

Lab3 时间

- Lab3 时间
 - 实验目的
 - 实验器材
 - 硬件
 - 软件
 - 实验原理
 - 连接电路图
 - TIM定时器
 - 实验步骤
 - 恢复实验2的连线和程序;
 - 编写程序, 从串口输出测量到的按钮按下的时间长度(同样要注意去抖动);
 - 编写程序, 从某个GPIO口输出方波;
 - 组内两位同学交叉连线, 一位同学的MCU输出单次脉冲, 另一位同学的MCU测量这个脉冲的高电平时间长度 (注意两块板子要共地) 。
 - 扩展内容
 - 用引脚中断来处理按钮和测量时间;
 - 试探能够产生和测量的最小脉冲宽度(以下脉冲宽度均代指高电平脉冲宽度)。
 - 定时器输出PWM波形以及捕捉PWM波形原理
 - 输出最小脉冲宽度
 - 接收最小脉冲宽度
 - 实验心得

实验目的

- 熟练掌握编写MCU程序的一般方法;
- 掌握MCU的中断响应程序编写方法;
- 熟练掌握MCU的时钟中断配置和时钟中断程序响应程序的编写方法;
- 掌握MCU程序的定时和获得时间的方法 (采用SysTick和定时器中断两种方法) 。

实验器材

硬件

- STM32F103核心板1块;
- ST-Link 1个;
- 杜邦线(孔-孔) 4根;
- 杜邦面包线 (孔-针) 3根;
- 面包线 (针-针) 若干;
- 按钮1个;
- 面包板1块。

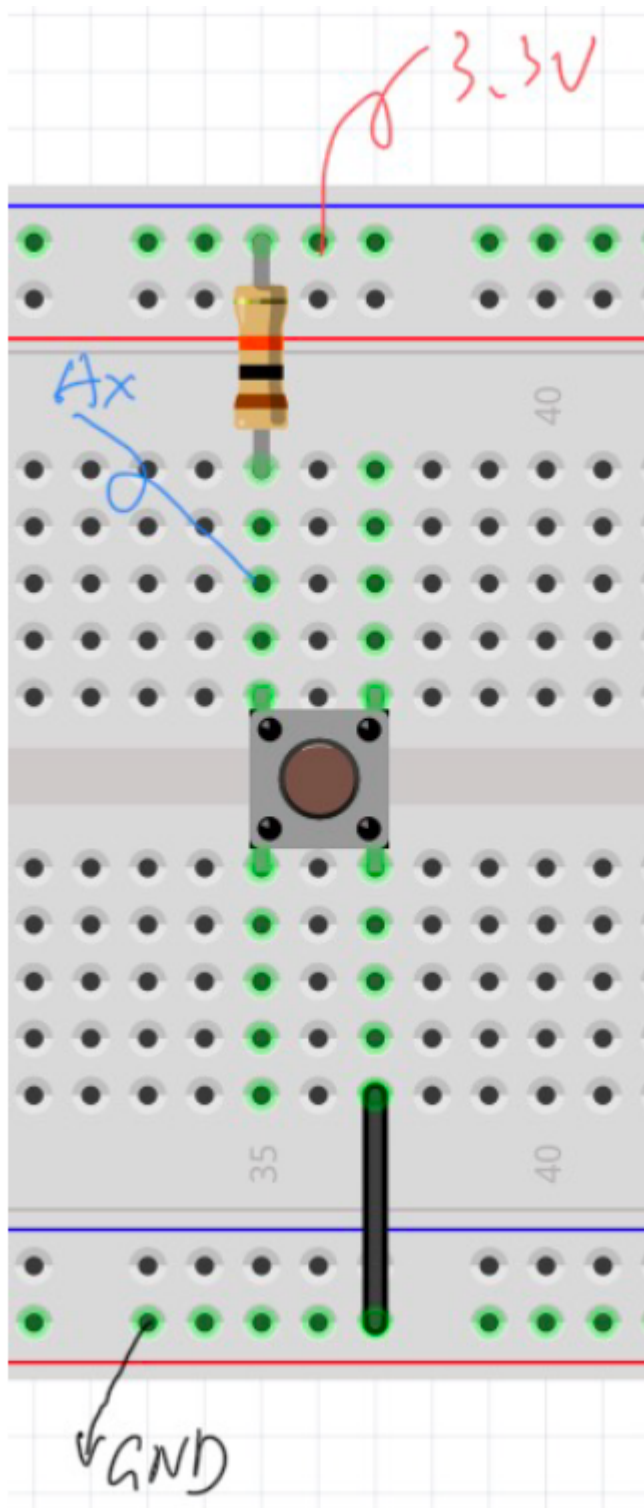
软件

- STM32CubeIDE;

- Putty
- Serial Port Utility(友善串口调试助手)

实验原理

连接电路图



- 按钮连接
- CP2102连接

TIM定时器

- STM32F1 系列中，除了互联型的产品，共有8个定时器，分为基本定时器，通用定时器和高级定时器。基本定时器TIM6和TIM7是一个16位的只能向上计数的定时器，只能定时，没有外部IO。通用定时器

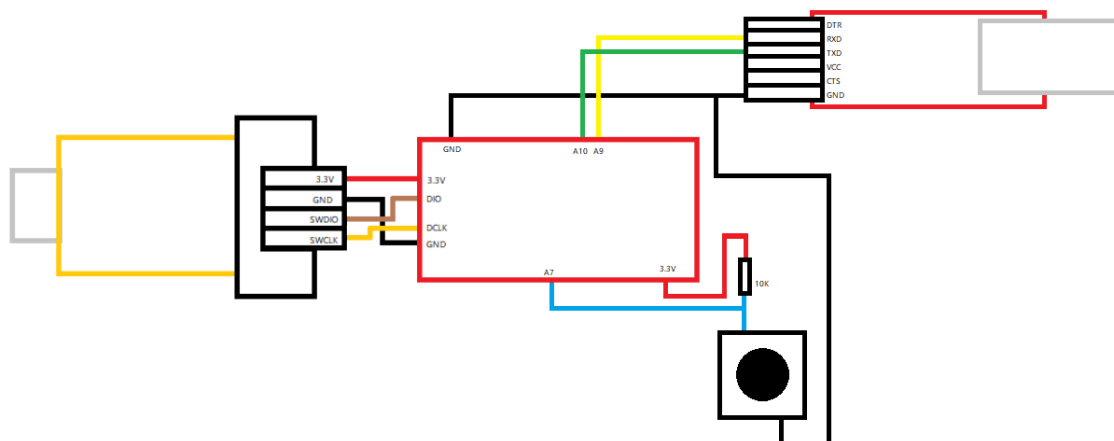
TIM2/3/4/5是一个16位的可以向上/下计数的定时器，可以定时，可以输出比较，可以输入捕捉，每个定时器有四个外部IO。高级定时器TIM1/8是一个16位的可以向上/下计数的定时器，可以定时，可以输出比较，可以输入捕捉，还可以有三相电机互补输出信号，每个定时器有8个外部IO。

- 基本定时器的核心是时基，通用计时器和高级定时器也有。
- 时钟源
 - 定时器时钟TIMxCLK，即内部时钟CK_INT，经APB1预分频器后分频提供，如果APB1预分频系数等于1，则频率不变，否则频率乘以2，库函数中APB1预分频的系数是2，即 $PCLK1 = 36M$ ，所以定时器时钟 $TIM_xCLK = 36 * 2 = 72M$ 。
- 计数器时钟
 - 定时器时钟经过PSC预分频器之后，即CK_CNT，用来驱动计数器计数。PSC是一个16位的预分频器，可以对定时器时钟TIMxCLK进行1~65536之间的任何一个数进行分频。具体计算方式为： $CK_CNT = TIM_xCLK / (PSC + 1)$ 。
- 计数器
 - 计数器CNT是一个16位的计数器，只能往上计数，最大计数值为65535。当计数达到自动重载寄存器的时候产生更新事件，并清零从头开始计数。
- 自动重载寄存器
 - 自动重载寄存器ARR是一个16位的寄存器，这里面装着计数器能计数的最大数值。当计数到这个值的时候，如果使能了中断的话，定时器就产生溢出中断。
- 定时时间的计算
 - 定时器的定时时间等于计数器的中断周期乘以中断的次数。计数器在CK_CNT的驱动下，
 - 计一个数的时间则是CK_CLK的倒数，等于： $1 / (TIM_xCLK / (PSC + 1))$ ，
 - 产生一次中断的时间则等于： $1 / (CK_CLK * ARR)$ 。如果在中断服务程序里面设置一个变量time，用来记录中断的次数，
 - 那么就可以计算出我们需要的定时时间等于： $1 / CK_CLK * (ARR + 1) * time$ 。

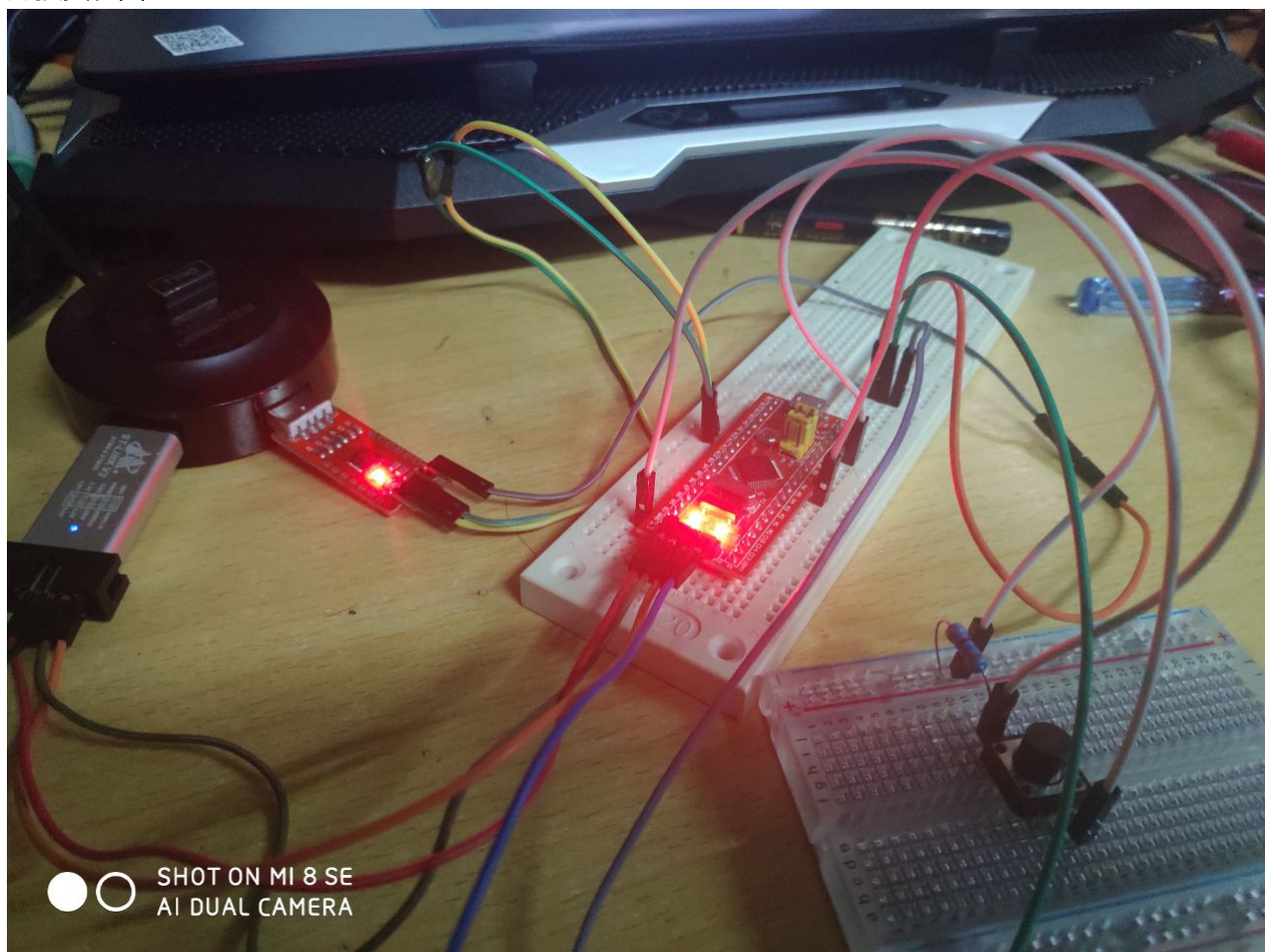
实验步骤

恢复实验2的连线 and 程序；

- 连接示意图



- 连接实物图



编写程序，从串口输出测量到的按钮按下的时间长度(同样要注意去抖动);

- 代码

```
/* stm32f1xx_it.c中的SysTick_Handler()函数中加入自定义函数Key_Scan进行去抖动操作 */
/* SysTick是一个24位的系统节拍定时器system tick timer, SysTick, 具有自动重载和溢出中断功能, 中断周期由函数HAL_SYSTICK_Config(SystemCoreClock / (1000U / uwTickFreq))确定, 默认值为1ms。*/
void SysTick_Handler(void)
{
    /* USER CODE BEGIN SysTick_IRQn 0 */
    Key_Scan();    // 每隔1ms, 触发中断扫描按钮状态, 若与上一次扫描结果相同, 则视为状态已经稳定
    /* USER CODE END SysTick_IRQn 0 */
    HAL_IncTick();

    HAL_SYSTICK_IRQHandler();
    /* USER CODE BEGIN SysTick_IRQn 1 */
    /* USER CODE END SysTick_IRQn 1 */
}
/* stm32f1xx_it.c中的版本1Key_Scan 使用的是RCC时钟进行计时, 缺点就是无法精确到ms。*/
extern uint8_t BtnState;
extern uint8_t BtnChangeFlag;
extern uint8_t hour_down, minute_down, second_down;
extern uint8_t hour_up, minute_up, second_up;
```

```

void Key_Scan(void)
{
    if(HAL_GPIO_ReadPin(BTN_GPIO_Port,BTN_Pin) == GPIO_PIN_RESET)
//判断有没有按下按钮
    {
        btnCount++;
        if(btnCount == 2)           // 上1ms的状态与当前状态都是按下按钮
        {
            BtnState=1;                //设置按钮为按下标志
            if(pushFlag == 0)          // 如果已经此前没有检测到按钮被按下过，则
            {
                HAL_RTC_GetTime(&hrtc, &Nowtime, RTC_FORMAT_BIN);    // RTC时钟获取
当前时间

                // 分别记录时分秒
                hour_down = Nowtime.Hours;
                minute_down = Nowtime.Minutes;
                second_down = Nowtime.Seconds;
                pushFlag = 1;          // 按键已经按下，避免按钮长按的时候计时出
问题

            }
            btnCount = 0;              // 重置
        }
    }
    else                             // 按钮没有按下
    {
        btnCount = 0;
        if(BtnState == 1)             // 上一个状态是按下按钮，说明此时按按钮动作结
束一次
        {
            HAL_RTC_GetTime(&hrtc, &Nowtime, RTC_FORMAT_BIN);    // 获取当前时间
            // 分别存储时分秒
            hour_up = Nowtime.Hours;
            minute_up = Nowtime.Minutes;
            second_up = Nowtime.Seconds;

            BtnChangeFlag = 1;         // 说明按钮状态发生了变化
            pushFlag = 0;

        }
        else                           // 上一个状态就是没有按下去的
        {
            BtnChangeFlag = 0;
        }
        BtnState = 0;
    }
}

/* main.c中的版本1串口通信模块，方法就是把获得数字各位放转换成ASCII码再输出 */
uint8_t BtnState = 0;
uint8_t BtnChangeFlag = 0;
uint8_t hour_down, minute_down, second_down;
uint8_t hour_up, minute_up, second_up;
uint8_t output[8];
if(BtnChangeFlag == 1)               // 按钮结束按下状态
{
    HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
}

```

```

// 计算时间
output[2] = ':';
output[5] = ':';
// 秒
if(second_up < second_down)
{
    second_up = 60;
    minute_up--;
}
output[6] = (second_up - second_down) / 10 + '0';
output[7] = (second_up - second_down) % 10 + '0';
// 分
if(minute_up < minute_down)
{
    minute_up = 60;
    hour_up--;
}
output[3] = (minute_up - minute_down) / 10 + '0';
output[4] = (minute_up - minute_down) % 10 + '0';
// 时
if(hour_up < minute_down)
{
    hour_down = -hour_down;
}
output[0] = (hour_up - hour_down) / 10 + '0';
output[1] = (hour_up - hour_down) % 10 + '0';
HAL_UART_Transmit(&huart1, &output, 8, 10000);
BtnChangeFlag = 0;
}
/* stm32f1xx_it.c中的版本2 Key_Scan 使用的是HAL_GetTick进行计时 */
extern uint32_t now, last;
void Key_Scan(void)
{
    if(HAL_GPIO_ReadPin(BTN_GPIO_Port,BTN_Pin) == GPIO_PIN_RESET)
//判断有没有按下按钮
    {
        btnCount++;
        if(btnCount == 2)                // 上1ms的状态与当前状态都是按下按钮
        {
            BtnState=1;                    //设置按钮为按下标志
            if(pushFlag == 0)              // 如果已经此前没有检测到按钮被按下过，则
            {
                last = HAL_GetTick();
                pushFlag = 1;              // 按键已经按下，避免按钮长按的时候计时出
            }
            btnCount = 0;                  // 重置
        }
    }
    else                                  // 按钮没有按下
    {
        btnCount = 0;
        if(BtnState == 1)                // 上一个状态是按下按钮，说明此时按按钮动作结

```

问题

束一次


```

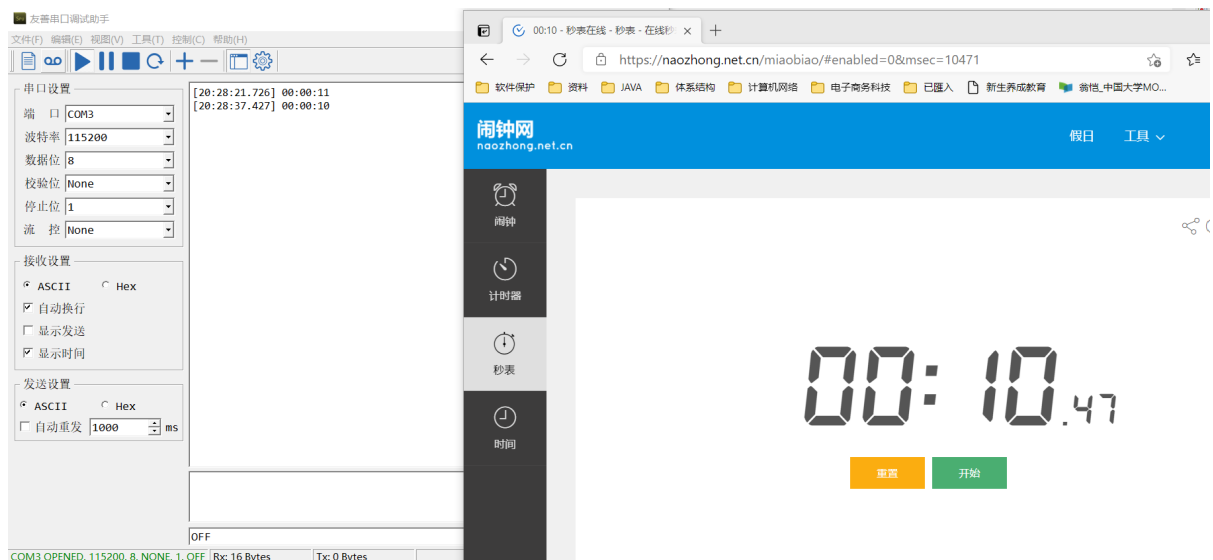
    {
        now = HAL_GetTick();
        BtnChangeFlag = 1;    // 说明按钮状态发生了变化
        pushFlag = 0;
    }
    else    // 上一个状态就是没有按下去的
    {
        BtnChangeFlag = 0;
    }
    BtnState = 0;
}

/* main.c中的版本2串口通信模块 */
uint32_t now, last;
if(BtnChangeFlag == 1)    // 按钮结束按下状态
{
    HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
    char buf[64];
    sprintf(buf, "%lu ms\r\n", now - last);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, 8, 10000);
    BtnChangeFlag = 0;
}

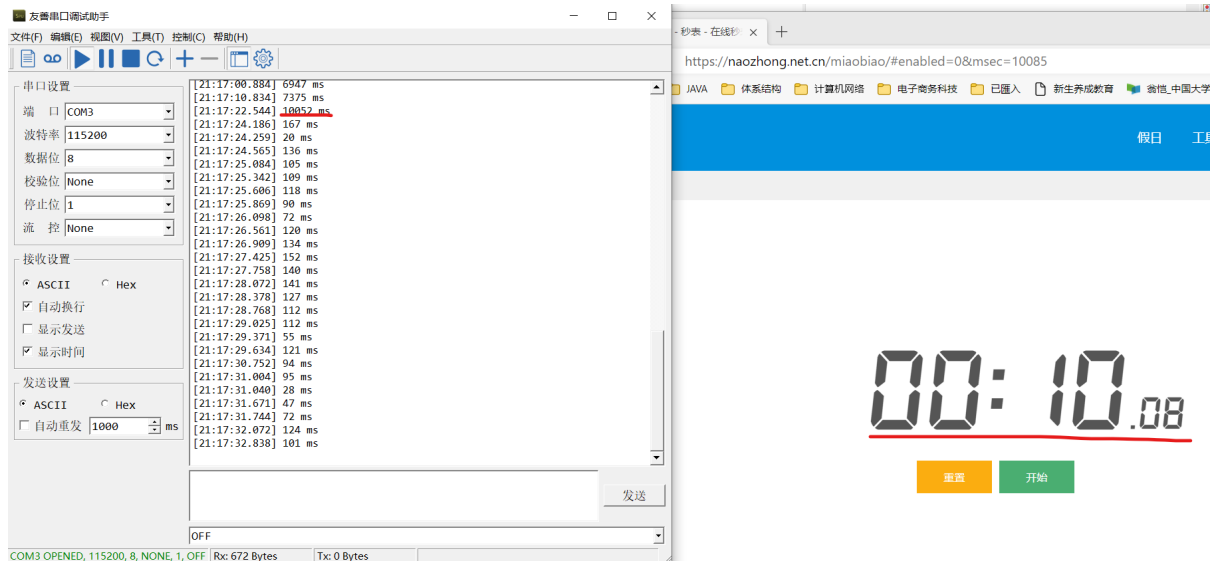
```

- 结果

- 版本1



版本2

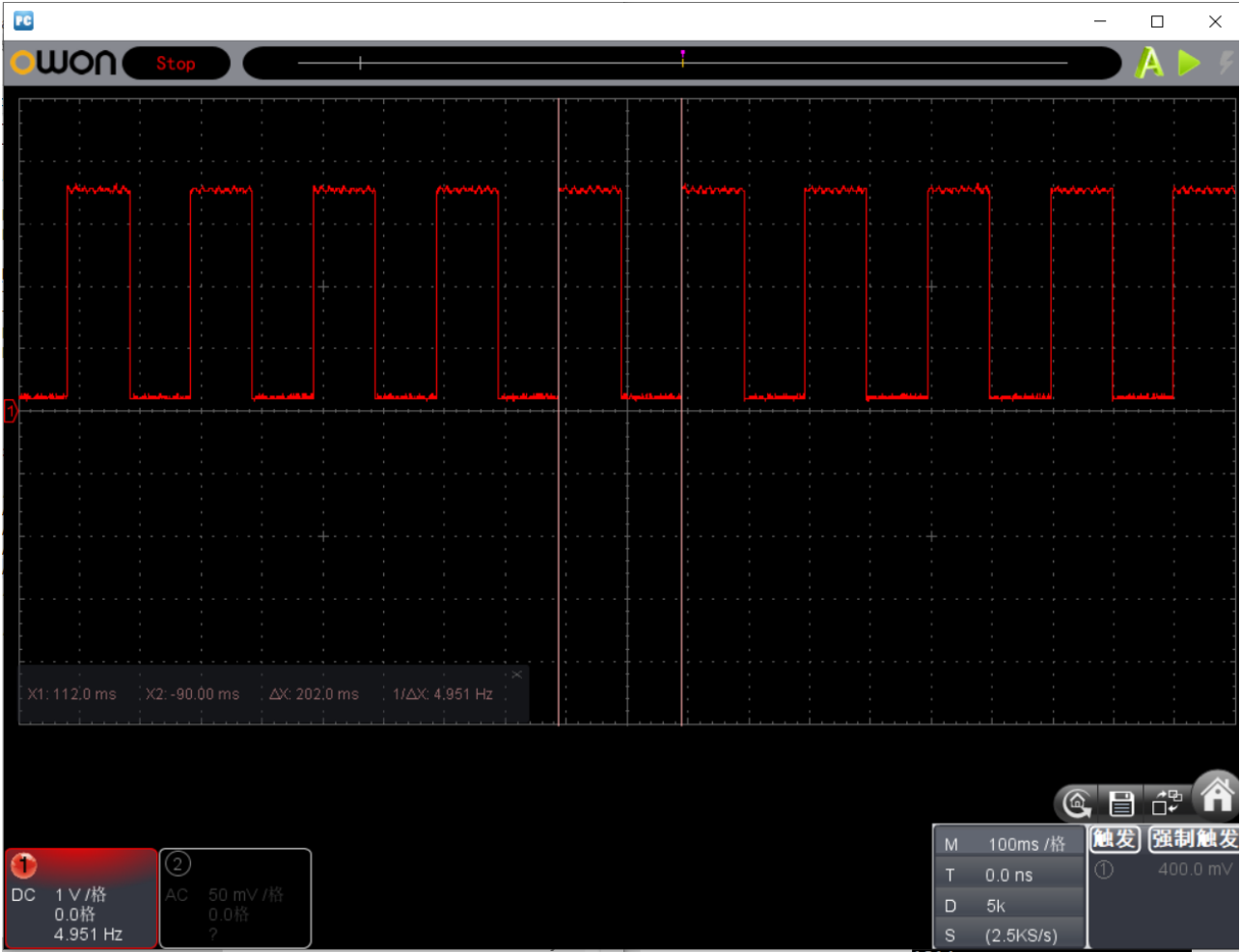


编写程序，从某个GPIO口输出方波；

- 代码

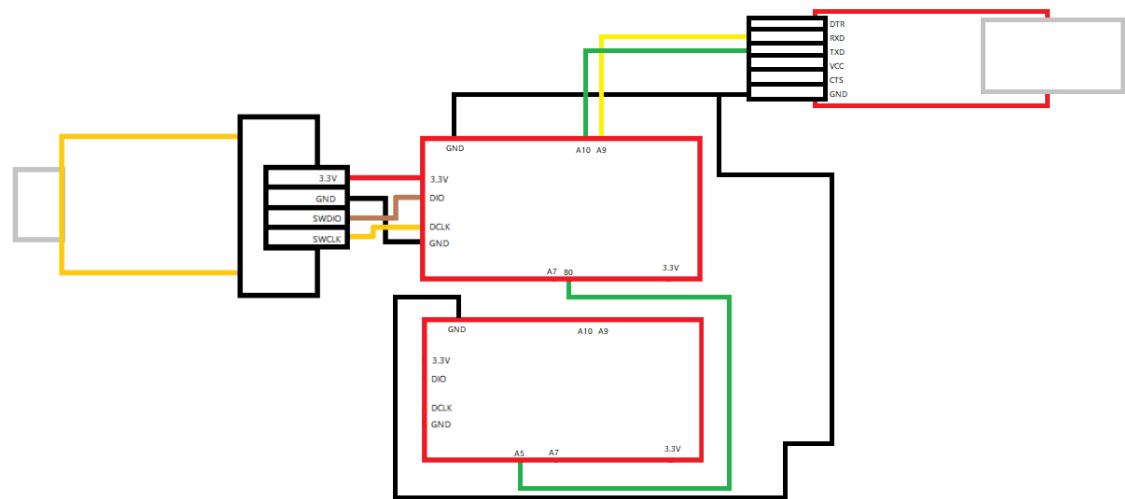
```
/* 配置PB0引脚为输出方波的GPIO口 */
HAL_GPIO_WritePin(Out_GPIO_Port, Out_Pin, GPIO_PIN_SET);
HAL_Delay(500);
HAL_GPIO_WritePin(Out_GPIO_Port, Out_Pin, GPIO_PIN_RESET);
HAL_Delay(500);
```


• 结果



组内两位同学交叉连线，一位同学的MCU输出单次脉冲，另一位同学的MCU测量这个脉冲的高电平时间长度（注意两块板子要共地）。

• 连线示意图



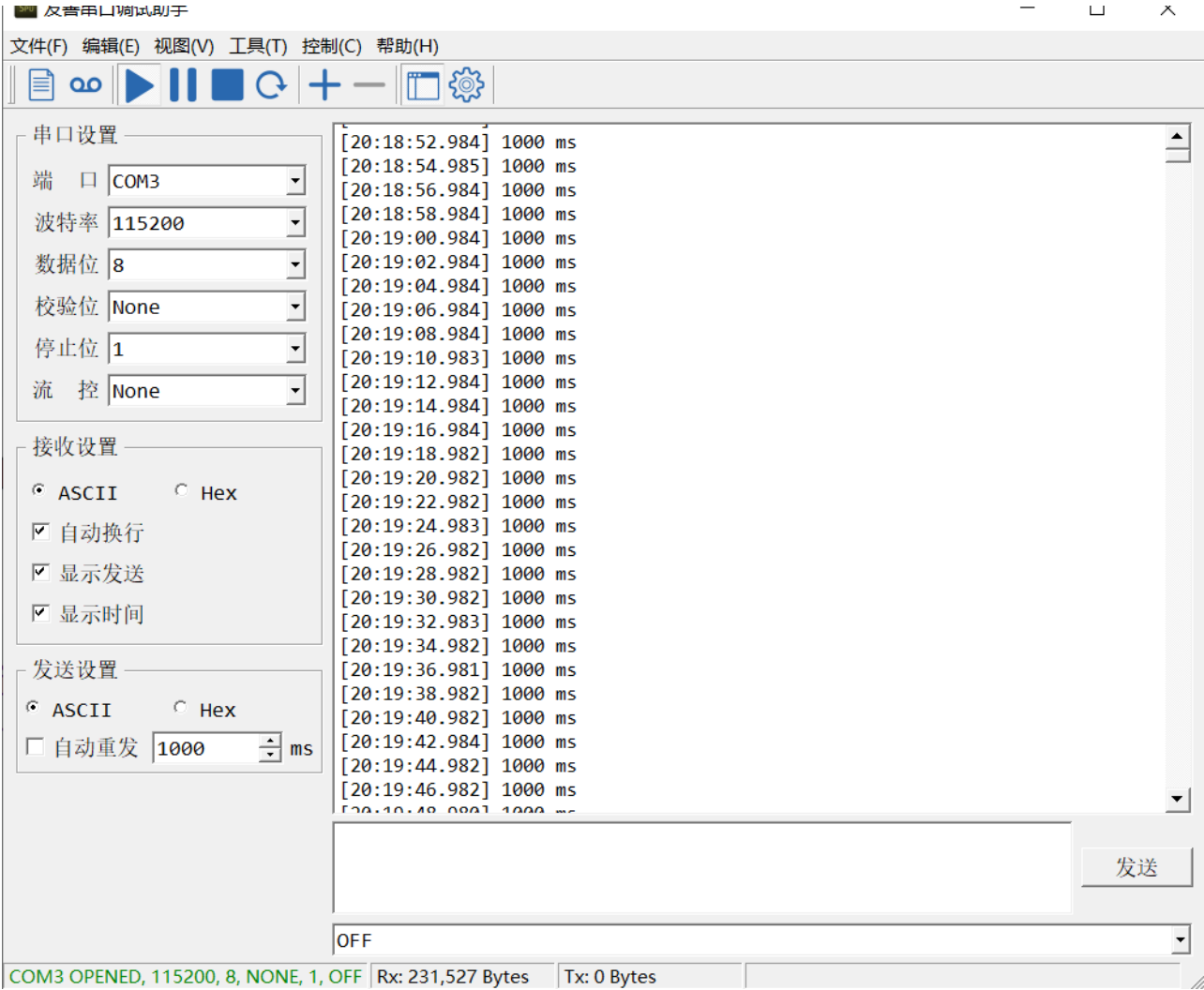
- 连线实物图



- 代码

```
/* main.c */
// 输出板
HAL_GPIO_WritePin(Out_GPIO_Port, Out_Pin, GPIO_PIN_SET);
HAL_Delay(500);
HAL_GPIO_WritePin(Out_GPIO_Port, Out_Pin, GPIO_PIN_RESET);
// 接收板, 配置PA5作为接收方波的GPIO口
char buf[64];
uint32_t last_time = 0;
if(HAL_GPIO_ReadPin(INPUT_GPIO_Port, INPUT_Pin) == GPIO_PIN_SET)
{
    last_time = HAL_GetTick();
    while(HAL_GPIO_ReadPin(INPUT_GPIO_Port, INPUT_Pin) == GPIO_PIN_SET);
    sprintf(buf, "%u ms\r\n", HAL_GetTick() - last_time);
    HAL_UART_Transmit_DMA(&huart1, (uint8_t*)buf, strlen(buf));
}
```

• 结果



扩展内容

用引脚中断来处理按钮和测量时间；

- 配置PA6引脚为External interrupt mode with rising / falling edge trigger detection

PA6	n/a	n/a	External I...	No pull-u...	n/a	BTN1	✓
PA7	n/a	n/a	Input mo...	Pull-up	n/a	BTN	✓
PB1	n/a	Low	Output P...	No pull-u...	Low	Out	✓
PC13-T...	n/a	Low	Output P...	No pull-u...	Low	LED	✓

PA6 Configuration :

GPIO mode

External Interrupt Mode with Rising/Falling edge trigger detec

GPIO Pull-up/Pull-down

No pull-up and no pull-down

User Label

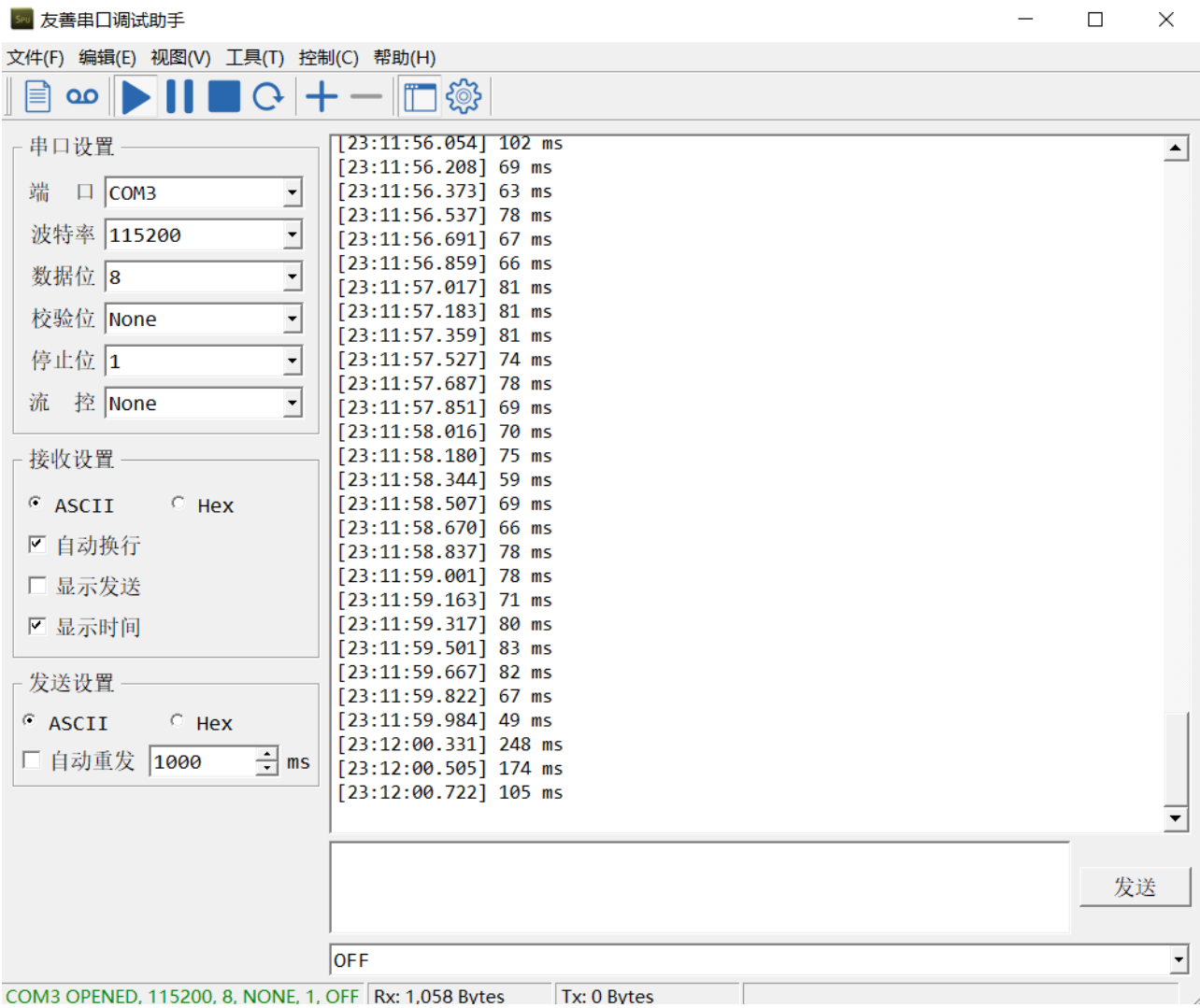
BTN1

- 代码

```
/* stm32f1xx_it.c */
/* 外部引脚中断处理函数 */
void EXTI9_5_IRQHandler(void)
{
    /* USER CODE BEGIN EXTI9_5_IRQn 0 */
    Key_Scan(); // 扫描按钮
    /* USER CODE END EXTI9_5_IRQn 0 */
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_6);
    /* USER CODE BEGIN EXTI9_5_IRQn 1 */

    /* USER CODE END EXTI9_5_IRQn 1 */
}
```

• 结果

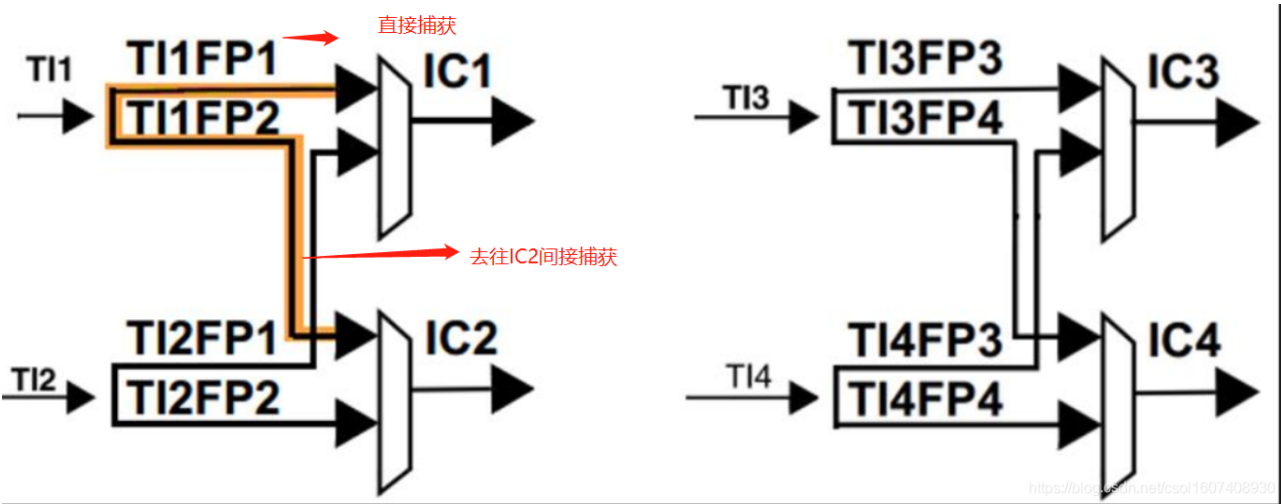


试探能够产生和测量的最小脉冲宽度(以下脉冲宽度均代指高电平脉冲宽度)。

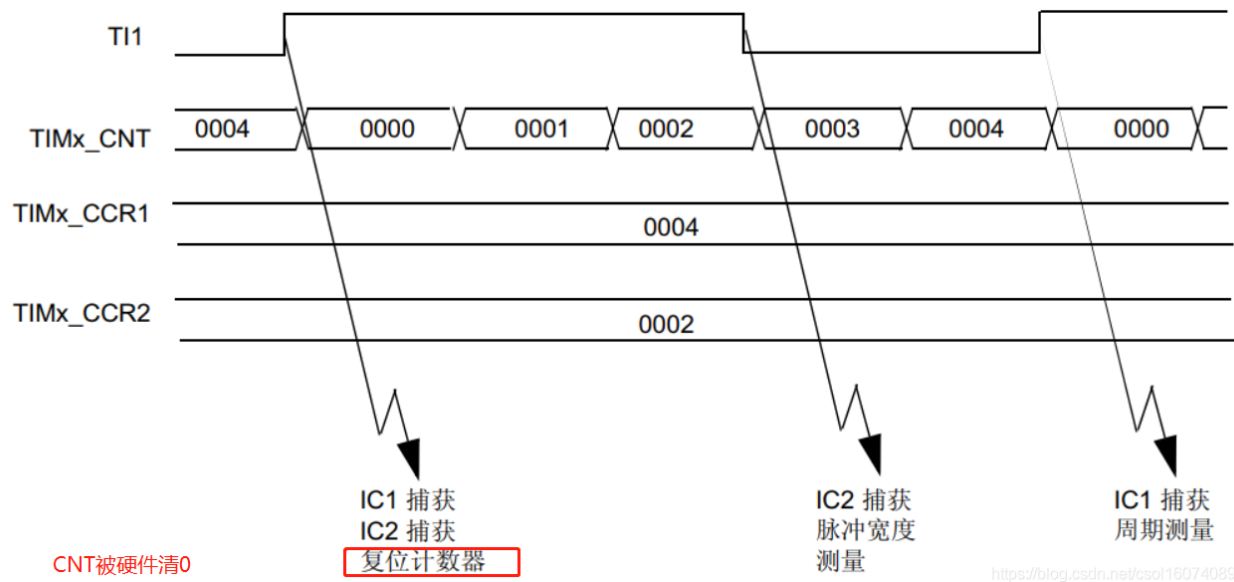
- 定时器输出的PWM波形能够达到代码输出波形更高的频率，获得更小的脉冲宽度。

定时器输出PWM波形以及捕捉PWM波形原理

- 触发源只能是TI1FP1或TI2FP2，即触发输入通道的中间信号TlxFPx，而TlxFPx去往ICx直接捕获,TlxFP(x+1) 去往IC(x+1)间接捕获。



- 下图假设TI1FP1是触发源：



- 触发源的直接捕获通道，要捕获什么边沿，可以由用户设置(上图举例上升沿)。当直接捕获通道捕获到目标边沿时，CNT会自动硬件清0，直接捕获通道捕获周期，间接捕获通道捕获脉宽。
- 同理，若触发源的直接捕获通道捕捉下降沿，那么间接捕获通道捕获的就是脉宽-，脉宽+就是周期减去脉宽-。

输出最小脉冲宽度

- TIM2作为输出PWM波形的定时器，CubeMX配置如下：

TIM2 Mode and Configuration

Mode

Slave Mode

Disable

Trigger Source

Disable

Clock Source

Internal Clock

Channel1

PWM Generation CH1

Channel2

Disable

Channel3

Disable

Channel4

Disable

Combined Channels

Disable

Configuration

Reset 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)

2

Counter Mode

Up

Counter Period (AutoReload Reg..

1

Internal Clock Division (CKD)

No Division

Configure the below parameters :

Search (Ctrl+F)

PWM Generation Channel 1

Mode

PWM mode 1

Pulse (16 bits value)

1

Output compare preload

Enable

Fast Mode

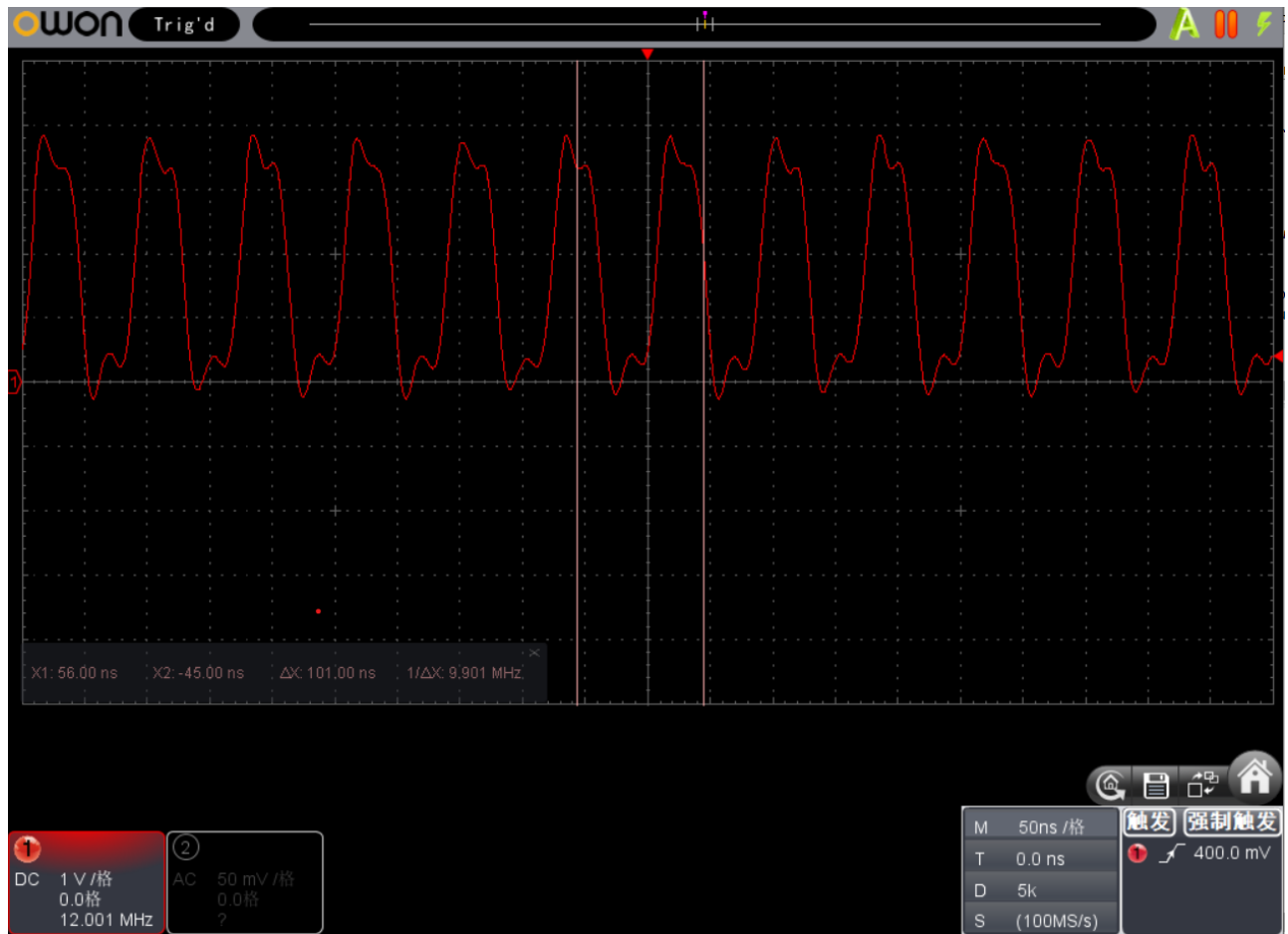
Disable

- 其中，Prescaler为2，ARR为1，计算可得定时器周期大约为83ns。Pulse为1，即占空比(脉冲宽度)为大概40ns。理论可知频率大约 $1/(80 * 10^{-9}) = 12.048MHz$ 。
- 代码

15 / 22


```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

- 结果



- 示波器测得的频率大概12.001MHz，与理论的相对误差为0.4%，误差小到基本可以忽略，所以可以认为脉冲宽度也基本准确。所以认为脉冲宽度最小约为40ns。(Prescaler为1的化出现的波形过于趋近正弦波)

接收最小脉冲宽度

- TIM1作为接收PWM波形的定时器，CubeMX配置如下：

TIM1 Mode and Configuration

Mode

Slave Mode

Disable

Trigger Source

Disable

Clock Source

Internal Clock

Channel1

Disable

Channel2

Disable

Channel3

Disable

Channel4

Disable

Combined Channels

PWM Input on CH1

Configuration

Reset 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)

71

Counter Mode

Up

Counter Period (AutoReload Reg..

10

Internal Clock Division (CKD)

No Division

▼ PWM Input CH1

Input Trigger

Slave Mode Controller

_____ Parameters for Chann.

Polarity Selection

IC Selection

Prescaler Division Ratio

Input Filter (4 bits value)

_____ Parameters for Chann.

Polarity Selection (opposite CH1)

IC Selection

Prescaler Division Ratio

Input Filter (4 bits value)

TI1FP1

Reset Mode

Rising Edge

Direct

No division

0

Falling Edge

Indirect

No division

0

Reset Configuration

✔ NVIC Settings	✔ DMA Settings	✔ GPIO Settings	
✔ Parameter Settings	✔ User Constants		
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM1 break interrupt	<input type="checkbox"/>	0	0
TIM1 update interrupt	<input type="checkbox"/>	0	0
TIM1 trigger and commutation interrupts	<input type="checkbox"/>	0	0
TIM1 capture compare interrupt	<input checked="" type="checkbox"/>	1	0

◦ 接收部分代码如下：

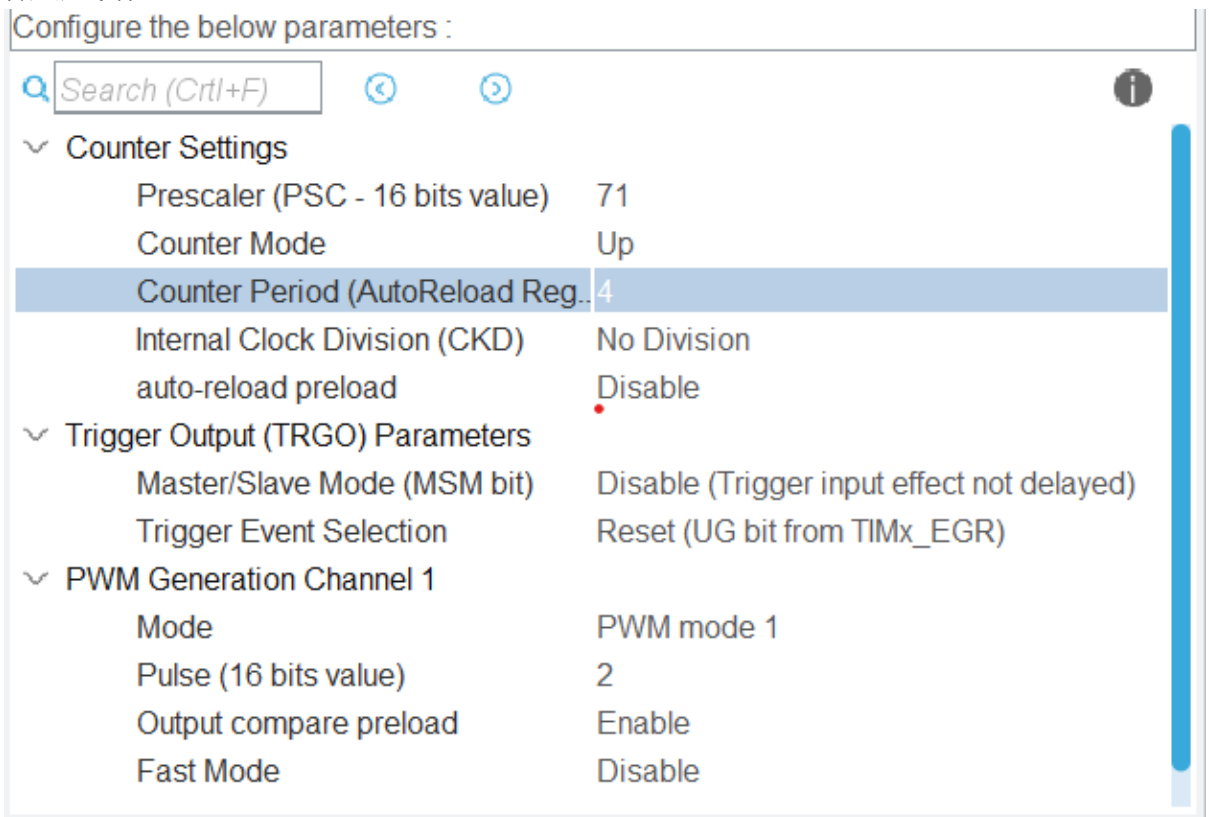
```
/* main.c */
uint32_t pwm_period=0;           //脉冲宽度
uint32_t IC2_VAL;                //记录寄存器CCR2的值
HAL_TIM_IC_Start_IT(&tim1,TIM_CHANNEL_2); // 打开CH2通道。用来读取脉冲宽度
if(IC2_VAL!=0)
{
    char buf[64];
    sprintf(buf, "%lu us\r\n", pwm_period);
    HAL_UART_Transmit_DMA(&huart1, (uint8_t *)buf, strlen(buf));
    HAL_Delay(100);
}
```

```

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Channel==HAL_TIM_ACTIVE_CHANNEL_2)
    {
        IC2_VAL=HAL_TIM_ReadCapturedValue(htim,TIM_CHANNEL_2);
        if(IC2_VAL!=0)
        {
            pwm_period=(IC2_VAL+1);
        }
    }
}

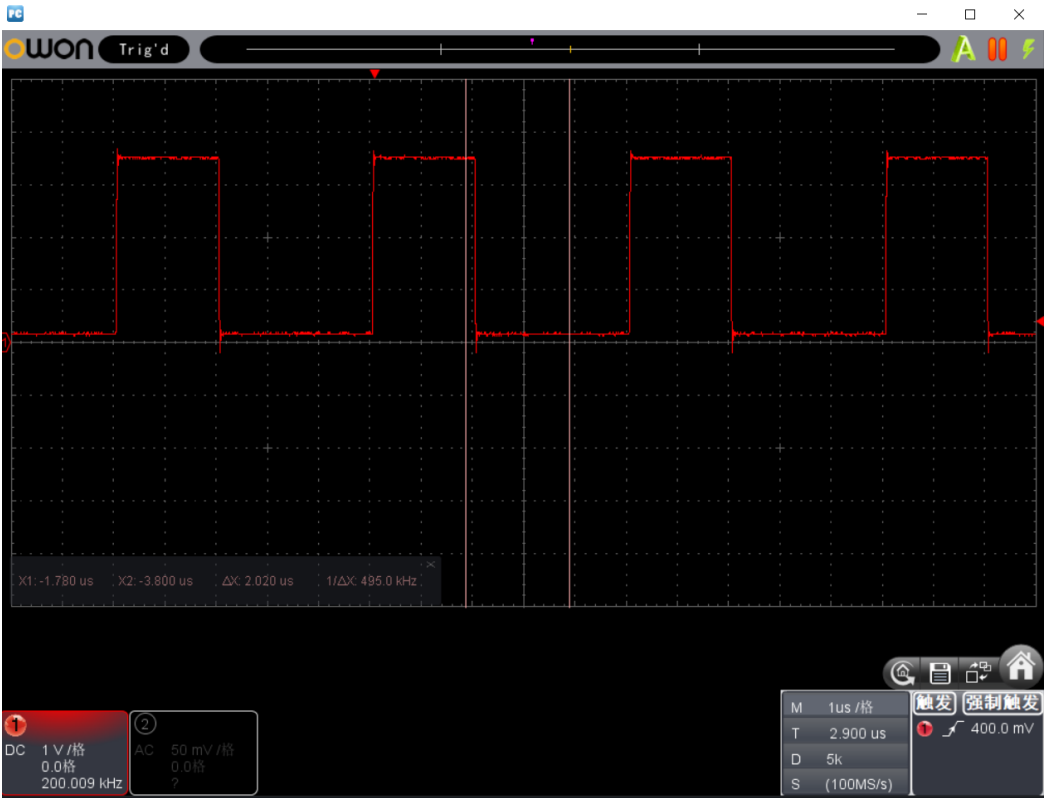
```

- TIM2作为输出PWM波形的定时器，经过配置测试发现接收脉冲宽度最小值为 $2\mu s$ 。
 - 由于脉冲宽度其实是可调制的，所以在不同的频率上一样能调制出相同宽度的脉冲。所以对TIM2做出如下配置：

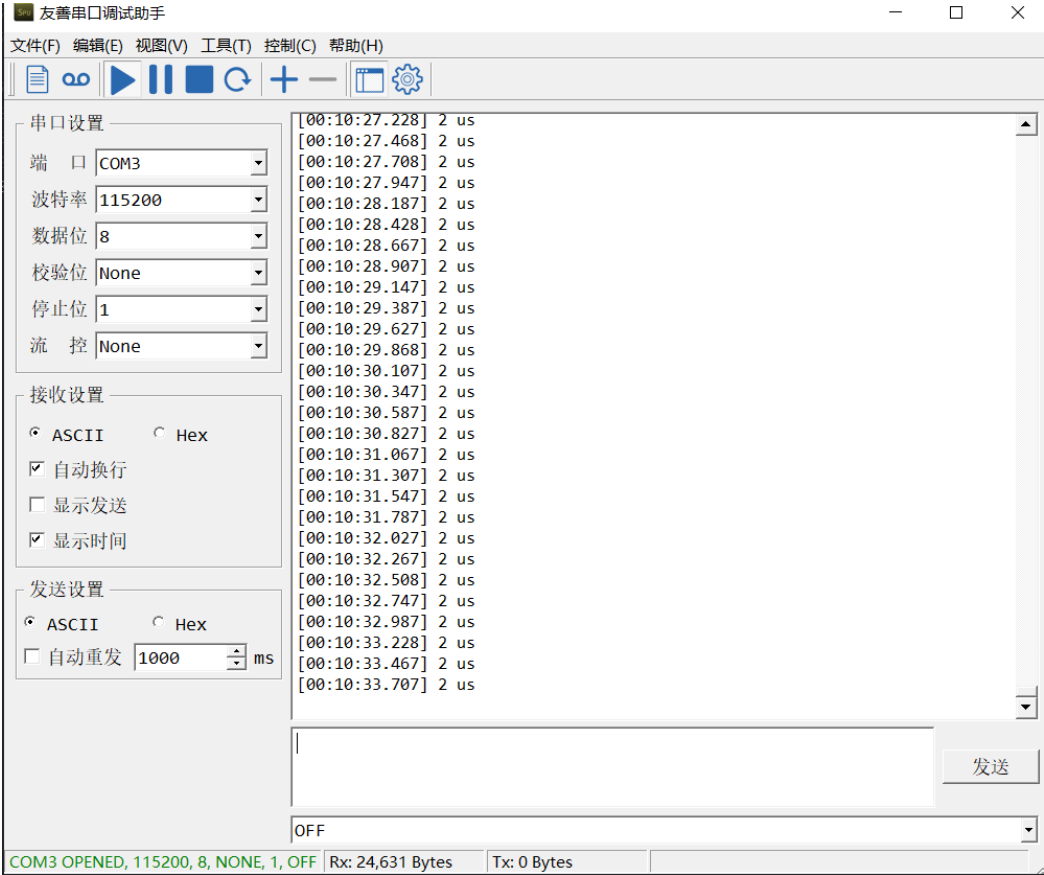


- Prescaler为71，即计数器跳1则过 $1\mu s$ 。ARR为4，表示总周期为 $5\mu s$ 。Pulse为2，表示脉冲宽度为 $2\mu s$
- 结果如图：

示波器






串口通信



- 测试设置为 $1\mu s$ 脉宽，配置如下：

Configure the below parameters :

- Counter Settings

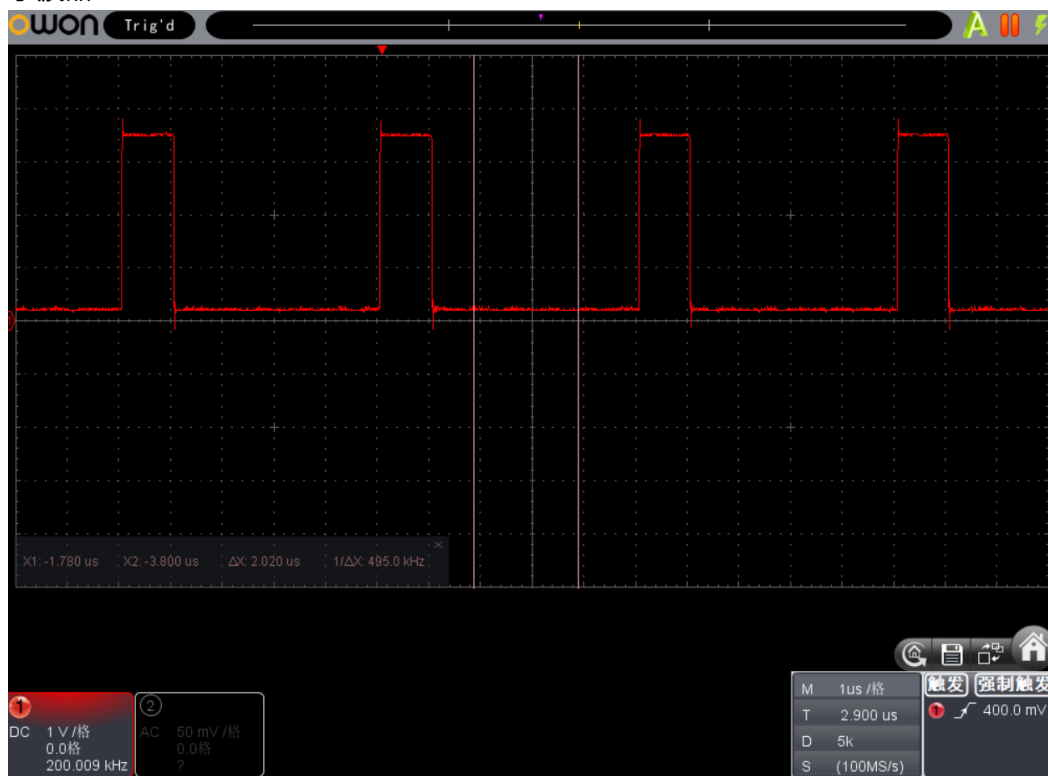
Prescaler (PSC - 16 bits value)	71
Counter Mode	Up
Counter Period (AutoReload Reg.. 4	
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
- Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
- PWM Generation Channel 1

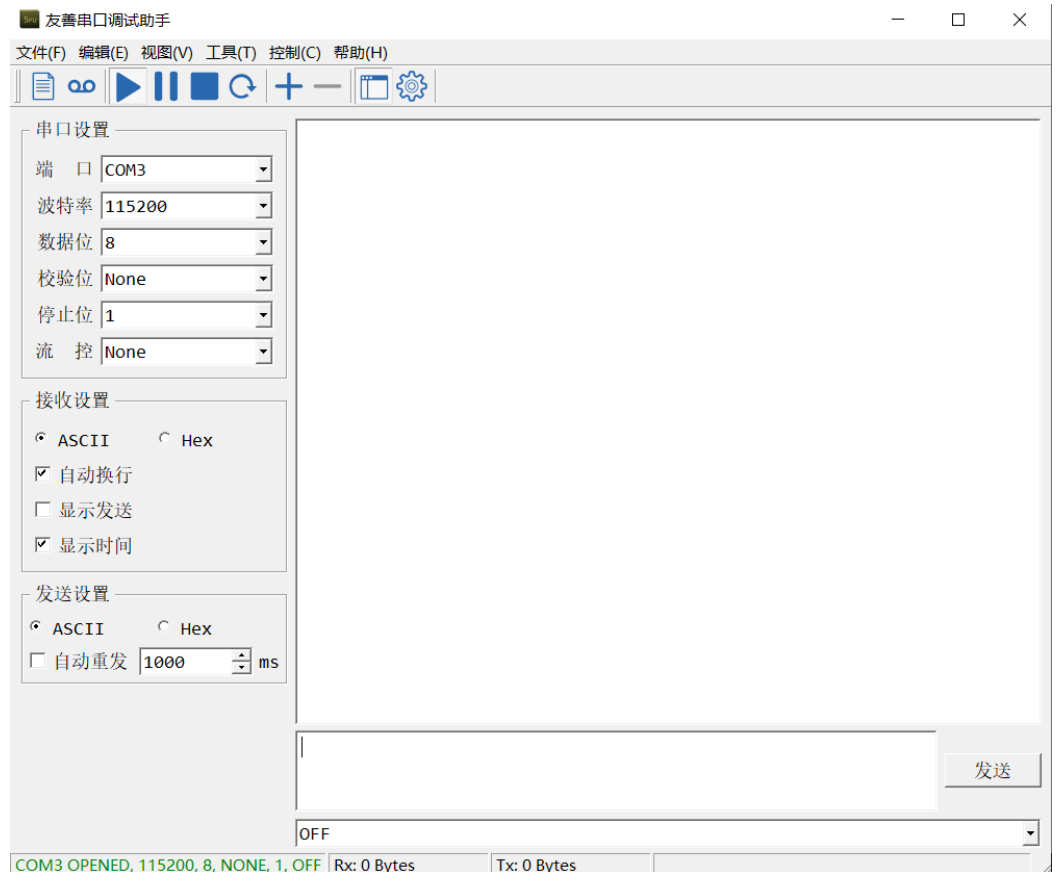
Mode	PWM mode 1
Pulse (16 bits value)	1
Output compare preload	Enable
Fast Mode	Disable

- 结果如图

- 示波器



■ 串口通信



- 示波器能够接收显示波形，但是串口通信却没有结果。

- 综上推断：能够接收的最小脉冲宽度约为 $1\mu s$

实验心得

- 本次实验还是很有难度的，主要是没有实验指南。那么许多内容都需要自己查找资料自学。遇到的困难有以下几个：
 - 计时在一开始不知道使用HAL_GetTick(), 只能使用RCC时钟。
 - PWM波的输入不知道该如何测量，资料多为使用定时器，但是并没有解释各个参数的含义，经过对不同资料对比和自己尝试，总算搞懂了部分。
 - 上次写的去抖动部分有bug，即无法测量按下的时间。又自己重写了该部分，并且搞懂了参考资料中使用SysTick的原因。
- 虽然本次实验花费了不少的时间，但是感觉收获颇丰。对定时器的学习有助于大程的气象模块唤醒机制的设计。对脉冲的读取有助于大程的气象模块对各种气象数据的读取。