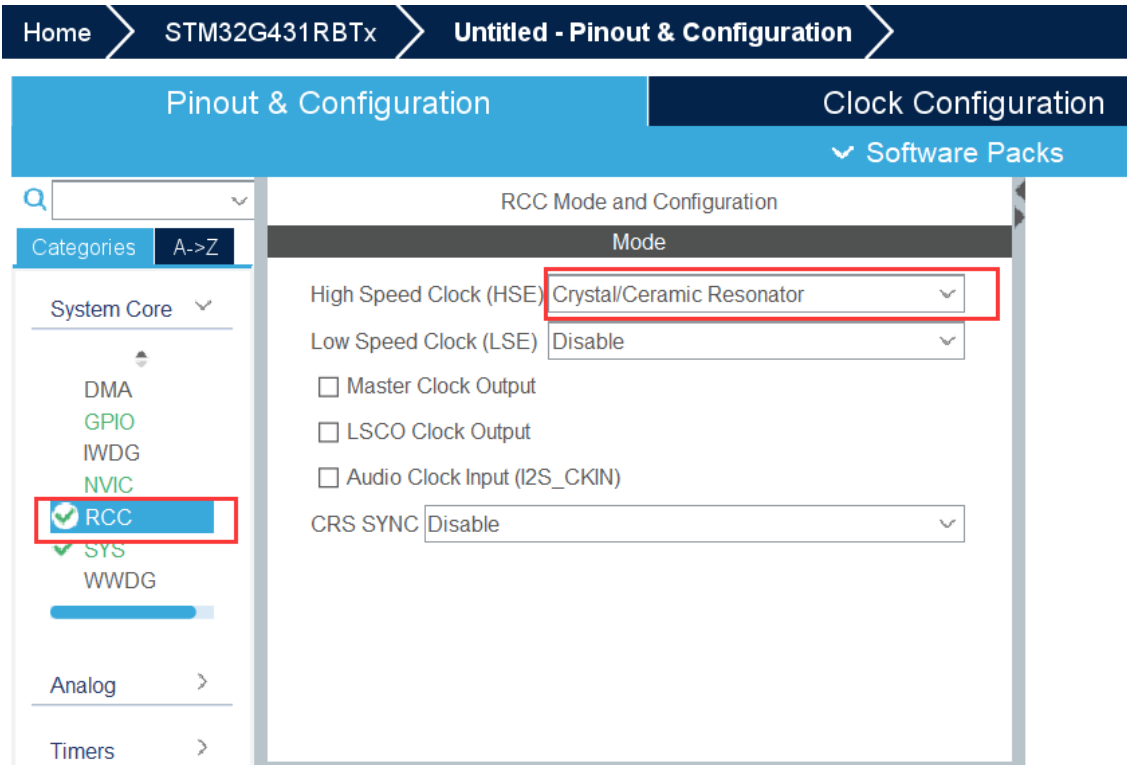
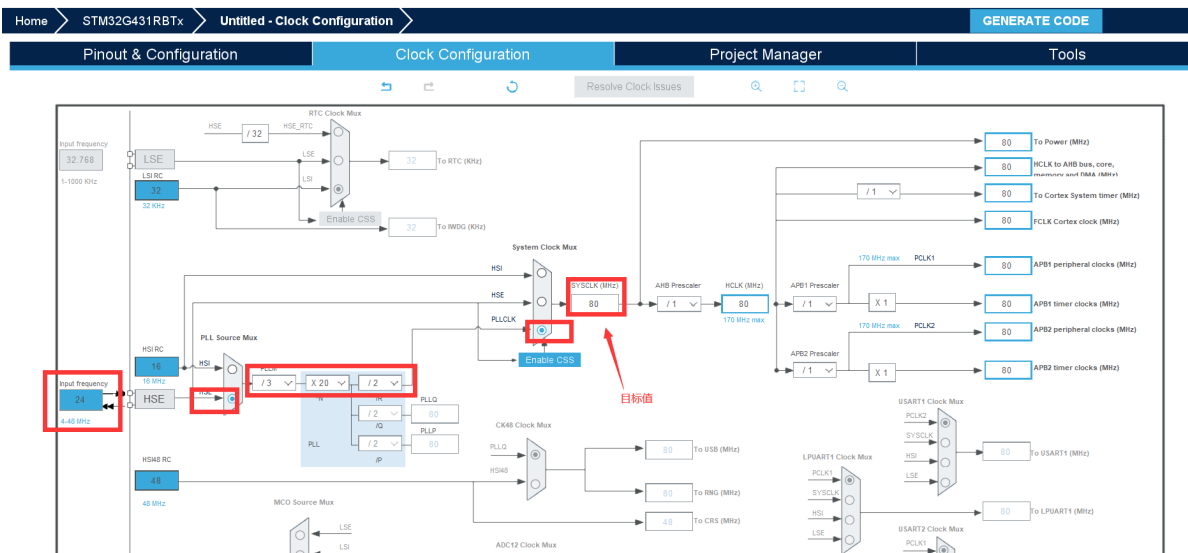


建立一个模板工程

- 步骤一：打开CubeMX后，初始化时钟RCC，使用无源外部晶振；



- 步骤二：调整系统主频至80MHz；

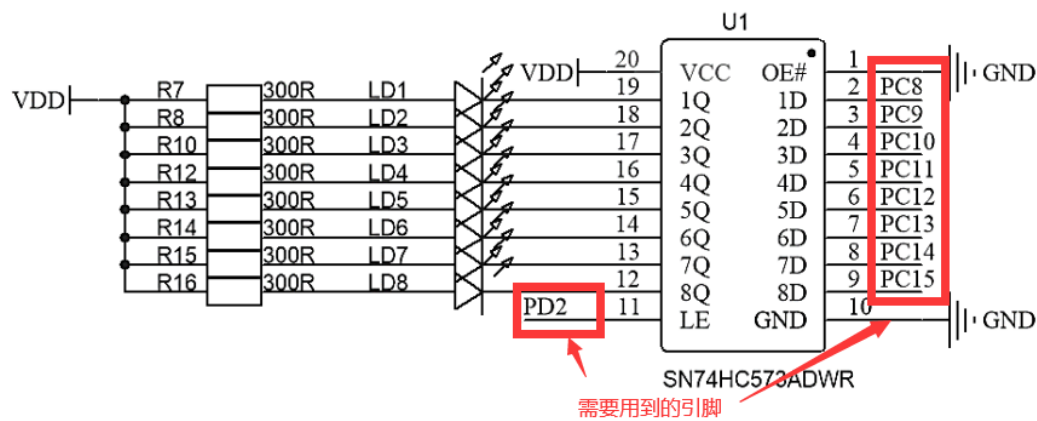


- 步骤三：配置工程信息；

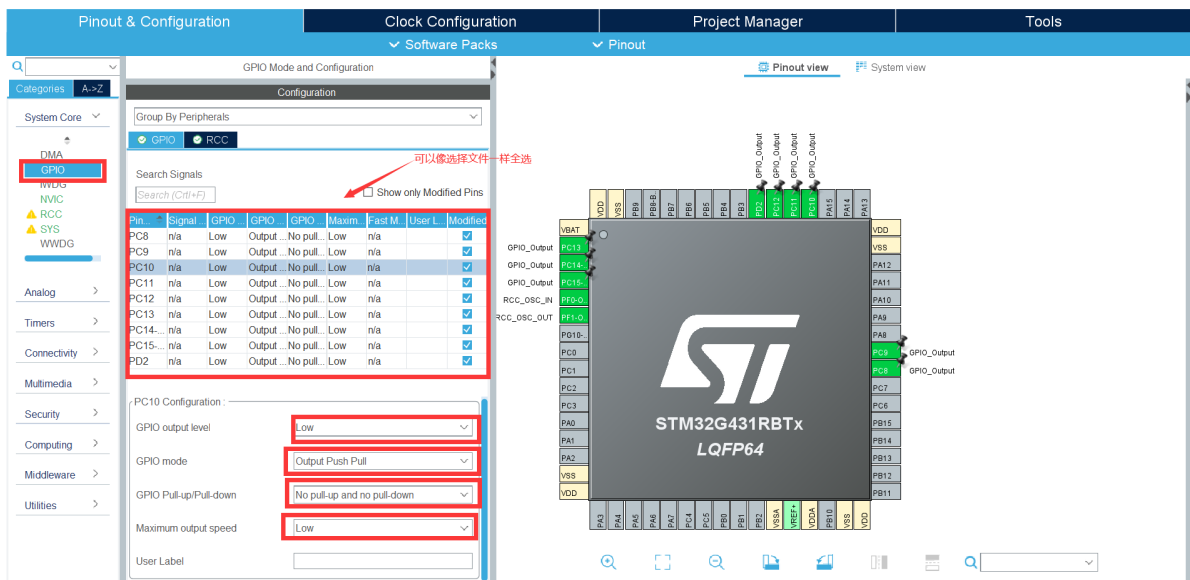
Pinout & Configuration	Clock Configuration	Project Manager
Project	<div>✓ STM32Cube MCU packages and embedded software packs</div> <div><input checked="" type="radio"/> Copy all used libraries into the project folder</div> <div><input type="radio"/> Copy only the necessary library files</div> <div><input type="radio"/> Add necessary library files as reference in the toolchain project configuration file</div>	
	<div>✓ Generated files</div> <div><input checked="" type="checkbox"/> Generate peripheral initialization as a pair of '.c/.h' files per peripheral</div> <div><input checked="" type="checkbox"/> Backup previously generated files when re-generating</div> <div><input checked="" type="checkbox"/> Keep User Code when re-generating</div> <div><input checked="" type="checkbox"/> Delete previously generated files when not re-generated</div>	
Code Generator	<div>✓ HAL Settings</div> <div><input type="checkbox"/> Set all free pins as analog (to optimize the power consumption)</div> <div><input type="checkbox"/> Enable Full Assert</div>	
	<div>✓ Template Settings</div> <div>Select a template to generate customized code</div> <div>Settings</div>	

LED

- 原理图



- CubeMX配置:
 - PC8-PC15 配置成 **GPIO_OutPut**,将默认电平设置成高电平,不加上拉下拉;
 - PD2配置成**GPIO_OutPut**,将默认电平设置成低电平,不加上拉下拉;



• 使用说明:

- 由于LCD与LED的部分引脚是重合的，初始化完成LCD后，还需要强制关闭LED；操作完LCD，再次操作LED时需要重置所有LED的状态，不然LED的工作状态就会出现问題；
- 每次使用LED时一定要记得将PD2拉高拉低，也就是打开关闭锁存器；

• 样例代码:

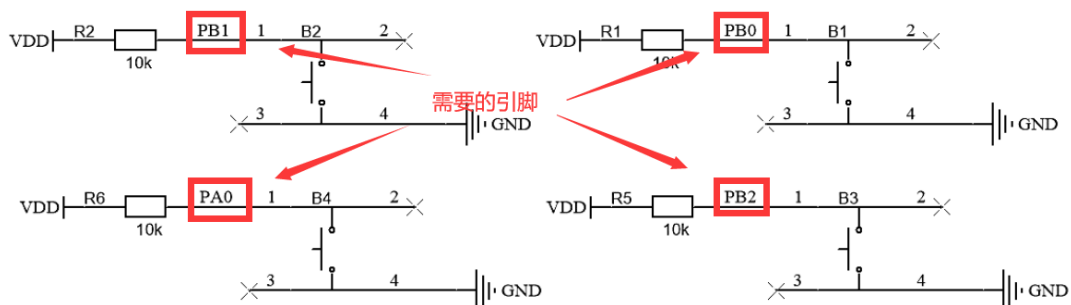
```

/*****
* 函数功能：改变所有LED的状态
* 函数参数：
*          char LEDSTATE: 0-表示关闭 1-表示打开
* 函数返回值：无
*****/
void changeAllLedByStateNumber(char LEDSTATE)
{
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15|GPIO_PIN_8
                      |GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12,
    (LEDSTATE==1?GPIO_PIN_RESET:GPIO_PIN_SET));
    //打开锁存器    准备写入数据
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_SET);
    //关闭锁存器 锁存器的作用为 使得锁存器输出端的电平一直维持在一个固定的状态
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_RESET);
}

```

按键

• 原理图



- 按键的引脚模式为**上拉模式输入模式(GPIO_Input)**



```

/*****
* 函数功能: 按键扫描 含按键消抖 无长按短按设计
* 函数参数: 无
* 函数返回值: 按键的位置
*
返回值说明: B1-1 B2-2 B3-3 B4-4
*****/
unsigned char scanKey(void)
{
    //按键锁
    static unsigned char keyLock = 1;
    //记录按键消抖时间
    // static uint16_t keyCount = 0;

    //按键按下
    if((HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_0) == RESET ||
    HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1) == RESET
    || HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2) == RESET ||
    HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0) == RESET)
    && keyLock == 1){
        //给按键上锁 避免多次触发按键
        keyLock = 0;

        //按键消抖 这里最好不要使用延时函数进行消抖 会影响系统的实时性
        // if(++keyCount % 10 < 5) return 0;
        // if(HAL_GetTick()%15 < 10) return 0;
        HAL_Delay(10);

        //按键B1
        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_0) == RESET){
            return 1;
        }
        //按键B2
        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1) == RESET){
            return 2;
        }
        //按键B3
        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2) == RESET){

```

```

        return 3;
    }
    //按键B4
    if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0) == RESET){
        return 4;
    }
}
//按键松开
if((HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_0) == SET &&
HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1) == SET
&& HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2) == SET &&
HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0) == SET)
&& keyLock == 0){
    //开锁
    keyLock = 1;
}
return 0;
}

```

LCD

- LCD**不需要经过CubeMX配置初始化**，官方会提供相关代码，直接使用官方代码即可。下面是初始化的示例：

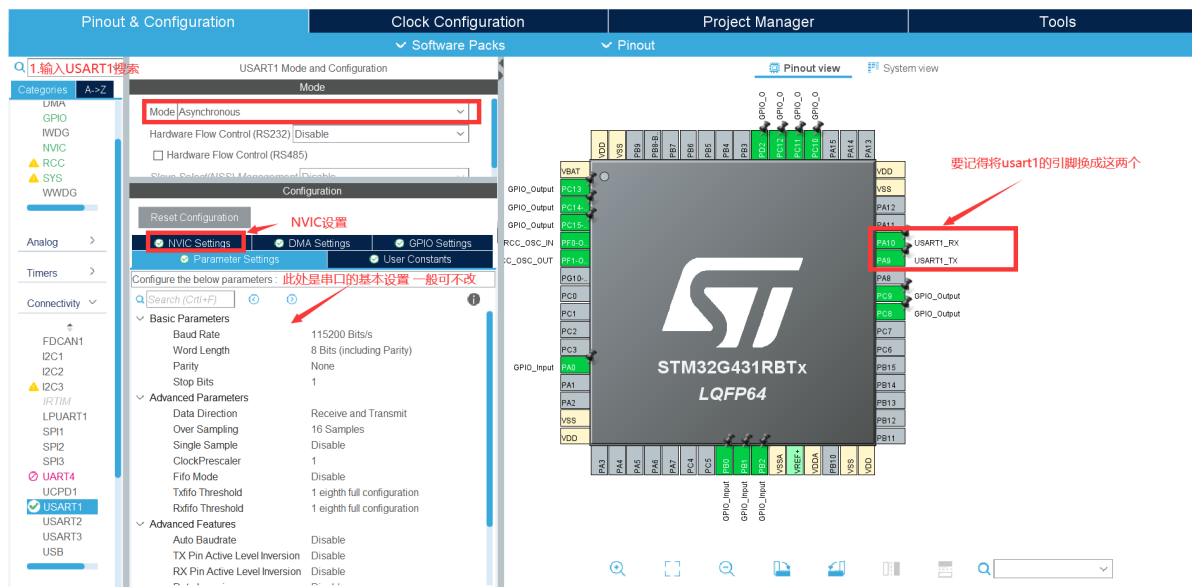
```

/*****
*  函数功能：LCD初始化
*  函数参数：无
*  函数返回值：无
*****/
void lcdInit(void)
{
    //HAL库的初始化
    LCD_Init();
    //设置LCD的背景色
    LCD_Clear(Blue);
    //设置LCD字体颜色
    LCD_SetTextColor(White);
    //设置LCD字体的背景色
    LCD_SetBackColor(Blue);
}

```

串口

- CubeMX配置
 - **usart1**串口默认配置是PC4、PC5，在这里我们要将其改成**PA9**、**PA10**；



- 串口接收数据

- 在使用下述**中断接收串口数据**时还需要注意的是需要在串口初始化完成后使用 **HAL_UART_Receive_IT(huart,(uint8_t *)&_rxBuff,1)**函数使能中断，使用中断接收数据的样例代码：

```

/** 存储串口1接收的数据
uint8_t Rxbuff[USARTMAXLENTH],_rxBuff[1];
//记录串口接收到的数据的大小
uint16_t RxCount = 0;

/**使用HAL_UART_Receive_IT中断接收数据 每次接收完成数据后就会执行该函数**/
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if(huart->Instance == USART1){
        //加上下面两句可以使得串口数据接收变成接收变长数据
        Rxbuff[RxCount++] = _rxBuff[0];
        RxCount %= 7;
        // 重新使能中断
        HAL_UART_Receive_IT(huart,(uint8_t *)&_rxBuff,1);
    }
}

/*如果需要定长的数据只需要传入接收数据的buff，不需要再定义变量记录长度。如上述的就可以将Rxbuff
的两句去掉*/

```

- 串口发送数据

- 使用**中断发送串口数据**需要用到的函数：

```

/**使用HAL_UART_Transmit_IT中断发送数据 每次发送完成数据后就会执行该函数**/
void HAL_UART_TxCpltCallback (UART_HandleTypeDef *huart)
{
}

```

不过，在实际情况中个人更偏向于使用**函数HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, const uint8_t *pData, uint16_t Size, uint32_t Timeout)**，函数解析：

```

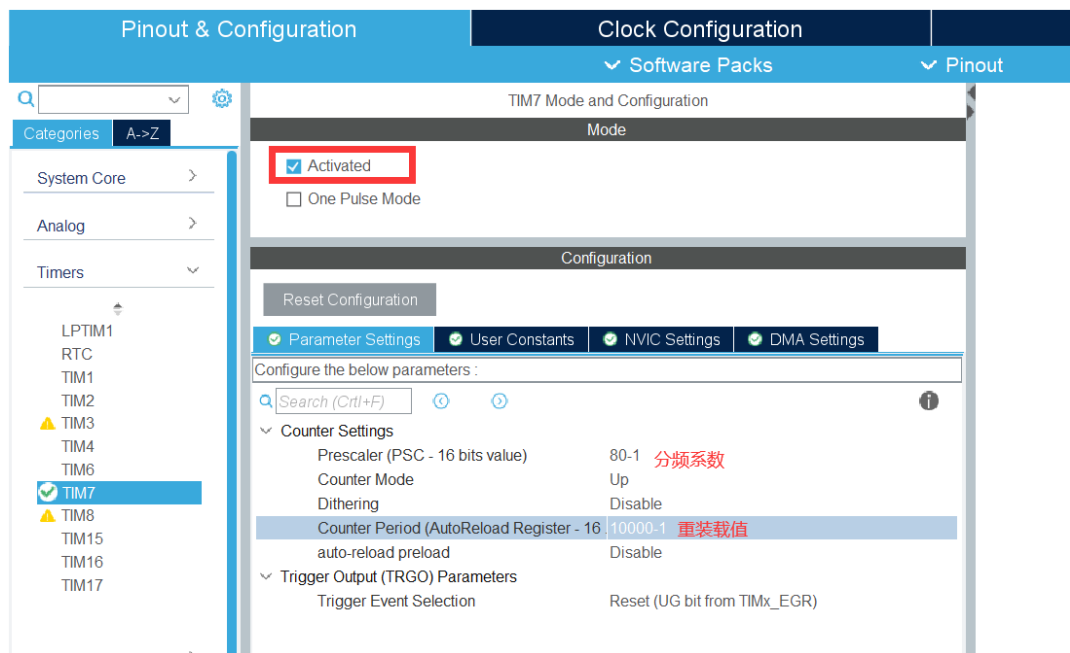
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, const uint8_t
*pData, uint16_t Size, uint32_t Timeout)
/*
UART_HandleTypeDef *huart:串口通道
const uint8_t *pData:发送的数据
uint16_t Size:发送数据的大小
uint32_t Timeout:发送数据超时时间
*/
/*
使用示例:
    发送一个字符串"hi",其超时时间为50ms
*/
HAL_UART_Transmit(&huart1,(unsigned char*)"hi",sizeof("hi"),50);

```

定时器

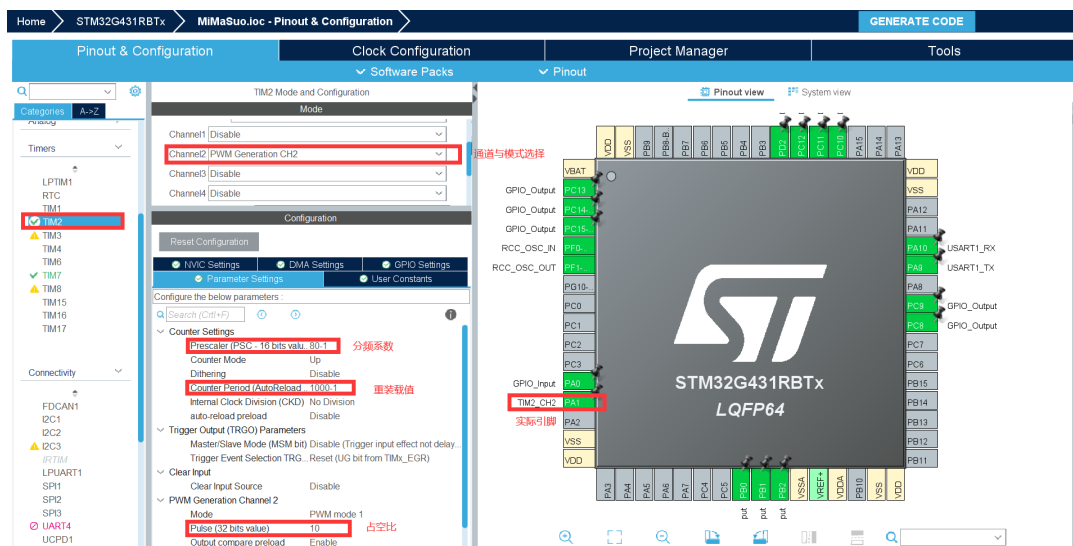
使用基本定时器的定时功能

- CubeMX配置:
 - 在使用cubem配置完成定时器后，定时器是无法正常使用定时与中断的，我们一定一定要使用函数**HAL_TIM_Base_Start_IT(&htim3)**打开定时器的中断。



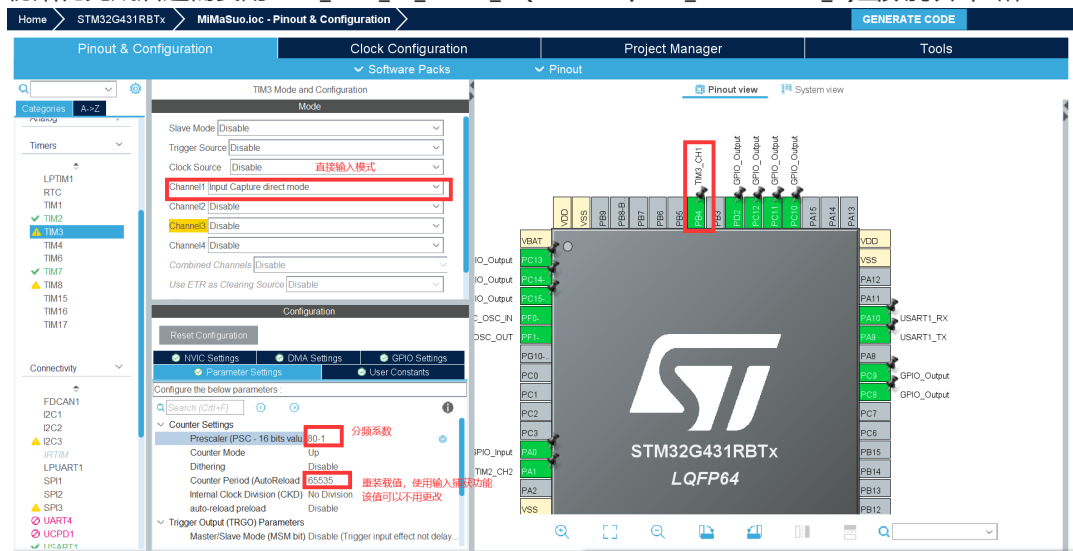
使用通用定时器的PWM输出功能

- CubeMX配置
 - 使用CubeMX配置完成后，还需要使用函数**HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_2)**打开定时器的PWM功能；



使用定时器的输入捕获功能

- 用于电位器R39与R40;
- CubeMX配置
 - 初始化完成后还需要用HAL_TIM_IC_Start_IT(&tim3,TIM_CHANNEL_1)函数打开中断;

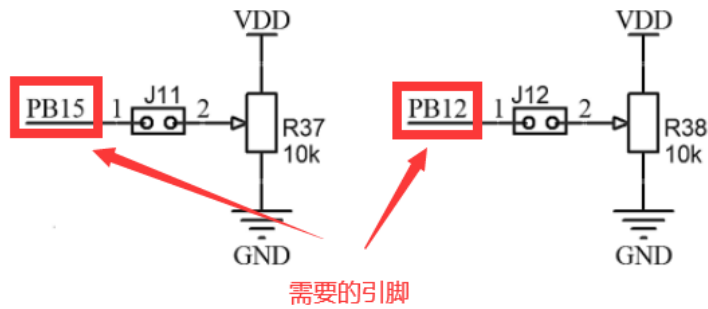


- 在程序运行时整PWM输出频率的方法:

```
//设置重装载值
__HAL_TIM_SetAutoreload(&htim2,500-1);
//设置比较值 用于得到占空比
__HAL_TIM_SetCompare(&htim2,TIM_CHANNEL_2,50);
```

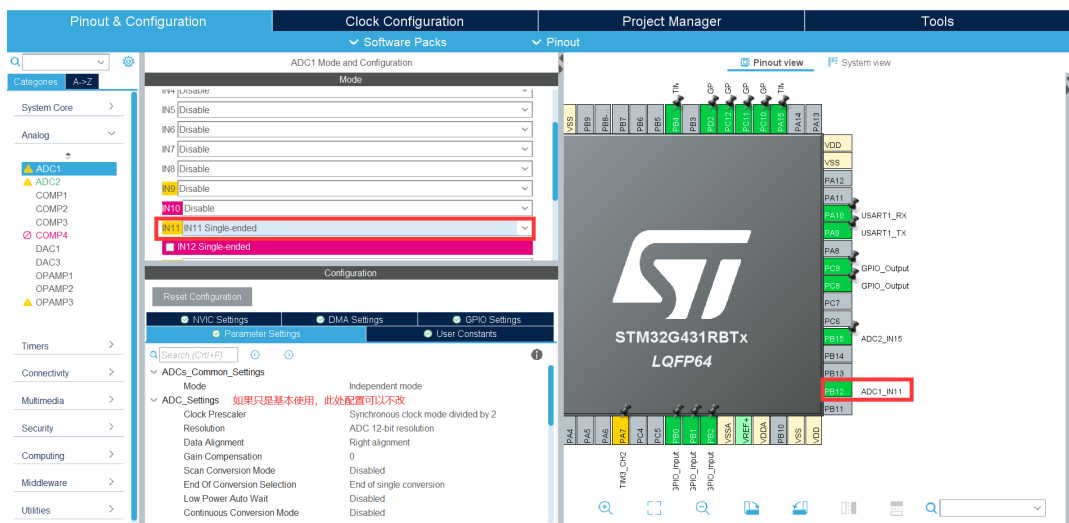
模拟输入ADC (R37、 R38)

- 原理图



- CubeMX配置

- R37与PB15直接相连接，位于ADC2的通道IN15
- R38*与PB12直接相连接，位于ADC1的通道IN11



- 获取ADC值的样例代码

```
//获取adc的值
double adcvalue = 0;

/*****
 * 函数功能: 获取ADC的值
 * 函数参数:
 *          ADC_HandleTypeDef *hadc: ADC的通道值
 * 函数返回值:
 *          double: 转换后的ADC值
 *****/

double getADC(ADC_HandleTypeDef *hadc)
{
    unsigned int value = 0;

    //开启转换ADC并且获取值
    HAL_ADC_Start(hadc);
    value = HAL_ADC_GetValue(hadc);

    //ADC值的转换 3.3V是电压 4096是ADC的精度为12位也就是2^12=4096
    return value*3.3/4096;
}
```

小tips

- 变量以百分数显示时显示%号的方法：可以使用函数`sprintf(temp,"%d%%",date)`，这里的三个百分号一个都不能少；
- 定时器中断服务函数总结

1. 输入捕获中断服务函数

```
/**  
 * @brief  Input Capture callback in non-blocking mode  
 * @param  htim TIM IC handle  
 * @retval None  
 */  
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim);
```

2. 定时器溢出后触发的中断服务函数

```
/**  
 * @brief  Period elapsed callback in non-blocking mode  
 * @param  htim TIM handle  
 * @retval None  
 */  
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);
```