

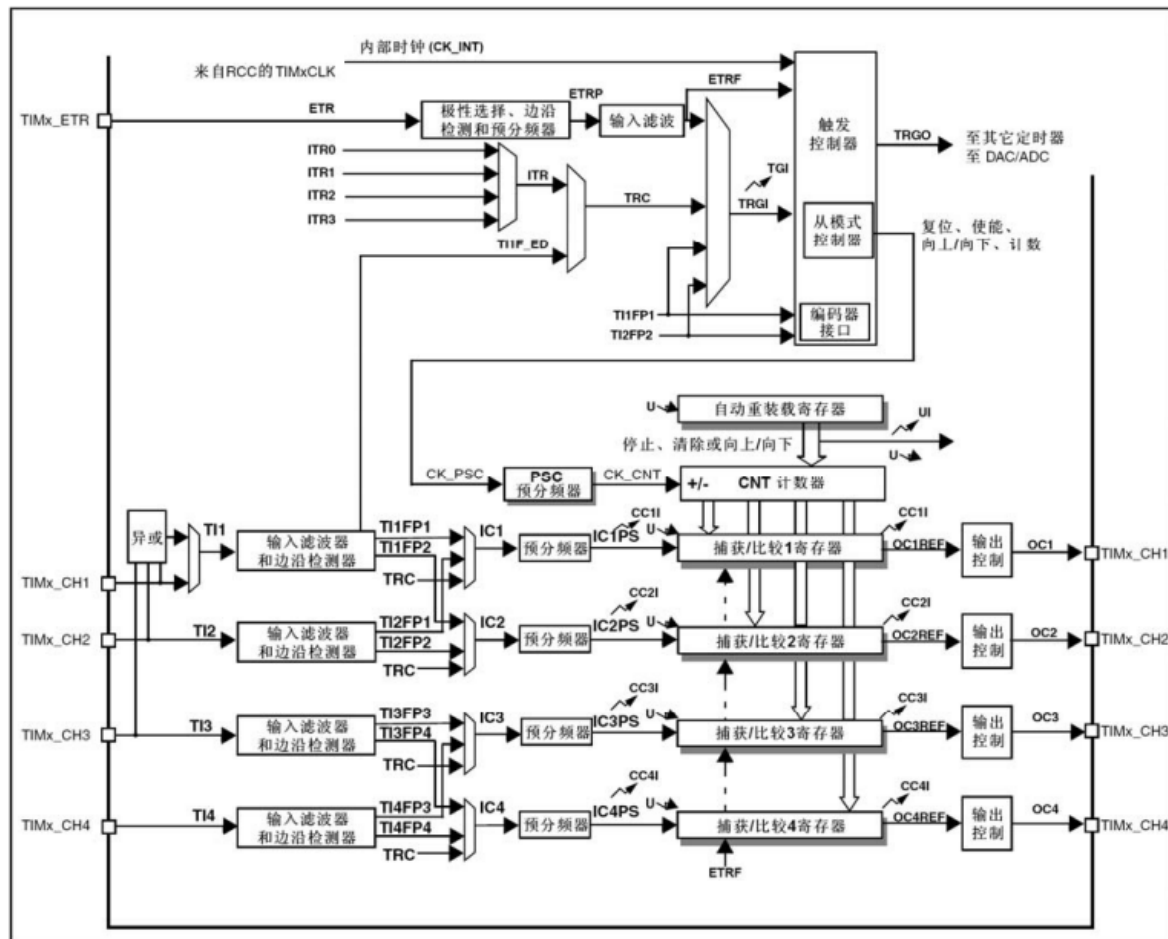
# Homework 4




## Week 7

### Introduce TIM\_IC

Write out the registers those are used(configured or read) by the function `TIM_ICInit()` `TIM_IRQHandler()` and their meanings one by one;

图98 通用定时器框图



注:  根据控制位的设定, 在U事件时传送预加载寄存器的内容至工作寄存器  
 事件  
 中断和DMA输出

### 结构体

```
typedef struct
{
    uint16_t TIM_Channel; //通道选择配置 CCMRx 寄存器
    uint16_t TIM_ICPolarity; //边沿触发选择配置 CCER 寄存器
    uint16_t TIM_ICSelection; //输入捕获选择配置 CCMRx 寄存器
    uint16_t TIM_ICPrescaler; //输入捕获预分频器关联 CCMRx 寄存器
    uint16_t TIM_ICFilter; //输入捕获滤波器关联 CCMRx 寄存器
} TIM_ICInitTypeDef;
```

过程中用到的基本的寄存器:

## 自动重载寄存器ARR

自动重载寄存器ARR用来存放与计数器CNT比较的值，如果两个值相等就递减重复计数器。可以通过TIMX\_CR1寄存器的ARPE位控制自动重载影子寄存器功能，如果ARPE位置1，自动重载影子寄存器有效，只有在事件更新时才把TIMX\_ARR值赋给影子寄存器。如果ARPE位为0，则修改TIMX\_ARR值马上有效。

## 预分频器PSC

预分频器PSC，有一个输入时钟CK\_PSC和一个输出时钟CK\_CNT。输入时钟CK\_PSC就是上面时钟源的输出，输出CK\_CNT则用来驱动计数器CNT计数。通过设置预分频器PSC的值可以得到不同的CK\_CNT，计算公式为： $f_{CK\_CNT} = \frac{f_{CK\_PSC}}{PSC[15:0]+1}$ ，可以实现1至65536分频。

## 计数器CNT

(1) 递增计数模式下，计数器从0开始计数，每来一个CK\_CNT脉冲计数器就增加1，直到计数器的值与自动重载寄存器ARR值相等，然后计数器又从0开始计数并生成计数器上溢事件，计数器总是如此循环计数。如果禁用重复计数器，在计数器生成上溢事件就马上生成更新事件；如果使能重复计数器，每生成一次上溢事件重复计数器内容就减1，直到重复计数器内容为0时才会生成更新事件。

(2) 递减计数模式下，计数器从自动重载寄存器ARR值开始计数，每来一个CK\_CNT脉冲计数器就减1，直到计数器值为0，然后计数器又从自动重载寄存器ARR值开始递减计数并生成计数器下溢事件，计数器总是如此循环计数。如果禁用重复计数器，在计数器生成下溢事件就马上生成更新事件；如果使能重复计数器，每生成一次下溢事件重复计数器内容就减1，直到重复计数器内容为0时才会生成更新事件。

(3) 中心对齐模式下，计数器从0开始递增计数，直到计数值等于(ARR-1)值生成计数器上溢事件，然后从ARR值开始递减计数直到1生成计数器下溢事件。然后又从0开始计数，如此循环。每次发生计数器上溢和下溢事件都会生成更新事件。

## 捕获寄存器

当发生捕获时（第一次），计数器CNT的值会被锁存到捕获寄存器CCR中，还会产生CCxI中断，相应的中断位CCxIF（在SR寄存器中）会被置位，通过软件（写CCxIF=0）或者读取CCR中的值可以将CCxIF清0。如果发生第二次捕获（即重复捕获：CCR寄存器中已捕获到计数器值且CCxIF标志已置1），则捕获溢出标志位CCxOF（在SR寄存器中）会被置位，CCxOF只能通过软件清零（写CCxOF=0）。

TIMX\_IRQHandler() :

```
void GENERAL_TIM_INT_FUN(void)
{
    if ( TIM_GetITStatus ( GENERAL_TIM, TIM_IT_Update) != RESET )
    {
        TIM_ICUserValueStructure.Capture_Period ++;
        TIM_ClearITPendingBit ( GENERAL_TIM, TIM_FLAG_Update );
    }

    if ( TIM_GetITStatus (GENERAL_TIM, GENERAL_TIM_IT_CCx ) != RESET)
    {
        if ( TIM_ICUserValueStructure.Capture_StartFlag == 0 )
        {
            TIM_SetCounter ( GENERAL_TIM, 0 );
            TIM_ICUserValueStructure.Capture_Period = 0;
            TIM_ICUserValueStructure.Capture_CcrValue = 0;
        }
    }
}
```

```

        GENERAL_TIM_OCxPolarityConfig_FUN(GENERAL_TIM,
TIM_ICPolarity_Falling);
        TIM_ICUserValueStructure.Capture_StartFlag = 1;
    }
    else
    {
        TIM_ICUserValueStructure.Capture_CcrValue =
        GENERAL_TIM_GetCapturex_FUN (GENERAL_TIM);

        GENERAL_TIM_OCxPolarityConfig_FUN(GENERAL_TIM,
TIM_ICPolarity_Rising);
        TIM_ICUserValueStructure.Capture_StartFlag = 0;
        TIM_ICUserValueStructure.Capture_FinishFlag = 1;
    }

    TIM_ClearITPendingBit (GENERAL_TIM,GENERAL_TIM_IT_CCx);
}
}

```

用到的寄存器基本是一致的，cnt 溢出时产生一次更新中断，使得自定义的 uint16\_t Capture\_Period 的值+1，记录一次定时周期时间。

## Flowchart

等待上升沿→检测到上升沿 → 置零计数器&清空计数变量&调整检测策略&开始捕获

等待下降沿→检测到下降沿 → 拷贝捕获寄存器的值，更新开始及完成捕获标志 → 等待上升沿  
(产生更新事件 → 周期计数变量加一)

主循环 → 完成捕获标志变为1 → 计算输出

## Program

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

### 1. 修改define

课件指定的project文件 GeneralTim.h 中没有对对应的引脚进行定义，查找发现引脚没有定义到.h 文件中而是直接写在了.c文件中：

```

// 开启定时器时钟, 即内部时钟CK_INT=72M
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5, ENABLE);

/*-----时基结构体初始化-----*/
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
// 自动重载寄存器的值, 累计TIM_Period+1个频率后产生一个更新或者中断
TIM_TimeBaseStructure.TIM_Period=GENERAL_TIM_PERIOD;
// 驱动CNT计数器的时钟 = Fck_int/(psc+1)
TIM_TimeBaseStructure.TIM_Prescaler= GENERAL_TIM_PSC;
// 时钟分频因子
TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;
// 计数器计数模式, 设置为向上计数
TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;
// 重复计数器的值, 没用到不用管
TIM_TimeBaseStructure.TIM_RepetitionCounter=0;
// 初始化定时器
TIM_TimeBaseInit(TIM5, &TIM_TimeBaseStructure);

```

猜测可能是文件指代出错了，找了一下另一个project是有相关的宏定义的，于是用另一个 bsp\_GeneralTim.h 文件修改作为示例（

```

/*****通用定时器TIM参数定义，只限TIM2、3、4、5*****/
// 当使用不同的定时器的時候，对应的GPIO是不一样的，这点要注意
// 我们这里默认使用TIM5

#define          GENERAL_TIM                TIM4
#define          GENERAL_TIM_APBxClock_FUN  RCC_APB1PeriphClockCmd
#define          GENERAL_TIM_CLK            RCC_APB1Periph_TIM4
#define          GENERAL_TIM_PERIOD         0xFFFF
#define          GENERAL_TIM_PSC            (72-1)

// TIM 输入捕获通道GPIO相关宏定义
#define          GENERAL_TIM_CH1_GPIO_CLK   RCC_APB2Periph_GPIOB
#define          GENERAL_TIM_CH1_PORT       GPIOB
#define          GENERAL_TIM_CH1_PIN        GPIO_Pin_6
#define          GENERAL_TIM_CHANNEL_x      TIM_Channel_1

// 中断相关宏定义
#define          GENERAL_TIM_IT_CCx         TIM_IT_CC1
#define          GENERAL_TIM_IRQ            TIM4_IRQn
#define          GENERAL_TIM_INT_FUN        TIM4_IRQHandler

```

将对应引脚指向PB6.

## 2. 删除更换捕获边沿

```

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
TIM_OC1PolarityConfig(TIM5, TIM_ICPolarity_Falling);
// 开始捕获标准置1
TIM_ICUserValueStructure.Capture_StartFlag = 1;

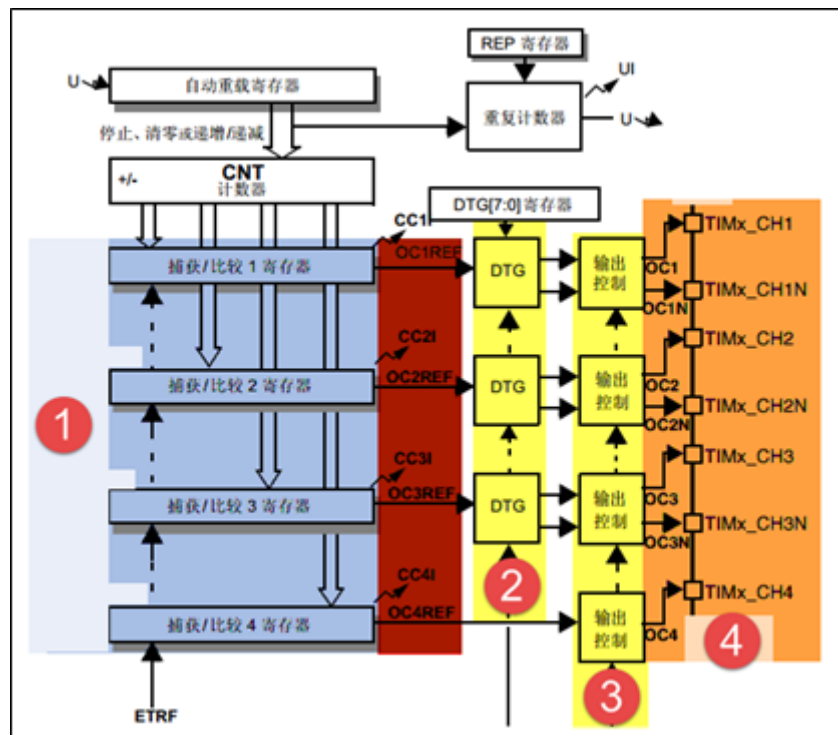
```

## 3. 计算频率

由：频率 = (周期)<sup>-1</sup>

```
printf ( "\r\n测得高电平脉宽时间: %f s\r\n", (float)TIM_PscCLK/number);
```

## Introduce TIM\_OC



## PWM 参数与寄存器值关系计算

频率  $\text{Freq} = \text{CK\_PSC} / (\text{PSC} + 1) / (\text{ARR} + 1)$

占空比  $\text{Duty} = \text{CCR} / (\text{ARR} + 1)$

## 结构体

```
typedef struct
{
    uint16_t TIM_OCMode;
    uint16_t TIM_OutputState;
    uint16_t TIM_OutputNState;
    uint16_t TIM_Pulse;
    uint16_t TIM_OCPolarity;
    uint16_t TIM_OCNPolarity;
    uint16_t TIM_OCIdleState;
    uint16_t TIM_OCNIdleState;
} TIM_OCInitTypeDef;
```

基本TIM使用的PSC/ARR/CNT/CCR寄存器详见第一部分。

## Program

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

1. 修改频率: GeneralTim.h

```
#define GENERAL_TIM TIM3
#define GENERAL_TIM_APBxClock_FUN RCC_APB1PeriphClockCmd
#define GENERAL_TIM_CLK RCC_APB1Periph_TIM3
#define GENERAL_TIM_Period 9
#define GENERAL_TIM_Prescaler 71
```

2. 修改占空比: GeneralTim.c

```
// 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;
```

## Program

Read the project “通用定时器-PWM呼吸灯-delay--示波器”, Sort out the program flow of this problem: How to constantly and regularly change the duty.

## Flowchart

配置定时器初始化与 PWM1 输出 & 配置GPIO引脚初始化 → 定时使用 TIM\_SetCompare1 改变占空比

## Modify

1. 挑选引脚

```
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);
```

2. TIM\_OC3Init(TIM, &TIM\_OCInitStruct);

```
TIM_OC3PreloadConfig(TIM, TIM_OCPreload_Enable);

while(1)
{
    for(kcnt=2001;kcnt>0;kcnt=kcnt-200)
    {
        TIM_SetCompare3(TIM,kcnt);
        Delay_nms(10000);
    }

    for(kcnt=1;kcnt<=2001;kcnt=kcnt+200)
    {
        TIM_SetCompare3(TIM,kcnt);
        Delay_nms(10000);
    }
}
```

3. 调整时间使平滑。

## Week 8

### Introduce TIM\_BDTR

TIM\_OCInit() 似乎是 Week 7 的作业欸 (

#### 13.4.18 TIM1 和TIM8 刹车和死区寄存器(TIMx\_BDTR)

偏移地址: 0x44

复位值: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MOE	AOE	BKP	BKE	OSSI	OSSI	LOCK[1:0]	DTG[7:0]								
1W	1W	1W	1W	1W	1W	1W	1W	1W	1W	1W	1W	1W	1W	1W	1W

注释: 根据锁定设置, AOE、BKP、BKE、OSSI、OSSI和DTG[7:0]位均可被写保护, 有必要在第一次写入TIMx\_BDTR寄存器时对它们进行配置。

位15	<b>MOE:</b> 主输出使能 (Main output enable) 一旦刹车输入有效, 该位被硬件异步清'0'。根据AOE位的设置值, 该位可以由软件清'0'或被自动置1。它仅对配置为输出的通道有效。 0: 禁止OC和OCN输出或强制为空闲状态; 1: 如果设置了相应的使能位(TIMx_CCER寄存器的CCxE、CCxNE位), 则开启OC和OCN输出。 有关OC/OCN使能的细节, 参见13.4.9节, TIM1和TIM8捕获/比较使能寄存器(TIMx_CCER)。
位14	<b>AOE:</b> 自动输出使能 (Automatic output enable) 0: MOE只能被软件置'1'; 1: MOE能被软件置'1'或在下一个更新事件被自动置'1'(如果刹车输入无效)。 注: 一旦LOCK级别(TIMx_BDTR寄存器中的LOCK位)设为'1', 则该位不能被修改。
位13	<b>BKP:</b> 刹车输入极性 (Break polarity) 0: 刹车输入低电平有效; 1: 刹车输入高电平有效。 注: 一旦LOCK级别(TIMx_BDTR寄存器中的LOCK位)设为'1', 则该位不能被修改。 注: 任何对该位的写操作都需要一个APB时钟的延迟以后才能起作用。
位12	<b>BKE:</b> 刹车功能使能 (Break enable) 0: 禁止刹车输入(BRK及CCS时钟失效事件); 1: 开启刹车输入(BRK及CCS时钟失效事件)。 注: 当设置了LOCK级别1时(TIMx_BDTR寄存器中的LOCK位), 该位不能被修改。 注: 任何对该位的写操作都需要一个APB时钟的延迟以后才能起作用。



位11	<b>OSSR:</b> 运行模式下“关闭状态”选择 (Off-state selection for Run mode) 该位用于当MOE=1且通道为互补输出时。没有互补输出的定时器中不存在OSSR位。 参考OC/OCN使能的详细说明(13.4.9节, TIM1和TIM8捕获/比较使能寄存器(TIMx_CCER))。 0: 当定时器不工作时, 禁止OC/OCN输出(OC/OCN使能输出信号=0); 1: 当定时器不工作时, 一旦CCxE=1或CCxNE=1, 首先开启OC/OCN并输出无效电平, 然后置OC/OCN使能输出信号=1。 注: 一旦LOCK级别(TIMx_BDTR寄存器中的LOCK位)设为2, 则该位不能被修改。
位10	<b>OSSI:</b> 空闲模式下“关闭状态”选择 (Off-state selection for Idle mode) 该位用于当MOE=0且通道设为输出时。 参考OC/OCN使能的详细说明(13.4.9节, TIM1和TIM8捕获/比较使能寄存器(TIMx_CCER))。 0: 当定时器不工作时, 禁止OC/OCN输出(OC/OCN使能输出信号=0); 1: 当定时器不工作时, 一旦CCxE=1或CCxNE=1, OC/OCN首先输出其空闲电平, 然后OC/OCN使能输出信号=1。 注: 一旦LOCK级别(TIMx_BDTR寄存器中的LOCK位)设为2, 则该位不能被修改。
位9:8	<b>LOOK[1:0]:</b> 锁定设置 (Lock configuration) 该位为防止软件错误而提供写保护。 00: 锁定关闭, 寄存器无写保护; 01: 锁定级别1, 不能写入TIMx_BDTR寄存器的DTG、BKE、BKP、AOE位和TIMx_CR2寄存器的OISx/OISxN位; 10: 锁定级别2, 不能写入锁定级别1中的各位, 也不能写入CC极性位(一旦相关通道通过CCxS位设为输出, CC极性位是TIMx_CCER寄存器的CCxP/CCNxP位)以及OSSR/OSSI位; 11: 锁定级别3, 不能写入锁定级别2中的各位, 也不能写入CC控制位(一旦相关通道通过CCxS位设为输出, CC控制位是TIMx_CCMRx寄存器的OCxM/OCxPE位); 注: 在系统复位后, 只能写一次LOCK位, 一旦写入TIMx_BDTR寄存器, 则其内容冻结直至复位。
位7:0	<b>UTG[7:0]:</b> 死区发生器设置 (Dead-time generator setup) 这些位定义了插入互补输出之间的死区持续时间。假设DT表示其持续时间: $DTG[7:5]=0xx \Rightarrow DT=DTG[7:0] \times T_{dtg}, T_{dtg} = T_{DTS};$ $DTG[7:5]=10x \Rightarrow DT=(64+DTG[5:0]) \times T_{dtg}, T_{dtg} = 2 \times T_{DTS};$ $DTG[7:5]=110 \Rightarrow DT=(32+DTG[4:0]) \times T_{dtg}, T_{dtg} = 8 \times T_{DTS};$ $DTG[7:5]=111 \Rightarrow DT=(32+DTG[4:0]) \times T_{dtg}, T_{dtg} = 16 \times T_{DTS};$ 例: 若 $T_{DTS} = 125ns(8MHZ)$ , 可能的死区时间为: 0到15875ns, 若步长为125ns; 16us到31750ns, 若步长为250ns; 32us到63us, 若步长为1us; 64us到126us, 若步长为2us; 注: 一旦LOCK级别(TIMx_BDTR寄存器中的LOCK位)设为1、2或3, 则不能修改这些位。

## 结构体

```
typedef struct
{
    uint16_t TIM_OSSRState; //运行时关闭状态选择
    uint16_t TIM_OSSIState; //空闲时关闭状态选择
    uint16_t TIM_LOCKLevel; //锁定等级
    uint16_t TIM_DeathTime; //死区时间
    uint16_t TIM_Break; //刹车使能
    uint16_t TIM_BreakPolarity; //刹车极性
    uint16_t TIM_AutomaticOutput; //自动输出使能
} TIM_BDTRInitTypeDef;
```

MOE由另外的函数开闭;

除上文所述基本寄存器外, 额外用到的寄存器全部属于BDTR寄存器。

系统复位启动都默认关闭断路功能, 将BDTR寄存器的BKE为置1, 使能断路功能。可通过 TIMx\_BDTR 寄存器的BKP位设置设置断路输入引脚的有效电平, 设置为1时输入BRK为高电平有效, 否则低电平有效。

## Program

Modify the frequency and duty of the output PWM, and verify its break function based on the project “高级定时器-输出PWM带死区”. frequency = 150K & duty = 5

```
// PWM 信号的频率 F = TIM_CLK/ {(ARR+1)*(PSC+1)}  
#define ADVANCE_TIM_PERIOD (10/1.5-1)  
#define ADVANCE_TIM_PSC (72-1)  
#define ADVANCE_TIM_PULSE (0.05*(10/1.5-1))
```

(奇怪的频率