本来只是路过，写详细一点。

我看楼主浮躁得不得了。现在什么都不要做了，先去看几遍《不要做浮躁的嵌入式工程师》这篇文章，想清楚了，

再动手吧。

我做了个实例，不用ST的库来点LED，解答你的问题

我的 KeilMDK 3.5

我的STM32板子奋斗版是 ，IC 是 STM32F103VET6

调试工具 JLINK V8

LED 接在 PB5 ，高电平点亮

既然楼主说一定懂C语言了，那么对于下面我的问题，不查百度，完全靠自己，懂多少？然后查了百度之后又能懂多少？

（一）新建 keil 工程，IC选择 ST 公司的 STM32F103VE，keil提示是否copy 启动文件，选择是。

这里有问题问楼主，

你有没有读过这个启动头文件？ 51 也是同样的启动文件，51的那个启动文件有没有读过？你知道

头文件里面做了什么吗？ C语言真的从 main 函数开始吗？运行时库是什么？这些资料从

什么地方知道？keil编译器的行为？

 （如果你说头文件是汇编的，没有必要看，那我当我没说）

例如启动文件里面有这么一句，我的问题是 \_\_main 这个标号在哪里实现的，注意，这里肯定不是 main 函数

这里跳到哪里去了？还有个问题 [WEAK] 这里是什么意思？有什么用？？？？

Reset\_Handler   PROC

                EXPORT  Reset\_Handler             [WEAK]

                IMPORT  \_\_main

                LDR     R0, =\_\_main

                BX      R0

（二）新建一个 main.c 并且写一个 main函数，什么都不做，这和51一样了。

void main(void)

{

while (1)

{

}

}

然后因为我需要调试，则设置jlink调试器，在项目属性里面 Debug 标签，Use J-LINK/J-TRACE ，然后到

utilities 标签，同样选择J-LINK /J-TRACK ，并且选择 Setting 按钮，里面的 Programming Algorithm

还是空的，表示keil 不知道目标是什么，我添加一个 STM32F10X High-density Flash ，问题，为什么是

High-desity ？依据是什么？？？

全部确认返回。

这个时候已经可以编译，开发板上电，已经可以下载仿真的，虽然程序什么都没有写

（三）既然硬件，仿真器，调试都准备好了，接着就开始写程序了。

我一直推荐新手花钱买学习板和仿真器，因为可以排除硬件的问题，让初学者集中精力去写程序，而不用怀疑

硬件有问题，这点很重要。

这阶段主要是看书，了解这个IC 的架构，了解指令集，了解寄存器（别跟我说你找不到这些资料？ .....）

Cortex-M3权威指南CnR2（电子书）.pdf

STM3210x参考手册.pdf

学习板原理图

博客，论坛等多个帖子，务必要对整个IC有个初步的了解。这个过程有点痛苦，但是值得花这个时间。

（四）开始写 LED

既然我们要操作 IO 口，当然就要看IO口相关的知识。打开 STM3210x参考手册.pdf ，我的目的只是操作 GPIO

所以我只需要将第五章看完就OK了。章节比较多，懒得看，根据一般的经验（楼主，你缺经验了吧？），不说多

就AVR 和 PIC 而已。操作IO一般是两个步骤，第一，操作IO控制寄存器，设置IO为输出，第二就是送数据。

那么很明显，只可能是 GPIOx\_CRL  GPIOx\_CRH ， GPIOx\_ODR 三个寄存器会有想要

仔细阅读这几个寄存器的介绍后知道，GPIOx\_CRL 是控制 PIN 0-7 的属性的，GPIOx\_CRH 控制PIN 8-15，ODR寄存器

当然就是输出数据了，将数据送到这里就行了。

然后，这几个寄存器的地址是多少？首先看 stm32f103ve.pdf 这个是官方的datasheet、，看第四章， Mmeory Mapping

为什么看这章？会英文都能猜到吧？，看 PORTB 的地址是 0x40010C00 - 0x40010FFF ，这个就是基地址了。基地址

加上偏移量就能找到具体的寄存器。

例如我需要操作 GPIOB\_CRL 的偏移为 00H ，（看STM3210x参考手册.pdf） ODR 寄存器的偏移为 0CH

那么很自然得出

GPIOB\_CRL = 0x40010C00

GPIOB\_ODR = 0x40010C0C

怎么验证我的结论正确？先看 keil 给的头文件 \Keil\ARM\INC\ST\STM32F10x\stm32f10x\_map.h

#define PERIPH\_BASE           ((u32)0x40000000)

#define APB2PERIPH\_BASE       (PERIPH\_BASE + 0x10000)

#define GPIOB\_BASE            (APB2PERIPH\_BASE + 0x0C00)

这样怎么算都能算出 0x40010C00 出来吧？？ODR 寄存器同理

为了点亮 LED ，我需要将 PB5 （也就是 GPIOB5）设置为输出，并且ODR相应的位写入 1 ，看资料得出 MODE5 是

bit 20 21 控制的，CNF5 是bit 22,23

MODE5应该设置 10（0x2) 选择 2MHZ 输出，CNF5 选择00（0x0），通用推挽模式，于是将这个值写入

(\*volatile unsigned long)0x40010C00 = (2<<20) | (0<<22);  // 为简单起见，不管其他位了

楼主你是否能看懂这句C语言？？volatile 什么意思什么用？指针的本质是什么？为什么能这样用？2<<20 是什么

意思，为什么能这样用？楼主我真的不是为难你，嵌入式都这么写的，ST的头文件也是这么定义

同理，设置 ODR 寄存器

\*(volatile unsigned long \*)0x40010C0C = 1<<5;

\*(volatile unsigned long \*)0x40010C0C = 0;

STM32 没有SFR ，没有bit，没有sbit 的概念的了。是不是就不如 51 了？

下载运行，还不行，因为GPIOB 的CLK 没有使能，这时其实 GPIOB 是不能工作的，这是 STM32 特殊的地方，上电

默认外设的时钟都是关的，初学者没有注意这里，是可以原谅的，多看看书，多实践，多问问就是了。

找到问题的原因，则再 RCC\_APB2ENR 设置，其中 BIT 3 就是 IOPBEN 是时钟使能位，同上，先找到 RCC\_APB2ENR

的地址

#define PERIPH\_BASE           ((u32)0x40000000)

#define AHBPERIPH\_BASE        (PERIPH\_BASE + 0x20000)

#define RCC\_BASE              (AHBPERIPH\_BASE + 0x1000)

RCC\_APB2ENR 的偏移是 18H ，所以最终得到地址为 0x40021018，操作方法同上

\*(volatile unsigned long \*)0x40021018 |= 1<<3;

最终的点LED的程序就完成了。

void main(void)

{

\*(volatile unsigned long \*)0x40021018 |= 1<<3;

\*(volatile unsigned long \*)0x40010C00 = (2<<20) | (0<<22);

\*(volatile unsigned long \*)0x40010C0C = 1<<5;

while (1)

{

}

}

如果将寄存器做一个定义，则程序变成如下

#define RCC\_APB2ENR \*(volatile unsigned long \*)0x40021018

#define GPIOB\_CRL  \*(volatile unsigned long \*)0x40010C00

#define GPIOB\_ODR \*(volatile unsigned long \*)0x40010C0C

void main(void)

{

RCC\_APB2ENR |= 1<<3;

GPIOB\_CRL = (2<<20) | (0<<22);

GPIOB\_ODR = 1<<5;

while (1)

{

}

}

RCC\_APB2ENR  RCC 是时钟寄存器 ， APB2 是外设2 ，ENR ，可以理解为 enable

GPIOB\_CRL  GPIO B control 控制寄存器

GPIOB\_ODR  GPIO(general purpose input output) B output data register 输出数据寄存器

都是有意义的名字，哪里难记了？？而且名字都来自 ST 的官方 datasheet、这个程序跟你用 51 写的程序我还真的

没看出差别有很大 .....

加入刚才的 GPIOB 寄存器，看看 ST 的官方库是怎么定义的，

\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\stm32f10x.h

用 UltraEdit 打开，搜索 GPIOB

#define PERIPH\_BASE           ((uint32\_t)0x40000000)

#define APB2PERIPH\_BASE       (PERIPH\_BASE + 0x10000)

#define GPIOB\_BASE            (APB2PERIPH\_BASE + 0x0C00)

没错，和keil 里面是一模一样的。

typedef struct

{

  \_\_IO uint32\_t CRL;

  \_\_IO uint32\_t CRH;

  \_\_IO uint32\_t IDR;

  \_\_IO uint32\_t ODR;

  \_\_IO uint32\_t BSRR;

  \_\_IO uint32\_t BRR;

  \_\_IO uint32\_t LCKR;

} GPIO\_TypeDef;

其中 \_\_IO  的定义在 \Libraries\CMSIS\CM3\CoreSupport\core\_cm3.h 为什么我知道在这个文件里面，因为我会

用 source insight ...

#define     \_\_IO    volatile

\_\_IO uint32\_t CRL 其实就是 volatile uint32\_t CRL

为什么用结构体？因为结构体的成员的地址分配（RAM中）是连续（不知道楼主是否懂得，这还是C语言的问题），

而 STM32 的一个模块的功能寄存器都是连续的，每个寄存器都是相当于 基地址加偏移，跟上面的理论一致

于是就有了结构体指针的用法

跟踪库函数的源代码，例如 GPIO 的 初始化函数

void GPIO\_Init(GPIO\_TypeDef\* GPIOx, GPIO\_InitTypeDef\* GPIO\_InitStruct)

以结构体指针的形式传递 IO 口 GPIO\_TypeDef\* GPIOx

访问 CRL 寄存器则用成员的形式 GPIOx->CRL;

不需要担心这样做的效率，因为都是地址，也就是指针，最终的效率是直接寄存器操作，效率是非常高的。

看不懂库函数，归根究底就是C语言功底不行。不要以为写过几行51就懂C语言了，远的很呢。

还有，STM 的库下载的时候包含了很多很多例子，库函数怎么使用在例子里面有很详细的介绍，不用写几行代码，

都是复制例子做实验，也很很容易的。

总结楼主的几个问题

1，ARM 没有SFR，也不需要，SFR 是51的关键字，没有理由 51 有 ARM 就要有。例如ACC，ARM 就没有，但是有

R0-R15 ，这些就是架构（architecture 的区别了）

2，STM32 的寄存器在官方头文件上面已经全部有定义了，上面已经阐述了。（你看不懂不代表没有吧？）

3，不带库函数的LED程序已经实现了。

想进步唯一的办法是多看书，多看代码，多写，多思考，少说话，楼主太浮躁了，反省一下吧。

Etual

2012-3-2

————————————————

版权声明：本文为CSDN博主「Etual」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/etual/article/details/8722120