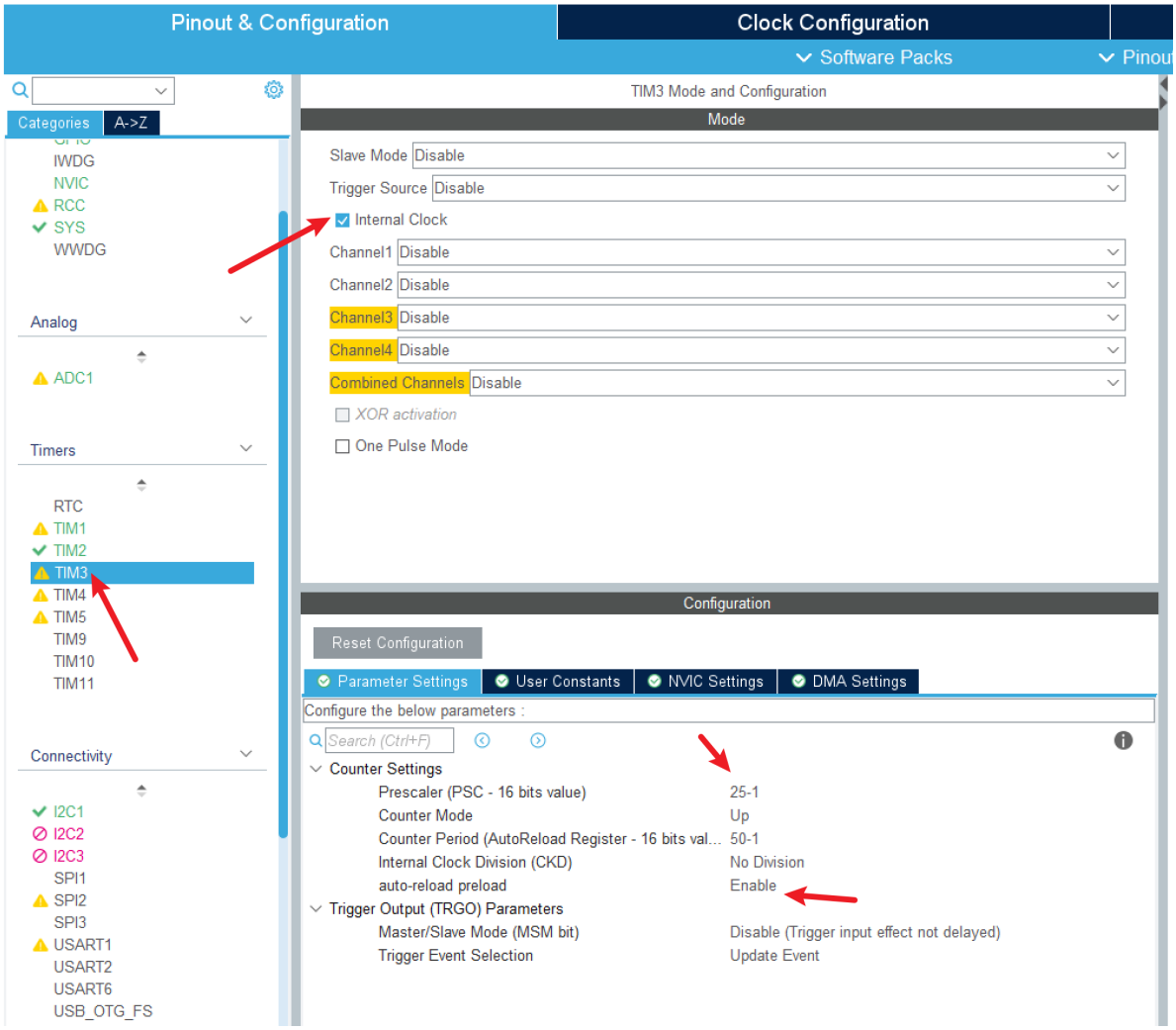
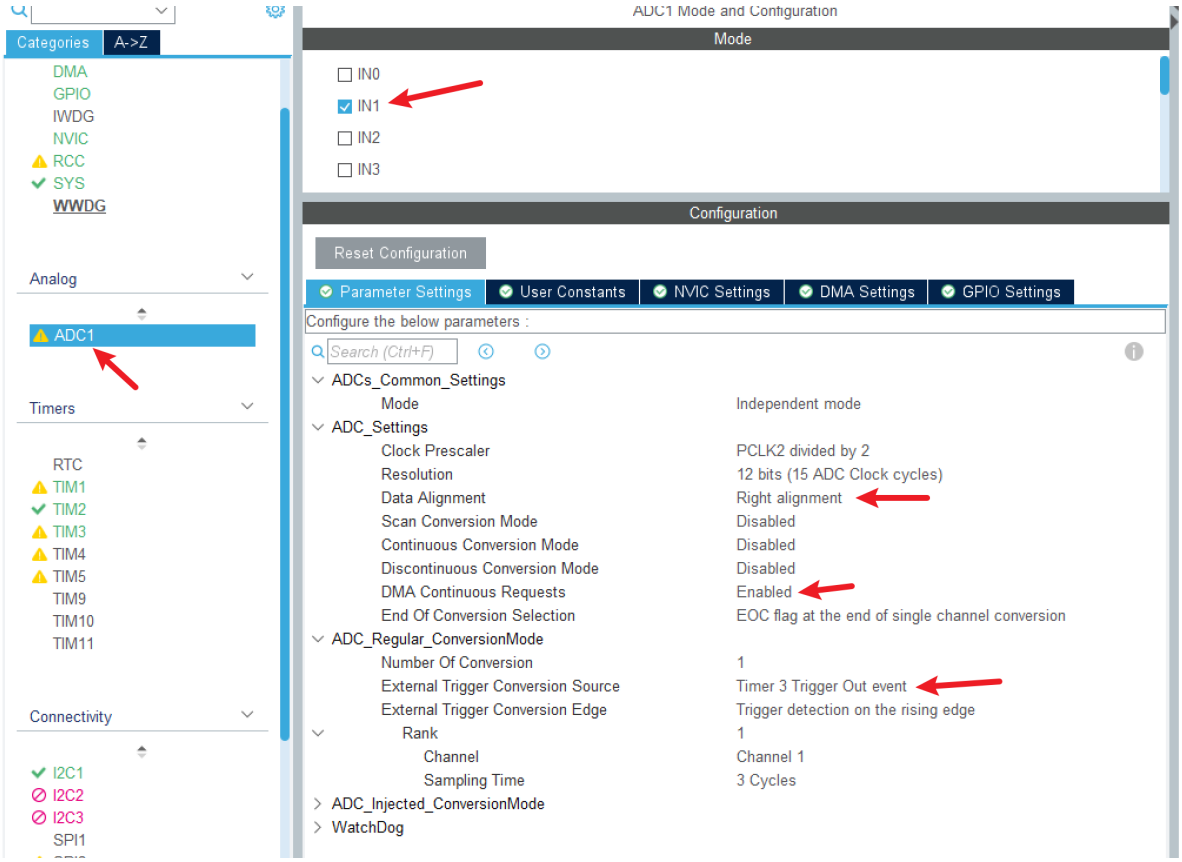
ADC+DMA采集波形

CubeMX配置

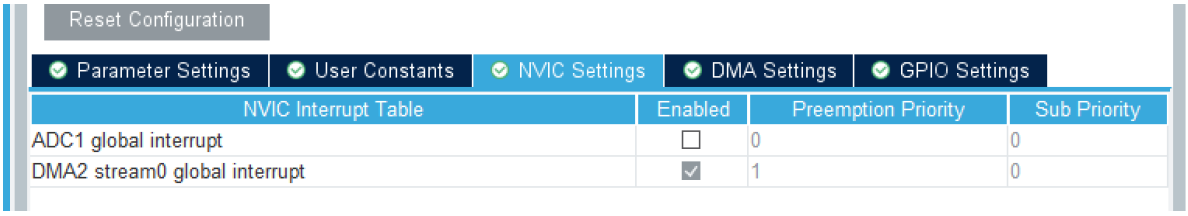
选择一个TIM定时器，分频系数和ARR根据自己的SYSCLK设置，我这里SYSCLK为25MHz，这个定时器作为采样的触发源。



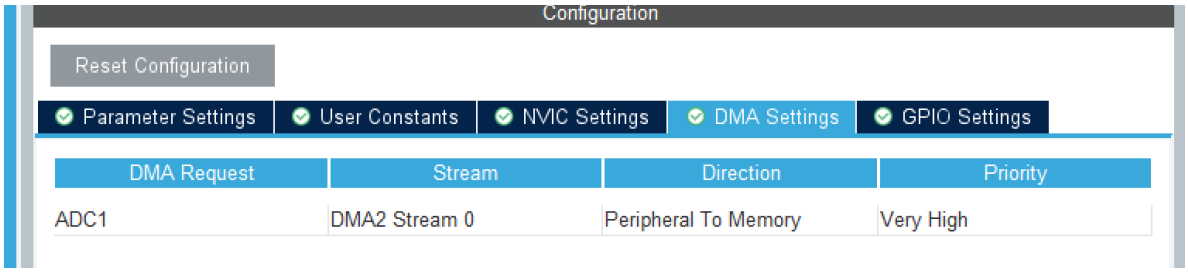
ADC随便开一个通道，设置右对齐，开启DMA连续请求，外部触发源设置为刚刚启用的TIM的Trigger Out Event



启动ADC的DMA中断（没错，后续ADC多次转换完成后，会通过DMA的中断回调）



添加DMA请求



代码

> 写在前面：代码截图中包含了大量的没有提到的变量，这些变量需要自己定义，自己理解用来干什么，如果都贴出来就没意思了。
>
> 示例代码只提供思路，在自己的代码中不一定适用，最好参考个思路然后自己编写，否则移植过去很容易出问题。

初始化的时候启动ADC+DMA模式，设置存放的数组以及采样点数（这里我用了宏定义，**自己编写，照抄无用**）

因为用不到转换完成一半的中断（Halftime）将其关闭。

然后启动刚刚设置的TIM，转换就开始了。

如下图：

```
 */
void osc_init() {
    gui_startup();

    HAL_ADC_Start_DMA( hadc: &hadc1,  pData: (uint32_t *) sample_value,  Length: OSC_SAMPLE_NUM);
    __HAL_DMA_DISABLE_IT(hadc1.DMA_Handle, DMA_IT_HT);

    for (int i = 0; i < 3; i++) {
        HAL_GPIO_WritePin( GPIOx: LED_GPIO_Port,  GPIO_Pin: LED_Pin,  PinState: GPIO_PIN_RESET);
        HAL_Delay( Delay: 500);
        HAL_GPIO_WritePin( GPIOx: LED_GPIO_Port,  GPIO_Pin: LED_Pin,  PinState: GPIO_PIN_SET);
        HAL_Delay( Delay: 500);
    }

    HAL_TIM_Base_Start( htim: &htim3);
}
```

接下来实现中断回调函数：

```
/**
 * @brief ADC转换完成回调函数
 * @details 会在完成全部点采样后被调用
 * @param hadc
 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {
```

这个函数在哪里找？

```
HAL_ADC_Start_DMA( hadc: &hadc1,  pData: (uint32_t *) sample_value,  Length: OSC_SAMPLE_NUM);
```

跳转到它的定义

```
    /* Set the DMA transfer complete callback */
    hadc->DMA_Handle->XferCpltCallback = ADC_DMAConvCplt;
```

继续跳转到DMA传输完成回调函数定义

```
#else
    HAL_ADC_ConvCpltCallback(hadc);
```

找到了

```c
/**
  * @brief  Regular conversion complete callback in non blocking mode
  * @param  hadc pointer to a ADC_HandleTypeDef structure that contains
  *         the configuration information for the specified ADC.
  * @retval None
  */
__weak void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(hadc);
  /* NOTE : This function Should not be modified, when the callback is needed,
            the HAL_ADC_ConvCpltCallback could be implemented in the user file
   */
}
```

我们发现这个函数是弱定义，意味着你需要**在自己的文件中实现它，不要在HAL库中修改！！！**

首先暂停TIM，在我们处理的过程中不要继续采样，不然可能会出一些bug。

```c
/**
 * @brief ADC转换完成回调函数
 * @details 会在完成全部点采样后被调用
 * @param hadc
 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {
    HAL_TIM_Base_Stop( htim: &htim3);
    // 触发
    uint8_t cnt = 0;
```

然后对采集到的点，即数组`sample_value`里的数据进行处理，必须说识别边沿，计算参数等等，自己编写。

在回调函数的最后，对一个标志位置1，然后在主函数中，对处理好的数据进行显示，**中断中不要进行耗时长的操作！！！**

```c
    show_flag = 1;
//    HAL_TIM_Base_Start(&htim3);
}
```

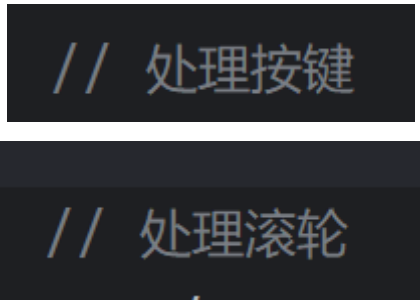
在主函数中进行波形显示的操作，读取标志位，复位标志位。

```
void osc_process(int16_t num, KeyInfo info) {
    static SelectOption option;
    static int16_t ver_ratio = 800;
    static int16_t last_num = 0;

    if (show_flag) {
        show_flag = 0;

        // 处理按键
```

然后，自己实现下面的部分：



复制点主要是为了实现暂停时，显示暂停的波形，同时后台可以继续检测波形。这里我用到了trig_pos这个变量，它是前面中断回调函数中，检测边沿获取的边沿位置，根据边沿位置进行平移，就可以实现触发，稳定显示波形。

```
// 复制需要显示的点
for (int i = 0; i < show_points_num; i++) {
    show_value[i] = sample_value[i + hor_pos + trig_pos] * 3300 * 10 / 4096 / ver_ratio;
}
```

绘制UI部分只简单介绍一下波形显示，

首先根据运行模式，选择主界面进行绘制，传入各种参数。

```
// 绘制UI
if (!is_measure) {
    if (run_state == OSC_RUN) {
        gui_main_interface( points: show_value,  num: show_points_num, run_state, trig_mode,  ver: ver_ratio,
                           hor: hor_arr_list[hor_index] * 10,  trig_level: show_trig_level,  trig_level_mV: show_trig_level_mV, option);
```

然后绘制网格，根据点连线形成波形

```c
/**
 * @brief 主界面GUI
 * @param points 波形的点
 * @param num 波形点的数量
 * @param run_state 运行状态
 * @param trig_mode 触发状态
 * @param ver 垂直偏转系数
 * @param hor 水平偏转系数
 */
void
gui_main_interface(uint16_t *points, uint16_t num, uint8_t run_state, uint8_t trig_mode, uint16_t ver, uint16_t hor,
                   uint16_t trig_level, uint16_t trig_level_mV, SelectOption option) {
    static char str[100];

    ssd1306_Fill( color: Black);

    // Grid
    gui_draw_grid();

    // Waveform
    for (uint16_t i = 1; i < 128; i++) {
        if (points[i - 1] < points[i]) {
            for (int j = points[i - 1]; j <= points[i]; j++) {
                ssd1306_DrawPixel( x: i,  y: 63 - j,  color: White);
            }
        } else {
            for (int j = points[i - 1]; j >= points[i]; j--) {
                ssd1306_DrawPixel( x: i,  y: 63 - j,  color: White);
            }
        }
    }
```

继续运行，启动TIM就会继续自动采集，然后重复上述循环。