线反转法行列键盘扫描

杜洋　2005-11-21

行列键盘的学习是单片机学习的必经之路，可是对了初学者来说学习起来并不容易。书上的资料不多，或是说明不细，抑或太复杂不易理解。我在学行列键盘时也有过此类的问题，近日我发现了一个非常好的行列扫描的方式——线反转法行列键盘扫描，它简单易懂，非常适合初学者学习，也可作为程序开发之用。我同时也写了汇编和 C 语言两个事例程序，不论你说什么语言都可以看懂。汇编和 C 的双语言例程也是我今后技术类文章的特点之一。

了解行列键盘扫描得从硬件开始学习，我们得知道行列扫描是什么意思。在单片机系统中为了扩大同一个 I/O 口的键盘个数，则采用了行列式键盘接法，就是交叉相接。如图：

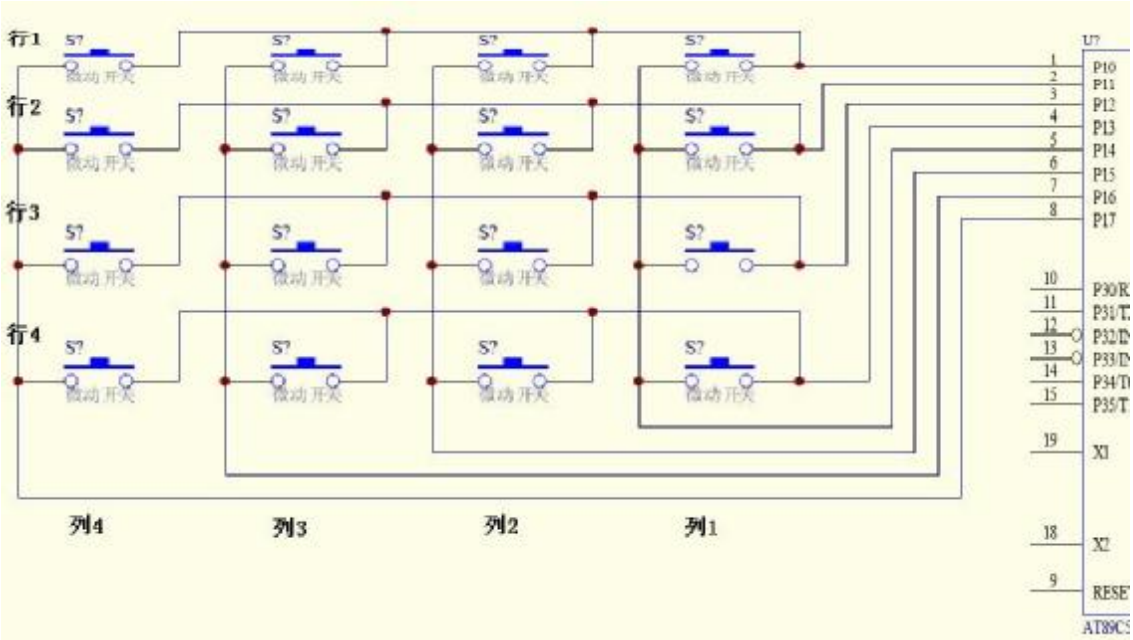

图 1

键盘接的前 4 个 I/O 口为行接线，后 4 个为列接线。这样的接法就构成了一个坐标，每一个键都对应这一个行的位置和一个列的位置。例如我们说左上角的那个所对应第 1 行和第 4 列，即单片机 P10 和 P17 两个 I/O 口。键盘的组成是用的微动开关，微动开关的特性是当有键按下时开关的两个引脚闭合导通。无按键时两个引脚是断开的状态。这样当我们按下图 1 中左上角的键时 P10 和 P17 在物理上是导通了，而其它的 I/O 口（P11~P16）都处于独立的状态。我们只要让单片机发现哪两个 I/O 口是导通的我们就可以知道是哪一个键被按下了。

这里我们用的一种方法是先将 4 个行线的 I/O 口置为“0”（低电平），将列线的 I/O 口置为“1”（高电平）。这样当有键按下时某一行的 I/O 口就和某一列

中的 I/O 口导通了，因为行线的口都是“0”（低电平）所以和行线导通的列线也将会变成“0”，而其余的列线因为开始时是“1”又没和其它的行线导通，所以依然是“1”。这样我们就可以找出了我们的按键所在的列了（因为列线中只有导通的列线变为了“0”，任何电平与低电平相导通都属于短路，短路的线将会是低电平）。

其实，所谓的“行”、“列”是我们人为规定的，如果试着把列看成行，将行看成列是一样的。这里我们规定 P10~P13 为行，P14~P17 为列。

现在我们知道了我们按下的键所在的列了，只要再知道它所在的行的话，我们就可以确定它的位置了。这时我们将 4 个行线的 I/O 口置为“1”（高电平），将列线的 I/O 口置为“0”（低电平），这是和最初的置式相反。被按着的那个按键还是导通的，还是属于短路，所以在被置“1”的行线中将会有一个变成了“0”，这样我们就确定了按键在行中的位置，到此我们还要确定什么呢？不需要了，我们已经找到了按下的键了。

举一个例子吧。例如我们按下了左上角的那个按键，首先当 4 个行线的 I/O 口置为“0”，列线的 I/O 口置为“1”时（即 P1 口的字节数据为 11110000），第 1 行和第 4 列导通，使第 4 列（P17）变为了“0”（此时的 P1 口的字节数据为 01110000）。之后当 4 个行线的 I/O 口置为“1”，列线的 I/O 口置为“0”时（即 P1 口的字节数据为 00001111），第 4 列和第 1 行导通，使第 1 行（P10）变为了“0”（此时的 P1 口的字节数据为 00001110）。通过了两次用单片机读出 2 个字节的数据，分别是“01110000”和“00001110”（从左向右，由高位到低位，P17—>P10）。前一个字节的低 4 位和后一个字节的高 4 位都是“0”，于是我们将它们去掉再将余下的数据拼成一个新的字节，即“01111110”。如果我们按下的是第一行的第 2 个键的话，我们得到的字节将会是“10111110”（第 1 行，第 3 列）。

每按一个键我们都得到不同的字节，比对我们的字节是什么就可以知道键值是什么了。

单片机行列键盘扫描程序流程：

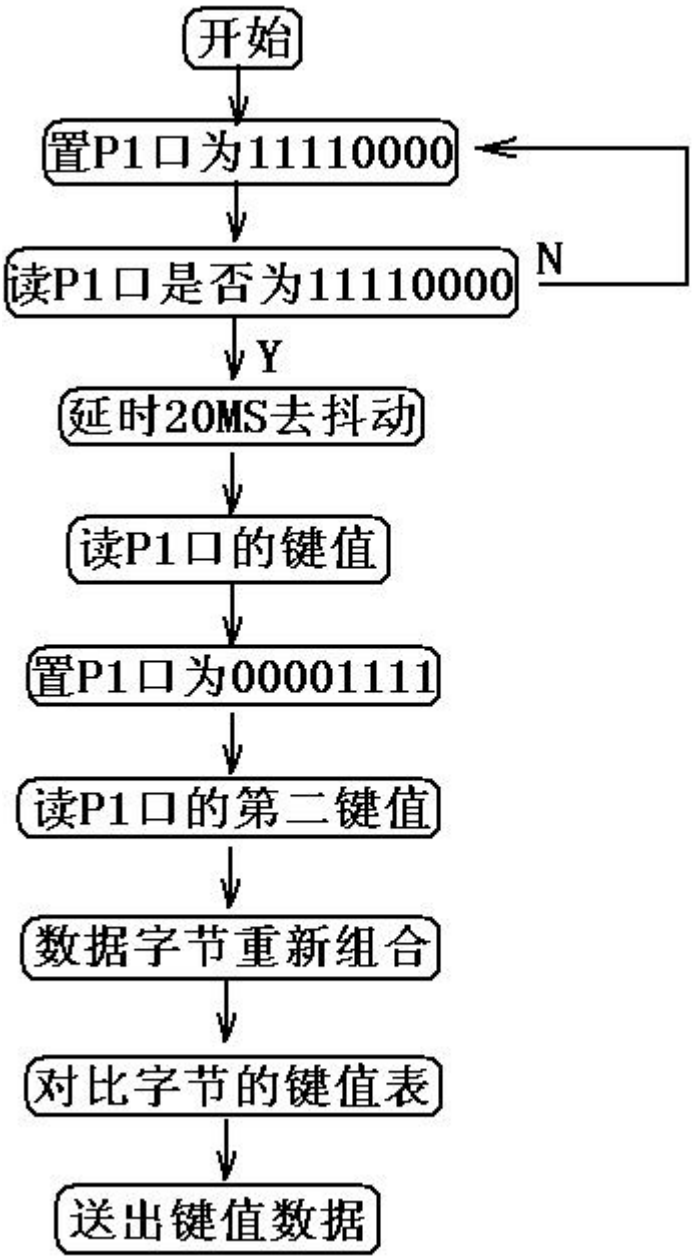

图 2

流程出来了，程序也就容易了许多，下面我们就来看一下扫描例程吧。程序是我自己写的，可能会有失误，仅供参考。（程序都已经通过测试）

C 语言例程：

```
/*----------------------------------------
项目名：        C 语言函数库
程序名：        4*4 扫描键盘
编写人：        杜洋
初写时间：      2005 年 11 月 18 日
程序功能：      调用后得出键盘数据返回值
实现方法：      占 CPU
CPU 说明：      MCS-51   （实验：AT89S52   12MHZ）
接口说明：      P3 口接 16 位行列键盘
信息说明：      将此函数放到工程文件夹中，在主程序中调用即可
函数声明：      调用延时函数 void Delay (uchar)
修改日志：
    NO.1-    2005-11-18   返回 P1 显示数据

----------------------------------------*/
//函数声明
#include <REG51.h>//////////////////MCS-51 头文件声明
#define  uchar unsigned char///////定义无符号变量简写式
#define  uint  unsigned int////////
sfr KEY = 0xb0;/////////////////////定义 P3 口为键盘接口
void Delay (uchar);////////////////延时程序声明（单位为毫秒）

Key ()//////////////////////////////键盘处理函数
{
uchar a,b,c;///////////////////////定义 3 个变量
KEY = 0x0f;////////////////////////键盘口置 00001111
if (KEY != 0x0f)///////////////////查寻键盘口的值是否变化
   {
   Delay (20);////////////////////延时 20 毫秒
   if  (KEY != 0x0f)//////////////有键按下处理
      {
     a = KEY;/////////////////键值放入寄存器 a
      }
    KEY = 0xf0;////////////////将键盘口置为 11110000
    c = KEY;///////////////////将第二次取得值放入寄存器 c
```

```
    a = a|c;//////////////////将两个数据熔合
    switch(a)//////////////////对比数据值
       {
       case 0xee: b = 0xee; break;///对比得到的键值给 b 一个应用数据
       case 0xed: b = 0xed; break;
       case 0xeb: b = 0xeb; break;
       case 0xe7: b = 0xe7; break;
       case 0xde: b = 0xde; break;
       case 0xdd: b = 0xdd; break;
       case 0xdb: b = 0xdb; break;
       case 0xd7: b = 0xd7; break;
       case 0xbe: b = 0xbe; break;
       case 0xbd: b = 0xbd; break;
       case 0xbb: b = 0xbb; break;
       case 0xb7: b = 0xb7; break;
       case 0x7e: b = 0x7e; break;
       case 0x7d: b = 0x7d; break;
       case 0x7b: b = 0x7b; break;
       case 0x77: b = 0x77; break;
       default:   break;//////////////////键值错误处理
       }
    }
return (b);////////////////////////////////////将 b 作为返回值
}

/*-----------------------------------------
函数名:         毫秒级延时函数
编写人:         杜洋
初写时间:       2005 年 9 月 29 日
程序功能:       空转延时
实现方法:       占用 CPU 的延时
CPU 说明:       MCS-51     外晶振 12MHZ   (非精确延时)
植入说明:      调用函数必须给延时函数一个 0~255 的延时值对应 0MS 到 255MS.
-----------------------------------------*/
void Delay(unsigned char a)
{
unsigned char i;
while( --a != 0)
{
for(i = 0; i < 125; i++);  //一个 ; 表示空语句,CPU 空转。
}       //i 从 0 加到 125, CPU 大概就耗时 1 毫秒
}
```

汇编程序：

```
/*----------------------------------------
项目名：        汇编子程序库
程序名：        行列键盘扫描子程序
编写人：        杜洋
初写时间：      2005 年 11 月 22 日
程序功能：      将 4*4 的键盘值送 LED 显示
实现方法：      线反转法
CPU 说明：       占 CPU
接口说明：      P3 口接键盘，P1 口接 LED
信息说明：
修改日志：
    NO.1-

----------------------------------------*/

;--------------------接口定义
LED    EQU    P1;;;;;LED 灯显示定义
KEY    EQU    P3;;;;;键盘接口定义

;--------------------程序入口定义
ORG    0000H
JMP    START;;;;;;;;芯片复位后的程序执行处
ORG    0030H

;--------------------初始化处理
START:
    MOV   LED,#0FFH;;;;LED 灯初始化全灭

;--------------------扫描循环体
LOOP:
    MOV   KEY,#0FH;;;;;装入键盘首次的扫描值
    MOV   A,KEY;;;;;;;;读出键盘值放入累加器 A
    CJNE A,#0FH,G01;;;读出的键值是否有变化，有变则跳到有键按下处理程序
    JMP   LOOP;;;;;;;;;无键按下跳回主循环
G01:
    CALL  DL20MS;;;;;;;有键按下延时 20 毫秒去抖
    MOV   A,KEY;;;;;;;;将键值给累加器
    CJNE A,#0FH,G02;;;重新判断键盘现状
    JMP   LOOP;;;;;;;;;是抖动则跳回主循环
G02:确定有键按下则
    MOV   KEY,#0F0H;;;;确定有键按下之后装入二次的扫描值
```

```
    MOV  B,KEY;;;;;;;;将二次扫描值放入寄存器 B 中
    ORL  A,B;;;;;;;;;;A 与 B 相或，得到一个字节的数据在 A 中

;----------------------键值对比处理
NE1:
    CJNE A,#0EEH,NE2;;;将 A 中的数据与键值表对比，不同则跳到下一个对比
    MOV  R2,#1;;;;;;;;;将输出显示数据送到寄存器 R2
    JMP  KEYEND;;;;;;;;跳出对比程序
NE2:
    CJNE A,#0EDH,NE3;;;（同上）
    MOV  R2,#2
    JMP  KEYEND
NE3:
    CJNE A,#0EBH,NE4
    MOV  R2,#3
    JMP  KEYEND
NE4:
    CJNE A,#0E7H,NE5
    MOV  R2,#4
    JMP  KEYEND
NE5:
    CJNE A,#0DEH,NE6
    MOV  R2,#5
    JMP  KEYEND
NE6:
    CJNE A,#0DDH,NE7
    MOV  R2,#6
    JMP  KEYEND
NE7:
    CJNE A,#0DBH,NE8
    MOV  R2,#7
    JMP  KEYEND
NE8:
    CJNE A,#0D7H,NE9
    MOV  R2,#8
    JMP  KEYEND
NE9:
    CJNE A,#0BEH,NE10
    MOV  R2,#9
    JMP  KEYEND
NE10:
    CJNE A,#0BDH,NE11
    MOV  R2,#10
    JMP  KEYEND
```

```
NE11:
    CJNE A,#0BBH,NE12
    MOV  R2,#11
    JMP  KEYEND
NE12:
    CJNE A,#0B7H,NE13
    MOV  R2,#12
    JMP  KEYEND
NE13:
    CJNE A,#07EH,NE14
    MOV  R2,#13
    JMP  KEYEND
NE14:
    CJNE A,#07DH,NE15
    MOV  R2,#14
    JMP  KEYEND
NE15:
    CJNE A,#07BH,NE16
    MOV  R2,#15
    JMP  KEYEND
NE16:
    CJNE A,#077H,KEYEND
    MOV  R2,#16
KEYEND:
    MOV  LED,R2;;;;;;;;;;;;将输出显示数据送出显示
    JMP  LOOP;;;;;;;;;;;;;;跳回主循环

;------------------20 毫秒延时，主要用于去抖动。(100,100)
DL20MS:
    MOV    R6,#100;;;;;;;;;
DL20MS_1:
    MOV    R7,#100;;;;;;;;;
    DJNZ   R7,$;;;;;;;;;;;;
    DJNZ   R6,DL20MS_1;;;;;
    RET

END
```