lab9 FreeRTOS温度计

实验目的

- 掌握FreeRTOS的应用程序编写；
- 理解如何直接操纵GPIO；
- 掌握单总线设备的访问方式；
- 掌握7段数码管的时分复用驱动方式。

实验器材

硬件

- STM32F103核心板1块；
- ST-Link板1块；
- USB串口板1块；
- 面包板1块；
- 四位7段数码管（共阳）1颗；
- 360Ω 1/8W电阻2颗；
- DHT-11 温湿度传感器1个；
- 杜邦线、面包线若干。

软件

- PC上的ST-Link的驱动程序；

- PC上的USB-TTL串口线配套的驱动程序；
- PC上的串口终端软件Serial Port Utility。

实验原理

GPIO寄存器

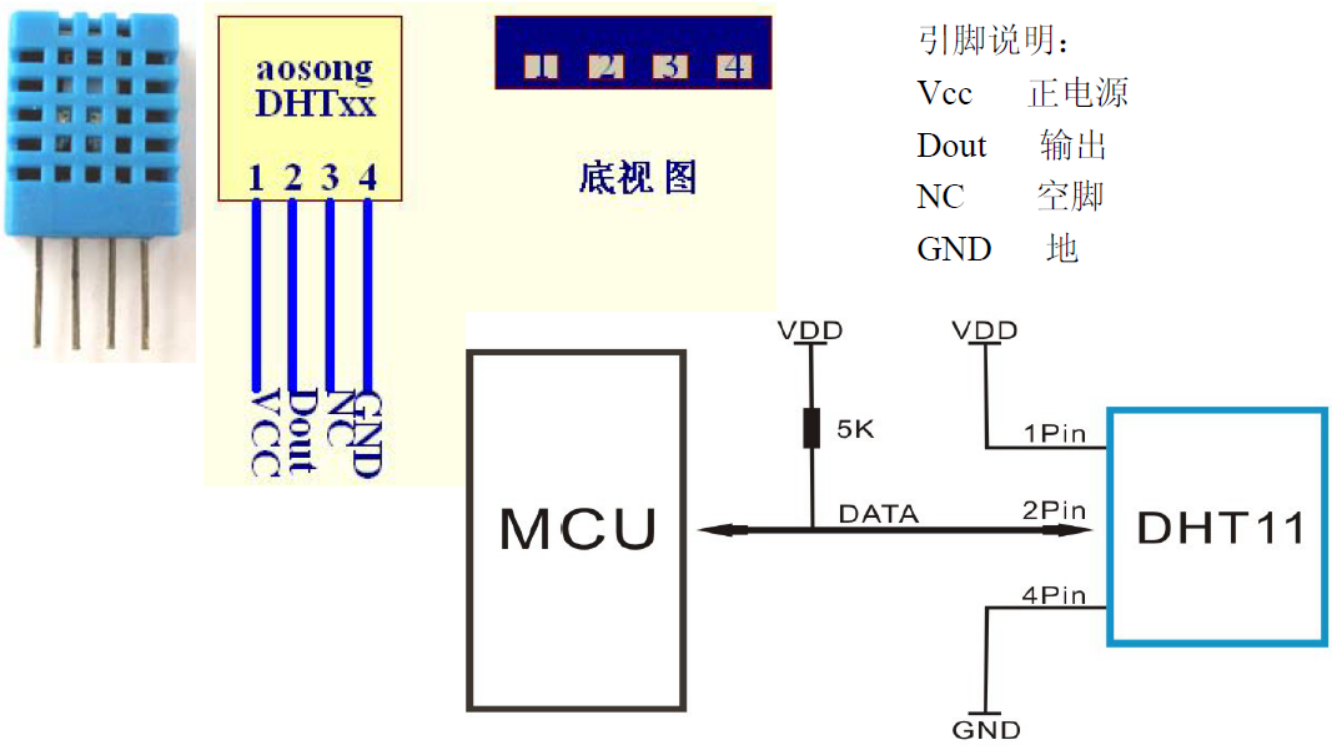
CRH和CRL

- CRH和CRL是GPIO的配置寄存器，CRH和CRL寄存器的每4个BIT用来配置一个IO的功能，CRL用来配置低8位IO的功能，CRH用来配置高8位IO的功能。
- IO的不同模式：
    - 输入浮空
    - 输入下拉
    - 输入上拉
    - 模拟输入
    - 开漏输出
    - 推免式输出
    - 推免式复用功能
    - 开漏复用功能

其他寄存器

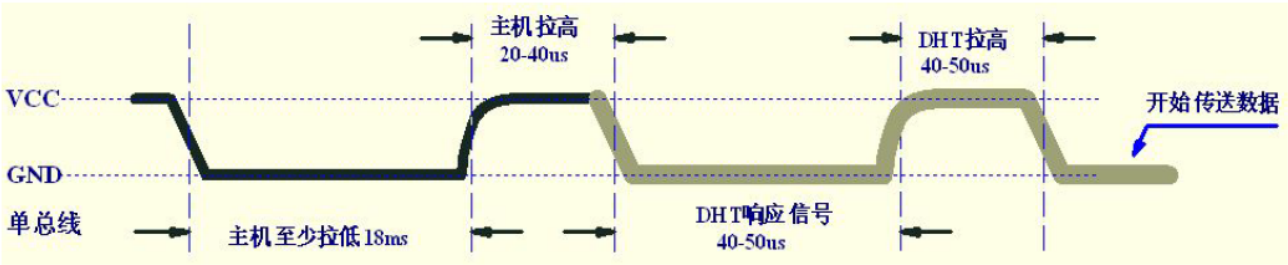
- IDR是GPIO的输入数据寄存器。通过IDR寄存器可以读出IO的状态。需要注意的是IDR寄存器只能以字（16位）的形式读出。
- ODR是GPIO的输出数据寄存器。通过ODR寄存器可以输出高低电平。
- BSRR是GPIO的端口位设置/清除寄存器。BSRR寄存器的高16位是清除IO位，低16位是置位IO位。需要注意的是BSRR只有在写入1的时候，相应的功能才有作用，写入0是无效的。
- BRR是GPIO的端口位清除寄存器。BRR基础只有在写入1的时候，相应的功能才有作用，写入0无效。
- LCKR是GPIO的端口配置锁定寄存器。LCKR寄存器用来锁定IO口的配置，设置后，除了复位后不能再配置IO的状态。

DHT11



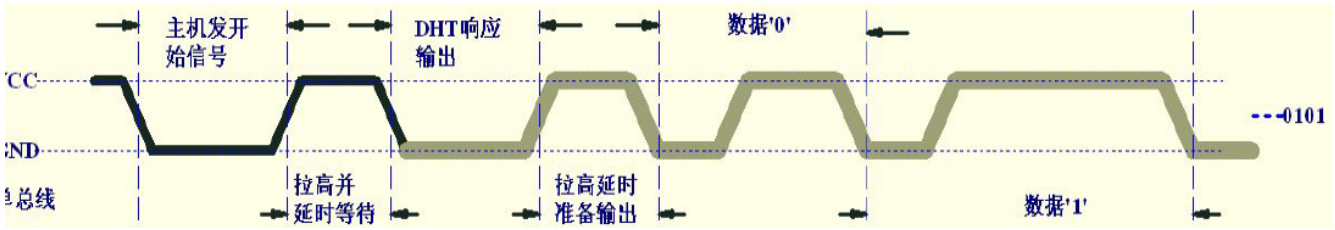
通信

- 主机发开始信号
  主机等待接收DHTxx响应信号
  主机连续接收40Bit的数据和校验和 数据



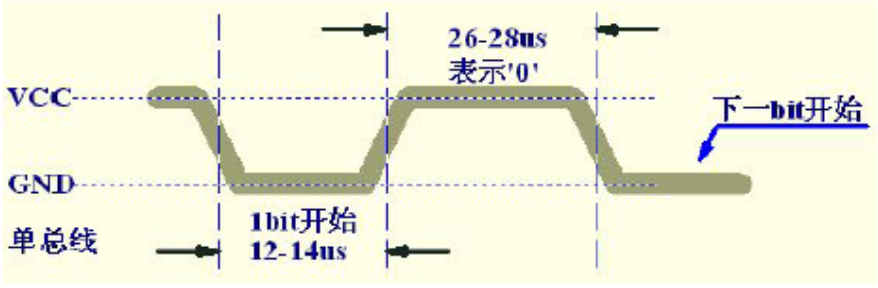
通信2

- DHT发送数据

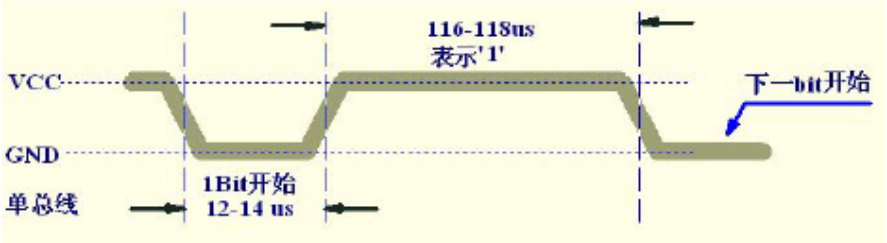


通信3

- 数据0



- 数据1



数据格式

byte4	byte3	byte2	byte1	byte0
00101101	00000000	00011100	00000000	01001001
整数	小数	整数	小数	校验和
湿度		温度		校验和

四位七段数码管

7段数码管

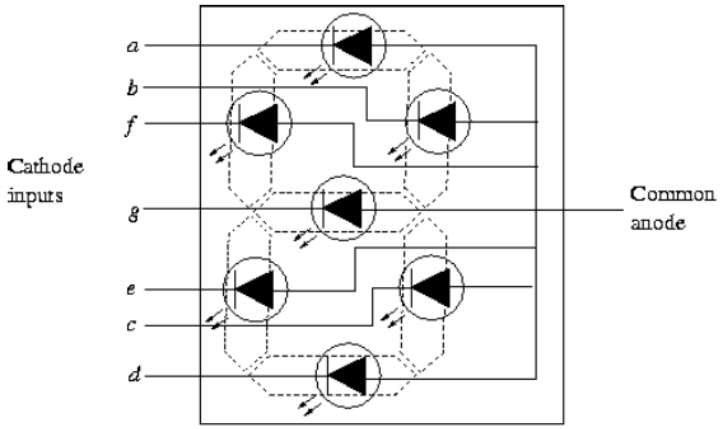


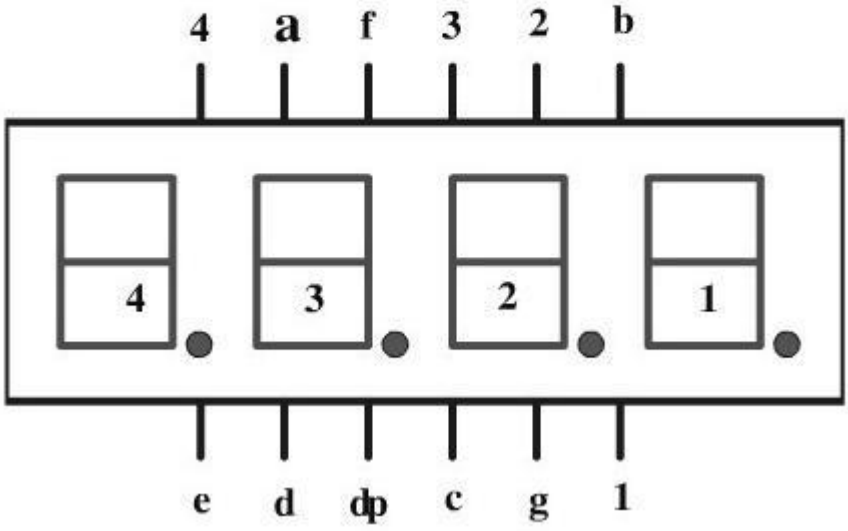
FIGURE 5 : Wiring of 7-segment LED display

DIGIT	SEGMENTS ACTIVATED
0	*a, b, c, d, e, f*
1	*b, c*
2	*a, b, d, e, g*
3	*a, b, c, d, g*
4	*b, c, f, g*
5	*a, c, d, f, g*
6	*a, c, d, e, f, g*
7	*a, b, c*
8	*a, b, c, d, e, f, g*
9	*a, b, c, d, f, g*

Table 2 : Active segments for each decimal digit.

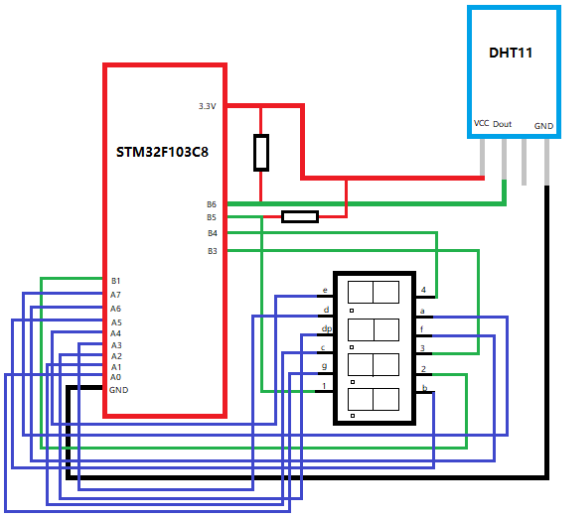
四位一体数码管的引脚图，共阳型

1、2、3、4 是从右边数起的

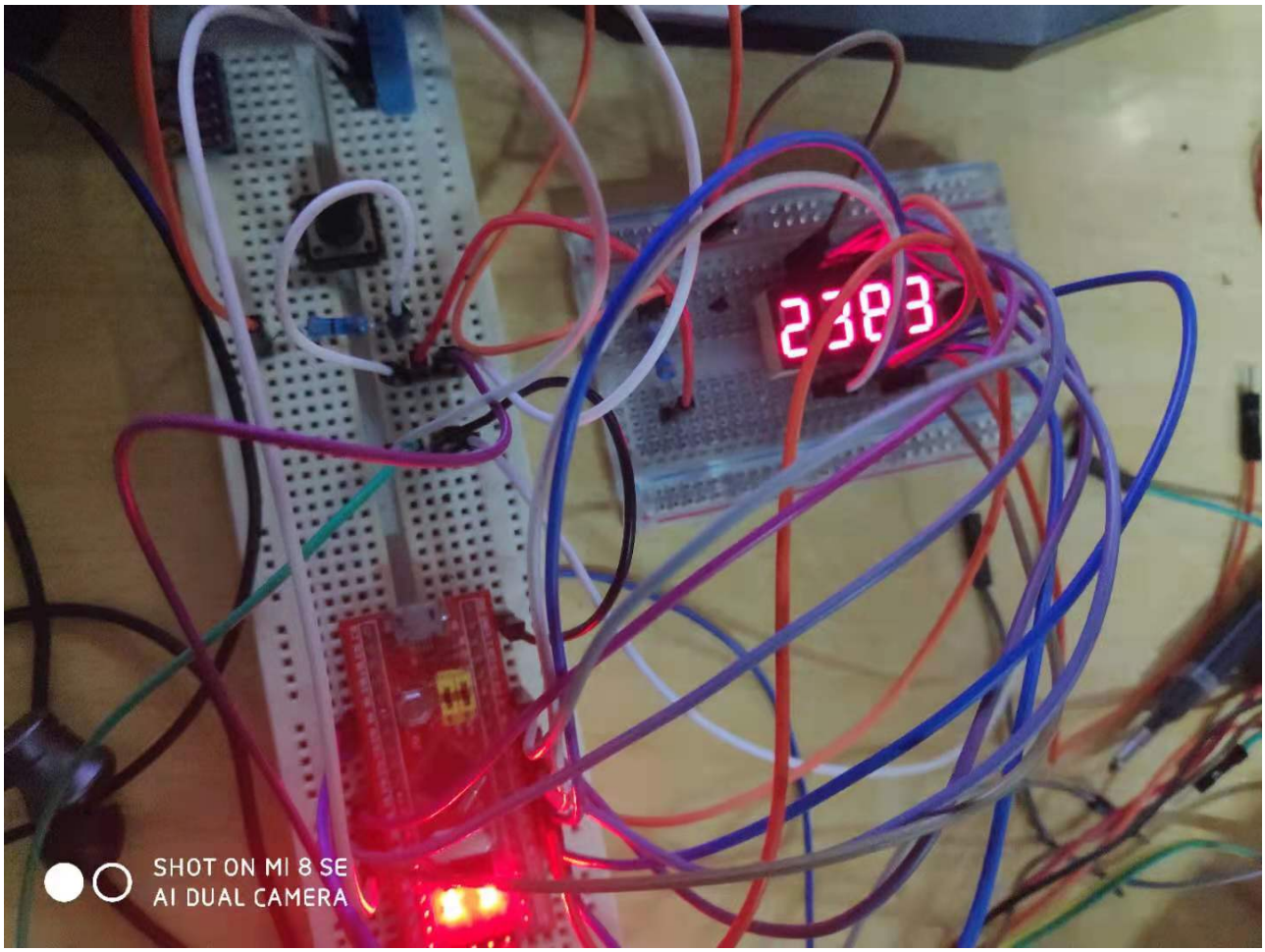


实验步骤

设计原理图



连线实物图

- 

测试、实现FreeRTOS对GPIO的访问；

- 修改上次实验的代码，通过GPIO寄存器来控制LED的亮灭。

- 代码

```
LED_GPIO_Port->ODR = 0x2000; /* 点亮 */
osDelay(500);
LED_GPIO_Port->ODR = 0x0000; /* 熄灭 */
osDelay(500);
```

实现DHT-11数据的读;

- 代码

```
/* 新建dht11.c和dht11.h来实现读取DHT11温湿度计的数据
 * 下面主要展示dht11.c的代码
 */
/* 自定义的微秒级计时器 */
void delay_us(uint16_t us){
    uint16_t differ = 0xffff-us-5;
    __HAL_TIM_SET_COUNTER(&htim3,differ);    //设定TIM3计数器起始值
    HAL_TIM_Base_Start(&htim3);     //启动定时器

    while(differ < 0xffff-5){   //判断
        differ = __HAL_TIM_GET_COUNTER(&htim3);     //查询计数器的计数值
    }
    HAL_TIM_Base_Stop(&htim3);
}
/* 将引脚PB6的GPIO状态转变为GPIO_Input */
void GPIO_Input(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* Configure GPIO pin : PB6 */
    GPIO_InitStruct.Pin = IODHT11_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    HAL_GPIO_Init(IODHT11_GPIO_Port, &GPIO_InitStruct);
}
/* 将引脚PB6的GPIO状态转变为GPIO_Output */
void GPIO_Output(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /*Configure GPIO pin : PB6 */
    GPIO_InitStruct.Pin = IODHT11_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(IODHT11_GPIO_Port, &GPIO_InitStruct);
}
/* 主机开始发送信号给DHT11 */
void DHT11_Rst(void)
{
    GPIO_Output();  // 将引脚状态设置为GPIO_Output，以便让主机发送信号
```

```
    HAL_GPIO_WritePin(IODHT11_GPIO_Port, IODHT11_Pin, GPIO_PIN_RESET);  // 主机拉低
    HAL_Delay(20);
    HAL_GPIO_WritePin(IODHT11_GPIO_Port, IODHT11_Pin, GPIO_PIN_SET);    // 主机拉高
    delay_us(30);
}
/* 主机接收DHT11回复的信号 */
uint8_t DHT11_Check(void)
{
    uint8_t retry = 0;
    GPIO_Input();   // 将引脚状态设置为GPIO_Input，以便主机接收信号
    while (HAL_GPIO_ReadPin(IODHT11_GPIO_Port, IODHT11_Pin) && retry < 100) //等待回应拉位低电平
    {
        retry++;
        delay_us(1);
    }
    //当变量值大于100 返回1 说明无响应  返回 0 则为正确响应
    if (retry >= 100)
        return 1;
    else
        retry = 0;
    while (!HAL_GPIO_ReadPin(IODHT11_GPIO_Port, IODHT11_Pin) && retry < 100)    //等待变为高电平
    {
        retry++;
        delay_us(1);
    }
    if (retry >= 100)
        return 1;
    return 0;
}
/* 建立通信初始化 */
uint8_t DHT11_Init(void)
{
    uint8_t ret = 1;
    DHT11_Rst();  //复位DHT11
    ret = DHT11_Check();
    uint8_t buffer[100];
    sprintf(buffer, "dht11 init %s. \r\n", ret == 0 ? "ok" : "failed");
    uart_transmit_str(buffer);
    return ret;
}
/* 读取一个位 */
uint8_t DHT11_ReadBit(void)
{
    uint8_t retry = 0;
    while (HAL_GPIO_ReadPin(IODHT11_GPIO_Port, IODHT11_Pin) && retry < 100) //等待变为低电平
    {
        retry++;
        delay_us(1);
    }
```

```
    retry = 0;
    while (!HAL_GPIO_ReadPin(IODHT11_GPIO_Port, IODHT11_Pin) && retry < 100) //等
待变为高电平
    {
        retry++;
        delay_us(1);
    }
    delay_us(40); //40us 后如果为低电平 数据为0    高电平数据为1
    if (HAL_GPIO_ReadPin(IODHT11_GPIO_Port, IODHT11_Pin))
        return 1;
    else
        return 0;
}
/* 读取一个字节  返回值位采集值 */
uint8_t DHT11_ReadByte(void)
{
    uint8_t i, dat;
    dat = 0;
    for (i = 0; i < 8; i++)
    {
        dat <<= 1;                //数据左移一位
        dat |= DHT11_ReadBit(); //每读取到一个位 放到dat的最后一位
    }
    return dat;
}
/* 读取数据，40个位 */
uint8_t DHT11_ReadData(uint8_t *h)
{
    uint8_t buf[5];
    uint8_t i;
    DHT11_Rst();
    if (DHT11_Check() == 0)
    {
        for (i = 0; i < 5; i++)
        {
            buf[i] = DHT11_ReadByte();
        }
        if (buf[0] + buf[1] + buf[2] + buf[3] == buf[4])    // 数据校验
        {
            *h = buf[0];
            h++;
            *h = buf[2];
        }
    }
    else
        return 1;
    return 0;
}
/* 创建一个FreeRtos的Task，用来读取数据，代码如下 */
void StartTask02(void const *argument)
{
  /* USER CODE BEGIN StartTask02 */
  /* Infinite loop */
  uint8_t buffer[100];
```

```
  for (;;)
  {
    while(DHT11_Init());
    DHT11_ReadData(temp);
    sprintf(buffer, "temp: %d C, humi: %d %% \r\n", temp[1], temp[0]);
    uart_transmit_str(buffer);
  }
```

- 结果

```
[19:13:33.935] temp: 23 C, humi: 75 %
[19:13:33.952] temp: 23 C, humi: 75 %
[19:13:34.948] temp: 23 C, humi: 75 %
[19:13:35.951] temp: 23 C, humi: 75 %
[19:13:36.948] temp: 23 C, humi: 75 %
[19:13:37.951] temp: 23 C, humi: 75 %
[19:13:38.949] temp: 23 C, humi: 74 %
[19:13:39.951] temp: 23 C, humi: 74 %
[19:13:40.948] temp: 23 C, humi: 74 %
[19:13:41.951] temp: 23 C, humi: 74 %
[19:13:42.947] temp: 23 C, humi: 74 %
[19:13:43.951] temp: 23 C, humi: 74 %
[19:13:44.947] temp: 23 C, humi: 74 %
[19:13:45.950] temp: 23 C, humi: 74 %
[19:13:46.947] temp: 23 C, humi: 74 %
[19:13:47.952] temp: 23 C, humi: 74 %
[19:13:48.947] temp: 23 C, humi: 74 %
[19:13:49.950] temp: 23 C, humi: 74 %
[19:13:50.947] temp: 23 C, humi: 74 %
```

实现以时分复用方式在四位7段数码管上依次显示0000-9999的数字；

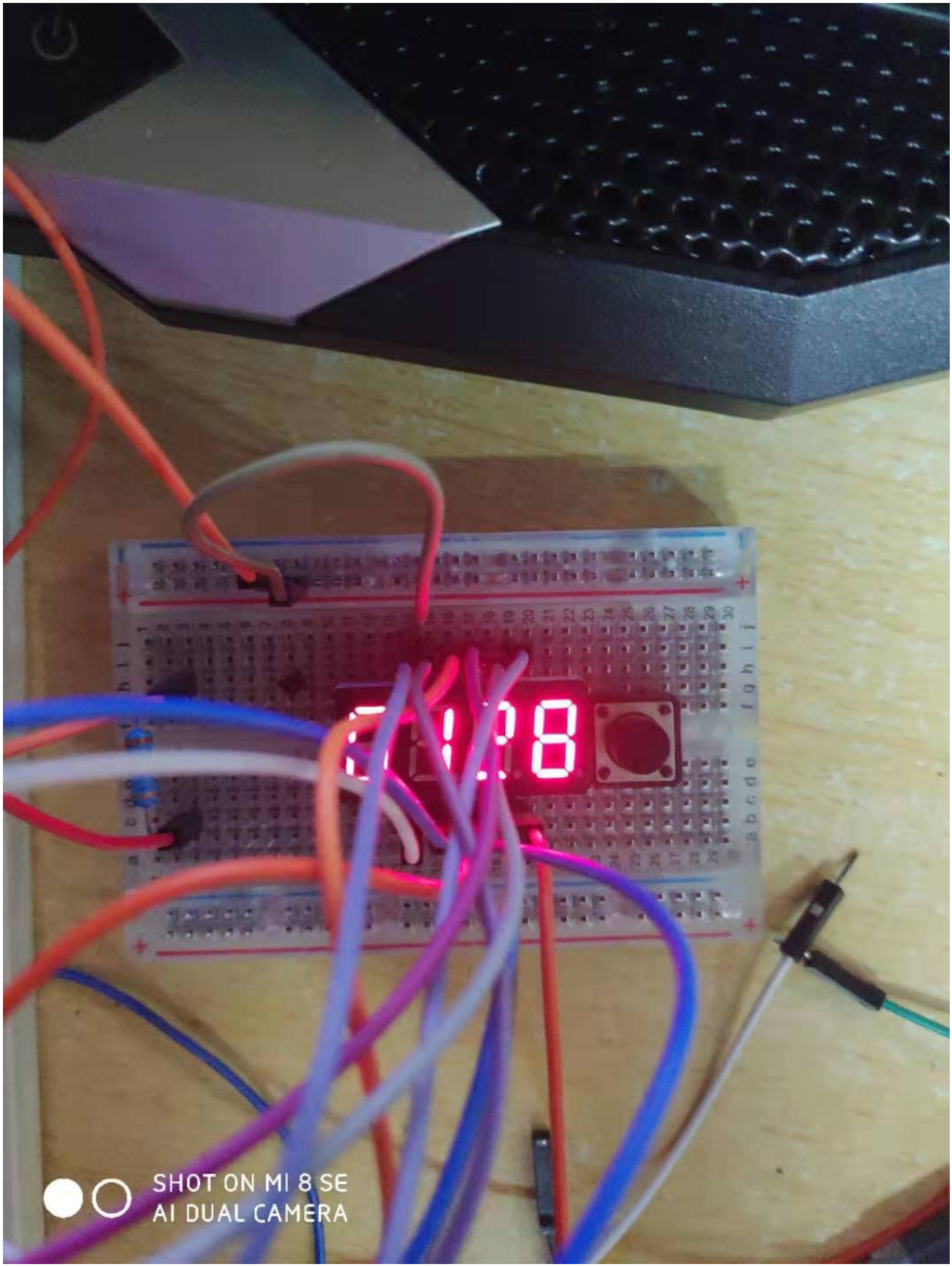
```
/* 新建led_seg.c和led_seg.h来实现四位七段数码管的显示
 * 下面主要展示led_seg.c的代码
 */
/* 显示初始化，将所有段引脚电平拉高，让七段数码管全部熄灭 */
void display_init()
{
    HAL_GPIO_WritePin(LED_A_GPIO_Port, LED_A_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_B_GPIO_Port, LED_B_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_C_GPIO_Port, LED_C_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_D_GPIO_Port, LED_D_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_E_GPIO_Port, LED_E_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_F_GPIO_Port, LED_F_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_DP_GPIO_Port, LED_DP_Pin, GPIO_PIN_SET);
}
/* 时分复用选择四位数字哪个位数字进行显示 */
void pick_digit(uint8_t pick)
{
    // 四个公共引脚电平拉低，全部熄灭
    HAL_GPIO_WritePin(C1_GPIO_Port, C1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(C2_GPIO_Port, C2_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(C3_GPIO_Port, C3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(C4_GPIO_Port, C4_Pin, GPIO_PIN_RESET);
```

```
    // 决定将哪个公共引脚点平拉高来显示对应位的数字
    switch (pick)
    {
    case 1:
        HAL_GPIO_WritePin(C1_GPIO_Port, C1_Pin, GPIO_PIN_SET);
        break;
    case 2:
        HAL_GPIO_WritePin(C2_GPIO_Port, C2_Pin, GPIO_PIN_SET);
        break;
    case 3:
        HAL_GPIO_WritePin(C3_GPIO_Port, C3_Pin, GPIO_PIN_SET);
        break;
    case 4:
        HAL_GPIO_WritePin(C4_GPIO_Port, C4_Pin, GPIO_PIN_SET);
        break;
    default:
        HAL_GPIO_WritePin(C1_GPIO_Port, C1_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(C2_GPIO_Port, C2_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(C3_GPIO_Port, C3_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(C4_GPIO_Port, C4_Pin, GPIO_PIN_RESET);
        break;
    }
}
/* 显示数字 */
void display_number(uint8_t number)
{
    switch (number)
    {
    case 1:
        display_one();
        break;
    case 2:
        display_two();
        break;
    case 3:
        display_three();
        break;
    case 4:
        display_four();
        break;
    case 5:
        display_five();
        break;
    case 6:
        display_six();
        break;
    case 7:
        display_seven();
        break;
    case 8:
        display_eight();
        break;
    case 9:
        display_nine();
```

```
            break;
        case 0:
            display_zero();
            break;
        }
}
/* 数字显示也采用时分复用的方式，即在同一时间只有一个数码管点亮
 * 由于代码长度过长，且0-9数字显示原理相同，所以下面只展示数字8的显
 * 示函数。
 */
void display_eight()
{
    display_init();
    HAL_GPIO_WritePin(LED_A_GPIO_Port, LED_A_Pin, GPIO_PIN_RESET);
    osDelay(1);
    HAL_GPIO_WritePin(LED_A_GPIO_Port, LED_A_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_B_GPIO_Port, LED_B_Pin, GPIO_PIN_RESET);
    osDelay(1);
    HAL_GPIO_WritePin(LED_B_GPIO_Port, LED_B_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_C_GPIO_Port, LED_C_Pin, GPIO_PIN_RESET);
    osDelay(1);
    HAL_GPIO_WritePin(LED_C_GPIO_Port, LED_C_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_D_GPIO_Port, LED_D_Pin, GPIO_PIN_RESET);
    osDelay(1);
    HAL_GPIO_WritePin(LED_D_GPIO_Port, LED_D_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_E_GPIO_Port, LED_E_Pin, GPIO_PIN_RESET);
    osDelay(1);
    HAL_GPIO_WritePin(LED_E_GPIO_Port, LED_E_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_F_GPIO_Port, LED_F_Pin, GPIO_PIN_RESET);
    osDelay(1);
    HAL_GPIO_WritePin(LED_F_GPIO_Port, LED_F_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_RESET);
    osDelay(1);
    HAL_GPIO_WritePin(LED_G_GPIO_Port, LED_G_Pin, GPIO_PIN_SET);
}
/* 创建一个FreeRtos的Task，用来显示0000-9999，代码如下 */
void StartTask03(void const *argument)
{
  /* USER CODE BEGIN StartTask03 */
  /* Infinite loop */
  //  uint16_t i;               //定义数字变量
  uint16_t temp_number, huni_number; //定义时间变量
  for (;;)
  {
    int i;                      //定义数字变量
    int time_cost, time;        //定义时间变量
    for (i = 0; i < 10000; i++) //循环遍历0000-9999，并输出
    {
        time = HAL_GetTick(); //确定当前时间
        //限制数字跳转速度，确保每个数字停留200ms
        for (time_cost = 0; time_cost < 200; time_cost = HAL_GetTick() - time)
        {
            //个位数
            pick_digit(1);
```

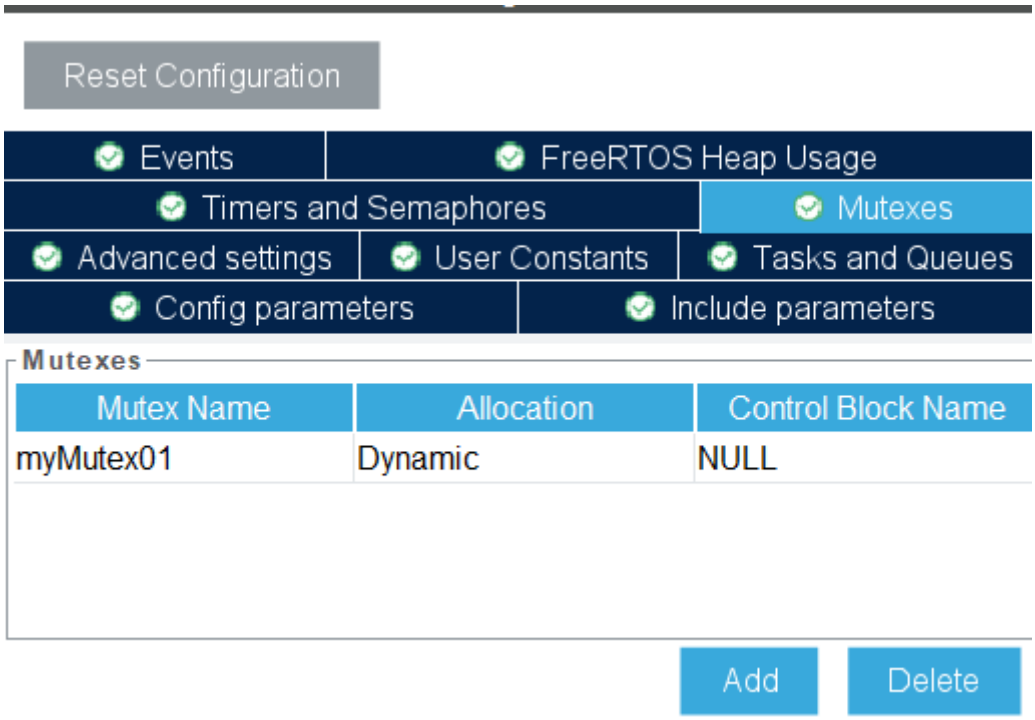
```
            display_number(i % 10);
            osDelay(1);
            // 十位数
            pick_digit(2);
            display_number((i / 10) % 10);
            osDelay(1);
            // 百位数
            pick_digit(3);
            display_number((i / 100) % 10);
            osDelay(1);
            // 千位数
            pick_digit(4);
            display_number((i / 1000) % 10);
            osDelay(1);
        }
    }
  }
  /* USER CODE END StartTask03 */
}
```

- 结果



用两个FreeRTOS任务，一个定时读DHT-11数据，一个轮流驱动数码管，一秒一次显示当前温度和湿度。注意处理好两个任务之间的数据共享。

- 设置定时器，

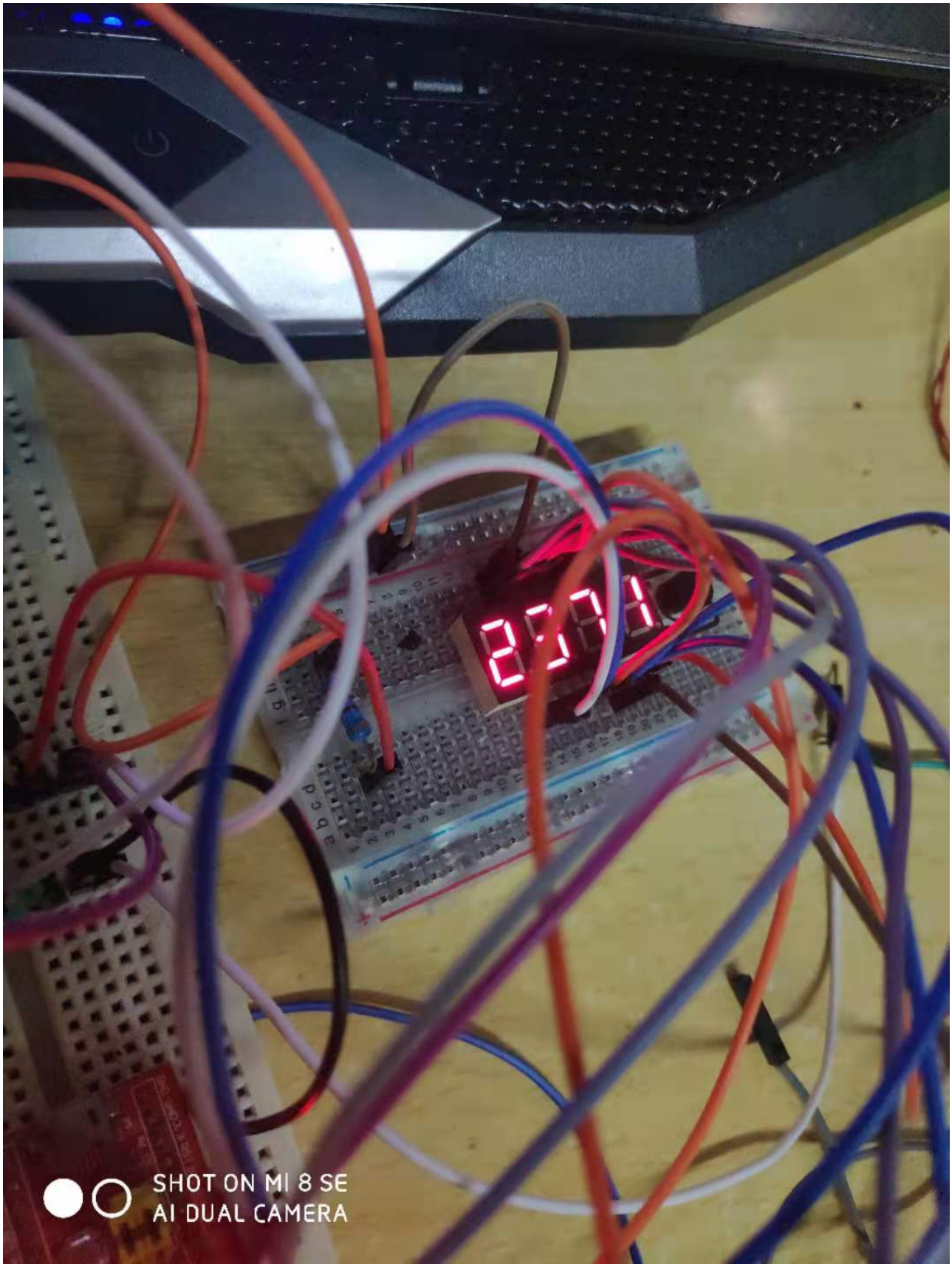


- 设置互斥锁
- 代码

```
/* 关于任务管理，要求做到一秒一次显示当前温度和湿度。所以必须协调好两个任务之间的关系
 * 协调任务的方法是：设置读取DHT11数据的任务拥有更高的优先级(其实更低优先级也可以，只要两个任务优先级不同即可)。
 * 因此开始运行时，会先从DHT11读取数据。
 * 该任务执行结束之前，选择挂起该任务，运行显示温度湿度的任务。这样能确保显示的温度和湿度准确。
 * 同时设置一个1s触发一次的定时器，触发后挂起显示温度湿度的任务，恢复运行读取DHT11数据的任务。
 */
/* 关于互斥锁，其实从理论上来说：两个任务——一个是修改共享变量的值，一个是读取共享变量的值，并不冲突。
 * 但考虑到题目特意提醒要注意两个任务之间的数据共享，
 * 且假设在运行显示数字的任务时，突然定时器中断触发，任务运行被切换，可能会导致数据不准确。
 * 因此加入互斥锁对共享变量进行保护。
 */
/* 全局变量记录温度和湿度 */
uint8_t temp[2];
/* MX_FREERTOS_Init(void) 函数中加入 */
osTimerStart(myTimer01Handle, 1000);    // 开启定时器
while(DHT11_Init());                    // DHT11通信初始化
/* 读取DHT11数据的任务 */
void StartTask02(void const *argument)
{
  /* USER CODE BEGIN StartTask02 */
  /* Infinite loop */
  uint8_t buffer[100];
  for (;;)
  {
    osMutexWait(myMutex01Handle, osWaitForever);    // 获得互斥锁
    DHT11_ReadData(temp);
```

```
    sprintf(buffer, "temp: %d C, humi: %d %% \r\n", temp[1], temp[0]);
    uart_transmit_str(buffer);
    osMutexRelease(myMutex01Handle);                // 释放互斥锁
    osThreadResume(Display_numberHandle);           // 恢复显示温度和湿度的任务
    osThreadSuspend(DHT11Handle);                   // 挂起读取DHT11数据的任务
  }
  /* USER CODE END StartTask02 */
}
/* 显示温度和湿度的任务 */
void StartTask03(void const *argument)
{
  /* USER CODE BEGIN StartTask03 */
  /* Infinite loop */
  //  uint16_t i;               //定义数字变量
  uint16_t temp_number, huni_number; //定义时间变量
  for (;;)
  {
    osMutexWait(myMutex01Handle, osWaitForever);    // 获得互斥锁
    temp_number = temp[1];
    huni_number = temp[0];
    osMutexRelease(myMutex01Handle);                // 释放互斥锁
    //个位
    pick_digit(1);
    display_number(huni_number % 10);
    osDelay(1);
    // 十位
    pick_digit(2);
    display_number((huni_number / 10) % 10);
    osDelay(1);
    // 百位
    pick_digit(3);
    display_number(temp_number % 10);
    osDelay(1);
    // 千位
    pick_digit(4);
    display_number((temp_number / 10) % 10);
    osDelay(1);
  }
  /* USER CODE END StartTask03 */
}
/* 定时器回调函数，1s触发一次 */
void Callback01(void const *argument)
{
  /* USER CODE BEGIN Callback01 */
  osThreadSuspend(Display_numberHandle);    // 挂起显示温度和湿度的任务
  pick_digit(0);
  osThreadResume(DHT11Handle);              // 恢复读取DHT11数据的任务
  /* USER CODE END Callback01 */
}
```

- 结果



```
[20:19:21.231] temp: 23 C, humi: 72 %
[20:19:22.233] temp: 23 C, humi: 72 %
[20:19:23.230] temp: 22 C, humi: 71 %
[20:19:24.234] temp: 22 C, humi: 71 %
[20:19:25.230] temp: 23 C, humi: 71 %
[20:19:26.233] temp: 23 C, humi: 71 %
[20:19:27.230] temp: 23 C, humi: 71 %
[20:19:28.233] temp: 23 C, humi: 71 %
[20:19:29.229] temp: 22 C, humi: 71 %
[20:19:30.233] temp: 22 C, humi: 71 %
[20:19:31.229] temp: 22 C, humi: 71 %
[20:19:32.233] temp: 22 C, humi: 71 %
[20:19:33.229] temp: 22 C, humi: 71 %
[20:19:34.233] temp: 22 C, humi: 71 %
[20:19:35.229] temp: 22 C, humi: 71 %
[20:19:36.232] temp: 22 C, humi: 71 %
[20:19:37.230] temp: 23 C, humi: 71 %
[20:19:38.233] temp: 23 C, humi: 71 %
[20:19:39.229] temp: 23 C, humi: 71 %
[20:19:40.233] temp: 23 C, humi: 71 %
```

实验心得

- 本次实验先对于Bootloader还是较为简单的。不过由于要做的内容比较多，所以略显复杂。幸得曾经在小学期的Arduino课程中曾经接触过DHT-11和四位七段数码管这两个器件，做起来较为轻松。
- 个人感觉本次实验主要难点是：如何协调两个任务的运行，实现一秒一次显示当前温度和湿度。