

温度数据存储与阈值监测设计

一、实验名称

数据传送实验

二、实验目的

通过本次实验掌握 Keil uVision3 开发环境和 C51 编程以及单片机汇编语言源程序编辑、编译、调试方式方法，熟悉 C51 程序结构及单片机汇编语言指令，掌握单片机片内外存储器间数据传送的方法，用 C51 或汇编语言编程实现单片机片内外存储器存取数据。

三、实验原理

本实验基于 MCS-51 单片机（以 AT89C51 为例）的存储结构和 I/O 口特性，利用片内 RAM、片外 RAM 以及端口引脚进行温度数据的生成、处理与结果指示。实验的核心原理如下：

（一）片内 RAM 与外部 RAM 的存取原理

MCS-51 内部含有 128B 低地址 RAM（00H~7FH）以及部分可位寻址区。

对于不同存储空间的访问：C51 用 data/idata/xdata/code 关键字，汇编用 MOV/MOVX/MOVC 指令分别访问。在本程序中，采用 `_at_` 关键字将 `temp[20]` 数组固定存放在片内 RAM 地址 `0x30~0x43`，便于直接访问。

对于片外 RAM 的访问，使用 `xdata` 存储类型说明，并通过指针变量完成 `MOVB` 指令方式的读写。

数据块的源头通常存放在片内 RAM 或程序存储器（code 区），作为原始数据或常量；传送的目的区域一般位于片外 RAM 或 I/O 端口，用于扩展存储或进行人机交互。在本程序中，`temp[20]` 数组作为数据块的源头，放在了片内的 ram 中，经过处理后的数据，例如：进行偏移后的温度值，平均值，最大值，最小值放在了片外 ram 中，同时在处理最大值，最小值和合适温度范围中进行了人机交互。

（二）温度数据的产生与存储

程序中通过简单的公式 $\text{temp}[i] = i + 20 + (i \% 3 - 1)$ 生成 20 组模拟温度数据，范围约在 $19\sim 39^{\circ}\text{C}$ 。

该数据一方面保存在片内数组 `temp[]`，另一方面逐一写入片外 RAM 的起始地址 `0x0000`，实现了数据在片内外存储器之间的传送过程。

（三）数据处理原理

平均值：通过累加 20 组数据，再除以样本数，计算平均温度，并存入外部 RAM `0x2000`。

最大值与最小值：通过逐一比较实现极值搜索，分别存入外部 RAM `0x2001` 和 `0x2002`。这些运算均利用循环结构和 RAM 寻址实现。

（四）阈值与范围判断

单片机 I/O 端口 `P1.0`、`P1.1`、`P1.2` 分别连接三色 LED，用于状态指示。低电平点亮，高电平熄灭。

（五）阈值判断

当 $\text{min} < 25$ 时，点亮蓝灯（`P1.1 = 0`）；

当 $\text{max} > 35$ 时，点亮黄灯（`P1.0 = 0`）。

（六）范围判断

当同时满足 $\text{min} \leq 25$ 且 $\text{max} \geq 35$ 时，点亮红灯（`P1.2 = 0`）。

（七）寄存器与端口状态变化

在运行过程中，片内 RAM（`0x30~0x43`）会存放温度数据；

片外 RAM（`0x0000~0x000F` 等区域）则反映数据传送结果；

指定的 RAM 地址（`0x2000~0x2002`）存放平均值、最大值、最小值。

I/O 口 `P1` 的低 3 位状态随数据判断条件而变化，从而直观反映温度是否越界。

四、实验设计

（一）实验内容

温度数据存储与显示系统设计：

- 1、采集温度数据并暂存于片内 RAM；
- 2、批量传送至片外 RAM 用于长期保存；
- 3、将关键值输出到 I/O 口（LED 显示/报警）；
- 4、在传送过程中进行简单算术与逻辑运算；

移位运算：温度数据读取后统一加上一个偏移值（例如：+1 或 -1），模拟传感器误差修正；

平均值：多次读取温度值后，取平均值再存入片外 RAM；

最大值，最小值：在片内 RAM 保存一批数据后，找出最大或最小值，作为“关键数据”输出到 I/O 口；

阈值判断：如果温度大于设定阈值，输出报警信号到 LED；

逻辑判断：判断温度是否在合理范围。

（二）实验硬件电路

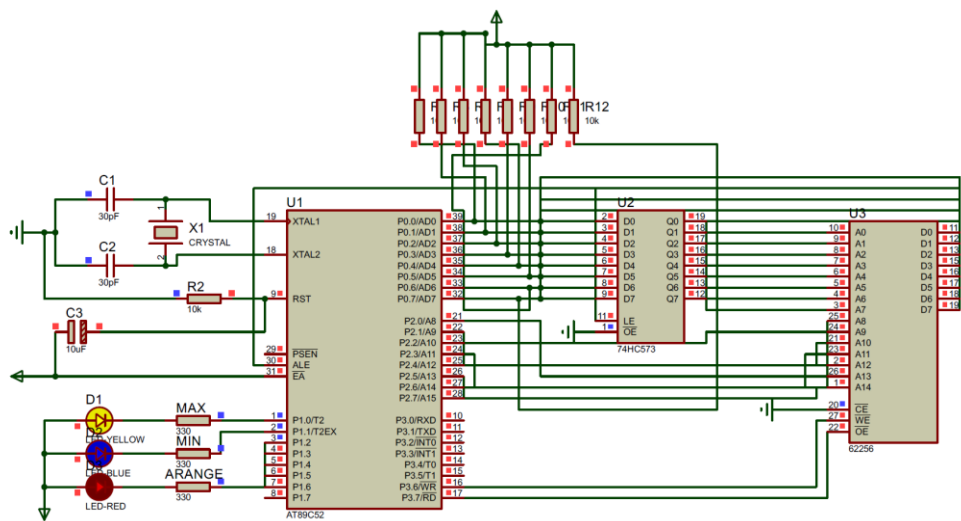


图 4-2-1 仿真电路图

（三）实验软件程序思路及程序流程图

（1）实验程序设计思路

本实验的目的是通过 C51 单片机的 C 语言编程，掌握 绝对地址访问的三种方式：指针访问、SFR 预定义宏访问以及 `_at_` 关键字固定地址访问。程序以温度数据处理为应用背景，演示了数据的生成、存储、运算以及 LED 指示灯的控制。

（2）程序功能说明

1、数据生成与存储

程序在片内 RAM 中定义了 `temp[20]` 数组，存放 20 个伪随机温度数据（20~39 之间，带小幅偏移）。同时，这些数据被写入片外 RAM 0x0000 开始的区域。

2、数据处理

平均值：计算 `temp[20]` 的平均值，并写入片外 RAM 0x2000。

最大值：求得温度最大值，并写入片外 RAM 0x2001。

最小值：求得温度最小值，并写入片外 RAM 0x2002。

3、结果显示

当 最小值 < 25 时，点亮蓝色 LED。

当 最大值 > 35 时，点亮黄色 LED。

当同时满足 最小值 ≤ 25 且 最大值 ≥ 35 时，点亮红色 LED。

(3) 三种绝对地址访问方式的应用

功能	地址访问方式	理由
片内 RAM 缓存温度	_at_	明确固定地址，便于存取
批量传送至片外 RAM	指针	指针便于遍历、批量搬运
I/O 口输出 LED	C51 预定义宏	P1 已是 SFR 宏定义，直接使用
移位运算修正	_at_	在片内固定地址数据上加偏移
平均值计算	指针	遍历数据求和，指针更方便
最大/最小值	指针	遍历数据比较，指针最合适
阈值判断	宏方式	直接控制 P1 输出
合理范围判断	宏方式	直接用逻辑判断操作 P1

(4) 程序整体流程图

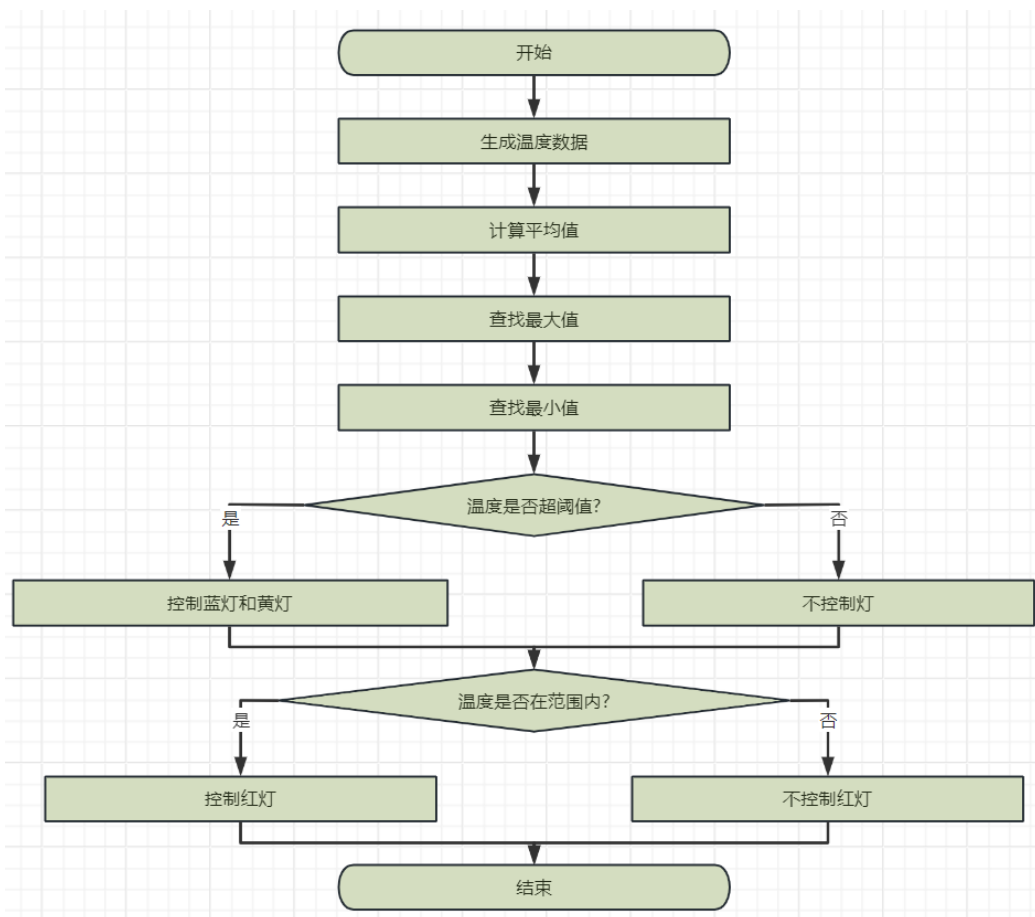
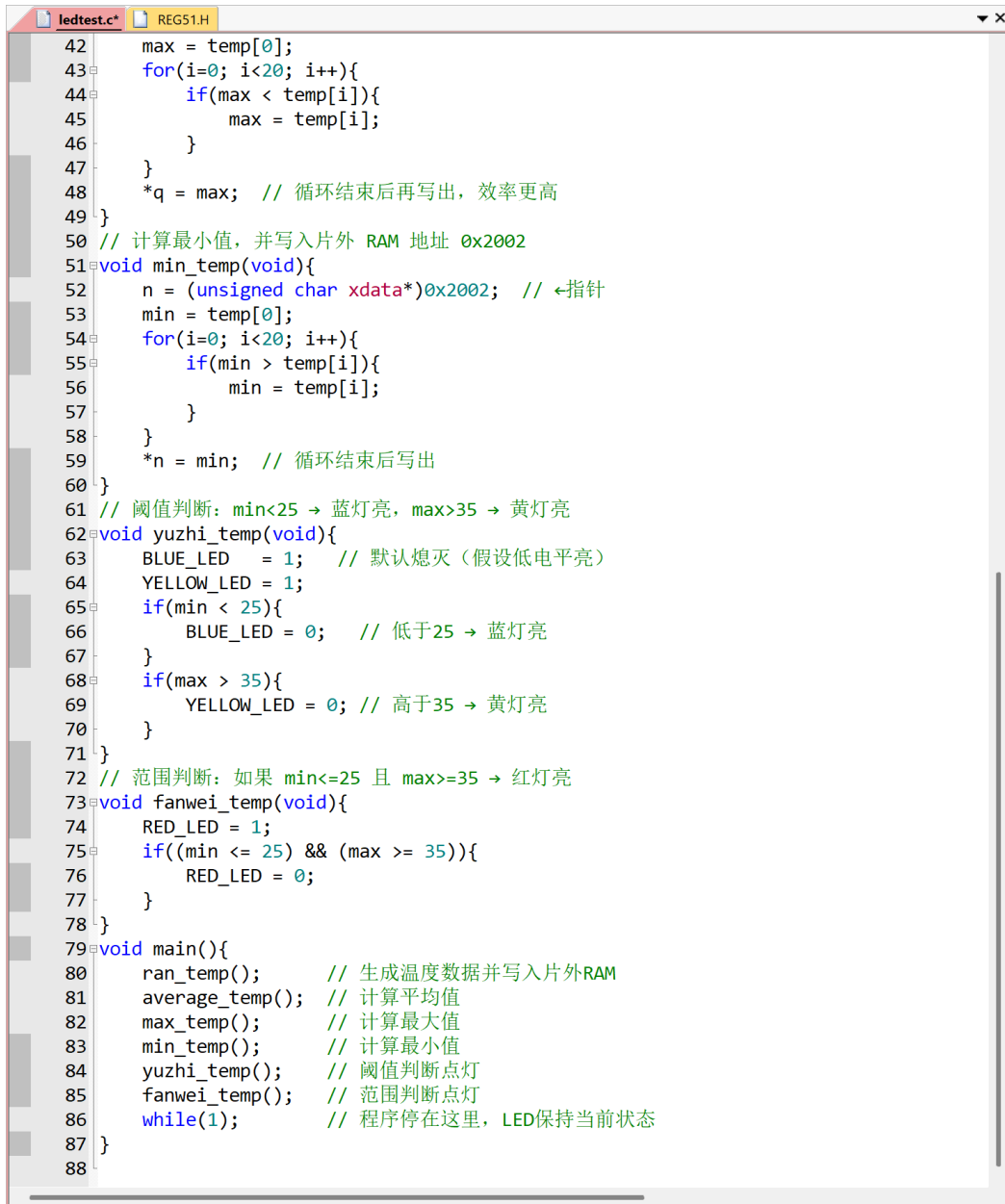


图 4-3-1 程序流程图

(四) 实验程序

```
ledtest.c* REG51.H
1 #include <REG51.H> // 包含C51运行库, 里面有P0、P1、P2、P3等SFR预定义宏
2
3 // 定义单个引脚控制 (LED)
4 sbit P1_0 = P1^0; // 对应 P1.0 ←预定义宏 (SFR位寻址)
5 sbit P1_1 = P1^1; // 对应 P1.1
6 sbit P1_2 = P1^2; // 对应 P1.2
7
8 #define YELLOW_LED P1_0
9 #define BLUE_LED P1_1
10 #define RED_LED P1_2
11 // 把temp数组固定放在片内RAM地址0x30起始位置 ←_at_ 方式
12 unsigned char temp[20] _at_ 0x30;
13 unsigned char i, aver, max, min;
14 unsigned int sum;
15 // 定义片外RAM指针 (xdata表示访问MOVX外部数据存储器) ←指针方式
16 unsigned char xdata *p;
17 unsigned char xdata *m;
18 unsigned char xdata *q;
19 unsigned char xdata *n;
20 // 生成一组伪随机温度数据, 存放到片内 temp[20], 同时写到片外 RAM 0x0000 开始
21 void ran_temp(void){
22     m = (unsigned char xdata*)0x0000; // 指向外部RAM 0x0000 ←指针
23     for(i=0; i<20; i++){
24         temp[i] = i + 20; // 基础数值 20~39
25         temp[i] = temp[i] + (i % 3 - 1); // 偏移
26         *(m+i) = temp[i]; // 写到片外RAM 0x0000+i ←指针
27     }
28 }
29 // 计算平均值, 并写入片外 RAM 地址 0x2000
30 void average_temp(void){
31     sum = 0;
32     for(i=0; i<20; i++){
33         sum += temp[i];
34     }
35     aver = sum / 20;
36     p = (unsigned char xdata*)0x2000; // 指针指向外部RAM 0x2000 ←指针
37     *p = aver; // 存放平均值
38 }
39 // 计算最大值, 并写入片外 RAM 地址 0x2001
40 void max_temp(void){
41     q = (unsigned char xdata*)0x2001; // ←指针
42     max = temp[0];
43     for(i=0; i<20; i++){
44         if(max < temp[i]){
45             max = temp[i];
46         }
47     }
48 }
```

图 4-4-1 代码 (1)



```
42     max = temp[0];
43     for(i=0; i<20; i++){
44         if(max < temp[i]){
45             max = temp[i];
46         }
47     }
48     *q = max; // 循环结束后再写出，效率更高
49 }
50 // 计算最小值，并写入片外 RAM 地址 0x2002
51 void min_temp(void){
52     n = (unsigned char xdata*)0x2002; // ←指针
53     min = temp[0];
54     for(i=0; i<20; i++){
55         if(min > temp[i]){
56             min = temp[i];
57         }
58     }
59     *n = min; // 循环结束后写出
60 }
61 // 阈值判断: min<25 → 蓝灯亮, max>35 → 黄灯亮
62 void yuzhi_temp(void){
63     BLUE_LED = 1; // 默认熄灭（假设低电平亮）
64     YELLOW_LED = 1;
65     if(min < 25){
66         BLUE_LED = 0; // 低于25 → 蓝灯亮
67     }
68     if(max > 35){
69         YELLOW_LED = 0; // 高于35 → 黄灯亮
70     }
71 }
72 // 范围判断: 如果 min<=25 且 max>=35 → 红灯亮
73 void fanwei_temp(void){
74     RED_LED = 1;
75     if((min <= 25) && (max >= 35)){
76         RED_LED = 0;
77     }
78 }
79 void main(){
80     ran_temp(); // 生成温度数据并写入片外RAM
81     average_temp(); // 计算平均值
82     max_temp(); // 计算最大值
83     min_temp(); // 计算最小值
84     yuzhi_temp(); // 阈值判断点灯
85     fanwei_temp(); // 范围判断点灯
86     while(1); // 程序停在这里，LED保持当前状态
87 }
88
```

图 4-4-2 代码（2）

（五）调试过程及实验结果

本实验在 Keil μ Vision + Proteus 环境下进行，采用单步调试与存储器窗口观测的方式，逐步验证各个函数功能是否实现正确。

（1）ran_temp() 函数调试

1) 调试方法

程序运行到 ran_temp(), 打开 片内 RAM Memory 窗口，观察 0x30~0x43 的存储单元内容。

同时，打开 片外 RAM Memory 窗口，观察 0x0000~0x0013 地址内容。

2) 观测结果

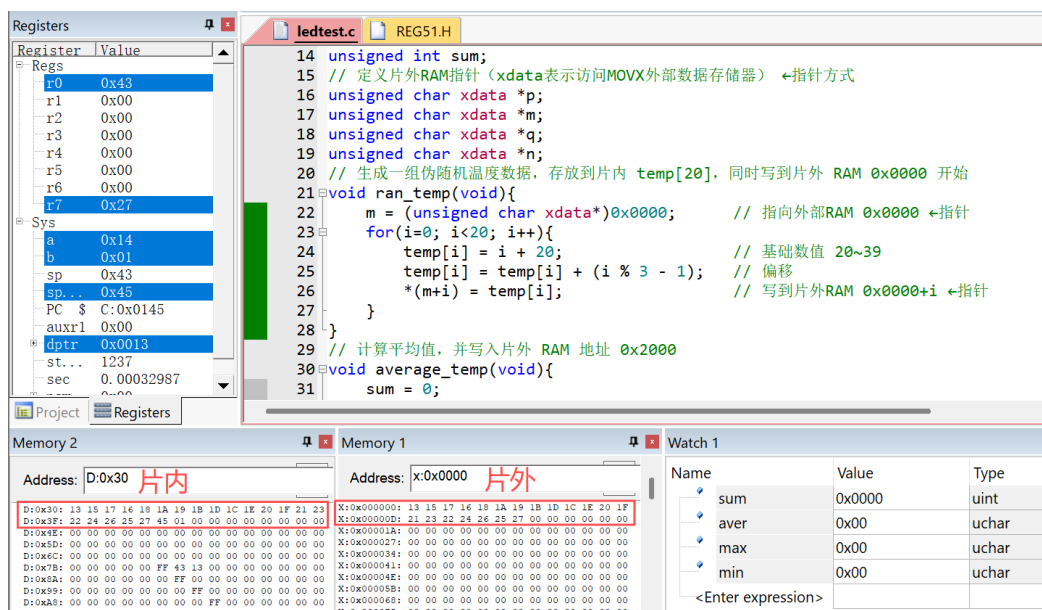


图 4-5-1 函数调试界面

片内 RAM 0x30~0x43 区域成功存放 20 个温度数据，数据范围在 19~40。

片外 RAM 0x0000~0x0013 与片内数组内容完全一致，说明数据传送正确。

3) 实验现象总结

ran_temp() 函数实现了温度数据的生成与片内→片外 RAM 的传送，验证了指针方式访问片外 RAM 的正确性。

(2) average_temp() 函数调试

1) 调试方法

单步执行 average_temp(), 在 寄存器窗口 观察变量 sum 和 aver 的变化。

在 片外 RAM Memory 窗口 观察 0x2000 地址的内容。

2) 观测结果

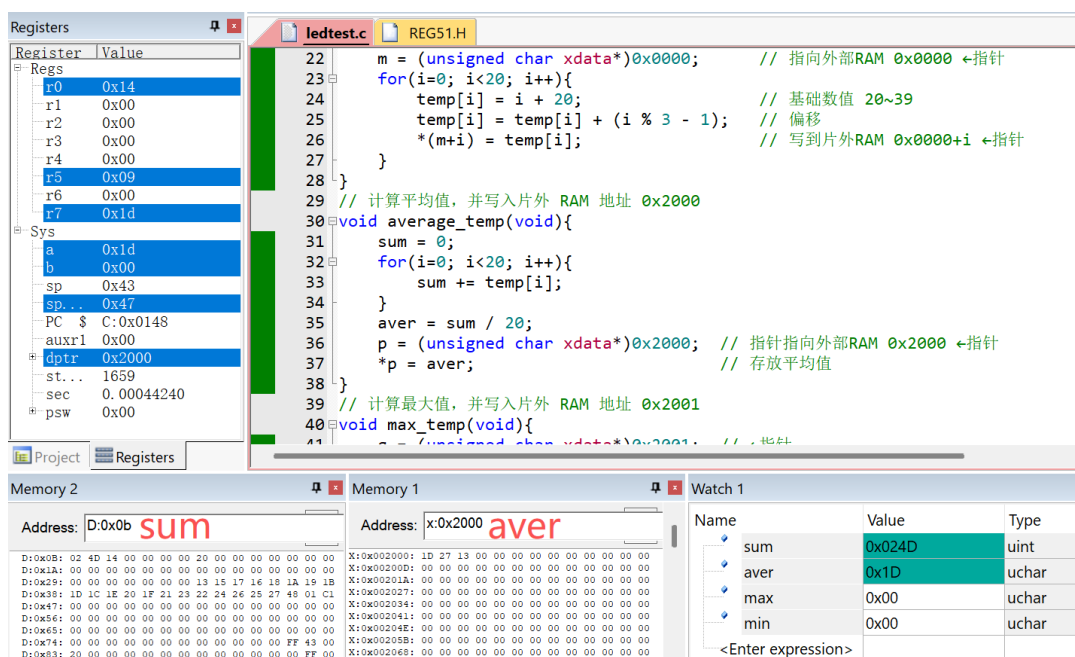


图 4-5-2 函数调试界面

从调试页面可以看出，sum 正确存储了数值，并且 aver 的地址得到了运算数值，片外 RAM 0x2000 地址正确存储了平均值。

3) 实验现象总结

average_temp() 函数实现了对温度数据的平均值计算，并通过指针将结果写入外部 RAM 指定地址。

(3) max_temp() 函数调试

1) 调试方法

单步执行 max_temp(), 在 Watch 窗口观察 max 变量的变化过程。

在片外 RAM Memory 窗口观察 0x2001 地址。

2) 观测结果

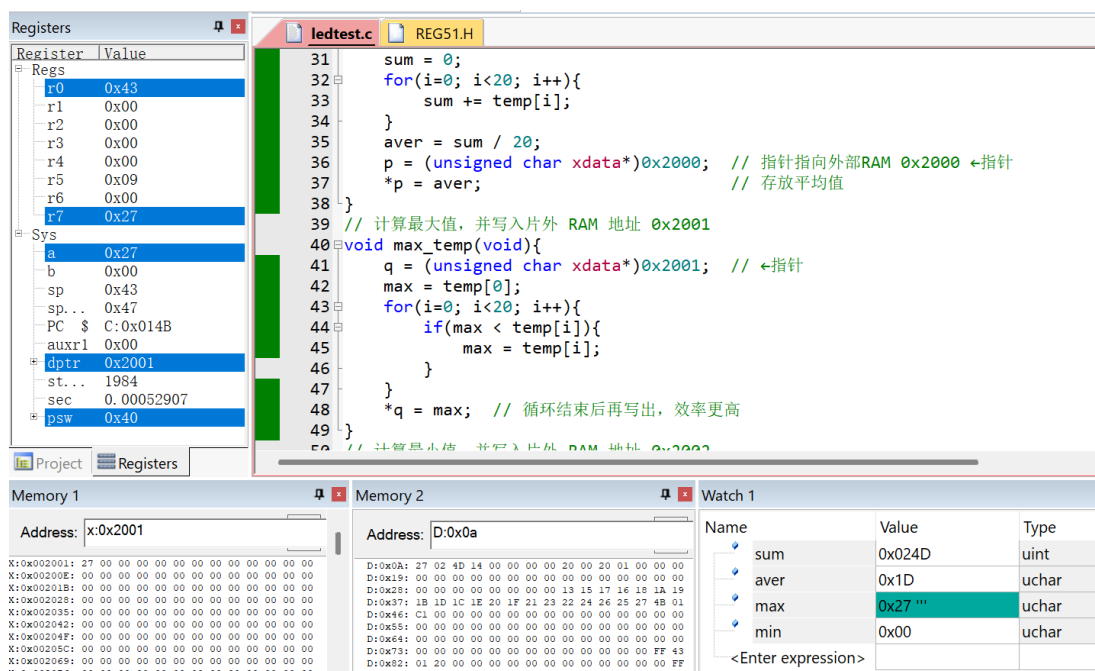


图 4-5-3 函数调试界面

程序循环过程中，max 逐步更新为当前的最大值。循环结束后，max 的值为 20 个温度数据中的最大值 0x29，片外 RAM 0x2001 地址写入该最大值。

3) 实验现象总结

max_temp() 函数实现了最大值的搜索与存储，验证了数据比较与外部 RAM 写操作的正确性。

(4) min_temp() 函数调试

1) 调试方法

单步执行 min_temp()，在 Watch 窗口观察 min 变量的变化过程。在片外 RAM Memory 窗口观察 0x2002 地址。

2) 观测结果

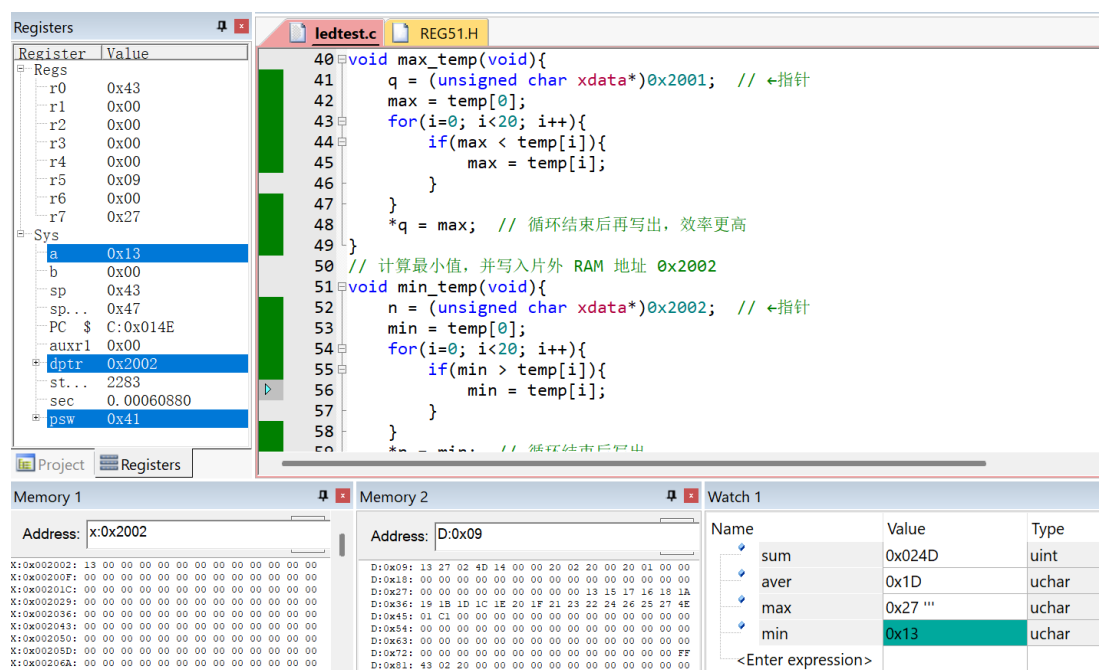


图 4-5-4 函数调试界面

程序循环过程中，min 逐步更新为当前的最小值。循环结束后，min 的值为 20 个温度数据中的最小值 0x13，片外 RAM 0x2002 地址写入该最小值。

3) 实验现象总结

min_temp() 函数实现了最小值的搜索与存储，程序功能正确。

(5) yuzhi_temp() 函数调试

1) 调试方法

在函数执行前，将 P1.0、P1.1 引脚添加到仿真观察中。单步执行 yuzhi_temp()，观察端口电平变化。在 Proteus 中观察对应 LED 的亮灭状态。

2) 观测结果

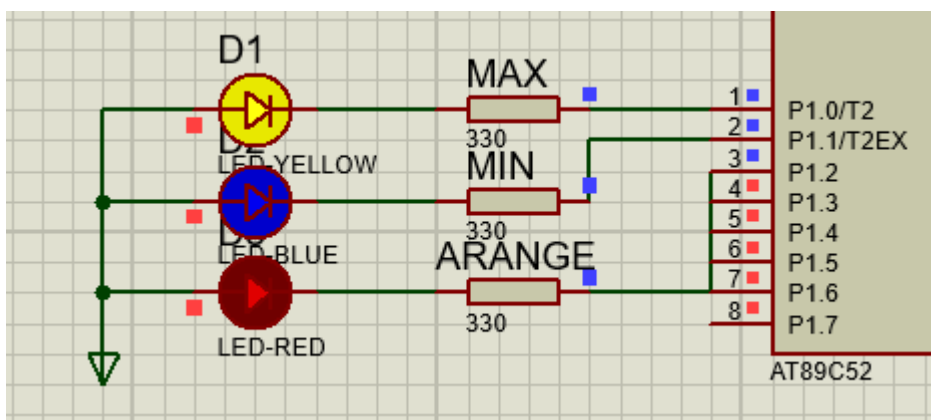


图 4-5-5 仿真界面

当 $\text{min} < 25$ 时，P1.1 输出低电平，蓝色 LED 点亮。

当 $\max > 35$ 时，P1.0 输出低电平，黄色 LED 点亮。

3) 实验现象总结

yuzhi_temp() 函数实现了基于阈值的指示灯控制，验证了 SFR 宏定义访问方式 的有效性。

(6) fanwei_temp() 函数调试

1) 调试方法

在函数执行前，将 P1.2 引脚添加到仿真观察中。

单步执行 fanwei_temp()，观察端口电平变化及 LED 状态。

2) 观测结果

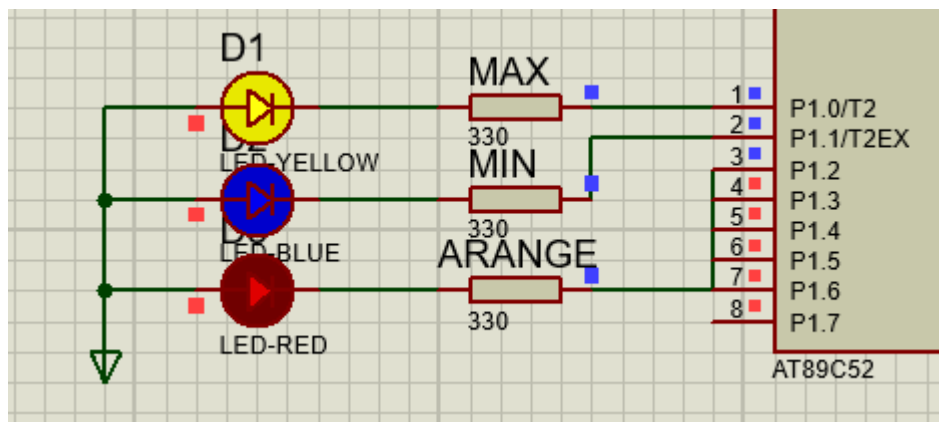


图 4-5-6 仿真界面

当条件 ($\min \leq 25$ 且 $\max \geq 35$) 成立时，P1.2 输出低电平，红灯点亮。

3) 实验现象总结

fanwei_temp() 函数正确实现了范围判断与 LED 控制，保证了温度数据在特定范围下的提示功能。

(7) 主函数整体调试

1) 调试方法

程序顺序执行 ran_temp()、average_temp()、max_temp()、min_temp()、yuzhi_temp()、fanwei_temp()。

最终在 while(1) 停止，LED 状态保持稳定。

2) 实验结果

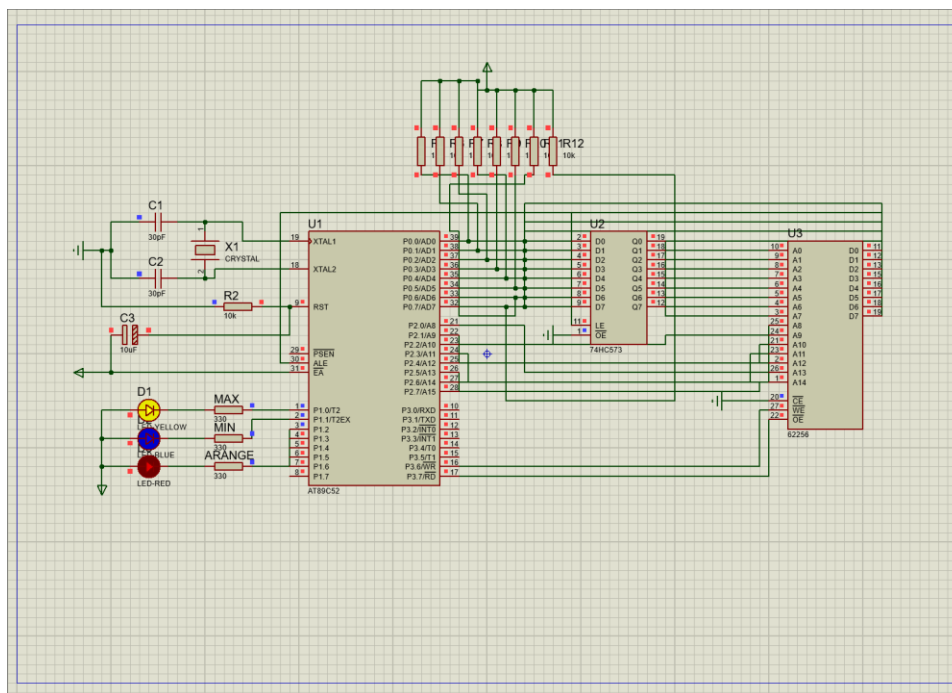


图 4-5-7 仿真运行界面

片内 RAM、片外 RAM 中数据均与程序设计一致。平均值、最大值、最小值存储正确。LED 状态与阈值逻辑完全符合预期。

五、实验总结

本次实验主要完成了 AT89C52 单片机与片外 RAM 的连接与数据存取。通过程序实现了温度数据的生成、平均值、最大值、最小值的计算，并将结果分别存入指定的片外 RAM 地址。同时，利用 P1.0、P1.1、P1.2 接三色 LED，对温度范围进行判断并显示。

在调试过程中，发现若 P0 口未加上拉电阻，片外 RAM 中数据会显示为全 FF；另外 LED 的接法也需要注意，单片机输出低电平时 LED 才会点亮。通过逐步修改电路与程序，最终实现了数据正确写入外部 RAM，LED 指示灯能根据阈值正确显示。

通过本实验，我加深了对单片机片内/片外存储器访问方式的理解，掌握了 74HC573 地址锁存器的作用和电路连接方法，同时也学会了在 Proteus 的 Memory 窗口观察调试数据。这对后续更复杂的单片机实验有了较好的实践基础。