**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。

**CNT**

**计数器**

**ARR**

**自动重装载值**

**自动重装载值**

**时间到**

溢出

**PSC**

**预分频器**

TIM CLK

CLK

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

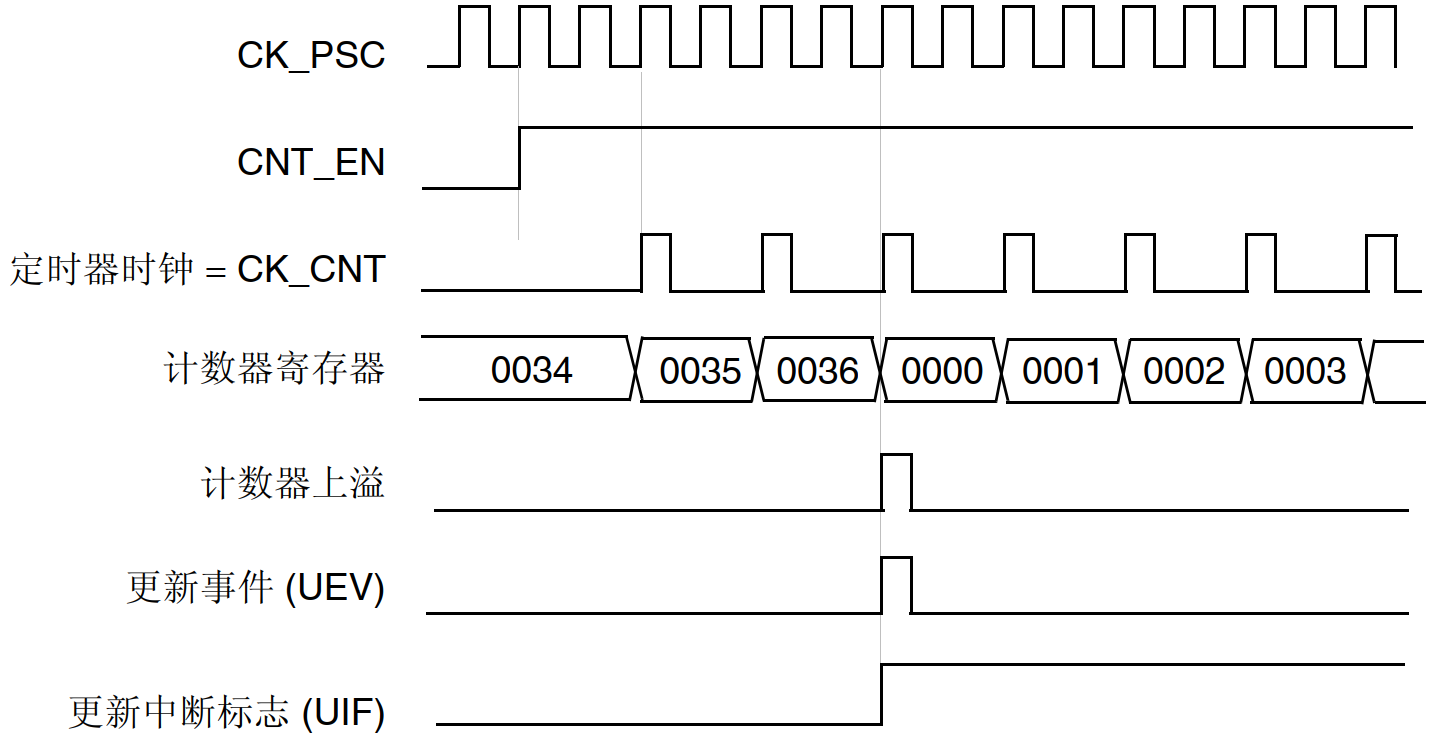
|  |  |
| --- | --- |
| 计数器模式 | 溢出条件 |
| 递增计数模式 | CNT= =ARR |
| 递减计数模式 | CNT= =0 |
| 中心对齐模式 | CNT= =ARR-1、CNT= =1 |





**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。

中断标志位置1，但不一定产生中断出

CNT= =ARR，溢出出

一个时钟周期



不工作

**计数器使能位**

**定时器时钟源**

图8.1.4 递增计数模式

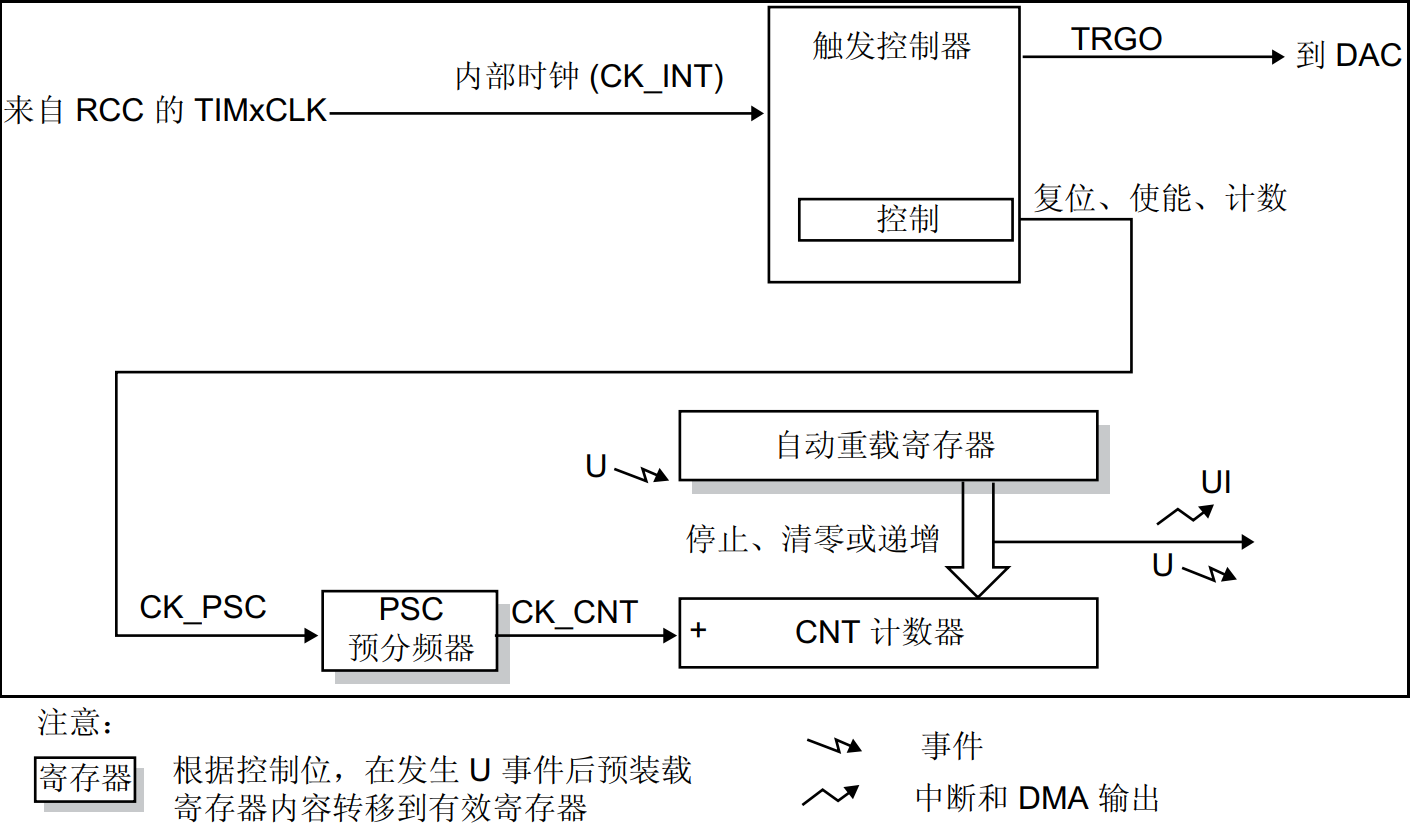
递减计数模式和中心对齐模式则与递增计数模式类似。

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**



更新事件的作用

预装载寄存器的值加载到对应影子寄存器

**③计数器（时基单元）**

**①时钟源**

默认不产生

默认产生

ARPE位决定ARR是否具有缓冲

更新事件

溢出条件：CNT = = ARR

**影子寄存器**

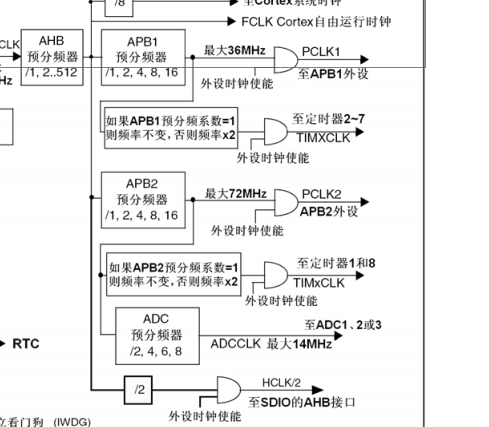
是实际起作用的寄存器，不可直接访问

中断



**②控制器**

图8.1.5 基本定时器框图



72MHZ

72MHZ

图8.1.6 时钟树部分图

**（2）定时器溢出时间计算公式**

是定时器溢出时间，是定时器的时钟源频率

是自动重装载寄存器的值，是预分频器寄存器的值

* **公式推导过程：**

计数频率 数一个数的时间

**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 可得到为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：

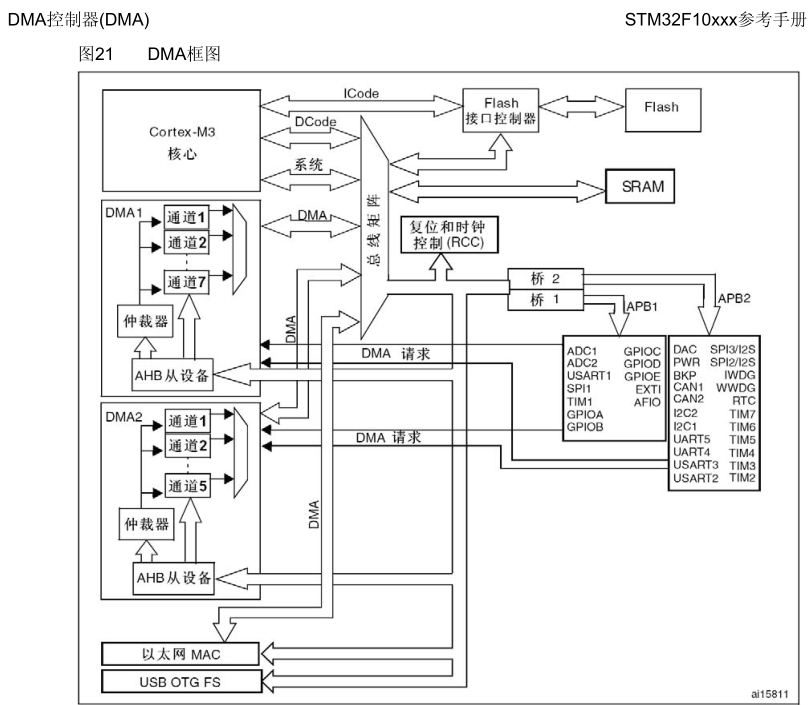
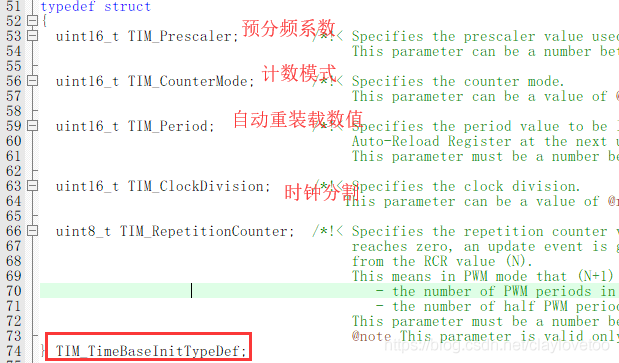


图8.1.7 DMA框图

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：

在这里插入图片描述

**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：



**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：**

****

**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：** 在这里插入图片描述

**步骤4：开启中断（中断初始化）：**在这里插入图片描述

**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：**



* **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：

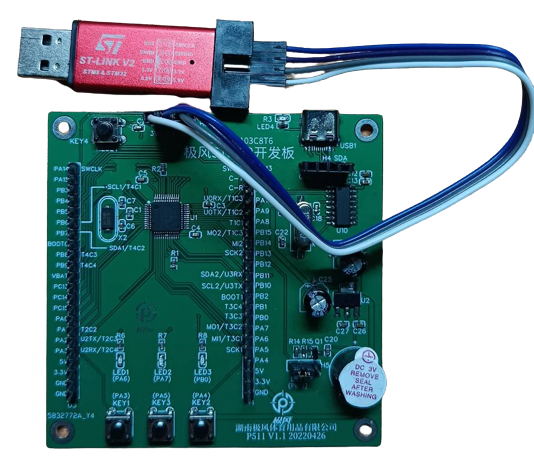


图8.2.1 总体连接实物图

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）

我们没有用到中断的嵌套，所以抢占优先级设置为0。

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。