第八章 定时器

本章主要介绍有关定时器的知识，将重点讲解基本定时器的原理，由于STM32F103C8T6 只有 TIM1-TIM4：分别是高级定时器和通用定时器，我们将利用 TIM4 来实现 1s 定时器更新中断实验——在中断完成翻转 LED2 的功能，即 LED2 一秒闪烁。本章分为如下几个部分：

8.1 定时器简介

8.2 硬件设计

8.3 软件设计

8.4 下载验证

8.1 定时器简介

定时器的核心就是计数器。

定时器工作原理：时钟源 CLK 相当于精准的时钟，来到 PSC 预分频器（相当于做除法，除以预分频系数），得到定时器一个真正工作的使用频率 TIM CLK（计时器工作的时钟频率），每来一个时钟后计数器计一个数，计到一定程度后会溢出，表示时间到，接着会产生一个事件或中断，然后 ARR 自动重装载值会重装到 CNT 计数器中，如图 8.1.1 所示。

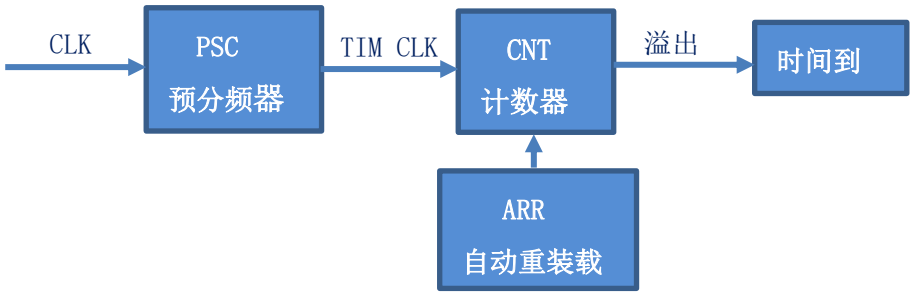


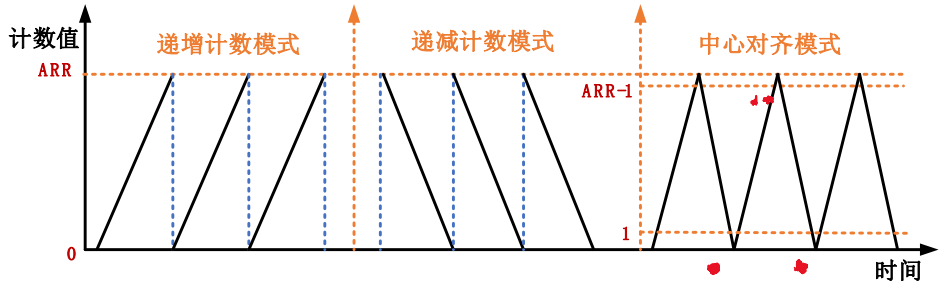
图 8.1.1 定时器定时原理图

表 8.1.2 STM32 定时器特性表（F1）

定时器类型	定时器	计数器位数	计数模式	预分频系数（整数）	产生 DMA	捕获/比较通道	互补输出
基本定时器	TIM6 TIM7	16	递增	1-65536	可以	0	无
通用定时器	TIM2 TIM3 TIM4 TIM5	16	递增、 递减、 中心对齐	1-65536	可以	4	无
高级定时器	TIM1 TIM8	16	递增、 递减、 中心对齐	1-65536	可以	4	有

表 8.1.3 STM32 定时器计数模式及溢出条件

计数器模式	溢出条件
递增计数模式	CNT= =ARR
递减计数模式	CNT= =0
中心对齐模式	CNT= =ARR-1、CNT= =1



递增计数模式实例说明：

比如 PSC = 1，ARR = 36。**分频系数=寄存器的值（PSC）+1**。

所以为二分频。

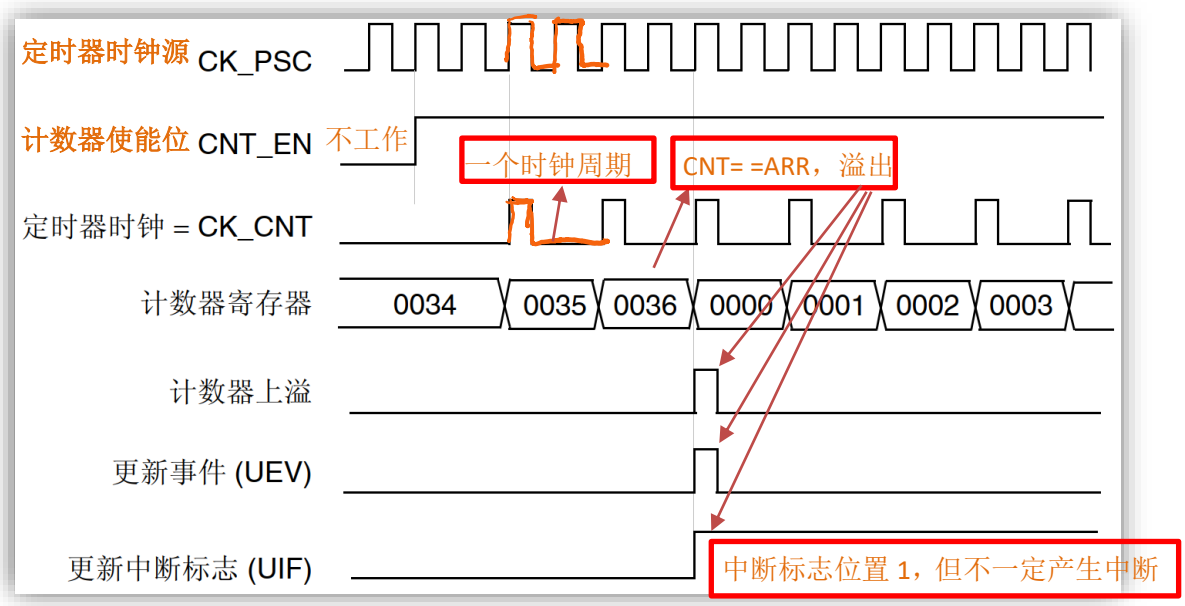


图 8.1.4 递增计数模式

递减计数模式和中心对齐模式则与递增计数模式类似。

表 8.1.3 三类定时器的主要功能介绍

定时器类型	主要功能
基本定时器	没有输入输出通道，常用作时基，即定时功能
通用定时器	具有多路独立通道，可用于输入捕获/输出比较，也可用作时基
高级定时器	除具备通用定时器所有功能外，还具备带死区控制的互补信号输出、刹车输入等功能（可用于电机控制、数字电源设计等）

功能：基础定时器<通用定时器<高级定时器

（1）基本定时器框图介绍

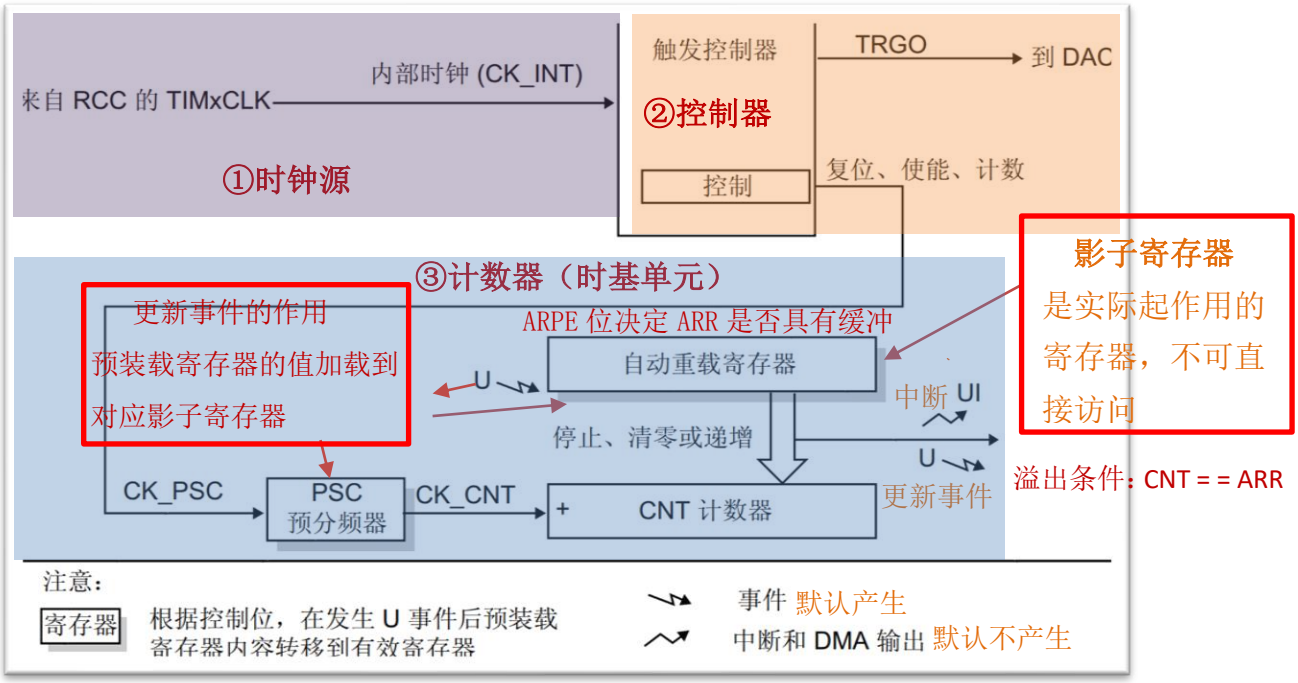


图 8.1.5 基本定时器框图

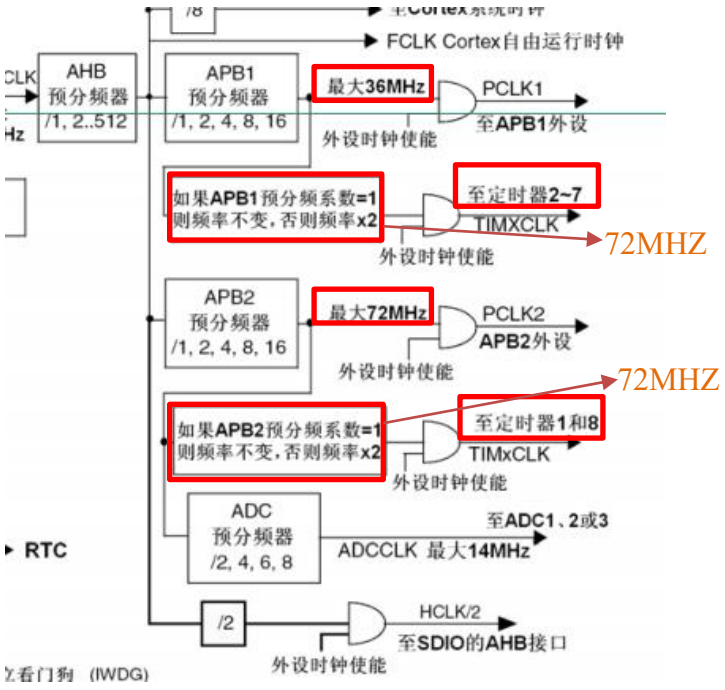


图 8.1.6 时钟树部分图

（2）定时器溢出时间计算公式

$$T_{OUT} = \frac{(ARR + 1) * (PSC + 1)}{F_t}$$

T_{OUT}是定时器溢出时间，F_t是定时器的时钟源频率

ARR是自动重装载寄存器的值，PSC是预分频器寄存器的值

- **公式推导过程：**

$$\frac{F_t}{PSC+1}\text{计数频率}\Rightarrow\frac{PSC+1}{F_t}\text{数一个数的时间}\Rightarrow\frac{PSC+1}{F_t}*(\text{ARR}+1)$$

说明：为什么 ARR+1？

至少需要有一个时钟周期到来才会溢出。

举例：在我们需要实现 LED2 一秒闪烁的实验中，我们将把 PSC 设置为 10000 - 1，把 ARR 的值设置为 7200 - 1，将这两个值代入定时器移出事件计算公式，即$T_{out}=\frac{(10000-1+1)*(7200-1+1)}{72000\,000}$ 可得到T_{out}为 1s。

说明：为什么将 PSC 设置为 10000 -1，把 ARR 设置为 7200 -1？

TIM4 是挂载在 APB1 总线上的，由图 8.2.2 可知 APB1 总线上最大频率为 36MHZ，定时器的时钟要乘以 2，所以可得到F_t为 72MHZ，接着可以求 PSC 和 ARR 的值，一个式子中有两个未知数，我们可以先代入一个系数，再求出另一个数，比如我们先带代入 PSC 的值，再求出 ARR 的值。因为F_t为 72MHZ，为了得到一个整数，就可以将 PSC 的值设为 7200 -1，接着求出 ARR 的值为 10000 -1。

（3）定时器中断实验配置步骤

在开始定时器的配置之前，我们首先要使能时钟：先看一下定时器是挂接在哪条总线上的，如图 8.1.7 所示：

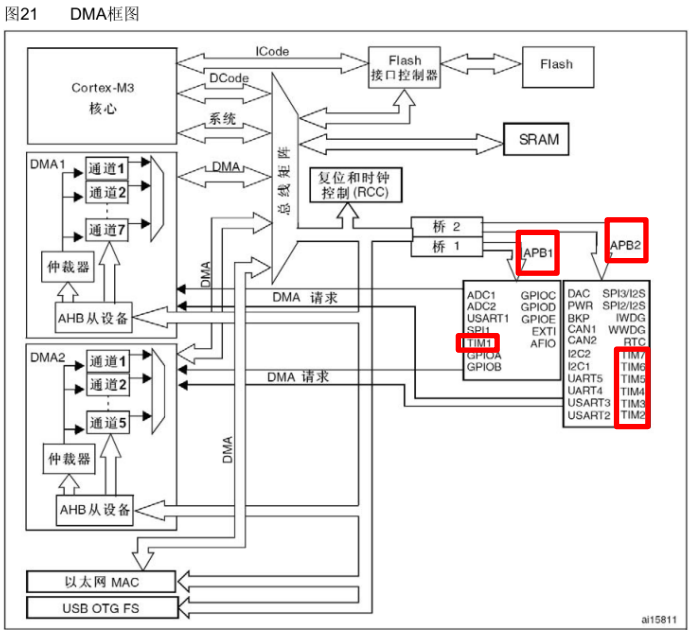


图 8.1.7　DMA 框图

可以看到**除 TIM1 外的定时器都是挂接在 APB1 总线上的**，所以我们要使能定时器时钟就要**使能 APB1 外设时钟寄存器**，看一下函数定义：

```
692 void RCC_AHBPeriphClockCmd(uint32_t RCC_AHBPeriph, FunctionalState NewState);
693 void RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph, FunctionalState NewState);
694 void RCC_APB1PeriphClockCmd(uint32_t RCC_APB1Periph, FunctionalState NewState);
```

步骤 1： 定时器的配置主要配置 4 个部分：自动重装载寄存器数值、预分频系数、时钟分割、计数模式，同样的，我们可以在库函数（stm32f10x_tim.h）中找到相关的结构体定义：

```
51 typedef struct
52 {
53   uint16_t TIM_Prescaler;         /*!< Specifies the prescaler value use
54                                        This parameter can be a number be
55
56   uint16_t TIM_CounterMode;       /*!< Specifies the counter mode.
57                                        This parameter can be a value of
58
59   uint16_t TIM_Period;            /*!< Specifies the period value to be
60                                        Auto-Reload Register at the next
61                                        This parameter must be a number b
62
63   uint16_t TIM_ClockDivision;     /*!< Specifies the clock division.
64                                        This parameter can be a value of @
65
66   uint8_t TIM_RepetitionCounter;  /*!< Specifies the repetition counter
67                                        reaches zero, an update event is
68                                        from the RCR value (N).
69                                        This means in PWM mode that (N+1)
70                                          - the number of PWM periods in
71                                          - the number of half PWM perio
72                                        This parameter must be a number b
73                                        @note This parameter is valid onl
74 } TIM_TimeBaseInitTypeDef;
```

预分频系数

计数模式

自动重装载数值

时钟分割

步骤 2：配置完定时器之后，我们还需要清除定时器状态标志位：

```
2600 void TIM_ClearFlag(TIM_TypeDef* TIMx, uint16_t TIM_FLAG)
```

我们的程序里用到的是更新标志（TIM_FLAG_Update）

步骤 3：使能定时器，即开启定时器，一旦使能了定时器，就会开始计数：

```
1067 void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState);
```

步骤 4：开启中断（中断初始化）：

```
1069 void TIM_ITConfig(TIM_TypeDef* TIMx, uint16_t TIM_IT, FunctionalState NewState);
```

步骤 5：加入中断优先级设置部分的代码。**优先级数字越小，优先级越高。**

- **中断优先级分组表：**

优先级分组	抢占优先级	响应优先级	bit[7:4] 分配情况	备注
NVIC_PriorityGroup_0	取值：0	取值：0~15	0:4	0bit抢占优先级、4bit响应优先级
NVIC_PriorityGroup_1	取值：0~1	取值：0~7	1:3	1bit抢占优先级、3bit响应优先级
NVIC_PriorityGroup_2	取值：0~3	取值：0~3	2:2	2bit抢占优先级、2bit响应优先级
NVIC_PriorityGroup_3	取值：0~7	取值：0~1	3:1	3bit抢占优先级、1bit响应优先级
NVIC_PriorityGroup_4	取值：0~15	取值：0	4:	4bit抢占优先级、0bit响应优先级

- **抢占优先级和响应优先级：**
  - ➢抢占优先级不同，会涉及到中断嵌套，抢占优先级高的会优先抢占优先级低的，优先得到执行。
  - ➢抢占优先级相同，不涉及到中断嵌套，响应优先级不同，响应优先级高的先响应（例如：两个中断同时响应，这里就会先执行响应优先级高的那个中断）。
  - ➢抢占优先级和响应优先级都相同，则比较它们的硬件中断编号，中断编号越小，优先级越高。

由于我们没有用到中断的嵌套，所以抢占优先级设置为 0.

8.2 硬件设计

本章需要用到的硬件资源有：

1）极风 STM32 开发板

2）STLINK 下载器

下面介绍一下 STM32 开发板和 STLINK 下载器的连接，STLINK 的 3.3V、SWCLK、SWDIO、GND 分别连在 STM32 开发板的 3V3、CLK、DIO、GND 上。总体连接实物图如图 8.2.1 所示：

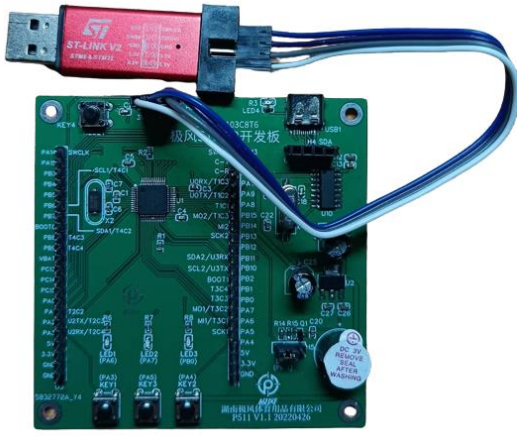


图 8.2.1 总体连接实物图

8.3 软件设计

（1）程序实现的功能

利用 TIM4 来实现 1s 定时器更新中断实验——在中断完成翻转 LED2。

（2）程序的实现

打开**实验 1 基本定时器\motor.uvprojx**，我们可以看到工程中拥有 5 个源文件，分别是 led.c、led.h、btim.c、btim.h、main.c。led.c 文件存放 led 驱动代码，btim.c 存放 TIM4 的驱动代码，main.c 文件存放应用代码。led 相关文件中都表明了注释，大家可自行查看，这里就不作详细解释。

打开 btim.c 文件，代码如下：

```
#include "led.h"
#include "btim.h"

/**********************************
* 函数全称:
* void Basic_TIM4_Init(uint16_t arr, uint16_t pre)
*
* 函数作用:
* 初始化 TIM4
* **********************************/
 void Basic_TIM4_Init(uint16_t arr, uint16_t pre)
 {
     TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
     NVIC_InitTypeDef NVIC_InitStructure;

      // 使能 APB1 外设时钟寄存器
     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
     TIM_TimeBaseStructure.TIM_Period = arr;
     TIM_TimeBaseStructure.TIM_Prescaler = pre; //CK_CNT=CKINT/(pre+1)
     // 时钟分频因子 ，基本定时器没有，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
     TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;
     // 重复计数器的值，基本定时器没有，不用管
     // TIM_TimeBaseStructure.TIM_RepetitionCounter=0;
     TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);   // 初始化定时器

     TIM_ClearFlag(TIM4,TIM_FLAG_Update);          // 清除定时器的状态标志位
     TIM_Cmd(TIM4, ENABLE);                        // 打开定时器
     TIM_ITConfig(TIM4,TIM_IT_Update,ENABLE);   // 开启更新中断（上溢中断）
```

```
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);   // 设置中断组
    NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;   // 选择中断通道
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; //设置抢占优先级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;    // 设置响应优先级
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;    // 使能中断通道
    NVIC_Init(&NVIC_InitStructure);   // 调用 NVIC 初始化函数
}

/**********************************
* 函数全称:
* void TIM4_IRQHandler(void)
*
* 函数作用:
* 每次产生一个中断溢出时间就会进入一次中断函数
* **********************************/
void TIM4_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM4, TIM_IT_Update) == SET) //检测是否溢出中断
    {
        // 清除中断标志位，否则启动时会进入中断服务函数
        TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
        LED2_Turn();
    }
}
```

下面我们看看头文 btim.h 的代码，代码中包含一些头文件定义，数据类型、函数声明，代码如下：

```
#ifndef _BTIM_H
#define _BTIM_H
#include "stdlib.h"
#include "stm32f10x.h"

void Basic_TIM4_Init(uint16_t arr, uint16_t pre);   /* 定时中断初始化函数 */

#endif
```

最后，我们在 main 函数里编写应用代码，main.c 文件如下：

```
#include "stm32f10x.h"
#include <string.h>
#include <stdlib.h>

#include "stdio.h"
#include "delay.h"
#include "led.h"
#include "btim.h"

// 主函数
 int main(void)
{
    LED_Init();                                 /* 初始化 LED */
    Basic_TIM4_Init(10000 -1, 7200 -1);   /* 10KHZ 的计数频率，计数 10K 次为 1s */

    while(1)
    {
        LED1_Turn();   //作为程序在运行的标志
        delay_ms(200);
    }
}
```

至此，我们的软件设计部分就结束了。

8.4 下载验证

在代码编译成功下载程序完成后，可观察到 LED1 闪烁表示程序正在运行，LED2 会呈现 1s 闪烁，即成功完成实验 1 基本定时器。