Homework for lecture 6

(1)Write out the registers those are used (configurated or read) and related with the Basic TIM by the project (LED_BasicTImer) and their meanings one by one

(2)Program to realize: PA4 output 100Hz square wave

(3)Complete the project that is to measure the frequency of the input signal based on the unfinished project on BB. Sort out the the code flow chart.

	定时器	计数器分辨率	计数器类型	预分频系数	产生DMA	捕获/比较通道	互补输出
高级定时器	TIM1	16位	向上/向下	1~65535	可以	4	有
	TIM8	16位	向上/向下	1~65535	可以	4	有
通用定时器	TIM2	16位	向上/向下	1~65535	可以	4	没有
	TIM3	16位	向上/向下	1~65535	可以	4	没有
	TIM4	16位	向上/向下	1~65535	可以	4	没有
	TIM5	16位	向上/向下	1~65535	可以	4	没有
基本定时器	TIM6	16位	向上	1~65535	可以	0	没有
	TIM7	16位	向上	1~65535	可以	0	没有

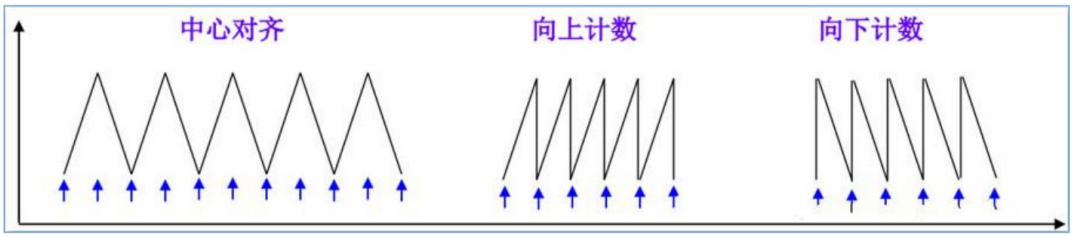


表 32-1 高级控制和通用定时器通道引脚分布

	高级定时器		通用定时器			
	TIM1	TIM8	TIM2	TIM5	TIM3	TIM4
CH1	PA8/PE9	PC6	PA0/PA15	PA0	PA6/PC6/PB4	PB6/PD12
CH1N	PB13/PA7/PE8	PA7				
CH2	PA9/PE11	PC7	PA1/PB3	PA1	PA7/PC7/PB5	PB7/PD13
CH2N	PB14/PB0/PE10	PB0				
CH3	PA10/PE13	PC8	PA2/PB10	PA2	PB0/PC8	PB8/PD14
CH3N	PB15/PB1/PE12	PB1				
CH4	PA11/PE14	PC9	PA3/PB11	PA3	PB1/PC9	PB9/PD15
ETR	PA12/PE7	PA0	PA0/PA15		PD2	PE0
BKIN	PB12/PA6/PE15	PA6				

Write out the registers those are used (configurated or read) and related with the Basic TIM by the project (LED_BasicTImer) and their meanings one by one

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);
使能定时器 TIM6 的时钟。
寄存器：RCC_APB1ENR（RCC APB1 peripheral clock enable register）。

TIM_TimeBaseStructure.TIM_Period = BASIC_TIM_Period;
设置自动重装载寄存器的值，即计数器计数到该值后会产生一个更新事件。
寄存器：TIM6_ARR（TIM6 Auto-Reload Register）。

TIM_TimeBaseStructure.TIM_Prescaler = BASIC_TIM_Prescaler;
设置时钟预分频数，用于控制计数器时钟的频率。
寄存器：TIM6_PSC（TIM6 Prescaler Register）。

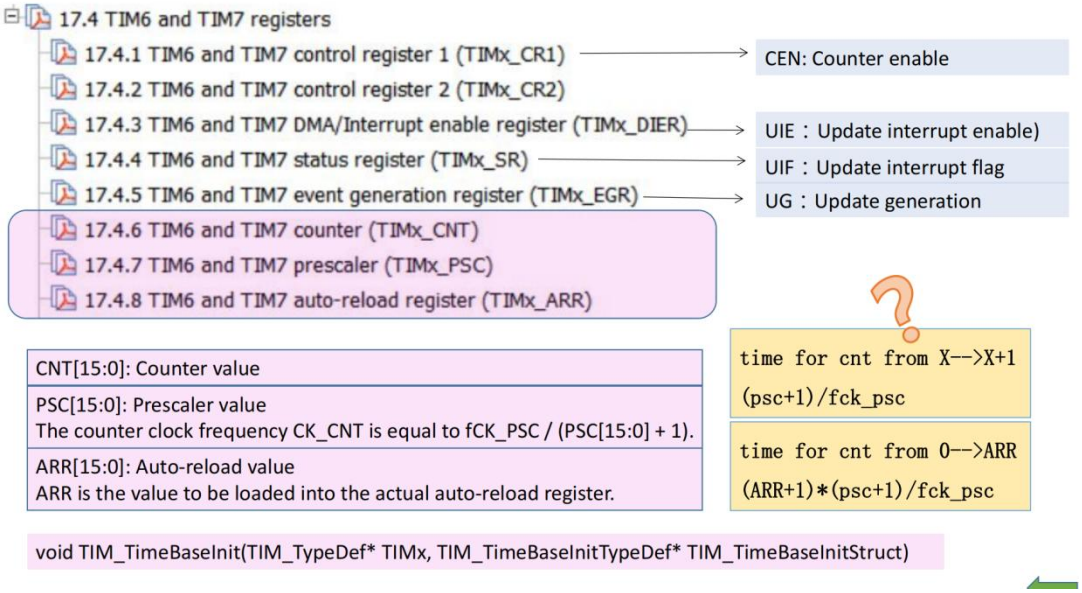
TIM_TimeBaseInit(TIM6, &TIM_TimeBaseStructure);
初始化定时器 TIM6 的基本配置。

NVIC_InitStructure.NVIC_IRQChannel = TIM6_IRQn;
设置中断向量为 TIM6_IRQn，即 TIM6 的中断。
寄存器：NVIC（Nested Vector Interrupt Controller）。
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
设置中断的主优先级。
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
设置中断的次优先级。
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
使能中断向量。

TIM_ClearFlag(TIM6, TIM_FLAG_Update);
清除计数器中断标志位，以确保在进入中断服务函数之前标志位被清除。
寄存器：TIM6_SR（TIM6 Status Register）。

TIM_ITConfig(TIM6, TIM_IT_Update, ENABLE);
开启定时器的更新中断。
寄存器：TIM6_DIER（TIM6 DMA/Interrupt Enable Register）。

TIM_Cmd(TIM6, ENABLE);
使能定时器 TIM6。
寄存器：TIM6_CR1（TIM6 Control Register 1）。



Program to realize: PA4 output 100Hz square wave

直接修改 LED_BasicTimer

```
while(1)
    {
            if( time == 5 )
            {
                time = 0;
                switch(i)
                {
                   case 0:LED_ON();i++;break;
                   case 1:LED_OFF();i=
                      0;break;

                }
            }
        }
```

100Hz 即 0.005s 变一次

然后偷偷修改宏定义

```
#define LED_GPIO_CLK     RCC_APB2Periph_GPIOA
#define LED_GPIO_Pin     GPIO_Pin_4
#define LED_GPIO_PORT    GPIOA
```

Complete the project that is to measure the frequency of the input signal based on the unfinished project on BB. Sort out the the code flow chart.

见 GeneralTImer_输入方波频率检测（finished）

Homework for lecture 7

(1) Write out the registers those are used (configurated or read) by the function TIM_ICInit() & TIMx_IRQHandler() and their meanings one by one

(2) Sort out the programme flow of TIMx_IRQHandler()

(3) Modify the project “TIM-通用定时器-脉宽测量” to measure the frequency of the input signal（TIM4-CH1）。

(4) Write out the registers those are used (configurated or read) by the function TIM_OCxInit() for General Tim and their meanings one by one;

(5) Modify the frequency and duty of output PWM in the project“通用定时器-输出 PWM”

(6) Read the project“通用定时器-PWM 呼吸灯-delay--示波器”, Sort out the programme flow of this problem: How to constantly and regularly change the duty. You can observe the output wave by Oscilloscope(Which GPIO?)

(7) Modify the project“通用定时器-PWM 呼吸灯-delay--示波器”， realize the breathing LED---Easy, change OCx-CH

(8) Modify the project“通用定时器-PWM 呼吸灯-delay--示波器”，use a TIM to realize timing instead of the dealy() function.

> 上述两个函数决定**定时时间**
>
> 定时频率=72M/（PSC+1）/(ARR+1)
>
> 例如：定时1秒，表示定时频率为1Hz，则PSC为7200，ARR为10000,其参数再均减1
>
> 因为预分频器与计数器都有1个数的偏差
>
> 如果PSC的值比较小，ARR的值比较大，就是表示是一个比较高的频率计比较多的数
>
> **注意：PSC，ARR的取值需要在0~65535之间**
>
> •计数器计数频率：CK_CNT = CK_PSC / (PSC + 1)
>
> •计数器溢出频率：CK_CNT_OV = CK_CNT / (ARR + 1)
>
> = CK_PSC / (PSC + 1) / (ARR + 1)

TIM_ETRClockMode2Config(TIM_TypeDef* TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TIM_ExtTRGPolarity, uint16_t ExtTRGFilter);

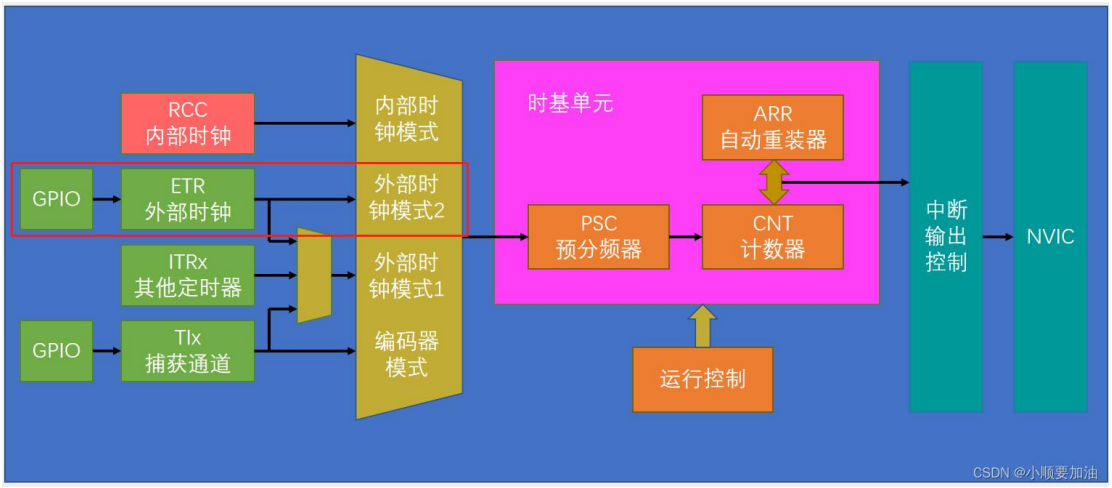
ETR 外部时钟模式2

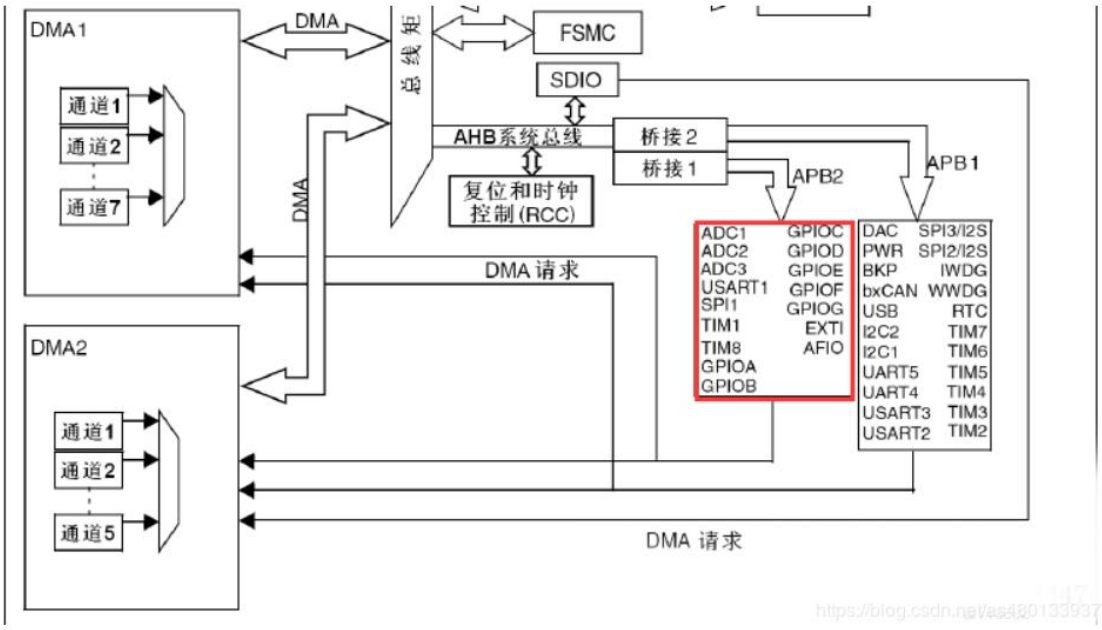
TIM_ExtTRGPrescaler:TIMx外部触发分频设置

TIM_ExtTRGPrescaler	描述
TIM_ExtTRGPSC_OFF	TIM ETRP 预分频 OFF
TIM_ExtTRGPSC_DIV2	TIM ETRP 频率除以 2
TIM_ExtTRGPSC_DIV4	TIM ETRP 频率除以 4
TIM_ExtTRGPSC_DIV8	TIM ETRP 频率除以 8

TIM_ExtTRGPolarity：TIMx外部触发极性设置

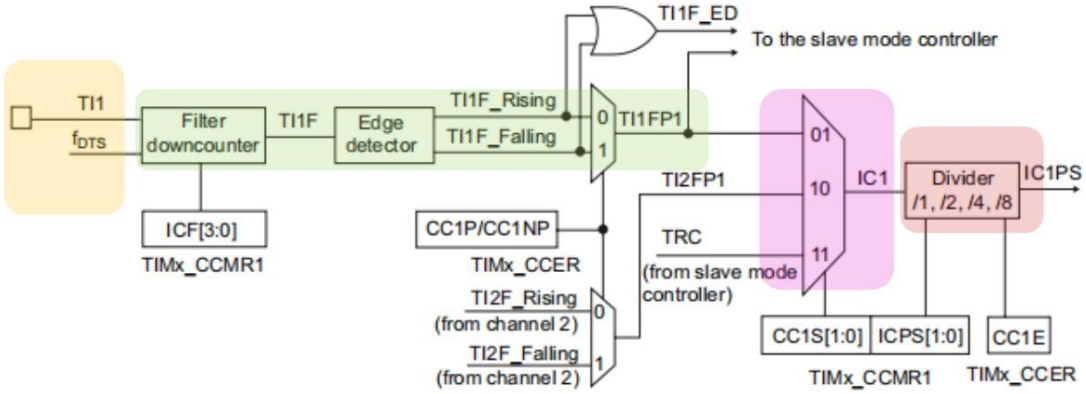
TIM_ExtTRGPolarity	描述
TIM_ExtTRGPolarity_Inverted	TIM 外部触发极性翻转：低电平或下降沿有效
TIM_ExtTRGPolarity_NonInverted	TIM 外部触发极性非翻转：高电平或上升沿有效





Write out the registers those are used (configurated or read) by the function TIM_ICInit() & TIMx_IRQHandler() and their meanings one by one

```
TIM_ICInitTypeDef TIM_ICInitStructure;
// 配置输入捕获的通道，需要根据具体的 GPIO 来配置
TIM_ICInitStructure.TIM_Channel = GENERAL_TIM_CHANNEL_x;
// 输入捕获信号的极性配置，TIMx_CCER 寄存器
TIM_ICInitStructure.TIM_ICPolarity = GENERAL_TIM_STRAT_ICPolarity;
// 输入通道和捕获通道的映射关系，有直连和非直连两种，TIMx_CCMR1 寄存器
TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
// 输入的需要被捕获的信号的分频系数，TIMx_CCER 寄存器
TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
// 输入的需要被捕获的信号的滤波系数，TIMx_CCMR1 寄存器
TIM_ICInitStructure.TIM_ICFilter = 0;
// 定时器输入捕获初始化
TIM_ICInit(GENERAL_TIM, &TIM_ICInitStructure);
```




```
void GENERAL_TIM_INT_FUN(void)
{
    // 当要被捕获的信号的周期大于定时器的最长定时时，定时器就会溢出，产生更新中断
    // 这个时候我们需要把这个最长的定时周期加到捕获信号的时间里面去
    if ( TIM_GetITStatus ( GENERAL_TIM, TIM_IT_Update) != RESET )
    {
     TIM_ICUserValueStructure.Capture_Period ++;
     TIM_ClearITPendingBit ( GENERAL_TIM, TIM_FLAG_Update );
    }

    // 上升沿捕获中断
    if ( TIM_GetITStatus (GENERAL_TIM, GENERAL_TIM_IT_CCx ) != RESET)
    {
     // 第一次捕获
     if ( TIM_ICUserValueStructure.Capture_StartFlag == 0 )
     {
          // 计数器清 0
          TIM_SetCounter ( GENERAL_TIM, 0 );
          // 自动重装载寄存器更新标志清 0
          TIM_ICUserValueStructure.Capture_Period = 0;
       // 存捕获比较寄存器的值的变量的值清 0
          TIM_ICUserValueStructure.Capture_CcrValue = 0;

          // 当第一次捕获到上升沿之后，就把捕获边沿配置为下降沿
          GENERAL_TIM_OCxPolarityConfig_FUN(GENERAL_TIM, TIM_ICPolarity_Falling);
       // 开始捕获标准置 1
          TIM_ICUserValueStructure.Capture_StartFlag = 1;
     }
     // 下降沿捕获中断
     else // 第二次捕获
     {
```

```
        // 获取捕获比较寄存器的值，这个值就是捕获到的高电平的时间的值
        TIM_ICUserValueStructure.Capture_CcrValue =
        GENERAL_TIM_GetCapturex_FUN (GENERAL_TIM);

        // 当第二次捕获到下降沿之后，就把捕获边沿配置为上升沿，好开启新的一轮捕获
        GENERAL_TIM_OCxPolarityConfig_FUN(GENERAL_TIM, TIM_ICPolarity_Rising);
      // 开始捕获标志清 0
        TIM_ICUserValueStructure.Capture_StartFlag = 0;
      // 捕获完成标志置 1
        TIM_ICUserValueStructure.Capture_FinishFlag = 1;
    }

     TIM_ClearITPendingBit (GENERAL_TIM,GENERAL_TIM_IT_CCx);
   }
}
```

用到的寄存器：

TIMx_CR1 - Timer Control Register 1：

TIM_ICInitStructure.TIM_Channel 配置捕获通道时可能涉及。

TIM_ICInitStructure.TIM_CounterMode 在时基结构体初始化时可能涉及。

TIMx_CCMR1 - Capture/Compare Mode Register 1：

TIM_ICInitStructure.TIM_ICSelection 和 TIM_ICInitStructure.TIM_ICPrescaler 配置输入捕获模式时涉及。

TIMx_CCER - Capture/Compare Enable Register：

TIM_ICInitStructure.TIM_ICPolarity 配置输入捕获信号的极性时涉及。

TIMx_SR - Status Register：

TIM_ClearITPendingBit 和 TIM_GetITStatus 用于清除和检查中断标志位。

TIMx_CCR1 - Capture/Compare Register 1：

TIM_ICUserValueStructure.Capture_CcrValue 存储捕获比较寄存器的值。

TIMx_EGR - Event Generation Register：

TIM_ClearITPendingBit 用于清除中断标志位。

Sort out the programme flow of TIMx_IRQHandler()

```
void GENERAL_TIM_INT_FUN(void)
{
   // 当要被捕获的信号的周期大于定时器的最长定时时，定时器就会溢出，产生更新中断
   // 这个时候我们需要把这个最长的定时周期加到捕获信号的时间里面去
   if ( TIM_GetITStatus ( GENERAL_TIM, TIM_IT_Update) != RESET )
```

```
    {
      TIM_ICUserValueStructure.Capture_Period ++;
      TIM_ClearITPendingBit ( GENERAL_TIM, TIM_FLAG_Update );
    }

    // 上升沿捕获中断
    if ( TIM_GetITStatus (GENERAL_TIM, GENERAL_TIM_IT_CCx ) != RESET)
    {
      // 第一次捕获
      if ( TIM_ICUserValueStructure.Capture_StartFlag == 0 )
      {
            // 计数器清 0
            TIM_SetCounter ( GENERAL_TIM, 0 );
            // 自动重装载寄存器更新标志清 0
            TIM_ICUserValueStructure.Capture_Period = 0;
          // 存捕获比较寄存器的值的变量的值清 0
            TIM_ICUserValueStructure.Capture_CcrValue = 0;

            // 当第一次捕获到上升沿之后，就把捕获边沿配置为下降沿
            GENERAL_TIM_OCxPolarityConfig_FUN(GENERAL_TIM, TIM_ICPolarity_Falling);
          // 开始捕获标准置 1
            TIM_ICUserValueStructure.Capture_StartFlag = 1;
      }
      // 下降沿捕获中断
      else // 第二次捕获
      {
            // 获取捕获比较寄存器的值，这个值就是捕获到的高电平的时间的值
            TIM_ICUserValueStructure.Capture_CcrValue =
            GENERAL_TIM_GetCapturex_FUN (GENERAL_TIM);

            // 当第二次捕获到下降沿之后，就把捕获边沿配置为上升沿，好开启新的一轮捕
获
            GENERAL_TIM_OCxPolarityConfig_FUN(GENERAL_TIM, TIM_ICPolarity_Rising);
          // 开始捕获标志清 0
            TIM_ICUserValueStructure.Capture_StartFlag = 0;
          // 捕获完成标志置 1
            TIM_ICUserValueStructure.Capture_FinishFlag = 1;
      }

      TIM_ClearITPendingBit (GENERAL_TIM,GENERAL_TIM_IT_CCx);
    }
}
```

计时器清零，捕获上升沿，存储，捕获下降沿，存储

Modify the project "TIM-通用定时器-脉宽测量" to measure the frequency of the input signal（TIM4-CH1）。

in GeneralTim.h

```
#define        GENERAL_TIM              TIM4
#define        GENERAL_TIM_IRQ          TIM4_IRQn
#define        GENERAL_TIM_INT_FUN      TIM4_IRQHandlerperiod:
```

```
// 上升沿捕获中断
    if ( TIM_GetITStatus (GENERAL_TIM, GENERAL_TIM_IT_CCx ) != RESET)
    {
     // 第一次捕获
     if ( TIM_ICUserValueStructure.Capture_StartFlag == 0 )
     {
          // 计数器清 0
          TIM_SetCounter ( GENERAL_TIM, 0 );
          // 自动重装载寄存器更新标志清 0
          TIM_ICUserValueStructure.Capture_Period = 0;
        // 存捕获比较寄存器的值的变量的值清 0
          TIM_ICUserValueStructure.Capture_CcrValue = 0;

          // 当第一次捕获到上升沿之后，就把捕获边沿配置为下降沿
          //GENERAL_TIM_OCxPolarityConfig_FUN(GENERAL_TIM, TIM_ICPolarity_Falling);
        // 开始捕获标准置 1
          TIM_ICUserValueStructure.Capture_StartFlag = 1;
     }
     // 下降沿捕获中断
     else // 第二次捕获
     {
          // 获取捕获比较寄存器的值，这个值就是捕获到的高电平的时间的值
          TIM_ICUserValueStructure.Capture_CcrValue =
          GENERAL_TIM_GetCapturex_FUN (GENERAL_TIM);

          // 当第二次捕获到下降沿之后，就把捕获边沿配置为上升沿，好开启新的一轮捕
获
          GENERAL_TIM_OCxPolarityConfig_FUN(GENERAL_TIM, TIM_ICPolarity_Rising);
        // 开始捕获标志清 0
          TIM_ICUserValueStructure.Capture_StartFlag = 0;
        // 捕获完成标志置 1
          TIM_ICUserValueStructure.Capture_FinishFlag = 1;
     }

     TIM_ClearITPendingBit (GENERAL_TIM,GENERAL_TIM_IT_CCx);
```

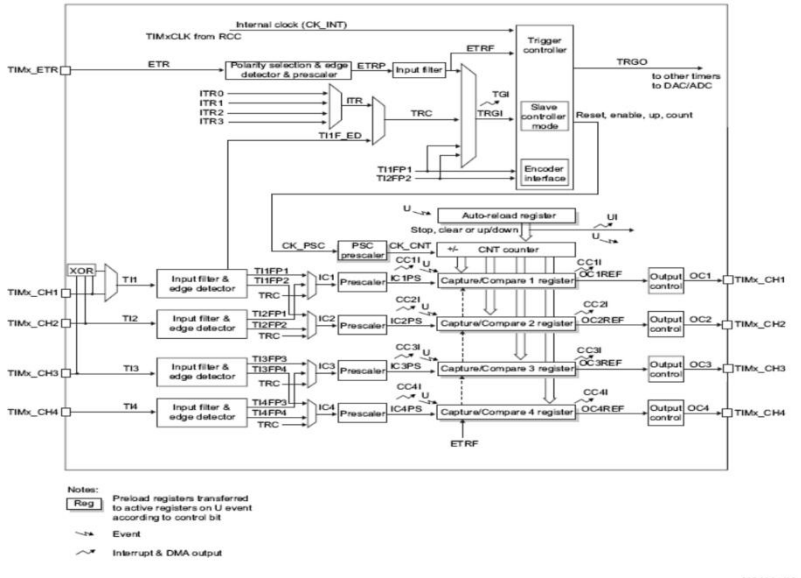
计算：
time = (TIM_ICUserValueStructure.Capture_Period * (GENERAL_TIM_PERIOD+1) +
(TIM_ICUserValueStructure.Capture_CcrValue+1)) / TIM_PscCLK
frequency = 1/time

Write out the registers those are used (configurated or read) by the function TIM_OCxInit() for General Tim and their meanings one by one

```
TIM_OCInitTypeDef    TIM_OCInitStructure;
// 配置为 PWM 模式 1
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
// 输出使能
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
// 输出通道电平极性配置
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

// 输出比较通道 1
TIM_OCInitStructure.TIM_Pulse = GENERAL_TIM_CCR1;
TIM_OC1Init(GENERAL_TIM, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(GENERAL_TIM, TIM_OCPreload_Enable);

// 输出比较通道 2
TIM_OCInitStructure.TIM_Pulse = GENERAL_TIM_CCR2;
TIM_OC2Init(GENERAL_TIM, &TIM_OCInitStructure);
TIM_OC2PreloadConfig(GENERAL_TIM, TIM_OCPreload_Enable);

// 输出比较通道 3
TIM_OCInitStructure.TIM_Pulse = GENERAL_TIM_CCR3;
TIM_OC3Init(GENERAL_TIM, &TIM_OCInitStructure);
TIM_OC3PreloadConfig(GENERAL_TIM, TIM_OCPreload_Enable);

// 输出比较通道 4
TIM_OCInitStructure.TIM_Pulse = GENERAL_TIM_CCR4;
TIM_OC4Init(GENERAL_TIM, &TIM_OCInitStructure);
TIM_OC4PreloadConfig(GENERAL_TIM, TIM_OCPreload_Enable);

// 使能计数器
TIM_Cmd(GENERAL_TIM, ENABLE);
```

TIMx->CCMR1：捕获/比较模式寄存器 1，用于配置输出比较通道的模式。
TIMx->CCER：捕获/比较使能寄存器，用于配置输出比较通道的使能状态。
TIMx->CCR1、TIMx->CCR2、TIMx->CCR3、TIMx->CCR4：捕获/比较寄存器，分别配置输出比较通道 1 到 4 的比较值。

Figure 100. General-purpose timer block diagram



Modify the frequency and duty of output PWM in the project“通用定时器-输出PWM”

// PWM 信号的周期 T = （ARR+1） * (1/CLK_cnt) = （ARR+1）*(PSC+1) / 72M

// 占空比 P=CCR/(ARR+1)

修改频率 frequency：

```
#define                GENERAL_TIM_Period                9
```

修改占空比 duty：

```
    // 占空比配置
    uint16_t CCR1_Val = 5;
    uint16_t CCR2_Val = 4;
    uint16_t CCR3_Val = 3;
    uint16_t CCR4_Val = 2;
```

Read the project“通用定时器-PWM 呼吸灯-delay--示波器”, Sort out the programme flow of this problem: How to constantly and regularly change the duty. You can observe the output wave by Oscilloscope(Which GPIO?)

观察——放到 PB5

```
while(1)
    {
        for(kcnt=2001;kcnt>0;kcnt=kcnt-200)
        {
            TIM_SetCompare1(TIM,kcnt);
            Delay_nms(100);
        }

        for(kcnt=1;kcnt<=2001;kcnt=kcnt+200)
        {
```

```
            TIM_SetCompare1(TIM,kcnt);
            Delay_nms(100);
        }
    }
```

设置寄存比较器的值，控制 PWM 占比

void TIM_SetComparex(TIM_TypeDef* TIMx, uint16_t Compare1)

TIM_TypeDef* TIMx：这是一个指向 TIM_TypeDef 结构体的指针，用于指定要操作的定时器实例。TIM_TypeDef 结构体是定时器的寄存器映射，包含了定时器的各种寄存器。

uint16_t Compare1：这是一个用于设置比较寄存器值的参数，即要设置的 PWM 占空比值。在呼吸灯效果中，该值通常在合适的范围内递增或递减，以实现 LED 的渐变亮度。

Modify the project“通用定时器-PWM 呼吸灯-delay--示波器”， realize the breathing LED---Easy, change OCx-CH

```
#include "timer_pwm.h"

//TIM3 CH1--PA6 引脚配置
static void GENERAL_TIM_GPIO_Config()
{
  GPIO_InitTypeDef GPIO_InitStructure;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
  GPIO_InitStructure.GPIO_Pin =     GPIO_Pin_0;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

}


static void GENERAL_TIM__PWM_Config()
{

    TIM_TimeBaseInitTypeDef   TIM_TimeBaseStructure;
      TIM_OCInitTypeDef   TIM_OCInitStruct;

     // 1 开启定时器时钟,即内部时钟 CK_INT=72M
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);


     TIM_DeInit(TIM);
      TIM_InternalClockConfig(TIM);
```

```
    // 2  自动重装载寄存器的值，累计 TIM_Period+1 个频率后产生一个更新或者中断
    TIM_TimeBaseStructure.TIM_Period = TIMx_Period;

     // 时钟预分频数为
    TIM_TimeBaseStructure.TIM_Prescaler= TIMx_Prescaler;

    // 时钟分频因子 ，
    TIM_TimeBaseStructure.TIM_ClockDivision=0x0;

    // 计数器计数模式，
    TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;

    // 重复计数器的值，
    //TIM_TimeBaseStructure.TIM_RepetitionCounter=0;

     // 初始化定时器
    TIM_TimeBaseInit(TIM, &TIM_TimeBaseStructure);

    //继续配置 PWM 功能。初始 CCR=72

    TIM_OCInitStruct.TIM_OCMode=TIM_OCMode_PWM1;
    TIM_OCInitStruct.TIM_OutputState=TIM_OutputState_Enable;
    TIM_OCInitStruct.TIM_OCPolarity=TIM_OCPolarity_High;
    TIM_OCInitStruct.TIM_Pulse=72;
     TIM_OC3Init(TIM,&TIM_OCInitStruct);

    //使能 TIMx 在 CCR 上的预装载寄存器
    TIM_OC3PreloadConfig(TIM,TIM_OCPreload_Enable);

    //设置 TIMx 的 PWM 输出为使能
    TIM_CtrlPWMOutputs(TIM,ENABLE);

    //使能 TIMx 在 ARR 上的预装载寄存器
    TIM_ARRPreloadConfig(TIM,ENABLE);

    //4  使能 TIMx 外设
        TIM_Cmd(TIM, ENABLE);

           TIM_ITConfig(TIM, TIM_IT_Update, ENABLE);
}



void GENERAL_TIM_PWM_Init(void)
```

```
{
    GENERAL_TIM_GPIO_Config();

    GENERAL_TIM__PWM_Config();


}
```

main.c:

```
/*********************************
芯片：STM32F103ZET6
实现功能：使用通用定时器 TIM1 实现 PWM 输出（）；
引脚：TIM3 CH1---PA6;
**********************************/

#include "stm32f10x.h"
#include "timer_pwm.h"

void Delay_nms(__IO uint32_t count )
{

    uint16_t i;
    while(count--)
    {
      i=12000;
      while(i--);
    }
}


int main(void)
{


       short int kcnt=2000;
      //uint8_t sign=1;
      GENERAL_TIM_PWM_Init();


       while(1)
           {
                for(kcnt=2001;kcnt>0;kcnt=kcnt-200)
```

```
            {
                TIM_SetCompare3(TIM,kcnt);
                Delay_nms(100);
            }

            for(kcnt=1;kcnt<=2001;kcnt=kcnt+200)
            {
                TIM_SetCompare3(TIM,kcnt);
                Delay_nms(100);
            }
        }
}
```