第 9 章　单片机系统配置及接口

单片机主要应用于测控系统中，对于测量系统，系统总要有被测量信号的输入通道，由计算机拾取必要的输入信息。系统需要的被测信号，一般可分为开关量和模拟量两种。所谓开关量，是指输入信号为状态信号，如继电器的吸合与断开、光电门的导通与截止等，其信号电平只有两种，即高电平或低电平。对于这类信号，只需经放大、整形和电平转换等处理后，即可直接送入计算机系统。对于模拟量输入，由于模拟信号的电压或电流是连续变化信号，其信号幅度在任何时刻都有定义，因此对其进行处理就较为复杂，在进行小信号放大、滤波量化等处理过程中需考虑干扰信号的抑制、转换精度及线性等诸多因素，而这种信号又是测控系统中最普通、最常碰到的输入信号，如对温度、湿度、压力、流量、速度、亮度等信号的处理等。

在控制系统中，通常将计算机的处理结果转换为模拟信号，驱动相应的执行机构，实现对被控对象的控制，根据被控对象的不同，控制信号也可分为开关量和模拟量两种。开关量（即数字信号）通过继电器、光电耦合器、可控硅等实现用数字信号控制，即弱电控制强电；模拟量通过 D/A 转换器实现数/模转换。

单片机应用系统通常需要人的干预，人—机对话通过键盘、显示器、打印机等进行。图 9-1 所示为具有模拟量输入、模拟量输出以及键盘、显示器、打印机等配置的 89C52 应用系统框图。为节省 I/O 口线，89C52 片外扩展应尽量采用串行外设接口芯片。

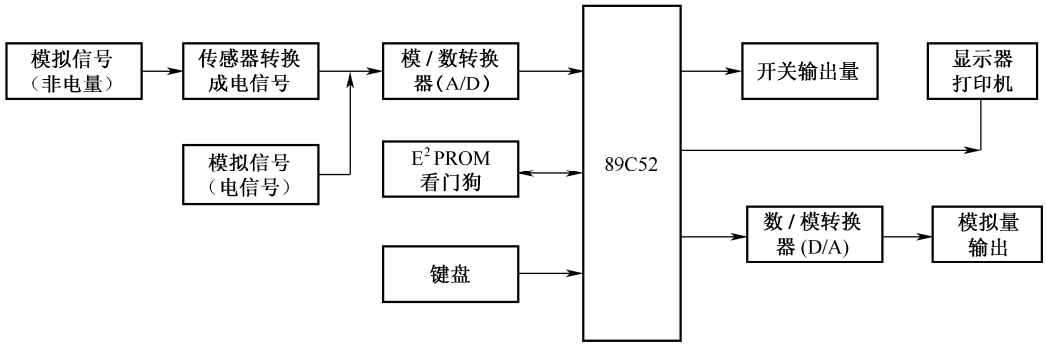


图 9-1　单片机应用系统配置框图

本章主要讲述键盘、显示器、A/D 转换器、D/A 转换器、开关器件等的工作原理及接口电路。

9.1　键盘接口

单片机应用系统通常都需要进行人—机对话。这包括人对应用系统的状态干预与数据输入等，所以应用系统大多数都设有键盘。键盘是由按键构成的。

9.1.1　键盘基本问题

键盘是一组按键的集合，它是最常用的单片机输入设备。操作人员可以通过键盘输入数

据或命令，实现简单的人—机通信。按键是一种常开型按钮开关。平时（常态时），按键的两个触点处于断开状态，按下键时它们才闭合（短路）。键盘分编码键盘和非编码键盘。键盘上闭合键的识别由专用的硬件译码器实现，并产生键编号或键值的称为编码键盘，如 BCD 码键盘、ASCII 码键盘等；靠软件识别的称为非编码键盘。

在单片机组成的测控系统及智能化仪器中，用得最多的是非编码键盘。本节讨论非编码键盘的原理、接口技术和程序设计。

键盘中每个按键都是一个常开开关电路，如图 9-2 所示。

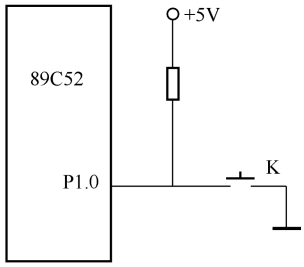


图 9-2　按键电路

当按键 K 未被按下时，P1.0 输入为高电平；当 K 闭合时，P1.0 输入为低电平。通常按键所用的开关为机械弹性开关，当机械触点断开、闭合时，电压信号波形如图 9-3 所示。由于机械触点的弹性作用，一个按键开关在闭合时不会马上稳定地接通，在断开时也不会一下子断开。因而在闭合及断开的瞬间均伴随有一连串的抖动，如图 9-3 所示。抖动时间的长短由按键的机械特性决定，一般为 5～10ms。这是一个很重要的时间参数，在很多场合都要用到。

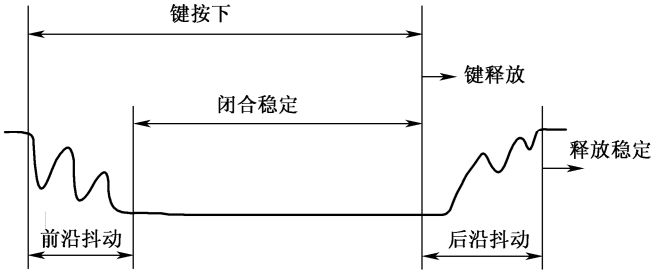


图 9-3　按键时的抖动

按键稳定闭合时间的长短则是由操作人员的按键动作决定的，一般为零点几秒。

键抖动会引起一次按键被误读多次。为了确保 CPU 对键的一次闭合仅做一次处理，必须去除键抖动。在键闭合稳定时，读取键的状态，并且必须判别；在键释放稳定后，再作处理。按键的抖动，可用硬件或软件两种方法消除。

如果按键较多，常用软件方法去抖动，即检测出键闭合后执行一个延时程序，产生 12～20ms 的延时，让前沿抖动消失后，再一次检测键的状态，如果仍保持闭合状态电平，则确认为真正有键按下。当确认有键按下或检测到按键释放后，才能转入该键的处理程序。

9.1.2 键盘结构及处理程序

键盘结构可以根据按键数目的多少分为独立式和行列式（矩阵式）两类，独立式键盘适用于按键数目较少的场合，结构和处理程序比较简单；行列式键盘适用于按键数目较多的场合，按键的常用识别方法有扫描法和反转法两种。

1. 独立式键盘

独立式按键是指各按键相互独立地接通一条输入数据线，如图 9-4 所示。这是最简单的键盘结构，该电路为查询方式电路。

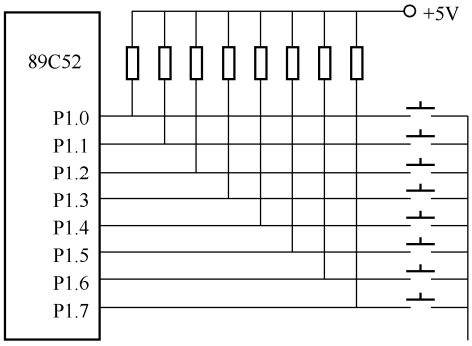


图 9-4 独立式键盘

当任何一个键按下时，与之相连的输入数据线即可读入数据 0，即低电平，而没有按下时读入 1，即高电平。要判别是否有键按下，用单片机的位处理指令十分方便。

这种键盘结构的优点是电路简单；缺点是当键数较多时，要占用较多的 I/O 线。

图 9-4 所示查询方式键盘的处理程序比较简单。实际应用当中，P1 口内有上拉电阻，图中电阻可以省去。

例 9-1 设计一个独立式按键的键盘接口，并编写键扫描程序，电路原理图如图 9-4 所示，键号从上到下分别为 0～7。

C 语言程序清单：

```
#include<reg52.h>
void key()
{
    unsigned char k;
    P1=0xff;                          //输入时 P1 口置全 1
    k=P1;                             //读取按键状态
    if(k==0xff)                       //无键按下，返回
        return;
    delay20ms();                      //有键按下，延时去抖
    k=P1;
    if(k==0xff)                       //确认键按下，抖动引起，返回
        return;
    while(P1!=0xff);                  //等待键释放
    switch(k)
    {
```

```
            case:0xfe
             ⋮                          //0 号键按下时执行程序段
            break;
            case:0xfd
             ⋮                          //1 号键按下时执行程序段
            break;
             ⋮                          //2～6 号键程序省略，读者可自行添上
            case:0x7f
             ⋮                          //7 号键按下时执行程序段
            break;
        }
}
```

汇编语言程序清单：

```
KEY:    MOV     P1,#0FFH            ;P1 口为输入口
        MOV     A,P1                ;读取按键状态
        CPL     A                   ;取正逻辑，高电平表示有键按下
        JZ      EKEY                ;A=0 表示无键按下，返回
        LCALL   DELAY20MS           ;有键按下，去抖
        MOV     A,P1
        CPL     A
        JZ      EKEY                ;抖动引起，返回
        MOV     B,A                 ;存键值
KEY1:   MOV     A,P1                ;以下等待键释放
        CPL     A
        JNZ     KEY1                ;未释放，等待
        MOV     A,B                 ;取键值送 A
        JB      ACC.0,PKEY0         ;K0 按下转 PKEY0
        JB      ACC.1,PKEY1         ;K1 按下转 PKEY1
         ⋮
        JB      ACC.7,PKEY7         ;K7 按下转 PKEY7
EKEY:   RET
PKEY1:  LCALL   K0                  ;K0 命令处理程序
        RET
PKEY2:  LCALL   K1                  ;K1 命令处理程序
        RET
         ⋮
PKEY4:  LCALL   K7                  ;K7 命令处理程序
        RET
```

由程序可以看出，各按键由软件设置了优先级，优先级顺序依次为 0～7。

2. 行列式键盘

为了减少键盘与单片机接口时所占用 I/O 线的数目，在键数较多时，通常都将键盘排列成行列矩阵形式，如图 9-5 所示。

每一水平线（行线）与垂直线（列线）的交叉处不相通，而是通过一个按键来连通。利用这种行列矩阵结构只需 N 条行线和 M 条列线，即可组成具有 N×M 个按键的键盘。

在这种行列矩阵式非编码键盘的单片机系统中，键盘处理程序首先执行有无键按下的程

序段，当确认有按键按下后，下一步就要识别哪一个按键被按下。对键的识别常用逐行扫描查询法或行列反转法。

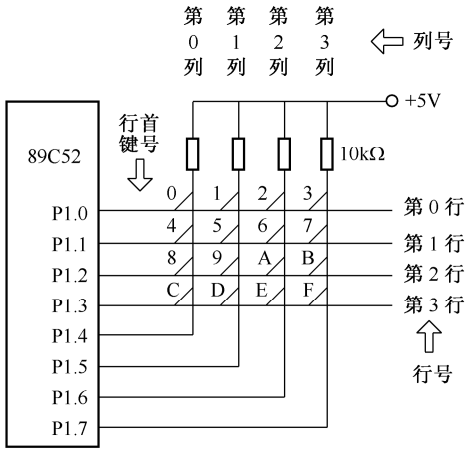


图 9-5　4×4 矩阵键盘接口

（1）行扫描法工作原理。以图 9-5 所示的 4×4 键盘为例，首先判别键盘中有无键按下，由单片机 I/O 口向键盘输出全扫描字，然后读入列线状态来判断。方法是：向行线（图中水平线）输出全扫描字 00H，把全部行线置为低电平，然后将列线的电平状态读入。如果有按键按下，总会有一根列线电平被拉至低电平，从而使列输入不全为 1。

判断键盘中哪一个键被按下是通过将行线逐行置低电平后，检查列输入状态实现的。方法是：依次给行线送低电平，然后查所有列线状态。如果全为 1，则所按下的键不在此行；如果不全为 1，则所按下的键必在此行，而且是在与零电平列线相交的交点上的那个键。

找到所按下按键的行列位置后，对按键进行编码，即求得按键键值，如图 9-5 所示。

在此指出，按键的位置码（即第几行第几列）并不等于按键的实际定义键值，因此还须进行转换。这可以借助查表或其他方法完成。这一过程称为键值译码，得到的是按键的顺序编号，然后再根据按键的编号（即 0 号键、1 号键、2 号键、……、F 号键）来执行相应的功能子程序，完成按键键帽上所定义的实际按键功能。

按键扫描的工作过程如下：

1）判断键盘中是否有键按下。

2）延时去抖。

3）进行行扫描，判断是哪一个键按下。

4）将按键的位置码转换为键值（键的顺序号）0、1、2…、F。

（2）键盘扫描识别子程序。

C 语言程序清单：

```
#include<reg52.h>
unsigned char key()                          //有键按下返回键值 0～15，无键按下返回-1
{
    unsigned char row,col=0,k =-1;           //定义变量行、列、返回值
    P1=0xf0;
    if((P1&0xf0)==0xf0)
```

```
        return k;                          //无键按下，返回
    delay20ms();                           //延时去抖
    if((P1&0xf0)==0xf0)
        return k;                          //抖动引起，返回
    for(row=0;row<4;row++)                 //行扫描
    {
        P1=~(1<<row);                      //扫描值送 P1
        k=P1&0xf0;
        if(k!=0xf0)                        //列线不全为 1，所按键在该列
        {
            while(k&(1<<(col+4)))
                col++;                     //查找为 0 列号
            k=row*4+col;                   //键值等于行号*4+列号
            P1=0xf0;
            while((P1&0xf0)!=0xf0);        //等待键释放
            break;
        }
    }
    return  k;                             //返回键值
}
```

汇编语言程序清单（返回值：在累加器 A 中，为键值）：

```
KEY:    LCALL   KS                 ;调用是否有键按下子程序
        JZ      EKEY               ;无键按下，返回
        LCALL   DELAY20MS          ;延时去抖
        LCALL   KS
        JZ      EKEY               ;抖动引起，返回
SKEY:   MOV     R0,#0              ;R0 作为行扫描计数器，开始为 0
        MOV     R1,#0              ;R1 作为列计数器，开始为 0
        MOV     R3,#0FEH           ;R3 为行扫描字暂存，低 4 位为扫描字
SKEY1:  MOV     P1,R3              ;输出行扫描字，高 4 位输入，为全 1
        MOV     A,P1               ;读列值
        MOV     R1,A               ;暂存列值
        CPL     A
        ANL     A,#0F0H
        JNZ     SKEY2              ;键在该列，转 SKEY2
        INC     R0                 ;行计数器加 1
        SETB    C
        MOV     A,R3
        RLC     A
        MOV     R3,A               ;进行下一行扫描
        CJNE    R0,#4,SKEY1        ;4 次扫描未完成，转 SKEY1 否则返回
EKEY:   MOV     A,#0FFH            ;无键按下时返回值 0FFH
        RET
SKEY2:  MOV     A,R1
        JNB     ACC.4,SKEY3
        JNB     ACC.5,SKEY4
```

```
        JNB     ACC.6,SKEY5
        JNB     ACC.7,SKEY6
SKEY3:  MOV     R2,#0                   ;存 0 列号
        SJMP    DKEY
SKEY4:  MOV     R2,#1                   ;存 1 列号
        SJMP    DKEY
SKEY5:  MOV     R2,#2                   ;存 2 列号
        SJMP    DKEY
SKEY6:  MOV     R2,#3                   ;存 3 列号
        SJMP    DKEY
DKEY:   MOV     A,R0                    ;键值译码，行号送 A
        MOV     B,#4
        MUL     AB
        ADD     A,R2                    ;行号×4+列号=键值，在 A 中
        PUSH    ACC
LK:     LCALL   KS                      ;等待键释放
        JNZ     LK
        POP     ACC
        RET
KS:     MOV     P1,#0F0H                ;是否有键按下，有返回非 0，无返回 0
        MOV     A,P1
        CPL     A
        ANL     A,#0F0H
        RET
```

（3）行列反转法工作原理。行列反转法接口电路和扫描法的接口电路一样。需要注意的是，图 9-5 所使用的 IO 口为 P1 口，内部有上拉电阻，因此列线的上拉电阻可以不用，如果使用 P0 口，则需要在行线和列线上都加上拉电阻。

行列反转法的按键识别过程如下：

1）判断是否有键按下、去抖的过程和行扫描法一样。

2）行为输出，列为输入，从行线输出全扫描字 0，读取列线值。

3）行列反转，列为输出，行为输入，将上一步读取到的列线输入值从列线输出，读取行线值，根据输出的列线值和读取到的行线值就可以确定按下键所在的位置，从而查表确定键值。

（4）行列反转法识别子程序。

仅给出 C 语言程序代码：

```
#include<reg52.h>
unsigned char key()
{
    unsigned char code keycode[]=
                            {0xee,0xde,0xbe,0x7e,
                            0xed,0xdd,0xbd,0x7d,
                            0xeb,0xdb,0xbb,0x7b,
                            0xe7,0xd7,0xb7,0x77
                            }                   //键盘表，定义 16 个按键的行列组合值
    unsigned char row,col,k=-1,i;               //定义变量行、列、返回值、循环变量
```

```
    P1=0xf0;
    if((P1&0xf0)==0xf0)
        return  k;                          //无键按下，返回-1
    delay20ms();                            //延时去抖
    if((P1&0xf0)==0xf0)
        return  k;                          //抖动引起，返回-1
    P1=0xf0;
    col=P1&0xf0;                            //行输出全 0，读取列值
    P1=col|0x0f;
    row=P1&0x0f;                            //列值输出，读取行值
    for(i=0;i<16;i++)
        if((row|col)==keycode[i])           //查找行列组合值在键盘表中位置
        {                                   //找到，该位置即为键值，否则，返回-1
            key=i;                          //对重复键，该方法处理为无键按下
            break;
        }
    P1=0xf0;
    while((P1&0xf0)!=0xf0);                 //等待键释放
    return  k;                              //返回键值
}
```

9.1.3　中断扫描方式

为了提高 CPU 的效率，可以采用中断扫描工作方式，即只有在键盘有键按下时才产生中断申请，CPU 响应中断，进入中断服务程序进行键盘扫描，并做相应处理。也可以采用定时扫描方式，即系统每隔一定时间进行键盘扫描，并做相应处理。

中断扫描工作方式的键盘接口如图 9-6 所示。该键盘直接由 89C52 P1 口的高、低字节构成 4×4 行列式键盘。键盘的行线与 P1 口的低 4 位相接，键盘的列线接到 P1 口的高 4 位。因此，P1.0～P1.3 作行输出线，P1.4～P1.7 作列输入线。扫描时，使 P1.0～P1.3 位清 0。当有键按下时，$\overline{\text{INT0}}$ 端为低电平，向 CPU 发出中断申请。若 CPU 开放外部中断，则响应中断请求，进入中断服务程序。中断服务程序除完成键识别、键功能处理外，还须有消除键抖动等功能。

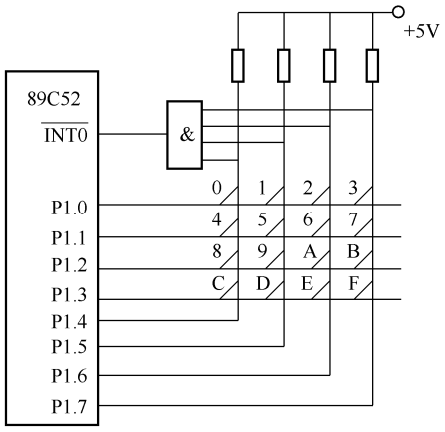


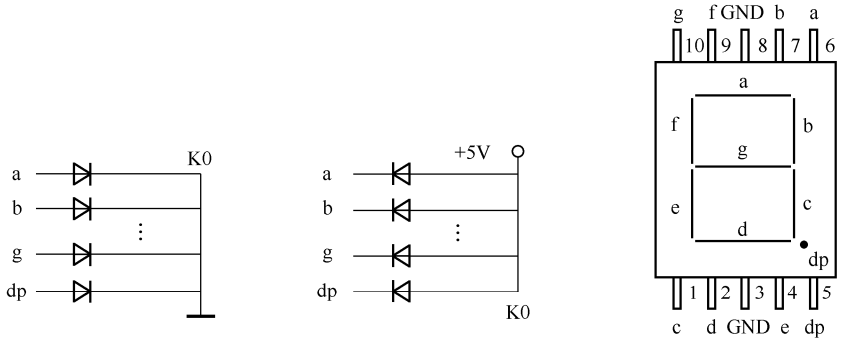
图 9-6　中断方式键盘接口

9.2　LED 显示接口

单片机应用系统中使用的显示器主要有发光二极管显示器，简称 LED（Light Emitting Diode）；液晶显示器，简称 LCD（Liquid Crystal Display）。本节主要讲述 LED 显示器的工作原理及接口。

9.2.1 LED 显示器结构原理

单片机中通常使用 7 段 LED 构成字型“8”，另外，还有一个小数点发光二极管，以显示数字、符号及小数点。这种显示器有共阴极和共阳极两种，如图 9-7 所示。发光二极管的阳极连在一起的（公共端 K0）称为共阳极显示器，阴极连在一起的（公共端 K0）称为共阴极显示器。一位显示器由 8 个发光二极管组成，其中，7 个发光二极管构成字型“8”的各个笔划（段）a～g，另一个小数点为 dp 发光二极管。当在某段发光二极管上施加一定的正向电压时，该段笔划即亮；不加电压则暗。为了保护各段 LED 不被损坏，须外加限流电阻。



（a）共阴极　　（b）共阳极　　（c）引脚配置外形图

图 9-7　LED 7 段显示器

以共阴极 LED 为例，如图 9-7(a)所示，各 LED 公共阴极 K0 接地。若向各控制端 a、b、…、g、dp 顺次送入 11100001 信号，则该显示器显示“7.”字型。

除上述 7 段“8”字型显示器以外，还有 14 段“米”字型显示器和发光二极管排成 m×n 个点矩阵的显示器。其工作原理都相同，只是需要更多的 I/O 口线控制。

共阴极与共阳极 7 段 LED 显示数字 0～F、“-”符号及“灭”的编码（a 段为最低位，dp 点为最高位）如表 9-1 所列。

表 9-1　共阴极和共阳极 7 段 LED 显示字型编码表

显示字符	0	1	2	3	4	5	6	7	8
共阴极段码	3F	06	5B	4F	66	6D	7D	07	7F
共阳极段码	C0	F9	A4	B0	99	92	82	F8	80
显示字符	9	A	B	C	D	E	F	-	熄灭
共阴极段码	6F	77	7C	39	5E	79	71	40	00
共阳极段码	90	88	83	C6	A1	86	8E	BF	FF

注：以上为 8 段，8 段最高位为小数点段。表中为小数点不点亮段码。

9.2.2 LED 显示器接口及显示方式

LED 显示器有静态显示和动态显示两种方式。

1. LED 静态显示方式

静态显示就是当显示器显示某个字符时，相应的段（发光二极管）恒定地导通或截止，直到显示另一个字符为止。例如，7 段显示器的 a、b、c 段恒定导通。其余段和小数点恒定截止时显示 7；当显示字符 8 时，显示器的 a、b、c、d、e、f、g 段恒定导通，dp 截止。

LED 显示器工作于静态显示方式时，各位的共阴极（公共端 K0）接地；若为共阳极（公共端 K0），则接+5V 电源。每位的段选线（a～dp）分别与一个 8 位锁存器的输出口相连，显示器中的各位相互独立，而且各位的显示字符一经确定，相应锁存的输出将维持不变。正因为如此，静态显示器的亮度较高。这种显示方式编程容易，管理也较简单，但占用 I/O 口线资源较多。因此，在显示位数较多的情况下，一般都采用动态显示方式。

2. LED 动态显示方式

在多位 LED 显示时，为了简化电路，降低成本，将所有位的段选线并联在一起，由一个 8 位 I/O 口控制。而共阴（或共阳）极公共端 K 分别由相应的 I/O 线控制，实现各位的分时选通。如图 9-8 所示为 6 位共阴极 LED 动态显示接口电路。

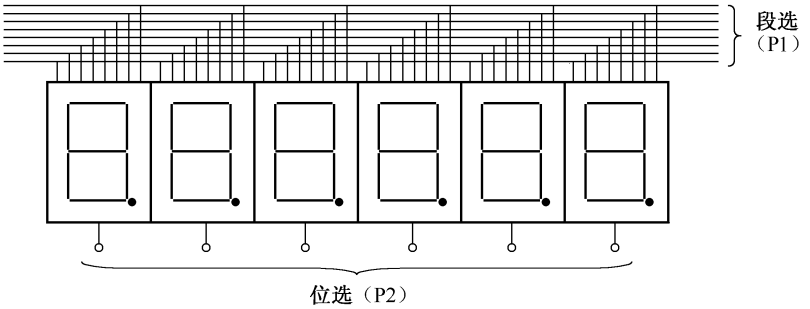


图 9-8　6 位 LED 动态显示接口电路

由于 6 位 LED 所有段选线皆由 P1 口控制，因此，在每一瞬间，6 位 LED 会显示相同的字符。要想每位显示不同的字符，就必须采用扫描方法轮流点亮各位 LED，即在每一瞬间只使某一位显示字符。在此瞬间，P1 口输出相应字符段选码（字型码），而 P2 口在该显示位送入选通电平（因为 LED 为共阴，故应送低电平），以保证该位显示相应字符。如此轮流，使每位分时显示该位应显示的字符。段选码、位选码每送入一次后延时 1ms，因人眼的视觉暂留时间为 0.1s（100 ms），所以每位显示的间隔不要超过 20ms，并保持延时一段时间，以造成视觉暂留效果，给人看上去每个数码管总在亮，这种方式称为软件扫描显示。

9.2.3　LED 显示器与 89C52 接口及显示子程序

图 9-9 所示为 89C52 P1 口和 P2 口控制的 6 位共阴极 LED 动态显示接口电路。图中，P1 口输出段选码，P2 口输出位选码，位选码占用输出口的线数决定于显示器位数，比如 6 位就要占 6 条。74LS245 是双向 8 位缓冲器，在此分别作为段选和位选驱动器。

逐位轮流点亮各个 LED，每一位保持 1ms，在 10～20ms 之内再一次点亮，重复不止。这样，利用人的视觉暂留，好像 6 位 LED 同时点亮一样。

C 语言程序清单：

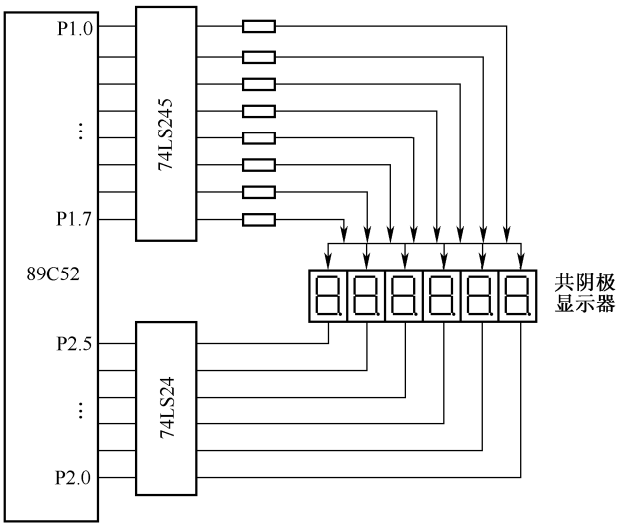


图 9-9　6 只 LED 动态显示接口

```
#include<reg52.h>
unsigned char code LED[]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,
                          0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71,0x40,0x00}
unsigned char dispbuf[6];                    //定义字型码和显示缓冲区
void disp()
{
    unsigned char i;
    for(i=0;i<6;i++)                         //6 位显示
    {
        P1=LED[dispbuf[i]];                  //段码送 P1 口
        P2=~(0x20>>i);                       //位码送 P2 口
        delay1ms();                          //延时 1ms
    }
}
```

汇编语言程序清单：

```
DISP:   MOV     R0,#DISPBUF         ;显示缓冲区首地址送 R0
        MOV     R2,#0DFH            ;位码 1101 1111B 送 R2
        MOV     R3,#6               ;6 位显示
        MOV     DPTR,#TAB           ;DPTR 指向段码表，先点亮最左边 LED
LOOP:   MOV     P2,R2               ;位码送 P2 口
        MOV     A,@R0               ;取显示数据
        MOVC    A,@A+DPTR           ;取出字型码
        MOV     P1,A                ;送出显示
        LCALL   DELAY1MS            ;延时 1ms
        INC     R0                  ;数据缓冲区地址加 1
        MOV     A,R2
        RR      A                   ;位码右移一位
        MOV     R2,A
```

```
        DJNZ    R3,LOOP              ;扫描到最左边显示位吗？
        RET
TAB:    DB      3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH
        DB      6FH,77H,7CH,39H,5EH,79H,71H,40H,00H
```

9.3　A/D 转换接口

模/数（A/D）转换电路的种类很多，例如，计数比较型、逐次逼近型、双积分型等。选择 A/D 转换器件主要是从速度、精度和价格上考虑。

逐次逼近型 A/D 转换器，在精度、速度和价格上都适中，是最常用的 A/D 转换器件。双积分 A/D 转换器，具有精度高、抗干扰性好、价格低廉等优点，但转换速度低。

近年来，串行输出的 A/D 芯片由于节省单片机的 I/O 口线，越来越多地被采用。如具有 SPI 三线接口的 TLC1549、T1C1543、TLC2543、MAX187 等，具有 2 线 I^2C 接口的 MAX127、PCF8591（4 路 8 位 A/D，还含 1 路 8 位 D/A）等，很多新型的单片机在片内集成有 A/D 转换器，从价格到使用性能都有很大的优越性，转换精度从 8 位到 16 位，如 Silicon 公司的 C8051 系列单片机、Philips 系列单片机、STC 系列单片机等。

这里，我们主要学习逐次逼近型 A/D 电路芯片，包括并行接口 A/D 转换器 ADC0809、串行接口 A/D 转换器 TLC2543、STC89LE516AD/X2 片内集成 A/D 转换器与 89C52 单片机的接口以及程序设计方法。

9.3.1　多通道串行输出 A/D 芯片 TLC2543 及接口

TLC2543 是 TI 公司生产的众多串行 A/D 转换器中的一种，它具有输入通道多、精度高、速度高、使用灵活和体积小的优点，为设计人员提供了一种高性价比的选择。

TLC2543 为 CMOS 型 12 位开关电容逐次逼近 A/D 转换器。它有 3 个控制输入端：片选（$\overline{CS}$）、输入/输出时钟（I/O CLOCK）和数据输入（DATA INPUT）端。其通过一个串行的三态输出端与主处理器或外围的串行口通信，可与主机高速传输数据，输出数据长度和格式可编程。片内含有一个 14 通道多路器，可从 11 个模拟输入或 3 个内部自测电压中选择一个。片内设有采样一保持电路。EOC 指示转换的完成。系统时钟由片内产生并由 I/O CLOCK 同步。正、负基准电压（REF_+、REF_-）由外部提供，通常为 V_{CC} 和地，两者差值决定输入电压范围。片内转换器的设计使器件具有高速（10μs 转换时间）、高精度（12 位分辨率、最大±1LSB 线性误差）和低噪声的特点。

TLC2543 与微处理器的接线很简单（用 SPI 接口只有 4 根连线），其外围电路也大大减少。TLC2543 的特性如下：

（1）12 位 A/D 转换器（可 8 位、12 位和 16 位输出）。

（2）在工作温度范围内转换时间为 l0μs。

（3）11 通道输入。

（4）3 种内建的自检模式。

（5）片内采样/保持电路。

（6）最大±l/4 096 的线性误差。

（7）内置系统时钟。

（8）转换结束标志位。

（9）单/双极性输出。

（10）输入/输出的顺序可编程（高位或低位在前）。

（11）可支持软件关机：

（12）输出数据长度可编程。

TLC1543 为 11 个输入端的 10 位 A/D 芯片，价格比 TLC2543 低。

1．TLC2543 的片内结构及引脚功能

TLC2543 封装为 20 引脚，有双列直插和方形贴片两种，引脚如图 9-10 所示，片内结构如图 9-11 所示。

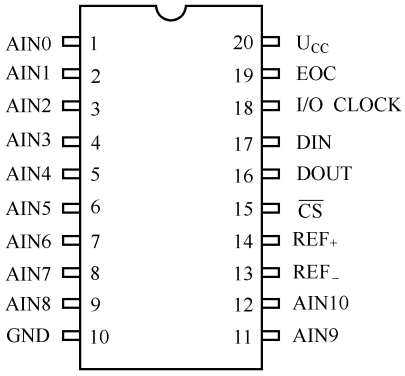


图 9-10　TLC2543 引脚排列

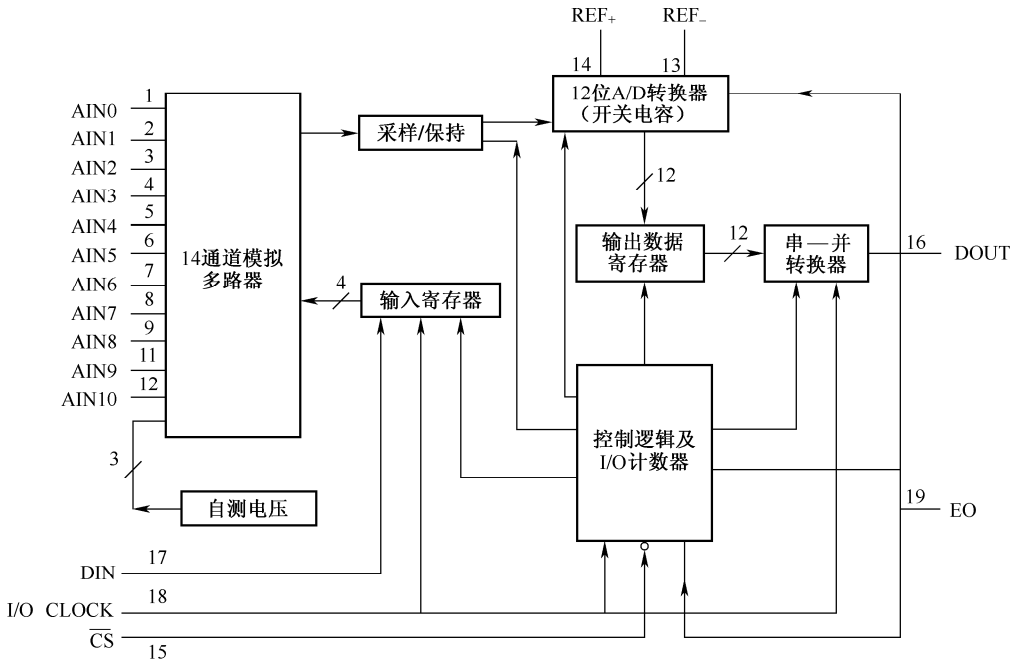


图 9-11　TLC2543 片内结构框图

TLC2543 片内由通道选择器、数据（地址和命令字）输入寄存器、采样/保持电路、12 位的模/数转换器、输出寄存器、并行到串行转换器以及控制逻辑电路 7 个部分组成。通道选择器根据输入地址寄存器中存放的模拟输入通道地址，选择输入通道，并将输入通道中的信号送到采样/保持电路中，然后在 12 位模/数转换器中将采样的模拟量进行量化编码，转换成数字量，存放到输出寄存器中。这些数据经过并行到串行转换器转换成串行数据，经 TLC2543 的 DOUT 输出到微处理器中。

TLC2543 的引脚意义如下：

（1）AIN0～AIN10 模拟输入通道。在使用 4.1MHz 的 I/O 时钟时，外部输入设备的输出阻抗应小或等于 50Ω。

（2）$\overline{CS}$片选端。一个从高到低的变化可以使系统寄存器复位，同时使能系统的输入/输出和 I/O 时钟输入。一个从低到高的变化会禁止数据输入/输出和 I/O 时钟输入。

（3）DIN 串行数据输入。最先输入的 4 位用来选择输入通道，数据是最高位在前，每一个 I/O 时钟的上升沿送入一位数据，最先 4 位数据输入到地址寄存器后，接下来的 4 位用来设置 TLC2543 的工作方式。

（4）DOUT 转换结束数据输出。有 3 种长度：8、12 和 16 位。数据输出的顺序可以在 TLC2543 的工作方式中选择。数据输出引脚在$\overline{CS}$为高时呈高阻状态，在$\overline{CS}$为低时使能。

（5）EOC 转换结束信号。在命令字的最后一个 I/O 时钟的下降沿变低，在转换结束后由低变为高。

（6）GND 地。

（7）SCLK（I/O CLOCK）输入/输出同步时钟，它有 4 种功能：

1）在它的前 8 个上升沿将命令字输入到 TLC2543 的数据输入寄存器。其中前 4 个是输入通道地址选择。

2）在第 4 个 I/O 时钟的下降沿，选中的模拟通道的模拟信号对系统中的电容阵列进行充电。直到最后一个 I/O 时钟结束。

3）I/O 时钟将上次转换结果输出。在最后一个数据输出完后，系统开始下一次转换。

4）在最后一个 I/O 时钟的下降沿，EOC 将变为低电平。

（8）REF_+正的转换参考电压，一般就用 U_{CC}。REF_- 负的转换参考电压。

（9）U_{CC} 设备的电源。

2. TLC2543 的命令字

TLC2543 的每次转换都必须给其写入命令字，以便确定下一次转换用哪个通道，下次转换结果用多少位输出，转换结果输出是低位在前还是高位在前。命令字的输入采用高位在前。命令字如下：

通道选择位	输出数据长度控制位	输出数据顺序控制位	数据极性选择位
D7D6D5D4	D3D2	D1	D0

输入到输入寄存器中的 8 位编程数据选择器件输入通道和输出数据的长度及格式。其选择格式如表 9-2 所列。

表 9-2 输入寄存器命令字格式

功能选择	输入数据字节								功能选择	输入数据字节							
	地址位				LI	L0	LSBF	BIP		地址位				L1	L0	LSBF	DIP
	D7	D6	D5	D4	D3	D2	D1	D0		D7	D6	D5	D4	D3	D2	D1	D0
选择输入通道									选择测试电压								
AIN0	0	0	0	0					（U_{ref+}-U_{ref-}）/2	1	0	1	1				
AIN1	0	0	0	1					U_{ref-}	1	1	0	0				
AIN2	0	0	1	0					U_{ref+}	1	1	0	1				
AIN3	0	0	1	1					软件断电	1	1	1	0				
AIN4	0	1	0	0					输出数据长度								
AIN5	0	1	0	1					8 位					0	1		
AIN6	0	1	1	0					12 位					X	0		
AIN7	0	1	1	1					16 位					1	1		
AIN8	1	0	0	0					输出数据格式								
AIN9	1	0	0	1					MSB 前导							0	
AIN10	1	0	1	0					LSB 前导							1	
									单极性（二进制）								0
									双极性（2 的补码）								1

注：X 表示无关项。

输入（控制字）前 4 位（D7～D4）从 11 个模拟输入选择一个进行转换，或从 3 个内部自测电压中选择一个，以对转换器进行校准，或者选择软件掉电方式。输入数据的 D3 D2 位选择输出数据长度。转换器的分辨率为 12 位，内部转换结果也总是 12 位长，选择 12 位数据长度时，所有的位都被输出。选择 8 位数据长度时，低 4 位被截去，转换精度降低，用以实现与 8 位串行接口快速通信。选择 16 位时，在转换结果的低位端增加了 4 个被置为 0 的填充位，可方便地与 16 位串行接口通信。输入数据的 D1（LSBF）位选择输出数据的传送方式，即下一个 I/O 周期数据以 LSB 前导或 MSB 前导输出。输入数据的 D0（BIP）选择转换结果以单极性或双极性二进制数码表示。

3. TLC2543 与 89C52 的 SPI 接口及程序

TLC2543 串行 A/D 转换器与 89C52 的 SPI 接口电路如图 9-12 所示。

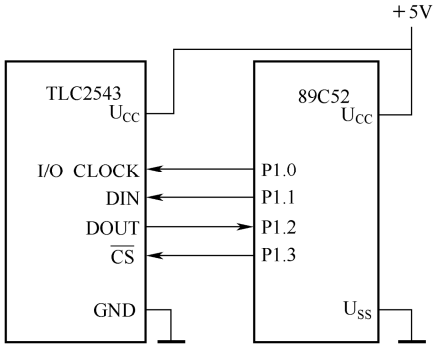


图 9-12 TLC2543 和 89C52 的接口电路

SPI（Serial Peripheral Interface）是一种串行外设接口标准，串行通信的双方用 4 根线进行通信。这 4 根连线分别是：片选信号、I/O 时钟、串行输入和串行输出。这种接口的特点是快速、高效，并且操作起来比 I^2C 要简单一些，接线也比较简单。TLC2543 提供 SPI 接口。

对不带 SPI 或相同接口能力的 89C52，须用软件合成 SPI 操作来和 TLC2543 接口。TLC2543 的 I/O CLOCK、DIN 和两端由单片机的 P1.0、P1.1 和 P1.3 提供。TLC2543 转换结果的输出（DOUT）数据由 P1.2 接收。89C52 将用户的命令字通过 P1.1 输入到 TLC2543 的输入寄存器中，等待 20μs 开始读数据，同时写入下一次的命令字。

（1）TLC2543 与 89C52 的 8 位数据传送程序。TLC2543 与 89C52 的 SPI 串行接口电路如图 9-12 所示。TLC2543 与 89C52 进行 1 次 8 位数据传送，选用 AIN0（即采集 1 次），高位在前。

C 语言程序清单：

```
#include<reg52.h>
#include<intrins.h>
sbit CS=P1^3;
sbit CLK=P1^0;
sbit DIN=P1^1;
sbit DOUT=P1^2;
unsigned char TLC2543(unsigned char command)
{                                          //定义函数，输入参数为命令字，输出转换结果
    unsigned char i,result=0;
    CS=0;                                  //片选有效
    for(i=0;i<8;i++)
    {
        DOUT=1;                            //P1.2 为输入口
        DIN=command&(0x80>>i);             //将命令字按位送出
        result<<=1;
        result|=DOUT;                      //按位接收转换结果
        CLK=1;                             //产生一个时钟
        _nop_();                           //高电平有一定宽度
        CLK=0;
    }
    return  result;                        //返回转换结果
}
```

汇编语言程序清单：

```
TLC2543:  MOV     R4,#04H          ;置控制字，AIN0，8 位数据高位在前
          MOV     A,R4
          CLR     P1.3             ;片选 CS 有效，选中 TLC2543
MSB:      MOV     R5,#08H          ;传送 8 位
LOOP:     MOV     P1,#04H          ;P1.2 为输入
          MOV     C,P1.2           ;将 A/D 转换的 8 位数据串行读入到 C 中
          RLC     A                ;带进位循环左移
          MOV     P1.1,C           ;将控制字（ACC 中）的一位经 DIN 送入
          SETB    P1.0             ;产生一个时钟
          NOP
```

```
        CLR     P1.0
        DJNZ    R5,LOOP
        MOV     R2,A            ;将 A/D 转换的数据存入 R2 中
        RET
```

执行上述子程序，经 8 次循环，执行“RLC A”指令 8 次，最后命令字 00000100 经 P1.1、DIN 进入 TLC2543 的输入寄存器，8 位 A/D 转换数据××××.××××读入累加器。

（2）TLC2543 与 89C52 的 12 位数据传送程序。用 TLC2543 的 AIN0 采集 10 个数据，放入 89C52 的 R2 中。89C52 采用 12MHz 的晶振，数据格式为 12 位，高位在前，单极性，命令字为 00H。

C 语言程序清单：

```
#include<reg52.h>
#include<intrins.h>
sbit CS=P1^3;
sbit CLK=P1^0;
sbit DIN=P1^1;
sbit DOUT=P1^2;
unsigned int TLC2543(unsigned char command)
{                                       //定义函数，输入参数为命令字，输出转换结果
    unsigned char i;
    unsigned int result=0;
    CS=0;                               //片选有效
    for(i=0;i<12;i++)
    {
        DOUT=1;                         //P1.2 为输入口
        DIN=command&(0x80>>i);          //将命令字按位送出
        result<<=1;
        result|=DOUT;                   //按位接收转换结果
        CLK=1;                          //产生一个时钟
        _nop_();                        //高电平有一定宽度
        CLK=0;
    }
    return  result;                     //返回转换结果
}
```

汇编语言程序清单：

```
        MOV     P1,#04H         ;P1.2 为输入
        MOV     R6,#0AH         ;转换 10 次
        MOV     R0,#2FH         ;置数据缓冲区指针
        CLR     P1.0            ;置 I/O 时钟为低
        SETB    P1.3            ;置 CS 为高
LOOP:   LCALL   TLC2543         ;调用转换子程序
        INC     R0              ;下一次存放单元
        DJNZ    R6,LOOP         ;不够 10，继续
        SJMP    $
TLC2543: MOV    A,#00H          ;设置通道选择和工作方式（IN0，12 位）
        CLR     P1.3            ;置 CS 为低
```

```
        MOV     R5,#0CH          ;置输出计数器初值
LOOP1:  MOV     C,P1.2           ;读入转换数据的一位到 C
        RLC     A                ;将进位位移给 A，同时将控制字的一位给 C
        MOV     P1.1,C           ;送出一位控制字
        SETB    P1.0             ;产生一个时钟
        NOP
        CLR     P1.0             ;
        CJNE    R5,#04H,LOOP2    ;剩 4 位了吗?
        MOV     @R0,A            ;存前 8 位
        INC     R0               ;
        CLR     A
LOOP2:  DJNZ    R5,LOOP1         ;未完，继续读剩余 4 位
        ANL     A,#0FH
        MOV     @R0,A            ;转换完存入单元
        RET
```

（3）TLC2543 与 89C52 的 16 位数据传送程序。TLC2543 为 12 位 A/D 转换器，可 8 位、12 位和 16 位输出。16 位输出是在 12 位的低 4 位填 4 个 0，为满足 16 位接口，其接口软件由一个主程序和 2 个子程序组成。主程序初始化 P1 口。子程序“TLC2543”包含合成 SPI 的操作以及 TLC2543 和单片机间交换数据的指令，检测命令字中 D1 位，以决定先传送转换结果的高字节还是低字节（选择 16 位数据长度方式）。SPI 功能的合成用累加器和带进位的左循环移位指令（RLC）模拟 SPI 的操作来实现。详细地说：读入转换结果的第一个字节的第一位到进位（C）位。累加器内容通过带进位左移，转换结果第一位移入 A 的最低位中，同时输入数据的第一位通过 P1.1 传输给 TLC2543。然后由 P1.0 位先高后低地翻转来提供第一个 I/O CLOCK 脉冲。这个时序再重复 7 次，完成转换数据的第一个字节的传送。TLC2543 和 89C52 之间的第二个字节的传送与第一个字节完全相同。高字节 MSByte 放在寄存器 R2，低字节 LSByte 放在寄存器 R3。子程序“STORE”用于映射对应于所选择的特定通道的 MSByte 和 LSByte 到偶或奇数的 RAM 地址。

C 语言程序清单：

```
#include<reg52.h>
#include<intrins.h>
sbit CS=P1^3;
sbit CLK=P1^0;
sbit DIN=P1^1;
sbit DOUT=P1^2;
unsigned int TLC2543(unsigned char command)
{                                       //定义函数，输入参数为命令字，输出转换结果
    unsigned char i;
    unsigned int result=0;
    CS=0;                               //片选有效
    for(i=0;i<16;i++)                   //16 位数据传送
    {
        DOUT=1;                         //P1.2 为输入口
        DIN=command&(0x80>>i);          //将命令字按位送出
        result<<=1;
```

```
            result|=DOUT;                   //按位接收转换结果
            CLK=1;                          //产生一个时钟
            _nop_();                        //高电平有一定宽度
            CLK=0;
        }
        if(command&0x02)                    //若输入数据 D1 为 1，高低 8 位交换
            return ((result<<8)|(result>>8));
        else
            return  result;                 //返回转换结果
}
```

汇编语言程序清单：

```
START:  MOV     P1,#04H         ;初始化 P1 口
        CLR     P1.0            ;置 I/O CLOCK 为低
        SETB    P1.3            ;置 CS 为高
        LCALL   TLC2543
        LCALL   STORE
        SJMP    START
TLC2543: MOV    R4,#0CH         ;读输入数据命令字到 R4、AIN0，16 位，高位在前
        MOV     A,R4            ;
DW0:    CLR     P1.3            ;置 CS 为低
        JB      ACC.1,LSB       ;若输入数据 D1 为 1，首先进行低字节传送
MSB:    MOV     R5,#8           ;以下传送高字节数据
LOOP1:  MOV     C,P1.2          ;读转换数据
        RLC     A
        MOV     P1.1,C          ;写命令字
        SETB    P1.0            ;产生一个时钟
        NOP
        CLR     P1.0
        DJNZ    R5,LOOP1        ;判断 8 位数据送完？未完，下一位
        MOV     R2,A            ;转换结果高字节放入 R2
        MOV     A,R4            ;读输入数据到 A
        JB      ACC.1,RETURN    ;若输入数据 D1 为 1，结束
LSB:    MOV     R5,#8           ;以下传送低字节数据
LOOP2:  MOV     C,P1.2
        RLC     A
        MOV     P1.1,C
        SETB    P1.0
        NOP
        CLR     P1.0
        DJNZ    R5,LOOP2
        MOV     R3,A            ;转换结果低字节放入 R3
        MOV     A,R4
        JB      ACC.1,MSB       ;若输入数据 D1 为 1，进行高字节数据传送
RETURN: RET
STORE:  MOV     A,R4            ;读输入数据到 A
        ANL     A,#0F0H         ;只保留地址位
```

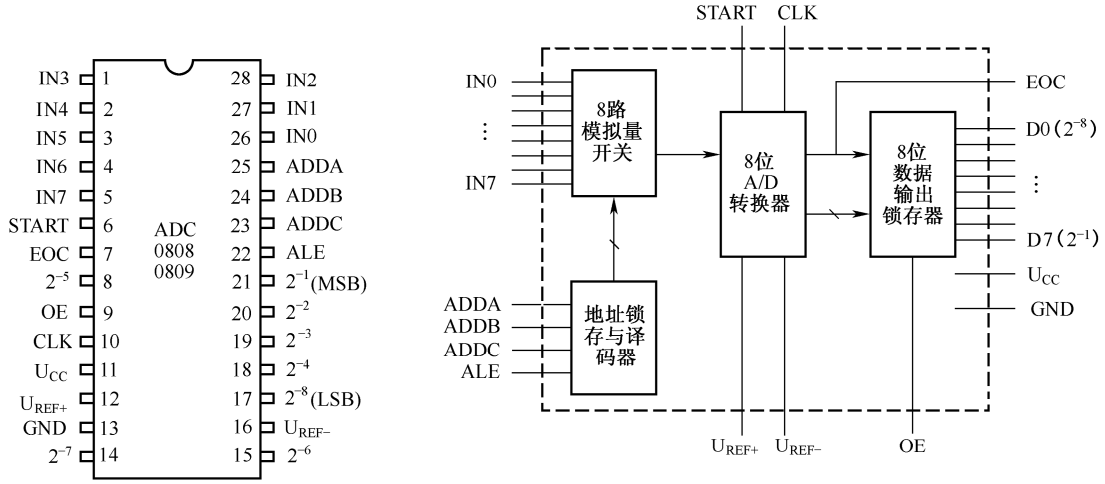
```
SWAP    A                    ;以下产生存储地址
RL      A
ADD     A,#30H
MOV     R1,A
MOV     A,R2
MOV     @R1,A
;把高字节放入相应的偶数地址 RAM：各通道地址依次为 30H、32H……
INC     R1
MOV     A,R3
MOV     @R1,A
;把低字节放入相应的奇数地址 RAM：各通道地址依次为 31H、33H……
RET
```

9.3.2　逐次逼近型并行输出 A/D 转换器及接口

1. ADC0809 的片内结构及引脚功能

ADC0809 是 CMOS 工艺，采用逐次逼近法的 8 位 A/D 转换芯片，28 引脚双列直插式封装，片内除 A/D 转换部分外还有多路模拟开关部分。

多路开关有 8 路模拟量输入端，最多允许 8 路模拟量分时输入，共用一个 A/D 转换器进行转换。图 9-13 所示为 ADC0809 的引脚图及内部逻辑结构图。它由 8 路模拟开关、8 位 A/D 转换器、三态输出锁存器以及地址锁存译码器等组成。



（a）ADC0809引脚　　（b）ADC0809内部逻辑结构

图 9-13　ADC0809 结构

引脚功能说明如下：

（1）IN0～IN7：8 个输入通道的模拟输入端。

（2）D0（2^{-8}）～D7（2^{-1}）：8 位数字量输出端。

（3）START：启动信号，加上正脉冲后，A/D 转换开始进行。

（4）ALE：地址锁存信号。由低至高电平时，把三位地址信号送入通道号地址锁存器，并经译码器得到地址输出，以选择相应的模拟输入通道。

（5）EOC：转换结束信号，是芯片的输出信号。转换开始后，EOC 信号变低；转换结束时，EOC 返回高电平。这个信号可以作为 A/D 转换器的状态信号来查询，也可以直接用作中断请求信号。

（6）OE：输出允许控制端（开数字量输出三态门）。

（7）CLOCK：时钟信号。最高允许值为 640kHz。

（8）U_{REF+}和 U_{REF-}：A/D 转换器的参考电压。

（9）U_{CC} 电源电压。由于是 CMOS 芯片，允许的电压范围较宽，可以是+5～+15V。

8 位模拟开关地址输入通道的关系见表 9-3。模拟开关的作用和 8 选 1 的 CD4051 作用相同。

表 9-3　8 位模拟开关功能表

ADDC	ADDB	ADDA	输入通道号
0	0	0	IN0
0	0	1	INl
0	1	0	IN2
⋮	⋮	⋮	⋮
1	1	1	IN7

ADC0809 芯片的转换速度在最高时钟频率下为 100μs 左右。在与 CPU 接口时，要求采用查询方式或中断方式。

2. ADC0809 与 89C52 接口

ADC0809 与 89C52 连接可采用查询方式，也可采用中断方式。图 9-14 为中断方式连接电路图。由于 ADC0809 片内有三态输出锁存器，因此可直接与 89C52 接口。

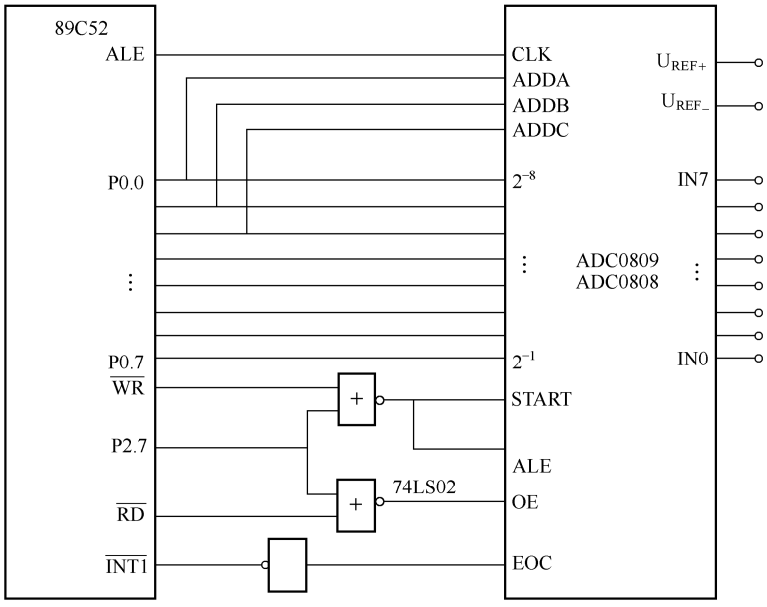


图 9-14　ADC0809 与 89C52 的连接

这里将 ADC0809 作为一个外部扩展并行 I/O 口，采用线选法寻址。由 P2.7 和 $\overline{WR}$ 联合控

制启动转换信号端（START）和 ALE 端，ADC0809 的 ADDA、ADDB 和 ADDC 端由 P0.0、P0.1、P0.2 送出，所以，ADC0809 的通道选择是由数据线控制的，ADC0809 的地址由 P2.7 一位控制，为 7FFFH。

启动 ADC0809 的工作过程是：先送数据到 ADC0809 地址，由 ALE 信号锁存通道号地址后，让 START 有效，同时启动 A/D 转换，即执行一条“MOVX　@DPTR,A”指令产生 $\overline{WR}$ 信号，A 中数据即为 ADC0809 通道号，使 ALE 和 START 有效；锁存通道号并启动 A/D 转换。A/D 转换完毕，EOC 端发出一正脉冲，申请中断。在中断服务程序中，“MOVX A,@DPTR”指令产生 $\overline{RD}$ 信号，使 OE 端有效，打开输出锁存器三态门，8 位数据便读入到 CPU 中。

ADC0809 的时钟取自 89C52 的 ALE 经二分频（也可用 74LS74 双 D 触发器之一）后的信号（接 CLK 端）。当 A/D 转换完毕，89C52 读取转换后的数字量时，须使用“MOVX A,@DPTR”指令。在图 9-14 所示的接口电路中，ADC0809 与片外 RAM 统一编址。

3．8 路巡回检测系统

例 9-2　某粮库或某冷冻厂需对 8 点（8 个冷冻室或 8 个粮仓）进行温度巡回检测。要求设计一个单片机巡回检测系统，使其能对各冷冻室或各粮仓的温度巡回检测并加以处理。

设被测温度范围为−30～+50℃，温度检测精度要求不大于±1℃。

温度传感器可选用热电阻、热敏电阻、PN 结或集成温度传感器 AD590 和 SL134 等芯片。将读数依次存放在片外数据存储器 A0H～A7H 单元。其主程序和中断服务程序如下：

C 语言程序清单：

```
#include<reg52.h>
#include<absacc.h>                          //绝对地址定位
define DAC0809 XBYTE[0x7fff]                //定义 DAC0809 地址为 7FFFH
unsigned char xdata buffer[8] _at_ 0xa0;    //数据存放定义
unsigned char i=0;
void main()
{
    IT1=1;                                  //边沿触发
    EA=1;
    EX1=1;
    DAC0809=i;                              //启动 0 通道转换
    while(1);
}
void int1_srv() interrupt 2
{
    buffer[i]=DAC0809;                      //读数存放
    if(++i!=8)                              //最后一个通道没结束
        DAC0809=i;                          //启动下一个通道转换
}
```

汇编语言主程序：

```
MAIN:  MOV     R0,#0A0H          ;数据暂存区首地址
       MOV     R2,#8             ;8 路计数初值
       MOV     R3,#0             ;R3 存放通道号
       SETB    IT1               ;边沿触发
       SETB    EA                ;开中断
```

```
        SETB    EX1
        MOV     DPTR,#7FFFH          ;指向 0809
        MOV     A,R3                 ;
        MOVX    @DPTR,A              ;送通道号，启动转换
        SJMP    $
中断服务程序：
        MOVX    A,@DPTR              ;读数
        MOVX    @R0,A                ;存数
        INC     R0                   ;更新存放单元
        INC     R3                   ;更新通道
        DJNZ    R2,DONE
        RETI
DONE:   MOV     A,R3
        MOVX    @DPTR,A
        RETI
```

9.3.3 单片机内部集成的 A/D 转换器

目前，市场上很多单片机在片内集成有 A/D 转换器，使用起来非常方便，下面以 STC89LE516AD/X2 为例，说明 A/D 转换的使用方法。

1. STC89LE516AD/X2 片内 A/D 转换器特点

STC89LE516AD/X2 的模拟量输入在 P1 口，有 8 位精度的高速 A/D 转换器，P1.0-P1.7 共 8 路，为电压输入型，可做按键扫描、电池电压检测、频谱检测等，17 个机器周期可完成一次转换，时钟在 40MHz 以下。

2. 特殊功能寄存器

（1）P1_ADC_EN 特殊功能寄存器。P1. x 作为 A/D 转换输入通道允许特殊功能寄存器，地址为 97H，复位值为 00000000B。格式如图 9-15 所示。

P1_ADC_EN

（97H）								
	ADC_P17	ADC_P16	ADC_P15	ADC_P14	ADC_P13	ADC_P12	ADC_P11	ADC_P10

图 9-15 P1_ADC_EN 特殊功能寄存器

相应位为“1”时，对应的 P1. x 口为 A/D 转换使用，内部上拉电阻自动断开。

（2）ADC_DATA 特殊功能寄存器。A/D 转换结果特殊功能寄存器，地址为 0C6H，复位值为 00000000B，模拟/数字转换结果计算公式如下：结果=256 * Uin / U_{CC}，Uin 为模拟输入通道输入电压，U_{CC} 为单片机实际工作电压，用单片机工作电压作为模拟参考电压。

（3）ADC_CONTR 特殊功能寄存器。A/D 转换控制特殊功能寄存器，地址为 0C5H，复位值为 xxx00000B。格式如图 9-16 所示。

ADC_CONTR

（C5H）	—	—	—	ADC_FLAG	ADC_START	CHS2	CHS1	CHS0

图 9-16 ADC_CONTR 特殊功能寄存器

相关位说明如下：

1）ADC_FLAG：模拟/数字转换结束标志位，当 A/D 转换完成后，ADC_FLAG=1。

2）ADC_START：模拟/数字转换（ADC）启动控制位，设置为“1”时，开始转换。

3）CHS2/CHS1/CHS0：模拟输入通道选择，如表 9-4 所示。

表 9-4　模拟输入通道选择

CHS2	CHS1	CHS0	模拟输入通道选择
0	0	0	选择 P1. 0 作为 A/D 输入来用
0	0	1	选择 P1. 1 作为 A/D 输入来用
0	1	0	选择 P1. 2 作为 A/D 输入来用
0	1	1	选择 P1. 3 作为 A/D 输入来用
1	0	0	选择 P1. 4 作为 A/D 输入来用
1	0	1	选择 P1. 5 作为 A/D 输入来用
1	1	0	选择 P1. 6 作为 A/D 输入来用
1	1	1	选择 P1. 7 作为 A/D 输入来用

3．A/D 转换程序

用 P1.0 为模拟量输入端进行 A/D 转换，程序如下：

```
#include <reg52.h>
/*定义与 ADC 有关的特殊功能寄存器*/
sfr   P1_ADC_EN=0x97;                    //A/D 转换功能允许寄存器
sfr   ADC_CONTR=0xC5;                    //A/D 转换控制寄存器
sfr   ADC_DATA=0xC6;                     //A/D 转换结果寄存器
/*延时函数                               */
void  delay(unsigned   char  delay_time)
{
      unsigned   char  i;
      unsigned   int   j;
      for(i=0;i<delay_time;i++)
            for(j=0;j<10000;j++);
}
/*AD 转换函数                            */
unsigned   char  ADC()
{
      delay(1);                          //使输入电压达到稳定
      ADC_CONTR=0x08;                    //P1.0 为模拟量输入端，启动 A/D 转换
      while((ADC_CONTR&0x10)==0);        //等待转换结束
      return   ADC_DATA;                 //返回转换结果
}
```

9.4 D/A 转换接口

9.4.1 8 位并行 D/A 转换器 DAC0832 接口技术

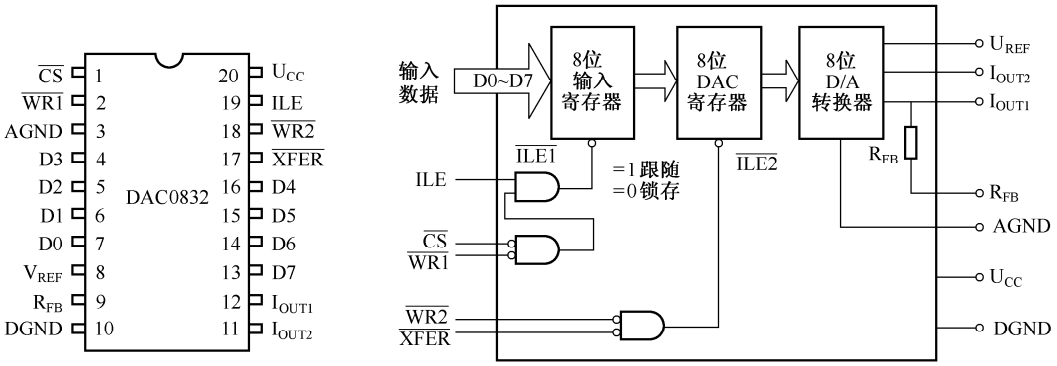
1. DAC0832 的结构原理

（1）DAC0832 的特性。美国国家半导体公司的 DAC0832 芯片是具有两级输入数据寄存器的 8 位单片 D/A 转换器，它能直接与单片机 89C52 相连接，采用二次缓冲方式，可以在输出的同时，采集下一个数据，从而提高转换速度，能够在多个转换器同时工作时，实现多通道 D/A 的同步转换输出。

主要的特性参数如下：

- 分辨率为 8 位。
- 只需在满量程下调整其线性度。
- 可与所有的单片机或微处理器直接接口。
- 电流稳定时间为 1μs。
- 可双缓冲、单缓冲或直通数据输入。
- 功耗低，约为 200mW。
- 逻辑电平输入与 TTL 兼容。
- 单电源供电（+5V～+15V）。

（2）DAC0832 的引脚及逻辑结构。DAC0832 的引脚如图 9-17（a）所示。其逻辑结构如图 9-17（b）所示，由 8 位锁存器、8 位 DAC 寄存器和 8 位 D/A 转换器构成。



（a） DAC0832引脚　　（b）DAC0832框图

图 9-17　DAC0832 结构

DAC0832 各引脚的功能说明如下：

- D0～D7：数字量数据输入线。
- ILE：数据锁存允许信号，高电平有效。
- $\overline{CS}$：输入寄存器选择信号，低电平有效。
- $\overline{WR1}$：输入寄存器的“写”选通信号，低电平有效。

- $\overline{WR2}$：DAC 寄存器的“写”选通信号，低电平有效。
- $\overline{XFER}$：数据传送信号，低电平有效。
- U_{REF}：基准电压输入线。
- R_{FB}：反馈信号输入线，芯片内已有反馈电阻。
- I_{OUT1} 和 I_{OUT2}：电流输出线。I_{OUT1} 与 I_{OUT2} 的和为常数，DAC 寄存器的内容线性变化。一般在单极性输出时，I_{OUT2} 接地。
- U_{CC}：工作电源。
- D_{GND}：数字地。A_{GND}：模拟信号地。

D/A 转换芯片输入是数字量，输出为模拟量，模拟信号很容易受到电源和数字信号等干扰而引起波动。为提高输出的稳定性和减小误差，模拟信号部分必须采用高精度基难电源 V_{REF} 和独立的地线，一般把数字地和模拟地分开。模拟地是模拟信号及基准电源的参考地，其余信号的参考地，包括工作电源地、数据、地址、控制等数字逻辑地都是数字地。

2. DAC0832 与单片机的接口

（1）单缓冲器方式接口。若应用系统中只有一路 D/A 转换或虽然是多路转换，但并不要求同步输出时，则采用单缓冲器方式接口，如图 9-18 所示。

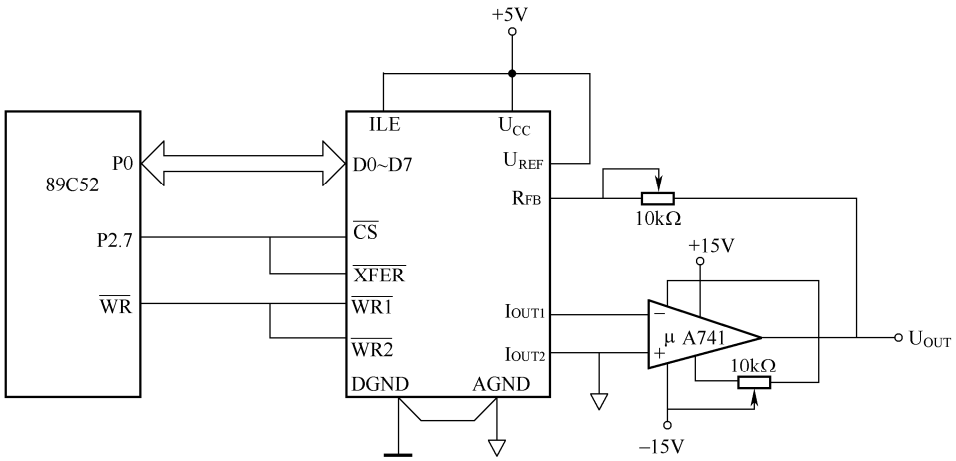


图 9-18　DAC0832 单缓冲方式接口

将 ILE 接+5V，寄存器选择信号 $\overline{CS}$ 及数据传送信号 $\overline{XFER}$ 都与地址选择线相连（图中为 P2.7），两级寄存器的写信号都由 89C52 的 $\overline{WR}$ 端控制。当地址线选通 DAC0832 后，只要输出控制信号，DAC0832 就能一步完成数字量的输入锁存和 D/A 转换输出。由于 DAC0832 具有数字量的输入锁存功能，故数字量可以直接从 89C52 的 P0 口送入。

C 语言程序清单：

```
#include<absacc.h>                          //头文件声明及端口地址定义
#define DAC0832 XBYTE[0x7FFF]

//在需要模拟量输出时
    DAC0832=data1;                          //data1 为输出模拟量的数字值
```

汇编语言程序清单：

```
MOV     DPTR,#7FFFH          ;地址只需 P2.7 为 0，其余为地址无关位，此处取 1
MOV     A,#DATA1             ;待输出模拟量的数字值送 A
MOVX    @DPTR,A
```

（2）双缓冲器同步方式接口。对于多路 D/A 转换接口，要求同步进行 D/A 转换输出时，必须采用双缓冲器同步方式接法。DAC0832 采用这种接法时，数字量的输入锁存和 D/A 转换输出是分两步完成的，即 CPU 的数据总线分时地向各路 D/A 转换器输入要转换的数字量并锁存在各自的输入寄存器中，然后 CPU 对所有的 D/A 转换器发出控制信号，使各个 D/A 转换器输入寄存器中的数据同时打入 DAC 寄存器，实现同步转换输出。

图 9-19 是一个二路同步输出的 D/A 转换接口电路。89C52 的 P2.5 和 P2.6 分别选择两路 D/A 转换器的输入寄存器，控制输入锁存；P2.7 连到两路 D/A 转换器的 $\overline{\text{XFER}}$ 端控制同步转换输出；$\overline{\text{WR}}$ 与所有的 $\overline{\text{WR1}}$、$\overline{\text{WR2}}$ 端相连，在执行 MOVX 指令时，89C52 自动输出 $\overline{\text{WR}}$ 信号。

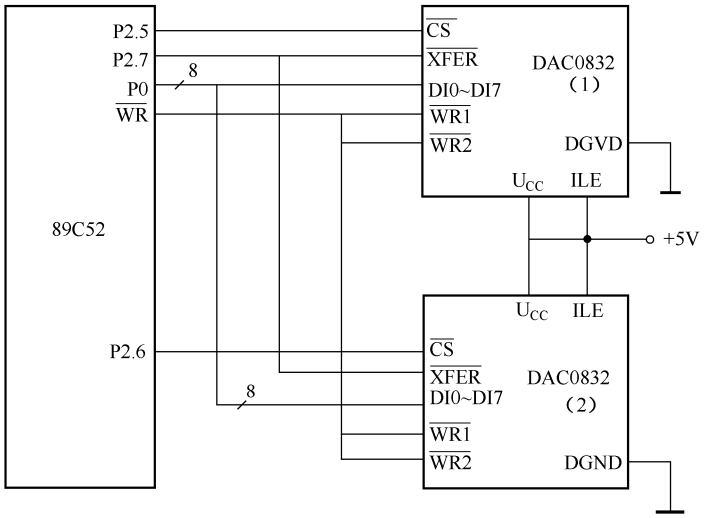


图 9-19　DAC0832 双缓冲方式接口

C 语言程序清单：

```
#include<absacc.h>
#define    DAC0832_1    XBYTE[0xDFFF]    //第一片 0832 输入寄存器端口地址
#define    DAC0832_2    XBYTE[0xBFFF]    //第二片 0832 输入寄存器端口地址
#define    DAC_ALL      XBYTE[0x7FFF]    //两路 D/A 转换器同步输出地址

//在需要模拟量同步输出时
DAC0832_1=data1;                         //data1 为第一片 0832 输出模拟量的数字值
DAC0832_2=data2;                         //data2 为第二片 0832 输出模拟量的数字值
DAC_ALL=0;                               //此处 0 没有意义，目的是使 XFER 同时有效
```

（3）DAC0832 应用——阶梯波的产生。阶梯波是在一定的时间内每隔一段时间输出的幅值递增一个恒定值。如图 9-20 所示，每隔 1ms 输出增长一个定值，经 10 ms 后循环。用 DAC0832 的单缓冲器方式就可以实现这样的波形。

具体的程序如下：

```
#include<absacc.h>
#define DAC0832 XBYTE[0x7FFF]
```

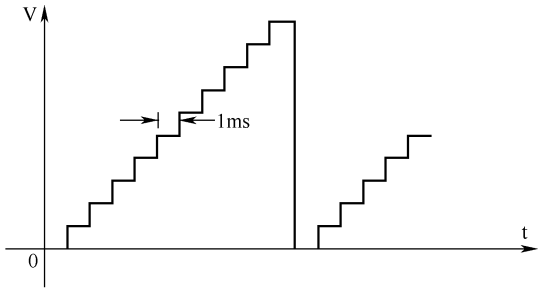


图 9-20　阶梯波波形

```
void main()
{
    unsigned char i;
    while(1)
        for(i=0;i<10;i++)
        {
            DAC0832=i*n;              //n 是为了使波形有一定高度，取值 1～25
            delay(1);                 //延时 1ms
        }
}
```

9.4.2　串行输入 D/A 转换器 TLC5615 接口技术

目前数/模转换器从接口上可分为两大类：并行接口数/模转换器和串行接口数/模转换器。并行接口转换器的引脚多，体积大，占用单片机的口线多；而串行数/模转换器的体积小，占用单片机的口线少。为减小线路板的面积，减少占用单片机的口线，越来越多地采用串行数/模转换器，例如 TI 公司的 TLC5615。

1．TLC5615 的结构原理

TLC5615 是具有 3 线串行接口的数/模转换器。其输出为电压型，最大输出电压是基准电压值的两倍。带有上电复位功能，上电时把 DAC 寄存器复位至全 0。TLC5615 的性价比较高，市场售价比较低。

（1）TLC5615 的特点。

- 10 位 CMOS 电压输出。
- 5V 单电源工作。
- 与微处理器 3 线串行接口（SPI）。
- 最大输出电压是基准电压的 2 倍。
- 输出电压具有和基准电压相同的极性。
- 建立时间 12.5μs。
- 内部上电复位。
- 低功耗，最高为 l.75 mW。
- 引脚与 MAX515 兼容。

（2）功能方框图。TLC5615 的功能方框图如图 9-21 所示。

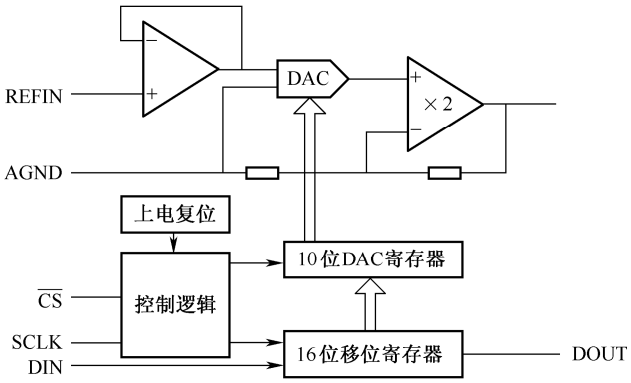


图 9-21　TLC5615 功能方框图

（3）引脚排列及功能。TLC5615 的引脚排列见图 9-22。

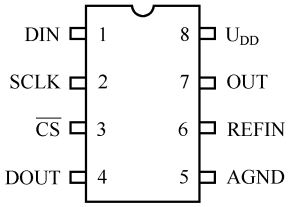


图 9-22　TLC5615 引脚

引脚功能说明：

- DIN：串行数据输入。
- SCLK：串行时钟输入。
- $\overline{CS}$：芯片选择，低电平有效。
- DOUT：用于菊花链（daisy chaining）的串行数据输出。
- AGND：模拟地。
- REFIN：基准电压输入。
- OUT：DAC 模拟电压输出。
- U_{DD}。正电源（4.5～5.5V）。

（4）TLC5615 的输入/输出关系。图 9-23 所示的 D/A 输入/输出关系如表 9-5 所列。

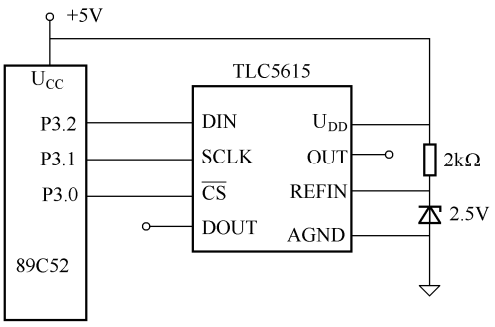


图 9-23　TLC5615 与 89C52 接口电路

表 9-5　D/A 转换关系表

数字量输入	模拟量输出
1111 1111 11（00）	$2V_{REFIN}$×1023/1024
…	…
1000 0000 01（00）	$2V_{REFIN}$×513/1024
1000 0000 00（00）	$2V_{REFIN}$×512/1024
0111 1111 11（00）	$2V_{REFIN}$×511/1024
…	…
0000 0000 01（00）	$2V_{REFIN}$×1/1 024
0000 0000 00（00）	0 V

因为 TLC5615 芯片内的输入锁存器为 12 位宽，所以要在 10 位数字的低位后面再填以数字 XX。XX 为不关心状态。串行传送的方向是先送出高位 MSB，后送出低位 LSB。

10 位	X	X
MSB		LSB

如果有级联电路，则应使用 16 位的传送格式，即在最高位 MSB 的前面再加上 4 个虚位，被转换的 10 位数字在中间。

4 个虚位	10 位	X	X

2. TLC5615 与 89C52 的串行接口电路及程序

图 9-23 为 TLC5615 和 89C52 单片机的接口电路。在电路中，89C52 单片机自 P3.0～P3.2 口分别控制 TLC5615 的片选 $\overline{CS}$、串行时钟输入 SCLK 和串行数据输入 DIN。

D/A 转换程序如下：

```
sbit CS=P3^0;
sbit SCLK=P3^1;
sbit DIN=P3^2;
void DAC(unsigned int adata)
{
    char i;
    adata<<=2;                    //10 位数据升位为 12 位，低 2 位无效
    CS=0;                         //片选有效
    for(i=11;i>=0;i--)
    {
        SCLK=0;                   //时钟低电平
        DIN=adata&(1<<i);         //按位将数据送入 TLC5616
        SCLK=1;                   //时钟高电平
    }
    SCLK=0;                       //时钟低电平
    CS=1;                         //片选高电平，输入的 12 位数据有效
}
```

9.5 开关器件接口

在单片机控制系统中，单片机总要对被控对象实现控制操作。后向通道是计算机实现控制运算处理后，对被控对象的输出通道接口。

系统的后向通道是一个输出通道，其特点是弱电控制强电，即小信号输出实现大功率控制。常见的被控对象有电机、电磁开关等。

单片机实现控制是以数字信号或模拟信号的形式通过 I/O 口送给被控对象的。其中，数字信号形态的开关量、二进制数字量和频率量可直接用于开关量、数字量系统及频率调制系统的控制；但对于一些模拟量控制系统，则应通过 D/A 转换器转换成模拟量控制信号后。才能实现控制。

1. 继电器及接口

单片机用于输出控制时，用得最多的功率开关器件是固态继电器，它将取代电磁式的机械继电器。

（1）单片机与继电器的接口。一个典型的继电器与单片机的接口电路如图 9-24 所示。继电器的工作原理很简单，只要让它的吸合线圈通过一定的电流，线圈产生的磁力就会带动衔铁移动，从而带动开关点的接通和断开，由此控制电路的通或断。

由于继电器的强电触点与吸合线圈之间是隔离的，所以继电器控制输出电路不需要专门设计隔离电路。图中二极管的作用是把继电器吸合线圈的反电动势吸收掉，从而保护晶体管。

（2）单片机与固态继电器接口。固态继电器简称 SSR（Solid State Relay），是一种四端器件：两端输入，两端输出，它们之间用光耦合器隔离。它是一种新型的无触点电子继电器，其输入端仅要求输入很小的控制电流，与 TTL、HTL、CMOS 等集成电路具有较好的兼容性，输入端可以控制输出端的通断。过零开关使得输出开关点在输出端电压在过零的瞬间接通或者断开，以减少由于开关电流造成的干扰。为了防止外电路中的尖峰电压或浪涌电流对开关器件造成的破坏，在输出端回路并联有吸收网络。固态继电器结构如图 9-25 所示。

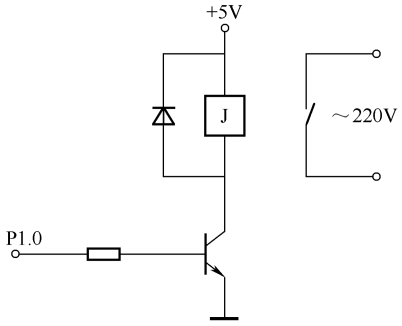


图 9-24 继电器接口

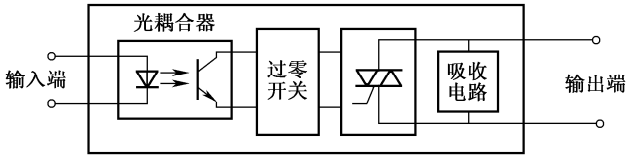


图 9-25 固体继电器内部结构

固态继电器的主要特点是：

1）低噪声。过零型固态继电器在导通和断开时都是在过零点进行的，因此具有最小的无线电干扰和电网污染。

2）可靠性高。因为没有机械触点，全封闭封装，所以耐冲击、耐腐蚀、寿命长。

3）承受浪涌电流大。一般可达额定值的 6～10 倍。

4）驱动功率小。驱动电流只须 10mA，因此可以很方便地与单片机直接连接使用。

5）对电源的适应性强。电源电压在有 20%波动的情况下能正常工作。

6）抗干扰能力强。输入端与输出端之间的电隔离，可以很好地避免强电回路的电污染对控制回路的影响。

图 9-26 是使用固态继电器实现控制单向伺服电动机可逆运转的实例。

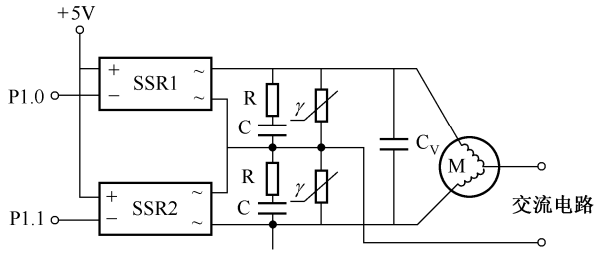


图 9-26　固体继电器使用实例

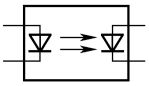
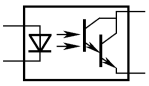
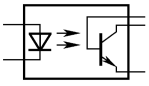
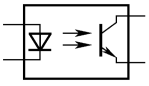
2. 光电耦合器（隔离器）件及驱动接口

后向通道往往所处环境恶劣，控制对象多为大功率伺服驱动机构，电磁干扰较为严重。为防止干扰窜入和保证系统的安全，常常采用光电耦合器，用以实现信号的传输，同时又可将系统与现场隔离开。

以光为媒介传输电信号也可实现在隔离的情况下完成不同系统之间的信号传递。通常是把发光器与受光器封装在同一管壳内，当输入端加电信号时发光器发出光线，光线强度正比于输入信号。受光器接收到光信号后产生与之对应的光电流，由输出端引出，由此实现“电—光—电”的隔离传送。

光耦合器是由一只发光二极管和一只光敏三极管组成的，当发光二极管加上正向电压时，发光二极管通过正向电流而发光，光敏三极管接收到光线而导通。由于发光端与接收端相互是隔离的，所以可以在隔离的情况下传递开关量的信号。光耦合器的种类很多，表 9-6 列出了几种常用的光耦合器。

表 9-6　几种常用的光耦合器

	光电二极管型光耦合器
	达林顿型光耦合器
	光电三极管型光耦合器
	光隔离器

晶体管输出型光电耦合器的受光器是光电晶体管，光电晶体管除了没有使用基极外，跟

普通晶体管一样，取代基极电流的是以光作为晶体管的输入。当光电耦合器的发光二极管发光时，光电晶体管受光的影响在 cb 间和 ce 间会有电流流过，这两个电流基本上受光照度的控制。常用 ce 间的电流作为输出电流，输出电流受 U_{ce} 的电压影响很小，在 U_{ce} 增加时，稍有增加。

光电耦合器在传输脉冲信号时，输入信号和输出信号之间有一定的时间延迟，不同结构光电耦合器的输入/输出延迟时间相差很大。4N25 的导通延迟 t_{ON} 是 2.8μs，关断延迟 t_{OFF} 是 4.5μs；4N33 的 t_{ON} 导通延迟 t_{ON} 是 0.6μs，关断延迟 t_{OFF} 是 45μs。

晶体管输出型光电耦合器可以用作开关，发光二极管和光电晶体管平常都处于关断状态。当发光二极管通过电流脉冲时，光电晶体管在电流脉冲持续的时间内导通（光电耦合器也可作线性耦合器用）。

图 9-27 是使用 4N25 的光电耦合器接口电路图。若 P1.0 输出一个脉冲，则在 74LS04 输出端输出一个相位相同的脉冲。4N25 起耦合脉冲信号和隔离单片机 89C52 系统与输出部分的作用，使两部分的电流相互独立。如输出部分的地线接机壳或接地，而 89C52 系统的电源地线浮空，不与交流电源的地线相接，这样可以避免输出部分电源变化时对单片机电源的影响，减少系统所受的干扰，提高系统的可靠性。4N25 输入/输出端的最大隔离电压大于 2500V。

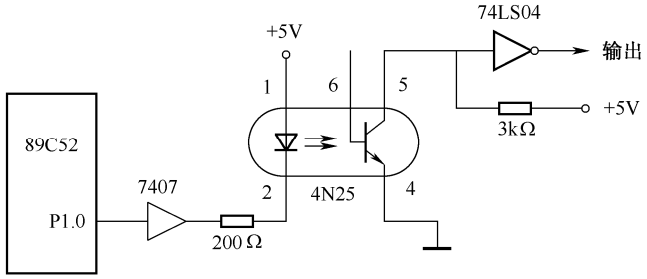


图 9-27　光电耦合器 4N25 的接口电路

图 9-27 所示的接口电路中，使用同相驱动器 OC 门 7407 作为光电耦合器 4N25 输入端的驱动。光电耦合器输入端的电流一般为 10～15mA，发光二极管的压降为 1.2～1.5V。限流电阻由下式计算：

$$R = \frac{U_{CC} - (U_F + U_{CS})}{I_F}$$

式中：U_{CC} 为电源电压；U_F 为输入端发光二极管的压降，取 1.5V；U_{CS} 为驱动器 7407 的压降，取 0.5V。

图 9-27 所示电路要求 I_F 为 15mA，则限流电阻值计算如下：

$$R = \frac{V_{CC} - (V_F + V_{CS})}{I_F} = \frac{5V - 1.5V - 0.5V}{0.015A} = 200\Omega$$

当 89C52 的 P1.0 端输出高电平时，4N25 输入端电流为 0A，三极管 ce 截止，74LS04 的输入端为高电平，7404 输出为低电平；当 89C52 的 P1.0 端输出低电平时，7407 输出端也为低电平，4N25 的输入电流为 15mA，输出端可以流过不小于 3mA 的电流，三极管 ce 导通（如果输出端负载电流小于 3mA），则 ce 间相当于一个接通的开关，74LS04 输出高电平。4N25 的第 6 脚是光电晶体管的基极，在一般的使用中该脚悬空。

思考题与习题

1．为什么要消除键盘的机械抖动？有哪些方法？

2．试说明非编码键盘的工作原理。如何判断键释放？

3．试述 A/D 转换器的种类和特点。

4．设计一个 2×2 的行列式键盘（同在 P1 口）电路并编写键扫描程序。

5．试设计一个 LED 显示器/键盘电路。

6．在一个 fosc 为 12MHz 的 89C52 系统中接有一片 ADC0809，它的地址为 7FFFH，试编写 ADC0809 初始化程序和定时采样特点 2 的程序（假设采样频率为 1ms/次，每次采样 4 个数据）。

7．试说明 TLC2543 的特点和与 89C52 的接口方式。

8．DAC0832 与 89C52 单片机连接时有哪些控制信号？其作用是什么？

9．在一个 89C52 单片机与 DAC0832 组成的应用系统中，DAC0832 的地址为 7FFFH，输出电压为 0～5V。试编写产生矩形波，其波形占空比为 1:4，高电平时电压为 2.5V，低电平时电压为 1.25V 的转换程序。

10．试说明 TLC5615 的特点。

附录 A　ASCII 码表

ASCII（美国标准信息交换码）表

行	列 0	1	2	3	4	5	6	7
位 654→ 位 3210↓	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	“	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	‘	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	–	=	M	]	m	}
E	1110	SO	RS	.	>	N	Ω	n	~
F	1111	SI	US	/	?	O	—	o	DEL

NUL	空	FF	走纸控制	CAN	作废
SOH	标题开始	CR	回车	EM	纸尽
STX	正文开始	SO	移位输出	SUB	减
ETX	正文结束	SI	移位输入	ESC	换码
EOT	传输结果	DLE	数据链换码	FS	文字分隔符
ENQ	询问	DC1	设备控制 1	GS	组分隔符
ACK	应答	DC2	设备控制 2	RS	记录分隔符
BEL	报警符（可听见）	DC3	设备控制 3	US	单元分隔符
BS	退一格	DC4	设备控制 4	SP	空格符
HT	横向列表	NAK	否定	DEL	作废
LF	换行	SYN	空转同步		
VT	垂直制表	ETB	信息组传送结束		

附录 B　MCS-51 指令表

十六进制代码	助记符	功能	对标志影响				字节数	周期数
			CY	AC	OV	P		
算术运算指令								
28～2F	ADD　A, Rn	(A)+(Rn)→A	√	√	√	√	1	1
25 direct	ADD　A,direct	(A)+(direct)→A	√	√	√	√	2	1
26,27	ADD　A, @Ri	(A)+((Ri))→A	√	√	√	√	1	1
24 data	ADD　A, #data	(A)+data→A	√	√	√	√	2	1
38～3F	ADDC　A, Rn	(A)+(Rn)+CY→A	√	√	√	√	1	1
35 direct	ADDC　A, direct	(A)+(direct)+CY→A	√	√	√	√	2	1
36,37	ADDC　A, @Ri	(A)+((Ri))+CY→A	√	√	√	√	1	1
34 data	ADDC　A, #data	(A)+data+CY→A	√	√	√	√	2	1
98～9F	SUBB　A, Rn	(A)−(Rn)−CY→A	√	√	√	√	1	1
95 direct	SUBB　A, direct	(A)−(direct)−CY→A	√	√	√	√	2	1
96,97	SUBB　A, @Ri	(A)−((Ri))−CY→A	√	√	√	√	1	1
94 data	SUBB　A, #data	(A)−data−CY→A	√	√	√	√	2	1
04	INC　A	(A)+1→A	×	×	×	√	1	1
08～0F	INC　Rn	(Rn)+1→Rn	×	×	×	×	1	1
05 direct	INC　direct	(direct)+1→direct	×	×	×	×	2	1
06,07	INC　@Ri	((Ri))+1→(Ri)	×	×	×	×	1	1
A3	INC　DPTR	(DPTR)+1→DPTR	×	×	×	×	1	2
14	DEC　A	(A)−1→A	×	×	×	√	1	1
18～1F	DEC　Rn	(Rn)−1→Rn	×	×	×	×	1	1
15 direct	DEC　direct	(direct)−1→direct	×	×	×	×	2	1
16,17	DEC　@Ri	((Ri))−1→(Ri)	×	×	×	×	1	1
A4	MUL　AB	(A)×(B)→AB	0	×	√	√	1	4
84	DIV　AB	(A)÷(B)→AB	0	×	√	√	1	4
D4	DA　A	对 A 进行十进制调整	√	√	×	√	1	1
逻辑运算指令								
58～5F	ANL　A, Rn	(A)∧(Rn)→A	×	×	×	√	1	1
55 direct	ANL　A, direct	(A)∧(direct)→A	×	×	×	√	2	1
56,57	ANL　A, @Ri	(A)∧((Ri))→A	×	×	×	√	1	1

续表

十六进制代码	助记符	功能	对标志影响				字节数	周期数
			CY	AC	OV	P		
逻辑运算指令								
54 data	ANL A, #data	(A)∧data→A	×	×	×	√	2	1
52 direct	ANL direct, A	(direct)∧(A)→direct	×	×	×	×	2	1
53 direct data	ANL direct, #data	(direct)∧data→direct	×	×	×	×	3	2
48～4F	ORL A, Rn	(A)∨(Rn)→A	×	×	×	√	1	1
45 direct	ORL A, direct	(A)∨(direct)→A	×	×	×	√	2	1
46,47	ORL A, @Ri	(A)∨((Ri))→A	×	×	×	√	1	1
44 data	ORL A, #data	(A)∨data→A	×	×	×	√	2	1
42 direct	ORL direct, A	(direct)∨(A)→direct	×	×	×	×	2	1
43 direct data	ORL direct, #data	(direct)∨data→direct	×	×	×	×	3	2
68～6F	XRL A, Rn	(A)⊕(Rn)→A	×	×	×	√	1	1
65 direct	XRL A, direct	(A)⊕(direct)→A	×	×	×	√	2	1
66,67	XRL A, @Ri	(A)⊕((Ri))→A	×	×	×	√	1	1
64 data	XRL A, #data	(A)⊕data→A	×	×	×	√	2	1
62 direct	XRL direct, A	(direct)⊕(A)→direct	×	×	×	×	2	1
63 direct data	XRL direct, #data	(direct)⊕data→direct	×	×	×	×	3	2
E4	CLR A	0→A	×	×	×	√	1	1
F4	CPL A	$\overline{(A)}$→A	×	×	×	×	1	1
23	RL A	A 循环左移一位	×	×	×	×	1	1
33	RLC A	A 带进位循环左移一位	√	×	×	√	1	1
03	RR A	A 循环右移一位	×	×	×	×	1	1
13	RRC A	A 带进位循环右移一位	√	×	×	√	1	1
数据传送指令								
E8～EF	MOV A, Rn	(Rn)→A	×	×	×	√	1	1
E5 direct	MOV A, direct	(direct)→A	×	×	×	√	2	1
E6,E7	MOV A, @Ri	((Ri))→A	×	×	×	√	1	1
74 data	MOV A, #data	data→A	×	×	×	√	2	1
F8～FF	MOV Rn, A	(A)→Rn	×	×	×	×	1	1
A8～AF direct	MOV Rn, direct	(direct)→Rn	×	×	×	×	2	2
78～7F data	MOV Rn, #data	data→Rn	×	×	×	×	2	1
F5 direct	MOV direct, A	(A)→direct	×	×	×	×	2	1
88～8F direct	MOV direct, Rn	(Rn)→direct	×	×	×	×	2	2
85 direct direct	MOV direct1, direct2	(direct2)→direct1	×	×	×	×	3	2

续表

十六进制代码	助记符	功能	对标志影响				字节数	周期数
			CY	AC	OV	P		
数据传送指令								
86,87 direct	MOV　direct, @Ri	((Ri))→direct	×	×	×	×	2	2
75 direct data	MOV　direct, #data	data→direct	×	×	×	×	3	2
F6,F7	MOV　@Ri, A	(A)→(Ri)	×	×	×	×	1	1
A6,A7 direct	MOV　@Ri, direct	direct→(Ri)	×	×	×	×	2	2
76,77 data	MOV　@Ri, #data	data→(Ri)	×	×	×	×	2	1
90 data16	MOV　DPTR, #data16	data16→DPTR	×	×	×	×	3	2
93	MOVC　A, @A+DPTR	((A)+(DPTR))→A	×	×	×	√	1	2
83	MOVC　A, @A+PC	((A)+(PC))→A	×	×	×	√	1	2
E2,E3	MOVX　A, @Ri	((P2,Ri))→A	×	×	×	√	1	2
E0	MOVX　A, @DPTR	((DPTR))→A	×	×	×	√	1	2
F2,F3	MOVX　@Ri, A	(A)→(P2,Ri)	×	×	×	×	1	2
F0	MOVX　@DPTR, A	(A)→(DPTR)	×	×	×	×	1	2
C0 direct	PUSH　direct	(SP)+1→SP, (direct)→(SP)	×	×	×	×	2	2
D0 direct	POP　direct	((SP))→direct, (SP)-1→SP	×	×	×	×	2	2
C8～CF	XCH　A, Rn	(A)←→(Rn)	×	×	×	√	1	1
C5 direct	XCH　A, direct	(A)←→ (direct)	×	×	×	√	2	1
C6,C7	XCH　A, @Ri	(A)←→((Ri))	×	×	×	√	1	1
D6,D7	XCHD　A, @Ri	$(A)_{0\sim3}$←→$((Ri))_{0\sim3}$	×	×	×	√	1	1
C4	SWAP　A	A 高、低半字节交换	×	×	×	×	1	1
位操作指令								
C3	CLR　C	0→C	√	×	×	×	1	1
C2 bit	CLR　bit	0→bit	×	×	×	×	2	1
D3	SETB　C	1→C	√	×	×	×	1	1
D2 bit	SETC　bit	1→bit	×	×	×	×	2	1
B3	CPL　C	$(\overline{C})$→C	√	×	×	×	1	1
B2 bit	CPL　bit	$\overline{(bit)}$→bit	×	×	×	×	2	1
82 bit	ANL　C, bit	(C)∧(bit)→C	√	×	×	×	2	2
B0 bit	ANL　C, $\overline{bit}$	(C)∧$\overline{(bit)}$→C	√	×	×	×	2	2
72 bit	ORL　C, bit	(C)∨(bit)→C	√	×	×	×	2	2
A0 bit	ORL　C, $\overline{bit}$	(C)∨$\overline{(bit)}$→C	√	×	×	×	2	2
A2 bit	MOV　C, bit	(bit)→C	√	×	×	×	2	1
92 bit	MOV　bit, C	(C)→bit	×	×	×	×	2	2

续表

十六进制代码	助记符	功能	对标志影响				字节数	周期数
			CY	AC	OV	P		
位操作指令								
40 rel	JC rel	若 CY=1，则(PC)+rel→PC	×	×	×	×	2	2
50 rel	JNC rel	若 CY=0，则(PC)+rel→PC	×	×	×	×	2	2
20 bit rel	JB bit, rel	若 bit=1，则(PC)+rel→PC	×	×	×	×	3	2
30 bit rel	JNB bit, rel	若 bit=0，则(PC)+rel→PC	×	×	×	×	3	2
10 bit rel	JBC bit, rel	若 bit=1，则(PC)+rel→PC，且 0→bit	×	×	×	×	3	2
控制转移指令								
CODE1	ACALL addr11	(SP)+1→SP,$(PC)_L$→(SP), (SP)+1→SP,$(PC)_H$→(SP), addr11→$PC_{10\sim0}$	×	×	×	×	2	2
12 addr16	LCALL addr16	(SP)+1→SP,$(PC)_L$→(SP), (SP)+1→SP,$(PC)_H$→(SP),addr16→PC	×	×	×	×	3	2
22	RET	(SP)→PC_H, (SP)-1→SP, (SP)→PC_L, (SP)-1→SP, 从子程序返回	×	×	×	×	1	2
32	RETI	(SP)→PC_H, (SP)-1→SP, (SP)→PC_L, (SP)-1→SP, 从中断返回	×	×	×	×	1	2
CODE2	AJMP addr11	addr11→$PC_{10\sim0}$	×	×	×	×	2	2
0 addr16	LJMP addr16	addr16→PC	×	×	×	×	3	2
80 rel	SJMP rel	(PC)+rel→PC	×	×	×	×	2	2
73	JMP @A+DPTR	(A)+(DPTR)→PC	×	×	×	×	1	2
60 rel	JZ rel	若(A)=0，则(PC)+rel→PC	×	×	×	×	2	2
70 rel	JNZ rel	若(A)≠0，则(PC)+rel→PC	×	×	×	×	2	2
B5 direct rel	CJNE A,direct,rel	若(A)≠(direct)，则(PC)+rel→PC, 若(A)<(direct)，则 1→CY	√	×	×	×	3	2
B4 direct rel	CJNE A,#data,rel	若(A)≠data，则(PC)+rel→PC, 若(A)< data，则 1→CY	√	×	×	×	3	2
B8～BF direct rel	CJNE Rn,#data,rel	若(Rn)≠data，则(PC)+rel→PC, 若(Rn)<data，则 1→CY	√	×	×	×	3	2
B6, B7 direct rel	CJNE @Ri,#data,rel	若((Ri))≠data，则(PC)+rel→PC, 若((Ri))<data，则 1→CY	√	×	×	×	3	2
D8～DF rel	DJNZ Rn, rel	(Rn)-1→Rn, 若(Rn)≠0，则(PC)+rel→PC	×	×	×	×	2	2
D5 direct rel	DJNZ direct, rel	(direct)-1→direct, 若(direct)≠0，则(PC)+rel→PC	×	×	×	×	3	2
00	NOP	空操作	×	×	×	×	1	1

CODE1：代表 $a_{10}a_9a_810001a_7a_6a_5a_4a_3a_2a_1a_0$，其中 a_{10}～a_0 为 addr11 各位。

CODE2：代表 $a_{10}a_9a_800001a_7a_6a_5a_4a_3a_2a_1a_0$，其中 a_{10}～a_0 为 addr11 各位。

附录 C　C51 库函数

C51 编译器的运行库中包含有丰富的库函数，使用库函数可以大大简化用户的程序设计工作，提高编程效率。下面介绍一些常用的库函数，如果用户使用这些库函数，必须在源程序的开始用预处理命令“#include”将相关的头文件包含进来。

C.1　寄存器头文件

寄存器头文件 regxxx.h（如 reg51.h）中定义了 MCS-51 所有特殊功能寄存器和相应位，定义时使用的是大写字母。在 C 语言源程序文件的开始，应该把对应的头文件 regxxx.h 包含进来，在程序中就可以直接使用 MCS-51 中的特殊功能寄存器和相应的位。

C.2　字符函数

字符函数在 ctype.h 头文件中声明，下面给出部分函数。

1. 检查英文字母函数 isalpha

函数原型：extern bit isalpha(char c)

再入属性：reentrant

功能：检查参数字符是否为英文字母，是则返回 1，否则返回 0。

2. 检查英文字母、数字字符函数 isalnum

函数原型：extern bit isalnum(char c)

再入属性：reentrant

功能：检查参数字符是否为英文字母或数字字符，是则返回 1，否则返回 0。

3. 检查数字字符函数 isdigit

函数原型：extern bit isdigit(char c)

再入属性：reentrant

功能：检查参数字符是否为数字字符，是则返回 1，否则返回 0。

4. 检查小写字母函数 islower

函数原型：extern bit islower(char c)

再入属性：reentrant

功能：检查参数字符是否为小写字母，是则返回 1，否则返回 0。

5. 检查大写字母函数 isupper

函数原型：extern bit isupper(char c)

再入属性：reentrant

功能：检查参数字符是否为大写字母，是则返回 1，否则返回 0。

6. 检查十六进制数字字符函数 isxdigit

函数原型：extern bit isxdigit(char c)

再入属性：reentrant

功能：检查参数字符是否为十六进制数字字符，是则返回 1，否则返回 0。

7. 数字字符转换十六进制数函数 toint

函数原型：extern char toint(char c)

再入属性：reentrant

功能：将 ASCII 字符的 0～9、A～F 转换成十六进制数，返回数字 0～F。

8. 转换小写字母函数 tolower

函数原型：extern char tolower(char c)

再入属性：reentrant

功能：将大写字母转换成小写字母，返回小写字母，如果输入的不是大写字母，则不作转换直接返回输入值。

9. 转换大写字母函数 toupper

函数原型：extern char toupper(char c)

再入属性：reentrant

功能：将小写字母转换成大写字母，返回大写字母，如果输入的不是小写字母，则不作转换直接返回输入值。

C.3 一般 I/O 函数

一般输入/输出函数在 stdio.h 头文件中声明，其中所有的函数都是通过单片机的串行口输入/输出的。在使用这些函数之前，应先对单片机的串行口进行初始化。例如串行通信的波特率 4800b/s，晶振频率为 11.0592MHz，初始化程序段为：

```
SCON=0x52;                //设置串行口方式 1、允许接收、启动发送
TMOD=0x20;                //设置定时器 T1 以模式 2 工作
TH1=0xfa;                 //设置 T1 重装初值
TR1=1;                    //开 T1
```

在 stdio.h 文件中声明的输入/输出函数，都是以_getkey 和 putchar 两个函数为基础，如果需要这些函数支持其他的端口，只需修改这两个函数即可。下面给出部分函数。

1. 从串行口输入字符函数_getkey

函数原型：extern char _getkey (void)

再入属性：reentrant

功能：从 51 单片机的串行口读入一个字符，如果没有字符输入则等待，返回值为读入的字符，不显示。

2. 从串行口输入字符并输出函数 getchar

函数原型：extern char getchar(void)

再入属性：reentrant

功能：使用_getkey 函数从 51 单片机的串行口输入一个字符，返回值为读入的字符，并且通过 putchar 函数将字符输出。

3. 从串行口输出字符函数 putchar

函数原型：extern char putchar(char)

再入属性：reentrant

功能：从 51 单片机的串行口输出一个字符，返回值为输出的字符。

4. 从串行口输入字符串函数 gets

函数原型：extern char *gets(char *string, int len)

再入属性：non-reentrant

功能：从 51 单片机的串行口输入一个长度为 len 的字符串（遇到换行符结束输入），并将其存入 string 指定的位置。输入成功返回存入地址的指针，输入失败则返回 NULL。

5. 从串行口格式输出函数 printf

函数原型：extern int printf(格式控制字符串,输出参数表)

再入属性：non-reentrant

功能：该函数是以一定的格式从 51 单片机的串行口输出数值和字符串，返回值为实际输出的字符数。

6. 格式输出到内存函数 sprintf

函数原型：extern int sprintf(char *,格式控制字符串,输出参数表)

再入属性：non-reentrant

功能：该函数与 printf 函数功能相似，但数据不是输出到串行口，而是送入一个字符指针指向的内存中，并且以 ASCII 码的形式存储。

7. 从串行口输出字符串函数 puts

函数原型：extern int puts(const char *)

再入属性：reentrant

功能：该函数将字符串和换行符输出到串行口，正确返回一个非负数，错误返回 EOF。

8. 从串行口格式输入函数 scanf

函数原型：extern int scanf(格式控制字符串,输入参数表)

再入属性：non-reentrant

功能：该函数在格式控制字符串的控制下，利用 getchar 函数从串行口读入数据，每遇到一个符合格式控制串规定的值，就将它顺序地存入由参数表中指向的存储单元。每个参数都必须是指针型。正确输入其返回值为输入的项数，错误则返回 EOF。

C.4　标准函数

标准函数在 stdlib.h 头文件中声明，下面给出部分函数。

1. 字符串转换浮点数函数 atof

函数原型：float atof (void *string)

再入属性：non-reentrant

功能：该函数把字符串转换成浮点数并返回。

2. 字符串转换整型数函数 atoi

函数原型：int atoi (void *string)

再入属性：non-reentrant

功能：该函数把字符串转换成整型数并返回。

3. 字符串转换长整数函数 atol

函数原型：long atol (void *string)

再入属性：non-reentrant

功能：该函数把字符串转换成长整数并返回。

4. 申请内存函数 malloc

函数原型：void *malloc (unsigned int size)

再入属性：non-reentrant

功能：该函数申请一块大小为 size 的内存，并返回其指针，所分配的区域不初始化。如果无内存空间可用，则返回 NULL。

5. 释放内存函数 free

函数原型：void free (void xdata *p)

再入属性：non-reentrant

功能：该函数释放指针 p 所指向的区域，p 必须是以前用 malloc 等函数分配的存储区指针。

C.5 数学函数

数学函数在头文件 math.h 中声明，下面给出部分函数。

1. 求绝对值函数 cabs、abs、fabs 和 labs

函数原型： extern char cabs (char i)
extern int abs (int i)
extern float fabs (float i)
extern long labs (long i)

再入属性：reentrant

功能：计算并返回 i 的绝对值。这 4 个函数除了变量和返回值类型不同之外，其功能完全相同。

2. 求平方根函数 sqrt

函数原型：extern float sqrt (float i)

再入属性：non-reentrant

功能：计算并返回 i 的平方根。

3. 产生随机数函数 rand 和 srand

函数原型： extern int rand (void)
extern void srand (int seed)

再入属性：reentrant，non-reentrant

功能：rand 函数产生并返回一个 0～32767 之间的伪随机数；srand 用来将随机数发生器初始化成一个已知的值，对函数 rand 的相继调用将产生相同序列的随机数。

4. 求三角函数 cos、sin 和 tan

函数原型： extern float cos (float i)

extern float sin (float i)

extern float tan (float i)

再入属性：non-reentrant

功能：3 个函数分别返回 i 的 cos、sin、tan 的函数值。3 个函数变量的范围都是−π/2～+π/2，变量的值必须在 ± 65535 之间，否则产生一个 NaN 错误。

5. 求反三角函数 acos、asin、atan 和 atan2

函数原型：　extern float acos (float i)

extern float asin (float i)

extern float atan (float i)

extern float atan2 (float i, float j)

再入属性：non-reentrant

功能：前 3 个函数分别返回 i 的反余弦值、反正弦值、反正切值，3 个函数的值域都是−π/2～+π/2；atan2 返回 i/j 的反正切值，其值域是−π～+π。

C.6　内部函数

内部函数在头文件 intrins.h 中声明。

1. 循环左移 n 位函数_crol_、_irol_、_lrol_

函数原型分别为：

unsigned char _crol_ (unsigned char, unsigned char n)

unsigned int _irol_ (unsigned int, unsigned char n)

unsigned long _lrol_ (unsigned long, unsigned char n)

再入属性：reentrant，intrinsic

功能：这些函数都是将第一个参数（无符号字符、无符号整型数、无符号长整型数）循环左移 n 位，返回被移动后的数。

2. 循环右移 n 位函数_cror_、_iror_、_lror_

函数原型分别为：

unsigned char _cror_ (unsigned char, unsigned char n)

unsigned int _iror_ (unsigned int, unsigned char n)

unsigned long _lror_ (unsigned long, unsigned char n)

再入属性：reentrant，intrinsic

功能：这些函数都是将第一个参数（无符号字符数、无符号整型数、无符号长整型数）循环右移 n 位，返回被移动后的数。

3. 空操作函数_nop_

函数原型：void _nop_ (void)

再入属性：reentrant，intrinsic

功能：该函数产生一个 MCS-51 单片机的空操作函数。

4. 位测试函数_testbit_

函数原型：bit _testbit_ (bit)

再入属性：reentrant，intrinsic

功能：该函数产生一个 MCS-51 单片机的位操作指令 JBC，对字节中的一个位进行测试，如果该位为 1，则返回 1，并且将该位清 0；如果该位为 0，则直接返回 0。

C.7 字符串函数

字符串函数在头文件 string.h 中声明，下面给出部分函数。

1. 存储器数据复制函数 memcopy

函数原型：void *memcopy (void *dest, void *src, int len)

再入属性：reentrant

功能：该函数将存储区 src 中的 len 个字符复制到存储区 dest 中，返回指向 dest 的指针。如果存储区 src 和 dest 有重叠，不能保证其正确性。

2. 存储器数据复制函数 memccpy

函数原型：void *memccpy (void *dest, void *src, char cc, int len)

再入属性：non-reentrant

功能：该函数将存储区 src 中的 len 个字符复制到存储区 dest 中，如果遇到字符 cc，则把 cc 复制后就结束。对于返回值，如果复制了 len 个字符，则返回 NULL，否则返回指向 dest 中下一个字符的指针。如果存储区 src 和 dest 有重叠，不能保证其正确性。

3. 存储器数据移动函数 memmove

函数原型：void *memmove (void *dest, void *src, int len)

再入属性：reentrant

功能：该函数将存储区 src 中的 len 个字符移动到存储区 dest 中，返回指向 dest 的指针。如果存储区 src 和 dest 有重叠，也能够正确移动。

4. 存储器字符查找函数 memchr

函数原型：void *memchr (void *buf, char cc, int len)

再入属性：reentrant

功能：该函数顺序搜索存储区 buf 中前 len 个字符，查找字符 cc，如果找到，则返回指向 cc 的指针，否则返回 NULL。

5. 存储器字符比较函数 memcmp

函数原型：char memcmp (void *buf1, void *buf2, int len)

再入属性：reentrant

功能：该函数逐个字符比较存储区 buf1 和 buf2 的前 len 个字符，如果相等则返回 0，如果不等，则返回第一个不等的字符的差值（buf1 的字符减 buf2 的字符）。

6. 存储器写字符函数 memset

函数原型：void *memset (void *buf, char cc, int len)

再入属性：reentrant

功能：该函数向存储区 buf 写 len 个字符 cc，返回 buf 指针。

7. 字符串挂接函数 strcat

函数原型：char *strcat (char *dest, char *src)

再入属性：non-reentrant

功能：该函数将字符串 src 复制到 dest 的尾部，返回指向 dest 的指针。

8. n 个字符挂接函数 strncat

函数原型：char *strncat (char *dest, char *src, int len)

再入属性：non-reentrant

功能：该函数将字符串 src 中的前 len 个字符复制到 dest 的尾部，返回指向 dest 的指针。

9. 字符串复制函数 strcpy

函数原型：char *strcpy(char *dest, char *src)

再入属性：reentrant

功能：该函数将字符串 src 复制到 dest 中，包括结束符，返回指向 dest 的指针。

10. n 个字符复制函数 strncpy

函数原型：char *strncpy (char *dest, char *src, int len)

再入属性：non-reentrant

功能：该函数将字符串 src 中的前 len 个字符复制到 dest 中，返回指向 dest 的指针。如果 src 的长度小于 len，则在 dest 中以 0 补齐到长度 len。

11. 字符串比较函数 strcmp

函数原型：char strcmp (char *string1, char *string2)

再入属性：reentrant

功能：该函数逐个字符比较字符串 string1 和 string2，如果相等则返回 0，如果不等，则返回第一个不等的字符的差值（string1 的字符减 string2 的字符）。

12. 字符串 n 个字符比较函数 strncmp

函数原型：char strncmp (char *string1, char *string2, int len)

再入属性：non-reentrant

功能：该函数逐个字符比较字符串 string1 和 string2 中的前 len 字符，如果相等则返回 0，如果不等，则返回第一个不等的字符的差值（string1 的字符减 string2 的字符）。

13. 字符串长度测量函数 strlen

函数原型：int strlen (char *src)

再入属性：non-reentrant

功能：该函数测试字符串 src 的长度，包括结束符，并将长度返回。

14. 字符串字符查找函数 strchr

函数原型：　void *strchr (const char *string, char cc)

　　int strpos (const char *string, char cc)

再入属性：reentrant

功能：strchr 函数顺序搜索字符串 src 中第一次出现的字符 cc（包括结束符），如果找到，则返回指向 cc 的指针，否则返回 NULL。strpos 的功能与 strchr 相似，但返回的是 cc 在字符串中出现的位置值，未找到则返回-1，第一个字符是 cc 则返回 0。

C.8 绝对地址访问函数

绝对地址访问函数在头文件 absacc.h 中声明。

1. 绝对地址字节访问函数 CBYTE、DBYTE、PBYTE、XBYTE

函数原型分别为：#define CBYTE ((unsigned char volatile code*) 0)

#define DBYTE ((unsigned char volatile data *) 0)

#define PBYTE ((unsigned char volatile pdata *) 0)

#define XBYTE ((unsigned char volatile xdata *) 0)

功能：上述宏定义用来对 MCS-51 单片机的存储空间进行绝对地址访问，可以作为字节寻址。CBYTE 寻址 CODE 区，DBYTE 寻址 DATA 区，PBYTE 寻址分页的 XDATA 区，XBYTE 寻址 XDATA 区。

2. 绝对地址字访问函数 CWORD、DWORD、PWORD、XWORD

函数原型分别为：#define CWORD ((unsigned int volatile code*) 0)

#define DWORD ((unsigned int volatile data *) 0)

#define PWORD ((unsigned int volatile pdata *) 0)

#define XWORD ((unsigned int volatile xdata *) 0)

这些宏的功能与前面的宏类似，区别在于这些宏的数据类型是无符号整型 unsigned int。

参考文献

[1] 李朝青编著．单片机原理及接口技术（第 3 版）．北京：北京航空航天大学出版社，2006．

[2] 谢维成，杨加国主编．单片机原理与应用及 C51 程序设计．北京：清华大学出版社，2006．

[3] 胡伟，季晓衡编著．单片机 C 程序设计及应用实例．北京：人民邮电出版社，2003．

[4] 周立功等编著．增强型 80C51 单片机速成与实战．北京：北京航空航天大学出版社，2004．

[5] 谭浩强编著．C 程序设计．北京：清华大学出版社，1999．

[6] 李建忠编著．单片机原理及应用．西安：西安电子科技大学出版社，2002．

[7] 吴黎明主编．单片机原理及应用技术．北京：科学技术出版社，2005．

[8] 何桥主编．单片机原理及应用．北京：中国铁道出版社，2004．