STM32 入门系列教程

定时器与蜂鸣器

Revision 0.01

(2010-04-24)

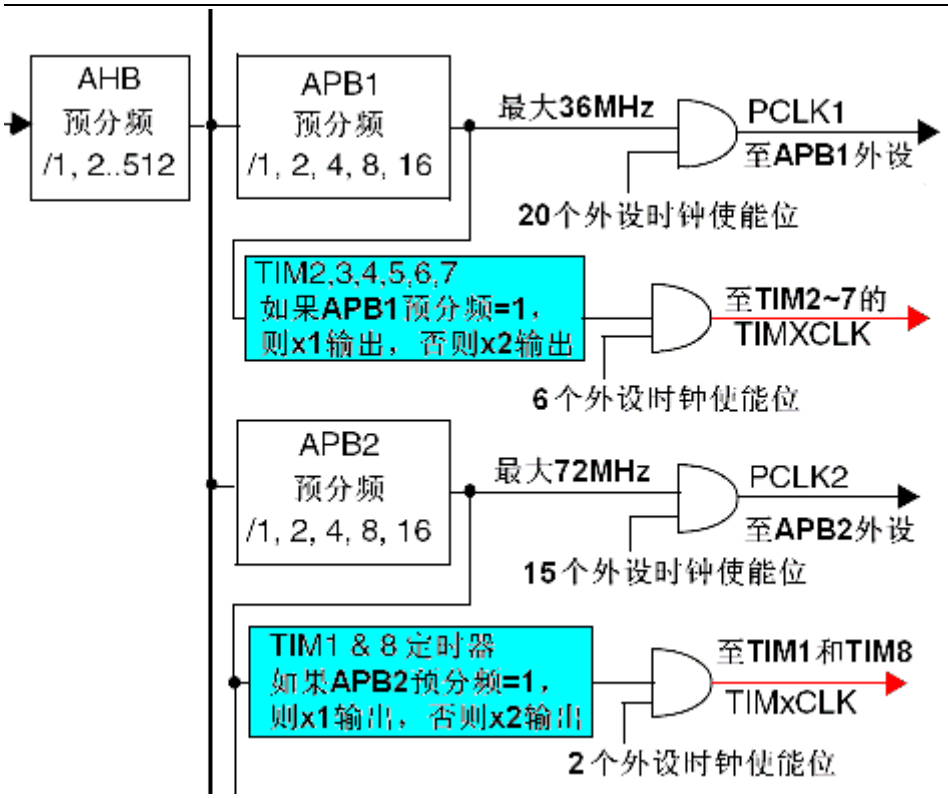
前一期教程已经详细讲述了 STM32 中断编程，本来不想再讲述定时器。因为定时器自然也是用到中断处理。但一想，既然作为入门系列教程，就应该具备完整性。实际上，笔者在网上搜了一下，发现仍然有许多网友卡在定时器这里，因此有必要专门列出一期教程讲述定时器。

相信您一定学习过单片机的定时器。没错~！STM32 系列的 CPU 定时器与单片机定时器操作类似。只要去配置自动装载寄存器、时钟预分频、溢出方式（向上溢出还是向下溢出）等等。当然 STM32 寄存器比较复杂，远不止这些，本系列教程主要是入门型，如果您希望深入学习 STM32 定时器，我们在论坛上传了一篇比较不错的文章，好像叫《STM32 入门篇之通用定时器彻底研究》，作者不详，网址如下：http://www.arm79.com/read.php?tid=1977。

一、STM32 通用定时器原理

STM32 系列的 CPU，有多达 8 个定时器，其中 TIM1 和 TIM8 是能够产生三对 PWM 互补输出的高级定时器，常用于三相电机的驱动，它们的时钟由 APB2 的输出产生。其它 6 个为普通定时器，时钟由 APB1 的输出产生。

下图是 STM32 参考手册上时钟分配图中，有关定时器时钟部分的截图：



实际上 STM32 的 CPU 文档给出的图与这个图略有区别。但是我们还是想研究这个图。原因是这个图对我们思路的理解比较有帮助。从图中可以看出，定时器的时钟不是直接来自 APB1 或 APB2，而是来自于输入为 APB1 或 APB2 的一个倍频器，图中的蓝色部分。

下面以通用定时器 2 的时钟说明这个倍频器的作用：当 APB1 的预分频系数为 1 时，这个倍频器不起作用，定时器的时钟频率等于 APB1 的频率；当 APB1 的预分频系数为其它数值(即预分频系数为 2、4、8 或 16)时，这个倍频器起作用，定时器的时钟频率等于 APB1 的频率两倍。

可能有同学还是有点不理解，OK，我们举一个例子说明。假定 AHB=36MHz，因为 APB1 允许的最大频率为 36MHz，所以 APB1 的预分频系数可以取任意数值；当预分频系数=1 时，APB1=36MHz，TIM2~7 的时钟频率=36MHz(倍频器不起作用)；当预分频系数=2 时，APB1=18MHz，在倍频器的作用下，TIM2~7 的时钟频率=36MHz。

有人会问，既然需要 TIM2~7 的时钟频率=36MHz，为什么不直接取 APB1 的预分频系数=1？答案是：APB1 不但要为 TIM2~7 提供时钟，而且还要为其它外设提供时钟；设置这个倍频器可以在保证其它外设使用较低时钟频率时，TIM2~7 仍能得到较高的时钟频率。

再举个例子：当 AHB=72MHz 时，APB1 的预分频系数必须大于 2，因为 APB1 的最大频率只能为 36MHz。如果 APB1 的预分频系数=2，则因为这个倍频器，TIM2~7 仍然能够得到 72MHz 的时钟频率。能够使用更高的时钟频率，无疑提高了定时器的分辨率，这也正是设计这个倍频器的初衷。

二、STM32 通用定时器编程

实际上，一开始笔者就提到，定时器编程，就是中断的编程。因为使用定时器必定要使用到中断。由于之前已经详细讲述过中断编程，因此本期部分代码的解释会简略讲述，您可以参考芯达 STM32 入门系列配套教程《初试 STM32 中断》。

步骤一　系统配置 SystemInit();，包括时钟 RCC 的配置，倍频到 72MHZ。

步骤二　GPIO 的配置，使用函数为 GPIO_Config();，该函数的实现如下：

```
void GPIO_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOD
                       | RCC_APB2Periph_AFIO, ENABLE);
/**
 *  LED1 -> PB8      ,   LED2 -> PB9 , LED3 -> PE0 , LED4 -> PE1
 */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 |GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 |GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
```

```
  GPIO_Init(GPIOE, &GPIO_InitStructure);

/**
 *  LED1-4-OFF
 */
  GPIO_SetBits(GPIOB , GPIO_Pin_8);
  GPIO_SetBits(GPIOB , GPIO_Pin_9);
  GPIO_SetBits(GPIOE , GPIO_Pin_0);
  GPIO_SetBits(GPIOE , GPIO_Pin_1);
}
```

实际上定时器的讲解，不需要配置 GPIO 的引脚，只是我们在定时器实验中，使用用户 LED 灯来做实验，所以需要配置对应的用户指示灯 LED。最后一步需要把 LED1-4 关掉，以使得实验效果更明显。

步骤三　嵌套中断控制器的配置，我们照样使用函数 NVIC_Config();，只是初始化的过程略有不同。详细解释请参考教程《初试 STM32 中断》。这里我们也把函数实现列出来：

```
void NVIC_Config(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQChannel;      //通道
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;//
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;    //
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

从以上函数实现来看，实际上只是改动了结构体成员 NVIC_IRQChannel 的

值，现在需要的通道是 TIM2 的通道，因此初始化值为 TIM2_IRQChannel。从这里也可以看出，这个函数实际上可以看做一个模型，可以拿去别的程序中改动后直接使用。

步骤四　定时器的初始化配置，使用 Timer_Config();。OK，关键部分出来了。我们来看下实现过程：

```
void Timer_Config(void)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2 , ENABLE);
    TIM_DeInit(TIM2);
    TIM_TimeBaseStructure.TIM_Period=2000;    //自动重装载寄存器的值
    TIM_TimeBaseStructure.TIM_Prescaler= (36000 - 1);        //时钟预分频数
    TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;    //采样分频
    TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
    TIM_ClearFlag(TIM2, TIM_FLAG_Update);                //清除溢出中断标志
    TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);
    TIM_Cmd(TIM2, ENABLE);                               /开启时钟
}
```

我们每个语句都来解释一下。首先我们想使用定时器，就必须使能定时器的时钟，这就是函数 RCC_APB1PeriphClockCmd();，通过它开启 RCC_APB1Periph_TIM2。

TIM_DeInit(TIM2); 该函数主要用于复位 TIM2 定时器，使之进入初始状态。然后我们对自动重装载寄存器赋值，TIM_Period 的大小实际上表示的是需要经过 TIM_Period 次计数后才会发生一次更新或中断。接下来需要设置时钟预分频数 TIM_Prescaler，这里有一个公式，我们举例来说明：例如时钟频率=72MHZ/(时钟预分频+1)。说明当前设置的这个 TIM_Prescaler，直接决定定时器的时钟频率。通俗点说，就是一秒钟能计数多少次。比如算出来的时钟频率是 2000，也就是一秒钟会计数 2000 次，而此时如果 TIM_Period 设置为 4000，即 4000 次计数后

就会中断一次。由于时钟频率是一秒钟计数 2000 次，因此只要 2 秒钟，就会中断一次。

再往后的代码，还有一个需要注意的，就是我们一般采用向上计数模式，即每次计数就会加 1，直到寄存器溢出发生中断为止。

最后别忘了，需要使能定时器!!

步骤五　编写中断服务程序。同样需要注意的，一进入中断服务程序，第一步要做的，就是清除掉中断标志位。由于我们使用的是向上溢出模式，因此使用的函数应该是：TIM_ClearITPendingBit(TIM2 , TIM_FLAG_Update);。芯达 STM32 开发板实现的中断服务程序如下：

```
void TIM2_IRQHandler(void)
{
    if ( TIM_GetITStatus(TIM2 , TIM_IT_Update) != RESET ) {
        TIM_ClearITPendingBit(TIM2 , TIM_FLAG_Update);
        switch(state){
            case 0:
              /*====LED1-ON=======*/
                GPIO_ResetBits(GPIOB , GPIO_Pin_8);
              GPIO_SetBits(GPIOB, GPIO_Pin_9);
              GPIO_SetBits(GPIOE, GPIO_Pin_0);
              GPIO_SetBits(GPIOE, GPIO_Pin_1);
                break;
            case 1:
                GPIO_SetBits(GPIOB , GPIO_Pin_8);
              GPIO_ResetBits(GPIOB, GPIO_Pin_9);
              GPIO_SetBits(GPIOE, GPIO_Pin_0);
              GPIO_SetBits(GPIOE, GPIO_Pin_1);
                break;
            case 2:
```

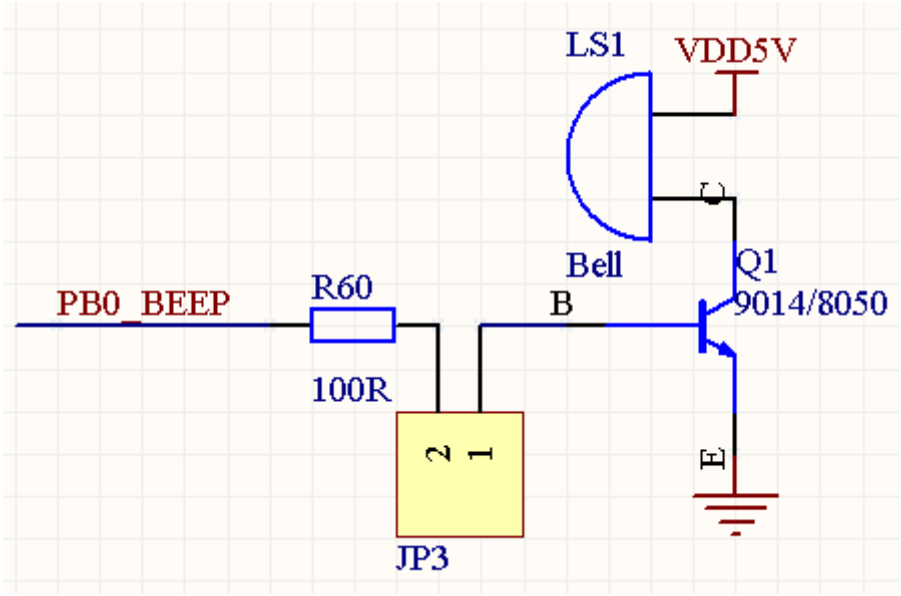
```
                    GPIO_SetBits(GPIOB , GPIO_Pin_8);
                GPIO_SetBits(GPIOB, GPIO_Pin_9);
                GPIO_ResetBits(GPIOE, GPIO_Pin_0);
                GPIO_SetBits(GPIOE, GPIO_Pin_1);
                    break;
            case 3:
                    GPIO_SetBits(GPIOB , GPIO_Pin_8);
                GPIO_SetBits(GPIOB, GPIO_Pin_9);
                GPIO_SetBits(GPIOE, GPIO_Pin_0);
                GPIO_ResetBits(GPIOE, GPIO_Pin_1);
                    break;
            default:
                    break;
        }
        if(++state >= 4){
            state = 0;
        }
    }
}
```

使用 switch(state)语句，使得芯达 STM32 开发板上的四个用户指示灯按顺序闪烁。当然，您也可以编写其他的中断服务程序。总之，在中断服务程序里，可以发挥您的聪明才智，编写出自己得意的代码。

最后需要说明的是，本期教程叫做《定时器与蜂鸣器》。有的同学可能会问，怎么光看定时器，没有蜂鸣器呢？呵呵，实际上，蜂鸣器的控制超级简单，就相当于一个 LED 的控制。在芯达 STM32 开发板上，我们使用 PB0(根据版本的不同，也有可能使用 PC5)连接蜂鸣器，那么我们可以配置下 PB0(根据版本的不同，也有可能使用 PC5)为输出推挽模式，然后使用如下代码：

GPIO_SetBits(GPIOB, GPIO_Pin_0);

OK，蜂鸣器在响啦~！就这么简单。看看电路图吧！



这里有一个 JP3，它存在的意义是：如果在别的系统实现中，使用到 PB0(根据版本的不同，也有可能使用 PC5)引脚，可能会使蜂鸣器意外的响起来，因此加上一个 JP3 跳线，当进行别的实验的时候，可以拔掉短路帽。

至此，STM32 的外部中断教程编写结束。如果您对串口操作还有不明白的地方，请直接到我们的合作网站：ARM 技术交流网 www.arm79.com，进行讨论。我们将会尽快给您做出答复。感谢您阅读本教程。

注：本期教程部分内容参考如下网址，对该作者表示感谢！

http://article.ednchina.com/Other/20090330085645.htm

附：

福州芯达工作室简介

福州芯达工作室成立于 2009 年 9 月，我们专注于嵌入式产品的研发与推广，目前芯达产品涉及 ARM9 系列、STM32 系列。

芯达团队成员均硕士研究生毕业，具有一定研发实力。我们的愿景在于把福州芯达打造成国内一流的嵌入式品牌。或许我们现在做的还不够，但是我们真的努力在做，希望通过我们的努力，能够在您学习和使用芯达产品的过程中带来或多或少的帮助。

这是芯达为了配合 STM32 开发板而推出的入门系列教程。如果您在看了我们的教程后，理清了思路，我们都会倍感欣慰！让我们一起学习，共同进步，在征服嵌入式领域的道路上风雨同行！

官方网站：http://www.arm79.com/

官方淘宝：http://shop36353570.taobao.com/