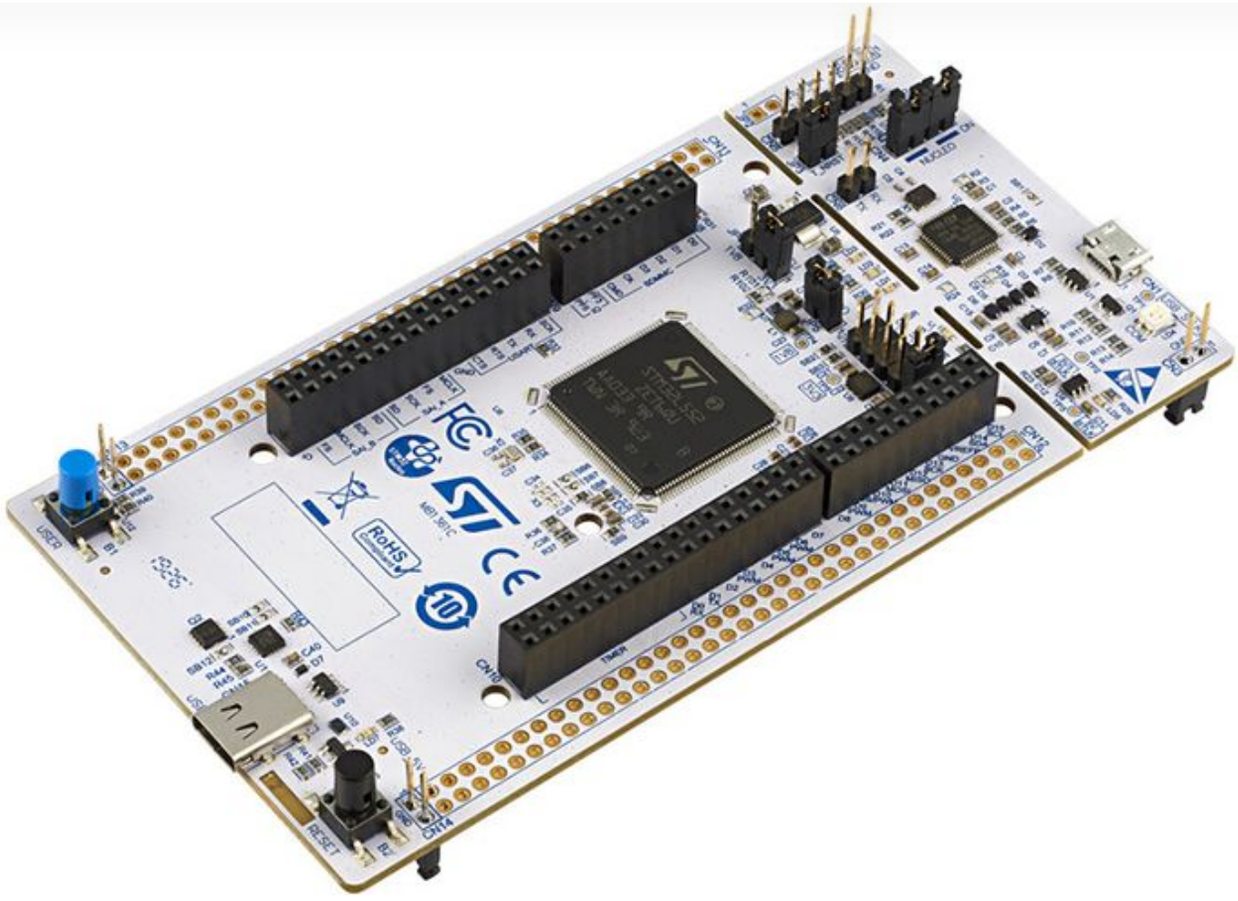


STM32位带操作全解

好奇
业精于勤荒于嘻（不爱回私信的答主）
关注他

本文属于知识整理，内容主要来自本人的学习笔记，主要参考书籍为《Cortex M3和M4权威指南》、《Cortex M3权威指南（中文版）》《野火零死角玩转STM32》，由于本人学习笔记很多为个人理解的内容，可能有错漏的地方，如果您阅读本文过程中发现有说得不对的，您可以在评论区留言指出，本人将感激不尽。

本文将从下面几个问题展开进行介绍：

什么是位带操作、什么是位带区、什么是位带别名区、怎么使用位带操作、为什么要使用位带操作（或者说位带操作有什么优点）、什么时候使用位带操作。

一、什么是位带操作?

使用过51单片机的小伙伴应该有过类似的操作，想要点亮一个LED灯只要一个P2^1 = 0；就可以实现了，非常方便，其实，所谓的位带操作就等价于51单片机的位操作，但是在STM32里面，内核的最小寻址单位是字节，一个字节里面有8bit，那么，STM32又是怎么实现位操作的？要搞清楚这个问题，就要先弄明白下面两个问题，什么是位带区、什么是位带别名区。

二、什么是位带区

介绍位带区之前，我们先来看一下Cortex-M内核里面寻址空间是怎么分配的

这是一张Cortex-M内核寻址空间映射图，来自《Cortex M3和M4权威指南》

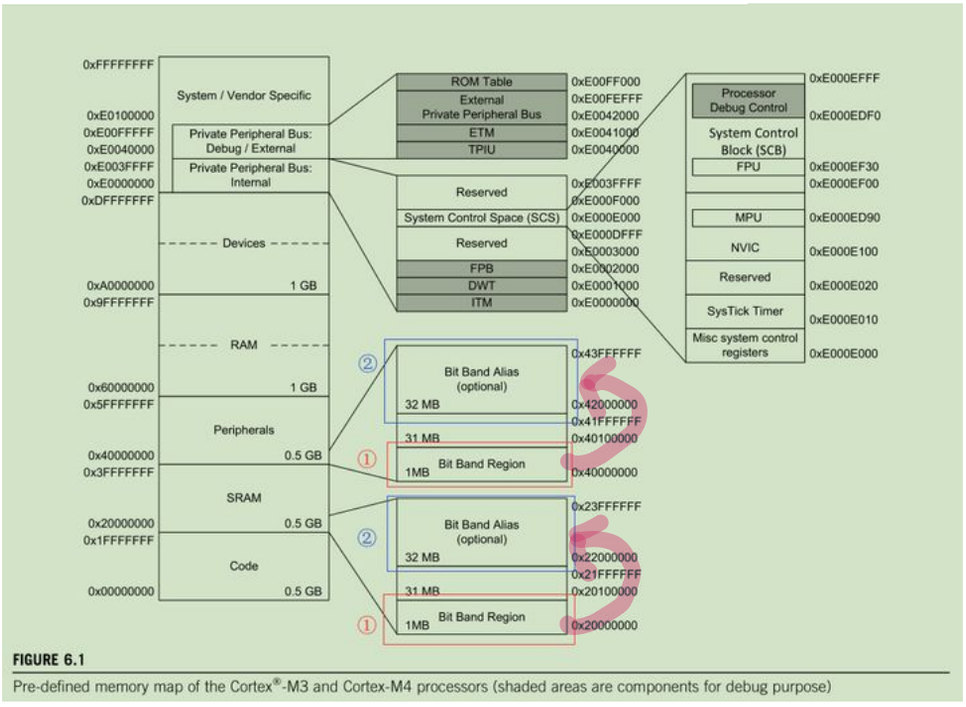


FIGURE 6.1

Pre-defined memory map of the Cortex®-M3 and Cortex-M4 processors (shaded areas are components for debug purpose)

我们可以看到图中有两个位带区，分别是SRAM区里的0X20000000-0X200FFFFF地址段和片内外设区里的0X40000000-0X400FFFFF地址段（图中标号①处），它们的地址空间大小都是1M字节，在SRAM和外设地址段内的这1M大小的空间就是位带区，说白了就是支持位带操作的区域就是位带区。我们上面已经说过了，内核的最小寻址单位是字节，那么怎么将寻址单位缩小到bit？要弄明白这个问题，那就要先弄明白什么是位带别名区了。

三、什么是位带别名区

从位带别名区名字上理解，感觉它像是别人的替身，实际使用上它就是别人的替身（电影里出名的武打明星都有自己的替身，STM32这么出名，肯定也有自己的替身啦，危险的操作都让替身去做，自己躺着就行了），**位带别名区就是为位带区服务的，对位带别名区的操作最终都会反映在位带区上，我们操作位带别名区的时候就等价于在操作位带区地址，**那么位带别名区与位带区又是什么样的关联关系呢？

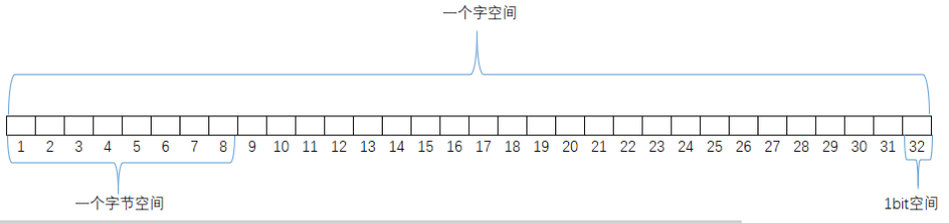
从上面映射图上可以看到，SRAM区里的0X22000000-0X23FFFFFF地址段和外设区里0X42000000-0X43FFFFFF地址段都是位带别名区，两个别名区空间大小都是32M。那么，这32M的**位带别名区**地址空间是怎么与1M的**位带区**地址空间对应起来的呢？其实，工程师们想出了一个很好的办法，地址映射，将1M字节里面的每一个bit映射到32M字节里面去，那么怎么映射呢？看到这里可能有些小伙伴就亿脸懵逼了，懵逼的话可以看一下下面的演算。

各种单位运算关系

1字节 = 8bit

1字 = 4字节

如果对这些单位没有什么概念，可以看下图



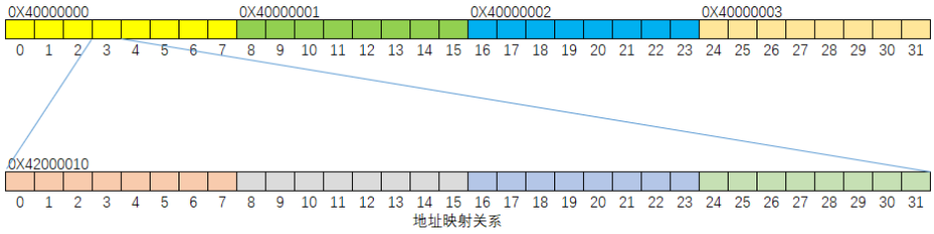
将1bit映射到1个字空间里（如下图）

映射前的一个字节 = 映射后的8个字

那么就有

映射前的一个字节 = 映射后的32个字节

映射前的1M字节 = 映射后的32M字节



好奇的小伙伴可能就要问了，为什么要将1bit空间要映射到一个字空间里去呢？我映射到1字节或者2字节的地址空间不行吗？我只能说，STM32是一个32位的机器，内核按字寻址的话寻址速度是最快的，所以别问这么多为什么，如果问了，答案就是为了速度。

四、怎么使用位带操作

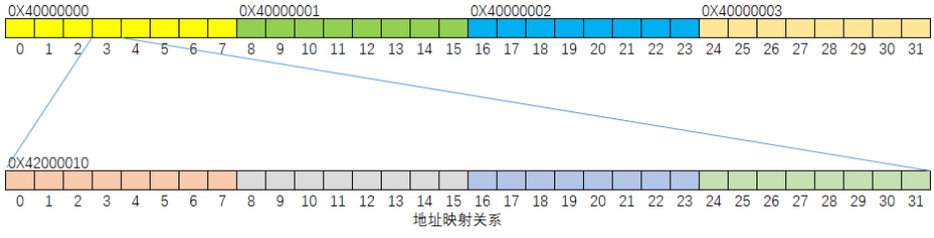
上面我们已经知道了位带区就是支持位操作的地址段，位带别名区就是位带区的地址映射，操作位带别名区就等价于操作位带区，并且我们知道了大致的映射过程，那么在STM32实际使用中又是怎么应用的呢？

在《Cortex M3和M4权威指南》里有这么一段话

> located in the first 1MB of the SRAM region, and the other is located in the first 1MB of the peripheral region. These two memory regions can be accessed like normal memory, but they can also be accessed via a separate memory region called the *bit-band alias*. When the bit-band alias address is used, each individual bit can be accessed separately in the least significant bit (LSB) of each word-aligned address (Figure 6.7).

通过这个，我们就知道了，位带别名区的每个地址对齐的字都是最低位有效的，也就是说在映射关系里，我们可以得到下面的结论

在位带别名区里，只要最低位是1，那么对应的位带区的位就是1。



上图的映射关系操作示例如下：

0X40000000地址位3置1操作：

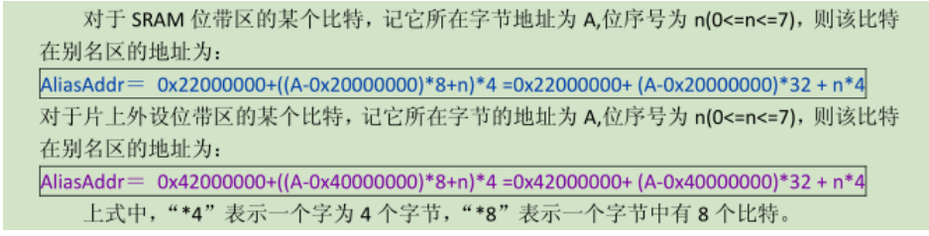
```
*(uint32_t*)(0x40000000) |= (1<<3);
//等价于
```

```
*(uint32_t*)(0x40000010) = 0x03;
//也等价于
*(uint32_t*)(0x40000010) = 0x05;
```

0X40000000地址位3置0操作：

```
*(uint32_t*)(0x40000000)  &= ~(1<<3);
 //等价于
*(uint32_t*)(0x40000010) = 0x00;//只要最低位是0即可
//也等价于
*(uint32_t*)(0x40000010) = 0x02;
//也等价于
*(uint32_t*)(0x40000010) = 0x04;
```

在《Cortex M3权威指南》中，前人已经整理出了位带别名区与位带区地址对应关系的表达式，使用的时候只要套用公式就可以，如下图：



将两个公式合并一下就得到：

AliasAddr = ((A & 0xF0000000)+0x02000000+((A &0x00FFFFFF)<<5)+(n<<2))

式中A为位带区地址，n为位序号

实际应用中，我们可以参考野火的方案，通过宏定义将公式进行封装，封装完后，所有的GPIO端口如下表示

```
// 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)

// 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))


// 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 GPIOH_ODR_Addr    (GPIOH_BASE+20)
#define GPIOI_ODR_Addr    (GPIOI_BASE+20)
#define GPIOJ_ODR_Addr    (GPIOJ_BASE+20)
#define GPIOK_ODR_Addr    (GPIOK_BASE+20)

#define GPIOA_IDR_Addr    (GPIOA_BASE+16)
#define GPIOB_IDR_Addr    (GPIOB_BASE+16)
#define GPIOC_IDR_Addr    (GPIOC_BASE+16)
```

```
#define GPIOF_IDR_Addr    (GPIOF_BASE+16)
#define GPIOG_IDR_Addr    (GPIOG_BASE+16)
#define GPIOH_IDR_Addr    (GPIOH_BASE+16)
#define GPIOI_IDR_Addr    (GPIOI_BASE+16)
#define GPIOJ_IDR_Addr    (GPIOJ_BASE+16)
#define GPIOK_IDR_Addr    (GPIOK_BASE+16)


// 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

#define PBout(n)   BIT_ADDR(GPIOB_ODR_Addr,n)  //输出
#define PBin(n)    BIT_ADDR(GPIOB_IDR_Addr,n)  //输入

#define PCout(n)   BIT_ADDR(GPIOC_ODR_Addr,n)  //输出
#define PCin(n)    BIT_ADDR(GPIOC_IDR_Addr,n)  //输入

#define PDout(n)   BIT_ADDR(GPIOD_ODR_Addr,n)  //输出
#define PDin(n)    BIT_ADDR(GPIOD_IDR_Addr,n)  //输入

#define PEout(n)   BIT_ADDR(GPIOE_ODR_Addr,n)  //输出
#define PEin(n)    BIT_ADDR(GPIOE_IDR_Addr,n)  //输入

#define PFout(n)   BIT_ADDR(GPIOF_ODR_Addr,n)  //输出
#define PFin(n)    BIT_ADDR(GPIOF_IDR_Addr,n)  //输入

#define PGout(n)   BIT_ADDR(GPIOG_ODR_Addr,n)  //输出
#define PGin(n)    BIT_ADDR(GPIOG_IDR_Addr,n)  //输入

#define PHout(n)   BIT_ADDR(GPIOH_ODR_Addr,n)  //输出
#define PHin(n)    BIT_ADDR(GPIOH_IDR_Addr,n)  //输入

#define PIout(n)   BIT_ADDR(GPIOI_ODR_Addr,n)  //输出
#define PIin(n)    BIT_ADDR(GPIOI_IDR_Addr,n)  //输入

#define PJout(n)   BIT_ADDR(GPIOJ_ODR_Addr,n)  //输出
#define PJin(n)    BIT_ADDR(GPIOJ_IDR_Addr,n)  //输入

#define PKout(n)   BIT_ADDR(GPIOK_ODR_Addr,n)  //输出
#define PKin(n)    BIT_ADDR(GPIOK_IDR_Addr,n)  //输入
```

以上代码来自野火开源例程

个人提示：

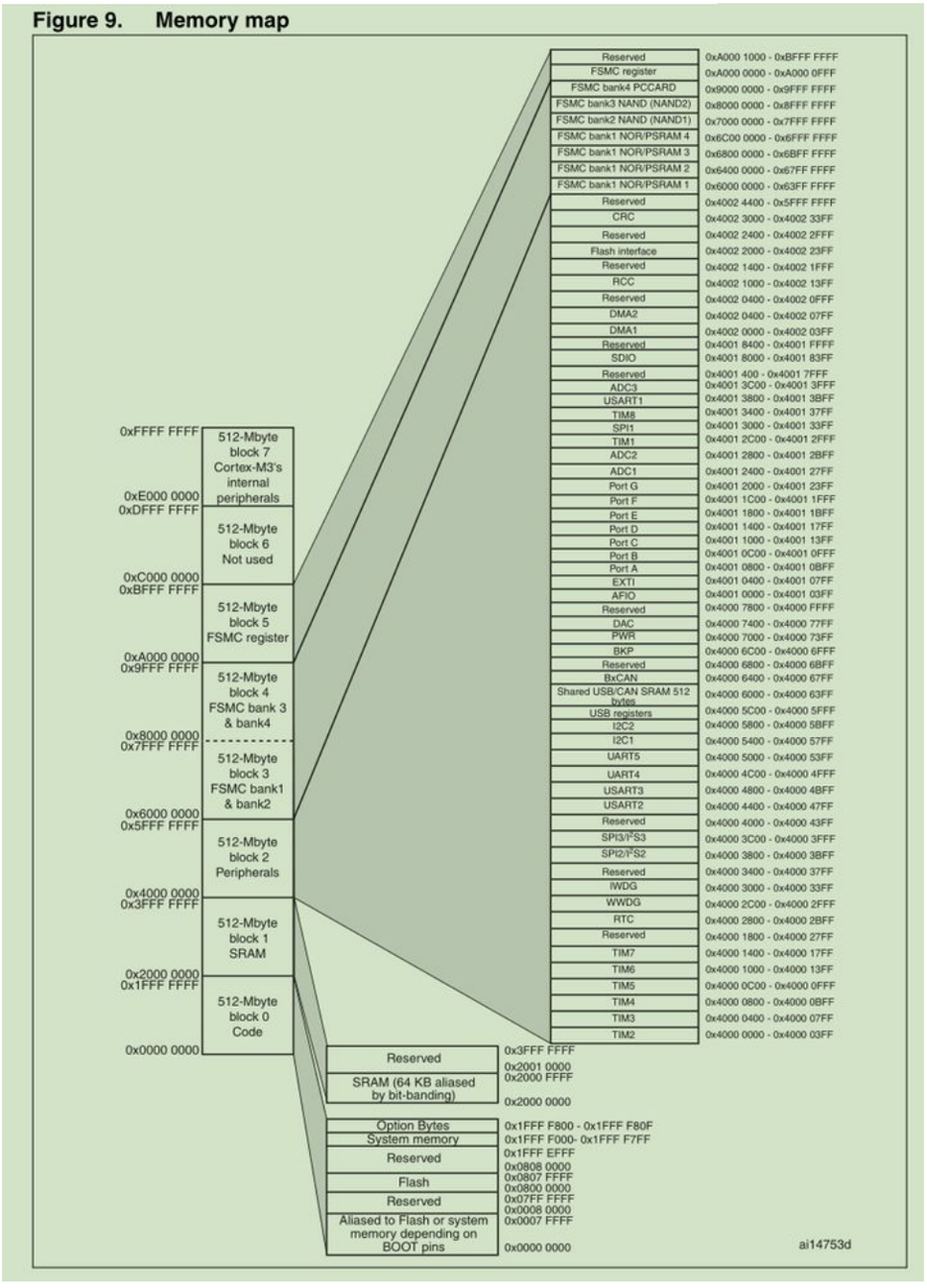
1、使用上面封装好的位带操作之前，要先对IO端口进行配置，否则操作结果不可预期。

2、PAout（n）作为左值使用，PAin（n）作为右值使用。（跟51单片机一样，你们应该知道我在说什么的）

3、如果觉得宏定义麻烦，也可以使用结构体对位带别名区进行封装，具体封装过程可以参考STM32标准库封装寄存器的方法，这里就不整理了。

4、另外，从STM32片内外设地址空间映射图（F10X系列）可以看到，我们不仅可以使用公式对所有GPIO端口进行封装，我们也可以对STM32所有片内外设的寄存器进行封装（FSMC除外），但是我们使用片内外设基本都是通过固件库配置一次就可以，不需要经常性的反复配置，所以就没有封装的必要了。

5、最后，使用的过程中要注意一点，强制地址转换的时候一定要使用volatile关键字进行修饰，否则这个操作可能会被编译器优化掉。



STM32F10X系列地址空间映射图

五、为什么要使用位带操作?

总结来说，一个是因为访问速度快，另一个是因为安全。

如果在裸机开发中，位带操作相比于直接的读-改-写操作除了访问速度快一点以外好像也没有什么可以说的了，但是如果在带操作系统的开发中，多任务并发运行的时候就有可能在任务切换的过程中发生不可预料的问题，而位带操作由于是属于硬件完成的不可被异常打断的操作（原子操作），所以相对于读-写-改的操作模式的话会更安全些。

六、什么时候使用位带操作?

在参考的《Cortex M3权威指南》中推荐的是在IO密集型的底层代码中使用，个人理解是：相比于库函数操作，位带操作访问速度会快很多（原子类型的寄存器操作），如果在时序要求较严格的情况下，可以使用位带操作，如使用IO口模拟某种通信协议。另外，由于位带操作异常不可打断（原

以上均为在阅读《Cortex M3权威指南》以后的学习总结，整理发表不易，喜欢的话可以点赞收藏。

学习过程中发现一些讲解位带操作比较好的视频，部分内容也来自一些视频，在这里也分享给大家。

【嵌入式】cortex系列单片机中的位带操作有什么用？_哔哩哔哩 (゜- ゜)つロ 干杯~...
www.bilibili.com/video/BV1y4411M71M

第17讲 入门篇—— 位带操作（1）_哔哩哔哩 (゜- ゜)つロ 干杯~-bilibili
www.bilibili.com/video/BV1ne411s7eN

第17讲 入门篇——位带操作理论分析（2）_哔哩哔哩 (゜- ゜)つロ 干杯~-bilibili
www.bilibili.com/video/BV1dT4y1g7my

第17讲 入门篇——位带操作理论分析（3）_哔哩哔哩 (゜- ゜)つロ 干杯~-bilibili
www.bilibili.com/video/BV1sC4y1W7FS

第17讲 入门篇——位带操作理论分析（4）_哔哩哔哩 (゜- ゜)つロ 干杯~-bilibili
www.bilibili.com/video/BV1Bt4y1y7Yq

编辑于 2020-05-21 17:06

STM32 嵌入式设计 嵌入式系统

推荐阅读

STM32及内部资源简介

本篇文章主要是让大家简单了解认识STM32 并了解本实验平台上的外设资源。 首先，我们要知道STM32 最小系统的组成有哪些，它包含一个 STM32 的主控芯片、电源电路、时钟电路、复位电路、...

AI电堂 发表于STM32...

【S
析
Boo

9 条评论 切换为时间排序

写下你的评论...

三人影 2020-12-24

解释的很好，不过在 0X40000000地址位3置1和置0操作 处的代码有误，应该将

▲ 赞同 50 ▼ 9 条评论 分享 喜欢 收藏 申请转载 ···

改为

” *(uint32_t*)(0x42000010) = 0x01; //只要最低位是1即可 “。

其他地方也是相应的修改。

1

糊毅樊 回复 三人影 2021-01-05

我觉得这里应该是0x4200000C才对

赞

王亚樵 2021-01-08

感谢分享，学到了

2

木图 2020-12-17

STM32是32位的，内核最小寻址单位应该是双字吧，32位

1

骑车上月球 01-13

全文精华：PAout（n）作为左值使用，PAin（n）作为右值使用。
提示：那如果之前已经把GPIOA设置为输出模式，这里再使用PAin(n)也是不矛盾的，因为即使IO是输出模式下，读取IO状态仍是可用的。

赞

123456789 2021-11-11

请问一下，现在哪里还能找到野火的零死角玩转stm32的pdf，我下载的都是坏了，打不开

赞

欣于所遇向之所欣 回复 123456789 2021-12-21

直接在野火论坛啊

赞

笑苏辰 2021-04-02

讲解的很好了！学习了！

赞

sxr1223 2020-08-16

请问能用-O0优化代替volatile关键字吗?

赞