**《PWM+TIM实现全彩灯》实验报告**

|  |  |  |
| --- | --- | --- |
| **课程名称**： **嵌入式系统设计** | **年级**：23级 | **上机实践成绩**： |
| **指导教师**：郭建 | **姓名**： **张建夫** |  |
| **上机实践名称**： **PWM+TIM实现全彩灯** | **学号：10235101477** | **上机实践日期**：2025/05/20 |
| **上机实践编号**： | **组号**： | **上机实践时间**： 14：50~16：30 |

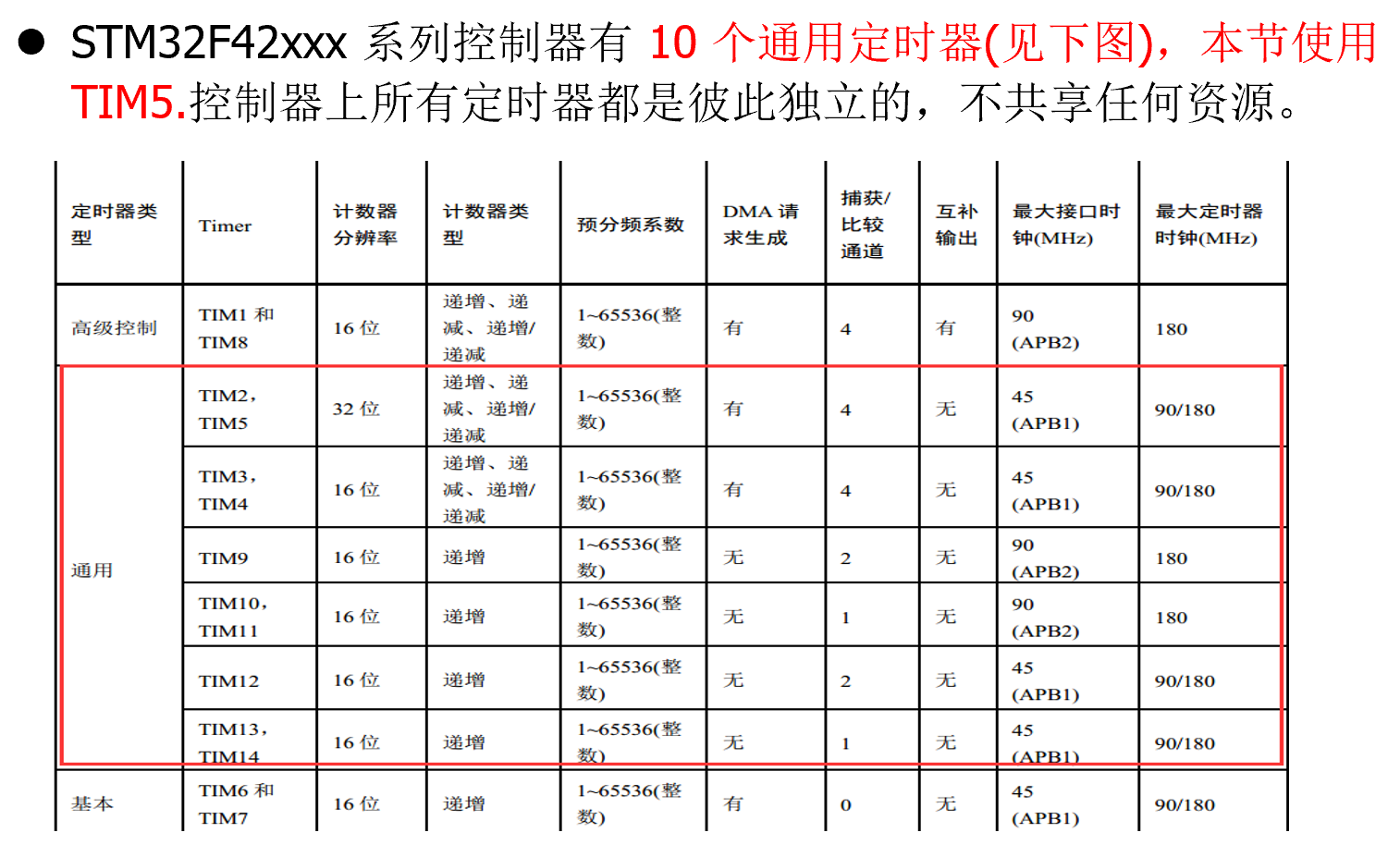
1. **目的与要求**

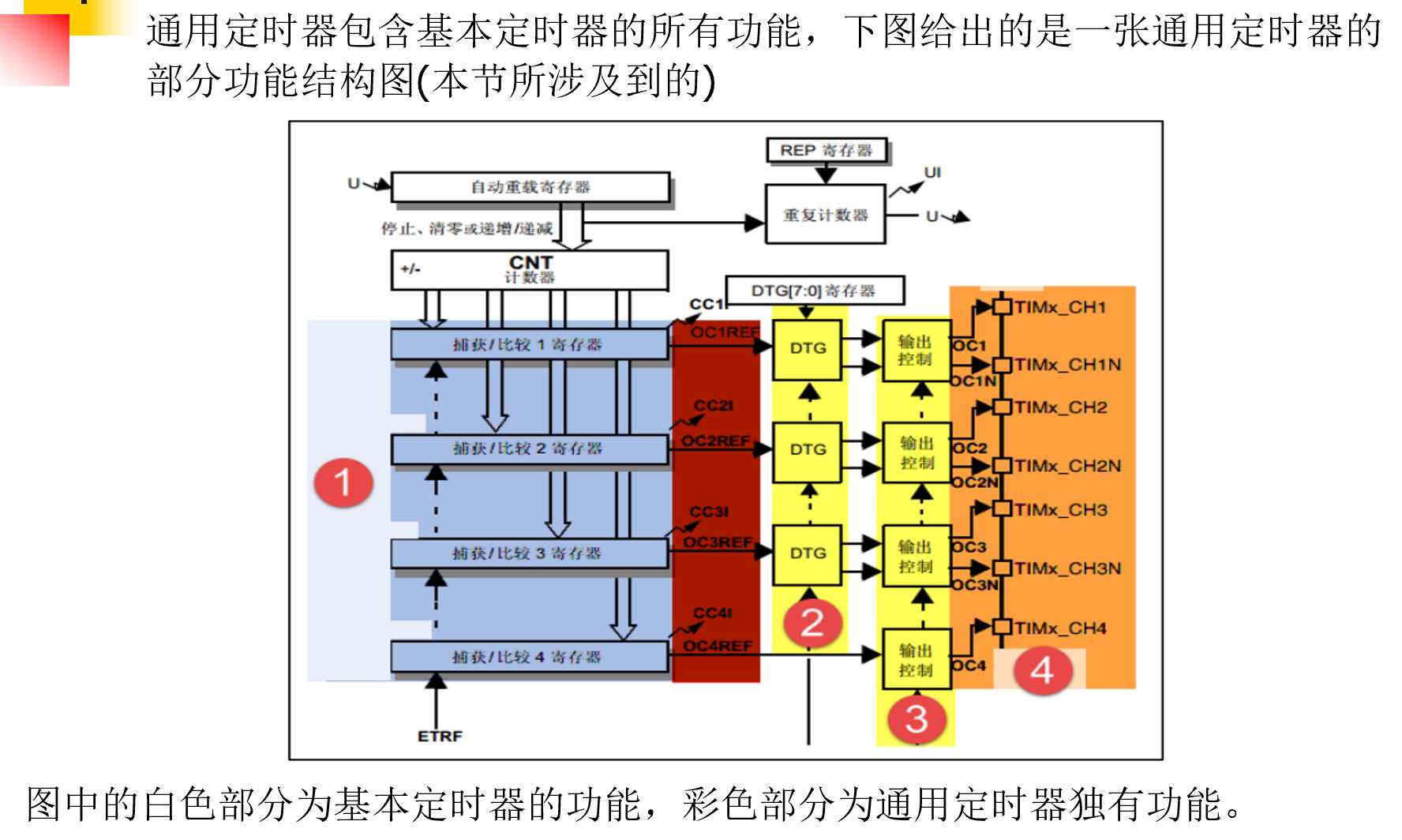
* 了解STM32通用定时器相关知识
* 学会使用通用定时器的PWM输出

1. **内容与实验原理**
2. **实验内容：**
   * + - 了解通用定时器的相关知识
       - 理解通用定时器的PWM输出功能
       - 学会使用PWM实现全彩灯

**2.实验原理：**

**（1）通用TIM介绍：**

****

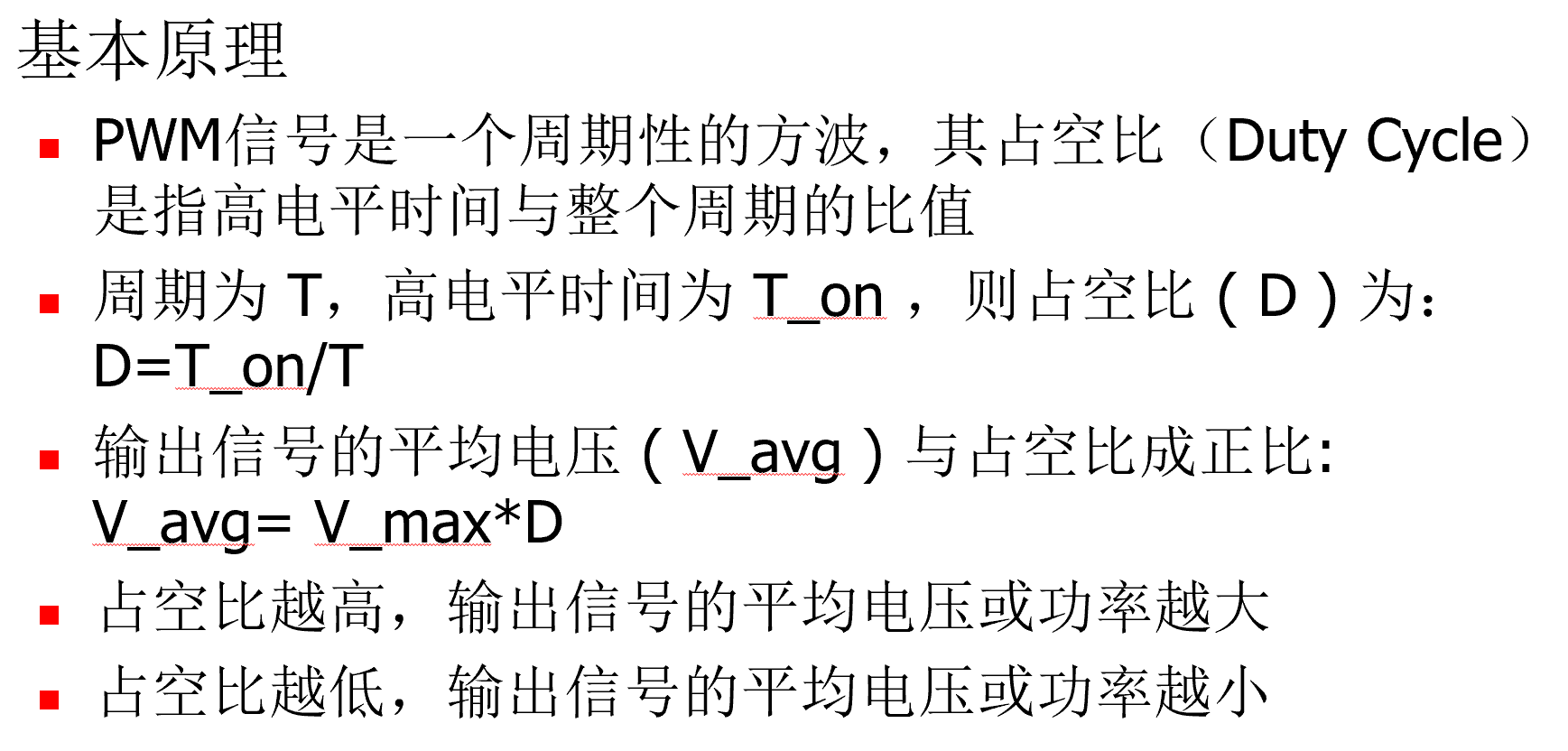
****

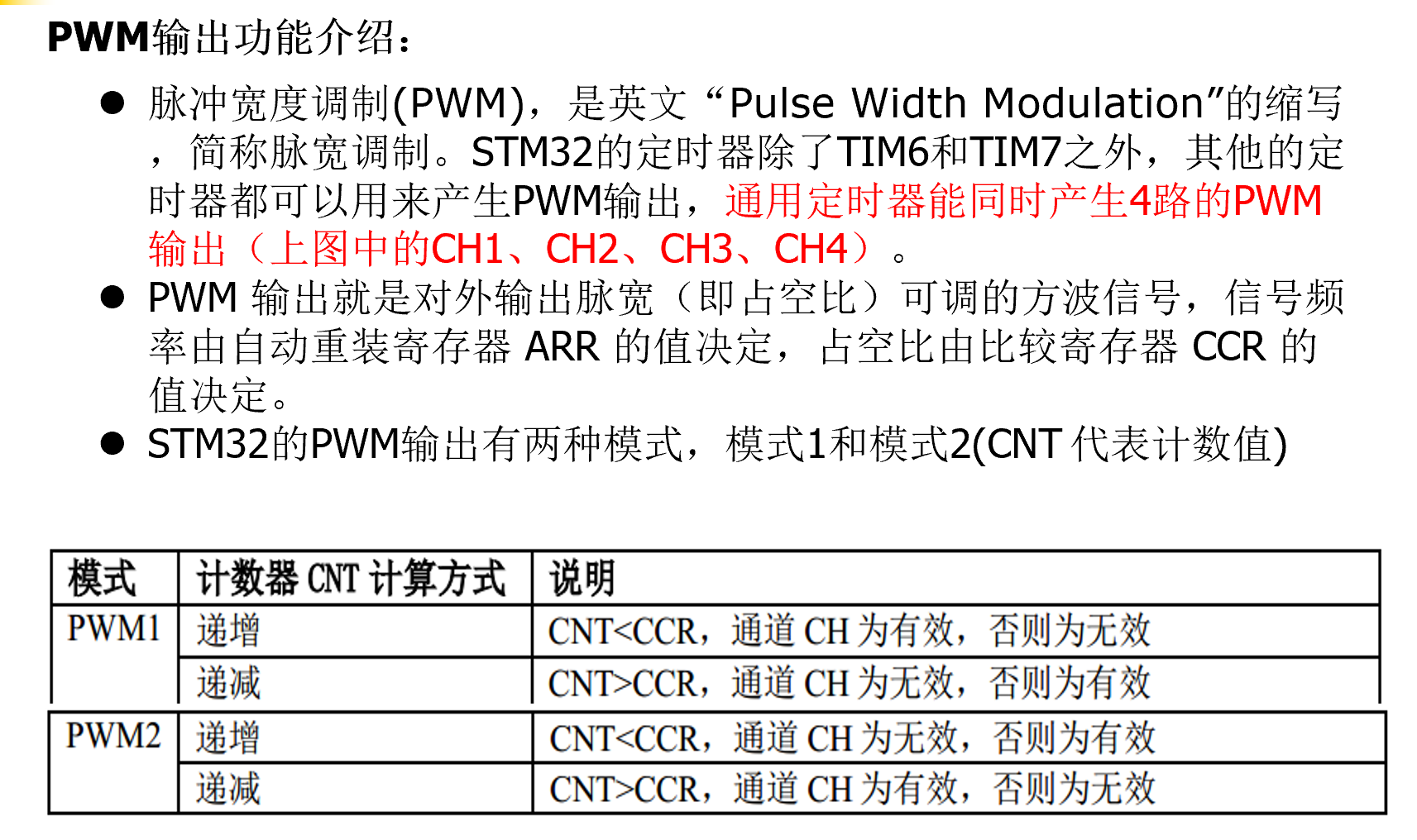
里面的tim原理是通过上图计数器 CNT 的值跟比较寄存器 CCR(捕获比较寄存器) 的值进行比较，相等的时候，输出参考信号 OCxREF 的信号的极性就会改变，其中 OCxREF=1（高电平）称之为有效电平， OCxREF=0（低电平）称之为无效电平，并且会产生比较中断 CCxI，相应的标志位 CCxIF（ SR 寄存器中）会置位。**然后 OCxREF 再经过一系列的控制之后就成为真正的输出信号 OCx/OCxN。**

而这些输出信号最终是通过定时器的外部io进行输出，分别为 CH1/2/3/4。

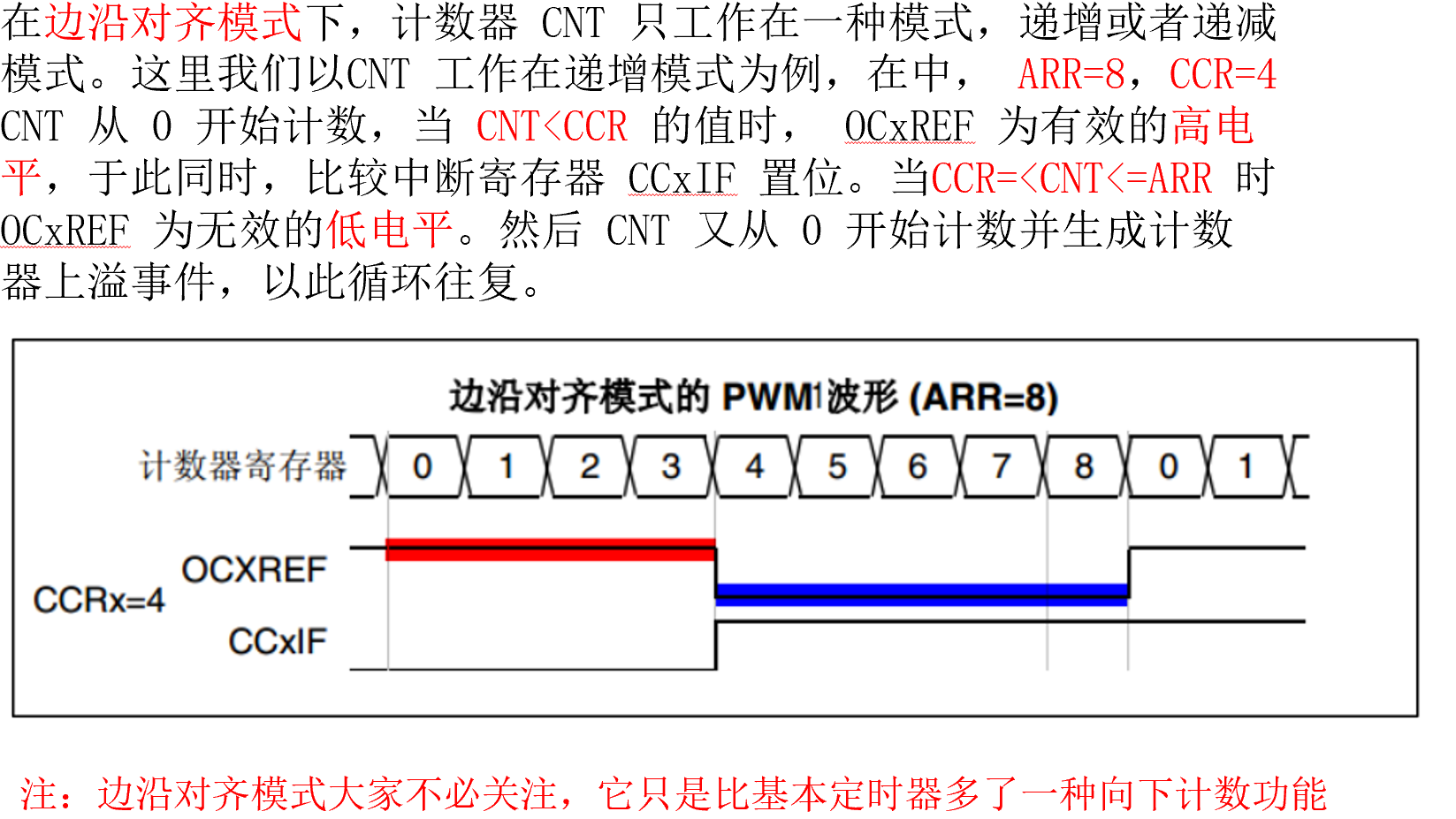
**（2）PWM介绍：**

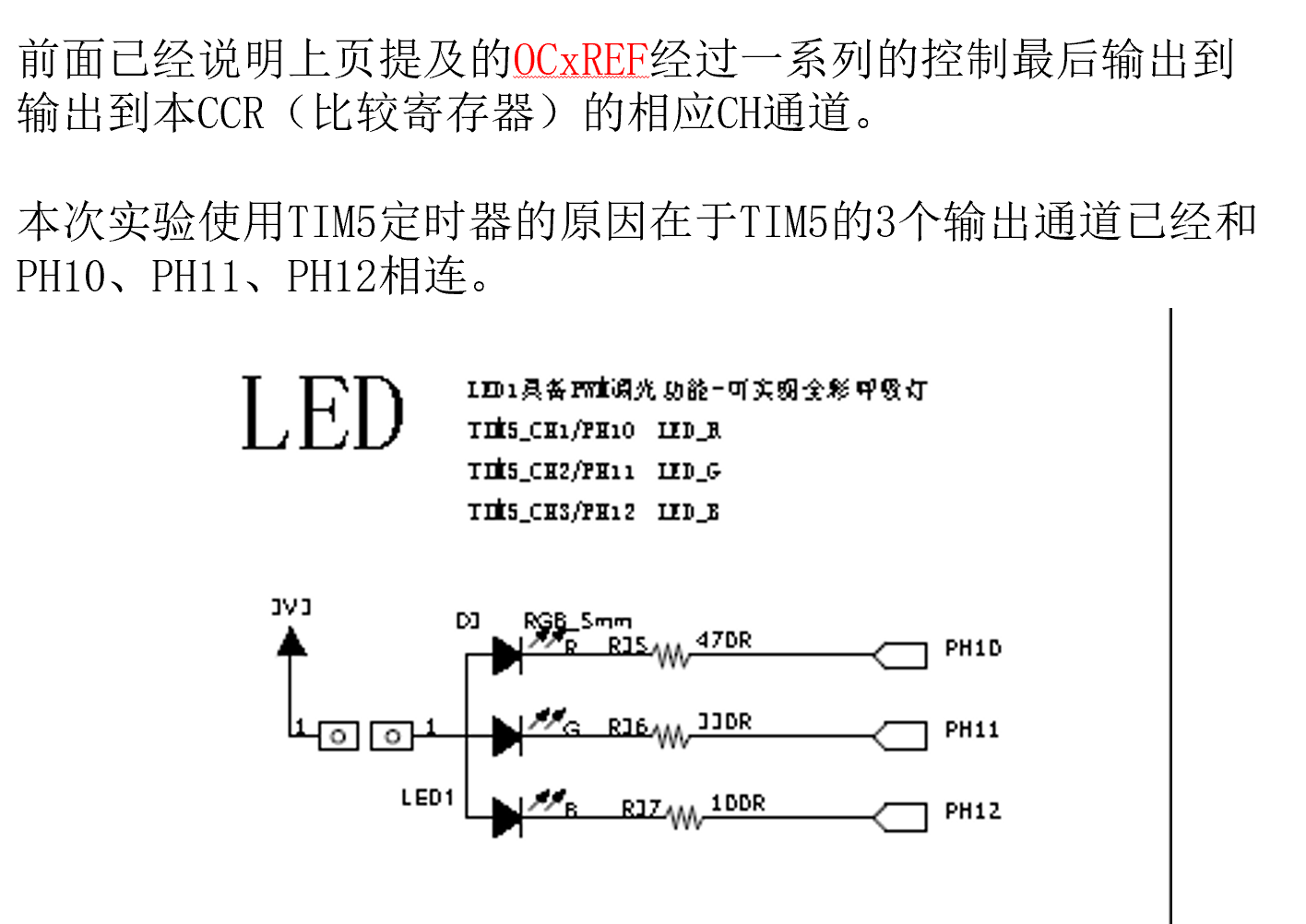
PWM指的是通过调节脉冲信号的占空比来控制输出信号平均电压或功率的技术。



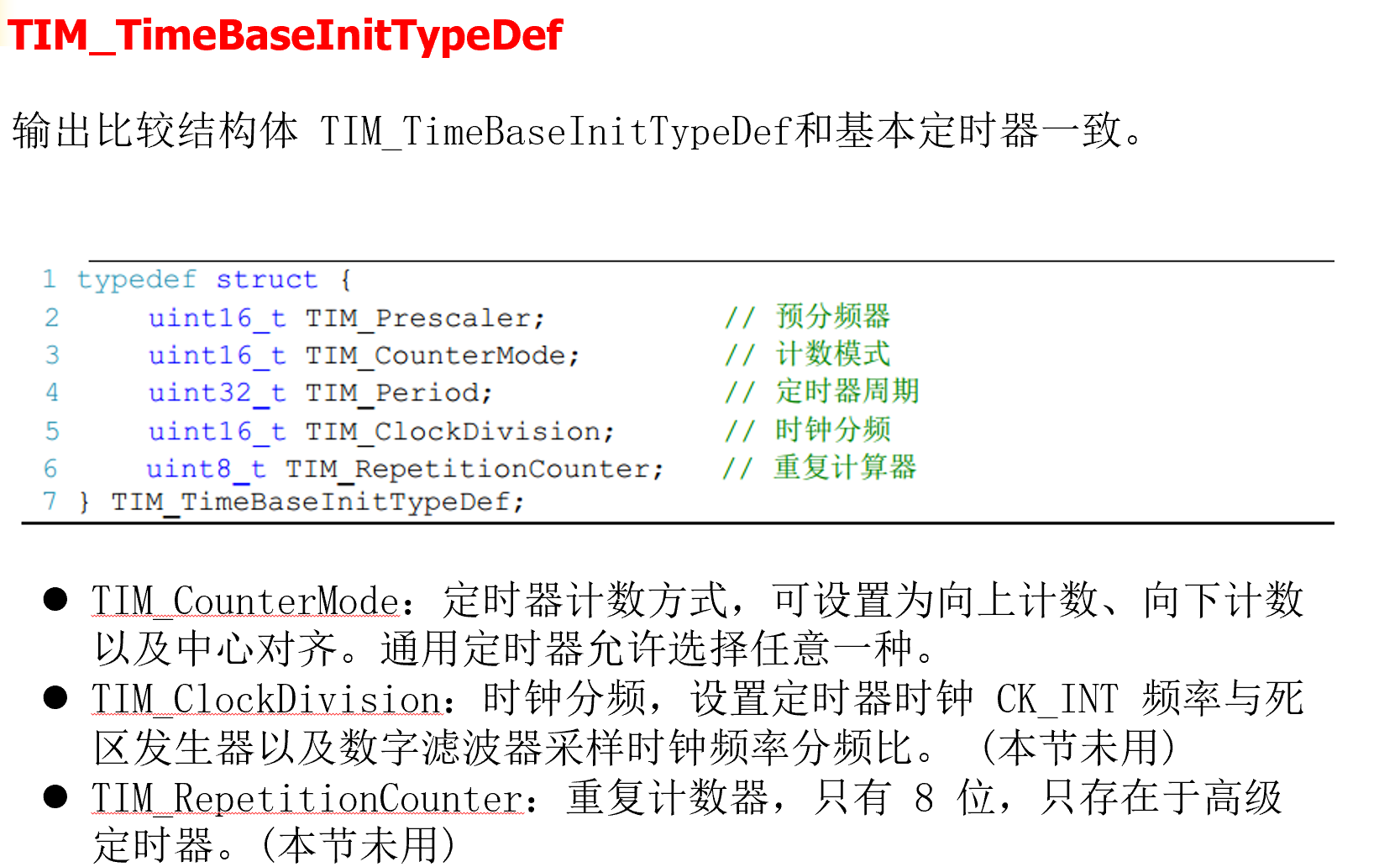


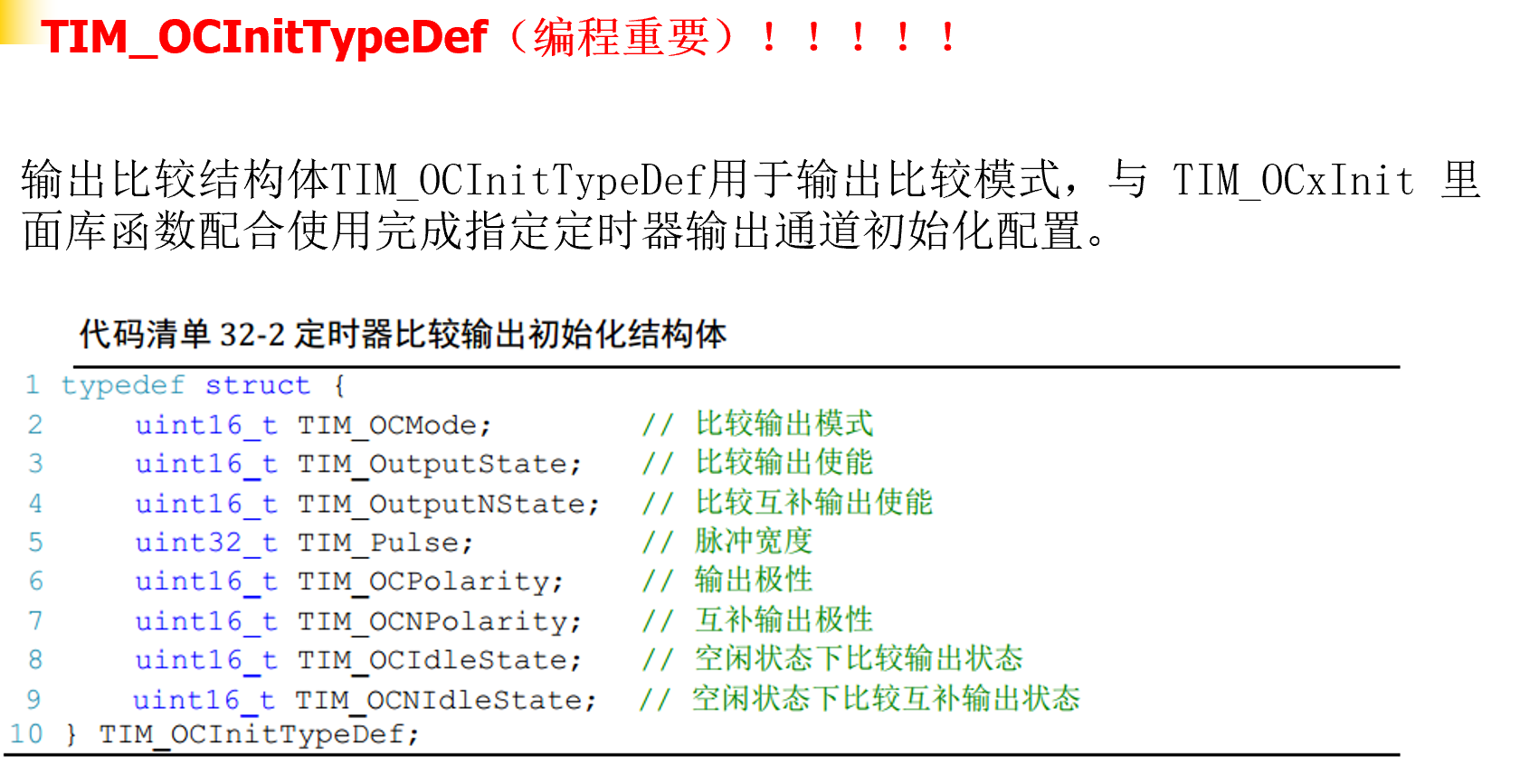
本次实验选用的是PWM1，其输出的工作原理如下：

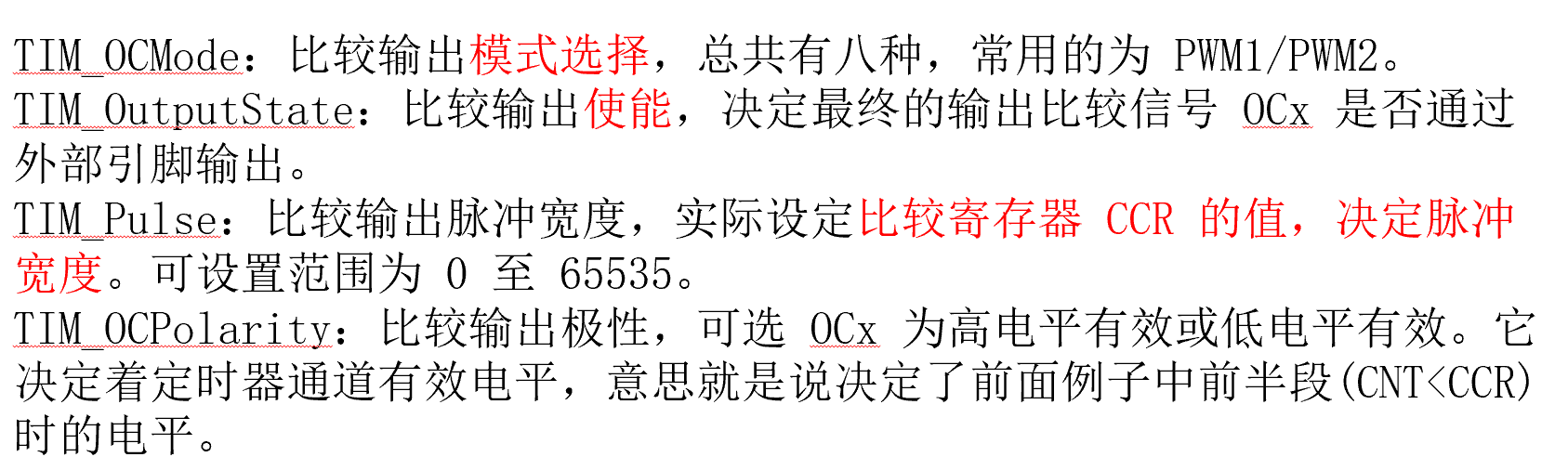




**（3）TIM编程介绍：**







1. **使用环境**

调用dxdiag工具：

Operating System: Windows 11 家庭中文版 64-bit (10.0, Build 22621) (22621.ni\_release.220506-1250)

Language: Chinese (Simplified) (Regional Setting: Chinese (Simplified))

System Manufacturer: HP

System Model: HP Pavilion Aero Laptop 13-be2xxx

BIOS: F.13 (type: UEFI)

Processor: AMD Ryzen 5 7535U with Radeon Graphics (12 CPUs), ~2.9GHz

Memory: 16384MB RAM

Available OS Memory: 15574MB RAM

Page File: 27604MB used, 5685MB available

Windows Dir: C:\WINDOWS

DirectX Version: DirectX 12

DX Setup Parameters: Not found

User DPI Setting: 144 DPI (150 percent)

System DPI Setting: 192 DPI (200 percent)

DWM DPI Scaling: UnKnown

Miracast: Available, with HDCP

Microsoft Graphics Hybrid: Not Supported

1. **主要实验内容和结果展示**

本次实验使用的是之前提供的模板，需要将bsp\_key的头文件和源文件均改成bsp\_color\_led的文件内容。另外，对于要求的拓展实验，还需要增加基本定时器的初始化相关头文件和源文件。

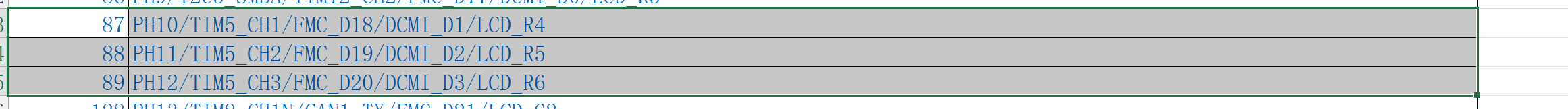
1. **示例实验：**

按照ppt上的教程，先要在头文件（bsp\_color\_led.h）对串口的变量进行映射（宏定义）：



该处是对各个灯引脚的定义，这里使用的三个的引脚分别是10，11，12号引脚，GPIO端口编号为H，这里使用这三个引脚的原因有两点：

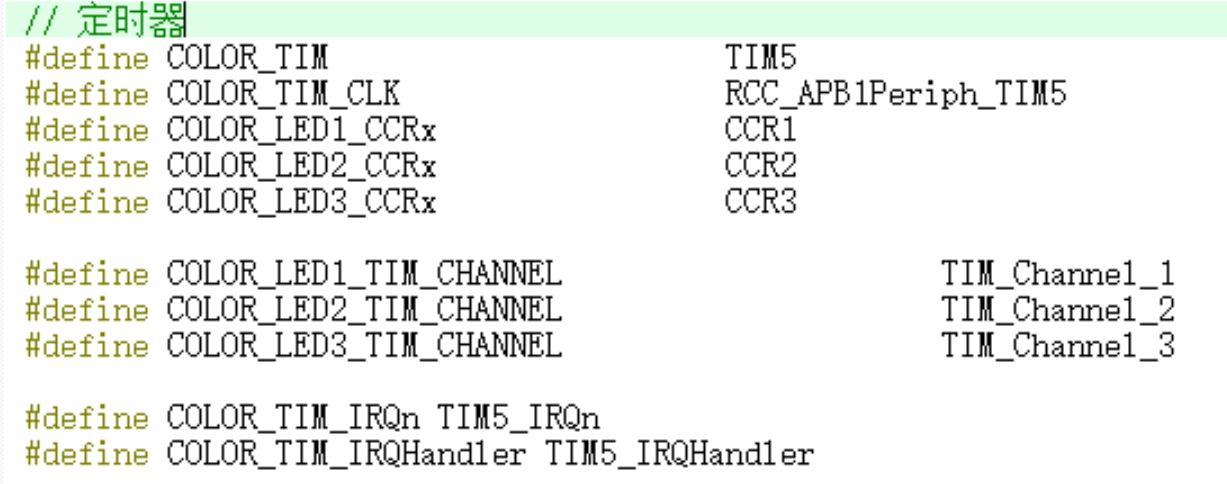
* 这三个引脚本身就对应着红绿蓝三个颜色的灯
* Tim5引脚的三个输出通道已经和PH10、PH11、PH12相连。在老师提供的“STM32F429IGT6管脚汇总（含第二功能）”文件中，可以看到：



这三个端口含有通用计时器tim5分别的三个输出通道

仔细观察可以发现，这几个宏的定义仅仅只有下面的pinsourse（用于配置引脚复用功能）和af（指定对应引脚的复用功能为tim5）与之前gpio的宏定义不一样，其他的端口和引脚实际上是一样的。

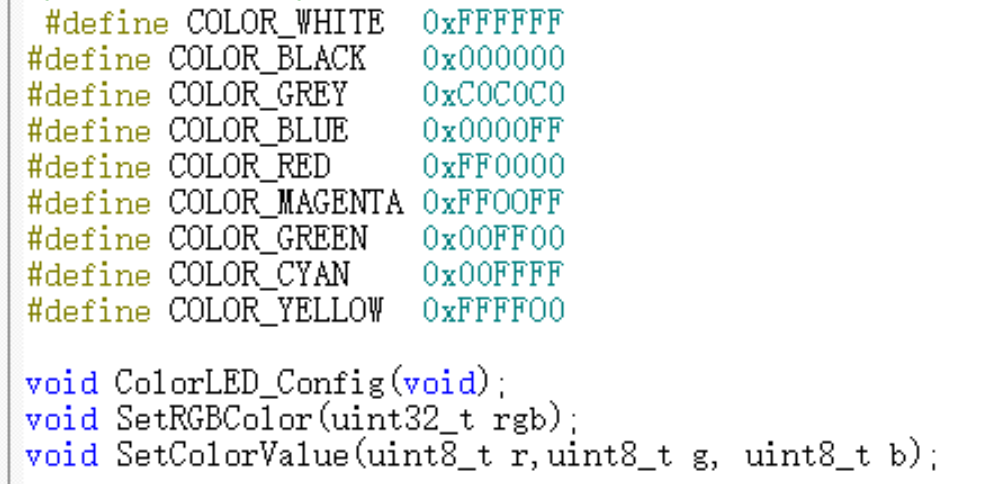
接下来定义定时器相关的宏：



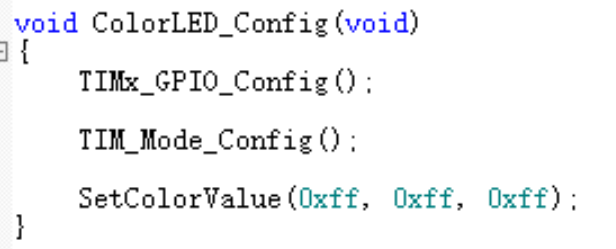
这一串代码的意思是指定对应的定时器tim5，并设置其时钟域为RCC\_APB1Periph\_TIM5，接着定义tim的三个通道1，2，3的比较寄存器CCR，后面改变这些值就可以改变相应的占空比。

最后两行代码在本次实验中是多余的，我看了整个ppt没有发现使用这两个中断定义的地方，应该在stm32f4xx\_it.c中书写相应的中断服务函数才能实现具体的功能。

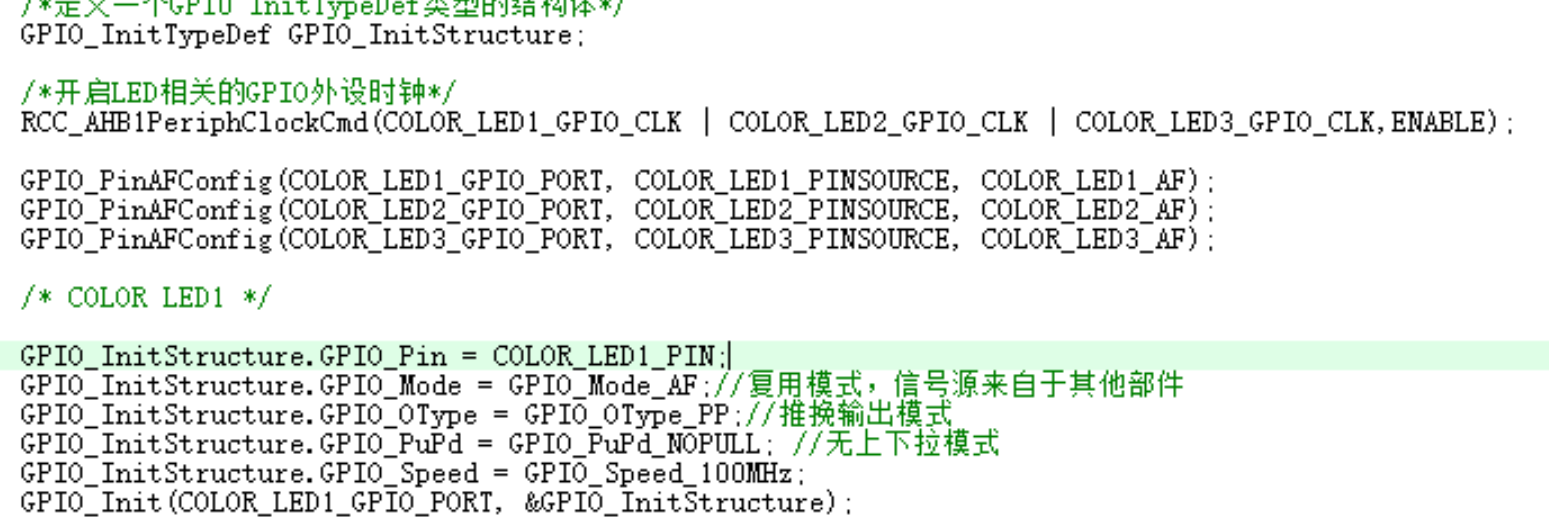
接着定义rgb颜色值并进行函数声明：



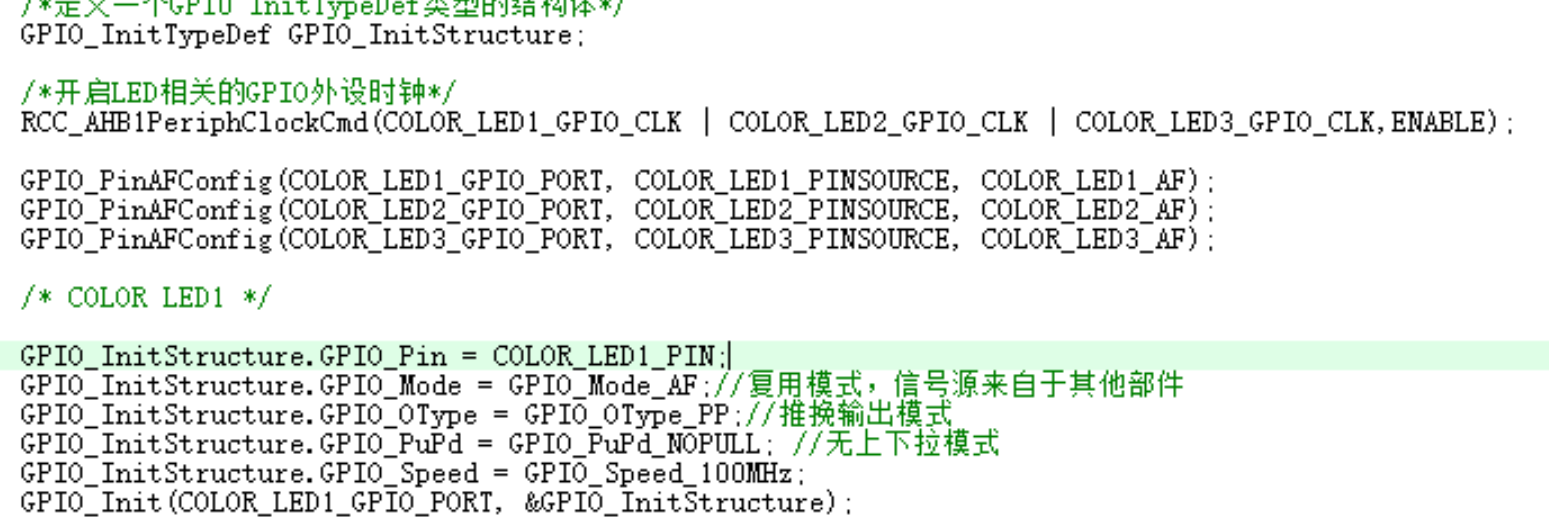
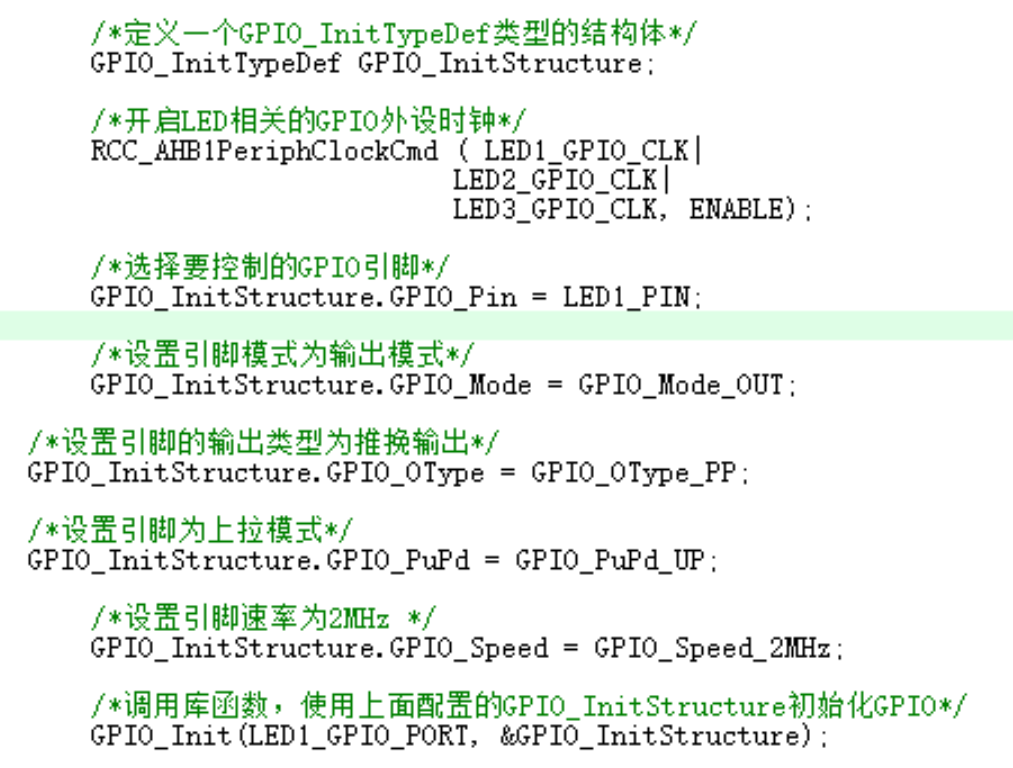
然后围绕这三个函数分别讲解，第一个是ColorLED\_Config：



这个函数用于tim5的初始化配置，并将初始颜色设为白色（setcolorvalue函数），接下来我们具体看timx\_gpio\_config的设置：



这个过程和实验三里面的gpio初始化过程大致一样，我们可以将两个初始化的过程做一个对比：

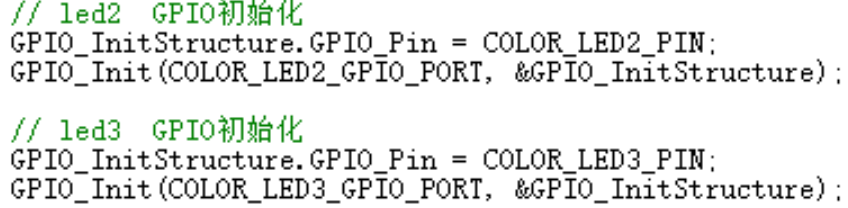


（左侧为实验3中的gpio初始化，右侧是本次实验的初始化）

可以看到引脚模式由输出模式变为复用模式，引脚改为无上下拉的模式（实际上这个不影响），速率由2MHz变为100MHz

除此之外，因为要进行引脚复用，这里还多出来了调用gpio\_pinafconfig函数进行引脚和tim5的绑定.

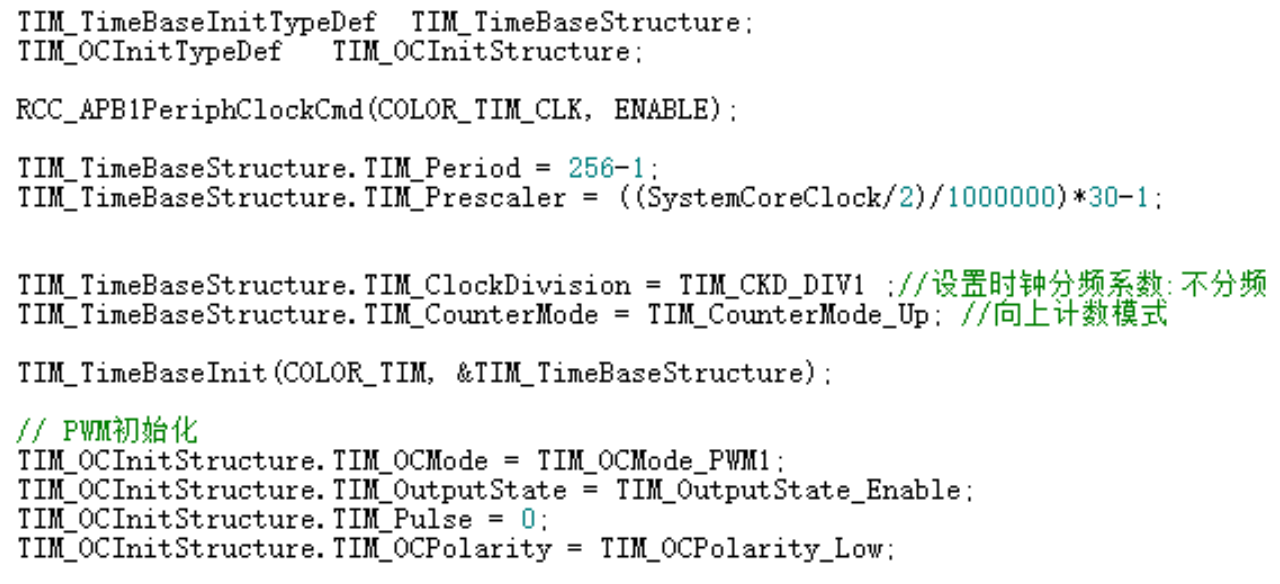
通过使用实验3中gpio初始化多个引脚的方法，对本次实验的绿灯和蓝灯也进行相应的初始化：



下一个初始化函数是TIM\_Mode\_Config，这个函数初始化了两个东西：

* 定时器的基本参数
* PWM初始化

先看定时器的初始化：



首先使能了本次实验的定时器的时钟，并对设置计数器的溢出值为255（由于计数是从0开始的，实际上会计数256次才结束一个计时周期）

接下来设置预分频，这里我在看代码的时候有些疑问，这个预分频的计算为什么这么复杂，它到底在算什么，而且还使用了SystemCoreClock，后面询问了大模型，大模型给出的答案是系统主频（即SystemCoreClock）过高，需要“提前降频”，这里的预分频器就是为了这个目的，大模型给出了一个计算公式：

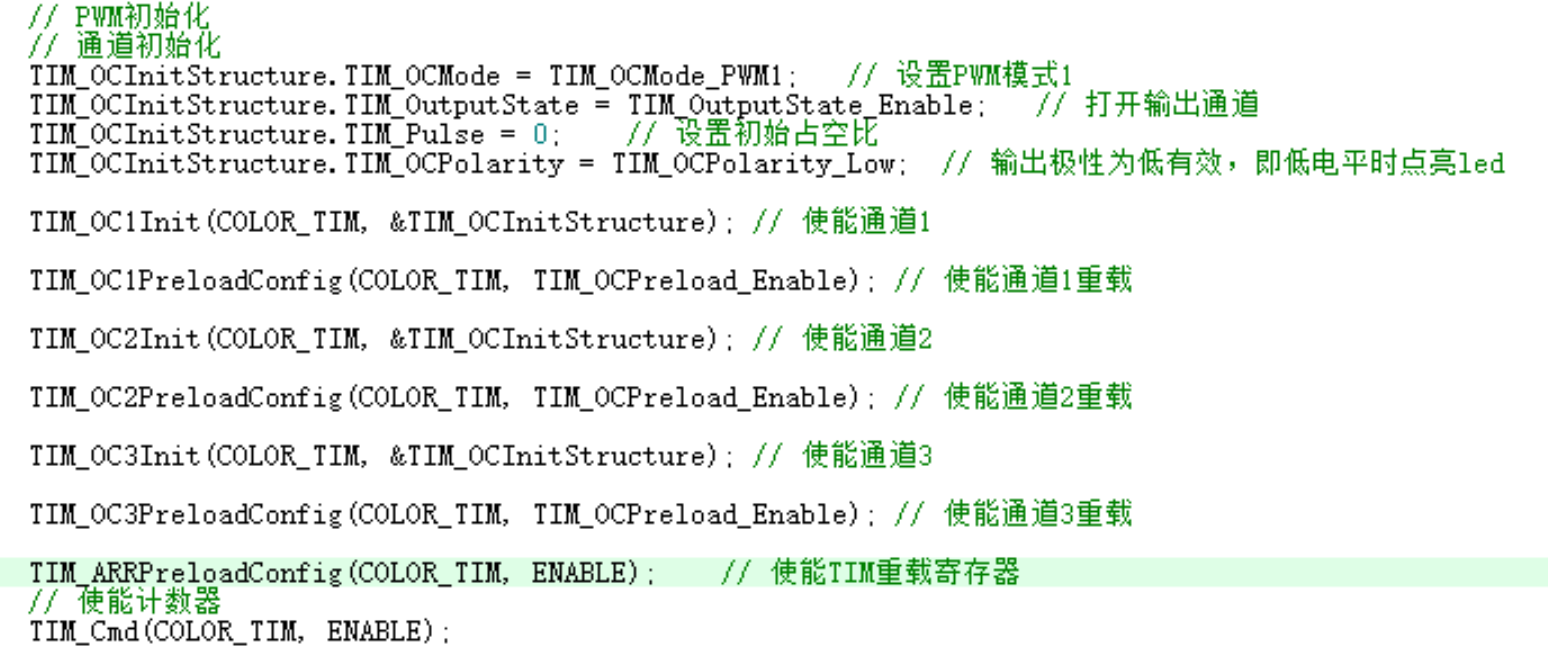


***计数频率 = 84MHz/（Prescaler的值 + 1）***

这里就解释了为什么使用SystemCoreClock，SystemCoreClock的值为168MHz，其除以2正好是84MHz，可以和公式分子的84MHz（实际上是该寄存器的时钟源频率）抵消，接着除以1百万乘以30，就相当于每次计数的时长为30 \* 10^-6 ，也就是30微秒

最后就是设置定时器时钟 CK\_INT 频率与死区发生器以及数字滤波器采样时钟频率分频比以及计数模式（向上计数）。

再看PWM的初始化：



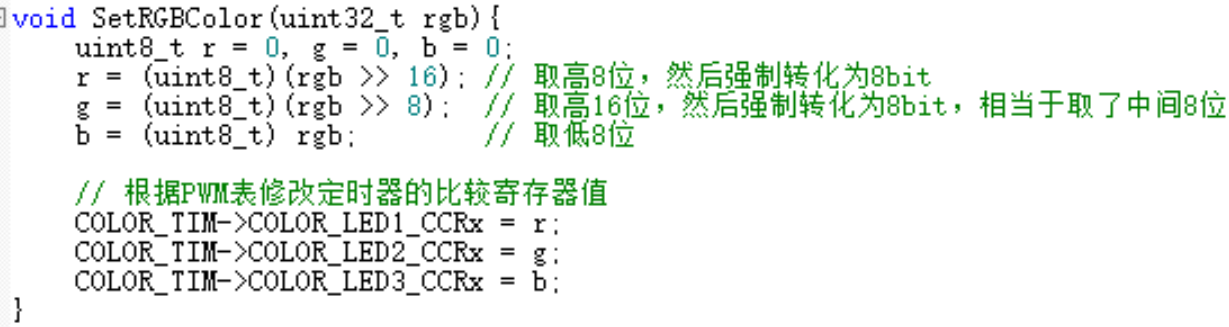
首先按照ppt上的要求设置PWM为模式1，并将输出通道打开（第二行的enable），接下来设置占空比为0%，即led初始为熄灭状态，并设置输出极性为低电平有效，最后就是使能通道并进行预装载（这里老师是用重载表示预装载，作为c++程序员感觉实在是不严谨，c里面没有重载机制，容易让人误解）

这里的预装载机制我专门查了一下，发现这种机制是基于一个影子寄存器的概念实现的，由于在用户更改灯的亮度（即更改占空比）时，有两个更改的时间，第一个是立即更改，另一个是本次PWM周期结束时更改，而预装载采用的是后者，因此，需要有一个寄存器将预装载的值存好，直到周期结束时才写入CCR比较寄存器，存储这个值的寄存器称为影子寄存器，每一个通道都一个影子寄存器。

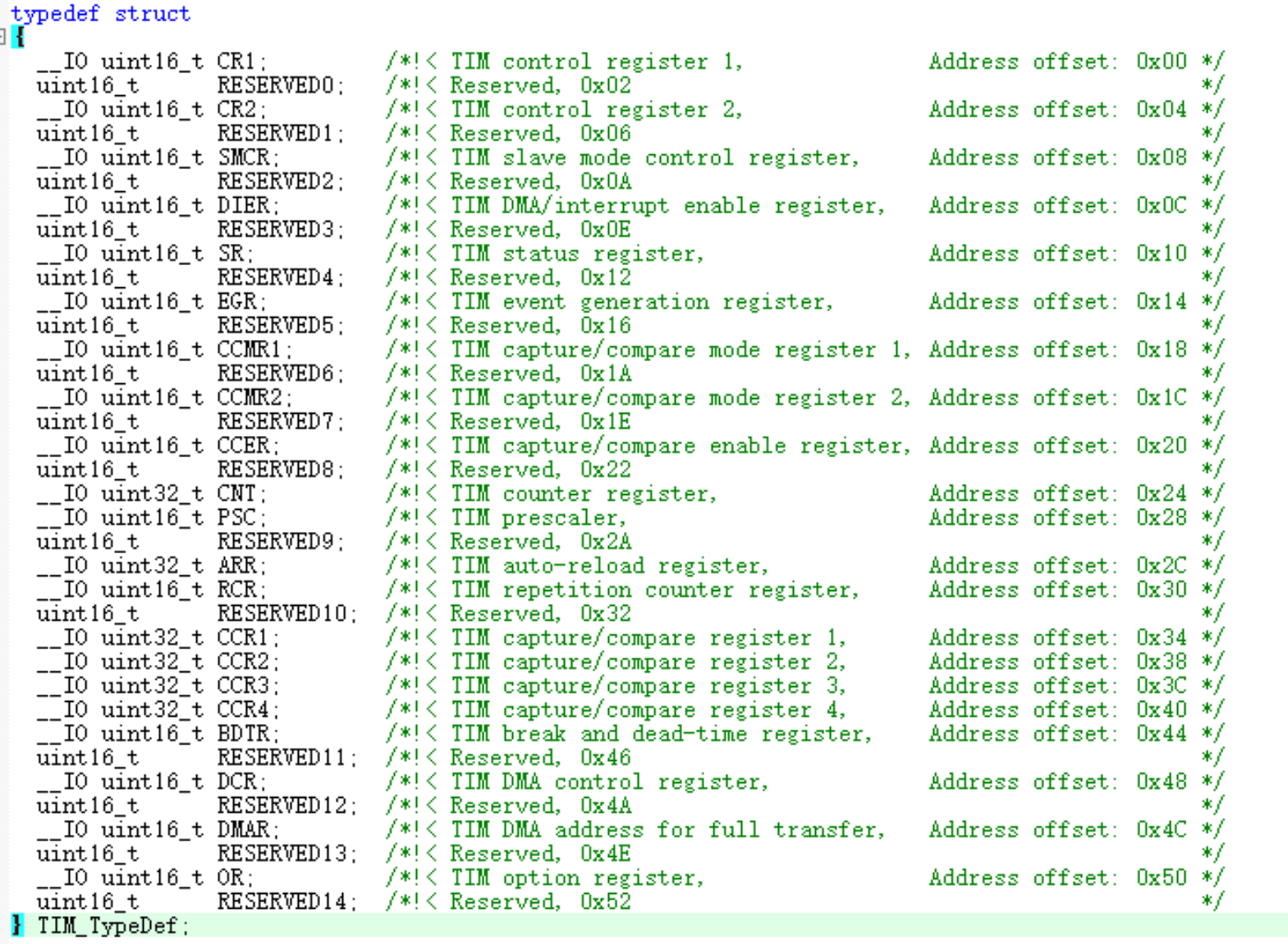
然后使能预装载寄存器，即开启影子寄存器功能。

最后的最后，需要使能计数器，不然颜色不会更改。

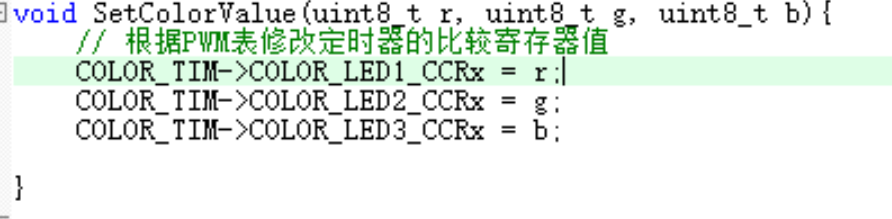
接下来看第二个函数：SetRGBColor



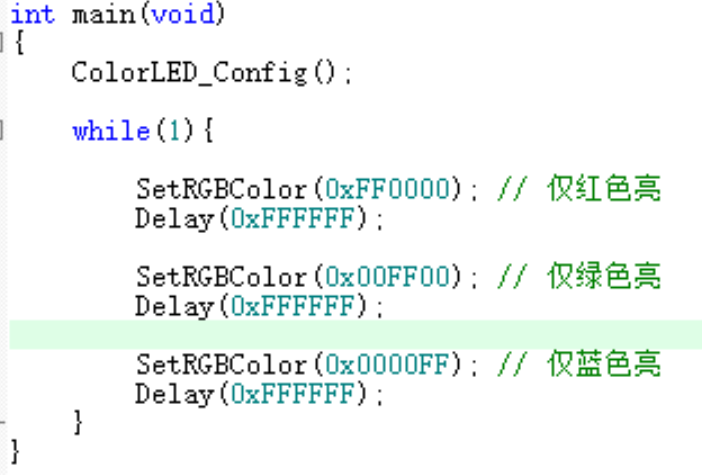
这个函数是传入一个4字节的数（实际上只有低24位有效），并取出相应位置的值作为对应颜色的占空比，例如高8位表示红色（rgb右移16位那一行），并用PWM表（实际上是一个结构体，如下图所示）找到对应的CCR比较寄存器，之后赋值。



最后一个函数是SetColorValue，直接分别传入三个字节，表示红绿蓝，其实就是上一个函数少了分解参数的过程：



然后看主函数的实现：



主函数的实现比较简单，在设置完灯的占空比后，使用一个延时函数保证占空比在该时间段内不会被修改即可。

演示视频见提交文件。

1. **参考所给项目，自行用PWM实现三个通道(从1-3)的占空比依次为自己学号后三个字节(从高-低)，观察LED的变化。(如学号为10156160233，则后三个字节为0x160233)**

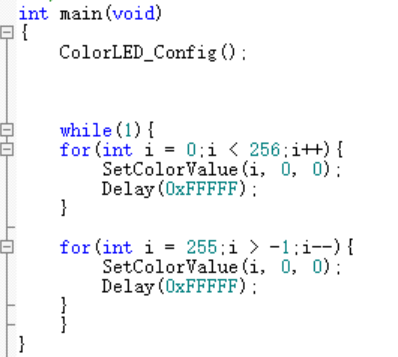
这个要求比较特别，每个人都有自己的学号，意味着呈现出来的亮度都不相同，我的学号后6位为0x101477，故将示例实验中对应红灯的亮度改为0x10，绿灯的亮度改为0x14，蓝灯的亮度改为0x77即可：



演示视频见提交文件。

1. **参考实验项目，完成红灯的呼吸灯，即使红灯由暗到亮，然后再回到暗，一直反复。（提示：修改红灯的占空比，也就是修改红灯对应的CCR寄存器的值）**

如提示所言，由于占空比决定了灯的亮度，实现呼吸灯的思路就是通过循环改变占空比的值，另外，由于占空比和灯的亮度是线性相关的，我们可以直接使用迭代时的计数器表示占空比（相当于哈希了），最后，题目要求红灯由暗到亮，再由亮到暗，因此一个周期内包含两个循环，一个循环递增，一个循环递减：



需要注意的是，0xFF = 255，这个值是占空比的最大值，因此迭代的计数器最大值是255，同时为了使灯的亮度变化明显，我将delay的时间改为了0xFFFFF。

演示视频见提交文件。

1. **把示例中的延时函数改用基本定时器定为1s。**

该实验的原理和之前那次拓展实验的原理一样，但是我在实现的过程却因为一个编译器的特性而一直实验失败，不过后面发现原因后实验也顺利成功，此处便记录相关尝试。

为实现延时函数为固定的一秒，而且又要使用之前的基本定时器设置时间，此处再次解释一遍基本定时器计算时间的方式，由于基本定时器的时钟源频率为90MHz（之前讲过通用寄存器的时钟源频率为84MHz，那个分子的值），首先定时器会根据这个时钟频率进行预分频，此处我才用的预分频就是按照ppt内的预分频：

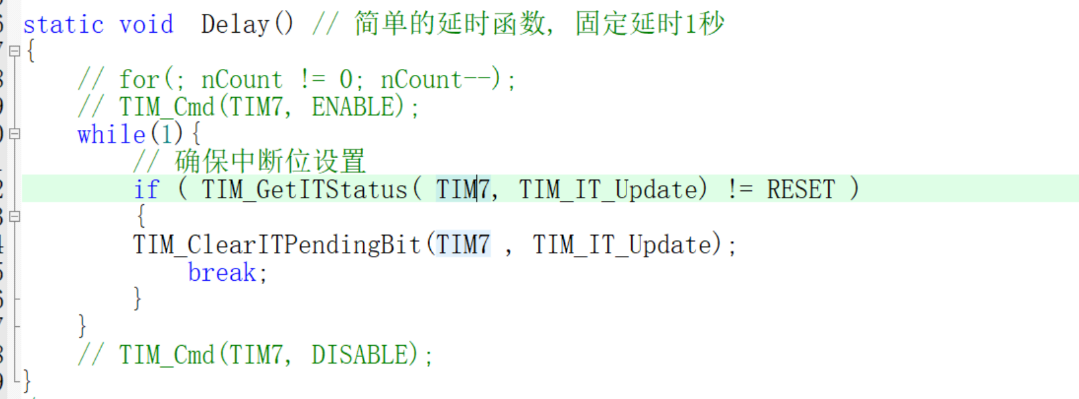
此处减1的原因不再赘述，9000\*1/90M，得到100微秒，也就是说，预分频后的tick+1的时间是100微秒，由于ppt的要求是1秒，100微秒 \* 10000就是1秒，因此分频改为10000 – 1即可：



其他的初始化过程和之前的扩展实验相同，此处不再赘述

接下来是延时函数的改造：

由于每个计时周期结束时只会跳转到中断函数来处理本周期，因此，我的第一个思路是把中断函数内的代码通过某种形式移到Delay函数中，其中我发现之前的中断函数有一个函数用于判断是不是真的产生了中断，便进行了如下改变：

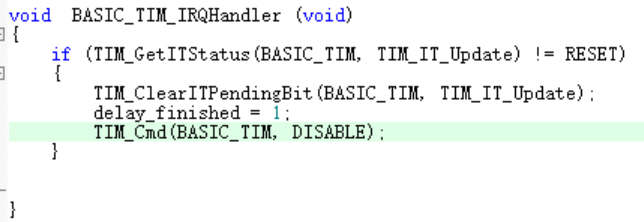


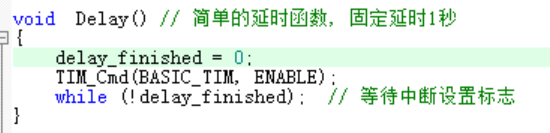
（上下各有一个TIM\_Cmd的原因是后面自己1.0版本没做对后加上的，我查了资料发现是通过这两个语句控制计数器的启用与停止，不过加上了也不对就是了）

但是这是无效的，而且相当危险，这个函数本身是***验证***中断是否产生，而不是***判断***中断是否产生，第一次实验失败。

后面我又想了一个办法，是通过一个额外的全局变量（定义在stm32f4xx\_it.c里面），当该变量为0时，主线程在delay函数中就会阻塞，只有在中断发生后，delay被置为1，回到主函数，才会停止delay函数的执行，下面是变量定义，中断函数的实现以及delay函数的实现：







（Delay\_finished定义在stm32f4xx\_it.c中，Delay函数定义在主文件中）

具体实现逻辑如下：

首先进入delay函数，将delay\_finished变量设为0，然后激活定时器，此时定时器开始计时，下面的while循环则是实现忙等。

接着等待时间结束触发中断，在中断函数内，delay\_finished被设置为1，并在同时关闭定时器。

回到Delay函数，while循环下一次判断delay\_finished的值时就会跳出循环，这样就成功设置了延时时间为1秒的延时函数。

这个2.0版本的实现逻辑上是没有问题的，问题出在delay\_finished没有加上阻止编译器将其优化的volatile关键字，这会导致编译器将while循环优化成一个死循环（delay\_finished的逻辑取反可以提前算出来，为1，而且编译器不知道我在中断函数内将其取反了），导致实验失败。

这个问题的解释相当简单但是找到问题的过程却相当困难，由于最开始并没有编译器优化而导致实验失败的例子，对编译器导致的问题是相当不敏感，一直以为是实现逻辑或者是理解不够透彻，将实验的代码改了很多次，甚至将extern的解释以及可能出现的问题看了很多次（因为我在不同文件里引用了同一个变量delay\_finished），最后甚至将Delay函数写到了stm32f4xx\_it.c文件中避免使用extern，但是很显然，这些方式都失败了，最后询问了同学，他们指出可能是编译器优化的问题才解决。

演示视频见提交文件。

**五、实验总结**

本次实验我收获颇多，尤其是工程方面的经验。对于ppt上的基础实验部分，我初步了解了PWM的原理以及其和通用定时器的结合使用，并对定时器的预分频机制有了更加深入的了解。在扩展实验中，我第一次遇到了因编译器优化产生的问题，从某种意义上说，这是一次特别好的实验经历，这为我之后进行嵌入式开发提供了非常宝贵的工程经验，我也真实体验到了遇到问题却查资料查不出来的情况，整个过程虽然相当崩溃，结果还是实验成功了，也知道了失败的原因。