定时器：LED闪烁实验

定时器和中断时嵌入式学习过程中，两个非常重要的概念。本实验还是对协处理器STM32F100进行操作。

1. **定时器说明**

STM32F1 系列中，除了互联型的产品，共有8个定时器，分为基本定时器，通用定时器和高级定时器。基本定时器 TIM6 和 TIM7 是一个 16 位的只能向上计数的定时器，只能定时，没有外部 IO。通用定时器 TIM2/3/4/5 是一个 16 位的可以向上/下计数的定时器，可以定时，可以输出比较，可以输入捕捉，每个定时器有四个外部 IO。高级定时器 TIM1/8是一个 16 位的可以向上/下计数的定时器，可以定时，可以输出比较，可以输入捕捉，还可以有三相电机互补输出信号，每个定时器有 8 个外部 IO。

1. **Stm32F100定时器的时钟来源**

参考STM32F100芯片数据手册的第13页，可以找到，每个定时器的时钟来源，如图1. 所示。

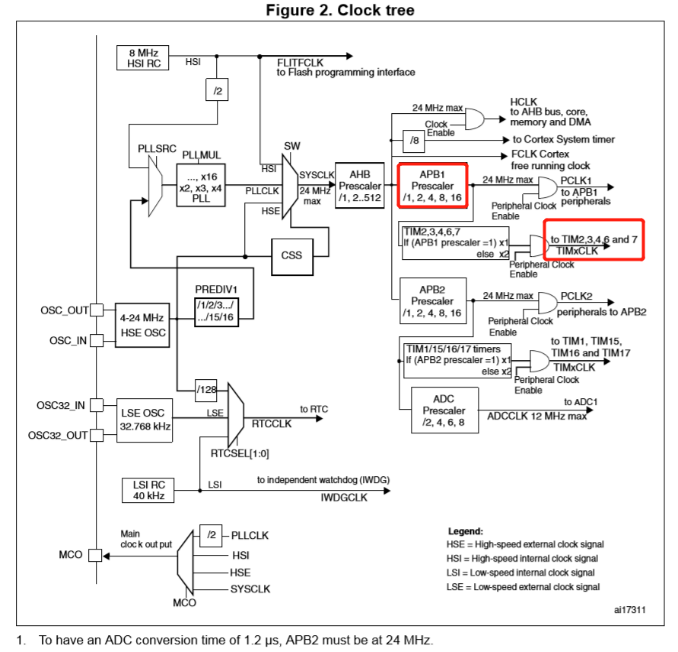


图1. 定时器相关的时钟来源

我们主要关注TIM2定时器，由图1. 可以知道，系统时钟（SYSCLK）经过AHB分频器给APB1外设。现在需要查看我们的时钟系统配置，来查看一下APB1外设的频率，如图2.所示。

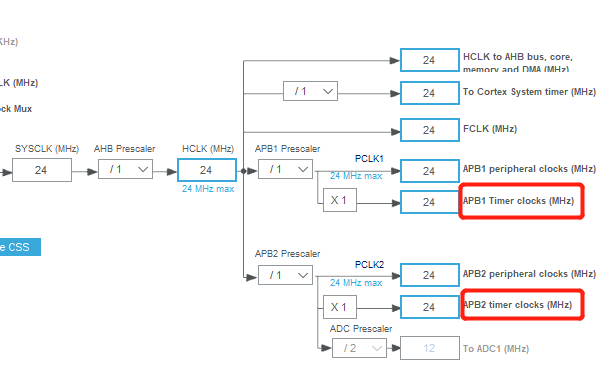


图2. APB1的时钟频率

查看时钟系统配置，APB1外设的频率为24MHz，故而时钟系统分配给定时器TIM2的频率为24MHz。

1. **设置定时器**

设置定时器主要是设置定时器的分频，以及定时器的定时值。下面通过解释定时器相关的额名词，来学习如何完成定时器的设置。

1）时钟源

定时器时钟TIMxCLK（x为定时器的需要数字）即内部时钟CK\_INT，经APB1预分频器后分频提供，如果APB1 预分频系数等于 1，则频率不变，否则频率乘以 2，在我们的实验中APB1的分频系数为1，故而定时器时钟TIMxCLK=24MHz\*1=24MHz

2）计数器时钟

定时器时钟经过 PSC 预分频器之后，即CK\_CNT，用来驱动计数器计数。PSC 是一个16 位的预分频器，可以对定时器时钟 TIMxCLK 进行 1~65536 之间的任何一个数进行分频。可理解为我们定时器时钟进行分频，指定分频值为PSC，就将我们的TIMxCLK分了PSC+1，定时器的最终频率即为：

CK\_CNT=TIMxCLK/（PSC+1）。

这里的频率的意思就是1s中记 TIMxCLK/（PSC+1）兆个数 (1M=10的6次方) ,每记一个数的时间为(PSC+1)/ TIMxCLK

**这里的PSC值需要我们自行设计。**

3）计数器

计数器 CNT 是一个 16 位的计数器，只能往上计数，最大计数值为 65535。当计数达到自动重装载寄存器的时候产生更新事件，并清零从头开始计数。

4）自动重装载寄存器

自动重装载寄存器 ARR 是一个 16 位的寄存器，这里面装着计数器能计数的最大数值。当计数到这个值的时候，如果使能了中断的话，定时器就产生溢出中断。

**这里的寄存器ARR的装载值，需要我们自行设计。定义这个数值为arr。**

5）定时时间的计算

定时器的定时时间等于计数器的中断周期乘以中断的次数。计数器在 CK\_CNT 的驱动下，计一个数的时间则是（PSC+1）/TIMxCLK，然后我们从0记到arr 的时间就是：

time=(arr+1)\*(PSC+1)/ TIMxCLK

time即为我们想要的定时时间。比如说我们要定时1s钟，由于TIMxCLK=24MHz，我们可以令arr=2399，PSC=9999，则：

time=（2399+1）\*（9999+1）/24000000=1s

1. **工程配置**
2. **定时器的设置**

按照点亮LED的实验完成工程配置后，依照图3. 所示配置定时器TIM2的参数。

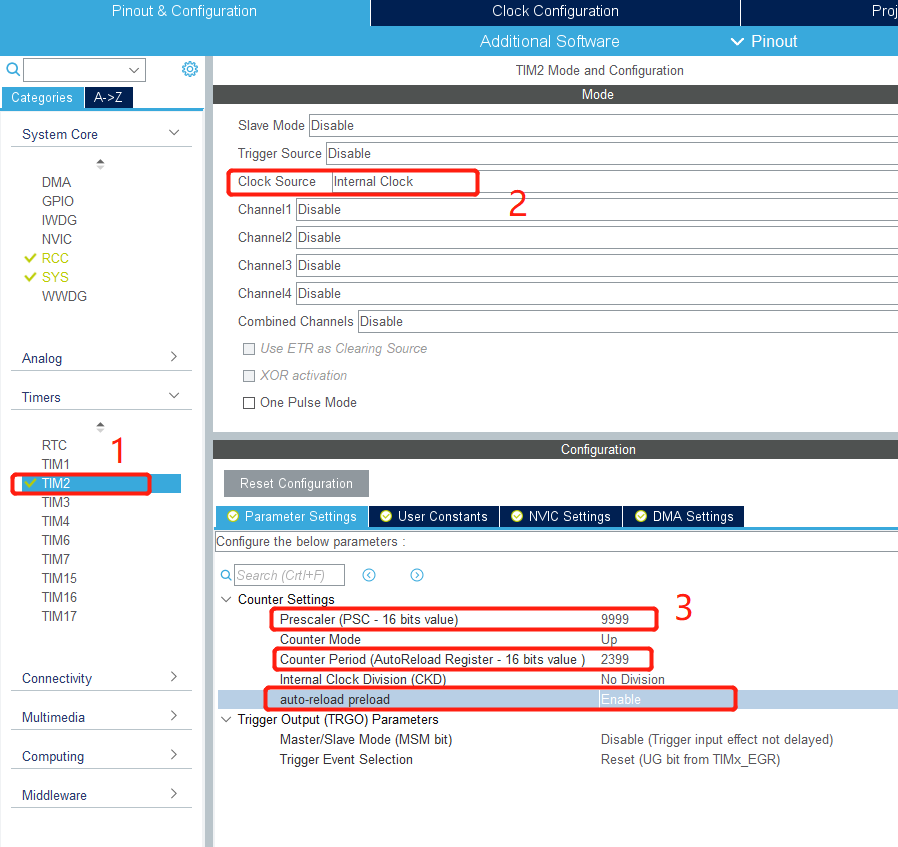


图3. TIM2定时1s的参数设置

Clock Source选择Internal Clock（内部时钟），ETR2 为外部触发输入(ETR)。

Counter Mode(计数模式) Up(向上计数模式)

auto-reload-preload(自动重装载) Enable 使能

1. **开启定时器中断**

完成定时器的设置之后，需要开启定时器中断。

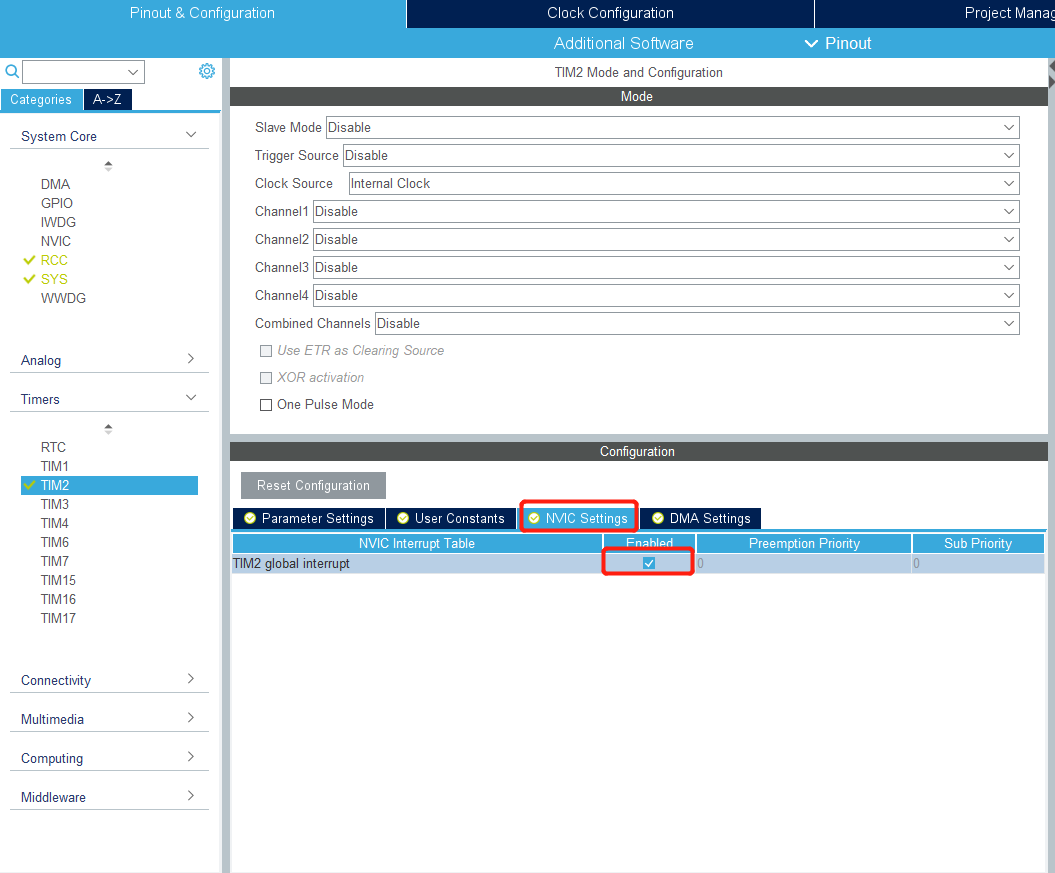
****

图4. 开启TIM2的定时器中断

1. **主函数中的修改**

完成上述配置后，生成代码。为了实现LED每1s闪烁一次的效果，我们需要修改主函数中的代码。在main.c中，写入下列代码。

|  |
| --- |
| **//在while（1）之前添加下面一行代码**  MX\_GPIO\_Init();  MX\_TIM2\_Init();  HAL\_TIM\_Base\_Start\_IT(&htim2);//初始化定时器2的中断  **//在mian函数之后，添加定时器中断函数**  **void** **HAL\_TIM\_PeriodElapsedCallback**(TIM\_HandleTypeDef \*htim)  {  **static** **unsigned** **char** ledState = 0;  **if** (htim == (&htim2))  {  **if** (ledState == 0)  HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_13,*GPIO\_PIN\_RESET*);//灯亮  **else**  HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_13,*GPIO\_PIN\_SET*);//灯灭  ledState = !ledState;  }  } |

1. **重点代码分析**

|  |
| --- |
| HAL\_TIM\_Base\_Start\_IT(&htim2);//初始化定时器2的中断 |

定时器中断处理函数，这个函数的具体作用是判断中断是否正常，然后判断产生的是哪一类定时器中断，然后进入相应的中断回调函数

|  |
| --- |
| **void** **HAL\_TIM\_PeriodElapsedCallback**(TIM\_HandleTypeDef \*htim) |

在HAL库中，每进行完一个中断，并不会立刻退出，而是会进入到中断回调函数中，这里我们是使用定时器溢出中断回调函数

|  |
| --- |
| **static** **void** **MX\_TIM2\_Init**(**void**)  {  TIM\_ClockConfigTypeDef sClockSourceConfig = {0};  TIM\_MasterConfigTypeDef sMasterConfig = {0};  htim2.Instance = TIM2;  htim2.Init.Prescaler = 9999; //PSC的值  htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP; //向上计数  htim2.Init.Period = 2399; //arr的值  htim2.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;  htim2.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;  **if** (HAL\_TIM\_Base\_Init(&htim2) != *HAL\_OK*)  {  Error\_Handler();  }  sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;  **if** (HAL\_TIM\_ConfigClockSource(&htim2, &sClockSourceConfig) != *HAL\_OK*)  {  Error\_Handler();  }  sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;  sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;  **if** (HAL\_TIMEx\_MasterConfigSynchronization(&htim2, &sMasterConfig) != *HAL\_OK*)  {  Error\_Handler();  }  } |

1. **延伸**
2. 在原有代码的基础上，实现TIM3的配置，TIM3定时时间为500ms，TIM3控制另一个LED灯的闪烁频率。
3. 对比定时器的初始化函数与GPIO的初始化函数，总结初始化程序的编程规律。