DSP f2812　键盘数码管 C 程序实例

一、说明

我们所使用的开发板中，选用了专用芯片 BC7281 来驱动的键盘和数码管，BC7281 通过外接移位寄存器（典型芯片如 74HC164,74LS595 等），最多可以控制 16 位数码管显示或 128 只独立的 LED。BC7281 的驱动输出极性及输出时序均为软件可控，从而可以和工作外部电路配合。

二、BC7281 引脚说明：

名称	引脚号	说明
DAT	1	与 MCU 串行通讯数据端，为双向数据传输口，作为输出时为漏极开路输出，需要外接上拉电阻。
KEY	2	键盘有效输出端，低电平有效，检测到有效按键后该引脚变为低电平，并一直保持到键值锁存器内容被读出
CLK	3	与 MCU 串行通讯时钟端，下降沿有效。
RST	4	复位端，低电平有效。芯片内部有上电复位电路，故该脚与 vcc 相连。
GND	5	接地
DIG0-DIG7	6-13	位驱动输出
VCC	14	电源输入端
OSCO	15	RC 振荡输出，一般悬空
RC	16	外接 RC 振荡器
SCLK	17	外接段驱动用移位寄存器时钟端
SDAT	18	外接段驱动用移位寄存器输出端，输出段驱动数据，低位在前

三、通讯模式：

1）指令格式

BC7281 与 DSP 之间的通讯采用 2 线高速串行接口，两根连线分别是数据线 DAT 和同步时钟线 CLK，其中 DAT 为双向数据传输线，BC7281 既用该线从 DSP 接收数据，也用该线向 DSP 发送数据。BC7281 的 DAT 引脚为漏极开路输出结构，使用时需要在该线上加上拉电阻。CLK 引脚为串行接口同步时钟，由 DSP 控制，下降沿有效。

串行接口数据宽度为 8 位，两个字节为一组，构成一条完整的指令。第一个字节为命令字，第二个字节为数据。字节在传送时高位（MSB）在前。串行接口数据结构如下：

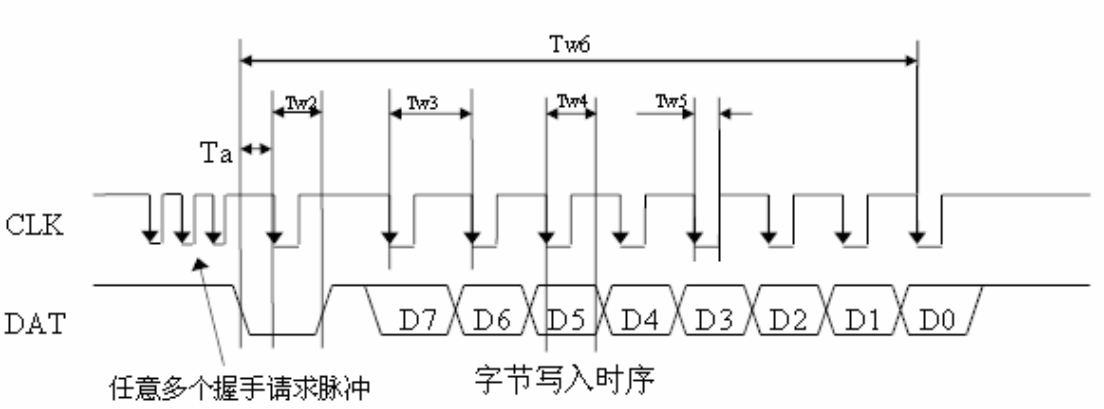
指令字节								数据字节							
D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
R/W	0	0	a4	a3	a2	a1	a0	d7	d6	d5	d4	d3	d2	d1	d0

指令字节中 R/W 为读写控制，当 R/W=0 时，由 DSP 向 BC7281 的内部寄存器内写入数据；当 R/W=1 时，DSP 读出 BC7281 内部寄存器的数据。a0-a4 为目标寄存器的地址，其范围为 00H-1FH

2）时序

1、字节写入 BC7281——指令字节及数据字节

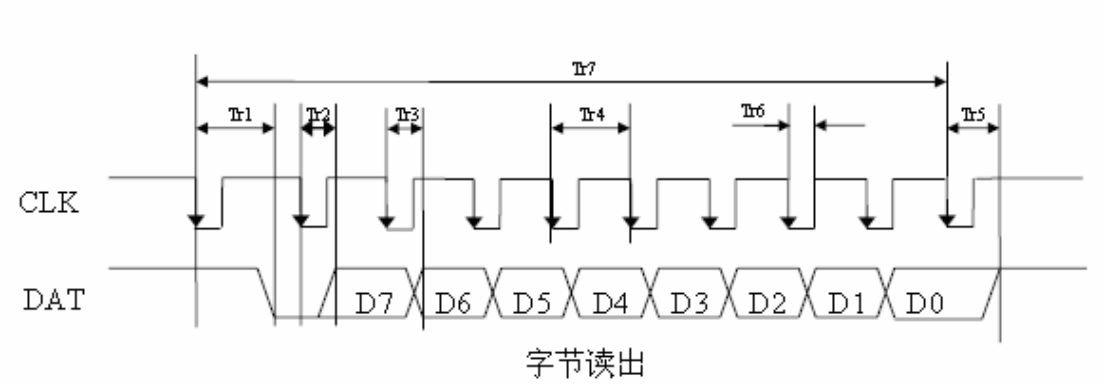
字节写入 BC7281,包括指令字节和写入 BC7281 的数据字节，一个写入寄存器的指令，由连个字节写入操作组成，第一个字节为指令字节，而第二个字节则为数据字节。在接口空闲的情况下，BC7281 的 DAT 引脚处于高阻输入状态，而 DSP 端也将 DAT 线置于输入状态，上拉电阻使得 DAT 线上为高电平。传输开始时，首先需要建立握手信号，DSP 先向 BC7281 发出一系列 CLK 脉冲，脉冲的数量可以是任意多个 DSP 同时监测 DAT 线，而 BC7281 在收到该握手脉冲后在 DAT 线输入一低电平，表示准备好可以接受 DSP 数据，DSP 一旦检测到 BC7281 的响应信号，立刻停止发送握手脉冲，并在规定时间之类，在 CLK 线上再发送一次脉冲，该脉冲使得 BC7281 的 DAT 引脚恢复高阻输入状态，因为 DAT 线上有上拉电阻，因此 DAT 线上回复陈为高电平，DSP 在测试到 DAT 线恢复成高电平后，即开始发送数据。发送是数据的高位（MSB）在前。每发送一 CLK 脉冲，开始部分已发送部分的 CLK 脉冲均为下降沿有效。需要注意：DSP 检测到 BC7281 的低电平握手信号后，应该在 Ta 时间之内给出下一个 CLK 脉冲，同时注意数据传输时的数据保持时间，如果数据保持时间小于规定的值，则有可能造成通讯错误。



2、字节从 BC7281 读出——读出数据字节

读寄存器操作，由一个字节写入操作和一个字节读出操作两部分，字节写入操作写入指令字，数据则有字节读出操作读出。DSP 在传送完指令字后，应将 DAT 线置于输入状态，以便从 BC7281 接收数据。读出数据时，也需要建立握手信号，过程与写入数据时相似，但又不太噢乖，数据读出是，DSP 仅发送一个单一的握手脉冲，而不是像希尔数据是不停的发送直到收到 BC7281 响应信号。具体过程是：DSP 首先向 BC7281 发出一个起始 CLK 脉冲，BC7281 在收到该脉冲后在 DAT 脚开始输出数据。此后 BC7281 每收到一个脉冲，即在 DAT 上输出一个数据为。一个与希尔指令不同的地

方是，当 8 个数据均读出了以后，DSP 还必须多发出一个 CLK 脉冲，表示数据接收完毕，BC7281 才能从数据输出状态转成输入状态，准备接收下一个指令。



注意：在数据传送期间，BC7281 不会进行显示和键盘扫描，因此如果通讯速度过慢（指令的传送时间>大于扫描周期），将会对显示造成影响。

四、外部电路

1）BC7281 的 16 脚为电源引脚，虽然可以直接将其与电路中的电源相连，但是为了获得更好的抗干扰能力，在 VCC 引脚和电源之间，串入了一个 RC 滤波电路。

2）BC7281B 采用外接的 RC 振荡电路为显示和键盘扫描体够时钟驱动，外接玩家的典型参数为 R=1.5K-3.3K,C=20PF，见下图。当采用比较小的电阻值是，振荡频率较高，可以获得更快的通讯速率和扫描速率，使得占用 cpu 时间跟随，显示也更稳定。当 R=1.5K 时，振荡频率为 9.5MHz。当 R=3.3K 时，振荡频率为 5.0MHz。BC7281 的 OSCO 端为内部真的电路的输出端，其输出的频率为 RC 振荡频率的一半，可以用作测量振荡频率之用（请不要直接测量 RC 引脚上的频率），因为测量仪器会引起振荡频率的改变，从而无法得到真实的结果），电路中一般此脚悬空即可。在电路板布线时，振荡电路的元件应尽可能地靠近 BC7281 芯片，并尽量使连线最短。

3）芯片的 RST 引脚为复位端。因为 BC7281 的内部有上电复位电路因此在一般情况下不需要特殊的复位电路，只需将 RST 引脚直接连接到 VCC 端就可以了。如果需要外部的复位电路，RST 撒谎个的复位脉冲的最小宽度为 20us，复位电路中电阻 R 值一般不要超过 40K。另外一种方法是直接由 DSP 控制 BC7281 的复位。BC7281 的复位过程大概需要 25ms 的时间，也即 RST 为高电平平均 25ms 后，BC7281 才开始工作。

4）BC7281 与 DSP 的接口共需要三根线，数据线 DAT,时钟线 CLK 和按键指示 KEY，其中 CLK 和 KEY 引脚分别为输入和输出引脚，而 DAT 脚则为双向口，器内部为 POEN DRAIN 结构，需要外接一个 10-20K 的上拉电阻，使其可靠地输出高电平。

5）DIG0-DIG7 为位驱动输出，BC7281 适合连接共阳式的数码管，外部驱动电路比较简单，只需要 8 只 NPN 型三极管即可，三极管结成射极跟随器形式，因此基级无需限流电阻。

6）BC7281 最多可以连接 64 个按键，按 8X8 矩阵排列，矩阵的‘行’连接到 BC7281 的位驱动

DIG0-DIG7，矩阵的‘列’连接到第 0-7 位显示的段驱动移位寄存器的输出，为了防止对显示部分的影响，键盘矩阵与显示电路之间必须加入二极管和 4.7K 的隔离电阻。当使用 BC7281 的键盘功能时，DIG0-DIG7 上应加以 100K 的下拉电阻，且 8 根引脚必须都接入下拉电阻，即使所用到的键比较少时，也不能将其中未连接键盘的引脚上的下拉电阻省略。

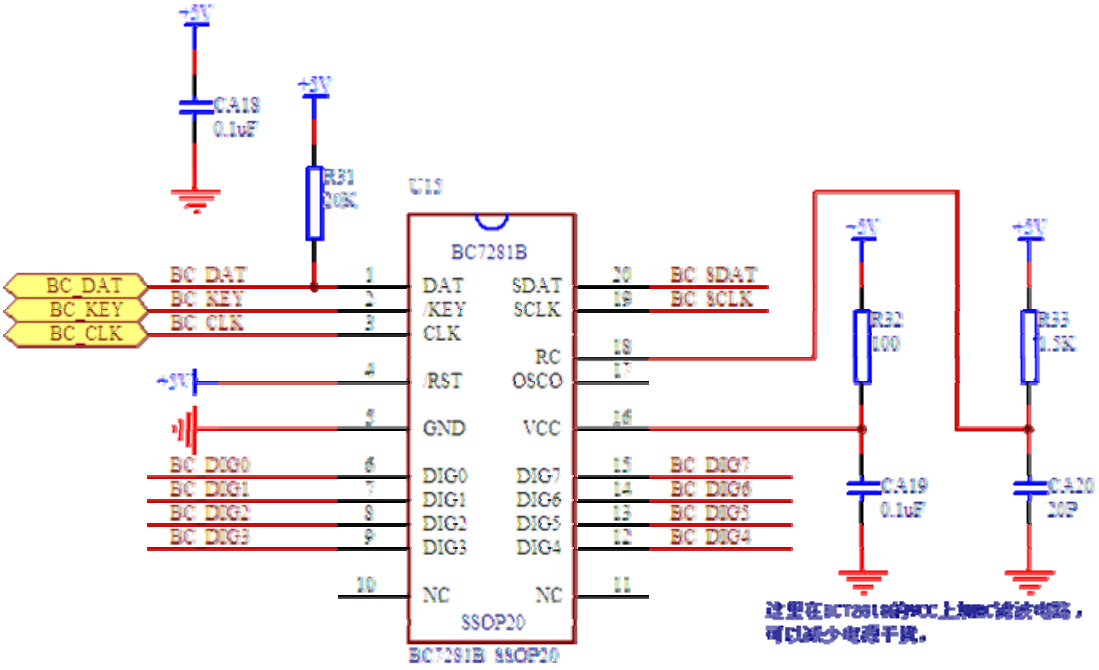


图 BC7281 外部电路接线图

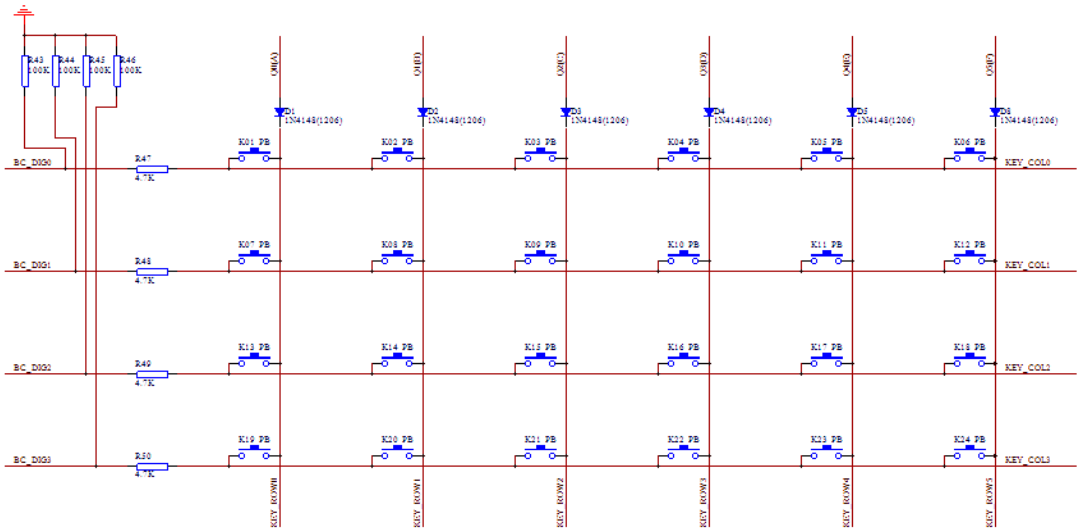


图 键盘接线图

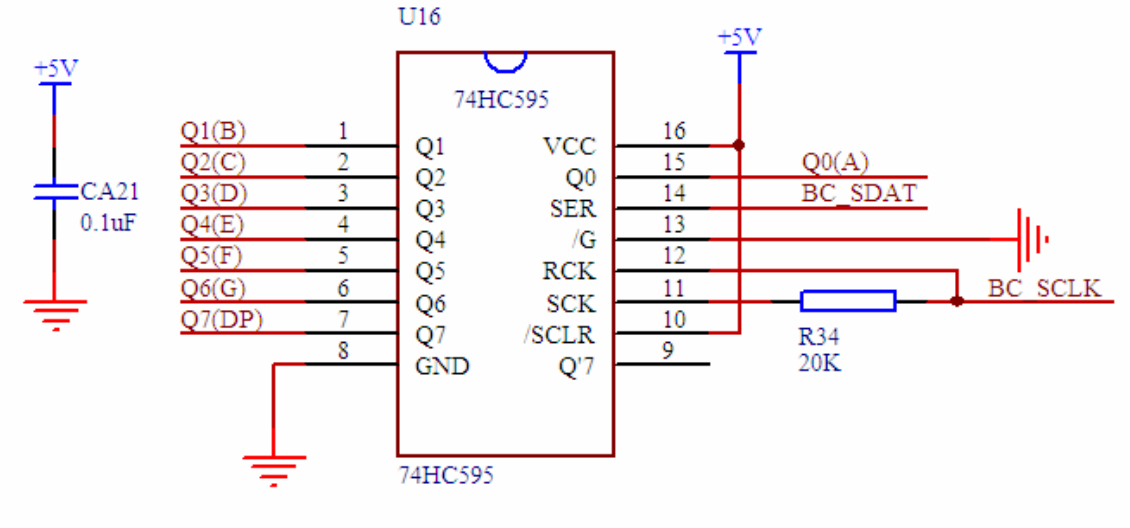


图 移位寄存器

四、C 程序

下面是数码管键盘的源程序，在本开发板上，采用的是三线模式，其中 BC7281 的 DAT 线连接在了 GPIOF10 接口上，KEY 线连接在了 GPIOF8 接口上，CLK 线连接在了 GPIOF9 接口上。

```
//---------------------------------------------设置 BC7281 端口程序--------------
// BC_DAT = P26 = GPIOF10
void set_BC_DAT(int state)
{
    EALLOW;
    GpioMuxRegs.GPFDIR.bit.GPIOF10 = 1;      //dat  口设置成输出
    EDIS;
    switch(state)
    {
        case 0:
            GpioDataRegs.GPFDAT.bit.GPIOF10 = 0;          // clear BC_DAT.
            break;
        case 1:
            GpioDataRegs.GPFDAT.bit.GPIOF10 = 1;          // setb BC_DAT.
            break;
    }
}
// BC_DAT = P26 = GPIOF10
```

```
unsigned char read_BC_DAT(void)
{
    EALLOW;
    GpioMuxRegs.GPFDIR.bit.GPIOF10 = 0;      //dat  口设置成输入
    EDIS;

    switch(GpioDataRegs.GPFDAT.bit.GPIOF10)
    {
        case 0:
            return 0x00;            // read, BC_DAT=0.
        case 1:
            return 0x01;            // read, BC_DAT=1.
    }
    return 0xff;
}
// BC_KEY = P28 = GPIOF8
void set_BC_KEY(int state)
{
    EALLOW;
    GpioMuxRegs.GPFDIR.bit.GPIOF8 = 1;      //dat  口设置成输出
    EDIS;

    switch(state)
    {
        case 0:
            GpioDataRegs.GPFDAT.bit.GPIOF8 = 0;            // clear BC_KEY.
            break;
        case 1:
            GpioDataRegs.GPFDAT.bit.GPIOF8 = 1;            // setb BC_KEY.
            break;
    }
}
// BC_KEY = P28 = GPIOF8
unsigned char read_BC_KEY(void)
```

```
{
    EALLOW;
    GpioMuxRegs.GPFDIR.bit.GPIOF8 = 0;      //dat 口设置成输入
    EDIS;

    switch(GpioDataRegs.GPFDAT.bit.GPIOF8)
    {
        case 0:
            return 0x00;            // read, BC_KEY=0.
        case 1:
            return 0x01;            // read, BC_KEY=1.
    }
    return 0xff;
}

// BC_CLK = P25 = GPIOF9
void set_BC_CLK(int state)
{
    EALLOW;
    GpioMuxRegs.GPFDIR.bit.GPIOF9 = 1;      //dat 口设置成输出
    EDIS;

    switch(state)
    {
        case 0:
            GpioDataRegs.GPFDAT.bit.GPIOF9 = 0;          // clear BC_CLK.
            break;
        case 1:
            GpioDataRegs.GPFDAT.bit.GPIOF9 = 1;          // setb BC_CLK.
            break;
    }
}
//-------------下面是 BC7281 子程序-------------------------
//-------------向 BC728X 发送一个字节--------------------------------
```

```
// output parameters:      bc7281_send_1_byte()                    // =0x00, 超时出错。0x01，成功。
unsigned char bc7281_send_1_byte(unsigned char send_byte,unsigned int overtime_in_us)
{
    unsigned char bit_counter;
    gen_timer1_counter_in_us=overtime_in_us;
    if(gen_timer1_counter_in_us==0)return 0x00; // 超时出错，直接返回。
    // 直接发一个时钟脉冲。
    set_BC_CLK(0);
    delay_gen_timer2_in_us(200); // BC7281 要求这个时钟脉冲宽度不小于 1us. 这里是 200us.
    set_BC_CLK(1);
    // 然后等待 BC_DAT 变低。
    gen_timer2_counter_in_us=5000; // 在 F2812 向 BC7281 发出一个时钟脉冲后，用 5ms 时间来等待 BC_DAT 变低。如果在 5ms 内没有变低，则重发一个时钟脉冲。
    while(read_BC_DAT()==1)             //读取 DAT 的状态线，当 DAT 为高时，继续读取状态。
    {
        if(gen_timer1_counter_in_us==0)return 0x00;             // 超时出错，直接返回。
        if(gen_timer2_counter_in_us==0)return 0x00;             // 超时出错，直接返回。
    }

    // 至此, BC7281 已响应. DAT 变低，要再发一个 CLK，等待 DAT 变高.
    set_BC_CLK(0);
    delay_gen_timer2_in_us(200);       // BC7281 要求这个时钟脉冲宽度不小于 1us. 这里是 200us.
    set_BC_CLK(1);

    gen_timer2_counter_in_us=5000;
    while(read_BC_DAT()==0)
    {
        if(gen_timer1_counter_in_us==0)return 0x00;                 // 超时出错，直接返回。
        if(gen_timer2_counter_in_us==0)return 0x00;                 // 超时出错，直接返回。
    }

    // BC_DAT 已变高
    delay_gen_timer2_in_us(200);       // BC7281 要求这个时钟脉冲宽度不小于 1us. 这里是 200us.
```

```
    // 下面开始发送 8 个 bits.
    // 至此, DAT 又变高. 可以开始发送 8bits.
    for (bit_counter = 0; bit_counter < 8; bit_counter++)
    {                   // 发送 8 个比特
        if ((send_byte & 0x80) == 0)
         {
              set_BC_DAT(0);      // 如果待发 bit 为 0, 置 dat 为 0
         }
         else
         {
              set_BC_DAT(1);      // 反之置为 1
          }
          send_byte = send_byte << 1;       // send_byte 左移一位
          set_BC_CLK(0);                    // 输出一 clk 脉冲
//          set_LED0(0);
        delay_gen_timer2_in_us(200);// BC7281 要求这个时钟脉冲宽度不小于 1us. 这里是 200us.

        set_BC_CLK(1);
//          set_LED0(1);
        delay_gen_timer2_in_us(200); // BC7281 要求这个时钟周期，即 0+1 的总时长，不小于 8us. 这里是共 400us.
        }
    return 0x00;      // 已成功发送一个字节。
}

//-----------从 BC728X 接收一个字节-----------------------------
unsigned char bc7281_receive_1_byte(unsigned int overtime_in_us)
{
    unsigned char bit_counter, in_byte;
    gen_timer1_counter_in_us=overtime_in_us;
    if(gen_timer1_counter_in_us==0)return 0x00;                    // 超时出错, 直接返回。

    // 直接发一个时钟脉冲,请求握手信号。
    set_BC_CLK(0);
```

```
    delay_gen_timer2_in_us(200);      // BC7281 要求这个时钟脉冲宽度不小于 1us. 这里是 200us.
    set_BC_CLK(1);
    while(read_BC_DAT()==1)                                          // 等待 DAT 响应
    {
        if(gen_timer1_counter_in_us==0)return 0x00;                  // 超时出错, 直接返回。
        if(gen_timer2_counter_in_us==0)return 0x00;                  // 超时出错, 直接返回。
    }
    set_BC_CLK(0);
    delay_gen_timer2_in_us(200);// BC7281 要求这个时钟脉冲宽度不小于 1us. 这里是 200us.
    set_BC_CLK(1);
    for (bit_counter=0;bit_counter<8;bit_counter++)
    {                   // 接收 8 个 bit

        in_byte=in_byte*2;      // in_byte 左移一位
        if (read_BC_DAT()==1)      // 如果 dat 为'1'
        {
            in_byte=in_byte|0x01;      // bit0=1
        }
        set_BC_CLK(0);
        delay_gen_timer2_in_us(200);// BC7281 要求这个时钟脉冲宽度不小于 1us. 这里是 200us.
        set_BC_CLK(1);
        delay_gen_timer2_in_us(200);      // 短暂延时
    }

    return(in_byte);
}
//----写入 BC728X, 第一个参数为目标寄存器地址, 第二个参数为要写入的数据--------
void write7281(unsigned char reg_add, unsigned char write_data)
{
    bc7281_send_1_byte(reg_add,3000);            // 发送寄存器地址
    delay_gen_timer2_in_us(100);
    bc7281_send_1_byte(write_data,3000);            // 发送数据字节
}
//-------读出 BC728X 内部寄存器的值, 调用参数为寄存器地址-----------------
```

```
unsigned char read7281(unsigned char reg_add)
{
    bc7281_send_1_byte(reg_add|0x80,3000);      // 发送读指令(bit7=1)
    return(bc7281_receive_1_byte(3000));        // 接收数据字节并返回
}
//--------调试 BC7281 程序----------------------------
void debug_bc7281()
{
        unsigned char key_number;

        write7281(0x12,0x81);           // initialize BC7281. 采用 74HC595.

            write7281(0x15,0x00);           // display '0' at postion 0.
            write7281(0x15,0x10);           // display '1' at postion 1.
            write7281(0x15,0x26);           // display '2' at postion 2.
            write7281(0x15,0x35);           // display '3' at postion 3.
            write7281(0x15,0x44);           // display '4' at postion 4.
            write7281(0x15,0x53);           // display '5' at postion 5.
            write7281(0x15,0x62);           // display '6' at postion 6.
            write7281(0x15,0x71);           // display '7' at postion 7.


        while(1)
        {
            delay_gen_timer4_in_ms(50);
            if(read_BC_KEY()==0)                // 等待按键
            {
                key_number=read7281(0x13);
               key_number = key_number & 0x00ff;
                switch(key_number)
                {
                    case 0x00:
                        key_number=0x01;
                        break;
```

```
            case 0x01:
                key_number=0x04;
                break;
            case 0x02:
                key_number=0x07;
                break;
           case 0x03:
                key_number=0x0a;
                break;
            case 0x08:
                key_number=0x02;
                break;
            case 0x09:
                key_number=0x05;
                break;
            case 0x0a:
                key_number=0x08;
                break;
            case 0x0b:
                key_number=0x00;
                break;
            case 0x10:
                key_number=0x03;
                break;
            case 0x11:
                key_number=0x06;
                break;
            case 0x12:
                key_number=0x09;
                break;
            case 0x13:
                key_number=0x0b;
                break;
            case 0x18:
```

```
            key_number=0x0c;
            break;
        case 0x19:
            key_number=0x0d;
            break;
        case 0x1a:
            key_number=0x0e;
            break;
        case 0x1b:
            key_number=0x0f;
            break;
        case 0x20:
            key_number=0x1a;
            break;
        case 0x21:
            key_number=0x1b;
            break;
        case 0x22:
            key_number=0x1c;
            break;
        case 0x23:
            key_number=0x1d;
            break;
        case 0x28:
            key_number=0x1e;
            break;
        case 0x29:
            key_number=0x1f;
            break;
        case 0x2a:
            key_number=0x2a;
            break;
        case 0x2b:
            key_number=0x2b;
```

```
break;
}
// 在第 1 位上以 HEX 译码方式显示键码的高 4 位
                write7281(0x15,0x10+(key_number&0xf0)/16);
               // 在第 0 位上以 HEX 译码方式显示键码的低 4 位
write7281(0x15,key_number&0x0f);                }
            }
}
//=====================================================================
// No more.
//=====================================================================
```

保证键盘和数码管成功驱动的要点：

1） 首先要保证时序波形的正确。在本程序中，数据流程图如下：

发送字节程序源码如下：

```
unsigned char bc7281_send_1_byte(unsigned char send_byte,unsigned int overtime_in_us)
{
    // 发送几个时钟脉冲，建立握手信号。
    set_BC_CLK(0);
    set_BC_CLK(1);
    set_BC_CLK(0);
    set_BC_CLK(1);
    delay_gen_timer2_in_us(200); // BC7281 要求这个时钟脉冲宽度不小于 1us. 这里是 200us.
    // 然后等待 BC_DAT 变低。
    while(read_BC_DAT()==1) {;;} //读取 DAT 的状态线，当 DAT 为高时，继续读取状态。
```

```
    // 至此, BC7281 已响应. DAT 变低, 要再发一个 CLK, 等待 DAT 变高.
    set_BC_CLK(0);
    delay_gen_timer2_in_us(200);   // BC7281 要求这个时钟脉冲宽度不小于 1us. set_BC_CLK(1);
    delay_gen_timer2_in_us(200); // BC7281 要求这个时钟脉冲宽度不小于 1us.
    // 下面开始发送 8 个 bits.
    // 至此, DAT 变高. 可以开始发送 8bits.
    for (bit_counter = 0; bit_counter < 8; bit_counter++)
    {                   // 发送 8 个比特
        set_BC_CLK(1);                  // 输出一 clk 脉冲
        set_LED0(1);      // 这里加入了显示灯亮的函数。
        delay_gen_timer2_in_us(200);// BC7281 要求这个时钟脉冲宽度不小于 1us
        if ((send_byte & 0x80) == 0)
        {
            set_BC_DAT(0);      // 如果待发 bit 为 0, 置 dat 为 0
        }
        else
        {
            set_BC_DAT(1);      // 反之置为 1
         }
         send_byte = send_byte << 1;       // send_byte 左移一位

         set_BC_CLK(1);
         set_LED0(1);      // 这里加入了显示灯亮的函数，可以用来检测按键是否按下。
       delay_gen_timer2_in_us(200); //延迟
       }
    return 0x00;      // 已成功发送一个字节。
}
```

在上面的程序中，注意在下降沿时钟时后发送的数据。

2） 延时问题

在本开发板的所有程序中都采用了统一的延迟程序，为了能够调试成功和优化键盘和显示程序，必须做好延时的工作。

在本程序中，在数据传送期间，BC7281 不会进行显示和键盘扫描，因此如果通讯速度过慢（指令的传送时间>大于扫描周期），将会对显示造成影响。

上海胜诺通信培训项目

完全实战的开发学习环境，在实践中找到 DSP 学习的诀窍。

培训项目	培训目标	学时	收费标准
DSP 初级设计	掌握 DSP 开发工具、方法和常用模块的编程设计	10 天	2500
DSP 高级设计	DSP 在产品设计中最小系统搭建、安全性设计和编程设计	20 天	3500
单片机培训	单片机应用、编程和各接口的使用	10 天	2000 元
单片机设计	独立设计单片机硬件应用系统、软件编程及调试	一个月	4000 元
CPLD/FPGA 培训	单片机应用、编程（Verilog HDL）和硬件最小系统	10 天	2000 元
CPLD/FPGA 设计	独立设计 CPLD/FPGA 硬件应用系统、软件编程及调试	一个月	4000
PCB 培训	基础设计、EMI/EMC 安全性及高速 PCB 等	5 天	1200 元
产品设计	实际的通信产品设计、包括上面的所有知识点。	4~6 个月	9800 元
企业订单式培训	按照企业需求培训学员	不限	面谈

上海胜诺通信技术有限公司针对想从事电子产品设计嵌入式应用工作的专业人员，推出高级电子产品设计真实培训项目。研发内容涵盖电路图设计、PCB 设计、焊接、调试、芯片编程、PC 接口等，具体内容有：Protel99SE、TI DSP TMS320F2812、TI CCS、MCU STC89C51、TI MCU MSP430、ALTERA CPLD/FPGA、LCD、USB、TCP/IP、产品系统设计等。

学员学成后，如果学员还想继续锻炼，公司欢迎参与公司产品研发项目的部分工作。

注意：开通周六、周末班，特殊情况可具体面谈，欢迎报名参加。

主将老师：沈丹诚高工具有 15 年通信产品研发经验，如智能用户线测试仪广泛用于中国电信并出口多个国家。

联系电话： 021-64387320 54590581

Email： sinolab@163.com

QQ: 903977475

地 址： 上海市徐汇区虹桥路 333 号 415 室

（交大慧谷创业中心）

网址：www.sinolab.com/sh