浙江大学 实验报告

课程名称：微机原理及应用

实验名称：实验 5 基于 DS18B20 的测温实验

指导老师：田翔/马永昌

实验类型：设计实验

专业：生物医学工程

姓名：

学号：

日期：2024 年 6 月 13 日

地点：东 1B-416

目录

1. 实验目的和要求

1.1. 实验要求

1.熟悉 MCS-51 教学实验系统硬件结构。

2.编写汇编代码。

3.对实验任务进行认真分析，写出需求分析报告和系统设计报告。

1.2. 实验任务

•阅读 DS18B20 的数据手册，了解其工作原理及时序结构；

•利用 DS18B20 实现测温功能，并将温度值以“123.45C”的形式显示在数码管上。

*可以不保留两位小数，只考虑整数部分。

*DS18B20 的测温范围是-55C 到 125C，只需考虑零上温度。

*本实验在开发板上实现，可以不用 Proteus 仿真。

*通过串口实现 PC 和单片机的通信，在 PC 端打印实时温度。

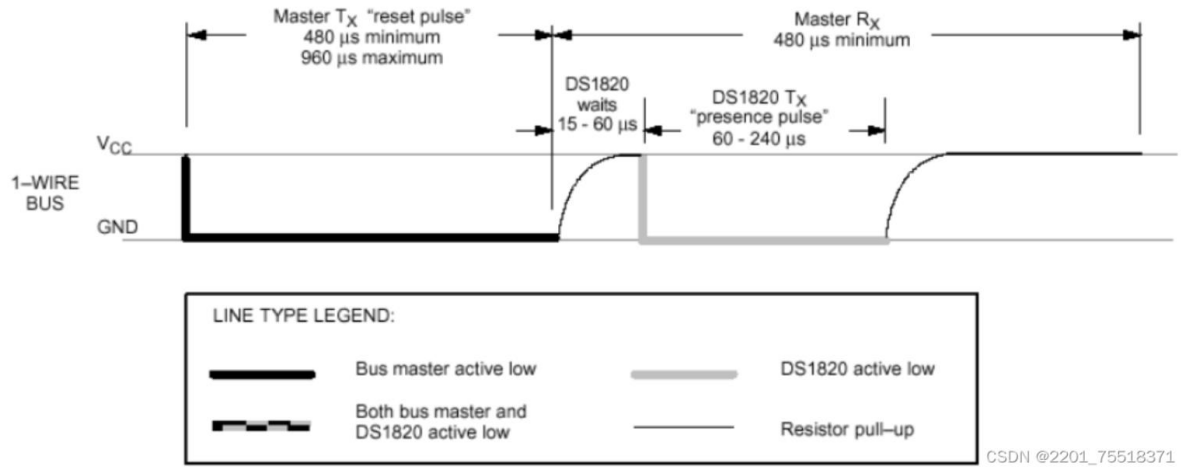
*自行设计和拓展该测温实验。比如，在 PC 端用图形化界面展示实时温度数据，绘制温度随时间变化的曲线，并能够进行一定的交互操作（修改温度的分辨率等)。

2. 实验内容和原理

DS18B20 通信时序

1.初始化序列

DS18B20 的所有通信都由由复位脉冲组成的初始化序列开始。该初始化序列由主机发出，后跟由 DS18B20 发出的存在脉冲（presence pulse）。当发出应答复位脉冲的存在脉冲后，DS18B20 通知主机它在总线上并且准备好操作了。

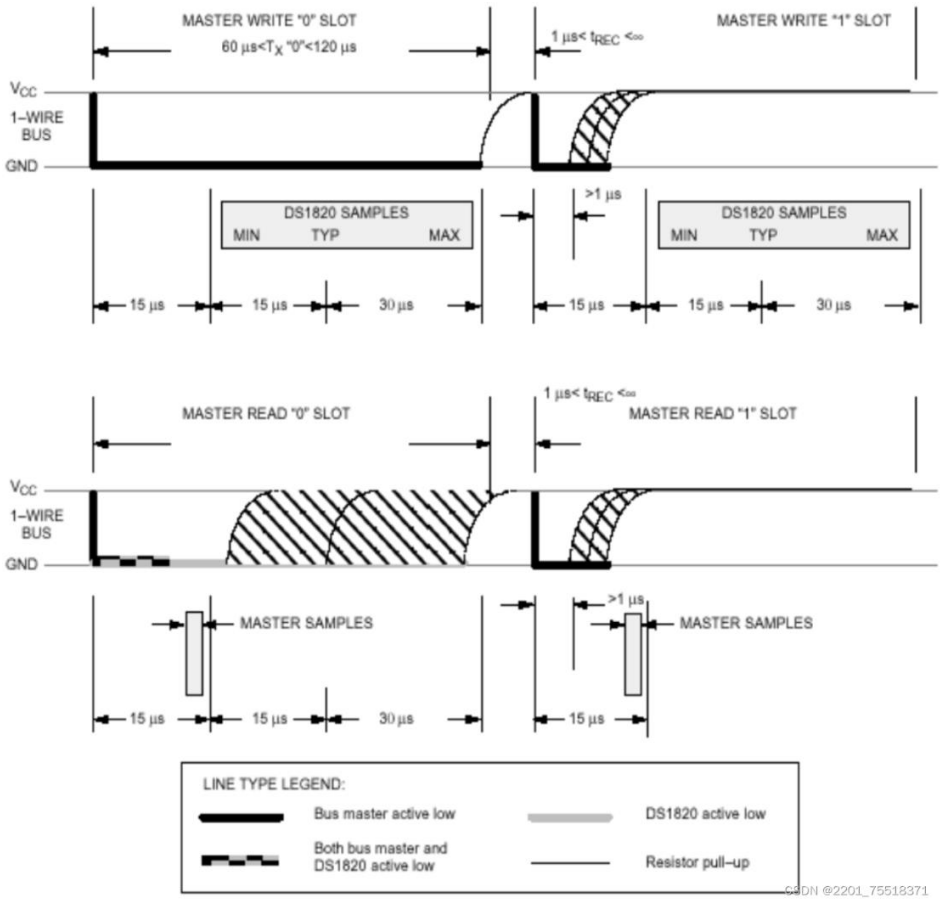


初始化过程“复位和存在脉冲”

在初始化步骤中，总线上的主机通过拉低单总线至少 480μs 来产生复位脉冲。然后总线主机释放总线并进入接收模式。

当总线释放后，5kΩ 的上拉电阻把单总线上的电平拉回高电平。当 DS18B20 检测到上升沿后等待 15 到 60us，然后以拉低总线 60-240us 的方式发出存在脉冲。

2.写时序



读/写时序

读写时隙

主机在写时隙向 DS18B20 写入数据，并在读时隙从 DS18B20 读入数据。在单总线上每个时隙只传送一位数据。

写时间隙

有两种写时隙：写“0”时间隙和写“1”时间隙。总线主机使用写“1”时间隙向 DS18B20 写入逻辑 1，使用写“0”时间隙向 DS18B20 写入逻辑 0。

所有的写时隙必须至少有 60us 的持续时间。相邻两个写时隙必须要有最少 1us 的恢复时间。所有的写时隙（写 0 和写 1）都由拉低总线产生。

为产生写 1 时隙，在拉低总线后主机必须在 15us 内释放总线（拉低的电平要持续至少 1us）。由于上拉电阻的作用，总线电平恢复为高电平，直到完成写时隙。

为产生写 0 时隙，在拉低总线后主机持续拉低总线即可，直到写时隙完成后释放总线（持续时间 60-120us）。

写时隙产生后，DS18B20 会在产生后的 15 到 60us 的时间内采样总线，以此来确定写 0 还是写 1。

3.读时序

读时序

DS18B20 只有在主机发出读时隙后才会向主机发送数据。因此，在发出读暂存器命令 [BEh]或读电源命令[B4h]后，主机必须立即产生读时隙以便 DS18B20 提供所需数据。另外，主机可在发出温度转换命令 T [44h]或 Recall 命令 E 2[B8h]后产生读时隙，以便了解操作的状态(在 DS18B20 操作指令这一节会详细解释)。

读时间隙

指令名称	指令代码	指令功能
温度转换	0x44	启动器件的温度转换，这个转换是需要时间的，准换之后的结果就存在暂存器的 Byte1 和 Byte0 中
读暂存器	0xBE	读暂存器全部 9 个 Byte 的内容
写暂存器	0x4E	往暂存器 Byte4 和 Byte3 写数据，就是设置温度上限和下限
复制暂存器	0x48	将暂存器 Byte4 和 Byte3 的数据复制到 EEPROM 中
重调 EEPROM	0xB8	将 EEPROM 中的内容恢复到暂存器的 Byte4 和 Byte3 中
读供电方式	0xB4	读器件的供电模式，寄生供电时，器件返回 0，外接电源供电时，器件返回 1

所有的读时隙必须至少有 60us 的持续时间。相邻两个读时隙必须要有最少 1us 的恢复时间。所有的读时隙都由拉低总线，持续至少 1us 后再释放总线(由于上拉电阻的作用，总线恢复为高电平)产生。在主机产生读时隙后，DS18B20 开始发送 0 或 1 到总线上。DS18B20 让总线保持高电平的方式发送 1，以拉低总线的方式表示发送 0.当发送 0 的时候，DS18B20 在读时隙的末期将会释放总线，总线将会被上拉电阻拉回高电平(也是总线空闲的状态)。DS18B20 输出的数据在下降沿(下降沿产生读时隙)产生后 15us 后有效。因此，主机释放总线和采样总线等动作要在 15μs 内完成。

DS18B20 操作流程

·初始化:从机复位,主机判断从机是否响应

·ROM 操作:ROM 指令+本指令需要的读写操作

·功能操作:功能指令+本指令需要的读写操作

ROM 操作就是访问 64 位 ROM,功能操作就是访问暂存器

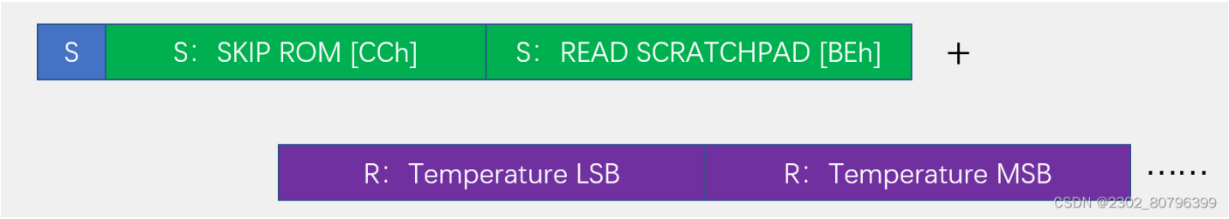
ROM指令	功能指令
SEARCH ROM [F0h]	CONVERT T [44h]
READ ROM [33h]	WRITE SCRATCHPAD [4Eh]
MATCH ROM [55h]	READ SCRATCHPAD [BEh]
SKIP ROM [CCh]	COPY SCRATCHPAD [48h]
ALARM SEARCH [ECh]	RECALL E2 [B8h]
	READ POWER SUPPLY [B4h]

本实验使用 SKIP ROM,因为本开发板从机只有一个 DS18B20,不需要 ROM 来进行寻址

DS18B20 数据帧

· 温度变换:初始化→跳过 ROM→开始温度变换

· 温度读取:初始化→跳过 ROM→读暂存器→连续的读操作



温度存储格式

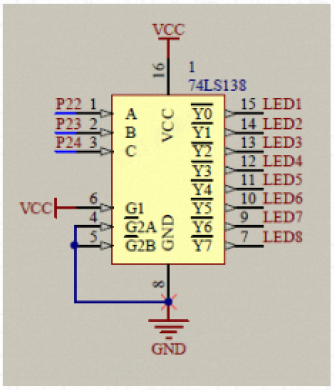
	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

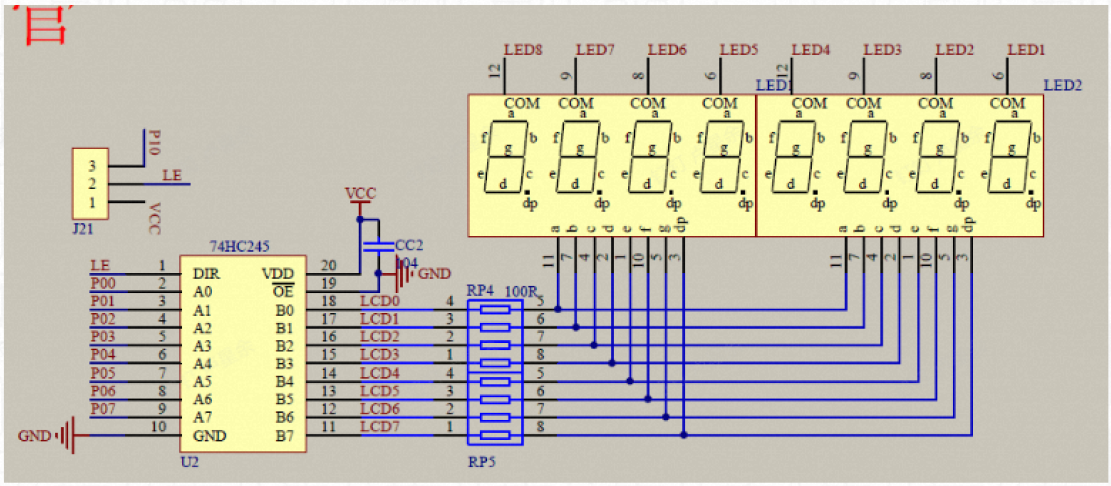
S = SIGN

其中 S 是符号位,温度为负,则 BIT11-15 全为 1,温度为正,则 BIT11-15 全为 0

数码管显示温度

- 动态扫描，回顾实验1。
- 数码管的位选信号由74LS138译码器控制。
- 译码器的地址输入为P22、P23、P24。
- 共阴极数码管。





3. 主要仪器设备

计算机、仿真软件、开发板

4. 操作方法和实验步骤

4.1. 需求分析

即实验任务部分

4.2. 系统设计

· 实验装置以单片机为核心

· 输入控制： PC 端上位机，DS18B20 总线（P3.7）

· 输出控制： DS18B20 总线（P3.7），PC 端上位机

4.3. 硬件设计

系统核心采用 8051 单片机，该单片机在不进行外扩数据存储器和程序存储器的条件下，共有 32 个 I/O 可供使用。功能强，应用面广，价格低。

Proteus 器件选择: AT89C51

4.3.1. 输入

PC 端上位机，DS18B20 总线（P3.7

4.3.2. 输出

DS18B20 总线（P3.7），PC 端上位机

4.4. 软件设计

4.4.1. 硬件资源规划

1. 寄存器分配

累加器 A (ACC): 用于临时存储和处理数据，如温度数据处理和 UART 通信中数据的发送和接收。

寄存器 R0, R1, R2, R3, R4, R5, R6, R7:

R0, R1, R2: 主要用于各类延时循环中的计数。

R3, R4, R5: 未直接使用。

R6, R7: 在数码管显示和温度数据处理中使用。

2. I/O 端口分配

P0: 用于数码管显示，通过该端口输出字符编码。

P1: Debug 使用。

P2.2, P2.3, P2.4: 用于控制数码管显示的位选，以选择显示的数码管。

P3.7 (DSPORT): 用于 DS18B20 温度传感器的数据传输。

3. 内存地址分配

30H, 31H: 用于存储与 DS18B20 通信的数据。

32H, 33H: 分别存储 DS18B20 的温度读取结果的低位和高位。

34H, 35H: 存储处理后的整数和小数部分的温度数据。

36H-40H: 存储处理后的温度数据的百位、十位、个位和小数部分的十位、个位。

41H, 42H: UART 通信中使用，41H 用于发送数据，42H 用于接收数据。

4. 定时/延时

Delay_1us, Delay_100us, Delay_1ms, Delay_100ms 等: 提供稳定的键盘扫描、DS18B20 传感器操作和数码管显示的延时。

5. 键码表 (KEY_TABLE)

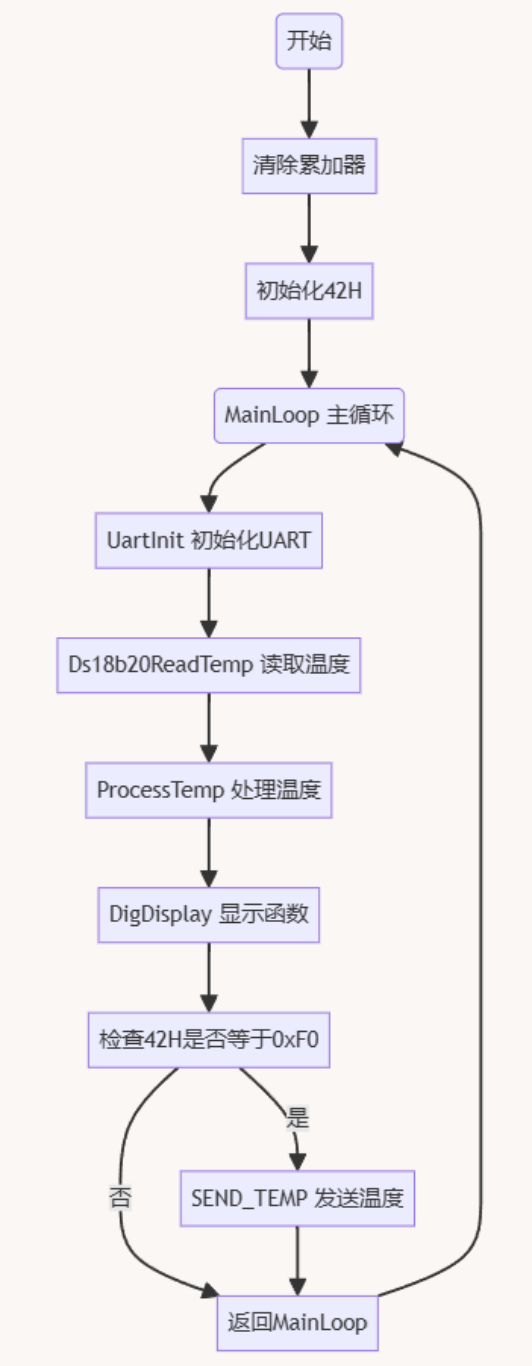
地址: 0x0300H

用途: 存储数码管显示的字符编码，对应不同的键盘输入显示。

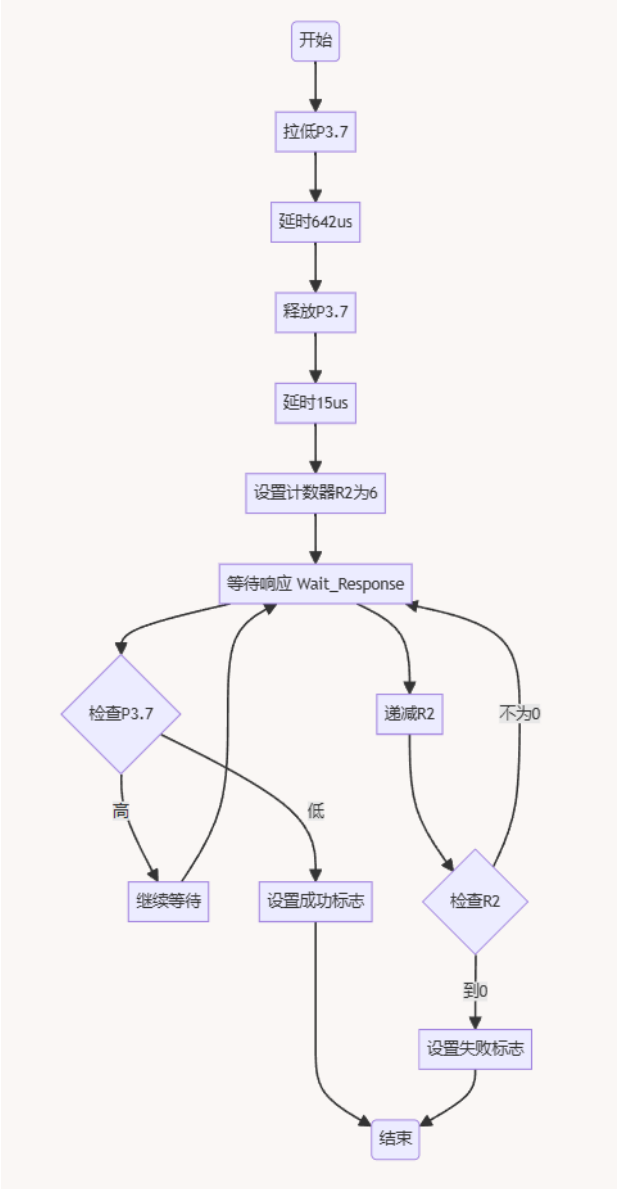
4.4.2. 流程图

下位机流程图

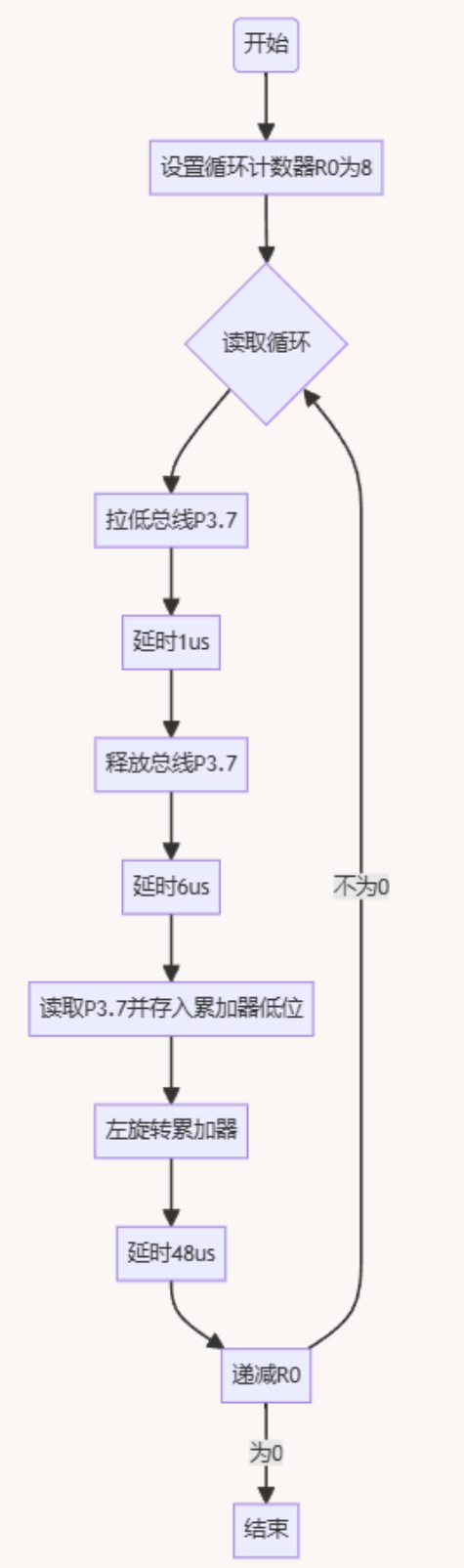
主循环



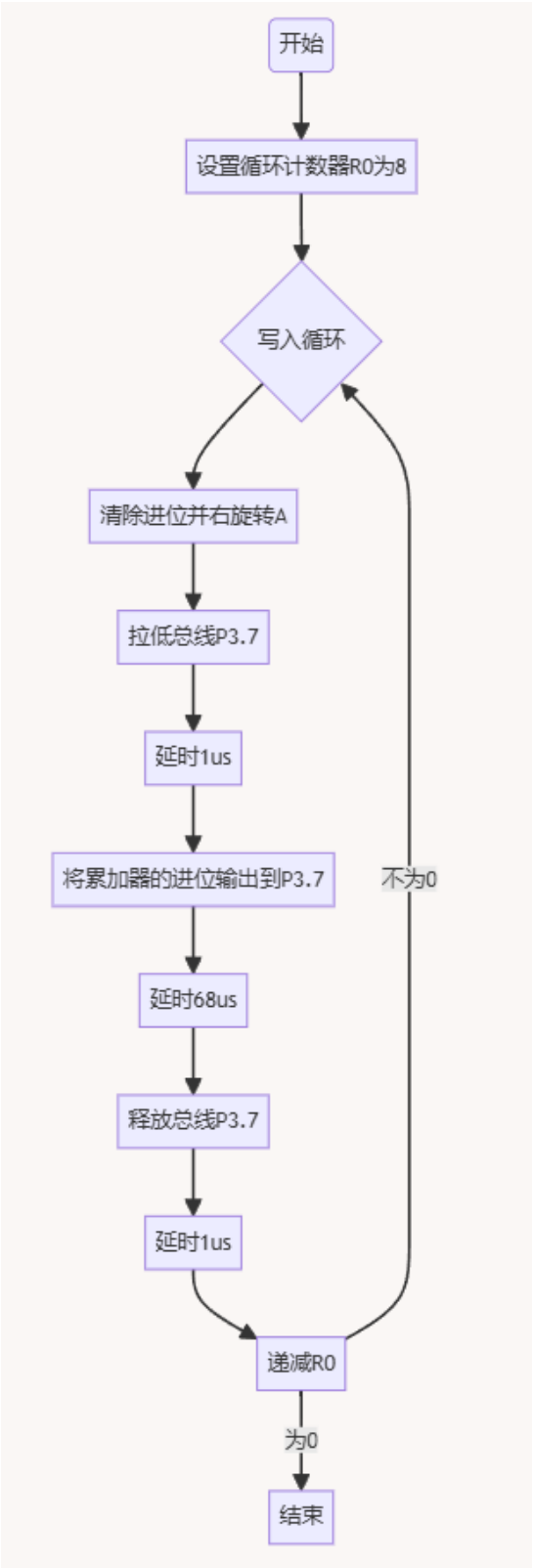
Ds18b20Init 初始化 DS18B20



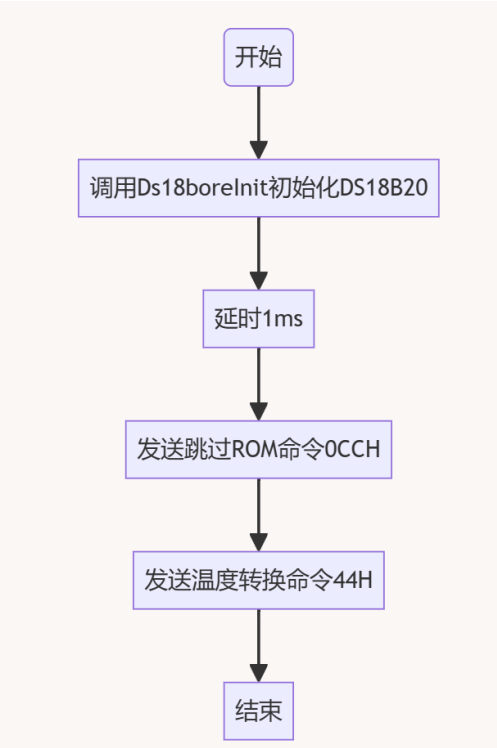
读取字节数据 (Ds18b20ReadByte)



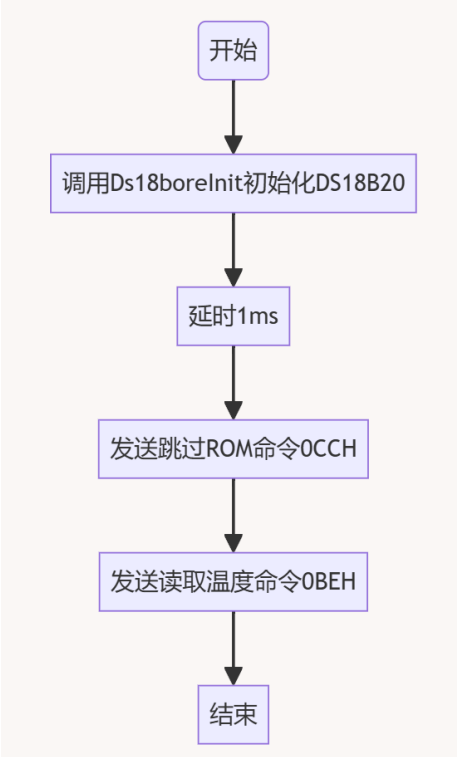
写入字节数据 (Ds18boreWriteByte)



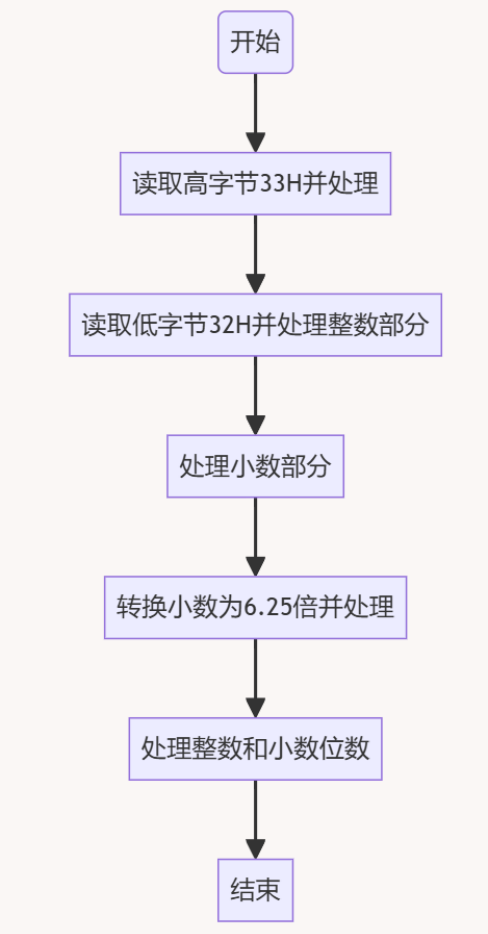
温度转换 (Ds18boreChangeTemp)



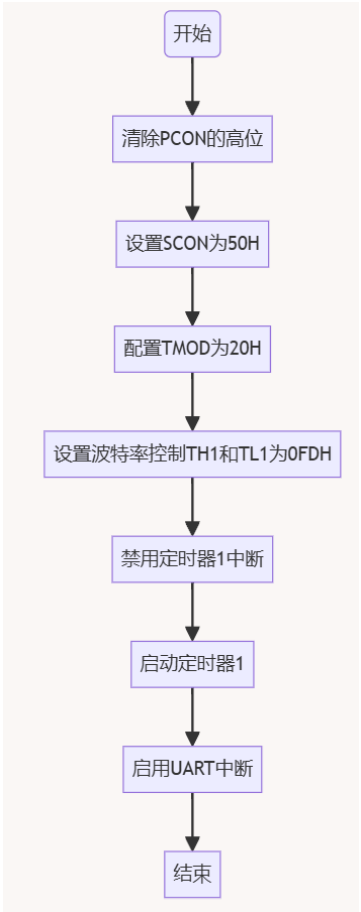
发送读取温度命令 (Ds18b20ReadTempCom)



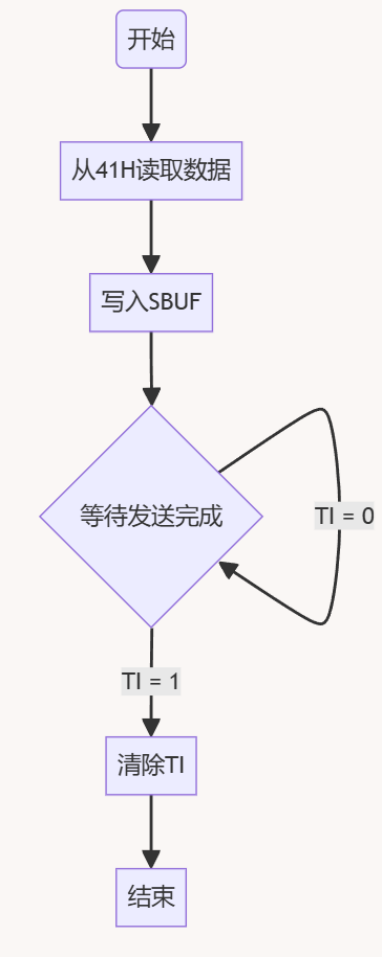
处理温度数据 (ProcessTemp)



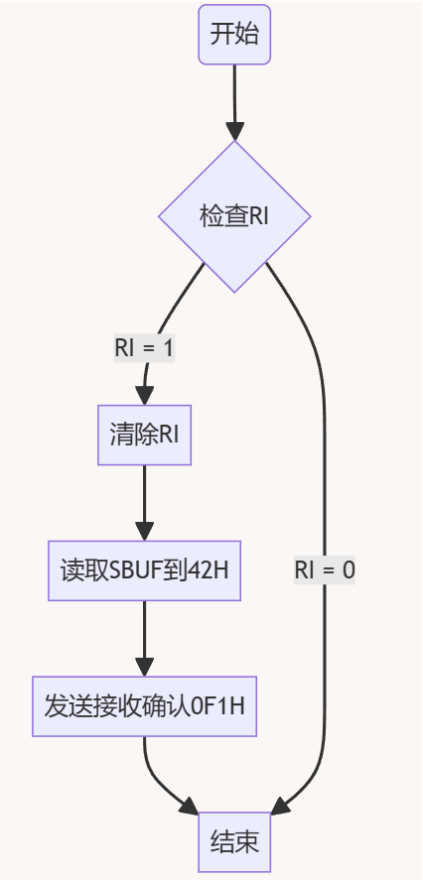
UART 初始化 (UartInit)



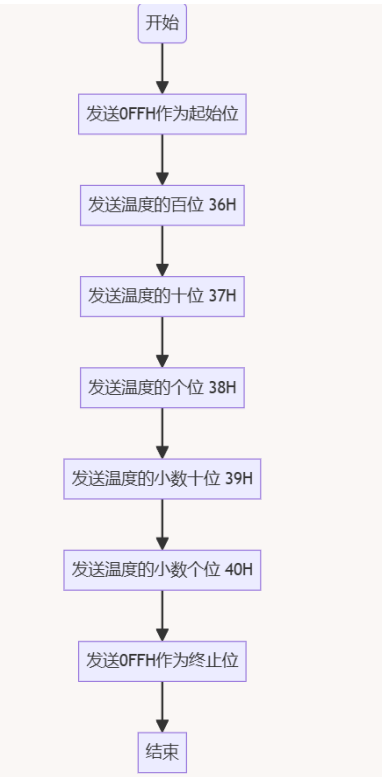
UART 发送字节 (UART_SendByte)



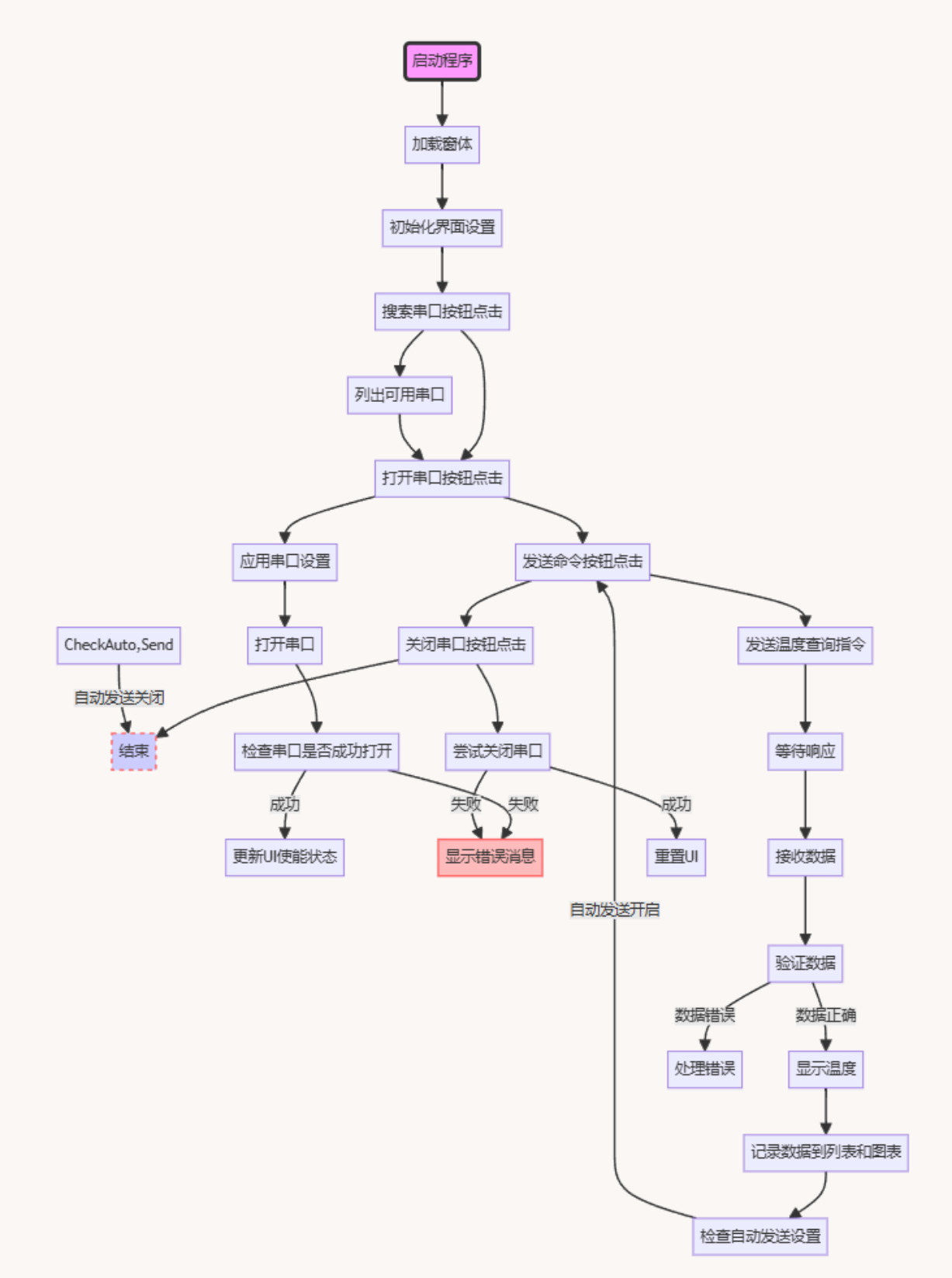
UART 中断服务例程 (UART_ISR)



读取并发送温度数据 (SEND_TEMP)



上位机流程图


启动程序
加载窗体
初始化界面设置
搜索串口按钮点击
列出可用串口
打开串口按钮点击
应用串口设置
发送命令按钮点击
CheckAuto,Send
打开串口
关闭串口按钮点击
发送温度查询指令
自动发送关闭
结束
检查串口是否成功打开
尝试关闭串口
等待响应
成功
失败
失败
成功
更新UI使能状态
显示错误消息
重置UI
接收数据
自动发送开启
验证数据
数据错误
数据正确
处理错误
显示温度
记录数据到列表和图表
检查自动发送设置

4.4.3. 代码

单片机代码

```
ORG 0000H             ; 程序入口点
    SJMP Main
ORG 0023H               ; 中断向量地址为 0023H（中断 4）
    LJMP UART_ISR        ; 跳转到中断服务例程
Main:
    CLR A               ; 清除累加器，用于安全起见
    MOV A,#0FFH
    MOV 42H,A           ;初始化 42H
    CLR A
MainLoop:
    ACALL UartInit      ; 调用初始化函数
    ACALL Ds18b20ReadTemp   ; 调用读取温度的函数
    ACALL ProcessTemp       ; 调用数据处理函数
    ACALL DigDisplay      ; 调用显示函数
    MOV A,42H
    CJNE A, #0F0H, NOT_SEND
;   ACALL Delay_100ms
    ACALL SEND_TEMP
NOT_SEND:
    LJMP MainLoop         ; 无限循环回到开始
;-------------------------------------------------------------------------------
-----
;初始化 DS18B20
Ds18b20Init:
    CLR  P3.7           ; 拉低总线
    ACALL Delay_642us   ; 延时 642us
    SETB P3.7           ; 释放总线
    ACALL Delay_15us    ; 短暂延时等待响应

    MOV  R2, #6         ; 初始化计数器，设置最大等待时间
Wait_Response:
    JB   P3.7, Continue_Wait ; 如果 P3.7 是高，继续等待
    MOV  A, #1          ; 总线被拉低，初始化成功，返回 1
    ;Debug 用，判断是否成功通信，测试通过
    MOV 41H,A
    RET

Continue_Wait:
    ACALL Delay_1ms     ; 延时 1ms
    DJNZ R2, Wait_Response ; 计数器递减，如果未到 0 继续循环
    MOV  A, #0          ; 计数器到 0 还未检测到低电平，初始化失败
    ;Debug 用，判断是否成功通信，测试通过
    MOV 41H,A
    RET
```

```
;-------------------------------------------------------------------------------
-----
; 数据字节存储在 RAM 地址 30H.向 18B20 写入一个字节
Ds18b20WriteByte:
    MOV R0, #8              ; 初始化循环计数器，准备写入 8 位数据
    MOV A, 30H              ; 将地址 30H 处的数据加载到累加器 A

Write_Loop:
    CLR C
    RRC A
    CLR P3.7                ; 拉低总线，准备写入一位数据
    ACALL Delay_1us         ; 延时 1 微秒，实际上是 5 微秒
;    MOV C, ACC.0            ; 将数据的最低位移到进位标志位
    MOV P3.7, C             ; 将进位标志位的值写入 P3.7（DSPORT）
    ACALL Delay_68us        ; 延时 68 微秒
    SETB P3.7               ; 释放总线
    ACALL Delay_1us         ; 延时 1 微秒以供总线恢复

;    RRC A                   ; 右旋转累加器，准备下一个数据位
    DJNZ R0, Write_Loop     ; 循环计数器递减并检查是否为 0，不为 0 则继续循环

    RET                     ; 返回
;-------------------------------------------------------------------------------
----
; 数据字节存储在 RAM 地址 31H.向 18B20 读取一个字节
Ds18b20ReadByte:
    MOV R0, #8              ; 初始化循环计数器，准备读取 8 位数据
    CLR A                   ; 清除 A 寄存器，准备存储数据字节

Read_Loop:
    CLR P3.7                ; 拉低总线
    ACALL Delay_1us         ; 延时 1 微秒
    SETB P3.7               ; 释放总线
    ACALL Delay_1us         ; 等待 6 微秒，数据稳定
    MOV C, P3.7             ; 将 P3.7 的值读取到进位标志位
    RRC A                   ; 左旋转累加器，将进位标志位移入 A 的最低位

    ACALL Delay_48us        ; 延时 48 微秒，准备下一次读取

    DJNZ R0, Read_Loop      ; 循环计数器递减并检查是否为 0，不为 0 则继续循环

    MOV 31H, A              ; 将最终读取的字节存储到 31H
    RET                     ; 返回
;-------------------------------------------------------------------------------
----
;让 18b20 开始转换温度
Ds18b20ChangTemp:
    ACALL Ds18b20Init     ; 调用初始化函数
```

```
    ACALL Delay_1ms        ; 延时 1ms，确保初始化稳定

    ; 发送跳过 ROM 操作命令
    MOV A, #0CCH           ; 将 0xCC 加载到累加器
    MOV 30H, A             ; 将 0xCC 写入 RAM 地址 30H
    ACALL Ds18b20WriteByte ; 调用写字节函数

    ; 发送温度转换命令
    MOV A, #44H            ; 将 0x44 加载到累加器
    MOV 30H, A             ; 将 0x44 写入 RAM 地址 30H
    ACALL Ds18b20WriteByte ; 调用写字节函数
    ; 通常需要等待转换完成，延时 100ms
    ; 如果持续监控，可能不需要此延时
;    ACALL Delay100ms    ; 为转换过程延时
    RET
;-------------------------------------------------------------------------------
----
; 函 数 名: Ds18b20ReadTempCom
; 函数功能: 发送读取温度命令
Ds18b20ReadTempCom:
    ACALL Ds18b20Init      ; 调用初始化函数

    ACALL Delay_1ms        ; 延时 1ms，确保初始化稳定

    ; 发送跳过 ROM 操作命令
    MOV A, #0CCH           ; 将 0xCC 加载到累加器
    MOV 30H, A             ; 将 0xCC 写入 RAM 地址 30H
    ACALL Ds18b20WriteByte ; 调用写字节函数

    ; 发送读取温度命令
    MOV A, #0BEH           ; 将 0xBE 加载到累加器
    MOV 30H, A             ; 将 0xBE 写入 RAM 地址 30H
    ACALL Ds18b20WriteByte ; 调用写字节函数
    RET
;-------------------------------------------------------------------------------
----
; 函 数 名          : Ds18b20ReadTemp
; 函数功能          : 读取温度,低位存储在 32H,高位存储在 33H
; 51 单片机汇编版本的 Ds18b20ReadTemp 函数

Ds18b20ReadTemp:
    ACALL Ds18b20ChangTemp    ; 开始温度转换
    ACALL Ds18b20ReadTempCom ; 发送读取温度命令

    ACALL Ds18b20ReadByte     ; 读取低字节
    MOV A,31H                 ; 将读取的低位字节存储在累加器 A
    MOV 32H, A                ; 将读取的低字节存储在地址 32H

    ACALL Ds18b20ReadByte     ; 读取高字节
```

```
    MOV A,31H                ; 将读取的高位字节存储在累加器 A
    MOV 33H, A               ; 将读取的高字节存储在地址 33H
    RET                      ; 返回
;-------------------------------------------------------------------------------
----
; 函 数 名: ProcessTemp
; 函数功能: 处理温度数据,将温度按照整数和小数部分分别存储在 34H 和 35H
; 假设:
; - 高字节存储在 33H
; - 低字节存储在 32H
; - 结果存储在内存地址 34H-35H，同时按数值储存到 36-40H

ProcessTemp:
    ; 清除高字节中无效的位并移位，将有效位移至低位
    MOV A, 33H  ; 将高字节加载到累加器 A
    ANL A, #07H ; 仅保留高字节的后三位
    SWAP A      ; 交换高低四位
    ANL A, #0F0H ; 清除现在的低四位（原高四位）
    MOV R0,A

    ; 提取低字节的高四位，处理整数部分
    MOV B, 32H  ; 将低字节加载到 B
    ANL B, #0F0H ; 提取低字节的高四位
    MOV A,B
    SWAP A
    MOV B,A
    MOV A,R0
    ORL A, B    ; 将其加入 A 中
    MOV 34H, A  ; 假设此处直接存储整数部分

    ; 处理小数部分，低字节的低四位
    MOV A, 32H  ; 将低字节加载到 A
    ANL A, #0FH ; 提取低字节的低四位
    MOV R0, A   ; 将提取的值存储到 R0
    ; 将提取的值乘以 6.25
    MOV R4,A            ; 将 A 的值复制到 R4
    MOV B,R4            ; 将 A 的值复制到 B
    ADD A, B            ; A = A * 2
    MOV R5, A            ; 将 A 的值复制到 R5,此时 R5 = A * 2
    MOV B, R5            ; 将 R5 的值复制到 B
    ADD A, B            ; A = A * 4 (此时 A 已经是原始值的 4 倍)
    ADD A, B            ; A = A * 4 + B * 2 = A * 6
    MOV R1, A            ; 将 A 的值复制到 R1,此时 R1 = A * 6
    ; 计算原值的 25% (A / 4)
    MOV A, R0           ; 恢复原始值到 A
    CLR C
    RRC A                ; A = A / 2
    CLR C
    RRC A                ; A = A / 4 (此时 A 为原始值的 25%)
```

```
    ; 将25%的结果加到A * 6的结果上
    ADD A, R1            ; A = A * 6 + A * 0.25 = A * 6.25
    MOV 35H, A  ; 存储小数部分
    ; 整数部分abc存储在34H，小数部分ef存储在35H
    ; 处理整数部分abc
    MOV A, 34H           ; 加载整数部分abc
    MOV B, #100          ; 准备除以100以分离百位
    DIV AB
    MOV 36H, A           ; 存储百位数到36H
    MOV A, B
    MOV B, #10           ; 准备除以10以分离十位和个位
    DIV AB               ; A = bc / 10, B = c
    MOV 37H, A           ; 存储十位数到37H
    MOV 38H, B           ; 存储个位数到38H
    ; 处理小数部分ef
    MOV A, 35H           ; 加载小数部分ef
    MOV B, #10           ; 准备除以10以分离十位和个位
    DIV AB               ; A = ef / 10, B = f
    MOV 39H, A           ; 存储小数部分的十位数到39H
    MOV 40H, B           ; 存储小数部分的个位数到40H
    RET
;-------------------------------------------------------------------------------
----
;函数名:DigDisplay()
;函数功能:数码管显示函数
DigDisplay:
    MOV R7, #6           ; 用于控制显示的数字数量

DISPLAY_LOOP:
    MOV DPTR, #KEY_TABLE     ; 设置DPTR指向查找表的开始地址
    ; 设置位选
    MOV A, R7
    CJNE A, #6, NOT_SIXTH
    CLR P2.2
    CLR P2.3
    CLR P2.4
    MOV A,#12
    MOVC A, @A+DPTR        ; 查找字符表
    MOV P0, A              ; 输出到数码管
    SJMP DISPLAYED
NOT_SIXTH:
    CJNE A, #5, NOT_FIFTH
    SETB P2.2
    CLR P2.3
    CLR P2.4
    MOV A,40H
    MOVC A, @A+DPTR        ; 查找字符表
    MOV P0, A              ; 输出到数码管
    SJMP DISPLAYED
```

```
NOT_FIFTH:
    CJNE A, #4, NOT_FOURTH
    CLR P2.2
    SETB P2.3
    CLR P2.4
    MOV A,39H
    MOVC A, @A+DPTR        ; 查找字符表
    MOV P0, A              ; 输出到数码管
    SJMP DISPLAYED
NOT_FOURTH:
    CJNE A, #3, NOT_THIRD
    SETB P2.2
    SETB P2.3
    CLR P2.4
    MOV A,38H
    MOVC A, @A+DPTR        ; 查找字符表
    MOV P0, A              ; 输出到数码管
    ACALL Delay_100us     ; 调用延时
    MOV P0,#0H           ; 清空显示以避免幽灵效应
    MOV A,#80H
    MOV P0, A              ; 输出到数码管
    SJMP DISPLAYED
NOT_THIRD:
    CJNE A, #2, NOT_SECOND
    CLR P2.2
    CLR P2.3
    SETB P2.4
    MOV A,37H
    MOVC A, @A+DPTR        ; 查找字符表
    MOV P0, A              ; 输出到数码管
    SJMP DISPLAYED
NOT_SECOND:
    CJNE A, #1, NOT_FIRST
    SETB P2.2
    CLR P2.3
    SETB P2.4
    MOV A,36H
    MOVC A, @A+DPTR        ; 查找字符表
    MOV P0, A              ; 输出到数码管
    SJMP DISPLAYED
NOT_FIRST:

DISPLAYED:
    ACALL Delay_100us     ; 调用延时
    MOV P0,#0H           ; 清空显示以避免幽灵效应
    DJNZ R7, DISPLAY_LOOP ; 继续显示下一个数字
    RET
;-------------------------------------------------------------------------
----
```

```
; 延时函数
Delay_100us:
    MOV R0, #20
DELAY_100_LOOP:
    ACALL Delay_1us     ; 调用 5 微秒延时 20 次
    DJNZ R0, DELAY_100_LOOP
    RET

; 实现 1ms 延时
Delay_1ms:
    MOV R1, #10
DELAY_1MS_LOOP:
    ACALL Delay_100us   ; 调用 100 微秒延时 10 次
    DJNZ R1, DELAY_1MS_LOOP
    RET


;100ms 的延时函数
Delay_100ms:
    MOV R2, #100
DELAY_100MS_LOOP:
    ACALL Delay_1ms     ; 调用 1 毫秒延时 100 次
    DJNZ R2, DELAY_100MS_LOOP
    RET



; 延时 6 微秒
Delay_6us:
    ; 实际上是 10us
    MOV R1, #2
DELAY_6_LOOP:
    ACALL Delay_1us     ; 调用 5 微秒延时 2 次
    DJNZ R1, DELAY_6_LOOP
    RET


; 延时 48 微秒,实际上是 60，下同
Delay_48us:
    MOV R1, #12
DELAY_48_LOOP:
    ACALL Delay_1us     ; 调用 5 微秒延时 12 次
    DJNZ R1, DELAY_48_LOOP
    RET

; 延时 15 微秒
Delay_15us:
    MOV R0, #3
DELAY_15_LOOP:
```

```
    ACALL Delay_1us     ; 调用 5 微秒延时 3 次
    DJNZ R0, DELAY_15_LOOP
    RET

Delay_642us:
    MOV R2, #70
DELAY_642_LOOP:
    ACALL Delay_6us     ; 调用 10 微秒延时 70 次
    DJNZ R2, DELAY_642_LOOP
    RET

; 延时 1 微秒
Delay_1us:
    ; 实际上是 5us
    RET

; 延时 68 微秒
Delay_68us:
    MOV R1, #14
DELAY_68_LOOP:
    ACALL Delay_1us     ; 调用 5 微秒延时 14 次
    DJNZ R1, DELAY_68_LOOP
    RET
;-------------------------------------------------------------------------------
-------------
; UartInit 函数：在 11.0592MHz 下初始化 UART
UartInit:
    ANL   PCON, #07H
    ; 设置 SCON = 50H（模式 1，REN 启用）
    MOV   SCON, #50H
    ; 设置 TMOD 为 20H，为定时器 1 设置模式 2（自动重载）
    ANL   TMOD, #0FH      ; 清除高 4 位，保留低位不变
    ORL   TMOD, #20H      ; 设置高 4 位为 2（0010），定时器 1 为模式 2
    ; 将 TH1 和 TL1 设置为 FDH，用于波特率生成
    MOV   TH1, #0FDH
    MOV   TL1, #0FDH
    ; 禁用定时器 1 中断
    CLR   ET1
    ; 启动定时器 1
    SETB  TR1
    ; 启用 UART 中断和全局中断
    SETB  ES
    SETB  EA
    ; 函数返回
    RET
;-------------------------------------------------------------------------------
---------------
; UART_SendByte 函数：发送一个字节
; 参数：数据字节存储在内存地址 41H
```

```
UART_SendByte:
    ; 读取内存地址 41H 中的数据
    MOV   A, 41H          ; 将地址 41H 的内容加载到累加器 A
    ; 将数据写入 SBUF 寄存器
    MOV   SBUF, A
    ; 等待发送完成，检测 TI 标志位
Wait_TI:
    JNB   TI, Wait_TI     ; 如果 TI=0，继续等待
    ; 清除 TI 标志位以准备下一次发送
    MOV A,#0
    MOV 41H,A
    CLR   TI
    ; 函数返回
    RET
;-------------------------------------------------------------------------------
----------------
; UART_Routine 中断服务例程：处理接收到的数据并回送
; 使用中断号 4
UART_ISR:
    JB    RI, Received    ; 检查接收中断标志位 RI，如果置位则处理接收
    SJMP  Exit_ISR        ; 否则直接退出中断服务例程
    Received:
    CLR   RI              ; 清除接收中断标志位 RI
    MOV   A, SBUF         ; 从 SBUF 寄存器读取接收到的数据
    MOV   42H, A          ; 将数据存储到内存地址 42H
    MOV   A  ,#0F1H       ;表示成功接受,DEBUG 用
    ; 调用发送函数将成功接受标志数据回送
    ACALL UART_SendByte

Exit_ISR:
    RETI                  ; 从中断返回
;-------------------------------------------------------------------------------
----------------
;SEND_TEMP
SEND_TEMP:
    MOV A,#0FFH   ;初始位
    MOV 41H,A
    ACALL UART_SendByte
    MOV A,#0FFH   ;初始位
    MOV 41H,A
    ACALL UART_SendByte
    MOV A,36H
    MOV 41H,A
    ACALL UART_SendByte
    MOV A,37H
    MOV 41H,A
    ACALL UART_SendByte
    MOV A,38H
    MOV 41H,A
```

```
    ACALL UART_SendByte
    MOV A,39H
    MOV 41H,A
    ACALL UART_SendByte
    MOV A,40H
    MOV 41H,A
    ACALL UART_SendByte
    MOV A,#0FFH   ;截止位
    MOV 41H,A
    ACALL UART_SendByte
    MOV A,#0FFH
    MOV 42H,A          ;初始化 42H
    RET
KEY_TABLE:
    DB
0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71
END                    ; 结束程序
```

上位机代码

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using System.Windows.Forms.DataVisualization.Charting;
using SeeSharpTools.JY.Report;
using SeeSharpTools.JY.ArrayUtility;
using System.Web.UI.WebControls;

namespace lab5pro2
{
    public partial class Form1 : Form
    {
        int count;//定义一个整型 count，用于定时器 1
        ExcelReport excel;
        public Form1()
        {
            InitializeComponent();
            button_closeport.Enabled = false;
            button_sendorder.Enabled = false;
            timer3.Enabled = false;
            timer3.Stop();
```

```
        }

        List<string> temp = new List<string>();
        private int delayTime = 1000;

        private void Form1_Load(object sender, EventArgs e)
        {
            comboBox_port.Text = "COM3";
            comboBox_baud.Text = "9600";
            comboBox_parity.Text = "None";
            comboBox_data.Text = "7";
            comboBox_stop.Text = "One";
            timer1.Stop();//暂停计时
        }

        private void button_searchport_Click(object sender, EventArgs e)//搜索可
用串口
        {
            string[] portname = SerialPort.GetPortNames();//定义一个字符串来获取
串口
            this.comboBox_port.Items.Clear();//清空 comboBox1 中的值
            foreach (string port in portname)//遍历串口
            {
                var serialPort = new SerialPort();//把串口赋给定义的 var 变量
                serialPort.PortName = port;
                serialPort.Open();//打开串口
                this.comboBox_port.Items.Add(port);//打开成功，则添加至下拉框
                serialPort.Close();//关闭串口
            }
        }

private void button_openport_Click(object sender, EventArgs e)//打开所选择的串口
{
    if (serialPort1.IsOpen)//如果串口是打开的
    {
        try
        {
                    serialPort1.Close();//先判断运行之前串口是否打开，若打开则要先
关闭
        }
        catch
        {
            // 可选：处理关闭串口时的异常
        }
    }
    try
    {
                // 应用用户从界面选择的设置
                serialPort1.PortName = comboBox_port.Text; // 设置串口号
```

```
                serialPort1.BaudRate = int.Parse(comboBox_baud.Text); // 设置波特率
                serialPort1.DataBits = int.Parse(comboBox_data.Text); // 设置数据位
                serialPort1.StopBits = (StopBits)Enum.Parse(typeof(StopBits), comboBox_stop.Text); // 设置停止位
                serialPort1.Parity = (Parity)Enum.Parse(typeof(Parity), comboBox_parity.Text); // 设置校验位

                // 打开串口
                serialPort1.Open();

        // 更新界面元素状态
        button_openport.Enabled = false;
        comboBox_port.Enabled = false;
        comboBox_baud.Enabled = false;
        comboBox_parity.Enabled = false;
        comboBox_data.Enabled = false;
        comboBox_stop.Enabled = false;
        button_searchport.Enabled = false;
        button_closeport.Enabled = true;
        button_sendorder.Enabled = true;
    }
    catch (Exception ex) // 捕获所有可能的异常
    {
        MessageBox.Show("串口打开失败: " + ex.Message, "错误");
    }
}


        private void button_closeport_Click(object sender, EventArgs e)//关闭所选择的串口
        {
            try
            {
                serialPort1.Close();
                button_openport.Enabled = true;
                comboBox_port.Enabled = true;
                comboBox_baud.Enabled = true;
                comboBox_parity.Enabled = true;
                comboBox_data.Enabled = true;
                comboBox_stop.Enabled = true;
                button_searchport.Enabled = true;
                button_closeport.Enabled = false;

            }
            catch (Exception err)//一般情况下关闭串口不会出错，加上以防万一
            {
                MessageBox.Show(err.Message);
```

```
            }
        }




        private void button_sendorder_Click_1(object sender, EventArgs e)
        {
           // this.sendAndDisplayHexCommand();
            this.sendCommand();
            button_sendorder.Enabled =false;
            timer1.Start();
            timer2.Start();
        }

        private void timer1_Tick(object sender, EventArgs e)//定时器
        {

        }



        private void timer3_Tick(object sender, EventArgs e)
        {
        }

        private double? lastTemperature = null; // 存储上一次读取的温度
        private double count_t = 0; // 温度次数计数

        private void sendCommand() // 构造发送指令函数
        {
 //           textBox1.Text = "Null"; // 初始化文本框
            if (this.serialPort1.IsOpen)
            {
                try
                {
                    byte[] commandByte = { 0xF0 }; // 直接定义字节命令
                    this.serialPort1.Write(commandByte, 0, commandByte.Length);
                    // 等待足够时间确保数据完整接收
                    System.Threading.Thread.Sleep(500); // 等待 500 毫秒，可调整以
适应具体情况

                    // 检查是否接收到足够的字节
                    if (this.serialPort1.BytesToRead >= 9)
                    {
                        byte[] byteReceive = new byte[9]; // 预期接收 9 个字节
                        this.serialPort1.Read(byteReceive, 0,
byteReceive.Length); // 读取缓冲区中的数据
```

```
                        // 检查帧头和帧尾是否正确
                        if (byteReceive[0] == 0x00 && byteReceive[1] == 0x7F &&
byteReceive[2] == 0x7F && byteReceive[8] == 0x7F)
                        {
                            // 解析温度数据
                            int integerPart = byteReceive[3] * 100 +
byteReceive[4] * 10 + byteReceive[5];
                            double fractionalPart = (byteReceive[6] * 10 +
byteReceive[7]) * 0.01;
                            double temperature = integerPart + fractionalPart;

                            // 检查温度变化
                            if(count_t>2)
                            {
                                if (lastTemperature.HasValue && ((temperature -
lastTemperature.Value) > 50|| (lastTemperature.Value- temperature  ) > 1))
                                {
                                 //   textBox1.Text = "Error4";
                                    return; // 放弃本次读取
                                }
                            }
                            // 更新上次温度记录
                            lastTemperature = temperature;
                            count_t++;
                            string tempString =
temperature.ToString("F2").ToUpper();
                            textBox1.Text = tempString; // 更新文本框显示温度
                            listBox1.Items.Add(tempString); // 添加到列表框
                            listBox1.SelectedIndex = listBox1.Items.Count - 1;
// 自动选择最新的项

                            double[] doubleArray =
listBox1.Items.Cast<string>().Select(s => double.Parse(s)).ToArray();
                            easyChartX1.Plot(doubleArray); // 绘制图表
                        }
                        else
                        {
                            textBox1.Text = "Error1"; // 帧头或帧尾错误
                        }
                    }
                    else
                    {
                        textBox1.Text = "Error2"; // 数据未完全接收
                    }
                }
                catch (Exception er)
                {
                    MessageBox.Show(er.Message); // 异常处理
```

```
                    textBox1.Text = "Error in receiving data";
                }
            }
        }


        private void sendAndDisplayHexCommand()
        {
            textBox1.Text = "Null"; // 初始化文本桉
            if (this.serialPort1.IsOpen)
            {
                try
                {
                    byte[] commandByte = { 0xF0 }; // 定义发送的字节命令
                    this.serialPort1.Write(commandByte, 0, commandByte.Length);
// 发送命令

                    // 等待足够时间确保数据完整接收
                    System.Threading.Thread.Sleep(500); // 等待 200 毫秒，可调整以
适应具体情冇

                    int bytesToRead = this.serialPort1.BytesToRead; // 读取可用
的字节长度
                    if (bytesToRead > 0)
                    {

                        byte[] byteReceive = new byte[9]; // 根据可读的字节长度创
建数组
                        this.serialPort1.Read(byteReceive, 0,
byteReceive.Length); // 读取缓冲区中的数据

                        // 转换接收到的字节到 16 进制字符串
                        var hexString =
BitConverter.ToString(byteReceive).Replace("-", " ");
                        textBox1.Text = hexString; // 显示 16 进制字符串
                    }
                    else
                    {
                        textBox1.Text = "No data received"; // 没有数据接收
                    }
                }
                catch (Exception er)
                {
                    MessageBox.Show(er.Message); // 显示异常信息
                    textBox1.Text = "Error in receiving data"; // 接收数据错误
                }
            }
            else
            {
```

```
                {
                    textBox1.Text = "Port not open"; // 串口未打开
                }
            }
        }


            private void chart1_Click(object sender, EventArgs e)//双击 chart 事
件来显示滑动条
        {

        }


        private void timer2_Tick(object sender, EventArgs e)
        {

        }




        private void easyChartX1_AxisViewChanged(object sender,
SeeSharpTools.JY.GUI.EasyChartXViewEventArgs e)
        {

        }

        private void timer1_Tick_1(object sender, EventArgs e)
        {
            button_sendorder_Click_1(button_sendorder, null);//调用
button_sendorder_Click_1 函数
            if (serialPort1.IsOpen)    //如果串口已经打开
            {
                count++;
            }

        }

        private void button_stop_Click_1(object sender, EventArgs e)
        {
            button_sendorder.Enabled = true;
            timer1.Stop();
            timer3.Stop();
            count_t = 0;
        }

        private void groupBox1_Enter(object sender, EventArgs e)
```

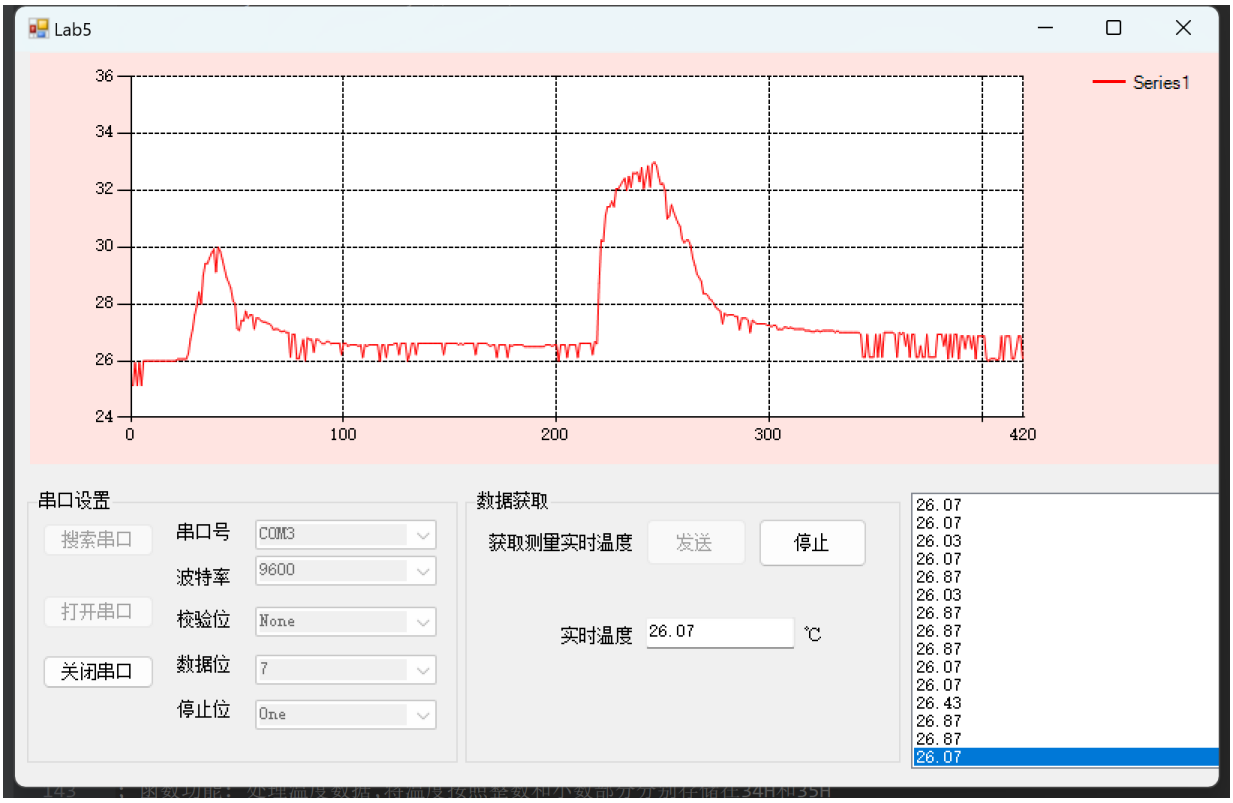
```
        {

        }

        private void comboBox_baud_SelectedIndexChanged(object sender,
EventArgs e)
        {

        }
    }
}
```

5. 实验结果和分析

5.1. 系统测试

见开发板测试，测试正常通过验收

上位机测试如图



5.2. 分析

系统运行结果表明，本实验设计的基于 51 单片机的硬件电路和软件组成的系统，完成了实验任务。

6. 思考和讨论

6.1. DEBUG

在第一版测试时遇到了和 DS18B20 通信失败的问题，通过在仿真中查看波形输出发现是波形的时序存在问题，后排查发现是最基础的延时函数设计出现错误

```
; 延时 1 微秒
Delay_1us:
    ; 实际上是 5us
    RET
```

后对应修正其他延时函数，最后可以正常通信

在调试串口通信以实现和上位机通信的过程中，发现下位机能够正确读取到上位机发送的发送温度指令，但是返回温度值存在有规律的错误（但是上位机可以通过二次处理将其映射为正确温度），经过调试发现是波特率设置问题（目前是 9600），当波特率设置为 62500 时则可正常接收。当然最后由于时间原因没有在上位机和单片机上统一修正，采用原先 9600 方案（不影响温度读取显示）。

6.2. 总结

作为这门课程的最后一个实验，本次实验工作量我认为可以是一个课程大作业的工作量。从零开始写基础温度读取功能加上 DEBUG 大约花费了四小时左右，上位机开发加调试大约花费两小时左右。同时在本次实验代码中我尝试使用了 C 语言当中的函数封装，个人认为这样复用了代码相较于前几次实验的写法可以显著提高效率。当然也学到了许多知识，包括串口通信在内的一些知识内容在课堂上不是很理解的部分在实际操作中得到了理解，总体来说加深了我对于微机原理与应用这门课程的理解。

单片机汇编语言源文件和上位机工程文件已经上传至 Github

LUXINGYU23/51-DS18B20 (github.com)