浙江大学

本科实验报告

课程名称：嵌入式系统

姓　　名：Jessie Peng

学　　院：计算机科学与技术学院

系：

专　　业：

学　　号：

指导教师：王总辉

2019　年　5　月　31　日

浙江大学实验报告

课程名称：嵌入式系统 实验类型：综合

实验项目名称：实验 5：uC/OS 室温计

学生姓名：Jessie Peng 专业： 学号：

实验地点：曹西 501 实验日期：2019 年 5 月 31 日 指导老师：王总辉

一、实验目的和要求

- 学习 uC/OS-II 的应用程序编写
- 理解如何直接操纵 GPIO，体会与 Linux 的不同
- 学习单总线设备的访问方式
- 学习 7 段数码管的时分复用驱动方式

二、实验内容和原理

【实验器材——硬件】

- STM32F103 核心板
- USB 串口板
- 面包版
- 两位 7 段数码管（共阳）
- 360 欧姆 1/8W 电阻 2 个
- DHT-11 温湿度传感器
- 面包线若干

【实验器材——软件】

- MDK 软件及扩展
    - mdk_513
    - Keil.STM32F1xx_DFP.2.0.0 pack
- Fritzing

【实验器材——文档】

- lab5.pdf

【实验内容——硬件平台搭建】

- 设计输出方案，画连线示意图
- 在面包板上连线，完成外部电路

【实验内容——嵌入式软件设计】

- 编写 C/C++程序，测设程序和电路
  - 测试、实现 uC/OS-II 对 GPIO 的访问
  - 实现 DHT-11 数据的读
  - 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
  - 用两个 uC/OS-II 任务，一个定时器读 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

三、实验步骤和结果记录

【下载 uC/OS-II 并配置工程环境】

下载源代码并安装，然后进行工程配置：

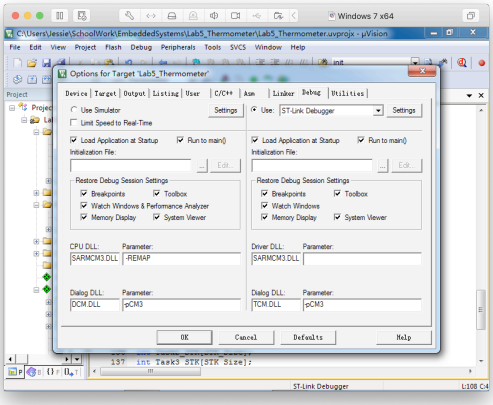
1. 新建工程，选择 STM32F103C8，选择 CMSIS 下的 CORE 和 Device 下的 Startup，以及 Device 下的 StdPeriph Drivers 下的 Framework，RCC 和 GPIO
2. 工程中和实际目录中都新建几个目录：APP、UCOS、BSP、LIB、CPU、Output
3. 工程上右键，Options，Output 页签，Select Folder for Objects，进入 Output 目录，点击 OK
4. 把 Micrium\Software\uCOS-II\Source 目录中的文件拷贝到 UCOS 目录下，并添加到工程中
5. 复制 Micrium\Software\EvalBoards\ST\STM3210B-EVAL\RVMDK\OS-Probe 目录下的文件 app_cfg.h，os_cfg.h 和 includes.h 到 APP 目录中
6. 复制 Micrium\Software\uCOS-II\Ports\arm-cortex-m3\Generic\RealView 目录下的所有文件到 CPU 目录
7. 复制 Micrium\Software\EvalBoards\ST\STM3210B-EVAL\RVMDK\BSP 目录下

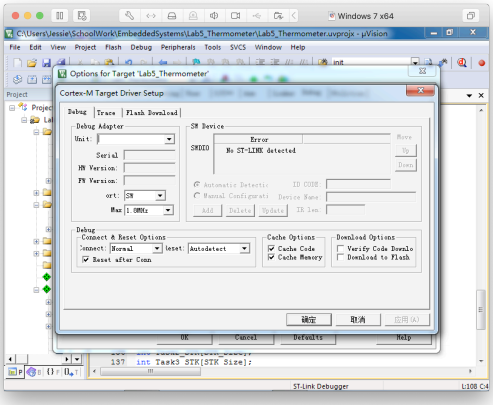
的 bsp.h 到 BSP 目录中

8. 复制 Micrium\Software\uC-CPU\ARM-Cortex-M3\RealView 目录和 Micrium\Software\uC-CPU 目录下的所有文件到 CPU 目录下
9. 复制 Micrium\Software\uC-LIB 目录下的所有.h 文件到 LIB 目录下
10. 工程 Options 中，C/C++页签，Defines 中添加 USE_STDPERIPH_DRIVER
11. 工程 Options 中，C/C++页签，Include Paths,点击后面省略号可选择 include 目录，添加 UCOS、APP、CPU、LIB、BSP、RTE、RTE/Device/STM32F103C8 路径到 include path 中，其中 RTE、RTE/Device/STM32F103C8 为工程目录下自动生成的目录
12. 修改 os_cfg.h 文件

```
1.  #define OS_APP_HOOKS_EN          0
```

13. 在工程 Options 的 Debug 选项卡中，选择 ST-Link Debugger，并点开旁边的 Settings，选择 ort 为 SW：





14. BSP 目录下新建 BSP.c 文件

```
#include <bsp.h>
CPU_INT32U  BSP_CPU_ClkFreq (void) {
    RCC_ClocksTypeDef  rcc_clocks;
    RCC_GetClocksFreq(&rcc_clocks);
    return ((CPU_INT32U)rcc_clocks.HCLK_Frequency);
}
INT32U  OS_CPU_SysTickClkFreq (void) {
    INT32U  freq;
    freq = BSP_CPU_ClkFreq();
    return (freq);
}
```

15. 注释掉 bsp.h 中的#include <stm32f10x_lib.h>和#include <lcd.h>

16. app_cfg.h 文件中，修改为

```
#define  APP_OS_PROBE_EN    DEF_DISABLED
#define  APP_PROBE_COM_EN   DEF_DISABLED
```

17. 注释掉 includes.h 文件中的#include <stm32f10x_lib.h>和#include <lcd.h>

18. APP 目录下新建 app.c 文件并创建时钟中断处理函数

```
#include <includes.h>

void SysTick_Handler(void)
{
    OS_CPU_SR cpu_sr;

    OS_ENTER_CRITICAL(); // Tell uC/OS-II that we are starting an ISR
    OSIntNesting++;
    OS_EXIT_CRITICAL();

    OSTimeTick(); // Call uC/OS-II's OSTimeTick()
    OSIntExit(); // Tell uC/OS-II that we are leaving the ISR
}

int main()
{
    OSInit();
    OSStart();
    return 0;
}
```

19. 在 startup_stm32f10x_md.s 中，将没有实现的 PendSV_Handler 替换为 uCOS

实现的 PendSV_Handler

```
; Vector Table Mapped to Address 0 at Reset
                AREA    RESET, DATA, READONLY
                EXPORT  __Vectors
                EXPORT  __Vectors_End
                EXPORT  __Vectors_Size
                IMPORT  OS_CPU_PendSVHandler
__Vectors       DCD     __initial_sp              ; Top of Stack
                DCD     Reset_Handler             ; Reset Handler
                DCD     NMI_Handler               ; NMI Handler
                DCD     HardFault_Handler         ; Hard Fault Handler
                DCD     MemManage_Handler         ; MPU Fault Handler
                DCD     BusFault_Handler          ; Bus Fault Handler
                DCD     UsageFault_Handler        ; Usage Fault Handler
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     SVC_Handler               ; SVCall Handler
                DCD     DebugMon_Handler          ; Debug Monitor Handler
                DCD     0                         ; Reserved
                DCD     OS_CPU_PendSVHandler      ; PendSV Handler
                DCD     SysTick_Handler           ; SysTick Handler
```

准备工作结束，之后我们的程序在 app.c 里面编写。

配置本实验需要用到的 GPIO 引脚：

```
1.  void GPIO_Configuration(void)
2.  {
3.      GPIO_InitTypeDef GPIO_InitStructure;
4.
5.      RCC_DeInit();
6.      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOB | RCC
    _APB2Periph_GPIOA, ENABLE);
7.
8.      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14;
9.      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
10.     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
11.     GPIO_Init(GPIOC, &GPIO_InitStructure);
12.
13.     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
14.     GPIO_Init(GPIOB, &GPIO_InitStructure);
15.
16.     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPI
    O_Pin_3 | GPIO_Pin_4 |
17.                                                                   GPIO_Pin_5 |
     GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_11 | GPIO_Pin_12;
18.     GPIO_Init(GPIOA, &GPIO_InitStructure);
19. }
```

利用标准库的延时函数构造毫秒级延时函数，标准库的延时是非阻塞的（其他类似 while 和 for 循环来延时是阻塞的，即执行的时候不能切换任务）：

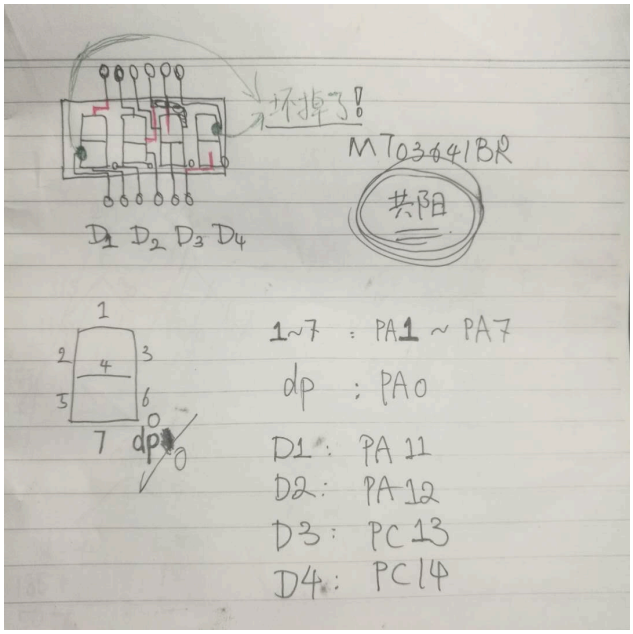
```
void Delay_ms(int times)
{
    OSTimeDly(OS_TICKS_PER_SEC/1000 * times);
}
```

自己写一个微秒级延时函数，实际上时间与真正的一微秒有出入，只能自己测试摸索：

```
void Delay_us(int times)
{
    unsigned int i;
    for (i = 0; i < times; i++)
    {
    }
}
```

【七段数码管映射】

使用共阳四位七段数码管，从左至右的四个使能信号引脚分别连接开发板的 PA11, PA12, PC13, PC14，编号 0～7 的 led 引脚分别连接开发板的 PA0 ~ PA7，引脚编号方式如下：



新建一个 task 用于在数码管上显示数字：

```
void task_show_segments(void* pdata)
{
```

```
    while (1)
    {
        segments_TDM_show(segments_val);
        Delay_ms(7);
    }
}
```

数码管一次只能显示一位数，要用时分复用（TDM）的方式轮流显示四个位上的数字，利用人的视觉暂留就可以达到同时看见四个数字的效果：

```
void segments_TDM_show(int num)
{
    int i;
    static int index = -1;
    index = (index + 1) % 4;

    for (i = 0; i < index; i++)
    {
        num /= 10;
    }

    digit_select(index);
    digit_show(num % 10, 0);
}
```

digit_select()用于选择显示哪一位：

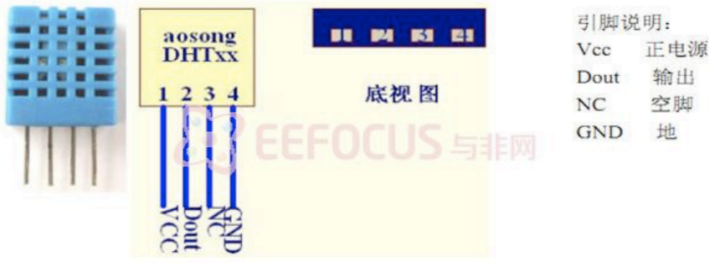
```
void digit_select(int index)
{
    BitAction v[4];
    int i;
    for (i = 0; i < 4; i++)
    {
        v[i] = Bit_RESET;
    }
    v[3-index] = Bit_SET;

    GPIO_WriteBit(GPIOA, GPIO_Pin_11, v[0]);
    GPIO_WriteBit(GPIOA, GPIO_Pin_12, v[1]);
    GPIO_WriteBit(GPIOC, GPIO_Pin_13, v[2]);
    GPIO_WriteBit(GPIOC, GPIO_Pin_14, v[3]);
}
```

digit_show()将 0-9 每个数字对应的数码管状态硬编码，显示对应的数字：

```
1.  void digit_show(int digit, int point)
2.  {
3.      int segment;
4.      BitAction v[8];
5.      int i;
6.      switch (digit)
7.      {
8.          case 0: segment = 0x11; break; // 0b00010001
9.          case 1: segment = 0xb7; break; // 0b10110111
10.         case 2: segment = 0x45; break; // 0b01000101
11.         case 3: segment = 0x25; break; // 0b00100101
12.         case 4: segment = 0xa3; break; // 0b10100011
13.         case 5: segment = 0x29; break; // 0b00101001
14.         case 6: segment = 0x09; break; // 0b00001001
15.         case 7: segment = 0xb5; break; // 0b10110101
16.         case 8: segment = 0x01; break; // 0b00000001
17.         case 9: segment = 0x21; break; // 0b00100001
18.         default: segment = 0x49; // 0b01001001 'E' for error
19.     }
20.     segment ^= (point & 0x01);
21.
22.     for (i = 0; i < 8; i++)
23.     {
24.         if (segment & (0x01 << i))
25.         {
26.             v[i] = Bit_SET;
27.         }
28.         else
29.         {
30.             v[i] = Bit_RESET;
31.         }
32.     }
33.
34.     GPIO_WriteBit(GPIOA, GPIO_Pin_0, v[0]);
35.     GPIO_WriteBit(GPIOA, GPIO_Pin_1, v[1]);
36.     GPIO_WriteBit(GPIOA, GPIO_Pin_2, v[2]);
37.     GPIO_WriteBit(GPIOA, GPIO_Pin_3, v[3]);
38.     GPIO_WriteBit(GPIOA, GPIO_Pin_4, v[4]);
39.     GPIO_WriteBit(GPIOA, GPIO_Pin_5, v[5]);
40.     GPIO_WriteBit(GPIOA, GPIO_Pin_6, v[6]);
41.     GPIO_WriteBit(GPIOA, GPIO_Pin_7, v[7]);
42. }
```

【温湿度传感器】

DHT11 是单总线设备，通过握手协议进行通信，引脚说明如下：



本实验中，1 号接开发板的 3.3V，2 号接开发板的 B0，4 号接开发板的 GND。

新建一个任务用于与 DHT11 通信：

```
1.  void task_read_DHT11(void* pdata)
2.  {
3.      uint8_t buf[5];
4.      int state;
5.      memset(buf, 0, sizeof(buf));
6.
7.      while (1)
8.      {
9.          state = DHT11_Read_Data(buf);
10.         switch (state)
11.         {
12.             case DHT11_CS_ERROR:
13.                 segments_val = 9002;
14.                 break;
15.             case DHT11_NO_CONN:
16.                 segments_val = 9001;
17.                 break;
18.             case DHT11_OK:
19.                 segments_val = DHT11_Temperature(buf);
20.                 break;
21.         }
22.         Delay_ms(1000);
23.     }
24. }
```

DHT11_Read_Data()尝试从 DHT11 接收数据并保存到 buf 中（buf[0:1]是湿度，buf[2:3]是温度，buf[5]=buf[0]+buf[1]+buf[2]+buf[3]是校验和），遵循如下的数据

传输协议：

1. stm32 输出低电平至少 18ms(只有此处为 ms，其余均为μs)。
2. stm32 输出高电平 20~40μs
3. DHT11 反馈低电平 80μs
4. DHT11 反馈高电平 80μs
5. 以上为双方握手，以下开始准备接受数据。数据总长 40 个 bit，输入为大端输入，即高位的 bit 先进行传输。每个 byte 表示一个数值。按照接受顺序分别表示湿度整数，湿度小数，温度整数，温度小数，校验码。校验码为前方四个 byte 的和。
6. 对于每个传输的 bit，DHT11 会首先输出 50μs 的低电平
7. 而后以输出高电平的时间决定每个 bit 的值。高电平持续时间为 20~30μs 的为 bit 0，高电平持续时间为 70μs 的表示 bit 1。

```
uint8_t DHT11_Read_Data(uint8_t *buf)
{
    int i;
    unsigned int cpu_sr;

    // enter critical section
    OS_ENTER_CRITICAL();

    val = 10;

    // handshake
    DHT11_Rst();

    // proceed response
    if (DHT11_Check() == 0)
    {
        // low
        DHT11_Wait(1, 2);
        // high
        DHT11_Wait(0, 3);
        // read 40 bits
        for (i = 0; i < 5; i++)
        {
            buf[i] = DHT11_Read_Byte();
        }

```

```
        DHT11_Pin_OUT();

        // exit critical section
        OS_EXIT_CRITICAL();

        // checksum
        if (buf[0] + buf[1] + buf[2] + buf[3] == buf[4])
        {
            return DHT11_OK;
        }
        else
        {
            return DHT11_CS_ERROR;
        }
        return DHT11_OK;
    }
    else
    {
        // exit critical section
        OS_EXIT_CRITICAL();

        return DHT11_NO_CONN;
    }
}
```

其中握手阶段封装在函数 DHT11_Rst()中：

```
void DHT11_Rst()
{
    // handshake: send
    DHT11_Pin_OUT();
    DHT11_Set(0);
    Delay_us(25000);
    DHT11_Set(1);
    Delay_us(40);
    DHT11_Set(0);

    // handshake: receive
    DHT11_Pin_IN();
}
```

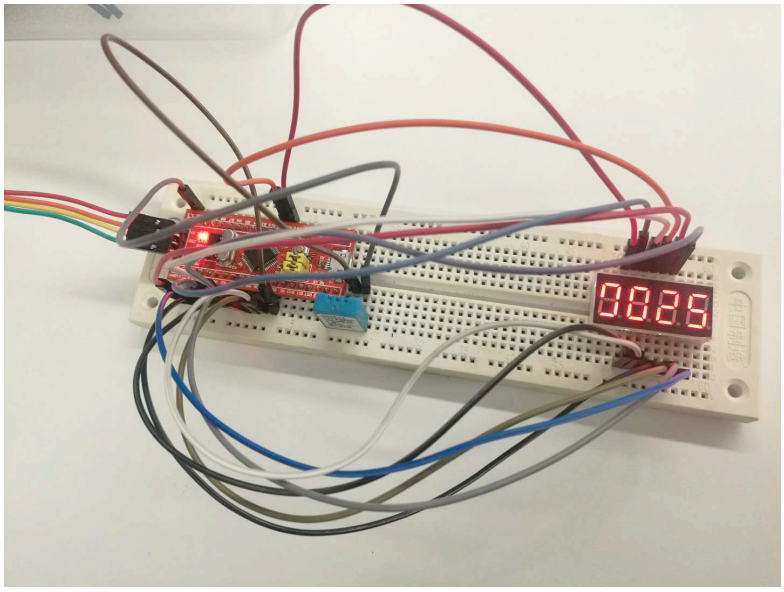
最后主函数调用两个 task 并发执行：

```
int segments_val = 0;
```

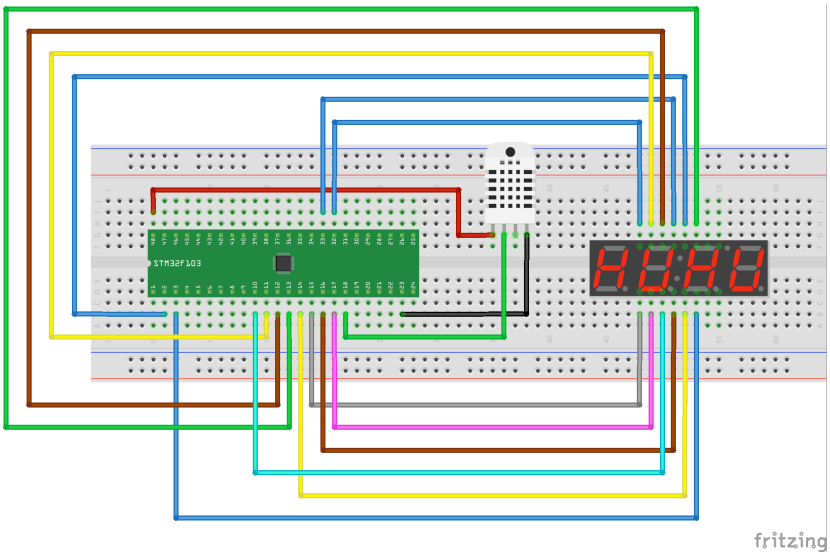
```
2.
3.  #define STK_Size 100
4.  int Task1_STK[STK_Size];
5.  int Task2_STK[STK_Size];
6.
7.  int main()
8.  {
9.      GPIO_Configuration();
10.     OSInit();
11.     OS_CPU_SysTickInit();
12.
13.     OSTaskCreate(task_read_DHT11, (void*)0, (OS_STK*)&Task2_STK[STK_Size-1],
    1);
14.     OSTaskCreate(task_show_segments, (void*)0, (OS_STK*)&Task1_STK[STK_Size-
    1], 2);
15.
16.     OSStart();
17.     return 0;
18. }
```

四、实验结果分析

实验连线和结果拍照（显示室温 25 摄氏度）：



连线示意图（由于 Fritzing 元件库中没找到完全匹配的 STM32 和数码管，所以连线不完全与实验对应，仅当示意）：



五、讨论与心得

本次实验需要使用到 uCOS，而且连线比较复杂，好在实验指导和网上教程都很多，在 uCOS 和数码管连线上都没有遇到什么困难。唯一遇到问题的是温湿度传感器，一开始我一直无法接收到 DHT 的数据，怀疑是握手和数据传输协议的时间控制方面出了问题。因为跟 DHT 握手必须严格控制高低电平的时长，但是微秒级的延时函数是自己通过估计编写的，并不能准确模拟真实的一微秒，这就可能导致拉高拉低电平的时间太短或者太长。经过不断调整和测试，最后发现只要将一部分毫秒级的延时也采用自己编写的那个微秒级延时函数，而不使用标准库的延时函数，就能正常接收传感器的数据了。原因可能跟非阻塞延迟有关，自己编写的循环是阻塞的而标准库的延时是非阻塞的，这就意味着用标准库延时的同时，可能切换为执行显示数码管的任务，而在没有接收到传感器的数据时数码管会显示一个错误码。