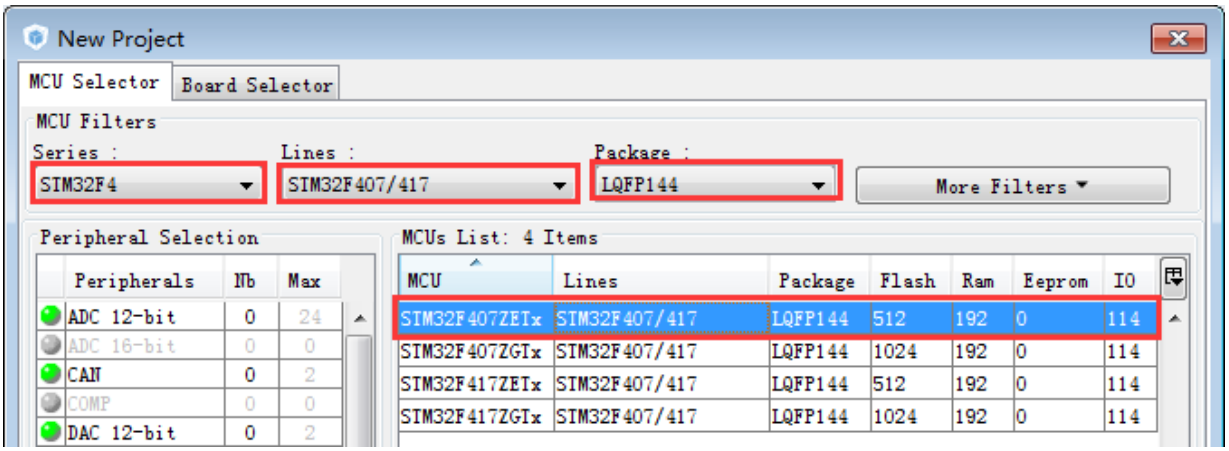
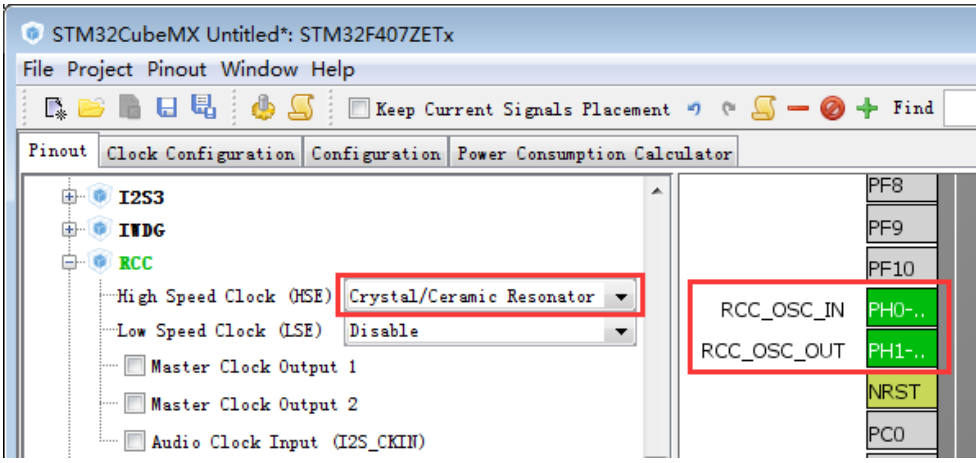
STM32Cube 学习之八：输入捕获

假设已经安装好 STM32CubeMX 和 STM32CubeF4 支持包。

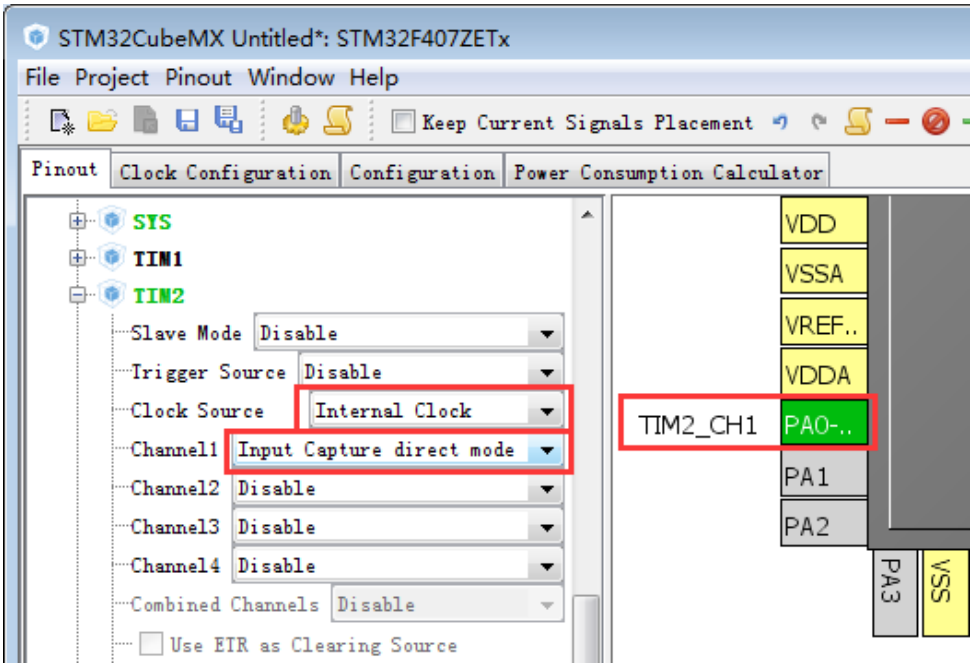
Step1.打开 STM32CubeMX，点击“New Project”，选择芯片型号，STM32F407ZETx。



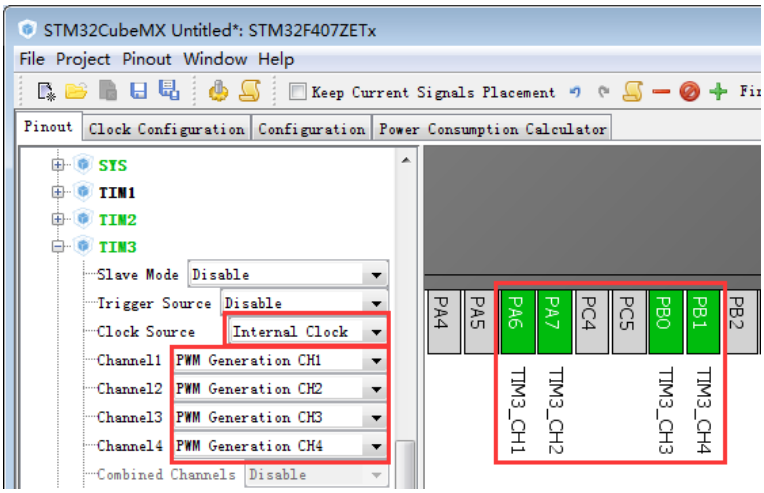
Step2. 在 Pinout 界面下配置晶振输入引脚。



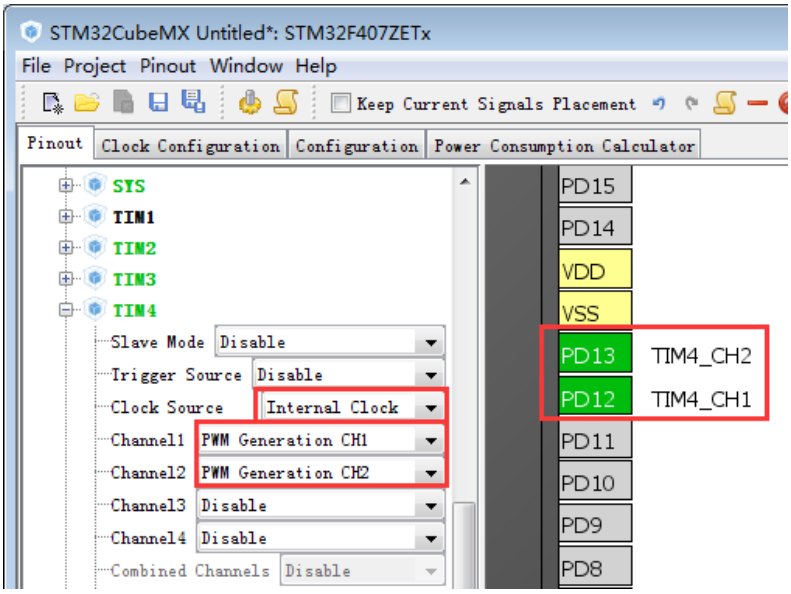
配置 TIM2 使用内部时钟源，CH1 作为输入捕获通道，默认映射到 PA0 引脚。



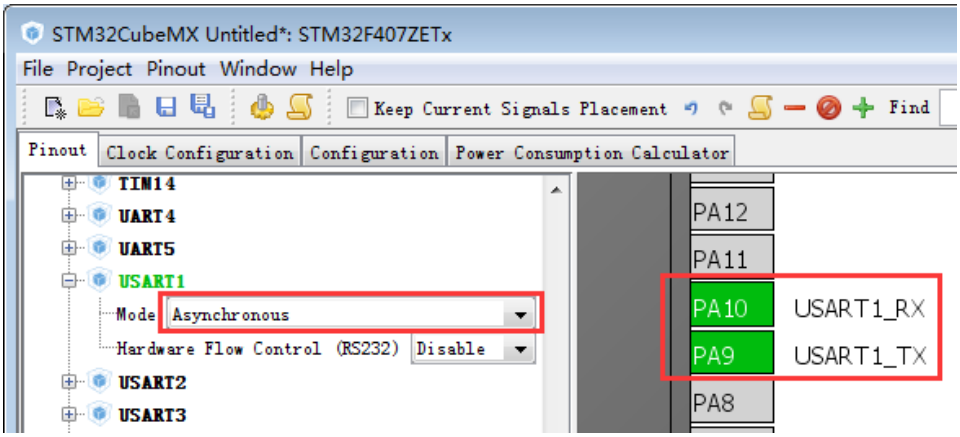
配置 TIM3 使用内部时钟，CH1~CH4 为 PWM 输出通道，默认映射引脚分别为 PA6，PA7，PB0，PB1。



配置 TIM4 使用内部时钟，CH1，CH2 为 PWM 输出通道，映射引脚分别为 PD12，PD13。

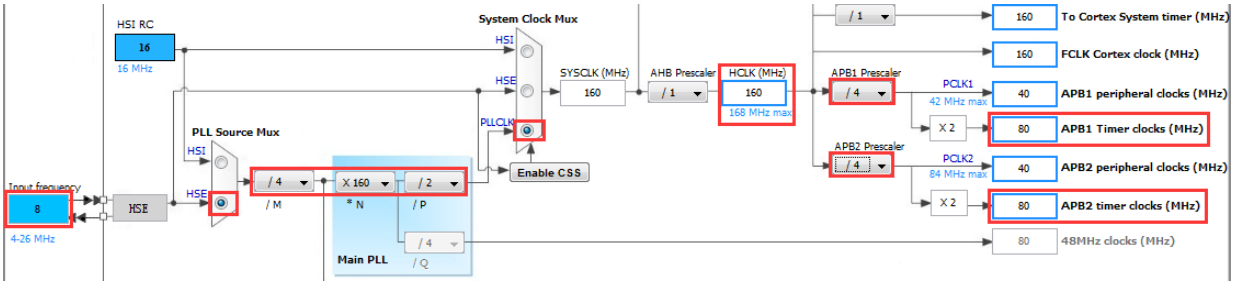


配置串口，作为信息输出接口。



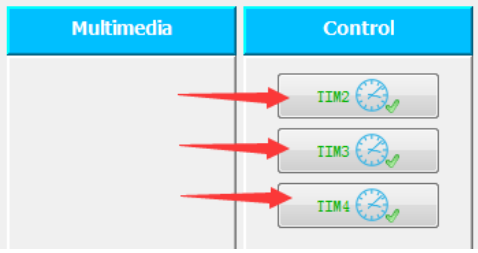
Step3.在 Clock Configuration 界面配置时钟源。

使用外部 8M 晶振作 PLL 时钟输入，并使用 PLL 输出作为系统时钟。为了后面的计算方便，将系统时钟配置成 160MHz。



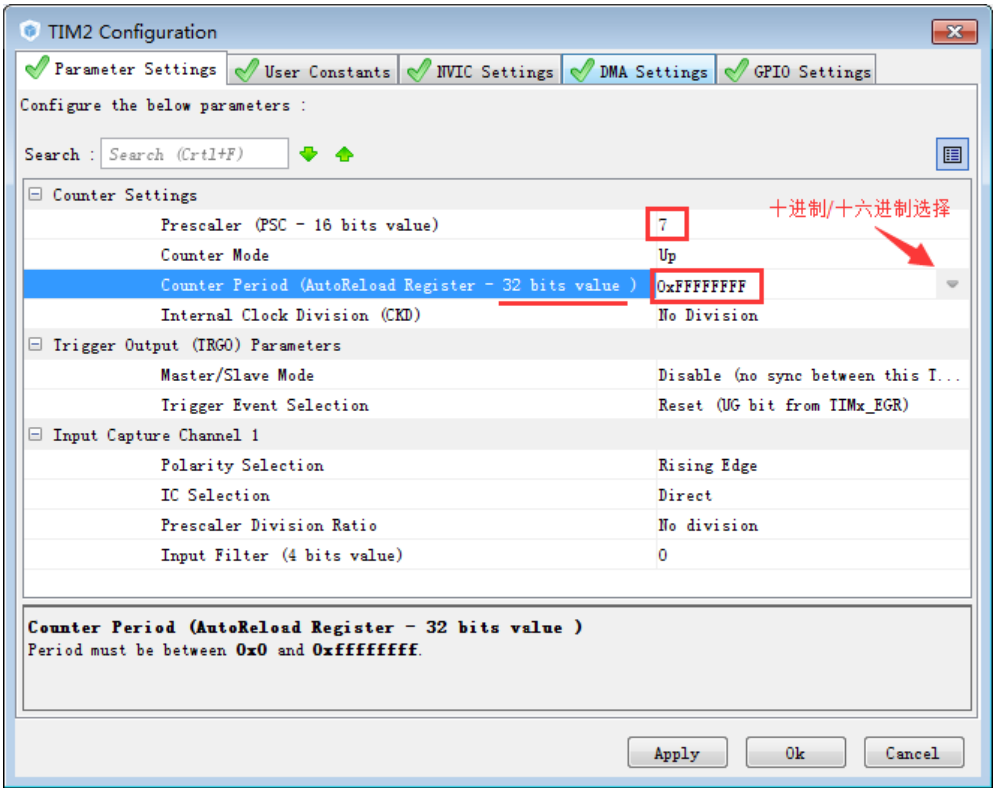
Step4.配置外设参数。

在 configuration 界面中点击 TIM2/ TIM3/ TIM4 按钮，可以进入参数配置界面。



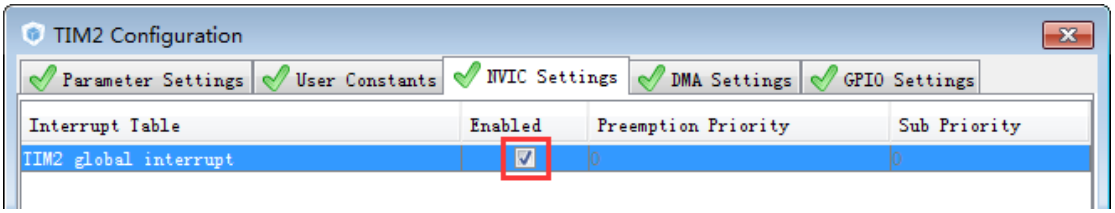
TIM2：在 Parameter Settings 页配置预分频系数为 7，其计数时钟就是 80MHz/(7+1)=10MHz。

计数周期(自动加载值)，转换为十六进制形式，输入 32bit 最大值 0xFFFFFFFF。

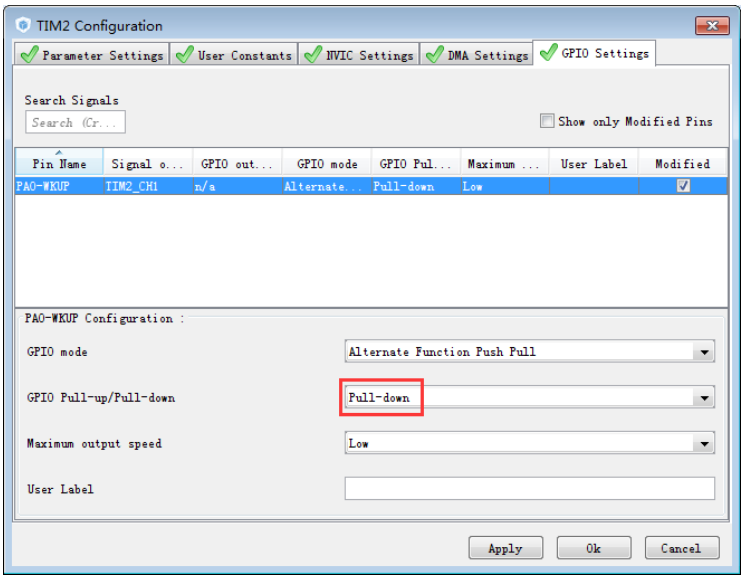


注意，TIM2 的自动加载寄存器 ARR 和各个通道的捕获/比较寄存器 CCRx 都是 32bit 的。

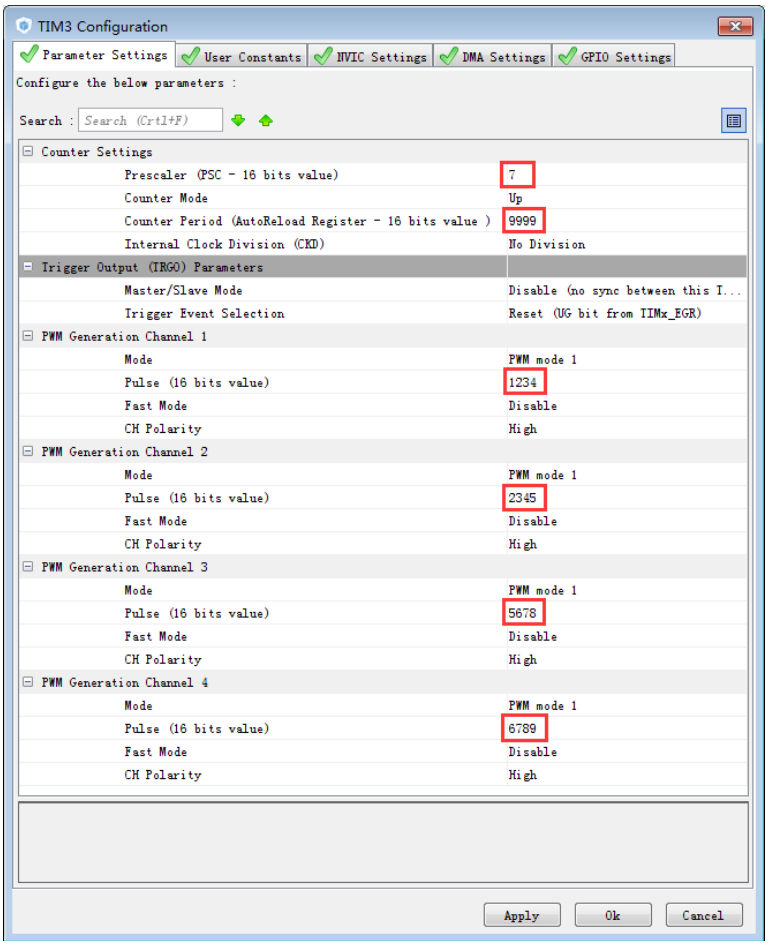
在 NVIC 页面使能捕获/比较中断。



在 GPIO 页面设置捕获输入引脚下拉电阻，设置成上拉也可以，主要是为了使在没有信号输入时在输入引脚上得到稳定的电平。

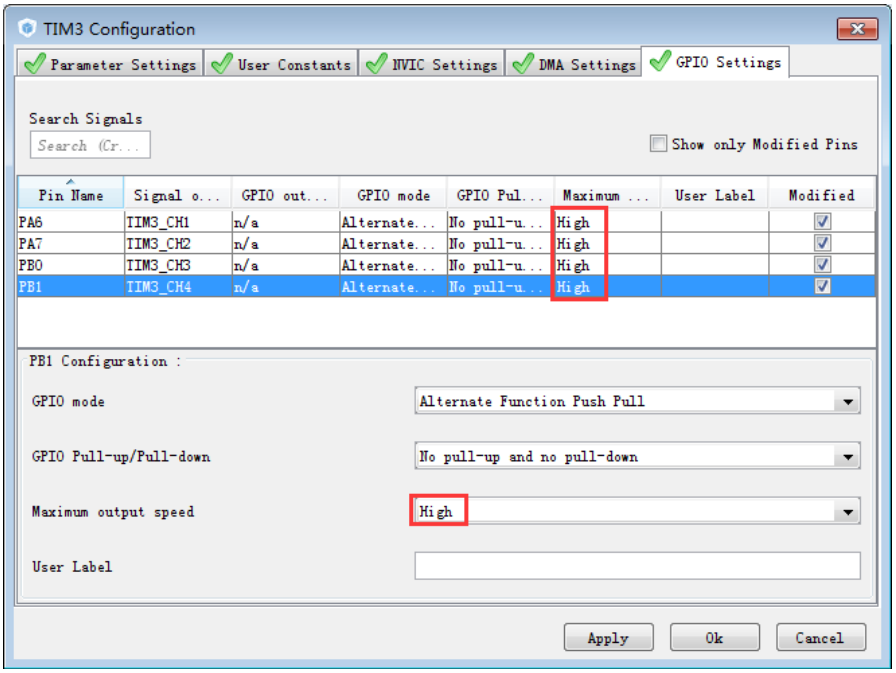


TIM3：在 Parameter 页配置预分频系数为 7，计数周期(自动加载值)为 9999。其溢出频率就是 80MHz/(7+1)/(9999+1)=1kHz，这就是 TIM3 各通道输出的 PWM 信号的频率。

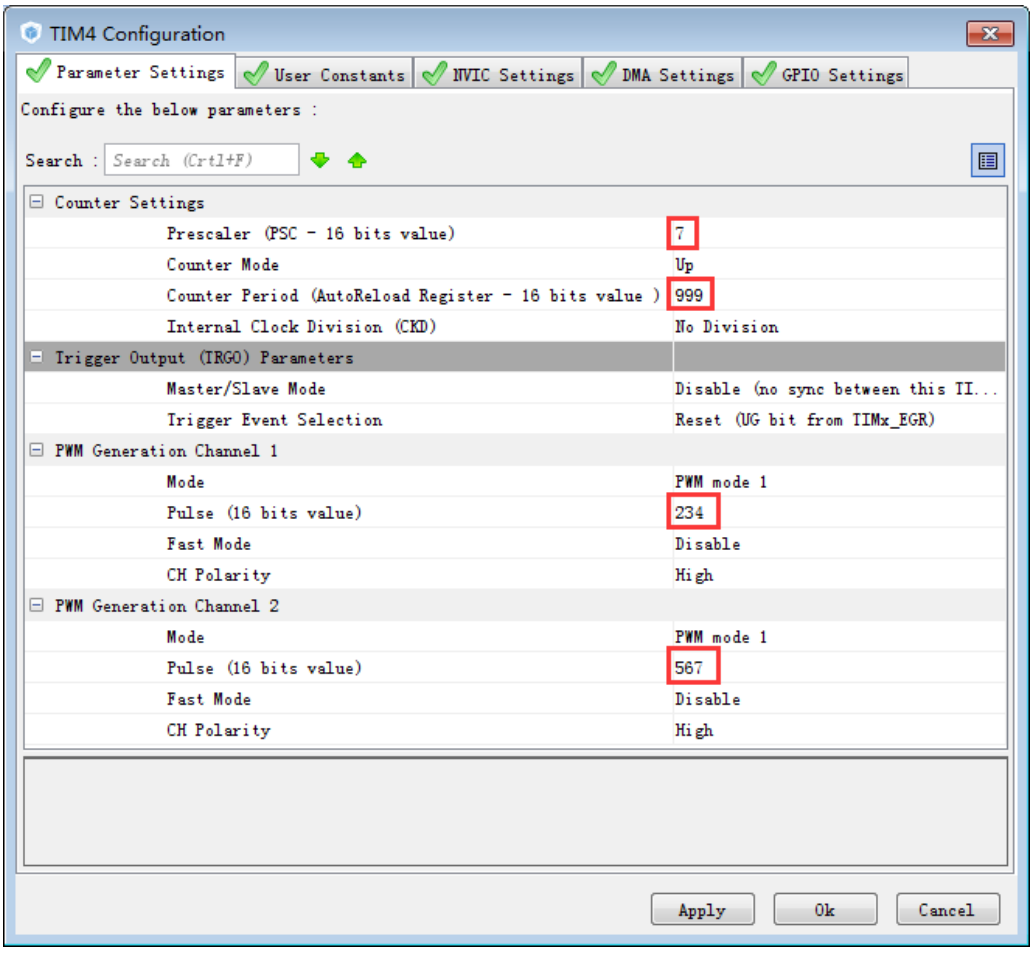


各通道输出 PWM 的占空比参数如上图红框标注，其他参数使用默认值。按照图中参数，CH1~CH4 输出的 PWM 周期都是 1ms，而高电平时间分别是 123.4us，234.5us，567.8us，678.9us。

在 GPIO 页面配置相关引脚的特性。

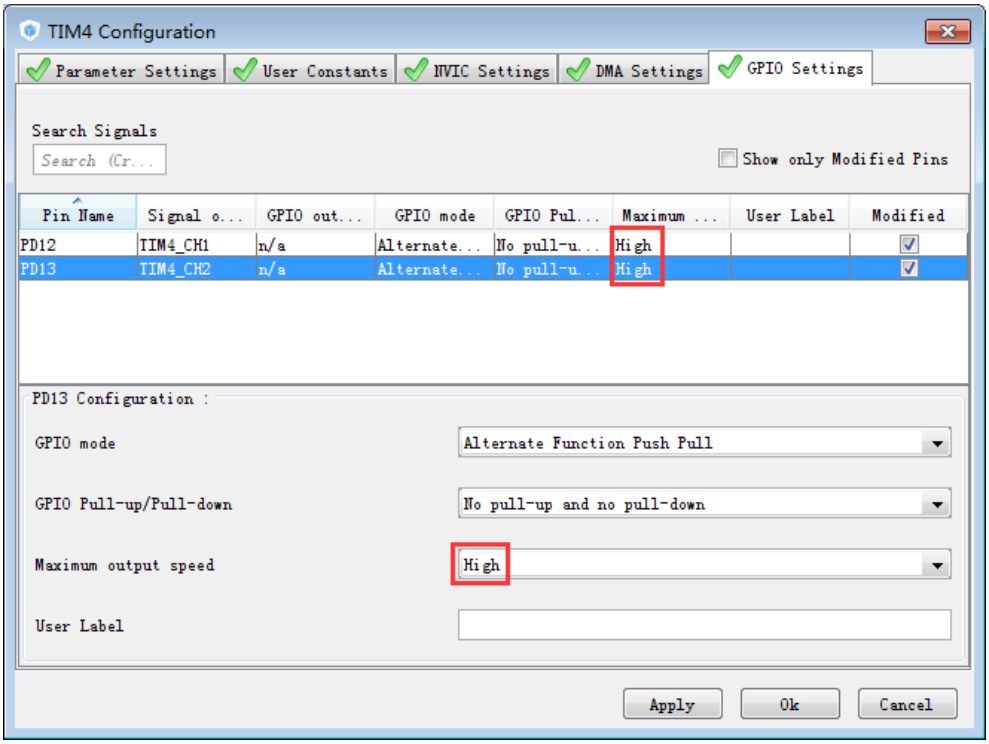


TIM4：在 Parameter 页配置预分频系数为 7，计数周期(自动加载值)为 999。其溢出频率就是 80MHz/(7+1)/(999+1)=10kHz，这就是 TIM4 各通道输出的 PWM 信号的频率。

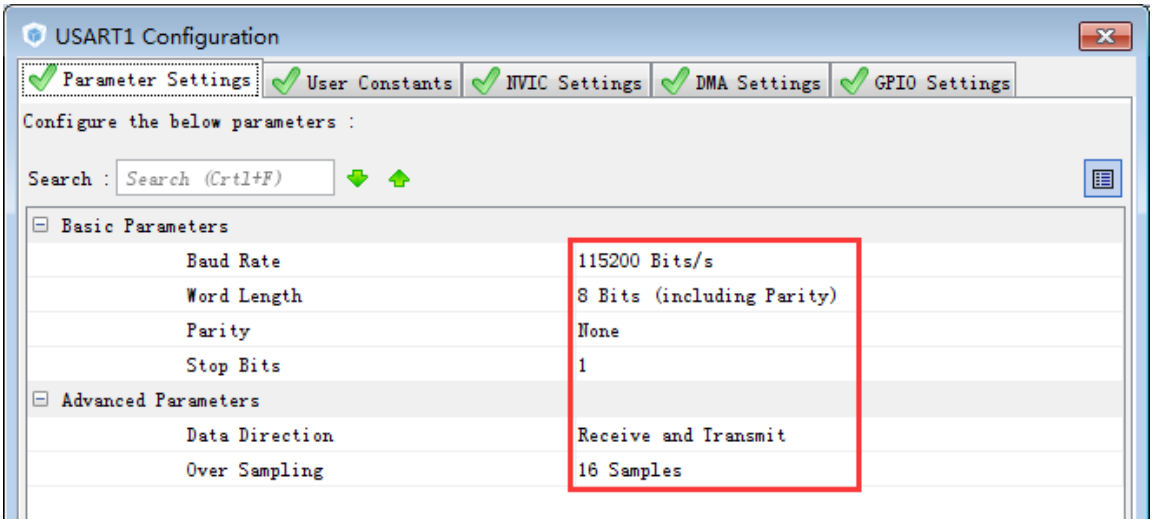


各通道输出 PWM 的占空比参数如上图红框标注，其他参数使用默认值。CH1，CH2 输出的 PWM 周期都是 100us，而高电平时间分别是 23.4us，56.7us。

在 GPIO 页面配置相关引脚的特性。

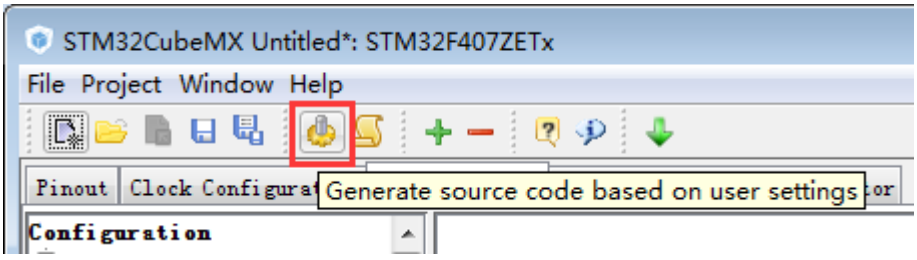


串口参数配置，使用默认值即可。

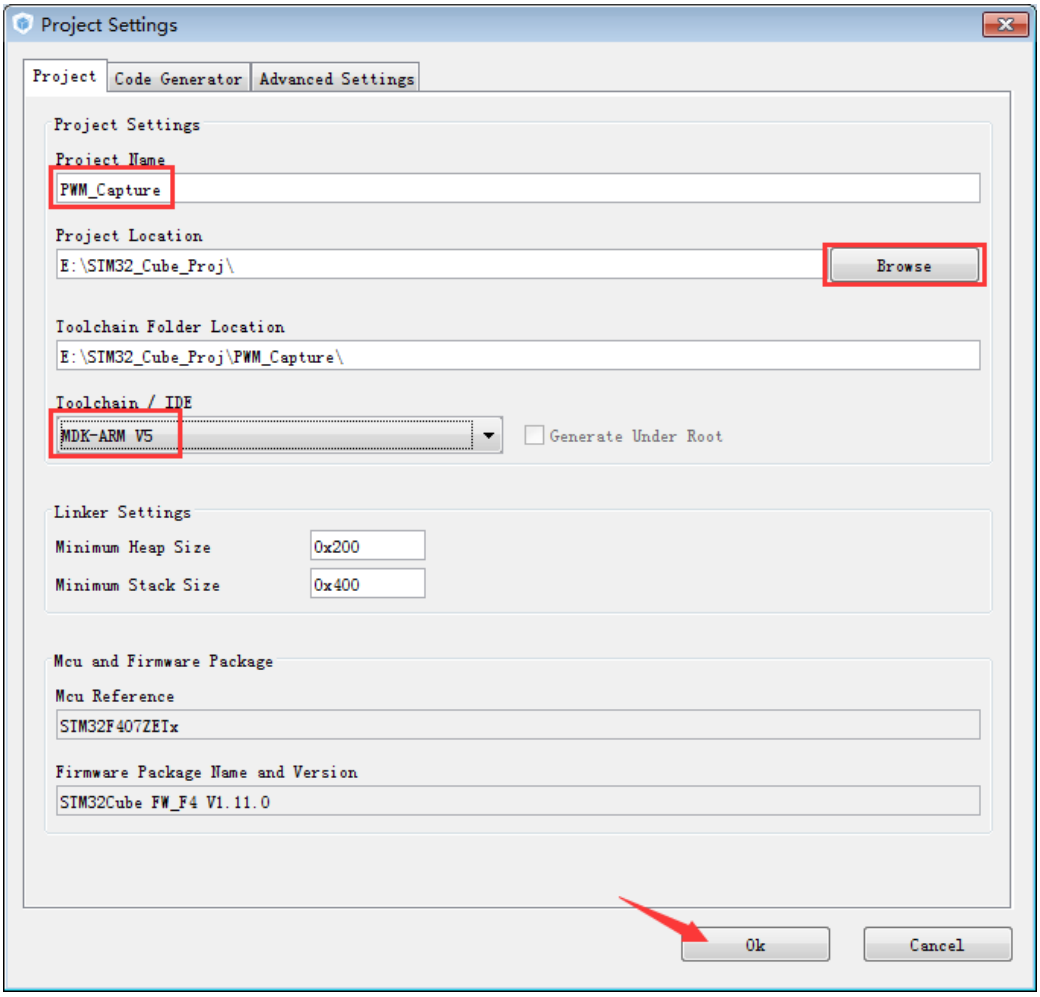


Step5.生成源代码。

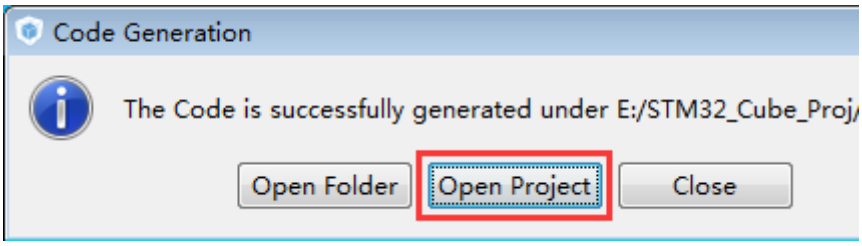
点击生成源代码按钮。



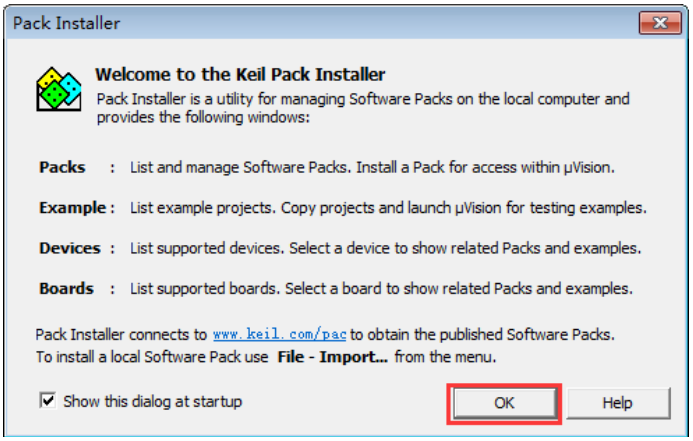
在设置界面中输入工程名，保存路径，工程 IDE 类型，点 OK 即可。



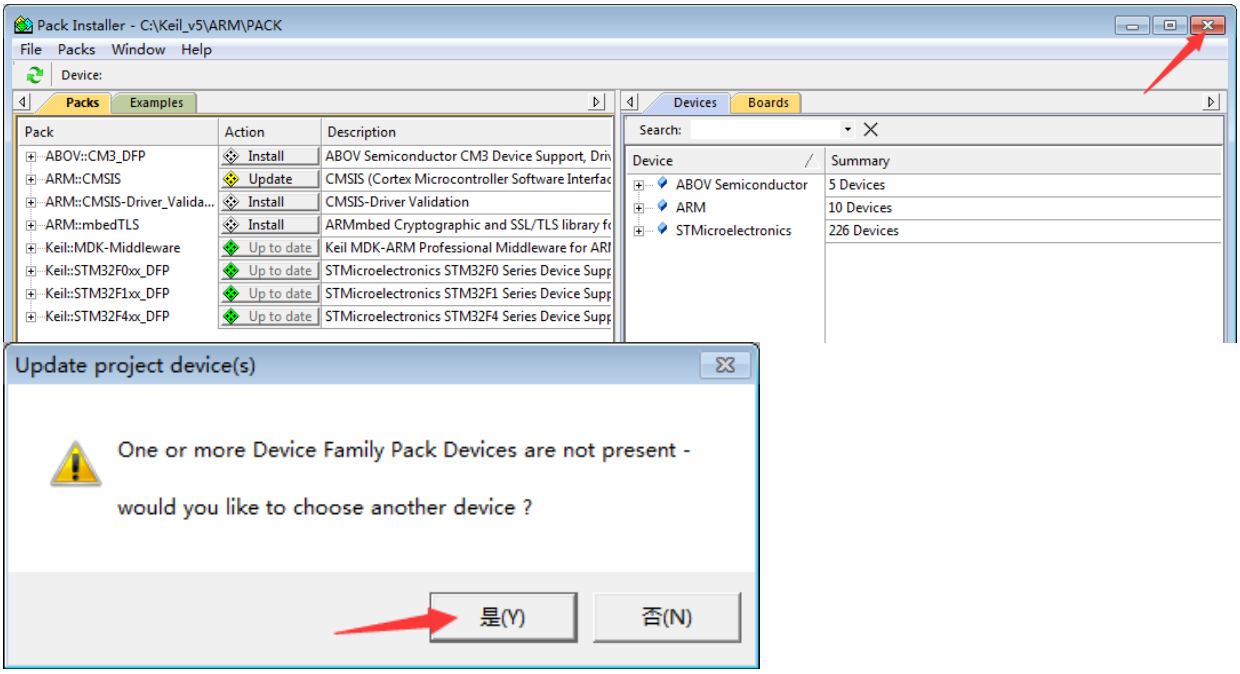
生成代码完成后可直接打开工程。



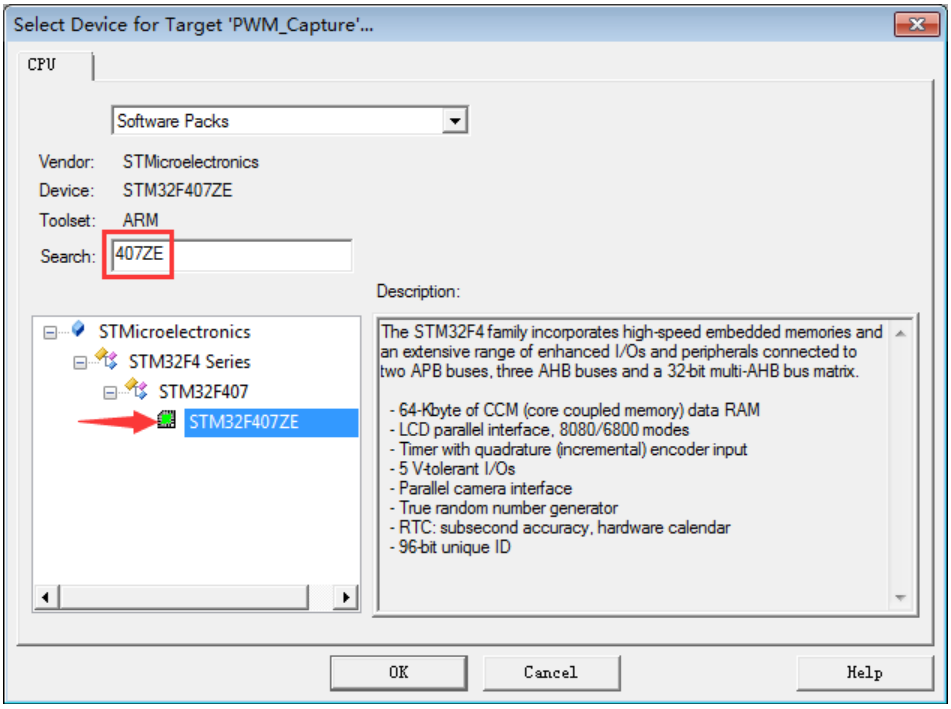
弹出如下对话框时，如果已经安装了 F4 的支持包，则点击 OK 关闭。如果没有安装，则点击界面中的 www.keil.com/...链接，找到芯片的支持包，然后安装。



关闭后面的界面。



点击“是”，然后选择芯片型号。可以在搜索框中输入关键字，加快选择速度。



Step6.添加功能代码。

先在 main.c 文件用户代码区输入包含标准输入输出头文件。

```
33  /* Includes ------------------------------------------------------------------*/
34  #include "stm32f4xx_hal.h"
35
36  /* USER CODE BEGIN Includes */
37  #include <stdio.h>
38  /* USER CODE END Includes */
39
```

在用户代码区 4 实现标准输出 printf()的底层驱动函数 fputc()，功能是在 UART1 输出一个字符。

```
/* USER CODE BEGIN 4 */
int fputc(int ch, FILE *f)
{
    HAL_UART_Transmit(&huart1, (uint8_t*)&ch, 1, 10);
    return ch;
}
```

在主函数前面的用户代码区 0，定义一些全局变量。

```

/* USER CODE BEGIN 0 */
uint32_t capture_buf[3] = {0};
uint8_t capture_cnt = 0;
uint32_t pwm_cycle, high_time;
uint64_t duty;
/* USER CODE END 0 */

```

在 while(1)之前的用户代码区 2，使能 TIM3、TIM4 的各个通道 PWM 输出。

```
  /* USER CODE BEGIN 2 */
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2);
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_4);

    HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
  /* USER CODE END 2 */
```

在 while(1)中的用户代码区 3，写入 TIM2 CH1 通道的输入捕获控制和数据处理。

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
      switch (capture_cnt) {
          case 0:
              capture_cnt++;
              __HAL_TIM_SET_CAPTUREPOLARITY(&htim2, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_RISING);
              HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_1);   // 启动捕获
              break;
          case 4:
              pwm_cycle = capture_buf[2] - capture_buf[0];
              printf("Cycle:%.4fms\r\n", pwm_cycle/10000.0);

              high_time  = capture_buf[1] - capture_buf[0];
              printf("High :%.4fms\r\n", high_time/10000.0);

              duty = high_time;
              duty *= 1000;
              duty /= pwm_cycle;
              printf("Duty :%.1f%%\r\n", duty/10.0);

              HAL_Delay(1000); // 延时1秒
              capture_cnt = 0;
              break;
      }
  }
  /* USER CODE END 3 */
```

在 main 文件的用户代码区 4，写入 TIM2 输入捕获中断处理回调函数。

```c
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if (TIM2 == htim->Instance) {
        if (HAL_TIM_ACTIVE_CHANNEL_1 == htim->Channel) {
            switch (capture_cnt) {
                case 1:
                    capture_buf[0] = __HAL_TIM_GET_COMPARE(htim, TIM_CHANNEL_1);
                    __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_FALLING);
                    capture_cnt++;
                    break;
                case 2:
                    capture_buf[1] = __HAL_TIM_GET_COMPARE(htim, TIM_CHANNEL_1);
                    __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_RISING);
                    capture_cnt++;
                    break;
                case 3:
                    capture_buf[2] = __HAL_TIM_GET_COMPARE(htim, TIM_CHANNEL_1);
                    HAL_TIM_IC_Stop_IT(htim, TIM_CHANNEL_1);   // 停止捕获
                    capture_cnt++;
                    break;
                default:
                    break;
            }
        }
    }
}
/* USER CODE END 4 */
```

至此，工程完成。功能是使用 TIM 的输入捕获功能，实现对 PWM 信号的周期和占空比测量，并将数据通过串口发送出去。用杜邦线将 PA0 和其他 PWM 信号输出脚相连，即可测量信号的周期，高电平所占时间，以及占空比，在串口 1 会输出这些信息。输出信息示例如下：

Cycle:1.0000ms
High :0.1234ms
Duty :12.3%
Cycle:0.1000ms
High :0.0567ms
Duty :56.7%

按照本例的配置，测量精度是 0.1us，测量信号周期范围是 0~0xFFFFFFFF*0.1us，即 0~429.4967295 秒

测量基本思路是：

1.设置 TIM2 CH1 为输入捕获功能；

2.设置上升沿捕获；

3.使能 TIM2 CH1 捕获功能；

4.捕获到上升沿后，存入 capture_buf[0]，改为捕获下降沿；

5.捕获到下降沿后，存入 capture_buf[1]，改为捕获上升沿；

6.捕获到上升沿后，存入 capture_buf[2]，关闭 TIM2 CH1 捕获功能；

7.计算：capture_buf[2] - capture_buf[0]就是周期，capture_buf[1] - capture_buf[0]就是高电平所占时间。

特别说明：

printf()函数的详细使用方法可到网上查找，在此仅解释本例的语句。

printf("Cycle:%.4fms\r\n", pwm_cycle/10000.0);

其中“%f”是输出浮点数的格式，加了“.4”就是保留 4 位小数。

pwm_cycle 是 32bit 无符号整形，除以 10000.0 就是先将其变成小数，在除以 10000。

printf("Duty :%.1f%%\r\n", duty/10.0);
因为%在 printf()函数的格式转换中是格式转换的特殊符号，因此要打印一个“%”时，就要写成“%%”。

HAL_TIM_PWM_Start()函数用于使能定时器某一通道的 PWM 输出。

HAL_TIM_IC_Start_IT()函数用于使能定时器某一通道的输入捕获功能，并使能相应的中断。对应的 HAL_TIM_IC_Stop_IT()函数和其功能相反，是关闭定时器某一通道的输入捕获功能和相应中断。

__HAL_TIM_SET_CAPTUREPOLARITY 不是函数，而是底层操作的一个宏定义。
在 stm32f4xx_hal_tim.h 文件中可以找到。其作用是修改定时器某一通道的输入捕获极性。

```
1117 #define __HAL_TIM_SET_CAPTUREPOLARITY(__HANDLE__, __CHANNEL__, __POLARITY__)                \
1118                        do{                                                                  \
1119                            TIM_RESET_CAPTUREPOLARITY((__HANDLE__), (__CHANNEL__));          \
1120                            TIM_SET_CAPTUREPOLARITY((__HANDLE__), (__CHANNEL__), (__POLARITY__)); \
1121                          }while(0)
```

__HAL_TIM_GET_COMPARE 也是一个宏定义。
在 stm32f4xx_hal_tim.h 文件中可以找到。其作用是获取定时器某一通道的捕获/比较寄存器值。

```
974 #define __HAL_TIM_GET_COMPARE(__HANDLE__, __CHANNEL__) \
975   (*(__IO uint32_t *)(&((__HANDLE__)->Instance->CCR1) + ((__CHANNEL__) >> 2U)))
```

根据我使用 CubeMX 开发的经验，发现 HAL 库并没有把所有的操作都封装成函数。对于底层的寄存器操作(如本例中的读取捕获/比较寄存器)，还有修改外设的某个配置参数(如本例中的改变输入捕获的极性)，HAL 库会使用宏定义来实现。而且会用__HAL_作为这类宏定义的前缀。获取某个参数，宏定义中一般会有_GET，而设置某个参数的，宏定义中就会有_SET。在开发过程中，如果遇到寄存器级别或者更小范围的操作时，可以到该外设的头文件中查找，一般都能找到相应的宏定义。

官方例程请参考 stm32cubef4.zip 解压后 STM32Cube_FW_F4_V1.11.0\Projects\STM324xG_EVAL\Examples\TIM\TIM_InputCapture 目录下的工程。

S.D.Lu 于 深圳
2016 年 8 月