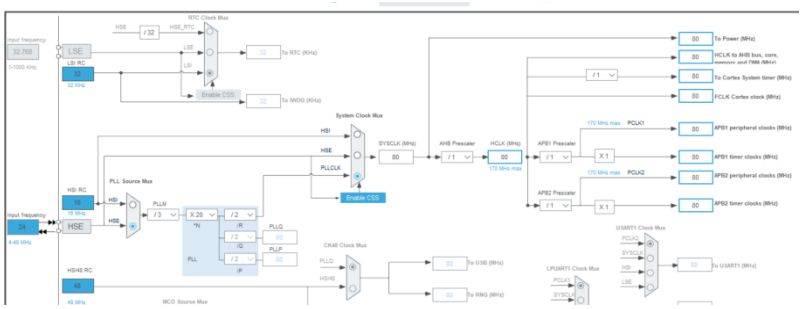
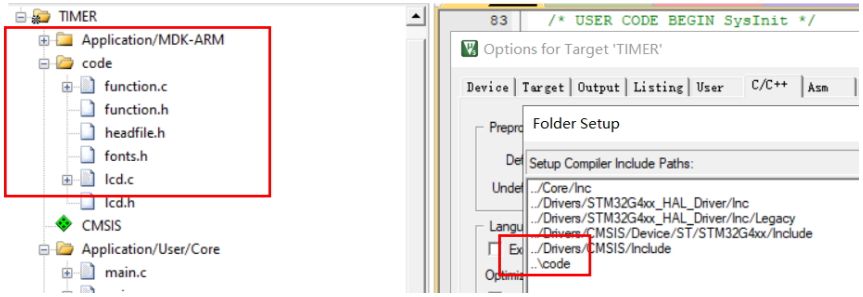
环境配置：

1. STM32G431RBT6
2. 时钟配置



3. 文件资源管理



4.

便捷操作&Q：

*Ctrl + F:find

*红色波浪线：[keil5 的源文件汇中出现红色波浪线 keil5 代码下面有红线-CSDN 博客]

*_IT?:

函数类型	中断使能?	是否需要手动配置中断?
`HAL_TIM_Base_Start/Stop`	❌ 不使能中断	❌ 无需
`HAL_TIM_Base_Start_IT/Stop_IT`	✅ 自动使能/禁用中断	⚠️ 需实现回调函数和 NVIC 配置（部分芯片需手动使能 NVIC 中断）

*利用 sprintf 完成各种奇怪的数据转换

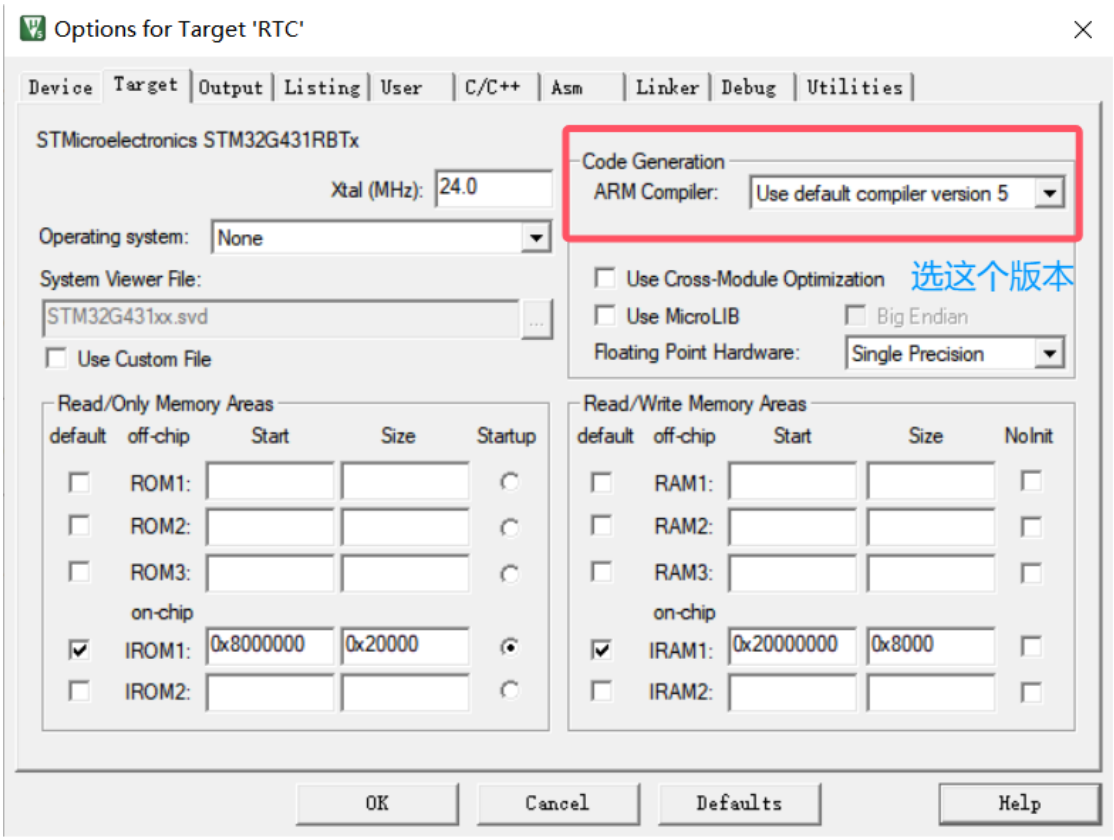
```
sprintf(parking_time_message,"%d",parking_time);
sprintf(cost,"%.2f",total_cost);
sprintf(output_message,"%s:%s:%s:%s\r\n",str1,str2,parking_time_message,cost);
```

*串口发小数转数据？

可以使用 atof（str*）函数->先包含 stdlib 库

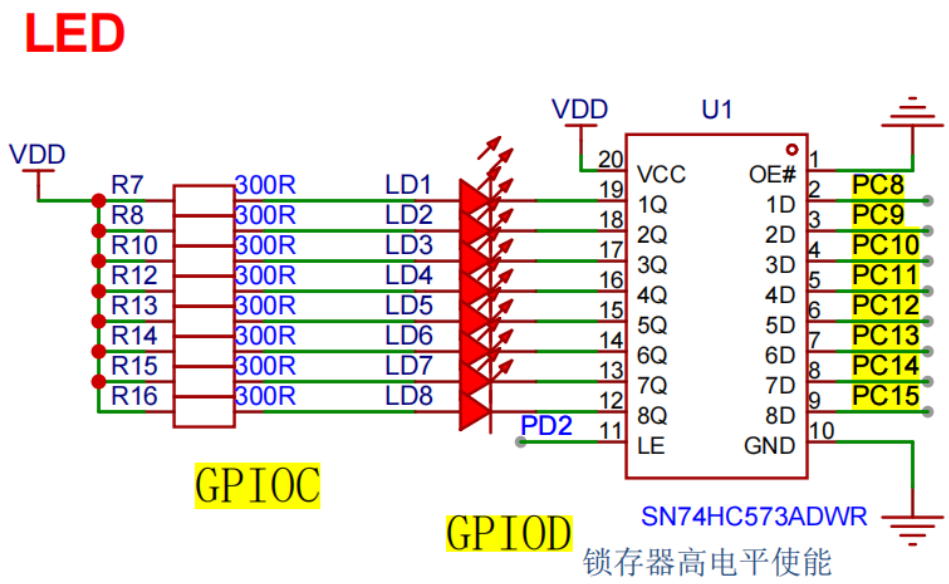
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM8 break interrupt	☐	0	0
TIM8 update interrupt	☐	0	0
TIM8 trigger and commutation interrupts	☐	0	0
TIM8 capture compare interrupt	☑	0	0

555 频率捕获也可用 TIM8,但要开启中断



E:\Keil5\ARM\ARMCC 资源包的 ARMCC 压缩包解压

LED 模块:



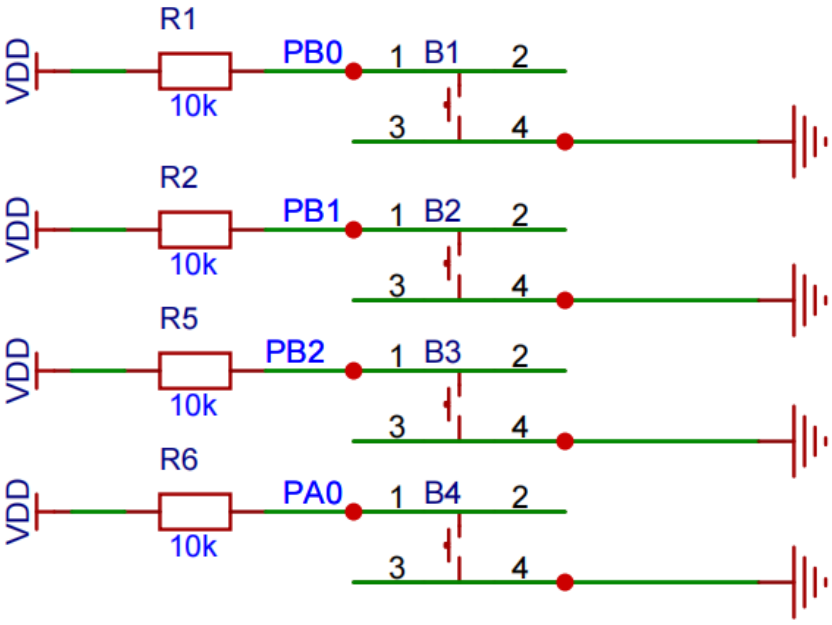

```c
//控制单个LED的状态
void light_led(uint8_t pos,int state)
{
  if(pos > 0 && pos < 9)
  {
    //控制GPIO_D PD2锁存器的状态，高电平使能
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_SET);

    if(state)
    {
      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8 << (pos - 1), GPIO_PIN_RESET);
    }
    else{
      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8 << (pos - 1), GPIO_PIN_SET);
    }
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_RESET);
  }
}
```

*初始化控制灯全不亮，锁存器使能，推挽输出

HAL 库中未提供直接操作 GPIO 口的标准函数，可操作 GPIOx->ODR 寄存器，操作 LED 的同时注意控制寄存器的锁存状态

按键模块：

按键



*初始化 GPIO 口需要使用上拉电阻，读入模式
按键按下为低电平 用定时器 50Hz 做

LCD 模块:

解决 LCD/LED 引脚冲突的方法：
*只要调用了 LCD 的函数就添加修改

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */
//注意需要修改lcd的内置函数以保证lcd和led不会发生冲突------3处
//LCD_DisplayStringLine(Line0, (uint8_t *)text);//*1
//uint16_t temp = GPIOx->ODR  ---- GPIO的输出寄存器
//LCD初始化一般写法
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_RESET);
LCD_Init(); //*2
LCD_Clear(Black); //*3
//这两函数比较简单，仅改了参数，没有函数调用
LCD_SetBackColor(Black);
LCD_SetTextColor(White);
/* USER CODE END 2 */
```

调用 function：

```
//LCD
char text[20] = {0};
void lcd_show(int key_num)
{
  sprintf(text," test");
  LCD_DisplayStringLine(Line0, (uint8_t *)text);//*
  sprintf(text,"key_num:%d",key_num);
  LCD_DisplayStringLine(Line2, (uint8_t *)text);
}
```

sprintf 函数：格式化字符串，将后面部分的内容写入前面的数组 buffer

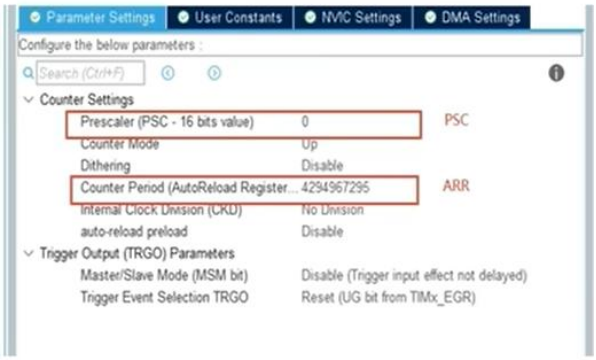
定时器:

定时器时钟频率: $f = \frac{f_{system}}{(ARR+1)(PSC+1)}$

ARR：自动重装载值
PSC：预分频器
CNT：计数器

定时器中断周期: $T = \frac{1}{f} = \frac{(ARR+1)(PSC+1)}{f_{system}}$

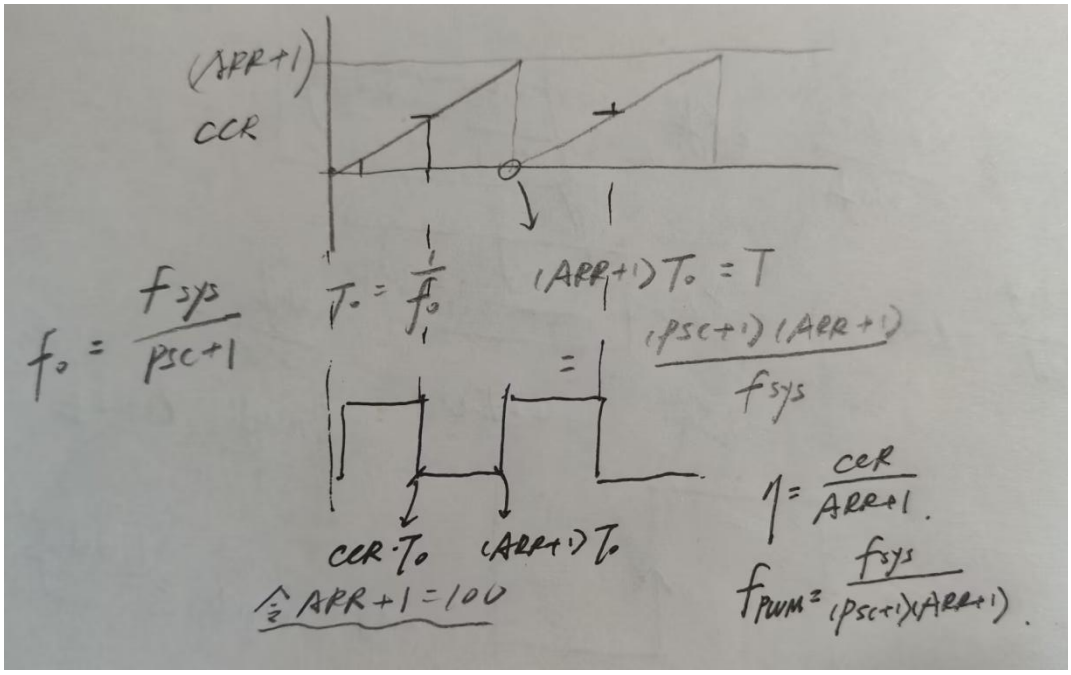
每隔 T 秒，进入一次定时器中断




```
HAL_TIM_Base_Start_IT(&htim2); //如果需要中断回调
```

进中断：

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == TIM2)
    {
        clock();
    }
}
```



PWM/捕获:

（引脚产生 PWM 波、555 产生 PWM 波）

1.PSC（Prescaler，预分频器）

作用：对定时器的输入时钟进行分频，降低计数频率。

定时器时钟 = 输入时钟 / (PSC + 1)

2.ARR（Auto-Reload Register，自动重载寄存器）

作用：设定定时器的计数上限（最大计数值），决定定时周期。

溢出时间 = (ARR + 1) * (PSC + 1) / 输入时钟频率

3.CCR（Capture/Compare Register，捕获/比较寄存器）

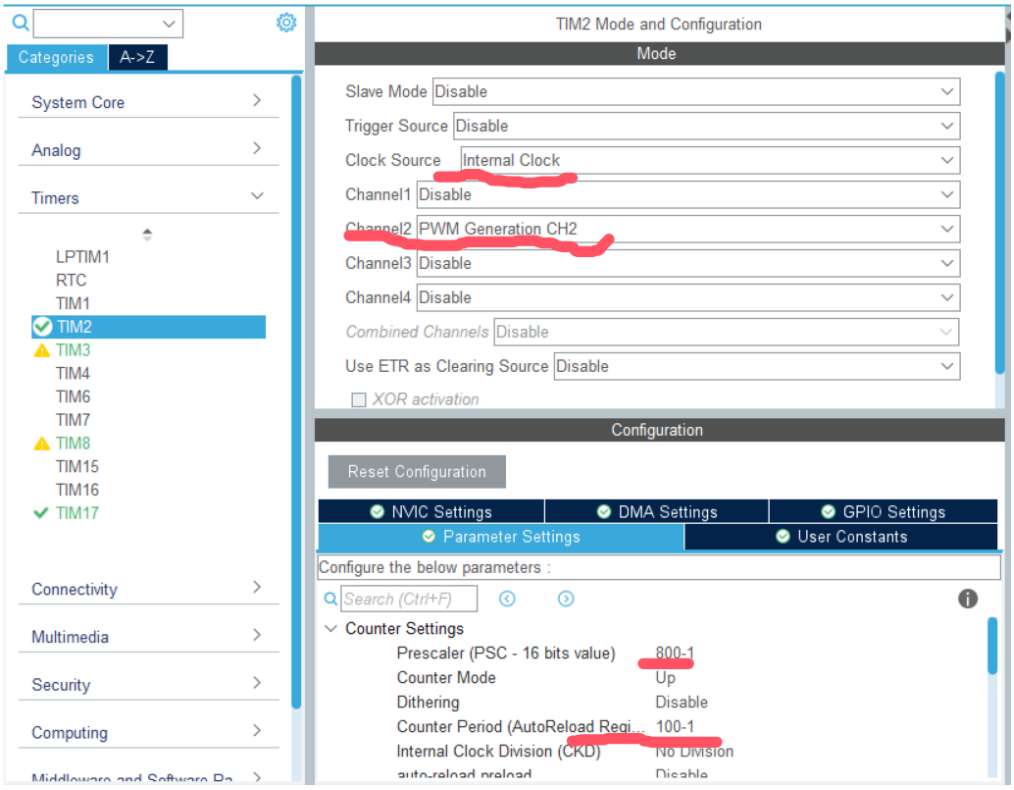
作用：输入捕获：记录外部信号边沿触发时计数器的值（用于测频/脉宽）。输出比较:当计数器值等于 CCR 时,触发动作(如引脚电平翻转、PWM 占空比控制)。

占空比 = CCR / (ARR + 1)

①PWM:

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
TIM2->CCR2 = 50;
```

单引脚利用定时器产生中断:(1KHz)



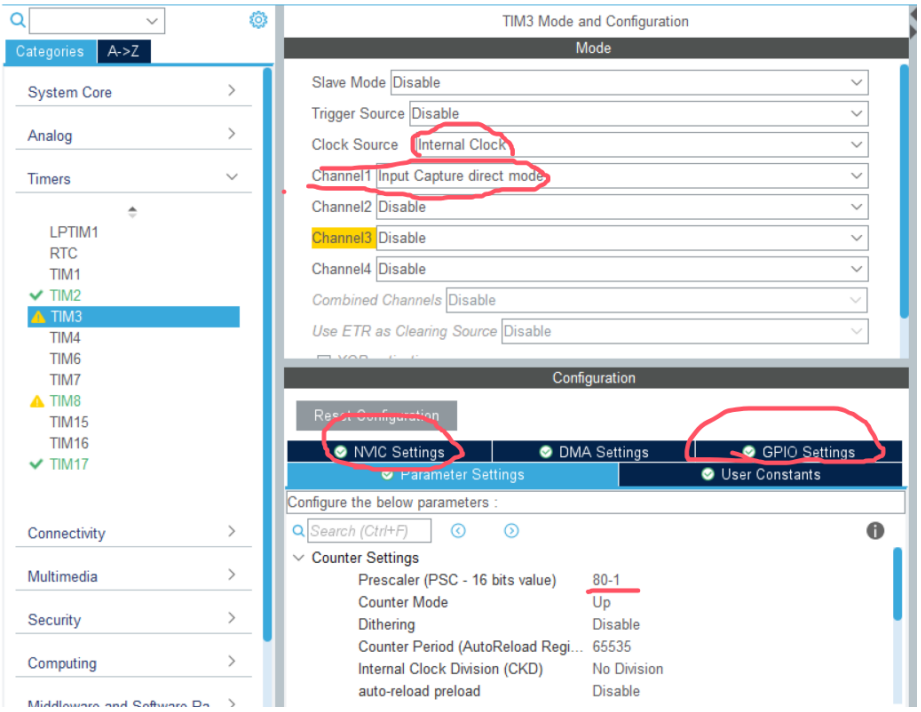
②捕获(PSC 给 80-1 让预分频尽量小 好捕获)：

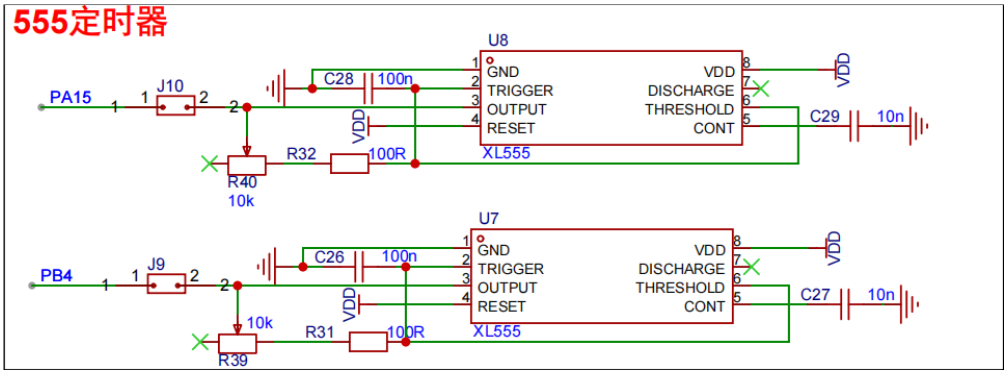
```
HAL_TIM_IC_Start_IT(&htim17, TIM_CHANNEL_1);
```

进中断：

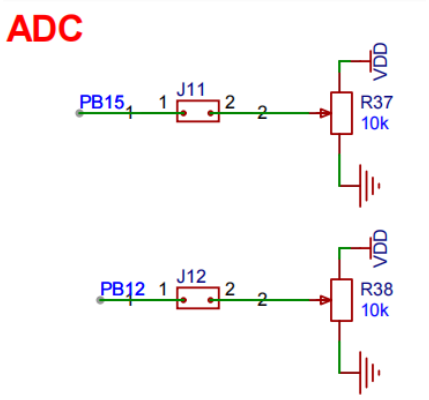
```
int capture_val,fre;
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == TIM17)
    {
        capture_val = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
        TIM17->CNT = 0; //清零计数器，直到下一个上升沿到来

        fre = 80000000/(80 * capture_val);
        // (80/80M) * capture_val = Tx -- 测量信号的一个周期
    }
}
```

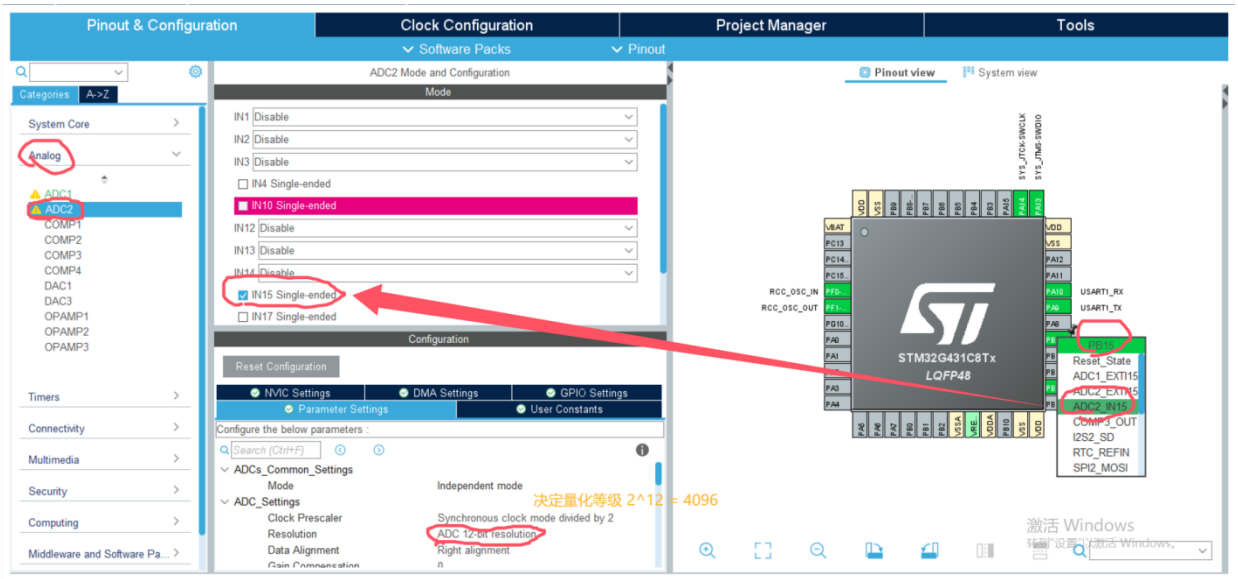
输入捕获配置，进中断，配置 GPIO 口和电路图一致（捕捉一个引脚）：


TIM3 Mode and Configuration
Mode
Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock
Channel1 Input Capture direct mode
Channel2 Disable
Channel3 Disable
Channel4 Disable
Combined Channels Disable
Use ETR as Clearing Source Disable
Configuration
NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
User Constants
Configure the below parameters :
Search (Ctrl+F)
Counter Settings
Prescaler (PSC - 16 bits value) 80-1
Counter Mode Up
Dithering Disable
Counter Period (AutoReload Regi... 65535
Internal Clock Division (CKD) No Division
auto-reload preload Disable
Categories
A->Z
System Core
Analog
Timers
LPTIM1
RTC
TIM1
TIM2
TIM3
TIM4
TIM6
TIM7
TIM8
TIM15
TIM16
TIM17
Connectivity
Multimedia
Security
Computing


555定时器
U8
GND
TRIGGER
OUTPUT
RESET
VDD
DISCHARGE
THRESHOLD
CONT
XL555
PA15
J10
C28
100n
R32
100R
R40
10k
C29
10n
VDD
U7
PB4
J9
C26
R31
R39
C27

ADC:

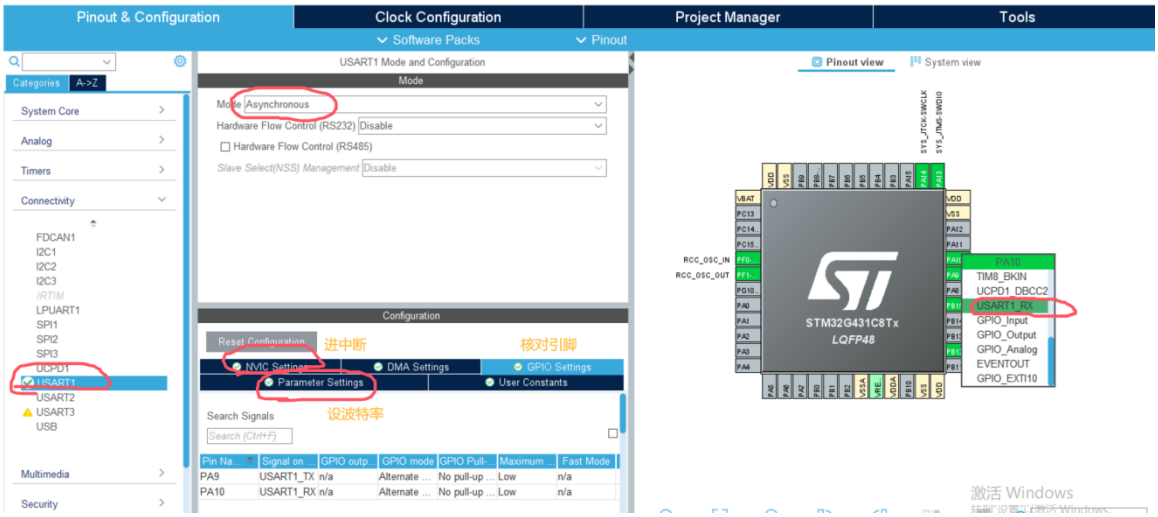

ADC
PB15
J11
R37
10k
PB12
J12
R38
10k
VDD

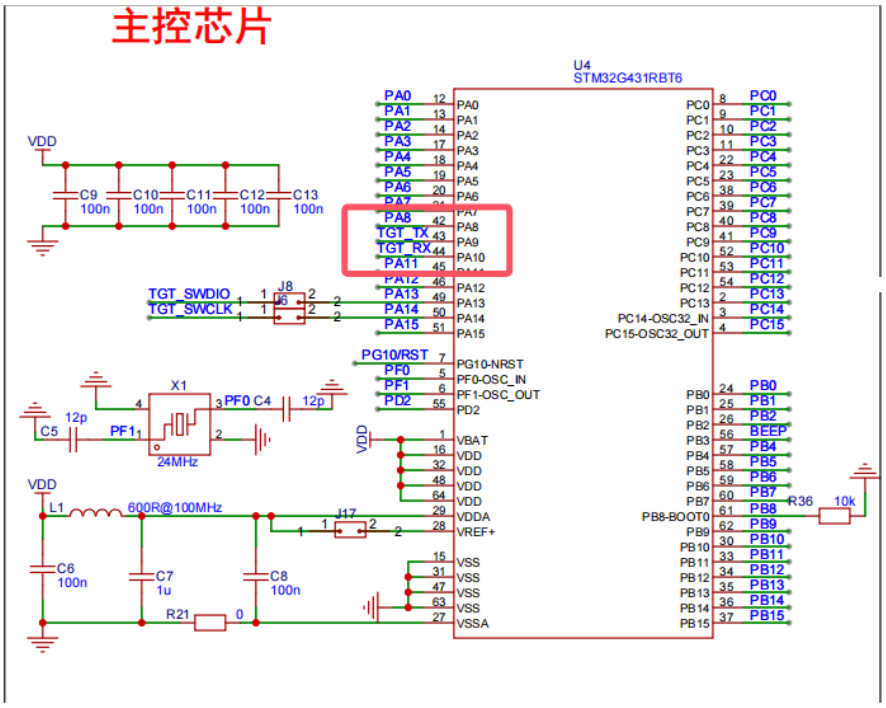


直接读值，量化计算：

```
int adc_val1,adc_val2;
char text[30] = {0};
void Get_voltage(void)
{
  HAL_ADC_Start(&hadc1);
  HAL_ADC_Start(&hadc2);

  adc_val1 = HAL_ADC_GetValue(&hadc1);
  adc_val2 = HAL_ADC_GetValue(&hadc2);

  sprintf(text,"voltage_R37:%.2f",3.3 * adc_val2 / 4096);
  LCD_DisplayStringLine(Line0,(uint8_t *)text);
  sprintf(text,"voltage_R38:%.2f",3.3 * adc_val1 / 4096);
  LCD_DisplayStringLine(Line1,(uint8_t *)text);
}
```

串口通信:





USART_SEND:

依赖于函数：

```
HAL_UART_Transmit(&huart1, //uart 句柄
                  (uint8_t *)"string\r\n", //待发送数据
                  sizeof("stri\r\n"), //待发送数据大小
                  100); //超时时间
```

USART_RECEIVE:

注意把所用定时器 PSC 设置为 8000-1

mian.c 中：
HAL_UART_Receive_IT(&huart1, &rx, 1); //不进 while 循环，初始化中断
//extern uint8_t rx 为全文件变量，在“.h”声明，负责缓存一个字节的数据
func.h 中：（注意计数器的重置和下一次的进中断使能）

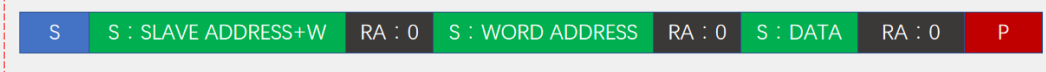
```
char rx_data[30] = {0};
int rx_count;
int rx_flag;
uint8_t rx;
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  if(huart->Instance == USART1)
  {
    TIM3->CNT = 0;
    if(rx_count < 30)
      rx_data[rx_count] = rx;
    rx_count ++;
    rx_flag = 1;

    HAL_UART_Receive_IT(&huart1, &rx, 1);
  }
}
```

```
void USART_RX_DATA(void)
{
  int i = 0;

  if(rx_flag)
  {
    if(TIM3->CNT >= 15)
    {
      //帧尾定时器到，没有下一个数据到来
      LCD_Clear(Line3);
      sprintf(text,"%s",rx_data);
      LCD_DisplayStringLine(Line3,(uint8_t *)text);
      sprintf(text,"RX_Len:%d",strlen(rx_data));
      LCD_DisplayStringLine(Line4,(uint8_t *)text);

      HAL_UART_Transmit(&huart1, (uint8_t *)rx_data, sizeof(rx_data), 100);
      //重置全局变量
      rx_flag = 0;
      rx_count = 0;
      for(i = 0;i < 30;i ++)
        rx_data[i] = '\0';
    }
  }
}
```

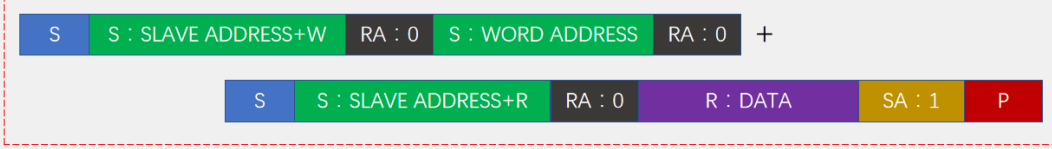
I2C 通信:(可以参考手册自己写)

AT24C02数据帧

- 字节写：在WORD ADDRESS处写入数据DATA



- 随机读：读出在WORD ADDRESS处的数据DATA



- AT24C02的固定地址为1010，可配置地址本开发板上为000
  所以SLAVE ADDRESS+W为0xA0，SLAVE ADDRESS+R为0xA1

I2CInit();//不要忘记初始化

```
void Write_EEPROM(uint8_t address,uint8_t data)
{
  I2CStart();
  //见手册，写0读1
  I2CSendByte(0xA0);
  I2CWaitAck();
  I2CSendByte(address);
  I2CWaitAck();
  I2CSendByte(data);
  I2CWaitAck();
  I2CStop();
  //给点延时让写入反应
  HAL_Delay(20);
}
```

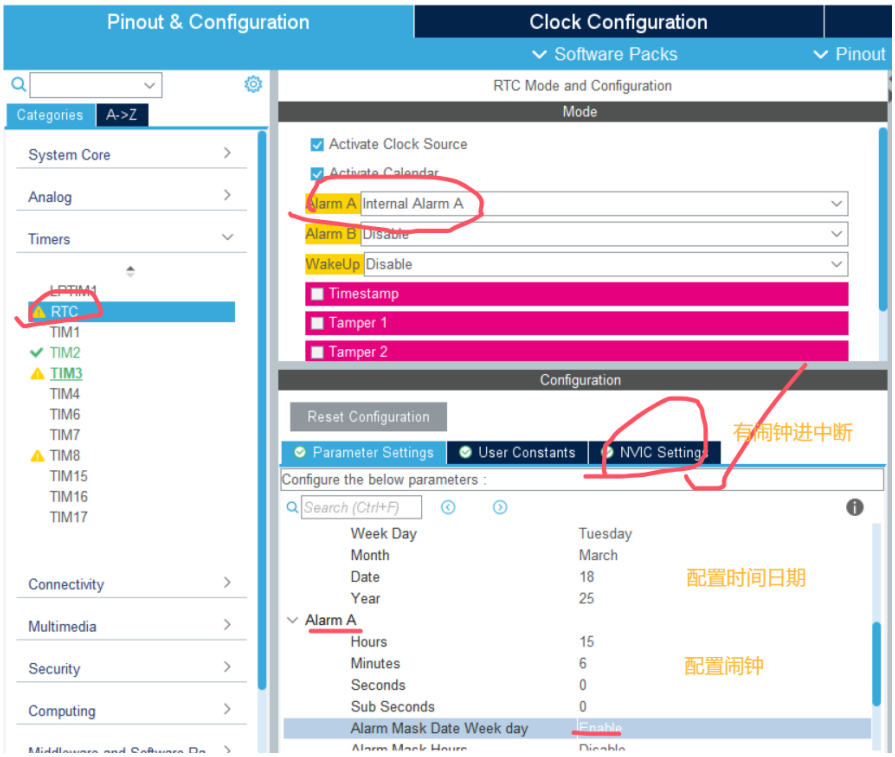
```
uint8_t Read_EEPROM(uint8_t address)
{
  uint8_t data = 0x00;
  //先转跳到对应地址
  I2CStart();
  I2CSendByte(0xA0);
  I2CWaitAck();
  I2CSendByte(address);
  I2CWaitAck();

  I2CStart();
  I2CSendByte(0xA1);
  I2CWaitAck();
  data = I2CReceiveByte();
  I2CSendNotAck();
  I2CStop();
  return data;
}
```

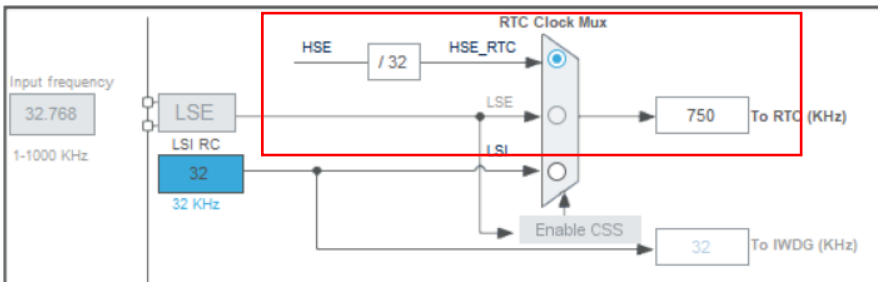
浮点数存入 EEPROM:(映射)

1.1 先转成 11，存进去，需要的时候拿出来/10

RTC 实时时钟:

* Hour Format->24/12 小时制、Date Format->Binary data format;
* Alarm Mask D/W/d：掩盖表示时钟每日触发。





选用 HSE 高速时钟 24MHz，125*6000=750k，正好 1s 分频

结构体格式定义可参考 cubemx 初始化文件：

```
RTC_TimeTypeDef sTime = {0};
RTC_DateTypeDef sDate = {0};
void Get_real_time()
{
  HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
  HAL_RTC_GetDate(&hrtc, &sDate, RTC_FORMAT_BIN);
}

int Alarm_flag;
void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc)
{
  //闹钟触发中断
  Alarm_flag = 1;
}
```

真题反馈：

1、float 转 int？

2、

```
    sprintf(text,"      A=%.2fKHz",((float) Fre_A) / 1000);
  //Fre_A_K先拿个变量存着不行    ((float) Fre_A) / 1000 可实现
```

3、第十四届 数据 2s 内不变化？ 频率均匀步进？

4、定时器方式处理按键 非阻塞 单击双击长按

5、Duty 记得双通道开 IT

6、PWM_Start()

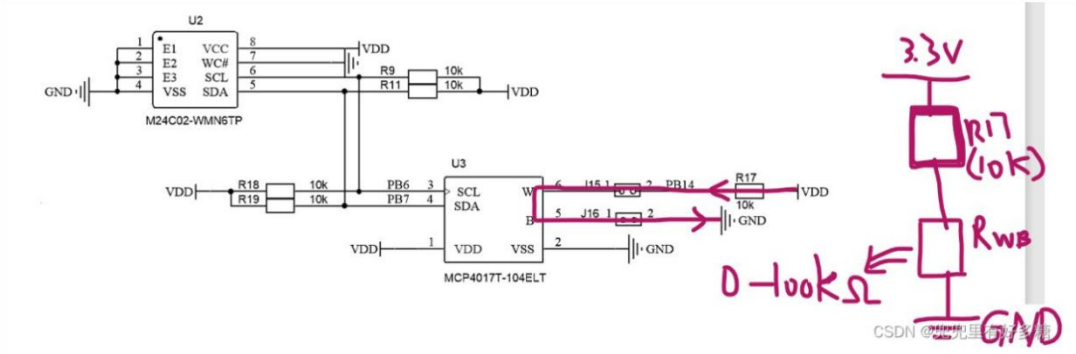
7、什么时候 Start 什么要中断 定时器、adc、ic、串口、pwm

```
//IC
HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_1);
HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_2);
/*
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
HAL_TIM_ReadCapturedValue(const TIM_HandleTypeDef *htim, uint32_t Channel)
*/
//PWM
HAL_TIM_PWM_Start(&htim15, TIM_CHANNEL_1);
TIM15->CCR1 = 50;
//TIMER
HAL_TIM_Base_Start(&htim2);
HAL_TIM_Base_Start_IT(&htim2);
/*
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
*/
//ADC
//可以省略初始化
HAL_ADC_Start(ADC_HandleTypeDef *hadc);
HAL_ADC_GetValue(ADC_HandleTypeDef *hadc)
//UART
HAL_UART_Receive_IT(&huart1, &rx, 1);
/*
HAL_TIM_Base_Start(&htim3); //辅助开一个定时器 0.1ms 计一次数
HAL_UART_Transmit(&huart1, (uint8_t *)"ERROR!\r\n", sizeof("ERROR!\r\n"), 50);
*/
```

8、解决 LCD 的残留问题：

```c
void LCD_DisplayStringLine(u8 Line, u8 *ptr)
{
  uint32_t temp = GPIOC->ODR;
    u32 i = 0;
    u16 refcolumn = 319;//319;

    //手动更改对'\0'后面数据的处理
    /*
      while条件
      if else
    */
    while ((i < 20))    // 20
    {
      if(*ptr == 0)
      {
        LCD_DisplayChar(Line, refcolumn, ' ');
      }
      else
      {
        LCD_DisplayChar(Line, refcolumn, *ptr);
        ptr++;
      }
        refcolumn -= 16;
        i++;
    }
  GPIOC->ODR = temp;
}
```

9、DMA IIC RTC



所以PB14处的电压就是：

$$V = 3.3 * \frac{R_{WB}}{R_{WB} + 10}$$

Rwb = 100kom * N / 127

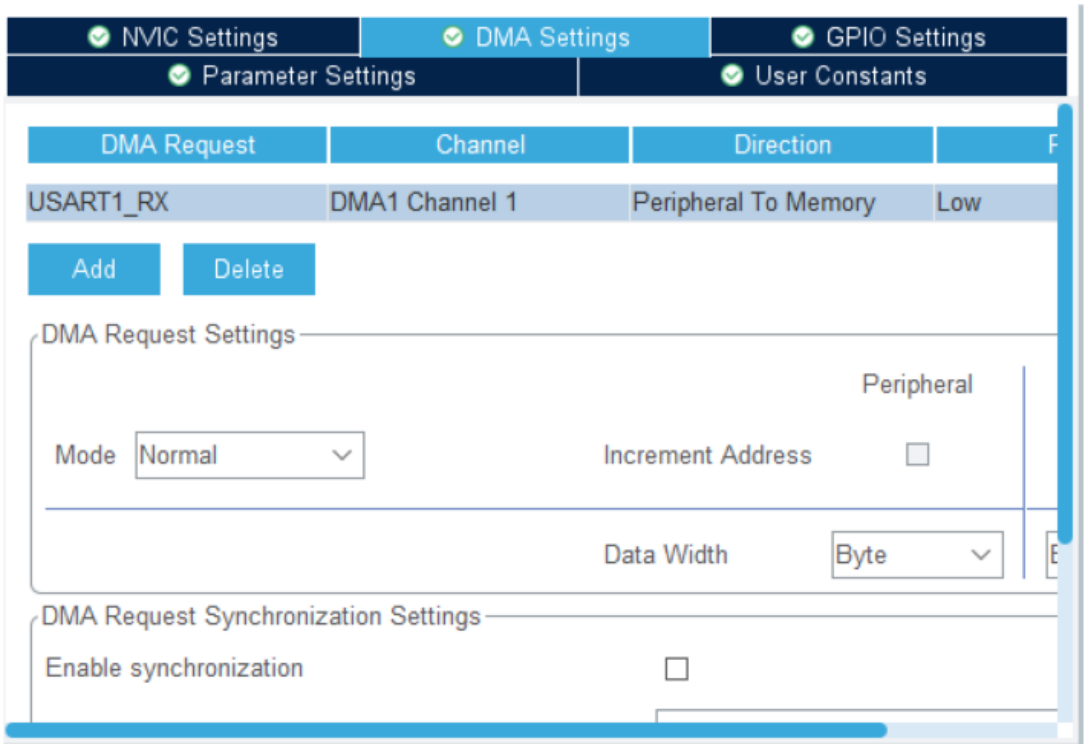
10、频率捕获上电刚开始可能 fre 为 0（没捕获这么快）

选择题：
微控制器参考手册
P75:G431 内部资源一览

国赛

一、串口-DMA 模式-字符串处理

无需开定时器 9600 波特率 需要开 NVIC 中断 DMA add RX 即可




```c
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size)
{
  //回调函数在uart
  if(huart->Instance == USART1)
  {
    rx_flag = 1;
    //传输函数在uart_ex
    HAL_UARTEx_ReceiveToIdle_DMA(huart, (uint8_t *)rx_data, RX_BUFFER_SIZE);
  }
}
```

HAL_UARTEx_ReceiveToIdle_DMA(huart, (uint8_t *)rx_data, RX_BUFFER_SIZE);

（在 main 函数初始化也要添加）

```c
void UAST_TASK(void)
{
    if(rx_flag) {
    ***
    rx_flag = 0;
    menset(rx_data,0,strlen(rx_data));
    }
}
```

```
for(i = 1;i < rx_length - 1;i ++) str_num[i - 1] = rx_data[i];
i = 0;
str_cut = strtok(str_num,",");
while(str_cut != 0)
{
  cut_data[i] = atoi(str_cut); //截断的字符转整数
  i ++;
  str_cut = strtok(NULL,",");  //继续截断
}
```

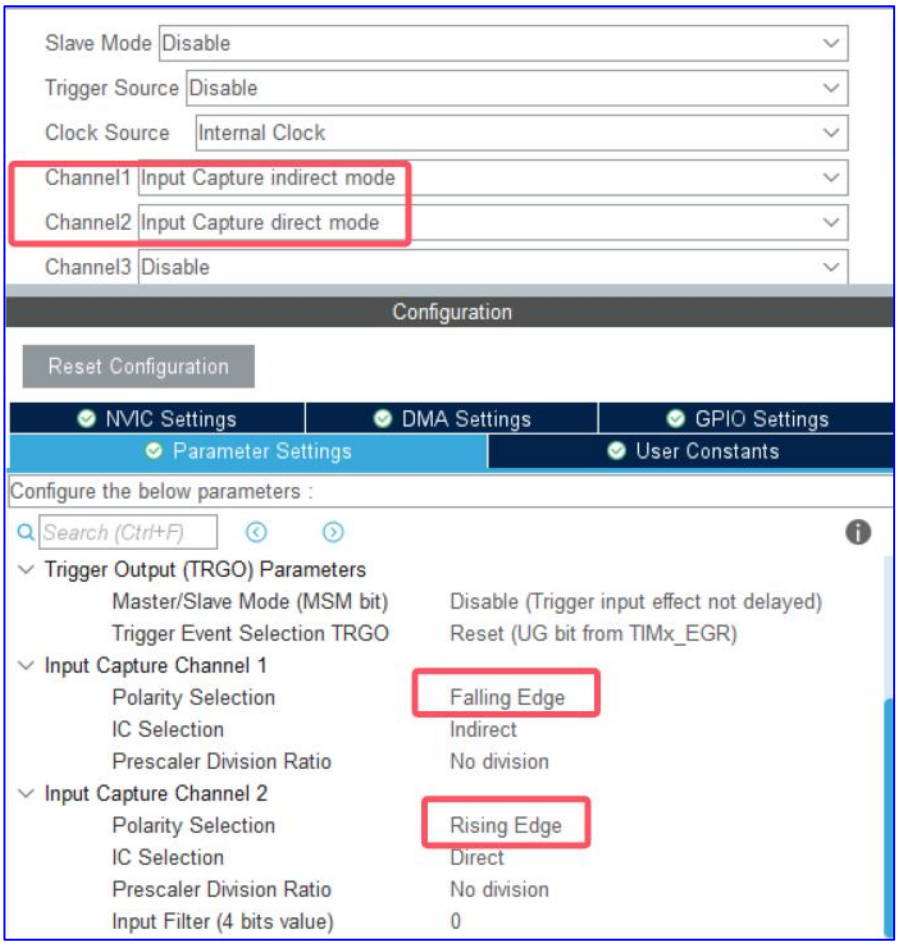
```
//删除一个点 {20,60}
str_cut = strtok(rx_data + 1,",");
delete_x = atoi(str_cut);
str_cut = strtok(NULL,"}");
delete_y = atoi(str_cut);
```

二、解决 LCD 不等长显示

```
298  void LCD_DisplayStringLine(u8 Line, u8 *ptr)
299  {
300    uint32_t temp = GPIOC->ODR;
301      u32 i = 0;
302      u16 refcolumn = 319;//319;
303
304      //手动更改对'\0'后面数据的处理
305      /*
306        while条件
307        if else
308      */
309      while ((i < 20))   // 20
310      {
311        if(*ptr == 0)
312        {
313          LCD_DisplayChar(Line, refcolumn, ' ');
314        }
315        else
316        {
317          LCD_DisplayChar(Line, refcolumn, *ptr);
318          ptr++;
319        }
320          refcolumn -= 16;
321          i++;
322      }
323    GPIOC->ODR = temp;
324  }
```

三、测量输入信号频率、占空比

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
  if(htim->Instance == TIM2)
  {
    if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2)
    {
      //捕获上升沿
      rising_time = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_2);
      falling_time = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
      TIM2->CNT = 0;

      Fre = 80000000 / ((rising_time) * (TIM2->PSC));
      Duty = falling_time * 100 / rising_time;
    }
  }
}
```



四、RTC 实时时钟 时钟树最上面用高速时钟 750kHz 6000 * 125 1s

五、按键 双击 组合式按键

```
struct key
{
 GPIO_PinState key_state;
 int short_flag;
 int double_flag;
 int long_flag;

 int kick_flag;
 int kick_cnt; //统计按下的次数

 int press_state;
 int long_press_cnt;
 int double_press_cnt; //统计按下的时间间隔
}KEY[4];
//组合按键
int key3_first_flag;
int key4_first_flag;
int combine_flag;
int combine_cnt;
void HAL_TIM_PeriodElapsedCallback(TIM_Han
dleTypeDef *htim)
{
 int i = 0;
 if(htim->Instance == TIM3)
 {
  //10ms
  KEY[0].key_state = HAL_GPIO_ReadPin(GPIOB,
GPIO_PIN_0);
  KEY[1].key_state = HAL_GPIO_ReadPin(GPIOB,
GPIO_PIN_1);
  KEY[2].key_state = HAL_GPIO_ReadPin(GPIOB,
GPIO_PIN_2);
  KEY[3].key_state = HAL_GPIO_ReadPin(GPIOA
,GPIO_PIN_0);

  for(i = 0;i < 4;i ++)
  {
   switch(KEY[i].press_state)
   {
    case 0:
    {
     if(KEY[i].key_state == GPIO_PIN_RESET)
     {
      KEY[i].press_state = 1;
     }
    }break;
    case 1:
    {
     if(KEY[i].key_state == GPIO_PIN_RESET)
     {
      if(i == 2 || i == 3)
      {
       //KEY3、4 组合按键
       if(i == 2)
        key3_first_flag = 1;
       else if(i == 3)
        key4_first_flag = 1;
       combine_cnt = 0;

      }
      KEY[i].press_state = 2;
      KEY[i].long_press_cnt = 0;
     }
     else
      KEY[i].press_state = 0;
    }break;
    case 2:
    {
     if(KEY[i].key_state == GPIO_PIN_RESET)
     {
      if(key3_first_flag == 1 && key4_first_flag =
= 1)
      {
       //KEY3、4 组合按键
       combine_cnt ++;
       if(combine_cnt >= 200)
        {
        combine_flag = 1;
        KEY[2].long_press_cnt = 201;
        KEY[3].long_press_cnt = 201;
        }
      }

      KEY[i].long_press_cnt ++;
      if(KEY[i].long_press_cnt >= 200 &&
combine_flag != 1)
       //2s
       KEY[i].long_flag = 1;
```

```
    }
    else
    {
     //KEY3、4 组合按键
     if(i == 2)
      key3_first_flag = 0;
     else if(i == 3)
      key4_first_flag = 0;

     KEY[i].press_state = 0;
     if(KEY[i].long_press_cnt < 200)
     {
      //准备双击
      KEY[i].kick_cnt ++;
      KEY[i].kick_flag = 1;
      KEY[i].double_press_cnt = 0;
     }
    }
   }break;
  }
  //双击 可拓展 N 击
  if(KEY[i].kick_flag)
  {
   KEY[i].double_press_cnt ++;

   if(KEY[i].double_press_cnt >= 25)
   {
    //250ms 后没有按键再被按下 结算
    if(KEY[i].kick_cnt == 1)
     KEY[i].short_flag = 1;
    else if(KEY[i].kick_cnt == 2)
     KEY[i].double_flag = 1;

    KEY[i].double_press_cnt = 0;
    KEY[i].kick_flag = 0;
    KEY[i].kick_cnt = 0;
   }
  }
 }
}
```

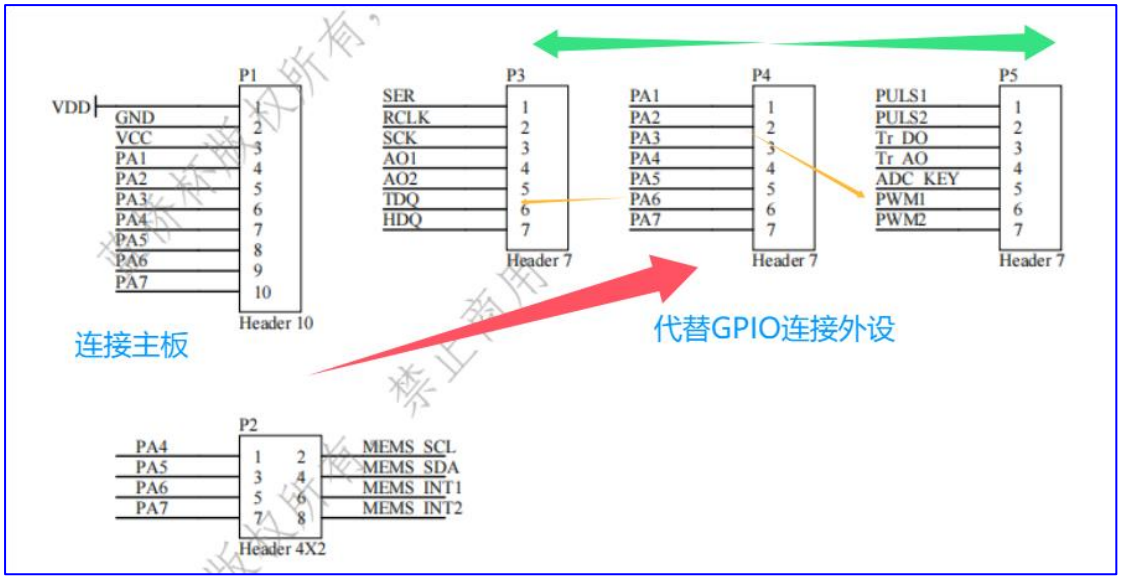
六、扩展板

1. DS18B20 初始化单总线用 PA6 可参考提供的底层

主函数调用 ds18b20_hal.h -> void ds18b20_init_x(void); //初始化外设

```
//自己写一个DS18B20温度读取
float ds18b20_read(void)
{
    uint8_t low, high;
    float temp;

    ow_reset(); //DS18B20开始转换
    ow_byte_wr(OW_SKIP_ROM); //跳过ROM
    ow_byte_wr(DS18B20_CONVERT);

    ow_reset(); //开始读取DS18B20数据
    ow_byte_wr(OW_SKIP_ROM); //跳过ROM
    ow_byte_wr(DS18B20_READ);

    low = ow_byte_rd();
    high = ow_byte_rd();
    temp = (high << 8 | low) * 0.0625; //转成实际温度

    return temp;
}
```

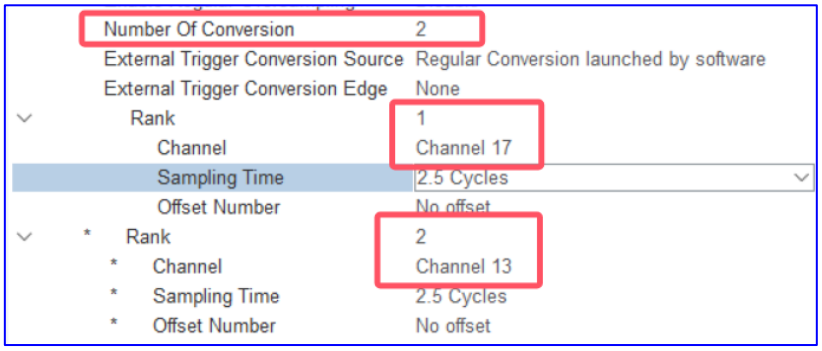
2. 接线



七、单 TIM 频率捕获/adc 多通道

双通道公用 CNT 无法清零

```
if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) // 1.找到他是哪个通道产生上升沿
{
  if(!state1)
  {
    tim1[0] = TIM3->CCR1;
    state1 = 1;
  }
  else
  {
    tim1[1] = TIM3->CCR1;
    uint16_t diff = (tim1[1] > tim1[0])?(tim1[1] - tim1[0]):
                    (65535 - (tim1[0] - tim1[1]));
    fre_R39 = 1000000/diff;
    state1 = 0;
  }
}
if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2) // 1.找到他是哪个通道产生上升沿
{
  if(!state2)
  {
    tim2[0] = TIM3->CCR2;
    state2 = 1;
  }
  else
  {
    tim2[1] = TIM3->CCR2;
    uint16_t diff = (tim2[1] > tim2[0])?(tim2[1] - tim2[0]):
                    (65535 - (tim2[0] - tim2[1]));
    fre_PA7 = 1000000/diff;
    state2 = 0;
  }
}
```



循环扫描

```c
void adc_read(double *adc_volt1,double *adc_volt2) // 传地址
{
  HAL_ADC_Start(&hadc1);
  uint16_t MCP_value = HAL_ADC_GetValue(&hadc1);
  *adc_volt1 = 3.3*MCP_value/4095;

  HAL_ADC_Start(&hadc1);
  uint16_t R38_value = HAL_ADC_GetValue(&hadc1);
  *adc_volt2 = 3.3*R38_value/4095;
}
```

```c
void selectionSort(int arr[], int n) {
    // 外层循环：控制当前需要放置的位置（从0到n-2）
    for (int i = 0; i < n - 1; i++) {
        // 假设当前索引i的元素是最小值
        int minIndex = i;

        // 内层循环：在未排序部分查找实际最小值
        for (int j = i + 1; j < n; j++) {
            // 如果找到更小的值，更新最小值的索引
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }

        // 将找到的最小值与当前位置i交换
        // 使用临时变量进行交换操作
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}
```