

第 12 章

定时器中断实验

这一章将介绍如何使用 STM32 的通用定时器。STM32 的定时器功能十分强大,有 TIME1 和 TIME8 等高级定时器,也有 TIME2~TIME5 等通用定时器,还有 TIME6 和 TIME7 等基本定时器。本章使用 TIM3 的定时器中断来控制 DS1 的翻转,在主函数用 DS0 的翻转来提示程序正在运行。本章会选择难度适中的通用定时器来介绍。

12.1 STM32 通用定时器简介

STM32 的通用定时器是一个通过可编程预分频器(PSC)驱动的 16 位自动装载计数器(CNT)构成。STM32 的通用定时器可以用于测量输入信号的脉冲长度(输入捕获)或者产生输出波形(输出比较和 PWM)等。使用定时器预分频器和 RCC 时钟控制器预分频器,脉冲长度和波形周期可以在几个微秒到几个毫秒间调整。STM32 的每个通用定时器都是完全独立的,没有互相共享的任何资源。

STM3 的通用 TIM_x (TIM2、TIM3、TIM4 和 TIM5)定时器功能包括:

- ① 16 位向上、向下、向上/向下自动装载计数器(TIM_x_CNT)。
- ② 16 位可编程(可以实时修改)预分频器(TIM_x_PSC),计数器时钟频率的分频系数为 1~65 535 之间的任意数值。
- ③ 4 个独立通道(TIM_x_CH1~4),这些通道可以用来作为输入捕获、输出比较、PWM 生成(边缘或中间对齐模式)、单脉冲模式输出。
- ④ 可使用外部信号(TIM_x_ETR)控制定时器和定时器互连(可以用一个定时器控制另外一个定时器)的同步电路。
- ⑤ 如下事件发生时产生中断/DMA:
 - 更新:计数器向上溢出/向下溢出,计数器初始化(通过软件或内部/外部触发);
 - 触发事件(计数器启动、停止、初始化或者由内部/外部触发计数);
 - 输入捕获;
 - 输出比较;
 - 支持针对定位的增量(正交)编码器和霍尔传感器电路;
 - 触发输入作为外部时钟或者按周期的电流管理。

由于 STM32 通用定时器比较复杂,这里不多介绍,可直接参考《STM32 参考手册》第 253 页。下面介绍与这章实验密切相关的几个通用定时器的寄存器。

首先是控制寄存器 1(TIMx_CR1),各位描述如图 12.1 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留						CKD[1:0]	ARPE	CMS[1:0]	DIR	OPM	URS	UDIS	CEN		
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
位0		CEN: 使能计数器 0: 禁止计数器; 1: 使能计数器。 注: 在软件设置了CEN位后,外部时钟、门控模式和编码器模式才能工作。触发模式可以自动地通过硬件设置CEN位。 在单脉冲模式下,当发生更新事件时,CEN被自动消除													

图 12.1 TIMx_CR1 寄存器各位描述

本实验只用到了 TIMx_CR1 的最低位(位 0),也就是计数器使能位;该位必须置 1,才能让定时器开始计数。接下来介绍第二个寄存器: DMA/中断使能寄存器(TIMx_DIER)。该寄存器是一个 16 位的寄存器,各位描述如图 12.2 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留	TDE	保留	CC4DE	CC3DE	CC2DE	CC1DE	UDE	保留	TIE	保留	CC4IE	CC3IE	CC2IE	CC1IE	UIE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
位0		UIE: 允许更新中断(Update interrupt enable) 0: 禁止更新中断; 1: 允许更新中断													

图 12.2 TIMx_DIER 寄存器各位描述

这里同样仅关心它的最低位,该位是更新中断允许位,本章用到的是定时器的更新中断,所以该位要设置为 1 来允许由于更新事件产生的中断。

接下来看第 3 个与这章有关的寄存器: 预分频寄存器(TIMx_PSC)。该寄存器用设置对时钟进行分频,然后提供给计数器,作为计数器的时钟。各位描述如图 12.3 所示。这里,定时器的时钟来源有 4 个:

- 内部时钟(CK_INT);
- 外部时钟模式 1: 外部输入脚(TIx);
- 外部时钟模式 2: 外部触发输入(ETR);
- 内部触发输入(ITRx): 使用 A 定时器作为 B 定时器的预分频器(A 为 B 提供时钟)。

这些时钟具体选择哪个可以通过 TIMx_SMCR 寄存器的相关位来设置。这里的 CK_INT 时钟是从 APB1 倍频来的,STM32 中除非 APB1 的时钟分频数设置为 1,否则通用定时器 TIMx 的时钟是 APB1 时钟的 2 倍。当 APB1 的时钟不分频的时候,通用定时器 TIMx 的时钟就等于 APB1 的时钟。注意,高级定时器的时钟不是来

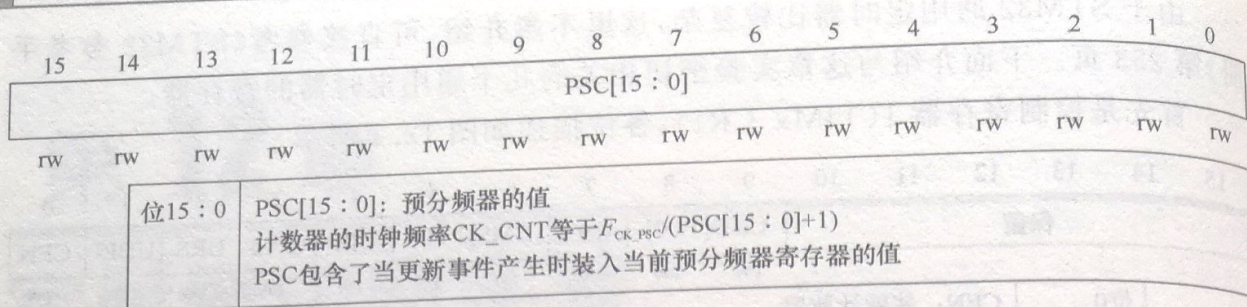


图 12.3 TIMx_PSC 寄存器各位描述

自 APB1,而是来自 APB2 的。

顺带介绍一下 TIMx_CNT 寄存器。该寄存器是定时器的计数器,存储了当前定时器的计数值。

接着介绍自动重装载寄存器(TIMx_ARR)。该寄存器在物理上实际对应着 2 个寄存器。一个是程序员可以直接操作的,另外一个程序员看不到的,这个看不到的寄存器在《STM32 参考手册》里面叫影子寄存器。事实上真正起作用的是影子寄存器。根据 TIMx_CR1 寄存器中 APRE 位的设置:APRE=0 时,预装载寄存器的内容可以随时传送到影子寄存器,此时二者是连通的;而 APRE=1 时,在每一次更新事件(UEV)时才把预装在寄存器的内容传送到影子寄存器。自动重装载寄存器的各位描述如图 12.4 所示。

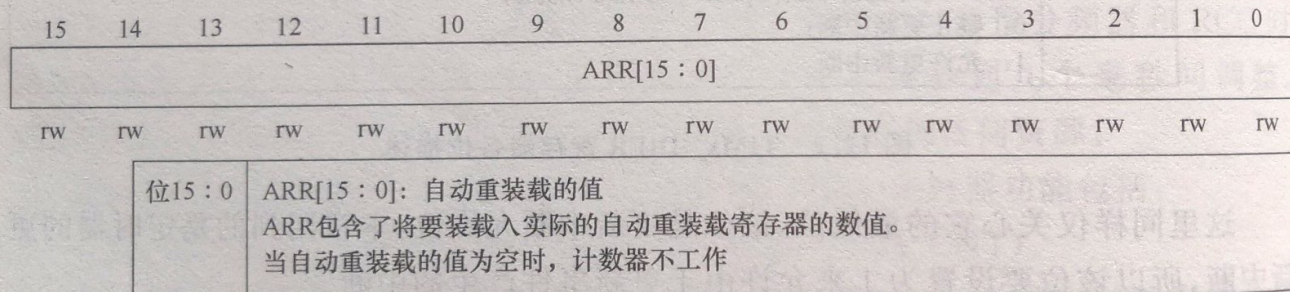


图 12.4 TIMx_ARR 寄存器各位描述

最后介绍的是状态寄存器(TIMx_SR)。该寄存器用来标记当前与定时器相关的各种事件/中断是否发生,各位描述如图 12.5 所示。

TIMx_SR 寄存器同样只用到了最低位,当计数器 CNT 被重新初始化的时候,产生更新中断标记,通过这个中断标志位就可以知道产生中断的类型。这些位的详细描述请参考《STM32 参考手册》第 282 页。只要对以上几个寄存器进行简单设置,就可以使用通用定时器了,并且可以产生中断。

这一章将使用定时器产生中断,然后在中断服务函数里面翻转 DS1 上的电平来指示定时器中断的产生。接下来以通用定时器 TIM3 为实例来说明要经过哪些步骤,才能达到这个要求并产生中断。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留		CC4OF	CC3OF	CC2OF	CC1OF	保留		TIF	保留	CC4IF	CC3IF	CC2IF	CC1IF	UIF	
		rc w0	rc w0	rc w0	rc w0			rc w0			rc w0	rc w0	rc w0	rc w0	rc w0
位0		UIF: 更新中断标记(Update interrupt flag) 当好产生更新事件时该位由硬件置“1”。它由软件清‘0’。 0: 无更新事件产生; 1: 更新中断等待响应。当寄存器被更新时该位由硬件置“1”: — 若TIMx_CR1寄存器的UDIS=0、URS=0, 当TIMx_EGR寄存器的UG=1时产生更新事件(软件对计数器CNT重新初始化); — 若TIMx_CR1寄存器的UDIS=0、URS=0, 当计数器CNT被触发事件重初始化时产生更新事件(参考同步控制寄存器的说明)													

图 12.5 TIMx_SR 寄存器各位描述

1) TIM3 时钟使能

这里通过 APB1ENR 的第 1 位来设置 TIM3 的时钟, 因为 Stm32_Clock_Init 函数里面把 APB1 的分频设置为 2 了, 所以 TIM3 时钟就是 APB1 时钟的 2 倍, 等于系统时钟(72 MHz)。

2) 设置 TIM3_ARR 和 TIM3_PSC 的值

通过这两个寄存器来设置自动重装的值以及分频系数。这两个参数加上时钟频率就决定了定时器的溢出时间。

3) 设置 TIM3_DIER 允许更新中断

因为要使用 TIM3 的更新中断, 所以设置 DIER 的 UIE 位为 1, 使能更新中断。

4) 允许 TIM3 工作

光配置好定时器还不行, 没有开启定时器照样不能用。配置完后要开启定时器, 通过 TIM3_CR1 的 CEN 位来设置。

5) TIM3 中断分组设置

定时器配置完之后, 因为要产生中断, 必不可少地要设置 NVIC 相关寄存器, 以使能 TIM3 中断。

6) 编写中断服务函数

最后还要编写定时器中断服务函数, 从而处理定时器产生的相关中断。在中断产生后, 通过状态寄存器的值来判断此次产生的中断属于什么类型。然后执行相关的操作, 这里使用的是更新(溢出)中断, 所以在状态寄存器 SR 的最低位。处理完中断之后应该向 TIM3_SR 的最低位写 0, 从而清除该中断标志。

通过以上几个步骤就可以达到我们的目的了, 使用通用定时器的更新中断来控制 DS1 的亮灭。

12.2 硬件设计

本实验用到的硬件资源有指示灯 DS0 和 DS1、定时器 TIM3。本章将通过

TIM3 的中断来控制 DS1 的亮灭,DS0 和 DS1 的电路在前面已经有介绍了。而 TIM3 属于 STM32 的内部资源,只需要软件设置即可正常工作。

12.3 软件设计

软件设计在之前的工程上面增加,不过没用到看门狗,所以先去掉 wdg.c(注意,此时 HARDWARE 组仅剩 led.c)。首先在 HARDWARE 文件夹下新建 TIMER 的文件夹,然后打开 USER 文件夹下的工程,新建一个 timer.c 的文件和 timer.h 的头文件,保存在 TIMER 文件夹下,并将 TIMER 文件夹加入头文件包含路径。在 timer.c 里输入如下代码:

```
#include "timer.h"
#include "led.h"
//定时器 3 中断服务程序
void TIM3_IRQHandler(void)
{
    if(TIM3->SR&0X0001) LED1 = ! LED1; //溢出中断
    TIM3->SR&= ~(1<<0); //清除中断标志位
}
//通用定时器 3 中断初始化
//这里时钟选择为 APB1 的 2 倍,而 APB1 为 36M
//arr: 自动重装值。
//psc: 时钟预分频数
//这里使用的是定时器 3
void TIM3_Int_Init(u16 arr,u16 psc)
{
    RCC->APB1ENR|= 1<<1; //TIM3 时钟使能
    TIM3->ARR= arr; //设定计数器自动重装值//刚好 1 ms
    TIM3->PSC= psc; //预分频器 7200,得到 10 kHz 的计数时钟
    TIM3->DIER|= 1<<0; //允许更新中断
    TIM3->CR1|= 0x01; //使能定时器 3
    MY_NVIC_Init(1,3,TIM3_IRQn,2); //抢占 1,子优先级 3,组 2
}
```

该文件下包含一个中断服务函数和一个定时器 3 中断初始化函数。中断服务函数比较简单,在每次中断后判断 TIM3 的中断类型,如果中断类型正确,则执行 LED1(DS1)的取反。

TIM3_Int_Init 函数就是执行我们上面介绍的 5 个步骤,使得 TIM3 开始工作,并开启中断。该函数的 2 个参数用来设置 TIM3 的溢出时间。因为 Stm32_Clock_Init 函数里面已经初始化 APB1 的时钟为 2 分频,所以 APB1 的时钟为 36 MHz,而

从 STM32 的内部时钟树图得知,当 APB1 的时钟分频数为 1 的时候,TIM2~7 的时钟为 APB1 的时钟;而如果 APB1 的时钟分频数不为 1,那么 TIM2~7 的时钟频率将为 APB1 时钟的两倍。因此,TIM3 的时钟为 72 MHz,再根据设计的 arr 和 psc 的值就可以计算中断时间了,计算公式如下:

$$T_{\text{out}} = ((\text{arr} + 1) \times (\text{psc} + 1)) / T_{\text{clk}}$$

其中, T_{clk} 为 TIM3 的输入时钟频率(单位为 MHz)。 T_{out} 为 TIM3 溢出时间(单位为 μs)。将 timer.c 文件保存,然后加入到 HARDWARE 组下。接下来,在 timer.h 文件里输入如下代码:

```
#ifndef __TIMER_H
#define __TIMER_H
#include "sys.h"
void TIM3_Int_Init(u16 arr,u16 psc);
#endif
```

最后,在主程序里面输入如下代码:

```
int main(void)
{
    Stm32_Clock_Init(9);           //系统时钟设置
    delay_init(72);                //延时初始化
    uart_init(72,9600);            //串口初始化
    LED_Init();                    //初始化与 LED 连接的硬件接口
    TIM3_Int_Init(5000,7199);      //10 kHz 的计数频率,计数到 5000 为 500 ms
    while(1)
    {
        LED0 = ! LED0;
        delay_ms(200);
    }
}
```

这里的代码和之前大同小异,此段代码对 TIM3 进行初始化之后进入死循环等待 TIM3 溢出中断,当 TIM3_CNT 的值等于 TIM3_ARR 的值时就会产生 TIM3 的更新中断,然后在中断里面取反 LED1,TIM3_CNT 再从 0 开始计数。

12.4 下载验证

完成软件设计之后,将编译好的文件下载到 MiniSTM32 开发板上,观看其运行结果是否与我们编写的一致。如果没有错误,则看到 DS0 不停闪烁(每 400 ms 闪烁一次),而 DS1 也是不停闪烁,但是闪烁时间较 DS0 慢(1 s 一次)。