第 5 章

数码显示与键盘接口

第 2～4 章分别讲述了 STC15 单片机的片内资源与内部结构，51 单片机的 C51 语言编程基础，单片机仿真与调试技术，分别对应单片机的硬件、软件和常用开发工具基础知识，以此为基础，可以将单片机内部集成的外围设备资源介绍与单片机应用实例相结合，探讨单片机应用系统开发技术。

单片机应用系统是由单片机和相关的外围设备共同组成的。不同的应用需求需要配置不同的外围设备，而人机对话设备通常是单片机应用系统中不可缺少的组成部分，数码显示与矩阵键盘是最常用的人机对话设备，本章专题研究其接口和编程技术。

视频

5.1 数码管及其显示接口

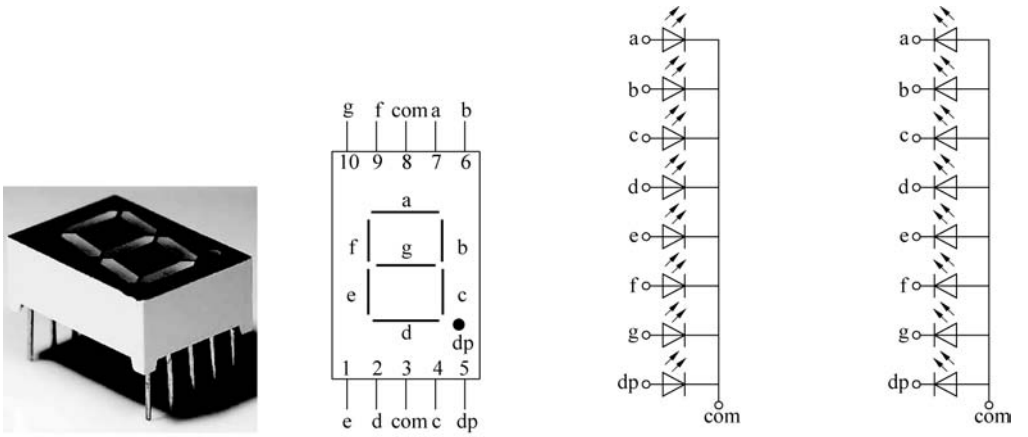
LED 七段数码显示器俗称数码管，具有价廉、可视性好、结构简单、品种丰富、接口灵活等特点，广泛应用于单片机系统中，是最常用的显示设备(输出设备)。

5.1.1 数码管及其分类

七段数码显示器(或数码管)是由 8 只发光二极管按一定空间位置排列构成的，有多种尺寸规格，封装有 1 位(单码)、2 位、4 位、6 位和 8 位等形式，不同的尺寸和封装的数码显示器引脚排列也不同。例如，0.5 英寸 1 位数码管的实物和引脚封装如图 5.1(a)和(b)所示，其中第 3 和 8 脚为公共端，称为数码管的位选线，其余 8 脚各对应内部 8 只 LED 的一个极，称为数码管的段选线 a，b，…，g，dp。根据内部 8 个 LED 的连接方式，数码管可分为共阴极和共阳极两种结构，分别如图 5.1(c)和(d)所示。

共阳极数码管是将 8 只笔段 LED 的阳极连接在一起，如图 5.1(c)所示。通常将公共阳极接高电平(一般接电源)，各笔段(阴极)引脚分别串联电阻后接段驱动电路输出端。当某笔段驱动电路的输出端为低电平时，则该笔段导通并被点亮，如图 5.2(a)所示。

共阴极数码管是将 8 只笔段 LED 的阴极连接在一起，如图 5.1(d)所示。通常将公共阴极接低电平(一般接地)，各笔段(阳极)引脚分别串联电阻后接笔段驱动电路输出端。当某笔段驱动电路的输出端为高电平时，该笔段导通并被点亮。

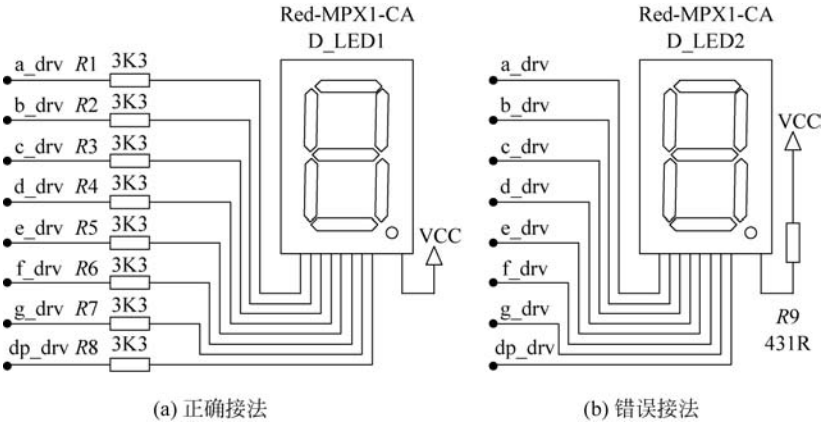


(a) 实物图　(b) 引脚封装　(c) 共阴极　(d) 共阳极

图 5.1　数码管结构图

5.1.2　数码管驱动电路

在 STC15 单片机应用系统中，准双向模式 I/O 口可直接驱动共阳极数码管；推挽模式 I/O 口可直接驱动共阴极或共阳极数码管。无论是采用共阴极还是共阳极连接，都需要根据各笔段的额定导通电流和外接电源来确定各笔段所需的限流电阻。由于二极管伏安特性的离散性导致二极管并联时电流分配是不均匀的，因此二极管一般不宜并联使用，数码管应避免在共阳极（或共阴极）串接公共的限流电阻，如图 5.2 所示，其中图(b)接法是错误的，8 只笔段 LED 的总电流是固定的，显示不同字符时，点亮的 LED 数目不同，数码管的亮度不同。



(a) 正确接法　(b) 错误接法

图 5.2　单只共阳极数码管驱动电路

要使数码管显示某个字符，必须从 8 个笔段驱动接口输出相应的字形编码。假设在单片机应用系统中，以单片机 Pn 口的 Pn.0～Pn.7 分别驱动数码管的 a～dp 笔段。当端口输出高电平时，共阴极数码管对应的笔段将被点亮；反之，当端口输出低电平时，共阳极数码

管对应的笔段将被点亮。据此可得数码管显示各种字符的字形编码如表 5.1 所示，字形编码也称作段码。

表 5.1　数码管显示字符的段码表（不带小数点）

显示字符	共阴极字形码	共阳极字形码	显示字符	共阴极字形码	共阳极字形码
0	3FH	C0H	9	6FH	90H
1	06H	F9H	A	77H	88H
2	5BH	A4H	b	7CH	83H
3	4FH	B0H	C	39H	C6H
4	66H	99H	d	5EH	A1H
5	6DH	92H	E	79H	86H
6	7DH	82H	F	71H	8EH
7	07H	F8H	全亮	7FH	80H
8	7FH	80H	全灭	00H	FFH

5.1.3　数码管显示方式

1. 数码管静态显示

静态显示是指数码管显示某一字符时，相应的发光二极管恒定导通或恒定截止。这种显示方式的各位数码管相互独立，公共端（即位选线）恒定接地（共阴极）或接正电源（共阳极）。每个数码管的 8 个笔段线分别串接电阻后与一个 8 位 I/O 口相连，I/O 口只要有段码输出，相应的字符即显示出来并保持不变，直到 I/O 口输出新的段码。采用静态显示方式，较小的电流即可获得较高的亮度，且显示驱动程序占用 CPU 时间少，编程简单，电路故障和软件错误易排查，但其占用的 I/O 口多，硬件电路复杂，成本高，只适用于显示位数较少的场合。

【例 5.1】 数码管静态显示。标准单片机 AT89C51 的 I/O 口是准双向口，灌电流驱动能力比较大，可直接驱动共阳极数码管。如图 5.3 所示，AT89C51 单片机主时钟 12.0MHz，用其 P1 和 P2 口直接驱动 2 位共阳极数码管，小数点不显示，dp 笔段悬空，数码管采用静态驱动。为该硬件系统设计一软件，要求实现以下功能：2 位数码管每隔 1 秒向左滚屏 1 位，依次显示 9，8，7，…，0，9，8，…

```
#include <reg51.h>

#define uchar unsigned char
#define uint unsigned int

uchar code segTab[ ] = {                    //在 CODE 区定义七段码译码表
    0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,
    0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e
};
uchar data disBuf[2] = {0,0};               //在 DATA 区定义显示缓冲区

void disp_s();                              //声明静态显示函数
```

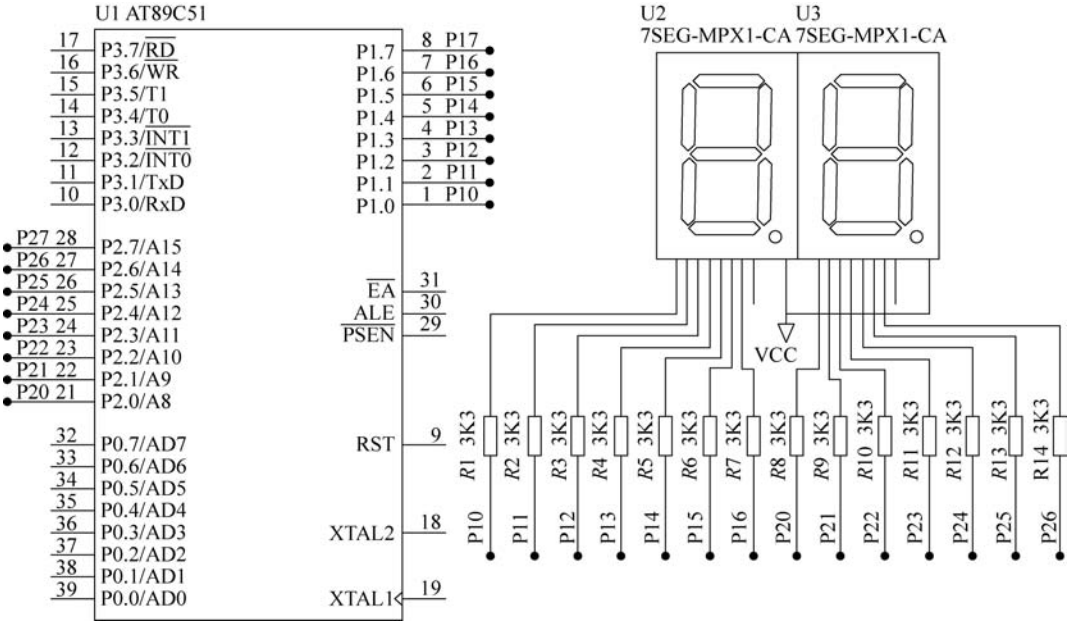


图 5.3　用 AT89C51 的 P1 和 P2 口直接驱动 2 只共阳极数码管(静态显示)

```
void delay(uchar);

void disp_s()                          //静态显示函数定义
{
    P1 = segTab[disBuf[0]&0x0f];
    P2 = segTab[disBuf[1]&0x0f];
}

void delay(uchar td)                   //定义延时函数,延时 100ms * td,主时钟 12.0MHz
{
    uint i;
    while(td--)
        for(i=0;i<8329;i++);
}

void main()
{
    uchar j;
    while(1)
    {
        for(j=0;j<10;j++)
        {   disBuf[0] = 9-j;           //显示缓冲区赋值
            disBuf[1] = (9-j)>0?(8-j):9;
            disp_s();                  //调用静态显示函数
            delay(10);                 //延时 1 秒
        }
    }
}
```

2. 数码管动态显示

动态显示是指多位 LED 数码管逐位被轮流点亮，这种逐位点亮的方式称为位扫描，2 位共阳数码管动态显示电路如图 5.4 所示。在数码管动态显示电路中，通常将各位数码管对应的段选线并联在一起，分别串接电阻后由 1 个 8 位的 I/O 口控制，各个数码管的位选线（即 com 端）由其他的 I/O 口线分别控制。当数码管以动态方式显示时，各位数码管分时轮流选通，即在某一时刻只选通一位数码管，并送出该位字符对应的段码，在下一时刻选通下一位数码管，再送出该位字符对应的段码。依此规律循环即可使各位数码管显示不同的字符，虽然这些字符是在不同的时刻分别显示，但由于人眼存在视觉暂留效应，只要所有数码管的循环扫描频率达到每秒 30 次以上，人眼就感觉不到闪烁，达到各位数码管连续稳定地显示不同的字符的效果。例如，4 位数码管动态显示方式其段选和位选输出数据流如表 5.2 所示，如果每位扫描时间为 2.5ms，则循环扫描周期为 10ms，扫描频率达到每秒 100 次。

表 5.2　4 位数码管动态显示方式段选和位选数据流

时间流	……→时间进程→……									
扫描	前个扫描周期			当前扫描周期				下个扫描周期		
时间片	……	2.5ms	2.5ms	2.5ms	2.5ms	2.5ms	2.5ms	2.5ms	2.5ms	……
段码数据（8 位）	……	第 3 位段码	第 4 位段码	第 1 位段码	第 2 位段码	第 3 位段码	第 4 位段码	第 1 位段码	第 2 位段码	……
第 1 位选线	……	禁止	禁止	允许	禁止	禁止	禁止	允许	禁止	……
第 2 位选线	……	禁止	禁止	禁止	允许	禁止	禁止	禁止	允许	……
第 3 位选线	……	允许	禁止	禁止	禁止	允许	禁止	禁止	禁止	……
第 4 位选线	……	禁止	允许	禁止	禁止	禁止	允许	禁止	禁止	……

与静态显示方式相比，动态显示方式有如下特点：

（1）动态显示方式节省 I/O 口，限流等硬件电路也比较简单，如 4 位数码管，静态显示需 32 个 I/O 口，而动态显示仅需要 12 个 I/O 口。

（2）各位数码管是轮流导通，如果不提高每位数码管导通时的瞬时电流，那么各位数码管的平时电流将下降，数码显示的亮度也下降，因此动态显示时，数码管导通的瞬时电流要提高（限流电阻降低）。

（3）CPU 要依次循环扫描各个数码管，导致显示驱动程序复杂，占用较多的 CPU 时间。

（4）每位数码管导通的时间必须是均等的，否则将导致数码管显示亮度不均匀，另外，单片机程序要避免在某一局部模块内循环。

【例 5.2】 将例 5.1 的数码管改为动态显示。如图 5.4 所示为 2 位共阳极数码管动态显示电路，AT89C51 的 P1 口为段选驱动，P2.6 和 P2.7 口为位选控制，程序修改如下：

```
#include <reg51.h>
#define uchar unsigned char
#define uint unsigned int
```

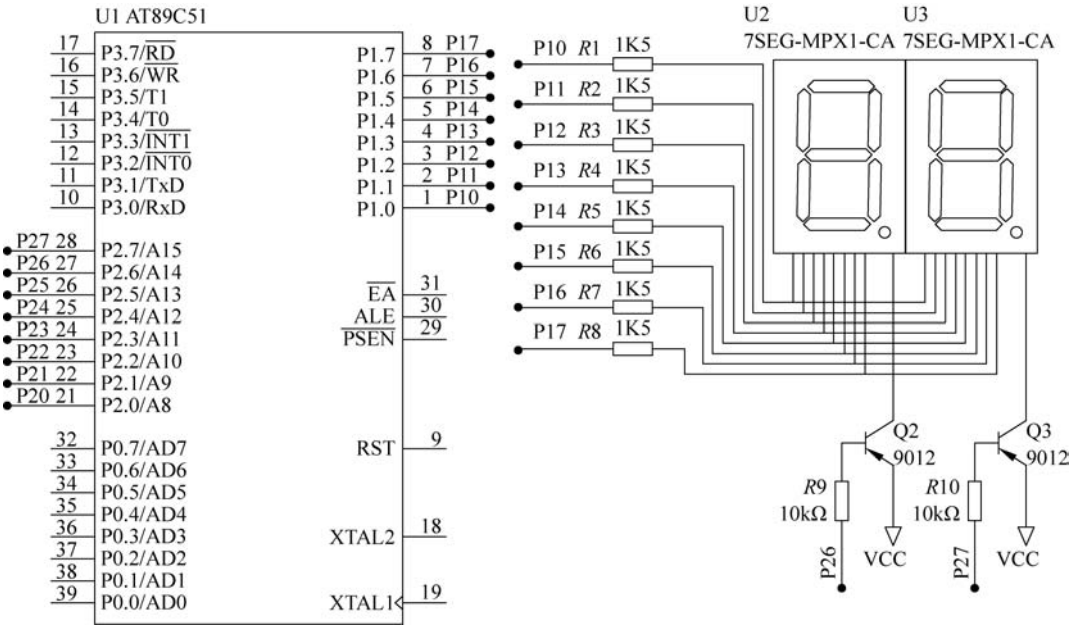


图 5.4　两只共阳极数码管动态显示电路(P1 段码驱动,P2.6 和 P2.7 位扫描控制)

```
uchar code segTab[ ] = {                          //在 CODE 区定义七段码译码表
    0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,
    0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e
};
uchar code disScan[2] = {~(1 << 6),~(1 << 7)};    //在 CODE 区定义位选码
uchar data disBuf[2];                             //在 DATA 区定义显示缓冲区
uchar data disNum;                                //在 DATA 区定义扫描位控制变量
void disp_d();                                    //声明动态显示函数
void delay2ms5();
void disp_d()                                     //动态显示函数定义
{   P2| = 3 << 6;                                 //关闭显示器
    P1 = segTab[disBuf[disNum]&0x0f];             //输出扫描位的段码
    P2& = disScan[disNum];                        //输出扫描位的扫描码
    disNum = (disNum + 1) % 2;                    //准备扫描下一位
}
void delay2ms5()
{   uint i;
    for(i = 0;i < 427;i++);                       //2.5ms 延时函数
}
void main()
{   uchar j = 0;
    uint tx = 0;
    while(1)
    {   delay2ms5();                              //延时 2.5ms
        disp_d();
        tx++;
```

```
            if(tx == 400)                                    //1s 到?
            {   tx = 0;
                disBuf[0] = 9 - j;                           //显示缓冲区赋值
                disBuf[1] = (9 - j)> 0?(8 - j):9;
                j = (j + 1) % 10;
            }
        }
}
```

如图 5.4 所示，在数码管动态显示接口电路中，不管数码管有几位，输出段码仅需要占用单片机的 1 个 8 位 I/O 口，当数码管位数增加时，所需的位选控制端口则相应增加。

5.1.4 用 74HC595 扩展数码显示接口

单片机应用系统常用带锁存功能的 8 位移位寄存器 74HC595 扩展 LED 显示接口，IAP15W4K58S4 实验板上的 8 位共阴极数码显示模块即采用此扩展方法，其电路如图 4.36 所示。为了不过多占用 IAP15W4K56S4 单片机的 I/O 资源，使用 2 片具有三态输出且带锁存功能的 8 位串入、并出或串出的移位寄存器 74HC595 扩展了一个串行输入转 16 位并行输出的接口，由于 74HC595 的输出端 Qn 的拉电流（或灌电流）最大可达 35mA，整个芯片的最大功率可达 500mW，可直接用 16 位的扩展并行输出口驱动数码管显示模块。这样仅用 STC15 单片机的 3 条 I/O 口线（最少 I/O 资源），实现 8 位数码管显示模块的扫描控制，其中 P4.0 作为串行数据输出线 DS、P4.3 作为移位脉冲输出线 SH_CP、P5.4 作为移位寄存器内部数据锁存脉冲线 ST_CP。图 5.5 给出了 74HC595 的内部结构与功能框图，其真值表如表 5.3 所示。

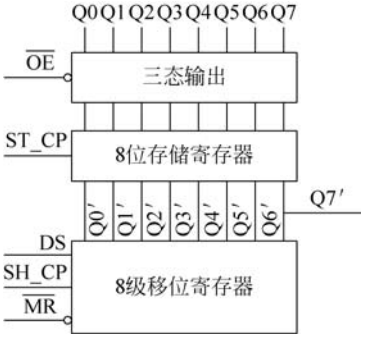


图 5.5 74HC595 的内部结构与功能框图

表 5.3 74HC595 的真值表

输入					内部			输出		功能
SH_CP	ST_CP	$\overline{OE}$	$\overline{MR}$	DS	$Q0'$	$Q1'$	…	$Q7'$	Qn	
×	×	L	L	×	L	L	…	L	不变	MR 低电平，移位寄存器清零
×	↑	L	L	×	L	L	…	L	L	ST_CP 上升沿将移位寄存器零存储
×	×	H	L	×	L	L	…	L	高阻	OE 高电平，输出悬浮（高阻态）
↑	**×**	**L**	**H**	**×**	**DS**	$Q0'$	**…**	$Q6'$	**不变**	**SH_CP 上升沿串行数据内部移位**
×	**↑**	**L**	**H**	**×**	**不变**	**不变**	**…**	**不变**	Qn'	**ST_CP 上升沿将移位寄存器值存储**
↑	↑	L	H	×	DS	$Q0'$	…	$Q6'$	Qn'	ST_CP 上升沿将移位寄存器先前值存储，SH_CP 上升沿串行数据内部移位

使用 2 片 74HC595 扩展 16 位的串入并出接口，其工作原理是数码显示模块扫描数据的串行传输，详细说明如下：

(1) 所谓串行数据传输，是指仅用 1 条信号线 DS 传输信息，如要传输 1 字节的二进制数据，那么数据只能按位依序(先高位后低位，或先低位后高位)分时用该信号线传输，每个时间片只传输 1 位数据。例如，将 1 字节的数据 0x4e 以先高位后低位的顺序从 P4.0 端口(即 DS 端)串行输出，则 P4.0 将依次输出 0，1，0，0，1，1，1，0，每位数据占用多少时间与传输速率有关。

(2) 74HC595 是 8 位串入/并出或串出的移位寄存器，带锁存和三态输出功能，从真值表 5.3 的倒数第 3 行可见，SH_CP 引脚的脉冲上升沿触发内部 8 级移位寄存器移位，原先内部 $Q0'$～$Q6'$的数据移到了 $Q1'$～$Q7'$，DS 端的数据移入 $Q0'$，即 DS→$Q0'$→$Q1'$→…→$Q6'$→$Q7'$，$Q7'$是 74HC595 的串行数据的输出端，用于级联下 1 个 74HC595 芯片；从真值表 5.3 的倒数第 2 行可见，当输出允许 $\overline{OE}$=0 时，ST_CP 引脚的脉冲上升沿触发内部移位数据 $Q0'$～$Q7'$锁存到 8 位存储寄存器，并从 8 位三态输出口 $Q0$～$Q7$ 输出。图 4.36 中，单片机的串行数据从 P4.0 输出，传输给 U6(即左边的 74HC595 电路)的串行数据输入端 DS，U6 的串行数据输出端 $Q7'$再传输给 U5(即右边的 74HC595 电路)的串行数据输入端 DS，U6 和 U5 两芯片的移位脉冲引脚 SH_CP 并接，数据锁存脉冲引脚 ST_CP 并接，两片 8 位移位寄存器 74HC595 级联，构成 16 位串入/并出的扩展 I/O 口。在 16 位的扩展并行输出口中，U5 的输出口 $Q7$～$Q0$ 对应高 8 位，作为数码管显示模块的 8 条位选线，U6 的输出口 $Q7$～$Q0$ 对应低 8 位，作为数码管显示模块的段码输出端口。

(3) 如有 2 字节(16 位)数据，其二进制数据位分别为 $B2.7$～$B2.0$ 和 $B1.7$～$B1.0$，要将该 2 字节数据从扩展的 16 位串入/并出接口输出，单片机输出的接口信号 P40_HC595_DS、P43_HC595_SH、P54_HC595_ST 如图 5.6 所示。$B2.7$ 是最早串行输出的数据位，经过 16 次移位(16 个 SH_CP 脉冲作用)传输，最终从 U5 的 $Q7$ 端口输出。

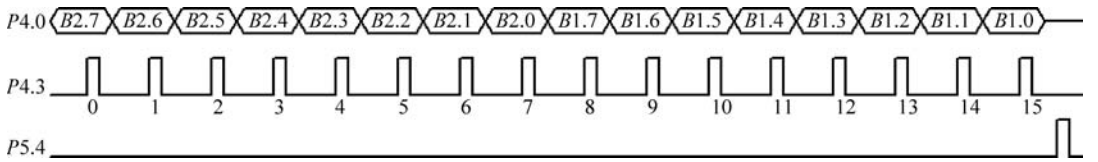


图 5.6　从 16 位串入/并出扩展接口输出 2 字节数据的接口信号

要控制实验板共阴极数码管显示模块以动态方式显示，核心的程序是如何将数码管扫描的段码和位选码从 16 位扩展并口输出。如图 5.6 所示，可先编写从 STC 单片机串行输出 1 字节数据的函数 send_595(uchar x)如下：

```
sbit P_595_DS = P4^0;              //定义 74HC595 的串行数据接口
sbit P_595_SH = P4^3;              //定义 74HC595 的移位脉冲接口
sbit P_595_ST = P5^4;              //定义 74HC595 的输出寄存器锁存信号接口
void send_595(uchar);              //声明移位输出 1 字节数据函数
```

```
void send_595(uchar x)              //从 STC 单片机移位输出 1 字节数据
{   uchar i;
    for(i = 0;i < 8;i++)            //循环移位,共 8 位
    {   x << = 1;                   //左移 1 位,最高位移出到 CY
        P_595_DS = CY;              //CY 从串行数据口输出
        P_595_SH = 1;               //输出移位脉冲
        P_595_SH = 0;
    }
}
```

先执行 send_595(disScan[disNum]),再执行 send_595(segTab[disBuf[disNum]]),即可将当前扫描位的位选码和段码数据分别发送到移位寄存器 U5 和 U6 的内部(未锁存),然后再从 P_595_ST(即 P5^4)引脚输出 1 个锁存脉冲 ST_CP,即将当前扫描位的位选码和段码数据从 16 位串入/并出接口输出,完成当前位的扫描。实验板 8 位共阴极数码管动态扫描函数如下:

```
uchar code segTab[ ] = {                          //在 CODE 区定义七段码译码表
    0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
    0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71
};
uchar code disScan[8] = {                         //在 CODE 区定义扫描码
    ~(1 << 0),~(1 << 1),~(1 << 2),~(1 << 3),~(1 << 4),~(1 << 5),~(1 << 6),~(1 << 7)
};
uchar data disBuf[8];                             //在 DATA 区定义显示缓冲区
uchar data disNum;                                //在 DATA 区定义位扫描控制变量

void disp_d();                                    //声明动态显示函数

void disp_d(void)                                 //显示驱动函数
{   send_595(disScan[disNum]);                    //将当前扫描位扫描码发送
    send_595(segTab[disBuf[disNum]]);             //将当前扫描位段码发送
    P_595_ST = 1;                                 //16 位数据移位后锁入输出寄存器中
    P_595_ST = 0;
    disNum = (disNum + 1) % 8;                    //调整扫描位的值,指向下一位
}
```

视频

5.2 键盘接口电路及其消抖动

按键或其他开关元件也是单片机应用系统最常用的信息或控制量输入元件,与通用计算机系统不同,单片机系统的按键没有统一固定的规格,也没有专用接口电路来处理按键等开关控制量,需要根据应用系统的需要配置按键等开关元件。

5.2.1 按键开关及其接口电路

1. 按键开关的结构与分类

单片机系统通常使用廉价的、简单的、小型化的按键等开关元件作为开关量输入装置,

常用的有：按钮、按钮式薄膜开关、触摸开关、拨码开关、数字拨码开关、拨动开关(或微动开关)、自锁按钮开关、限位开关、干簧管等。

按键或开关按工作原理可以分为两类：一类是触点式开关按键，如机械式开关和导电橡胶式开关等；另一类是无触点开关按键，如开关管、晶闸管和固态继电器等。按结构特点可分为按钮型开关和闸刀型开关(或拨动型)。按钮开关按其开关状态可以分为两类：一类是常开型，即按下闭合，释放则断开；另一类是常闭型，即按下断开，释放则闭合。

2．按键开关与单片机的简单接口

按键开关总是通过一定的接口电路与 CPU 相连，以两种不同的逻辑电平表示“闭合”和“断开”两种状态，或用逻辑电平跃变的上升沿和下降沿表示“按下”和“释放”状态。CPU 可以通过查询或中断的方式确定是否有按键被按下以及是哪个按键被按下，进而根据系统既定的功能执行相应的程序代码。图 5.7 是开关及其与单片机的简单接口电路，不论是按钮开关还是闸刀开关，与单片机的接口均可采用下拉或上拉方式。

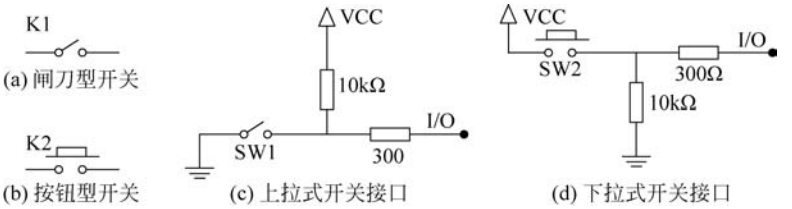


图 5.7　开关及其与单片机的简单接口电路

图 5.7(c)和(d)中 10kΩ 的电阻为上拉或下拉电阻(量级为 10kΩ 级)，其作用是保证开关断开时，单片机的 I/O 端口有确定的电平，上拉式为高电平，下拉式为低电平；300Ω 电阻可以省略(量级为百欧级)，其作用是 I/O 端口保护。采用上拉式开关接口时，开关导通时，I/O 端口逻辑电平为 0，断开为 1，开关状态用负逻辑表示；采用下拉式开关接口时，开关导通时，I/O 端口逻辑电平为 1，断开为 0，开关状态用负正逻辑表示。

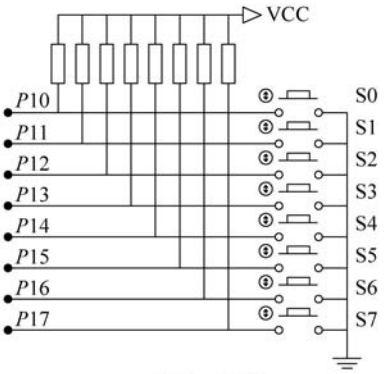
3．键盘的结构和工作方式

键盘通常由一组规则排列的按键组成，根据按键的接线方式的不同可分为独立式键盘和矩阵式键盘，其结构如图 5.8 所示。

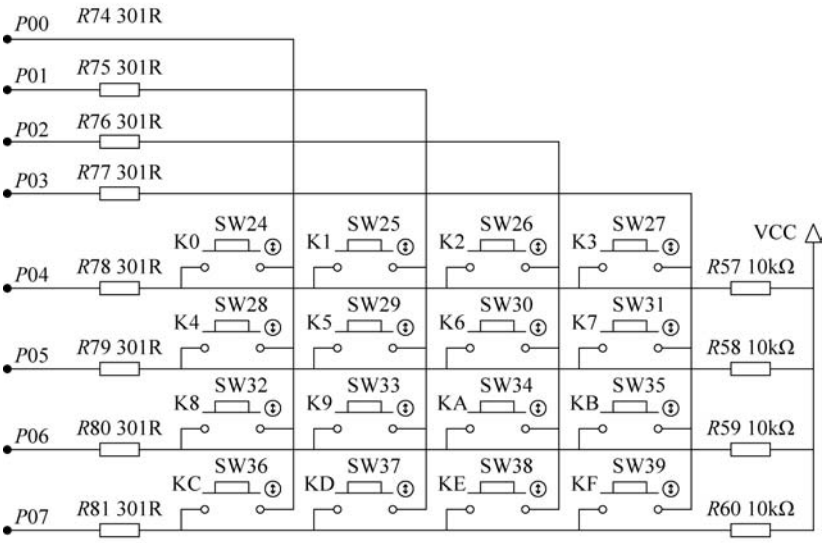
独立式键盘是直接用 I/O 端口构成简单的按键接口电路，如图 5.8(a)所示，其特点是每个按键(或开关)单独占用一根 I/O 口线，每个按键的工作不会影响其他 I/O 口线的状态。独立式键盘电路配置灵活，软件结构简单，但是占用 I/O 端口较多，一般只在按键数量较少的场合下使用。

矩阵式键盘由行列控制线组成，按键(或开关)跨接在行列控制线之间，如图 5.8(b)所示，行列控制线简称行线、列线。矩阵式键盘可以在使用比较少的 I/O 端口的情况下得到比较多的按键数量，但是软件结构相对比较复杂，一般在对按键数量要求比较多的场合下使用。

键盘电路中上拉电阻的作用是确保在没有按键按下的情况下将 CPU 端口上拉为高电



(a) 独立式键盘



(b) 矩阵式键盘

图 5.8 键盘结构

平。实际应用中可根据单片机 I/O 口内部电路结构进行取舍，若单片机 I/O 口内部本身已经有上拉电阻，则键盘中的上拉电阻可省略。

键盘的工作方式应根据实际应用系统中 CPU 的工作状况而定，其选取的原则是既要保证 CPU 能及时响应按键操作，又不要过多地占用 CPU 的工作时间。通常，键盘的工作方式有 3 种，即编程扫描、定时扫描和中断扫描。

编程扫描方式是利用 CPU 完成其他工作的空余时间调用键盘扫描子程序来响应键盘输入的要求。在执行键功能程序时，CPU 不再响应键输入要求，直到 CPU 重新扫描键盘为止。

定时扫描方式是指每隔一段时间对键盘扫描一次，根据键盘的输入要求执行相应的程序功能。在对扫描时间精度要求不高的情况下，可利用主软件延时程序的方法实现定时扫

描。在对扫描时间精度要求较高的情况下，可利用定时器定时中断实现，有关定时器中断内容详见第 6 章。

中断扫描方式是指将键盘电路采用一定的方式与 CPU 的外中断信号输入引脚进行关联。当有键盘操作的时候产生相应的外中断请求，CPU 在中断响应中进行相应的键盘处理，有关外中断内容详见第 6 章。

5.2.2　按键抖动与键信号消抖动处理

机械式按钮按下或释放时，由于机械弹性作用的影响，总是伴随有一定时间的触点机械抖动，之后触点才稳定下来。在抖动期间，触点的连接状态、导电特性不稳定，接口信号的电平也不稳定，其抖动过程如图 5.9 所示，图中 t_1 和 t_3 为抖动时间，其时长与按钮开关的机械特性有关，一般小于 20ms；t_2 为按键闭合的稳定期，其时间由使用者按键的动作确定，一般为几百毫秒以上，t_0 和 t_4 为按键释放期。

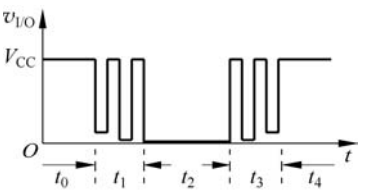


图 5.9　机械抖动导致下拉式按键接口信号电平的抖动

与处理速度为微秒级的单片机相比而言，这种机械抖动是不可忽略的。如果在触点抖动期间进行按键的通断状态检测，那么可能会导致判断出错，即按键一次操作（按下或释放）被错误地认为是多次操作，从而使单片机产生错误的动作，这是不允许出现的。因此，为了避免按键触点机械抖动所导致的检测误判，必须采取相应的去抖动措施。消除按键抖动可以采用硬件方法，如在按键电路中增加 RS 触发器电路或 RC 积分电路进行消抖；也可采用软件方法，在按键扫描程序中增加相应的代码进行消抖。前者需要增加电路成本，且设备体积也随之增大；后者仅占用少量的 CPU 时间，单片机应用系统多采用软件方法消抖。

软件实现键信号去抖动处理的基本思想是：延时法，即当 CPU 检测到有按键按下时，执行一个 20ms 左右（时长可按键类型适当调整）的延时程序后再进行按键检测，如果检测到按键仍处于被按下状态，则确认按键被按下；反之，则认为是机械抖动引起的状态变化。对按键释放识别也是采用相同的办法处理。需要注意的是，如果单片机软件系统采用按键定时扫描方式，且扫描周期比软件去抖动的延时时间短，则需要对去抖动的延时程序做特殊的处理，否则可能会引起键盘误读错误。

【例 5.3】　使用实验板的以下硬件资源：

（1）单片机 IAP15W4K58S4；

（2）两个连接到 P3.2 和 P3.3 端口的独立式下拉按键 SW17 和 SW18，如图 4.29 所示；

（3）数码显示模块的最左边 2 位，如图 4.32 所示。

试用这些硬件资源构成一个键控计数器，要求实现以下功能：

（1）启动时系统显示 50；

（2）SW17 键作为“加 1”键，按该键一次，显示数值加 1，最大计数值 99；

(3) SW18 作为“减 1”键,按该键一次,显示数值减 1,最小计数值 01。

```
#include <stc15.h>
#define uchar unsigned char
#define uint unsigned int
uchar code segTab[] = {                    //在 CODE 区定义七段码译码表
    0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
    0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71
};
uchar code disScan[2] = {~(1<<0),~(1<<1)};      //在 CODE 区定义扫描码仅最左边 2 位
uchar data disBuf[2];                      //在 DATA 区定义显示缓冲区
uchar data disNum;                         //在 DATA 区定义位扫描控制变量
sbit SW17 = P3^2;
sbit SW18 = P3^3;
sbit P_595_DS = P4^0;                      //定义 74HC595 的串行数据接口
sbit P_595_SH = P4^3;                      //定义 74HC595 的移位脉冲接口
sbit P_595_ST = P5^4;                      //定义 74HC595 的输出寄存器锁存信号接口
void send_595(uchar);                      //声明移位输出 1 字节数据函数
void disp_d();                             //声明动态显示函数
void delay2ms5();
void delay20ms();

void send_595(uchar x)                     //从 STC 单片机移位输出 1 字节数据
{   uchar i;
    for(i = 0;i < 8;i++)                   //循环移位,共 8 位
    {   x <<= 1;                           //左移 1 位,最高位移出到 CY
        P_595_DS = CY;                     //CY 从串行数据口输出
        P_595_SH = 1;P_595_SH = 0;         //输出移位脉冲
    }
}

void disp_d()                              //动态显示函数定义
{   send_595(disScan[disNum]);             //将当前扫描位扫描码发送
    send_595(segTab[disBuf[disNum]]);      //将当前扫描位段码发送
    P_595_ST = 1;P_595_ST = 0;             //16 位数据移位后锁入输出寄存器中
    disNum = (disNum + 1) % 2;             //准备扫描下一位,仅 2 位
}

void delay2ms5()
{   uint i;
    for(i = 0;i < 2930;i++);               //2.5ms 延时函数
}
void delay20ms()
{   uint i;
    for(i = 0;i < 24000;i++);              //20ms 延时函数
}
```

```
void main()
{   uchar cnt = 50;                    //定义计数变量 cnt
    P3M1& = ～(3 << 2);P3M0& = ～(3 << 2);
    P4M1& = ～9;P4M0& = ～9;P5M1& = ～16;P5M0 = ～16;
    while(1)
    {   disBuf[0] = cnt/10;disBuf[1] = cnt % 10;
        delay2ms5();                   //延时 2.5ms
        disp_d();
        SW17 = 1;                      //P3.2 读之前写 1
        if(SW17 == 0)                  //有 SW17 键?
        {   delay20ms();               //延时消抖
            while(!SW17);              //等 SW17 键释放
            delay20ms();               //延时消抖
            if(cnt < 99)cnt++;         //计数未达上限时,执行 + 1
        }
        SW18 = 1;                      //P3.3 读之前写 1
        if(SW18 == 0)                  //有 SW18 键?
        {   delay20ms();               //延时消抖
            while(!SW18);              //等 SW18 键释放
            delay20ms();               //延时消抖
            if(cnt > 1)cnt -- ;        //计数未达下限时,执行 - 1
        }
    }
}
```

其中加黑的 3 个程序行是仅使用数码显示最左边 2 位时动态显示程序的相应修改。将代码下载到实验板中试运行,可验证系统功能正确性。

程序中使用编程扫描方式读取键状态,结合延时消抖、等待键释放等措施实现键每按下一次,增减一个计数值。但这个程序存在致命问题,即延时消抖、等待键释放与数码管动态显示的循环扫描驱动相冲突,导致键按下时,数码管显示停留在某一位上。

5.3　数码动态显示与键信号消抖动处理的协同


视频

例 5.3 中有数码动态显示与键信号消抖动处理两个任务,前者需要定时循环,后者需要延时并等待键释放,二者的冲突源自两个任务处理程序是独立编写的,没有从多任务系统的角度考虑整个系统编程问题,本节尝试解决这一问题。与操作系统所述的多任务处理有所不同,本节所述的多任务都不是并发的,是可以采用轮询方式分时处理的。一个可在实际系统中运用得好的键信号消抖动处理程序应具备以下几个特点:

(1) 能正确识别按键信息;

(2) 具有软件消抖动功能;

(3) 与其他程序模块(特别是数码管动态显示程序)能协同工作,不互相影响;

(4) 可以给出多种按键信息,如键状态变化前后沿提取。

5.3.1 多任务系统程序结构

含数码动态显示和键信号消抖动处理的多任务单片机系统流程如图 5.10 所示，程序每隔 2.5ms 启动一次主循环，完成定时读键盘保存键状态、数码显示动态扫描、完成按键状态消抖动等处理，及其他任务（如按键发出“嘀”声响、按键解释与响应等）。主程序在 2.5ms 延时后完成所有各项子任务，若所有任务都比较简单，则处理时间可忽略不计，可大致认为程序的循环时间就是 2.5ms。主程序的循环之所以取 2.5ms，是考虑到若将本例的方法移植到 8 位数码显示模块的系统，这样的循环扫描速度仍不会出现显示器频闪。在第 6 章，2.5ms 延时改用定时器实现，CPU 就有更多的时间处理更复杂的任务。

5.3.2 键信号处理

1. 消抖动处理

所有的机械按键在按下的最初 20ms 内，触点未达到稳定接触，连接按键的 I/O 端口的电平不稳定，CPU 读到的键状态不稳定，即所谓的键抖动，如图 5.11 所示。按键消抖动处理方法有多种，软件延时消抖动是最常用的方法，其算法原理是 CPU 一旦检测到键状态非零，表明有键按下，延时一段时间（不小于 20ms）后，等待键状态稳定后，再读取有效的键状态。

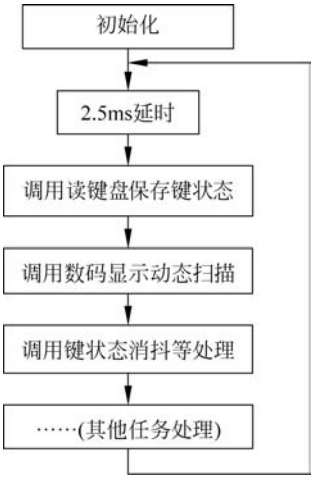


图 5.10 数码动态显示与键信号处理流程

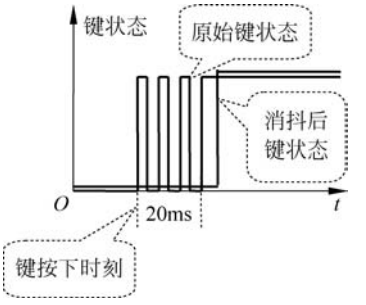


图 5.11 键抖动及处理

定义全局变量 edgk 用于保存从 I/O 接口读取的键状态，此处规定键状态统一用正逻辑表示，即按下或闭合为 1，释放或断开为 0。edgk 应是“可位寻址的”，每位对应一个键，以便主程序可对各键状态进行查询，另定义全局变量 ktmr 作为消抖计时器。若采用如图 5.10 所示的主程序循环，则系统每 2.5ms 调用一次“扫描键盘保存键状态”及“键状态消

抖等处理”函数，其中“键状态消抖等处理”算法流程图如图 5.12 所示。用已读取的键状态判断，无键时消抖计时器清零；当有键按下时，消抖计时器加 1，由于主循环为 2.5ms，因此消抖计时器的每个计数值相当于 2.5ms，消抖计时器计数达到 8 即已延时 20ms。消抖计时未达 20ms，则丢弃不稳定的键状态；若 20ms 计时已到，则启用新读取的稳定的键状态。

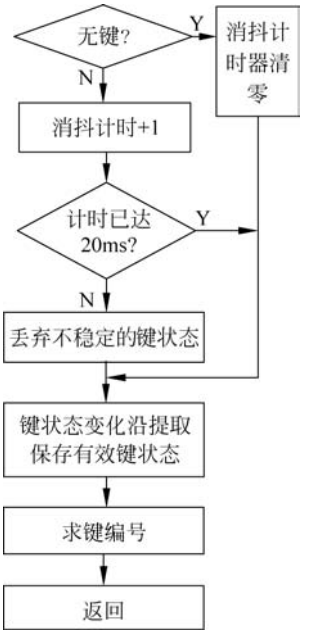


图 5.12　键状态消抖等处理子程序流程图

2. 键状态变化沿提取

在微机系统中，按键通常有两种用法。其一是不论按键按下时间长短，按一次只起一次作用，这种按键是键状态变化的前沿或后沿（对应按下或释放）起作用，不妨称这种按键为触发键；其二是按键只在按下时才有作用，一旦按键释放其作用消失，这种按键是键状态（持续的导通或断开）起作用，不妨称这种键为开关键。如前所述，用变量 edgk 保存键状态变化沿，有变化沿为 1，无变化沿为 0，即 edgk 保存触发键的键状态；另定义变量 key 用于保存键状态，它保存的是开关键的键状态。若前后两次循环的键状态不同（异或为 1），且本次循环的键状态为 1，则表明出现了键状态变化前沿，或发生了键按下；若前后两次循环的键状态不同（异或为 1），且前次循环的键状态为 1，则表明出现了键状态变化后沿，或发生了键释放；若前后两次循环的键状态相同（异或为 0），则表明没有键动作发生。综上，键状态变化沿提取算法可表示为：

```
键状态变化前沿 = [(前次循环键状态)^(本次循环键状态)]&(本次循环键状态)
键状态变化后沿 = [(前次循环键状态)^(本次循环键状态)]&(前次循环键状态)
```

【例 5.4】 按本节所述方法，重新设计例 5.3 软件。

新程序代码清单如下，与例 5.3 比较，其中加黑的程序行为新增代码。

```
#include<stc15.h>
#define uchar unsigned char
#define uint unsigned int
uchar code segTab[] = {                          //在 CODE 区定义七段码译码表
    0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
    0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71
};
uchar code disScan[2] = {~(1<<0),~(1<<1)};       //在 CODE 区定义扫描码
uchar data disBuf[2];                            //在 DATA 区定义显示缓冲区
uchar data disNum;                               //在 DATA 区定义位扫描控制变量
uchar bdata key;                                 //声明变量,键状态
uchar bdata edgk;                                //声明变量,键变化前沿
uchar data ktmr;                                 //定义变量,消抖计时器
uchar data kcode;                                //定义变量,键号
```

```
sbit EK1 = edgk^0;                          //SW17 的键状态(触发型)
sbit EK2 = edgk^1;                          //SW18 的键状态(触发型)
sbit P_595_DS = P4^0;                       //定义 74HC595 的串行数据接口
sbit P_595_SH = P4^3;                       //定义 74HC595 的移位脉冲接口
sbit P_595_ST = P5^4;                       //定义 74HC595 的输出寄存器锁存信号接口
void send_595(uchar);                       //声明移位输出 1 字节数据函数
void disp_d();                              //声明动态显示函数
void delay2ms5();                           //声明 2.5ms 延时函数
void readkey();                             //读键盘保存键状态函数
void keytrim();                             //键状态消抖动处理函数

void send_595(uchar x)                      //从 STC 单片机移位输出 1 字节数据
{   uchar i;
    for(i = 0;i < 8;i++)                    //循环移位,共 8 位
    {   x <<= 1;                            //左移 1 位,最高位移出到 CY
        P_595_DS = CY;                      //CY 从串行数据口输出
        P_595_SH = 1;P_595_SH = 0;          //输出移位脉冲
    }
}

void disp_d()                               //动态显示函数定义
{   send_595(disScan[disNum]);              //将当前扫描位扫描码发送
    send_595(segTab[disBuf[disNum]]);       //将当前扫描位段码发送
    P_595_ST = 1;P_595_ST = 0;              //16 位数据移位后锁入输出寄存器中
    disNum = (disNum + 1) % 2;              //准备扫描下一位
}

void delay2ms5()
{   uint i;
    for(i = 0;i < 2930;i++);                //2.5ms 延时函数
}

void readkey()                              //扫描键盘存键状态
{   P3| = 3 << 2;                           //P3.3、P3.2 准双向,读之前先写 1
    edgk = (~P3 >> 2)&0x03;                 //读键状态,求反转正逻辑
}

void keytrim()                              //键状态消抖动,键前沿提取,求键号
{    uchar temp;                            //本行以下为: 消抖动
    if(edgk == 0)ktmr = 0;                  //无键,消抖计时器清零
    else
    {   if(ktmr < 255)ktmr++;               //有键,消抖计时器 + 1(防溢出)
        if(ktmr < 8)edgk = 0;               //延时未到弃不稳定键
    }
    temp = edgk;                            //本行以下为: 键前沿提取.键状态暂存
    edgk = (key^edgk)&edgk;                 //此时 key 还保存着上次循环键状态
    key = temp;                             //暂存的本次循环健状态移至 key
    if(edgk!= 0)                            //本行以下为: 求键编号,无键为 0x10
    {   temp = edgk;
```

```
        for(kcode = 0;temp&1 == 0;kcode++)temp >> = 1;
    }
    else kcode = 0x10;
}
void main()
{   uchar cnt = 50;                                    //定义计数变量 cnt
    P3M1& = ~(3 << 2);P3M0& = ~(3 << 2);
    P4M1& = ~9;P4M0& = ~9;P5M1& = ~16;P5M0 = ~16;
    while(1)
    {   disBuf[0] = cnt/10;disBuf[1] = cnt % 10;
        delay2ms5();                                   //调用延时 2.5ms
        disp_d();                                      //调用数码动态显示扫描
        readkey();                                     //调用读键状态保存健状态
        keytrim();                                     //调用键信号消抖动等处理
        if(EK1)                                        //有 SW17 键?
            if(cnt < 99)cnt++;                         //计数未达上限时,执行 + 1
        if(EK2)                                        //有 SW18 键?
            if(cnt > 1)cnt -- ;                        //计数未达下限时,执行 - 1
    }
}
```

将代码下载到实验板中试运行,可验证系统功能的正确性。

在例 5.3 的新软件中,键状态消抖动和键状态变化沿提取(或触发型键状态生成)是 keytrim()函数中的重要算法。为加深对键状态变化前沿提取算法的理解,下面以 SW17 键按动一次为例,每次调用 keytrim()函数前后,键状态信号处理过程中的各种重要中间结果如表 5.4 所示。从表 5.4 可清晰地看出键状态的消抖动和键状态变化沿提取的算法实现。

表 5.4　SW17 键按动一次,键状态消抖动的变化沿提取过程一些重要的中间结果

键动作	消抖前键状态 edgk	消抖计时器 ktmr	消抖后键状态 edgk	触发型键状态 edgk(沿提取后)	开关型键状态 key(沿提取后)
无键	0000 0000	0	0000 0000	0000 0000	0000 0000
SW17 按下	0000 0001	1	0000 0000	0000 0000	0000 0000
SW17 抖动	0000 0000	0	0000 0000	0000 0000	0000 0000
SW17 按稳	0000 0001	1	0000 0000	0000 0000	0000 0000
…	…	…	…	…	…
SW17 按稳	0000 0001	7	0000 0000	0000 0000	**0000 0000**
SW17 按稳	0000 0001	8	**0000 0001**	0000 0001	0000 0001
SW17 按稳	0000 0001	9	0000 0001	0000 0000	0000 0001
…	…	…	…	…	…
SW17 按稳	0000 0001	<=255	0000 0001	0000 0000	0000 0001
SW17 释放	0000 0000	0	0000 0000	0000 0000	0000 0000
SW17 抖动	0000 0001	1	0000 0000	0000 0000	0000 0000
SW17 抖动	0000 0001	2	0000 0000	0000 0000	0000 0000
SW17 抖动	0000 0000	0	0000 0000	0000 0000	0000 0000
SW17 抖动	0000 0001	1	0000 0000	0000 0000	0000 0000
SW17 释稳	0000 0000	0	0000 0000	0000 0000	0000 0000

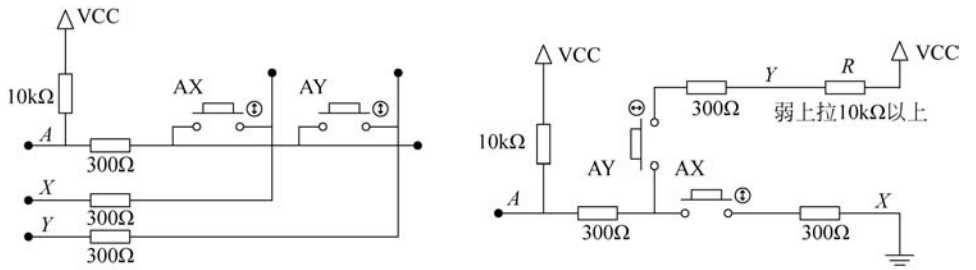
视频

5.4 矩阵键盘及其应用

当单片机应用系统的输入开关型控制量数量超过 8 个，采用矩阵键盘接法可以节省单片机的 I/O 口资源，本节讲解矩阵键盘的读键（或键的识别）方法，即键盘扫描方法。

5.4.1 矩阵键盘的扫描方法

在矩阵键盘中，通常行线必须有上拉（若行线所用 I/O 端口没有内部弱上拉，则必须外部上拉），以保证无键按下时，行线处在高电平状态；列线可以设置为准双向或开漏模式。为说明键盘扫描识别按键的原理，以图 5.13(a)所示 1 行×2 列最小矩阵键盘为例。矩阵键盘扫描是分时逐列识别各行与该列跨接的键的状态，扫描某一列时，仅当前列线输出低电平，其他列线均输出高电平。如图 5.13(a)，当扫描 X 列时，X 列线输出低电平，Y 列线输出高电平，此时键盘接口的等效电路如图 5.13(b)所示，不论按键 AY 是否按下，行线 A 为的信号电平只与按键 AX 的状态有关，键 AX 按下，行线 A 逻辑电平 0；键 AX 释放，行线 A 逻辑电平 1。若直接用行线逻辑电平表示键状态，则按键状态用负逻辑表示，即逻辑 0 表示键导通，逻辑 1 表示键断开；也可用行线逻辑电平的“非”表示键状态，则按键状态用正逻辑表示。



(a) 1行×2列最小矩阵键盘　　(b) 扫描X列等效电路

图 5.13　矩阵键盘扫描方法

上述矩阵键盘识别方法就是所谓的行列扫描法，是单片机系统读矩阵键盘最常用的方法，以图 5.8(b)所示的 4×4 矩阵键盘为例，其扫描过程如下：

(1) 图 5.8(b)所示矩阵键盘，P0 口应设置成准双向口，由于有内部弱上拉，图中的电阻 R57、R58、R59、R60 可省略，不妨以 P0.7～P0.4 为列线，P0.3～P0.0 为行线。

(2) 准备扫描 P0.7 列，给 P0 口送该列扫描字～(1 << 7)(即 0x7f)，将 P0.7 置 0，其余列置 1，行线 P0.3～P0.0 是准双向口，作为输入口，读之前也要先写 1。

(3) 从行线 P0.3～P0.0 读取与 P0.7 列线跨接的四键(KF、KE、KD、KC)的状态，若某行线逻辑电平为 0，则表示该行线与 P0.7 列线跨接的键被按下，如 KD 键被按下，则行线 P0.1 的逻辑电平为 0。

(4) 准备扫描下一列，修改列扫描字并从 P0 口输出，P0.6、P0.5、P0.4 列扫描字分别为～(1 << 6)、～(1 << 5)、～(1 << 4)。

(5) 重复步骤(3)和(4),直到矩阵键盘中所有的列被扫描完成后,退出键盘扫描。

键盘扫描程序的编写有 3 种方式:编程扫描方式、定时扫描方式、中断扫描方式,其中定时扫描方式最为常用。

5.4.2　矩阵键盘应用举例

【例 5.5】 按键显示系统硬件如图 5.14 所示,用 AT89C51 的 $P0$ 口外接 4×4 的矩阵键盘,其中 $P0.7$～$P0.4$ 为列线,$P0.0$～$P0.3$ 为行线;用反相器 74HC04 驱动 4 位共阳数码显示模块的阳极,显示位扫描由 $P2.0$～$P2.3$ 控制,$P1$ 口作为动态数码显示器的笔段驱动;$P2.4$ 经反相器驱动蜂鸣器,用于产生按键提示音。试为该按键显示系统设计软件,要求实现以下功能:

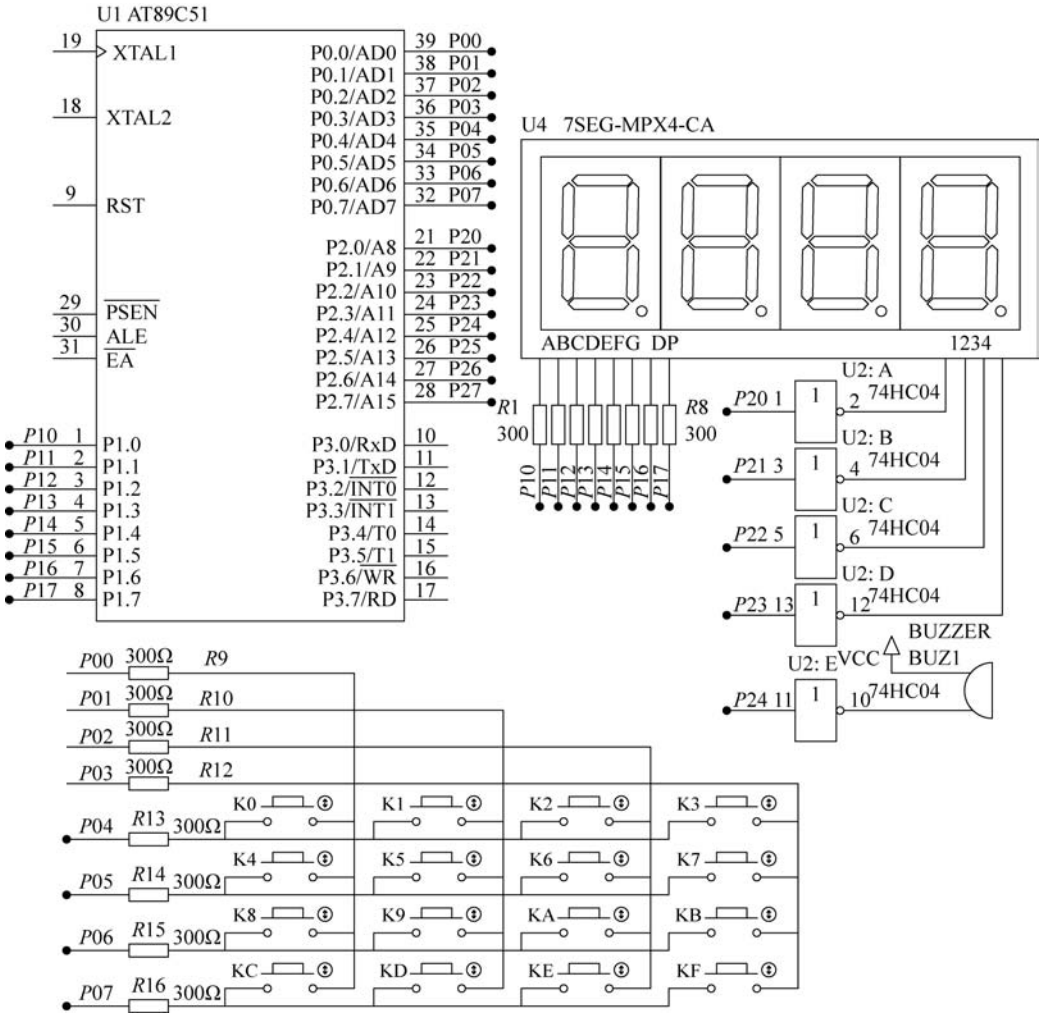


图 5.14　按键显示系统硬件(蜂鸣器选用:DC Operated Buzzer-Output Via Sound Card)

(1) 启动时显示 0123;

(2) 4×4 键盘对应十六进制数码 0～9、A～F,当按动按键时,与该键对应的数码从数码显示器的右边滚入;

(3) 按键处理程序应能与动态显示程序模块协同工作、有键消抖功能、有按键提示音、按键仅在前沿起作用(即每次按键仅在按下时发生作用)。

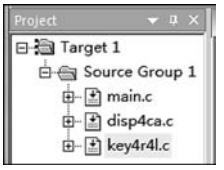


图 5.15 模块化软件结构

为便于将键盘扫描和键状态消抖动、数码动态显示等程序移植到其他系统中,本例运用模块化编程思想,将软件分为 3 个模块。其一 key4r4l.c,含与键信号处理相关的函数;其二 disp4ca.c,含数码动态显示;其三 main.c,为系统主程序。将 3 个模块添加到本例的软件工程项目中,结构如图 5.15 所示。

(1) key4r4l.c 是定义了与键信号处理相关的函数,含 4 行×4 列矩阵键盘扫描保存键状态、键状态消抖动、键状态变化前沿提取、求按键编号、发按键提示音"嘀"等,算法原理如前所述,代码如下:

```
#include <reg51.h>
#define uchar unsigned char
#define uint unsigned int
extern uint bdata key;                  //声明外部变量,键状态
extern uint bdata edgk;                 //声明外部变量,键变化前沿
uchar data kcode;                       //定义变量,键编号
uchar data ktmr;                        //定义变量,消抖计时器
uchar data beeftmr;                     //定义变量,蜂鸣计时器
sbit BEEF = P2^4;                       //定义变量,蜂鸣器控制 I/O

void readkey()                          //扫描键盘存键状态
{   uchar i,j;
    for(i = 7;i > 3;i -- )
    {   P0 = ~(1 << i);                 //扫描 P0.i 列,该列线输出 0
        for(j = 0;j < 3;j++);           //延时约 10μs 待列信号稳定,键盘引线较长应延长
        edgk << = 4;                    //空出 edgk 的低 4 位
        edgk| = (~P0)&0x0f;             //读键转正逻辑,新读 4 个键状态填补 edgk 低 4 位
    }
}

void keytrim()                          //键状态消抖动,键前沿提取,求键号
{   uint temp;                          //本行以下为: 消抖动
    if(edgk == 0)ktmr = 0;              //无键,消抖计时器清零
    else
    {   if(ktmr < 255)ktmr++;           //有键,消抖计时器 + 1(防溢出)
        if(ktmr < 8)edgk = 0;           //延时未到弃不稳定键
    }
    temp = edgk;                        //本行以下为: 键前沿提取.键状态暂存
    edgk = (key^edgk)&edgk;             //此时 key 还保存着上次循环键状态
    key = temp;                         //暂存的本次循环键状态移至 key
    if(edgk!= 0)                        //本行以下为: 求键号
```

```
    {    temp = edgk >> 1;                    //kcode 初值,temp = 待查 16 个键位
         for(kcode = 0;temp!= 0;kcode++)temp >> = 1;          //逐位查键,未查出 kcode + 1
    }
    else kcode = 0x10;                        //无键,kcode = 0x10
}

void keysound()                               //按键发出"嘀"的声响
{    if(edgk!= 0)beeftmr = 40;                //有变化沿,蜂鸣 100ms 初值
     if(beeftmr == 0)BEEF = 0;                //蜂鸣时间已到,蜂鸣关
     else {beeftmr -- ;BEEF = 1;}             //蜂鸣时间未到,走时、蜂鸣开
}
```

与例 5.4 不同,本例键盘共有 16 键,保存键状态的全局变量 edgk 和 key 的数据类型定义为 16 位的 unsigned int 型,K0 键的键状态处理在最低位,KF 键的键状态处理在最高位,由于键状态 edgk 和 key、按键编号 kcode 是主程序的重要输入开关型控制变量,因此这 3 个变量在主程序 main.c 模块中定义,本模块使用 extern 声明引用。

(2) disp4ca.c 是 4 位共阳极数码显示器扫描驱动子程序。

```
#include < reg51.h >
#define uchar unsigned char
uchar code segTab[ ] = {                      //在 CODE 区定义七段码译码表
    0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,
    0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e
};
uchar code disScan[4] = {~(1 << 0),~(1 << 1),~(1 << 2),~(1 << 3)}; //4 种位选码数据
uchar data disBuf[4];                         //定义变量,显示缓冲器
uchar data disNum;                            //定义变量,当前扫描位
void disp_d(void)                             //显示驱动函数
{    P2| = 0x0f;                              //关闭所有位
     P1 = segTab[disBuf[disNum]];             //将当前扫描位七段码送 P1 口
     P2& = disScan[disNum];                   //将当前扫描位选线信号送 P2 口
     disNum = (disNum + 1) % 4;               //调整扫描位的值,指向下一位
}
```

同理,由于显示缓冲器 disBuf[4]存放的是显示器所要显示的内容,当前显示扫描位 disNum 也是主程序中很重要的控制变量,主程序中要频繁使用该变量,因此这 2 个变量在主程序 main.c 模块中定义,本模块使用 extern 声明引用。

(3) main.c 是主程序,完成 2.5ms 定时器初始化,显示内容选择和简单的按键解释与响应,代码如下:

```
#include < reg51.h >
#define uchar unsigned char
#define uint unsigned int
extern void disp_d();                         //声明函数,显示扫描函数
extern void readkey();                        //声明函数,扫描键盘存键状态
extern void keytrim();                        //声明函数,键状态消抖等处理
extern void keysound();                       //声明函数,有键发出"嘀"的声响
```

```
extern uchar data disBuf[];              //声明外部变量,显示缓冲器
extern uchar data disNum;                //声明外部变量,当前扫描位
extern uchar data kcode;                 //声明外部变量,键编号
uint bdata key;                          //定义变量,键状态
uint bdata edgk;                         //定义变量,键状态变化前沿
sbit K0 = key^8;                         //定义变量,开关型键状态位
sbit K8 = key^0;                         //定义变量,开关型键状态位
sbit EK0 = edgk^8;                       //定义变量,触发型键状态位
sbit EK8 = edgk^0;                       //定义变量,触发型键状态位
sbit BEEF = P2^4;                        //定义变量,蜂鸣器控制 I/O
void delay2ms5()
{   uint i;
    for(i = 0;i < 427;i++);              //2.5ms 延时函数
}
void main(void)
{   BEEF = 0;                            //关闭蜂鸣器
    disBuf[0] = 0x0; disBuf[1] = 0x1;    //默认显示"0123"
    disBuf[2] = 0x2; disBuf[3] = 0x3;
    while(1)
    {   delay2ms5();
        readkey();                       //调用扫描键盘存键状态函数
        disp_d();                        //调用显示扫描函数
        keytrim();                       //调用键状态消抖等处理函数
        keysound();                      //调用有键发出"嘀"声响函数
        if(kcode < 16)
        {   disBuf[0] = disBuf[1]; disBuf[1] = disBuf[2];          //键号右边滚入
            disBuf[2] = disBuf[3]; disBuf[3] = kcode;
        }
    }
}
```

用 μVision 平台完成上述软件项目设计,经编译和调试,在 Proteus 中验证系统软件功能的正确性。

本章小结

键盘和数码管显示模块是单片机应用系统最重要的输入/输出通道,由于成本问题,绝大多数的系统都不采用专门的键盘和数码显示接口电路,用单片机的 I/O 端口直接驱动数码管动态显示,或从 I/O 端口直接读入键盘信息,然后再作软件消抖动处理。为此,本章主要介绍数码管结构、驱动电路、显示方式和常用的扩展数码显示接口、按键开关及其接口电路、键信号消抖动处理、矩阵键盘扫描方法等基本知识;着重讨论数码动态显示与键信号消抖动处理等的软件协同问题,从多任务系统软件结构出发,引入触发型键状态和开关型键状态的概念,用键状态变化沿提取算法生成前沿触发型键状态或后沿触发型键状态。

习题

5.1　数码管显示一个字符时，用一字节的数据表示其各笔段的驱动电平，从高位到低位各笔段排序如下：dp,g,f,e,d,c,b,a，则称该字节数据为段码。试分析数码管显示以下字符的段码。

（1）共阳极数码管，显示字符“4.”和“H”；

（2）共阴极数码管，显示字符“7”和“L.”。

5.2　如图 5.4 所示，2 只共阳极数码管动态显示电路，如果稳定显示“87”字符，试填写以下数据表，列出 CPU 分时从 $P1$ 口输出的段码流，和从 $P2.7 \sim P2.6$ 输出的位选码流。

时　间　流	…→时间进程→…						
扫描	…	前个扫描周期		当前扫描周期		下个扫描周期		…
时间片	…	2.5ms	2.5ms	2.5ms	2.5ms	2.5ms	2.5ms	…
段码数据(P1)	…							…
位选码 P2[7:6]	…							…

5.3　在例 5.5 中，变量 edgk、key、ktmr、kcode、beeftmr 为什么必须定义为全局变量？

5.4　在例 5.5 中，键状态消抖动和键状态变化沿（或触发型键状态）提取是 keytrim() 函数（键状态消抖等处理）的重要算法，采用后沿提取算法，为加深对该算法的理解，请以 K9 键按动一次为例填写下表，分析键状态信息处理过程的几个重要的中间结果。

键　动　作	原始键状态 Edgk 高字节	消抖计时器 ktmr	消抖后键状态 edgk 高字节	触发型键状态 edgk 高字节	开关型键状态 key 高字节
无键	0000 0000	0	0000 0000	0000 0000	0000 0000
K9 按下	0000 0010	1			
K9 抖动	0000 0000	0			
K9 按稳	0000 0010	1			
…	…	…	…	…	…
K9 按稳		7			
K9 按稳		8			
K9 按稳		9			
…	…	…	…	…	…
K9 按稳		<=255			
K9 释放	0000 0000				
K9 抖动	0000 0010(假设)				
K9 抖动	0000 0000(假设)				
K9 抖动	0000 0010(假设)				
K9 抖动	0000 0010(假设)				
K9 释放	0000 0000				

5.5　试以图 5.8(b)所示的 4 行×4 列矩阵键盘为例说明矩阵键盘的识别方法——扫描法，以及读键盘的步骤。

5.6　如题图 5.6 所示，使用以下硬件资源：

(1) 单片机 AT89C51，主频率为 12.0MHz；

(2) 两个连接到 $P3.2$ 和 $P3.3$ 端口的独立式下拉按键 SW1 和 SW2，$R11$ 和 $R12$ 属性设置为 DIGITAL；

(3) 2 位共阳极数码显示器，动态显示驱动；

(4) 用 $P2.5$ 驱动蜂鸣器，蜂鸣器的元件模型选用经声卡输出的直流运行式(ACTIVE 库，DC Operated Buzzer-Output Via Sound Card)，属性设置：运行电压+5/负载阻抗 330/频率 500Hz。

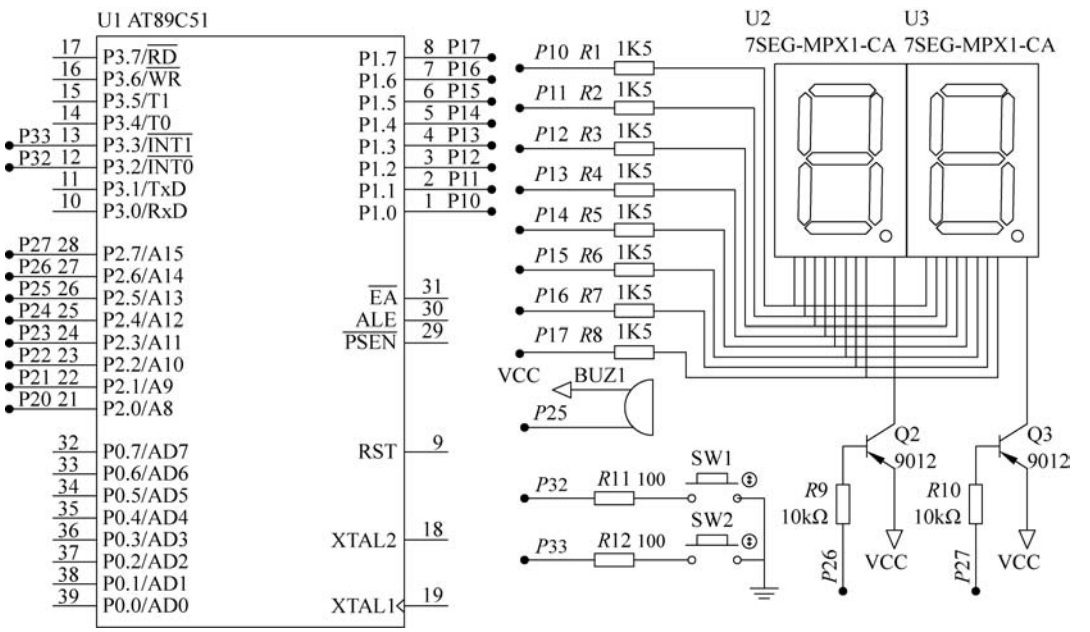
试用这些硬件资源设计一个键控计数器，要求实现以下功能：

(1) 启动时系统显示“50”；

(2) SW1 键作为“加 1”键，按该键一次，显示数值加 1，最大计数值 99；

(3) SW2 作为“减 1”键，按该键一次，显示数值减 1，最小计数值 01；

(4) 键信号需有去抖动处理、键状态变化前沿提取处理，按键每按动一次，蜂鸣器发出一声时长为 100ms 的提示音“嘀”。



题图 5.6　用 AT89C51 构成键控计数器