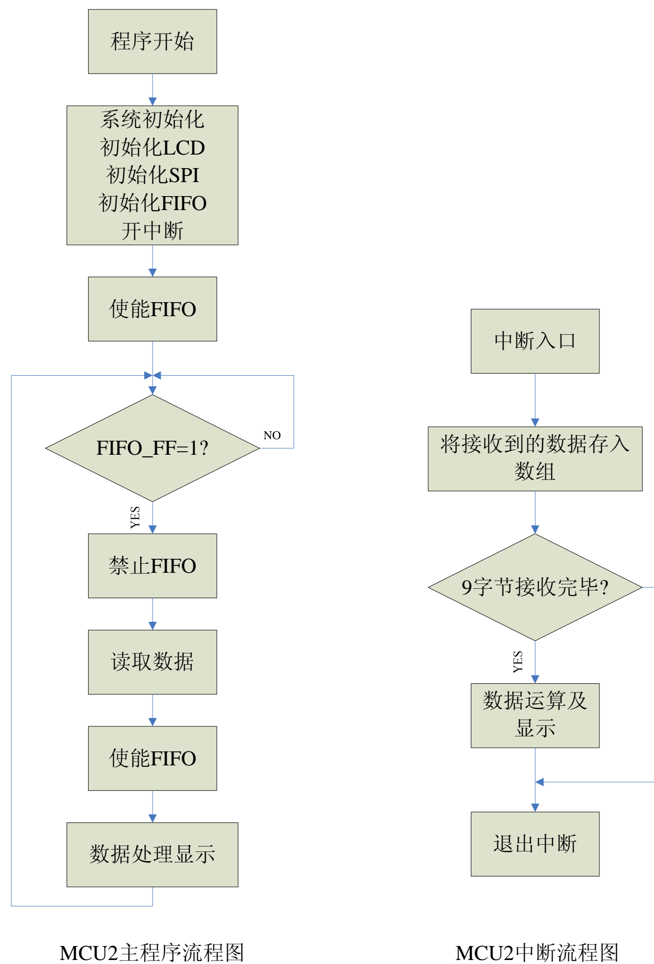
资料整理自互联网，版权归原作者！ 欢迎访问 www.XinShiLi.net 新势力单片机、嵌入式

给你的电路注入灵魂

——程序设计

没有程序那一堆堆电路起不到任何作用，完全是一堆废板子！就像一台没有操作系统的电脑一样，只能废电。程序设计是整个示波器制作中的难点，本文将详细讲解程序的设计。

该示波器中的程序全部是用 c 语言编写的，开发环境为 CodeVisionAVR C，原程序在附件中，下面就各个重要的子程序的设计一一叙述，其它程序见原程序。MCU2 与 MCU1 的程序流程图分别见图 1 和图 2。



已修订
09-01-07，21:49

图 1：MCU2 程序流程图

已修订
09-01-07，21:49

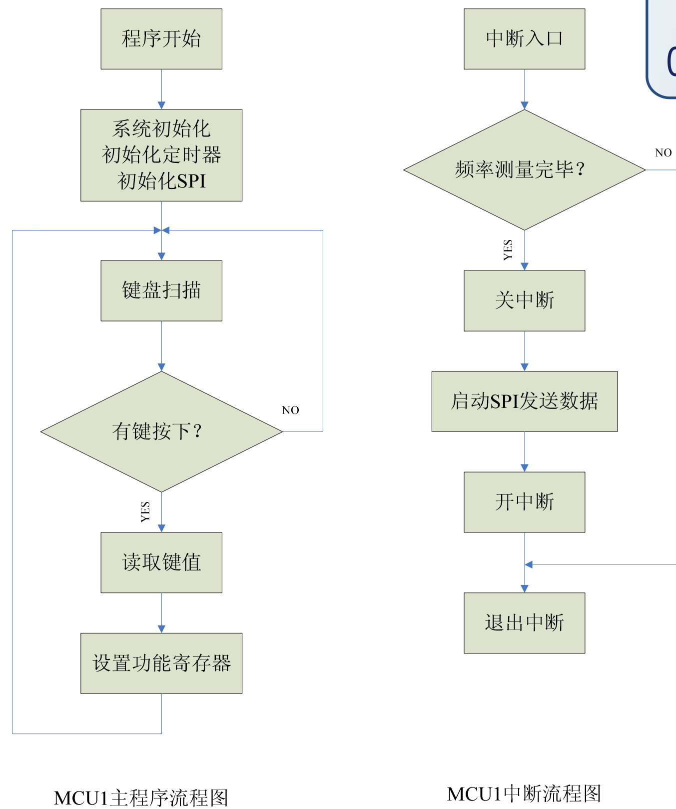


图 2：MCU1 程序流程图

1．同步触发的软实现

细心的朋友会发现这个示波器电路中缺少一部分电路，就是硬件触发电路，为了降低电路的复杂性我在做这个示波器时没有做这个电路，而是用软件实现同步触发的，这样做有个弊端就是几乎无法实现单次触发，因为我基本不用这个功能，需用这个功能的朋友只需在程控放大器部分加上一个由高速比较器构成的迟滞比较器然后将输出端接到一个外部中断的输入口即可。当然程序和电路就要做相应的变化，这里就不多讲了。软件触发的好处是触发条件更易调整，只需调整比较语句中的参数即可。保证可以用软件触发的条件是要有足够大的存储空间，显示一屏的数据为 240 个，但每次读进单片机的数据为 500 个，多余 260 个数据就是作为不满足触发条件的舍弃余量，为了以防万一，当从 500 个数据中以经读出超过 260 个数据但还没有符合触发条件的数据时，将跳出触发比较循环，重新从 FIFO 存储器中读出 500 个数据，因为 FIFO 存储器为 4K 容量，最多可以这样重复读取 8 次数据，所以软触发可以非常稳定的工作，在该示波器的 MCU2 中控制触发的语句见以下程序段：

```
read:
for(i=0;i<500;i++)      //从 FIFO 存储器中读 500 个数据
{
```

已修订
09-01-07，21:49

资料整理自互联网，版权归原作者！ 欢迎访问 www.XinShiLi.net 新势力单片机、嵌入式

```
    FIFO_R=0;
    add[i]=FIFO_bus;
    FIFO_R=1;
   }
   while(!(add[q]<=m&&add[q+1]>=m))     //满足幅度为 m 且为上升沿则触发
     {
     q+=1;
     if(q>=260)          //若存储数据不足则重新读数据
       goto read;
     }
```

程序的意思是只有当此时采样信号的数值是 m 且为上升沿时才可以触发，改变触发沿只需改变运算符，改变触发电压只需改变 m 的值即可，m 的取值范围是 0~255。

2. 从采样数据中测信号峰峰值

本示波器就能够测量输入信号电压的峰峰值，并显示在屏幕上。这个功能由峰峰值测量子程序完成，见下面的程序段。

在程序开始时给a中赋值128，即基线电压值。因为一屏幕的显示数据为240个，所以用for（）循环将if…else…判断语句执行240次，在a中存放最大值，在b中存放最小值。对每个数据进行比较，如果该数据比a大则将这个数据存入a，如果小于a，则将这个数据与b进行比较，比b大则抛弃，比b小则存入b。故当240个循环执行完后a中存放的是这一屏幕数据中的最大值，b中存放的是这一屏幕显示数据中的最小值。在比较完后用a减去b，得到差值存入c中，则c中保存的值就是电压的峰峰值，调用电压计算显示子程序根据当前的垂直灵敏度给c乘以不同的倍数，得到实际的峰峰值。当前垂直灵敏度的判断由一个switch（）选择结构完成。biao寄存器中的数据是当前的垂直灵敏度，case 4：后面没有运算是因为程控放大器在此状态下的放大倍数为1，即没有放大也没有衰减。

在计算完峰峰值后，设置LCD显示器，使其工作在文本模式（因为只有在文本模式下对字库的调用才有效），然后设置屏幕上显示电压峰峰的坐标（对该LCD模块的控制是先送命令，后送参数。例如设置X坐标“SdCmd(0x60);SdCmd(30);”中，第一个SdCmd（）送的0x60是设置X坐标的命令，第二个SdCmd（）送的30是X轴的坐标，其他设置相同。具体见光盘中LCD显示屏的资料。），在设置完LCD后约定显示格式，小数点后保留2为有效数字，显示单位为Vpp，显示完毕后需重新设置LCD工作状态使其工作在图形模式用于波形显示。

资料整理自互联网，版权归原作者！ 欢迎访问 www.XinShiLi.net 新势力单片机、嵌入式

```
a=128;
for(i=0;i<240;i++)        //取数据中的最大值与最小值
 {
     if(add[i]>a)
         {
          a=add[i];
         }
      else if(add[i]<b)
           {
            b=add[i];
           }
 }          //取得最大值存于a中，最小值存于b中
c=a-b;        //取差值存入c中
if(e>5)       //避免频繁换数据
    {
   disp_volt();    //调用电压值计算显示子程序
   e=0;
    }
    e++;
void disp_volt()    //电压值计算显示子程序
{
    c=c*0.667;
    switch(biao)//根据不同的垂直灵敏度计算峰峰值
           {
           case 0:c=c*25;break;
           case 1:c=c*10;break;
           case 2:c=c*5;break;
           case 3:c=c*2.5;break;
           case 4:;break;
           case 5:c=c*0.5;break;
```

已修订
09-01-07, 21:49

资料整理自互联网，版权归原作者！欢迎访问 www.XinShiLi.net 新势力单片机、嵌入式

```
            case 6:c=c*0.25;break;
            case 7:c=c*0.1;break;
            case 8:c=c*0.05;break;
            default:break;
            }
SdCmd(0x00);SdCmd(0xcd); //设置WLCR 寄存器，使LCD工作与显示文本状态
SdCmd(0xf1);SdCmd(0x1f); //字型水平、垂直方向各放大2 倍
SdCmd(0x60);SdCmd(30);  //设置显示X坐标
SdCmd(0x70);SdCmd(50);  //设置显示Y坐标
sprintf(lcd_buffer,"%3d.%02dVpp",c/100,c%100); //约定显示格式，小数点后保留两位
ShowText(lcd_buffer);                          //有效数字
SdCmd(0x00);SdCmd(0xc5); //设置WLCR 寄存器,使LCD工作于图形显示模式
}
```

3．将采样数据转换成显示数据

LCD显示屏为320×240点阵的图形显示模块，内置RA8803 控制器。模块不仅可以显示单一的文本、图形，而且可以实现双图层的（“或”、“异或”、“同或”、“与”四种逻辑关系)合成显示。在本示波器中显示格线与波形是在不同的层上显示，显示关系为“或”，画方格线的程序见原程序，比较简单就不多说了，着重解释一下如何将采样数据转换成显示数据。

显示屏的地址结构见图3，由图可知对显示数据的操作最小单位为字节，因为Mega32的内存为2K字节，显示波形的区域为240*240，显示一屏波形所需处理的数据为7.2K,故Mega32不可能同时处理一屏波形的全部数据，所以将一屏波形按字节分为30列，每次处理一列，处理完后直接显示，然后处理下一列。将AD转换所得的数据作为给LCD显示器写数据的列地址，因为一列数据位240字节，所以定义一个容量为240字节的数组lcd_buffer[240]，lcd_buffer[]在初始时数据全为00H，因为每次对数据的操作至少是一个字节，而每次处理数据处理的是所显示一个点，所以对每列数据处理8次，定义一个变量m，在一列数据处理之前将其赋值为m=10000000B,处理该列第1个点时让该点垂直地址所对应的数组中的数据（00H）与m相或并将结果存入数组，再将变量m右移一位，即m=01000000B。让第2点垂直地址所对应的数组中的数据与m相或并将结果存入数组，再将变量m右移一位，即m=00100000B ……，这样直到一列数据中的8个点全处理完，重新给m赋值为m=10000000B,

资料整理自互联网，版权归原作者！ 欢迎访问 www.XinShiLi.net 新势力单片机、嵌入式

已修订
09-01-07, 21:48

然后送显示。为了有较好显示效果，将显示相邻的点用线连接起来，在处理第一个点时预读出第二个点的垂直坐标，与第一个点的垂直坐标进行比较，如果比第一个点的垂直坐标小则从第一个点向第二个点拉线，如果比第一个点的垂直坐标大则从第二个点向第一个点拉线。具体程序如下所示：

```
for(j=0;j<30;j++)          //将一屏数据分为30列
    {
      m=0b10000000;          //
      for(i=j*8;i<(j+1)*8;i++)   //处理每列中的8个点
       {
        k=add[i]; // 读出采样数据作为垂直坐标
        lcd_buffer[k]=(lcd_buffer[k]|m); //让该坐标对应数据与m相或并原位保存
        if(add[i+q]<add[i+q+1])          //判断拉线方向
          {
           for(k=add[i+q];k<add[i+q+1];k++)
            {
            lcd_buffer[k]=(lcd_buffer[k]|m);
            }
          }
        else
          {
           for(k=add[i+q];k>add[i+q+1];k--)
            {
            lcd_buffer[k]=(lcd_buffer[k]|m);
            }
          }
           m>>=1;          //将m的值右移一位
         }

      for(h=0;h<240;h++)      //送显示

       {

          SdCmd(0x60);SdCmd(j);           //设置显示X坐标

          SdCmd(0x70);SdCmd(h);           //设置显示Y坐标

          SdData(lcd_buffer[h]);          //传送显示数据

          lcd_buffer[h]=0;                //将已送出数据的存储器单元清零

        }

  }
```

资料整理自互联网，版权归原作者！欢迎访问 www.XinShiLi.net 新势力单片机、嵌入式

图 3：LCD 显示器地址结构图

4．用MCU1频率测量

用Mega8测量频率使用了其中的两个计数器/定时器。设置TCCR1B=6使16位计数器/定时器T/C1工作在计数器方式，对外部T1（PD5）引脚输入的脉冲信号进行计数（下降沿触发）。设置TCCR2=15使T/C2工作在CTC模式，内部时钟1024分频（16M/1024=15.625KHz），设置OCR2=124使中断时间为(124+1)/15.625=8ms，在低水平扫速时每隔8ms中断一次，在高水平扫速时通过重新设置TCCR2=14则每隔2ms中断一次，在这里以低水平扫速时为例，每次T/C2的中断中都首先记录下T/C1寄存器TCNT1当前的计数值，因此前后两次寄存器TCNT1的差值（time1_new-time1_old）或（65536-time1_old+time1_new）就是8ms时间内T1引脚输入的脉冲个数，为了提高测量精度程序对125个8ms内的脉冲个数进行累计，将累计值存入变量freq中，即可知限定时间为1s内有多少个脉冲，这样就将T1脚上的脉冲频率测量出来了，而T1脚上的频率是经过4分频后的所以真正的频率是将测量的频率的4倍。具体程序如下所示：

```
interrupt [TIM2_COMP] void timer2_comp_isr(void)
{
    time1_new = TCNT1;          // 8ms到，记录当前T/C1的计数值
    time_8ms_ok = 1;
```

已修订
09-01-07, 21:48

资料整理自互联网，版权归原作者！欢迎访问 www.XinShiLi.net 新势力单片机、嵌入式

```
}
void time()
{
 if (time_8ms_ok)
        {   // 累计T/C1的计数值
            if (time1_new >= time1_old) freq = freq + (time1_new - time1_old);
            else freq = freq + (65536 - time1_old + time1_new);
            time1_old = time1_new;
            if (++freq_time >=125)
            {
                freq_time = 0;  // 1s到,
                freq_to_spibuff();// 将1s内的脉冲数送计算并通过SPI发送
                freq = 0;
            }
                time_8ms_ok = 0;
        }
}
```

5．将两个单片机联系起来

将两个单片机联系起来就是实现两个单片机之间的通信，在这里实际就是让 MCU1 控制 MCU2,为了完成这一功能所以使用 SPI 通信。

首先介绍一下 SPI 的通信协议：

SPI（串行外设接口）总线系统是一种同步串行外设接口，允许 MCU 与各种外围设备以串行方式进行通信、数据交换，广泛应用于各种工业控制领域。基于此标准，SPI 系统可以直接于各个厂家生产的多种标准外围器件直接接口。SPI 接口通常包含有 4 根线：串行时钟（SCK）、主机输入/从机输出数据线（MISO）、主机输出/从机输入数据线（MOSI）和低电平有效的从机选择线 SS。在从机选择线 SS 使能的前提下，主机的 SCK 脉冲将在数据线上传输主/从机的串行数据。主/从机的典型连接图如图 4 所示：

资料整理自互联网，版权归原作者！ 欢迎访问 www.XinShiLi.net 新势力单片机、嵌入式

图 4：主/从机的连接图

串行外设接口 SPI 允许 AVR 单片机和外设之间进行高速的同步数据传输。AVR 单片机 SPI 的特点如下：全双工，3 线同步数据传输，主/从机操作，LSB 首先发送或 MSB 首先发送，7 种可编程的比特率，传送中断结束，写碰撞标志检测，可以从闲置模式唤醒。SPI 主机-从机的互连如图 5 所示，系统包括两个移位寄存器和一个主时钟发生器。通过将需要的从机的 SS 引脚拉低，主机启动一次通信过程。主机和从机将需要的数据放到相应的移位寄存器，主机在 SCK 引脚上产生时钟脉冲以交换数据。主机的数据从 MOSI 移出，从从机 MISO 移入。从机的数据从 MISO 移出，从从机 MOSI 移入。主机通过将从机的 SS 拉高实现与从机的同步。

图 5：SPI 主机-从机的互连

下面将介绍 SPI 的几个特殊寄存器：

（1）SPI 的控制寄存器—SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

资料整理自互联网，版权归原作者！欢迎访问 www.XinShiLi.net 新势力单片机、嵌入式

SPIE 为 SPI 中断使能，置位后，只要 SPSR 寄存器的 SPIF 和 SREG 寄存器的全局中断使能位置位，就会引发 SPI 中断。SPE 置位将使能 SPI，DORD 置位时数据的 LSB 首先发送；否则数据的 MSB 首先发送。MSTR 置位时选择主机模式，否则为从机。CPOL 置位表示空闲 SCK 为高电平；否则空闲时 SCK 为低电平。CPHA 决定数据是在 SCK 的起始沿采样还是在 SCK 的结束沿采样。通过对 SPR1、SPR0 进行设计，确定主机的 SCK 速率。

（2）SPI 的状态寄存器—SPSR

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
读/写	R	R	R	R	R	R	R	R/W	
初始值	0	0	0	0	0	0	0	0	

SPIF 为中断标志位，串行发送结束后，SPIF 置位。若此时寄存器 SPCR 的 SPIE 和全局中断使能位置位，SPI 中断即产生。进入中断例程后 SPIF 将自动清零。在发送当中对 SPI 数据寄存器 SPDR 写数据将置位 WCOL，SPI2X 置位后 SPI 的速度加倍。

（3） SPI 的数据寄存器—SPDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
读/写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	X	X	X	X	X	X	X	X	Undefined

SPDR 数据寄存器为读/写寄存器，用来在寄存器文件 SPI 移位寄存器之间传输数据。写寄存器将启动数据传输，读寄存器将读取寄存器的接收缓冲器。SPI 系统的发送方向只有一个缓冲器，而在接收方向有两个缓冲器。也就是说，在发送时一定要等到移位过程全部结束后才能对 SPI 数据寄存器执行写操作。而在接收数据时，需要在下一个字符移位过程结束之前通过访问 SPI 数据寄存器读取当前接收到的字符。否则第一个字节将丢失。

本示波器中只用 MCU1 控制 MCU2，所以 MCU1 只用于发送控制数据，而 MCU2 只用于接收控制数据，所以将 MCU1 配制成 SPI 主机，将 MCU2 配制成 SPI 从机即可。在实际的程序设计中由于 MCU1 启动 SPI 通信是在中断服务程序中完成，所以在执行完后相应寄存器会被清零，导致数据错误，所以 MCU1 并没有使用其中的 SPI 控制器，而是使用一个子程序模拟 SPI 通信，解决了控制寄存器被清零的问题。MCU2 则使用了本身的 SPI 控制器进行数据接收。具体程序见以下程序段：

资料整理自互联网，版权归原作者！ 欢迎访问 www.XinShiLi.net 新势力单片机、嵌入式

（1）MCU1 模拟 SPI 主机程序段：

spi_out（）为 SPI 发送子程序，带有参数 j，即 j 为要发送的数据，发送数据时先拉低 ss，让从机开始接收数据，然后用 for（）循环将数据按由左至右的顺序（即高位先发送）发送给从机，具体方法是将 j 与 0b10000000 相与，屏蔽低 7 位，是 1 则将 dat 拉高，否则置低，然后拉高 clk，延时 1us 再置低 clk，模拟时钟信号，再将 j 左移一位，再与 0b10000000 相与，然后判断发送……直到 8 位数据发送完毕，拉高 ss 告诉从机数据发送完毕进行数据存储。发送数据时约定数据格式，即两个单片机之间的通信协议：每次发送 9 个字节，前 4 个字节是测得的频率数据，且高位在前；第 5 个字节为垂置灵敏度数据；第 6 个字节为触发控制数据；第 7 个字节为同步控制数据；第 8 个字节为水平扫速数据；第 9 个字节为功能复用键的当前功能标志。从机再接收到数据后按照这样的顺序对数据进行处理，实现相应的功能。spi_out()这个子程序还可以用于其他需要 SPI 控制的芯片，只需在调用前对 IO 口进行定义即可。

已修订 09-01-07，21:48

```
spi_out(unsigned char j)
{
  unsigned char u;
  ss=0;
  for(u=0;u<8;u++)
      {
        if(j&0b10000000) {  dat=1; }
        else {dat=0;}
        delay_us(1);
        clk=1;
        delay_us(1);
        clk=0;
        delay_us(1);
        j<<=1;
      }
```

```
        delay_us(1);
    ss=1;
}
void freq_to_disbuff()
{               if(fr==0)
             {
             freq=freq*4;
             }
              eep=freq>>24;//取频率高 8 位
              spi_out(eep);
              delay_us(10);
              eep=(freq>>16)&0xff;
              spi_out(eep);
              delay_us(10);
              eep=(freq>>8)&0xff;
              spi_out(eep);
              delay_us(10);
              eep=freq&0xff;//取频率低 8 位
              spi_out(eep);
              delay_us(10);
              spi_out(w[i]);  //垂直灵敏度数据
              delay_us(10);
              spi_out(tri);   //触发数据
              delay_us(10);
              spi_out(hold);  //同步数据
              delay_us(10);
              spi_out(kr);    //扫速数据
```

```
delay_us(10);
spi_out(zhi);   //复用键功能标志数据
delay_us(10);
}
```

（2）MCU2 从机 SPI 程序段：

init_spi()函数是将 MCU2 配制成 SPI 从机，每接收一个字节的数据中断一次，中断服务程序中将接收到的数据存入数组，并将数组地址加 1，然后判断 9 个字节是否接收完毕，若没接收完则继续等待接收，接收完后则将数据按约定格式处理显示。大家可以根据自己的需求改变这些格式为其增加新的功能。

```
void init_spi()     //SPI 初始化子函数
{
DDRB.7=0;
PORTB.7=1;
DDRB.5=0;
PORTB.5=1;
DDRB.4=0;
PORTB.4=1;              //将 SPI 端口设置成输入
SPCR=0b11000101;   //设置 SPI 为从机
SPSR=0X00;              //清零 SPSR 寄存器
}
interrupt[SPI_STC] void spi_isr(void)     //SPI 接收数据中断
{
 if(j!=9)
 {
     i[j]=SPDR;        //将接收到的数据存入数组
     j++;                  //给数组地址加 1 }
```

资料整理自互联网，版权归原作者！ 欢迎访问 www.XinShiLi.net 新势力单片机、嵌入式

```
}
        if(j>=9)          //判断 9 个字节数据是否接受完毕
    {   eep=i[0];
    freq=eep;
    freq=freq<<8;
    eep=i[1];
    freq=freq|eep;
    freq=freq<<8;
    eep=i[2];
    freq=freq|eep;
    freq=freq<<8;
    eep=i[3];
    freq=freq|eep;      //将频率值整和到 freq 寄存器
    disp_freq();         //显示频率值
    eep=i[4];
    disp_volt();         //显示垂直灵敏度
    tri=i[5];            //显示触发方式
    hold=i[6];           //显示同步
    biao=i[7];           //显示扫速
    disp_time();
    zhi=i[8];         //显示复用键当前功能
    disp_cond();
    j=0;}
```

已修订
09-01-07，21:48

魏坤

2008 11 15