实验 8　定时器实验

这一章，我们将向大家介绍如何使用 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。在《 STM32 参考手册》里面，定时器的介绍占了 1/5 的篇幅，足见其重要性。在本章中，我们将利用 TIM3 的定时器中断来控制板子上面 8 个 LED 循环点亮的效果。本章分为以下学习目标：

1、学会操作 STM32 的定时器。

1.1 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 的通用定时器可以被用于：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 TIMx (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（ TIMx_CNT）。

2)16 位可编程(可以实时修改)预分频器(TIMx_PSC)，计数器时钟频率的分频系数为 1～65535 之间的任意数值。

3） 4 个独立通道（ TIMx_CH1~4)，这些通道可以用来作为：

A．输入捕获

B．输出比较

C． PWM 生成(边缘或中间对齐模式)

D．单脉冲模式输出

4）可使用外部信号（ TIMx_ETR）控制定时器和定时器互连（可以用 1 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：

A．更新：计数器向上溢出/向下溢出，计数器初始化(通过软件或者内部/外部触发)

B．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)

C．输入捕获

D．输出比较

E．支持针对定位的增量(正交)编码器和霍尔传感器电路

F．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，这里我们不再多介绍，请大家直接参考《 STM32 参考手册》第 253 页，通用定时器一章。 为了深入了解 STM32 的通用寄存器， 下面我们先介绍一下与我们这章的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），该寄存器的各位描述如下所示：

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15:10	保留，始终读为0。
位9:8	**CKD[1:0]**: 时钟分频因子 定义在定时器时钟(CK_INT)频率与数字滤波器(ETR,TIx)使用的采样频率之间的分频比例。 00：$t_{DTS} = t_{CK\_INT}$ 01：$t_{DTS} = 2 \times t_{CK\_INT}$ 10：$t_{DTS} = 4 \times t_{CK\_INT}$ 11：保留
位7	**ARPE**：自动重装载预装载允许位 0：TIMx_ARR寄存器没有缓冲； 1：TIMx_ARR寄存器被装入缓冲器。
位6:5	**CMS[1:0]**：选择中央对齐模式 00：边沿对齐模式。计数器依据方向位(DIR)向上或向下计数。 01：中央对齐模式1。计数器交替地向上和向下计数。配置为输出的通道(TIMx_CCMRx寄存器中CCxS=00)的输出比较中断标志位，只在计数器向下计数时被设置。 10：中央对齐模式2。计数器交替地向上和向下计数。计数器交替地向上和向下计数。配置为输出的通道(TIMx_CCMRx寄存器中CCxS=00)的输出比较中断标志位，只在计数器向上计数时被设置。 11：中央对齐模式3。计数器交替地向上和向下计数。计数器交替地向上和向下计数。配置为输出的通道(TIMx_CCMRx寄存器中CCxS=00)的输出比较中断标志位，在计数器向上和向下计数时均被设置。 注：在计数器开启时(CEN=1)，不允许从边沿对齐模式转换到中央对齐模式。
位4	**DIR**：方向 0：计数器向上计数； 1：计数器向下计数。 注：当计数器配置为中央对齐模式或编码器模式时，该位为只读。

位3	**OPM**：单脉冲模式 0：在发生更新事件时，计数器不停止； 1：在发生下一次更新事件(清除CEN位)时，计数器停止。
位2	**URS**：更新请求源 软件通过该位选择UEV事件的源 0：如果允许产生更新中断或DMA请求，则下述任一事件产生一个更新中断或DMA请求： – 计数器溢出/下溢 – 设置UG位 – 从模式控制器产生的更新 1：如果允许产生更新中断或DMA请求，则只有计数器溢出/下溢才产生一个更新中断或DMA请求。
位1	**UDIS**：禁止更新 软件通过该位允许/禁止UEV事件的产生 0：允许UEV。更新(UEV)事件由下述任一事件产生： – 计数器溢出/下溢 – 设置UG位 – 从模式控制器产生的更新 被缓存的寄存器被装入它们的预装载值。 1：禁止UEV。不产生更新事件，影子寄存器(ARR、PSC、CCRx)保持它们的值。如果设置了UG位或从模式控制器发出了一个硬件复位，则计数器和预分频器被重新初始化。
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被自动清除。

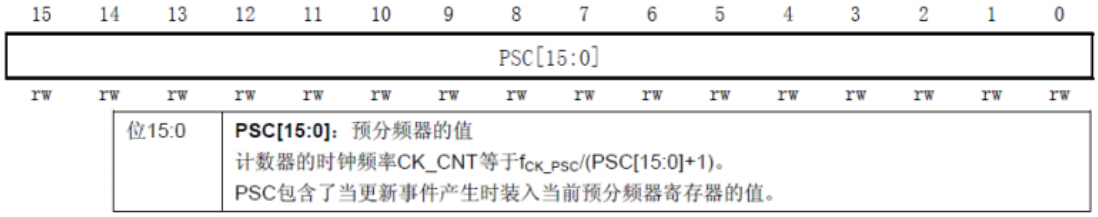
首先我们来看看 TIMx_CR1 的最低位，也就是计数器使能位，该位必须置 1，才能让定时器开始计数。从第 4 位 DIR 可以看出默认的计数方式是向上计数，同时也可以向下计数，第 5,6 位是设置计数对齐方式的。 从第 8 和第 9 位可以看出，我们还可以设置定时器的时钟分频因子为 1,2,4。

接下来介绍第二个与我们这章密切相关的寄存器： DMA/中断使能寄存器（ TIMx_DIER)。该寄存器是一个 16 位的寄存器，其各位描述如图所示：

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

这里我们同样仅关心它的第 0 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所产生的中断。

接下来我们看第三个与我们这章有关的寄存器：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。该寄存器的各位描述如图所示：



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位15:0	**PSC[15:0]**：预分频器的值 计数器的时钟频率CK_CNT等于f_{CK_PSC}/(PSC[15:0]+1)。 PSC包含了当更新事件产生时装入当前预分频器寄存器的值。

这里，定时器的时钟来源有 4 个：

1） 内部时钟（ CK_INT）

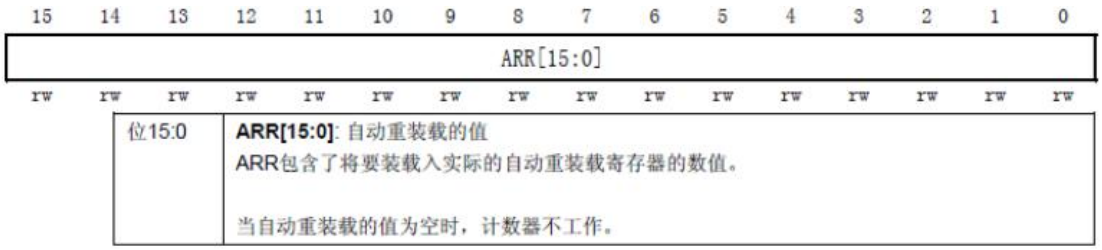
2） 外部时钟模式 1：外部输入脚（ TIx）

3） 外部时钟模式 2：外部触发输入（ ETR）

4） 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 倍频的来的，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。这里还要注意的就是高级定时器的时钟不是来自 APB1，而是来自 APB2 的。这里顺带介绍一下 TIMx_CNT 寄存器，该寄存器是定时器的计数器，该寄存器存储了当前定时器的计数值。

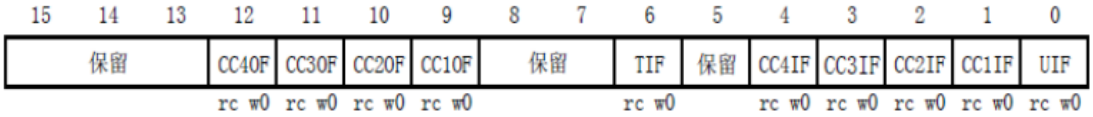
接着我们介绍自动重装载寄存器（ TIMx_ARR），该寄存器在物理上实际对应着 2 个寄存器。一个是程序员可以直接操作的，另外一个是程序员看不到的，这个看不到的寄存器在《 STM32 参考手册》里面被叫做影子寄存器。事实上真正起作用的是影子寄存器。 根据 TIMx_CR1 寄存器中 APRE 位的设置： APRE=0 时，预装载寄存器的内容可以随时传送到影子寄存器，此时 2 者是连通的；而 APRE=1 时，在每一次更新事件（ UEV）时，才把预装在寄存器的内容传送到影子寄存器。自动重装载寄存器的各位描述如图所示：



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位15:0	**ARR[15:0]**: 自动重装载的值 ARR包含了将要装载入实际的自动重装载寄存器的数值。 当自动重装载的值为空时，计数器不工作。

最后，我们要介绍的寄存器是：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。该寄存器的各位描述如图所示：



关于这些位的详细描述，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的设置，我们就可以使用通用定时器了，并且可以产生中断。

1.2 定时器库函数

这一章，我们将使用定时器产生中断，然后在中断服务函数里面对 LED 进行移位显示，也就可以知道定时器是否进入中断。接下来我们以通用定时器 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。 这里我们就对每个步骤通过库函数的实现方式来描述。 首先要提到的是，定时器相关的库函数主要集中在固件库文件 stm32f10x_tim.h 和 stm32f10x_tim.c 文件中。

1） TIM3 时钟使能。

TIM3 是挂载在 APB1 之下，所以我们通过 APB1 总线下的使能使能函数来使能 TIM3。调用的函数是：

```
/* 开启定时器 3 时钟 */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);
```

2） 初始化定时器参数,设置自动重装值， 分频系数，计数方式等。

在库函数中，定时器的初始化参数是通过初始化函数 TIM_TimeBaseInit 实现的：

```
voidTIM_TimeBaseInit(TIM_TypeDef*TIMx,TIM_TimeBaseInitTypeDef*
TIM_TimeBaseInitStruct);
```

第一个参数是确定是哪个定时器，这个比较容易理解。 第二个参数是定时器初始化参数结构体指针，结构体类型为 TIM_TimeBaseInitTypeDef，下面我们看看这个结构体的定义：

```
typedef struct
{
uint16_t TIM_Prescaler;
uint16_t TIM_CounterMode;
uint16_t TIM_Period;
uint16_t TIM_ClockDivision;
```

```
uint8_t TIM_RepetitionCounter;
} TIM_TimeBaseInitTypeDef;
```

这个结构体一共有 5 个成员变量，要说明的是，对于通用定时器只有前面四个参数有用，最后一个参数 TIM_RepetitionCounter 是高级定时器才有用的，这里不多解释。

第一个参数 TIM_Prescaler 是用来设置分频系数的，刚才上面有讲解。

第二个参数 TIM_CounterMode 是用来设置计数方式，上面讲解过，可以设置为向上计数，向下计数方式还有中央对齐计数方式， 比较常用的是向上计数模式 TIM_CounterMode_Up 和向下计数模式 TIM_CounterMode_Down。

第三个参数是设置自动重载计数周期值，这在前面也已经讲解过。

第四个参数是用来设置时钟分频因子。

针对 TIM3 初始化范例代码格式：

```
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_TimeBaseStructure.TIM_Period = 5000;
TIM_TimeBaseStructure.TIM_Prescaler =7199;
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
```

3） 设置 TIM3_DIER 允许更新中断。

因为我们要使用 TIM3 的更新中断， 寄存器的相应位便可使能更新中断。 在库函数里面定时器中断使能是通过 TIM_ITConfig 函数来实现的：

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

第一个参数是选择定时器号，这个容易理解，取值为 TIM1~TIM17。

第二个参数非常关键，是用来指明我们使能的定时器中断的类型，定时器中断的类型有很多种，包括更新中断 TIM_IT_Update，触发中断 TIM_IT_Trigger，以及输入捕获中断等等。

第三个参数就很简单了， 就是失能还是使能。

例如我们要使能 TIM3 的更新中断，格式为：

TIM_ITConfig(TIM3,TIM_IT_Update,ENABLE);

4） TIM3 中断优先级设置。

在定时器中断使能之后，因为要产生中断，必不可少的要设置 NVIC 相关寄存器，设置中断优先级。之前多次讲解到用 NVIC_Init 函数实现中断优先级的设置，这里就不重复讲解。

5） 允许 TIM3 工作，也就是使能 TIM3。

光配置好定时器还不行，没有开启定时器，照样不能用。我们在配置完后要开启定时器，通过 TIM3_CR1 的 CEN 位来设置。在固件库里面使能定时器的函数是通过 TIM_Cmd 函数来实现的：

void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)

这个函数非常简单，比如我们要使能定时器 3，方法为：

TIM_Cmd(TIM3, ENABLE); //使能 TIMx 外设

6） 编写中断服务函数。

在最后，还是要编写定时器中断服务函数，通过该函数来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，我们这里使用的是更新（溢出）中断，所以在状态寄存器 SR 的最低位。在处理完中断之后应该向 TIM3_SR 的最低位写 0，来清除该中断标志。在固件库函数里面， 用来读取中断状态寄存器的值判断中断类型的函数是：

ITStatus TIM_GetITStatus(TIM_TypeDef* TIMx, uint16_t)

该函数的作用是，判断定时器 TIMx 的中断类型 TIM_IT 是否发生中断。 比如，我们要判断定时器 3 是否发生更新（溢出）中断，方法为：

```
if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET)
{
}
```

固件库中清除中断标志位的函数是：

void TIM_ClearITPendingBit(TIM_TypeDef* TIMx, uint16_t TIM_IT)

该函数的作用是，清除定时器 TIMx 的中断 TIM_IT 标志位。 使用起来非常简单，比如我们在

TIM3 的溢出中断发生后，我们要清除中断标志位，方法是：

TIM_ClearITPendingBit(TIM3, TIM_IT_Update);

这里需要说明一下，固件库还提供了两个函数用来判断定时器状态以及清除定时器状态标志位的函数 TIM_GetFlagStatus 和 TIM_ClearFlag，他们的作用和前面两个函数的作用类似。只是在 TIM_GetITStatus 函数中会先判断这种中断是否使能，使能了才去判断中断标志位，而 TIM_GetFlagStatus 直接用来判断状态标志位。通过以上几个步骤，我们就可以达到我们的目的了，使用通用定时器的的更新中断定时 1 秒控制 LED 逐个点亮。

1.3 例程程序

1) 初始化函数

```
/*************************************************************************
***********
* 函 数 名          : time_init
* 函数功能          : 定时器 3 端口初始化函数
* 输    入          : 无
* 输    出          : 无
*************************************************************************
**********/
void time_init()
{
   TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;    //声明一个结构体变量，用来初始化 GPIO

   NVIC_InitTypeDef NVIC_InitStructure;

   /* 开启定时器 3 时钟 */
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);
```

TIM_ClearITPendingBit(TIM3,TIM_IT_Update);//清除 TIMx 的中断待处理位:TIM 中断源

TIM_TimeBaseInitStructure.TIM_Period = 2000;//设置自动重装载寄存器周期的值

TIM_TimeBaseInitStructure.TIM_Prescaler = 35999;//设置用来作为 TIMx 时钟频率预分频值，100Khz 计数频率

TIM_TimeBaseInitStructure.TIM_ClockDivision = 0; //设置时钟分割:TDTS = Tck_tim

TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;//TIM 向上计数模式

TIM_TimeBaseInit(TIM3,&TIM_TimeBaseInitStructure);

TIM_Cmd(TIM3,ENABLE); //使能或者失能 TIMx 外设

/* 设置中断参数，并打开中断 */

TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE); //使能或者失能指定的 TIM 中断

/* 设置 NVIC 参数 */

NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

NVIC_InitStructure.NVIC_IRQChannel=TIM3_IRQn; //打开 TIM3_IRQn 的全局中断

NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=0; //抢占优先级为 0

NVIC_InitStructure.NVIC_IRQChannelSubPriority=1; //响应优先级为 1

NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //使能

NVIC_Init(&NVIC_InitStructure);

}

2) 定时器中断函数

```
void TIM3_IRQHandler()  //定时器 3 中断函数
{
    static u8 i=0;
```

```
    TIM_ClearITPendingBit(TIM3,TIM_IT_Update);
    GPIO_Write(GPIOC,(u16)~(0x01<<i++));
    if(i==8)i=0;
}
```

程序设计中我们所有中断函数都保存在 stm32f10x_it.c 中。

3) 主函数很简单，这里就贴出来了，大家参考例程就可以了。