

# 该文章将记录蓝桥杯的模块使用方案 等

---

## 模板概述

---

模板已经配置了基本所有的驱动了，包括但不限于

LED灯

按键扫描

LCD屏幕

UART

EEPROM

DHT11

板载RES

**但是测试不尽完整，还请认真阅读理解原理**

开始使用模板后，建议将不需要的**函数**删除

别反初始化啥函数，毕竟别搞不清楚这个外设也有别的用途

删完之后确定能编译，能跑起来

再接着往下写

如果后面某个外设寄了别急，看看下面的手册

搞不懂就重新从模板开始

## 省赛

---

### 按键

别老惦记着你那外部中断啦，不行的，老老实实按键扫描就好

记得在cubemx里初始化，但是不需要上下拉（因为官方也没有做）

消抖是必须要做的，但是只需要10ms就好（因为官方这么写的）

按键的一种写法：

```
#define B1 HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_0)
#define B2 HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1)
#define B3 HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2)
#define B4 HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)

u8 keyScan()
{
    if(B1==0)
    {
        HAL_Delay(10);
```

```

        if(B1==0)
            return 1;
    }
    if(B2==0)
    {
        HAL_Delay(10);
        if(B2==0)
            return 2;
    }
    if(B3==0)
    {
        HAL_Delay(10);
        if(B3==0)
            return 3;
    }
    if(B4==0)
    {
        HAL_Delay(10);
        if(B4==0)
            return 4;
    }
    return 0;
}

u8 nowKey=0,oldKey=0;
while(1)
{
    nowKey=keyScan();
    if(oldKey!=nowKey)
    {
        oldKey=nowKey;
        switch(nowKey)
        {
            case 0:
                break;
            case 1:
                break;
            case 2:
                break;
            case 3:
                break;
            case 4:
                break;
        }
    }
}

```

消抖，分离按键，防重复触发一体化

# UART

UART使用UART1

空闲中断需要如下配置

Configuration

Reset Configuration

Parameter Settings

User Constants

NVIC Settings

DMA Settings

GPIO Settings

USART1\_RXDMA1 Channel 1Peripheral To MemoryHigh

AddDelete

DMA Request Settings

ModeNormal

Increment Address

Peripheral

Memory

Data WidthByte

Byte

DMA Request Synchronization Settings

Enable synchronization

Reset Configuration

Parameter Settings

User Constants

NVIC Settings

DMA Settings

GPIO Settings

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
DMA1 channel1 global interrupt	<input checked="" type="checkbox"/>	0	0
USART1 global interrupt / USART1 wake-up interrupt through EXTI line 25	<input checked="" type="checkbox"/>	0	0

```
int recLen=100;
char rec[100]={0};
int main(void)
{
    __HAL_UART_ENABLE_IT(&huart1,UART_IT_IDLE);
    HAL_UART_Receive_DMA(&huart1,(uint8_t*)rec,recLen);
}
```

```
void USART1_IRQHandler(void)
{
    /* USER CODE BEGIN USART1_IRQn 0 */
    if(__HAL_UART_GET_IT_SOURCE(&huart1,UART_IT_IDLE)!=RESET){
        __HAL_UART_CLEAR_IDLEFLAG(&huart1);
        HAL_UART_DMAstop(&huart1);
        int recCNT=recLen-__HAL_DMA_GET_COUNTER(huart1.hdmarx);
        //int recCNT=recLen-__HAL_DMA_GET_COUNTER(&hdma_usart1_rx);//等价于上面一句，下面这个变量好找一点
        char tmp[100]="";
        strncpy(tmp,rec,recCNT);
        HAL_UART_Transmit(&huart1,(uint8_t*)tmp,strlen(tmp),255);//测试发回去
        memset(rec,0,recLen);
        HAL_UART_Receive_DMA(&huart1,(uint8_t*)rec,recLen);
    }
    /* USER CODE END USART1_IRQn 0 */
    HAL_UART_IRQHandler(&huart1);
    /* USER CODE BEGIN USART1_IRQn 1 */

    /* USER CODE END USART1_IRQn 1 */
}
```

```
}
```

## LCD

该部分已经在模板里写好

io: 请使用LCD例程模板

为保证在LCD与灯状态的共存, 需要做出如下修改

lcd.c

```
LCD_WriteReg函数  
LCD_WriteRAM_Prepare函数  
LCD_WriteRAM函数  
一共三个函数每个函数前后两行供六行  
  
u16 tmp=GPIOC->ODR; //不能是u8  
//函数原本内容  
GPIOC->ODR=tmp;
```

LCD每一行最大上限是19个字符, 超过上限编译时不会报错, 但是会重启炸Fault (推测为内存越界)

## 写入单字符

```
sprintf(tmp, "      :%02d:%02d  ", min, sec); //给要写入的带颜色字符留空  
LCD_DisplayStringLine(Line3, (uint8_t*)tmp);  
LCD_SetTextColor(Red);  
LCD_DisplayChar(Line3, 319-16*4, hour/10+'0'); //对写入单一字符  
LCD_DisplayChar(Line3, 319-16*5, hour%10+'0');  
LCD_SetTextColor(White);
```

第二个参数计算方法是: 319-16\*字符索引 (从0开始索引)

第三个参数需要填写ASCII码, 可以使用'0'表示字符0

**注意: 若连续使用上述代码, 会导致一直画面不稳定, 应按需刷新**

## 翻转显示

[https://blog.csdn.net/qg\\_44920338/article/details/125573284](https://blog.csdn.net/qg_44920338/article/details/125573284)

原代码 (注释是它的) :

```
LCD_WriteReg(R1 , 0x0000); // set SS and SM bit //0x0100  
LCD_WriteReg(R96 , 0x2700); // Gate Scan Line 0xA700
```

修改后:

```
LCD_WriteReg(R1 , 0x0100); // set SS and SM bit    (从下往上)
LCD_WriteReg(R96 , 0xA700); // Gate Scan Line      (从右往左)
```

实现翻转两个都要，否则只是镜像

诶：原来它注释提醒了我？？

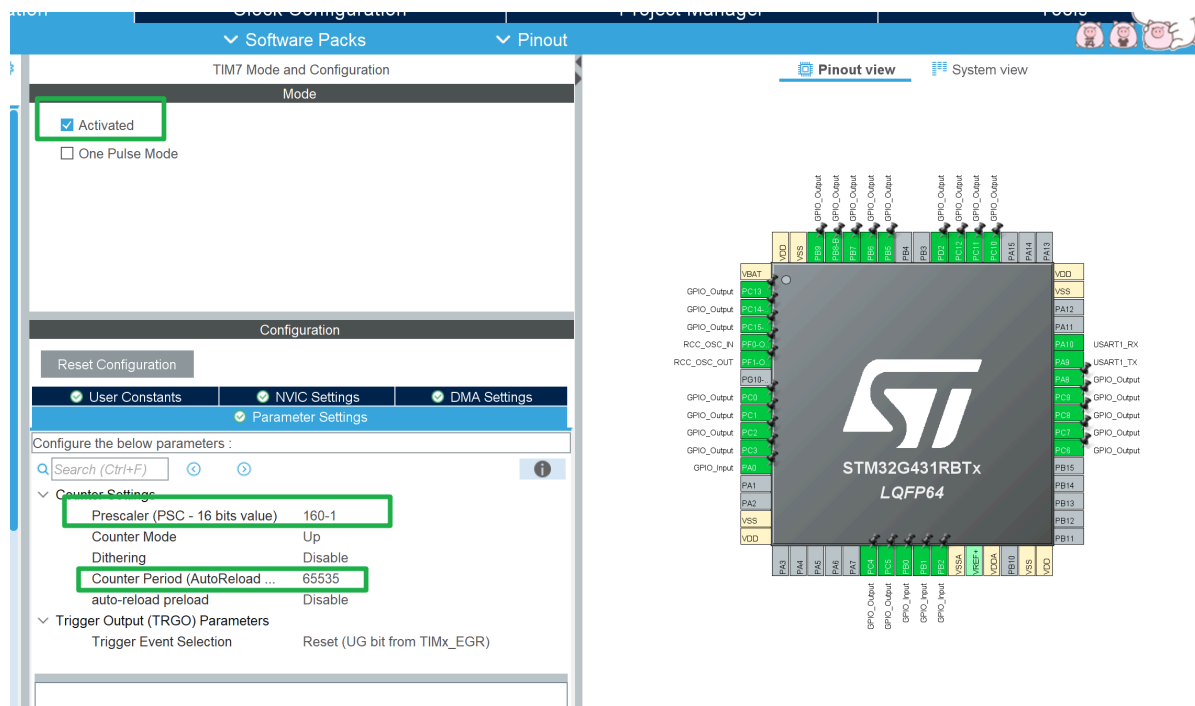
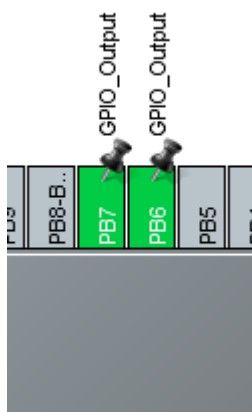
## 板载EEPROM

该部分驱动已经在模板里写好

型号：M24C02-WMN6TP

I2C软件模拟

io：PB6 PB7，GPIO OUTPUT不需要做额外配置。为了delay准确，配置TIM7用于延时控制



不用打开中断

移植i2c.h后

默写如下函数到main.c

```
uint8_t IICRead(uint8_t address)
```

```

{
    unsigned char val;

    I2CStart();
    I2CSendByte(0xa0);
    I2CWaitAck();

    I2CSendByte(address);
    I2CWaitAck();

    I2CStart();
    I2CSendByte(0xa1);
    I2CWaitAck();
    val = I2CReceiveByte();
    I2CWaitAck();
    I2CStop();

    return(val);
}

//
void IICWrite(unsigned char address,unsigned char info)
{
    I2CStart();
    I2CSendByte(0xa0);
    I2CWaitAck();

    I2CSendByte(address);
    I2CWaitAck();
    I2CSendByte(info);
    I2CWaitAck();
    I2CStop();
}

```

启动->写入元器件写地址->写入写入地址->写入数据->结束

启动->写入元器件写地址->写入读取地址->启动->写入元器件读地址->读取->结束

i2c.c内的delay1修改

```

/**
 * @brief I2C的短暂延时
 * @param None
 * @retval None
 */
extern TIM_HandleTypeDef htim7;
static void delay1(uint32_t nus)
{
    uint16_t differ = 0xffff-nus-500;
    //设置定时器2的技术初始值
    __HAL_TIM_SetCounter(&htim7,differ);
    //开启定时器
    HAL_TIM_Base_Start(&htim7);
}

```

```

while( differ<0xffff-500)
{
    differ = __HAL_TIM_GetCounter(&htim7);
};
//关闭定时器
HAL_TIM_Base_Stop(&htim7);
}

```

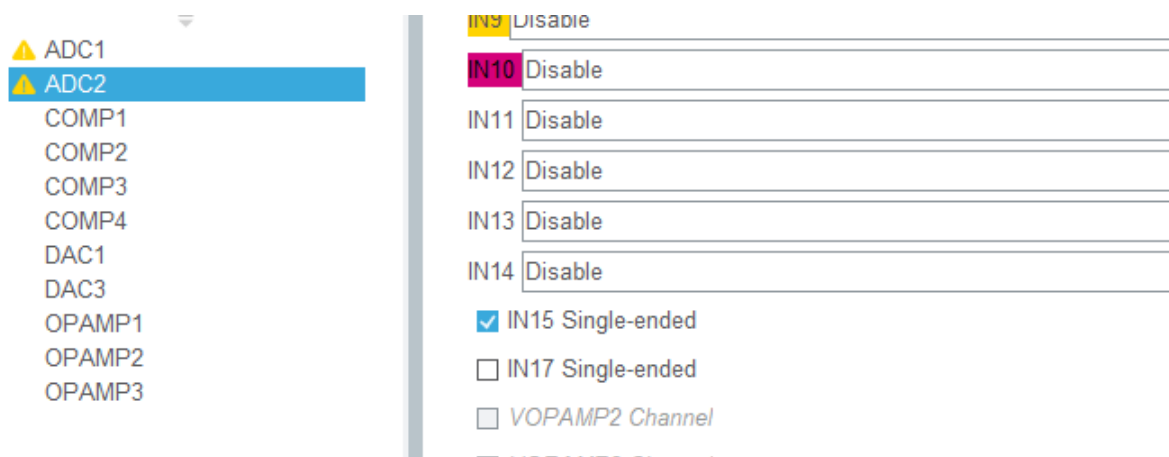
自己处理一下写入时的Delay，如果在IICWrite函数内加Delay，然后一不小心在中断里一个调用！卡死了

连续调用IICWrite会导致卡死

## 板载ADC

ADC硬件

io：打开IN15即可，不需要额外配置



main.c

```

HAL_ADCEx_Calibration_Start(&hadc2,ADC_SINGLE_ENDED);//校准

uint16_t Get_ADC2(void)
{
    uint16_t adc = 0;

    HAL_ADC_Start(&hadc2);
    adc = HAL_ADC_GetValue(&hadc2);

    return adc;
}

sprintf(tmp, "VR37: %.02fv", (float)Get_ADC2()/4096*3.3);

```

第二个旋钮用的是ADC1，根本不冲突

```
double Get_ADC1()
{
    HAL_ADC_Start(&hadc1);
    int tmp=HAL_ADC_GetValue(&hadc1);
    return (double)(tmp)/4096*3.3;
}
double Get_ADC2()
{
    HAL_ADC_Start(&hadc2);
    int tmp=HAL_ADC_GetValue(&hadc2);
    return (double)(tmp)/4096*3.3;
}
```

## 板载LED灯

**LED灯修改前必须进行LCD\_Init初始化**

锁存器，PD2高电平同步，低电平锁存

**警告，例程默认不初始化PD2，记得开（但是模板开了）**

使用：

main.c

```
HAL_GPIO_WritePin(GPIOC,GPIO_PIN_All,GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOD,GPIO_PIN_2,GPIO_PIN_SET);
HAL_GPIO_WritePin(GPIOD,GPIO_PIN_2,GPIO_PIN_RESET);
//全关灯
```

RTC

Mode

☒ Activate Clock Source

☒ Activate Calendar

Alarm A

Disable

Alarm B

Disable

WakeUp

Disable

☐ Timestamp

☐ Tamper 1

☐ Tamper 2

Calibration

Disable

☐ Reference clock detection

Configuration

Reset Configuration

☒ Parameter Settings

☒ User Constants

Configure the below parameters :

Search (Ctrl+F)

<

>

i

General

Hour Format

Hourformat 24

Asynchronous Predivider value

127

Synchronous Predivider value

249

Calendar Time

在读取RTC时，读完Time必须要读Date

```
RTC_DateTypeDef DateTmp;
RTC_TimeTypeDef TimeTmp;
while(1)
{
    HAL_RTC_GetTime(&hrtc,&TimeTmp,RTC_FORMAT_BIN);
    HAL_RTC_GetDate(&hrtc,&DateTmp,RTC_FORMAT_BIN);
}
```

RTC的报警定时器的使用

☒ Activate Calendar

Alarm A

Internal Alarm A

Alarm B

Disable

☒ Parameter Settings

☒ User Constants

☒ NVIC Settings

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
RTC alarm interrupt through EXTI line 17	<input checked="" type="checkbox"/>	1	0

Year	0
Alarm A	
Hours	0
Minutes	0
Seconds	0
Sub Seconds	0
Alarm Mask Date Week day	Enable
Alarm Mask Hours	Disable
Alarm Mask Minutes	Disable
Alarm Mask Seconds	Disable
Alarm Sub Second Mask	All Alarm SS fields are masked.
Alarm Date Week Day Sel	Date
Alarm Date	1

其中Alarm Mask Date Week day 指报警忽略日期，必须要开，否则报警要匹配年月日，永远不能报警  
若这样配置，默认就会打开一次报警中断

使用回调，或放IQR（虽然我没试过）

```
void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc)
{
    char tmp[20]="";

    sprintf(tmp,"%02f+%.1f+%02d%02d%02d",Get_ADC(),k,Alarm.AlarmTime.Hours,Alarm.AlarmTime.Minutes,Alarm.AlarmTime.Seconds);
    HAL_UART_Transmit(&huart1,(uint8_t*)tmp,strlen(tmp),255);
}
```

需要再次配置报警时要注意！从MX\_RTC\_Init抄出来配置更好

```
RTC_AlarmTypeDef Alarm;
Alarm.Alarm=RTC_ALARM_A;
Alarm.AlarmTime.Hours=0;//报警时间
Alarm.AlarmTime.Minutes=0;
Alarm.AlarmTime.Seconds=0;
Alarm.AlarmTime.SubSeconds = 0x0;
Alarm.AlarmMask = RTC_ALARMMASK_DATEWEEKDAY;//这个是关键，要求忽略年月日!!!
Alarm.AlarmSubSecondMask = RTC_ALARMSUBSECONDMASK_ALL;
Alarm.AlarmDateWeekDaySel = RTC_ALARMDATEWEEKDAYSEL_DATE;
Alarm.AlarmDateWeekDay = 0x1;
Alarm.Alarm = RTC_ALARM_A;//指定哪个报警器
```

最后启动中断

```
HAL_RTC_SetAlarm_IT(&hrtc,&Alarm,RTC_FORMAT_BIN);//启动中断
```

板载RES（电位器）

TABLE 6-2: STEP RESISTANCES

Part Number	Resistance (Ω)		
	Case	Total (R <sub>AB</sub> )	Step (R <sub>S</sub> )
MCP4017/18/19-502E	Min.	4000	31.496
	Typical	5000	39.370
	Max.	6000	47.244
MCP4017/18/19-103E	Min.	8000	62.992
	Typical	10000	78.740
	Max.	12000	94.488
MCP4017/18/19-503E	Min.	40000	314.961
	Typical	50000	393.701
	Max.	60000	472.441
MCP4017/18/19-104E	Min.	80000	629.921
	Typical	100000	787.402
	Max.	120000	944.882

必须要修改i2c的delay函数

```
/**
 * @brief I2C的短暂延时
 * @param None
 * @retval None
 */
extern TIM_HandleTypeDef htim7;
static void delay1(uint32_t nus)
{
    uint16_t differ = 0xffff-nus-500;
    //设置定时器2的技术初始值
    __HAL_TIM_SetCounter(&htim7,differ);
    //开启定时器
    HAL_TIM_Base_Start(&htim7);

    while( differ<0xffff-500)
    {
        differ = __HAL_TIM_GetCounter(&htim7);
    };
    //关闭定时器
    HAL_TIM_Base_Stop(&htim7);
}
```

电位器的步长：0.787402千欧，0-127步

这玩意误差挺大，所以即便实测不准也是正常（实测方法为J15和J16接到欧姆表）

```
void write_resistor(uint8_t value)
{
    I2CStart();
    I2CSendByte(0x5E);
    I2CWaitAck();

    I2CSendByte(value);
    I2CWaitAck();
    I2CStop();
}

uint8_t read_resistor(void)
{
    uint8_t value;
    I2CStart();
    I2CSendByte(0x5F);
    I2CWaitAck();

    value = I2CReceiveByte();
    I2CSendNotAck();
    I2CStop();

    return value;
}

sprintf(buf, "  RES VAL:%.1fk  ", (0.78740*read_resistor()));
LCD_DisplayStringLine(Line8, (uint8_t *) (buf));
```

## 板载NE555频率采集

可以参考国赛拓展板的频率采集

或者附加内容里面的双路捕获（但是你只用一路就行）

## 国赛扩展板

国赛板子记得看跳线帽

## 数码管

跳线帽：SCK, RCK, RES

io：不需要配置，在SEG\_Init会做

复制文件：seg.c

初始化函数：main.c

```
SEG_Init();
```

使用：

```
SEG_DisplayValue(1,1,1); //1,1,1  
SEG_DisplayValue(10,10,10); //A,A,A  
SEG_DisplayValue(16,16,16); //全灭，达到灭灯的效果!!!
```

## BUTTON(ADC)

跳线帽：AKEY

io：需要配置ADC2，IN13 Single-ended

复制文件：button.c，其中getADC()函数需要自己写

使用：

```
u8 key_val = Scan_Btn();
```

## 温度传感器（DS18B20）

跳线帽：TDQ

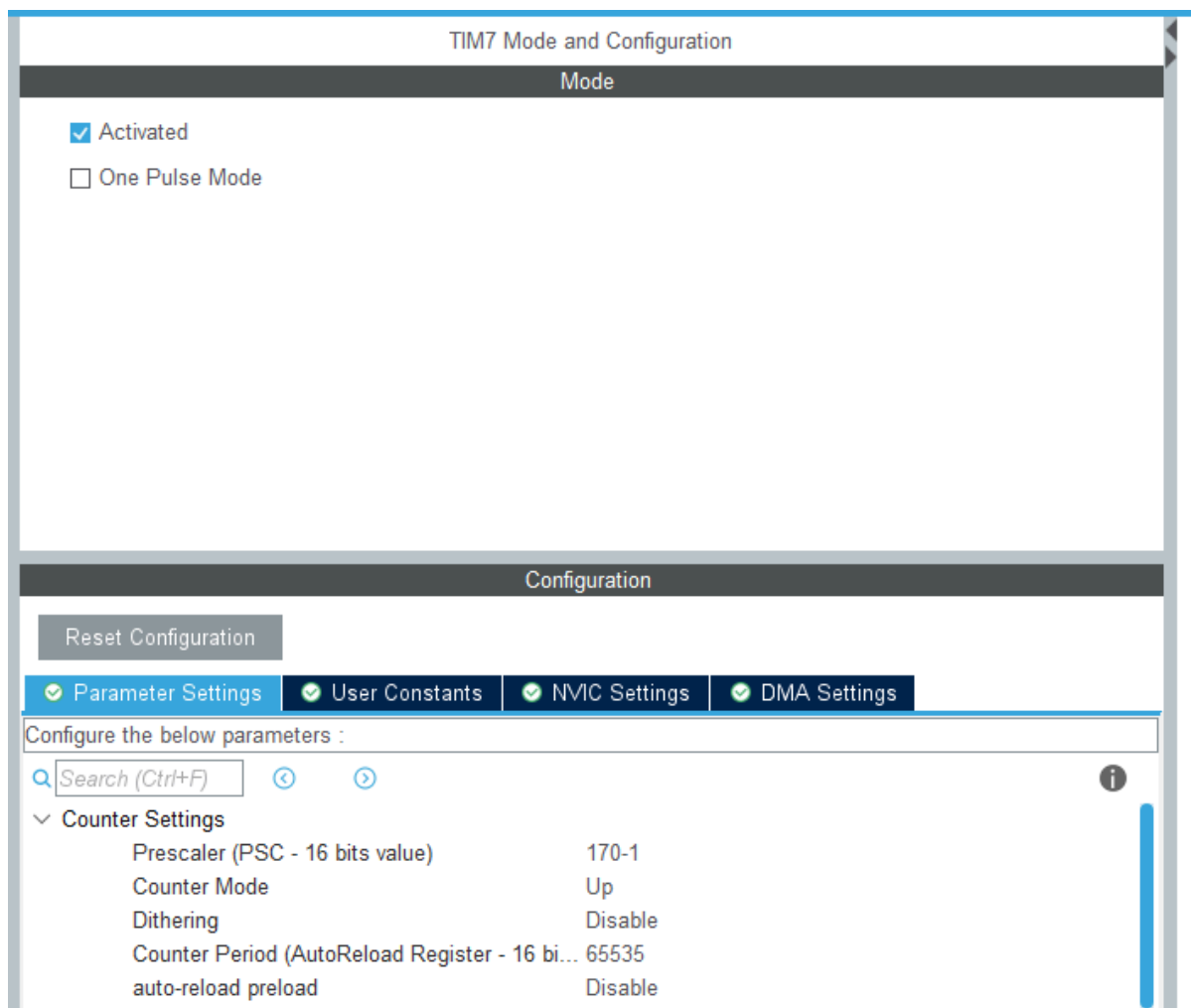
io：不需要配置，在ds18b20\_init\_x会做

复制文件：ds18b20.c

使用：（视情况可能需要改写其中的u8 u16 u32等类型）

如果需要上外部晶振，需要自己重写delay

开TIM7，1us分频，无需中断



```
static void delay_us(uint32_t nus)
{
    uint16_t differ = 0xffff-nus-500;
    //设置定时器2的技术初始值
    __HAL_TIM_SetCounter(&htim7,differ);
    //开启定时器
    HAL_TIM_Base_Start(&htim7);

    while( differ<0xffff-500)
    {
        differ = __HAL_TIM_GetCounter(&htim7);
    };
    //关闭定时器
    HAL_TIM_Base_Stop(&htim7);
}
```

理论上需要自己写读取函数

```
uint16_t ds18b20_read(void)
{
    uint8_t val[2];
    uint8_t i = 0;

    uint16_t x = 0;
```

```

ow_reset();
ow_byte_wr(OW_SKIP_ROM);
ow_byte_wr(DS18B20_CONVERT);
delay_us(750000);

ow_reset();
ow_byte_wr( OW_SKIP_ROM );
ow_byte_wr ( DS18B20_READ );

for ( i = 0 ; i < 2; i++)
{
    val[i] = ow_byte_rd();
}

x = val[1];
x <=< 8;
x |= val[0];

return x;
}

```

初始化方法

```

ds18b20_init_x();
read = (ds18b20_read() & 0x07FF);
tem = read / 16.;

```

## 温湿度传感器 (DHT11)

跳线帽: HDQ

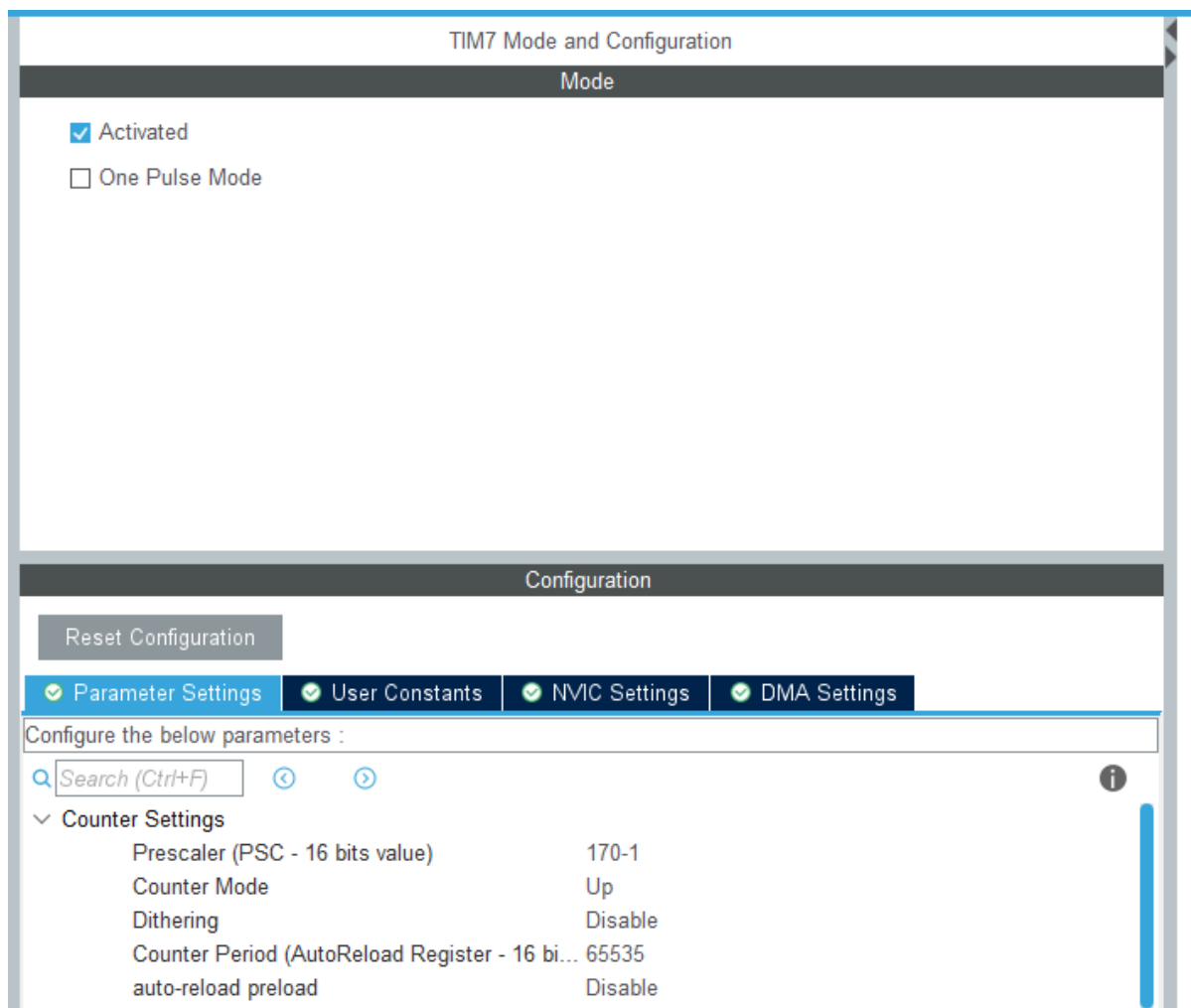
io: 不需要配置, 在DHT11\_Init会做

复制文件: dht11.c

使用: (视情况可能需要改写其中的u8 u16 u32等类型)

如果需要上外部晶振, 需要自己重写delay

开TIM7, 1us分频, 无需中断



```
static void delay_us(uint32_t nus)
{
    uint16_t differ = 0xffff-nus-500;
    //设置定时器2的技术初始值
    __HAL_TIM_SetCounter(&htim7,differ);
    //开启定时器
    HAL_TIM_Base_Start(&htim7);

    while( differ<0xffff-500)
    {
        differ = __HAL_TIM_GetCounter(&htim7);
    };
    //关闭定时器
    HAL_TIM_Base_Stop(&htim7);
}
```

检查两个delay，改为后面注释的us数

```
void DHT11_Rst(void)
    delay_us(40);          //主机拉高20~40us
uint8_t DHT11_Read_Bit(void)
    delay_us(40); //等待40us
```

理论上需要自己写读取函数

```
//从DHT11读取一次数据
uint8_t DHT11_Read_Data(uint8_t *temp, uint8_t *humi)
{
    uint8_t buf[5];
    uint8_t i;
    DHT11_Rst();
    if(DHT11_Check() == 0)
    {
        for(i = 0; i < 5; i++)
        {
            buf[i] = DHT11_Read_Byte();
        }
        if((buf[0] + buf[1] + buf[2] + buf[3]) == buf[4])
        {
            *humi = buf[0];
            *temp = buf[2];
        }
    }
    else return 1;
    return 0;
}
```

初始化方法：

```
u8 temper ;
u8 humi ;
DHT11_Init();
DHT11_Read_Data(&temper, & humi);
```

## MEMS传感器(LIS302DL)

???

## 光敏电阻 (DO)

跳线帽：TRDO

io：配置PA3为输入口

使用：低电平为亮，高电平为不亮

```

if(!HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_3) )
{
    LCD_DisplayStringLine(Line7, (u8 *)"          DO:High      ");
}
else
{
    LCD_DisplayStringLine(Line7, (u8 *)"          DO:Low       ");
}

```

## 光敏电阻 (AO)

跳线帽: TRAO

io: 配置ADC2 IN17 Single-end

使用:

```

tmp = getADC();
sprintf((char *)str, " R-P:%.2fk ", tmp / (4096. - tmp) * 10);

```

## AD采集×2

跳线帽: AO1 AO2 对应RP5 RP6 对应PA4 PA5

io: 配置ADC2 IN17 Single-end

使用:

## 官方方法

(例程的写法多少有些暴力)

```

uint16_t getADC_RP5(void)
{
    ADC_ChannelConfTypeDef sConfig = {0};
    uint16_t adc = 0;
    sConfig.Channel = ADC_CHANNEL_13;
    sConfig.Rank = ADC_REGULAR_RANK_1;
    sConfig.SamplingTime = ADC_SAMPLETIME_640CYCLES_5;
    sConfig.SingleDiff = ADC_SINGLE_ENDED;
    sConfig.OffsetNumber = ADC_OFFSET_NONE;
    sConfig.Offset = 0;
    if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
    {
        Error_Handler();
    }
    HAL_ADC_Start(&hadc2);
    adc = HAL_ADC_GetValue(&hadc2);

    return adc;
}

```

```

uint16_t getADC_RP6(void)
{
    ADC_ChannelConfTypeDef sConfig = {0};
    uint16_t adc = 0;
    sConfig.Channel = ADC_CHANNEL_17;
    sConfig.Rank = ADC_REGULAR_RANK_1;
    sConfig.SamplingTime = ADC_SAMPLETIME_640CYCLES_5;
    sConfig.SingleDiff = ADC_SINGLE_ENDED;
    sConfig.OffsetNumber = ADC_OFFSET_NONE;
    sConfig.Offset = 0;
    if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
    {
        Error_Handler();
    }
    HAL_ADC_Start(&hadc2);
    adc = HAL_ADC_GetValue(&hadc2);

    return adc;
}

```

## AD另法，连续循环DMA

Configuration

Reset Configuration

Parameter Settings User Constants NVIC Settings DMA Settings GPIO Settings

Configure the below parameters :

Search (Ctrl+F)

ADCs\_Common\_Settings

Mode Independent mode

ADC\_Settings

Clock Prescaler Synchronous clock mode divided by 4

Resolution ADC 12-bit resolution

Data Alignment Right alignment

Gain Compensation 0

Scan Conversion Mode Enabled 扫描模式 (自动开的)

End Of Conversion Selection End of single conversion

Low Power Auto Wait Disabled

Continuous Conversion Mode Enabled 连续模式

Discontinuous Conversion Mode Disabled

DMA Continuous Requests Enabled DMA请求

Overrun behaviour Overrun data preserved

ADC\_Regular\_ConversionMode

Enable Regular Conversions Enable

Enable Regular Oversampling Disable 改为通道数，改了以后扫描模式自动启动

Number Of Conversion 2

External Trigger Conversion Source Regular Conversion launched by software

External Trigger Conversion Edge None

Rank 1

Channel Channel 13 低于这个数，可能会

Sampling Time 24.5 Cycles 导致程序运行卡死

Offset Number No offset

Rank 2

Channel Channel 17 记得改成不一样的通

Sampling Time 24.5 Cycles 道，否则数据一样了

Offset Number No offset

Configuration

Reset Configuration

✓ Parameter Settings
✓ User Constants
✓ NVIC Settings
✓ DMA Settings
✓ GPIO Settings

DMA Request	Channel	Direction	Priority
ADC2	DMA1 Channel 1	Peripheral To Memory	Low

Add Delete

DMA Request Settings

循环模式，否则会溢出

Mode	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">Circular ▼</div>	Increment Address	<input type="checkbox"/>
		Peripheral	<input type="checkbox"/>
		Memory	<input checked="" type="checkbox"/>
Data Width	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">Half Word ▼</div>		<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">Half Word ▼</div>

DMA Request Synchronization Settings

Enable synchronization ☐

```

uint16_t testbuffer[2]={0};
HAL_ADCEX_Calibration_Start(&hadc2,ADC_SINGLE_ENDED);
HAL_ADC_Start_DMA(&hadc2,(uint32_t*)&testbuffer,2);
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    //LCD
    sprintf(tmp,"v1:%.02f",testbuffer[0]/4096.0*3.3);
    LCD_DisplayStringLine(Line1,(uint8_t*)tmp);
    sprintf(tmp,"v2:%.02f",testbuffer[1]/4096.0*3.3);
    LCD_DisplayStringLine(Line2,(uint8_t*)tmp);

}
/* USER CODE END 3 */

```

记得是u16存放数据，传进去强转u32

啥？程序好卡(没卡死)？试着把DMA采集ADC的函数注释掉试试。

如果不卡了，那证明就是DMA中断跑的太欢了

关闭DMA全部中断

```

void MX_DMA_Init(void)
{

    /* DMA controller clock enable */
    __HAL_RCC_DMAMUX1_CLK_ENABLE();

```

```

__HAL_RCC_DMA1_CLK_ENABLE();

/* DMA interrupt init */
/* DMA1_Channel1_IRQn interrupt configuration *///注释掉下面这些
// HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
// HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
// /* DMA1_Channel2_IRQn interrupt configuration */
// HAL_NVIC_SetPriority(DMA1_Channel2_IRQn, 0, 0);
// HAL_NVIC_EnableIRQ(DMA1_Channel2_IRQn);

}

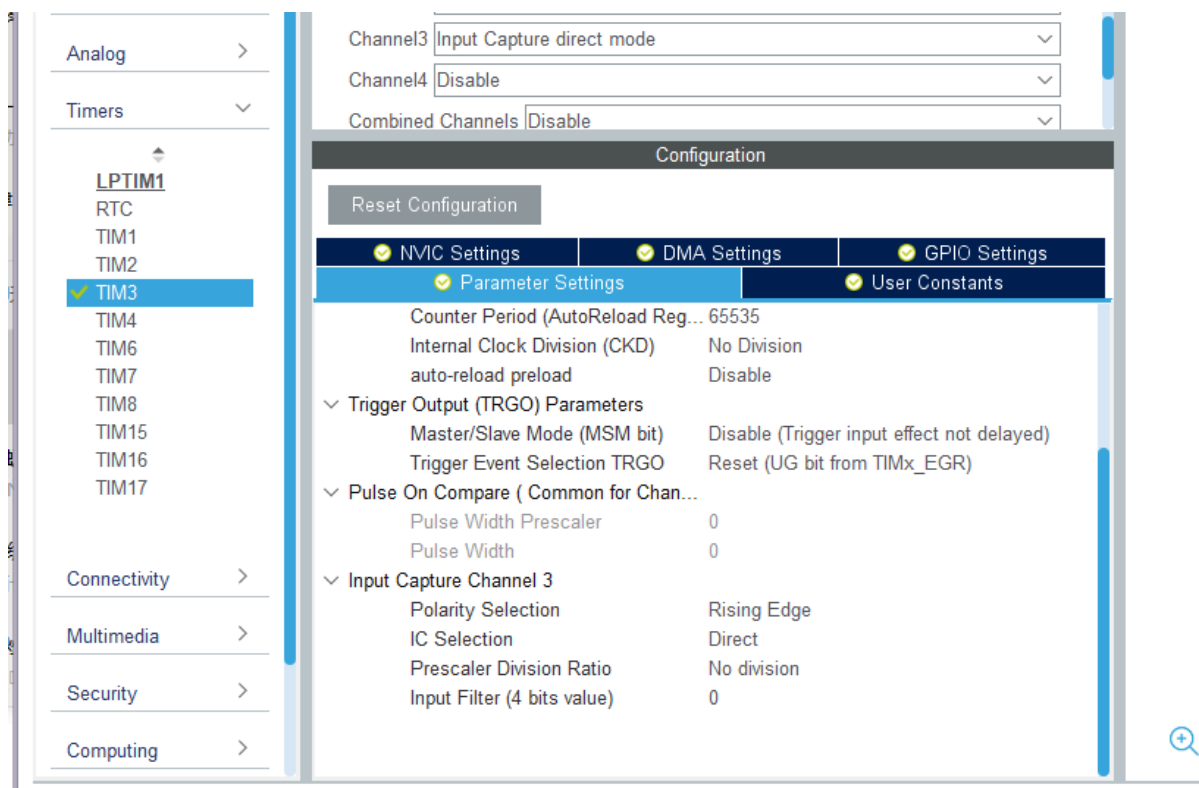
```

或者直接你可以找找如何默认不开启DMA（在NVIC里）

## 占空比采集

跳线帽：PWM1 PWM2 对应旋钮RP1 RP2 对应PA6 PA7

io：配置TIM3 CH2 上升沿 PSR：170-1 ARR：65535 自动重装关，这样使得CNT为1us加一次



使用：

main.c

```

//启动一次中断先
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
uint32_t cc1_value_2 = 0; // TIMX_CCR1 的值
uint32_t RP2 = 0;

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    cc1_value_2 = __HAL_TIM_GET_COUNTER(&htim3);
    __HAL_TIM_SetCounter(&htim3, 0);
    RP2 = 1000000 / cc1_value_2; //频率
}

```

```

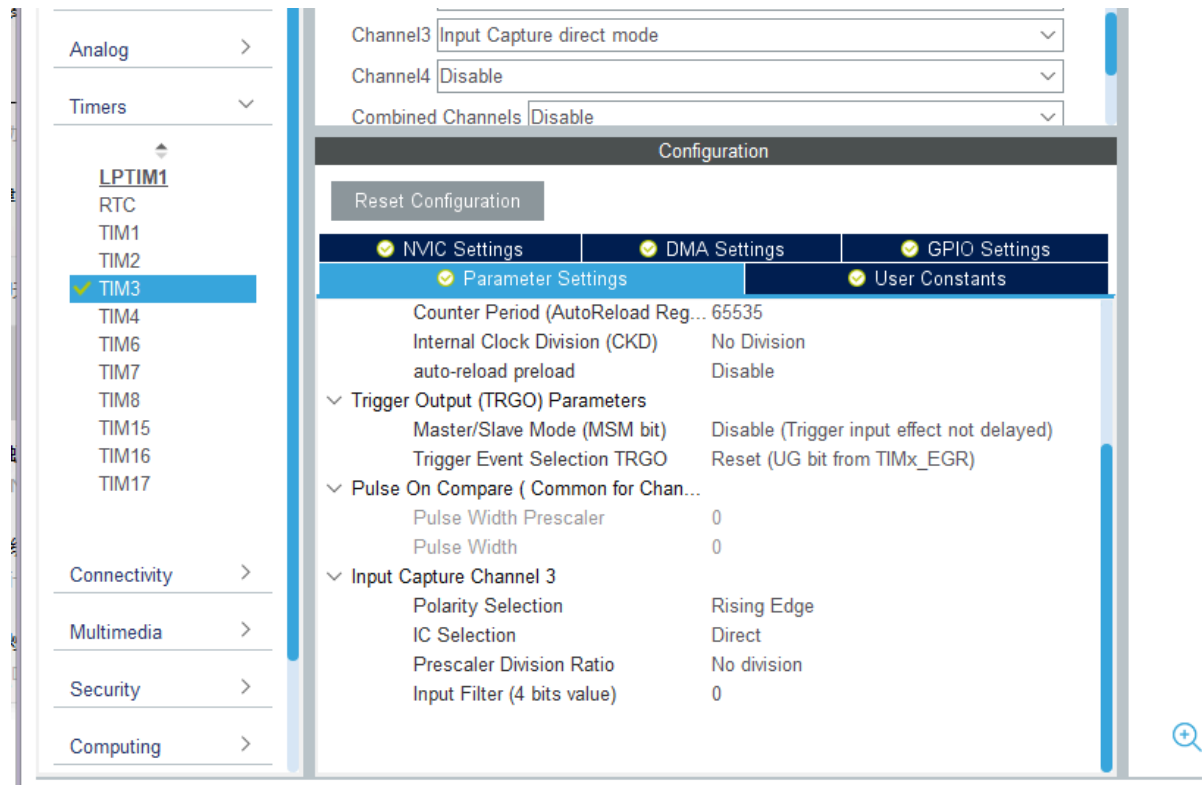
    HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);
}

```

## 频率采集

跳线帽：PLUSE1 PLUSE2 对应RP3 RP4 对应PA1 PA2

io：配置TIM3 CH2 上升沿 PSR：160-1 ARR：65535 自动重装关，这样使得CNT为1us加一次



使用：

main.c

```

//启动一次中断先
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
uint32_t cc1_value_2 = 0; // TIMx_CCR1 的值
uint32_t RP2 = 0;

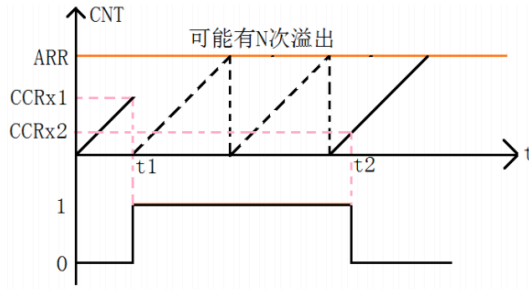
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    cc1_value_2 = __HAL_TIM_GET_COUNTER(&htim3);
    __HAL_TIM_SetCounter(&htim3, 0);
    RP2 = 1000000 / cc1_value_2; //频率

    HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);
}

```

原理如下：

溢出时间计算：



t1时刻检测到高电平，发生中断，在中断里将计数值置0，开始记溢出次数N，

其中每计数0xFFFF次溢出一次，直到t2时刻跳变回低电平，

获取最后一次溢出时到t2时刻的计数值TIM5CH1\_CAPTURE\_VAL

则 高电平时间 = 溢出次数\*65535+TIM5CH1\_CAPTURE\_VAL us；根据定时器初始化时的频率即可计算出溢出总次数所占用的时间，即为高电平时间。

如果计数器值为 32 bit 那么最大为0xFFFFFFFF

高电平时间：

$$T = N * 0xFFFFFFFF + captureVal$$

此时CNT不能用TIM->CNT读出，而是读CCR1，CCR2（此时CCR不再是PWM输出时不变的情况）

所以上述程序要求所捕获的PWM频率不得低于1s/65535us，即15.25Hz，否则由于出现重装不准确，也同时根据采样定理不能高于1s/2us，即500KHz

## 附加内容

### 输出双路PWM不同频率与占空比

```
static int PC1=100,PC4=300,CC1=0,CC4=0;//频率，临时变量
static double ZC1=0.3,ZC4=0.4;//占空比%
static u8 C1=0,C4=0;//高低电平转换
void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim)//重写回调
{
    if(htim->Instance==htim2.Instance)//哪个tim
    {
        if(htim->Channel==HAL_TIM_ACTIVE_CHANNEL_1)//哪个通道
        {
            if(C1)
            {
                C1=0;//高低电平转换
                CC1=__HAL_TIM_GET_COUNTER(&htim2)+(int)(1000000/PC1*(1-ZC1));
                if(CC1>99999)CC1-=99999;//防止CCR>ARR
                __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1,CC1);//设定下一个CCR
            }
            else
            {
                C1=1;
                CC1=__HAL_TIM_GET_COUNTER(&htim2)+(int)(1000000/PC1*(ZC1));
                if(CC1>99999)CC1-=99999;
                __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1,CC1);
            }
        }
        if(htim->Channel==HAL_TIM_ACTIVE_CHANNEL_4)
        {
            if(C4)
```

```

    {
        C4=0;
        CC4=__HAL_TIM_GET_COUNTER(&htim2)+(int)(1000000/PC4*(1-ZC4));
        if(CC4>99999)CC4-=99999;
        __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_4,CC4);
    }
    else
    {
        C4=1;
        CC4=__HAL_TIM_GET_COUNTER(&htim2)+(int)(1000000/PC4*(ZC4));
        if(CC4>99999)CC4-=99999;
        __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_4,CC4);
    }
}
}
}

```

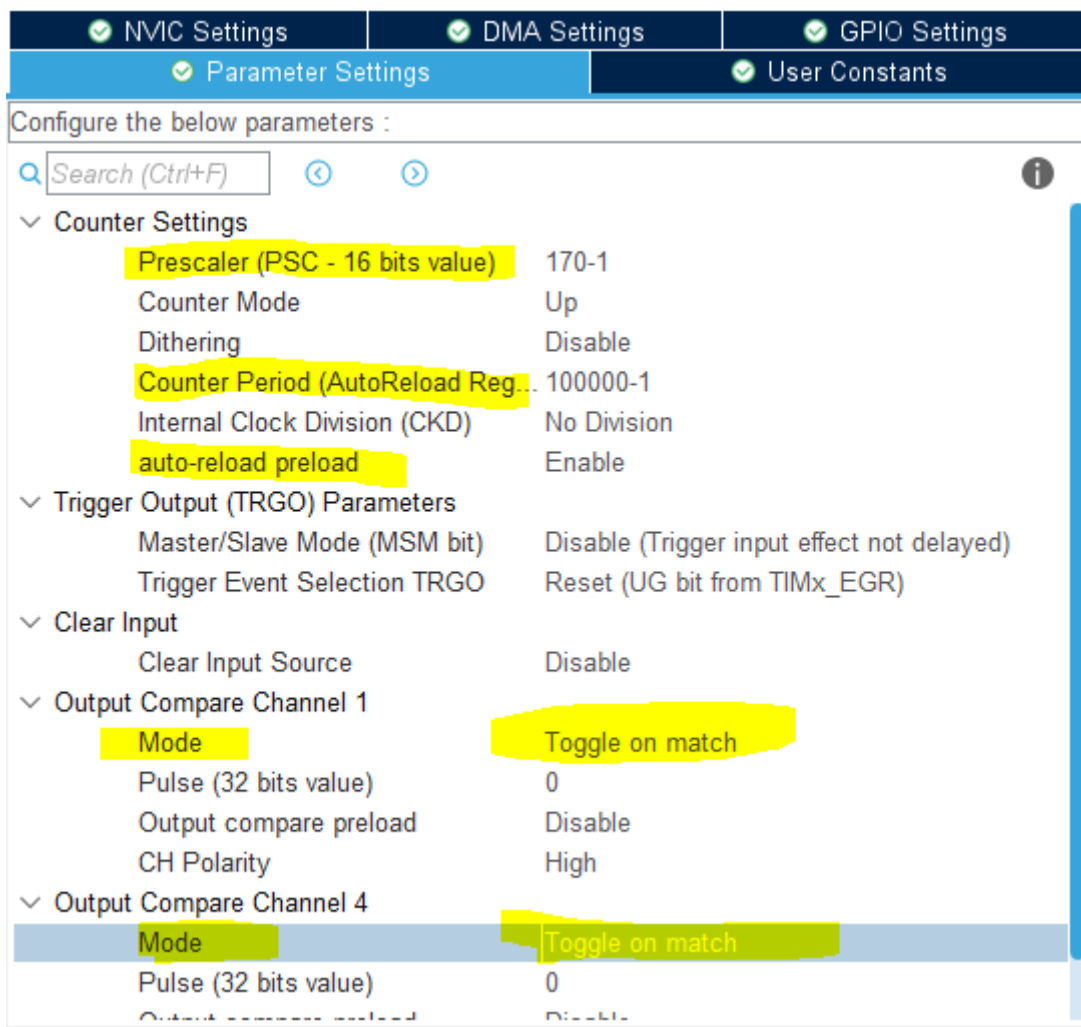
不使用重写回调函数的话，改在IQR里写测试过不行，原因未知

The screenshot shows the 'TIM2 Mode and Configuration' window in STM32CubeMX. The 'Mode' tab is active, displaying various configuration options for the TIM2 peripheral. The settings are as follows:

- Slave Mode:** Disable
- Trigger Source:** Disable
- Clock Source:** Internal Clock
- Channel1:** Output Compare CH1
- Channel2:** Disable
- Channel3:** Disable
- Channel4:** Output Compare CH4
- Combined Channels:** Disable
- Use ETR as Clearing Source:** Disable
- ☐ XOR activation
- ☐ One Pulse Mode

The 'Configuration' tab is visible at the bottom of the window.

OC CH1别开错



记得OC的Mode要调整!!!

最后启动OC中断

```
HAL_TIM_OC_Start_IT(&htim2, TIM_CHANNEL_1);
HAL_TIM_OC_Start_IT(&htim2, TIM_CHANNEL_4);
```

直接启动OC IT即可，无需再启动OC

## 捕获双路PWM

```
u8 CH2Step = 1, CH3Step = 1;
int CH2Buf[3], CH3Buf[3];
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2) //这里要选择active否则进不去
    {
        if (CH2Step == 1)
        {
            CH2Buf[0] = HAL_TIM_ReadCapturedValue(&htim2, TIM_CHANNEL_2);
            __HAL_TIM_SET_CAPTUREPOLARITY(&htim2, TIM_CHANNEL_2,
TIM_ICPOLARITY_FALLING);
            CH2Step++;
        }
        else if (CH2Step == 2)
```

```

    {
        CH2Buf[1] = HAL_TIM_ReadCapturedValue(&htim2, TIM_CHANNEL_2);
        __HAL_TIM_SET_CAPTUREPOLARITY(&htim2, TIM_CHANNEL_2,
TIM_ICPOLARITY_RISING);
        CH2Step++;
    }
    else if (CH2Step == 3)
    {
        CH2Buf[2] = HAL_TIM_ReadCapturedValue(&htim2, TIM_CHANNEL_2);
        __HAL_TIM_SET_CAPTUREPOLARITY(&htim2, TIM_CHANNEL_2,
TIM_ICPOLARITY_FALLING);
        CH2Step++;
        HAL_TIM_IC_Stop_IT(&htim2, TIM_CHANNEL_2);
    }
}
else
{
    if (CH3Step == 1)
    {
        CH3Buf[0] = HAL_TIM_ReadCapturedValue(&htim2, TIM_CHANNEL_3);
        __HAL_TIM_SET_CAPTUREPOLARITY(&htim2, TIM_CHANNEL_3,
TIM_ICPOLARITY_FALLING);
        CH3Step++;
    }
    else if (CH3Step == 2)
    {
        CH3Buf[1] = HAL_TIM_ReadCapturedValue(&htim2, TIM_CHANNEL_3);
        __HAL_TIM_SET_CAPTUREPOLARITY(&htim2, TIM_CHANNEL_3,
TIM_ICPOLARITY_RISING);
        CH3Step++;
    }
    else if (CH3Step == 3)
    {
        CH3Buf[2] = HAL_TIM_ReadCapturedValue(&htim2, TIM_CHANNEL_3);
        __HAL_TIM_SET_CAPTUREPOLARITY(&htim2, TIM_CHANNEL_3,
TIM_ICPOLARITY_FALLING);
        CH3Step++;
        HAL_TIM_IC_Stop_IT(&htim2, TIM_CHANNEL_3);
    }
}
}
}

```

```

int main(void)
{
    HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_2);
    HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_3);
    double fp2 = 0, fp3 = 0;
    double zhan2 = 0, zhan3 = 0;
    while (1)
    {
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */

```

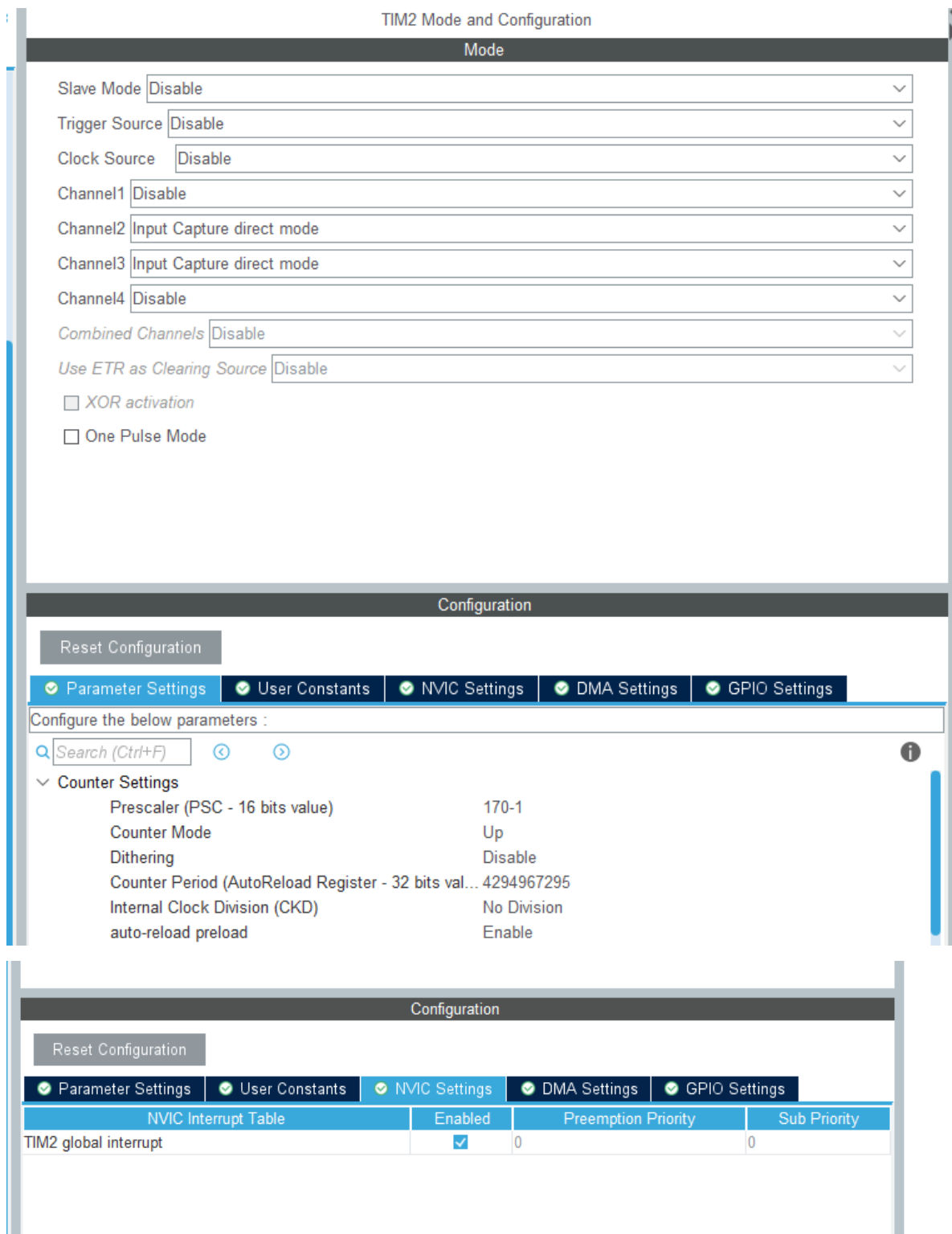
```

if (CH2Step == 4)
{
    CH2Step = 1;
    fp2 = 1/((double)(CH2Buf[2] - CH2Buf[0])/1000000.0);
    zhan2 = (double)(CH2Buf[1] - CH2Buf[0]) / (double)(CH2Buf[2] - CH2Buf[0]);
    memset(CH2Buf, 0, sizeof(CH2Buf));
    //    __HAL_TIM_SET_COUNTER(&htim2, 0);
    __HAL_TIM_SET_CAPTUREPOLARITY(&htim2, TIM_CHANNEL_2,
TIM_ICPOLARITY_RISING);
    HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_2);
}
if (CH3Step == 4)
{
    CH3Step = 1;
    fp3 = 1/((double)(CH3Buf[2] - CH3Buf[0])/1000000.0);
    zhan3 = (double)(CH3Buf[1] - CH3Buf[0]) / (double)(CH3Buf[2] - CH3Buf[0]);
    __HAL_TIM_SET_CAPTUREPOLARITY(&htim2, TIM_CHANNEL_3,
TIM_ICPOLARITY_RISING);
    HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_3);
}
}

```

使用的是TIM2，拥有u32的ARR计数器，所以可测量的频率下限很低很低，也不需要考虑更新中断带来的影响

记得测试时高电平要3.2v左右，2.2v测试时会乱跳

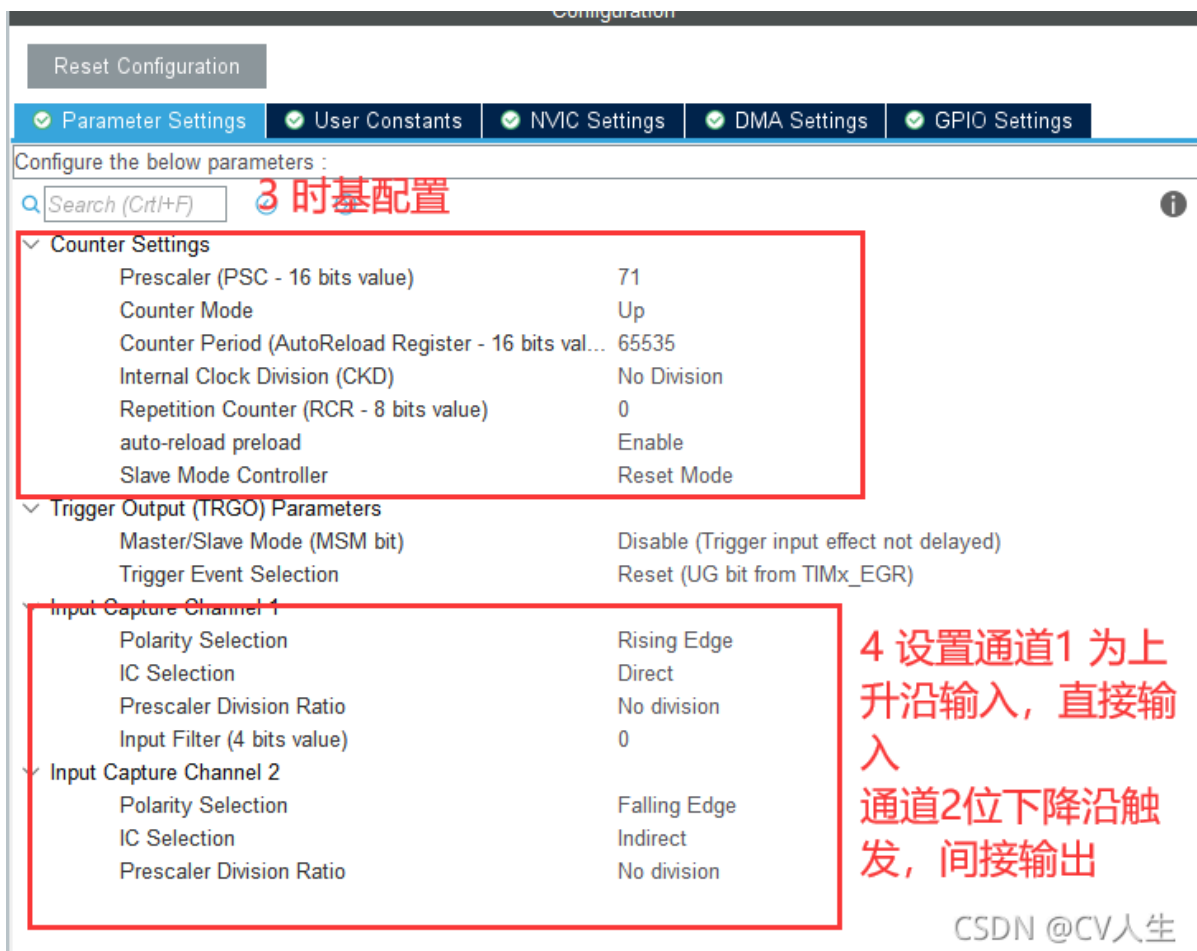
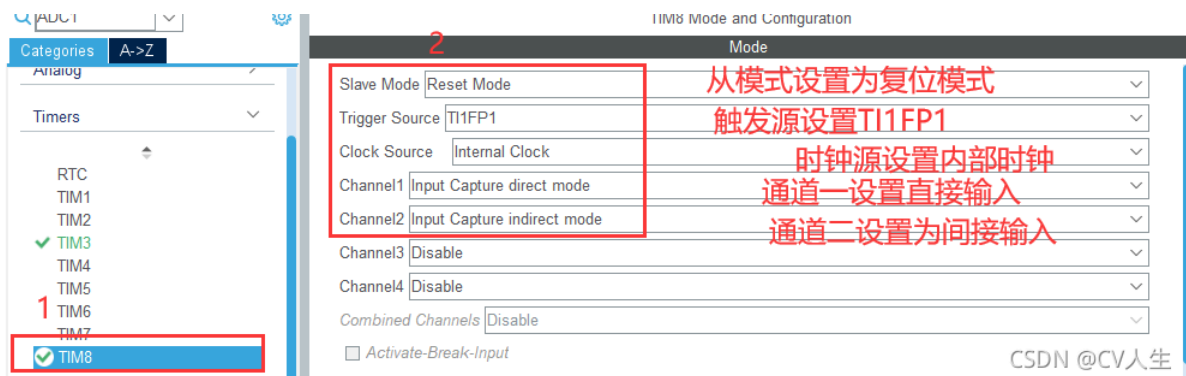


## 另一种PWM输入捕获的办法（仅单路，且存在硬件限制但不多）

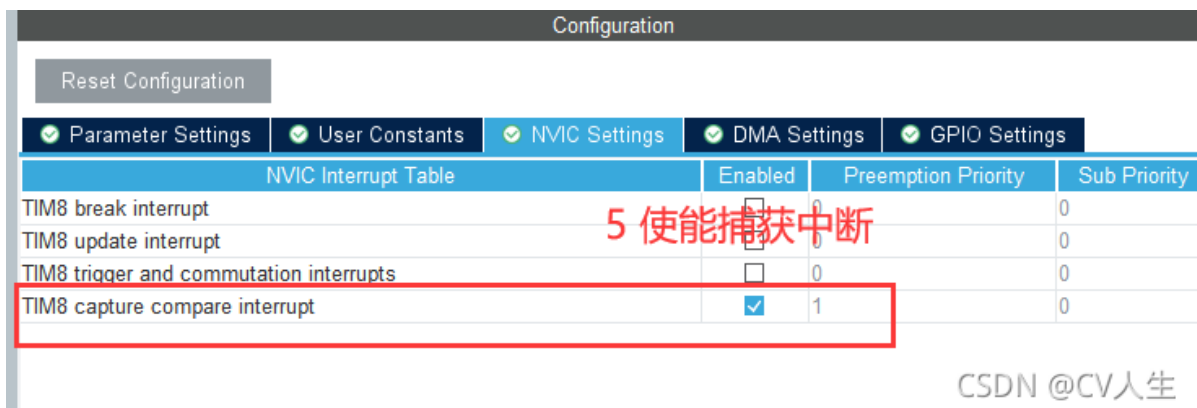
同时占用CH1CH2通道

CH1采集输入

[https://blog.csdn.net/qg\\_29031103/article/details/120023388](https://blog.csdn.net/qg_29031103/article/details/120023388)



(此处时PSC自己改为160-1等，保证为1us)



Configuration

Reset Configuration

Parameter Settings

User Constants

NVIC Settings

DMA Settings

GPIO Settings

Search Signals

Search (Ctrl+F)

6 确认IO口

☐ Show only Modified Pins

Pin Name	Signal on Pin	GPIO output ...	GPIO mode	GPIO Pull-up...	Maximum ou...	User Label	Modified
PC6	TIM8_CH1	n/a	Input mode	No pull-up an...	n/a		<input type="checkbox"/>

CSDN @CV人生

此处只有一个口才是对的，才是对的

```
//初始化
HAL_TIM_IC_Start_IT(&htim8,TIM_CHANNEL_1);
HAL_TIM_IC_Start_IT(&htim8,TIM_CHANNEL_2);
while(1)
{
    sprintf(tmp, "  %.2f  %.2f  ", Duty, Frequency);
    LCD_DisplayStringLine(Line8, (uint8_t *) (tmp));
}

//中断回调
float Duty, Frequency;
int Cap_val1, Cap_val2;
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Channel==HAL_TIM_ACTIVE_CHANNEL_1)
    {
        Cap_val1=HAL_TIM_ReadCapturedValue(&htim4,TIM_CHANNEL_1);
        Cap_val2=HAL_TIM_ReadCapturedValue(&htim4,TIM_CHANNEL_2);
        if(Cap_val1!=0)
        {
            Duty=(float)(Cap_val2+1)*100/(Cap_val1+1);
            F=170000000/170/(float)(Cap_val1+1);
        }
    }
}
```

那如果是CH2接收呢？

### TIM2 Mode and Configuration

Mode

Slave Mode	Reset Mode	▼
Trigger Source	Tl2FP2	▼
Clock Source	Internal Clock	▼
Channel1	Input Capture indirect mode	▼
Channel2	Input Capture direct mode	▼
Channel3	Disable	▼
Channel4	Disable	▼
Combined Channels	Disable	▼
Use ETR as Clearing Source	Disable	▼

☐ XOR activation

☐ One Pulse Mode

### Configuration

Reset Configuration

✓ Parameter Settings
✓ User Constants
✓ NVIC Settings
✓ DMA Settings
✓ GPIO Settings

Configure the below parameters :

i

- ▼ Counter Settings
 

Prescaler (PSC - 16 bits value)	160-1
Counter Mode	Up
Dithering	Disable
Counter Period (AutoReload Register - 32 bit...	4294967295
Internal Clock Division (CKD)	No Division
auto-reload preload	Enable
Slave Mode Controller	Reset Mode
- ▼ Trigger Output (TRGO) Parameters
 

Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection TRGO	Reset (UG bit from TIMx_EGR)
- ▼ Input Capture Channel 1
 

Polarity Selection	Falling Edge
IC Selection	Indirect
Prescaler Division Ratio	No division
- ▼ Input Capture Channel 2
 

Polarity Selection	Rising Edge
--------------------	-------------

▼ Input Capture Channel 1

Polarity Selection	Falling Edge	indirect的通道应该选下降沿
IC Selection	Indirect	
Prescaler Division Ratio	No division	

▼ Input Capture Channel 2

Polarity Selection	Rising Edge	如果上面选TIFP正确，那不该有一个可以填数据的选项
IC Selection	Direct	
Prescaler Division Ratio	No division	
Input Filter (4 bits value)	0	

✓ Parameter Settings
✓ User Constants
✓ NVIC Settings
✓ DMA Settings
✓ GPIO Settings

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM2 global interrupt	✓	0	0

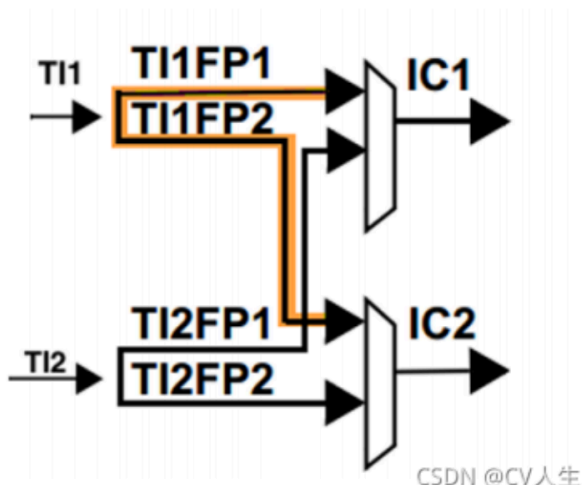


```
float Duty, Frequency;
int Cap_val1, Cap_val2;
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2) //对应输入的通道
    {
        /*读取寄存器的值*/
        Cap_val1 = HAL_TIM_ReadCapturedValue(&htim2, TIM_CHANNEL_2); //val1 对应输入的通道
        Cap_val2 = HAL_TIM_ReadCapturedValue(&htim2, TIM_CHANNEL_1);
        /*计算占空比，频率*/
        if (Cap_val1 != 0)
        {
            Duty = (float)(Cap_val2 + 1) * 100 / (Cap_val1 + 1);
            Frequency = 1000000 / (float)(Cap_val1 + 1);
        }
    }
}
```

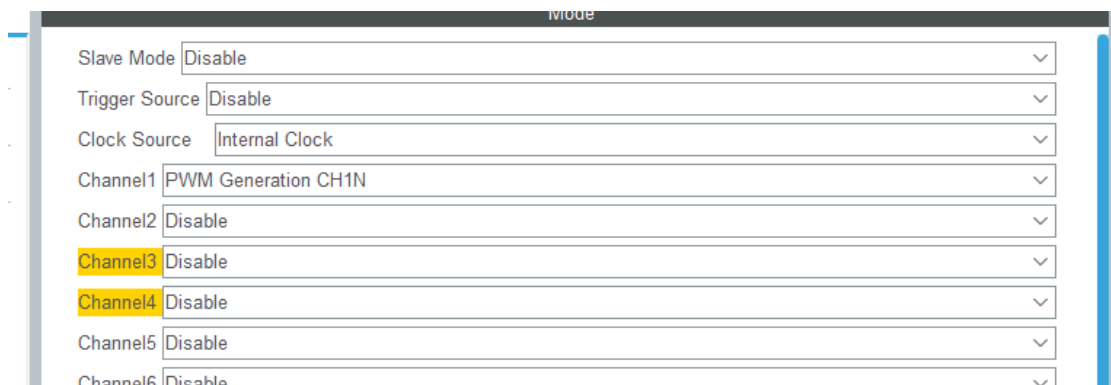
你可能想问：CH3CH4呢？不能用吗？

应该不能（不过原文用的是F103所以不能完全确定）

一、PWM模式只能使用通道1和通道2，PWM输入模式工作原理如下图：



## CHXN用法



然后使用下面的代码就可以输出正常的PWM了（极性正常，不用做额外修改）

```
HAL_TIMEx_PWMN_Start(&htim1,TIM_CHANNEL_1);
```

注意是TIMEx

## SRAM?

本届试题还出现了一个可能困惑大家的地方：SRAM至少记录100条数据。

2) 要求至少可记录 100 条数据，记录的数据保存在微控制器 SRAM 存储器中。

CSDN @黑心萝卜三条杠

小编乍一看，还以为是需要将数据存储在一个掉电也不会丢失的内存中呢。（小编忘记ROM与RAM的区别啦😂😂😂）百度后才知道：SRAM，一种静态随机存取存储器只要保持通电，里面储存的数据就可以恒常保持；掉电后，数据还是会消失，这与在断电后还能储存资料的ROM或闪存是不同的。所以，也就是说，将这100条以上的数据存储到数据中就好啦！😂😂😂

## 移位

```
&=~ (1<<3)
```

使得第四位清零

```
|=(1<<1)
```

使得第二位置位

```
^=(1<<2)
```

使得第三位翻转

## 使定时器接近精确的重设

```
TIM2->CNT=0;
TIM2->SR=0; //https://blog.csdn.net/weixin_44788542/article/details/113111139
HAL_TIM_Base_Start_IT(&htim2);
```

## 关于时钟

160MHz可能更好，因为能做到整除4,8等数据，而不是170MHz

## 关于屏幕显示

小心百分号！！

```
sprintf("%%")
```

## 关于自动重装的作用

如果开启了自动重装，新设置的ARR的值会在CNT=ARR的时候即触发更新中断时装载

即ARR下周期生效，这样的作用是如果不开启自动重装而你修改了ARR

且CNT>ARR时，你可能需要等ARR寄存器自然溢出，那在溢出前你的定时器会停摆

如果ARR是32bit计数器，那所有和该定时器相关的功能大概都寄了

## 细节：

---

### 第六届省赛

不同屏幕按键逻辑要分离，不能在屏幕1操作屏幕2

到屏幕二应该默认回到调整小时

好像还真没什么别的？，RTC自己配置好中断就好

### 第七届省赛

液位1不应该能比2高，以此类推，我觉得也不能等于

ADC是1秒才刷新一次，且第一次显示时候就要有值和level，而不是等待第一秒

LD2和LD3闪烁要分离，不能满足条件后同步闪

不同屏幕按键逻辑要分离，不能在屏幕1操作屏幕2

一上电不该是Level发生变化（因为这时候算是初值）

到屏幕二应该默认回到第一行突出

LD2和LD3应该做好需要重新闪5次的准备，不会出现需要先闪完当前的

（可能？）在上一条的基础上需要尽可能快开始下一次5次闪烁

给出我认为需要尽快的方法

LD2:

```

while(1)
{
    if(LEDFlag2==1)
    {
        LEDFlag2=0;
        HAL_GPIO_TogglePin(GPIOC,GPIO_PIN_9);
        HAL_GPIO_WritePin(GPIOD,GPIO_PIN_2,GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOD,GPIO_PIN_2,GPIO_PIN_RESET);
    }
}

```

```

void TIM3_IRQHandler(void)//0.2s中断，常开
{
    /* USER CODE BEGIN TIM3_IRQn 0 */

    /* USER CODE END TIM3_IRQn 0 */
    HAL_TIM_IRQHandler(&htim3);
    /* USER CODE BEGIN TIM3_IRQn 1 */
    if(flashTime2!=0)
    {
        flashTime2--;
        LEDFlag2=1;
    }
    /* USER CODE END TIM3_IRQn 1 */
}

```

```

while(1)
{
    if(Hlevel!=level)
    {
        if(Hlevel>level)
            sprintf(tmp,"A:H%d+L%d+D\r\n",nowHeight,level);
        else
            sprintf(tmp,"A:H%d+L%d+U\r\n",nowHeight,level);
        HAL_UART_Transmit(&huart1,(uint8_t*)tmp,strlen(tmp),255);
        Hlevel=level;
        HAL_GPIO_WritePin(GPIOC,GPIO_PIN_9,GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOD,GPIO_PIN_2,GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOD,GPIO_PIN_2,GPIO_PIN_RESET);//重置目前的灯状态
        TIM3->CNT=0;//确保不会在设定刷9次到打开标志位间进中断了
        flashTime2=9;//刷9次即可，由于下面把标志位打开了
        LEDFlag2=1;//打开标志位
    }
}

```

对LD3，由于在中断内不能使用LED，因此再用标志位拉出来while处理

```

void USART1_IRQHandler(void)
{
    /* USER CODE BEGIN USART1_IRQn 0 */

    /* USER CODE END USART1_IRQn 0 */
    HAL_UART_IRQHandler(&huart1);
    /* USER CODE BEGIN USART1_IRQn 1 */
    if(rec[0]=='c')
    {
        char tmp[30]="";
    }
}

```

```

        sprintf(tmp, "C:H%d+L%d\r\n", nowHeight, level);
        HAL_UART_Transmit(&huart1, (uint8_t*)tmp, strlen(tmp), 255);
        LD3=1;
    }
    else if(rec[0]=='s')
    {
        char tmp[30]="";
        sprintf(tmp, "S:TL%d+TM%d+TH%d\r\n", l1, l2, l3);
        HAL_UART_Transmit(&huart1, (uint8_t*)tmp, strlen(tmp), 255);
        LD3=1;
    }
    rec[0]=0;
    HAL_UART_Receive_IT(&huart1, (uint8_t*)rec, 1);
    /* USER CODE END USART1_IRQn 1 */
}

```

```

void TIM3_IRQHandler(void)//0.2s中断，常开
{
    /* USER CODE BEGIN TIM3_IRQn 0 */

    /* USER CODE END TIM3_IRQn 0 */
    HAL_TIM_IRQHandler(&htim3);
    /* USER CODE BEGIN TIM3_IRQn 1 */
    if(flashTime2!=0)
    {
        flashTime2--;
        LEDFlag2=1;
    }
    /* USER CODE END TIM3_IRQn 1 */
}

```

```

while(1)
    if(LD3==1)
    {
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_RESET);
        LD3=0;
        TIM4->CNT=0;
        flashTime3=9;
        LEDFlag3=1;
    }

```

```

while(1)
    if(LEDFlag3==1)
    {
        LEDFlag3=0;
        HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_10);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_RESET);
    }

```

## 第八届省赛

恭喜来到最难一届

逻辑实现本身就是一件很困难的事情了，建议用状态机

在电梯上升下降，开关门时不能按按键

最后一次按键按下后开始2秒计时

电梯是先上后下，即便在3楼也是先上4楼再下2，下1

控制电机转向的高低电平那里可以一直保持，直到下次需要更改（否则你想改成什么，高阻态？）

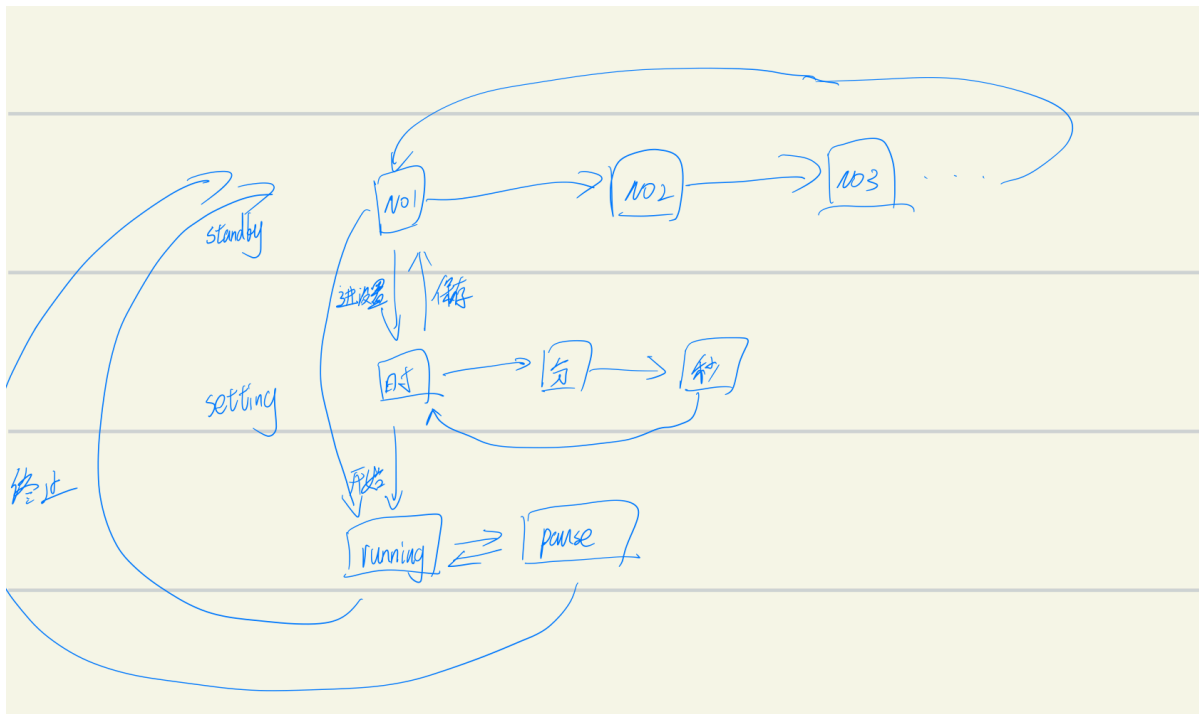
还好流水灯不会与按键冲突（因为两个不会同时运行，但是建议想想如果冲突了该怎么做）

记得到楼层闪两下

## 第九届省赛

这届题和上一届不分上下，具有一定挑战性

先给出我认为的逻辑情况



（关于在计时状态下能不能切换，这是个好问题）

（计时结束后是否该读取原来的计时前的时间状态，这也是个好问题，我觉得不用，否则你怎么知道运行完毕了）

屏幕逻辑分离（逻辑多了容易忘）

屏幕字符高亮不能一直闪动，但也不能该更新不更新

长按的处理，还好没有一边长按加数字一边闪的情况

但是存在终止定时器的长按时闪灯和屏幕计时显示的情况(如你在长按0.8秒内，你的灯0.5秒要开关不能停，如果屏幕倒计时建议秒也得显示)

切换后定时时间设定为当前（切换后的）位置存储的时间

不先设置也可以进入计时模式

计时暂停时，应该很精确，不该出现暂停计时后，重新计算这一秒或者跳过这一秒的剩余时间

推荐方法为HAL\_TIM\_Base\_Stop

## 第十届省赛

之前的太难了？来点简单的！

屏幕逻辑分离都已经写在那里了

注意按键不能重复，max不能小于或者等于min

注意测一下min是否会有小于0的显示出现

LED逻辑嘛，直接多点代码，能实现就行

还有就是灯的设置两个不能相同，但是又要能循环。建议多点测试，bug主要集中在边界（1和8）

## 第十一届省赛

还是太难了？再简单点

3.29v和0v建议处理一下

屏幕逻辑分离都已经写在那里了啊！

AUTP MANU大写哦

啊？还有啥难的？

## 第十二届省赛

太简单了？该来点又难又烦的东西了

串口收到的错误有很多情况，如下：

串口理论上是收22位，如果指令大于22或小于22，应该认定为错误更好（此时需要开定时器或空闲中断）

时间本身不合理，包括如下情况：

年是0-99

月1-12

天要考虑闰年和月份

小时0-23

分钟0-59

秒0-59

时间前后不合理：即入场时间应该早于出场时间，相等我觉得也不行（量子叠加态是吧）

出场时车的类型不合理，即入场C车出场V车

然后算时间，要算年月日，逻辑很复杂？那就对了！

放心，即便是100年，费用3.5也是一个double存的下的

所以是可以存在费用很高的情况

## 第十三届省赛

三次密码输入错误后，第四次算三次连续么？算的

第四次应该重新计时5秒

5秒内密码输入正确应停止LD2，B1234都不该能在sta界面使用

## 第十四届省赛

我觉得难点只在采集和输出波形的准确性

还有就是无信号时应该频率归0吧

最后百分号别漏了

仔细读题就好

## 第十四届国赛

持续处于报警状态不累加

从报警参数界面退出时，新的 FH、AH 和TH 参数生效（这个好理解）

从回放设置界面退出时，新的 FP、VP 和 TT 参数生效（这个不好理解它的作用，就是说在回放时该参数也得这样生效）

长按不影响屏幕（这个对国赛是基操了），也不影响回放

初始状态包括回到实时数据界面和退出回放模式

关于保障频率回放的准确性，我建议是分三段PSC，100以下，100-10000,10000以上

由于占空比要求精度为1%，换言之ARR知道得要是100以上

# 来自xtx的最后碎碎念

---

虽然但是，xtx也是拿过国一的

以下是几个最值得参考的程序

project2023\xtx\GuoSai\Base 这个文件夹里面是国赛（包含拓展板驱动的）基准编写项目文件夹（就是说没有功能只有驱动和模板）

project2023\xtx\GuoSai\14 国一程序

project2023\xtx\ShengSai\14 省一程序