这样就可以做到诸如GPIOx->MODER，这样的操作来访问到相对应GPIO的MODER寄存器了

GPIO port mode register	GPIO port output type register
偏移地址：0x00	偏移地址：0x04

像这种通过一层层的将 **地址映射** 起来的操作32里面一大推，道理都是一样。

4：位带操作原理

eg:STM32F407ZGT6注：字=4字节=32比特STM32F407ZGT6是基于Cortex_M4处理器内核的(我们以M3为例是一样的)，在芯片制造商得到CM3处理器内核的使用授权之后，它们就可以把CM3内核用在自己的硅片设计中，添加存储器，外设，I/O以及其它功能块。



5: 位操作代码实现

```
#define LED0 PFout(9) // D50

A: #define PFout(n) BIT_ADDR(GPIOF_ODR_Addr,n) // 输出

#define GPIOF_ODR_Addr (GPIOF_BASE+20) //0x40021414

#define GPIOF_BASE (AHB1PERIPH_BASE + 0x1400)
#define AHB1PERIPH_BASE (PERIPH_BASE + 0x00020000)
#define PERIPH_BASE ((uint32_t)0x40000000)

得到GPIOF_ODR的值为0x4002 1414 (PF的首地址)

B: #define BIT_ADDR(addr, bitnum) MEM_ADDR(BITBAND(addr, bitnum))

#define BITBAND(addr, bitnum) ((addr & 0xF0000000)+0x20000000+((addr & 0xFFFF)<<5)+(bitnum<<2))

经过计算, 有错请指正: 0x4200 0000+0x0048 8280+0x0000 0024=0x424282a4;

#define MEM_ADDR(addr) *((volatile unsigned long *) (addr))

将上述算出地址强制转换为unsigned long 类型的指针, 前面加*, 得到该地址中的数据然后返回给B, B再返回给A, A再返回给LED0,就实现的类似于51的位操作。
https://blog.csdn.net/qq_41867145
```

可以看到最终是访问地址0X4242 82A4,正是在片上外设区的位带别名区的地址区间内, 可以看我上面发的图。这样, 一顿操作就将PF9偷天换日到了片上外设别名区上。仔细一想, 有点像C语言里面的函数传地址的概念。一间房子 (地址), 我复制了一把钥匙放在另一个人的手上, 这样, 这个人就可以获取, 修改这个房间 (地址) 里面的内容了。