例程十　定时器 TIM3

定时器的使用是很重要的，可以说是单片机的灵魂来的，它的功能非常强大，单片机可以利用它来做很多事情。这个例程主要解析一下 STM8S 里面定时器 TIM2 的主要功能特点。TIM2 是个通用定时器，和 TIM3 是一样的。

通用定时器由带有可编程预分频器的16位自动装载计数器构成。

它适用于多种场合，包括：

- 基本的定时
- 测量输入信号的脉冲长度(输入捕获)
- 产生输出波形(输出比较，PWM和单脉冲)
- 与其他定时器或外部信号同步(外部时钟，复位，触发和使能信号)(仅针对带有TIM5的芯片)

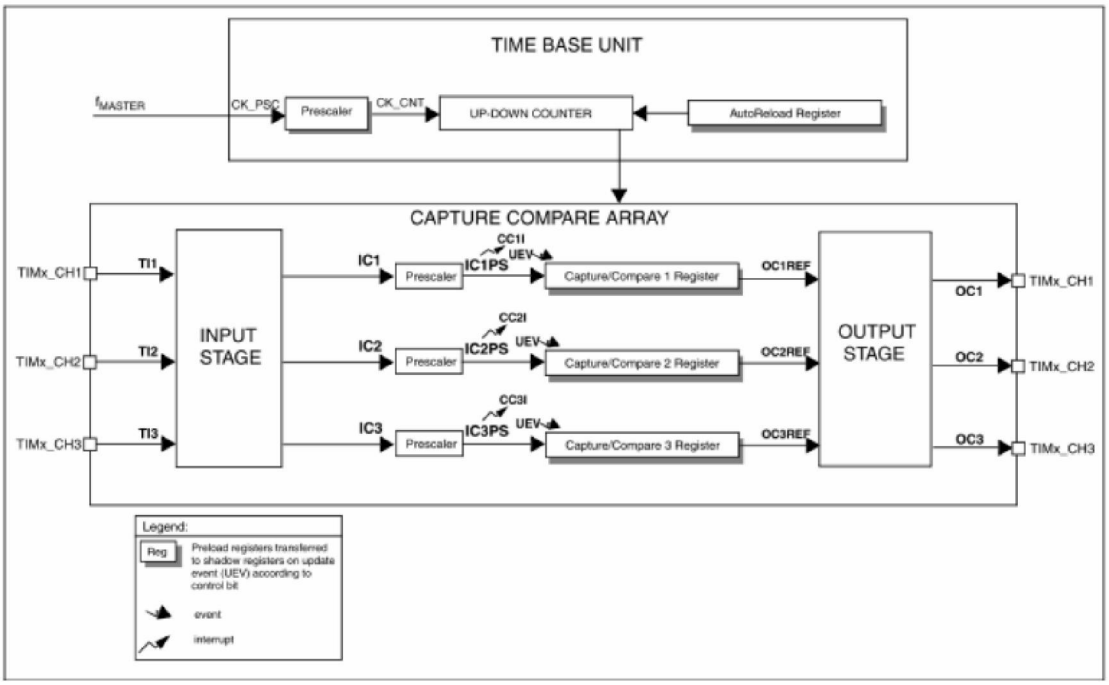
定时器可由内部时钟驱动。

TIM2/TIM3的主要功能

TIM2/TIM3的功能包括：

- 16位向上计数和自动装载计数器
- 4位可编程(可以实时修改的)预分频器，计数器时钟频率的分频系数为1～32768之间的2的幂
- 3个独立通道：
  - 输入捕获
  - 输出比较
  - PWM 生成(边缘对齐模式)
  - 单脉冲模式输出
- 如下事件发生时产生中断：
  - 更新：计数器向上溢出，计数器初始化(通过软件)
  - 输入捕获
  - 输出比较

图79　TIM2/TIM3框图



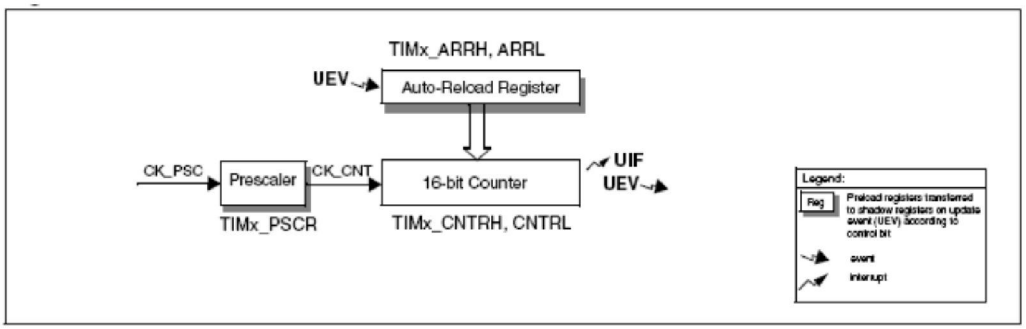
时基单元包含：

- 16位向上计数器
- 预分频器
- 16位自动装载寄存器

没有重复寄存器。

计数器使用内部时钟(f_{MASTER})，它由CK_PSC提供，并经过预分频器分频产生计数器时钟CK_CNT。

图81　时基单元



预分频器

预分频器的实现：

- 预分频器基于4位寄存器控制的16位计数器，由于寄存器带有缓冲器因此可以随时修改预分频的数值。计数器可以取值为1到32768之间的2的幂进行分频。

计数器时钟频率的计算公式：

$f_{CK\_CNT} = f_{CK\_PSC}/2^{(PSCR[3:0])}$

预分频器的值由预装载寄存器写入。一旦写入预装载寄存器的LS字节时，带有当前使用值的影子寄存器就被写入了新的值。

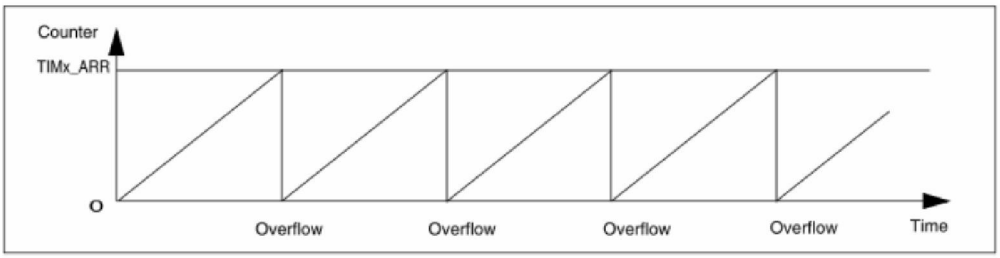
新的预分频值在下一个周期时生效(在下一个更新事件之后)。

对TIMx_PSCR寄存器的读操作通过预装载寄存器实现，因此可以随时读取不受限制。

计数模式

在向上计数模式中，计数器从0计数到用户定义的比较值(TIMx_ARR寄存器的值)，然后重新从0开始计数并产生一个计数器溢出事件，同时，如果TIM1_CR1寄存器的UDIS位是0，将会产生一个更新事件(UEV)。*0*描述了向上计数模式。

图31　向上计数模式的计数器



置位TIMx_EGR寄存器的UG位(通过软件方式或者使用从模式控制器)也同样可以产生一个更新事件。

使用软件置位TIMx_CR1寄存器的UDIS位，可以禁止更新事件，这样可以避免在更新预装载寄存器时更新影子寄存器。在UDIS位被清除之前，将不产生更新事件。但是在应该产生更新事件时，计数器仍会被清0，同时预分频器的计数也被清0(但预分频器的数值不变)。此外，如果设置了TIMx_CR1寄存器中的URS位(选择更新请求)，设置UG位将产生一个更新事件UEV，但硬件不设置UIF标志(即不产生中断请求)。这是为了避免在捕获模式下清除计数器时，同时产生更新和捕获中断。

当发生一个更新事件时，所有的寄存器都被更新，硬件同时(依据URS位)设置更新标志位(TIMx_SR寄存器的UIF位)：

自动装载影子寄存器被重新置入预装载寄存器的值(TIMx_ARR)。

预分频器的缓存器被置入预装载寄存器的值(TIMx_PSC寄存器的内容)。

下图给出一些例子，说明当TIMx_ARR=0x36时，计数器在不同时钟频率下的动作。

*图32*的预分频为2，因此计数器的时钟(CK_CNT)频率是预分频时钟(CK_PSC)频率的一半。

*图32*禁止了自动装载功能(ARPE=0)，所以在计数器达到0x36时，计数器溢出，影子寄存器立刻被更新，同时产生一个更新事件。

图32　当ARPE=0(ARR不预装载)，预分频为2时的计数器更新。

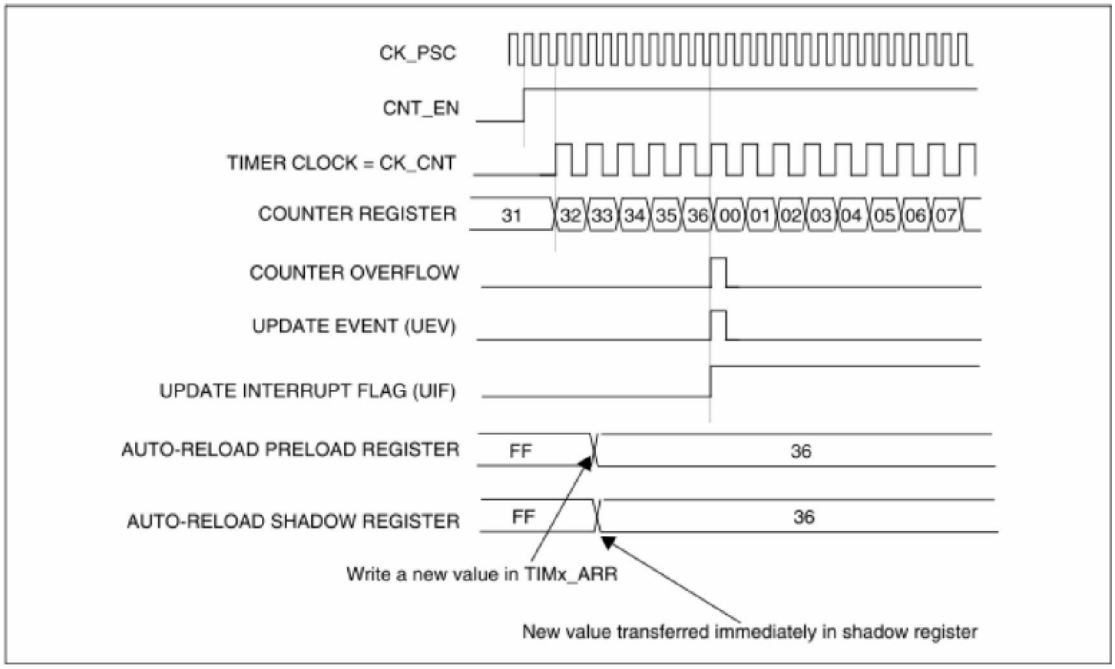
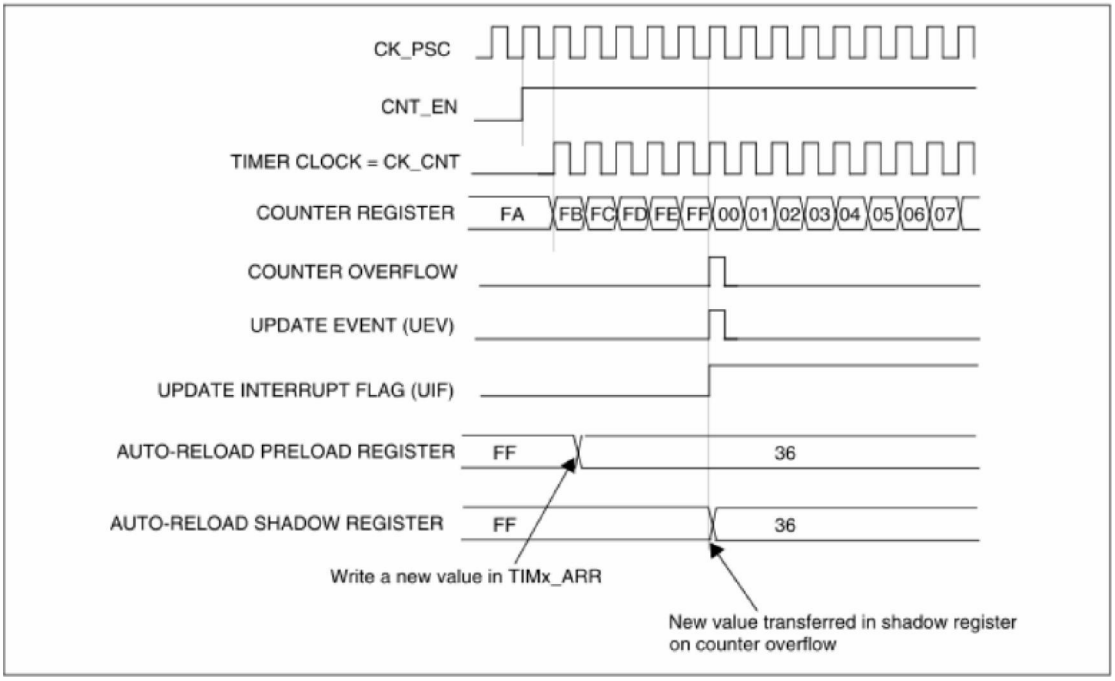


*图33*的预分频为1，因此CK_CNT的频率与CK_PSC一致。

*图33*使能了自动重载(ARPE=1)，所以在计数器达到0xFF产生溢出。0x36将在溢出时被写入，同时产生一个更新事件。

图33　ARPE=1(TIM1_ARR预装载)时的计数器更新



计数模式就介绍一种向上技术模式，其他模式可以参考 STM8 寄存器.pdf 文档第 115 页至 118 页

捕获/比较通道

输入部分

请参考*17.5*。

如*图82*输入部分框图所示，定时器带有两个输入通道，通道1在内部链接到比较器。

图82　输入部分框图

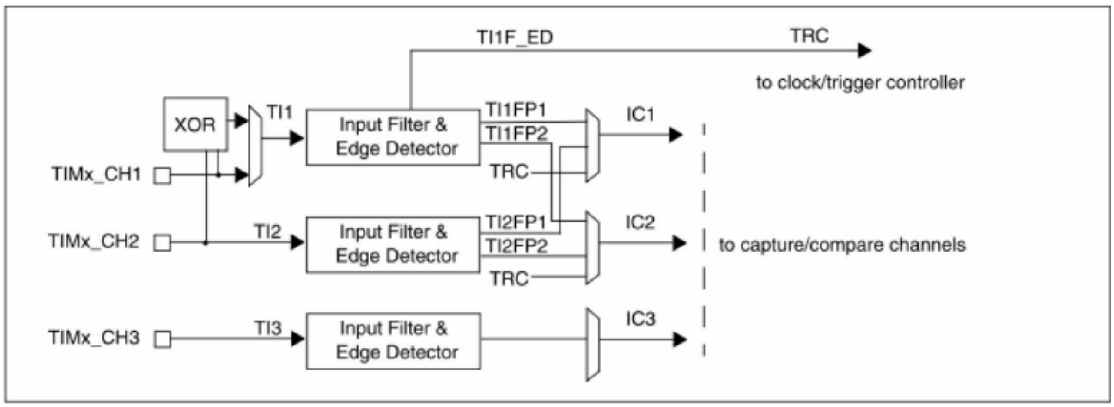


图83　　TIM2通道1的输入部分框图

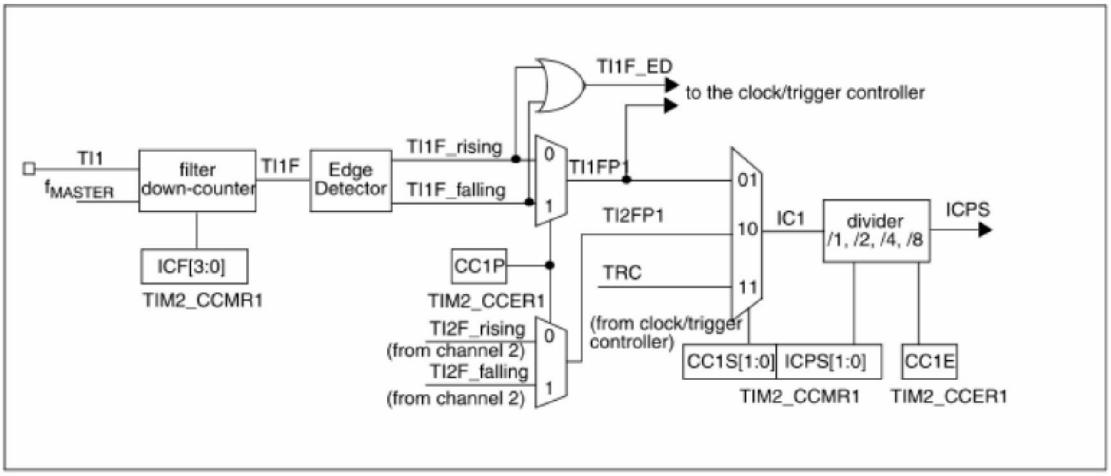


图84　　输出部分框图

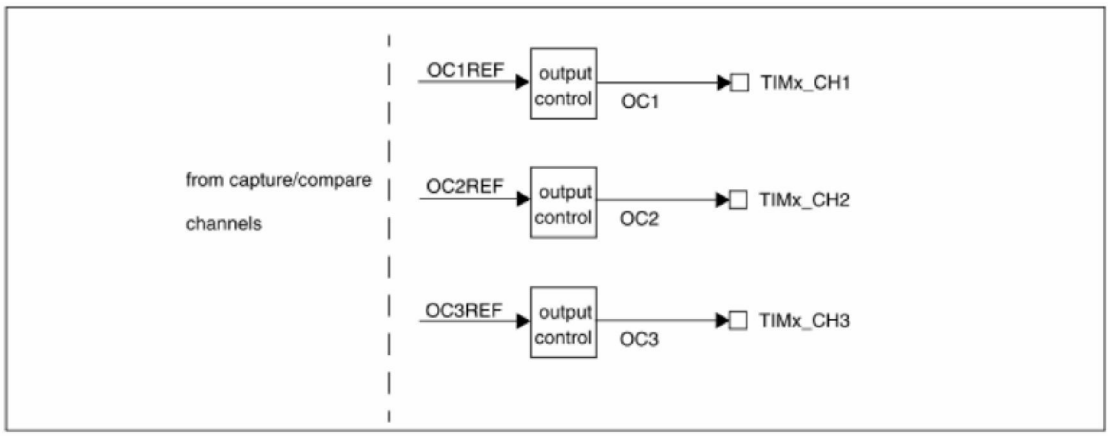
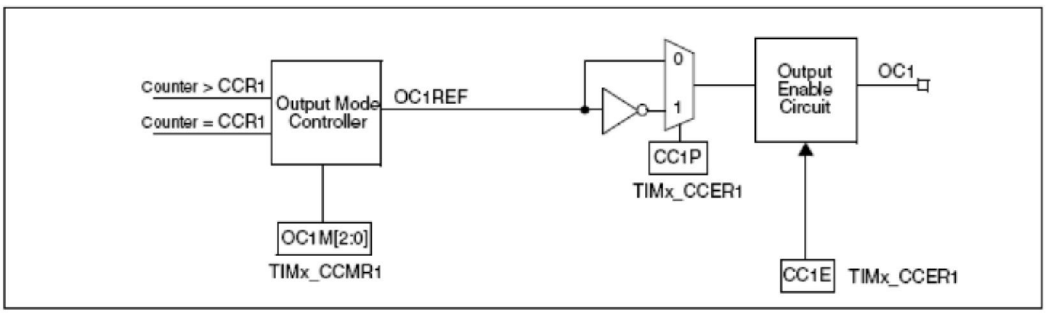


图85　　通道1的输出部分框图



通道输出的具体的原理可以参考 STM8 寄存器.pdf 文档第 133 页至 138 页

中断

通用定时器包括4个中断源：

- 捕获/比较3中断
- 捕获/比较2中断
- 捕获/比较1中断
- 更新中断

在使用中断功能时，需要先设置TIMx_IER寄存器的CC3IE位或CC2IE位或CC1IE位使能中断请求。

通过软件设置TIMx_EGR寄存器的相应位也能产生不同的中断源。

了解通用定时器 TIM3，现在关键是学会怎么去用，怎样去设置定时器的寄存器。好的，跟以前一样从主函数看起

```
int main(void)
{
  /* Infinite loop */

  /*设置内部高速时钟16M为主时钟*/
  CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
   /*!<Set High speed internal clock */
  #if TIM3_Mode == BaseTime
   Tim3_Init();
   LED_Init();
   SetLedOFF();
   __enable_interrupt();
   #elif TIM3_Mode == PWM
   PWM_Init();
  #endif
  while (1)
  {
    /*  添加你的代码  */
    #if TIM3_Mode ==BaseTime
    LED_Display();
    #elif TIM3_Mode==PWM
    TestPWM_LED();
    #endif

  }
}
```

看过这主函数，是否觉得这和前面的例程写得有点不一样呢？是的，这个 TIM3 的功能强大，功能比较多，在我的例程中，我写了几个常用和比较重要的功能，所以才有了预定义的方法，这种形式对调试程序比较重要，呆会在详细解释。这种#if …#endif 或#if…#elif…#endif。这就预定义，只有后面为真，编译器才会编译那些代码，否则就不会编译。大家学过 C 语言的都是知道这种编程方法，希望大家能领会这种方法。下面还是主要看看的我们 STM8S 的 TIM3 的各个功能的初始话。在这个例程中，介绍了定时器的基本定时，主要利用基本定时来实现最精准的 1ms 的延迟函数和 TIM3 的 PWM 的功能。

```
#if TIM3_Mode == BaseTime
 Tim3_Init();
 LED_Init();
 SetLedOFF();
 __enable_interrupt();
 #elif TIM3_Mode == PWM
 PWM_Init();
#endif
```

这个就是初始化，如果是 TIM3_Mode==BaseTime 的话就是利用 TIM3 的定时器功能。看看里面的初始化吧。

```
void Tim3_Init(void)
{
  TIM3_DeInit();
  TIM3_TimeBaseInit( TIM3_PRESCALER_16 ,999);
  TIM3_PrescalerConfig(TIM3_PRESCALER_16,TIM3_PSCRELOADMODE_IMMEDIATE);
  TIM3_ARRPreloadConfig(ENABLE);
  TIM3_ITConfig(TIM3_IT_UPDATE , ENABLE);
  TIM3_Cmd(ENABLE);
}
```

这个初始化函数定义了 TIM3 的计数频率为主时钟的 16 分频，就是 1M,当从 0 计数到 999 就是 1ms,但计数到 999 是马上又自动重装系数，又从 0 开始计数，当计数到 999，会产生一个中断。
__enable_interrupt();这个就是开总中断
其他的 2 个初始化就 LED 的初始化，前面已经讲过了，这里就不再重复了。
下面看看中断服务子程序

```
__interrupt void TIM3_UPD_OVF_BRK_IRQHandler(void)
{
  TimingDelay_Decrement();
  TIM3_ClearITPendingBit(TIM3_IT_UPDATE);
}
```

这个 TIM3 计数溢出中断的中断号为 0xF,里面的函数是放中断后要处理的语句，最后要清楚标志位，TIM3_ClearITPendingBit(TIM3_IT_UPDATE);否则就跳不出中断。这里主要是实现 1ms 的延迟函数,所以只调用了 TimingDelay_Decrement();这个函数。看看它的函数原型

```
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
     {
        TimingDelay--;
      }
}
```

就是每近一次中断 TimingDelay 就减 1，TimingDelay 定义为静态全局变量，这样的话就可以实现 1ms 精准延迟。下面看看延迟函数的原型

```
void delay_ms( u16 nTime)
{
  TimingDelay = nTime;

  while(TimingDelay != 0);
}
```

整个延迟思想是 TimingDelay 不等于 0 的话，就在这等到 TimingDelay 为 0，TimingDelay 减 1 是进去中断子程序，TimingDelay 是全局静态变量。这个功能使用流水灯 LED 来演示，所以大家在主函数的文件头部那里这样定义就行

```
#define TIM3_Mode   1
#define PWM         0
#define BaseTime    1
```

这个就是表示 #if TIM3_Mode == BaseTime 当你编译的时候只编译定

时器这个功能的代码。编译后下载到风驰电子 STM8 开发板的时候就会见到 LED 不断的流，每隔 100ms。

下面接着写 TIM3 的 PWM 功能

```
#elif TIM3_Mode == PWM
PWM_Init();
```

```
void PWM_Init(void)
{

  #if TIM3_Channel==TIM3_Channel1
    /*TIM3 Frequency = TIM3 counter clock/(ARR + 1) */
    TIM3_TimeBaseInit(TIM3_PRESCALER_16, 499);
    /*TIM3 Frequency=16M/16/(499+1)=2K*/
    /* PWM1 Mode configuration: Channel1
    TIM3 Channel1 duty cycle = [TIM3_CCR1/(TIM3_ARR + 1)] * 100 = 50%*/
    TIM3_OC1Init(TIM3_OCMODE_PWM1, TIM3_OUTPUTSTATE_ENABLE,250, TIM3_OCPOLARITY_HIGH);
    TIM3_OC1PreloadConfig(ENABLE);

  #elif TIM3_Channel==TIM3_Channel2
    TIM3_TimeBaseInit(TIM3_PRESCALER_16, 999);
    TIM3_OC2Init(TIM3_OCMODE_PWM2, TIM3_OUTPUTSTATE_ENABLE,500, TIM3_OCPOLARITY_HIGH);
    TIM3_OC2PreloadConfig(ENABLE);

  #elif TIM3_Channel==TIM3_Channel3
    TIM3_TimeBaseInit(TIM3_PRESCALER_16, 499);
    TIM3_OC3Init(TIM3_OCMODE_PWM3, TIM3_OUTPUTSTATE_ENABLE,250, TIM3_OCPOLARITY_HIGH);
    TIM3_OC3PreloadConfig(ENABLE);
  #endif

  TIM3_Cmd(ENABLE);

}
```

这个初始化函数定义了 3 个通道的初始化，初始化里面最重要的是设置了 PWM 的频率和占空比，对于怎么设，大家看上面的例程代码和注释就清楚了。在官方库里面没有单独对 PWM 设置频率和占空比，所以我就封装了 2 个这样的函数，大家来看看。

```
void SetTIM3_PWM_Frequency(uint16_t TIM3_Period)
{
      /* Set the Autoreload value */

    TIM3->ARRH = (uint8_t)(TIM3_Period >> 8);
    TIM3->ARRL = (uint8_t)(TIM3_Period);
}
```

大家看到这个函数是否和官方库封装的很相似呢？这是个很标准的函数，其实大家要培养成良好的编程习惯，这样让人感觉你的编程是很规范的。这个函数是实现 PWM 的频率的设置。频率的设置还要跟时钟有关系。

TIM3 Frequency = TIM3 counter clock/(ARR + 1)= TIM3 counter clock/(TIM3_Period+ 1)。

```
void SetTIM3_PWM_DutyCycle( uint16_t TIM3_Pulse)
{


  #if TIM3_Channel==TIM3_Channel1
    /* Set the Pulse value */
    TIM3->CCR1H = (uint8_t)(TIM3_Pulse >> 8);
    TIM3->CCR1L = (uint8_t)(TIM3_Pulse);

  #elif TIM3_Channel==TIM3_Channel2
    TIM3->CCR2H = (uint8_t)(TIM3_Pulse >> 8);
    TIM3->CCR2L = (uint8_t)(TIM3_Pulse);

  #elif TIM3_Channel==TIM3_Channel3
    TIM3->CCR3H = (uint8_t)(TIM3_Pulse >> 8);
    TIM3->CCR3L = (uint8_t)(TIM3_Pulse);

  #endif
}
```

这个函数设置了 3 个通道的占空比，这个函数也是用预定义来实现了，至于大家要用哪个通道的就在 pwm.h 的头文件里面改下宏定义

```
#define TIM3_Channel    1
#define TIM3_Channel1   1
#define TIM3_Channel2   2
#define TIM3_Channel3   3
```

TIM3 Channel duty cycle = [TIM3_CCRx/(TIM3_ARR + 1)]= [TIM3_Period/(TIM3_ARR + 1)] 所以先设好频率，才能设置占空比。

TIM3 的通道 1 在风驰电子 STM8 开发板链接到 LED，可以利用不同的占空比实现 LED 的不同的亮度。

```
#define TIM3_Mode   0
#define PWM         0
#define BaseTime    1
```

```
#define TIM3_Channel    1
#define TIM3_Channel1   1
#define TIM3_Channel2   2
#define TIM3_Channel3   3
```

当是这样设置的话，就是实现 PWM 不同的占空比，对应的是 LED 的亮度不同的变化。

风驰电子祝您学习愉快～～～!!!!!