

第 10 章 竞赛试题设计与实现

本章以竞赛试题——“温湿度监控设备”为例介绍 STM32 MCU 系统的设计与实现。

“温湿度监控设备”通过采集传感器输出电压信号和信号频率得到环境温湿度数据，并能够根据设定的温湿度阈值执行相应动作，系统框图如图 10.1 所示。

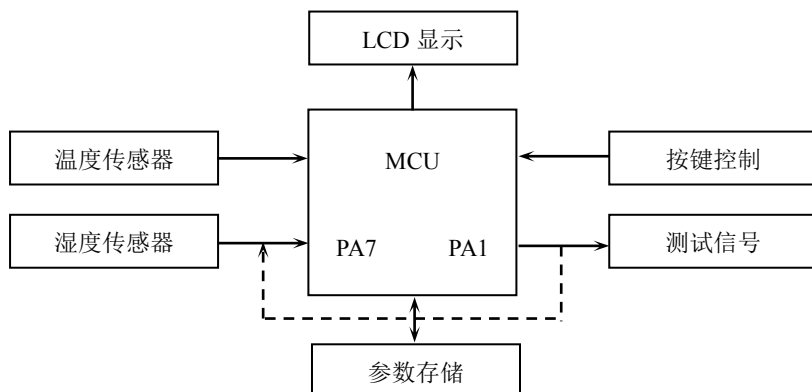


图 10.1 系统框图

10.1 设计任务及要求

1. 温度测量功能

通过竞赛板上电位器 R37 输出电压模拟温度传感器输出信号，温度测量范围为-20℃~60℃，电压温度关系如下式所示：

$$T = k * VR37 + b$$

其中 T 代表环境温度，VR37 为电位器输出信号，k、b 为常数，0V 时对应温度为-20℃，3.3V 对应 60℃。

2. 湿度测量功能

通过竞赛板 PA7 引脚检测输入信号频率，相对湿度测量范围为 10%~90%，频率湿度关系如下式所示：

$$H = m * F + n$$

其中 H 代表环境湿度，F 为传感器输入到设备信号频率，m、n 为常数，1KHz 对应相对湿度为 10%，10KHz 对应 90%。

3. 测试信号

通过竞赛板 PA1 引脚输出频率 1KHz 到 10KHz 方波，模拟湿度传感器输出信号。

4. 参数设置与数据记录功能

可以通过按键设定温、湿度上限和数据采样间隔，温湿度数据记录间隔可设置为 1~5 秒，要求至少保存 60 组数据，数据不需要写入 FLASH 或 E2PROM 存储器。

5. RTC 功能

通过单片机片内 RTC 设计实现实时时钟功能。

6. 按键功能定义

设备上电后通过 LCD 显示实时温、湿度数据和数据记录次数，显示格式如图 10.2 所示。

实时数据	
当前温度:	-20℃
当前湿度:	60%
实时时钟:	12-50-00
记录次数:	20

图 2 LCD 显示界面参考图（实时数据显示）

B1 按键：“功能”按键，按下后进入功能设定界面（如图 10.3 所示），再次按下 B1 按键时退出设置界面，保存用户设定的结果到 E2PROM，并返回图 2 所示的实时数据显示界面。

参数设置	
温度上限:	40℃
湿度上限:	80%
采样间隔:	1s
测试信号:	1.5KHz

图 3 LCD 显示界面参考图（阈值设置界面）

B2 按键：“切换”按键，切换选择 4 个待修改的运行参数，被选中的参数突出显示（如图 10.3 “湿度上限”）。

B3 按键：“加”按键，当前选择的参数是温度时，参数加 1℃；选择采样间隔时，参数加 1 秒；选择参数为湿度时，参数加 5%；选择测试信号时，测试信号频率加 500Hz；

B4 按键：“减”按键，当前选择的参数是温度时，参数减 1℃；选择采样间隔时，参数减 1 秒，选择参数为湿度时，参数减 5%；选择测试信号时，测试信号频率减 500Hz；

备注：“加”、减按键应根据当前调整的参数属性，设计合理的边界值。

7. 串口功能

设备通过串口完成阈值和数据查询功能，使用 STM32 USART2 完成上述串口功能，波特率设置为 9600。

7.1 阈值查询

通过 PC 机给设备发送字符 ‘C’，设备返回包含当前温湿度阈值和当前时间的字符串，格式可自定义。

7.2 数据查询

通过 PC 机给设备发送字符 ‘T’，设备返回包含当前采集到的所有温、湿度数据的字符串，每条温、湿度数据应包含该条数据的记录时间，格式可自定义。

8. 报警指示功能

当前温度值超过温度上限时，指示灯 L1 闪烁报警；

当前湿度值超过湿度上限时，指示灯 L2 闪烁报警；

每次数据采集时，指示灯 L3 亮、灭的状态反转。

10.2 系统设计

通过分析设计任务及要求，可以得到系统详细框图如图 10.4 所示。

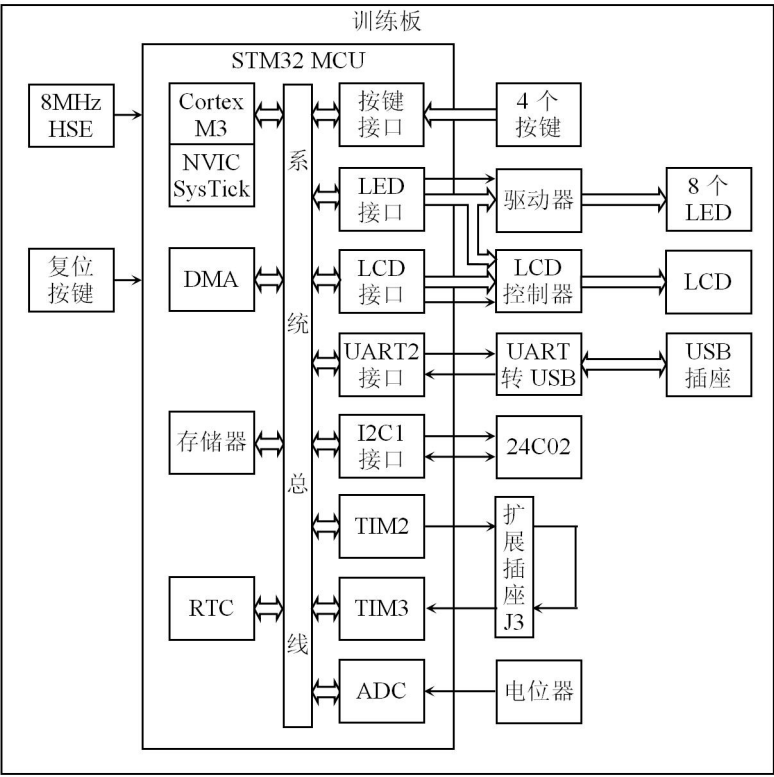


图 10.4 系统详细框图

温度测量使用的电位器 R37 连接在 PB0 (ADC_IN8)，数模转换值与温度的关系为：

$$T = k * N + b$$

其中 T 代表环境温度，N 为模数转换值，k、b 为常数，N=0 对应温度为-20℃，N=4095 对应 60℃，代入上式可得数模转换值与温度的关系为：

$$T = 80 * N / 4095 - 20$$

湿度测量通过 PA7 检测输入信号的频率，使用的是 TIM3_CH2 的输入捕捉功能，捕捉值与湿度的关系为：

$$H = m * C + n$$

其中 H 代表环境湿度，C 为捕捉值，m、n 为常数，捕捉值的确定与 TIM3 的设置有关：系统的默认频率是 8MHz，设 TIM3 的预分频值为 80，则计数器的输入频率为 100KHz，1KHz 的捕捉值为 100，对应湿度为 10%，10KHz 的捕获值为 10，对应湿度为 90%，代入上式可得捕获值与湿度的关系为：

$$H = (890 - 8 * C) / 9$$

测试信号通过 PA1 输出频率 1KHz 到 10KHz 方波（间隔 500Hz），使用的是 TIM2_CH2 的输出比较功能：系统的默认频率是 8MHz，设 TIM2 的预分频值为 80，则计数器的输入频率为 100KHz，1KHz 的重装值为 100，10KHz 的重装值为 10，频率与重装值的关系为：

$$R = 110 - F / 100$$

参数存储通过 I2C1 接口访问 24C02 实现,串口功能通过 UART2 和 UART 转 USB 实现,按键接口、LED 接口和 LCD 接口均通过 GPIO 实现。

系统主程序、按键处理程序和 LCD 处理程序的流程图如图 10.5 所示。

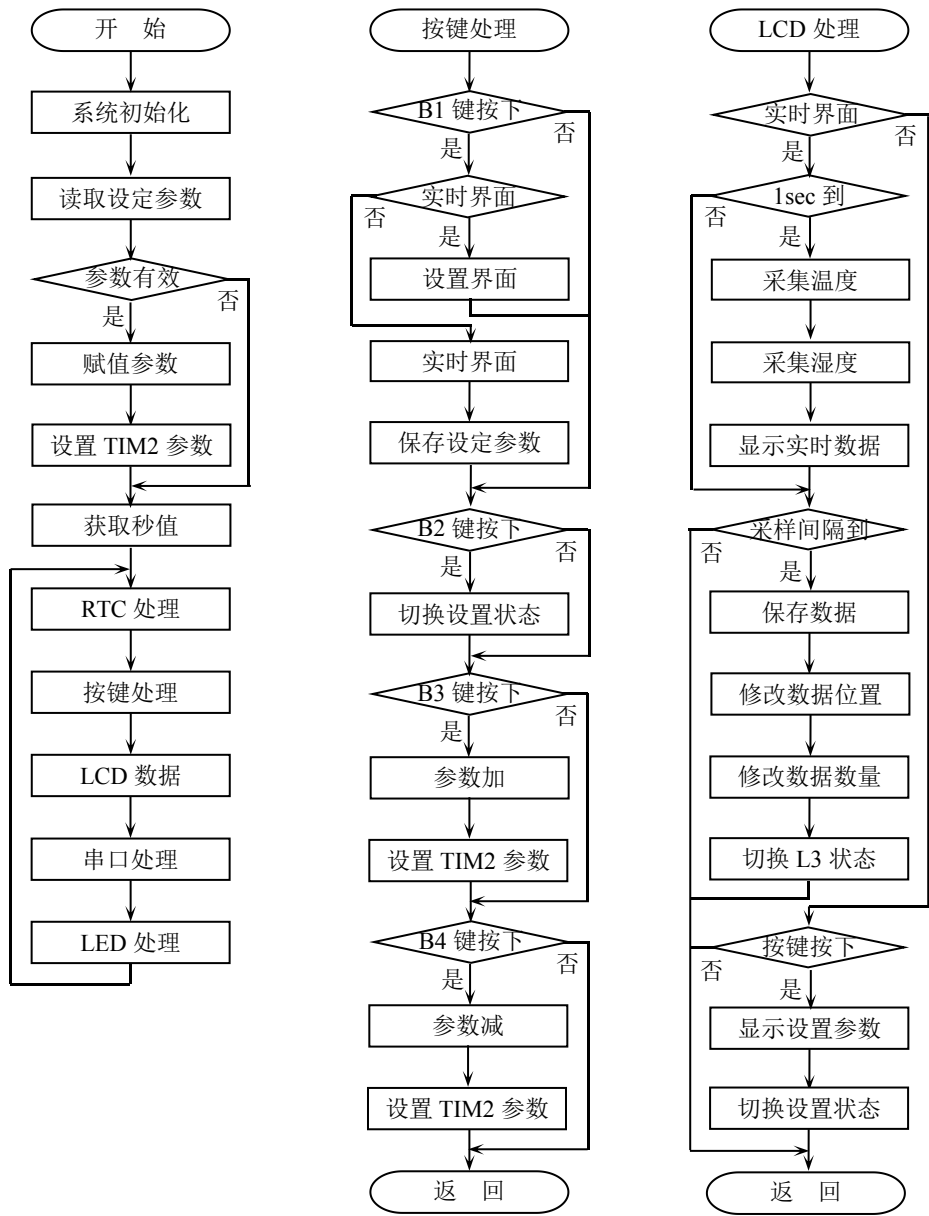


图 10.5 实时钟系统程序流程图

主程序首先对系统进行初始化,包括 RTC 初始化、ADC 初始化、TIM2 初始化、TIM3 初始化、按键接口初始化、LED 接口初始化、LCD 接口初始化、USART2 接口初始化和 I2C1 接口初始化等。系统初始化后从 24C02 中读取设定参数,并判断参数是否有效,如果有效则赋值参数,否则使用程序默认参数,然后获取秒值作为后续程序判断时间间隔的初值。

主循环的操作包括 RTC 处理、按键处理、LCD 处理、串口处理和 LED 处理等,RTC 处理实现实时时钟功能,按键处理和 LCD 处理采用状态机实现相应功能,串口处理根据接收

的字符是‘C’还是‘T’分别按要求发送字符串，LED 处理实现超限闪烁和采集指示。

保存设定参数到 24C02 和根据设置调整 TIM2 参数包含在按键处理中，采集温度和采集湿度包含在 LCD 处理中。

10.2.1 初始化程序设计

初始化程序包括 RTC 初始化、ADC 初始化、TIM2 初始化、TIM3 初始化、按键接口初始化、LED 接口初始化、LCD 接口初始化、USART2 接口初始化和 I2C1 接口初始化等。

(1) RTC 初始化程序设计

RTC 初始化程序设计如下：

```
void RTC_Init(void)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_BKP, ENABLE);

    PWR_BackupAccessCmd(ENABLE);           // 允许访问备份域
    BKP_DeInit();                           // 复位备份域

    RCC_LSICmd(ENABLE);                     // 允许 LSI
    while(!RCC_GetFlagStatus(RCC_FLAG_LSIRDY)); // 等待 LSI 准备好
    RCC_RTCCLKConfig(RCC_RTCCLKSource_LSI); // 选择 LSI 作为 RTC 时钟
    RCC_RTCCLKCmd(ENABLE);                  // 允许 RTC 时钟

    RTC_WaitForSynchro();                   // 等待 RTC 寄存器同步
    RTC_WaitForLastTask();                  // 等待 RTC 寄存器操作完成

    RTC_SetPrescaler(39999);                // 设置 RTC 周期为 1s
    RTC_WaitForLastTask();
}
```

RTC 功能也可用 SysTick 和 TIM 实现。

(2) ADC 初始化程序设计

ADC 初始化程序设计如下：

```
void ADC1_Init(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
    // PB0 (IN8) 模拟输入
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

```

ADC_InitStruct.ADC_Mode = ADC_Mode_Independent;    // 默认值
ADC_InitStruct.ADC_ScanConvMode = DISABLE;        // 默认值
ADC_InitStruct.ADC_ContinuousConvMode = ENABLE;    // 连续转换
ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;
ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right; // 默认值
ADC_InitStruct.ADC_NbrOfChannel = 1;              // 默认值
ADC_Init(ADC1, &ADC_InitStruct);                  // 初始化 ADC1

ADC_RegularChannelConfig(ADC1, ADC_Channel_8, 1,
    ADC_SampleTime_1Cycles5);                      // 配置通道 8

ADC_Cmd(ADC1, ENABLE);                             // 开启 ADC1
ADC_Cmd(ADC1, ENABLE);                             // 开始转换
}

```

(3) TIM2 初始化程序设计

TIM2 初始化程序设计如下：

```

void TIM2_Init(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
    // PA1 复用推挽输出
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    TIM_TimeBaseInitStruct.TIM_Prescaler = 79;    // 预分频值 (8MHz/80=100kHz)
    TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up; // 默认值
    TIM_TimeBaseInitStruct.TIM_Period = 110-csxx/100;    // 自动装载值
    TIM_TimeBaseInitStruct.TIM_ClockDivision = TIM_CKD_DIV1;    // 默认值
    TIM_TimeBaseInitStruct.TIM_RepetitionCounter = 0;    // 默认值
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseInitStruct);    // 初始化 TIM2 时基

    TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStruct.TIM_Pulse = (110-csxx/100)>>1;    // 占空比 1/2
    TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;    // 默认值
    TIM_OC2Init(TIM2, &TIM_OCInitStruct);    // 初始化 TIM2 输出比较 2

    TIM_Cmd(TIM2, ENABLE);    // 允许 TIM2
}

```

```
}
```

(4) TIM3 初始化程序设计

TIM3 初始化程序设计如下:

```
void TIM3_Init(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
    /* PA7 浮空输入 (复位状态, 可以省略)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure); */

    TIM_TimeBaseInitStruct.TIM_Prescaler = 79;    // 预分频值 (8MHz/80=100kHz)
    TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up; // 默认值
    TIM_TimeBaseInitStruct.TIM_Period = 65535;
    TIM_TimeBaseInitStruct.TIM_ClockDivision = TIM_CKD_DIV1;    // 默认值
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseInitStruct);    // 初始化 TIM3 时基

    TIM_ICInitStruct.TIM_Channel = TIM_Channel_2;
    TIM_ICInitStruct.TIM_ICPolarity = TIM_ICPolarity_Falling;
    TIM_ICInitStruct.TIM_ICSelection = TIM_ICSelection_DirectTI;
    TIM_ICInit(TIM3, &TIM_ICInitStruct);                // 初始化 TIM3 通道 2

    TIM_SelectInputTrigger(TIM3, TIM_TS_TI2FP2);        // TI2FP2 触发
    TIM_SelectSlaveMode(TIM3, TIM_SlaveMode_Reset);    // 复位模式
    TIM_Cmd(TIM3, ENABLE);                             // 允许 TIM3
}
```

(5) 按键接口初始化程序设计

按键接口初始化程序设计如下:

```
void KEY_Init(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    /* PA0、PA8 浮空输入 (复位状态, 可以省略)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure); */
    /* PB1、PB2 浮空输入 (复位状态, 可以省略)
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
```

```

    GPIO_Init(GPIOB, &GPIO_InitStructure); */
}

```

(6) LED 接口初始化程序设计

LED 接口初始化程序设计如下：

```

void LED_Init(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    // PD2 推挽输出（PC8~PC15 初始化在 LCD 接口初始化中完成）
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
}

```

(7) LCD 接口初始化程序设计

LCD 接口初始化程序使用 lcd.c 中的 STM3210B_LCD_Init()实现。

(8) USART2 接口初始化程序设计

USART2 接口初始化程序设计如下：

```

void USART2_Init(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
    // TX2-PA2 复用推挽输出
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    /* RX2-PA3 浮空输入（复位状态，可以省略）
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure); */
    // 初始化 USART2（波特率 115200，8 个数据位，1 个停止位，无校验，无硬件流控）
    USART_InitStructure.USART_BaudRate = 115200;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_InitStructure.USART_HardwareFlowControl
        = USART_HardwareFlowControl_None;
    USART_Init(USART2, &USART_InitStructure);
    // 允许 USART2
}

```



```

    USART_Cmd(USART2, ENABLE);
}

```

(9) I2C1 接口初始化程序设计

I2C1 接口初始化程序使用 i2c.c 中的 i2c_init()实现。

10.2.2 处理程序设计

处理程序包括 RTC 处理、按键处理、LCD 处理、串口处理和 LED 处理等，其中按键处理和 LCD 处理采用的状态机如图 10.6 所示。

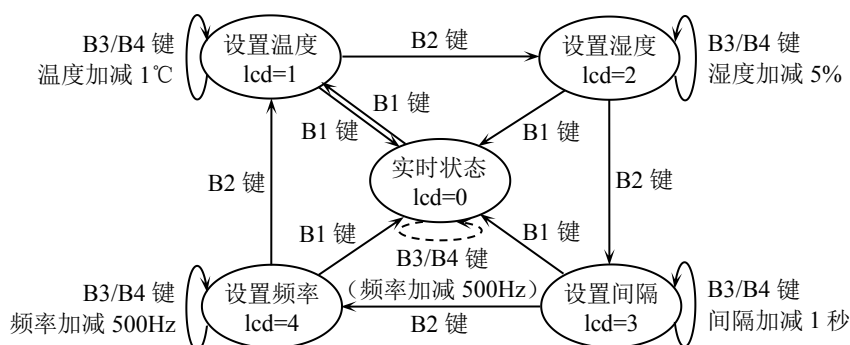


图 10.6 按键处理和 LCD 处理状态机

系统共有 5 个状态：1 个实时状态（显示实时数据）和 4 个设置状态（分别设置温度上限、湿度上限、采样间隔和测试频率），实时状态和设置状态间用 B1 键转换，4 个设置状态间用 B2 键转换，4 个设置状态均可用 B3/B4 键加减参数，实时状态的 B3/B4 键操作是增加功能，用于实时状态下的频率加减，方便湿度变化的观察，温度变化的观察通过改变电位器 R37 的值实现。

(1) RTC 处理程序设计

RTC 处理程序设计如下：

```

void RTC_Proc(void)
{
    if(RTC_GetFlagStatus(RTC_FLAG_SEC))
    {
        RTC_WaitForSynchro(); // 等待 RTC 寄存器同步
        RTC_ClearFlag(RTC_FLAG_SEC);
        sec0 = RTC_GetCounter(); // 获取计数值
        hour = sec0 / 3600;
        min = (sec0 % 3600) / 60;
        sec = (sec0 % 3600) % 60;
    }
}

```

(2) 按键处理程序设计

按键处理程序设计如下：

```

void KEY_Proc(void)

```

```

{
    if(!GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0))           // B1 键按下
    {
        RTC_Delay(10);                                       // 延时 10ms 消抖
        if((!GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)) && (key == 0))
        {
            key = 1;                                         // 设置按下标志
            if(lcd == 0)
            {
                lcd = 1;                                     // 切换功能状态
            }
            else
            {
                lcd = 0;
                para[0] = wdsx;
                para[1] = sdsx;
                para[2] = cyjg;
                para[3] = csxh/100;
                I2C_EE_BufferWrite(para, 0, 4);              // 保存设置参数
                sec1 = sec0;
                sec2 = sec0;
            }
        }
    }
    else if(!GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_8))      // B2 键按下
    {
        RTC_Delay(10);                                       // 延时 10ms 消抖
        if((!GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_8)) && (key == 0))
        {
            key = 1;                                         // 设置按下标志
            if(lcd) lcd++;                                   // 切换设置状态
            if(lcd == 5) lcd = 1;
        }
    }
    else if(!GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_1))      // B3 键按下
    {
        RTC_Delay(10);                                       // 延时 10ms 消抖
        if((!GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_1)) && (key == 0))
        {
            key = 1;                                         // 设置按下标志

```

```

if(lcd) // 设置状态
{
    switch(lcd)
    {
        case 1:
        {
            if(wdsx < 60) wdsx++; // 温度上限加 1℃
            break;
        }
        case 2:
        {
            if(sdsx < 90) sdsx += 5; // 湿度上限加 5%
            break;
        }
        case 3:
        {
            if(cyhg < 5) cyhg++; // 采样间隔加 1s
            break;
        }
        case 4:
        {
            if(csxh < 10000) csxh += 500; // 测试频率加 500Hz
            break;
        }
    }
}
else
    if(csxh < 10000) csxh += 500;
TIM_SetAutoreload(TIM2, 110-csxh/100); // 自动装载值
TIM_SetCompare2(TIM2, (110-csxh/100) >> 1); // 匹配值
}
}
else if(!GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_2)) // B4 键按下
{
    RTC_Delay(10); // 延时 10ms 消抖
    if((!GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_2)) && (key == 0))
    {
        key = 1; // 设置按下标志
        if(lcd) // 设置状态
        {

```

```

switch(lcd)
{
    case 1:
    {
        if(wdsx > 0) wdsx--;           // 温度上限减 1℃
        break;
    }
    case 2:
    {
        if(sdsx > 10) sdsx -= 5;      // 湿度上限减 5%
        break;
    }
    case 3:
    {
        if(cyjpg > 1) cyjpg--;        // 采样间隔减 1s
        break;
    }
    case 4:
    {
        if(csxh > 1000) csxh -= 500;  // 测试频率减 500Hz
        break;
    }
}
else
    if(csxh > 1000) csxh -= 500;
TIM_SetAutoreload(TIM2, 110-csxh/100); // 自动装载值
TIM_SetCompare2(TIM2, (110-csxh/100) >> 1); // 匹配值
}
}
else // 按键松开
{
    key = 0; // 清除按下标志
}
}

```

(3) LCD 处理程序设计

LCD 处理程序设计如下:

```

void LCD_Proc(void)
{
    if(!lcd) // 实时数据

```

```

{
    if(sec0 != sec1)                                // 1sec 到
    {
        sec1 = sec0;
        LCD_DisplayStringLine(Line1, "    Shishi Shuju    "); // 显示实时数据
        wd = (80*ADC1->DR)/4095-20;                    // 采集温度
        sprintf((char*)lcdstr, "    DQWD:    %3dC    ", wd);
        LCD_DisplayStringLine(Line3, lcdstr);
        sd = (890-8*TIM3->CCR2)/9;                    // 采集湿度
        sprintf((char*)lcdstr, "    DQSD:    %3d%%    ", sd);
        LCD_DisplayStringLine(Line5, lcdstr);
        sprintf((char*)lcdstr, "    DQSJ: %02u:%02u:%02u ", hour, min, sec);
        LCD_DisplayStringLine(Line7, lcdstr);
        sprintf((char*)lcdstr, "    JLCS:    %2u    ", datsum);
        LCD_DisplayStringLine(Line9, lcdstr);
    }

    if((sec0-sec2) >= cyjg)                            // 采样间隔到
    {
        sec2 = sec0;
        data[datnum][0] = hour;                        // 保存时
        data[datnum][1] = min;                        // 保存分
        data[datnum][2] = sec;                        // 保存秒
        data[datnum][3] = wd;                        // 保存温度
        data[datnum][4] = sd;                        // 保存湿度

        if(++datnum == 60) datnum = 0;                // 修改数据位置
        if(datsum != 60) datsum++;                    // 修改数据数量
        led ^= 1;
    }
}

else                                                    // 设置参数
{
    if(key)                                            // 按键按下
    {
        LCD_DisplayStringLine(Line1, "    Canshu Shezhi    "); // 显示参数设置
        sprintf((char*)lcdstr, "    WDSX:    %3dC    ", wdsx);
        LCD_DisplayStringLine(Line3, lcdstr);
        sprintf((char*)lcdstr, "    SDSX:    %3d%%    ", sdsx);
        LCD_DisplayStringLine(Line5, lcdstr);
    }
}

```

```

sprintf((char*)lcdstr, " CYJG:    %lus    ", cyjg);
LCD_DisplayStringLine(Line7, lcdstr);
sprintf((char*)lcdstr, " CSXH: %2u.%1ukHz    ",
        csxh/1000, csxh%1000/100);
LCD_DisplayStringLine(Line9, lcdstr);

LCD_SetTextColor(Red);                                // 突出显示
switch(lcd)
{
    case 1:
    {
        sprintf((char*)lcdstr, " WDSX:  %3d%    ", wdsx);
        LCD_DisplayStringLine(Line3, lcdstr);          // 温度上限
        break;
    }
    case 2:
    {
        sprintf((char*)lcdstr, " SDSX:  %3d%%    ", sdsx);
        LCD_DisplayStringLine(Line5, lcdstr);          // 湿度上限
        break;
    }
    case 3:
    {
        sprintf((char*)lcdstr, " CYJG:    %lus    ", cyjg);
        LCD_DisplayStringLine(Line7, lcdstr);          // 采样间隔
        break;
    }
    case 4:
    {
        sprintf((char*)lcdstr, " CSXH: %2u.%1ukHz    ",
                csxh/1000, csxh%1000/100);
        LCD_DisplayStringLine(Line9, lcdstr);          // 测试信号
    }
}
LCD_SetTextColor(White);
}
}
}

```

(4) 串口处理程序设计

串口处理程序设计如下：

```

void USART2_Proc(void)
{
    u8 chr, i;
    chr = USART_ReceiveData_NonBlocking(USART2);           // 非阻塞接收数据
    if(chr == 'C')
        printf("C %02u:%02u:%02u %3dC %2u%%\r\n", hour, min, sec, wdsx, sdsx);
    if(chr == 'T')
        for(i=0; i<datsum; i++)                           // 发送存储数据
            printf("T %02u:%02u:%02u %3dC %2u%%\r\n",
                data[i][0], data[i][1], data[i][2], data[i][3], data[i][4]);
}

```

(5) LED 处理程序设计

LED 处理程序设计如下:

```

void LED_Proc(void)
{
    GPIO_Write(GPIOC, 0xFF00);
    if((wd >= wdsx) && (sec0 & 1))
    {
        GPIO_ResetBits(GPIOC, GPIO_Pin_8);               // 点亮 L1
    }
    else
    {
        GPIO_SetBits(GPIOC, GPIO_Pin_8);                  // 熄灭 L1
    }
    if((sd >= sdsx) && (sec0 & 1))
    {
        GPIO_ResetBits(GPIOC, GPIO_Pin_9);               // 点亮 L2
    }
    else
    {
        GPIO_SetBits(GPIOC, GPIO_Pin_9);                  // 熄灭 L2
    }
    if(led & 1)
    {
        GPIO_ResetBits(GPIOC, GPIO_Pin_10);              // 点亮 L3
    }
    else
    {

```

```

        GPIO_SetBits(GPIOC, GPIO_Pin_10);           // 熄灭 L3
    }
    GPIO_SetBits(GPIOD, GPIO_Pin_2);                 // 选通显示
    GPIO_ResetBits(GPIOD, GPIO_Pin_2);
}

```

10.3 系统实现

系统的实现方法前面已多次介绍，这里将主要步骤介绍如下：

(1) 在 D:\下新建文件夹 Test，将竞赛目录中的下列文件复制到 Test 文件夹：

- lcd.c
- lcd.h
- fonts.h
- i2c.c
- i2c.h

(2) 将 lcd.h、fonts.h 和 i2c.c 中的下列语句：

```
#include "stm32f10x.h"           // 适用于 V3.5.0
```

修改为：

```
#include "stm32f10x_lib.h"       // 适用于 V2.0.1
```

将 i2c.c 中 SDA_Output() 和 SCL_Output() 函数的自变量类型以及 SDA_Input() 函数的返回值类型修改为 “u8”。

(3) 在 D:\Test 中新建工程 test，将下列文件添加到工程中：

- lcd.c
- i2c.c

(4) 在工程中新建添加 main.c 文件，在 main.c 中添加如下内容：

// 包含头文件

```
#include "stm32f10x_lib.h"
```

```
#include "stdio.h"
```

```
#include "lcd.h"
```

```
#include "i2c.h"
```

// 定义符号常量

```
#define FAILURE 0
```

```
#define SUCCESS 1
```

// 定义全局变量

```
u8 para[4];
```

```
s8 wd, wdsx = 40;
```

```
u8 sd, sdsx = 80, cyjg = 1;
```

```
u16 csxh = 1000;
```

```
u8 key = 0, led = 0, lcd = 0;
```

```
u8 hour = 0, min = 0, sec = 0;
```



```

u32 sec0 = 0, sec1 = 0, sec2 = 0;
u8 data[60][5], datnum = 0, datsum = 0;
u8 lcdstr[20];
// 定义结构体
ADC_InitTypeDef ADC_InitStruct;
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
TIM_ICInitTypeDef TIM_ICInitStruct;
TIM_OCInitTypeDef TIM_OCInitStruct;
GPIO_InitTypeDef GPIO_InitStructure;
USART_InitTypeDef USART_InitStructure;
// 声明函数
void RTC_Init(void);
void RTC_Proc(void);
void ADC1_Init(void);
void TIM2_Init(void);
void TIM3_Init(void);
void KEY_Init(void);
void KEY_Proc(void);
void LCD_Proc(void);
void USART2_Init(void);
void USART2_Proc(void);
void LED_Init(void);
void LED_Proc(void);
void RTC_Delay(u32 ms);
int fputc(int ch, FILE *f);
u16 USART_SendData(USART_TypeDef* USARTx, u16 Data);
u16 USART_ReceiveData_NonBlocking(USART_TypeDef* USARTx);
u8 I2C_EE_BufferRead(u8* pBuffer, u8 ReadAddr, u8 NumByteToRead);
u8 I2C_EE_BufferWrite(u8* pBuffer, u8 ReadAddr, u8 NumByteToRead);
// 主函数
int main(void)
{
    RTC_Init();
    ADC1_Init();
    TIM2_Init();
    TIM3_Init();
    KEY_Init();
    STM3210B_LCD_Init();
    LCD_Clear(Blue);
    LCD_SetBackColor(Blue);

```

```

LCD_SetTextColor(White);
USART2_Init();
LED_Init();
i2c_init();
I2C_EE_BufferRead(para, 0, 4);
if(para[2] <= 5)
{
    wdsx = para[0];
    sdsx = para[1];
    cyjg = para[2];
    csxh = para[3]*100;
    TIM_SetAutoreload(TIM2, 110-para[3]);          // 自动装载值
    TIM_SetCompare2(TIM2, (110-para[3]) >> 1); // 匹配值
}
sec1 = RTC_GetCounter();
sec2 = sec1;
// 主循环
while(1)
{
    RTC_Proc();
    KEY_Proc();
    LCD_Proc();
    USART2_Proc();
    LED_Proc();
}
}

```

(5) 将 10.2 中的下列初始化程序：

- RTC_Init()
- ADC1_Init()
- TIM2_Init()
- TIM3_Init()
- KEY_Init()
- USART2_Init()
- LED_Init()

和下列处理程序：

- RTC_Proc()
- KEY_Proc()
- LCD_Proc()
- USART2_Proc()
- LED_Proc()

追加到 main.c 中。

(6) 将 3.4.1 中的下列 USART 发送数据程序和 USART 非阻塞接收数据程序：

- USART_SendData1()
- USART_ReceiveData_NonBlocking()

3.4.3 中的下列 Printf 调用函数：

- fputc()

5.3.4 中的下列 I2C 读写函数：

- I2C_EE_BufferRead()
- I2C_EE_BufferWrite()


和 6.4.3 中的下列延时程序：


- RTC_Delay()


追加到 main.c 中。

(7) 将下列编译库文件复制添加到工程中：

- C:\Keil\ARM\RV31\LIB\ST\STM32F10xR.LIB

(8) 单击生成工具栏中的 Target Option（目标选项）按钮  打开“目标选项”对话框，在 Target(目标)标签中选中 Code Generation(代码生成)下的 Use MicroLIB(使用 MicroLIB)。

(9) 单击生成工具栏中的 Build（生成）按钮 ，生成目标程序文件 test.axf。

(10) 单击生成工具栏中的“Download”（下载）按钮 ，将目标程序文件 test.axf 下载到训练板并运行。