手把手教你学单片机的C语言程序设计(十三)

◆吕超亚

数组

基本数据类型(如字符型、整型、浮点型)的一个重要特征是只能具有单一的值。然而,许多情况下我们需要一种类型可以表示数据的集合,例如:如果使用基本类型表示整个班级学生的数学成绩,则30个学生需要30个基本类型变量。如果可以构造一种类型来表示30个学生的全部数学成绩,将会大大简化操作。

C语言中除了基本的的数据类型(例如整型、字符型、浮点型数据等属于基本数据类型)外,还提供了构造类型的数据,构造类型数据是由基本类型数据按一定规则组合而成的,因此也称为导出类型数据。C语言提供了三种构造类型:数组类型、结构体类型和共用体类型。构造类型可以更为方便地描述现实问题中各种复杂的数据结构。

数组是一组有序数据的集合,数组中的每一个数据都属于同一个数据类型。

数组类型的所有元素都属于同一种类型,并且是按顺序存放在一个连续的存储空间中,即最低的地址存放第一个元素,最高的地址存放最后的一个元素。数组类型的优点主要有两个:1.让一组同一类型的数据共用一个变量名,而不需要为每一个数据都定义一个名字;2.由于数组的构造方法采用的是顺序存储,极大方便了对数组中元素按照同一方式进行的各种操作。此外需要说明的是数组中元素的次序是由下标来确定的,下标从0开始顺序编号。

数组中的各个元素可以用数组名和下标来唯一地确定。数组可以是一维数组、二维数组或者多维数组。常用的有一维、二维数组和字符数组等。一维数组只有一个下标,多维数组有两个以上的下标。在C语言中数组必须先定义,然后才能使用。

一维数组

一维数组的定义形式如下:

数据类型 [存储器类型] 数组名[常量表达式];

其中,"数据类型"说明了数组中各个元素的类型。"数组名"是整个数组的标识符,它的定名方法与变量的定名方法一样。"常量表达式"说明了该数组的长度,即该数组中的元素个数。常量表达式必须用方括号"[]"括起来,而且其中不能含有变量。

例如定义数组 char math [30],则该数组可以用来描述30个学生的数学成绩。

二维及多维数组

定义多维数组时,只要在数组名后面增加相应于维数的常量表达式即可。

对于二维

数组的定义形式为:

数据类型 [存储器类型] 数组名[常量表达式1][常量表达式2];

例如要定义一个3行5列共3*5=15个元素的整数矩阵first,可以采用如下的定义方法:

```
int first [3][5];
```

再如我们要在点阵液晶上显示"爱我中华"四个汉字,可这样定义点阵码:

```
char code Hanzi[4][32]=
{
0x00,0x40,0x40,0x20,0xB2,0xA0,0x96,0x90,0x9A,0x4C,0x92,0x47,0xF6,0x2A,0x9A,0x2A,0x93,0x12,0x91,0x1A,0x99,0x26,0x97,0x22,0x91,0x40,0x90,0xC0,0x30,0x40,0x00,0x00,/* 爱 */
0x20,0x04,0x20,0x04,0x22,0x42,0x22,0x82,0xFE,0x7F,0x21,0x01,0x21,0x01,0x20,0x10,0x20,0x10,0xFF,0x08,0x20,0x07,0x22,0x1A,0xAC,0x21,0x20,0x40,0x20,0xF0,0x00,0x00,/* 我 */
0x00,0x00,0x00,0x00,0xFC,0x07,0x08,0x02,0x08,0x02,0x08,0x02,0x08,0x02,0xFF,0xFF,0x08,0x02,0x08,0x02,0x08,0x02,0x08,0x02,0xFC,0x07,0x08,0x00,0x00,0x00,0x00,0x00,/* 中 */
0x20,0x00,0x10,0x04,0x08,0x04,0xFC,0x05,0x03,0x04,0x02,0x04,0x10,0x04,0x10,0xFF,0x7F,0x04,0x88,0x04,0x88,0x04,0x84,0x04,0x86,0x04,0xE4,0x04,0x00,0x04,0x00,0x00/* 华 */
}
```

点阵码可使用专用的汉字或图形点阵生成软件产生。

数组的定义要注意以下几个问题:

1.数组名的命名规则同变量名的命名,要符合C语言标识符的命名规则。

2.数组名后面的"[]"是数组的标志,不能用圆括号或其他符号代替。

3.数组元素的个数必须是一个固定的值,可以是整型常量、符号常量或者整型常量表达式。

字符数组

基本类型为字符类型的数组称为字符数组。字符数组是用来存放字符的。字符数组是C语言中常用的一种数组。字符数组中的每个元素都是一个字符,因此可用字符数组来存放不同长度的字符串。字符数组的定义方法与一般数组相同,下面是定义字符数组的例子:

```
char second [6]={'H','E','L','L','O','\0'};
char third[6]={"HELLO"};
```

在C语言中字符串是作为字符数组来处理的。一个一维的字符数组可以存放一个字符串,这个字符串的长度应小于或等于字符数组的长度。为了测定字符串的实际长度,C语言规定以'\0',作为字符串结束标志,对字符串常量也自动加一个'\0'作为结束符。因此字符数组char second[6]或char third[6]可存储一个长度≤5的不同长度的字符串。在访问字符数组时,遇到'\0'就表示字符串结束,因此在定义字符数组时,应使数组长度大于它允许存放的最大字符串的长度。

对于字符数组的访问可以通过数组中的元素逐个进行访问，也可以对整个数组进行访问。

数组元素赋初值

数组的定义方法，可以在存储器空间中开辟一个相应于数组元素个数的存储空间，数组的赋值除了可以通过输入或者赋值语句为单个数组元素赋值来实现，还可以在定义的同时给出元素的值，即数组的初始化。如果希望在定义数组的同时给数组中各个元素赋以初值，可以采用如下方法：

数据类型 [存储器类型] 数组名[常量表达式]={常量表达式表};

其中，“数据类型”指出数组元素的数据类型。“存储器类型”是可选项，它指出定义的数组所在存储器空间。“常量表达式表”中给出各个数组元素的初值。

例如：

```
uchar code SEG7 [10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
```

有关数组初始化的说明如下：

1.元素值表列，可以是数组所有元素的初值，也可以是前面部分元素的初值。

如：

```
int a[5]={1,2,3};
```

数组 a 的前 3 个元素 a[0]、a[1]、a[2]分别等于 1、2、3，后 2 个元素未说明。但是系统约定：当数组为整型时，数组在进行初始化时未明晚设定初值的元素，其值自动被设置为 0。所以 a[3]、a[4]的值为 0。

2.当对全部数组元素赋初值时，元素个数可以省略。但“[]”不能省。例如：

```
char c[]={'a','b','c'};
```

此时系统将根据数组初始化时大括号内值的个数，决定该数组的元素个数。所以上例数组 c 的元素个数为 3。但是如果提供的初值小于数组希望的元素个数时，方括号内的元素个数不能省。

数组作为函数的参数

除了可以用变量作为函数的参数之外，还可以用数组名作为函数的参数。一个数组的数组名表示该数组的首地址。数组名作为函数的参数时，此时形式参数和实际参数都是数组名，传递的是整个数组，即形式参数数组和实际参数数组完全相同，是存放在同一空间的同一个数组。这样调用的过程中参数传递方式实际上是地址传递，将实际参数数组的首地址传递给被调函数中的形式参数数组。当形式参数数组修改时，实际参数数组也同时被修改了。

用数组名作为函数的参数，应该在主调函数和被调函数中分别进行数组定义，而不能只在一方定义数组。而且在两个函数中定义的数组类型必须一致，如果类型不一致将导致编译出错。实参数组和形参数组的长度可以一致也可以不一致，编译器对形参数组的长度不作检查，只是将实参数组的首地址传递给形参数组。如果希望形参数组能得到实参数组的全部元素，则应使两个数组的长度一致。定义形参数组时可以不指定长度，只在数组名后面跟一个空的方括号[]，但为了在被调函数中处理数组元素的需要，应另外设置一个参数来传递数组元素的个数。

下面做几个有关数组的实验。

实验一

在 LED/128*64 图形液晶试验板上，输入数字“5”、“6”、“7”、“8”，将它们存入数组 shuzu[4]，然后让它们在数码管上显示。

在我的文档中建立一个文件目录(cs32)，然后建立 cs32.uv2 的工程项目，最后建立源程序文件(cs32.c)。

输入下面的程序：

```
#include <REG51.H>// 序号(以下同):1
#define uchar unsigned char //2
#define uint unsigned int //3
uchar code SEG7[10]={0x3f,0x06,0x5b,
0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};//4
uchar code ACT[4]={0xef,0xdf,0xbf,0x7f}; //5
uchar data shuzu[4]; //6
/*************************************7*******/
uchar status; //8
uchar temp,f; //9
/*********************10*****************/
void delay_1ms(void) //11
{ //12
uint k; //13
for(k=0;k<121;k++); //14
} //15
/************************16*****************/
void dis_all(void) //17
{ P0=SEG7[shuzu[0]];P2=ACT[0];//18
  delay_1ms(); //19
  P0=SEG7[shuzu[1]];P2=ACT[1];//20
  delay_1ms(); //21
  P0=SEG7[shuzu[2]];P2=ACT[2];//22
  delay_1ms(); //23
  P0=SEG7[shuzu[3]];P2=ACT[3];//24
  delay_1ms(); //25
} //26
/************************27*****************/
void dis_shuzu0(void) //28
{ P0=SEG7[f];P2=ACT[0]; //29
  delay_1ms(); //30
} //31
/*********************32*****************/
void dis_shuzu1(void) //33
{ P0=SEG7[f];P2=ACT[1]; //34
  delay_1ms(); //35
} //36
/*************************37**************/
void dis_shuzu2(void) //38
{ P0=SEG7[f];P2=ACT[2]; //39
  delay_1ms(); //40
} //41
/*********************42************************/
void dis_shuzu3(void) //43
{ P0=SEG7[f];P2=ACT[3]; //44
  delay_1ms(); //45
} //46
/************47**********************/
void key_s1(void) //48
{ //49
P2=0xff; 50
 if(P2==0xfe) //51
 {if(temp>50)temp=0; //52
  if(temp==0)f++; //53
  if(f>9)f=0; //54
  temp++; //55
  delay_1ms(); //56
 } //57
} //58
/**********************59*****************/
  void key_s2(void) //60
{ //61
P2=0xff; //62
 if(P2==0xfd) //63
 { //64
  switch (status) //65
  { //66
    case 1:shuzu[0]=f;break;//67
    case 2:shuzu[1]=f;break;//68
    case 3:shuzu[2]=f;break;//69
    case 4:shuzu[3]=f;break;//70
    default:break; //71
```

```
    }                                   //72
  }                                     //73
}                                       //74
/*****************75*********************/
void key_s4(void)                       //76
{                                       //77
P2=0xff;                                //78
    if(P2==0xf7)                        //79
     {if(temp>50)temp=0;                //80
    if(temp==0)status++;                //81
    if(status>4)status=0;               //82
    temp++;                             //83
    delay_1ms();                        //84
     }                                  //85
}                                       //86
/*********************87*******************/
void main(void)                         //88
{                                       //89
    while(1)                            //90
    {                                   //91
        key_s1();                       //92
        key_s2();                       //93
        key_s4();                       //94
            switch (status)             //95
        {                               //96
    case 0:dis_all();break;             //97
    case 1:dis_shuzu0();break;          //98
    case 2:dis_shuzu1();break;          //99
    case 3:dis_shuzu2();break;          //100
    case 4:dis_shuzu3();break;          //101
    default:break;                      //102
        }                               //103
    }                                   //104
}                                       //105
```

编译通过后，将生成的 cs32.hex 文件烧录到 89S51 芯片中，将芯片插入到 LED/128*64 图形液晶试验板上，试验板上接通 5V 开关稳压电源。刚开始 4 个 LED 数码管显示“0000”。按一下 S4 键，只有个位数码管亮（显示“0”）。再按一下 S4 键，变成十位数码管亮（显示“0”）。……。这样按动 S4 键后，显示状态变成：显示全部四位→显示个位→显示十位→显示百位→显示千位，循环进行。在个位数码管点亮时，按下 S1 键（按住不放），则显示值开始递增（0-9）并循环，调整到我们需要的某个值（例如“8”）时，放开 S1 键，然后按动一下 S2 键，即将该值存入数组的 shuzu[0]。同理，在十、百、千位数码管点亮时，我们也可将“7”、“6”、“5”三个值存入数组的 shuzu [1] 、shuzu [2] 、shuzu[3]。再按动 S4 键，让四位数码管全部点亮，显示数组的内容“5678”。这样实现了对数组的读写操作。

我们对程序进行分析。

序号 1（程序解释，以下同）：包含头文件 REG51.H。

序号 2、3：数据类型的宏定义。

序号 4：定义数组 SEG7[10]，存放数码管 0~9 的字形码，由于进行只读操作，因此存储区可选定在 code 区。

序号 5：定义数组 ACT4[4]，存放四位数码管的位码，由于进行只读操作，因此存储区选定在 code 区。

序号 6：定义数组 shuzu[4]，存放输入的数据，由于要进行读写操作，因此存储区选定在 *data* 区。

序号 7：程序分隔。

序号 8：定义工作状态 status。

序号 9：全局变量 temp、f。

序号 10：程序分隔。

序号 11~15：延时 1mS 的子函数。

序号 16：程序分隔。

序号 17：定义函数名为 dis_all 的子函数。

序号 18：*dis_all* 子函数开始。取出数组的 shuzu[0]内容送 P0 口，同时点亮个位数码管。

序号 19：延时 1mS，维持数码管点亮。

序号 20：取出数组的 shuzu[1]内容送 P0 口，同时点亮十位数码管。

序号 21：延时 1mS，维持数码管点亮。

序号 22：取出数组的 shuzu[2]内容送 P0 口，同时点亮百位数码管。

序号 23：延时 1mS，维持数码管点亮。

序号 24：取出数组的 shuzu[3]内容送 P0 口，同时点亮千位数码管。

序号 25：延时 1mS，维持数码管点亮。

序号 26：dis_all 函数子结束。

序号 27：程序分隔。

序号 28：定义函数名为 dis_shuzu0 的子函数。

序号 29：dis_shuzu0 子函数开始。取出数组的 shuzu[0]内容送 P0 口，同时点亮个位数码管。

序号 30：延时 1mS，维持数码管点亮。

序号 31：dis_shuzu0 子函数子结束。

序号 32：程序分隔。

序号 33：定义函数名为 dis_shuzu1 的子函数。

序号 34：dis_shuzu1 子函数开始。取出数组的 shuzu[1]内容送 P0 口，同时点亮十位数码管。

序号 35：延时 1mS，维持数码管点亮。

序号 36：dis_shuzu0 子函数子结束。

序号 37：程序分隔。

序号 38：定义函数名为 dis_shuzu2 的子函数。

序号 39：dis_shuzu2 子函数开始。取出数组的 shuzu[2]内容送 P0 口，同时点亮百位数码管。

序号 40：延时 1mS，维持数码管点亮。

序号 41：dis_shuzu2 子函数子结束。

序号 42：程序分隔。

序号 43：定义函数名为 dis_shuzu3 的子函数。

序号 44：dis_shuzu3 子函数开始。取出数组的 shuzu[3]内容送 P0 口，同时点亮千位数码管。

序号 45：延时 1mS，维持数码管点亮。

序号 46：dis_shuzu3 子函数子结束。

序号 47：程序分隔。

序号 48：定义函数名为 key_s1 的子函数。

序号 49：key_s1 子函数开始。

序号 50：P2 口置全 1，准备读取输入值。

序号 51：进行 If(P2==0xfe)语句判别。读入 P2 口的输入值，如果为 FEH，说明 S1 键被按下。

序号 52：进入 If(P2==0xfe)语句。temp 的计数范围为 0~50。

序号 53：在 temp 为 0 时，对 f 进行递增。f 是我们要取用并写入数组的一个变量。

序号 54：f 的范围为 0~9。

序号 55：temp 递增。

序号 56：延时 1mS。

序号 57：If(P2==0xfe)语句结束。

序号 58：key_s1 子函数子结束。说明：由于按下键后，程序做完判断处理（如对 f 递增）后要延时 1mS（如序号 55），因此下一次对 f 的递增需等待 50x1mS 之后才进行，这样比较适合人的直觉，否则按下键后的 f 递增太快，眼睛无法看清。

序号 59：程序分隔。

序号 60：定义函数名为 key_s2 的子函数。

序号 61：key_s2 子函数开始。

序号 62：P2 口置全 1，准备读取输入值。

序号 63：进行 If(P2==0xfd)语句判别。读入 P2 口的输入值，如果为 FDH，说明 S2 键被按下。

序号 64：进入 If(P2==0xfd)语句。

序号 65：进行 switch (status) 开关语句判断。

序号 66：进入开关语句。

序号 67：如果 status 的值为 1，将 f 值赋 shuzu[0]，然后退出开关语句。

序号 68：如果 status 的值为 2，将 f 值赋 shuzu[1]，然后退出开关语句。

序号 69：如果 status 的值为 3，将 f 值赋 shuzu[2]，然后退出开关语句。

序号 70：如果 status 的值为 4，将 f 值赋 shuzu[3]，然后退出开关语句。

序号 71：如果 status 的值一项也不符合，则直接退出开关语句。

序号 72：开关语句结束。

序号 73:If(P2==0xfd)语句结束。
序号 74:key_s2 子函数子结束。
序号 75:程序分隔。
序号 76:定义函数名为 key_s4 的子函数。
序号 77:key_s4 子函数开始。
序号 78:P2 口置全 1,准备读取输入值。
序号 79:进行 If(P2==0xf7)语句判别。读入 P2 口的输入值,如果为 F7H,说明 S4 键被按下。
序号 80:temp 的计数范围为 0~50。
序号 81:在 temp 为 0 时,对 status 进行递增。status 是程序运行过程中的一个状态标志。
序号 82:status 的范围为 0~3。
序号 83:temp 递增。
序号 84:延时 1mS。
序号 85:If(P2==0xf7)语句结束。
序号 86:key_s4 子函数子结束。
序号 87:程序分隔。
序号 88:定义函数名为 main 的主函数。
序号 89:main 主函数开始。
序号 90:无限循环。
序号 91:无限循环语句开始。
序号 92:调用 key_s1 子函数。
序号 93:调用 key_s2 子函数。
序号 94:调用 key_s4 子函数。
序号 95:进行 switch(status) 开关语句判断。
序号 96:进入开关语句。
序号 97:如果 status 的值为 0,调用 dis_all 子函数,,然后退出开关语句。
序号 98:如果 status 的值为 1,调用 dis_shuzu0 子函数,,然后退出开关语句。
序号 99:如果 status 的值为 2,调用 dis_shuzu1 子函数,,然后退出开关语句。
序号 100:如果 status 的值为 3,调用 dis_shuzu2 子函数,,然后退出开关语句。
序号 101:如果 status 的值为 4,调用 dis_shuzu3 子函数,,然后退出开关语句。
序号 102:如果 status 的值一项也不符合,则直接退出开关语句。
序号 103:开关语句结束。
序号 104:无限循环语句结束。
序号 105:main 主函数结束。

实验二

在 LED/128*64 图形液晶试验板上,输入 10 个整数(0~999 之间),输出其中的最大数。

在我的文档中建立一个文件目录(cs33),然后建立 cs33.uv2 的工程项目,最后建立源程序文件(cs33.c)。

输入下面的程序:

```
#include <REG51.H>// 序号(以下同):1
#define uchar unsigned char          //2
#define uint unsigned int            //3
uchar code SEG7[10]={0x3f,0x06,0x5b,
0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};//4
uchar code ACT[4]={0xef,0xdf,0xbf,0x7f};//5
uint data shuzu[10];                 //6
/*****************7*****************/
uchar status;                        //8
uchar temp;                          //9
int max,f;                           //10
/*****************11****************/
void delay_1ms(void)                 //12
{                                    //13
uint k;                              //14
for(k=0;k<121;k++);                  //15
}                                    //16
/*****************17****************/
void dis_max(void)                   //18
{    P0=SEG7[max%10];P2=ACT[0];//19
        delay_1ms();                 //20
 P0=SEG7[(max/10)%10];P2=ACT[1];//21
     delay_1ms();                    //22
     P0=SEG7[max/100];P2=ACT[2];//23
     delay_1ms();                    //24
}                                    //25
/*****************26****************/
void dis_input(void)                 //27
{    P0=SEG7[f%10];P2=ACT[0];//28
     delay_1ms();                    //29
  P0=SEG7[(f/10)%10];P2=ACT[1];//30
     delay_1ms();                    //31
     P0=SEG7[f/100];P2=ACT[2];  //32
     delay_1ms();                    //33
     P0=SEG7[status];P2=ACT[3];//34
     delay_1ms();                    //35
}                                    //36
/*****************37****************/
void key_s1(void)                    //38
{                                    //39
P2=0xff;                             //40
 if(P2==0xfe)                        //41
 {if(temp>50)temp=0;                 //42
     if(temp==0)f++;                 //43
     if(f>999)f=999;                 //44
     temp++;                         //45
     delay_1ms();                    //46
 }                                   //47
}                                    //48
/*****************49****************/
void key_s2(void)                    //50
{                                    //51
P2=0xff;                             //52
 if(P2==0xfd)                        //53
 {if(temp>50)temp=0;                 //54
     if(temp==0)f--;                 //55
     if(f<0)f=0;                     //56
     temp++;                         //57
     delay_1ms();                    //58
 }                                   //59
}                                    //60
/*****************61****************/
void key_s3(void)                    //62
{                                    //63
P2=0xff;                             //64
 if(P2==0xfb)                        //65
 {                                   //66
     switch (status)                 //67
     {    case 0:shuzu[0]=f;break;   //68
          case 1:shuzu[1]=f;break;   //69
          case 2:shuzu[2]=f;break;   //70
          case 3:shuzu[3]=f;break;   //71
     case 4:shuzu[4]=f;break;        //72
     case 5:shuzu[5]=f;break;        //73
     case 6:shuzu[6]=f;break;        //74
     case 7:shuzu[7]=f;break;        //75
     case 8:shuzu[8]=f;break;        //76
     case 9:shuzu[9]=f;break;        //77
          default:break;             //78
     }                               //79
 }                                   //80
}                                    //81
/*****************82****************/
void key_s4(void)                    //83
{                                    //84
P2=0xff;                             //85
     if(P2==0xf7)                    //86
      {if(temp>50)temp=0;            //87
     if(temp==0)status++;            //88
     if(status>10)status=0;          //89
     temp++;                         //90
     delay_1ms();                    //91
      }                              //92
}                                    //93
/*****************94****************/
void conv(void)                      //95
{                                    //96
uchar i;                             //97
max=shuzu[0];                        //98
     for(i=1;i<10;i++)               //99
     {                               //100
     if(shuzu[i]>max)max=shuzu[i];//101
     }                               //102
}                                    //103
/*****************104***************/
void main(void)                      //105
{                                    //106
     while(1)                        //107
     {                               //108
          key_s1();                  //109
```

```
        key_s2();                    //110
        key_s3();                    //111
        key_s4();                    //112
        conv();                      //113
        if(status==10)dis_max();     //114
        else dis_input();            //115
    }                                //116
}                                    //117
```

编译通过后，将生成的 cs33.hex 文件烧录到 89S51 芯片中，将芯片插入到 LED/128*64 图形液晶试验板上，试验板上接通 5V 开关稳压电源。刚开始 4 个 LED 数码管显示“0000”，其中最左的数码管显示状态(状态为 0)，右边三个数码管显示数据。按动 S1 键，数据递增(最大到 999)。按动 S2 键，数据递减(最小到 000)。当我们选择某个数据(例如 159)后，按一下 S3 键，即将此数据(159)存入数组的 shuzu[0]；按一下 S4 键，最左的数码管数据递增(状态为 1)，按动 S1 或 S2 键选择数据，再按一下 S3 键，将此数据存入数组的 shuzu[1]；------同理，我们可将 shuzu[0]~ shuzu[9]存入 0-999 之间的不同数据。最后按一下 S4 键，最左的数码管熄灭，右边三个数码管显示从数组 shuzu 找出的最大数。

我们对程序进行分析。

序号 1（程序解释，以下同）：包含头文件 REG51.H。

序号 2、3：数据类型的宏定义。

序号 4：定义数组 SEG7[10]，存放数码管 0~9 的字形码，由于进行只读操作，因此存储区可选定在 code 区。

序号 5：定义数组 ACT4[4]，存放四位数码管的位码，由于进行只读操作，因此存储区可选定在 code 区。

序号 6：定义数组 shuzu[10]，存放输入的数据，由于要进行读写操作，因此存储区选定在 data 区。

序号 7：程序分隔。

序号 8：定义工作状态 status。

序号 9：全局变量 temp。

序号 10：全局变量 max、f。

序号 11：程序分隔。

序号 12~16：延时 1mS 的子函数。

序号 17：程序分隔。

序号 18：定义函数名为 dis_max 的子函数。

序号 19：dis_max 子函数开始。取出 max 的个位数送 P0 口，同时点亮个位数码管。

序号 20：延时 1mS，维持数码管点亮。

序号 21：取出 max 的十位数送 P0 口，同时点亮十位数码管。

序号 22：延时 1mS，维持数码管点亮。

序号 23：取出 max 的百位数送 P0 口，同时点亮百位数码管。

序号 24：延时 1mS，维持数码管点亮。

序号 25：dis_max 子函数子结束。

序号 26：程序分隔。

序号 27：定义函数名为 dis_input 的子函数。

序号 28：dis_input 子函数开始。取出 f 的个位数送 P0 口，同时点亮个位数码管。

序号 29：延时 1mS，维持数码管点亮。

序号 30：取出 f 的十位数送 P0 口，同时点亮十位数码管。

序号 31：延时 1mS，维持数码管点亮。

序号 32：取出 f 的百位数送 P0 口，同时点亮百位数码管。

序号 33：延时 1mS，维持数码管点亮。

序号 34：取出 f 的千位数送 P0 口，同时点亮千位数码管。

序号 35：延时 1mS，维持数码管点亮。

序号 36：dis_input 子函数子结束。

序号 37：程序分隔。

序号 38：定义函数名为 key_s1 的子函数。

序号 39：key_s1 子函数开始。

序号 40：P2 口置全 1，准备读取输入值。

序号 41：进行 If(P2==0xfe)语句判别。读入 P2 口的输入值，如果为 FEH，说明 S1 键被按下。

序号 42：进入 If(P2==0xfe)语句。temp 的计数范围为 0~50。

序号 43：在 temp 为 0 时，对 f 进行递增。f 是我们要取用并写入数组的一个变量。

序号 44：f 最大为 999。

序号 45：temp 递增。

序号 46：延时 1mS。

序号 47：If(P2==0xfe)语句结束。

序号 48：key_s1 子函数子结束。

序号 49：程序分隔。

序号 50：定义函数名为 key_s2 的子函数。

序号 51：key_s2 子函数开始。

序号 52：P2 口置全 1，准备读取输入值。

序号 53：进行 If(P2==0xfd)语句判别。读入 P2 口的输入值，如果为 FDH，说明 S2 键被按下。

序号 54：进入 If(P2==0xfd)语句。temp 的计数范围为 0~50。

序号 55：在 temp 为 0 时，对 f 进行递增。f 是我们要取用并写入数组的一个变量。

序号 56：f 最小为 0。

序号 57：temp 递增。

序号 58：延时 1mS。

序号 59：If(P2==0xfd)语句结束。

序号 60：key_s2 子函数子结束。

序号 61：程序分隔。

序号 62：定义函数名为 key_s3 的子函数。

序号 63：key_s3 子函数开始。

序号 64：P2 口置全 1，准备读取输入值。

序号 65：进行 If(P2==0xfb)语句判别。读入 P2 口的输入值，如果为 FBH，说明 S3 键被按下。

序号 66：进入 If(P2==0xfb)语句。

序号 67：进行 switch (status) 开关语句判断。

序号 68：进入开关语句。如果 status 的值为 0，将 f 值赋 shuzu[0]，然后退出开关语句。

序号 69：如果 status 的值为 1，将 f 值赋 shuzu[1]，然后退出开关语句。

序号 70：如果 status 的值为 2，将 f 值赋 shuzu[2]，然后退出开关语句。

序号 71：如果 status 的值为 3，将 f 值赋 shuzu[3]，然后退出开关语句。

序号 72：如果 status 的值为 4，将 f 值赋 shuzu[4]，然后退出开关语句。

序号 73：如果 status 的值为 5，将 f 值赋 shuzu[5]，然后退出开关语句。

序号 74：如果 status 的值为 6，将 f 值赋 shuzu[6]，然后退出开关语句。

序号 75：如果 status 的值为 7，将 f 值赋 shuzu[7]，然后退出开关语句。

序号 76：如果 status 的值为 8，将 f 值赋 shuzu[8]，然后退出开关语句。

序号 77：如果 status 的值为 9，将 f 值赋 shuzu[9]，然后退出开关语句。

序号 78：如果 status 的值一项也不符合，则直接退出开关语句。

序号 79：开关语句结束。

序号 80：If(P2==0xfb)语句结束。

序号 81：key_s3 子函数子结束。

序号 82：程序分隔。

序号 83：定义函数名为 key_s4 的子函数。

序号 84：key_s4 子函数开始。

序号 85：P2 口置全 1，准备读取输入值。

序号 86：进行 If(P2==0xf7)语句判别。读入 P2 口的输入值，如果为 F7H，说明 S4 键被按下。

序号 87：temp 的计数范围为 0~50。

序号 88：在 temp 为 0 时，对 status 进行递增。status 是程序运行过程中的一个状态标志。

序号 89：status 的范围为 0~10。

序号 90：temp 递增。

序号 91：延时 1mS。

序号 92：If(P2==0xf7)语句结束。

序号 93：key_s4 子函数子结束。

序号 94：程序分隔。

序号 95：定义函数名为 conv 的子函数。

序号 96：conv 子函数开始。

序号 97：定义局部变量 i。

序号 98：将数组的 shuzu[0]送入 max 中。

序号 99：for 循环语句。

序号 100：进入 9 次循环。

序号 101：如果 shuzu[i] 的值大于 max，则此值赋于 max。

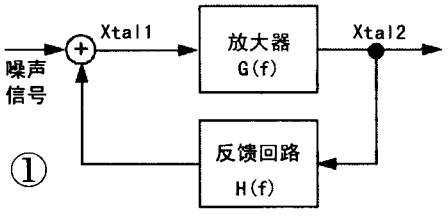
80C51 时钟振荡器的原理分析和设计考虑

◆李学海 刘治山 高笑飞

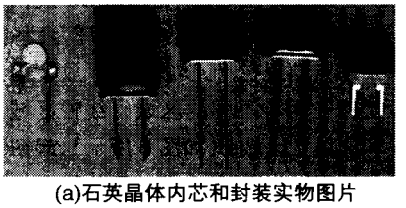
本文对于时钟振荡器的原理分析和电路设计分为两个题目分别论述。

一、时钟振荡器的原理分析

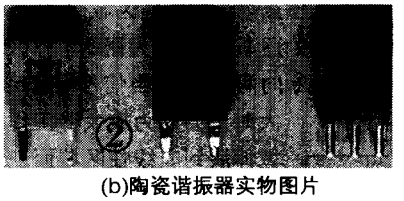
如果对于一个用作时钟源的多谐振荡器(即方波振荡器)进行理论分析，则可以利用一个如图 1 的分析模型。电路主要包含 2 个部分：放大器 G(f) 和反馈回路 H(f)。



其中，反馈回路 H(f)既可以利用一只石英晶体担当，也可以利用一只陶瓷谐振器担当，它们都具有频率选择作用的（实物图片如图 2 所示）。这里以应用更为广泛的石英晶体为例(其特征分析在后面马上有介绍)。而放大器 G(f) 既可以由一个同相放大电路担当，也可以由一个反相放大电路担当。另外，噪声作为振荡电路起振的原始激励信号。或者说，振荡器的工作还要依赖于噪声信号的帮忙。噪声信号可以是晶



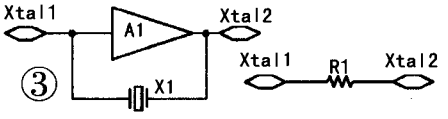
(a)石英晶体内芯和封装实物图片



(b)陶瓷谐振器实物图片

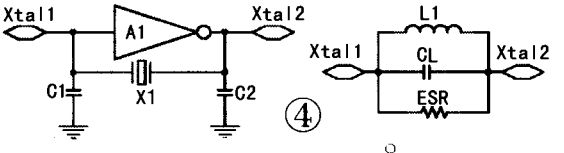
体管和电阻器件的热噪声、外来电磁辐射在输入端上形成的杂波噪声或电源刚刚接通时的瞬态噪声。

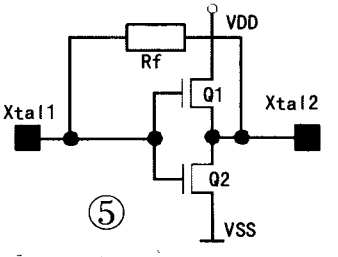
如图 3 所示，当采用一个同相放大电路 A1 担当放大器时，就构成了一个串联谐振式振荡器。这时的反馈回路表现为一只纯电阻性元件 R1。



如图 4 所示，当采用一个反相放大电路 A1 担当放大器时，就构成了一个并联谐振式振荡器。这时利用了石英晶体的电感特性，因此需要对地连接 2 只电容 C1 和 C2。这时的反馈回路表现为一个如图 4 右侧所描绘的等效结构。这种振荡器形式所需要的反相放大器，非常适合利用 80C51 内部现有的电路实现。

其实，图 4 中的反相放大器 A1(不要看作一只逻辑非门)，就是利用 80C51 芯片内部现有的 3 个器件等效而成的。如图 5 所示，构成逻辑非门的 PMOS 管 Q1 和 NMOS 管，以及作为





工作点偏置电阻的 Rf（同时也是一只直流负反馈电阻）。

作为反馈回路的石英晶体属于一种复合型器件，其内部等效电路如图 6 所示。其中，串联支路 R1、L1 和 C1 可

序号 102:for 循环语句结束。
序号 103:conv 子函数子结束。
序号 104:程序分隔。
序号 105:定义函数名为 main 的主函数。
序号 106:main 主函数开始。
序号 107:无限循环。
序号 108:无限循环语句开始。
序号 109:调用 key_s1 子函数。
序号 110:调用 key_s2 子函数。
序号 111:调用 key_s2 子函数。
序号 112:调用 key_s4 子函数。
序号 113:调用 conv 子函数。
序号 114: 如果状态标志为 10，调用 dis_max 子函数。
序号 115:否则调用 dis_input 子函数。
序号 116:无限循环语句结束。
序号 117:main 主函数结束。