第 1 章　通用定时器 TIM6/7

1.1 STM32 定时器分类

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。

表格 1 STM32 定时器分类

	定时器	计数器分辨率	计数器类型	预分频系数	产生DMA	捕获/比较通道	互补输出
高级定时器	TIM1	16位	向上/向下	1~65535	可以	4	有
	TIM8	16位	向上/向下	1~65535	可以	4	有
通用定时器	TIM2	16位	向上/向下	1~65535	可以	4	没有
	TIM3	16位	向上/向下	1~65535	可以	4	没有
	TIM4	16位	向上/向下	1~65535	可以	4	没有
	TIM5	16位	向上/向下	1~65535	可以	4	没有
基本定时器	TIM6	16位	向上	1~65535	可以	0	没有
	TIM7	16位	向上	1~65535	可以	0	没有

1.2 定时器框图剖析

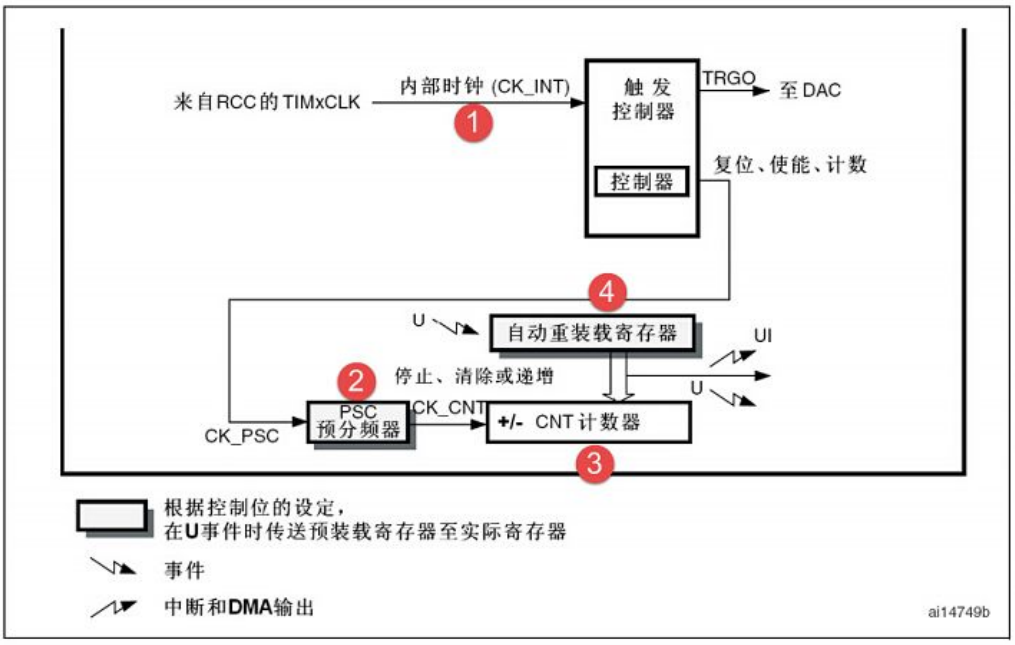


图 1 基本定时器框图

基本定时器主要由 PSC 预分频器、CNT 计数器、自动重装载寄存器这三部分组成。16 位的计数器在预分频器时钟 CK_CNT 的驱动下从 0 往上计数，计数器的计数模式只能是向上，最大是 2^16=65535,当计数值等于自动重装载寄存器中的数值时，产生一个更新事件，该事件可以是计数器溢出中断或者是 DMA 请求。更新事件之后，计数器清零，重新开始计数。

1. 定时器时钟

定时器时钟 TIMxCLK，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，库函数中 APB1 预分频的系数是 2，即 PCLK1=36M，所以定时器时钟 TIMxCLK=36*2=72M。

2. 计时器时钟

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。具体计算方式为：CK_CNT=TIMxCLK/(PSC+1)。

3. 计时器

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。

5. 定时时间计算

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 CK_CLK 的倒数，等于：1/（TIMxCLK/(PSC+1)），产生一次中断的时间则等于：1/（CK_CLK * ARR）。如果在中断服务程序里面设置一个变量 time，用来记录中断的次数，那么就可以计算出我们需要的定时时间等于：1/（CK_CLK * ARR）* time。

基本定时器除了可以产生基本的定时之外，还可以用来触发 DAC 的同步电路，实际上 TIM6 和 TIM7 在内部已经连接到了各自的 DAC 中，关于这部分的知识我们在 DAC 章节再详细讲解。

1.3 定时实验

1.3.1 硬件设计

本实验利用基本定时器 TIM6/7 定时 1s，1s 时间到 LED 翻转一次。基本定时器是单片机内部的资源，没有外部 IO，不需要接外部电路，现只需要一个 LED 即可。

1.3.2 软件设计

这里只讲解核心的部分代码，有些变量的设置，头文件的包含等并没有涉及到，完整的代码请参考本章配套的工程。

我们编写两个定时器驱动文件，bsp_TiMbase.h 和 bsp_TiMbase.h，用来配置定时器中断优先级和和初始化定时器。

1. 定时编程要点

1、开定时器时钟 TIMx_CLK, x[6,7]
2、设置自动重装载寄存器 ARR 的值
3、确定计数器的时钟，设置 PSC 的分频系数
4、初始化 TIMx, x[6,7]
5、使能 TIMx, x[6,7] update 中断
6、打开定时器
7、编写中断服务程序

通用定时器和高级定时器的定时编程要点跟基本定时器差不多，只是还要再选择下计数器的计数模式，是向上还是向下。因为基本定时器只能向上计数，且没有配置计数模式的寄存器，默认是向上。

2. 代码分析

定时器宏定义

代码 1 基本定时器宏定义

```
#define macTIM6 // 如果使用 TIM7，注释掉这个宏即可

#ifdef  macTIM6 // 使用基本定时器 TIM6
#define              macTIMx                    TIM6
#define              macTIM_APBxClock_FUN       RCC_APB1PeriphClockCmd
#define              macTIM_CLK                 RCC_APB1Periph_TIM6
#define              macTIM_IRQ                 TIM6_IRQn
#define              macTIM_INT_FUN             TIM6_IRQHandler

#else  // 使用基本定时器 TIM7
#define              macTIMx                    TIM7
#define              macTIM_APBxClock_FUN       RCC_APB1PeriphClockCmd
#define              macTIM_CLK                 RCC_APB1Periph_TIM7
#define              macTIM_IRQ                 TIM7_IRQn
#define              macTIM_INT_FUN             TIM7_IRQHandler

#endif
```

基本定时器有 TIM6 和 TIM7，我们可以有选择的使用，为了提高代码的可移植性，我们把当需要修改定时器时需要修改的代码定义成宏，默认使用的是定时器 6，如果想修改成定时器 7，只需要把宏 macTIM6 注释掉即可。

定时器初始化

代码 2 基本定时器初始化

```
void TIMx_Configuration(void)
{
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

```

```
    // 开启 TIMx_CLK,x[6,7],即内部时钟 CK_INT=72M
    macTIM_APBxClock_FUN (macTIM_CLK, ENABLE);

    // 自动重装载寄存器周的值(计数值)
    TIM_TimeBaseStructure.TIM_Period=1000;

    // 累计 TIM_Period 个频率后产生一个更新或者中断
    // 时钟预分频数为 71，则驱动计数器的时钟 CK_CNT = CK_INT / (71+1)=1M
    TIM_TimeBaseStructure.TIM_Prescaler= 71;

    // 时钟分频因子 ，基本定时器 TIM6 和 TIM7 没有，不用管
    //TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;

    // 计数器计数模式，基本定时器只能向上计数，没有计数模式的设置，不用管
    //TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;

    // 重复计数器的值，基本定时器 TIM6 和 TIM7 没有，不用管
    //TIM_TimeBaseStructure.TIM_RepetitionCounter=0;

    // 初始化定时器 TIMx, x[6,7]
    TIM_TimeBaseInit(macTIMx, &TIM_TimeBaseStructure);

    // 清除计数器中断标志位
    TIM_ClearFlag(macTIMx, TIM_FLAG_Update);

    // 开启计数器中断
    TIM_ITConfig(macTIMx,TIM_IT_Update,ENABLE);

    // 使能计数器
    TIM_Cmd(macTIMx, ENABLE);

    // 暂时关闭 TIMx,x[6,7]的时钟，等待使用
    macTIM_APBxClock_FUN (macTIM_CLK, DISABLE);
}
```

我们把定时器设置自动重装载寄存器 ARR 的值为 1000，设置时钟预分频器为 71，则驱动计数器的时钟：CK_CNT = CK_INT / (71+1)=1M，则计数器计数一次的时间等于：1/CK_CNT=1us，当计数器计数到 ARR 的值 1000 时，产生一次中断，则中断一次的时间为：1/CK_CNT*ARR=1ms。

在初始化定时器的时候，我们定义了一个结构体：TIM_TimeBaseInitTypeDef，TIM_TimeBaseInitTypeDef 结构体里面有 5 个成员，TIM6 和 TIM7 的寄存器里面只有 TIM_Prescaler 和 TIM_Period，另外三个成员基本定时器是没有的，所以使用 TIM6 和 TIM7 的时候只需初始化这两个成员即可， 另外三个成员是通用定时器和高级定时器才有，具体说明如下：

```
typedef struct {
    TIM_Prescaler             //都有
    TIM_CounterMode           //TIMx,x[6,7]没有，其他都有
    TIM_Period                //都有
    TIM_ClockDivision         //TIMx,x[6,7]没有，其他都有
    TIM_RepetitionCounter     //TIMx,x[1,8,15,16,17]才有
} TIM_TimeBaseInitTypeDef;
```

其中 TIM15/16/17 只存在与互联型产品中，在 F1 大/中/小容量型号中没有。

定时器中断优先级配置

代码 3 基本定时器中断优先级配置

```
/// TIMx,x[6,7]中断优先级配置
void TIMx_NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    // 设置中断组为 0
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

    // 设置中断来源
    NVIC_InitStructure.NVIC_IRQChannel = macTIM_IRQ;

    // 设置主优先级为 0
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;

    // 设置抢占优先级为 3
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

我们设置中断组为 0，主优先级为 0，抢占优先级为 3。

定时器中断服务程序

代码 4 基本定时器中断服务程序

```
void  macTIM_INT_FUN (void)
{
    if ( TIM_GetITStatus( macTIMx, TIM_IT_Update) != RESET ) {
        time++;
        TIM_ClearITPendingBit(macTIMx , TIM_FLAG_Update);
    }
}
```

定时器中断一次的时间是 1ms，每当进一次中断的时候我们设置一个变量 time，让 time 来记录进入中断的次数。如果我们想实现一个 1s 的定时，我们只需要判断 time 是否等于 1000 即可，1000 个 1ms 就是 1s。然后把 time 清 0，重新计数，以此循环往复。

在中断服务程序的最后，要把相应的中断标志位清除掉，切记。

主函数

代码 5 主函数

```
int main(void)
{
    /* led 端口配置 */
    LED_GPIO_Config();

    /* 基本定时器 TIMx,x[6,7] 定时配置 */
    TIMx_Configuration();

    /* 配置基本定时器 TIMx,x[6,7]的中断优先级 */
    TIMx_NVIC_Configuration();

    /* 基本定时器 TIMx,x[6,7] 重新开时钟，开始计时 */
```

```
13      macTIM_APBxClock_FUN (macTIM_CLK, ENABLE);
14
15      while (1) {
16          if ( time == 1000 ) { /* 1000 * 1 ms = 1s 时间到 */
17              time = 0;
18              /* LED1 取反 */
19              LED1_TOGGLE;
20          }
21      }
22  }
```

主函数做一些必须的初始化，然后在一个死循环中不断的判断 time 的值，time 的值在定时器中断改变，每加一次表示定时器过了 1ms，当 time 等于 1000 时，1s 时间到，LED1 翻转一次，并把 time 清 0。

1.3.3 下载验证

把编写好的程序下载到开发板，可以看到 LED1 以 1s 的频率闪烁一次。