目录

图目录

摘要

随着经济的飞速发展和科学技术水平的不断提高,智能数据采集系统在工业生产以及科学研究中得到了广泛的应用.在信息化时代,数据和信息无疑成为一种重要的资源,而数据采集系统的出现更是进一步促进了人机交互、对设备的自动检测控制等的实现,为现代化工业生产提供了方便。本次课程设计将阐述基于单片机的智能数据采集系统的设计要点及其具体方法,以期对基于单片机的智能数据采集系统的改造和创新做出应有的贡献

此次课程设计利用型号为 STC89C52RC 的单片机，ADC0809，LCD1602 等电子元器件来实现对 8 通道模拟电压的采集的功能。

关键字：数据采集系统、单片机、STC89C52、模数转换、lcd 显示

Abstract

With the rapid development of economy and constantly improve the level of science and technology, intelligent data acquisition system in the industrial production and scientific research has been widely used. In the era of information, data and information has become a kind of important resources, and the emergence of the data acquisition system is further promoting the human-computer interaction, the realization of the automated testing of equipment control and so on, provides the convenience for modern industrial production. This course design will elaborate the design points and specific methods of the intelligent data acquisition system based on the single chip microcomputer, in order to make due contributions to the transformation and innovation of the intelligent data acquisition system based on the single chip microcomputer

This course design USES the single-chip microcomputer model STC89C52RC, ADC0809, LCD1602 and other electronic components to achieve the function of 8 channel analog voltage acquisition.

Keywords: Data acquisition system, single chip microcomputer, STC89C52, analog-to-digital conversion, LCD display

多路数据采集控制系统

1 设计题目

单片机多路数据采集控制系统—Proteus 软件仿真

2 设计目的

运用单片机原理及其应用等课程知识,根据题目要求进行进行软件仿真的设计和调试,从而加深对本课程知识的理解, 把学过的比较零碎的知识系统化，比较系统的学习开发单片机应用系统的基本步骤和基本方法，使学生应用知识能力、设计能力、调试能力以及报告撰写能力等有一定的提高。

3 设计内容及要求

用 8051 单片机设计数据采集控制系统，基本要求如下：

1、 可实现 8 路数据的采集，假设 8 路信号均为 0-5V 的电压信号；

2、 采集数据可通过 LCD 显示，显示格式为：[通道号] 电压值，如[0 1]： 4 .5V

3、 可通过键盘设置采集方式；（单点采集、多路巡测、采集时间间隔*）

4、 具有异常数据声音报警功能：对第一路数据可设置正常数据的上限值和下限值，当采集的数据出现异常，发出报警信号。

4 设计过程

4.1 总体设计

此次多路数据采集控制系统包括：

单片机为 STC89C52RC，8 位的 AD 转换芯片 0809，LCD1602 液晶显示屏，按键电路由按键、上拉电阻构成，晶振选择外接 12MHz，部分电容，电阻，二极管。其电路总设计框图如下：

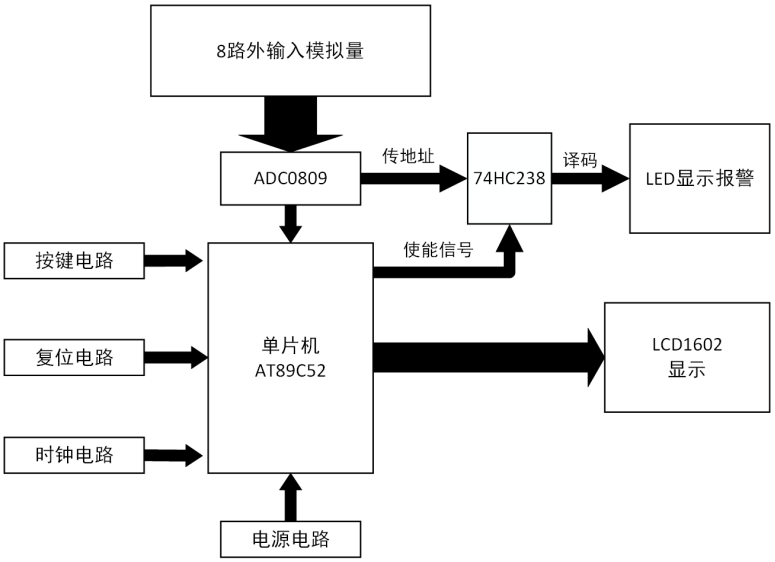


图 1-系统框图

4.2 关键元件选取

(一) STC89C52RC

STC89C52RC 是 STC 公司生产的一种低功耗、高性能 CMOS8 位微控制器，具有 8K 字节系统可编程 Flash 存储器。STC89C52 使用经典的 MCS-51 内核，但是做了很多的改进使得芯片具有传统的 51 单片机不具备的功能。在单芯片上，拥有灵巧的 8 位 CPU 和在系统可编程 Flash，使得 STC89C52 为众多嵌入式控制应用系统提供高灵活、超有效的解决方案。

32 位 I/O 口线，看门狗定时器，内置 4KB EEPROM，MAX810 复位电路，3 个 16 位定时器/计数器，4 个外部中断，一个 7 向量 4 级中断结构（兼容传统 51 的 5 向量 2 级中断结构），全双工串行口。另外 STC89C52 可降至 0Hz 静态逻辑操作，支持 2 种软件可选择节电模式。空闲模式下，CPU 停止工作，允许 RAM、定时器/计数器、串口、中断继续工作。

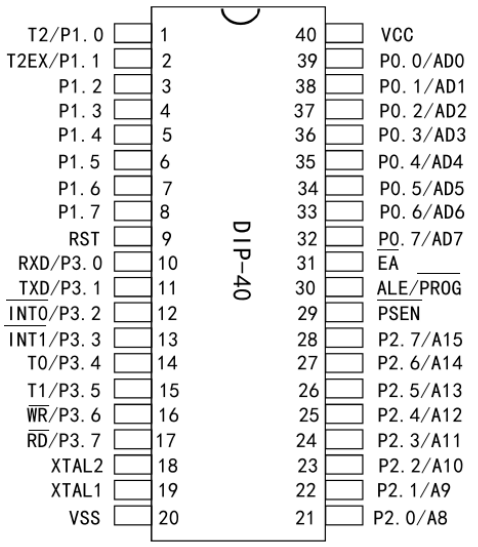


图 2- STC89C52 引脚图

图 3- STC89C52 实物图

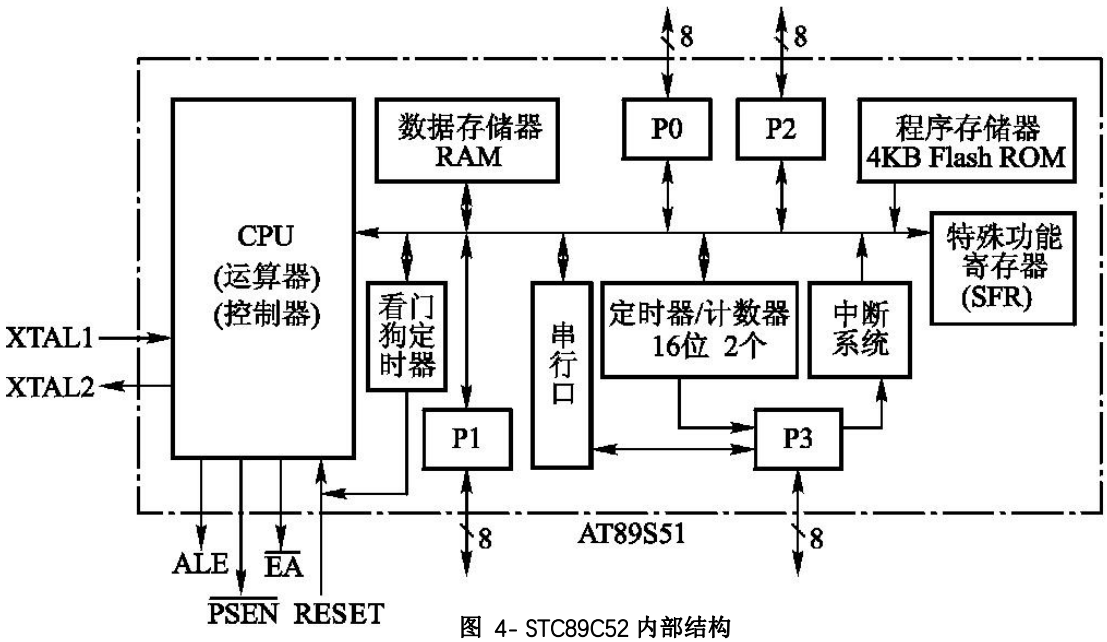


图 4- STC89C52 内部结构

1、数据存储器（RAM）：片内为 128B（52 子系列为 256B），片外最多可扩 64KB。片内 128B 的 RAM 以高速 RAM 的形式集成，可加快单片机运行的速度和降低功耗

2、程序存储器（Flash ROM）：片内集成有 8KB 的 Flash 存储器，如片内容量不够，片外可外扩至 64KB

3、中断系统：具有 6 个中断源，2 级中断优先权

4、定时器/计数器：2 个 16 位定时器/计数器（52 子系列有 3 个），4 种工作方式

5、看门狗定时器 WDT：当 CPU 由于干扰使程序陷入死循环或跑飞时，WDT 可使程序恢复正常运行

关键引脚

RESET：复位信号输入，在引脚加上持续时间大于 2 个机器周期的高电平，可使单片机复位。正常工作，此脚电平应 ≤ 0.5V。当看门狗定时器溢出输出时，该脚将输出长达 96 个时钟振荡周期的高电平

$\overline{EA}$：1，读取内部存储器。0，读取外部存储器。

ALE：为 CPU 访问外部程序存储器或外部数据存储器提供地址锁存信号，将低 8 位地址锁存在片外的地址锁存器中。

(二) ADC0809

ADC0809 是采用 COMS 工艺制造的双列直插式单片 8 位 A/D 转换器。分辨率 8 位，精度 7 位，带 8 个模拟量输入通道，有通道地址译码锁存器，输出带三态数据锁存器。启动信号为脉冲启动方式，最大可调节误差为±1LSB。

ADC0809 内部没有时钟电路，故 CLK 时钟需由外部输入，fclk 允许范围为 500kHz-1MHz，典型值为 640kHz。每通道的转换需要 66~73 个时钟脉冲，大约 100-110us。（转换时间）工作温度范围为-40℃—+85℃。功耗为 15mW，输入电压范围为 0－5V，单一+5V 电源供电。

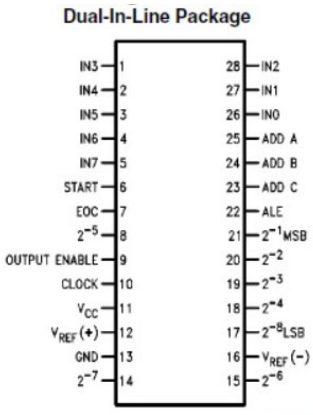


图 6-ADC0809 引脚

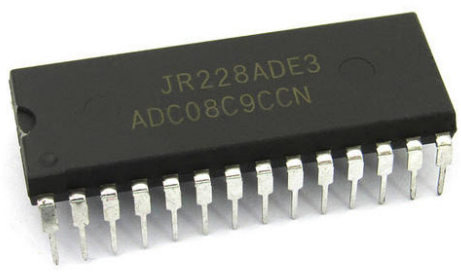


图 5-ADC0809 实物图

IN0~IN7：8 路模拟信号输入端。D0~D7：8 位数字量输出端。START：启动控制输入端，高电平有效，用于启动 ADC0809 内部的 A/D 转换过程。ALE：地址锁存控制输入端，ALE 端可与 START 端连接在一起，通过软件输入一个正脉冲，可立即启动 A/D 转换。EOC：转换结束信号输出端，开始 A/D 转换时为低电平，转换结束是输出高电平。OE：输出允许控

制端，用于打开三态输出锁存器，当 OE 为高电平时，打开三态数据输出锁存器，将转换后的数据凉输送到数据总线上。CLK：始终信号输入端。ADDA(ADDB、ADDC)：8 路模拟选通开关的 3 位地址选通输入端。

(三) LCD1602

LCD1602 液晶显示器是广泛使用的一种字符型液晶显示模块。它是由字符型液晶显示屏（LCD）、控制驱动主电路 HD44780 及其扩展驱动电路 HD44100，以及少量电阻、电容元件和结构件等装配在 PCB 板上而组成。

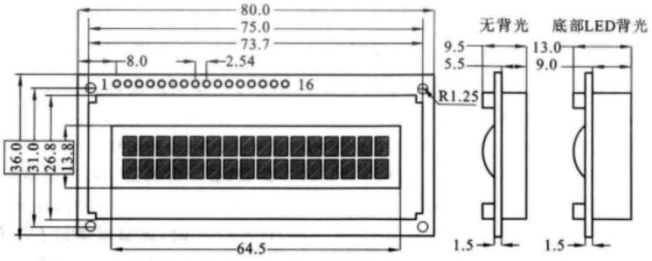


图 9-LCD 俯视图

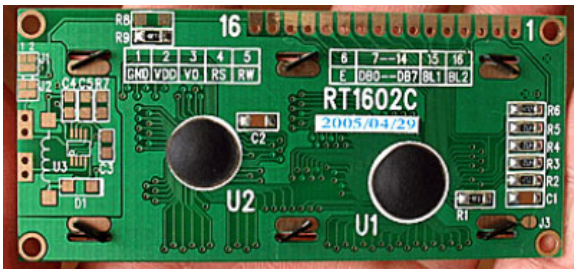


图 8-背部视图

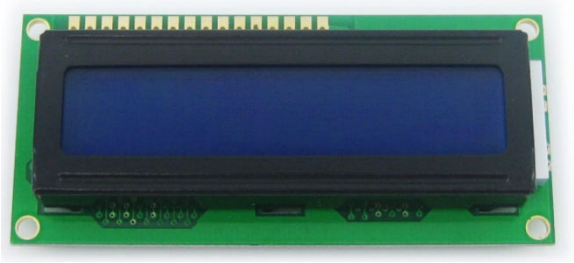


图 10-LCD 实物图

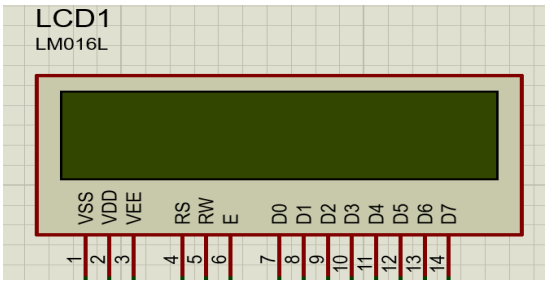


图 7-对应引脚

• 引脚 1：VSS 为地电源。

• 引脚 2：VDD 接 5V 正电源。

• 引脚 3：VL 为液晶显示器对比度调整端，接正电源时对比度最弱，接地时对比度最高，对比度过高时会产生“鬼影”现象，使用时可以通过一个 10kQ 的电位器调整其对比度。

• 引脚 4：RS 为寄存器选择脚，高电平时选择数据寄存器、低电平时选择指令寄存器。

• 引脚 5：R/W 为读/写信号线，高电平时进行读操作，低电平时进行写操作。当 RS 和 R/W 共同为低电平时可以写入指令或显示地址；当 RS 为低电平，R/W 为高电平时，可以读忙信号；当 RS 为高电平，R/W 为低电平时，可以写入数据。

• 引脚 6：E 端为使能端，当 E 端由高电平跳变为低电平时，液晶模块执行命令。

• 引脚 7~14：D0~D7 为 8 位双向数据线。

• 引脚 15：背光源正极。

• 引脚 16：背光源负极

(四) 74HC238

74HC238 能够将 3 地址输入，译码出 8 个输出，输出的电平刚好和 74LS138 相反。

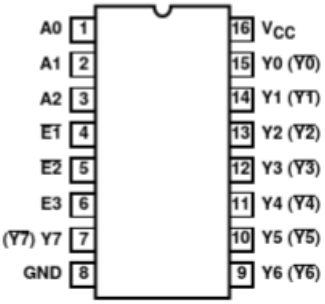


图 13-引脚图

图 12-实物图

INPUTS						OUTPUTS							
$\overline{E}_1$	$\overline{E}_2$	E_3	A_0	A_1	A_2	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
H	X	X	X	X	X	L	L	L	L	L	L	L	L
X	H	X	X	X	X	L	L	L	L	L	L	L	L
X	X	L	X	X	X	L	L	L	L	L	L	L	L
L	L	H	L	L	L	H	L	L	L	L	L	L	L
L	L	H	H	L	L	L	H	L	L	L	L	L	L
L	L	H	L	H	L	L	L	H	L	L	L	L	L
L	L	H	H	H	L	L	L	L	H	L	L	L	L
L	L	H	L	L	H	L	L	L	L	H	L	L	L
L	L	H	H	L	H	L	L	L	L	L	H	L	L
L	L	H	L	H	H	L	L	L	L	L	L	H	L
L	L	H	H	H	H	L	L	L	L	L	L	L	H

图 11-74HC238 真值表

用作 IO 扩展与复用

用 3 个 IO，可以控制 8 个输出

A0~A2：3 个输入

E1、E2：拉低使能，可以接地

E3：拉高使能，可以接 VCC

Y0~Y7：8 个输出

4.3 具体电路设计，基于 Proteus8

(一) 晶振模块

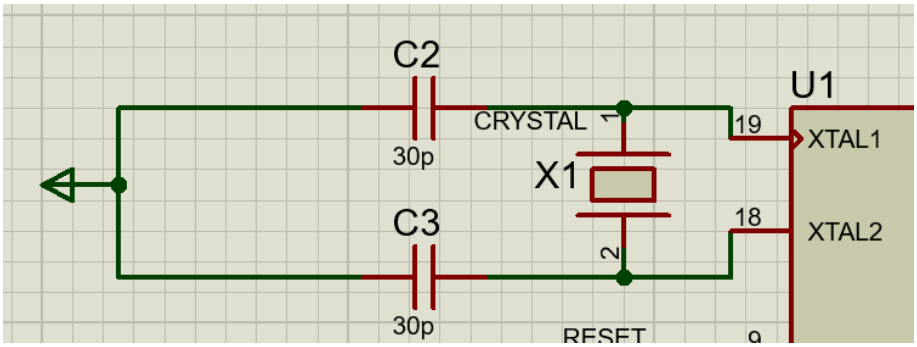


图 14-晶振模块

晶振与单片机的脚 XTAL0 和脚 XTAL1 构成的振荡电路中会产生谐波（也就是不希望存在的其他频率的波）这个波对电路的影响不大，但会降低电路的时钟振荡器的稳定性，为了电路的稳定性起见 ATMEL 公司只是建议在晶振的两引脚处接入两个 10pf-50pf 的瓷片电容接地来削减谐波对电路的稳定性的影响，所以晶振所配的电容在 10pf-50pf 之间都可以的。

(二) 单片机复位模块

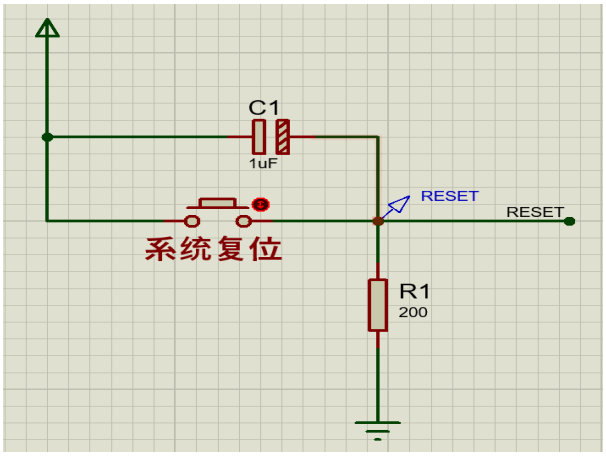


图 15-复位模块

复位电路工作原理如上所示，VCC 上电时，使极性电容 C 充电，在 200 欧电阻上出现高电位电压，使得单片机复位；几个毫秒后，C 充满，200 欧电阻上电流降为 0，电压也为 0，使得单片机进入工作状态。工作期间，按下 S1，C 放电，放电结束后，又充电，在 200 电阻上出现高电压，使得单片机进入复位状态，直到 S1 松手，C 又充电完毕，随后，单片机进入工作状态。

(三) ADC 模块

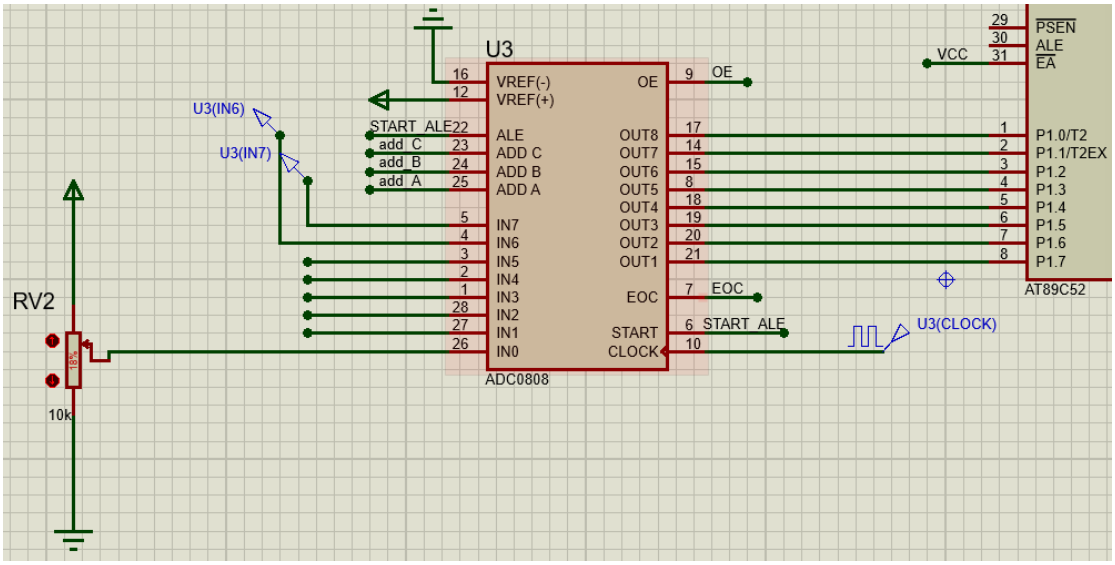


图 16-ADC 模块

如上图，ADC0809 的地址输入 CBA、ALE、EOC、START、OE 由 AT89C52 上的 IO 引脚来控制。8 位 OUT 输出直接由 AT89C52 的 P1 口来接收

(四) LCD 显示模块

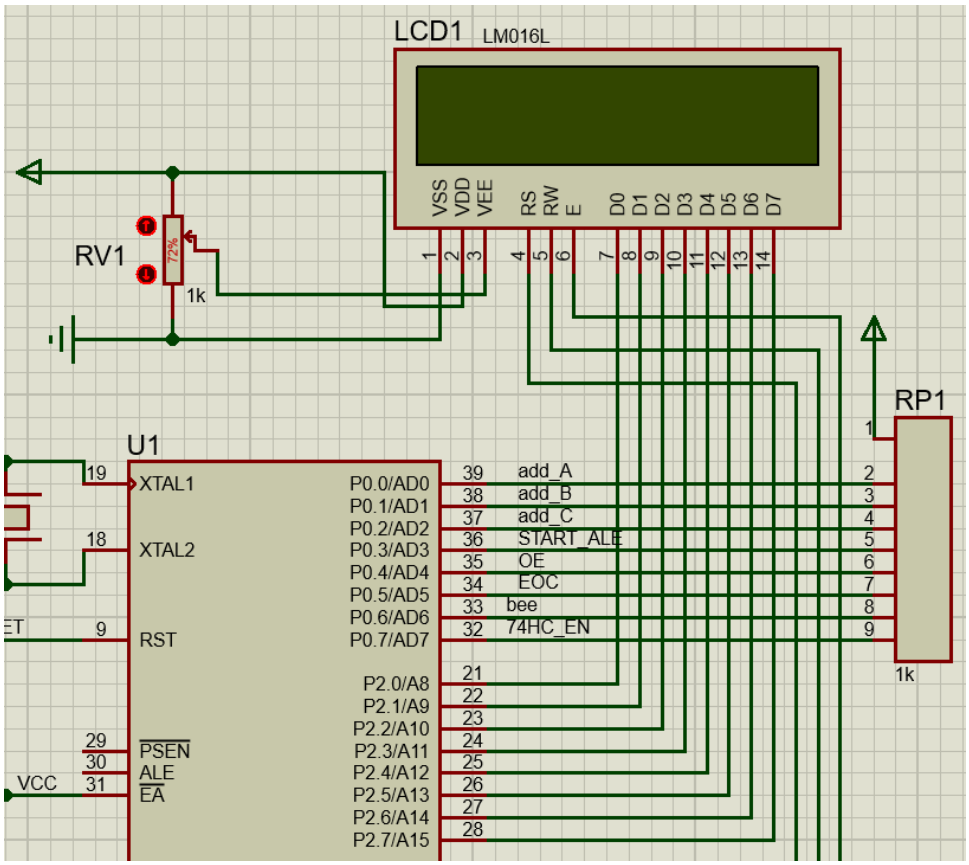


图 17-LCD 显示

如上图所示 Vss、Vdd、Vee 构成分压，STC89C52RC 的 P2 口驱动 D0~D1，另外的 3 个 IO 分别控制 RS、RW、E。

(五) 译码器与 LED 报警模块

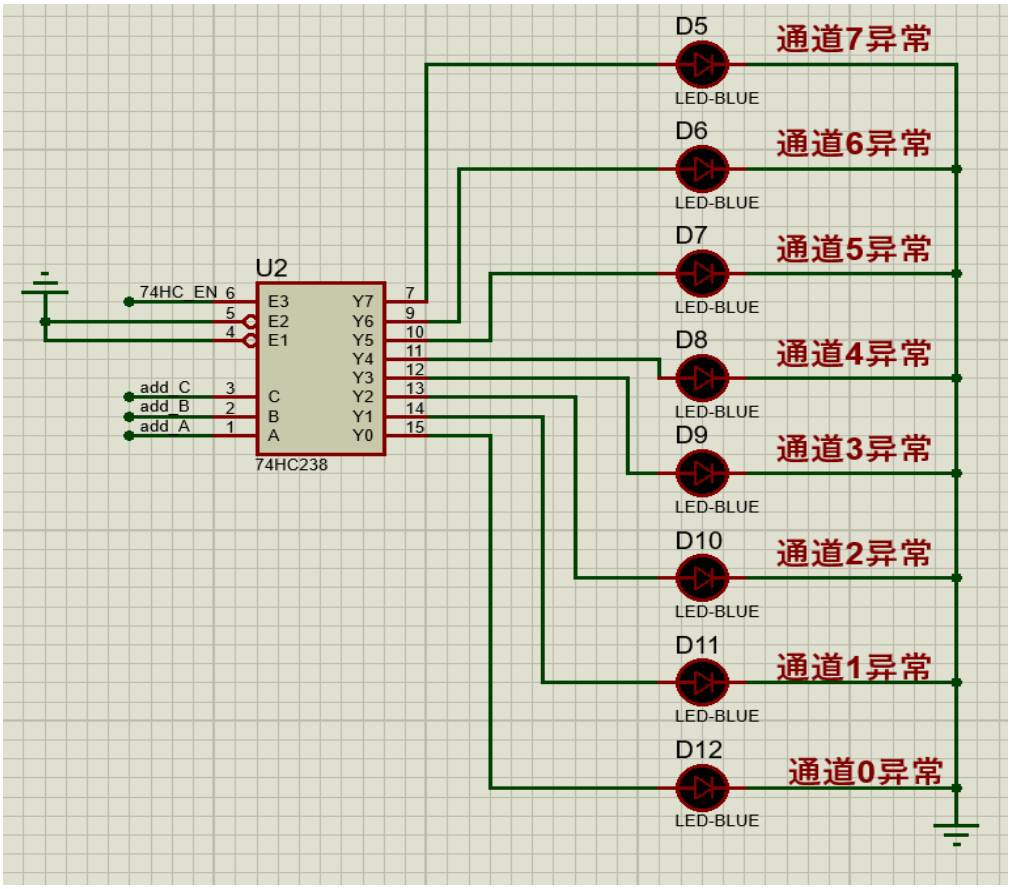


图 18-译码器与 LED 报警

如上图所示，74HC238 的 CBA 由 ADC0809 的 CBA 传输而来，使能端 E3 交由 AT89C52 的一个引脚控制，一旦切换到对应通道，如果该通道的电压异常，那么久点亮该通道的 LED，进行报警。

(六) 按键模块

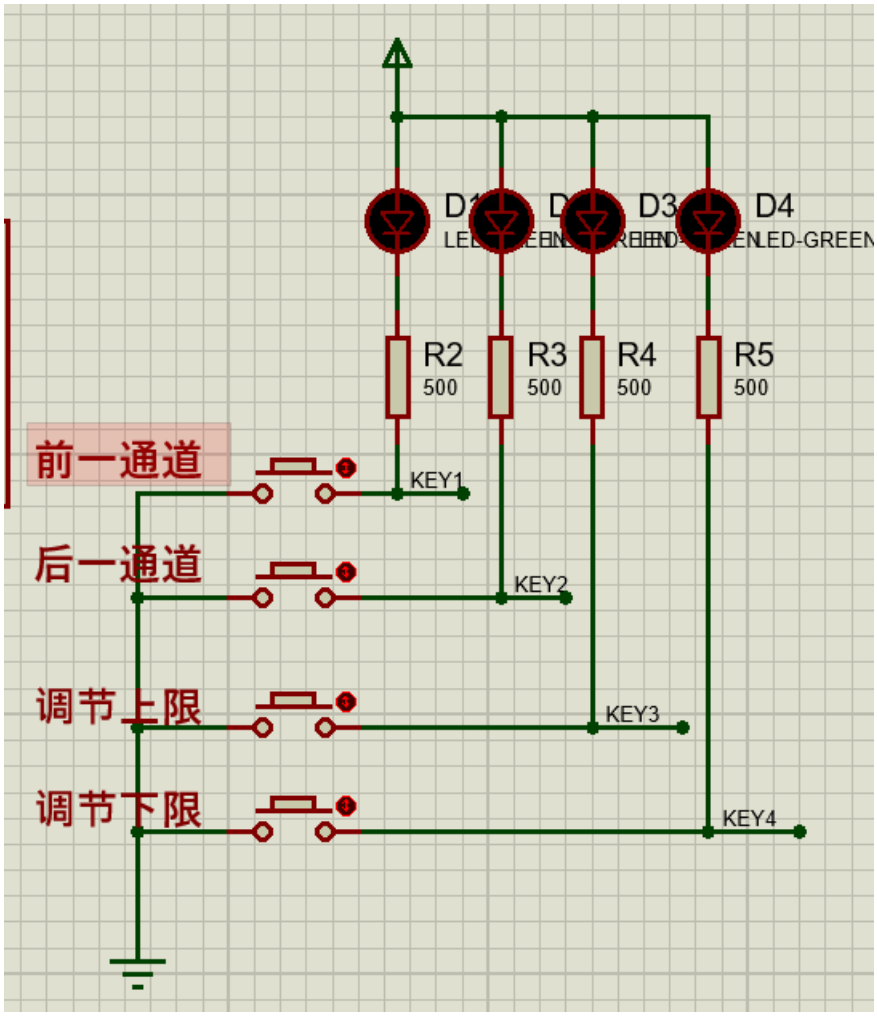


图 19-按键模块

按键模块如上图所示，上拉电阻在上电时默认将按键右端置为高电平。一旦按键按下，电阻下方为低电平并传给 STC89C52 对应引脚，执行操作。

4.4 程序设计

此次的程序设计借助 uVersion4 实现。自定义了 7 个头文件，1 个全局变量
它们分别为：

1、#include "AT89C52_init.h"　　//AT89C52 初始化相关实现

2、#include "lcd_init.h"　　// LCD 初始化相关实现

3、#include "lcd_display.h"　　// LCD 显示相关功能实现

4、#include "delay.h"　　//延时函数

5、#include "adjust_voltage.h"　　//电压调节相关功能实现

6、#include "key_check.h"　　//按键检测相关功能实现

7、#include "led_warning.h"　　//led 报警显示相关功能实现

8、unsigned char flag=0;　　//全局通道号

以下是本次程序设计的流程图，分为主函数执行流程图和中断函数执行流程图：

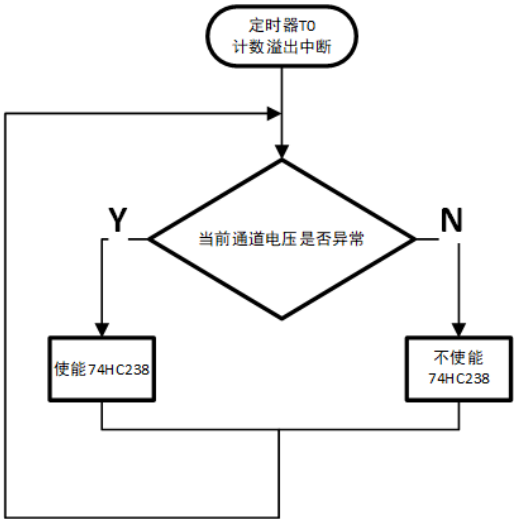


图 20-定时器 T0 中断

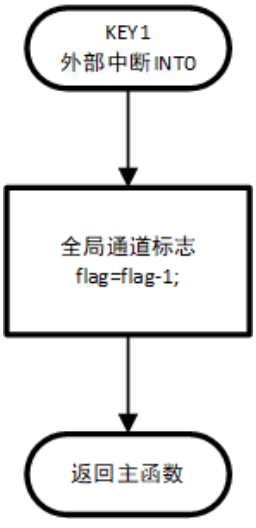


图 22-外部中断 T0

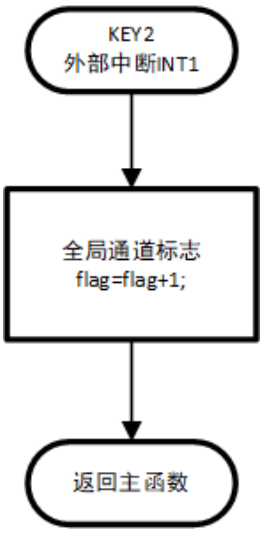


图 21-外部中断 T1

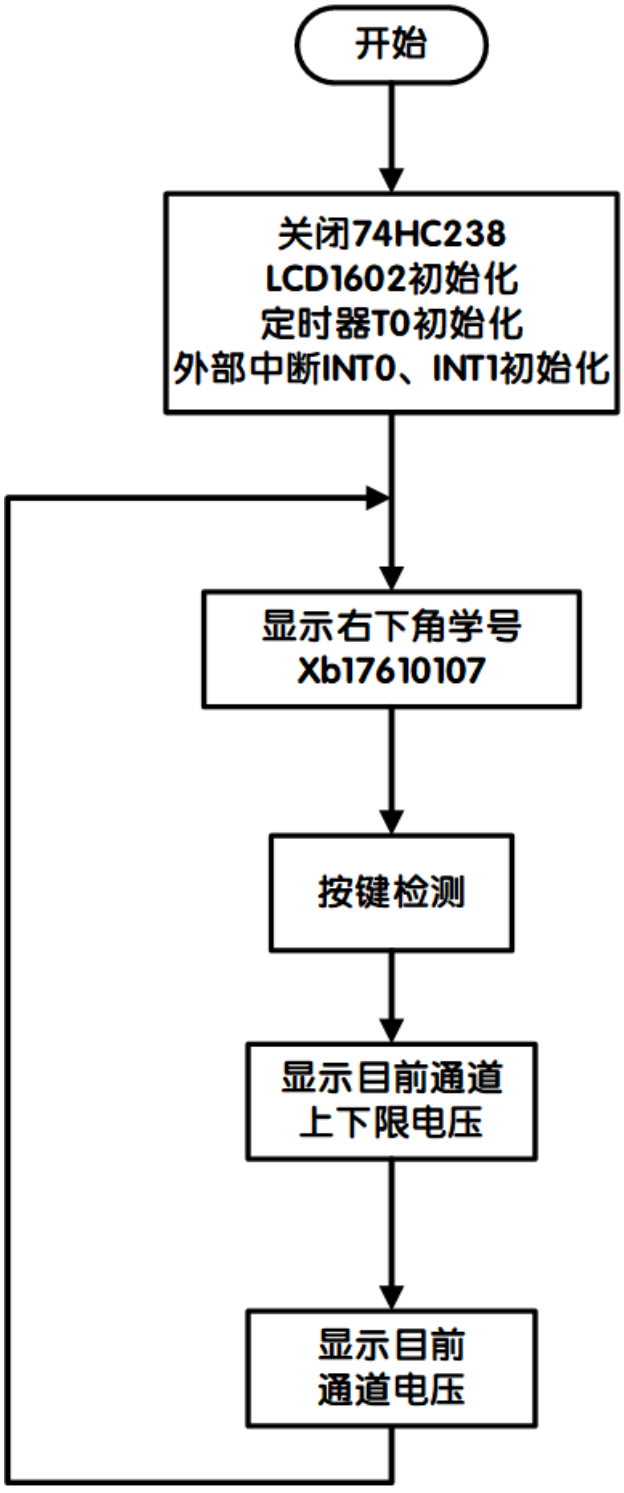

开始
关闭74HC238
LCD1602初始化
定时器T0初始化
外部中断INT0、INT1初始化
显示右下角学号
Xb17610107
按键检测
显示目前通道
上下限电压
显示目前
通道电压

图 23-主程序流程图

4.5 主要程序介绍

1) LCD 显示程序介绍

```
#include <reg52.h>
#include "lcd_init.h"
#include "delay.h"
sbit LCD_RS=P3^7;
sbit LCD_RW=P3^6;
sbit LCD_EN=P3^5;
sbit add_A=P0^0;
sbit add_B=P0^1;
sbit add_C=P0^2;//ABC 为 3 为地址输入线，译码出 8 位通道
sbit START_ALE=P0^3;//地址锁存，锁存 CBA 地址
sbit OE=P0^4;//转换后数据输出使能
sbit EOC=P0^5;//是否转换完成，0 正忙，1 转换成功
void lcd_init()
{
    LCD_Write_Command(0x38);//功能设置指令 6：8 位数据，双行显示，5*7 字形
    LCD_Write_Command(0x0c);//显示开关控制指令 4：开启显示屏，关光标，光标不闪烁
    LCD_Write_Command(0x06);//置输入模式指令 3：数据读写后光标右移，画面不移动
    LCD_Write_Command(0x01);//清屏指令 1
}

void LCD_Write_Command(unsigned char com)//写命令函数
{
    LCD_RS=0;//选择指令寄存器
    LCD_RW=0;//选择写
    P2=com;//传命令
    delay(5);
    LCD_EN=1;//将使能线置高
    delay(5);//将高电平延时 1 段时间
    LCD_EN=0;//产生下降沿，让 LCD 执行命令
}

void LCD_Write_Value(unsigned char value)//写数据函数
{
    LCD_RS=1;//选择数据寄存器
```

```
    LCD_RW=0;//选择写
    P2=value;//传输显示数据
    delay(5);
    LCD_EN=1;//将使能线置高
    delay(5);//将高电平延时 1 段时间
    LCD_EN=0;//产生下降沿，让 LCD 执行命令
}
```

上述代码为 LCD1602 相关的所有命令。LCD1602 可以显示 2*16 个字符，每个字符又是 5*7 的点阵组成。每个字符都有固定的显示段码，想要在 LCD1602 上指定位置显示指定字符分 2 个步骤，首先需要写命令到指令寄存器，然后再写数据到数据寄存器，才可以正确显示。以下是 LCD1602 写操作的时序图。

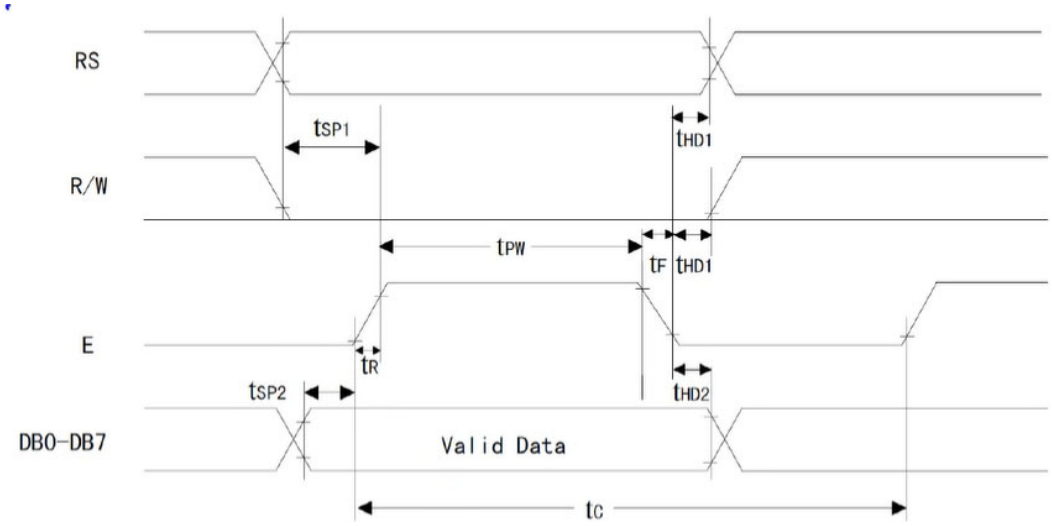


图 24-写操作时序图

RS 为寄存器选择脚，1 时选择数据寄存器、0 时选择指令寄存器

R/W 为读/写信号线，1 时进行读操作，0 时进行写操作。当 RS 和 R/W 共同为 0 时可以写入指令或显示地址；当 RS=1，R/W=0 时，可以写入数据。

如果是写命令或显示地址，刚开始 E=0，RS=0，RW=0，之后 P2 将数据传给 D0~D7，接下来 E=1 延时一段时间将数据读入，之后 E=0 命令或地址传输完成。

如果是写显示数据，刚开始 E=0，RS=1，RW=0，之后 P2 将数据传给 D0~D7，接下来 E=1 延时一段时间将数据读入，之后 E=0 数据传输完成。

D0~D7 一共是 8 位，CGROM 用来保存已有字符，写已有字符操作时，最高位 D7≡1，剩下的 7 位，若是写显示地址，其他 7 位为对应 LCD 点阵的地址外还要加上 0x80H 即 D7；若是写数据，7 位为对应 CGROM 中的字符的译码即可，查表可得。

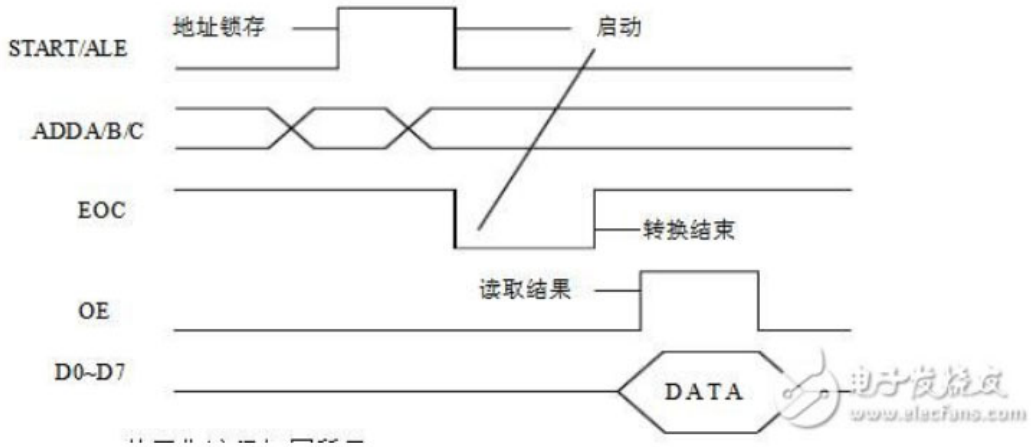
CGRAM 用来保存用户自定义的字符，但是只有 8 个空间。使用 CGRAM 时，首先得写 CGRAM 对应字符的地址，之后将自定义字符写入 CGRAM。之后再写显示地址命令，然后写入显示的是 CGRAM 的 8 个字符中的第几个字符。

2) ADC 转化程序介绍

```
START_ALE=0;
OE=0;
select_channel(1);//选择通道 1 进行转化
START_ALE=1;
delay(5);
START_ALE=0;
delay(5);//等待 AD 转换完成
while(EOC==1)
{
P1=0xff;//读数据之前将 P1 口的电压置高
OE=1;
temp1=P1/51.0;//将转换后的 16 进制准换成电压
temp2=temp1*100;
delay(5);
OE=0;
break;
}

void select_channel(unsigned char n)//通道选择函数
{
    switch(n)
    {
        case 0: add_C=0; add_B=0; add_A=0; break;//开启通道 0
        case 1: add_C=0; add_B=0; add_A=1; break;//开启通道 1
        case 2: add_C=0; add_B=1; add_A=0; break;//开启通道 2
        case 3: add_C=0; add_B=1; add_A=1; break;//开启通道 3
        case 4: add_C=1; add_B=0; add_A=0; break;//开启通道 4
        case 5: add_C=1; add_B=0; add_A=1; break;//开启通道 5
        case 6: add_C=1; add_B=1; add_A=0; break;//开启通道 6
        case 7: add_C=1; add_B=1; add_A=1; break;//开启通道 7
    }
}
```

模数转换部分核心程序如上所示，ADC0809 的 START 和 ALE 引脚由 AT89C52 一个引脚同时控制，unsigned float temp1 用来保存转换以后的电压值，temp2 用来保存扩大 100 倍后的 temp1。add_C、add_B、add_A 为地址输入，以下是 ADC0809 的时序图。



模数转换的编程思想按照 ADC0809 的时序图来进行。

刚开始，OE=1，关闭三态输出锁存器。输入 3 位地址 ABC，8 种不同的组合对应不同的通道。将 START=1，ALE=1，将地址存入地址锁存器中，经地址译码器译码从 8 路模拟通道中选通一路模拟量送到比较器，同时由于 START 上升沿使逐次逼近寄存器复位。

延时一段时间，START=0，下降沿启动 A/D 转换，转换期间 EOC 为低电平。

当转换结束时，转换的结果送入到输出三态锁存器，并使 EOC 信号自动回到高电平，通知 CPU 已转换结束。程序中判断 EOC 是否为 1，如果 EOC=1，将 OE=1，单片机从输出端 D0~D7 读取数据，并延时一段时间后 OE=0，准备下一次 AD 转换。

注意到在读取数据之前，将 P2=0xff 的原因是防止外部干扰。

5 仿真过程，Proteus 8 与 uVersion 联调

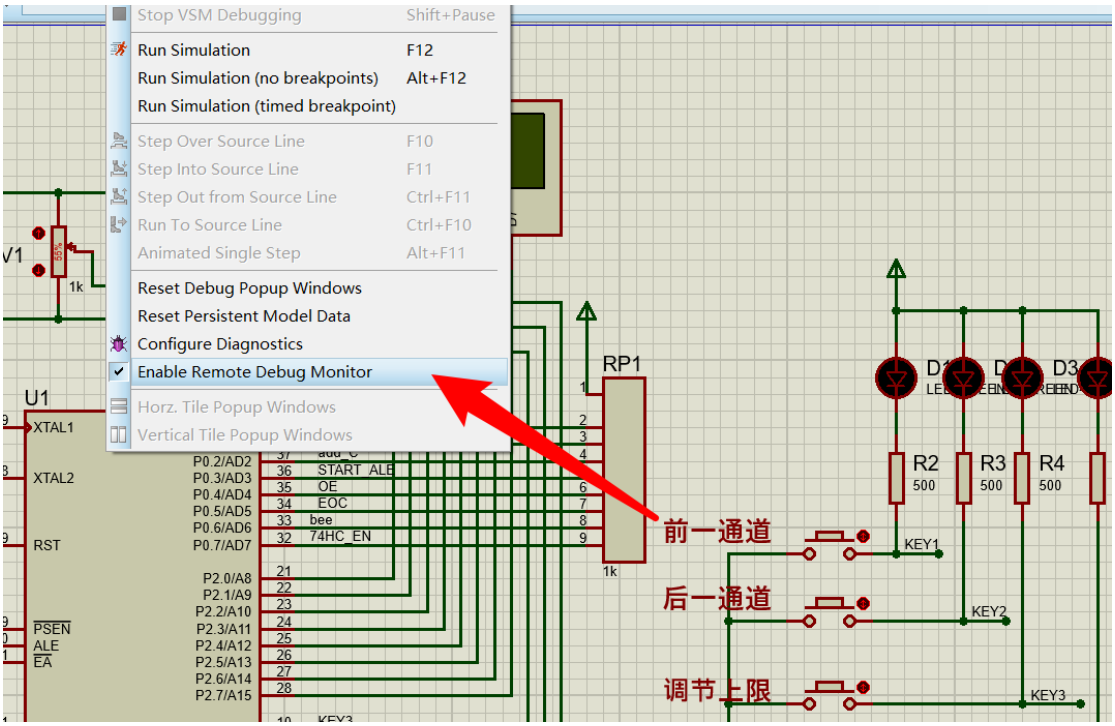


图 26-启用 Enable Remote

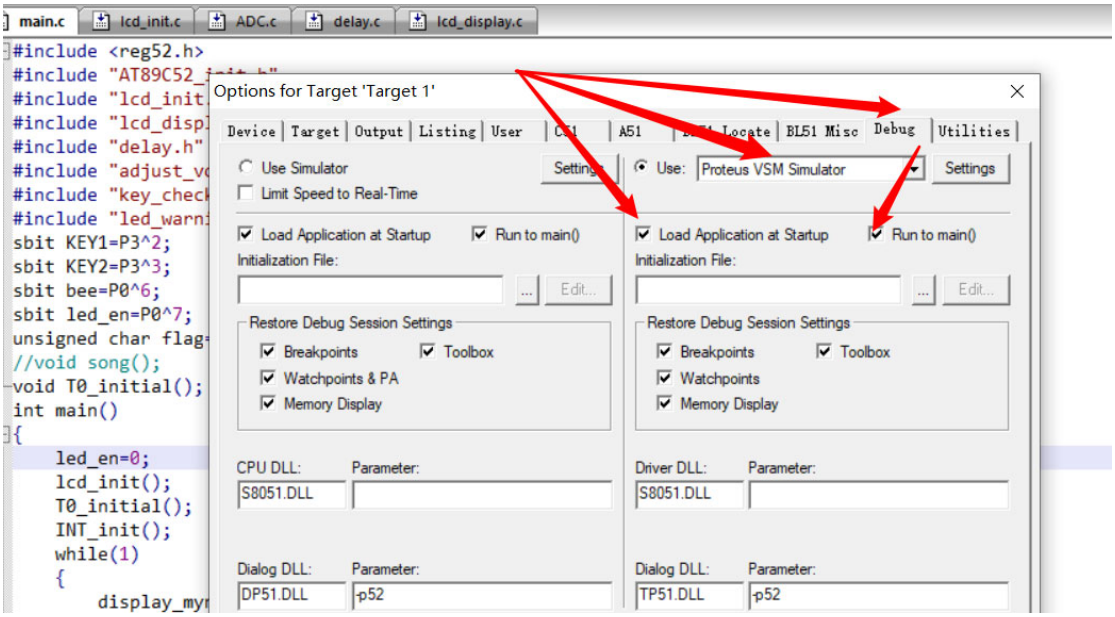


图 25-激活 uVersion4 中的 Proteus 选项

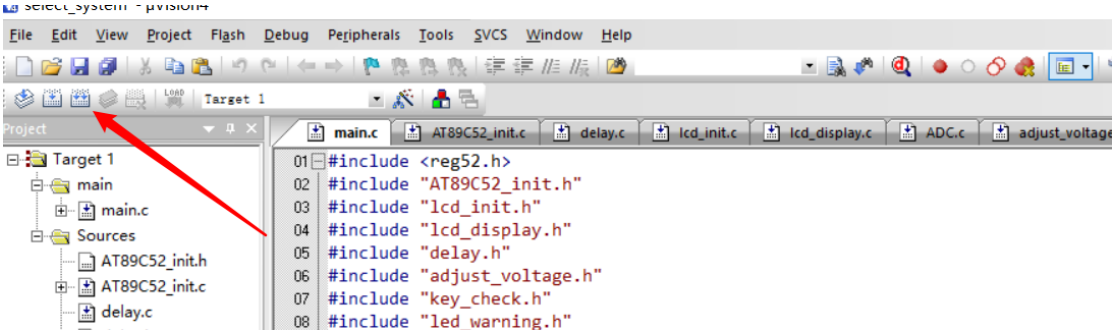


图 30-点击编译

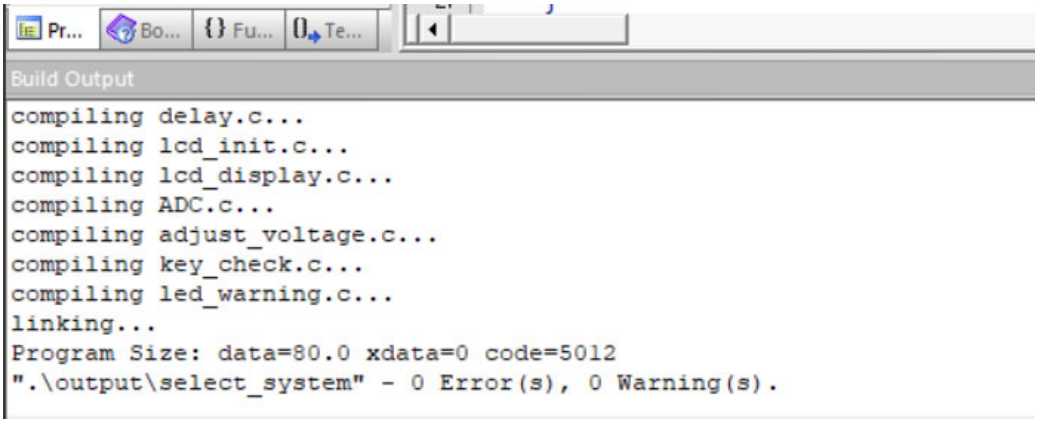


图 29-编译通过

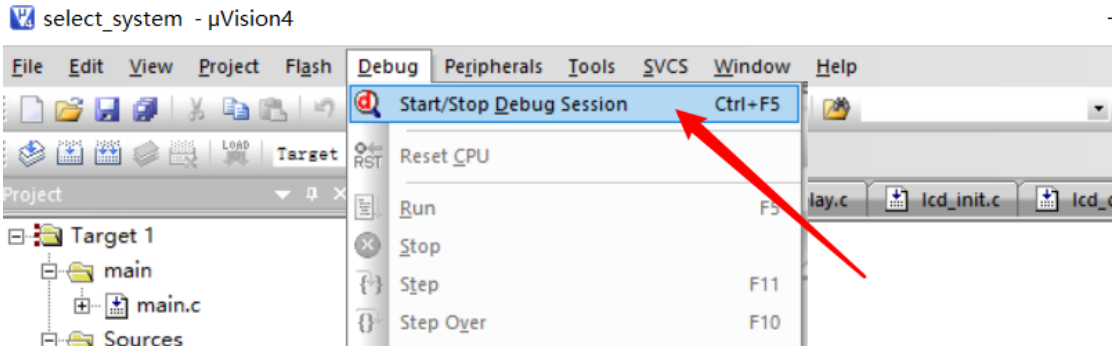


图 28-点击调试

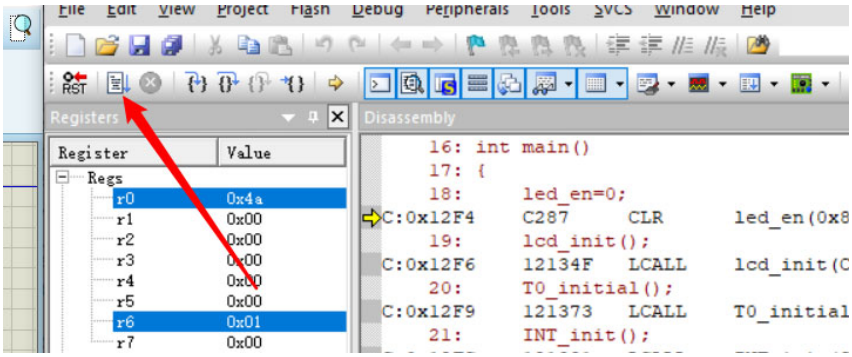


图 27-全速运行

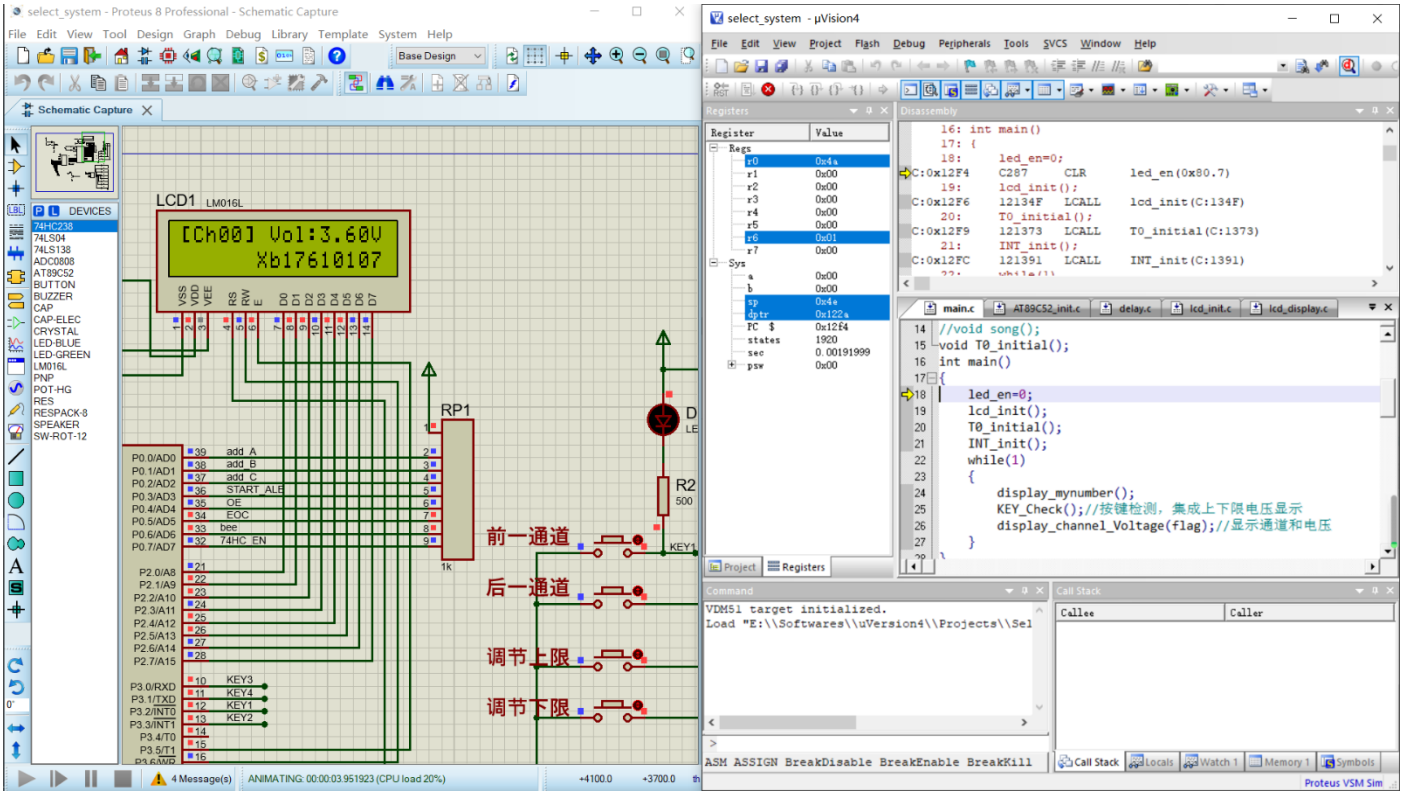
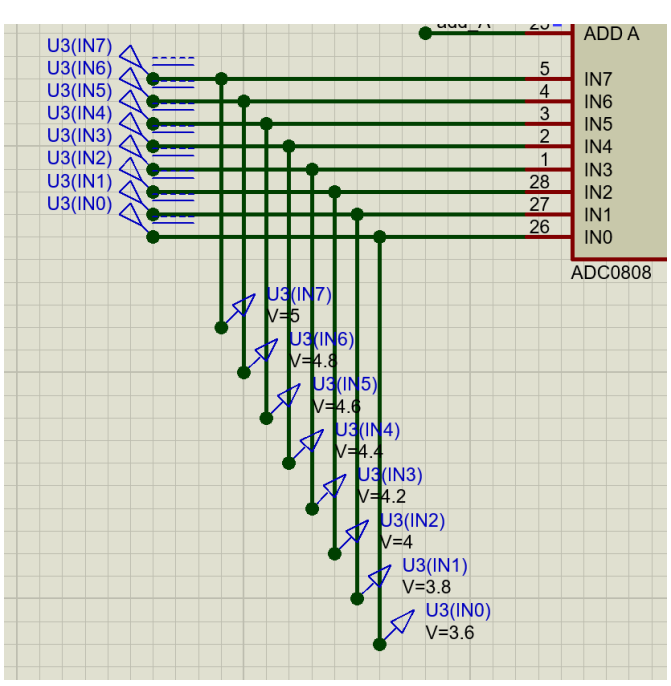


图 31-全速运行



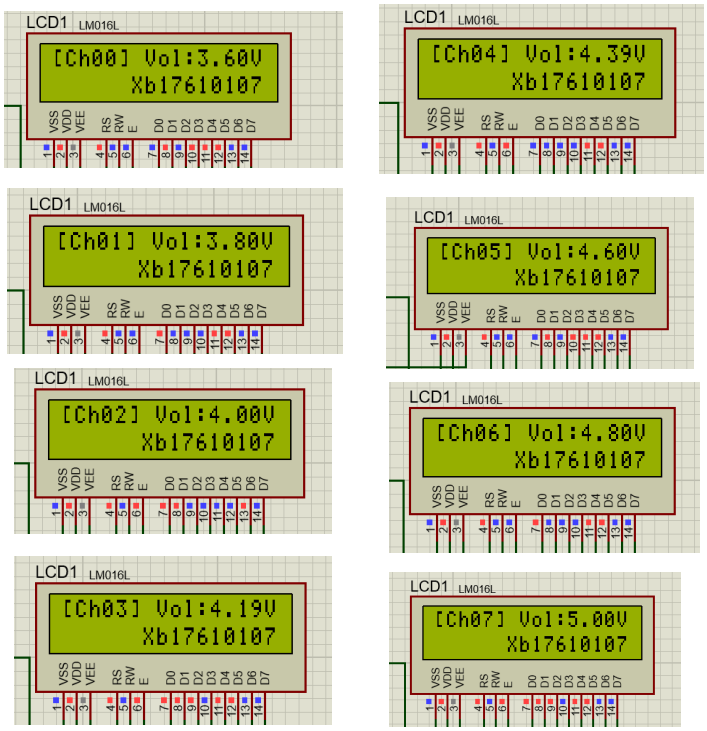


图 32-输入 8 个不同的电压值

如上图所示，IN0~IN7 的电压为 3.6~5，公差为 0.2 的等差数列，ADC0809 工作正常，LCD 能够显示对应通道的电压值。至于通道的切换由按键实现。

接下来是通道的上下限电压调节，我能够对分别对 8 个通道的上下限电压进行控制，实验要求利用蜂鸣器实现只对 0 通道的异常电压声音报警。在仿真的过程中，我发现蜂鸣器分为有源和无源 2 种，有源只要加上电压就会发出声音；而无源则需要 PWM 驱动。要实现发出音乐报警的功能对于毫无音乐天赋的我来讲太难了。于是我突发奇想，换了一种思维方式，利用 LED 进行显示报警，而且我不光做到了对单一通道的检测，还可以对任意当前通道的显示报警，实现该功能所需要的元器件是 74HC238，它的输出刚好和 74LS138 相反，刚好符合检测特点。碰巧的是，ADC0809 的 3 位地址可以与 74HC238 共用，接下来剩下三个使能端，2 个接地，一个交由 AT89C52 的一个 IO 控制即可。仿真电路如下：

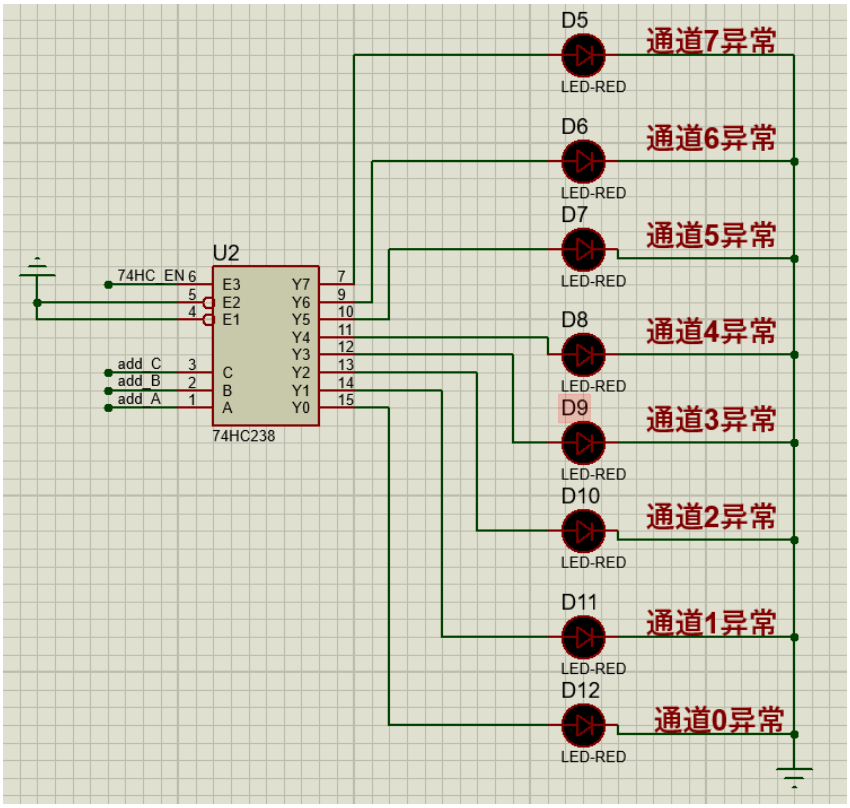
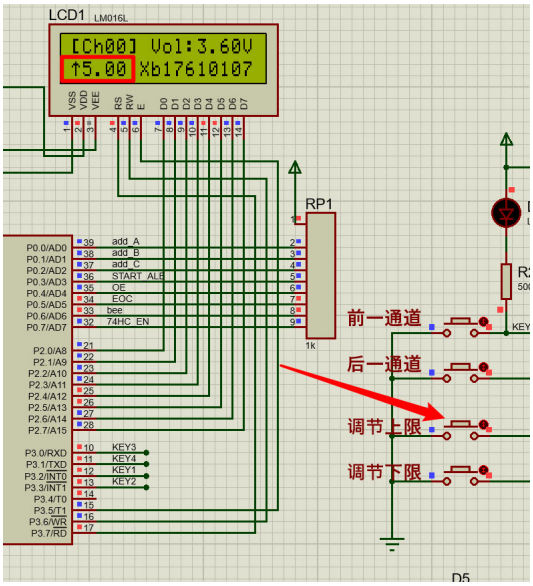


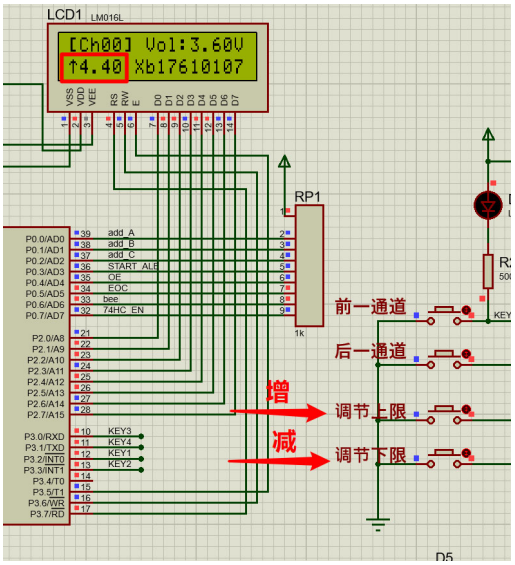
图 33-LED 报警电路

具体工作原理是，ABC 为地址输入，Y0~Y7 为译码输出，系统刚开始运行时，6 脚受单片机控制 E3=0，此时 Y0~Y7 输出为 00000000，LED 全灭。当处在某一通道下，并且该通道的电压异常，即超出上限或低于下限时，E3=1，译码输出，点亮该通道的报警 LED 实现与人的交互。

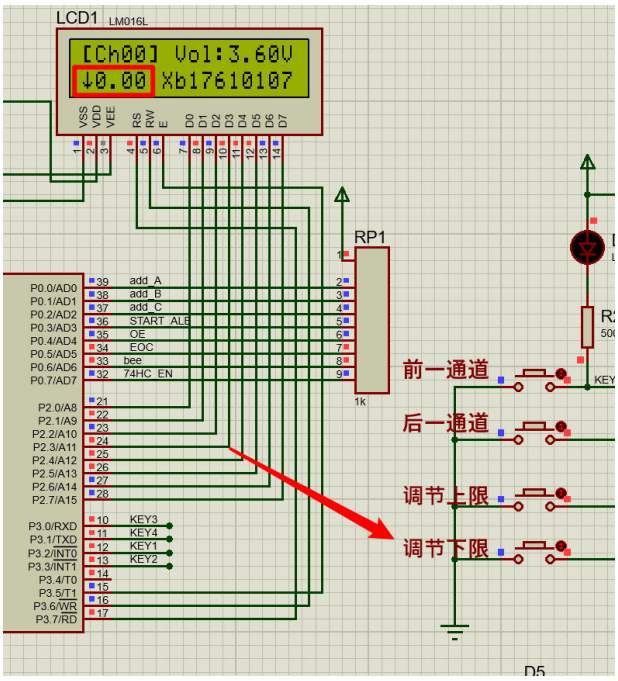
接下来时具体的调试现象，上下限电压由按键 KEY3 和 KEY4 控制，刚开始，第一次按下如果是 KEY3，则进入上限电压调节界面，第二次按下任意 KEY3 或 KEY4 可以控制上限电压的增减。同理第一次按下如果是 KEY4，则进入下限电压调节界面，第二次按下任意 KEY3 或 KEY4 可以控制下限电压的增减。

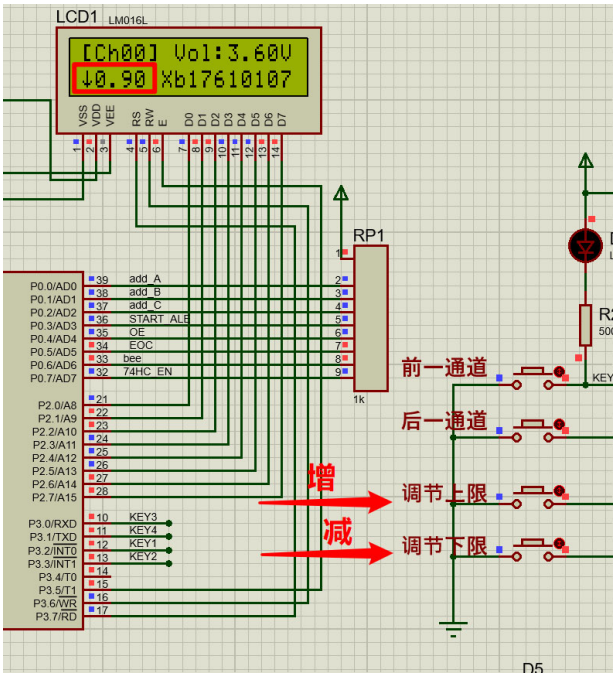
如果 3 秒没有操作，调节界面消失，但是该通道的上下限电压已更改。用户还需调节时，只需重复上述步骤即可。



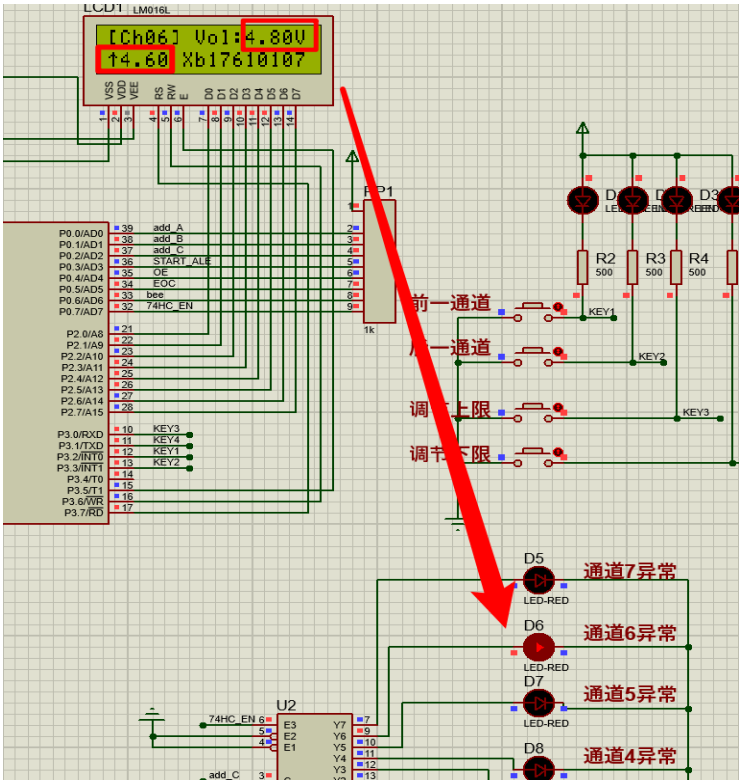


如上图所示，第一次按下如果是 KEY3 进入上限电压调节界面(最左边有上限电压箭头↑提示)，第二次按下任意 KEY3 或 KEY4 可以控制上限电压的增减。

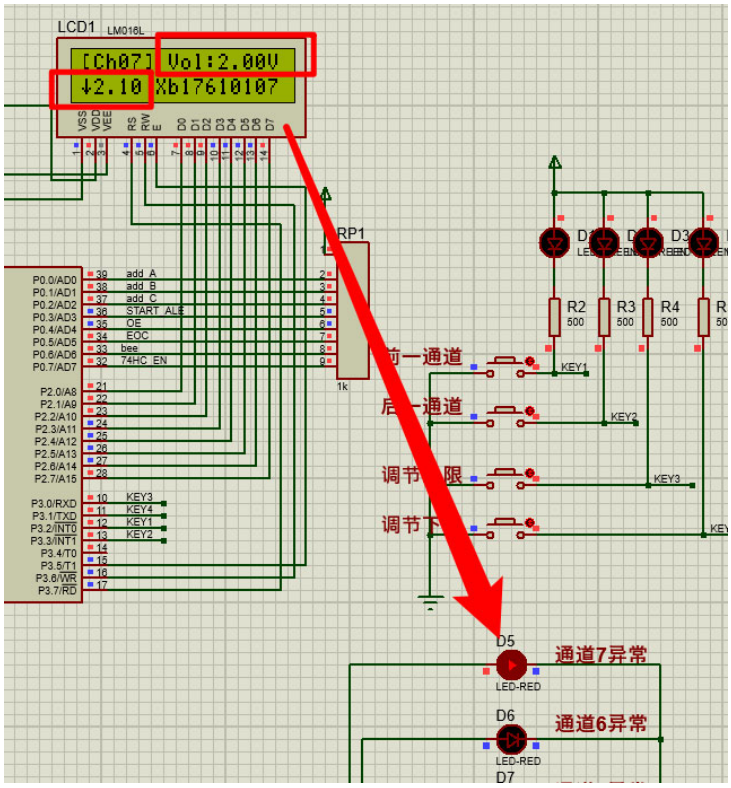




如上图所示，第一次按下如果是 KEY4 进入下限电压调节界面(最左边有下限电压箭头↓提示)，第二次按下任意 KEY3 或 KEY4 可以控制下限电压的增减。



如上图所示，此时通道 6 的电压已经超过了用户调节的上限电压值，通道 6 的报警 LED 亮起。



如上图所示，此时通道 7 的电压已经低于了用户调节的下限电压值，通道 7 的报警 LED 亮起。其他通道的工作情况同上，此处不再进行赘述。

6 遇到的问题与解决方法

(一) 按键模块按下输出 0 滞留

标准的按键模块由上拉电阻构成，正常输出为 1，按键按下时，0 持续一瞬间，立马恢复高电平。但是仿真时，按键按下后低电平 0 会保持 3 秒，之后才恢复高电平 1。将电压探头放置于按键端，发现会有惊人的充放电的效果。之后发现是由于仿真中加入了示波器检测波形，将示波器删去后恢复正常。将此问题归结为 Proteus 软件的问题。

(二) LCD 电压显示始终为整数

不管怎么改变输入电压的值，显示的电压始终是整数。发现在读取经 ADC0809 转化的数字量时，变量的类型没有考虑仔细，由于是整形 int，参与运算必然丢位。准换成 float，并且进行除法运算时注意除以浮点数 51.0，解决问题。

7 心得体会

我始终认为电子信息工程是一个非常有潜力的专业，并且我对该专业充满了浓厚的兴趣。虽然说专业课让我学得喘不过气来，难度颇大，但我相信这只是一个阶段性的过渡期。

属于工科的电子信息工程需要我有缜密的分析思维，强大的动手能力和学习能力，这些素养还需后期慢慢培养。时光荏苒，如今我已是一名大三的人了，在大学度过的两年里，我收获了许许多多宝贵的经验，积累了一定的专业知识与素养，回首看看两年前的自己，我发现自己已不再是刚进大学的是那个对专业一无所知的小白了。我清楚地知道目前我学到的只是专业的冰山一角，剩下的还需要我跟随老师的脚步，课外自己慢慢摸索。

此次的多路数据采集控制系统运用到了很多之前专业课的知识，尤其是数字电子技术，模拟电子技术。落实到具体的课程设计我才恍然大悟专业知识并不是白学的，原来是这么运用的，也难怪很久以前专业课老师每次上课都会给我们埋下伏笔。以前自己还没有这么深的感触，但越到后来的专业课程设计我渐渐感觉到专业课知识起到对课程设计的支配与指导作用。

一连 2 次 AT89C52 的课程设计让我对单片机有了更加深入的理解。本次课程设计最突出的特点，也是最关键的地方是时序分析，严格按照时序图来编写程序。有 ADC0809 的时序图，还有 LCD1602 的时序图，之前一直看不懂这个时序图干些什么作用，现在可恍然大悟了。

2 年下来，算算做过的课程实验加课程设计大概有 30 个左右，之前都是老师带着我们做着课程设计与实验。现在我逐渐摸清了要想做出一个能实现具体功能的成品需要哪几个步骤，常用哪几个模块，怎么把这几个模块联系起来。以我现在的能力，我能够自己设计出能够实现简易功能的成品，当然是建立在查阅资料的基础上。以我自己现在的知识储备量，我知道做出某种功能的成品，电路中运放（OP07 逃不掉）是一定要用到的，运放电路常有电压比较器、差分比例放大电路（需要两个输入信号）、同向比例运算电路（其中一个特例可实现电压跟随器）；有运放就会涉及到正负供电，供电的话就需要电源模块，电源模块有两类，恒流源和恒压源。

我相信之后我还会遇到更大的困难，更大的挑战，不过本身也不就是一路解决问题，克服困难，提升自我的过程么。未来是未知的，我只有不断进步，不断去适应困苦的环境，才

能超越自我，达到新的高度。

感谢一路上帮助过我的同学和老师。

附件

模块化程序代码

Proteus 8 Professional 仿真布局图一张

主函数 main.c

```
#include <reg52.h>
#include "AT89C52_init.h"
#include "lcd_init.h"
#include "lcd_display.h"
#include "delay.h"
#include "adjust_voltage.h"
#include "key_check.h"
#include "led_warning.h"
sbit KEY1=P3^2;
sbit KEY2=P3^3;
sbit bee=P0^6;
sbit led_en=P0^7;
unsigned char flag=0;//flag 用来保存通道序号
//void song();
void T0_initial();
int main()
{
    led_en=0;
    lcd_init();
    T0_initial();
    INT_init();
    while(1)
    {
        display_mynumber();
        KEY_Check();//按键检测，集成上下限电压显示
        display_channel_Voltage(flag);//显示通道和电压
    }
}


void key1() interrupt 0//按键 1 显示前 1 通道，递减 1
{
    if(flag==0)   flag=7;//到顶则从 7 开始
    else flag-=1;
}

void timer0() interrupt 1
{
    led_warn();
```

```
    TH0=0x3C;
    TL0=0x80;//每次进入中断赋初值 15536,50ms 检测一次
}

void key2() interrupt 2//按键 2 显示后 1 通道，递增 1
{
    if(flag==7)   flag=0;//到底则从 0 开始
    else flag+=1;
}


void T0_initial()
{
    TMOD=0x01;//无需配置 C/(T 的反)，因为单片机默认上电是低电平
    TH0=0x3C;
    TL0=0x80;//每次进入中断赋初值 15536,50ms 检测一次
    ET0=1;//允许 T0 中断
    TR0=1;//启动 T0 计数器
    EA=1;//打开总中断
}

/* void song()
{
    bee=1;
    delay(500);
    bee=0;
    delay(50000);
} */
```

AT89C52_init.c

```
#include <reg52.h>
void INT_init()//使能 AT89C52 外部中断 0 和外部中断 1
{
    EA=0;//关闭总中断屏蔽外部中断
    IT0=1;//下降沿中断
    EX0=1;//开启分中断开关
    IT1=1;//下降沿中断
    EX1=1;//开启分中断开关
    EA=1;//开启总中断
}
```

AT89C52_init.h

```
#ifndef _AT89C52_INIT_H_
#define _AT89C52_INIT_H_
extern void INT_init();
#endif
```

delay.c

```
void delay(unsigned char n)
{
    unsigned char i,j;
    for(i=n;i>0;i--)
        for(j=110;j>0;j--);
}
```

delay.h

```
#ifndef _DELAY_H_
#define _DELAY_H_
extern void delay(unsigned char n);
#endif
```

lcd_init.c

```
#include <reg52.h>
#include "lcd_init.h"
#include "delay.h"
sbit LCD_RS=P3^7;
sbit LCD_RW=P3^6;
sbit LCD_EN=P3^5;
sbit add_A=P0^0;
sbit add_B=P0^1;
sbit add_C=P0^2;//ABC 为 3 为地址输入线，译码出 8 位通道
sbit START_ALE=P0^3;//地址锁存，锁存 CBA 地址
sbit OE=P0^4;//转换后数据输出使能
sbit EOC=P0^5;//是否转换完成，0 正忙，1 转换成功
void lcd_init()
{
    LCD_Write_Command(0x38);//功能设置指令 6：8 位数据，双行显示，5*7 字形
    LCD_Write_Command(0x0c);//显示开关控制指令 4：开启显示屏，关光标，光标不闪烁
    LCD_Write_Command(0x06);//置输入模式指令 3：数据读写后光标右移，画面不移动
    LCD_Write_Command(0x01);//清屏指令 1
}

void LCD_Write_Command(unsigned char com)//写命令函数
{
    LCD_EN=0;
    LCD_RS=0;//选择指令寄存器
    LCD_RW=0;//选择写
    P2=com;//传命令
    delay(5);
    LCD_EN=1;//将使能线置高
    delay(5);//将高电平延时 1 段时间
    LCD_EN=0;//产生下降沿，让 LCD 执行命令
}

void LCD_Write_Value(unsigned char value)//写数据函数
{
    LCD_EN=0;
    LCD_RS=1;//选择数据寄存器
    LCD_RW=0;//选择写
    P2=value;//传输显示数据
    delay(5);
```

```
    LCD_EN=1;//将使能线置高
    delay(5);//将高电平延时 1 段时间
    LCD_EN=0;//产生下降沿，让 LCD 执行命令
}
```

lcd_init.h

```
#ifndef _LCD_INIT_H_
#define _LCD_INIT_H_
extern void lcd_init();
extern void LCD_Write_Command(unsigned char com);
extern void LCD_Write_Value(unsigned char value);
#endif
```

lcd_display.c

```
#include "lcd_init.h"
#include "lcd_display.h"
#include "ADC.h"
#include "delay.h"
#include "adjust_voltage.h"
unsigned char selfchar_up[]={0x04,0x0E,0x15,0x04,0x04,0x04,0x04,0x00};
unsigned char selfchar_down[]={0x04,0x04,0x04,0x04,0x15,0x0E,0x04,0x00};
unsigned char selfchar_clear[]={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
void display_mynumber()//显示学号
{
    LCD_Write_Command(0x46+0x80);//学号从第 2 行第 7 个开始显示
    LCD_Write_Value(0x58);//显示 X
    //delay(20);
    LCD_Write_Value(0x62);//显示 b
    //delay(20);
    LCD_Write_Value(0x31);//显示 1
    //delay(20);
    LCD_Write_Value(0x37);//显示 7
    //delay(20);
    LCD_Write_Value(0x36);//显示 6
    //delay(20);
    LCD_Write_Value(0x31);//显示 1
    //delay(20);
    LCD_Write_Value(0x30);//显示 0
```

```
    //delay(20);
    LCD_Write_Value(0x31);//显示 1
    //delay(20);
    LCD_Write_Value(0x30);//显示 0
    ///delay(20);
    LCD_Write_Value(0x37);//显示 7
    //delay(20);
}

void display_channel_Voltage(unsigned char m)//显示通道和电压
{
    LCD_Write_Command(0x00+0x80);//送通道号显示初地址
    LCD_Write_Value(0x5b);//显示[
    LCD_Write_Value(0x43);//显示 C
    LCD_Write_Value(0x68);//显示 h
    LCD_Write_Value(0x30);//显示 0
    LCD_Write_Value(flag+0x30);//显示 0
    LCD_Write_Value(0x5d);//显示]
    //LCD_Write_Value(0x3A);//显示:
    LCD_Write_Command(0x07+0x80);//送电压显示初地址
    LCD_Write_Value(0x56);//显示 V
    LCD_Write_Value(0x6F);//显示 o
    LCD_Write_Value(0x6c);//显示 l
    LCD_Write_Value(0x3A);//显示:
    switch(m)
    {
        case 0:
            ADC(0);
            LCD_Write_Value(temp2/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
            LCD_Write_Value((temp2/10)%10+0x30);//显示第 1 位小数
            LCD_Write_Value(temp2%10+0x30);//显示第 2 位小数
            break;
        case 1:
            ADC(1);
            LCD_Write_Value(temp2/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
            LCD_Write_Value((temp2/10)%10+0x30);//显示第 1 位小数
            LCD_Write_Value(temp2%10+0x30);//显示第 2 位小数
            break;
```

```
case 2:
    ADC(2);
    LCD_Write_Value(temp2/100+0x30);//显示最高位
    LCD_Write_Value(0x2E);//显示小数点
    LCD_Write_Value((temp2/10)%10+0x30);//显示第 1 位小数
    LCD_Write_Value(temp2%10+0x30);//显示第 2 位小数
    break;
case 3:
    ADC(3);
    LCD_Write_Value(temp2/100+0x30);//显示最高位
    LCD_Write_Value(0x2E);//显示小数点
    LCD_Write_Value((temp2/10)%10+0x30);//显示第 1 位小数
    LCD_Write_Value(temp2%10+0x30);//显示第 2 位小数
    break;
case 4:
    ADC(4);
    LCD_Write_Value(temp2/100+0x30);//显示最高位
    LCD_Write_Value(0x2E);//显示小数点
    LCD_Write_Value((temp2/10)%10+0x30);//显示第 1 位小数
    LCD_Write_Value(temp2%10+0x30);//显示第 2 位小数
    break;
case 5:
    ADC(5);
    LCD_Write_Value(temp2/100+0x30);//显示最高位
    LCD_Write_Value(0x2E);//显示小数点
    LCD_Write_Value((temp2/10)%10+0x30);//显示第 1 位小数
    LCD_Write_Value(temp2%10+0x30);//显示第 2 位小数
    break;
case 6:
    ADC(6);
    LCD_Write_Value(temp2/100+0x30);//显示最高位
    LCD_Write_Value(0x2E);//显示小数点
    LCD_Write_Value((temp2/10)%10+0x30);//显示第 1 位小数
    LCD_Write_Value(temp2%10+0x30);//显示第 2 位小数
    break;
case 7:
    ADC(7);
    LCD_Write_Value(temp2/100+0x30);//显示最高位
    LCD_Write_Value(0x2E);//显示小数点
    LCD_Write_Value((temp2/10)%10+0x30);//显示第 1 位小数
```

```
            LCD_Write_Value(temp2%10+0x30);//显示第 2 位小数
            break;
    }
    LCD_Write_Value(0x56);//显示 V
}

void display_down(unsigned char p)
{
    switch(p)
    {
        case 0:
            LCD_Write_Value(down0/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
            LCD_Write_Value((down0/10)%10+0x30);//显示第 1 位小数
            LCD_Write_Value(down0%10+0x30);//显示第 2 位小数
            break;
        case 1:
            LCD_Write_Value(down1/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
            LCD_Write_Value((down1/10)%10+0x30);//显示第 1 位小数
            LCD_Write_Value(down1%10+0x30);//显示第 2 位小数
            break;
        case 2:
            LCD_Write_Value(down2/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
            LCD_Write_Value((down2/10)%10+0x30);//显示第 1 位小数
            LCD_Write_Value(down2%10+0x30);//显示第 2 位小数
            break;
        case 3:
            LCD_Write_Value(down3/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
            LCD_Write_Value((down3/10)%10+0x30);//显示第 1 位小数
            LCD_Write_Value(down3%10+0x30);//显示第 2 位小数
            break;
        case 4:
            LCD_Write_Value(down4/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
            LCD_Write_Value((down4/10)%10+0x30);//显示第 1 位小数
            LCD_Write_Value(down4%10+0x30);//显示第 2 位小数
            break;
```

```
        case 5:
            LCD_Write_Value(down5/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
            LCD_Write_Value((down5/10)%10+0x30);//显示第 1 位小数
            LCD_Write_Value(down5%10+0x30);//显示第 2 位小数
            break;
        case 6:
            LCD_Write_Value(down6/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
            LCD_Write_Value((down6/10)%10+0x30);//显示第 1 位小数
            LCD_Write_Value(down6%10+0x30);//显示第 2 位小数
            break;
        case 7:
            LCD_Write_Value(down7/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
            LCD_Write_Value((down7/10)%10+0x30);//显示第 1 位小数
            LCD_Write_Value(down7%10+0x30);//显示第 2 位小数
            break;
    }
}

void display_up(unsigned char p)
{
    switch(p)
    {
        case 0:
            LCD_Write_Value(up0/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
            LCD_Write_Value((up0/10)%10+0x30);//显示第 1 位小数
            LCD_Write_Value(up0%10+0x30);//显示第 2 位小数
            break;
        case 1:
            LCD_Write_Value(up1/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
            LCD_Write_Value((up1/10)%10+0x30);//显示第 1 位小数
            LCD_Write_Value(up1%10+0x30);//显示第 2 位小数
            break;
        case 2:
            LCD_Write_Value(up2/100+0x30);//显示最高位
            LCD_Write_Value(0x2E);//显示小数点
```

```
                LCD_Write_Value((up2/10)%10+0x30);//显示第 1 位小数
                LCD_Write_Value(up2%10+0x30);//显示第 2 位小数
                break;
            case 3:
                LCD_Write_Value(up3/100+0x30);//显示最高位
                LCD_Write_Value(0x2E);//显示小数点
                LCD_Write_Value((up3/10)%10+0x30);//显示第 1 位小数
                LCD_Write_Value(up3%10+0x30);//显示第 2 位小数
                break;
            case 4:
                LCD_Write_Value(up4/100+0x30);//显示最高位
                LCD_Write_Value(0x2E);//显示小数点
                LCD_Write_Value((up4/10)%10+0x30);//显示第 1 位小数
                LCD_Write_Value(up4%10+0x30);//显示第 2 位小数
                break;
            case 5:
                LCD_Write_Value(up5/100+0x30);//显示最高位
                LCD_Write_Value(0x2E);//显示小数点
                LCD_Write_Value((up5/10)%10+0x30);//显示第 1 位小数
                LCD_Write_Value(up5%10+0x30);//显示第 2 位小数
                break;
            case 6:
                LCD_Write_Value(up6/100+0x30);//显示最高位
                LCD_Write_Value(0x2E);//显示小数点
                LCD_Write_Value((up6/10)%10+0x30);//显示第 1 位小数
                LCD_Write_Value(up6%10+0x30);//显示第 2 位小数
                break;
            case 7:
                LCD_Write_Value(up7/100+0x30);//显示最高位
                LCD_Write_Value(0x2E);//显示小数点
                LCD_Write_Value((up7/10)%10+0x30);//显示第 1 位小数
                LCD_Write_Value(up7%10+0x30);//显示第 2 位小数
                break;
        }
}

void display_adjust(char s)//显示调节后上下限电压的内容，s 用来选择显示上限电压还是下限电压
{
    unsigned char m;
    LCD_Write_Command(0x40);//设定 CGRAM 地址，把自定义字符存储进去
```

```
switch(s)
{
    case -1:
    for(m=0;m<8;m++) //将 selfchar_down[]中的数据依次写入 CGRAM 的 8 位里
    {
        LCD_Write_Value(selfchar_down[m]);
        delay(5);
    }
    LCD_Write_Command(0x40+0x80);
    LCD_Write_Value(0x00);
    LCD_Write_Command(0x41+0x80);//送下限电压显示初地址
    display_down(flag);//显示对应通道的下限电压
    break;

    case 1:
    for(m=0;m<8;m++) //将 selfchar_up[]中的数据依次写入 CGRAM 的 8 位里
    {
        LCD_Write_Value(selfchar_up[m]);
        delay(5);
    }
    LCD_Write_Command(0x40+0x80);
    LCD_Write_Value(0x00);
    LCD_Write_Command(0x41+0x80);//送上限电压显示初地址
    display_up(flag);//显示对应通道的上限电压
    break;

    case 0:
    for(m=0;m<8;m++) //将 selfchar_clear[]中的数据依次写入 CGRAM 的 8 位里
    {
        LCD_Write_Value(selfchar_clear[m]);
        delay(5);
    }
    LCD_Write_Command(0x40+0x80);
    LCD_Write_Value(0x00);
    LCD_Write_Command(0x41+0x80);//送上限电压显示初地址
    LCD_Write_Value(0x20);//显示空
    LCD_Write_Value(0x20);//显示空
    LCD_Write_Value(0x20);//显示空
    LCD_Write_Value(0x20);//显示空
    break;
```

```
    }
}
```

lcd_display.h

```
#ifndef _LCD_DISPLAY_H_
#define _LCD_DISPLAY_H_
extern void display_mynumber();//显示学号
extern void display_channel_Voltage(unsigned char m);//显示通道和电压
extern void display_down(unsigned char p);//显示下限电压值
extern void display_up(unsigned char p);//显示上限电压值
extern void display_adjust(char s);//显示调节后上下限电压的内容，s 用来选择显示上限电压还是下限电压
extern unsigned char flag;//flag 用来保存通道序号
#endif
```

ADC.c

```
#include <reg52.h>
#include "ADC.h"
#include "delay.h"
sbit add_A=P0^0;
sbit add_B=P0^1;
sbit add_C=P0^2;//ABC 为 3 为地址输入线，译码出 8 位通道
sbit START_ALE=P0^3;//地址锁存，锁存 CBA 地址
sbit OE=P0^4;//转换后数据输出使能
sbit EOC=P0^5;//是否转换完成，0 正忙，1 转换成功
float temp1=0;//temp1 用来读取转换后的 16 进制
unsigned int temp2=0;//用来保存扩大啊 100 后的 temp1
void ADC(unsigned char n)
{
    char i;
    switch(n)
    {
        case 0:
        temp1=0;//由于 0 通道采取多次采样的方法，所以需要复位
        for(i=5;i>0;i--)
        {
            START_ALE=0;//
            OE=0;
            select_channel(0);//选择通道 0 进行转化
```

```
        START_ALE=1;
        delay(5);
        START_ALE=0;
        delay(5);//等待 AD 转换完成
        while(EOC==1)
        {
            P1=0xff;//读数据之前将 P1 口的电压置高
            OE=1;
            temp1=temp1+P1/51.0;//将转换后的 16 进制准换成电压
            delay(5);
            OE=0;
            break;
        }
    }
    temp2=(temp1/5.0)*100;
    break;
    case 1:
    START_ALE=0;
    OE=0;
    select_channel(1);//选择通道 1 进行转化
    START_ALE=1;
    delay(5);
    START_ALE=0;
    delay(5);//等待 AD 转换完成
    while(EOC==1)
    {
        P1=0xff;//读数据之前将 P1 口的电压置高
        OE=1;
        temp1=P1/51.0;//将转换后的 16 进制准换成电压
        temp2=temp1*100;
        delay(5);
        OE=0;
        break;
    }
    break;
    case 2:
    START_ALE=0;
    OE=0;
    select_channel(2);//选择通道 2 进行转化
    START_ALE=1;
```

```
delay(5);
START_ALE=0;
delay(5);//等待 AD 转换完成
while(EOC==1)
{
    P1=0xff;//读数据之前将 P1 口的电压置高
    OE=1;
    temp1=P1/51.0;//将转换后的 16 进制准换成电压
    temp2=temp1*100;
    delay(5);
    OE=0;
    break;
}
break;
case 3:
START_ALE=0;
OE=0;
select_channel(3);//选择通道 3 进行转化
START_ALE=1;
delay(5);
START_ALE=0;
delay(5);//等待 AD 转换完成
while(EOC==1)
{
    P1=0xff;//读数据之前将 P1 口的电压置高
    OE=1;
    temp1=P1/51.0;//将转换后的 16 进制准换成电压
    temp2=temp1*100;
    delay(5);
    OE=0;
    break;
}
break;
case 4:
START_ALE=0;
OE=0;
select_channel(4);//选择通道 4 进行转化
START_ALE=1;
delay(5);
START_ALE=0;
```

```
delay(5);//等待 AD 转换完成
while(EOC==1)
{
    P1=0xff;//读数据之前将 P1 口的电压置高
    OE=1;
    temp1=P1/51.0;//将转换后的 16 进制准换成电压
    temp2=temp1*100;
    delay(5);
    OE=0;
    break;
}
break;
case 5:
START_ALE=0;
OE=0;
select_channel(5);//选择通道 5 进行转化
START_ALE=1;
delay(5);
START_ALE=0;
delay(5);//等待 AD 转换完成
while(EOC==1)
{
    P1=0xff;//读数据之前将 P1 口的电压置高
    OE=1;
    temp1=P1/51.0;//将转换后的 16 进制准换成电压
    temp2=temp1*100;
    delay(5);
    OE=0;
    break;
}
break;
case 6:
START_ALE=0;
OE=0;
select_channel(6);//选择通道 6 进行转化
START_ALE=1;
delay(5);
START_ALE=0;
delay(5);//等待 AD 转换完成
while(EOC==1)
```

```
            {
                P1=0xff;//读数据之前将 P1 口的电压置高
                OE=1;
                temp1=P1/51.0;//将转换后的 16 进制准换成电压
                temp2=temp1*100;
                delay(5);
                OE=0;
                break;
            }
            break;
            case 7:
            START_ALE=0;
            OE=0;
            select_channel(7);//选择通道 7 进行转化
            START_ALE=1;
            delay(5);
            START_ALE=0;
            delay(5);//等待 AD 转换完成
            while(EOC==1)
            {
                P1=0xff;//读数据之前将 P1 口的电压置高
                OE=1;
                temp1=P1/51.0;//将转换后的 16 进制准换成电压
                temp2=temp1*100;
                delay(5);
                OE=0;
                break;
            }
            break;
        }
}

void select_channel(unsigned char n)//通道选择函数
{
    switch(n)
    {
        case 0: add_C=0; add_B=0; add_A=0; break;//开启通道 0
        case 1: add_C=0; add_B=0; add_A=1; break;//开启通道 1
        case 2: add_C=0; add_B=1; add_A=0; break;//开启通道 2
        case 3: add_C=0; add_B=1; add_A=1; break;//开启通道 3
```

```
        case 4: add_C=1; add_B=0; add_A=0; break;//开启通道 4
        case 5: add_C=1; add_B=0; add_A=1; break;//开启通道 5
        case 6: add_C=1; add_B=1; add_A=0; break;//开启通道 6
        case 7: add_C=1; add_B=1; add_A=1; break;//开启通道 7
    }
}
```

ADC.h

```
#ifndef _ADC_H_
#define _ADC_H_
extern void ADC(unsigned char n);
extern void select_channel(unsigned char n);//通道选择函数
extern unsigned int temp2;
#endif
```

adjust_voltage.c

```
#include "adjust_voltage.h"
unsigned int down0=0,up0=500;//通道 0 的上下限值
unsigned int down1=0,up1=500;//通道 1 的上下限值
unsigned int down2=0,up2=500;//通道 2 的上下限值
unsigned int down3=0,up3=500;//通道 3 的上下限值
unsigned int down4=0,up4=500;//通道 4 的上下限值
unsigned int down5=0,up5=500;//通道 5 的上下限值
unsigned int down6=0,up6=500;//通道 6 的上下限值
unsigned int down7=0,up7=500;//通道 7 的上下限值
void adjust_up_add(unsigned char p)//调节对应通道上限电压值向上
{
    switch(p)
    {
        case 0:
            if((up0+10)<=500)       up0+=10;
            break;
        case 1:
            if((up1+10)<=500)       up1+=10;
            break;
        case 2:
            if((up2+10)<=500)       up2+=10;
```

```
            break;
        case 3:
            if((up3+10)<=500)       up3+=10;
            break;
        case 4:
            if((up4+10)<=500)       up4+=10;
            break;
        case 5:
            if((up5+10)<=500)       up5+=10;
            break;
        case 6:
            if((up6+10)<=500)       up6+=10;
            break;
        case 7:
            if((up7+10)<=500)       up7+=10;
            break;
    }
}

void adjust_up_reduce(unsigned char p)//调节对应通道上限电压值向下
{
    switch(p)
    {
        case 0:
            if(up0>down0+10)        up0-=10;
            break;
        case 1:
            if(up1>down1+10)        up1-=10;
            break;
        case 2:
            if(up2>down2+10)        up2-=10;
            break;
        case 3:
            if(up3>down3+10)        up3-=10;
            break;
        case 4:
            if(up4>down4+10)        up4-=10;
            break;
        case 5:
            if(up5>down5+10)        up5-=10;
```

```
            break;
        case 6:
            if(up6>down6+10)        up6-=10;
            break;
        case 7:
            if(up7>down7+10)        up7-=10;
            break;
    }
}

void adjust_down_add(unsigned char p)//调节对应通道下限电压值向上
{
    switch(p)
    {
        case 0:
            if(down0<(up0-10))      down0+=10;
            break;
        case 1:
            if(down1<(up1-10))      down1+=10;
            break;
        case 2:
            if(down2<(up2-10))      down2+=10;
            break;
        case 3:
            if(down3<(up3-10))      down3+=10;
            break;
        case 4:
            if(down4<(up4-10))      down4+=10;
            break;
        case 5:
            if(down5<(up5-10))      down5+=10;
            break;
        case 6:
            if(down6<(up6-10))      down6+=10;
            break;
        case 7:
            if(down7<(up7-10))      down7+=10;
            break;
    }
}
```

```
void adjust_down_reduce(unsigned char p)//调节对应通道下限电压值向下
{
    switch(p)
    {
        case 0:
            if((down0-10)>=0)       down0-=10;
            break;
        case 1:
            if((down1-10)>=0)       down1-=10;
            break;
        case 2:
            if((down2-10)>=0)       down2-=10;
            break;
        case 3:
            if((down3-10)>=0)       down3-=10;
            break;
        case 4:
            if((down4-10)>=0)       down4-=10;
            break;
        case 5:
            if((down5-10)>=0)       down5-=10;
            break;
        case 6:
            if((down6-10)>=0)       down6-=10;
            break;
        case 7:
            if((down7-10)>=0)       down7-=10;
            break;
    }
}
```

adjust_voltage.h

```
#ifndef _ADJUST_VOLTAGE_H_
#define _ADJUST_VOLTAGE_H_
extern void adjust_up_add(unsigned char p);//调节对应通道上限电压值向上
extern void adjust_up_reduce(unsigned char p);//调节对应通道上限电压值向下
extern void adjust_down_add(unsigned char p);//调节对应通道下限电压值向上
extern void adjust_down_reduce(unsigned char p);//调节对应通道下限电压值向下
externunsigned int down0,down1,down2,down3,down4,down5,down6,down7,up0,up1,up2,up3,up4,up5,up6,up7,temp2;
#endif
```

key_check.c

```
#include <reg52.h>
#include "adjust_voltage.h"
#include "lcd_display.h"
#include "key_check.h"
sbit KEY3=P3^0;
sbit KEY4=P3^1;
void KEY_Check()
{
    static unsigned char now=1,n=0,i=0,j=0;//n 用来控制刚开始按下进入哪个界面，第二次按下是对固定界面电压调节,i 的作用是若按键长时间没变化,隐藏调节界面
    //j 的作用是停留在哪个界面上持续控制
    unsigned char past,key=0;//key 来区别那个按键按下
    past=now;
    if(KEY3==0)     {key=3;now=0;}
    else if(KEY4==0)    {key=4;now=0;}
    else    now=1;
    if((past==1)&&(now==0))
    {
        i=0;
        if(n==0)///第一次按两个键中的任意一个是判断进入那个界面
        {
            if(key==3)  {display_adjust(1);  j=1;}//1 进入上限电压调节界面
            else if(key==4) {display_adjust(-1);  j=-1;}//-1 进入下限电压调节界面
            n+=1;
        }
        else//第二次以后开始调节电压
        {
            switch(j)
            {
                case 1://对上限电压进行调节
                    if(key==3)
                    {
                        adjust_up_add(flag);
                        display_adjust(1);
                    }
                    else if(key==4)
                    {
                        adjust_up_reduce(flag);
                        display_adjust(1);
```

```
                        }
                        break;
                    case -1://对下限电压进行调节
                        if(key==3)
                        {
                            adjust_down_add(flag);
                            display_adjust(-1);
                        }
                        else if(key==4)
                        {
                            adjust_down_reduce(flag);
                            display_adjust(-1);
                        }
                        break;
                }
            }
        }
        else if((past==1)&&(now==1))
        {
            i++;
            if(i==30)    {display_adjust(0); n=0;}
        }
}
```

key_check.h

```
#ifndef _KEY_CHECK_H_
#define _KEY_CHECK_H_
extern void KEY_Check();//检测调节电压上下限的按键
#endif
```

led_warning.c

```
#include <reg52.h>
#include "led_warning.h"
#include "adjust_voltage.h"
sbit led_en=P0^7;
void led_warn()
{
    switch(flag)
    {
        case 0:
```

```
                if((temp2>up0) || (temp2<down0)) led_en=1;
                else led_en=0;
                break;
            case 1:
                if((temp2>up1) || (temp2<down1)) led_en=1;
                else led_en=0;
                break;
            case 2:
                if((temp2>up2) || (temp2<down2)) led_en=1;
                else led_en=0;
                break;
            case 3:
                if((temp2>up3) || (temp2<down3)) led_en=1;
                else led_en=0;
                break;
            case 4:
                if((temp2>up4) || (temp2<down4)) led_en=1;
                else led_en=0;
                break;
            case 5:
                if((temp2>up5) || (temp2<down5)) led_en=1;
                else led_en=0;
                break;
            case 6:
                if((temp2>up6) || (temp2<down6)) led_en=1;
                else led_en=0;
                break;
            case 7:
                if((temp2>up7) || (temp2<down7)) led_en=1;
                else led_en=0;
                break;
        }
}
```

led_warning.h

```
#ifndef _LED_WARNING_H_
#define _LED_WARNING_H_
extern void led_warn();//led 警告
extern unsigned char flag;//flag 用来保存通道序号
#endif
```