9.3.1 矩阵键盘的工作原理和扫描确认方式

来源：《AVR 单片机嵌入式系统原理与应用实践》M16 华东师范大学电子系 马潮

当键盘中按键数量较多时，为了减少对 I/O 口的占用，通常将按键排列成矩阵形式，也称为行列键盘，这是一种常见的连接方式。矩阵式键盘接口见图 9-7 所示，它由行线和列线组成，按键位于行、列的交叉点上。当键被按下时，其交点的行线和列线接通，相应的行线或列线上的电平发生变化，MCU 通过检测行或列线上的电平变化可以确定哪个按键被按下。

图 9-7 为一个 4 x 3 的行列结构，可以构成 12 个键的键盘。如果使用 4 x 4 的行列结构，就能组成一个 16 键的键盘。很明显，在按键数量多的场合，矩阵键盘与独立式按键键盘相比可以节省很多的 I/O 口线。

矩阵键盘不仅在连接上比单独式按键复杂，它的按键识别方法也比单独式按键复杂。在矩阵键盘的软件接口程序中，常使用的按键识别方法有行扫描法和线反转法。这两种方法的基本思路是采用循环查循的方法，反复查询按键的状态，因此会大量占用 MCU 的时间，所以较好的方式也是采用状态机的方法来设计，尽量减少键盘查询过程对 MCU 的占用时间。

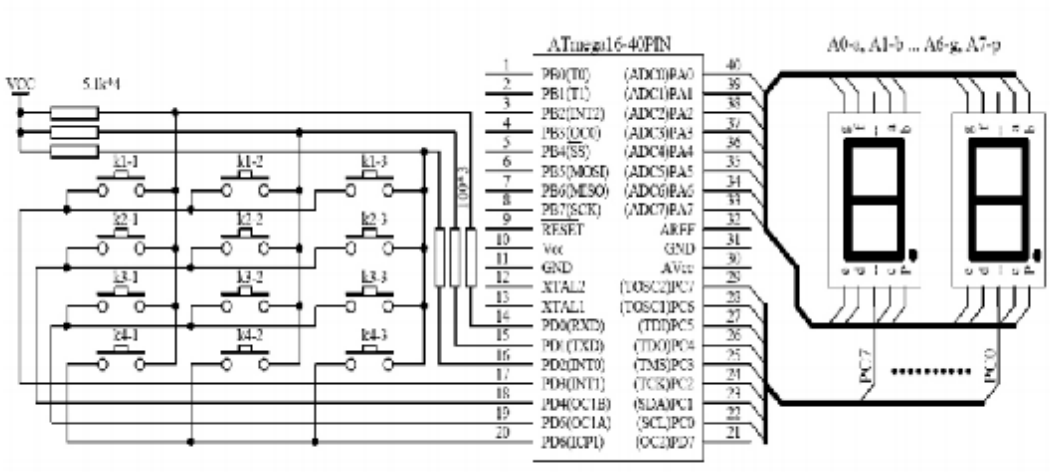

图 9-7 4*3 矩阵键盘的组成

下面以图 9-7 为例，介绍采用行扫描法对矩阵键盘进行判别的思路。图 9-7 中，PD0、PD1、PD2 为 3 根列线，作为键盘的输入口（工作于输入方式）。PD3、PD4、PD5、PD6 为 4 根行线，工作于输出方式，由 MCU（扫描）控制其输出的电平值。行扫描法也称为逐行扫描查询法，其按键识别的过程如下。

- √ 将全部行线 PD3－PD6 置低电平输出，然后读 PD0－PD2 三根输入列线中有无低电平出现。只要有低电平出现，则说明有键按下（实际编程时，还要考虑按键的消抖）。如读到的都是高电平，则表示无键按下。
- √ 在确认有键按下后，需要进入确定具体哪一个键闭合的过程。其思路是：依

次将行线置为低电平，并检测列线的输入（扫描），进而确认是具体的按键位置。如当 PD5 输出低电平时（PD3=1、PD4=1、PD5=0、PD6=1），测到 PD1 的输入为低电平（PD0=1、PD1=0、PD2=1），则可确认按键 K3-2 处于闭合状态。通过以上分析可以看出，MCU 对矩阵键盘的按键识别，是采用扫描方式控制行线的输出和检测列线输入的信号相配合实现的。

√ 矩阵按键的识别仅仅是确认和定位了行和列的交叉点上的按键，接下来还要考虑键盘的编码，即对各个按键进行编号。在软件中常通过计算的方法或查表的方法对按键进行具体的定义和编号。

在单片机嵌入式系统中，键盘扫描只是 MCU 的工作内容之一。MCU 除了要检测键盘和处理键盘操作之外，还要进行其他事物的处理，因此，MCU 如何响应键盘的输入需要在实际系统程序设计时认真考虑。

通常，完成键盘扫描和处理的程序是系统程序中的一个专用子程序，MCU 调用该键盘扫描子程序对键盘进行扫描和处理的方式有三种：程序控制扫描、定时扫描和中断扫描。

√ 程序控制扫描方式。在主控程序中的适当位置调用键盘扫描程序，对键盘进行读取和处理。

√ 定时扫描方式。在该方式中，要使用 MCU 的一个定时器，使其产生一个 10ms 的定时中断，MCU 响应定时中断，执行键盘扫描，当在连续两次中断中都读到相同的按键按下（间隔 10ms 作为消抖处理），MCU 才执行相应的键处理程序中断方式。使用中断方式时，键盘的硬件电路要做一定的改动，增加一个按键产生中断信号的输入线，当键盘有按键按下时，键盘硬件电路产生一个外部的中断信号，

MCU 响应外部中断，进行键盘处理。来

下面我们介绍基于状态机并采用定时键盘扫描的键盘处理系统的设计方法。

9.3.2 定时扫描方式的键盘接口程序

根据图 9-7，下面的键盘接口函数 read_keyboaed() 完成了对 4*3 键盘的扫描识别和键盘的编码。编码键盘的定义使用 define 语句定义，键盘的形式类似电话和手机键盘，如图 9-8 所示。

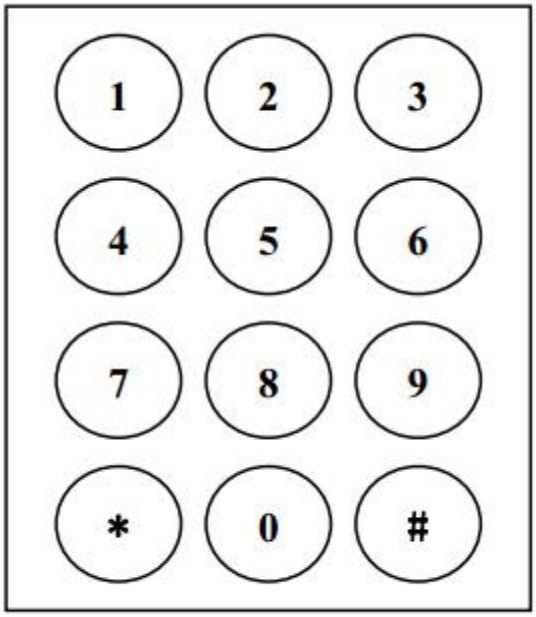

图 9-8 手机键盘

```
#define No_key      255
#define K1_1    1
#define K1_2    2
#define K1_3    3
#define K2_1    4
#define K2_2    5
#define K2_3    6
#define K3_1    7
#define K3_2    8
#define K3_3    9
#define K4_1    10
#define K4_2    0
#define K4_3    11
#define Key_mask      0b00000111


char read_keyboard()
{
   static char key_state = 0, key_value, key_line;
   char key_return = No_key,i;
   switch (key_state)
    {
      case 0:
         key_line = 0b00001000;
         for (i=1; i<=4; i++)                          //  扫描键盘
          {
```

```
      PORTD = ~key_line;                    // 输出行线电平
      PORTD = ~key_line;                    // 必须送 2 次!!!（注 1
          key_value = Key_mask & PIND;      // 读列电平
      if (key_value == Key_mask)
          key_line <<= 1;                   // 没有按键，继续扫描
      else
       {
        key_state++;                        // 有按键，停止扫描
        break;                              // 转消抖确认状态
       }
     }
    break;
case 1:
    if (key_value == (Key_mask & PIND))     // 再次读列电平，
     {
      switch (key_line | key_value)         // 与状态 0 的相同，确认按键
       {                                    // 键盘编码，返回编码值
         case 0b00001110:
              key_return = K1_1;
              break;
       case 0b00001101:
              key_return = K1_2;
              break;
       case 0b00001011:
              key_return = K1_3;
              break;
       case 0b00010110:
              key_return = K2_1;
              break;
       case 0b00010101:
              key_return = K2_2;
              break;
       case 0b00010011:
              key_return = K2_3;
              break;
       case 0b00100110:
              key_return = K3_1;
              break;
       case 0b00100101:
              key_return = K3_2;
              break;
       case 0b00100011:
              key_return = K3_3;
              break;
```

```
            case 0b01000110:
                key_return = K4_1;
                break;
            case 0b01000101:
                key_return = K4_2;
                break;
            case 0b01000011:
                key_return = K4_3;
                break;
          }
          key_state++;          // 转入等待按键释放状态
        }
        else
          key_state--;          // 两次列电平不同返回状态 0，（消抖处理）
        break;
      case 2:                   // 等待按键释放状态
        PORTD = 0b00000111;     // 行线全部输出低电平
        PORTD = 0b00000111;     // 重复送一次
        if ( (Key_mask & PIND) == Key_mask)
          key_state=0;          // 列线全部为高电平返回状态 0
        break;
    }
    return key_return;
}
```

系统主程序应每隔 10ms 调用该键盘接口函数 read_keyboaed()，函数返回值为 255 时表示无按键按下。检测和确认按键按下时，函数返回值为 0 到 11 之间的一个，该返回值已经是经过了键盘编码的值。

键盘接口函数 read_keyboaed()是基于状态机实现的，将键盘扫描处理过程化分成三个状态，每个状态的功能为：

- √ 状态 0，键盘扫描检测。控制 PD3-PD6，4 根行线逐行输出低电平，对键盘进行扫描检测。一旦检测到有键按下（key_value），立即停止键盘的扫描，状态转换到状态 1。注意此时变量 key_value 中保存着读到的列线输入值，而且该行线低电平的输出是保持不变的。
- √ 状态 1，消抖处理和键盘编码。再次检测键盘列线的输入，并与状态 0 时的 key_value 比较，不相等则返回状态 0，实现了消抖处理。相等则确认该键的输入，进行键盘编码和设置函数的返回值，状态转化到状态 2。
- √ 状态 2，等待按键释放。控制 PD3-PD6，4 根行线全部输出低电平，检测 3 根列线输入全部为高电平（无按键按下）时状态返回到状态 0。

读者在阅读该段程序时，请注意 key_mask、key_value、key_line 的作和用

它们不仅与键盘的硬件连接有关系，同时还要注意他们在程序中是如何使用的，其值的保存等等。

通过这个例子也可以看出，在硬件设计的过程中，也需要认真考虑软件的写。比如，你也可以将 4 根键盘行线与 PD0、PD2、PD3、PD6 连接，列线使用 PD1、PD4、PD5，从硬件的角度看是完全可以的，但会给软件编写造成很多麻烦。所以，一个好的嵌入式系统工程师必须同时具备良好的软件设计编写能力和硬件设计能力。

实际上 read_keyboaed()还是一个比较简单的键盘接口函数，还不能处理类似多键按下的识别、按键“连发”等功能。但当你真正掌握了基本键盘接口的设计思想和方法后，就可以在这个简单的键盘接口函数基础上，设计出功能更加完善的键盘系统了。

（注 1）当 AVR 的 I/O 口处于输入方式工作时，其对应的 PINx 就是外部引脚上的实际电平。为了防止读入 PINx 时数据的不稳定和不确定，（例如在读的期间，正好引脚电平发生改变），AVR 的 I/O 输入端到总线之间加入一个同步锁存器，用以保证读入数据的稳定和确定。同步锁存器在系统 I/O 时钟的作用下，经过 1/2-3/2 个系统时钟周期，将引脚电平锁定，提供 MCU 读取。也就是说外部引脚的电平变化要经过 1/2-3/2 个时钟周期才能真正的被读到，见图 9-9。

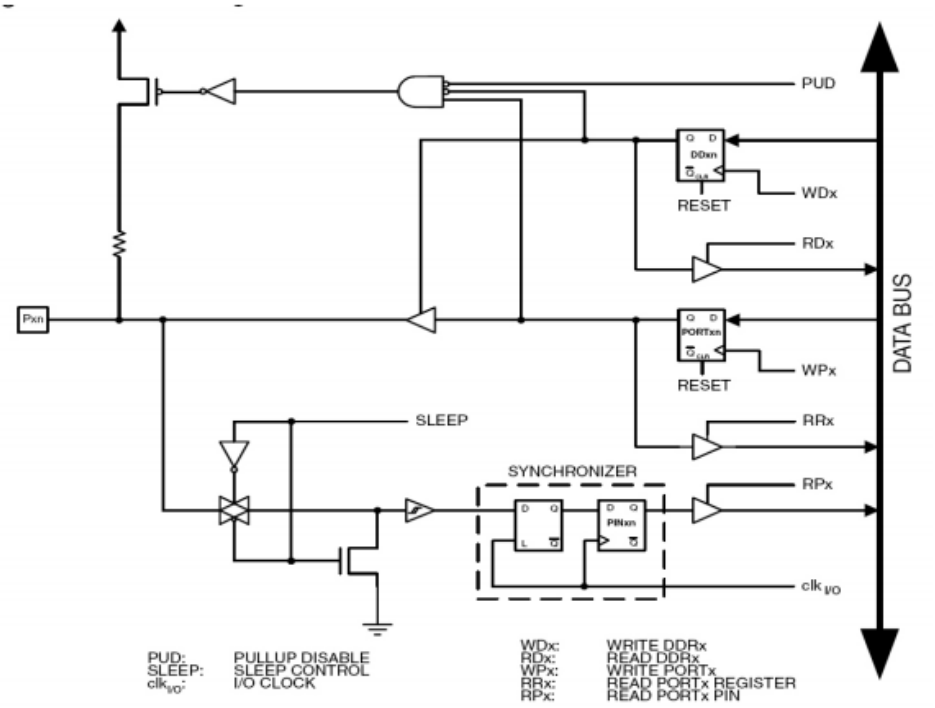

图 9-9 AVR I/O 内部结构图

因此，当编写程序读取 AVR 的 I/O 口 PINx 电平值时应注意：

当 I/O 口从输出方式改成输入方式后，应延时一个 CLK 再读。

当 I/O 外部引脚电平改变后，也要延时一个 CLK 后再读取。

在本例的键盘扫描程序中，根据扫描条件，首先改变行线输出的电平，如果按键按下的话，那么对应点的列线（输入口）电平也马上改变了，此时需要延时一个 CLK 后读列线的输入电平值才是正确的。程序中采用输出 2 次相同的行电平的方式，第 2 次输出的实际作用是起到延时的作用（其实相当于 2 个 NOP）。当然，也可以输出 1 次行电平，在读 I/O 口时采用读 2 次的方式，只要满足规定的延时时间就可以的。

例 9.3 简单电话拨号键盘的设计

1 ）硬件电路

在这个例子中，结合图 9-7 的硬件电路和图 9-8 定义的键盘，实现一个简单的电话拨号键盘。系统由 PA、PC 口控制的 8 个 LED 数码管和 PD 口的 4*3 键盘组成，系统上电时，8 个 LED 数码管显示“--------”8 条横线，每按下一个号码后，原 8 位 LED 数码管的显示内容向左移动一位，最右边一位则显示键盘上刚按下的数字（“*”键用“A”表示，“#”键用“b”表示）。要求：对键盘按键操作的反应迅速而且无误，同时按键操作过程中应保证 LED 的扫描显示均匀、连续不间断。

2）软件设计

```
/*********************************************
File name              : demo_9_3.c
Chip type              : ATmega16
Program type           : Application
Clock frequency        : 4.000000 MHz
Memory model           : Small
External SRAM size     : 0
Data Stack size        : 256
*********************************************/

#include <mega16.h>
flash char led_7[13]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,   // 字型码
0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x40};
                  // 后 3 位为 “A”，“b”，“-”
flash char position[8]={0x7f,0xbf,0xdf,0xef,0xf7,0xfb,0xfd,0xfe};

char dis_buff[8];            // 显示缓冲区，存放要显示的 8 个字符的段码值
```

```
char key_stime_counter;
char posit;
bit    key_stime_ok;

void display(void)          // 8 位 LED 数码管动态扫描函数
{
    PORTC = 0xff;
    PORTA = led_7[dis_buff[posit]];
    PORTC = position[posit];
    if (++posit >=8 ) posit = 0;
}
// Timer 0 比较匹配中断服务,2ms 定时
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
    display();             // 调用 LED 扫描显示
    if (++key_stime_counter >=5)
   {
        key_stime_counter = 0;
        key_stime_ok = 1;              // 10ms 到
   }
}
#define No_key     255
#define K1_1    1
#define K1_2    2
#define K1_3    3
#define K2_1    4
#define K2_2    5
#define K2_3    6
#define K3_1    7
#define K3_2    8
#define K3_3    9
#define K4_1    10
#define K4_2    0
#define K4_3    11
#define Key_mask    0b00000111
```

```
char read_keyboard()
{
    static char key_state = 0, key_value, key_line;
    char key_return = No_key,i;
   。。。。。。          //  同上面 read_keyboard()
 }

void main(void)
{
    char i, key_temp;

    PORTA = 0x00;              //  显示控制 I/O 端口初始化
    DDRA = 0xFF;
    PORTC = 0xFF;
    DDRC = 0xFF;
    PORTD = 0xFF;              //  键盘接口初始化
    DDRD = 0xF8;              // PD2、PD1、PD0 列线，输入方式，上拉有效
    // T/C0 初始化
    TCCR0=0x0B;          //  内部时钟，64 分频（4M/64=62.5KHz），CTC 模式
    TCNT0=0x00;
    OCR0=0x7C;           // OCR0 = 0x7C(124),(124+1)/62.5=2ms
    TIMSK=0x02;          //  允许 T/C0 比较匹配中断

    for (i=0; i<8 ;i++)
    {dis_buff[i]= 12;}     // LED 初始显示 8 个“-”
    #asm("sei")          //  开放全局中断

    while (1)
   {
       if (key_stime_ok)
    {
          key_stime_ok = 0;              // 10ms 到
          key_temp = read_keyboard();     //  调用键盘接口函数读键盘
          if (key_temp != No_key)
```

```
        {           // 有按键按下
                for (i=0; i<7; i++)
                {dis_buff[i] = dis_buff[i+1];}   // LED 显示左移一位
                dis_buff[7] = key_temp;          // 最右显示新按下键的键值
        }
      }
    };
}
```