《微机原理与接口技术》
实验指导书

基于 STC15 原厂实验箱

V1.0

编写人：赵文义、刘　柳

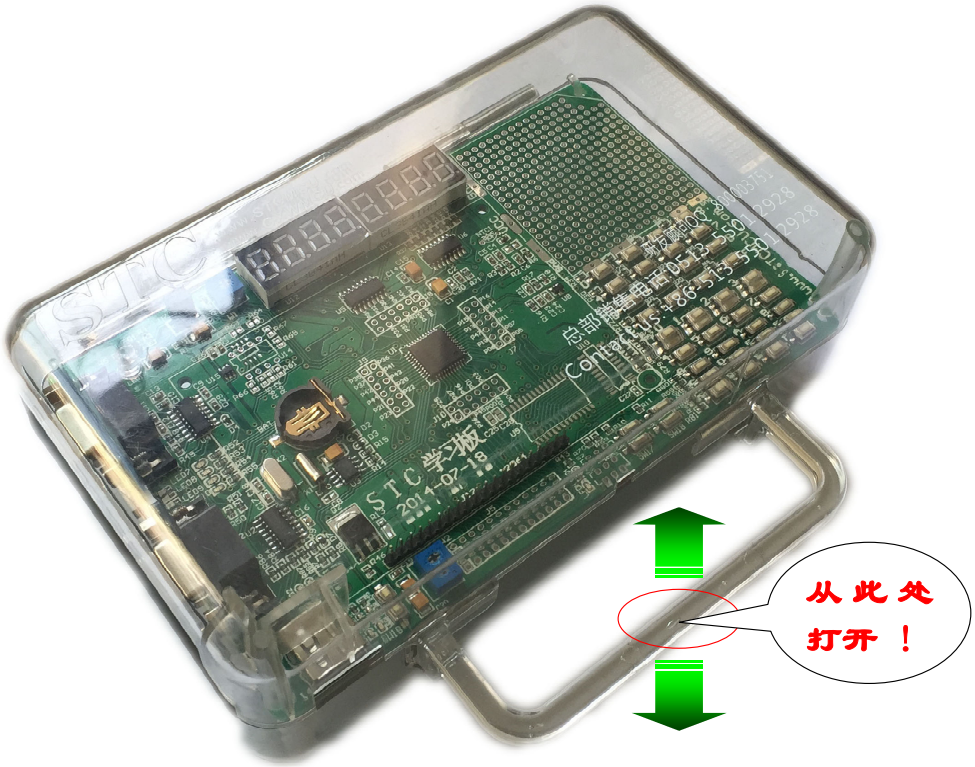
梁宜勇、王立强

浙江大学光电学院

2022.5.1

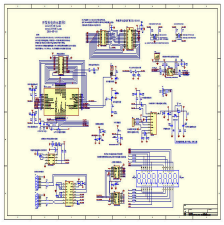
!!! 注意事项!!!

1. 实验箱壳体为亚克力制品，经不住摔碰、挤压、刮擦，设备损坏无处购买补充，请小心保护。
2. 使用中尽量不要将板子从实验箱中拿出来，板子的安装卡脚及固定脚套容易折断，板子没有支撑，放在金属表面或碰到金属物容易产生短路。
3. USB 双公头线连接电脑时应注意连接到可靠耐用的 USB 口上，程序下载和调试时不要插拔 USB 线。



实验相关软件和资料：

1. 集成开发环境：Keil uvision for C51
2. 烧录和调试软件：STC-ISP
3. USB 串口驱动程序：CH340
4. 芯片手册：STC15.pdf
5. 实验箱原理图：STC-STUDY-4.pdf
6. 参考例程：STC-开发板 4-程序
7. 参考书：



《STC 单片机原理及应用》，何宾，清华大学出版社

《单片微机原理与接口技术》，丁向荣，电子工业出版社

8. STC 官网地址：

http://www.stcmcudata.com

http://www.stcmcu.com

http://www.gxwmcu.com

夏学期《微机》实验安排

周次	实验名称	实验内容和目的
4	认识实验	熟悉 keil 集成开发环境，熟悉 STC-ISP 功能软件实验，跑马灯实验
5	定时器实验	熟悉显示驱动函数接口，实现计时和显示
6	键盘显示实验	熟悉行列式键盘编程和动态扫描显示
7	AD 和串行实验	ADC 温度采集，串行口上传到 PC 机记录

实验设备：STC 原厂实验箱 2014/2015 版

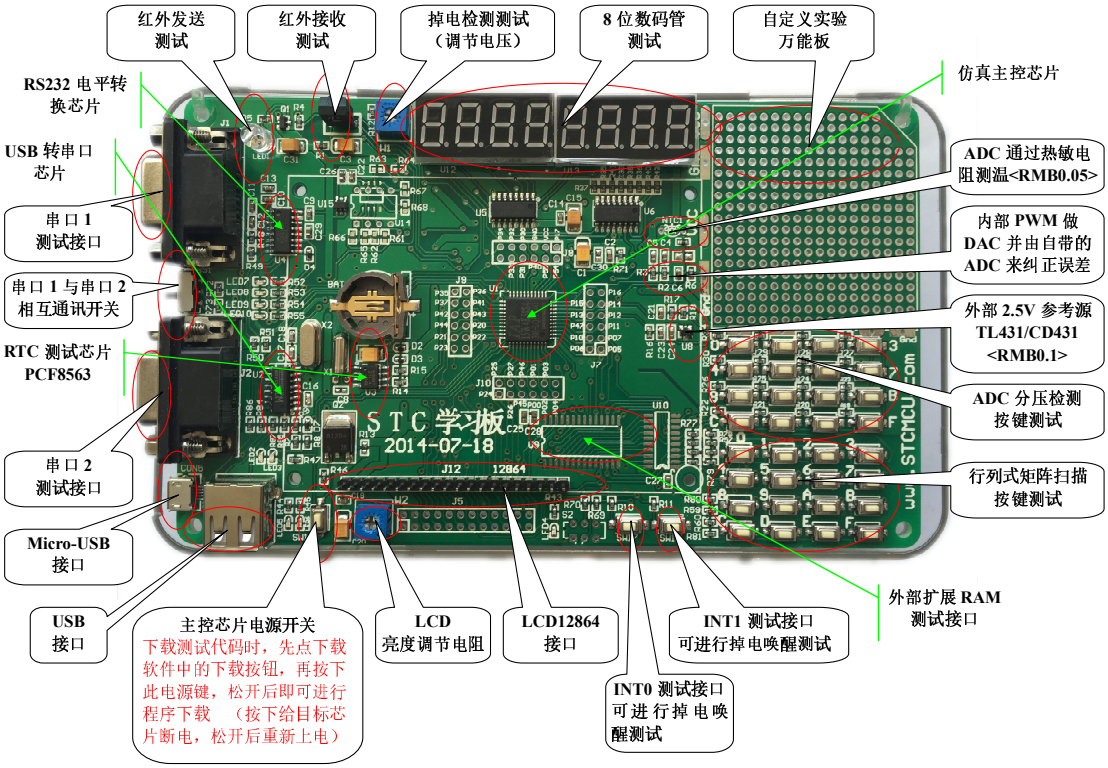


图 1. STC 实验箱 PCB 和元件布局

2014/2015 版芯片、器件和连接完全相同，不同在于布局上：15 版去除了主芯片 STC15W4K58S4 周围的引脚排针，而将引脚以 DIP 排针方式引出，造成了对 NTC 区、洞洞板区、和 AD 键盘区的推挤。

实验一

- **实验目的：**安装 keil for C51，STC-ISP，CH340

  配置使用 keil 对 STC 单片机进行调试

  熟练掌握 Keil 集成开发环境的用法

  熟练掌握 STC-ISP 的程序下载和调试方法

  掌握汇编和 C 语言的软件编程方法

  编制程序控制 LED 灯成跑马灯流转

- **实验内容：**

软件程序设计（多字节数存放方式：低字节在前，高字节在后）

1） 数据传输：设置片内 RAM 40H 开始的 20 个字节内容为 00-19，传输并逆序存放到片内 50H，90H 和 XRAM 0040H 开始的地方。（汇编编程）

2） 数值运算：将片内 RAM 20H 和 30H 开始的两个双字节数相加和相乘，结果分别放在 40H 和 50H 开始的地方。（汇编）

3） 代码转换：将片内 RAM 20H 开始的 8 个字节压缩 BCD 码，转换为等值的 16 进制数，存放在 30H 开始的地方。（汇编）

4） 数值查表：将片内 RAM 20H 开始的 8 个 16 进制数，转换成 ASCII 码，输出到 30H 开始的内存中。（汇编）

5） 使用 C51 重新设计程序 1~4，完成汇编软件实验类似内容。

硬件程序设计：并行口 LED 灯控制

6） 设计一个程序，使实验箱中 LED7/8/9/10 从上而下循环往复实现流水灯功能，周期为 1s，每个灯点亮时间均分。

7） 将上下两个灯分别看作红灯和绿灯，中间 2 灯看作黄灯。设：红灯亮 30S，直接熄灭转绿灯；绿灯亮 25S，闪烁 3S 熄灭（周期为 1Hz），转黄灯亮 2S，再转为红灯亮。循环该过程仿真路口某一方向的交通灯变化。

- **实验步骤：**

1. Keil 集成开发环境安装和配置

以管理员身份运行 Keil+4 完美破解版.zip（学在浙大提供）里面的 

，步骤见 https://www.bilibili.com/video/BV1mu411Z77G/。安装完 demo 版后，破解不要使用原压缩包内的 keygen，这个只能破解到 2020 年，而使用独立的“2020 版 keil 最新注册机.zip”，亲测可以破解到 2032 年。破解后，软件全功能使用，代码大小不受限制。破解后的软件 “仅供学习交流，严禁用于商业用途”。

2. **Keil 软件的使用，**见：keil_μVision4 使用详解教程.pdf

3. USB 驱动和 STC-ISP 的安装

CH340 的驱动安装详见 STC-ISP 包里面的“STC-USB 驱动安装说明.pdf”。安装完后，如果正常，通过 USB 线接上 STC 实验箱，查看设备管理器，会看到多出来一个虚拟串口，如图 2。在我的电脑上是 COM4，这个就是 CH340 芯片产生的 USB 串口，不同电脑此串口号不一定相同，后面使用根据自己电脑作相应修改。

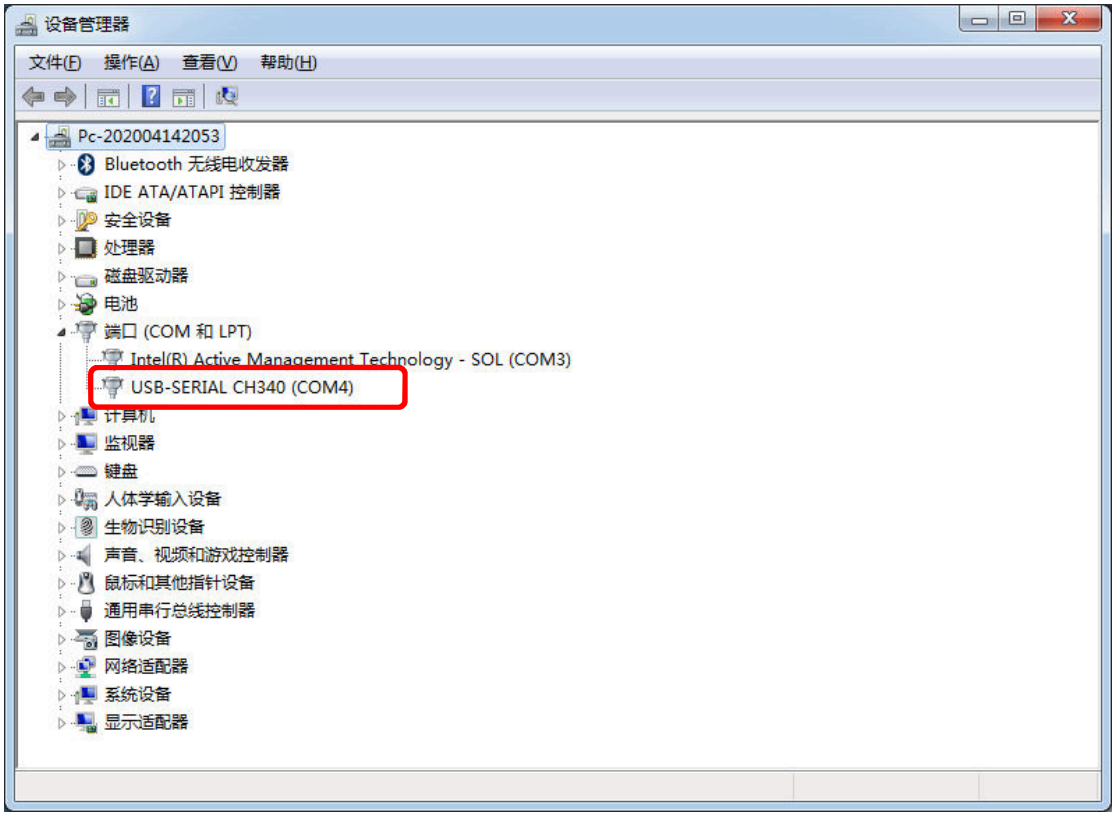


图 2. CH340 虚拟串口

4. STC-ISP 用于程序下载（/烧录）

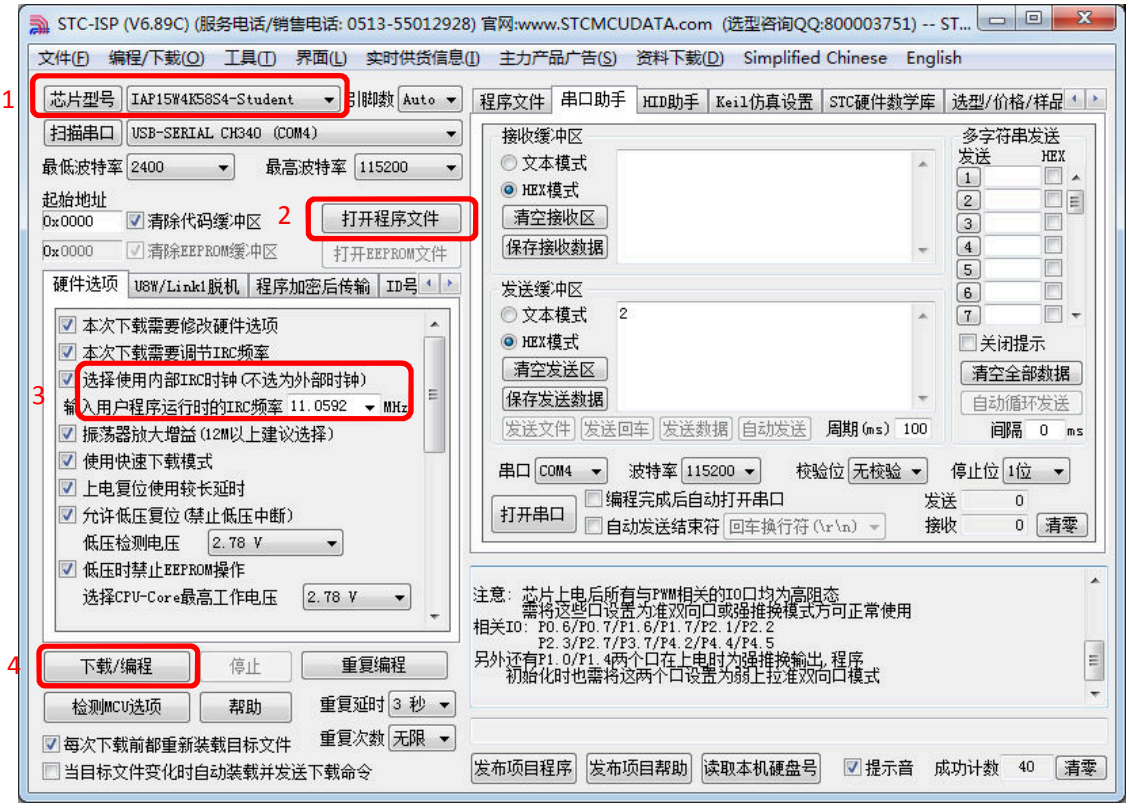


图 3. STC-ISP 程序下载

1. 实验箱的芯片型号为 IAP15W4K58S4，部分实验箱芯片上有激光打码，部分芯片上没有，但都是这个型号，student 指为实验箱专用，其实与普通同型号商用芯片没有什么区别；
2. 找到 Keil 编译生成的.Hex 二进制机器码文件，选中即可；
3. STC 芯片可以使用外部晶振，也可以使用内部 RC 振荡器。实验箱采用内部时钟，需要勾选 IRC。振荡频率决定了芯片的运行速度和功耗，需要根据程序实际情况设定时钟振荡频率。
4. 依次设定了前面 3 项后，其他硬件选项不需要更改，即可以按【下载/编程】按钮，将程序机器码下载到芯片中运行。注意此时在 STC-ISP 程序的右下角窗口会出现（图 4）：

图 4. 等待单片机上电

等待芯片上电时监测目标单片机是否与选中的单片机匹配。这一工作须在上电时完成，需要将实验板上的主芯片完全断电并重新上电，实验板上的 reset 按键就是完成这个功能的（见图 1 下面红色字体对应的按键）。

5. 复位后，程序会进行芯片版本检测，程序下载（/烧写）、校验，在图 5 下方有进度条指示，下载完成成功计数+1。

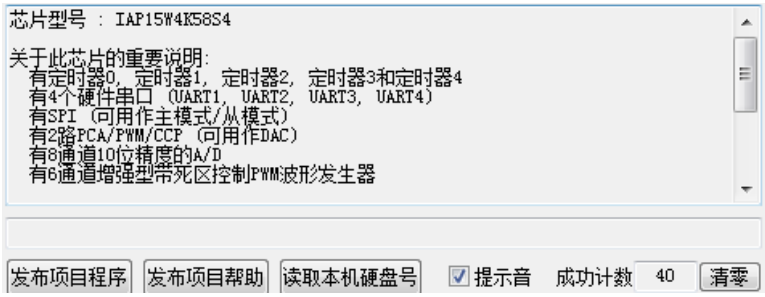


图 5. 芯片检测和程序下载

5. STC-ISP 制作仿真芯片

图 6. 添加头文件和库文件到 keil 中

在 STC-ISP 软件界面中，选择“keil 仿真设置“ ”选项卡（如图 6），点击 “添加型号和头文件…”按钮，在弹出的“浏览文件夹”窗口中，选中 Keil 的安装目录（Keil 的默认安装目录为“c:\keil”），确定后，若弹出“STC MCU 型号添加成功”则表示所有 STC 型号的头文件、库文件和 keil 下的硬件调试驱动均已安装完成。

STC 各系列中，IAP 打头型号的单片机属于可仿真单片机，也就是说，使用该类芯片，不用去另外购买价格昂贵的仿真器，就可以在 keil 下实现单步、断点、步入、步出等调试，随时监控 CPU 和片内寄存器、存储器和其他资源的工作状态，监控变量的变化、程序的运行是否符合设计逻辑。实验箱的单片机正好是 IAP15W4K58S4，属于宽电压的仿真芯片。出厂时，IAP 单片机的仿真功能默认是关闭的，若要使用仿真功能，则需使用 STC-ISP 下载软件将目标单片机设置为仿真芯片。

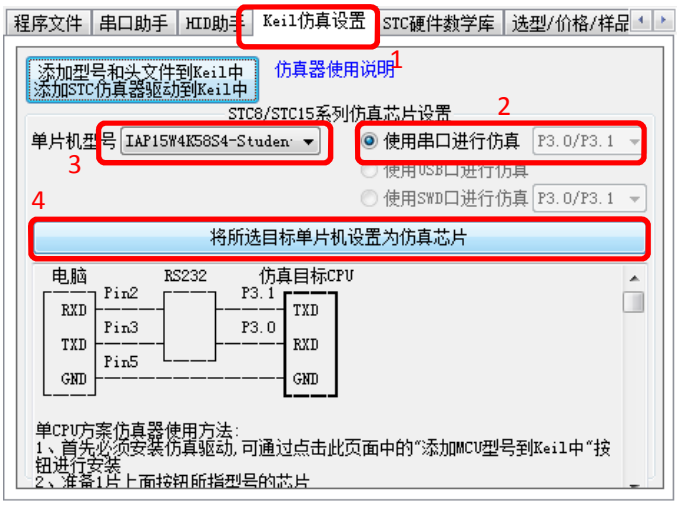


图 7. 设置实验箱单片机为仿真芯片

仍然在“keil 仿真设置”选项卡中，选择仿真口为串行口，单片机为 IAP15W4K58S4，点击“将所选目标单片机设置为仿真芯片”。当下载滚动条达到终点后，仿真芯片制作完成。仿真芯片一旦制作完成，在不烧写其他程序的情况下，仿真模式不会被改变。

6. Keil 软件 STC 仿真调试设置

打开 Keil uv4，新建 project，输入项目名，确认，会弹出：

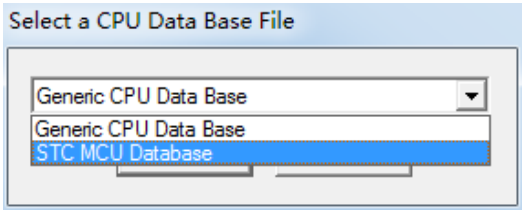


图 8. MCU 类型选择

这是提示选择 CPU（MCU）类型，我们下拉选取 STC MCU Database。接着弹出一个型号选择窗口：

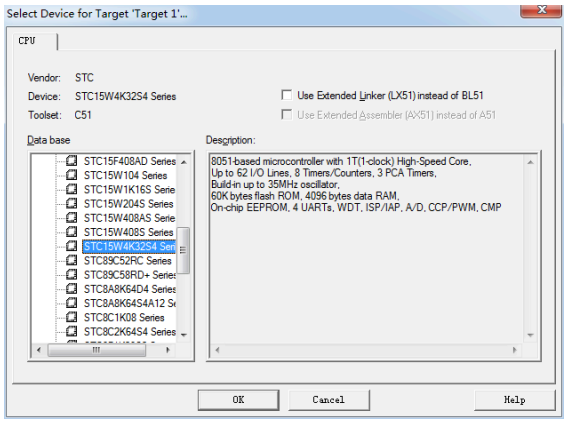


图 9. MCU 型号选择

可选型号非常多，但是没有我们实验箱主芯片完全一样的型号，可选择核心和资源非常相近的 stc15w4k32 系列。

建立项目时，会提示是否拷贝一个标准初始化文件 startup.a51 到项目中，可选择“否”。这样建立的是一个纯净项目。

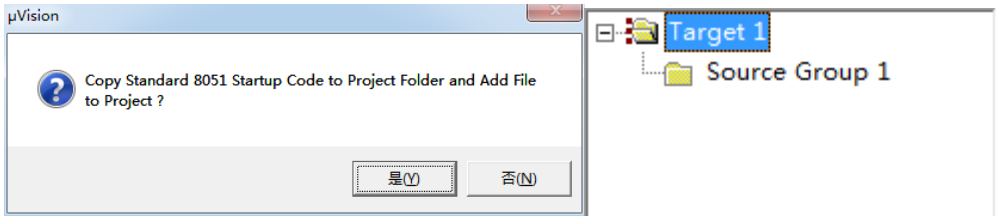


图 10. 启动文件拷贝选择

新建文件（New）,会建立一个空白文本文件 Text1，可以在文件中输入程序源代码。如果是汇编程序，输入代码后请保存为.asm 或者.a51，如果是 C51 程序，输入代码后请保存为.c。Windows 系统秉承了 Microsoft 目录文件命名规则，不区分大小写。Keil 以项目为单位组织文件，源文件和用户库文件需要加入到项目中才能生成目标代码文件。头文件只需要包含在源代码中，不需要单独加入项目。

在 Keil 软件中打开项目文件，1.点击魔法棒按钮（options）。

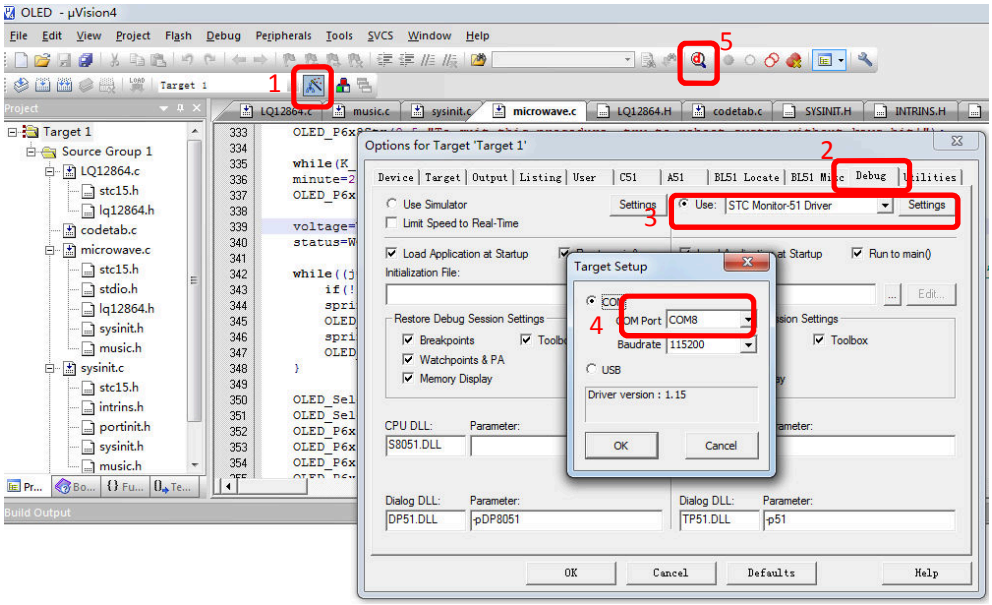


图 11. 设置实验箱单片机为仿真芯片

2.在弹出的选项窗口中选择”debug”选项卡，页面左边为软件仿真设置，右边为硬件调试设置。

3.选中右边的单选项，点击 use 右侧的下拉菜单，选中“stc monitor 51 driver”作为仿真驱动，点击其右侧的 settings。

4.在弹出窗口中选择”COM”调试，COM 口拉选 CH340 虚拟串口，如前所述，我这台电脑应该选择 COM4 ，波特率 115200。确认无误后，点击 OK 保存并关闭选项卡。

5.保持实验箱通过 usb 线连接到电脑，程序编译通过无错误后，点击 debug 按钮，进入在线调试。

注意：如果更换电脑连接的 usb 口，驱动不需要重新安装，但是虚拟串口号会发生改变，需要在各处作相应调整。

7. 汇编程序实例

数据传输：设置片内 RAM 40H 开始的 20 个字节内容为 00-19，传输并逆序存放到片内 50H，90H 和 XRAM 0040H 开始的地方。

例：汇编源代码保存为 movdata.asm,加入项目文件如下图所示：

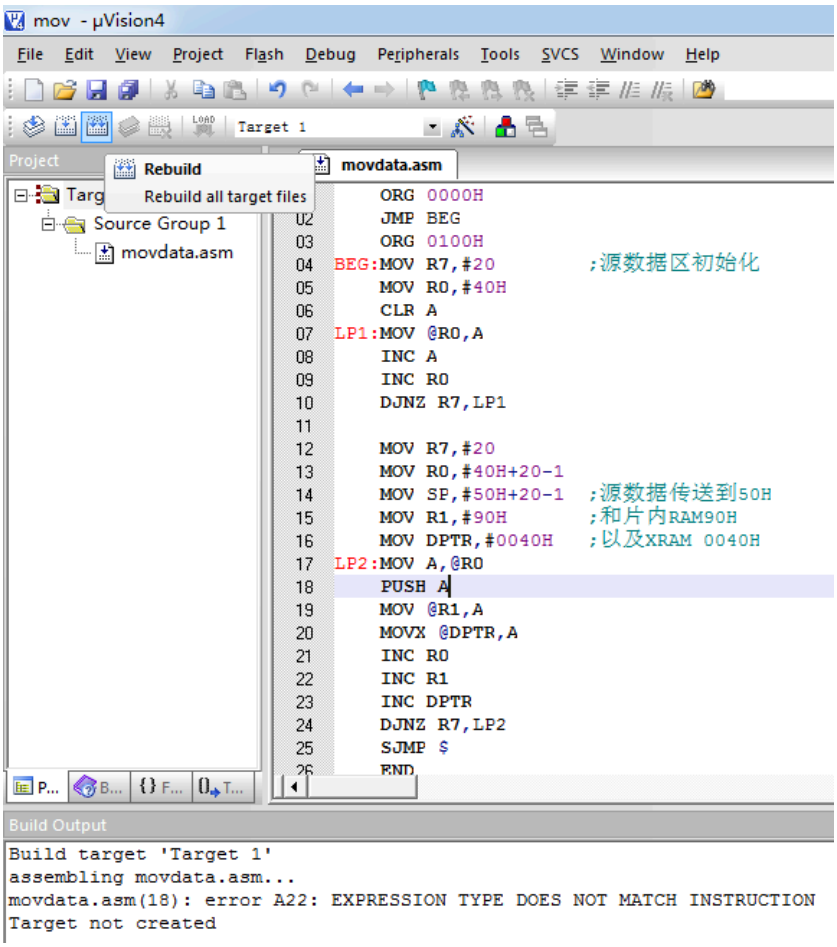


图 12. 数据传输汇编程序实例

点击工具栏第二排第三个按钮 build，编译出现语法错误，提示为“指令操作数不匹配”，经查指令格式，发现 PUSH/POP 指令只接受 direct 寻址，将 A 改为 ACC，重新编译通过。

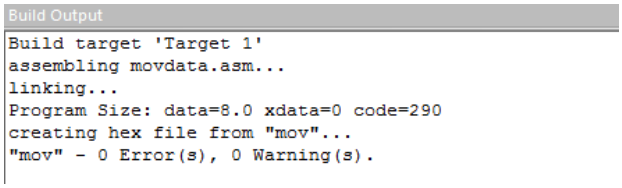


图 13. 编译结果输出窗口

从输出窗口可以看到，程序片内数据使用了 8 个字节，没有使用片外数据段（XRAM），代码段占用 290 个字节（包含 0~100H 空出的 256 字节）。Keil 是不统计数据传输所使用的字节数的，在汇编中

如此，在 C 语言中亦是如此，C51 程序所占用空间只计变量，尤其是数组变量，指针根据存储模式只占用 1~3 个字节，而可以操作的空间且很大，这就是为什么要慎用指针，防止指针越界的原因之一。

软件实验在 keil 中可使用软件仿真来进行，如图 14。

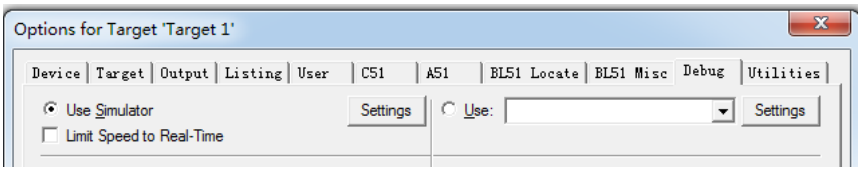


图 14. 软件仿真点选 Use Simulator

使用断点运行，程序执行初始化源数据区前后的区别见图 15。

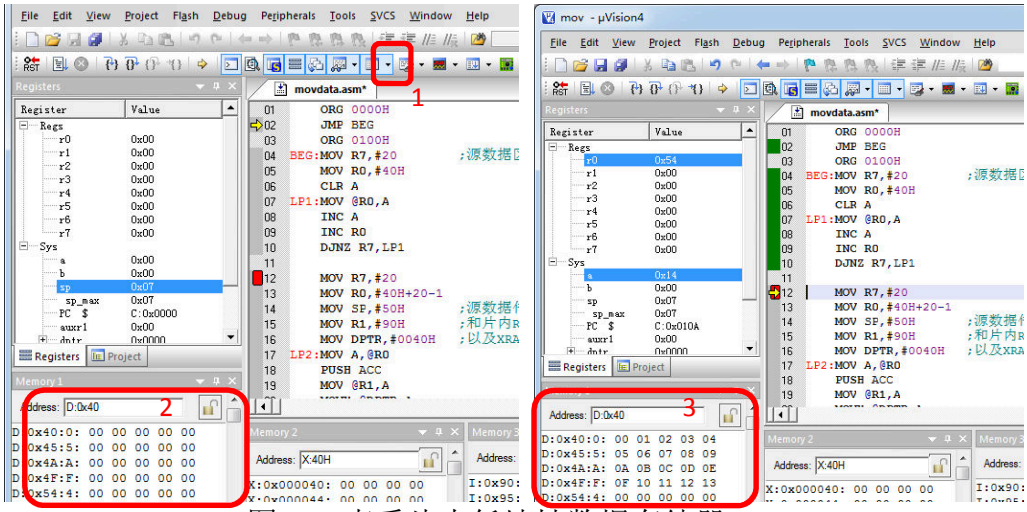


图 15. 查看片内低地址数据存储器 DATA

在调试状态下，1.打开内存窗口，可以查看片内 RAM、片外 RAM 和程序存储器中存储的内容。查看方式为，在地址栏输入“存储类型：存储地址”，如要查看片内 DATA 空间 40H 单元开始的内容，输入：D:0x40，查看片内高 128 字节中 90H 开始的内容：I:0x90，片外 RAM 0040 开始的内容：X:0x40，程序段 100H 内容：C:0x100。从图 15 中 2 和 3 可见，40H 开始的 20 个单元已经被正常初始化。

一个 LP2 循环，程序从 R0 指向的源数据区 4FH 取出一个字节，暂存到累加器 A，压入 SP 指针指向的单元，通过 R1 传送到片内高地址 90H 单元，通过 DPTR 写入 XRAM 40H 单元。

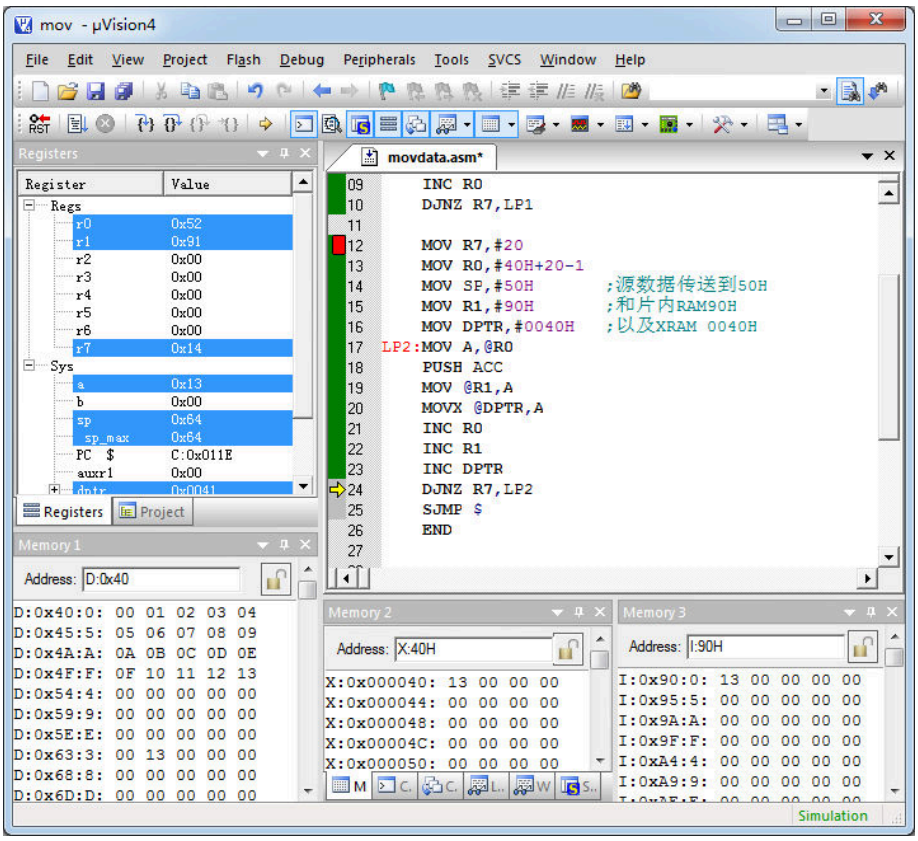


图 16. 查看寄存器和存储器

执行一个循环，XRAM 和片内高地址中的结果正确，片内 50H 中结果不对，数据存放到了 51H 单元。出现这个问题的原因是由于 51 单片机的堆栈是满顶法向上生长的，SP 初始化应为 4FH。一次循环结束后指针修改见图 15 左边窗口 registers。第二次循环发现并没有改变各目标数据区下一个地址的数据，仔细检查源程序（可单步跟踪）发现寄存器中的源指针 R0 并没有按照我们的设计从后往前移动，修改第 21 行 INC R0 为 DEC R0，问题解决。

全速运行程序完成，发现传送的数据区最后几位数据还是不对，经查，是因为 DATA 中的源数据区和目标数据区有交叠，修改程

序分两次传送，将 IDATA 90H 中的传送结果作为第二次传送的数据源，问题解决。

注：第 1 行 ORG 0000H 为复位地址，其中存放一条跳转指令转向真正的用户程序，空出中断向量表；末 SJMP $为防止程序跑飞。

8. C51 程序实例

数据传输：设置片内 RAM 40H 开始的 20 个字节内容为 00-19，传输并逆序存放到片内 50H，90H 和 XRAM 0040H 开始的地方。

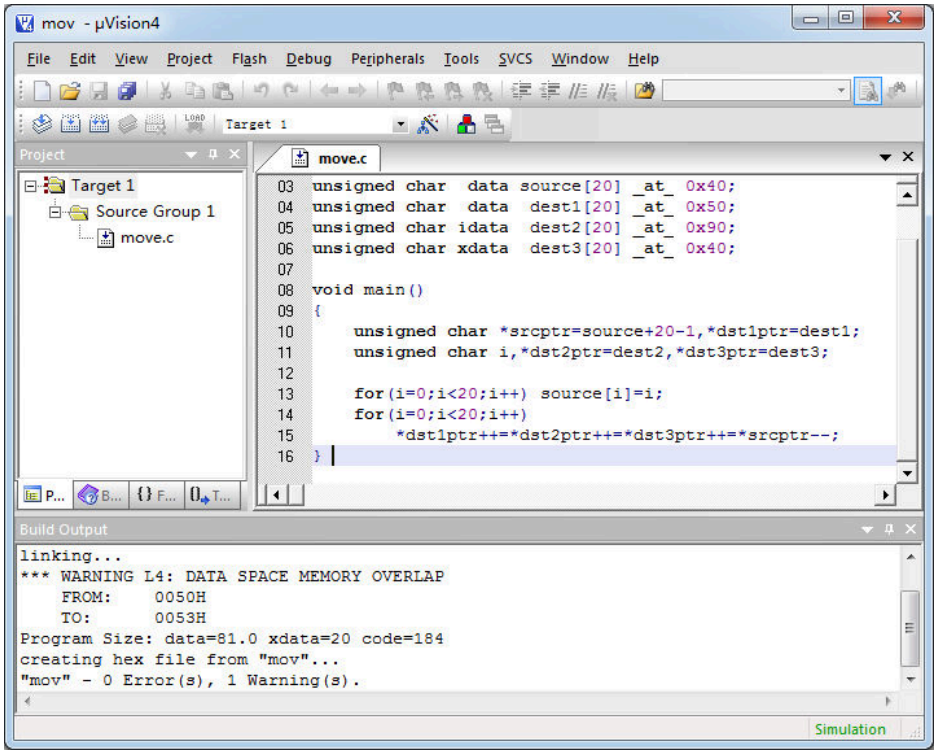


图 17. C51 绝对地址访问数据传送

C 程序是用数组变量方式编程的，预先分配了存储空间，好处是一旦遇到了空间交叠，编译程序会抛出一个警告信息，如上图。

C51 程序调试与汇编调试一个最大的不同是可以跟踪观察变量，如果图 18，函数的局部变量可以在 local 窗口中跟踪到，全局变量可以右键点击加入到 watch 窗口。

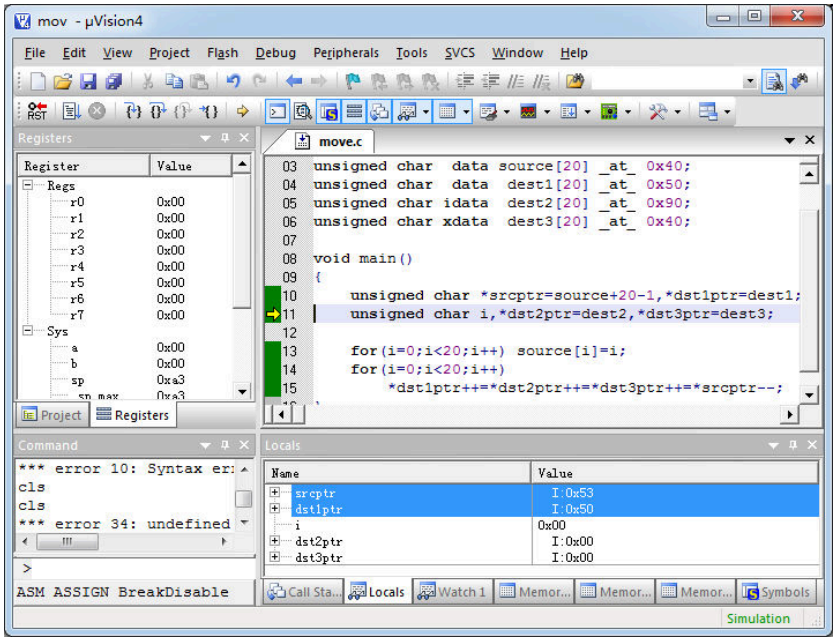


图 18. Keil 的 local 调试窗口可以跟踪 C51 函数局部变量

从上图可以看出，初始化后，srcptr 指向 0x53，dst1ptr 指向 0x50，dst1ptr 指向单元中写入的内容冲掉了 srcptr 指向源数据段后面部分的数据。通过如汇编示例中说的的修改，结果正常。

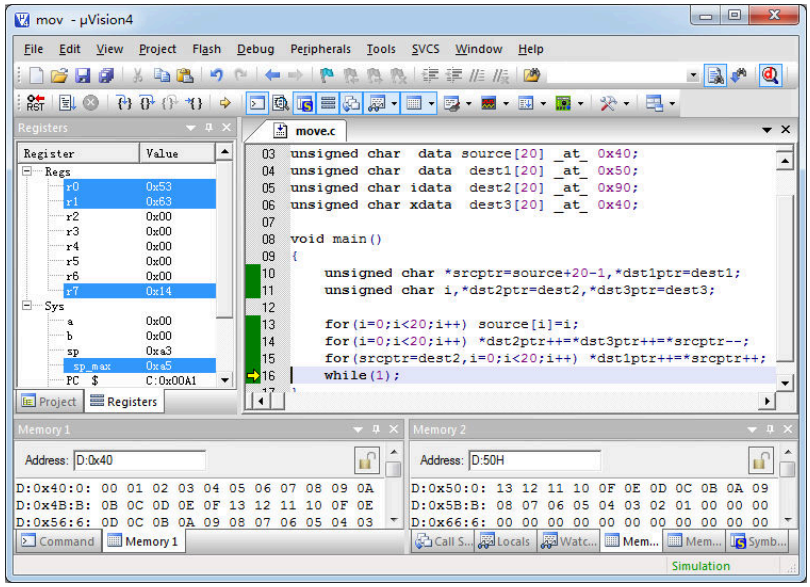


图 19. 修改后传送到目标地址中的数据正常

9. 硬件实验：流水灯控制实例

流水灯又称跑马灯，常用在节日彩灯中。流水灯程序设计，也是单片机硬件最简单的入门程序设计，涉及到 IO 口的配置和使用。

STC 实验箱没有设计完整的一排 8 个流水灯，而且毗邻的 4 个 LED 灯也不是由一组 PIO 口单独控制的，分属于 P1 口的 P1.7、 P1.6 和 P4 口的 P4.7、P4.6，如图 20。

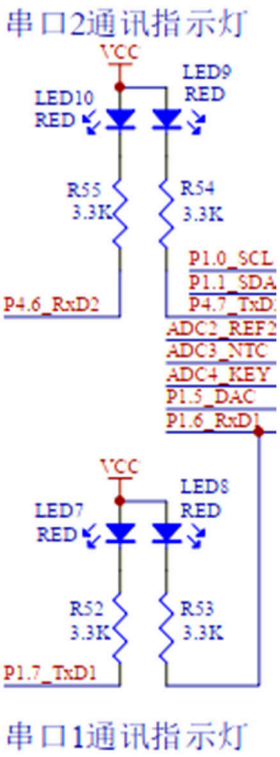


图 20. 流水灯硬件

实际 PCB 板上自上而下排列顺序为 LED7、LED8、LED9、LED10，也就是 P1.7、P1.6、P4.7 和 P4.6。

可以参考 STC 官方提供的“STC-开发板 4-程序-C 语言-V2\01-P17 P16 P47 P46-跑马灯”略作修改作为 1S 轮转的流水灯控制程序。

```
#include     <STC15.H>

#define u8 unsigned char
#define u16 unsigned
#define MAIN_Fosc        22118400L   //定义主时钟

void    delay_ms(u8 ms)
{
      u16 i;
      do{
            i = MAIN_Fosc / 16000;
           while(--i)  ;     //16T per loop
       } while(--ms);
}

void setLED(u8 i)
{
    P17 = P16 = P47 = P46 =1;
    switch(i) {
         case 0:   P17 = 0; break;
         case 1:    P16 = 0; break;
         case 2:   P47 = 0; break;
         case 3:   P46 = 0; break;
    }
```

```
    delay_ms(250);
}
/******************** 主函数 **************************/
void main(void)
{
    u8 i;

    P1M1 = 0;   P1M0 = 0;   //设置为准双向口
    P4M1 = 0;   P4M0 = 0;   //设置为准双向口
    while(1)
        for(i=0;i<4;i++)
            setLED(i);
}
```

STC-ISP 设置仿真调试的时候已经帮助我们安装了 STC15.H，路径在<keil 安装目录>/C51/INC/STC 下面，这个头文件定义了 15 系列芯片所有的 SFR 和 SBIT 位，比如实验程序中使用到的 P16，P17 等。需要在编译选项的 C51 选项页中告诉 keil 到哪儿去找到这个头文件，如图 20，编译才能顺利通过。

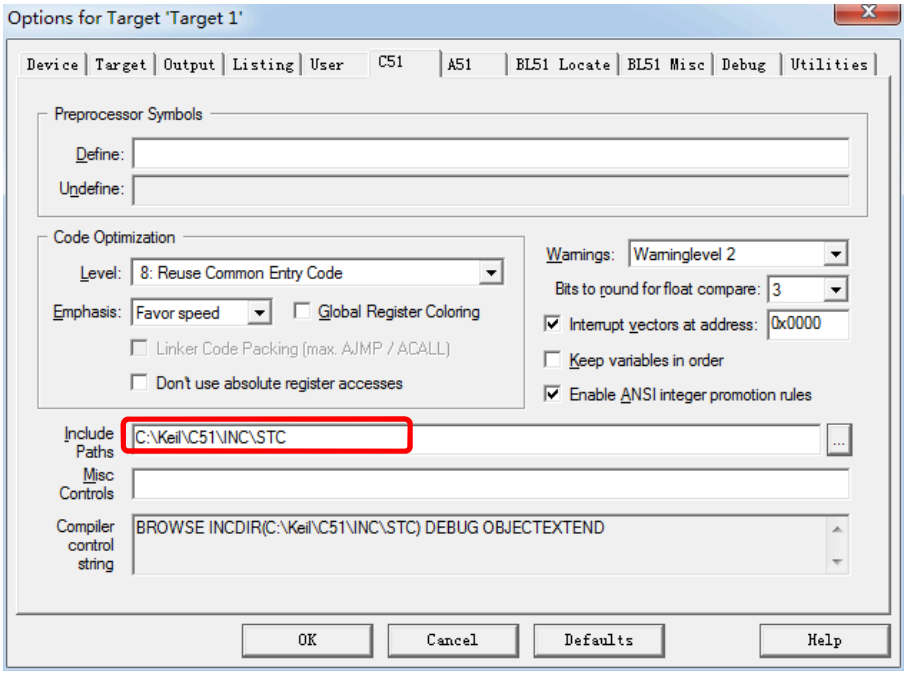


图 21. 修改后传送到目标地址中的数据正常

delay_ms 是 ms 级的软件延时程序，主频为 MAIN_Fosc,记为 f，程序内循环约 16T=16/f，循环次数 f/16000，大致延时 16/16000 约 1ms。软件延时不会很准，除了计算误差，还会受到中断影响。

注：实验如果遇到双灯同亮，需要将旁边的开关拨向上方。

- **练习与思考：**

1. 为什么每做一个实验，最好新建一个项目进行管理？
2. 如何用汇编语言实现多字节数的加、减、乘、除？
3. 如果用大端方式存取多字节数据，程序应该如何修改？
4. 表格多于 256 项，每项不是一个字节，程序如何修改？
5. C51 如何实现 51 单片机的 7 种寻址方式，使用变量和指针都可以访问存储单元，其访问方式有什么区别？
6. 用 C51 编程，要访问某一个确定起始地址的单元或数据块，如果不使用绝对地址方式和用_at_开辟数组，该如何做？
7. P0M0=P0M1=0 做了什么工作，如果去掉对程序有什么影响？

实验二

- **实验目的：**

  掌握单片机定时计数器的原理和使用

  掌握单片机外部中断的原理和使用

  了解 8 位 LED 显示驱动函数的接口方法

  使用硬件定时器实现秒表正/反计时

  使用外部中断按键控制定时器秒表

- **实验内容：**

  1） 在 8 位 LED 的低 3 位显示秒表，范围为 0~200 不含 200，当数值增加达到 199 后，下一秒恢复到 0S 重新计秒。

  2） 使用外部中断 0 控制秒表的启停，使用外部中断 1 控制秒表在正计时秒表和倒计时秒表之间切换。按键不改变秒表的当前计数值。

  3） 将秒表的位数扩大为 5 位，小数点后两位分别为 0.1S 和 0.01S，重复实验 1，2。

  4） 设计 24 小时时钟，使时钟同样具有实验 2 的功能。

  5） 使用 LED 显示学号，学号超出 8 位部分通过中断按键左右移动以便能够显示完整。

- **实验步骤：**

1. 8 位 LED 驱动函数

实验箱上部有 8 位“8”字 LED 数码管，每位都带有小数点，适于显示 16 进制数和一些简单段图，比如“-”，“_”，“ ”，“o”等。

本实验拟用这些数位显示秒表、时钟和学号。8 位 LED 数码管显示模块是由两块 74HC595 分别驱动的，这部分内容在实验三介绍。本实验已经将 8 位 LED 的驱动函数独立出来，存放在 LED8.C 中，实验需要将 LED.c 文件包含在项目中，这也有别于实验一。

尽管一个项目可以包含多个源文件，但是 main 函数只能有一个，它决定了程序的入口和运行流程。其他函数都是被 main 函数直接或者间接调用的函数，如果某个函数始终没有被调用到，编译器会弹出警告。警告不会影响交叉编译并正常生成.hex 机器码文件。有些程序员喜欢将所有错误和警告都清除后再下载调试，这无疑会使程序逻辑更加清晰，而有些程序员则喜欢保留着没有被调用到的函数，以便于今后继续积木式扩展。这只是个人习惯问题，见仁见智无所谓优劣。项目窗口如图 22 左侧。LED8 打包了显示驱动，提供一个数组 u8 LED8[8]作为显示缓冲区给主程序 chronograph.c 使用。

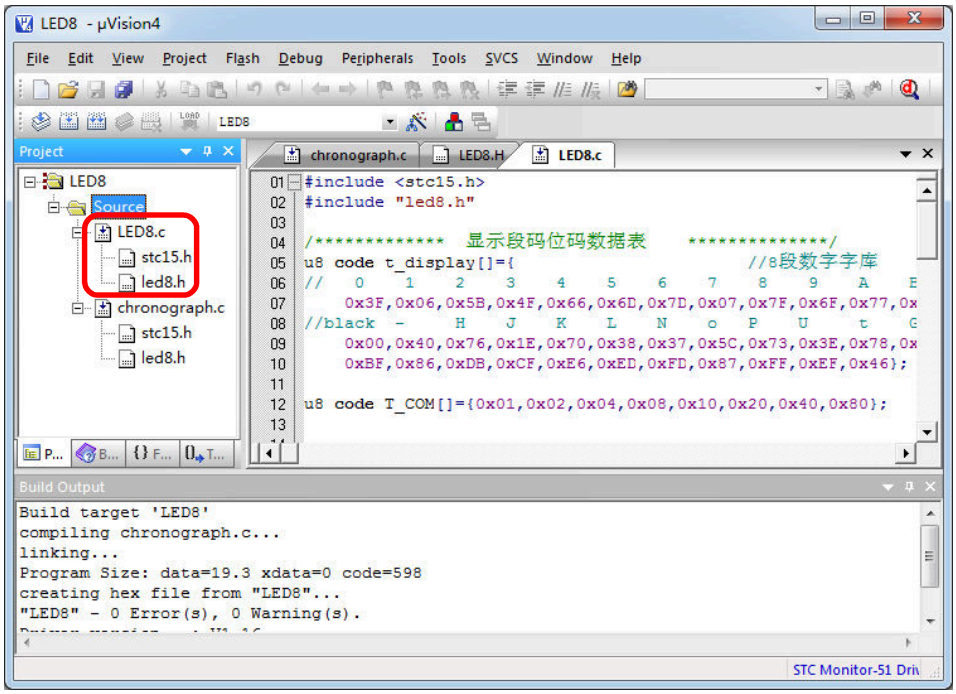
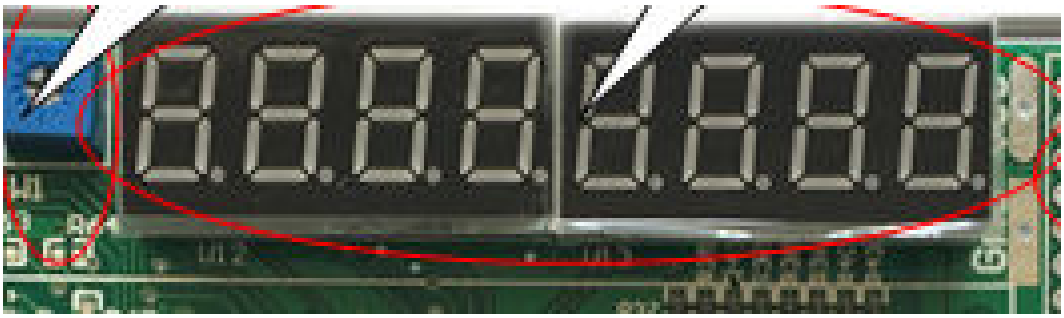


图 22. 多文件项目管理

显示缓冲区的组织方式是，低位在左高位在右：



LED[0] LED[1] LED[2] LED[3] LED[4] LED[5] LED[6] LED[7]

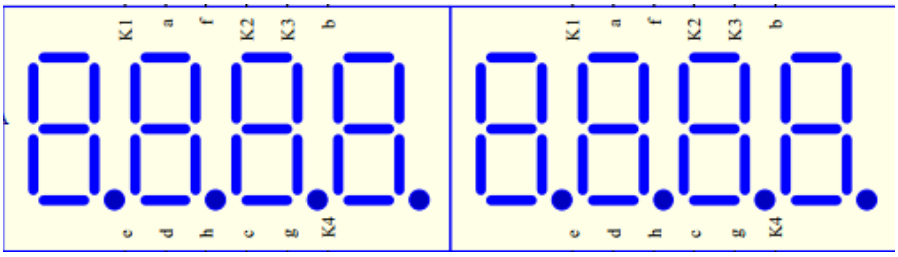


图 23. 显示缓冲区与实际显示位置的对应关系

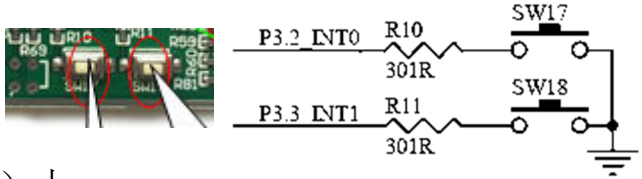
显示缓冲区中存放的是要显示的数据，一个 16 进制数占用一个字节单元，如果我们要显示“12345678”，在缓冲区中存放的方式是 LED[0]=1，LED[1]=2，LED[2]=3，LED[3]=4，LED[4]=5，LED[5]=6，LED[6]=7，LED[7]=8。如果要显示带小数点的数字，可以在置入显示缓冲区中的值中加上 0x20，比如 LED[5]=0x26，则显示“6.”。

驱动程序使用 1ms 中断刷新一位显示，一个周期 8 位为 8ms，达到了 125 次/秒的帧率，人眼是看不到抖动的。

2. 秒表[0,200)循环正逆计数器例程

方案设计：1.秒走时；2.数值范围为 0~199 循环计数；3.通过 sw17 改变正逆方式，0 为正计时，1 为倒计时；4.通过 sw18 可暂停和恢复计数。

硬件支持(如图 24)：



SW17 接在 P3.2（int0）上，

图 24. 按键硬件

SW18 接在 P3.3（int1）上，按键释放时中断线被准双向口内部弱上拉到高电平，按键被按下时，中断线被 300 欧姆电阻拉低。

设置 int0、int1 外部中断为下降沿触发方式，打开中断，按键就可以触发相应的外部中断。

定时/计算器 T0 设定工作在定时方式，每 1ms 产生一次中断，1000 次中断为 1S，如果计数允许，就更新一次计数值 Count，同时将计数值更新到显示缓冲区。

根据要求，显示缓冲区的 LED[5]存放计数值的百位值，LED[5]存放十位值，LED[5]存放个位值。

这样问题就简化为三个基本求解过程：1.显示缓冲区的更新，触发事件每 s； 3.显示的定时刷新，触发事件每 ms。2.按键的处理，触发事件外部中断。

程序分析（程序中有详细注释）：

led8.h：

```
typedef       unsigned char     u8;
typedef       unsigned int      u16;
typedef       unsigned long     u32;

#define DIS_DOT        0x20
#define DIS_BLACK      0x10
#define DIS_           0x11
```

```
void DisplayScan(void);
```

头文件在多模块项目设计中有着举足轻重的作用，它是连接模块间函数的桥梁。led8.h 提供了类型的别名定义，即 typedef 定义的新类型，方便书写。该头文件还包含了特殊段码所在位置的符号定

义，方便程序设计中不需要去记忆、查询特殊段码的地址偏移量。头文件还可包含模块中函数的原型声明，这个原型声明必须与函数定义相一致：有相同个数和类型的输入参数，相同的返回值类型。在编译过程中，语法检查器会检查调用和定义是否一致。模块内部调用的函数，其原型声明不需要出现在头文件中，有些甚至显式定义为静态函数。但是需要被其他模块调用的函数，必须在调用模块中声明其原型，最简单的组织方式是在头文件中声明，需要调用此函数的模块包含该头文件，如 void DisplayScan(void);

chronograph.c：

```
#include <stc15.h>
#include "led8.h"

/*************  外部变量和本地变量声明      **************/
extern u8   LED8[8];          //显示缓冲，在 LED8.c 中声明
bit F_T0=0;                   //1ms 中断软标志
bit F_incdec=0,F_stop=0;     //运行参数，初始为增量计数器，不停止
u8   Count=0;                 //unsigned char 与 s8,u16 在 led8.h 中定义

/***************** T0 中断初始化函数  ******************/
void T0_Init()
{
    #define       MAIN_Fosc          22118400L     //定义主时钟
    #define       Timer0_Reload     (65536UL -(MAIN_Fosc / 1000))

    AUXR = 0x80;       //Timer0 set as 1T, 16 bits timer auto-reload,
    TH0 = (u8)(Timer0_Reload / 256);
    TL0 = (u8)(Timer0_Reload % 256);
    ET0 = 1;            //T0 interrupt enable
    PT0 = 1;            //设置 T0 我高优先级
    TR0 = 1;            //启动 T0
}
/***************** INT0/INT1 中断初始化函数  ******************/
void INT_Init()
{
    IT0 = 1;       //INT0  下降沿中断
```

```
    IT1 = 1;       //INT1  下降沿中断
    IE1   = 0;     //外中断 1 标志位
    IE0   = 0;     //外中断 0 标志位

    EX1 = 1;       //INT1 Enable
    EX0 = 1;       //INT0 Enable
    EA = 1;        //允许总中断
}

/********************** 端口初始化 ************************/
void Port_Init()
{
     P0M1 = 0;     P0M0 = 0;     //设置为准双向口
     P1M1 = 0;     P1M0 = 0;     //设置为准双向口
     P2M1 = 0;     P2M0 = 0;     //设置为准双向口
     P3M1 = 0;     P3M0 = 0;     //设置为准双向口
     P4M1 = 0;     P4M0 = 0;     //设置为准双向口
     P5M1 = 0;     P5M0 = 0;     //设置为准双向口
     P6M1 = 0;     P6M0 = 0;     //设置为准双向口
     P7M1 = 0;     P7M0 = 0;     //设置为准双向口
}

void   delay_ms(u8 ms)
{
      u16 i;
      do{
            i = MAIN_Fosc / 16000;
            while(--i)      ;     //16T per loop
      }while(--ms);
}

/********************** INT0 中断函数 ************************/
void INT0_int (void) interrupt 0          //进中断时已经清除标志
{
     F_incdec^=1;          //sw17 乒乓键，改变计数方向标志
     delay_ms(20);         //软件延时消抖动
}

/********************** INT1 中断函数 ************************/
void INT1_int (void) interrupt 2          //进中断时已经清除标志
{
     F_stop^=1;       //sw18 乒乓键，改变计数启停标志
     delay_ms(20);  //软件延时消抖动
}
```

```
/********************** T0 1ms 中断函数 ************************/
void T0_int (void) interrupt 1
{
    DisplayScan();   //1ms 扫描显示一位
    F_T0 = 1;        //1ms 时间到标志
}

/********************** 显示计数函数 ************************/
void    DisplayCount(void)
{
    LED8[5] = Count/100;            //百位
    LED8[6] = (Count%100) / 10;     //十位
    LED8[7] = Count % 10;           //个位
}

/********************** 主函数 ************************/
void main(void)
{
    u16 i=0;

    Port_Init();            //初始化
    T0_Init();
    INT_Init();

    for(i=0; i<8; i++)   LED8[i] = DIS_BLACK;     //上电消隐
    while(1) {
        if(F_T0) {   //1ms 到
            F_T0 = 0;
            if(!F_stop)
                if(++i >= 1000) {               //1 秒到
                    i=0;
                    if(!F_incdec){              //根据计数方向更新计数
                        if(++Count==200) Count=0;
                    } else {
                        if(Count==0) Count=199;
                        else Count--;
                    }
                    DisplayCount();             //更新显示缓冲区
                }
        }
    }
}
```

定时器溢出时间的计算：

```
#define     MAIN_Fosc       22118400L     //定义主时钟
#define     Timer0_Reload   (65536UL -(MAIN_Fosc / 1000))
```

根据 51 基本知识，定时器缺省工作在方式 0。STC 修改了传统 51 比较鸡肋的 13b 定时器方式为 16b 可重载定时/计数器。STC15 的定时器时钟源有 1T 方式和 12T 方式，在 1T 方式下，时钟源 T=1/f。设定时时间为 n*T，其中 n 为溢出脉冲数，单位为秒，若要使 n*T =1/1000 秒，则 n=f/1000。因为 51 定时器为加法计数器，定时初值需要设置为脉冲数的补码，重载时间 Timer0_Reload 的计算如程序。

```
AUXR = 0x80;        //Timer0 set as 1T, 16 bits timer auto-reload,
TH0 = (u8) (Timer0_Reload / 256);
TL0 = (u8) (Timer0_Reload % 256);
```

辅助寄存器 AUXR 的最高位是定时器时钟源分频设置位。如=1，表示不分频，即与我们前面计算相一致，如=0，则兼容传统 MCS-51 的时钟 12 分频方式。TH0 为定时器初值高字节，TL0 为定时器初值低字节，分别设置为整型数 Timer0_Reload 的高低字节。

- **练习与思考：**

1. 设置 PT0=1 的目的是什么？试去掉该行观察运行效果。
2. 当制作仿真芯片或者程序下载（/烧写）时，如果设置的 IRC 频率与程序中设置的频率不一致，会出现什么情况？
3. 尝试使用 8 位 LED 数码管的中间横杠，实现跑马灯功能。
4. 如何让 8 位 LED 产生动态效果？比如线段绕着边框追逐。
5. 尝试设置闹钟，在闹钟时间到达的 1 分钟内出现提示信息。
6. 如何在断电时保存闹钟信息，断电后恢复时钟正常运行？
7. 如果需要多个功能，而只有两个按键，你将如何设计？

实验三

- **实验目的：**熟悉独立式和行列式键盘工作原理

  了解实验箱的各种键盘接口方式

  熟练掌握独立式键盘程序设计

  熟练掌握行列式键盘程序设计

  熟悉静态扫描和动态扫描显示原理

  了解实验箱 LED 数码管的接口方式

  熟练掌握 8 位 LED 显示模块程序设计

- **实验内容：**

1） 独立键盘：使用独立按键进行 0-199 循环计数，SW17 用于计数加，SW18 用于计数减。计数从 0 开始，当前计数值使用实验 2 的显示接口方式，显示在 8 位 LED 数码管末 3 位。

2） 行列键盘：读取实验箱行列键盘的编码，显示在 LED 数码管的末位上，编码须与键盘附近标示的数字相同。

3） 学号移动：按一次 sw17 学号开始循环左移，再按一次停在当前位置；按一次 sw18 学号开始循环右移，再按一次停在当前位置。左移过程中，按右移键，不停止，直接从当前位置开始循环右移，反之亦然。学号首末位数字间空一格。

4） 时钟走时：给实验二的 24 小时时钟，增加键盘修改时间功能。要求：按 SW17 进入/退出修改，进入后时钟暂停，退出后按照新时间开始走时。修改过程中当前位闪烁，按 SW18 在 6 位时、分、秒中切换修改位，按数字键修改当前位。

● 实验步骤：

1. 实验箱的键盘布局

实验箱提供了三种方式的键盘，独立键盘、行列键盘和 AD 键盘。独立键盘是指一个按键占用一根 I/O 口线的键盘，适用于按键较少的场合，如 MP3、显示器设置键。行列键盘又称为矩阵式键盘，常用于键盘比较多的场合，如银行数字键盘、电脑键盘、数控机械控制台键盘。还有一种键盘，既有独立键盘占用口线少的优点，又有行列键盘按键数量多的优点，就是 AD 键盘，它使用的是一个模拟通道线，在以前单片机内没有集成 ADC 等模拟电路时不常见，可见于各种日本随身听，如 sony Walkman, discman。

STC 实验箱中，有三个独立按键，一个复位按键 sw19，仅用于断电和重新上电，一般无法用于读取按键状态。另外两个按键 sw17 和 sw18，分别接于 P3.2 和 P3.3（图 24），实验二中使用它们的第二功能 INT0 和 INT1 产生中断，用于计数器的启停和运行方式。如果关闭中断，也可以将 P3.2 和 P3.3 当作普通 I/O 口来读取按键状态，判断独立按键是否被按下。尝试在关闭中断下对按键进行计数。

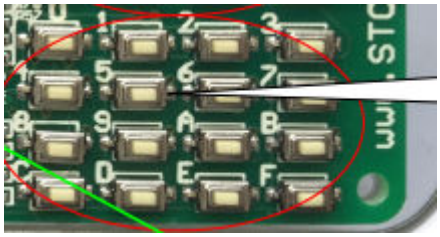


图 25. 行列式键盘

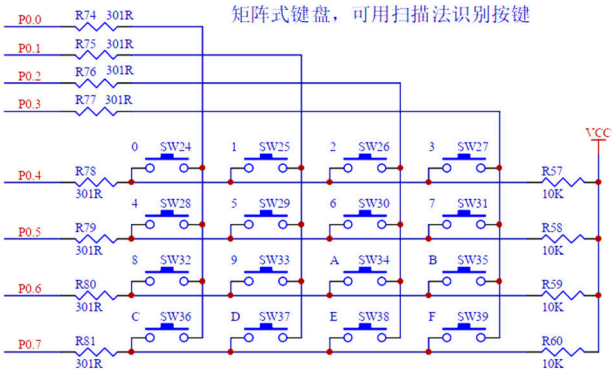


图 26. 行列式键盘原理图

STC 实验箱的行列键盘位于实验箱的右下角，其实物如图 25，键盘接连接原理如图 26：行

列键盘的 16 个按键被连接到 P0 口，其中行线位于 P0 口的高 4 位，列线位于 P0 口的低 4 位，而且是不用跳线直接连接的，也就是说占用了整个 P0 口。原理图接在行列线上的 8 个 300 欧电阻 R74~R81 是为了防止端口不小心被损坏。假设 P0 端口被初始化为推挽方式，行线输出高电平，列线输出低电平，如果没有限流电阻，当某个按键被按下时，连接到这个按键上的两条口线就会形成尖峰贯穿电流，烧毁端口。4 个上拉电阻 R57~R60 的作用是，与限流电阻之间形成分压，以确定正确的输入电平。按键附近有按键编号，实验内容 2 就是请同学们通过扫描方式或者反转方式获取键盘编码，并实时显示出来。

在行列键盘的上方是 AD 键盘。AD 键盘与行列键盘最大的区别是每个按键附近都有电阻，这些电阻为分压电阻，以此确定每个按键所在位置的电压值。ADC 通过读取电压值，确定按键位置。实验箱的 AD 键盘使用一阶 RC 滤波用于在一定范围内消除按键抖动。前两种按键可以通过软件方法处理串键，AD 键盘天生对串键无能为力，这也是为什么一些年份比较久远的随身听串键比较严重。

2. LED 数码管驱动方式

实验 2 中已经提到，实验箱带有 8 位 LED 数码管，并提供了接口方式。本节根据其硬件连接，简要说明 8 位 LED 数码管的工作方式、工作原理以及接口软件的工作方式。

实验箱上的 8 位 LED 数码管由 2 块 3461AS 构成，3461AS 是 4 位 0.36 英寸共阴极红色高亮 LED 数码管，其引脚如图 27。

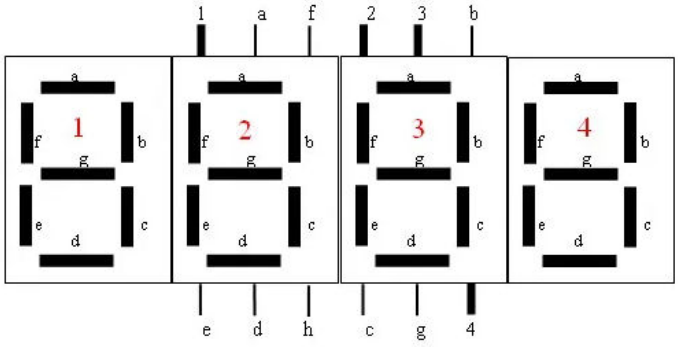


图 27. 3461AS 共阴极 LED 数码管引脚

3461AS 数码管与 MCU 连接，构成 8 位 LED 显示器如图 28。

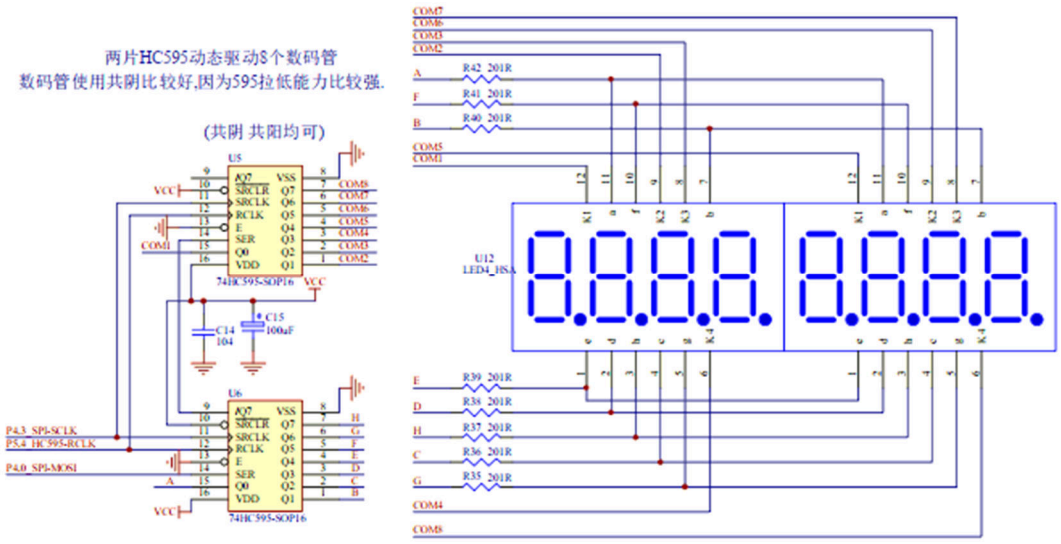


图 28. 8 位 LED 数码管显示器

大多数微机书上动态扫描一脉相承使用并行 74LS373 或者串行 74LS164 作为段锁存，使用 74LS138 等作为位扫描，这些芯片集成度低、数据传输速度慢、功耗相对比较大，供货也有问题，已经不适合在现代各种设备中大量使用。当前大屏点阵显示器一般都大量使用廉价的低功耗高速 CMOS 移位寄存器 74HC595 作为输出锁存器。595 串口采用 SPI 方式，采用串入并出移位寄存器的好处是占用口线资源少。

有必要将 HC595 芯片的引脚图、逻辑图、功能图和真值表贴到指导书上供大家参考，如图 29 和表 1。

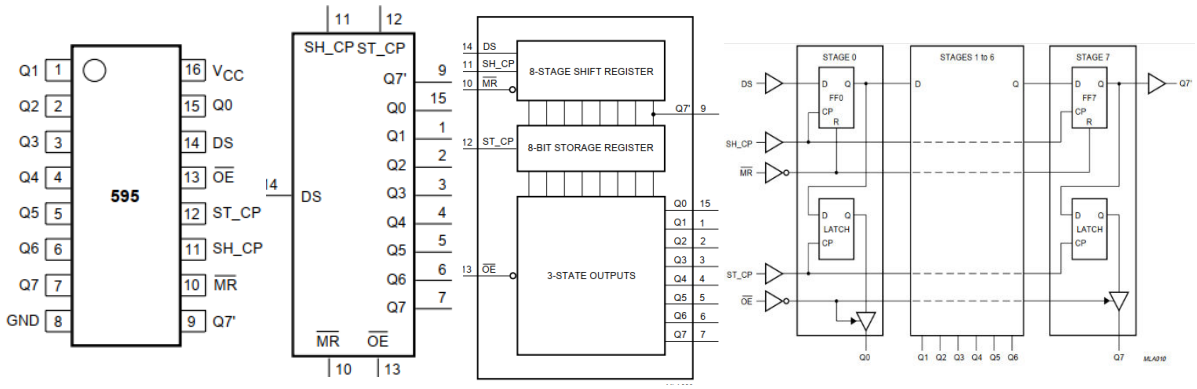


图 29. HC595 移位寄存器引脚、结构和功能

表 1. HC595 移位寄存器真值表

INPUT					OUTPUT		FUNCTION
SH_CP	ST_CP	$\overline{OE}$	$\overline{MR}$	DS	Q7'	Qn	
X	X	L	L	X	L	n.c.	a LOW level on $\overline{MR}$ only affects the shift registers
X	↑	L	L	X	L	L	empty shift register loaded into storage register
X	X	H	L	X	L	Z	shift register clear; parallel outputs in high-impedance OFF-state
↑	X	L	H	H	Q6'	n.c.	logic high level shifted into shift register stage 0; contents of all shift register stages shifted through, e.g. previous state of stage 6 (internal Q6') appears on the serial output (Q7')
X	↑	L	H	X	n.c.	Qn'	contents of shift register stages (internal Qn') are transferred to the storage register and parallel output stages
↑	↑	L	H	X	Q6'	Qn'	contents of shift register shifted through; previous contents of the shift register is transferred to the storage register and the parallel output stages

不同书籍和手册上对引脚的命名可能不一样，但是同一引脚的功能是完全相同的，比如 14 引脚在图 29 表 1（来自数据手册）中命名为 DS，在图 28 中（来自 STC 官方资料）命名为 SER，他们的意义都是串行数据输入。在指导书中先对各引脚进行映射和说明：

表 2. HC595 引脚对应和功能说明表

图 28 引脚	图 29 引脚	引脚功能说明
SER	DS	移位寄存器串行数据输入
SRCLK	SH_CP	串行时钟信号，上升沿产生移位，不影响输出数据
RCLK	ST_CP	并行时钟信号，上升沿将移位寄存器置入输出锁存器
Q0~Q7	Q0~Q7	并行输出口线，与输出锁存器中的各位一一对应
/Q7	Q7'	移位寄存器中 Q7 输出线，用于级联，不受 RCLK 影响
SRCLR	MR	异步清 0 端，低电平有效，仅清除移位寄存器中的值
E	OE	输出允许端，低电平有效，高电平时输出为高阻状态
VDD、VSS		电源、地线

STC 实验箱中，有两片 595 芯片，其中 U6 的串行输出线被连接到 U5 的串行输入线，两片 595 级联构成 16 位的串行移位寄存器。

实验箱中，一方面，两片 595 构成的 16 位移位寄存器的功能是通过串行 SPI 总线（软件模拟）完成 MCU 的 16 位并行口扩展。U6 的串行输入端 SER，也就是 16 位移位寄存器串行数据源，为 P4.0，移位寄存器的串行移位时钟端为 P4.3，并行打入时钟端为 P5.4。另一方面，此 16 位移位寄存器也担当着 LED 数码管显示驱动的功能。

U6 的输出口线被连接到 LED 数码管的段码总线 ABCDEFGH 上，用于进行段码驱动，高电平有效，ABCDEFG 对应的驱动段见图 27，H 没有在图 27 中标注，为 7 段 LED 数码管的小数点段。

数码管的段码驱动采用软件译码的方式，码表如表 3。

表 3. 3461AS 十六进制数段码表

符号	段码	符号	段码
0	0x3F	8	0x7F
1	0x06	9	0x6F
2	0x5B	A	0x77
3	0x4F	B	0x7C
4	0x66	C	0x39
5	0x6D	D	0x5E
6	0x7D	E	0x79
7	0x07	F	0x71

表三列出了 0~F 对应的段码，只要 ABCDEFGH 按序接到 Q0~Q7 位，所有共阴极 LED 数码管的段码都是相同的，共阳极 LED 需对表 3 的段码取反。如果要显示小数点，则需在取出段码后，将其最高位置 1。实验二中，有特别说明要显示小数点，可以将编码+0x20，这是显示小数位的另一种方法。LED8.C 中，段码表是这样组织的：

```
u8 code t_display[]={                          //8段数字字库
//   0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
    0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,
// black  -    H    J    K    L    N    o   P    U     t    G    Q    r   M    y
    0x00,0x40,0x76,0x1E,0x70,0x38,0x37,0x5C,0x73,0x3E,0x78,0x3d,0x67,0x50,0x37,0x6e,
//   0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  -1
    0xBF,0x86,0xDB,0xCF,0xE6,0xED,0xFD,0x87,0xFF,0xEF,0x46};
```

图 30. LED8.c 中 LED 显示段码表

在程序段预制的 t_display 表，将不带小数点的 16 进制数放在偏移量 0 开始的位置，特殊符号段码放在偏移量 0x10 开始的位置，带小数点的十进制数段码放在偏移量 0x20 开始的位置。这是一种以存储代替计算的方式，在取段码前，只要使用正确偏移量，就可以取出带小数点数字的段码。其他符号的段码可以自行绘图推敲。

U5 的输出口线被连接到 8 位数码管各位的公共端 com1~com8，作为数码管的位驱动，低电平有效。一般芯片的灌电流能力都要远远大于拉电流能力，高亮 LED 数码管单段允许 10ma 级电流，公共端就有 100ma 级电流，这也是实验箱采用共阴极数码管的原因。当然，采用动态显示时，各口线的平均电流均要低于此值。

由图 27 和图 28 我们已经知道，Q0（COM1）~Q7（COM8）分别连接到两片 3461AS 的最左~最右各位，这也就是在 u8　LED8[8]; 显示缓冲区中，各单元与实际显示位置的对应关系。

3. LED 数码管的驱动程序

595 移位寄存器具有两级缓冲，也就是在移位过程中，输出锁存器中的数据不会改变，作为显示驱动芯片的好处是不会因为移位引起显示紊乱或暗影。

在剖析了 LED 数码管的驱动芯片和硬件连接后，再来看 LED 数码管的驱动程序就变得水到渠成了。

```
u8 code T_COM[]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};          //位码

/*************  IO 口定义      **************/
sbit      P_HC595_SER     = P4^0;     //pin 14      SER         data input
sbit      P_HC595_RCLK    = P5^4;     //pin 12      RCLk        store (latch) clock
sbit      P_HC595_SRCLK = P4^3;     //pin 11      SRCLK     Shift data clock
```

```
/*************  本地变量声明  **************/
u8   LED8[8];            //显示缓冲
u8   display_index=0;    //显示位索引
```

sbit 定义了软件模拟 SPI 的各信号，定义与前节所述硬件一致。

T_COM 定义了位码，也是以存储空间换移位时间。每次扫描使能其中一位，因为是共阴接法，所以实际使用时取出 T_COM 的值以后需要取反。

```
/***************** 向 HC595 发送一个字节函数 *****************/
void Send_595(u8 dat)
{
    u8  i;
    for(i=0; i<8; i++)
    {
        dat <<= 1;
        P_HC595_SER   = CY;
        P_HC595_SRCLK = 1;
        P_HC595_SRCLK = 0;
    }
}

/********************** 显示扫描函数 ************************/
void DisplayScan(void)
{
    Send_595(~T_COM[display_index]);                    //输出位码
    Send_595(t_display[LED8[display_index]]);    //输出段码

    P_HC595_RCLK = 1;
    P_HC595_RCLK = 0;                                   //锁存输出数据
    if(++display_index >= 8)      display_index = 0;   //8 位结束回 0
}
```

Send_595(u8)为字节传送函数，作用是向 595 发送一个字节数据，高位在前，低位在后，每准备一位数据到 SER 后，SRCLK 产生一个正脉冲，其上升沿产生一次 16 位移位，SER 数据进入 u6 最低位。循环 8 次后，一个字节的数据就被送入到 U6 移位寄存器，而 U6 移位寄存器中原先存放的数据则被推入 U5 中。

DisplayScan(void)为显示驱动函数。它根据 display_index 所指，取出位码取反，发送给 595，再取出 LED8 中对应的数据进行译码，发送给 595，原先发送的位码被推入 u5，新发送的段码保存在 u6 之中，两次发送分别完成了刷新位 LED 的段码和位码准备，接着给 RCLK 一个正脉冲，在其上升沿将移位寄存器中的数据锁入输出锁存器，完成该位显示更新。

显示驱动函数将位刷新过程分成移位和输出两步进行，也是配合了 595 芯片的两级寄存器，使得显示内容清晰无暗影。

每调用显示驱动函数一次，LED 数码管显示器上就有一位得到刷新，刷新的内容为 display_index 对应 LED8 缓冲区中的内容，刷新后，驱动程序更新 display_index 中的值指向下一个需要刷新的显示位。

调用 8 次显示驱动函数，即完成 LED8 位数码管的一次刷新，可称为一帧（frame）。当帧率超过 50fps 的时候，一般认为，因为人眼的视觉暂留，我们看到的场景是连贯的。在实际显示中，50fps 还有一定的闪烁感，这就是为什么护眼电视的帧率需要>100Hz（或 fps）的原因。

实验二示例中，使用 T0 产生 1ms 时间基准中断，用于刷新 1 位显示，一帧的刷新时间为 8ms，也就是达到了 125Hz 帧率，显示是十分稳定的。但一旦这个过程被打断，比如键盘中断的优先级高于 T0 时基刷新，而键盘处理又不能很快完成，就会出现显示不稳定甚至黑屏的情况。这种情况应该通过程序设计避免出现。

4. 行列式键盘和显示例程

例 1 读取实验箱行列键盘的编码，显示在 LED 数码管的末位上，编码须与键盘附近标示的数字相同。实验例程参见 lab3/exp2。其中使用了 STC 原厂例程，并做了一些细微的改动，为了方便同学理解，指导书对关键部分有详细的讲解。

首先，继续实验二模块化程序设计的思想，将项目下的文件根据功能进行组织，分为：LED8.c 显示驱动，key16.c 键盘扫描和 keydisp.c 主程序。

```
extern u8 KeyCode;       //给用户使用的键码, 1~16 有效
void Key_Init(void);
void IO_KeyScan(void);

/********************** 主函数 ************************/
void main(void)
{
    u8 i;

    Port_Init();
    T0_Init();
    Key_Init();

    for(i=0;i<8;i++) LED8[i] = DIS_BLACK;     //初始熄灭
    while(1) {
        if(F_KS) {                    //50ms，扫描键盘
            IO_KeyScan();
            F_KS=0;
        }
         if(KeyCode > 0) {            //有键按下，处理键
            LED8[7] = KeyCode-1;
            KeyCode=0;
        }
    }
}
```

主程序通过 Key_Init()函数进行键码和连发初始化，然后进入键盘扫描和键码处理循环。

F_KS 为 T0 定时器产生的 50ms 中断，用于键盘扫描，同时兼顾按键的软件消抖。当 F_KS 有效时，主程序调用 IO_KeyScan()函数进行键盘扫描，但是一次调用不会直接返回一个键码，否则由于机械抖动的存在，可能在按键的前后沿都会有多个键码产生。只有当前后 2 次调用（间隔了 50ms）产生的键码相同时，才能确认按键有些，并产生键码。

IO_KeyScan()返回时，将键码保存在全局变量 KeyCode 中，其值为 1~16，如果 KeyCode 为 0 表示无按键或者同时按下多键。键码减 1 即为键盘标示的 0~F 码，程序直接将它送入 LED8 的末位，用于 LED 刷新显示。

这个例程的关键是键盘扫描模块 key16.c 中的 IO_KeyScan 函数。STC 原厂例程的程序是对的，但是程序注释中的键盘布局图是错误的，正确的键盘连接见图 26，实际键盘布局应如图 31：行线连接到 P0 口的高 4 位，列线连接到 P0 口的低四位。

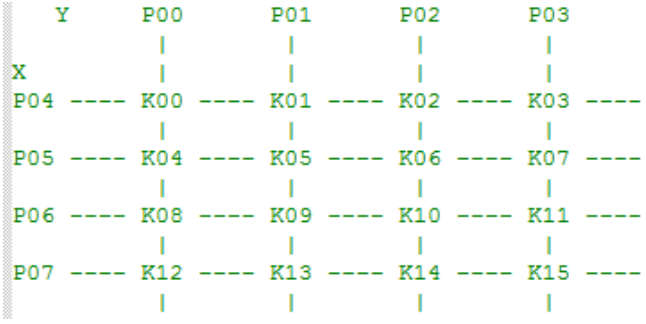


图 31. STC 实验箱 P0 口键盘连接

如果按键行列值为 0~3，则：键码=行*4+列。例程程序取值用 1~4，有：键码=（行-1）*4+（列-1），而 KeyCode 没有-1，是为了让返回值带有“有键（>0）”和“无键（==0）”属性，不用另设变量。

翻转法获取键码需要扫描键盘 2 次：一次确定行坐标，一次确定列坐标，两个坐标组合才能获得完整的键码。如果使用扫描法，则不管是扫描行线还是列线，都需要扫描四次才能确定键码。程序使用翻转法获取键码。

```
u8 code T_KeyTable[16] = {0,1,2,0,3,0,0,0,4,0,0,0,0,0,0,0};
void    IO_KeyScan(void)     //50ms call
{
    u8  i,j;

    j = IO_KeyState1;   //保存上一次状态

    P0 = 0xf0;  //Y低，读X
    for(i=0;i<60;i++);
    IO_KeyState1 = P0 & 0xf0;

    P0 = 0x0f;  //X低，读Y
    for(i=0;i<60;i++);
    IO_KeyState1 |= (P0 & 0x0f);
    IO_KeyState1 ^= 0xff;   //取反
```

图 32. 翻转法获取键码

局部变量 j 暂存上一个键值，全局临时变量 IO_KeyState1 保存当前键值。键码获取时，首先列线置零，读取按键所在行信息，置入变量高 4 位，然后翻转为行线置零，读取按键所在的列信息，置入变量低 4 位。行列信息都是有按键的位为 0，无按键的位为 1，比如 0x7E 表示低 4 行第 1 列有按键，为了方便查表，将其取反为 0x81。

```
    if(j == IO_KeyState1)   //连续两次读相等
    {
        j = IO_KeyState;
        IO_KeyState = IO_KeyState1;
        if(IO_KeyState != 0)    //有键按下
        {
            F0 = 0;
            if(j == 0)  F0 = 1; //第一次按下
            else if(j == IO_KeyState)
            {
                if(++IO_KeyHoldCnt >= 20)   //1秒后重键
                {
                    IO_KeyHoldCnt = 18;
                    F0 = 1;
                }
            }
            if(F0)
            {
                j = T_KeyTable[IO_KeyState >> 4];                           //有键
                if((j != 0) && (T_KeyTable[IO_KeyState& 0x0f] != 0))
                    KeyCode = (j - 1) * 4 + T_KeyTable[IO_KeyState & 0x0f];     //计算键码
            }
        }
        else    IO_KeyHoldCnt = 0;
    }
    P0 = 0xff;
```

图 33. 处理连发和产生键码

IO_KeyScan()函数还具有长按连发功能。如果本次按键与上次相同，说明没有抖动，则分两种情况，如果键码为 0，说明没有按键，连发计时器 IO_KeyHoldCount 复原，否则应有键码产生。j 暂存上一个保存的键码 IO_KeyState，IO_KeyState 取当前键码，51 单片机 PSW 中的用户位 F0 作为是否产生键码的标志，如果前面没有按键，那按下就会直接产生一个键码（可防止按键迟钝），否则，需要根据长按情况确定是否产生连发，首次连发计时器 IO_KeyHoldCount 的值为 20，也就是长按 1 秒后才能产生连发，接着此值被设为 18，也就是如果继续保存长按不放，后面会以 1 秒 10 键的方式产生键码连发。

最后解释一下 T_KeyTable 表，它是通过查表法将按键的行列置位值转化为对应的行列序号，比如 0x81，行值 0x8 查得表中的行号为 4，列值 0x1 查得表中的列号为 1，这样由(4-1)*4+1 得到左下角按键键码 13，显示为“C”。其他类推。

例 2.按一次 sw17 学号开始循环左移，再按一次停在当前位置；按一次 sw18 学号开始循环右移，再按一次停在当前位置。左移过程中，按右移键，不停止，直接从当前位置开始循环右移，反之亦然。学号首末位数字间空一格。实验例程参见 lab3/exp1。

根据实验二，左键 SW17 对应外部中断 0，右键 SW18 对应外部中断 1。为防止中断执行时间太长，干扰到其他中断事件的快速响应，一般的方法是，在中断内设标志，表明中断事件已经发生，如 F_L 表示左键，F_R 表示右键，在主程序中，再根据这些事件的紧急情况决定处理的优先级别。这只是一种编程的习惯。

```
062 void Delay20ms()        //@22.1184MHz
063 {
064     unsigned char i, j, k;
065
066     _nop_(); _nop_();
067     i = 2;
068     j = 175;
069     k = 75;
070     do  {
071         do  {
072             while (--k);
073         } while (--j);
074     } while (--i);
075 }
076
077 /******************** INT0中断函数 ************************/
078 void INT0_int (void) interrupt 0      //进中断时已经清除标志
079 {
080     F_L^=1; //乒乓键
081     Delay20ms();
082 }
083
084 /******************** INT1中断函数 ************************/
085 void INT1_int (void) interrupt 2      //进中断时已经清除标志
086 {
087     F_R^=1;     //乒乓键
088     Delay20ms(); //软件延时消抖动
089 }
```

图 34. SW17，SW18 外部中断函数

中断按键也需要消抖。在例程中，将 20ms 的软件延时函数放在中断服务程序中，用于软件消抖。在这个比较简单的程序中，如此做是没有问题的。如果程序复杂，可能会引起两个问题，一个是，增加了中断处理时间，使其他事件不能及时响应，另一个是，如果两个中断不同级别，可能会引发 Delay20ms()被重入。

实验一有同学反映软件延时不准，有两个原因：1，烧写时用的 IRC 频率不对，要知道，软件延时对应机器周期 T。2，确实算得不准。本示例用 STC 软件计算器得到 20ms 精确软件延时，见图 35。

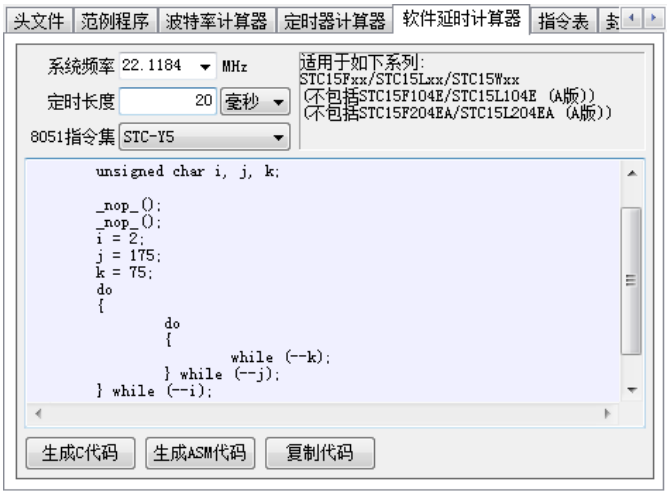


图 35. STC 官方软件定时计算器

```
u8 ASC2num(u8 asc)
{
    if(asc<='9'&&asc>='0')
        return asc-'0';
    else return DIS_BLACK;
}

/********************** 显示数串函数 ************************/
void ReflashDisp(void)
{
    u8 i;

    for(i=0;i<8;i++)
        LED8[i] = ASC2num(disp_string[(disp_index+i)%StrLen]);
```

图 36. 字符转数字和产生循环字符串

ReflashDisp()函数根据 disp_index 指针刷新显示内容。显示屏是 8 位 LED，一次只能显示 8 位数字。取出字符串 disp_string 中的字符后，通过调用 ASC2num 得到对应的数字，如果取出的不是数字字符，则应补空，防止产生乱码。StrLen 为调用 string.h 中的 strlen 函数得到的字符串长度。disp_index+i 可能会超出字符串 disp_string 的末位，使用取模将指针回环，形成一个 StrLen 长度的循环字串。

```
    ReflashDisp();    //显示手机号
    while(1) {
        if(F_L) {
            switch(F_Dir) {
                case 0: break;
                case 1: F_Dir=0; break;
                case 2: F_Dir=1;
            }
            F_L=0;
        }
        if(F_R) {
            switch(F_Dir) {
                case 0: F_Dir=1; break;
                case 1: F_Dir=2; break;
                case 2: break;
            }
            F_R=0;
        }

        if(F_T0) {
           F_T0=0;
            if(++i >= 1000) {  //1秒到
                i=0;
                disp_index=(disp_index-F_Dir+1+StrLen)%StrLen;
                ReflashDisp();
            }
        }
    }
```

图 37. 字符串连续移动处理

主程序中，通过事件循环，处理按键和定时器软标志，如图 37，有：F_L 左键事件、F_R 右键事件和 F_T0 定时事件。

F_Dir 表示移动方向，0 左移，1 静止，2 右移。左键的作用是：如果原先在 0 左移状态则不起作用，如果原先是 1 暂停，则开始左移，如果原先处在右移状态 2 则暂停。右键作用类似不再赘述。

定时事件 F_T0 每 1ms 产生一次，如实验二。计数到 1000 次，也就是 1s，按照 F_Dir 确定的移动方向产生一次数字串移动。移动主要是在 disp_string 字串环中，确定下一个 disp_index 的位置，F_Dir-1 将（0，1，2）映射到（-1，0，1）正好对应了 F_Dir 的增量，在正整数范围内取模，+StrLen 不影响取模结果。

例 3.按键改时的 24 小时时钟。要求的时钟设置方式与 STC 官方例程不一样，难度比例程更大。有了 LED 显示驱动和独立、行列键盘程序，相信有兴趣的同学多花点时间一定能做出来。实验不提供源代码，仅在学在浙大上提供.Hex 文件，供同学们下载和参考。如有需要，改天会在本指导书中补充流程图。

- **练习与思考：**

1. 你还熟悉哪些人机接口输入方式？试查阅编码器飞梭原理。
2. 你还熟悉哪些人机接口输出方式？试查阅 OLED 工作原理。
3. 实验 1 独立键盘查询方式与实验二使用的外中断的异同？
4. 根据动态显示原理，分析酒店门头屏移动显示的工作原理。
5. 试用扫描法改写按键译码程序，比较与实验例程的区别。
6. 如何让 24 小时时钟的修改方式更合理，用户体验更好？
7. 如何设计出更美观、更准确、更省电、掉电不停的时钟。

实验四

- **实验目的：**熟悉微机 ADC/DAC 工作方式

  熟悉 STC 片内 ADC 的原理和接口

  自学了解 ADC 键盘的工作原理

  自学了解温敏电阻 NTC 的特点

  熟练掌握 STC15 片内 ADC 的使用

  熟悉串行通信的概念和工作原理

  熟练掌握使用 UART 进行串行通信

  了解上下位机协同现场测控的方法

- **实验内容：**

  1） 读取 ADC 键盘编码，以 16 进制方式显示在 8 位 LED 数码管的末位，码值与按键附近的数字相同。

  2） 使用 STC15 片内 BGV 基准电压测量当前电源电压值，和板上 NTC 温度值，同时显示在 LED 数码管上。

  3） 将 NTC 温度值通过 UART 串口传送给 PC 机实时显示、画出温度曲线，并保存一段时间的温度数据。

  4） PC 机通过串口给单片机发指令，改变实验三中学号循环移位的移动方向和速度，设 0.5S、1S 和 2S 三档速度。

  5） 将实验三 24h 时钟的当前时间，通过串口发送给 PC 机。

  6） 使用 PC 机实时修改实验 3 中 LED 显示的 24h 时钟时间。

7） （选做）PC 机给 24h 时钟设定 3 个闹钟，闹钟时间可在单片机中通过 SW17 查询。当闹钟时间到时，在下位机进行现场警示，在上位机进行消息提醒。

- **实验步骤：**

1. ADC 键盘的构成和工作原理

在很久以前，根据电子系统按键数量的多少，聪明的工程师分别使用独立键盘和行列式键盘的方式，使用有限的 MCU 并行口线，解决了多键输入响应问题，包括上一个实验同学们练习过的键盘编码和键处理。后来，针对键盘矩阵，为了节省主 MCU 口线，还设计了专用的键盘扫描芯片，如 7279，或者使用单独的从 MCU 来进行键处理，如电脑键盘里的 8048。

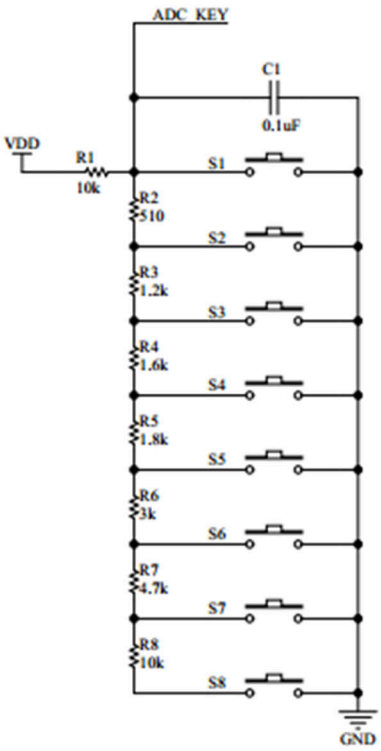


图 38. 某 ADC 键盘

随着高性能 ADC 大量集成到 MCU 中，也随之出现了 ADC 键盘。其特点是无论按键多少，仅占用一个模拟通道。

某款嵌入式开发板上的 ADC 键盘，共有 8 个按键 S1~S8，见图 38。上分压电阻 $R_{上}$=10K，S1~S8 对应的下分压电阻 $R_{下}$分别为 0，510，1710，3310, 5110，8110，12810，22810，则有分压值：$V_o=V_{dd}*R_{下}/(R_{上}+R_{下})$。

该款嵌入式系统使用 STM32 M3，内置 12 位 ADC，满量程转换值 4096，

各档理论转换值，也就是按键编码应为：0，198，598，1019，1385，1834，2300，2847。从中可知，ADC 键盘的设计，需遵循使按键 ADC 编码间保持一定间隙，才能有效区分不同按键。好的设计是兼顾了均匀间隙和使用标称值电阻分压。

STC 实验箱的 ADC 键盘位于实验三行列式键盘的上方，有 16 个按键，其电路原理如图 39。

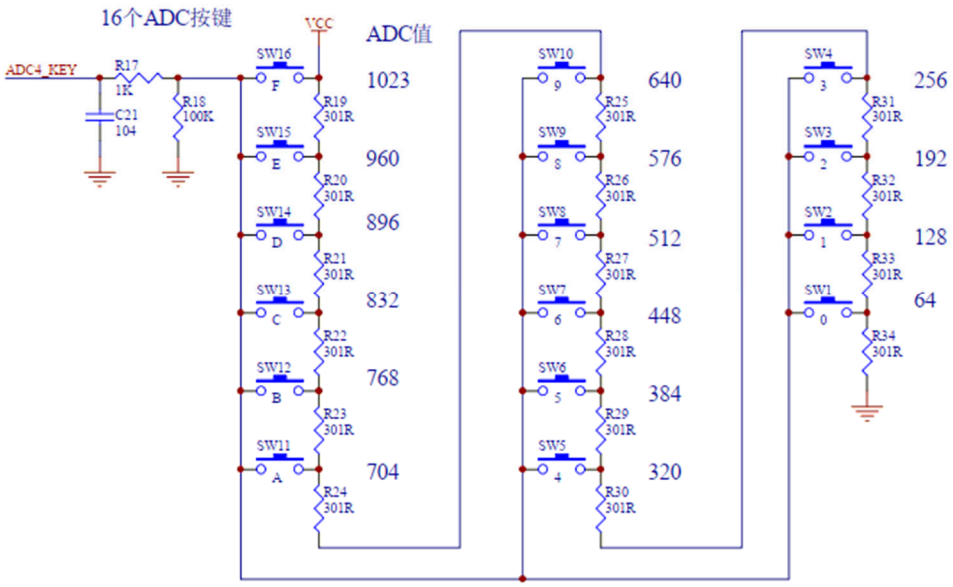


图 39. STC 实验箱 16 按键 ADC 键盘

由图可见，ADC 键盘没有设置额外的固定分压电阻，采用了同值电阻 300 欧姆均匀分压方式。R17，R18 和 C21 构成硬件滤波器用于按键消抖。

STC15 片内有 10b 的 ADC，满量程 1024。分压间隔为 64。ADC 键盘的读取见 STC 例程，程序设计中为了防止抖动也设置了专门的措施，如连续获取按键，只有当键值差别比较小时才确认为有效键，键值的获取设计了±16 字的容差，为键码间隙的一半，保证了按键判别的准确性。

2. 负温度系数热敏电阻 NTC

工业温度传感器根据工作原理有热电阻、热电偶、红外热释电、半导体结效应和带隙式温度传感器等，已经有多种被制作成 IC 方式供应的集成温度传感器产品，在方便性、准确性、稳定性和可替换性上都有着很大的优势，如常见的模拟输出的 LM35，数字输出的 DS18B20 等。

尽管现在集成温度传感器大行其市，但也有一些限制，如高低温测量范围、特殊测量、高精度测量等方面还有欠缺；在需要大量部署测温点，对测温要求也不是很高的场合，集成温度传感器成本也相对比较高。

热敏电阻式温度传感器商业化已有近 100 年，一直以来，以性价比极高、灵敏度高、体积小的特点被广泛应用于后一种场合，如：电脑主板、工业温控器、电源逆变器、干式变压器等。热敏电阻一般有两种，PTC 正温度系数器件和 NTC 负温度系数器件，PTC 用于过流、过载、过热、过温保护，如充电器、空调、开关电源等的保护开关。

NTC 一般用于抑制浪涌电流，温度测量及控制。开机瞬间，电路对整流桥后的电容充电，会出现很大的浪涌电流，NTC 冷电阻可用于抑制浪涌大电流，开机后，电流流过 NTC 导致 NTC 发热，电阻下降到一个恒定的小阻值，对电源供电不产生影响。比如手机电池保护，办公自动化设备的电路保护等。NTC 作为测温元件，也常用于电子温度计、空调、冰箱、饮水机等的电器中。

热敏电阻长期被广泛应用的原因是显而易见的。首先，热敏电阻的制作成本低，通过混合镍、锰、钴和铁等氧化物烧结而成陶瓷半导体，工艺成熟适合批量生产。然后，热敏电阻的检测灵敏度高，相比热电偶使用铂等贵金属材料，每 1° C 变化只有几个百分点的阻值变化，NTC 随温度变化表现出很大的阻值变化，根据掺杂的不同，一般每 1℃温度改变能产生 1~5%的阻值变化。

根据迈克尔法拉第发现的热敏原理，NTC 的电阻随温度变化公式为： $R_T = R * EXP(B*(1/T-1/T_0))$。其中： R_T 是热敏电阻在 T 温度下的阻值，R 是热敏电阻在 T_0 常温下（一般是 25 摄氏度）的标称阻值，B 值是热敏电阻的灵敏度系数，与掺杂成分有关，某一种 NTC 的 B 为固定值。T 和 T_0 用的是绝对温度，开尔文温度 T=273.15+摄氏度。比如：NTC 热敏电阻 MF52 ,10K@25,B= 3950, 1%，10K 是 25 度时的阻值，当 t=0 度时，T=273.15，T_0=298.15，有：R_T=10000*exp(3950*(1/(273.15)-1/298.155))≈33620.6 欧姆。

在实际使用中，NTC 一般需要使用一个分压电阻，将阻值变化转化为输出电压的变化来进行测量。STC 实验箱使用了型号为 MF52 的 NTC 作为温度传感器，上拉电阻 R6 为 1%精度的 10K 金属膜电阻，作为 NTC 的分压电阻，如图 40，输出接到 MCU 的 P1.3(ADC3_NTC)。需要一提的是，由于常用碳膜电阻的精度低（10~20%），温度系数相对比较大（5-20%%），而且一致性不好，一般不适合作为测量分压电阻。

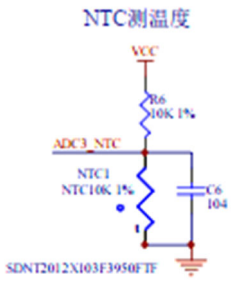


图 40. 实验箱 NTC

3. STC 的 ADC 通道配置

STC15 系列单片机的片内 ADC 是 10 位 8 通道逐次逼近型 AD 转换器。ADC 通道与 P1 端口复用，P1.7-P1.0 对应 ADC0~ADC7，速度可达到 300KHz（30 万次/秒）。

表 4. 与 ADC 相关的寄存器

符号	描述	地址	位地址及其符号 MSB LSB								复位值
P1ASF	P1 Analog Function Configure register	9DH	P17ASF	P16ASF	P15ASF	P14ASF	P13ASF	P12ASF	P11ASF	P10ASF	0000 0000B
ADC_CONTR	ADC Control Register	BCH	ADC_POWER	SPEED1	SPEED0	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0	0000 0000B
ADC_RES	ADC Result high	BDH									0000 0000B
ADC_RESL	ADC Result low	BEH									0000 0000B
CLK_DIV PCON2	时钟分频寄存器	97H	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0	0000 0000B
IE	Interrupt Enable	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0	0000 0000B
IP	Interrupt Priority Low	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000 0000B

P1ASF 为模拟通道配置寄存器，某一位设为 1，则该位用于 AD 通道，不作为 AD 通道的位设为 0，仍可作为普通 IO 口使用。

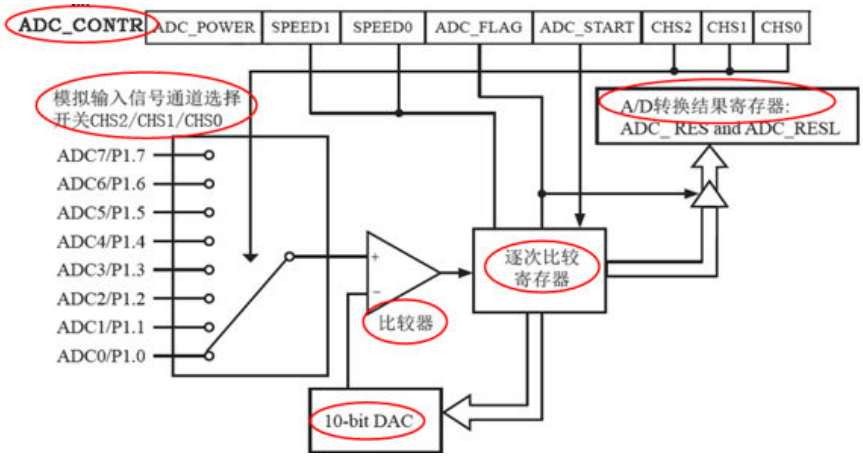


图 41. STC15 片内 ADC 结构

片内 ADC 的结构如图 41，主要通过 ADC 控制寄存器来选择通道，确定转换速率，启动转换和查询转换状态。

表 5. ADC 控制寄存器 ADC_CONTR

位号	B7	B6	B5	B4	B3	B2	B1	B0
位名称	ADC_ POWER	SPEED1	SPEED0	ADC_ FLAG	ADC_ START	CHS2	CHS1	CHS0

ADC 控制寄存器中，B7 为 ADC 电源控制位，0 关闭 ADC 电源，1 打开 ADC 电源。设为 0，有利于在不使用 ADC 时降低功耗。片内 ADC 部件默认是关闭的，如果需要使用，为保证转换准确，启动 ADC 转换前一定要预先要置位 ADC_POWER 打开 ADC 电源，设置 ADC 转换速率，待电源稳定后，再启动 ADC 转换。

B6/B5（SPEED1/SPEED0）位决定 ADC 转换速率，如表 6。

表 6. ADC 转换速率

SPEED1	SPEED0	A/D 转换所需时间
1	1	90 个时钟周期转换一次，设 CPU 工作频率 27MHz， A/D 转换速度 300KHz（27MHz÷90）
1	0	180 个时钟周期转换一次
0	1	360 个时钟周期转换一次
0	0	540 个时钟周期转换一次

B4（ADC_FLAG）为 ADC 结束标志位。ADC 每完成一次转换，该标志就会自动置 1，如果允许中断（EADC=1），该标志会引发一次 ADC 中断。否则，可以查询此标志以确定转换结果是否有效。该标志由软件清 0。

B3(ADC_START)为 ADC 启动控制位，设置为“1”时，开始转换，转换结束该位自动清 0。B2~B0 用于选择 ADC 通道，如表 7。

表 7. ADC 通道选择

CHS2	CHS1	CHS0	模拟输入通道选择
0	0	0	选择 P1.0 作为 A/D 输入来用 ADC0
0	0	1	选择 P1.1 作为 A/D 输入来用 ADC1
0	1	0	选择 P1.2 作为 A/D 输入来用 ADC2
0	1	1	选择 P1.3 作为 A/D 输入来用 ADC3
1	0	0	选择 P1.4 作为 A/D 输入来用 ADC4
1	0	1	选择 P1.5 作为 A/D 输入来用 ADC5
1	1	0	选择 P1.6 作为 A/D 输入来用 ADC6
1	1	1	选择 P1.7 作为 A/D 输入来用 ADC7

ADC 转换的结果可通过读取 ADC_RES 寄存器中存放的高位和 ADC_RESL 寄存器中存放的低位组合获得。STC15 对 ADC 转换结果有两种存储格式，分别为左对齐和右对齐存储。对齐方式由时钟分频寄存器 CLK_DIV 中的 ADRJ 位来控制，如表 8。

表 8. ADC 结果对齐方式控制

Name	7	6	5	4	3	2	1	0
时钟分频寄存器	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	Tx2_Rx2	CLKS2	CLKS1	CLKS0

当 ADRJ=0 时，ADC 转换结果采用左对齐方式存放，如表 9。单片机复位后默认 ADC 结果左对齐。

表 9. ADC 结果左对齐

位号	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	ADC_RES9	ADC_RES8	ADC_RES7	ADC_RES6	ADC_RES5	ADC_RES4	ADC_RES3	ADC_RES2
ADC_RESL							ADC_RES1	ADC_RES0

此时，8 位精度 ADC 值可直接取 ADC_RES。10 位全精度 ADC 值为：(ADC_RES<<2)|ADC_RESL。

当 ADRJ=1 时，ADC 转换结果采用右对齐方式，如表 10。

表 10. ADC 结果右对齐

位号	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES							ADC_RES9	ADC_RES8
ADC_RESL	ADC_RES7	ADC_RES6	ADC_RES5	ADC_RES4	ADC_RES3	ADC_RES2	ADC_RES1	ADC_RES0

此时，8 位精度 ADC 值为：(ADC_RES<<6)|(ADC_RESL>>2)。10 位全精度 ADC 值为：(ADC_RES<<8)|ADC_RESL。

一般来说，8 位 AD 使用 ADRJ=0，10 位 AD 使用 ADRJ=1。

如前所述，在中断允许条件下，ADC 结束可以由 ADC_FLAG 触发中断。与中断相关的寄存器有 IE 和 IP，如表 11。

表 11. ADC 中断相关寄存器

SFR	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0
IP	B8H	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

这两个寄存器是可位寻址的特殊功能寄存器，可设置 ADC 中断允许和中断优先级。

STC15 片内 ADC 的编程要点：

1） 打开 ADC 电源
2） 延时等待 ADC 电源电压稳定
3） 将 P1 口相应口线配置为 AD 输入口
4） 配置转换结果对齐方式
5） 配置中断方式
6） 设置 ADC 通道并开启转换
7） 等待 ADC 转换结束，清除 ADC_FLAG
8） 读取 ADC 值并转化为对应输入值

4. STC 的内部 BGV 基准

ADC/DAC 电路需要一个不受电源电压、温度影响的参考电压，各种精密电源也需要使用一个准确的电压作为所有输出电压的参考点，这个参考电压就是基准电压。基准电压电路中，由 Robert Widla 在 1971 年发明的带隙电压基准源（Bandgap voltage Reference，或 BGV/BGR）是一种常用的基准源。BGV 由于具有很高的精确性和稳定性，常用作高精度的参考电压。

所谓带隙，指的是半导体导带的最低点至价带最高点之间的压差。最早发明的 BGV，其输出电压约为 1.25V，与硅的带隙电压相仿，因而被称为带隙基准。随着技术的发展，带隙基准也发

展出多种多样的结构，有些 Bandgap 的输出电压与带隙电压并不一致。

带隙基准源这种优异的温度稳定性，并不是与生俱来的，而是来自于巧妙的电路设计。其核心思想为：既然温度变化不可避免，那就补偿温度带来的变化。不同材料或电路，具有不同的温度系数，有的为正温度系数，有的为负温度系数，合理配置正温度系数和负温度系数的材料，使温度带来的电压变化相互抵消，就可以得到一个零温度系数的基准电压。

尽管 STC 官网没有给出 STC15 片内 ADC 的具体电路，但从 STC15.pdf 大致可以知道，其 ADC/DAC 的电压基准 V_{ref} 采用的是电源电压 V_{dd}。我们知道，对于同样一个 V_{in}，当 V_{ref} 变化时，得到的 ADC 结果是不同的，10 位 AD 有：$ADC_{in}=1024*V_{in}/V_{ref}$。

只有确定了 V_{ref} 的大小，才能根据 ADC 结果得到准确的 V_{in}。

STC15 内部集成了一个 BGV 电压基准，这个电压很稳定，约为 1.25V，不会随芯片的工作电压改变而变化。ADC 的第 9 通道是用来测试内部 BGV 参考电压的。通过在标准 5V 条件下和实际工作状态下两次测量内部 BGV 基准电压，即可反推出实际工作条件下的电源电压，用于校准实际测量值。

设 $V_{bg}\approx1.25$，在 5.0V 下使用 10 位 ADC 测量 BGV 基准得到 GBV5，在实际电源电压 V_x 下测得 GBV 基准值为 BGVX，则有：$BGV5/1024= V_{bg}/5.0$，$BGVX/1024= V_{bg}/V_x \rightarrow V_x = 5.0 *BGV5 / BGVX$，

V_{in}=5.0 *BGV5 / BGVX *ADC_{in}/1024，V_{in} 即为精确的输入电压测量结果。

STC15 官方还提供了另外一种测量方法。即在不知电源电压 V_x 的情况下连续测量两次，第一次测量基准电压 V_{bg}，第二次测量输入电压 V_{in}，有：$ADC_{bg}/1024=V_{bg}/V_x$，$ADC_{in}/1024=V_{in}/V_x$，两式相除得到：$ADC_{bg}/ADC_{in}=V_{bg}/V_{in}$ ，$V_{in}= V_{bg}* ADC_{in}/ADC_{bg}$，其中 V_{bg} 用近似值 1.25 代入。

STC15 内部的带隙基准连接在所谓的 ADC 第 9 通道上，然而 ADC_CONTR 只有前 8 个通道可以选择，如何测量第 9 通道呢？stc15.pdf 给出了测量方法：首先将 P1ASF 设置为 0，即关闭 P1 口的模拟功能，然后通过 ADC 转换读取第 0 通道的值，其结果即为第 9 通道内部 BandGap 基准的参考电压值。

此节内容是对 STC15.pdf 和参考书相关章节的补充。

5. 用基准测量电源电压

片内 BGV 基准带隙电压为：V_{bg}≈1.25，测量当前的 BGV 值，$ADC_{bg}/1024=V_{bg}/V_{dd}$，有：$V_{dd}$ =1.25*1024/ ADC_{bg}=1280/ ADC_{bg}。

```
for(j=0,i=0; i<16; i++)                 //采样 16 次取平均
    j += Get_ADC10bitResult(8); //内部 BGV，ADC  第 9 通道
j = (u32)128000UL*16 / j;           // Vdd = 1280 / ADC，计算放大 100 倍
LED8[0] = j / 100 + DIS_DOT;        //显示 MCU 电压，百位后加小数点
LED8[1] = (j % 100) / 10;
LED8[2] = j % 10;
```

测量调用 Get_ADC10bitResult 函数进行第 9 通道片内 BGV 电压的 ADC 转换。结果作平滑滤波，在计算电源电压时，放大 100 倍，用整数表示 2 位小数，在显示的时候在百位后加上小数点。

Get_ADC10bitResult 函数如图 42，清除前一次转换结果，设置通道启动 ADC 转换，等待转换结束标志 ADC_FLAG 置位，根据左对齐方式获取 10 位 ADC 值并返回。因为 CLK_DIV 中的 ADRJ 没有设置，默认为 0 即结果左对齐。通道 9 和通道 0 共用一个通道号，只是在进行第 9 通道 ADC 时，需要将 P1ASF 设置为全 0，而进行其他 8 个通道 ADC 时，对应的通道 P1ASF 位需要设为 1。

```
/******************查询法读一次ADC结果******************/
u16 Get_ADC10bitResult(u8 channel)  //channel = 0~8,8为内部BGV
{
    ADC_RES = 0;
    ADC_RESL = 0;

    ADC_CONTR = (ADC_CONTR & 0xe0) | 0x08 | (channel&7);    //start the ADC

    while((ADC_CONTR & 0x10) == 0)  ;   //wait for ADC finish
    ADC_CONTR &= ~0x10;                 //清除ADC结束标志
    return  (((u16)ADC_RES << 2) | (ADC_RESL & 3));
}
```

图 42. Get_ADC10bitResult 函数

6. 通过外接 NTC 测量环境温度值

如前所述，迈克尔法拉第效应表明，NTC 的阻值随温度变化呈现一种指数形式，可以用公式 $R_T = R * EXP(B*(1/T-1/T_0))$来计算。我们知道，MCU 的乘除运算比较慢，进行浮点数运算更慢，以 e 为底的指数运算需要调用 math 库函数，执行效率非常低。而且使用公式只能得到 B 值恒定的 2 参考测温点之间的温度值。

在有限温度范围内，当前温度的 NTC 阻值，一般通过查分度表的方式获得。分度表是 NTC 厂家实际校准后提供的，每一个整数温度下 NTC 的阻值。对于精度要求高于 1 摄氏度的应用，也就是 ADC 值介于 2 个相邻分度值之间，则可以通过线性插值的方式得到温度的小数值。厂家提供的分度表如图 43，使用分度表，是工程上常用的 NTC 测温方法。

NTC 热敏电阻 R-T参数表
阻值@25度：10KΩ 允许公差：1%
B值25/50℃：3950

T(温度)	Rmin	Rnom	Rmax
-40	268250	281160	294650
-39	252370	264320	276810
-38	237540	248610	260170
-37	223690	233950	244650
-36	210750	220260	230170
-35	198640	207460	216640
-34	187310	195490	204010
-33	176710	184290	192190
-32	166770	173810	181140
-31	157460	164000	170790
-30	148730	154800	161110
-29	140460	146100	151950
-28	132730	137960	143390
-27	125480	130350	135390
-26	118690	123210	127890
-25	112320	116530	120880

图 43. MF52 分度表

示例程序参考 STC 官方实例，使用了查分度表加插值的方式得到 0.1 度精度的温度值。官方实例提供的分度表为从-40 度到 120 度共 161 项 12b 温度采样值表，对应于厂家提供的典型电阻值。其中-40 度放在第 0 项，与 0 度有 40 的偏移，40 之前为负温度，之后为正温度。温度值计算如图 44，输入为 12b 采样值，输出 0~1600，除了偏移还放大了 10 倍，对应于-40~120 度。

```
// 计算结果：0对应-40.0度，400对应0度，650对应25.0度，最大1600对应120.0度.
// 为了通用，ADC输入为12bit的ADC值.
/*********************************************/
#define     D_SCALE     10      //1位小数，为了避免出现小数，结果放大倍数
u16 get_temperature(u16 adc)
{
    u16 code *p;
    u8  j,k,min,max;

    adc = 4096 - adc;   //Rt接地
    p = temp_table;
    if(adc < p[0]||adc > p[160]) //超出量程
        return (0xffff);

    min = 0;        //-40度
    max = 160;      //120度

    for(j=0; j<8; j++)  //对分查表，2^8可查256项>160
    {
        k = (min + max) / 2;
        if(adc < p[k]) max = k;
        else           min = k;
    }
    if(adc == p[min]) return min * D_SCALE;
//    else if(adc == p[max]) return max * D_SCALE;
    else // min < temp < max   max=min+1
        return min* D_SCALE+(adc - p[min]) * D_SCALE / (p[max] - p[min]);
}
```

图 44. 温度值计算函数

分度表对应于 NTC 接电源分压电阻接地的情况，而实际电路中，NTC 和分压电阻的接法正好相反，取补码正好转换为分度表相应的电路接法应该获得的采样值。分度表是单调递增的表格，函数首先判断输入是不是在分度表温度范围内，如果超出了范围，直接返回错误值 0xFFFF。

分度表为 160 项有序递增表格，对输入值进行二分查找，每次缩小查找区间，最多 8 次（可查 256 项）一定能够找到输入值，或者确定输入值所在的相邻温度区间[min,max=min+1)。如果找到，直接返回扩大 10 倍的温度值；否则，在区间内进行线性插值，因为区间代表了 1 度，乘以 10 插值且保留整数，即保留了小数点后一位温度，显示时将小数点放在 10 位后即可。

```
for(j=0,i=0; i<4; i++) j += Get_ADC10bitResult(3);  //通道3, NTC
j = get_temperature(j); //计算温度值

if(j >= 400)     F0 = 0, j -= 400;       //温度 >= 0度
else             F0 = 1, j  = 400 - j;   //温度 <  0度
LED8[4] = j / 1000;     //显示温度值
LED8[5] = (j % 1000) / 100;
LED8[6] = (j % 100) / 10 + DIS_DOT;
LED8[7] = j % 10;

if(LED8[4] == 0)     //可能是3位或者2位温度
    if(LED8[5] == 0)  {
        LED8[5] = DIS_BLACK;
        if(F0)  LED8[5] = DIS_;     //显示-
    } else {
        LED8[4] = DIS_BLACK;
        if(F0)  LED8[4] = DIS_;     //显示-
    }
```

图 44. 温度值计算函数

如图 14，主程序令 10b 的 ADC 重复采样 4 次，虽然精度仍然是 10b，但是数值范围被扩大为 12b。调用温度计算函数获得有偏移 400 且放大了 10 倍的温度值后，符号保存在 F0 中，温度值去偏移求绝对值，提取各位显示在 LED8 的低 4 位。显示时，去掉前导 0，如测到零下温度(F0=1)则在合适位置加上负号。

7. UART 串口发送数据到上位机

STC15 的串行通信初始化程序如图 45。

```
/*************** T1定时器初始化函数 *****************/
void T1_Init(u16 brt)
{
    #define Timer_Reload   (65536UL - MAIN_Fosc/4/brt) //Timer1 波特率发生器

    AUXR |=  (1<<6);    //T1x12=1: Timer1 1T, 16 bits timer auto-reload
    TH1 = (u8)(Timer_Reload / 256);
    TL1 = (u8)(Timer_Reload % 256);
    TR1 = 1;    //Timer1 run
}

void Uart_Init(u16 brt)     //19200bps@22.1184MHz
{
    SCON = (SCON & 0x3f) | 0x40;    //UART1模式, SM0 SM1=01: 8位数据,可变波特率
    REN = 1;    //允许接收
    AUXR &= ~0x01;      //S1 BRT Use Timer1;
    T1_Init(brt);
}
```

图 45. 串行口初始化

初始化函数中，SCON 的高 3 位 SM0/SM1/SM2 中，SM2 用于多机通信，SM0 和 SM1 用于选择串行通信模式，00 是移位寄存器，02/03 是 9 位数据，01 是 8 位数据可变波特率通信，波特率由定时器决定。B4 位 REN=1 允许接收，全双工串口可以收发同时进行，SBUF 收发缓冲寄存器同名不同物理单元。AUXR 的 B0 为 0，选择 T1 作为波特率发生器。

根据手册，STC15 系列 MCU 的通信波特率为定时器溢出率的 4 分频。设置 AUXR 的 B6=1，T1 的输入时钟不分频，为系统主频 MAIN_Fosc（或 Fosc）。溢出率为 Fosc/(65536-$T_{初值}$)，则有自动波特率计算：brt=Fosc/ (65536- $T_{初值}$)/ 4， $T_{初值}$=65536- Fosc/4/brt。

我们知道，单片机串口收发有中断和查询两种方式。中断效率高，CPU 在通信的同时还可以进行数据采集、数值处理、本地显示和控制等。实验例程 lab4\exp1 串口通信采用了中断方式。MCS-51 将串行发送和串行接受归并到同一个串行中断（中断 4）下，通过在服务程序中根据 TI 和 RI 标志判断是何种事件发生。

例程的中断服务程序如图 46 所示。

```
29  void S1_int (void) interrupt UART1_VECTOR
30  {
31      if(RI)
32      {
33          RI = 0;
34          if((RX_Ptr+1)%UART_BUF_LENGTH==RX_Get) return;  // 缓冲满
35          RX_Ptr=(RX_Ptr+1)%UART_BUF_LENGTH;
36          RX_Buffer[RX_Ptr] = SBUF;
37      }
38
39      if(TI)
40      {
41          TI = 0;
42          TX_Busy = 0;
43      }
44  }
```

图 46. 串行中断服务程序

UART1_VECTOR 为在 gdef.h 中定义的中断号 4。串行中断的中断请求标志 TI 和 RI，在进入中断服务后不会自动硬件清除，必须由串行中断服务程序软件清除。

发送结束时，也就是发送了一个 8bit 数据，TI=1 进入中断，中断服务程序清除 TX_Busy 软标志，通知主程序发送已完成，可以进行新数据发送。接收数据后，RI=1 进入中断，考虑到主程序可能在做其他工作，不一定能及时取走串行数据，双缓冲也无法保证接收后面数据后不冲掉 SBUF 中的前一个数据。

串行缓冲队列是一种行之有效的解决串行接受数据丢失的方法。队列可以开辟在片内 RAM 低端 data（最快）和高端 idata（很快），也可以开辟在片外 XRAM 区（较大），队列长度应根据芯片物理内存的容量和串行数据的处理要求进行权衡，一般在串行速率不高、突发数据量不是特别大的情况下，30~100 字节比较合适。例程仿效 STC 实例，在 XRAM 中开辟了 50 字节的环形缓冲队列 RX_Buffer，以保证数据有 50*11/brt 秒以上的缓冲时间。队列的具体实现策略，在后面有一节会有比较详细的描述。

MCS-51 的串行口允许我们将数据用数据流的方式逐字节往 SBUF 推送，如果要进行格式化数据传输，也可以用 sprintf 预先在字符串中将数据进行格式化、规范化，以获得需要的效果。这里介绍另外一种直接向串口格式化输出数据的方法。

资料表明，Keil C 函数库中的 printf 打印字符串时，调用了 putchar 进行单字符的输出，而 scanf 读取字符串时，调用了 _getkey 进行单字符输入（getchar()只是为了对应 putchar，对函数_getkey 所做的宏定义）。只要改变和重载底层的字符输入输出函数，就可以将 printf 和 scanf 的标准输入输出设备重定向为 UART 串口、IIC 总线、CAN 总线甚至各种液晶屏模块。

各种开发实践表明，C51 虽然在函数库里封装了 putchar 函数和_getkey 函数，只要根据自己的设备重写此两个函数，在链接的时候，编译器会选择使用当前目录下的函数覆盖库函数，有点类似于 OOP 的方法重载？重写的 putchar 函数如下：

```
s8 putchar(s8 ch)
{
    SBUF = ch;
    TX_Busy = 1;
    while(TX_Busy);
    return ch;              //返回已经发送的数据
}
```

函数所做的工作就是往串行口 SBUF 写入一个字节数据启动发送，置发送软标志，等待发送完成串行中断服务程序清除软标志，向调用程序回传已发送的数据。注意函数定义应与原 getchar 相一致，发送完成后返回参数值，参数和返回值必须是 char 类型，否则会跟 stdio.h 中的函数原型声明相冲突，无法通过编译。

```
if(++msecond >= 500) {    //500ms测一个数据
    msecond = 0;
    if(!NTC) {
        printf("Vdd:%3.2f\t",DispVdd()/100.0);
        P1ASF = 0x01<<3;        //设置P1.3用于ADC通道3，NTC连接到ADC3
    }  else {
        printf("NTC:%4.1f\n",DispNTC()/10.0);
        P1ASF = 0;              //准备P1用于第9通道测试内部BGV
    }
    NTC^=1;
}
```

图 47. 主程调用 printf

在主程序中，就可以跟平时调用 printf 向屏幕打印数据一样，通过 UART 串口随心所欲向上位机发送数据了。

8. 上位机显示接受到的电压和温度值

RS232 串口通信虽然工作距离不长，是一种比较古老的通信协议，但是目前仍然是很常用的电脑与设备、设备与设备之间的近程控制方式，一些先进的路由器、机顶盒设备，尽管外部已经使用了 usb 通信接口，但是在机内仍然保留了源自 RS232 的 4 针 TTL 刷机接口，供厂家量产和维护人员救砖。

TTL UART 与 RS232 接口的区别是电平不同，RS232 是负逻辑±15V 电平，TTL UART 用的是 5V 正逻辑电平，两者之间的信号转换在上世纪 90 年代采用独立收发芯片 MC1488、MC1489，后来转为双功能合一的 232 芯片，如 MAX232，以及高速、高性能的 MAX3232。实验箱使用一片芯力特 SP3232，用于连接 9 针 RS232 串行通信接口。随着接口和硬件的换代，本世纪大多数电脑，尤其是笔记本，在物理上已经不再配置 DB9 接口，转而使用 USB 接口+USB 转串口芯片来进行串行通信。USB 转 232 芯片常用的有 FT232，CP210X，PL230X，CH34X，实验箱使用了一片 CH340G 作为 UART 通信的接口。从实验一可知，我们下载烧写程序和仿真

调试都是使用了此芯片虚拟出的电脑串口，在本实验中，串行通信仍然使用同一个接口。

根据实验箱原理图可知，实验箱的 CH340G 被连接到 MCU 的串口 1（UART1）。作为下位机，MCU 通过串口 1 收发串行数据，作为上位机，PC 电脑通过双公头 USB typeA 线和 CH340 枚举的虚拟串口收发数据，完成全双工串口通信，其设置如图 45。

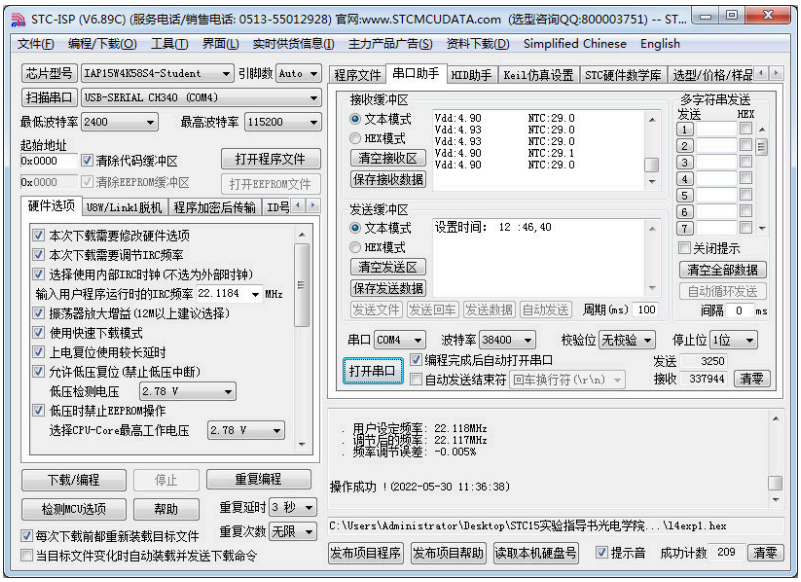


图 48. STC-ISP 串口助手

在 PC 上有很多著名的通用串口调试助手，如 ComTool 和 SSCOM 等，实验板开发者一般也有自己类似的串口/网络工具，如野火和原子 STM32。宏晶作为基于 51 的最大芯片制造企业和 STC 实验箱的提供者，其 STC-ISP 软件一直以界面友好、包罗齐整、集成工具多而著称，在选项页上的串口工具就是一个简单而且功能齐全的串口调试助手，可以文本和 HEX 码收发，可以单次和周期性发送数据，也可以发送预编制命令字串和文件。

串口打开前，一般需要设置好串口号为 CH340 虚拟串口，波特率、校验位数和停止位数与下位机通信程序相一致。如果只使

用一个串口 S1（UART1），在实验调试中，可以勾选上选项页上的“编程完成后自动打开串口”，则软件能在下载程序和进行串口通信之间完美无缝切换。当然，在调试过程中，为防止接受信息滚屏太快捕捉不到问题所在，也可以随时打开和关闭串口。另外，建议使用文本方式进行通信，便于我们直接观察传输的内容是否正确。

9. 上位机根据接受到的数据绘制曲线

实验要求记录和画出 NTC 温度值曲线。例程 lab4\exp1 中，下位机间隔 0.5 秒分别发送测得的电源电压 Vdd 和 NTC 温度值，上位机 STC 串口助手已经将接收到的串口数据直接原样显示出来。因为使用了通用的串口助手，数据在接受后是无法进行处理的，显示的数据格式也无法改变。

有能力的同学可以用 C 或者 python 自己编写上位机程序，对接受到的数据进行加工处理，显示、保存和绘制曲线，琢磨同屏画出电源电压和 NTC 温度两条曲线，对接受到的前导字符串进行处理并作为曲线的名称显示在对应曲线附近。如果不熟悉上位机程序开发，可以使用网上介绍或提供的现成上位机软件，如：https://zhuanlan.zhihu.com/p/109941792。但一般需要改变下位机程序中输出数据的格式，符合这些软件的接收要求。这里推荐一块比较经典的软件 serialchart，下载网站和使用介绍可以在网上搜索，学在浙大上也已经有上传。软件是开源的，有兴趣的同学甚至可以研究源码：https://github.com/peng-zhihui/SerialChart。

10.环形串行数据接收队列

修改 24h 时钟时间时，上位机为数据发送方，单片机为数据接收方。前节谈到了环形串行数据接受缓冲队列，环形是为了重复利用队列空间，这里先来谈谈其工作原理和具体实现。

队列使用了双指针，RX_Ptr 为串行接受指针，RX_Get 为软件取数指针。RX_Ptr 指针总是 RX_Get 指针的前面，虽然在环形队列中无所谓前后，但是我们在心里一定要牢记 RX_Get 不能跑到 RX_Ptr 之前，也就是当队列空时不能再试图从中取数，当队列满时不能再接受数据往里填充，否则会引起混乱。当然，也有一些方案是不管队列的空/满情况的，只要有新数据，一定覆盖旧数据，比如监控录像就是这种，我们这里不使用这种方案。

设计并没有增加另外的单元或标记以记录队列的空、满状态，而是直接根据双指针的相互关系来判断。当两个指针的值相等时表示缓冲队列为空，此时只能接受不能取走数据，当 RX_Ptr 为 RX_Get-1 时，表示缓冲队列已满，此时不能再接受数据。

```
s8 _getkey(void)
{
    u8 ch;
    while(RX_Ptr == RX_Get);   //缓冲空，等地数据接收
    ES = 0;
    ch = RX_Buffer[RX_Get];
    RX_Get = (RX_Get + 1) % UART_BUF_LENGTH;
    ES = 1;
    return ch;
}
```

字符输入_getkey()函数重定向为从接收队列中取字符。首先判断队列是否为空，如果不空，则取得字符、修改指针并返回。

11.上位机修改 24h 时钟时间

串行队列将数据流扁平化后，接下来的工作就是访问串行队列以获取数据。除了直接编程从队列中读取数据进行格式解析和数据分析以外，还可以借助于标准输入输出库 stdio.h 中的 scanf 函数，通过函数中的格式化字符串，如前节所述的 printf，从串行缓冲队列中读取、分割和分析数据。

实验例程具有比较好的输入容错能力。上位机的时间修改命令分为两个部分，第一部分为命令字“设置时间”，这部分允许前后有其他字符，只要连续 4 个汉字正确就认为命令正确；第二部分为拟修改的时钟值，正确的写法是“HH:MM:SS”，用’:’作为时分秒的分隔符，允许其他的西文分隔符，但不接受中文时间和中文分隔符。两部分之间需要由西文分隔字符分开，如空格等。

虽然程序有很好的容错性，但是如果命令格式不符合要求，也不能完成正确解析，原来的时间不修改，并上传一个错误提示。解析分两部分进行，如图 49。

```
while(RXready()) {
    scanf("%s",string);
    if(strstr(string,"设置时间")) valid=1; break;
}
if(!valid) return 0;

while(RXready()) {
    scanf("%s",string); sptr=string;
    for(endline=0;!endline;) {
        dtype=sscanf(sptr,"%d%n",&dvalue,&dlenth);
        switch(dtype) {
            case 0: sptr++; break;
            case 1: sptr+=dlenth;
                    switch(hhmmss) {
                        case HH:if(HH<24) tmpH=dvalue; else valid=0; break;
                        case MM:if(MM<60) tmpM=dvalue; else valid=0; break;
                        case SS:if(SS<60) tmpS=dvalue; else valid=0; break;
                        default: break;
                    }
                    hhmmss++;
                    break;
            default:endline=1;
        }
    }
}
if(valid)
    *hour=tmpH, *minute=tmpM, *second=tmpS;
```

图 49. 串行接收数据解析

数据解析函数的主体如图 49。其中 RXready()函数查看缓冲队列是否接收到了一整行数据，判断依据是缓冲队列不空且出现了行末符'\n'。

第一个 RXready()循环是连续提取字符串，判断其中有无带有“设置时间”命令的子串。如果直到行末都没有命令字，则结束解析返回错误提示。否则，设置有效标记 valid=1，完成第一部分解析，进入第二部分。

第二个 RXready()循环也是连续读取字符串，并从字符串中提取数值，按个分配给局部变量 tmpH，tmpM 和 tmpS，直到解析完字符串，并直至解释完整行数据。

这里要注意只能使用 sscanf 对字符串进行文本解析，不能使用 scanf 直接对缓冲队列进行解析，为了这个问题，我进行了多种尝试，浪费了两天时间，最后结果还是不尽人意，大家有兴趣可以在我注释的源码基础上继续尝试。另外，一定要全部解析完整行数据，否则会出现缓冲数据越来越多，无法重新接收直至队列卡死的情况。

在串行通信方式下，这种 scanf 封装方式没有正确的代码参考，调试也不是特别方便，有时候出现了一个小问题，导致发送出现乱码，或者没有什么反应，甚至有印象以前在电脑里能正常解析的方法，写到单片机上却出不来正常的结果，只能一步步查看，效率也很低下。实验四的例程和指导书拖得比较晚，最后指导书也是草草完工，请同学们海涵。

- **练习与思考：**

1. ADC 中，双积分型、逐次逼近型和闪烁型各有什么特点？
2. 是否可以将两个同品牌的 8b 逐次逼近型 ADC 组装成 16b?
3. 设计一个简单实验方案，对测得的 NTC 温度值进行校准。
4. 串行发送是等待方式进行的，尝试使用发送缓冲队列。
5. 串行接收是使用缓冲队列的，尝试不用缓冲的情况。
6. 例程使用了 38400 作为波特率，其最高值受什么影响？
7. 如何通过程序设计保证串行数据传输的正确性？
8. 思考如何利用 SCON 中的 SM2 位进行多机通信。