8051单片机 C语言 彻底应用

赖麒文 编著

特色

- ◆突破艰涩难懂的汇编语言编写，改用易学易懂的C语言
- ◆由浅入深介绍程序结构、设计技巧与理念
- ◆循序渐进扩展程序设计的逻辑推理与思考方法
- ◆参考范例设计或直接选取实例函数套用到实际开发中
- ◆从基础、硬件电路设计和软件编程方法彻底剖析实用的商品化程序

科学出版社 文魁资讯股份有限公司

8051 单片机

C 语言彻底应用

赖麒文　编著

科学出版社

2002

内 容 简 介

本书介绍 8051 单片机 C 语言结合硬件编程应用的工程方法。本书通过一个个实用的例子分析，讲解了 C 语言实现自动控制和界面的设计方法、技巧以及常见问题剖析。

本书适合 8051 单片机应用设计人员参考。

图书在版编目(CIP)数据

8051 单片机 C 语言彻底应用/赖麒文编著. -北京:科学出版社, 2002

ISBN 7-03-009054-3

Ⅰ.8… Ⅱ.赖… Ⅲ.单片微型计算机, 8051-C 语言-程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2001）第 074091 号

图字：01-2001-2190 号

科 学 出 版 社 出版

北京东黄城根北街16号

邮政编码:100717

http://www.sciencep.com

新 蕾 印 刷 厂 印刷

科学出版社发行 各地新华书店经销

*

2002 年 1 月第 一 版 开本：710×1000 1/16

2002 年 1 月第一次印刷 印张：32

印数：1—5 000 字数：628 000

定价：42.00 元

（如果有印装质量问题，我社负责调换（环伟））

序

现在，市场上的 8051 单片机开发类书籍讲述的多是汇编语言，有关 C 语言的书也是琳琅满目。但是，像本书这样结合硬件讲述用 C 语言开发单片机的应用图书却不多见。本书以 8051 单片机为例，对 C 语言在单片机开发应用相关的程序和电路设计进行了深入浅出的讲述。

8051 单片机广泛应用于键盘、电话机、显示器、电冰箱、洗衣机、空调等等电器中。本书提供了来自产品的现成实例，读者可以把本书作为 8051 软硬件设计的工具书。本书中的这些电路和函数可以非常简便地应用在商品化软硬件开发过程中。也许书中的某些范例程序对你而言一时还派不上用场，但也请你仔细阅读，了解到底有哪些函数、用途是什么以及程序的设计理念。

在本书提供的大量函数中，部分函数的功能和目的是相同的，罗列程序的不同写法并对重要部分一再重复说明，是希望使读者更明了地理解函数，本书的宗旨是提供一些灵感启发读者活用 C 语言。我们希望读者在设计某种功能的程序时，可以方便地参考本书，快速了解程序设计技巧和观念。无论是吸收书中的方法直接套用某个函数,还是将书中的范例融会贯通、举一反三。这样对提高自己的编程水平会有很大帮助。本书附录中给出了书中所有 C 语言范例的源程序编译成的 8051 汇编代码，以方便使用不同类型计算机的读者参考。

目　录

第 1 章　C 语言基本概念

1-1　程序的初步

/* -------*/: C 程序的注释

➢ void main(void)

C 程序从 main 开始执行，前一个 void，表示无返回值；后一个 void，表示不传参数。

int i　;

int：声明整数类型

i ：变量名

; ：语句结束符号

表 1-1　数据类型的长度

数据类型	位数	字节数	值　域
bit	1		0 ~ 1
signed char	8	1	-128 ~ +127
unsigned char	8	1	0 ~ 255
enum	16	2	-32768 ~ +32767
signed short	16	2	-32768 ~ +32767
unsigned short	16	2	0 ~ 65535
signed int	16	2	-32768 ~ +32767
unsigned int	16	2	0 ~ 65535
signed long	32	4	-2147483648 ~ 2147483647
unsigned long	32	4	0 ~ 4294967295
float	32	4	0.175494E-38 ~0 .402823E+38
sbit	1		0 ~ 1
sfr	8	1	0 ~ 255
sfr16	16	2	0 ~ 65535

1-2 C 语言的运算符

表 1-2 C 语言运算

运算符	范 例	说 明
+	a+b	a 变量值和 b 变量值相加
-	a-b	a 变量值和 b 变量值相减
*	a*b	a 变量值乘以 b 变量值
/	a/b	a 变量值除以 b 变量值
%	a%b	取 a 变量值除以 b 变量值的余数
=	a=6	将 6 设定给 a 变量，即 a 变量值等于 6
+=	a+=b	等同于 a=a+b，将 a 和 b 相加的结果又存回 a
-=	a-=b	等同于 a=a-b，将 a 和 b 相减的结果又存回 a
*=	a*=b	等同于 a=a*b，将 a 和 b 相乘的结果又存回 a
/=	a/=b	等同于 a=a/b，将 a 和 b 相除的结果又存回 a
%=	a%=b	等同于 a=a%b，a 变量值除以 b 变量值的余数又存回 a
++	a++	a 的值加 1，即 a=a+1
--	a--	a 的值减 1，即 a=a-1
>	a>b	测试 a 是否大于 b
<	a<b	测试 a 是否小于 b
==	a==b	测试 a 是否等于 b
>=	a>=b	测试 a 是否大于或等于 b
<=	a>=b	测试 a 是否小于或等于 b
!=	a!=b	测试 a 是否不等于 b
&&	a && b	a 和 b 作逻辑 AND，两个变量是都是“真”，结果才为“真”否则结果为“0”
||	a||b	a 和 b 作逻辑 OR，只要有任何一个变量为“真”，结果就为“真”
!	!a	将 a 变量的值取反，即原来为“真”则变“假”，为“假”则变为“真”
>>	a>>b	将 a 按位右移 b 个位
<<	a<<b	将 a 按位左移 b 个位，右侧补“0”
|	a|b	a 和 b 的按位做 OR 运算
&	a & b	a 和 b 的按位做 AND 运算
^	a ^ b	a 和 b 的按位做 XOR 运算
~	~a	将 a 的每一位取反
&	a=&b	将 b 变量的地址存入 a 寄存器
*	*a	用来取寄存器所指地址内的值

注意：在逻辑运算中，凡是结果为非“0”的数值即为真，等于“0”为假。

◆ **范例一**

```
a=1;
b=++a;
```

其运算过程是 a 值加 1 变为 2，然后再将 2 赋值给 b，所以 b=2，a=2。

◆ **范例二**

```
a=1;
b=a++;
```

其运算过程是 a 原先的值 1，先赋值给 b，然后 a 再加 1 变为 2，所以 b=1，a=2。

1-3　C 程序的流程控制

➢ if 语句

1.　if　(条件表达式)
　　{　　动作　　}

如果条件表达式的值为真(非零的数)，则执行{ }内的动作，如果条件表达式为假，则略过该动作而继续往下执行。

◆ **范例**

```
01 void IfDemo1(void)
02 {
03    Byte  i,j;
04
05    if ( DisplayState<10 )
06
07      for(i=0; i<5; i++)
08      {
09        j=5 * DisplayState;
10        P2=DISPLAY_TABLE10[j+i];
11        P1=0x01 << i;
12        DelayX1ms(3);
13      }
```

```
14     }
15 }
```

2. if (条件表达式)
   { 动作 1 }
   else
   { 动作 2 }

如果条件表达式为真，则执行动作 1，略过 else 的部分，接着往下执行，如果条件表达式为假，则略过 if 的部分而执行 else 的动作 2，然后再往下执行。

◆ 范例

```
01 void IfDemo2(void)
02 {
03     if (LedCount<26)
04       LedOn( );
05     else
06       LedOff( );
07 }
```

3. if (条件表达式 1)
   if (条件表达式 2)
   if (条件表达式 3)
   { 动作 A }
   else
   { 动作 B }
   else
   { 动作 C }
   else
   { 动作 D }

动作 A：是条件表达式 1、2、3 都成立时才会执行。
动作 B：是条件表达式 1、2 成立，但条件表达式 3 不成立时才会执行。
动作 C：是条件表达式 1 成立，条件表达式 2 不成立时才会执行。
动作 D：是条件表达式 1 不成立时才会执行。

◆ 范例

```
01 void IfDemo3(void)
02 {
```

```
03     if ( FgPulse==0 )
04     {
05       if ( P1_7==0 )
06       {
07         FgPulse=1;
08         LedOn( );
09       }
10       else
11       {
12         FgPulse=0;
13         LedOff( );
14       }
15     }
16     else
17     {
18       if ( P1_7!=0 )
19         FgPulse=0;
20     }
21 }
```

4. if　　(条件表达式 1)
　　{　　动作 A　　}
　else if　(条件表达式 2)
　　{　　动作 B　　}
　else if　(条件表达式 3)
　　{　　动作 C　　}
　else
　　{　　动作 D　　}

动作　A：是条件表达式 1 成立时立即执行。
动作　B：是条件表达式 1 不成立，但条件表达式 2 成立时才会执行。
动作　C：是条件表达式 1、2 不成立，条件表达式 3 成立时才会执行。
动作　D：是条件表达式 1、2、3 都不成立时才会执行。

◆ **范例**

```
01 void IfDemo4(void)
02 {
```

```
03      if (LedCount<6)
04        LedOn( );
05      else if (LedCount<11)
06        LedOff( );
07      else if (LedCount<16)
08        LedOn( );
09      else if (LedCount<68)
10        LedOff( );
11      else
12        LedCount=0;
13  }
```

➢ switch-case 语句

```
switch   (条件表达式)
{
case 条件值 1:
        动作 1
        break;
case 条件值 2:
        动作 2
        break;
case 条件值 3:
        动作 3
        break;
default:
        动作 4
        break;
}
```

switch 内的条件表达式的结果必须为整数或字符。switch 以条件表达式的值来与各 case 的条件值对比，如果与某个条件值相符合，则执行该 case 的动作，如果所有的条件值都不符合，则执行 default 的动作，每一个动作之后一定要写 break，否则会有错误。另外 case 之后的条件值必须是数据常数，不能是变量，而且不可以重复，即条件值必须各不同，如果有数种 case 所做的动作一样时，也可以写在一起，即上下并列。一般当程序必须作多选 1 时，可以采用 switch

语句。

break：是跳出循环的指令，任何由 switch、for、while、do-while 构成的循环，都可以用 break 来跳出，必须注意的是 break 一次只能跳出一层循环，通常都和 if 连用，当某些条件成立后就跳出循环。

default：当所有 case 的条件值都不成立时，就执行 default 所指定的动作，作完后也要使用 break 指令跳出 switch 循环。

◆ **范例**

```
01 void SwitchDemo(void)
02 {
03     switch( OutputState )
04     {
05      case 1 :
06         P1_0=0;
07         P1_1=0;
08         P1_2=0;
09         break;
10      case 2 :
11         P1_0=1;
12         P1_1=0;
13         P1_2=0;
14         break;
15      default :
16         break;
17     }
18 }
```

➢ while 循环语句

while　(条件表达式)
　　{　　动作　　}

先测试条件表达式是否成立，当条件表达式为真时，则执行循环内动作，做完后又继续跳回条件表达式作测试，如此反复直到条件表达式为假为止，使用时要避免条件永真，造成死循环。

◆ 范例

do-while 循环语句

```
01 void WhileDemo(void)
02 {
03     FgPulseShort=0;
04     FgPulseLong=0;
05     DutyCount=0;
06
07     while ( P1_0==0 )          //先测试条件表达式，成立后再执行动作
08     {
09       DelayX1ms(1);
10       if ( P1_0==0 )
11         DutyCount++;
12     }
13
14     if ( ((0+3)<DutyCount) && (DutyCount<(10+3)) )
15       FgPulseShort=1;
16     else if (((10+3)<DutyCount) && (DutyCount<(20+3)))
17       FgPulseLong=1;
18 }
```

➢ do-while 循环语句

do { 动作 }
while (条件表达式);

先执行动作后，再测试条件表达式是否成立。当条件表达式为真，则继续回到前面执行动作，如此反复直到条件表达式为假为止，不论条件表达式的结果为何，至少会做一次动作，使用时要避免条件永真，造成死循环。

◆ 范例

```
01 void  DowhileDemo(void)
02 {
03     Byte  i=0,j,k;
04     Byte  SoundLong,SoundTone;
05     Word  m;
```

```
06
07    do            //先执行动作后
08
09       SoundLong=MUSIC_SOUNDLONG1[i];
10       SoundTone=MUSIC_SOUNDTONE1[i];
11       i++;
12
13       for(j=0; j<SoundLong; j++)    //sound long
14
15          for(k=0; k<12; k++)
16          {
17             for(m=0; m<SoundTone*1; m++)
18                P1_0 = 0;
19             for(m=0; m<SoundTone*1; m++)
20                P1_0 = 1;
21          }
22       }
23       Delay50uS(6);
24
25    } while ( MUSIC_SOUNDTONE1[i]!=0x00 ); //最后再测试条件表达式
                                              是否成立
26 }
```

➢ for 循环语句

for (表达式 1；表达式 2；表达式 3)

{　　动作　　}

表达式 1：通常是设定起始值。

表达式 2：通常是条件判断式，如果条件为真时，则执行动作，否则终止循环。

表达式 3：通常是步长表达式，执行动作完毕后，必须再回到这里做运算，然后再到表达式 2 做判断。

◆ 范例

```
01 void ForDemo(Byte ontime,Byte offtime)
02 {
03    Byte  i;
04
05    for(i=0; i<3; i++)
06    {
07       LedOn( );
08       DelayX10ms(ontime );
09       LedOff( );
10       DelayX10ms(offtime);
11    }
12    i++;
13 }
```

◆ 说明

从 i=0 开始执行，当 i<3 时(即表达式 2 条件成立)，则执行括号内(行号 06~11)的动作，否则跳出循环，继续执行。当动作执行完毕后，即执行到行号 11，必须回到表达式 3 作运算，i 加 1，然后又回到表达式 2 作判断是否成立，如此反复直到表达式 2 不成立为止。

continue：当程序执行 for 或 while 时，则可使用 continue 叙述来略过该循环，使得程序跳到下一次循环的开始，但并不跳出循环，只是忽略了 continue 后面的语句。

goto 标签

编写程序，尽量不要使用 goto 语句，以避免程序阅读困难。但是，break 一次只能跳离一个循环，如果需要跳离很多循环，则可使用 goto 语句，但 goto 的目的位置必须在同一个程序文件内，不能跳到其它程序文件，标签的写法和变量是一样的，标签后面必须加一个冒号。goto 经常和 if 连用，如果程序中检查到异常时，即使用 goto 语句去处理。

◆ 范例

```
01 void GotoDemo(void)
02 {
```

```
03      LedOn( );
04      DelayX10ms(50);
05
06      if (FgError) goto ERROR;
07
08      LedOff( );
09      DelayX10ms(50);
10  }
11
12  void Function(void)
13  {
14      Byte i;
15
16  ERROR:
17      i++;
18  }
```

行号 06 是不合法的 goto 叙述，因 goto 叙述不能跳到其它子程序上。

行号 16 ERROR 是标名，在最后面要加冒号。

➢ 函数(FUNCTION)

类型　函数名称 (类型　参数 1,类型　参数 2,类型　参数 3,....)

所谓函数，即子程序，也就是“语句的集合”，就是说把经常使用的语句群定义成函数，在程序中用到时调用，这样就可以减少重复编写程序的麻烦，也可以减少程序的长度。当一个程序太大时，建议将其中的一部分程序改用函数的方式调用较好，因为大程序过于繁杂容易出错，而小程序容易调试，也易于阅读和修改。

➢ 使用函数的注意事项

1. 函数定义时要同时声明其类型。
2. 调用函数前要先声明该函数。
3. 传给函数的参数值，其类型要与函数原定义一致。
4. 接受函数返回值的变量，其类型也要与函数一致。

➢ 函数的声明

```
void function1(void)
```

此函数无返回值，也不传参数。

```
void function2(unsign char i,int j)
```

此函数无返回值，但需要 unsign char 类型的 i 参数和 int 类型的 j 参数。

```
unsign char function3(unsign char i)
```

此函数有 unsign char 类型的返回值给原调用程序。

➢ 函数的返回值

```
return
```

return 是用来使函数立即结束以返回原调用程序的指令，而且可以把函数内的最后结果数据传回给原调用程序。

◆ 范例一

```
01 void FunctionDemo1(void)    //原调用程序
02 {
03     unsign int  i,j;
04
05     LedOn( );
06     for(i=0; i<500; i++)
07       for(j=0; j<120; j++)
08           ;
09     LedOff( );
10     for(i=0; i<500; i++)
11       for(j=0; j<120; j++)
12           ;
13 }
14
15 void LedOn(void)          //函数 LedOn，无返回值，也不传参数
16 {
17     P1_0=1;
18 }
```

```
19 void LedOff(void)        //函数 LedOff，无返回值，也不传参数
20 {
21     P1_0=0;
22 }
```

◆ 范例二

```
01 void FunctionDemo2(void)          //原调用程序
02 {
03     LedOn( );
04     DelayX1ms(500);
05     LedOff( );
06     DelayX1ms(500);
07 }
08
09 void DelayX1ms(unsign int count)   //函数 DelayX1ms，无返回值，但是
10                                    //传了一个 unsign int 类型的参数 count
11 {
12     unsign int  i,j;
13
14     for(i=0; i<count; i++)
15       for(j=0; j<120; j++)
16         ;
17 }
```

◆ 范例三

```
01 void FunctionDemo3(unsign int addr)
02 {
03     unsign int  i,temp=0x00;   //初始值等于 0x00
04
05     for( i=0; i<127; i++)          //total 127 bytes
06       temp+=EepromByteRead( i );     //temp=temp+返回值，作 127 次
07     Checksum =(~temp)+1;          //Checksum =0xff-(total
                                     sum)+1
08 }
09
```

```
10 unsign char EepromByteRead(unsign int addr)//会有unsign char类型的返回值
11 {
12     EepromRead( addr,1 );
13     return (TrmBuf[0]);
14 }
```

◆ **范例四**

```
01 void FunctionDemo4(void)          //原调用程序
02 {
03     GetKey2( );              //按键检查函数，在其后说明，
04                                 //此处并未列出
05     KeyMgr( );
06 }
07
08 void KeyMgr(void)
09 {
10     if ( (FgKEY3==0) && FgKEY2_ONOFF )
11     {
12       FgKEY3 = 1;
13       Beep4(2,17,10);
14       return;          //立即返回原调用程序，不执行行号16以后语句
15     }
16     if ( FgKEY3==0 )
17     {
18       FgKEY3 = 1;
19       Beep4(1,17,10);
20     }
21 }
```

➢ #define

#define 宏名 字符串

以一个宏名称来代表一个字符串，即当程序任何地方使用到宏名称时，则将以所代表的字符串来替换。宏的定义可以是一个常数、表达式，或含有参数的表达式都可，在程序中如果多次使用宏，则会占用较多的内存，但执行速度较快。

◆ 范例

```
01 #define  DATA      data
02 #define  IDATA     idata
03 #define  PDATA     pdata
04 #define  XDATA     xdata
05 #define  RDATA     code
06
07 #define    CLOCK_BASE   1
08 #define    CLOCK_40MS        (65536 - 40000 * CLOCK_BASE)
09 #define    CLOCK_4096us      (65536 - 4096 * CLOCK_BASE)
10 #define    CLOCK_5000us      (65536 - 5000 * CLOCK_BASE)
11 #define    CLOCK_10000us     (65536 - 10000 * CLOCK_BASE)
12
13 #define    HIBYTE_REF(addr) (* ((Byte *) & addr) )
14 #define    LOBYTE_REF(addr) (* ((Byte *) & addr + 1) )
15 #define    MAKEWORD(a,b)     ( ( ((Word)(a))<<8 ) + (Word)(b) )
16 #define    HIBYTE(a)         ( (Byte) ( (a) >>8    ) )
17 #define    LOBYTE(a)         ( (Byte) ( (a) & 0xff ) )
```

➢ 条件编译

#if 表达式
#else
#endif

如果表达式成立，则编译 #if 下的程序，否则编译 #else 下的程序，#endif 为结束条件表达式编译。

#ifdef　宏名　;如果宏名称已被定义过，则编译以下的程序
#ifndef　宏名　;如果宏名称未被定义过，则编译以下的程序

条件表达式编译通常用来调试，保留程序(但不编译)，或者有两种状况而须做不同处理的程序编写时使用，如以下所示。

◆ 范例一

```
01 #if     0   //永远不成立，行号 02~行号 05，只是保留程序，但不编译
02    if ( FgAudioFlag==1 )
```

```
03          LedOn( );
04      else
05          LedOff( );
06  #endif
```

◆ 范例二

```
01  #if     0    //#if ---#else -----的应用
02      P1 = P1 | 0xfc;
03      keyvalue = P1 & 0xfc;
04  #else
05      P1 = P1 | 0x7c;
06      keyvalue = P1 & 0x7c;
07  #endif
```

◆ 范例三

```
01  #if     RED_BLUE     //if RED_BLUE 未定义为零，则编译行号03~行号05的
02                          语句，否则，做行号07~行号09语句
03      BgainValue = TrmBuf[0];
04      GgainValue = TrmBuf[1];
05      RgainValue = TrmBuf[2];
06  #else
07      RgainValue = TrmBuf[0];
08      GgainValue = TrmBuf[1];
09      BgainValue = TrmBuf[2];
10  #endif
```

◆ 范例四

```
//如果 _DECLARE_H 未被定义过则开始定义行号02~行号11
01  #ifndef    _DECLARE_H
02  #define    _DECLARE_H
03  #define SUCCESS        0
04  #define FAILURE        1
05  #define FALSE          (Bool)0
06  #define TRUE           (Bool)!FALSE
07  #define TIME_BASE      40
```

```
08 #define TIME_10SEC      (10000/TIME_BASE)
09 #define TIME_3SEC       ( 3000 /TIME_BASE)
10 #define TIME_2SEC       ( 2000 /TIME_BASE)
11 #define TIME_1SEC       ( 1000 /TIME_BASE)
12 #endif
```

➢ typedef

typedef：可以用来定制自己的类型名称，如下所示。

typedef bit	Bit;	可以用 Bit 作为数据类型名作变量声明
typedef bit	Bool;	可以用 Bool 作为数据类型名作变量声明
typedef unsigned char	Byte;	可以用 Byte 作为数据类型名作变量声明
typedef unsigned int	Word;	可以用 Word 作为数据类型名作变量声明
typedef unsigned long	Long;	可以用 Long 作为数据类型名作变量声明

◆ 范例一

```
01 void TypedefDemo1(void)
02 {
03    Bool  i;
04
05    i=0;
06    j=~i;
07 }
```

◆ 范例二

```
01 void TypedefDemo2(void)
02 {
03    Byte  i=0x00;
04
05    i++;
06 }
```

◆ 范例三

```
01 void TypedefDemo3(void)
02 {
```

```
03    Word  i=0x00;
04
05    i++;
06 }
```

范例一至范例三的程序写法都是合法的。

第 2 章　程序的开始

➢ 主程序 main(　)

一个商品化的成品，在电源打开时其微处理器所执行的第一个程序进入点，就是 main( )函数，程序中的重复执行循环，可让微处理器不断地轮询外部状态，以进行实时反应。C 语言程序文件通常要在编写程序之前先指定头 H 文件(include 文件名.h)，因为所有的公共变量及函数，必须声明成“外部”(extern)，以利其它模块或函数调用。

例如 main.c 列表如下：

```
/*******************************/
/* include files               */
/*******************************/
#include  "define.h"
#include  "cpu8052.h"
#include  "global.h"
#include  "poweron.h"
#include  "delay.h"
#include  "input.h"
#include  "utility.h"
#include  "led.h"
#include  "beep.h"
#include  "music.h"
#include  "bcd.h"
#include  "dot5x7.h"
#include  "ic74138.h"
#include  "ic4094.h"
#include  "pulse.h"
#include  "ic4051.h"
```

```
20 #include  "key.h"
21 #include  "dac08.h"
22 #include  "eep93c66.h"
23 #include  "iic.h"
24 #include  "pwm.h"
25 #include  "eep24c08.h"
26 #include  "eep24c32.h"
27 #include  "osd.h"
28
29 #include  <intrins.h>
30 #include  <math.h>
31
32 /*******************************************/
33 void SubRoutine11(void);
34 void SubRoutine22(void);
35 void SubRoutine33(void);
36
37 /*******************************************/
38 //program start from main( ) while power on
39 void Main(void)
40 {
41    PowerOnInitial( );  //power on initial
42
43    while( 1 )
44    {
45
46       KeyProcess( );
47
48       SubRoutine11( );
49
50       SubRoutine22( );
51
52       if ( FgLedFlag==0 )
53       {
54          if ( LedOffCount==0 )  //timer off,led off
```

```
55          {
56              FgLedFlag = 1;
57              LedOff( );
58          }
59      }
60
61      SubRoutine33( );
62    }
63 }

64 void SubRoutine11(void)
65 {
66 }
67
68 void SubRoutine22(void)
69 {
70 }
71
72 void SubRoutine33(void)
73 {
74 }
75
```

行 号	说　　明
33~35	被 main 主过程调用的子程序，如果不利用包含文件设为外部函数，则必须写在程序开头，才能正确的编译
41	系统由多个模块的硬件组成，而大部分主要的 IC 皆由微处理器控制，因此开机后一般要做微处理器内部的初始化及外围控制 IC 的初始化
43	重复循环，才能不断的侦测外界的变化而立即反应
46	按键的处理程序
48,50	列举其它的处理例程，为空叙述只是说明用
52~59	计时中断应用的写法和程序框架，当计时的时间终了，则使 LED 熄灭并设定标志，以避免程序不断的执行而产生误动作

行 号	说 明
61	空叙述，表示为其它的处理例程
64~74	空叙述，仅说明用

其中：

➢ #include "define.h"

将定义文件包含进来，其内容如下：

```
#ifndef   _DEFINE_H
#define   _DEFINE_H

//declare
typedef bit   Bit;
typedef bit   Bool;
typedef unsigned char Byte;
typedef unsigned int  Word;
typedef unsigned long Long;

#define DATA  data
#define IDATA idata

#define PDATA pdata
#define XDATA xdata
#define RDATA code

#define HIBYTE_REF(addr)  (* ((Byte *) & addr) )
#define LOBYTE_REF(addr)  (* ((Byte *) & addr + 1) )
#define MAKEWORD(v1,v2)   ( ( ((Word)(v1))<<8 ) + (Word)(v2) )
#define HIBYTE(v1)     ( (Byte) ( (v1) >>8    ) )
#define LOBYTE(v1)     ( (Byte) ( (v1) & 0xff ) )

#define TIME_BASE 40
#define TIME_1MIN (60000/TIME_BASE)
#define TIME_25SEC    (25000/TIME_BASE)
```

```
#define TIME_20SEC    (20000/TIME_BASE)
#define TIME_15SEC    (15000/TIME_BASE)
#define TIME_10SEC    (10000/TIME_BASE)
#define TIME_8SEC ( 8000/TIME_BASE)
#define TIME_6SEC ( 6000/TIME_BASE)
#define TIME_5SEC ( 5000/TIME_BASE)
#define TIME_4SEC ( 4000/TIME_BASE)
#define TIME_3SEC ( 3000/TIME_BASE)
#define TIME_2SEC ( 2000/TIME_BASE)
#define TIME_1SEC ( 1000/TIME_BASE)
#define TIME_400MS    ( 400/TIME_BASE)
#define TIME_80MS (  80/TIME_BASE)

#define CLOCK_BASE    1
#define CLOCK_40MS    (65536 -  40000 * CLOCK_BASE)
#define CLOCK_4096us  (65536 -  4096  * CLOCK_BASE)
#define CLOCK_5000us  (65536 -  5000  * CLOCK_BASE)
#define CLOCK_10000us (65536 - 10000  * CLOCK_BASE)

#define SUCCESS   0
#define FAILURE   1
#define FALSE (Bool)0
#define TRUE  (Bool)!FALSE

//key
#define  NO_KEY   0
#define  KEY1 1
#define  KEY2 2
#define  KEY3 3
#define  KEY4 4
#define  KEY5 5
#define  KEY6 6
#define  KEY7 7
#define  KEY8 8
#define  KEY9 9
```

```
#define  KEY1010
#define  KEY1111
#define  KEY1212
#define  DOUBLE_KEY    99

//eeprom
#define VERSION_ID    0x55aa
#define VERSION_ADR   1024-2  //eeprom last two bytes
#define VARIABLE1_ADDRESS 0
#define START_ADDRESS 80

//osd
#define DEVICE_ADR    0x7a
#define BLACK 0x00
#define BLUE  0x01
#define GREEN 0x02
#define CYAN  0x03
#define RED   0x04
#define MAGENTA   0x05
#define YELLOW    0x06
#define WHITE 0x07
#define BLINK 0x08
#define ATTRIBUTE_ROW 0xa0
#define ATTRIBUTE_COLUMN  0x40
#define DISPLAY_ROW   0x80
#define DISPLAY_COLUMN    0x40
#define USERFONT_ROW  0xc0
#define USERFONT_COLUMN   0x40

//osd font
#define _SPACE    0x00
#define _MINUS    0x20
#define _PLUS 0x21
#define _DOT  0xe6
#define _COLON    0x3a
```

```
#define _SLASH    0x5f
#define _DITTO    0x13
#define _COMMA    0x2b
#define _HBarR    0x77
#define _HBar0    0x76
#define _HBar1    0x75
#define _HBar2    0x74
#define _HBar3    0x73
#define _HBar4    0x72
#define _HBar5    0x71
#define _HBar6    0x70
#define _HBarL    0x6f

#define _EOF  0xFF
#define _ESC  0xFE
#define _1ST  0x01
#define _2ND  0x02
#define __    0x00
#define _0    0x4f
#define _1    0x31
#define _2    0x32
#define _3    0x33
#define _4    0x34
#define _5    0x35
#define _6    0x36
#define _7    0x37
#define _8    0x38
#define _9    0x39
#define _A    0x41
#define _B    0x42
#define _C    0x43
#define _D    0x44
#define _E    0x45
#define _F    0x46
```

```
#define _G    0x47
#define _H    0x48
#define _I    0x49
#define _J    0x4a
#define _K    0x4b
#define _L    0x4c
#define _M    0x4d
#define _N    0x4e
#define _O    0x4f
#define _P    0x50
#define _Q    0x51
#define _R    0x52
#define _S    0x53
#define _T    0x54
#define _U    0x55
#define _V    0x56
#define _W    0x57
#define _X    0x58
#define _Y    0x59
#define _Z    0x5a
#define _a    0x08
#define _b    0x09
#define _c    0x0a
#define _d    0x0b
#define _e    0x0c
#define _f    0x0d
#define _g    0x0e
#define _h    0x0f
#define _i    0x10
#define _j    0x24
#define _k    0x25
#define _l    0x3b
#define _m    0x60
#define _n    0x62
#define _o    0x63
```

```
#define _p    0x66
#define _q    0x6b
#define _r    0x7a
#define _s    0x7b
#define _t    0xa0
#define _u    0xa1
#define _v    0xb8
#define _w    0xb9
#define _x    0x01
#define _y    0xc5
#define _z    0x1a
//multi-language ID
#define LANGUAGE_ENGLISH  0
#define LANGUAGE_FRENCH   1
#define LANGUAGE_GERMAN   2
#define LANGUAGE_ITALY    3
#define LANGUAGE_SPANISH  4

#endif
```

➢ #include "cpu8052.h"

将8052缓存器的地址定义文件包含进来，其内容如下：

```
// 8052 sfr address declare
sfr   ACC = 0xE0;
sfr   B      = 0xF0;
sfr   PSW = 0xD0;
sfr   SP     = 0x81;
sfr   DPL    =0x82;
sfr   DPH    = 0x83;
sfr   P0     = 0x80;
sfr   P1     = 0x90;
sfr   P2     = 0xA0;
sfr   P3     = 0xB0;
sfr   IE     = 0xA8;
```

```
sfr   IP      = 0xB8;
sfr   PCON    = 0x87;
sfr   TCON    = 0x88;
sfr   TMOD    = 0x89;
sfr   TL0     = 0x8A;
 sfr  TL1     = 0x8B;
sfr   TH0     = 0x8C;
sfr   TH1     = 0x8D;
sfr   T2CON   = 0xC8;
sfr   RCAP2L  = 0xCA;
sfr   RCAP2H  = 0xCB;
sfr   TL2     = 0xCC;
sfr   TH2     = 0xCD;
sfr   SCON    = 0x98;
sfr   SBUF    = 0x99;

//---PSW---
sbit  CY      = 0xD7;
sbit  AC      = 0xD6;
sbit  F0      = 0xD5;
sbit  RS1     = 0xD4;
sbit  RS0     = 0xD3;
sbit  OV      = 0xD2;
sbit  P       = 0xD0;

//---TCON---
sbit  TF1     = 0x8F;
sbit  TR1     = 0x8E;
sbit  TF0     = 0x8D;
sbit  TR0     = 0x8C;
sbit  IE1     = 0x8B;
sbit  IT1     = 0x8A;
sbit  IE0     = 0x89;
sbit  IT0     = 0x88;
```

```
//---T2CON---
sbit  TF2     = 0xCF;
sbit  EXF2    = 0xCE;
sbit  RCLK    = 0xCD;
sbit  TCLK    = 0xCC;
sbit  EXEN2   = 0xCB;
sbit  TR2     = 0xCA;
sbit  CT2     = 0xC9;
sbit  CPRL2   = 0xC8;

//---PCON---
sbit  SMOD    = 0x8e;
sbit  GF1     = 0x8A;
sbit  GF0     = 0x89;
sbit  PD      = 0x88;
sbit  IDL     = 0x87;

//---SCON---
sbit  SM0     = 0x9F;
sbit  SM1     = 0x9E;
sbit  SM2     = 0x9D;
sbit  REN     = 0x9C;
sbit  TB8     = 0x9B;
sbit  RB8     = 0x9A;
sbit  TI      = 0x99;
sbit  RI      = 0x98;

//---IE---
sbit  EA      = 0xAF;
sbit  ET2     = 0xAD;
sbit  ES      = 0xAC;
sbit  ET1     = 0xAB;
sbit  EX1     = 0xAA;
sbit  ET0     = 0xA9;
sbit  EX0     = 0xA8;
```

```
//---IP---
sbit  PT2     = 0xBD;
sbit  PS      = 0xBC;
sbit   PT1     = 0XBB
sbit  PX1     = 0xBA;
sbit  PT0     = 0xB9;
sbit  PX0     = 0xB8;

//---P0---
sbit  P0_0    = P0 ^ 0;
sbit  P0_1    = P0 ^ 1;
sbit  P0_3    = P0 ^ 3;
sbit  P0_2    = P0 ^ 2;
sbit  P0_4    = P0 ^ 4;
sbit  P0_5    = P0 ^ 5;
sbit  P0_6    = P0 ^ 6;
sbit  P0_7    = P0 ^ 7;

//---P1---
sbit  OSD_SCL= P1 ^ 0;
sbit  SCL_PIN = P1 ^ 0;
sbit  ISCL     = P1 ^ 0;

sbit  OSD_SDA     = P1 ^ 1;
sbit  SDA_PIN     = P1 ^ 1;
sbit  ISDA          = P1 ^ 1;
sbit  STROBE1_PIN   = P1 ^ 0;  //ic4094
sbit  DATA_PIN        = P1 ^ 1;
sbit  CLK_PIN     = P1 ^ 2;
sbit  OE_PIN          = P1 ^ 3;
sbit  STROBE2_PIN   = P1 ^ 4;
sbit  COLUMN1_PIN = P1 ^ 4;   //scan line
sbit  COLUMN2_PIN = P1 ^ 5;
sbit  COLUMN3_PIN = P1 ^ 6;
```

```
sbit  COLUMN4_PIN = P1 ^ 7;

sbit  CS_PIN         = P1 ^ 0;//93c66
sbit  XSCK_PIN       = P1 ^ 1;
sbit  XSDO_PIN    = P1 ^ 2;
sbit  XSDI_PIN    = P1 ^ 3;

sbit  P1_0     = P1 ^ 0;
sbit  P1_1     = P1 ^ 1;
sbit  P1_2     = P1 ^ 2;
sbit  P1_3     = P1 ^ 3;
sbit  P1_4     = P1 ^ 4;
sbit  P1_5     = P1 ^ 5;
sbit  P1_6     = P1 ^ 6;
sbit  P1_7     = P1 ^ 7;

//---P2---
sbit  PULSE_PIN   = P2 ^ 0;   //pulse input
sbit  P2_0    = P2 ^ 0;
sbit  P2_1    = P2 ^ 1;
sbit  P2_2    = P2 ^ 2;
sbit  P2_3    = P2 ^ 3;
sbit  P2_4    = P2 ^ 4;
sbit  P2_5    = P2 ^ 5;
sbit  P2_6    = P2 ^ 6;
sbit  P2_7    = P2 ^ 7;

//---P3---
sbit  P3_0    = P3 ^ 0;
sbit  P3_1    = P3 ^ 1;
sbit  P3_2    = P3 ^ 2;
sbit  P3_3    = P3 ^ 3;
sbit  P3_4    = P3 ^ 4;
sbit  P3_5    = P3 ^ 5;
sbit  P3_6    = P3 ^ 6;
```

```
sbit  P3_7    = P3 ^ 7;
```

➢ #include "global.h"

将公共变量的外部声明文件包含进来，请参照附录 B，而公共变量文件内容如下：

```
//global.c

//input
Bool  FgP1_0;
Bool  FgP1_1;
Bool  FgP1_2;
Bool  FgP1_3;
Bool  FgP1_4;
Bool  FgP1_5;
Bool  FgP1_6;
Bool  FgP1_7;
Bool  FgP2_0;
Bool  FgExtInput1;
Bool  FgExtInput2;
Bool  FgExtInput3;
Bool  FgExtInput4;
Bool  FgExtInput5;
Bool  FgExtInput6;
Bool  FgExtInput7;
Bool  FgExtInput8;
Bool  FgExtInput9;
Bool  FgExtInput10;
Bool  FgExtInput11;
Bool  FgExtInput12;
Bool  FgExtInput13;
Bool  FgExtInput14;
Bool  FgExtInput15;
Bool  FgExtInput16;
```

```
Byte  IDATA    TypeState;
Byte  IDATA    InputState;
//led
Bool  FgExtOut1;
Bool  FgExtOut2;
Bool  FgExtOut3;
Bool  FgExtOut4;
Bool  FgExtOut5;
Bool  FgExtOut6;
Bool  FgExtOut7;
Bool  FgExtOut8;
Bool  FgLedFlag;
Bool  FgLed2On;
Bool  FgLed3On;
Bool  FgLed2Off;
Bool  FgLed1Off;
Bool  FgChange;
Byte  IDATA    LedCount;
Byte  IDATA    OutputState;
Byte  IDATA    OutPutData1;

//utility
Bool  FgBeepOff;
Bool  FgAlarmOff;
Bool  FgExtFreq1;
Bool  FgExtFreq2;
Bool  FgExtFreq3;
Bool  FgExtFreq4;
Bool  Fgdirection;
Byte  IDATA    *ByteVariablePtr;
Byte  IDATA    UpdateValue;
Byte  IDATA    BcdData1;
Byte  IDATA    BcdData2;
Byte  IDATA    BcdData3;
Byte  IDATA    BcdData4;
```

```
Byte  IDATA   DisplayState;
Byte  IDATA   BcdVarible;
Word  IDATA   MaxValue;
Word  IDATA   MinValue;
Word  IDATA   *WordVariablePtr;
Word  IDATA   BcdVarible1;
//pulse
Bool  FgPulse;
Bool  FgPulseShort;
Bool  FgPulseLong;
Bool  FgTimeout;
Byte  IDATA   DutyCount;
Byte  IDATA   DutyCycleValue;
Byte  IDATA   PulseData;
Byte  IDATA   LowPulseCount;
Byte  IDATA   HiPulseCount;
Byte  IDATA   WheelNow;
Byte  IDATA   WheelOld;
Byte  IDATA   RightCount;
Byte  IDATA   LeftCount;
Byte  IDATA   EncoderCnt;
Byte  IDATA   SquareCount;

//timer
Byte  IDATA   T40msTimer;
Word  IDATA   LedOffCount;
Word  IDATA   Timer40msCount;
Word  IDATA   PulseCount;

//key
Bool  FgDoubleKey;
Bool  FgKEY1;
Bool  FgKEY2;
Bool  FgKEY3;
Bool  FgKEY4;
```

```
Bool  FgKEY5;
Bool  FgKEY2_ONOFF;
Byte  IDATA    KeyData;
Byte  IDATA    KeyData1;
Byte  IDATA    KeyData2;
Byte  IDATA    KeyData3;
Byte  IDATA    KeyData4;
Byte  IDATA    KeyCount;
Byte  IDATA    ScanKeyCounter;
Byte  IDATA    KeyBuffer;
Byte  IDATA    KeyCountTimer;
Byte  IDATA    CoarseAdjCount;
//iic
Byte  IDATA    ByteCnt;
Byte  IDATA    SlvAdr;

//eeprom
Bool  FgVariable1;
Bool  FgVariable2;
Bool  FgVariable3;
Byte  IDATA    TrmBuf[34];
Byte  IDATA    Variable1;
Byte  IDATA    Variable2;
Byte  IDATA    Variable3;
Byte  IDATA    Variable4;
Byte  IDATA    Variable5;
Byte  IDATA    Variable6;
Byte  IDATA    Variable7;

//93c66
Byte  IDATA    Buffer;

//osd
Byte  IDATA    OsdLanguage;
Byte  RDATA    *DataPointer;
```

```
Byte  IDATA   CurrentValue;
Byte  IDATA   OSDMinValue;
Byte  IDATA   OSDMaxValue;
Byte  IDATA   CurrentValue;
Byte  IDATA   NextColumn;

//external ram
Byte  PDATA   XFR_ADC _at_ 0x10;
Byte  PDATA   DAC0    _at_ 0x20;
```

行号	说　明
07~27	将所有文件（文件名.c）的函数外部声明文件（文件名.c）包含进来，请参照附录 B。

➢ #include <intrins.h>

将 keil_c 的外部函数 intrins.h(逻辑运算文件)包含进来，程序中才能使用这些函数，使用说明请参照 keil_c 资料手册，其内容如下：

```
//INTRINS.H
extern void   _nop_   (void);
extern bit    _testbit_   (bit);
extern unsigned char  _cror_   (unsigned char, unsigned char);
extern unsigned int   _iror_  (unsigned int,  unsigned char);
extern unsigned long  _lror_   (unsigned long, unsigned char);
extern unsigned char  _crol_   (unsigned char, unsigned char);
extern unsigned int   _irol_   (unsigned int,  unsigned char);
extern unsigned long  _lrol_   (unsigned long, unsigned char);
extern unsigned char  _chkfloat_   (float);
```

➢ #include <math.h>

将 keil_c 的外部函数 math.h (数值运算文件)包含进来，程序中才能使用这些函数，使用说明请参照 keil_c 资料手册，其内容如下：

```
//MATH.H
```

```
#pragma SAVE
#pragma REGPARMS
extern char   cabs    (char val);
extern int    abs (int    val);
extern long   labs    (long val);
extern float  fabs    (float val);
extern float  sqrt    (float val);
extern float  exp (float val);
extern float  log (float val);
extern float  log10   (float val);
extern float  sin (float val);
extern float  cos (float val);
extern float  tan (float val);
extern float  asin    (float val);
extern float  acos    (float val);
extern float  atan    (float val);
extern float  sinh    (float val);
extern float  cosh    (float val);
extern float  tanh    (float val);
extern float  atan2   (float y, float x);

struct FPBUF  {
 unsigned char save[16];
};

extern void   fpsave  (struct FPBUF *);
extern void   fprestore   (struct FPBUF *);
extern float  ceil    (float val);
extern float  floor   (float val);
extern float  modf    (float val, float *n);
extern float  pow (float x, float y);

#pragma RESTORE
```

第 3 章　开机后的启动流程

PowerOnInitial(　)

目的

打开电源后的初始动作。

程序

```
01 void PowerOnInitial(void)
02 {
03     InitialCpu( );
04
05     InitialCpuIO( );
06
07     EepromRead(VERSION_ADR,2); //power on check EEPROM id
08     if ( (TrmBuf[0]!=HIBYTE(VERSION_ID)) ||
09        (TrmBuf[1] !=LOBYTE(VERSION_ID)) )
10       InitialEeprom( );
11
12     InitialVariable( );
13 }
```

说明

初始化的动作完全依据系统的需求及复杂度而有所不同，系统简单也许仅是做微处理器内部的初始化，包括 CPU(Central Processing Unit：中央处理单元)的缓存器，I/O(Input/Output：输出入端口)的起始值以及撰写软件所需要的变量等，如果系统大复杂度高，除了微处理器内部初始化外，每一个被微处理器控制的 IC(Integrated Circuit：集成电路)，也都要做初始化，即设定控制 IC 内部缓存器的起始值。

行　号	说　明
03	微处理器内部缓存器的初始化
05	I/O 的设定或清除
07,08	EEPROM(Electrically Erasable Programmable Read-Only Memory 电子可擦写程序化只读存储器)内容初始默认值，首先读取 EEPROM 内容的版本数值，如果版本不一致，表示第一次开机要加载默认值，或者想要重新加载默认值
10	载入 EEPROM 内容的预设值，预设值是写在内存里的
12	共享变量的初始化，一般清除为 0，但有时也会设定为某数值，取决于软件写法和如何应用

➢ InitialCpu( )

目的

初始微处理器内部的缓存器

程序

```
01 void InitialCpu(void)
02 {
03     IE  = 0;          //disable all interrupt
04     PSW = 0;          //bank 0
05     IP=0x0b;          //hi priority:int0,count0,timer1
06     TMOD=0x15;  //set timer,counter mode
07
08     TR0 = 0;     //stop count0
09     TR1 = 1;          //start timer1
10     IT0 = 1;     //set int0:falling eage trigger
11
12     TL0 = 0xff;      //counter0=0xffff,count 1 time,execute
                  service route
13     TH0 = 0xff;
14     TL1 = CLOCK_40MS & 0xff;  //CLOCK_40MS=(65536-40000)
                           for 12Mhz
```

```
15    TH1 = CLOCK_40MS >> 8;
16
17    EX0 = 1;          //enable int0 interrupt
18    ET1 = 1;          //enable timer1 interrupt
19    ET0 = 1;          //enable count0 interrupt
20    EA = 1;       //enable all interrupt
21 }
```

说明

微处理器内部缓存器也许未完全被系统使用到，只要初始化程序中使用到的部分即可，而微处理器本身重置动作完成后，其 CPU 内部缓存器的数值设定如下：

寄存器	内　容	寄存器	内　容
PC	0x00	TCON	0x00
ACC	0x00	TH0	0x00
B	0x00	TL0	0x00
PSW	0x00	TH1	0x00
SP	0x07	TL1	0x00
DPTR	0x00	TH2	0x00
P0~P3	0xff	TL2	0x00
IP	XXX00000B	SCON	0x00
IE	0XX00000B	SBUF	未定
TMOD	0x00		

行　号	说　明
03	除能所有中断
04	选择 BANK0
05	设定高优先权
06	设定 Timer1、Count0 模式
08,09	开始计数、计时
10	外部中断为下降缘触发，即 INT0 的脚由“1”电平下降至“0”电平，则立即作中断例程
12,13	外部 T0 脚计数一次，则执行计数中断例程
14,15	内部系统时钟每 40mS，则执行计时中断例程

➢ InitialCpuIO( )

目的

初始化 CPU 的 I/O 设定或清除

程序

```
01 void InitialCpuIO(void)
02 {
03     P1_0 = 0;       //according hardware set or reset
04     P1_1 = 1;
05     P1_2 = 0;
06     P1_3 = 0;
07     P1_4 = 1;
08     P1_5 = 1;
09     P1_6 = 0;
10     P1_7 = 1;
11 }
```

说明

CPU I/O 的用途是控制系统的外围电路，输出为“1”电平或输出“0”电平，或是将系统的信号当成是 CPU I/O 的输入脚，反应给 CPU，因此 CPU I/O 的初始化完全要依据系统硬件设计而作，在此仅举例说明。

行 号	说　明
03	清除 P1.0，C 语言中将“0”设定给 P1.0，程序的写法为 P1_0=0，其中 P1_0 就是 I/O P1.0 的脚，那么 CPU I/O P1.0 就会输出为“0”电平，同理，“1” 电平也是一样
04	将 I/O P1.1 脚设定为“1”电平
05	将 I/O P1.2 脚清除为“0”电平
06	将 I/O P1.3 脚清除为“0”电平
07	将 I/O P1.4 脚设定为“1”电平
08	将 I/O P1.5 脚设定为“1”电平

行 号	说 明
09	将 I/O P1.6 脚清除为“0”电平
10	将 I/O P1.7 脚设定为“1”电平

➢ InitialEeprom( )

目的

将预设数值加载至 IC EEPROM 内。

程序

```
01 Byte RDATA EEPROM_DEFAULT_TABLE1[ ][6] =
02 {
03   {0x11 , 0x12 , 0x13 , 0x14 , 0x15 , 0x16},
04   {0x81 , 0x82 , 0x83 , 0x84 , 0x85 , 0x86},
05   {0xC1 , 0xC2 , 0xC3 , 0xC4 , 0xC5 , 0xC6}
06 };
07 Byte RDATA EEPROM_DEFAULT_TABLE2[ ] =
08 {
09   0x80, 0x00,
10   0x0f, 0x10,   0x10,   0x00,
11   0x01,     0x00,   0x00,   0x00,
12   0x00,     0x00
13 };
14
15 void InitialEeprom(void)
16 {
17    Byte  i,j;
18
19    for(i=0; i<3; i++)
20    {
21       DataPointer = EEPROM_DEFAULT_TABLE1[i];
22       for(j=0; j<6; j++)
23          TrmBuf[j+1] = *(DataPointer+j);
```

```
24        EepromWrite( i*6,6 );
25    }
26
27    for( i=0; i<12; i++)
28        TrmBuf[i+1] = EEPROM_DEFAULT_TABLE2[i];
29    EepromWrite( 0x80,12 );
30
31    TrmBuf[1] = HIBYTE( VERSION_ID );   //EEPROM id:0x55AA
32    TrmBuf[2] = LOBYTE( VERSION_ID );
33    EepromWrite( VERSION_ADR,2 );
34 }
```

说明

使用 eeprom 最主要的目的就是记忆，其最大的特点就是即使不供应电源(+5V)，其地址里面的资料仍然不会消失，因此可用来记忆系统中前一次的状态，以达到最佳化或自动化的性能。

行　号	说　　明
01~06	内定表 1 的数据，是属于二维数组
07~13	内定表 2 的数据，是属于一维数组
19	表 2 有三列数据，因此作三次
21	表 1 数据的第一个地址
22~24	每一列上有 6 个数据，依次将数据存入 TrmBuf 的数组中，再将 TrmBuf 数组中的数值，依据每一列的地址存入 EEPROM 内
27~29	读取内定表二的资料，并依序写入 EEPROM 0x80 的地址内
31~33	读取 EEPROM 版本的代号，并写入 EEPROM 内，表示已经初始化完成，而 EEPROM 内也有预设的资料值了

➢ InitialVariable( )

目的

变量的初始化。

程序

```
01 void InitialVariable(void)
02 {
03     LedCount   = 0x00;
04     OutputState= 0xff;
05     DutyCount  = 0;
06     KeyData    = 0;
07
08     FgKEY1  = 0;
09     FgKEY2  = 0;
10     FgKEY3  = 0;
11     FgKEY4  = 0;
12     FgKEY5  = 0;
13 }
```

说明

程序里所使用到的变量一一作初始化的设定，这样会比较麻烦，占用程序记忆空间大，但对设计者而言会比较明了，而且测试或修改容易，其时也可以程序一开始就将 8051 或 8052 的内部 RAM(Random-Access Memory:随机存取内存)清除掉，子程序 RamClear 就具备这个功能，之后再将非 0 的变量重新设定数值即可，要如何使用或设计，完全视设计者本身的习惯与思考逻辑而定。

行 号	说　　明
03	将 LedCount 的变量设定为 0，即 LedCount 的内容变成 0
04	将 OutputState 状态变量设为 0xff，也就是十进制 255
05	将 DutyCycle 计数值设为 0，即计数值从 0 开始累加
06	将按键检查例程的按键值初始为 0
08~12	将每一个按键的对应标志初始为 0

第 4 章　延时例程

➢ DelayX1ms(count)

目的

利用程序执行循环来作为短暂延迟(一)

参数

count：次数，总共延迟时间为 1ms 乘以 count

程序

```
01 void DelayX1ms(Word count) //crystal=12Mhz
02 {
03    Word i,j;           //variable:declare Word
04
05    for(i=0; i<count; i++)
06       for(j=0; j<120; j++)
07          ;
08 }
```

说明

程序中短暂时间的延迟，常以 DELAY 子程序来完成，因为简单。利用二层 for 循环来完成延迟的目的，其中行号 07 的分号，为 for 循环里的空叙述，行号 06 ，j 变量里的条件式 j<120 是试验出来的，要依据 CPU 的时钟频率不同而修正，本书所使用的 CPU 时钟频率为 12MHz。

➢ DelayX1ms1(count)

目的

短暂延迟(二)

参数

count：次数，总共延迟时间为 1ms 乘以 count

程序

```
01 void DelayX1ms1(Word count)
02 {
03    Word j;
04
05    while (count-- != 0)
06    {
07       for(j=0; j<72; j++)
08          ;
09    }
10 }
```

说明

此程序利用 while 循环，和上一个程序使用 for 循环来比较，其执行速度也是有差异性的，为了延迟 1ms 其循环的条件式修改成 j<72，由此可见，程序的功用一样，写法设计技巧却可以不一样。

➢ DelayX1ms2(count)

目的

短暂延迟(三)

参数

count：次数，总共延迟时间为 1ms 乘以 count

程序

```
01 void DelayX1ms2(Byte count)
02 {
03    Byte i,j,k;     //declare Byte,assembly different
04
05    for(i=0; i<count; i++)
06       for(j=0; j<40; j++)
07          for(k=0; k<120; k++)
08             ;
09 }
```

说明

LOCAL 变量(子程序暂时使用的变量，随着子程序返回调用程序而消失)改成了 Byte 型态，所以 i、j、k 变量值不能超过 255，为了达到 1ms 的延迟，将 DELAY 例程修改成三层的循环。综上所述，DELAY 例程完全取决于 CPU 的机器周期的长短，而不同写法其编译器所编译出来的机器码也不同，执行速度也就跟着不同，因而需要修正循环条件式的数值。

➢ DelayX10ms(count)

目的

10ms 延迟的应用(一)

参数

count：次数，总共延迟时间为 10ms 乘以 count

程序

```
01 void DelayX10ms(Word count)
02 {
03    Word i,j,k;
04
05    for(i=0; i<count; i++)
06       for(j=0; j<10; j++)
07          for(k=0; k<120; k++)
```

```
08            ;
09 }
```

说明

本程序为 DelayX1ms 子程序的延伸，程序中要延迟 1 秒以上时，使用了 DelayX1ms 而参数传 1000 比较不好，而使用 DelayX10ms 而参数传 100 感觉较好，在此只是将延迟例程的时间单位扩大而已，并不影响整体软件架构。

➢ DelayX10ms1(count)

目的

10ms 延迟的应用(二)

参数

count：次数，总共延迟时间为 10ms 乘以 count

程序

```
01 void DelayX10ms1(Word count)
02 {
03    Word j,k;
04
05    while (count-- != 0)
06    {
07       for(j=0; j<10; j++)
08          for(k=0; k<72; k++)
09             ;
10    }
11 }
```

说明

改成 while 循环的写法，并注意行号 08 已修改成 k<72，也就是说使用不同的循环写法有不同的执行速度。

➢ Delay50uS (count)

目的

极短延迟的应用(一)

参数

count：次数，总共延迟时间为 50uS 乘以 count

程序

```
01 void Delay50uS(Byte count)
02 {
03    Byte i,j;
04
05    for(j=0; j<count; j++)
06       for(i=0; i<6; i++)
07          ;
08 }
```

说明

12MHz 时钟频率的机器周期为 1uS，一个 NOP 指令也是为 1uS，如果要写很多个 NOP 指令，就可以利用一个极短延迟子程序来替代，以避免程序编写冗长，而且不易维护。

➢ ShortDelay(count)

目的

极短延迟的应用(二)

参数

count：次数，总共延迟时间为 50uS 乘以 count

程序

```
01 void ShortDelay(Byte count)
```

```
02 {
03     Byte i,j;
04
05     for(i=0; i<count; i++)
06       for(j=0; j<=140; j++)
07         _nop_( );      //include  <intrins.h>
08 }
```

说明

另外一种程序的写法，程序中行号 07 的叙述_nop_( )，相当于汇编程序的 NOP 指令，则必须在程序前面写上#include<intrins.h>的叙述，即将_nop_( )函数包含进来，其中 intrins.h 为 Keil-C 链接库的头文件。

➢ Timer40msDelay(count)

目的

利用定时器来做延迟例程

参数

count：次数，总共延迟时间为 40ms 乘以 count

程序

```
01 void Timer40msDelay(Byte count)
02 {
03     T40msTimer = count;
04     while ( T40msTimer != 0 ); //do loop until T40msTimer=0
05 }
06
07 // Description : 40ms interrupt
08 void Timer1ISR_40ms (void) interrupt 3 using 2
09 {
10     TL1 = CLOCK_40MS & 0xff;    //timer1:40ms
11     TH1 = CLOCK_40MS >> 8; //CLOCK_40MS=(65536 - 40000)
12     TF1 = 0;
```

```
13
14      if (T40msTimer != 0 )
15          T40msTimer--;
16  }
```

行 号	说　明
03	将参数值设定给 40ms 的计数值
04	不断的等待，程序并不会在此“死机”，因为计时中断每 40ms 就执行一次，而在计时中断例程会将 T40msTimer 减 1，直到 T40msTimer 等于 0 为止，因此，while 循环就会跳出而返回原调用程序了，即需要延迟的时间已经终了，就跳出 while 循环，当然在主程序要允许计时中断功能，否则程序会“死机”
08	计时中断例程

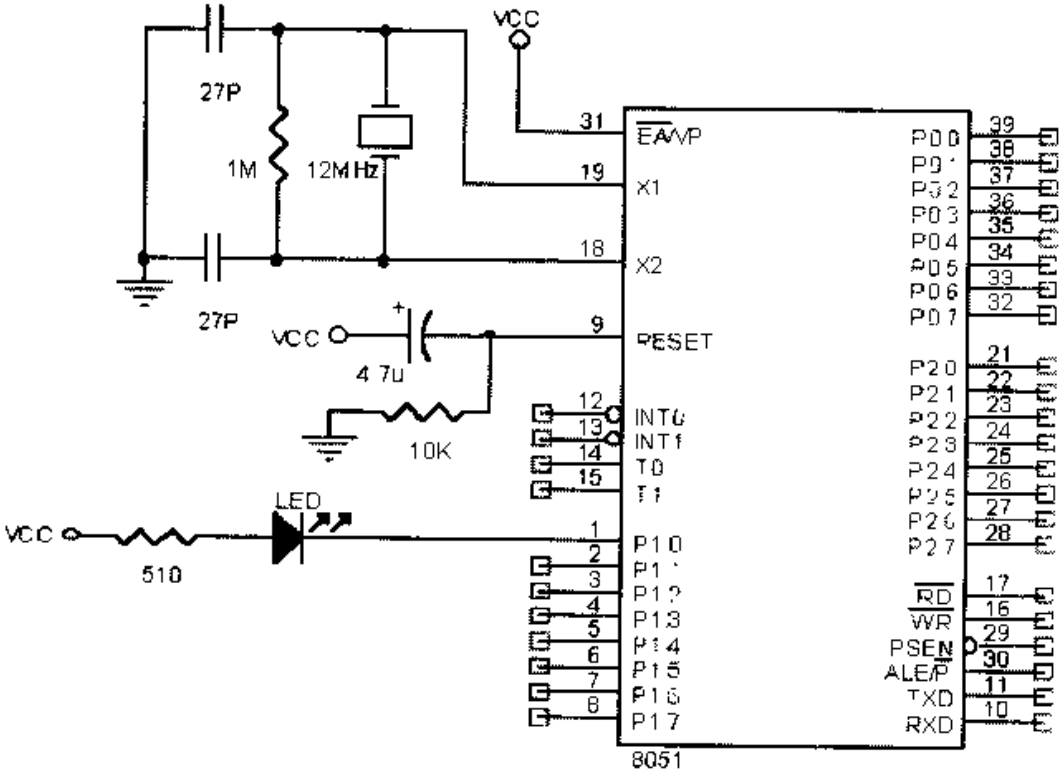
第 5 章　基本输入输出

➢ Led_1(　)

目的

通过 I/O 的控制，让 LED 亮或熄

电路



程序

```
01 void  Led_1(void)
02 {
03     P1_0=0;      //P1.0="0",led on
04 }
```

说明

任何控制 I/O 都是最基本的，而且大部分的应用程序也都需要用到 I/O，8051 共有 4 个 I/O 口，分别为 P0、P1、P2、P3，此 4 个接口都可做为单独的输入或输出使用，即每一支 I/O 脚位都可以做为输入用，也可以做为输出使用。当做为输出时，则每一支脚的外部电路可以高电平“1”驱动或低电平“0”驱动，如本程序即是低电平，当 P1.0 输出为“0”时，则 LED 亮，当 P1.0 输出为“1”时，则 LED 熄，8051 的 P3 又同时做其他功能接口使用，当用于其它功能时，则不能做为 I/O 使用，当做为 I/O 使用，则就无法做为其它功能使用了。而且 P0 是开集电极的 I/O 接口。

➢ LedOn( )

目的

通过晶体管的驱动，而使外部 LED 或继电器动作

电路

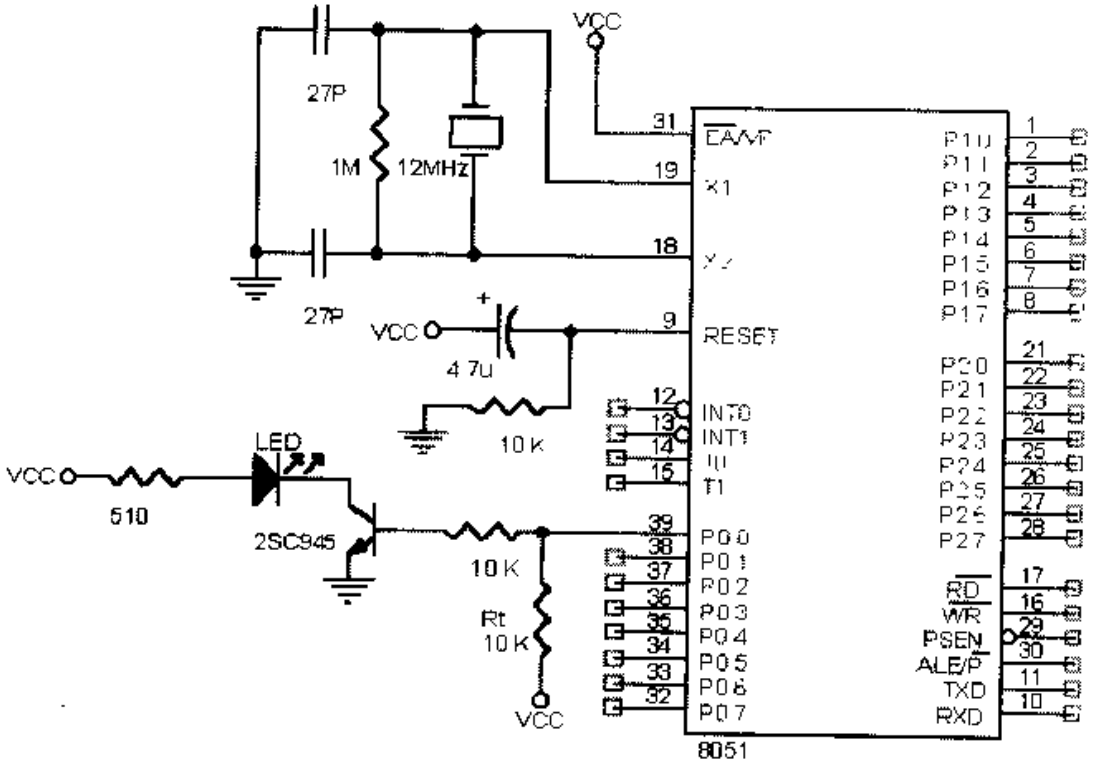
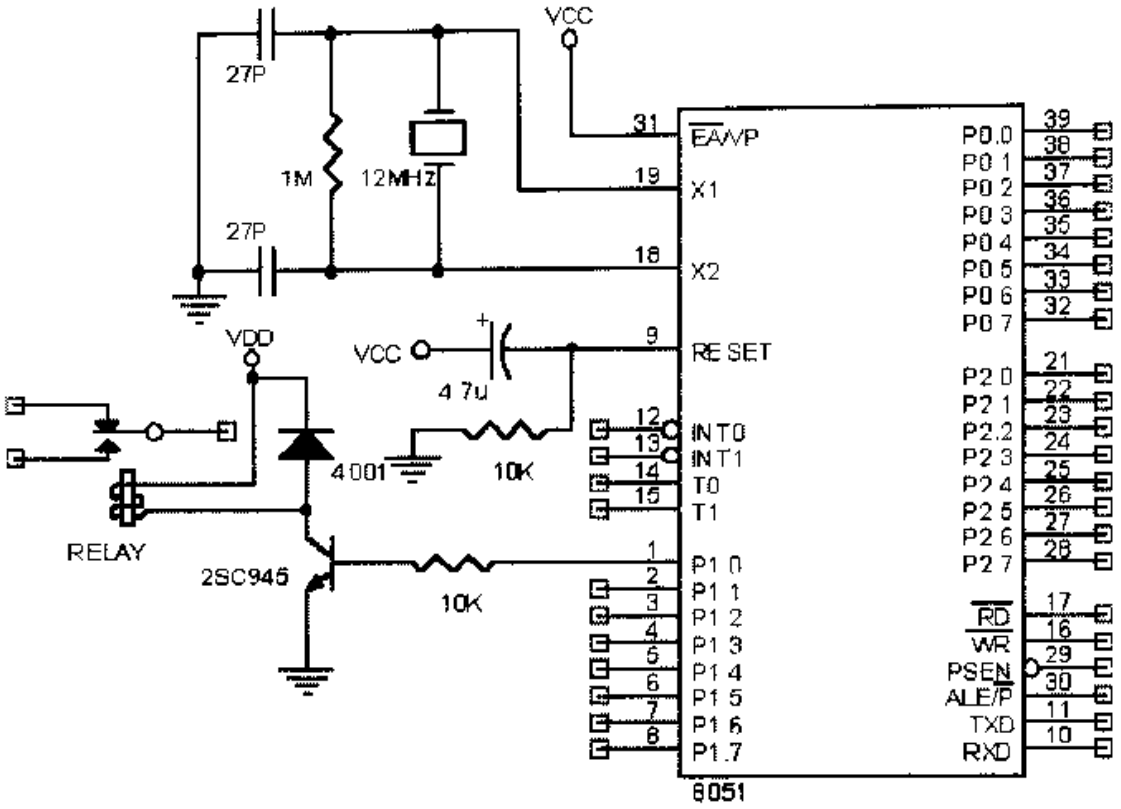
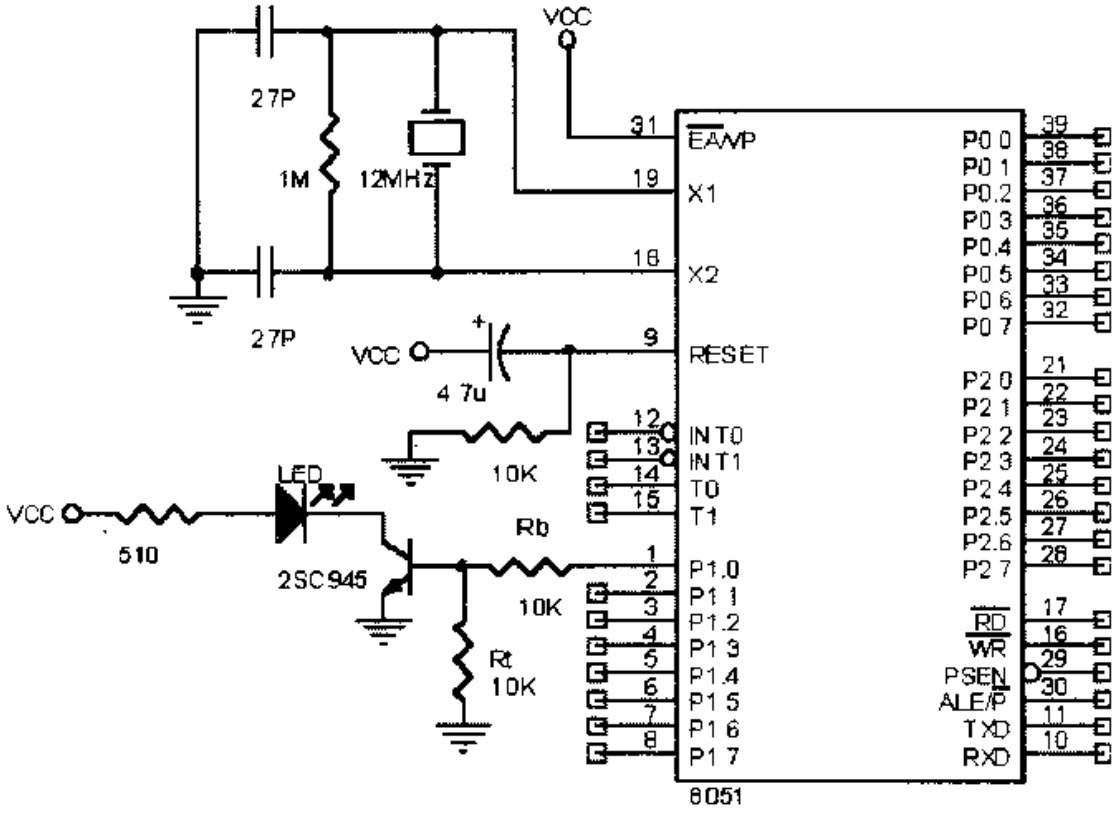


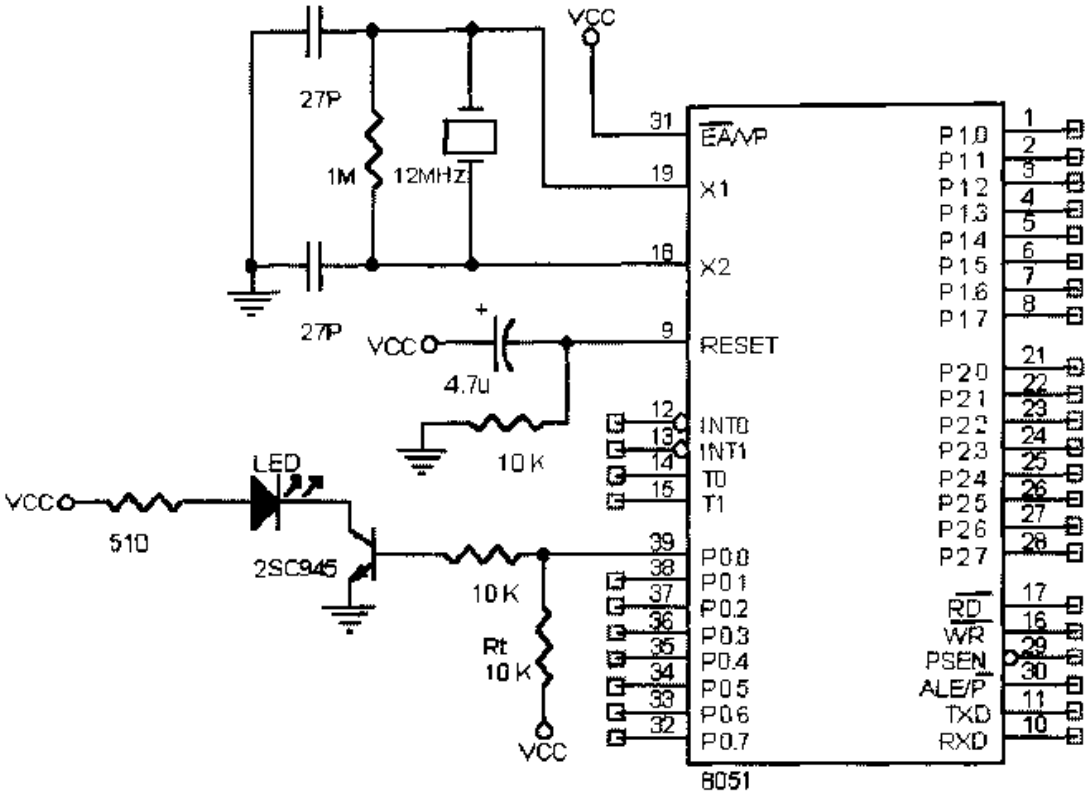
图---


VCC
27P
1M
12MHZ
27P
31
EA/VP
19
X1
18
X2
VDD
VCC
4.7u
9
RESET
4001
10K
12
INT0
13
INT1
14
T0
15
T1
RELAY
2SC945
10K
1
P1.0
2
P1.1
3
P1.2
4
P1.3
5
P1.4
6
P1.5
7
P1.6
8
P1.7
P0.0
P0.1
P0.2
P0.3
P0.4
P0.5
P0.6
P0.7
P2.0
P2.1
P2.2
P2.3
P2.4
P2.5
P2.6
P2.7
RD
WR
PSEN
ALE/P
TXD
RXD
8051

图二


VCC
27P
1M
12MHZ
27P
31
EA/VP
19
X1
18
X2
VCC
4.7u
9
RESET
10K
12
INT0
13
INT1
14
T0
15
T1
LED
VCC
510
Rb
2SC945
10K
Rt
10K
1
P1.0
2
P1.1
3
P1.2
4
P1.3
5
P1.4
6
P1.5
7
P1.6
8
P1.7
P0.0
P0.1
P0.2
P0.3
P0.4
P0.5
P0.6
P0.7
P2.0
P2.1
P2.2
P2.3
P2.4
P2.5
P2.6
P2.7
RD
WR
PSEN
ALE/P
TXD
RXD
8051

图三



图四

程序

```
01 void LedOn(void)
02 {
03    P1_0=1;            //P1.0="1",led on
04 }
05
06 void LedOff(void)
07 {
08    P1_0=0;            //P1.0="0",led off
09 }
```

说明

当要以“1”电平来做为输出时，由于 CPU I/O “1”输出的驱动能力不足，需要外加晶体管做为放大，如图一、图二。为了更稳定，可以在晶体管的基极接一个所谓的接地电阻，如图三 Rt 所示，接地电阻其电阻值可以等于基极电阻即 Rb，一般设计只要让电阻分压大约等于 1/2 Vcc 即可。

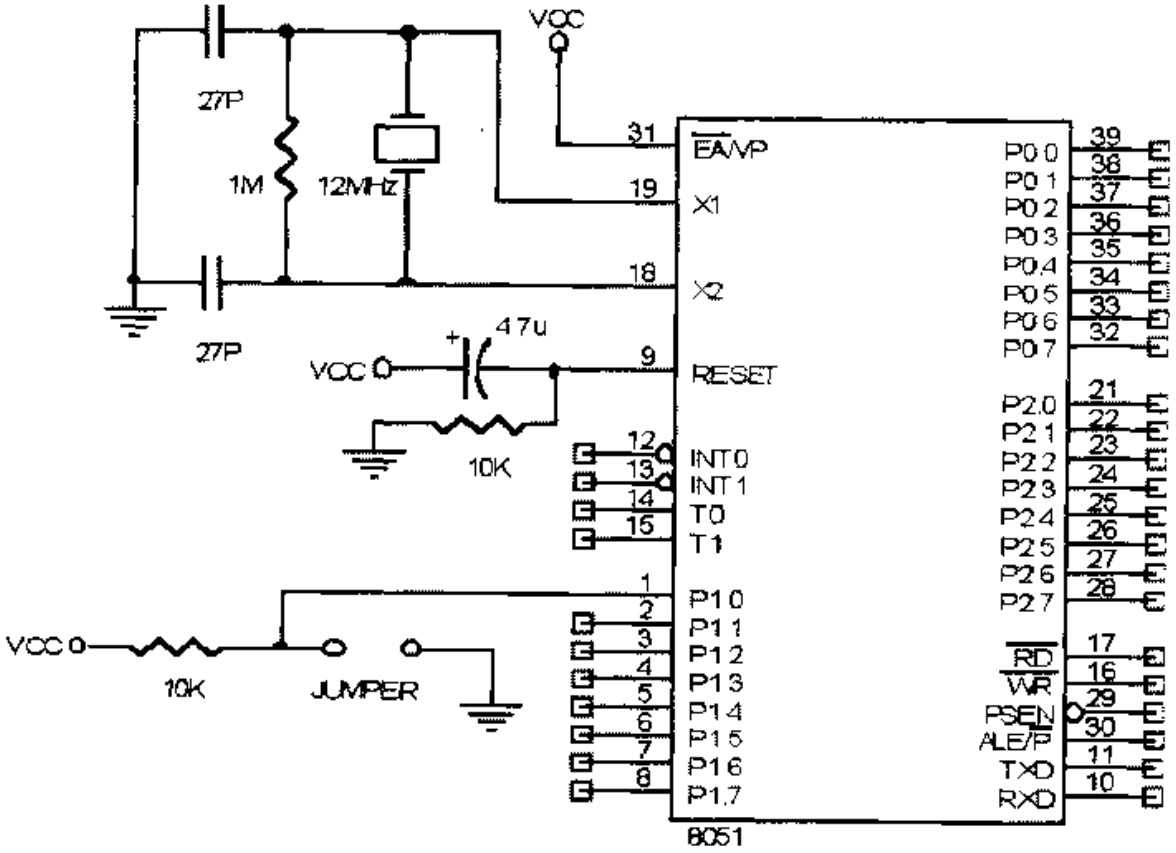
另外，有些输出端是开集电极(Open Collector)，则必须在输出端接上一个提升电阻，如图四 Rt 所示，一般为 10KΩ，但也要按电路的具体情况确定电阻值，如果不接上提升电阻 10KΩ，即使 P0.0=“1”时，晶体管仍然不会导通，LED 也不会亮，而造成电路无法正常动作。

➢ Input1( )

目的

I/O 作为输入接口使用的控制—“0”触发

电路



程序

```
01 //set flag and state
02 void Input1(void)
```

```
03 {
04     if ( P1_0==0 )
05     {
06        FgP1_0=1;
07        TypeState=0xff;
08     }
09     else
10     {
11        FgP1_0=0;
12        TypeState=0x00;
13     }
14 }
```

说明

一支脚的输入，即称为 1 bit 的输入控制，可以应用在传感器上的检测或是系统功能的选择上。当选择 I/O 接口的一支脚作为输入或输出时，此 I/O 接口最好有 bit 的寻址能力，以利程序的设计。而且要以一个 bit 的变量来储存外部状态，以便日后修改或维护程序而变得更容易，因为在其它子程序或函数模块中，时常会对此变量作判别及存取的动作。

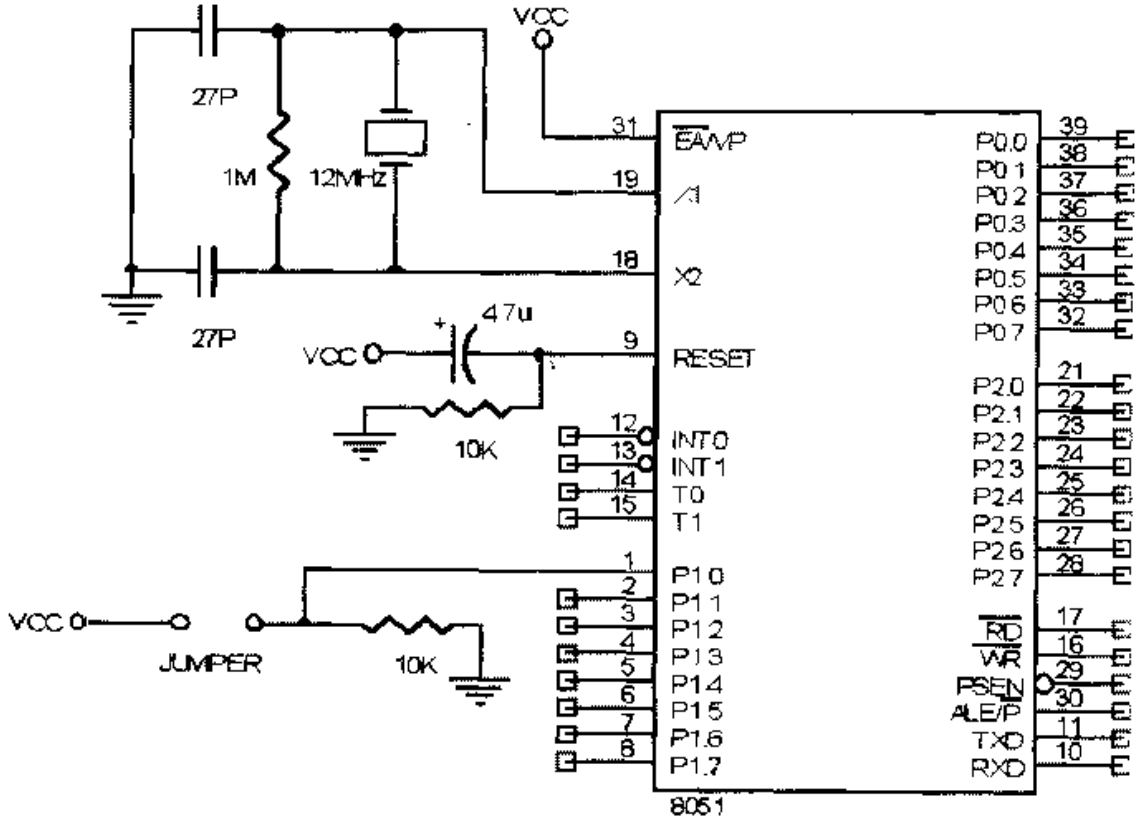
行 号	说　　明
04	判别 P1.0 是否为“0”输入，即上图 JUMPER 是否为短路
06~07	如果 JUMPER 短路，则设定标志变量为“1”，状态变量为“0xff”
11~12	如果 JUMPER 开路，即 P1.0 为“1”输入，则执行 else 的动作，即将标志变量设为“0”，状态变量也设为“0x00”，其中 FgP1_0 为标志变量，也就是 Bit 或 Bool 变量，而 TypeState 是状态变量

➢ Input2( )

目的

“1”动作的输入检测

电路



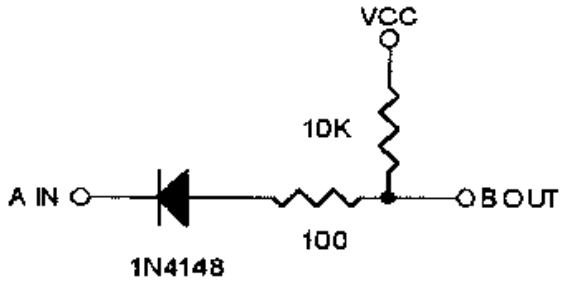
程序

```
01 void Input2(void)
02 {
03    if ( P1_0==1 )
04       FgP1_0=1;
05    else
06       FgP1_0=0;
07 }
```

说明

当 JUMPER 开路时，则 P1.0 输入为“0”并设定标志为 0，当 JUMPER 短路时，则 P1.0 输入为“1”，即“1”输入动作并设定标志为 1。

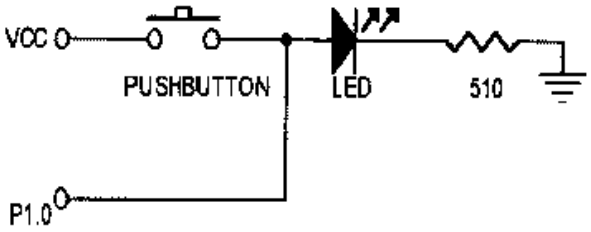
电路



说明

以二极管来阻隔 CPU I/O 和外部输入的控制电路，以避免外部电路损毁时破坏微处理器。当 A 输入端="0" 时，则电流通过二极管流向外部，因此 B 端="0"；当 A 输入端="1" 时，二极管为逆向无法导通，Vcc 经由 10KΩ 电阻而流向 B 端，因此 B 端="1" 电平。

电路



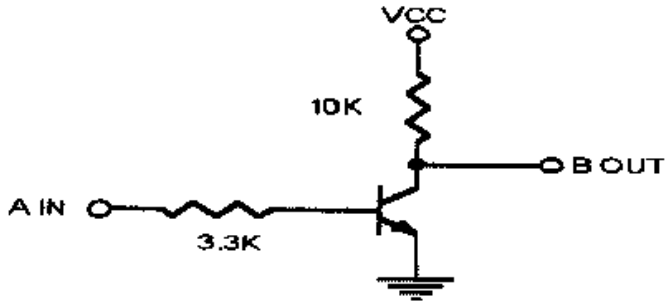
说明

以 LED 来监视外部开关的动作。

按下 PUSHBUTTON，则 LED 亮，P1.0="1"。

放开 PUSHBUTTON，则 LED 熄，P1.0="0"。

电路



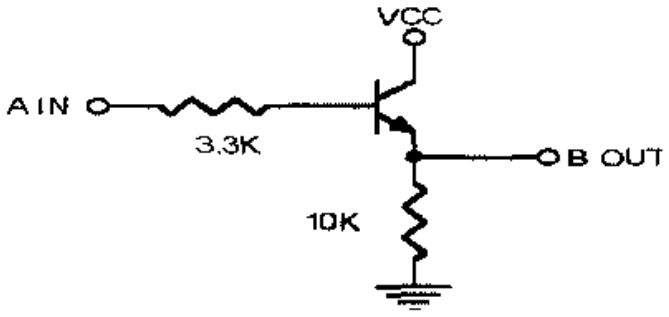
说明

利用晶体管集电极电阻的方法。

当 A 输入端="1"时，则晶体管 ON，所以 B 端="0"电平。

当 A 输入端="0"时，则晶体管 OFF，Vcc 经由 10KΩ流向 B 端，所以 B 端="1"。

电路



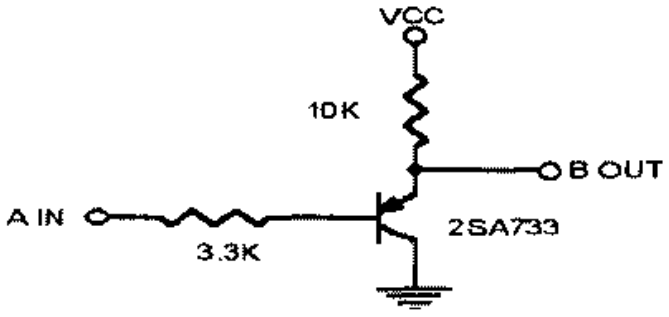
说明

利用晶体管发射极电阻的方法。

当 A 输入端="1"时，则晶体管 ON，10KΩ有 Vcc 的压降，所以 B 端="1"。

当 A 输入端="0"时，晶体管 OFF，B 端经由电阻接地，所以 B 端="0"。

电路



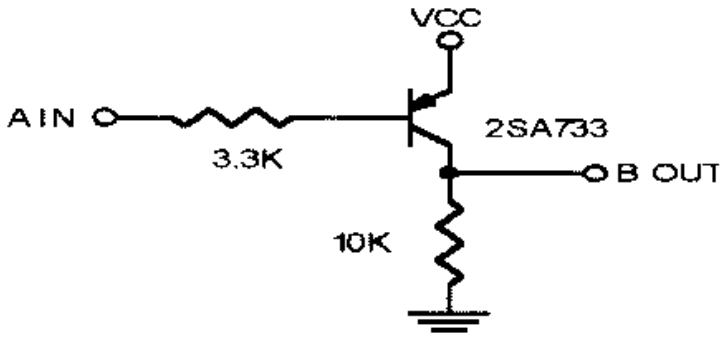
说明

利用 PNP 晶体管的转换。

当 A 输入端="1"时，则晶体管 OFF，所以 B 端="1"。

当 A 输入端="0"时，则晶体管 ON，所以 B 端="0"。

电路

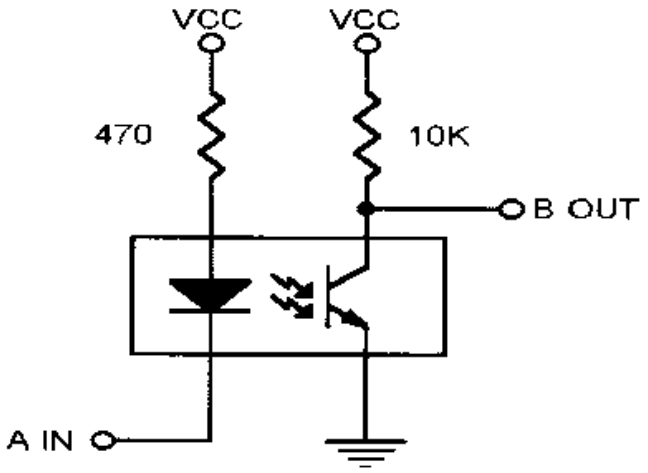


说明

当 A 输入端=“1”时，则晶体管 OFF，所以 B 端=“0”。

当 A 输入端=“0”时，则晶体管 ON，则 10K Ω将会有 Vcc 的压降，所以 B 端=“1”。

电路



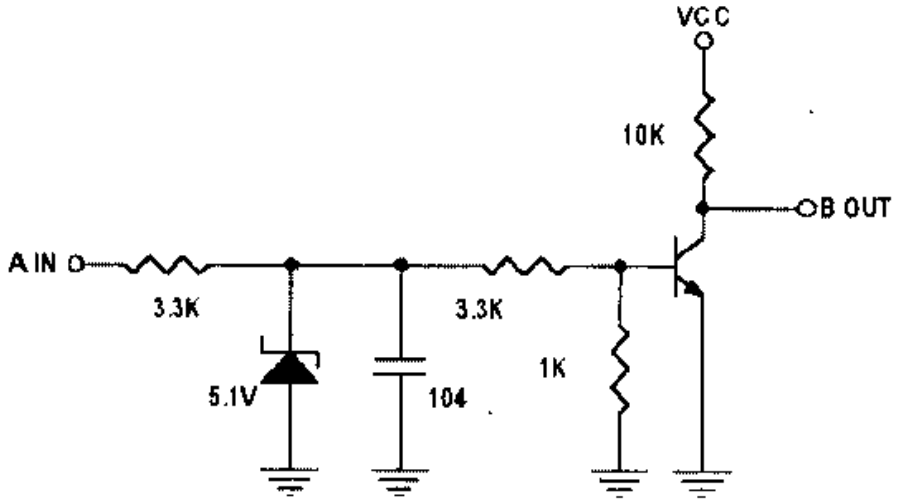
说明

利用光耦合器最主要的目的是隔离数字信号的噪音，以避免噪声的干扰，框线内是光耦合器组件。

当 A 输入端=“0”时，则光耦合器 ON，所以 B 端=“0”。

当 A 输入端=“1”时，则光耦合器 OFF，B 端=“1”。

电路

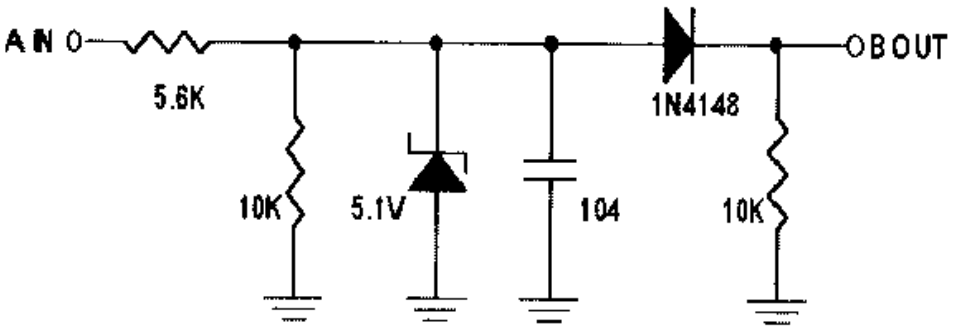


说明

一般 TTL 的输入电平为+5V，如果要检测外部电平为+12V 的输入时，则需要作转换电路，如上图所示。

当 A 输入端= +12V 输入经由齐纳二极管稳压至+5.1V，再经过电阻分压，晶体管基极电压大约等于 1.18V，足以让晶体管 ON，因此，B 端= “0”；当 A 输入端=0V 时，则晶体管 OFF，所以 B 端= “1” 电平。

电路

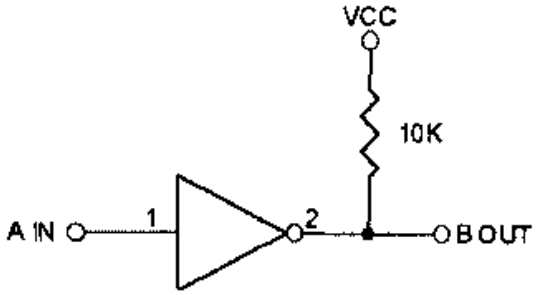


说明

当 A 输入端= +12V，经过电阻分压及齐纳二极管稳压至+5.1V，则二极管 ON，所以 B 端= “1” 电平。

当 A 输入端=0V 时，则二极管 OFF，B 端因为有 10KΩ的接地电阻，所以 B 端= “0” 电平。

电路



说明

利用 IC 作为缓冲器或电位的转换成本较高，如果有很多组的外部电路需要作转换，其零件种类管理成本及电路板插件的成本大于 IC 材料的成本，可考虑使用 IC 来做转换，但如果只有一组外部电路要转换，则以晶体管的方式成本较节省。

当 A 输入端= +12V，则 B 端= “0” 电平。

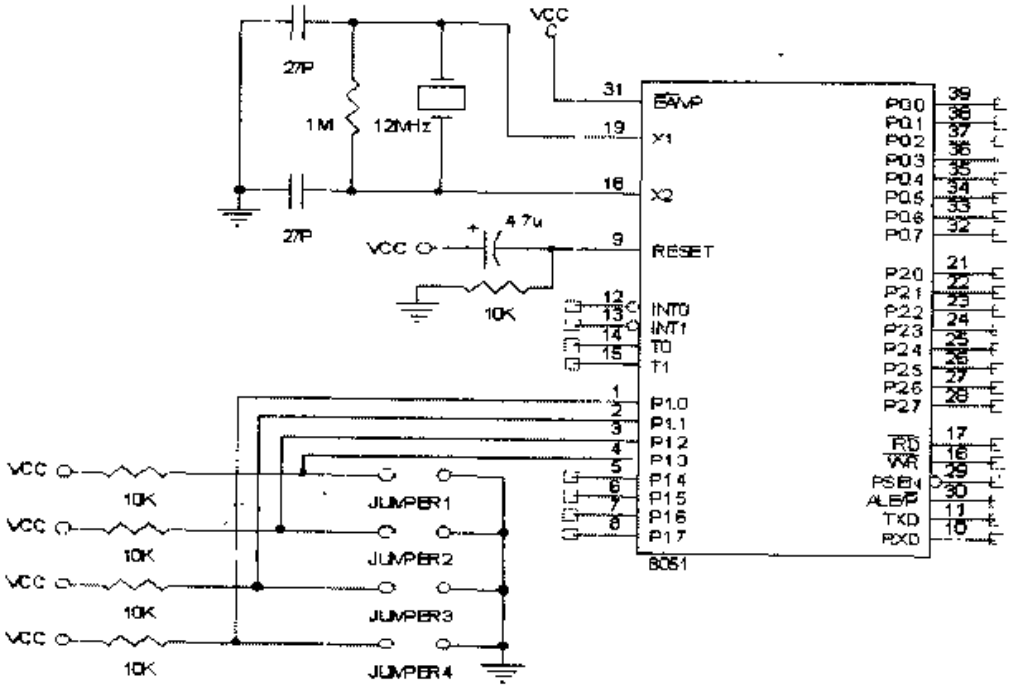
当 A 输入端= 0V，则 B 端= “1” 电平。

➢ Input3(　)

目的

多种输入的功能选择

电路



程序

```
01 void Input3(void)
02 {
03     Byte  keytmp;
04
05     keytmp = ~(P1) & 0x0f;  //if input,i/o="0"
06
07     //independent
08     FgP1_0=0;
09     FgP1_1=0;
10     if ( keytmp & 0x01 )   FgP1_0=1;          //P1.0="0" input
11     if ( keytmp & 0x02 )   FgP1_1=1;          //P1.1="0" input
12
13     //two bit test
14     TypeState = 0;
15     if ( (keytmp & 0x0C)==0x04 )       TypeState = 1;
  //bit2="0"
16     else if( (keytmp & 0x0C)==0x08 )   TypeState = 2;
  //bit3="0"
17 }
```

说明

如果有多种输入，则以相同接口作为输入较为简便，程序写法会较为简单明了，程序内存空间也较节省。

行 号	说　明
05	将外部输入状态暂存至变量 keytmp，因为是“0”动作而且是 P1.0~P1.3 作输入，所以将 P1 取反相，并屏蔽 P1.4~P1.7，只留下低 4 个位 P1.0~P1.3。
08,09	先将标志变量初始化，设定为 0
10	如果 JUMPER1 短路，即 P1.0="0"，反相后变为“1”，所以条件(keytmp & 0x01) 成立，并设定 FgP1_0=1
11	如同行号 10 的说明

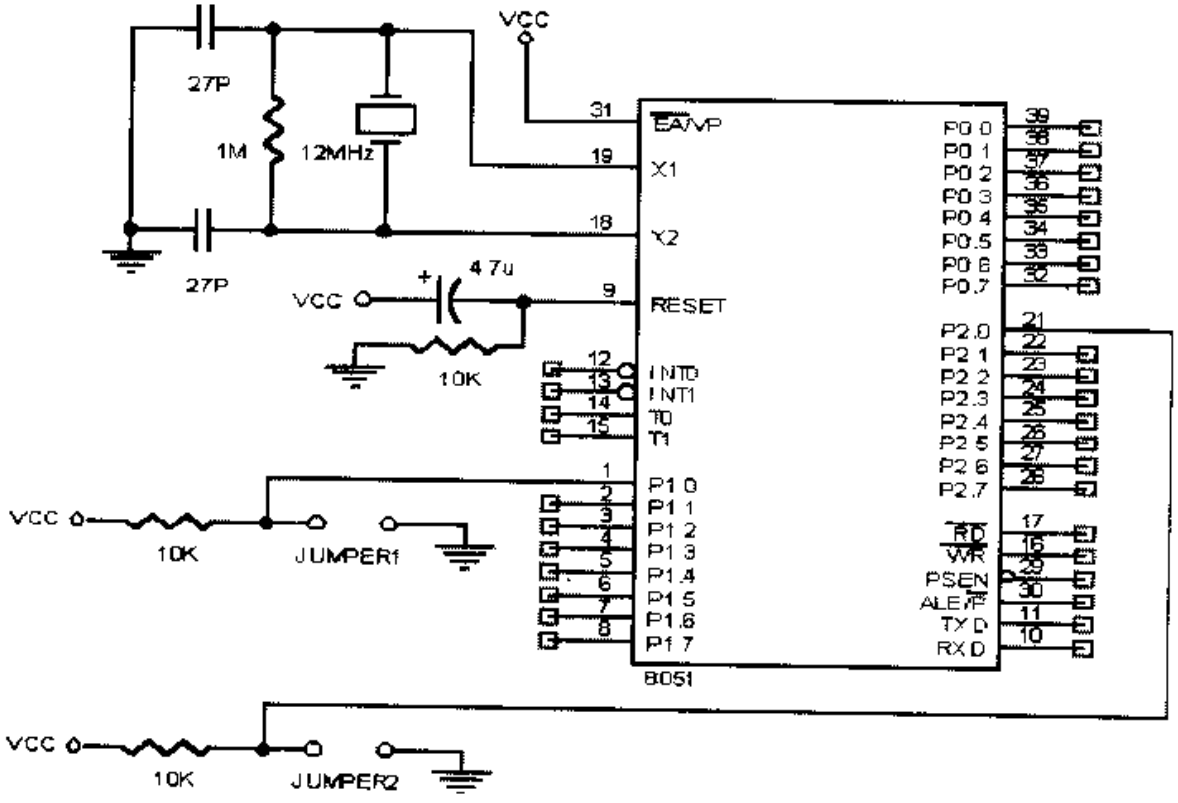
行　号	说　　明
14	状态变量初始化，设定为 0
15,16	判别 JUMPER3 和 JUMPER4，如果 JUMPER3 短路，则 TypeState = 1，如果 JUMPER4 短路，则 TypeState = 2

➢ Input4( )

目的

使用不同 I/O 接口作为输入的判别

电路



程序

```
01 //independent input
02 void Input4(void)
```

```
03 {
04     if ( P1_0==0 )
05        FgP1_0=1;
06     else
07        FgP1_0=0;
08
09     if ( P2_0==0 )
10        FgP2_0=1;
11     else
12        FgP2_0=0;
13 }
```

说明

当有 2 种输入时，硬件电路也可以使用不同的 I/O 端口，只是程序要将输入脚分别作判断是否动作。

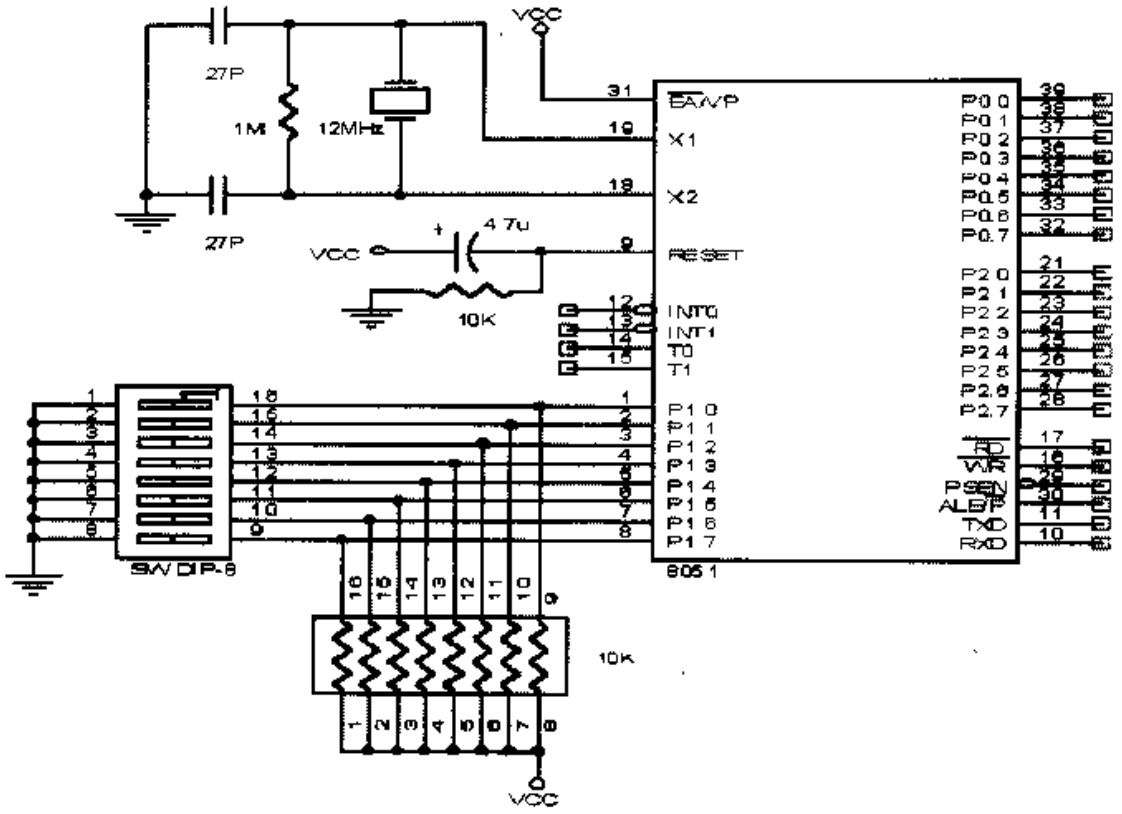
行 号	说 明
04~07	判别 JUMPER1 是否为短路，当 JUMPER1 短路则 P1.0 脚电位等于“0”电平，因此条件式成立，所以标志变量 FgP1_0=1；如果 JUMPER1 开路则 P1.0 脚位等于“1”电平，条件式不成立，因而执行 else，则标志变量 FgP1_0=0。
09~12	如同行号 04~07 的说明，只是以 P2.0 作为输入来判别

➢ Input5( )

目的

开关组的输入用法

电路



程序

```
01 void Input5(void)
02 {
03     Byte  keytmp;
04
05     keytmp = ~(P1);          //if input,i/o="0"
06     FgP1_0=0;
07     FgP1_1=0;
08     FgP1_2=0;
09     FgP1_3=0;
10     FgP1_4=0;
11     FgP1_5=0;
12     FgP1_6=0;
13     FgP1_7=0;
14     if ( keytmp & 0x01 )   FgP1_0=1;    //P1.0="0" input
15     if ( keytmp & 0x02 )   FgP1_1=1;    //P1.1="0" input
```

```
16    if ( keytmp & 0x04 )   FgP1_2=1;    //P1.2="0" input
17    if ( keytmp & 0x08 )   FgP1_3=1;    //P1.3="0" input
18    if ( keytmp & 0x10 )   FgP1_4=1;    //P1.4="0" input
19    if ( keytmp & 0x20 )   FgP1_5=1;    //P1.5="0" input
20    if ( keytmp & 0x40 )   FgP1_6=1;    //P1.6="0" input
21    if ( keytmp & 0x80 )   FgP1_7=1;    //P1.7="0" input
22 }
```

说明

开关组有 8 个输入，正好等于一个 I/O 接口有 8 个 bit，硬件设计上 I/O 端口最好使用同一个，不要各自分开，这样程序会很简单明了。

行 号	说 明
05	P1 端口取反，因为硬件设计上开关组为负逻辑
06~13	初始化标志变量都设为 0
14~21	将指拨开关的每一个输入分别做判断，如果开关短路则相对的标志变量设为 1，否则设为 0

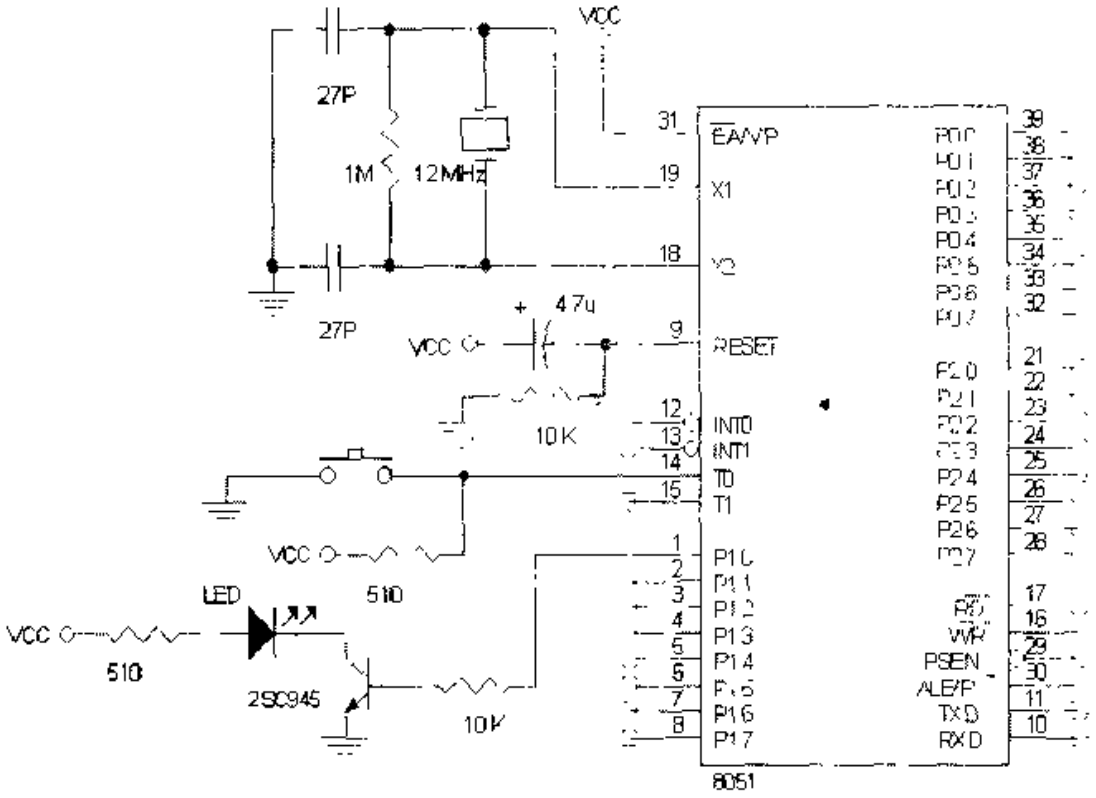
第 6 章　中断的应用

➢ CountMain1(　)

目的

利用计数器中断实现按键一次 LED 就闪烁一次。

电路



程序

```
01 void  InitialMain(void)
02 {
03     IE  = 0;          //disable all interrupt
```

```
04     IP  = 0x0b; //hi priority:int0,count0,timer1
05     TMOD= 0x15; //set timer1:model 16 bit
                   timer,timer0:counter0
06
07     TR0 = 1;          //start count0
08     TR1 = 1;          //start timer1
09     IT0 = 1;          //set int0:falling eage trigger
10
11     TL1 = CLOCK_40MS & 0xff;    //timer1:40mS
12     TH1 = CLOCK_40MS >> 8;
13     TL0 = 0xff; //counter0=0xffff,count 1 time,execute
            service route
14     TH0 = 0xff;
15
16     EX0 = 1;          //enable int0   interrupt
17     ET1 = 1;          //enable timer1 interrupt
18         ET0 = 1;          //enable count0 interrupt
19
20     EA  = 1;          //enable all interrupt
21 }
22
23 void  CountMain1(void) //non-bounce
24 {
25     IE  = 0;          //disable all interrupt
26     IP  = 0x0b; //hi priority:int0,count0,timer1
27     TMOD= 0x15; //set timer1:model 16 bit
            timer,timer0:counter0
28
29     TL0 = 0xff; //counter0=0xffff,count 1 time,execute
            service route
30     TH0 = 0xff;
31
32     ET0 = 1;          //enable count0 interrupt
33     EA  = 1;          //enable all interrupt
34
```

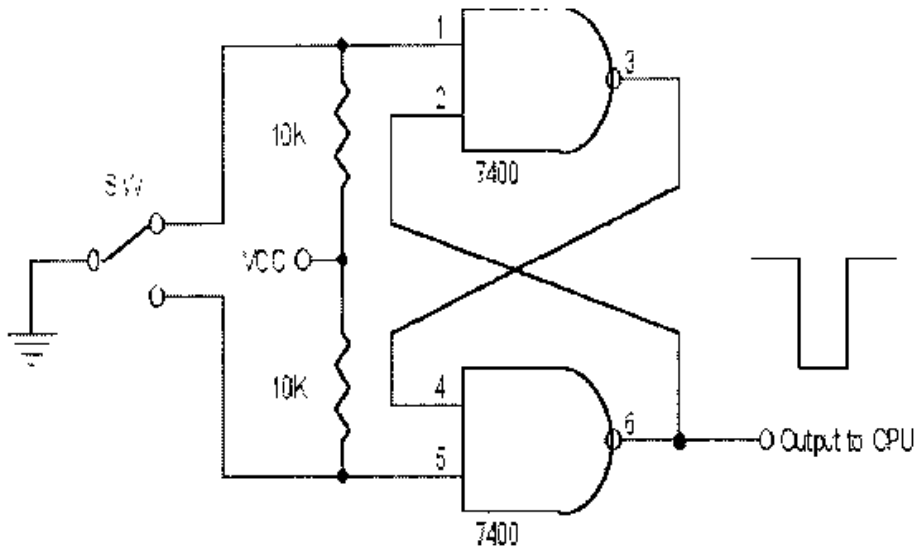
```
35      TR0 = 1;          //start count0
36
37      PulseCount=0;     //initial count
38  }
39
40  //count 1 time,execute service route
41  void  Timer0ISR_1 (void) interrupt 1 using 3
42  {
43      LedOn( );         //P1.0=1
44      DelayX10ms(50);
45      LedOff( );        //P1.0=0
46      DelayX10ms(50);
47
48      TL0 = 0xff;
49      TH0 = 0xff;
50      TF0 = 0;
51  }
```

说明

当外部有一脉冲或待检测的输入信号产生时，可利用计数器中断的方式立即作响应，如果使用轮询(Polling)的方式，往往容易遗漏掉此信号，或是反应时间太慢，无法实时响应，因此实际应用中首先考虑的是使用中断的方法。

行　号	说　　明
01~21	初始化微处理器的内部缓存器，即设定控制缓存器的数值，指定计数器为 T0 脚，定时器为计时计数器 1，即 COUNT0、TIMER1，每 40mS 为系统计时单位、设定优先权、使能计数器，使能定时器及允许中断。
23~30	主程序，设定 TH0、TL0 数值为 0xff，表示 T0 的脚位有一个负脉冲，就产生计数器中断，而执行计数器中断子程序
32~37	允许计数器中断，开始计数，并初始化计数器变量 PulseCount
41	计数器中断子程序
43~46	LED 闪烁一次
48,49	重新加载 TL0,TH0 值等于 0xff，即只要计数一次就立即中断
50	将中断溢出标志归零

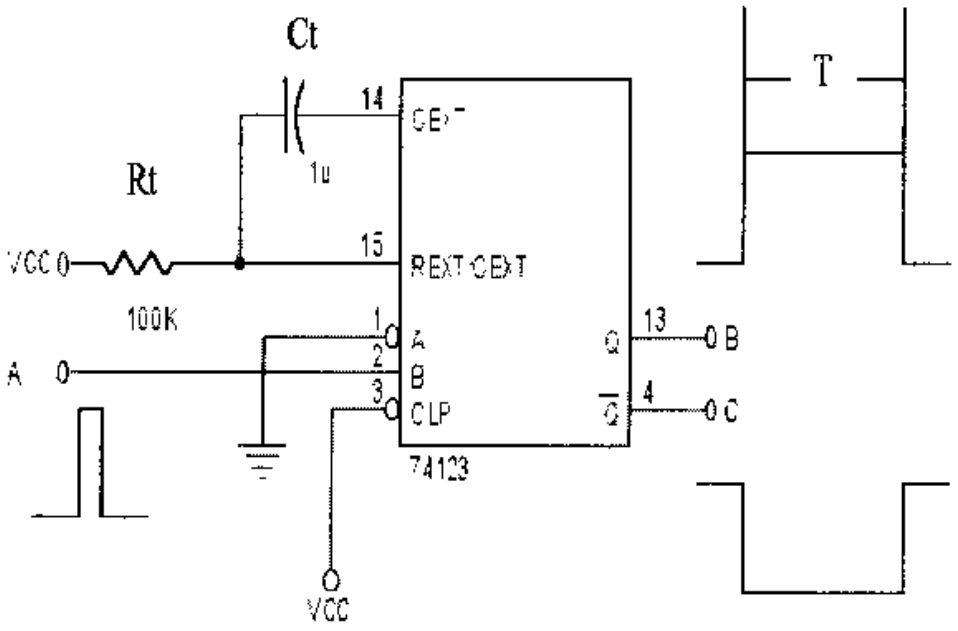
电路



说明

使用如按扭式开关则会产生杂波，并不会按一次开关只产生一个负脉冲，而会产生多个时间很短的方波，需要等待一段时间之后才会稳定，要解决杂波可用软件来延迟大约 20~40mS，或使用硬件电路来处理，如上图所示。当 S.W 拨动一次，则将会产生一个稳定的负脉冲，因此可避开杂波，以避免程序在执行时而造成误动作。

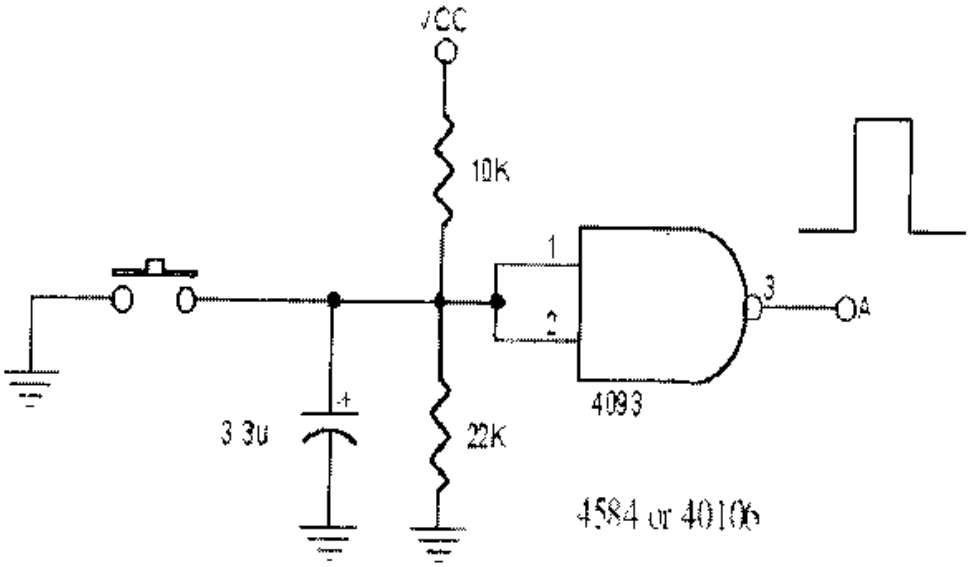
电路



说明

当 A 端有一个短暂的脉冲或杂波(必须比输出波形的脉冲宽度还要窄)，则将会有一稳定的输出正脉冲(Q 脚位输出)及负脉冲，而脉冲宽度 T 的时间由 Rt、Ct 所决定，当 Rt、Ct 的数值越大，则脉冲宽度 T 的时间也越长。

电路



说明

使用史密特触发器也可以输出稳定的脉冲，当开关按一下则 A 端将会有一个稳定的正脉冲。

➢ Timer0ISR_2( )

目的

按三次开关才让 LED 闪烁一下。

程序

```
01 //count 3 time,execute service route
02 void  Timer0ISR_2 (void) interrupt 1 using 3
03 {
04     PulseCount++;
05     if (PulseCount == 3 )
06
```

```
07        PulseCount=0;
08        LedOn( );
09        DelayX10ms(50);
10        LedOff( );
11        DelayX10ms(50);
12    }
13
14    TL0 = 0xff;
15    TH0 = 0xff;
16    TF0 = 0;
17 }
```

说明

初始程序和主程序的写法如同上一个范例，此程序也是每输入一个脉冲就立即产生中断，只是将计数器一直累加到三，才执行LED闪烁一次的程序。

行 号	说 明
04	每中断1次，就将计数值加1
05~12	如果计数值等于3，则执行LED闪烁一次，并清除计数值
14~16	重新加载计数值，因为TL0及TH0都等于0xff，所以计数值=0xffff-(TH0.TL0)+1=0xffff-0xffff+1=1，即计数1次就会产生中断

➢ CountMain2( )

目的

当计数器产生溢出时，则发生中断。

程序

```
01 //count 3 time,execute service route
02 void  CountMain2(void) //non-bounce
03 {
04    IE  = 0;          //disable all interrupt
```

```
05      IP  = 0x0b; //hi priority:int0,count0,timer1
06      TMOD= 0x15; //set timer1:model 16 bit
              timer,timer0:counter0
07
08      TL0 = 0xfd; //counter0=0xfffd,count 3 time,execute
              service route
09      TH0 = 0xff;
10
11      ET0 = 1;          //enable count0 interrupt
12      EA  = 1;          //enable all interrupt
13
14      TR0 = 1;          //start count0
15  }
16
17  void  Timer0ISR_3 (void) interrupt 1 using 3
18  {
19      LedOn( );
20      DelayX10ms(50);
21      LedOff( );
22      DelayX10ms(50);
23
24      TL0 = 0xfd; //count=3,then interrupt
25      TH0 = 0xff;
26      TF0 = 0;
27  }
```

说明

直接变更计数器的内容，当按了三次开关之后才产生中断，并让 LED 闪烁一下，但是开关的动作必须要经过处理，不能产生杂波，即按一次开关只能产生一个脉冲，如果开关的动作不经过硬件处理，则第一次按开关就会有杂波，就变成已经有三个以上的脉冲输入，则立即产生中断，LED 也会闪烁一下而造成误动作。

行 号	说 明
02~15	主程序，其中 TH0、TL0 设定为计数 3 次负脉冲才产生中断，并开始计数
17~27	计数中断例程，让 LED 闪烁一下并恢复计值，清除溢出标志

➢ CountMain3( )

目的

以中断方式计数外界输入脉冲的个数。

程序

```
01 void  CountMain3(void) //non-bounce
02 {
03     IE  = 0;          //disable all interrupt
04     IP  = 0x0b; //hi priority:int0,count0,timer1
05     TMOD= 0x15; //set timer1:model 16 bit
           timer,timer0:counter0
06
07     TL0 = 0xff; //counter0=0xffff,count 1 time,execute
           service route
08     TH0 = 0xff;
09
10     ET0 = 1;          //enable count0 interrupt
11     EA  = 1;          //enable all interrupt
12
13     TR0 = 1;          //start count0
14
15     PulseCount=0;     //initial count
16 }
17
18 //calculate pulse numbers
19 void  Timer0ISR_4 (void) interrupt 1 using 3
20 {
```

```
21      PulseCount++;   //16 bit data,PulseCount+1
22
23      TL0 = 0xff;
24      TH0 = 0xff;
25      TF0 = 0;
26  }
```

说明

当要计算外部的输入脉冲数量时，使用中断方式可以实时反应目前的状态，而且不用担心会遗漏或算不准的情况。

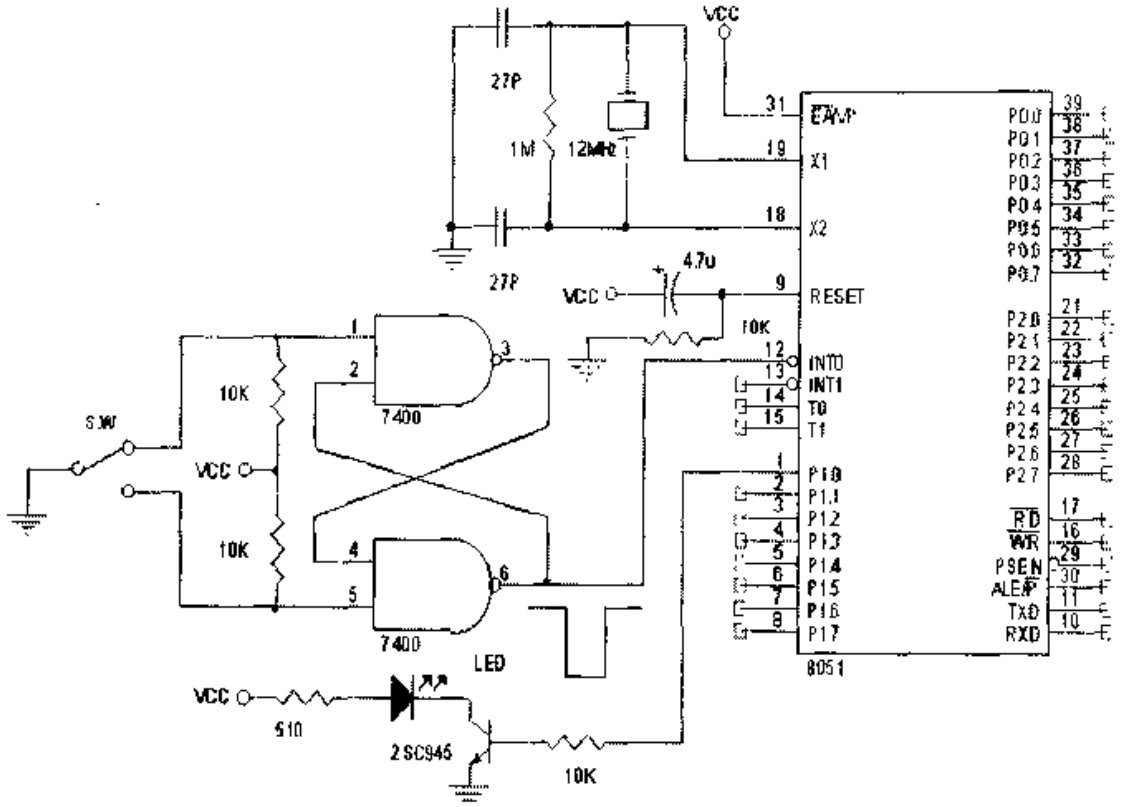
行　号	说明
04	设定中断来源的优先权
05	设定计时，计数模式为 TIMER1 及 COUNT0
07,08	计数一次就产生中断的缓存器数值
10~13	允许中断并开始计数
15	脉冲计数的内容先设定为 0
19	计数器中断例程
21	每计数器中断 1 次，则计数值就加 1
23,24	重新加载计数缓存器的内容
25	清除计数器中断溢出标志

➢ One_INT0ISR( )

目的

外部中断的应用。

电路



程序

```
/*******************************/
void  Int0Main1(void)  //non-bounce
{
    IE  = 0;     //disable all interrupt
    IP  = 0x0b; //hi priority:int0,count0,timer1

    IT0 = 1;         //set int0:falling eage trigger
    EX0 = 1;     //enable int0   interrupt
    EA  = 1;         //enable all interrupt
}

//One interrupt
void  One_INT0ISR (void) interrupt 0 using 3
{
    EX0 = 0;         //disable int0
```

```
16
17      LedOn( );
18      DelayX10ms(50);
19      LedOff( );
20      DelayX10ms(50);
21
22      EX0 = 1;          //enable next interrupt
23      IE0 = 0;          //set TCON.1 flag
24  }
```

说明

外部中断一般为下降缘的触发模式，而且为最高优先权。在实际应用中时常需要立即反应或异常处理，这时可以考虑用中断方式。

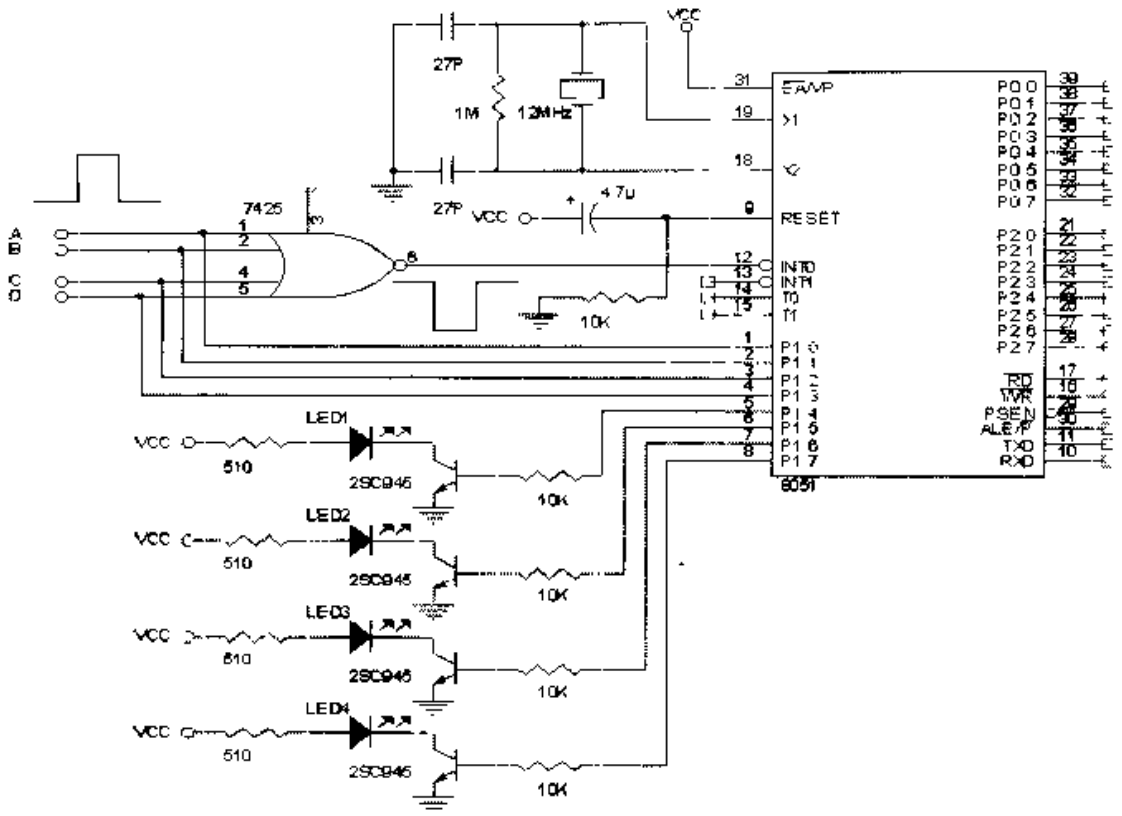
行　号	说　　明
04~08	中断应用的主程序，设定中断下降缘的触发模式并使能外部中断
09	必须使能所有的中断标志，才允许个别的中断动作
13	只有一个外部输入的中断例程
15	在执行中断例程时，不允许再产生同一来源的中断，以避免中断例程误动作
17~20	LED 闪烁一下
22	中断例程执行完毕后，恢复允许下一次的中断
23	清除外部中断溢出标志

➢ More_INT0ISR( )

目的

多个外部中断的应用。

电路



程序

```
01 //More interrupt
02 void  More_INT0ISR (void) interrupt 0 using 3
03 {
04    EX0 = 0;
05
06    if ( P1_0==1 )      //interrupt 1
07    {
08       Led1On( );
09       DelayX10ms(50);
10       Led1Off( );
11       DelayX10ms(50);
12    }
13    else if ( P1_1==1 ) //interrupt 2
14    {
15       Led2On( );
```

```
16        DelayX10ms(50);
17        Led2Off( );
18        DelayX10ms(50);
19     }
20     else if ( P1_2==1 ) //interrupt 3
21     {
22        Led3On( );
23        DelayX10ms(50);
24        Led3Off( );
25        DelayX10ms(50);
26     }
27     else if ( P1_3==1 ) //interrupt 4
28     {
29        Led4On( );
30        DelayX10ms(50);
31        Led4Off( );
32        DelayX10ms(50);
33     }
34
35     EX0 = 1;
36     IE0 = 0;              //set TCON.1 flag
37 }
38
39 void Led1On(void)
40 {
41     P1_4=1;
42 }
43 void Led1Off(void)
44 {
45     P1_4=0;
46 }
47
48 void Led2On(void)
49 {
50     P1_5=1;
```

```
51 }
52 void Led2Off(void)
53 {
54     P1_5=0;
55 }
56
57 void Led3On(void)
58 {
59     P1_6=1;
60 }
61 void Led3Off(void)
62 {
63     P1_6=0;
64 }
65
66 void Led4On(void)
67 {
68     P1_7=1;
69 }
70 void Led4Off(void)
71 {
72     P1_7=0;
73 }
```

说明

一般微处理器的外部中断输入源可能只有一组或二组而已，但往往在实际应用中常常有很多个外界的输入，都必须作实时的反应与处理。这时，可以在硬件线路上使用 NOR 门，只要有任何一个输入源有输入即产生中断，再由程序对不同的输入分别处理。

行 号	说 明
02	多个中断输入源处理例程
04	不允许中断
06~12	判断是否由 A 输入源来产生中断，如果是，则 LED1 闪烁一下
13~19	判断是否由 B 输入源来产生中断，如果是，则 LED2 闪烁一下

行　号	说　　明
20~26	判断是否由 C 输入源来产生中断，如果是，则 LED3 闪烁一下
27~33	判断是否由 D 输入源来产生中断，如果是，则 LED4 闪烁一下
35,36	允许下一次中断并清除溢出标志
41	LED1 亮
45	LED1 熄
50	LED2 亮
54	LED2 熄
59	LED3 亮
63	LED3 熄
68	LED4 亮
72	LED4 熄

➢ Timer1ISR_1( )

目的

用 TIMER1 来作为系统的内部计时器。

程序

```
01 //timer=40ms,interrupt 1 time
02 void  Timer1ISR_1 (void) interrupt 3 using 2
03 {
04     TL1 = CLOCK_40MS & 0xff;  //CLOCK_40MS=(65536 - 40000)
05     TH1 = CLOCK_40MS >> 8;
06     TF1 = 0;
07
08     Timer40msCount++;
09
10     if ( LedOffCount != 0 )   //every 40ms,then coumter-1
11        LedOffCount--;       //if coumter=0 then timer ok
12 }
```

说明

在设计程序时，大部分应用都需要定时器，例如 10 秒后将 LED 熄灭，或者在不按键的情况下 15 秒后恢复 OSD(On Screen Display)界面等等，因此在构想规划程序时，必须要有计时的例程，而且大部分的微处理器本身都有定时器，也提供使用者多种的应用模式。

行 号	说　　明
04,05	每 40mS 中断一次，即系统中最小的计时单位
06	清除溢出标志
08	每 40mS 之后则变量值加 1
10,11	计时的时间数值每 40mS 减 1，当此变量 LedOffCount 的数值等于 0，就表示计时的时间终了，也不用再减 1 了。

第 7 章　公用函数

➢ UnSignVar(　)

目的

无符号整数进位时数值的变化。

程序

```
01 //datatype:unsigned char (Byte)
02 void UnSignVar(void)
03 {
04    Byte  i=0x00;   //when call the routine, initial i=0xCC,
05
06    while( 1 )
07                .
08       i++;      //i=0x00~0xff,value=(0~255)
09    }
10 }
```

说明

Byte 是 unsigned char 类型，数值范围从 0~255，如果 i 变量一直累加至 255 时，当 i 又加 1 时，i 变量的值会变成 0。

➢ SignVar(　)

目的

有符号整数进位时数值的变化。

程序

```
01 //datatype:char
02 void SignVar(void)
03 {
04     char  i=0x00;
05
06     while( 1 )
07
08        i++;      //i=0x00~0xff,value=(-128~+127)
09     }
10 }
```

说明

char是signed char的类型，数值范围从-128~127，以bit7来作为正负号，当bit7="0"则为正，bit7="1"则为负，如果i变量一直累加至127时，当i又加一，则i变量的内容会变成-128，以十六进制来表示则数值为0x80。

➢ ByteVariableAdd1( )

目的

一个字节的变量和固定常数相加，即8 bit数值作加法运算。

程序

```
01 //byte operate,MaxValue=Word datatype
02 void ByteVariableAdd1(void)
03 {
04     Byte  temp;
05
06     temp = (*ByteVariablePtr);
07     if ( (temp + UpdateValue) <  (Byte)MaxValue )
08        temp += UpdateValue;
09     else
10        temp  = LOBYTE(MaxValue);
```

```
11
12      *ByteVariablePtr = temp;
13  }
```

说明

在数值运算中变量值加上一个固定常数或减去一个固定常数是非常普遍的，程序设计上一般都有值域的限制，即变量值如果小于最小值，则变量值将设定为最小值，变量值如果大于最大值，则变量值将限制在最大值。

行 号	说 明
06	将变量值的内容暂存至 temp，ByteVariablePtr 会指到实际要运算的 Byte 变量，通常此变量会在调用程序被指定。
07~10	变量值的内容加上固定常数，如果小于最大值则相加起来，如果大于最大值则此变量等于最大值，其中 UpdateValue 是固定常数值的变量，而 MaxValue 为最大值的变量
12	将运算的结果存入变量内，以指针类型作运算其弹性大，可随意指向需要作运算的变量的地址

➢ ByteVariableAdd2( )

目的

一个 Byte 的变量和固定常数相加，而以 Word 的方式运算。

程序

```
01 //word operate,MaxValue=Word datatype
02 void ByteVariableAdd2(void)
03 {
04     int  temp;
05
06     temp = (Word) (*ByteVariablePtr);
07     if ( (temp + UpdateValue) < (int)MaxValue )
08        temp += UpdateValue;
09     else
10        temp  = MaxValue;
```

```
11
12      *ByteVariablePtr = LOBYTE(temp);
13  }
```

说明

此程序为有符号整数 16 bit 相加，只要运算结果小于 0x8000 都是正确的，但如果运算结果有可能超过 0x8000，则需要将有符号整数类型 int 改成无符号整数类型 Word，会用到有符号整数的运算，是因为在别的函数里，程序会检测正负号而作不同的应用。

行 号	说 明
04	设定 LOCAL 变量 temp 为有符号整数，LOCAL 变量的特性为在此子程序执行完毕退出后，此变量就消失了；而 GLOBAL 变量的特性是不管程序执行到哪里，GLOBAL 变量的内容都还存在不会消失。
06	暂存变量的内容
07~10	判断相加后的结果是否小于最大值，是则相加；否则变量的内容等于最大值
11	将运算后的低字节存入变量中

➢ ByteVariableSub( )

目的

一个 Byte 的变量和固定常数相减，即 8 bit 数值作减法而用 Word 类型作运算。

程序

```
01 void ByteVariableSub(void)
02 {
03     int  temp;
04
05     temp = (Word) (*ByteVariablePtr);
06     if ( (temp - UpdateValue) > (int)MinValue )
07        temp -= UpdateValue;
08     else
```

```
09        temp  = MinValue;
10
11     *ByteVariablePtr = LOBYTE(temp);
12 }
```

说明

此程序相似于作加法运算的说明，也是以 Word 的方式作运算，但最后结果取其低字节。

行 号	说　　明
05	暂存变量的内容至 temp
06	判断相减后的结果是否大于最小值
07	是，则执行相减运算
09	否，则变量的内容等于最小值
11	将运算后的结果，取低字节存入指针变量中

➢ ByteProcess(　)

目的

将 Byte 变量的加减法运算写成一个函数，以利阅读和节省程序内存空间。

程序

```
01 void ByteProcess(void)
02 {
03     int  temp;
04
05     temp = (Word) (*ByteVariablePtr);
06     if ( Fgdirection ==1 )
07     {
08       if ( (temp - UpdateValue) > (int)MinValue )
09          temp -= UpdateValue;
10       else
11          temp  = MinValue;
```

```
12     }
13     else
14     {
15        if ( (temp + UpdateValue) < (int)MaxValue )
16           temp += UpdateValue;
17        else
18           temp  = MaxValue;
19     }
20
21     *ByteVariablePtr = LOBYTE(temp);
22 }
```

说明

依据 Fgdirection 方向标志变量作加法或减法的运算，即将上列单独的加法程序和减法程序合并在一起。

行 号	说 明
05	暂存变量的内容至 temp
06~12	方向标志变量等于 1，则作减法的运算
13~19	方向标志变量不等于 1 (可以等于 0)，则作加法的运算
21	将运算后的结果，取低字节存入指针变量中

➢ WordVariableAdd1( )

目的

二个 Byte 的变量和固定常数相加，即 16 bit 加法运算（有潜在错误可能）。

程序

```
01 //error coding,please notics,MaxValue=Word datatype
02 void WordVariableAdd1(void)
03 {
04    Word  temp;
05
```

```
06     temp = *WordVariablePtr;
07
08     if ( (temp + UpdateValue) < MaxValue )
                          //0xffff+0x0001=0x0000
09        temp += UpdateValue;
10     else
11        temp  = MaxValue;
12
13     *WordVariablePtr = temp;
14 }
```

说明

此程序有潜在的错误，请程序设计人员注意。将 temp 声明成无符号整数变量，则当十六进制 0xffff 加 1 后，结果会变成 0，而造成错误，正确的写法应该将 temp 变量声明成 int 类型，如下面程序所示。

行 号	说　　明
06	暂存 Word 变量的内容至 temp
08~11	如果 MaxValue 最大值为 0xffff，则此条件会造成错误，因为当变量 temp 等于 0xffff 又再加 1 后，则 temp 变量会变成 0，导致指针内的变量也变成 0，正确的 temp 变量应该还是要等于 0xffff，即最大值变量 MaxValue。
13	将运算后的结果，存入指针变量中

➢ WordVariableAdd2( )

目的

二个 Byte 的变量和固定常数相加，即 16 bit 加法运算（正确完整的程序写法）。

程序

```
01 //modify following
02 void WordVariableAdd2(void)
03 {
```

```
04     int  temp;
05
06     temp = *WordVariablePtr;
07
08     if ( (temp + UpdateValue) < (int)MaxValue )
09        temp += UpdateValue;
10     else
11        temp  = MaxValue;
12
13     *WordVariablePtr = temp;
14 }
```

说明

Word 类型变量的加法运算，此程序没有虫（BUG）可作为实际的应用程序，Byte 类型的变量范围数值从 0~255 常常不够使用，因而要使用到 Word 类型的变量来做运算，此程序即是一例。

行 号	说　　明
04	暂存变量 temp 为有符号整数的类型
08	假如相加的结果小于最大值
09	条件成立，则执行相加，即变量值加上固定常数
11	条件不成立，变量取最大值
13	将运算后的结果，存入 Word 指针变量中

➢ WordVariableSub1( )

目的

二个 Byte 的变量和固定常数相减，即 16 bit 减法运算（有潜在错误）。

程序

```
01 //error coding,please notics,MinValue=Word datatype
02 void WordVariableSub1(void)
03 {
04     Word  temp;
```

```
05
06     temp = *WordVariablePtr;
07
08     if ( (temp - UpdateValue) > MinValue ) //0x0000
0x0001=0xffff
09        temp -= UpdateValue;
10     else
11        temp  = MinValue;
12
13     *WordVariablePtr = temp;
14 }
```

说明

如果 temp 变量等于 0 再减 1 后，temp 变量会变成 0xffff 而造成错误，不要使用此程序的设计写法，请参考下一个范例才是正确的。

➢ WordVariableSub2( )

目的

二个 Byte 的变量和固定常数相减，即 16 bit 减法运算（正确完整的程序写法）。

程序

```
01 //modify following
02 void WordVariableSub2(void)
03 {
04     int  temp;
05
06     temp = *WordVariablePtr;
07
08     if ( (temp - UpdateValue) > (int)MinValue )
09        temp -= UpdateValue;
10     else
11        temp  = MinValue;
```

```
12
13      *WordVariablePtr = temp;
14  }
```

说明

将LOCAL变量temp声明成int类型，使得程序变得更完整和确实。

行 号	说 明
04	temp声明成int类型
08~11	如果相减的结果大于最小值，则执行减法运算，否则取最小值
13	将运算后的结果，存入Word指针变量中

➢ WordProcess( )

目的

将Word变量的加减法运算写成一个函数，以利阅读和节省程序内存空间。

程序

```
01 void WordProcess(void)
02 {
03     int  temp;
04
05     temp = *WordVariablePtr;
06     if (  Fgdirection ==1 )
07     {
08        if ( (temp - UpdateValue) > (int)MinValue )
09           temp -= UpdateValue;
10        else
11           temp  = MinValue;
12     }
13     else
14     {
15        if ( (temp + UpdateValue) < (int)MaxValue )
```

```
16          temp += UpdateValue;
17        else
18          temp  = MaxValue;
19      }
20
21      *WordVariablePtr = temp;
22  }
```

行　号	说　　明
03	temp 声明成 int 类型
05	将所指到的变量的内容暂存至 temp 变量内
06~12	方向标志变量等于 1，则做减法的运算，即变量值减去常数值
13~19	方向标志变量等于 0，则做加法的运算，即变量值加上常数值
21	将运算后的结果，存入 Word 指针变量中

➢ Hex2Bcd1(value)

目的

将变量值的个位数和十位数分别取出来，以利应用。

参数

value：待转换的数值，范围从 0~99

程序

```
01 //2 digital,Keydata1,Keydata2 is global varible
02 void Hex2Bcd1(Byte value)
03 {
04     Byte number2,number1;
05
06     number2=value/10;
07     number1=(value-=number2*10);
08
09     KeyData2=number2;
```

```
10     KeyData1=number1;
11 }
```

说明

有许多应用需要将变量值显示出来，例如：OSD 的应用，要将其计数值显示在屏幕上，则须将变量一个字符紧接一个字符的依次显示，所以必须将个位数、十位数、百位数或千位数分别取出来。例如: value=0x43，其十进制的数值为 67，当作完此程序后则 KeyData2=6、KeyData1=7、其中 KeyData2、KeyData1 是公用变量或称为全域变量(Global Variable)，在此为说明使用并无其它目的。

行 号	说　明
04	声明 LOCAL 变量 number2、number1 作为暂存变量
06	自变量(即待转换的数值)的十位数将之取出来
07	自变量(即待转换的数值)的个位数将之取出来
09,10	公用变量，可观察程序执行结果和验证程序的正确性与否

➢ Hex2Bcd2(value)

目的

将变量值的个位数、十位数和百位数分别取出来，以利应用。

参数

value：待转换的数值，范围从 0~999。

程序

```
01 //3 digital,Keydata1,Keydata2,Keydata3 is global varible
02 void Hex2Bcd2(Word value)
03 {
04     Byte number3,number2,number1;
05
06     number3=value/100;
07     number2=(value-=number3*100)/10;
08     number1=(value-=number2*10);
09
```

```
10      KeyData3=number3;
11      KeyData2=number2;
12      KeyData1=number1;
13  }
```

行 号	说 明
04	暂存 LOCAL 变量
06	自变量(即待转换的数值)的百位数将之取出来
07	自变量(即待转换的数值)的十位数将之取出来
08	自变量(即待转换的数值)的个位数将之取出来
10~12	取出来的数值将之存入公用变量内，以利验证

➢ Hex2Bcd3(value)

目的

将变量值的个位数、十位数、百位数和千位数分别取出来，以利应用。

参数

value：待转换的数值，范围从 0~9999。

程序

```
01 //4 digital,Keydata1,Keydata2,Keydata3,Keydata4 is global variblc
02 void Hex2Bcd3(Word value)
03 {
04      Byte number4,number3,number2,number1;
05
06      number4=value/1000;
07      number3=(value-=number4*1000)/100;
08      number2=(value-=number3*100)/10;
09      number1=(value-=number2*10);
10
11      KeyData4=number4;
```

```
12      KeyData3=number3;
13      KeyData2=number2;
14      KeyData1=number1;
15  }
```

行 号	说　　明
04	暂存 LOCAL 变量
06	自变量(即待转换的数值)的千位数将它取出来
07	自变量(即待转换的数值)的百位数将它取出来
08	自变量(即待转换的数值)的十位数将它取出来
09	自变量(即待转换的数值)的个位数将它取出来
11~14	取出来的数值将之存入公用变量内，以利验证

➢ Value255_100(value)

目的

将变量值范围从 0~255 比例压缩到 0~100 的数值范围。

参数

value：待转换的变量，数值范围从 0~255。

程序

```
01 //value=0~255 change to  0~100
02 void Value255_100(Byte value)
03 {
04     CurrentValue = value;
05     MinValue = 0;
06     MaxValue = 255;
07     CurrentValue = (Byte)( ( (CurrentValue-MinValue)*100)
                       /(MaxValue-MinValue) );
08
09 }
```

说明

一个以 Byte 类型声明的变量值，其数值范围从 0~255，但数值显示在屏幕上反而以十进制 0~100 的变化较为亲切，也因此才需要将变量值作转换。例如：如果 value=0xff 则转换后的数值为 100；value=0x80 则转换后的数值为 50；value=0x00 则转换后的数值也为 0，请参考行号 07 的转换公式。

行　号	说　　明
04	暂存要转换的变量
05	待转换变量的最小值
06	待转换变量的最大值
07	变量值转换成 0~100 的公式，请利用比例缩小的数学式来运算

➢ Value100_128a(value)

目的

将变量值范围从 0~100 比例放大到 0~128 的数值范围（有潜在错误）。

参数

value：待转换的变量，数值范围介于 0~100 才会正确。

程序

```
01 //value=0~100 change to  0~128
02 //error condition:when value>100,then CurrentValue is error
03 void Value100_128a(Byte value)
04 {
05     CurrentValue = value;
06     MinValue = 0;
07     MaxValue = 100;
08     CurrentValue = (Byte)( ( (CurrentValue-MinValue)*128)
                        /(MaxValue-MinValue) );
09
10 }
```

说明

当 value 大于 100 时，则转换后的数值也必大于 128，而当 value 大于 200 时，则转换后的数值会溢出反而更小，因此待转换数值必须要作判别，才能避免错误，如下一个范例所示。

➢ Value100_128b(value)

目的

将变量值范围从 0~100 比例放大到 0~128 的数值范围（实用）。

参数

value：待转换的变量，数值范围介于 0~100。

程序

```
01 void Value100_128b(Byte value)
02 {
03    if (value>100)
04       value=100;
05    CurrentValue = value;
06    MinValue = 0;
07    MaxValue = 100;
08    CurrentValue = (Byte)( ( (CurrentValue-MinValue)*128)
                  /(MaxValue-MinValue) );
09
10 }
```

行 号	说　明
03,04	事先将自变量做判断，如果大于 100 则将设定为最大值等于 100，以避免错误
05	暂存要转换的变量
06	待转换变量的最小值
07	待转换变量的最大值
08	变量值放入成 0~128 的公式

➢ RamClear( )

目的

将 8052 内部 RAM，从 0x20~0xff 的地址清除为 0。

程序

```
01 //clear cpu internal ram address:0x20~0xff
02 Byte idata address _at_ 0x20;
03 void RamClear(void)
04
05    Byte idata *index=&address;
06    Byte i;
07
08    for(i=0;i<(0xff-0x20+1);i++)
09
10       *index=0x00;
11       index++;
12    }
13 }
```

说明

系统开机即进入 main( )的起始函数，通常需要将微处理器的内部 RAM (Random-Access Memory：随机存取内存)都清除为 0，以确保变量起始值的正确性，也可以将使用到的变量一一清除为 0，但如果系统大，使用的变量多，反而会占用程序较多的内存空间。

行 号	说　　明
02	将绝对地址 0x20 设定给 address 变量，即索引的起始地址，因为 8052 微处理器可用的 RAM 地址介于 0x20~0xff 之间
05	使用指针的方式来处理，将 RAM 的起始地址指给 index
08	为了清除 0x20~0xff 之间地址的循环次数
10	将目前指到的地址 0x20 清除为 0
11	将 RAM 地址加 1

➢ ZeroContinue(counter)

目的

数组中有数值不等于 0 的，则立即跳出返回调用程序，如果数值等于 0 则继续测试下一个数组。

参数

counter：数组的大小

程序

```
01 //if TrmBuf[  ]=0,then next,total 4 times
02 //if not zero,then return
03 void ZeroContinue(Byte counter)
04 {
05    Byte i;
06
07    for(i=0; i<4; i++)
08    {
09       if (TrmBuf[counter]!=0)
10          break;     //if not zero then exit loop
11       counter++;
12       if (counter>4) counter=1;
13    }
14 }
```

行 号	说　明
02	将绝对地址 0x20 设定给 address 变量，即索引的起始地址，因为 8052 微处理器可用的 RAM 地址介于 0x20~0xff 之间
05	使用指针的方式来处理，将 RAM 的起始地址指给 index
08	为了清除 0x20~0xff 之间地址的循环次数
10	将目前指到的地址 0x20 清除为 0
11	将 RAM 地址加 1

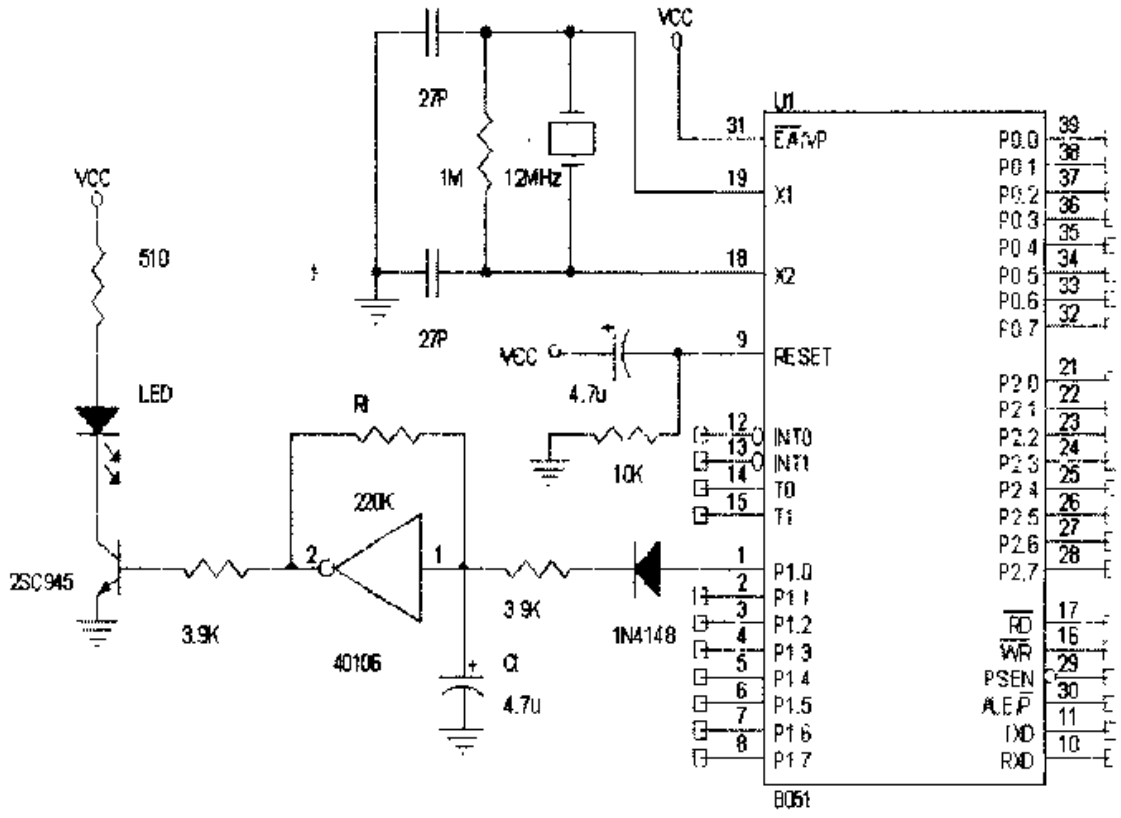
第 8 章　显示器的应用

➢ LedFlash0(　)

目的

以硬件来控制 LED 闪烁

电路



程序

```
01 void LedFlash0(void)
02 {
03     P1_0=0;              //hardware,"0":flash
```

```
04 }
```

说明

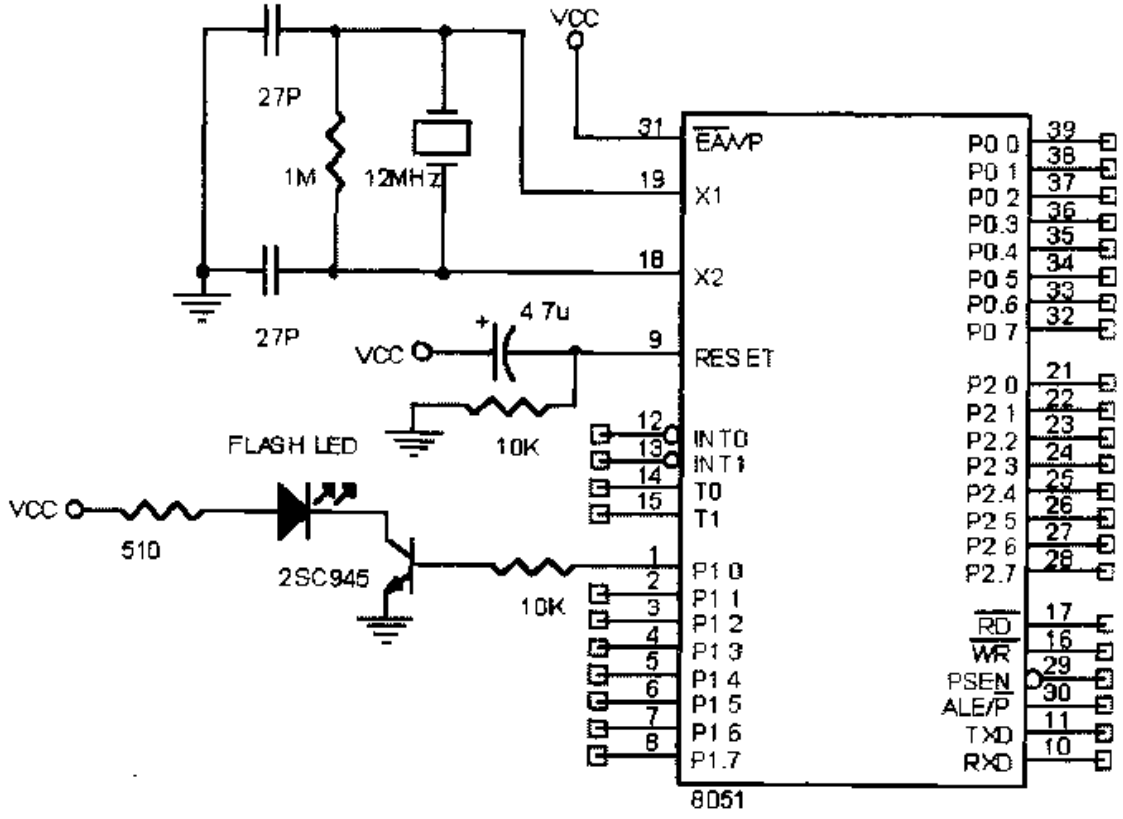
要让 LED (Light Emitting Diode：发光二极管) 闪烁除了设计软件外，也可以使用振荡电路来完成，仅使用硬件线路其成本较高，其中 Rt 反馈电阻及 Ct 控制着 LED 闪烁的频率，当 P1.0 输出为“0”电平时，则 LED 开始闪烁，当 P1.0 输出为 “1”电平时，则 LED 熄灭。

➢ LedFlash1( )

目的

直接使用闪烁型 LED，以作为提示作用。

电路



程序

```
01 void LedFlash1(void)
02 {
```

```
03     P1_0=1;              //flash led
04 }
```

说明

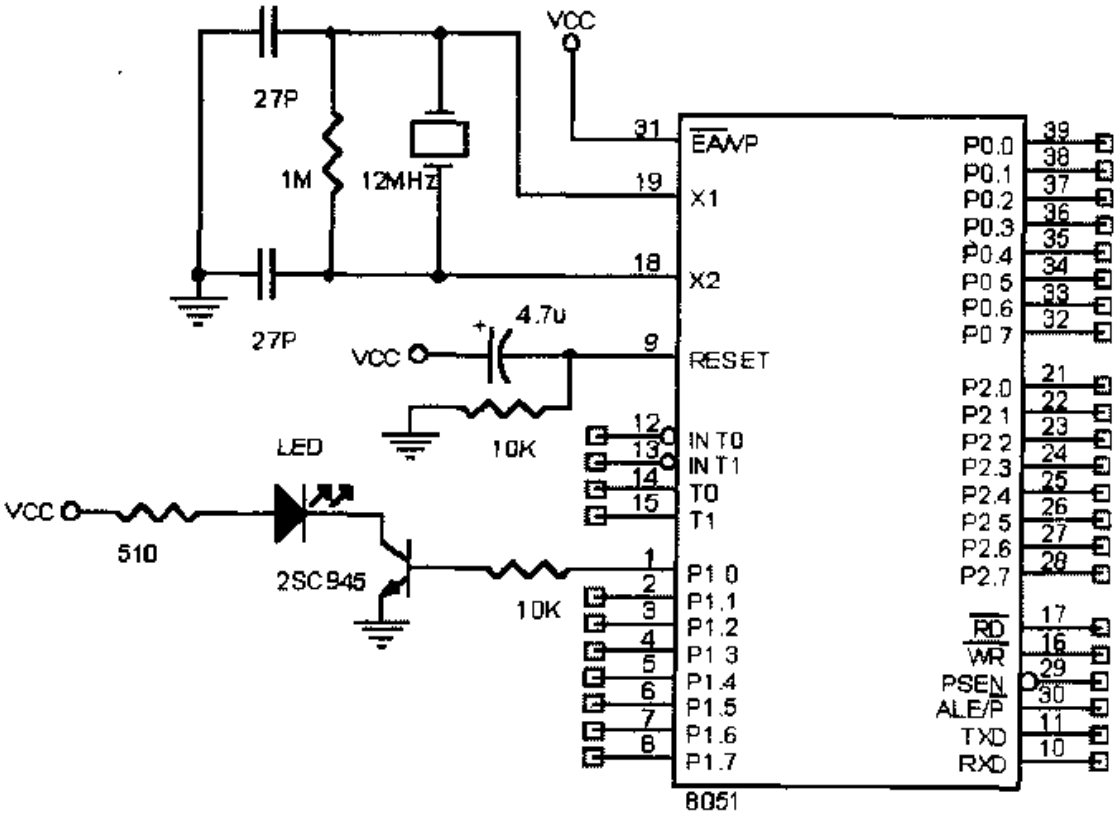
直接使用闪烁型 LED，程序写法变得非常简单，只须将 P1.0 输出为“1”电平，则 LED 就会开始闪烁，闪烁频率固定，大约 1 秒闪烁一下，当 P1.0 输出为“0” 电平，则 LED 熄灭，使用闪烁型 LED 缺点是成本较高。

➢ LedFlash2( )

目的

使用一般型 LED，以软件达到闪烁一下的功效。

电路



程序

```
01 void  LedFlash2(void)
```

```
02 {
03     LedOn( );
04     DelayX10ms(50); //ledon 500ms
05     LedOff( );
06     DelayX10ms(50); //ledoff 500ms
07 }
```

说明

使用一般型 LED，其优点是零件成本低，但软件不像硬件方式或闪烁型 LED 那么简单，只须设定 I/O 口即可，当 P1.0 输出为“1”则 LED 亮，当 P1.0 输出为“0”则 LED 熄，因此软件必须作“1”及“0”的变化，以达到闪烁的功效。

行 号	说　明
03	LedOn( )函数，即是设定 I/O 口 P1.0=“1”，则 LED 亮
04	当 LED 亮时，则程序作延迟的时间，就是 LED 亮的时间，此函数为延迟子程序，总共延迟 10mS 乘以参数 50 等于 500mS，即 0.5 秒。
05	LedOff( )函数，即是设定 I/O 口 P1.0=“0”，则 LED 熄灭
06	如同行号 04，为延迟子程序

➢ LedFlash3( )

目的

让 LED 一直闪烁不停。

程序

```
01 void  LedFlash3(void)
02 {
03    while( 1 )
04    {
05       LedOn( );
06       DelayX10ms(50);
07       LedOff( );
08       DelayX10ms(50);
09    }
```

```
10 }
```

说明

由于行号 03 的 while 循环且条件永远成真(零是假，非零即真)，因此会一直作行号 05~08 的动作，即 LED 闪烁不停，此程序只是为说明用，实际应用并不会如此。

➢ LedFlash4(ontime,offtime)

目的

以参数的方式，让 LED 闪烁一下。

参数

ontime：LED 亮的持续时间
offtime：LED 熄灭的持续时间

程序

```
01 void  LedFlash4(Byte ontime,Byte offtime)
02 {
03     LedOn( );
04     DelayX10ms(ontime );
05     LedOff( );
06     DelayX10ms(offtime);
07 }
```

说明

此程序可用参数控制 LED 亮的时间和 LED 熄灭的时间，行号 04 即控制着 LED 亮的时间，参数 ontime 数值小则 LED 亮的时间短，而参数 ontime 数值大则 LED 亮的时间长；同理行号 06，控制着 LED 熄的时间，当参数 offtime 数值小则 LED 熄的时间短，当参数 offtime 数值大则 LED 熄的时间长。在 C 语言里使用参数来传递数据是很好的程序设计方式而且其弹性非常大，可以节省很多重复的程序编写过程，建议读者常以此方式来设计程序。

➢ LedFlash5(count,ontime,offtime)

目的

可控制 LED 闪烁几次的程序。

参数

count：LED 闪烁几次
ontime：LED 亮的持续时间
offtime：LED 熄灭的持续时间

程序

```
01 void  LedFlash5(Byte count,Byte ontime,Byte offtime)
02 {
03    Byte  i;
04
05    for(i=0; i<count; i++)
06    {
07       LedOn( );
08       DelayX10ms(ontime );
09       LedOff( );
10       DelayX10ms(offtime);
11    }
12 }
```

说明

此程序如同上一个范例，只是多了一个控制 LED 闪烁几次的参数，行号 05 是让 LED 闪烁几次的控制循环，而以参数 count 是否结束循环的判断值作为依据。

关于控制 LED 闪烁的程序，是希望编写程序时以循序渐进的方式，学得如何正确的推理，并扩展思路，臻于细密与完整。其它大部分的程序，我也将不厌其烦的利用此渐进方式，来说明编写程序的技巧与思考方法。

➢ LedFlash6(count,ontime,offtime)

目的

LED 闪烁的总时间，并不影响计时时间。

参数

count：LED 闪烁几次
ontime：LED 亮的持续时间
offtime：LED 熄灭的持续时间

程序

```
01 void  LedFlash6(Byte count,Byte ontime,Byte offtime)
02 {
03     Byte  i;
04
05     TR1 = 0;          //stop timer,while led flash
06     for(i=0; i<count; i++)
07     {
08        LedOn( );
09        DelayX10ms(ontime );
10        LedOff( );
11        DelayX10ms(offtime);
12     }
13     TR1 = 1;          //led flash finished,then start timer
14 }
```

说明

一般在实际应用中都会有计时例程，为了避免 LED 闪烁的总时间影响计时时间，致使造成动作不确实，因此要做停止计时的处理。

行　号	说　　明
05	停止计时例程，在执行 LED 闪烁的程序前先设定停止计时
13	当 LED 闪烁执行完毕时，就恢复计时

➢ LedFlashGetkey(count,ontime,offtime)

目的

在 LED 闪烁例程中，一样可以接受按键的动作。

参数

count：LED 闪烁几次

ontime：LED 亮的持续时间

offtime：LED 熄灭的持续时间

程序

```
01 void  LedFlashGetkey(Byte count,Byte ontime,Byte offtime)
02 {
03    Byte  i;
04
05    TR1 = 0;
06    for(i=0; i<count; i++)
07    {
08       LedOn( );
09       DelayX10ms(ontime );
10       GetKey( );
11       if ( KeyData != NO_KEY )
12          break;  //if any key input,then exit "for" loop
13
14       LedOff( );
15       DelayX10ms(offtime);
16       GetKey( );
17       if ( KeyData != NO_KEY )
18          break;
19    }
20
21    LedOff( );  //exit loop,then led off
22    TR1 = 1;
23 }
```

说明

LED 闪烁一般作为提示或警示之用，如果 LED 闪烁 20 次，一次一秒钟，则 LED 闪烁例程将花费 20 秒的时间，在这 20 秒内，如果有异常也要作检测与处理，也因此需要在 LED 闪烁例程中加入判断。

行 号	说　　明
05	停止计时
06	LED 闪烁循环次数
08	LED 亮
09	LED 亮的时间
10	检测是否有按键输入的处理子程序
11,12	如果按任意键，则立即跳出 for 循环
14	LED 熄
15	LED 熄灭的延迟子程序
16	再一次检查是否有按键输入
17,18	按了任意键，则立即跳离 for 循环
21	跳离 for 循环后，必须让 LED 熄灭
22	恢复计时

➢ LedMain1( )

目的

将 LED 持续亮 5 秒钟。

程序

```
01 void  LedMain1(void)
02 {
03     Subroute1( );        //other program
04
05     LedOn( );
06     DelayX10ms(500);     //will be trouble,because wait 5s
```

```
07      LedOff( );
08
09      Subroute2( );
10  }
```

说明

此程序只为说明问题，并不是一个很好的应用程序，如果在 LED 亮 5 秒钟内，系统有任何的变化或事件要处理时，则将会漏掉此事件，因为此程序没有作其它事件的检测，很容易出错。

行 号	说　　明
03	实际应用中会调用其它子程序，仅供范例说明
05	LED 亮
06	延迟子程序，参数传了 500，即 10mS 乘以 500 等于 5 秒，占用了 CPU 5 秒的时间，什么也没有做
07	LED 熄灭
09	其它子程序，仅供范例说明

➢ LedMain2( )

目的

使用计时中断例程，可以达到 LED 亮 5 秒的功能，又可以处理任何事件而不会造成遗漏或错误。

程序

```
01 void  LedMain2(void)    //timer method
02 {
03      IE  = 0;          //disable all interrupt
04      IP  = 0x0b;       //hi priority:int0,count0,timer1
05      TMOD= 0x15;       //set        timer1:mode1        16        bit
timer,timer0:counter0
06
07      TL1 = CLOCK_40MS &0xff;    //timer1:40mS
```

```
08      TH1 - CLOCK_40MS >> 8;
09
10      ET1 = 1;              //enable timer1 interrupt
11      EA  = 1;                  //enable all interrupt
12
13      LedOn();
14      FgLedFlag  = 0;               //clr flag
15      LedOffCount=TIME_5SEC;        //set led off time=5s
16      TR1 = 1;                  //start timer
17
18      while( 1 )
19      {
20
21        Subroute1();
22
23        if ( FgLedFlag==0 )
24        {
25          if ( LedOffCount==0 ) //timer complete,led off
26          {
27            FgLedFlag = 1;        //set flag
28            LedOff();
29          }
30        }
31
32        Subroute2();
33      }
34  }
35  void  LedOff_Timer1ISR (void) interrupt 3 using 2
36  {
37      TL1 = CLOCK_40MS & 0xff;  //timer=40ms,interrupt 1 time
38      TH1 = CLOCK_40MS >> 8;
39      TF1 = 0;
40
41      Timer40msCount++;
42
```

```
43     if ( LedOffCount != 0 )
44       LedOffCount--;     //after 40ms,then coumter-1
45 }
```

说明

建议大部分需要长时间延迟的，都改用计时例程的方式来处理，虽然程序结构较为复杂，但绝对不会出错，且任何的事件都可立即处理，是一个完整且实用的程序架构，千万不要贪求程序简单好写，而使用上一个以延迟函数。

行 号	说 明
01	利用 TIMER 的主程序
03~05	设定 8051 的控制缓存器，设定 TIMER1 为 16 bit 定时器
07,08	每 40mS TIMER1 中断一次，其中 TH1 为高位，TL1 为低位
10,11	使能 TIMER1 中断，并使能所有的中断
13	LED 亮
14	清除 LED 标志，用以做判别
15	设定 LED 熄灭的计时时间为 5 秒
16	开始计时
18	此条件式永远成立，因此会不断的重复此循环
21	其它子程序
23	如果 LED 标志等于 0 则继续做下去，如果 LED 亮 5 秒的时间终了，则此标志变量会设定为“1”，因此就不会继续执行括号内的语句了
25	5 秒的计时例程终了，此 LedOffCount 将会等于 0，也就会执行括号内的叙述，令 LED 熄灭
27	设定标志变量为“1”，即表示 5 秒的计时时间终了
28	令 LED 熄灭，因为计时 5 秒的时间已经结束了
35	计时中断例程
37,38	重新又加载中断的缓存器内容
39	中断发生后，则相对应的 bit 会自动设定为“1”，因此执行中断例程后，必须用软件来清除之，以利下一次的中断

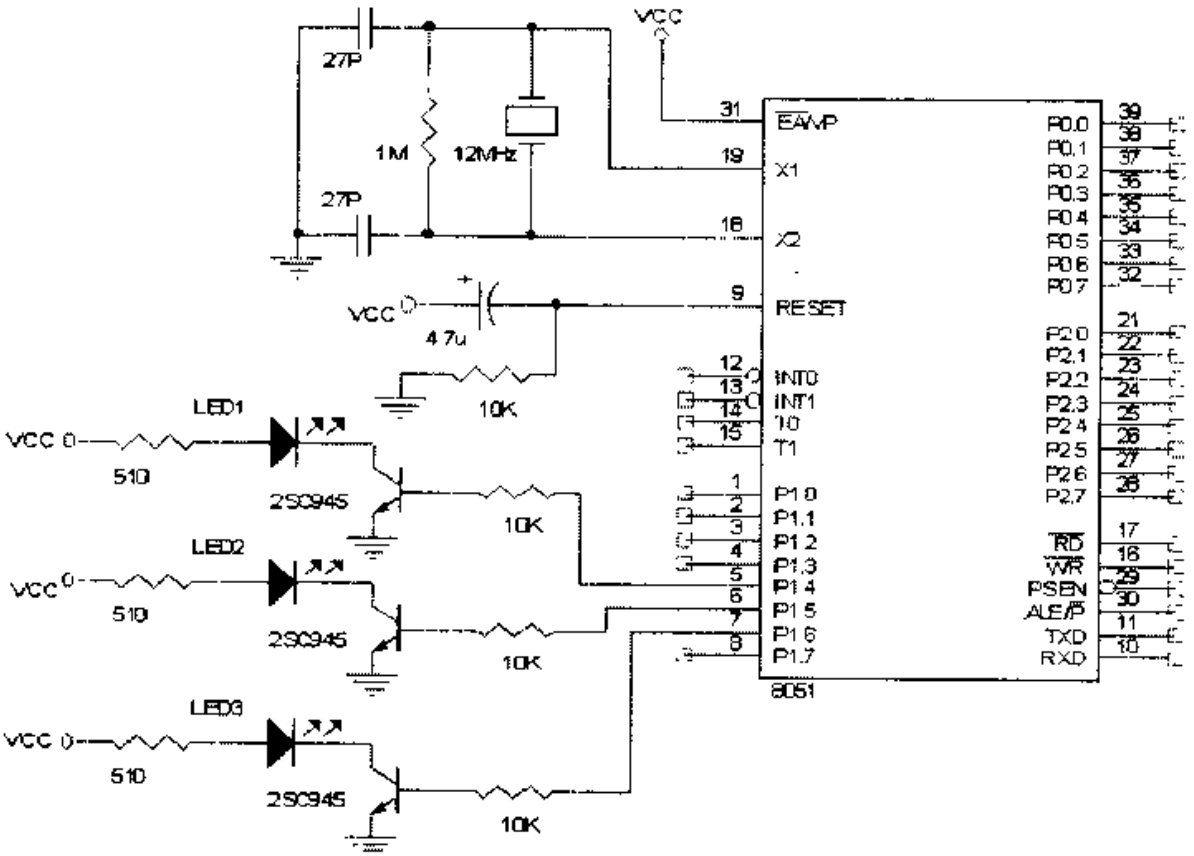
行　号	说　　明
41	每 40mS 此变量就加 1，目前未使用，只用来表示总共经过了几次 40mS，即可以作为计时的时间变量使用
43,44	计时的时间数值每 40mS 之后就减 1，即当此变量 LedOffCount 的数值等于 0，就表示计时的时间终了，也不用再减 1 了。

➢ LedTimming( )

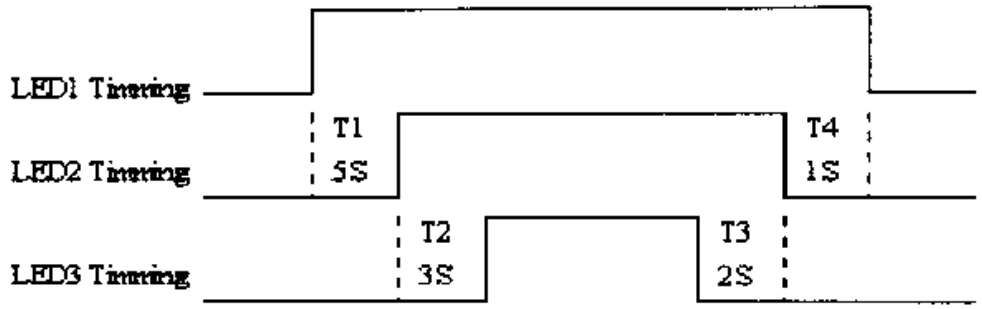
目的

ON / OFF 的循序动作流程

电路



On/Off 流程：



程序

```
01 //timming demo:On/Off Sequence
02 //led1 on ,5s led2 on ,3s led3 on
03 //led3 off,3s led2 off,5s led1 off
04 void LedTimming(void)
05 {
06     IE  = 0;              //disable all interrupt
07     IP  = 0x0b;           //hi priority:int0,count0,timer1
08         TMOD=  0x15;                  //set  timer1:mode1  16  bit
timer,timer0:counter0
09
10     TL1 = CLOCK_40MS & 0xff;      //timer1:40mS
11     TH1 = CLOCK_40MS >> 8;
12
13     ET1 = 1;              //enable timer1 interrupt
14     EA  = 1;              //enable all interrupt
15
16     Led1On( );
17     FgLed2On  = 0;
18     FgChange  = 0;
19     LedOffCount=TIME_5SEC;
20     TR1 = 1;
21
22     while( 1 )
23     {
24        if ( FgChange==0 )
25        {
```

```
26          if ( FgLed2On==0 )
27          {
28            if ( LedOffCount==0 )
29            {
30              Led2On( );
31              FgLed2On = 1;
32              FgLed3On = 0;
33              LedOffCount=TIME_3SEC;
34            }
35          }
36          else if ( FgLed3On==0 )
37          {
38            if ( LedOffCount==0 )
39            {
40              Led3On( );
41              FgLed3On = 1;
42              FgLed2Off= 0;
43              DelayX10ms(500);          //test:led3 on 5s
44              FgChange=1;
45
46              Led3Off( );
47              FgLed2Off= 0;
48              FgLed1Off= 1;
49              LedOffCount=TIME_2SEC;
50            }
51          }
52        }
53        else if ( FgChange==1 )
54        {
55          if ( FgLed2Off==0 )
56          {
57            if ( LedOffCount==0 )
58            {
59              Led2Off( );
60              FgLed2Off = 1;
```

```
61                  FgLed1Off = 0;
62                  LedOffCount=TIME_1SEC;
63               }
64            }
65            else if ( FgLed1Off==0 )
66            {
67               if ( LedOffCount==0 )
68               {
69                  Led1Off( );
70                  FgLed1Off = 1;
71                  FgChange  = 0;
72               }
73            }
74         }
75      }
76 }
77
78 void Led1On(void)
79 {
80     P1_4=1;
81 }
82 void Led1Off(void)
83 {
84     P1_4=0;
85 }
86
87 void Led2On(void)
88 {
89     P1_5=1;
90 }
91 void Led2Off(void)
92 {
93     P1_5=0;
94 }
95
```

```
96 void Led3On(void)
97 {
98     P1_6=1;
99 }
100 void Led3Off(void)
101 {
102     P1_6=0;
103 }
```

说明

在很多实际应用中，时常会依据系统特性而必须作 ON/OFF 的顺序流程，以稳定其系统品质与功能，例如：电源开机后一些硬件外围装置的 ON/OFF 顺序动作，或 LCD(Liquid Crystal Display：液晶显示)监视器面板(PANEL)的电源供给流程，或 PC 计算机监视器变换频率模式的信号控制流程等。

行 号	说　　明
06~14	如同上一个程序 LedMain2( )的说明
16~19	设定 LED1 亮，并清除 FgLed2On 及转变标志 FgChange 及设定 T1=5 秒
20	开始计时
22	不断的执行循环
24	当 FgChange=0 则表示在 ON 的计时阶段，当 FgChange=1 则表示在 OFF 计时阶段
28~34	如果 T1=5 秒的计时终了，则设定 LED2 亮并设定 LED2 标志变量 FgLed2On=1，下一个标志 FgLed3On = 0，并设定 T2=3 秒的计时时间，那么，第 26 行的条件就不成立，也就不会再重复执行括号内的语句，因为计时已经终了当然不需要再做，而使用此技巧可以达到这个功效
36	要开始计时 T2 时，FgLed3On 标志变量一定会等于 0，因为 T1 计时终了会将此标志变量清除为 0
38	判别 T2 时间是否终了
40~42	如果 T2 时间终了，则设定 LED3 亮，并设定 FgLed3On = 1 及标志变量 FgLed2Off= 0
43,44	LED3 持续亮 5 秒钟，并设定由 ON 计时变成 OFF 计时的标志变量 FgChange=1

行 号	说 明
46~49	设定 LED3 熄灭及标志变量 FgLed1Off= 1，并设定 T3=2 秒的计时时间
53	作 OFF 计时的流程
55~64	判断 T3=2 秒的计时时间是否终了，如果是，则设定 LED2 熄灭，并设定 T4=1 秒的计时时间
65	T4 计时的判别
67~72	判断 T4=1 秒的时间是否终了，如果时间终了，则设定 LED1 熄灭，完成整个 ON/OFF 的时序
78~81	LED1 亮
82~85	LED1 熄
87~90	LED2 亮
91~94	LED2 熄
96~99	LED3 亮
100~103	LED3 熄

➢ LedMain3( )

目的

可随时检测或执行其它程序，而让 LED 不断的闪烁一下。

程序

```
01 void  LedMain3(void)
02 {
03    LedCount=0;
04    while( 1 )
05    {
06       Subroute1( );
07
08       LedDelayFlash( );          //delay method
09       DelayXlms(40);          //delay 40ms
```

```
10        LedCount++;
11
12        Subroute2( );
13     }
14 }
15
16 void  LedDelayFlash(void)
17 {
18     if (LedCount<26)          //40ms * 25=1S
19     {
20        LedOn( );
21        return;           //exit
22     }
23     else if (LedCount<50)
24
25        LedOff( );
26        return;           //exit
27     }
28     else
29        LedCount=0;           //clr led count
30 }
```

说明

此程序比直接使用延迟方式让 LED 闪烁还要实用。要让 LED 不断的 1 秒钟闪烁一次时(例如某一检测信号发生时)，请千万不要用延迟 1 秒的方式。如果只是闪烁一次且闪烁时间并不是很长，为了程序简单容易明了，可以使用延迟的方式，但如果需要不断的闪烁，则必须写在主程序内，以延迟小单位时间的间隔方式去计数和判断。

行 号	说　　明
01	执行 LED 不断闪烁的主程序
03	计数值先初始化清除为 0
04	重复执行循环
06	其它子程序

行 号	说 明
08	借着计数值而设定 LED 亮或熄的子程序
09,10	每延迟 40mS 则计数值加 1
12	其它子程序
16	令 LED 不断闪烁的子程序
18~22	40mS * 25=1 秒，假如计数值在 25 之内，则设定 LED 亮并返回原调用程序
23	当 LED 亮 1 秒了，则会执行此判断式
25,26	设定 LED 熄并返回原调用程序
28,29	当 LED 亮 1 秒也熄灭 1 秒，即 LED 闪烁一次了，因此要重新计数

➢ LedMain4( )

目的

以计时中断的方式来让 LED 闪烁。

程序

```
01 void  LedMain4(void)
02 {
03     IE  = 0;          //disable all interrupt
04     IP  = 0x0b;       //hi priority:int0,count0,timer1
05     TMOD= 0x15; //set timer1:model 16 bit timer,timer0:counter0
06
07     TL1 = CLOCK_40MS & 0xff;    //timer1:40mS
08     TH1 = CLOCK_40MS >> 8;
09
10     ET1 = 1;          //enable timer1 interrupt
11     EA  = 1;          //enable all interrupt
12
13     TR1 = 1;          //start timer1
14
15     LedCount=0;
```

```
16      while( 1 )
17      {
18          Subroute1( );
19
20          LedTimerFlash( );     //timer method
21
22          Subroute2( );
23      }
24  }
25
26  void  LedTimerFlash(void)
27  {
28      if (LedCount<26)    //40ms * 25=1S
29      {
30          LedOn( );
31          return;       //exit
32      }
33      else if (LedCount<50)
34
35          LedOff( );
36          return;          //exit
37      }
38      else
39          LedCount=0;          //clr led count
40  }
41
42  void  Led_Timer1ISR (void) interrupt 3 using 2
43  {
44      TL1 = CLOCK_40MS & 0xff;  //timer=40ms,interrupt 1 time
45      TH1 = CLOCK_40MS >> 8;
46      TF1 = 0;
47
48      LedCount++;
49  }
```

说明

上一个范例，每延迟 40mS 则计数值加 1，并用子程序来判断 LED 亮或 LED 熄灭的时间是否终了。本范例程序用的是定时器方式，即先设定每 40mS 中断一次，且计时时间常数先设定好，在中断例程中将计数值减 1，直到计数值等于 0，也就是计时时间终了。用如此的方式可以达到 LED 闪烁的目的，其它功能也都建议用定时器的程序架构来实现。

行 号	说　　明
03~13	设定一些控制缓存器并开始计时
15	计数值先初始化清除为 0
20	调用子程序，其中 LedCount 在计时中断例程递增，即每 40mS 之后则计时变量值就加 1
26~40	LED 闪烁子程序，LED 亮 1 秒之后变成 LED 熄 1 秒，并将计数值清除为 0
42	计时中断例程
44,45	重新加载计时缓存器的内容，以便下一次又是 40mS 之后，就计时中断一次
48	每 40mS 之后，计时变量值就加 1，以便在 LedTimerFlash( )函数作判断

➢ LedMain5( )

目的

以 LED 闪烁方式的不同，来判别外部的触发信号。

程序

```
01 void  LedMain5(void)        //timer method
02 {
03    while( 1 )
04    {
05       if ( P2_0==0 )
06          Led_1Flash( );
07       else if ( P2_1==0 )
08          Led_2Flash( );
```

```
09        else if ( P2_2==0 )
10           Led_3Flash( );
11     }
12 }
13
14 //flash 1 time for 2 seconds
15 void  Led_1Flash(void)
16 {
17     if (LedCount<6)          //0~5,led on 0.2s
18        LedOn( );
19     else if (LedCount<58)        //6~58,led off 2s
20        LedOff( );
21     else
22        LedCount=0;           //re-start
23 }
24
25 //flash 2 time for 2 seconds
26 void  Led_2Flash(void)
27 {
28     if (LedCount<6)          //0~5,led on 0.2s
29        LedOn( );
30     else if (LedCount<11)        //6~10,led off 0.2s
31        LedOff( );
32     else if (LedCount<16)        //11~15,led on 0.2s
33        LedOn();
34     else if (LedCount<68)        //16~68,led off 2s
35        LedOff( );
36     else
37        LedCount=0;
38 }
39
40 //flash 3 time for 2 seconds
41 void  Led_3Flash(void)
42 {
43     if (LedCount<6)          //0~5,led on 0.2s
```

```
44        LedOn( );
45     else if (LedCount<11)        //6~10,led off 0.2s
46        LedOff( );
47     else if (LedCount<16)        //11~15,led on 0.2s
48        LedOn( );
49     else if (LedCount<21)        //16~20,led off 0.2s
50        LedOff( );
51     else if (LedCount<26)        //21~25,led on 0.2s
52        LedOn( );
53     else if (LedCount<78)        //26~78,led off 2s
54        LedOff( );
55     else
56        LedCount=0;
57 }
```

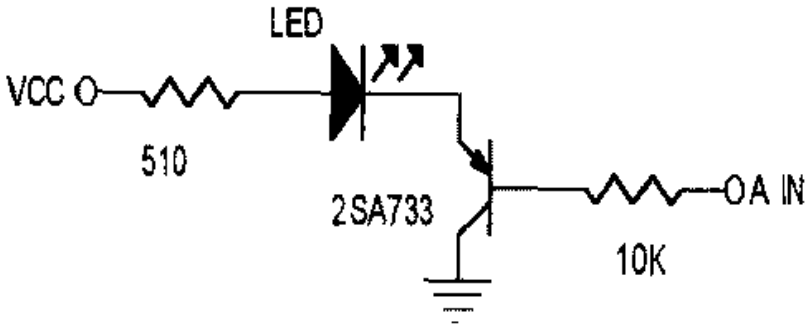
说明

如何区分外在的触发信号呢？如果有很多的输入信号，代表着在系统中不同的状态或异常时，最简单且最节省成本的方式，就是使用 LED 分别以不同的闪烁方式来代表不同的触发信号，如以下说明：

行 号	说　明
01	主程序
05,06	当外部信号动作，即 P2.0 输入“0”电平时，执行 Led_1Flash( )函数。
07,08	当外部信号动作，即 P2.1 输入“0”电平时，执行 Led_2Flash( )函数。
09,10	当外部信号动作，即 P2.2 输入“0”电平时，执行 Led_3Flash( )函数。
15	在 2 秒后，LED 快速闪烁一下的例程
17,18	LED 亮 0.2 秒
19,20	LED 熄 2 秒
21,22	将计数器 LedCount 重新清除为 0
26	在 2 秒后，LED 快速闪烁两下的例程
28,29	LED 亮 0.2 秒
30,31	LED 熄 0.2 秒
32,33	LED 又亮 0.2 秒

行　号	说　　明
34,35	LED 熄 2 秒，达到 LED 熄 2 秒后快速闪烁二下，如此一直循环的功能
37	将计数器重新清除为 0
41	在 2 秒后，LED 快速闪烁二下的例程
43,44	LED 亮 0.2 秒
45,46	LED 熄 0.2 秒
47,48	LED 亮 0.2 秒
49,50	LED 熄 0.2 秒
51,52	LED 亮 0.2 秒
53,54	LED 熄 2 秒，达到 LED 熄 2 秒后快速闪烁三下的功能
56	将计数器重新清除为 0

电路



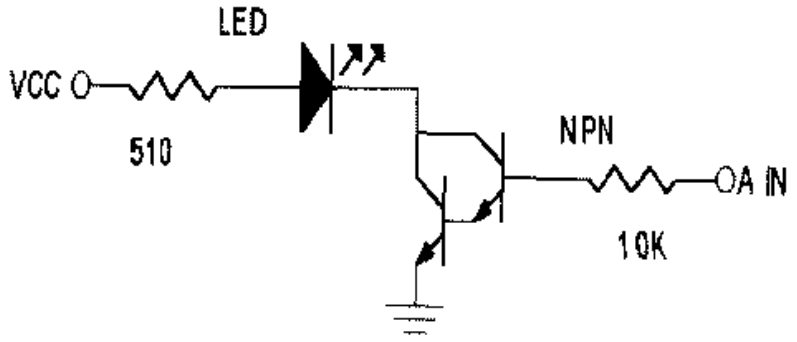
说明

以 PNP 型晶体管的驱动放大电路，来让 LED 亮或熄的电路。

当 A 端为“1”电平，则晶体管 OFF，LED 熄。

当 A 端为“0”电平，则晶体管 ON，LED 亮。

电路



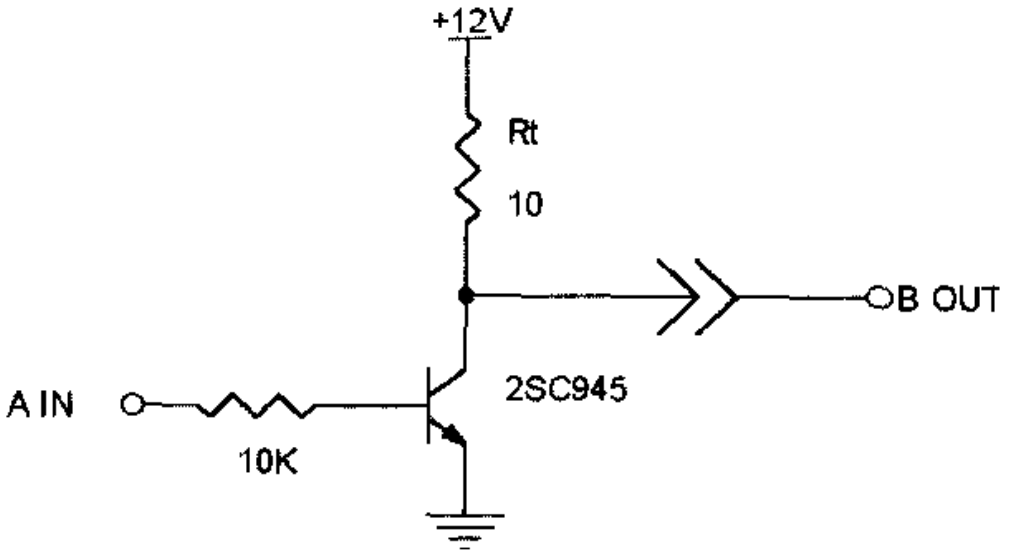
说明

此为 NPN 晶体管达灵顿对的放大电路，其流入晶体管的电流较大，因此可以带动需要较大电流的负载(例如：电感性负载之继电器)。

当 A 端为“1”电平，则晶体管 ON，LED 亮。

当 A 端为“0”电平，则晶体管 OFF，LED 熄。

电路



说明

用 TTL 准位来控制外部+12V 电压的供给与否。

当 A 端为“1”电平，则晶体管 ON，B 端为 0V。

当 A 端为“0”电平，则晶体管 OFF，B 端为+12V。

请注意：

此电路设计看似没有问题，其实是非常不适当的，其缺点如下：

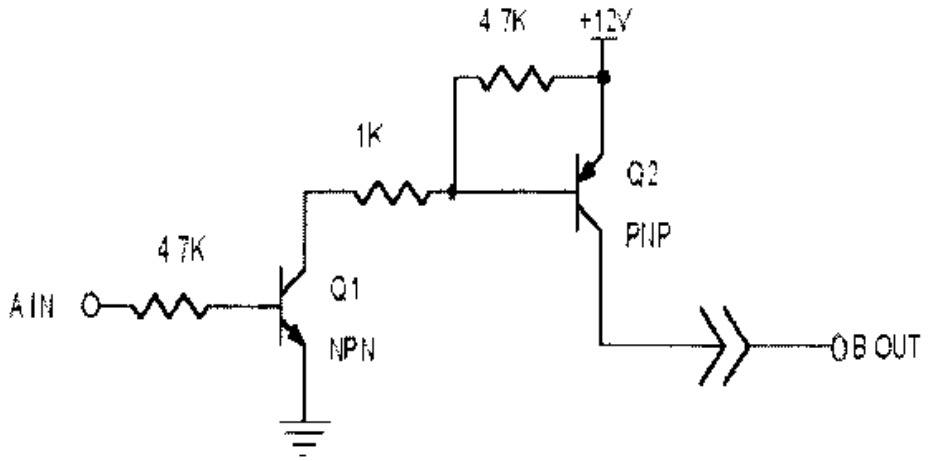
1. 如果电阻 Rt 较大，则 B 端的输出电平会降低；如果电阻 Rt 较小，则集极电流会太大，使得电阻 Rt 的功率变大，成本会增高。

2. 此电路的转态反应动作太慢，不适合需要快速反应的电路。

3. 具有负载效应，易造成输出端的不稳定和不确定性。

由以上得知，请切记不要使用此种电路设计，应改成如下：

电路



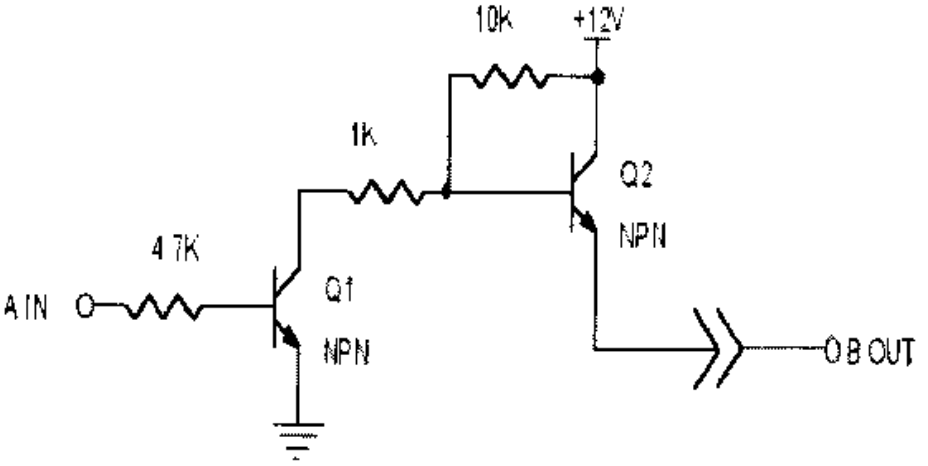
说明

用 PNP 型晶体管来控制 B 输出端电源的 ON/OFF。

当 A 端为“1”电平，Q1、Q2 晶体管 ON，则 B 输出端等于+12V。

当 A 端为“0”电平，Q1、Q2 晶体管 OFF，则 B 输出端等于 0V

电路



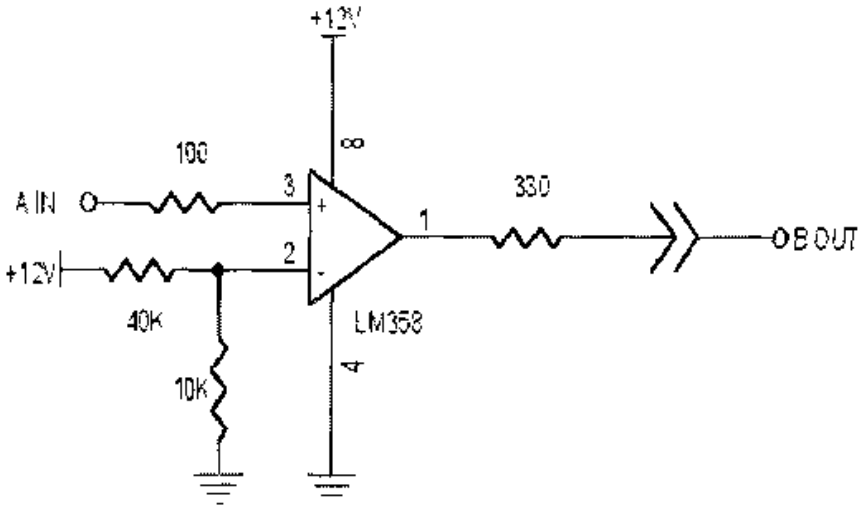
说明

用 NPN 型晶体管来控制 B 输出端电源的 ON/OFF。

当 A 端为“1”电平，Q1 晶体管 ON，Q2 晶体管 OFF，则 B 输出端等于 0V。

当 A 端为“0”电平，Q1 晶体管 OFF，Q2 晶体管 ON，则 B 输出端等于+12V

电路



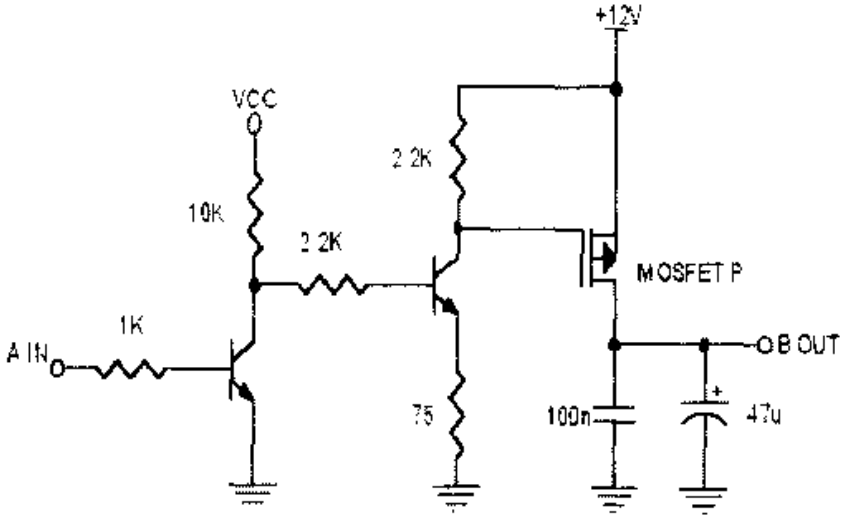
说明

使用运算放大器 IC，因其具有高输出阻抗的特性，可以杜绝负载效应，由于 IC 的第二支脚经由电阻分压具有 2.4V 的输入电压，因此：

当 A 端为“1”电平，则 B 输出端等于+12V。

当 A 端为“0”电平，则 B 输出端没有电压，即等于 0V

电路



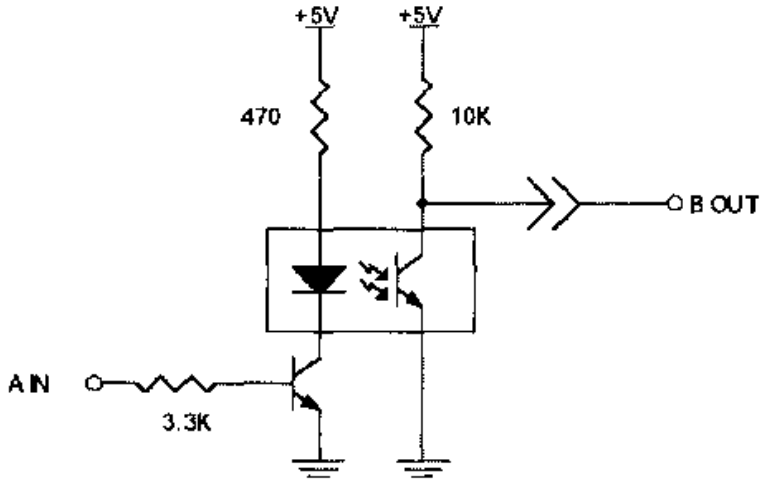
说明

用 MOSFET 来控制电源电压。

当 A 端为“1”电平，MOSFET ON，则 B 输出端等于+12V。

当 A 端为“0”电平，MOSFET OFF，则 B 输出端没有电压。

电路



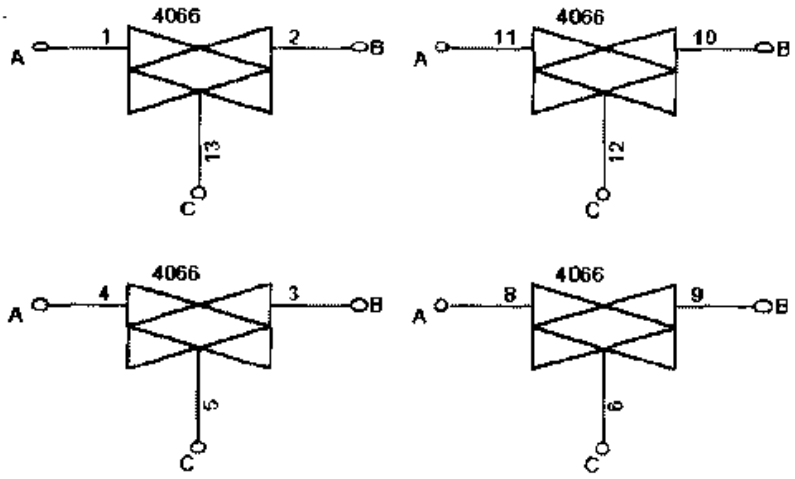
说明

用光耦合器来隔离输入和输出间的控制动作。

当 A 端为“1”电平，光耦合器 ON，则 B 输出端没有电压。

当 A 端为“0”电平，光耦合器 OFF，则 B 输出端等于+5V。

电路



说明

用电子开关来控制信号的导通与否。IC4066 共有 4 组开关，其接脚如上图所示，其中 A 端或 B 端可视为信号源的流入或流出，反之亦可，C 端为控制信号，如下说明：

当 C 端为“1”电平，则 A、B 短路，信号导通。

当 C 端为“0”电平，则 A、B 开路，信号截止。

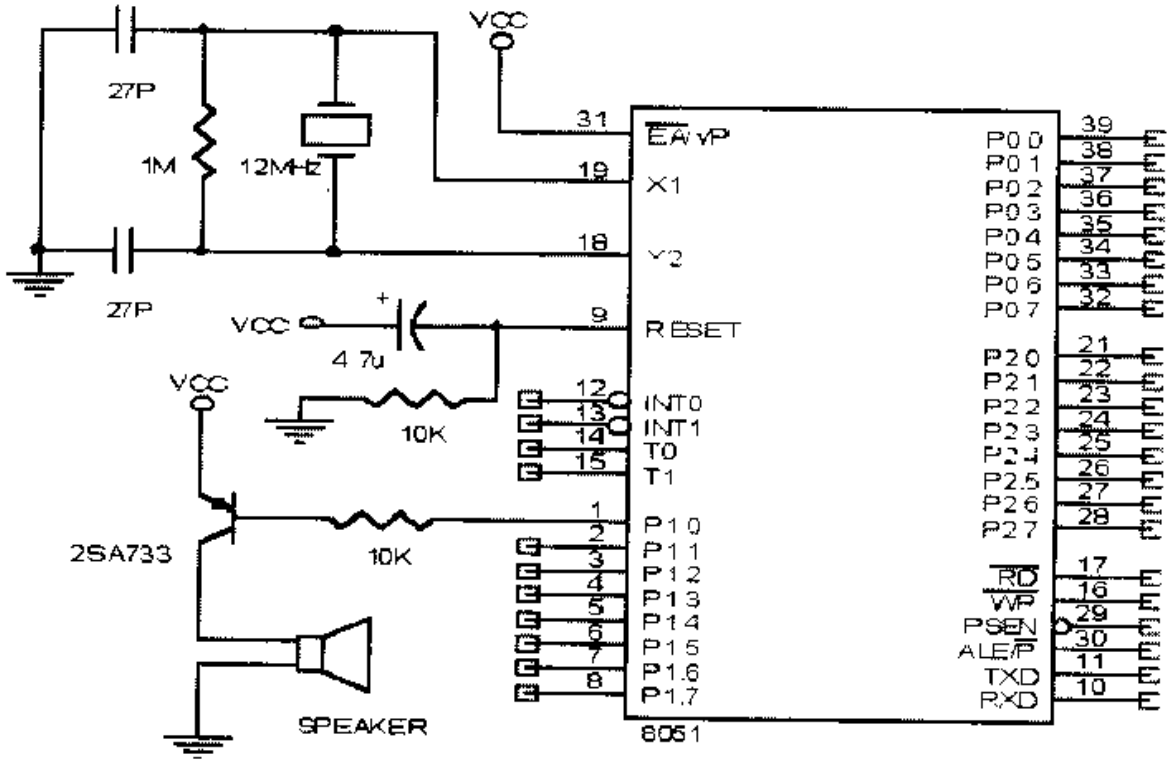
第 9 章　蜂鸣器的应用

➢ Beep1(　)

目的

喇叭或蜂鸣器叫一声的应用。

电路



程序

```
01 void Beep0(void)
02 {
03     P1_0=0;                  //speaker not sound
04 }
05
```

```
06 void Beep1(void)
07 {
08     Word  j,k;
09
10     for(j=0; j<17; j++)
11     {
12        for(k=0; k<50; k++)
13           P1_0 = 0;       //P1.0 on
14        for(k=0; k<50; k++)
15           P1_0 = 1;       //P1.0 off
16     }
17 }
```

说明

微处理器 I/O 脚当作输出时，其电位只有“1”或“0”，使用软件控制其弹性非常大，但也可以使用电子电路来达到相同的目的，这完全要看整体性能的评估、复杂度及成本效益如何。总之，软硬件的相互配合，才能达到系统的最佳化。

要让喇叭产生声音，必须要让晶体管不断的 ON/OFF 交互变化，即 P1.0 不断的输出“1”电位及“0”电位，如果 P1.0 输出一直是“1”电位或输出一直是“0”电位，则喇叭将无法产生声音，如 Beep0( )例程。而 Beep1( )例程，只是单独的产生一个声音音调，因其频率固定变化性比较少，当要产生不同频率的音调，以作为不同功能警报时，则程序又必须再写一次，不但维护不易且占用程序内存空间。

行 号	说　　明
03	P1.0=“1”或 P1.0=“0”，喇叭将无法产生声音
10	晶体管 ON/OFF 的变化共 17 次，即产生 17 周期的方波，这样才能发出声音
12,13	晶体管 ON 一段时间
14,15	晶体管 OFF 一段时间

➢ Beep2(tone)

目的

可发出不同音调的声音。

参数

tone：以 KHz 为单位的频率参数

程序

```
01 void Beep2(Byte tone)      //Khz unit
02 {
03    Word  j,k,SpFreq;
04
05    SpFreq  = (1000/tone)/2;
06
07    for(j=0; j<17; j++)
08    {
09       for(k=0; k<SpFreq; k++)
10          P1_0 = 0;
11       for(k=0; k<SpFreq; k++)
12          P1_0 = 1;
13    }
14 }
```

说明

此程序是上一个程序的扩充，利用参数取代程序中固定的数值，使得程序本身更具弹性，更适合各种不同的应用，无须再重新编写程序。利用参数来达到各种需求的函数，才是实用完整的。

行 号	说　明
05	参数是以 KHz 为单位来传递频率，而算出半周期的时间
07	作 17 次，即音长，发出 BEEP 声的时间
09~12	产生一个方波

➢ Beep3(soundlong,tone)

目的

可发出不同音长及音调的声音。

参数

soundlong：音长，声音的长短

tone：音调，声音的高低，以 KHz 为单位

程序

```
01 void Beep3(Byte soundlong,Byte tone)
02 {
03    Word  j,k,SpFreq;
04
05    SpFreq  = (1000/tone)/2;
06
07    for(j=0; j<soundlong; j++)
08    {
09       for(k=0; k<SpFreq; k++)
10          P1_0 = 0;
11       for(k=0; k<SpFreq; k++)
12          P1_0 = 1;
13    }
14 }
```

说明

此程序的弹性空间更大，不但可以改变声音的长短，发出急促或深长的声音，而且频率可任意变更，即听起来可以比较低沉或比较高亢。

行 号	说　　明
05	参数是以 KHz 为单位来传递频率，而算出半周期的时间
07	产生方波的个数，即发出 BEEP 声的时间
09~12	由 I/O 脚产生一个方波，方波频率由参数 tone 所决定

➢ Beep4(count,soundlong,tone)

目的

可连续发出不同音长及音调的声音

参数

count：发出 BEEP 的个数
soundlong：音长，声音的长短
tone:：音调，声音的高低，以 KHz 为单位

程序

```
01 void Beep4(Word count,Byte soundlong,Byte tone)
02 {
03    Word  i,j,k,SpFreq;
04
05    SpFreq  = (1000/tone)/2;          //Khz unit
06
07    for(i=0; i<count; i++)
08    {
09       for(j=0; j<soundlong; j++)
10       {
11          for(k=0; k<SpFreq; k++)
12             P1_0 = 0;
13          for(k=0; k<SpFreq; k++)
14             P1_0 = 1;
15       }
16       DelayX10ms(12);
17    }
18 }
```

说明

此程序须传递三个参数，利用三层嵌套 for 循环，可解决 BEEP 声的应用范围，算是比较趋于完整且实用的程序，在实务应用中也是足足有余的。

基于以上四种范例的分析，足以了解程序设计也是需要一步一步验证后，思

想空间再慢慢的扩大，逻辑推理的能力逐渐养成，就可循序渐进的设计出成功又实用的程序了。

行　号	说　　明
05	方波的半周期转换表达式
07	发出 BEEP 的个数的循环
09	音长的循环
11~14	产生一个方波
16	每发出一个 BEEP 声就延迟一段时间，即发出 BEEP 声的间隔

➢ BeepGetkey(count,soundlong,tone)

目的

在发出 BEEP 声时，一样可以接受按键。

参数

count：发出 BEEP 的个数
soundlong：音长，声音的长短
tone：音调，声音的高低，以 KHz 为单位

程序

```
01 void BeepGetkey(Word count,Byte soundlong,Byte tone)
02 {
03    Word  i,j,k,SpFreq;
04
05    FgBeepOff =0;
06    SpFreq  = (1000/tone)/2;
07
08    for(i=0; i<count; i++)
09    {
10       for(j=0; j<soundlong; j++)
11       {
12          for(k=0; k<SpFreq; k++)
```

```
13              P1_0 = 0;
14            for(k=0; k<SpFreq; k++)
15              P1_0 = 1;
16        }
17        DelayX10ms(12);
18
19        GetKey2( );
20        if ( (KeyData==KEY1)||(KeyData==KEY10) )
21        {
22           FgBeepOff=1;
23           break; //only KEY1,KEY10 press,exit for loop
24        }
25     }
26 }
```

说明

如果要产生连续 30 次 BEEP 声，以上一个范例 Beep4 的写法，一旦调用 Beep4 例程，则必须发完 30 次 BEEP 声后，程序才会执行完毕，如果在这 30 次的 BEEP 声中想要停止或作异常处理，将无法解决此现象，而此范例即可解决此现象。

行号	说　明
05	初始 BEEP 声的标志
06	计算半周期的时间
08	发出 BEEP 的个数的循环
10~16	连续产生方波的时间，即音长的循环
17	停留一段时间
19	每发出一次 BEEP 声，就检测是否有按键
20~24	如果按了 KEY1 或 KEY10，则设定 BEEP 声结束标志 FgBeepOff，并跳离外围的 for 循环，返回原调用程序，即停止发出 BEEP 声

➢ Alarm1(soundlong,tone)

目的

不断的发出 BEEP 声，如同警报响叫不停(一)

参数

soundlong：音长，声音的长短
tone：音调，声音的高低，以 KHz 为单位

程序

```
01 void Alarm1(Byte soundlong,Byte tone)
02 {
03    Word  i,k,SpFreq;
04
05    SpFreq  = (1000/tone)/2;
06
07    while( 1 )
08
09       for(i=0; i<soundlong; i++)
10       {
11          for(k=0; k<SpFreq; k++)
12             P1_0 = 0;
13          for(k=0; k<SpFreq; k++)
14             P1_0 = 1;
15       }
16       DelayX10ms(12);
17    }
18 }
```

说明

响叫不停的示范程序，应用中千万不能只是这样的编写程序，也要考虑要如何中止，例如当按键的时候。

行 号	说 明
05	音调变量
07	不断的重复执行循环
09	音长
11~14	产生一个方波
16	每发出一次 BEEP 声后，延迟 120mS 再继续发出 BEEP 声，一直重复下去

➢ Alarm2(count,soundlong,tone)

目的

警报的应用(二)

参数

count：发出 BEEP 的个数
soundlong：音长，声音的长短
tone：音调，声音的高低，以 KHz 为单位

程序

```
01 void Alarm2(Word count,Byte soundlong,Byte tone)
02 {
03    Word  i,j,k,SpFreq;
04
05    SpFreq  = (1000/tone)/2;
06
07    while( 1 )
08
09       for(j=0; j<count; j++)
10       {
11          for(i=0; i<soundlong; i++)
12          {
13             for(k=0; k<SpFreq; k++)
14                P1_0 = 0;
```

```
15              for(k=0; k<SpFreq; k++)
16                  P1_0 = 1;
17          }
18          DelayX10ms(12);
19      }
20
21      DelayX10ms(100);      //interval 1s
22  }
23 }
```

说明

如同上一个范例的说明，此程序也不要随意应用在实际工程中，因为程序在开始执行后停不住，即一直发出 BEEP 声。上一个程序和此程序都是为了循序渐进的方式来说明警报例程的注意事项，此程序多加了第 09 行的循环及第 21 行的语句，在于连续发出 count 个数的 BEEP 声后，就间隔 1 秒，才又重复的发出 count 个数的 BEEP 声。

➢ AlarmGetkey(count,soundlong,tone)

目的

可解除警报的方法（完整且实用的应用）

参数

count：发出 BEEP 的个数
soundlong：音长，声音的长短
tone：音调，声音的高低，以 KHz 为单位

程序

```
01 void AlarmGetkey(Word count,Byte soundlong,Byte tone)
02 {
03    Word  i,j,k,SpFreq;
04
05    FgBeepOff =0;
06    FgAlarmOff=0;
07
```

```
    SpFreq  = (1000/tone)/2;

    while( 1 )

      for(j=0; j<count; j++)
      {
        for(i=0; i<soundlong; i++)
        {
          for(k=0; k<SpFreq; k++)
            P1_0 = 0;
          for(k=0; k<SpFreq; k++)
            P1_0 = 1;
        }
        DelayX10ms(12);

        GetKey2( );
        if ( (KeyData==KEY1)||(KeyData==KEY10) )
        {
          FgBeepOff=1;
          break;       //exit for loop
        }
      }

      if (FgBeepOff)

        FgAlarmOff=1;
        break;     //exit while loop,end subroutine
      }
      DelayX10ms(100); //interval 1s
    }
}
```

说明

系统中如发生轻微的异常现象，通常利用喇叭响一声，LED 闪烁一下作为提示，如果发生较严重的异常现象时，则以发出警报作为警示，当异常处理完毕

或想要停止警报时，则在警报中也要有按键检测才能达到要求。

行　号	说　　明
05,06	初始化标志
08	计算半周期的时间
10	不断的重复执行循环
12	BEEP 声的次数
14	BEEP 声的音长
16~19	BEEP 声的音调，产生一个方波
21	每发出 BEEP 声后就延迟 120mS
23~28	检测按键，如果按下 KEY1 或 KEY10 则跳出行号 12 的 for 循环
31~35	设定警报结束标志 FgAlarmOff，并继续跳出行号 10 的 while 循环
36	如果未按下 KEY1 或 KEY10，则间隔 1 秒钟后继续警报

➢ BeepLed(count,soundlong,tone)

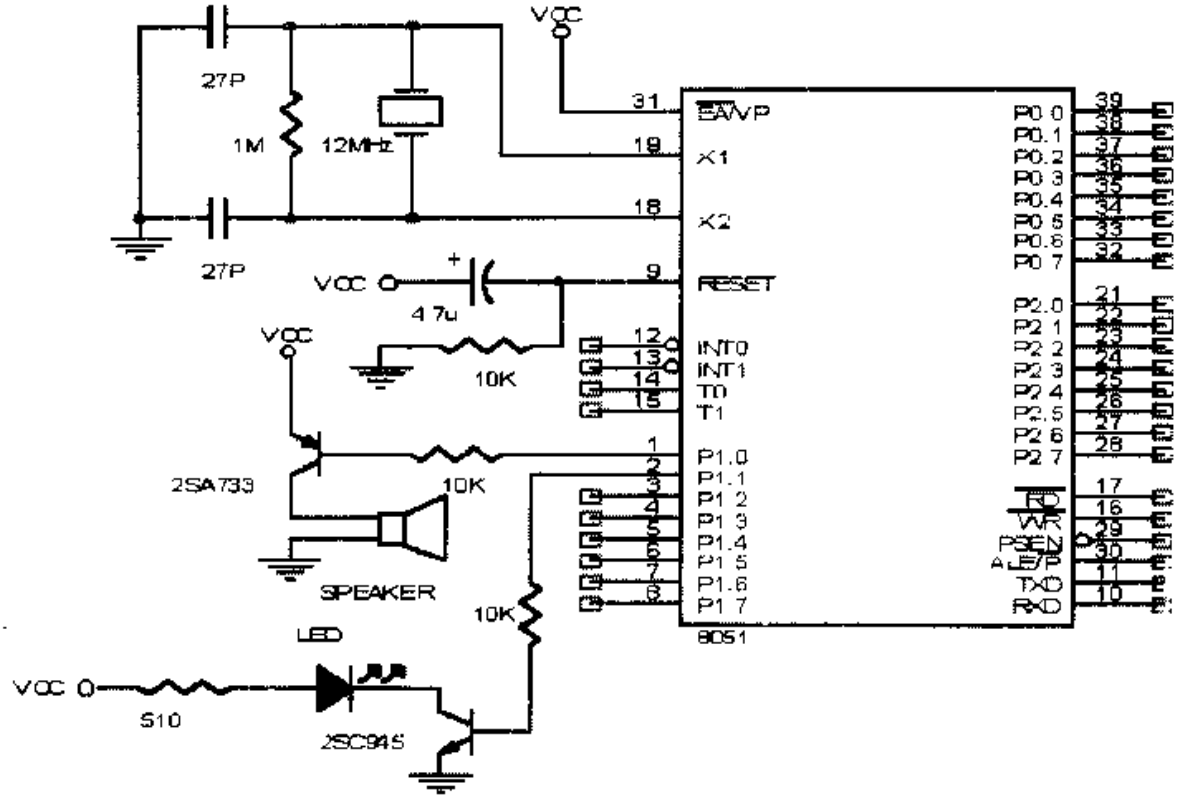
目的

在发出 BEEP 声的同时，LED 也闪烁一下。

参数

count：发出 BEEP 的个数
soundlong：音长，声音的长短
tone：音调，声音的高低，以 KHz 为单位

电路



程序

```
01 void BeepLed(Word count,Byte soundlong,Byte tone)
02 {
03    Word  i,j,k,SpFreq;
04
05    SpFreq  =  (1000/tone)/2;
06
07    for(i=0; i<count; i++)
08    {
09       P1_1=1;            //led on
10       for(j=0; j<soundlong; j++)
11       {
12          for(k=0; k<SpFreq; k++)
13             P1_0 = 0;
14          for(k=0; k<SpFreq; k++)
15             P1_0 = 1;
```

```
16        }
17        P1_1=0;           //led off
18
19        DelayX10ms(12);
20    }
21 }
```

说明

作为提示作用有下列几种方式：

1. 单独的喇叭响叫
2. 单独的 LED 闪烁
3. 喇叭响叫后立即 LED 闪烁
4. 喇叭响叫和 LED 闪烁同时动作

本程序即是喇叭响叫和 LED 闪烁同时动作的范例，要领是在要发出 BEEP 声前先设定 LED 亮，等 BEEP 声结束后才让 LED 熄，即可达到要求。

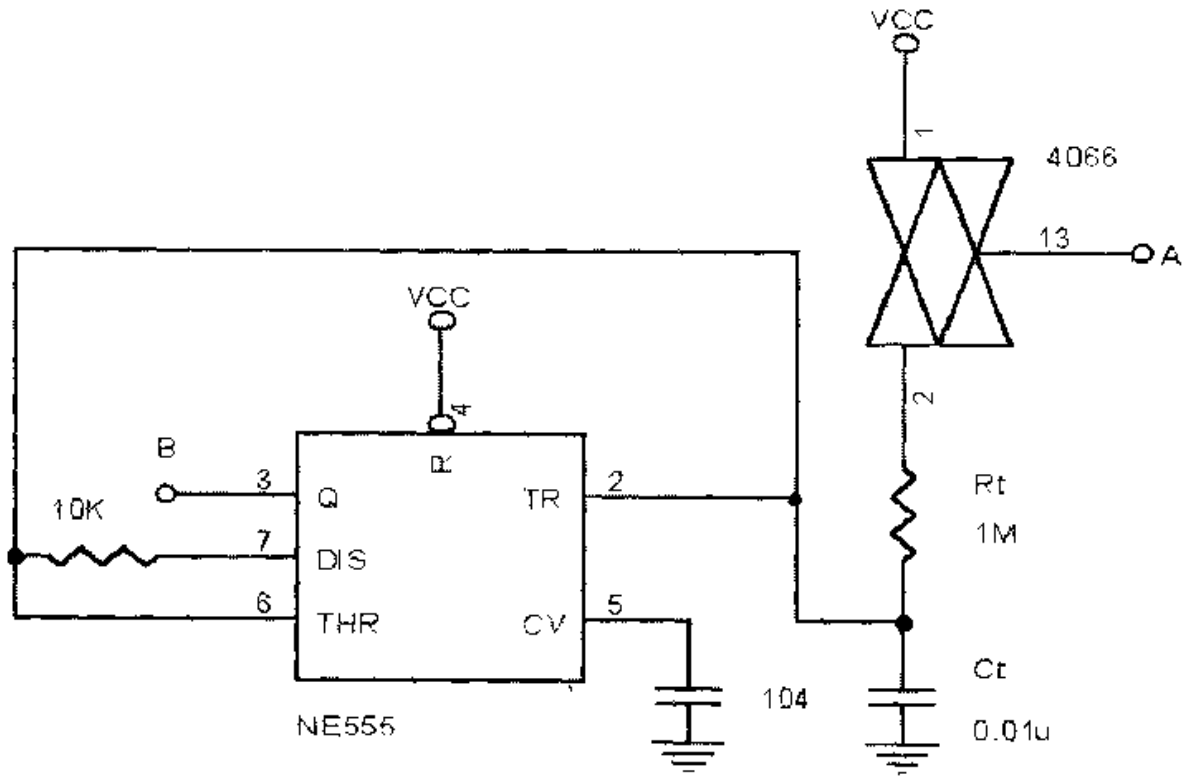
行　号	说　　明
05	音调变量
07	BEEP 声的个数
09	LED 亮
10~16	发出 BEEP 声
17	LED 熄
19	每发出 BEEP 声后就延迟 120mS

➢ HardWareBeep1(　)

目的

以硬件电路来发出 BEEP 声。

电路



程序

```
01 void HardWareBeep1(void)
02 {
03     P1_0=1;          //4066 on,square generate
04 }
```

说明

以硬件电路来发出 BEEP 声，其中 Rt、Ct 控制方波频率，但有缺点如下：

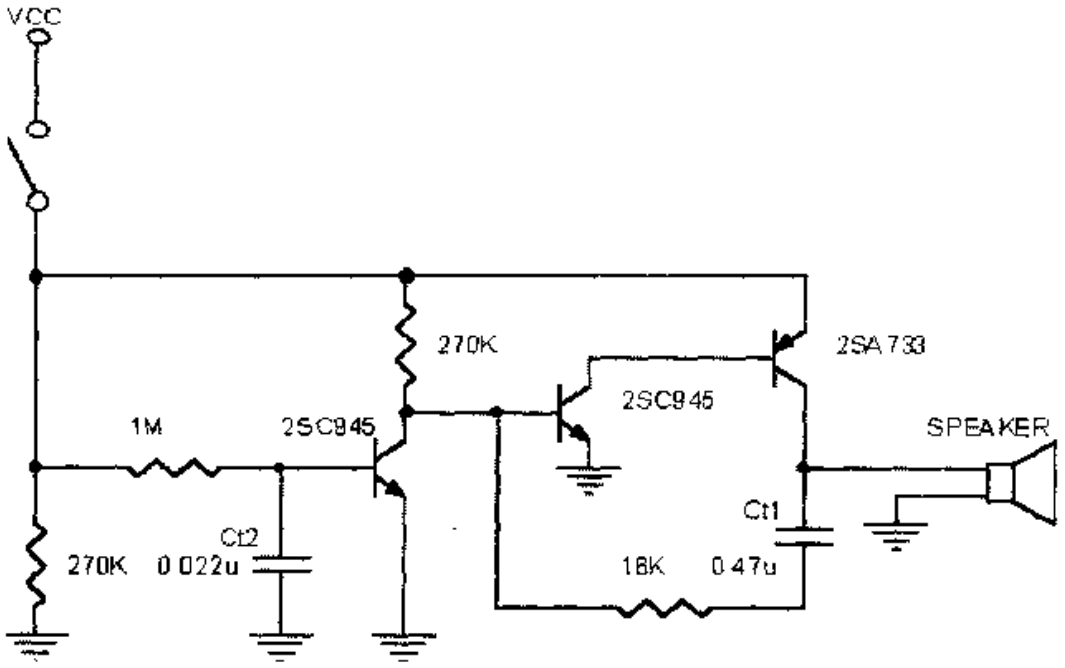
1. 材料成本高

2. 只能产生固定音调

3. 当电路需要更多组不同频率的音调时，也需要更多的电路来配合，则其材料成本更高

其优点为：软件简单，仅控制电源而已。

电路



说明

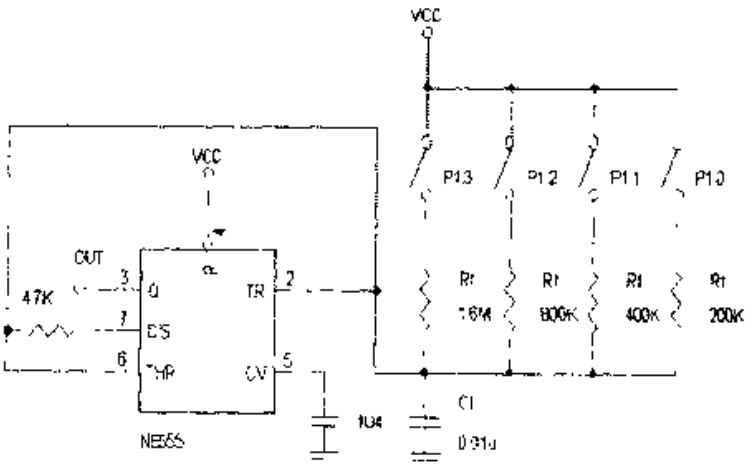
利用晶体管使喇叭发出警报的电路，虽然材料很多，但如果系统简单，仍然可以利用逻辑电路来控制电源的开或关，即控制喇叭发出警报与否，而无须编写软件，则可节省微处理器的成本。

➢ HardWareBeep2(　)

目的

利用四组开关来产生四组不同的方波频率(一)

电路



程序

```
01 void HardWareBeep2(void)
02 {
03    if ( FgExtFreq1==1 )
04    {
05       P1_0=1;          //square frequent 1
06       P1_1=0;
07       P1_2=0;
08       P1_3=0;
09    }
10    else if ( FgExtFreq2==1 )
11    {
12       P1_0=0;          //square frequent 2
13       P1_1=1;          //4066 on
14       P1_2=0;
15       P1_3=0;
16    }
17    else if ( FgExtFreq3==1 )
18    {
19       P1_0=0;          //square frequent3
20       P1_1=0;
21       P1_2=1;
22       P1_3=0;
23    }
24    else if ( FgExtFreq4==1 )
25    {
26       P1_0=0;          //square frequent4
27       P1_1=0;
28       P1_2=0;
29       P1_3=1;
30    }
31 }
```

说明

利用硬件来产生方波，并配合软件 I/O 来控制不同的频率，即控制音调，其中 Rt、Ct 为频率的参数，I/O P1.0~P1.3 是选择哪一段的频率输出。

行　号	说　　明
03~09	当外部频率标志 FgExtFreq1="1"，则设定 P1.0="1"的动作，来产生第一种音调
10~16	当外部频率标志 FgExtFreq2="1"，则设定 P1.1="1"的动作，来产生第二种音调
17~23	当外部频率标志 FgExtFreq3="1"，则设定 P1.2="1"的动作，来产生第三种音调
24~30	当外部频率标志 FgExtFreq4="1"，则设定 P1.3="1"的动作，来产生第四种音调

➢ HardWareBeep3(　)

目的

利用四组开关来产生四组不同的方波频率(二)

程序

```
01 void HardWareBeep3(void)
02 {
03    P1_0=0;
04    P1_1=0;
05    P1_2=0;
06    P1_3=0;
07
08    if ( FgExtFreq1==1 )
09       P1_0=1;          //4066 on,square frequent1
10    else if ( FgExtFreq2==1 )
11       P1_1=1;          //4066 on,square frequent2
12    else if ( FgExtFreq3==1 )
13       P1_2=1;          //4066 on,square frequent3
```

```
14     else if ( FgExtFreq4==1 )
15        P1_3=1;           //4066 on,square frequent4
16 }
```

说明

此程序的目的和上一个范例一样，但程序写法以这种模式较为简单明了，请读者多加利用及思考这种方式。

第 10 章　演奏歌曲的应用

当有了 BEEP 声的概念之后，即可以设计更复杂的音乐了，因为音乐是由连续的音符所串联而来的，只是每个音符的音长、频率不同而已，音乐中 4/4 表示每一小节有 4 拍，以四分音符为一拍，因此，在程序中音长以四分音符定为 12，而演奏速度设为 12 个循环，可依据需要而定。

音名和频率对照表如下：

音名	频率	音名	频率	音名	频率	音名	频率
C2	65	A3	220	F#5	740	D#7	2489
C#2	69	A#3	233	G5	784	E7	2637
D2	73	B3	247	G#5	831	F7	2794
D#2	78	C4	262	A5	880	F#7	2960
E2	82	C#4	277	A#5	932	G7	3136
F2	87	D4	294	B5	988	G#7	3322
F#2	93	D#4	311	C6	1047	A7	3520
G2	98	E4	330	C#6	1109	A#7	3729
G#2	104	F4	349	D6	175	B7	3951
A2	110	F#4	370	D#6	1245	C8	4186
A#2	116	G4	392	E6	1319	C#8	4435
B2	123	G#4	415	F6	1397	D8	4699
C3	131	A4	440	F#6	1480	D#8	4978
C#3	139	A#4	466	G6	1568	E8	5274
D3	147	B4	494	G#6	1661	F8	5587
D#3	156	中央 C5	523	A6	1760	F#8	5919
E3	165	C#5	554	A#6	1865	G8	6271
F3	175	D5	587	B6	1976	G#8	6645
F#3	185	D#5	622	C7	2093	A8	7040
G3	196	E5	659	C#7	2217	A#8	7459
G#3	208	F5	698	D7	2349	B8	7902

关于音乐的演奏，其应用程序如下：

➢ Sound( )

目的

以取表方式发出一种声音。

程序

```
01 //sound
02 Byte RDATA SOUND_LONG[  ] =
03
04     4,4,4,4,4,4,4,4,4,4,
05     4,4,4,4,4,
06 };
07 Byte RDATA SOUND_TONE[  ] =
08
09     100,91,83,77,71,67,62,59,55,52,
10     55,62,71,83,100
11 };
12
13 void Sound(void)
14 {
15     Byte  i,j,k;
16     Byte  SoundLong,SoundTone;
17     Byte  m;
18
19     for(i=0; i<15; i++)              //sound count
20
21       SoundLong=SOUND_LONG[i];
22       SoundTone=SOUND_TONE[i];
23
24       for(j=0; j<SoundLong; j++)     //sound long
25
26         for(k=0; k<8; k++)           //sound speed=tempo
27         {
28           for(m=0; m<SoundTone/3; m++)
```

```
29                P1_0 = 0;          //speaker on
30              for(m=0; m<SoundTone/3; m++)
31                P1_0 = 1;          //speaker off
32            }
33        }
34        Delay50uS(6);          //cut sound 300us
35    }
36 }
```

行 号	说　　明
02~06	每一个音符的音长，即发出声音的长短
07~11	每一个音符的周期，即音调的高低
13	发音子程序
19	音符的循环次数
21,22	以取表方式，存入音长和音调
24	音长的循环
26	发出声音的速度感
28~31	喇叭 ON/OFF，即发出声音
34	稍为延迟一下作为断音

➢ Music1( )

目的

演奏歌曲 —— for 循环

程序

```
01 //三 轮 车
02 Byte RDATA MUSIC_SOUNDLONG[  ] =
03
04     6,6,9,3,6,6,12,
05     6,6,6,6,6,6,12,
06     6,6,9,3,6,6, 9,3,
07     6,3,3,6,3,3, 6,6,9
```

```
08 };
09 Byte RDATA MUSIC_SOUNDTONE[  ] =
10
11  // 1.  1.  2.  3.  5.  5.  3.  简谱记号
12     120,120,106,95, 80, 80, 95,
13  // 5.  5.  6.  7.  1.. 1.. 5.
14     80, 80, 71, 63, 60, 60, 80,
15  // 1.. 1.. 6.  5.  3.  6.  5.  3.
16     60, 60, 71, 80, 95, 71, 80, 95,
17  // 1.  2.  3.  5.  6.  5.  3.  2.  1.
18     120,106, 95, 80, 71, 80, 95,106,120
19 };
20
21 void Music1(void)
22 {
23     Byte  i,j,k;
24     Byte  SoundLong,SoundTone;
25     Word  m;
26
27     for(i=0; i<31; i++)              //sound count
28
29       SoundLong=MUSIC_SOUNDLONG[i];
30       SoundTone=MUSIC_SOUNDTONE[i];
31
32       for(j=0; j<SoundLong; j++)    //sound long
33
34         for(k=0; k<12; k++)         //sound speed=tempo
35         {
36           for(m=0; m<SoundTone*1; m++)
37             P1_0 = 0;
38           for(m=0; m<SoundTone*1; m++)
39             P1_0 = 1;
40         }
41       }
42       Delay50uS(6);
```

```
43      }
44  }
```

行　号	说　　明
02~08	歌曲三轮车每一音符的长度
09~19	歌曲三轮车每一音符的音调
27	总共有 31 个音符，所以循环作 31 次
29,30	以取表方式将音长和音调分别存入变量内
32	音长的循环
34	演奏歌曲的节奏感，即速度
36~39	产生一个方波
42	稍为延迟一下作为断音

➢ Music2(　)

目的

演奏歌曲 —— do while 循环

程序

```
01 Byte RDATA MUSIC_SOUNDLONG1[  ] =
02
03      6,6,9,3,6,6,12,
04      6,6,6,6,6,6,12,
05      6,6,9,3,6,6, 9,3,
06      6,3,3,6,3,3, 6,6,9,
07      0
08 };
09 Byte  RDATA MUSIC_SOUNDTONE1[  ] =
10
11  // 1.  1.  2.  3.  5.  5.  3.
12      120,120,106,95, 80, 80, 95,
13  // 5.  5.  6.  7.  1.. 1.. 5.
14      80, 80, 71, 63, 60, 60, 80,
```

```
15  // 1.. 1.. 6.  5.  3.  6.  5.  3.
16     60, 60, 71, 80, 95, 71, 80, 95,
17  // 1.  2.  3.  5.  6.  5.  3.  2.  1.
18     120,106,95, 80, 71, 80, 95,106,120,
19  // end
20     0
21  };
22
23  void  Music2(void)
24  {
25     Byte  i=0,j,k;
26     Byte  SoundLong,SoundTone;
27     Word  m;
28
29     do
30
31        SoundLong=MUSIC_SOUNDLONG1[i];
32        SoundTone=MUSIC_SOUNDTONE1[i];
33        i++;
34
35        for(j=0; j<SoundLong; j++)    //sound long
36
37           for(k=0; k<12; k++)        //sound speed=tempo
38           {
39              for(m=0; m<SoundTone*1; m++)
40                 P1_0 = 0;
41              for(m=0; m<SoundTone*1; m++)
42                 P1_0 = 1;
43           }
44        }
45        Delay50uS(6);
46
47     } while ( MUSIC_SOUNDTONE1[i]!=0x00 );
48  }
```

行　号	说　　明
01~08	歌曲三轮车每一音符的长度
09~21	歌曲三轮车每一音符的音调
29	do 循环，先执行，后判断是否结束
31,32	将音长和音调从表中取出
33	每取出一个音符则将变量加 1，以指到下一个音符的数组
35	音长的循环
37	演奏速度
39~42	产生一个方波
45	稍为延迟一下作为断音
47	判断数组是否到了最后一个，其内容为 0，如果还未演奏完毕，则继续取表并发出声音，直到音符的结束符号出现为止。

➢ Music3( )

目的

演奏歌曲 —— 以两个字节来表示音调的数值。

程序

```
01 //2 Byte
02 Byte  RDATA MUSIC_SOUNDTONE2[  ] =
03
04  //1.    1.    2.    3.    5.    5.    3.
05    0,120,0,120,0,106,0,95 ,0,80 ,0,80 ,0,95 ,
06  //5.    5.    6.    7.     1..   1..  5.
07    0,80 ,0,80 ,0,71 ,0,63 ,0,60 ,0,60 ,0,80 ,
08  //1..   1..   6.    5.    3.    6.    5.    3.
09    0,60 ,0,60 ,0,71 ,0,80 ,0,95 ,0,71 ,0,80 ,0,95,
10  //1.    2.    3.    5.    6.    5.    3.    2.    1.
11    0,120,0,106,0,95 ,0,80 ,0,71 ,0,80 ,0,95 ,0,106,0,120
12 };
13
14 void Music3(void)
```

```
15 {
16    Byte  i,j,k,SoundLong;
17    Word  m,SoundTone;
18          .
19    for(i=0; i<31; i++)              //sound count
20
21       SoundLong=MUSIC_SOUNDLONG[i];
22       SoundTone = MUSIC_SOUNDTONE2[i*2] << 8 ;
23       SoundTone += MUSIC_SOUNDTONE2[i*2+1];
24
25       for(j=0; j<SoundLong; j++)    //sound long
26
27          for(k=0; k<12; k++)        //sound speed=tempo
28          {
29             for(m=0; m<SoundTone; m++)
30                P1_0 = 0;
31             for(m=0; m<SoundTone; m++)
32                P1_0 = 1;
33          }
34       }
35       Delay50uS(6);
36    }
37 }
```

行 号	说　明
02~12	以两个字节来表示音调的数值，因此可以演奏很低或很高的音阶，第一个字节为高字节，第二个字节为低字节
19	音长，循环作 31 次
21	以取表方式得到音长的数值
22,23	第一个字节为高字节，所以要左移 8 次后并加上第二个低字节，并存入变量内
25~34	喇叭发出声音
35	间隔 300uS 后继续发出下一个音符，直到全部演奏完毕

➢ Music4(number)

目的

自动选曲演奏

参数

number：要演奏的曲目编号，从 1 开始

程序

```
01 Byte  RDATA MUSIC_SOUNDLONG3[  ] =
02
03     //三 轮 车
04     6,6,9,3,6,6,12,
05     6,6,6,6,6,6,12,
06     6,6,9,3,6,6, 9,3,
07     6,3,3,6,3,3, 6,6,9,
08     0,
09     //红 彩 妹 妹
10     12, 6, 6,12, 6, 6, 6,12, 6,24,
11     12, 6, 6,12, 6, 6, 6,12, 6,24,
12     12, 6, 6, 6, 6, 6, 6,
13     6 ,12, 6,24,12,12,12, 6, 6,
14     6,12, 6,24,
15     0
16 };
17 Byte  RDATA MUSIC_SOUNDTONE3[  ] =
18
19     //三 轮 车
20     239,239,212,189,159,159,189,
21     159,159,142,126,120,120,159,
22     120,120,142,159,189,142,159,189,
23     239,212,189,159,142,159,189,212,239,
24     0,//end
25     //红 彩 妹 妹
```

```
    142,159,189,142,159,189,142,142,159,142,
    142,159,189,142,159,189,212,212,239,212,
    189,189,159,142,120,142,159,
    189,189,159,239,189,189,189,189,189,
    142,142,159,142,
    0//end
};

void  Music4(Byte number)
{
    Byte  k,n;
    Byte  SoundLong,SoundTone;
    Word  i=0,j=0,m;

    for(k=0; k<(number-1); k++)
    {
      while( MUSIC_SOUNDLONG3[i] != 0x00 )

        i++;
      }
      i++;
    }
    for(k=0; k<(number-1); k++)
    {
      while( MUSIC_SOUNDTONE3[j] != 0x00 )

        j++;
      }
      j++;
    }

   do
    {
      SoundLong=MUSIC_SOUNDLONG3[i];
      SoundTone=MUSIC_SOUNDTONE3[j];
```

```
61        i++;
62        j++;
63
64        for(n=0; n<SoundLong; n++)    //sound long
65
66          for(k=0; k<12; k++)         //sound speed=tempo
67          {
68            for(m=0; m<SoundTone/2; m++)
69              P1_0 = 0;
70            for(m=0; m<SoundTone/2; m++)
71              P1_0 = 1;
72          }
73        }
74        Delay50uS(6);
75
76      } while ( (MUSIC_SOUNDLONG3[i]!=0x00) ||
77            (MUSIC_SOUNDTONE3[j]!=0x00) );
78  }
```

行 号	说　　明
01~16	歌曲三轮车和红彩妹妹每一音符的长度
17~32	歌曲三轮车和红彩妹妹每一音符的音调
40	计算要演奏歌曲编号的起始地址
42	如果代表音符的数值不是 0，则指针变量就累加 1，直到一首歌的结尾，即演奏完毕
46	当数组的内容为 0，即表示这首歌已经没有音符了，因此也要跳过此结尾符号
48~55	也要计算出要演奏歌曲音调的起始地址
57	do 循环，直到音符结束符号出现为止
59~62	音长和音调的数值以取表方式得之，每演奏完一个音符就加 1，以指到下一个要演奏的音符
64~74	发出声音并间隔 300uS 的时间
76,77	如果音长和音调的结束符号(内容为 0)出现则停止，否则继续演奏

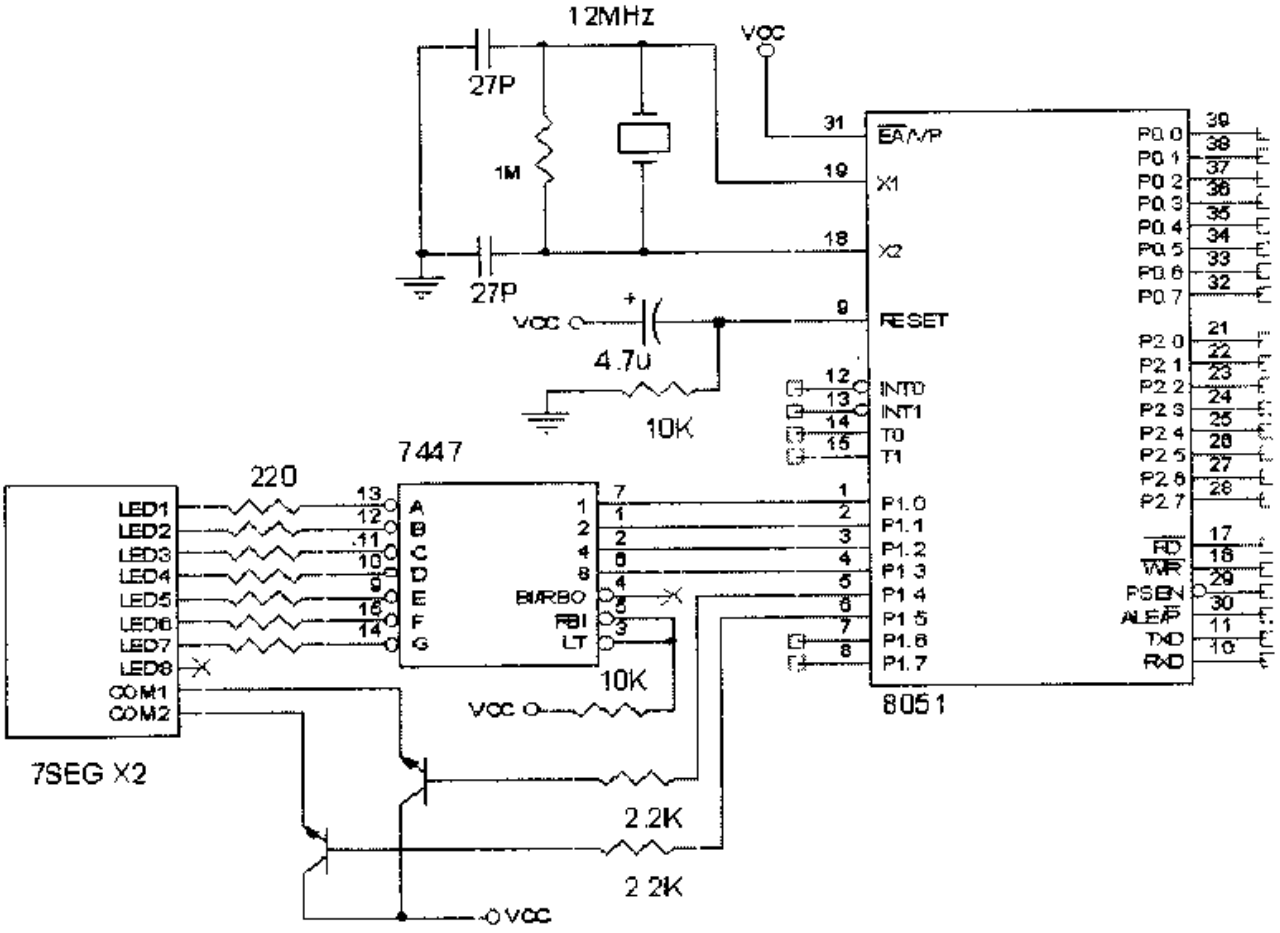
第 11 章　七段显示器的应用

➢ BcdDisplay1(　)

目的

两个 7 段显示器的应用

电路



程序

```
01 //BcdVarible:1 byte,2 digital
02 //vision suspend time
03 void BcdDisplay1(void)
04 {
05    Byte temp;
06
07    P1=0;                          //reset
08    while( 1 )
09    {
10       P1=BcdVarible & 0x0f;     //one unit
11       P1_4=1;           //transistor on
12       P1_5=0;
13       DelayX1ms(3);
14
15       temp=BcdVarible & 0xf0;  //ten unit
16       P1=temp >> 4;     //shift right 4 digital
17       P1_4=0;
18       P1_5=1;           //transistor on
19       DelayX1ms(3);
20    }
21 }
```

说明

在系统中往往有一些状况，希望借着输出设备来作为提示，以明了各种不同的情形，由于人类视觉停留的因素，利用 7 段显示器来显示，并不断的交替轮流点亮，即可达到同时显示的目的。

行 号	说　　明
07	清除数据和控制线
08	不断的重复执行循环，则可以看到数字的显示
10	将个位数取出并作为数据输出至 IC7447 作转换
11,12	个位数，7 段显示器的控制线使能

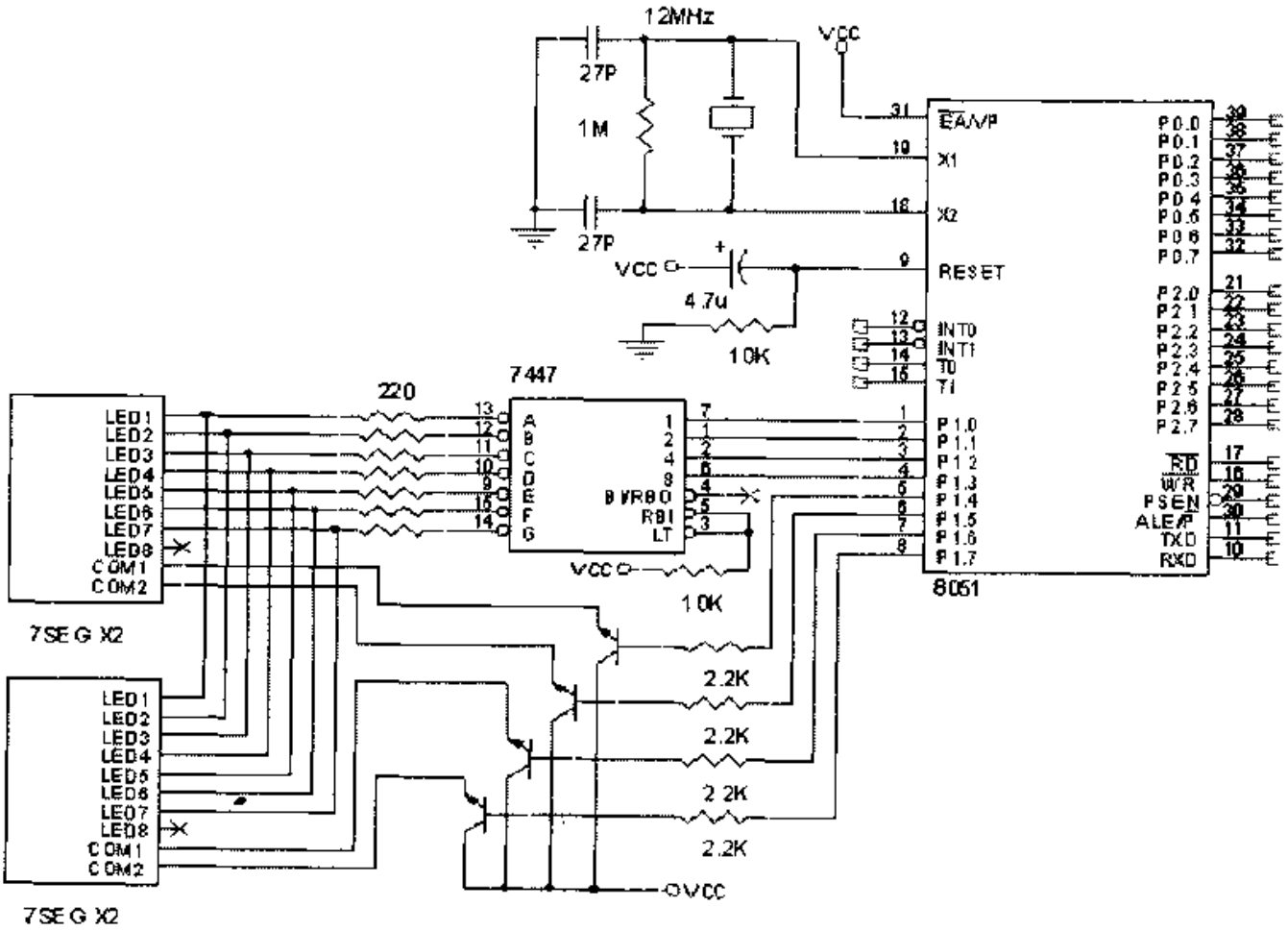
行 号	说 明
13	显示亮 3mS
15	将十位数取出
16	向右移位 4 个位，正好可以由 P1 端口的低 4 个位作输出
17,18	十位数的 7 段显示器的控制线使能
19	显示亮 3mS

➢ BcdDisplay2( )

目的

4 个 7 段显示器的应用。

电路



程序

```
01 //BcdVarible1:2 byte,4 digital
02 void BcdDisplay2(void)
03 {
04    Byte temp1,temp2;
05
06    P1=0;
07    while( 1 )
08    {
09       temp1=BcdVarible1 & 0xff;
10       temp2=BcdVarible1 >> 8;
11
12       P1=temp1 & 0x0f;      //one
13       P1_4=1;
14       P1_5=0;
15       P1_6=0;
16       P1_7=0;
17       DelayX1ms(3);
18
19       P1=(temp1 & 0xf0) >> 4;  //ten,shift right 4 digital
20       P1_4=0;
21       P1_5=1;
22       P1_6=0;
23       P1_7=0;
24       DelayX1ms(3);
25
26       P1=temp2 & 0x0f;      //hundred
27       P1_4=0;
28       P1_5=0;
29       P1_6=1;
30       P1_7=0;
31       DelayX1ms(3);
32
33       P1=(temp2 & 0xf0) >> 4;  //thousand,shift right 4 digital
```

```
34          P1_4=0;
35          P1_5=0;
36          P1_6=0;
37          P1_7=1;
38          DelayX1ms(3);
39      }
40  }
```

说明

如果需要显示的计数值超过 999 则需要以 4 个 7 段显示器来显示，此程序只是上一个范例的延伸，观念及程序技巧也和上一个范例一致，只是需要两个字节(Word)的变量来储存 BCD (Binary Coded Decimal：二进制编码的十进制) 的数码，此变量名称 BcdVarible1 即是 Word 变量。

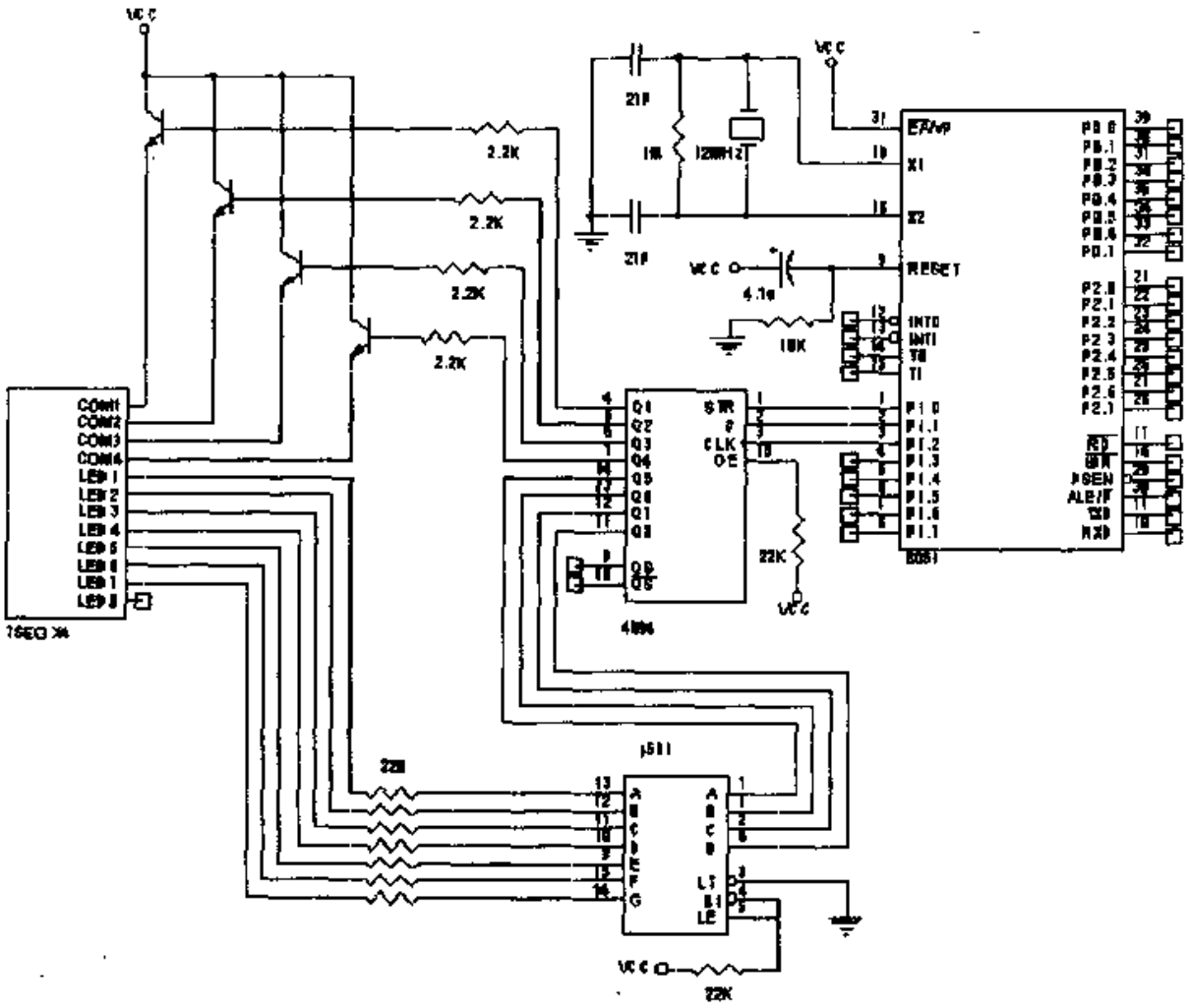
行 号	说　　明
06	P1 断口输出为 0，4 个 7 段显示器皆熄灭
09,10	将存有 4 个 BCD 数码的 Word 变量，分别用 Byte 来做暂存，temp1 为低字节，temp2 为高字节，将 Word 变量向右移 8 个位，即是取其高字节
12~17	将个位数的 BCD 数码显示 3mS
19~24	将十位数的 BCD 数码显示 3mS
26~31	将百位数的 BCD 数码显示 3mS
33~38	将千位数的 BCD 数码显示 3mS
39	while 循环的结束括号，因不断的重复执行循环，所以可以看到 4 个 7 段显示器都会显示数值出来，其实在程序中只是依次显示出个位数、十位数、百位数和千位数，但由于视觉关系看到数字同时显示出来

➢ BcdDisplay3( )

目的

用三个 I/O 脚，即可同时显示 4 个 7 段显示器。

电路



程序

```
01 //BcdData1~BcdData4
02 void BcdDisplay3(void)
03 {
04     Byte temp;
05
06     while( 1 )
07
08       BcdData1 &= 0x0f;
09       temp=(BcdData1 << 4) | 0x01; //data1,scan1="1"
10       Output4094_1(temp);    //demo programs on next sections
11       DelayX1ms(3);
```

```
12
13        BcdData2 &= 0x0f;
14        temp=(BcdData2 << 4) | 0x02; //data2,scan2="1"
15        Output4094_1(temp);
16        DelayX1ms(3);
17
18        BcdData3 &= 0x0f;
19        temp=(BcdData3 << 4) | 0x04; //data3,scan3="1"
20        Output4094_1(temp);
21        DelayX1ms(3);
22
23        BcdData4 &= 0x0f;
24        temp=(BcdData4 << 4) | 0x08; //data4,scan4="1"
25        Output4094_1(temp);
26        DelayX1ms(3);
27    }
28 }
```

说明

由于系统较复杂，致使微处理器 I/O 脚不够使用来控制周边电路时，往往需要以硬件电路来扩充 I/O 端口，只是软件会变得相对复杂，也就是说硬件及软件人员须要互相沟通配合，讨论哪一种方式较简单，成本较节省，及完成日期的达成期望值。

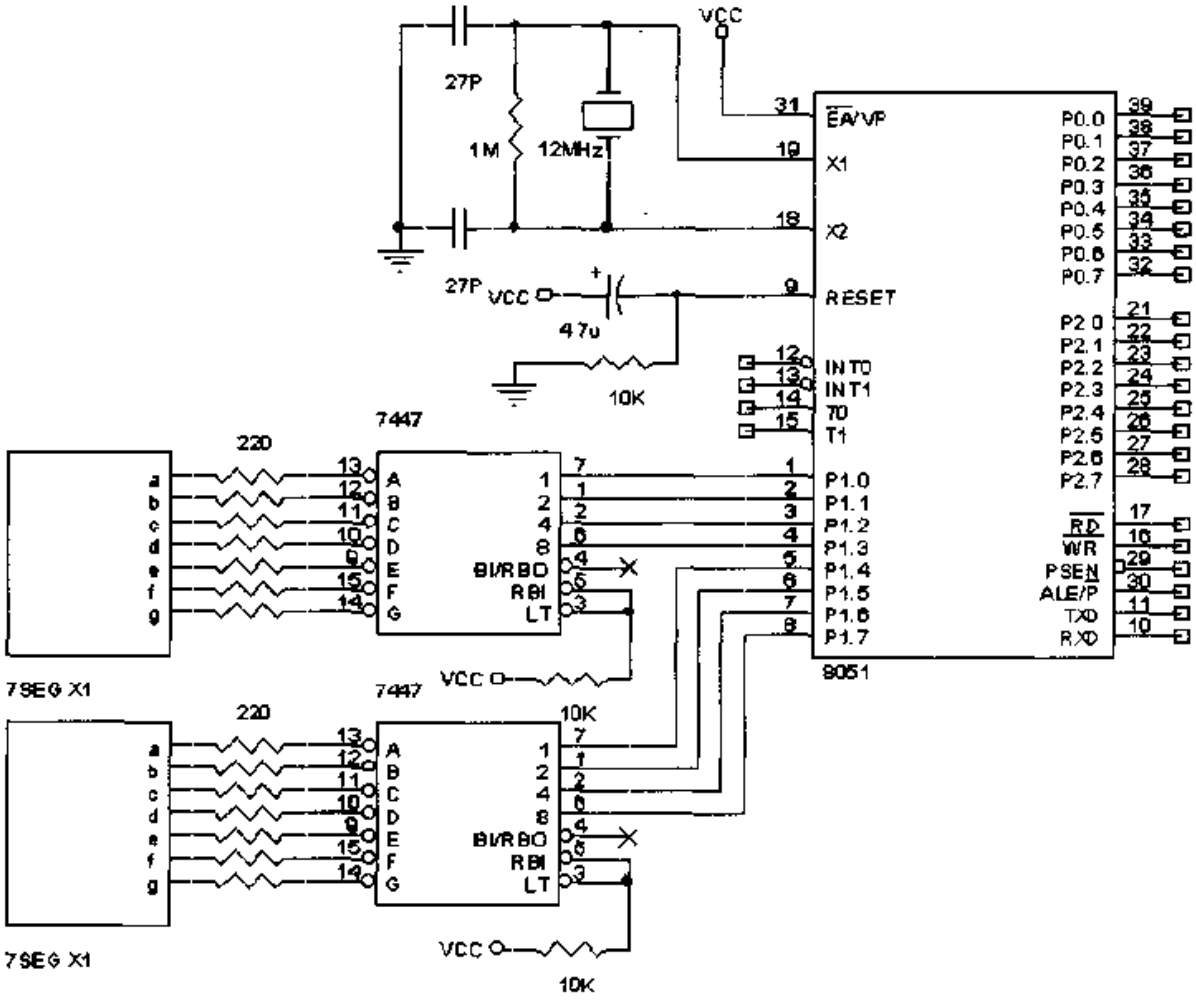
行 号	说　明
06	不断的重复执行循环
08	取个位数的 BCD 数码
09	将 BCD 数码移入 4 个高位内，设定个位数的控制线功能，即输出为“1”电平
10	将数据通过 IC4094 输出至 7 段显示器
11	显示亮 3mS
13~16	将十位数的 BCD 数码显示 3mS
18~21	将百位数的 BCD 数码显示 3mS
23~26	将千位数的 BCD 数码显示 3mS

➢ BcdDisplay4(　)

目的

由 P1 端口高低 4 个位置，来控制 2 个 7 段显示器.

电路



程序

```
01 void BcdDisplay4(void)
02 {
03     P1=BcdVarible;          //P1 output directly
04 }
```

说明

此电路结构较简单，由 IC7447 分别驱动 7 段显示器，而软件将 BCD 数码的变量由 P1 端口直接输出即可，只要 P1 端口的数据内容没有变更，则将会一直显示着目前的数值，如果 P1 端口做变更，则显示器也立即依据 P1 端口所做变更来显示，此方法唯一的缺点是必须占用 8 支 I/O 脚。

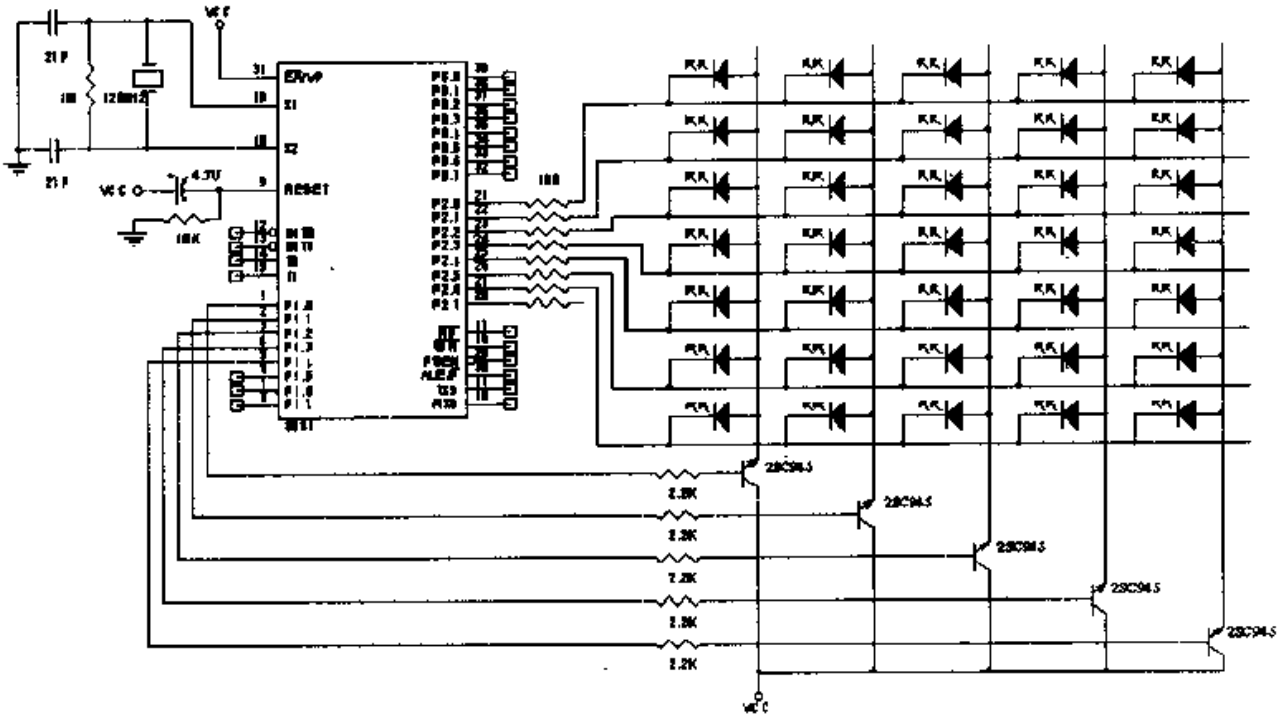
第 12 章　点阵显示器的应用

➢ Dot5x7_Display1(　)

目的

点阵显示器的应用。

电路



程序

```
01 //display "2" data=0x39,0x1e,0x2e,0x36,0x39
02 void Dot5x7_Display1(void)
03 {
04     while( 1 )
05     {
```

```
06      P2=0x39;      //column 1 data
07      P1_0=1;       //control pin="1" active, "1"动作
08      P1_1=0;
09      P1_2=0;
10      P1_3=0;
11      P1_4=0;
12      DelayX1ms(3);
13
14      P2=0x1e;          //column 2 data
15      P1_0=0;
16      P1_1=1;
17      P1_2=0;
18      P1_3=0;
19      P1_4=0;
20      DelayX1ms(3);
21
22      P2=0x2e;          //column 3 data
23      P1_0=0;
24      P1_1=0;
25      P1_2=1;
26      P1_3=0;
27      P1_4=0;
28      DelayX1ms(3);
29
30      P2=0x36;          //column 4 data
31      P1_0=0;
32      P1_1=0;
33      P1_2=0;
34      P1_3=1;
35      P1_4=0;
36      DelayX1ms(3);
37
38      P2=0x39;          //column 5 data
39      P1_0=0;
40      P1_1=0;
```

```
41          P1_2=0;
42          P1_3=0;
43          P1_4=1;
44          DelayX1ms(3);
45      }
46  }
```

说明

点阵显示器的每一点，都是由 LED 所组成，总共有 5 行 7 列，直排称为“行”，横排称为“列”，构成 5x7 点矩阵，要使之显示须先送出第一行的数据，再使第一行的控制线使能，停留一段时间后，再送出第二行的数据，再使第二行的控制线使能，依此类推至第五行控制线使能，并不断的重复执行循环，就可以完全显示出一个字符或数目字，只要将所要显示的字符或数目字规划出来，再依次分解出每一行的数据由 I/O 端口送出，并将控制线使能即可，称之为扫描方式。

行 号	说　　明
02	显示数字 2 的程序
04	条件式永远成立，因此会不断的重复执行循环
06~12	送出第一行的数据，并使第一行的扫描线使能，则相对应的 LED 会点亮，并停留一段时间，此时间常数要依据显示的数字是否会闪烁而适度修正，不能闪烁才可以
14~20	送出第二行的数据，并使第二行的扫描线使能
22~28	送出第三行的数据，并使第三行的扫描线使能
30~36	送出第四行的数据，并使第四行的扫描线使能
38~44	送出第五行的数据，并使第五行的扫描线使能

➢ Dot5x7_Display2(　)

目的

使用取表法来送出数据并显示出来。

程序

```
01 Byte RDATA DISPLAY_TABLE2[  ] = {  0x39,0x1e,0x2e,0x36,0x39 };
02 void Dot5x7_Display2(void)
```

```
03 {
04    Byte  i;
05
06    while( 1 )
07    {
08       for(i=0; i<5; i++)
09       {
10          P2=DISPLAY_TABLE2[i]; //look-up table,display "2"
11          P1=0x01 << i;       //control port=1,2,4,8,16
12          DelayX1ms(3);
13       }
14    }
15 }
```

说明

有一些动作方式如果无法找出相关性的逻辑法则或算法(algorithm)，致使撰写程序变得非常困难，则使用取表法将会变得更简洁方便，也因此时常被软件设计人员所使用。

行 号	说　　明
01	显示数字 2 的数据表
06	条件永远成立，因此会不断的重复执行循环
08	循环作 5 次，因为有 5 行的数据
10	数据表从第一个开始取出来并输出至 P2 端口
11	控制线或称为扫描线使能为“1”动作，将 0x01 左移 i 个位，即分别依次设定 P1.0、P1.1、P1.2 及 P1.3 为“1”电位
12	点亮一段时间

➢ Dot5x7_Display3( )

目的

以各自取表的方式显示出数字 0~9。

程序

```
01 Byte RDATA DISPLAY_TABLE0[  ] = { 0x41,0x3e,0x3e,0x3e,0x41 };//"0"
02 Byte RDATA DISPLAY_TABLE1[  ] = { 0x3f,0x3d,0x00,0x3f,0x3f };//"1"
03 Byte RDATA DISPLAY_TABLE2[  ] = { 0x39,0x1e,0x2e,0x36,0x39 };//"2"
04 Byte RDATA DISPLAY_TABLE3[  ] = { 0x5d,0x36,0x36,0x36,0x49 };//"3"
05 Byte RDATA DISPLAY_TABLE4[  ] = { 0x67,0x6b,0x6d,0x00,0x6f };//"4"
06 Byte RDATA DISPLAY_TABLE5[  ] = { 0x30,0x36,0x36,0x36,0x41 };//"5"
07 Byte RDATA DISPLAY_TABLE6[  ] = { 0x00,0x36,0x36,0x36,0x06 };//"6"
08 Byte RDATA DISPLAY_TABLE7[  ] = { 0x7c,0x7e,0x0e,0x76,0x78 };//"7"
09 Byte RDATA DISPLAY_TABLE8[  ] = { 0x41,0x36,0x36,0x36,0x41 };//"8"
10 Byte RDATA DISPLAY_TABLE9[  ] = { 0x30,0x36,0x36,0x36,0x00 };//"9"
11
12 //one dimension
13 //according DisplayState display 0~9
14 void Dot5x7_Display3(void)
15 {
16    Byte  i;
17
18    while( 1 )
19    {
20       if ( DisplayState==0 )
21       {
22          for(i=0; i<5; i++)
23          {
24             P2=DISPLAY_TABLE0[i];   //"0" table
25             P1=0x01 << i;
26             DelayX1ms(3);
27          }
28       }
29       else if ( DisplayState==1 )
30       {
31          for(i=0; i<5; i++)
32          {
33             P2=DISPLAY_TABLE1[i];   //"1" table
```

```
34            P1=0x01 << i;
35            DelayX1ms(3);
36         }
37      }
38      else if ( DisplayState==2 )
39      {
40         for(i=0; i<5; i++)
41         {
42            P2=DISPLAY_TABLE2[i];   //"2" table
43            P1=0x01 << i;
44            DelayX1ms(3);
45         }
46      }
47      else if ( DisplayState==3 )
48      {
49         for(i=0; i<5; i++)
50         {
51            P2=DISPLAY_TABLE3[i];   //"3" table
52            P1=0x01 << i;
53            DelayX1ms(3);
54         }
55      }
56      else if ( DisplayState==4 )
57      {
58         for(i=0; i<5; i++)
59         {
60            P2=DISPLAY_TABLE4[i];   //"4" table
61            P1=0x01 << i;
62            DelayX1ms(3);
63         }
64      }
65      else if ( DisplayState==5 )
66      {
67         for(i=0; i<5; i++)
68         {
```

```
69            P2=DISPLAY_TABLE5[i];   //"5" table
70            P1=0x01 << i;
71            DelayX1ms(3);
72         }
73      }
74      else if ( DisplayState==6 )
75      {
76         for(i=0; i<5; i++)
77         {
78            P2=DISPLAY_TABLE6[i];   //"6" table
79            P1=0x01 << i;
80            DelayX1ms(3);
81         }
82      }
83      else if ( DisplayState==7 )
84      {
85         for(i=0; i<5; i++)
86         {
87            P2=DISPLAY_TABLE7[i];   //"7" table
88            P1=0x01 << i;
89            DelayX1ms(3);
90         }
91      }
92      else if ( DisplayState==8 )
93      {
94         for(i=0; i<5; i++)
95         {
96            P2=DISPLAY_TABLE8[i];   //"8" table
97            P1=0x01 << i;
98            DelayX1ms(3);
99         }
100     }
101     else if ( DisplayState==9 )
102     {
103        for(i=0; i<5; i++)
104        {
```

```
105             P2=DISPLAY_TABLE9[i];  //"9" table
106             P1=0x01 << i;
107             DelayX1ms(3);
108           }
109       }
110   }
111 }
```

说明

此程序会依据介于 0~9 之间的变量值 DisplayState，在点阵显示器上显示数字，方法是分别将数字 0~9 的数据表先建立好，以便个别数字的取表作业。

行 号	说 明
01~10	数字 0~9 的数据表，为"0"则点矩阵 LED 亮，为“1”则点矩阵 LED 熄灭
18	不断的重复执行循环
20~28	如果状态变量 DisplayState 等于 0，则分别将数字 0 的数据依次输出至 P2 口
29~37	如果状态变量 DisplayState 等于 1，则分别将数字 1 的数据依次输出至 P2 口
38~46	如果状态变量 DisplayState 等于 2，则分别将数字 2 的数据依次输出至 P2 口
47~55	如果状态变量 DisplayState 等于 3，则分别将数字 3 的数据依次输出至 P2 口
56~64	如果状态变量 DisplayState 等于 4，则分别将数字 4 的数据依次输出至 P2 口
65~73	如果状态变量 DisplayState 等于 5，则分别将数字 5 的数据依次输出至 P2 口
74~82	如果状态变量 DisplayState 等于 6，则分别将数字 6 的数据依次输出至 P2 口
83~91	如果状态变量 DisplayState 等于 7，则分别将数字 7 的数据依次输出至 P2 口
92~100	如果状态变量 DisplayState 等于 8，则分别将数字 8 的数据依次输出至 P2 口
101~109	如果状态变量 DisplayState 等于 9，则分别将数字 9 的数据依次输出至 P2 口

➢ Dot5x7_Display4(　)

目的

将数字 0~9 的数据，以取表方式显示。

程序

```
01 //one dimension
02 Byte RDATA DISPLAY_TABLE10[  ] =
03 {
04     0x41,0x3e,0x3e,0x3e,0x41,   //"0"
05     0x3f,0x3d,0x00,0x3f,0x3f,   //"1"
06     0x39,0x1e,0x2e,0x36,0x39,   //"2"
07     0x5d,0x36,0x36,0x36,0x49,   //"3"
08     0x67,0x6b,0x6d,0x00,0x6f,   //"4"
09     0x30,0x36,0x36,0x36,0x41,   //"5"
10     0x00,0x36,0x36,0x36,0x06,   //"6"
11     0x7c,0x7e,0x0e,0x76,0x78,   //"7"
12     0x41,0x36,0x36,0x36,0x41,   //"8"
13     0x30,0x36,0x36,0x36,0x00    //"9"
14 };
15 void Dot5x7_Display4(void)
16 {
17    Byte  i,j;
18
19    if ( DisplayState<10 )
20
21      while( 1 )
22      {
23        for(i=0; i<5; i++)
24        {
25       j=5 * DisplayState;
26           P2=DISPLAY_TABLE10[j+i];
27           P1=0x01 << i;
```

```
28              DelayX1ms(3);
29          }
30      }
31  }
32 }
```

说明

上一个范例由于以各自取表方式，使得程序变得太冗长且不易维护，又占用太多程序内存空间，虽然可以达到功能要求，但仍不宜应用。本范例将数字 0~9 的数据全部放在同一个表内，而以状态变量值 DisplayState 来计算出取表的起始地址，是比较完整实用的程序，程序结构(Architecture)简单明了易维护。

行 号	说　　明
02~14	以一个表来存入数字 0~9 的数据
19	状态变量值 DisplayState 要介于 0~9 之间，否则返回原调用程序
21	不断的重复执行循环 ，
23	每一个数字需要 5 个数据，所以循环作 5 次
25	依据状态变量值 DisplayState 来计算出取表的起始地址
26,27	取表后的数据从 P2 端口输出，并设定扫瞄线使能动作
28	点亮一段时间

➢ Dot5x7_Display5( )

目的

二维取表方式的应用(一)

程序

```
01 //two 'dimension
02 Byte RDATA DISPLAY_TABLE11[ ][5] =
03 {
04    0x41,0x3e,0x3e,0x3e,0x41,        //"0"
05    0x3f,0x3d,0x00,0x3f,0x3f,        //"1"
```

```
06    0x39,0x1e,0x2e,0x36,0x39,        //"2"
07    0x5d,0x36,0x36,0x36,0x49,        //"3"
08    0x67,0x6b,0x6d,0x00,0x6f,        //"4"
09    0x30,0x36,0x36,0x36,0x41,        //"5"
10    0x00,0x36,0x36,0x36,0x06,        //"6"
11    0x7c,0x7e,0x0e,0x76,0x78,        //"7"
12    0x41,0x36,0x36,0x36,0x41,        //"8"
13    0x30,0x36,0x36,0x36,0x00     //"9"
14 };
15 void Dot5x7_Display5(void)
16 {
17    Byte  i;
18
19    if ( DisplayState<10 )
20    {
21       while( 1 )
22       {
23          for(i=0; i<5; i++)
24          {
25             P2=DISPLAY_TABLE11[DisplayState][i];
26             P1=0x01 << i;
27             DelayX1ms(3);
28          }
29       }
30    }
31 }
```

说明

二维取表的方法也时常被广泛应用，其中表内的“列”，例如行号 04~13，为第一维即是名称 DISPLAY_TABLE11 的第一个中括号的下标，列中的数据为“行”，例如第一列中的 0x41、0x3e，为第二维即是名称 DISPLAY_TABLE11 的第二个中括号的下标，请参考范例。

行 号	说 明
02	自行定义的二维矩阵名称，，其中第一个中括号为“列”，第二个中括号为“行”
04~13	显示数字 0~9 的数据，其中行号 04 为第 0 列、行号 05 为第 1 列、行号 06 为第 2 列，依此类推
19	先判断 DisplayState 是否介于 0~9 之间
21	不断的重复执行循环
23	每一列的数据代表一个数字 0~9，而每一列有 5 个数据
25	依据状态变量值 DisplayState 来决定到底是哪一列，即哪一个数字。依据 i 变量来依次取出数据
26,27	扫描线使能并显示一段时间

➢ Dot5x7_Display6( )

目的

二维取表方式的应用(二)。

程序

```
01 //sequent display 0~9
02 void Dot5x7_Display6(void)
03 {
04    Byte  i,j;
05    Word  count;
06
07    while( 1 )
08    {
09      for(j=0; j<10; j++)       //display 0~9
10      {
11        for(count=0; count<66; count++)
                    //display 66*(3ms*5) = 1s
12        {
13          for(i=0; i<5; i++)
```

```
14          {
15            P2=DISPLAY_TABLE11[j][i];
16            P1=0x01 << i;
17            DelayX1ms(3);
18          }
19        }
20        P2=0xff             //reset data
21        P1=0;          //reset control pin
22        DelayX10ms(200);      //interval 2s,
                          display off
23        }
24      }
25  }
```

行 号	说　明
07	不断的重复执行循环
09	依次显示 0~9 的数字
11	每一个数字显示 1 秒钟
13	每一列的数据代表一个数字 0~9，而每一列有 5 个数据
15	依据变量 j 来决定到底是哪一列，即哪一个数字。依据 i 变量来依次取出数据
16,17	扫描线使能并显示一段时间
20,21	重新显示下一个数字时，需要清除数据及扫描线
22	每间隔 2 秒钟依次显示数字 1 秒钟，并不断重复

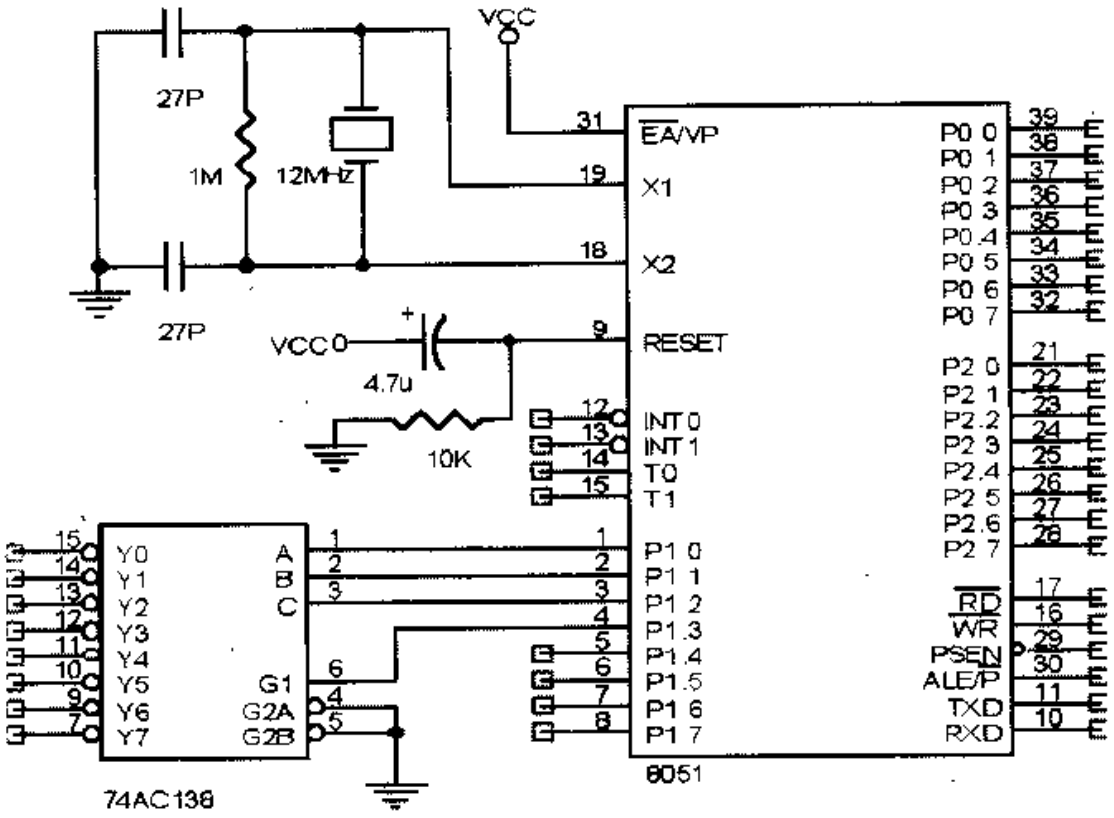
第 13 章　解码器的应用

➢ Output74138_1(　)

目的

用 IC74138 作为译码器，从 8 种状况中选择某一个输出。

电路



程序

```
01 //output2="0",other output="1"
02 void Output74138_1(void)
```

```
03 {
04     P1_3=1;          //enable
05     P1_0=1;
06     P1_1=0;
07     P1_2=0;
08 }
```

说明

以 3 条 I/O 线来选择 IC74138 Y0~Y7 的某一个输出，当被选到的输出脚位为“0”电平，其余脚皆为“1”电平。

行 号	说　明
04	使能控制线为“1”电平时动作
05~07	P1.0=“1”、P1.1=“0”、P1.2=“0”，即选择 Y1 输出为"0" 电平，其余输出为“1” 电平

➢ Output74138_2( )

目的

利用标志变量来选择输出(一)

程序

```
01 //error design:each if all julge,if true then return
02 void Output74138_2(void)
03 {
04     P1_3=1;              //enable
05     if ( FgExtOut1==1 )
06     {
07        P1_0=0;           //output1
08        P1_1=0;
09        P1_2=0;
10        return;
11     }
12     if ( FgExtOut2==1 )
```

```
13    {
14       P1_0=1;           //output2
15       P1_1=0;
16       P1_2=0;
17       return;
18    }
19    if ( FgExtOut3==1 )
20    {
21       P1_0=0;           //output3
22       P1_1=1;
23       P1_2=0;
24       return;
25    }
26    if ( FgExtOut4==1 )
27    {
28       P1_0=1;           //output4
29       P1_1=1;
30       P1_2=0;
31       return;
32    }
33    if ( FgExtOut5==1 )
34    {
35       P1_0=0;           //output5
36       P1_1=0;
37       P1_2=1;
38       return;
39    }
40    if ( FgExtOut6==1 )
41    {
42       P1_0=1;           //output6
43       P1_1=0;
44       P1_2=1;
45       return;
46    }
47    if ( FgExtOut7==1 )
```

```
48      {
49          P1_0=0;              //output7
50          P1_1=1;
51          P1_2=1;
52          return;
53      }
54      if ( FgExtOut8==1 )
55      {
56          P1_0=1;              //output8
57          P1_1=1;
58          P1_2=1;
59          return;
60      }
61      P1_3=0;                  //disable
62  }
```

说明

此程序的每一个 if 叙述都是各自独立的，如果 if 叙述的条件式不成立，则程序会一直执行判断下去，但只要有一个条件式成立，则相对应的输出立即动作后也立即退出，然而此程序流程不太结构化，应该使用下一个范例程序的结构较易懂，简单明了，此程序仅作说明之用。

➢ Output74138_3( )

目的

利用标志变量来选择输出(二)。

程序

```
01 //correct design
02 void Output74138_3(void)
03 {
04      P1_3=1;              //enable
05      if ( FgExtOut1==1 )
06      {
```

```
07        P1_0=0;           //output1
08        P1_1=0;
09        P1_2=0;
10    }
11    else if ( FgExtOut2==1 )
12    {
13        P1_0=1;           //output2
14        P1_1=0;
15        P1_2=0;
16    }
17    else if ( FgExtOut3==1 )
18    {
19        P1_0=0;           //output3
20        P1_1=1;
21        P1_2=0;
22    }
23    else if ( FgExtOut4==1 )
24    {
25        P1_0=1;           //output4
26        P1_1=1;
27        P1_2=0;
28    }
29    else if ( FgExtOut5==1 )
30    {
31        P1_0=0;           //output5
32        P1_1=0;
33        P1_2=1;
34    }
35    else if ( FgExtOut6==1 )
36    {
37        P1_0=1;           //output6
38        P1_1=0;
39        P1_2=1;
40    }
41    else if ( FgExtOut7==1 )
```

```
42    {
43       P1_0=0;           //output7
44       P1_1=1;
45       P1_2=1;
46    }
47    else if ( FgExtOut8==1 )
48    {
49       P1_0=1;           //output8
50       P1_1=1;
51       P1_2=1;
52    }
53    else
54       P1_3=0;           //disable
55 }
```

说明

条件判断式使用 if ...else 的叙述较严谨，只要表列中的任何一个条件式成立，则不再继续作其它 if 的判断，当执行完括号内的动作也无须使用 rerurn 指令，一样可返回原调用程序。

行　号	说　　明
04	使能控制线为“1”电平动作，IC74138 可以正常运作
05~10	如果标志变量 FgExtOut1=“1”，则 Y0 输出为“0”，其余为“1”电平
11~16	如果标志变量 FgExtOut2=“1”，则 Y1 输出为“0”，其余为“1”电平
17~22	如果标志变量 FgExtOut3=“1”，则 Y2 输出为“0”，其余为“1”电平
23~28	如果标志变量 FgExtOut4=“1”，则 Y3 输出为“0”，其余为“1”电平
29~34	如果标志变量 FgExtOut5=“1”，则 Y4 输出为“0”，其余为“1”电平
35~40	如果标志变量 FgExtOut6=“1”，则 Y5 输出为“0”，其余为“1”电平

行 号	说 明
41~46	如果标志变量 FgExtOut7=“1”，则 Y6 输出为“0”，其余为“1”电平
47~52	如果标志变量 FgExtOut8=“1”，则 Y7 输出为“0”，其余为“1”电平
53,54	如果没有任何标志设定即无须输出，因此除能 IC74138，即不动作

➢ Output74138_4( )

目的

使用 switch...case 语句来选择输出。

程序

```
01 //according OutputState
02 void Output74138_4(void)
03 {
04    P1_3=1;              //enable
05    switch( OutputState )
06    {
07     case 1 :
08        P1_0=0;          //output1
09        P1_1=0;
10        P1_2=0;
11        break;           //exit switch loop
12     case 2 :
13        P1_0=1;          //output2
14        P1_1=0;
15        P1_2=0;
16        break;
17     case 3 :
18        P1_0=0;          //output3
19        P1_1=1;
20        P1_2=0;
```

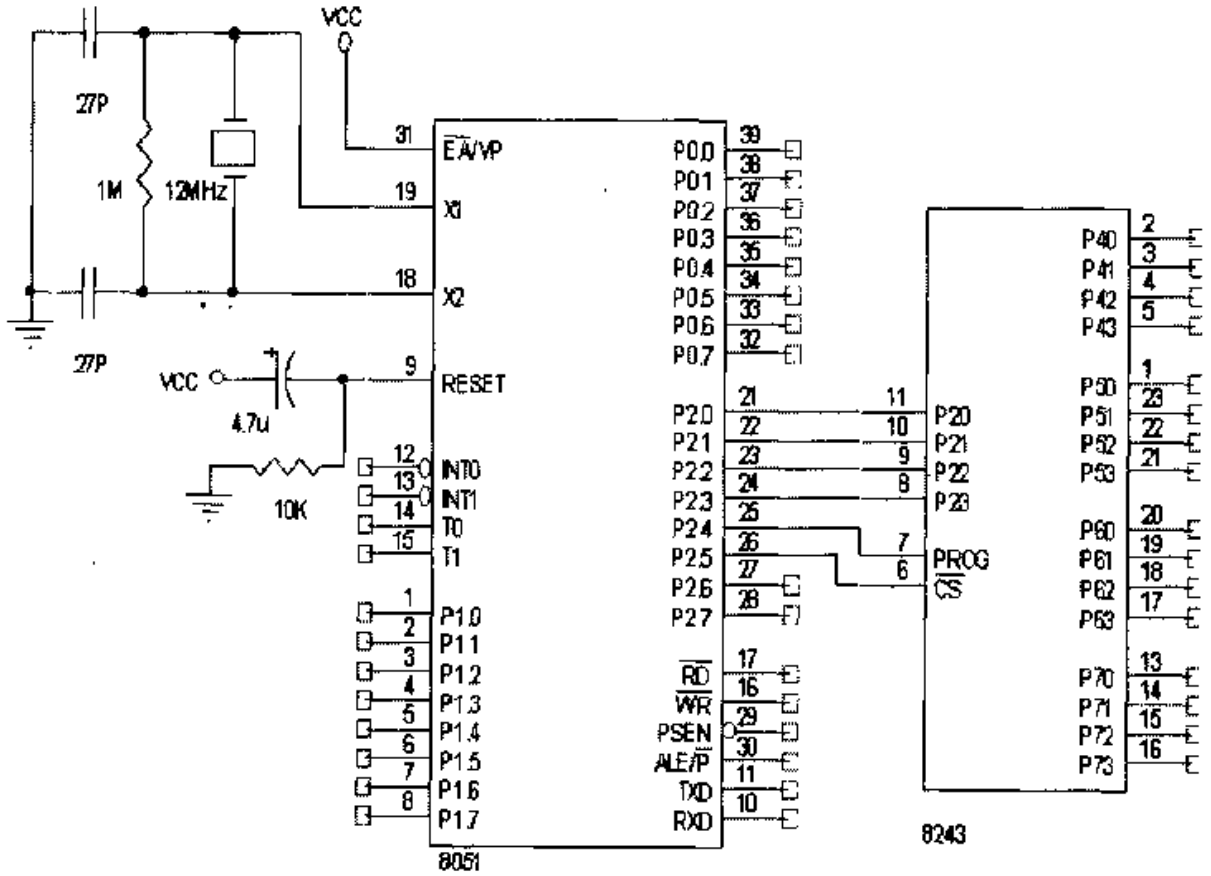
```
21          break;
22      case 4 :
23          P1_0=1;          //output4
24          P1_1=1;
25          P1_2=0;
26          break;
27      case 5 :
28          P1_0=0;          //output5
29          P1_1=0;
30          P1_2=1;
31          break;
32      case 6 :
33          P1_0=1;          //output6
34          P1_1=0;
35          P1_2=1;
36          break;
37      case 7 :
38          P1_0=0;          //output7
39          P1_1=1;
40          P1_2=1;
41          break;
42      case 8 :
43          P1_0=1;          //output8
44          P1_1=1;
45          P1_2=1;
46          break;
47      default :
48          P1_3=0;          //disable
49          break;
50      }
51  }
```

说明

以 switch 语句来选择输出是另外一种程序结构，其条件式不能使用标志变量，在此是依据变量 OutputState 的数值来执行相对应 case 内的动作，并以 break

指令来中止选择，如果都没有相符合的数值，即 OutputState 并不是介于 1~8 之间，则执行 default 内的叙述，即除能 IC74138 并跳出。

电路



说明

如上图最主要的目的是为了扩充 I/O 端口，因而使用了扩充 I/O 端口 IC8243，不但其成本昂贵软件也比较不好编写，成本较高，建议使用 IC4094，它可以扩充一个 I/O 端口，如果需要二个 I/O 端口则可以使用二个 IC4094，依此类推，请参考下一章节 IC4094 的说明。

第 14 章　扩充输出端口的应用

➢ Output4094_1(value)

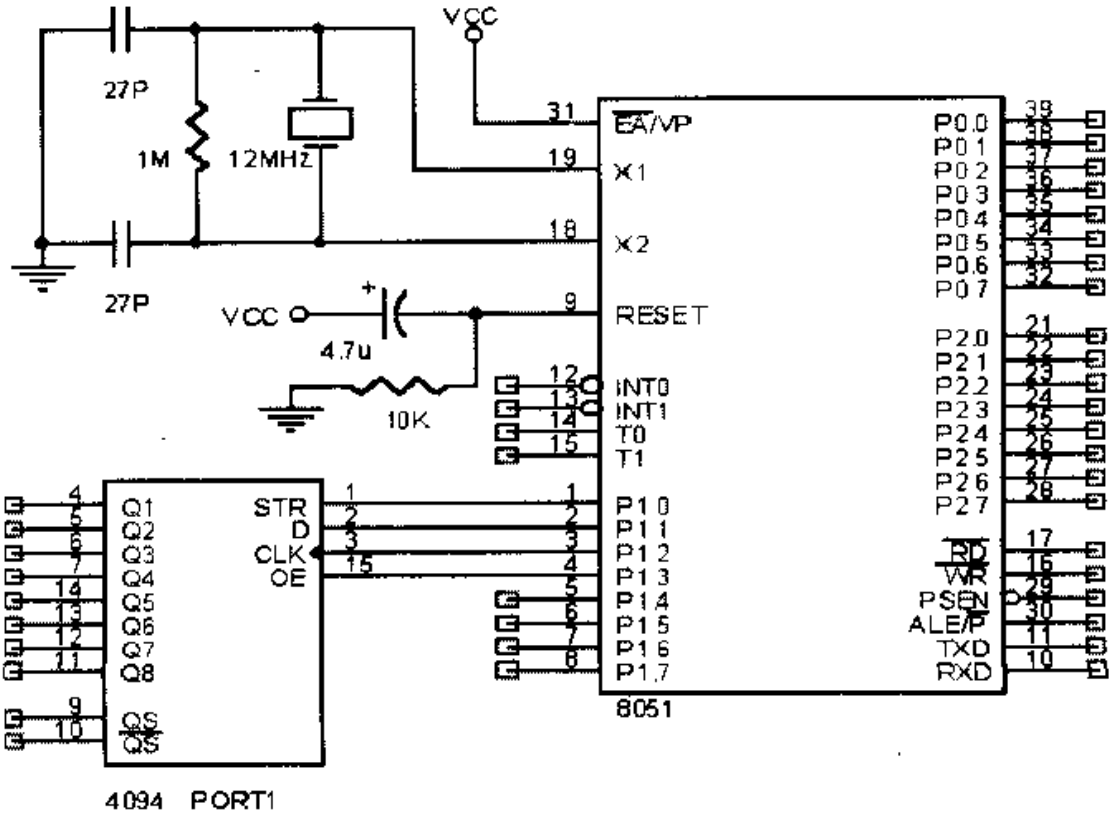
目的

用 IC4094 来扩充一个 I/O 端口。

参数

value：输出端口的数值

电路



程序

```
01 void Output4094_1(Byte value)
02 {
03     OE_PIN=1;            //P1.3=1,enable
04     DataTx8bit(value);
05     Strobe1( );
06     OE_PIN=0;            //P1.3=0,disable
07 }
08
09 //1 bit output
10 void Output4094_2(void)
11 {
12     OE_PIN=1;            //P1.3=1,enable
13     OutPutData1 |= 0x02;         //Q2:output "1"
14     DataTx8bit(OutPutData1);
15     Strobe1( );
16     OE_PIN=0;            //P1.3=0,disable
17 }
18
19 void Output4094_3(void)
20 {
21     OE_PIN=1;            //P1.3=1,enable
22     OutPutData1 &= ~(0x02); //Q2:output "0"
23     DataTx8bit(OutPutData1);
24     Strobe1( );
25     OE_PIN=0;            //P1.3=0,disable
26 }
27
28 //2 bit output
29 void Output4094_4(void)
30 {
31     OE_PIN=1;            //P1.3=1,enable
32     OutPutData1 |= 0x02;         //Q2:output "1"
```

```
33     OutPutData1 &= ~(0x04); //Q3:output "0"
34     DataTx8bit(OutPutData1);
35     Strobe1( );
36     OE_PIN=0;            //P1.3=0,disable
37 }
38
39 void DataTx8bit(Byte value)
40 {
41     Byte  i;
42
43     for(i=0; i<8; i++)
44     {
45       if ( value & 0x80 )
46         DATA_PIN = 1;  //P1.1=1
47       else
48         DATA_PIN = 0;
49       value <<= 1;
50
51       ClkDelay( );
52
53       CLK_PIN = 1;      //clk="0"->"1"->"0"
54       ClkDelay( );
55
56       CLK_PIN = 0;      //P1.2=0
57       ClkDelay( );
58     }
59 }
60
61 void Strobe1(void)
62 {
63     STROBE1_PIN = 0;    //P1.0="0"->"1"->"0"
64     ClkDelay( );
65
66     STROBE1_PIN = 1;
```

```
67     ClkDelay( );
68
69     STROBE1_PIN = 0;
70     ClkDelay( );
71 }
72
73 //50uS
74 void ClkDelay(void)
75 {
76     Byte i;
77
78     for(i=0; i<6; i++)
79       ;
80 }
```

说明

当微处理器的 I/O 端口不够使用，可用 IC4094 来扩充 I/O，其程序的编写设计必须是字节寻址模式，不能像 8051 的 I/O 一样可以位寻址，而且不能作为输入端口只能作为输出端口，因而软件设计会较复杂。IC4094 以串行的数据作传输，也必须注意 data 和 clk 的时钟是否符合，上图中 IC4094 的 OE 脚，即第 15 支脚也可以不由微处理器控制，直接接上+5V 即可。

行 号	说　明
03	使能 IC4094
04	传送数据到 IC4094 内部的例程，一次传送一个字节
05	必须再送一个 strobe 信号，数据值才会由 Q1~Q8 输出
06	除能 IC4094，即不能动作了
10	示范 Q2 输出为“1”电平，其余脚并不会改变原来的电平
13	将数据值和 0x02 作 OR 门，即数据值加上 0x02 的数值，所以 Q2 输出为“1”电平
14,15	将数据输出至 Q1~Q8
19	屏蔽 Q2 的示范程序，即 Q2 输出为“0”电平

行　号	说　　明
22	将数据值的位 1(bit1)清除为“0”
23,24	将屏蔽后的数据输出至 Q1~Q8
29	同时要作屏蔽为“0”和强迫为“1”的示范程序
32,33	Q2 输出为“1”电平、Q3 输出为“0”电平
34,35	传送数据
39	传送 8 位数据的例程，从高位开始
43	循环作 8 次
45~48	当 bit7 等于“1”，则 data pin 输出为“1”，否则 data pin 输出为“0”
49	数值左移一个位，即继续测试 bit6，依此类推，直到 bit0 为止
51~57	不管是“0”或“1”位，当 data pin 已经准备好，则须将 clk 作正脉波的输出，即"0"->"1"->"0"的变化，ClkDelay( )函数为时序而做一小段的延迟
61~71	STR 脚作正脉波的输出，即"0"->"1"->"0"的变化
74~80	一小段的延迟，只要符合 IC4094 CLK 和 DATA 脚的时序即可

➢ Output4094_5(outputstate,value)

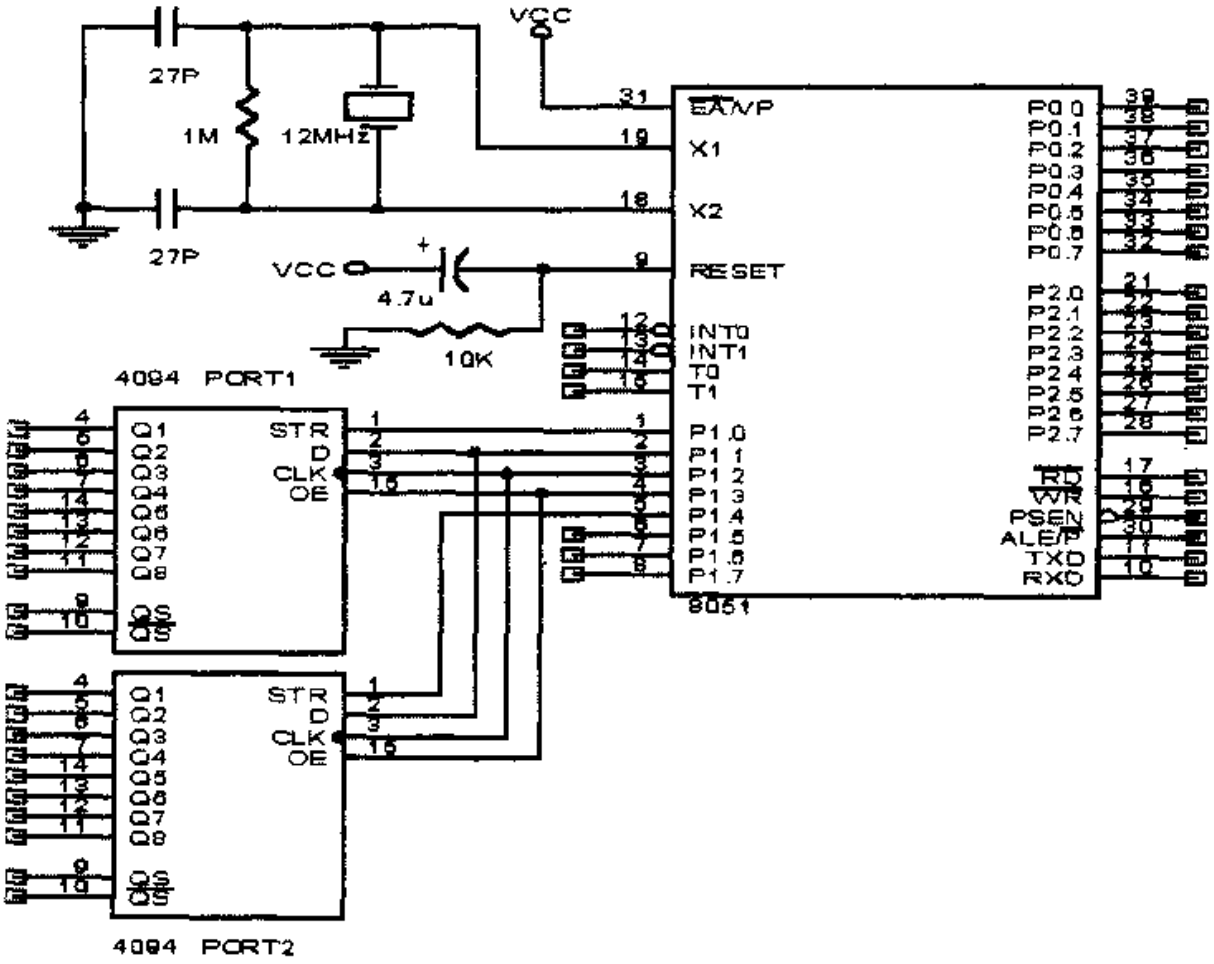
目的

以两个 IC4094 来扩充二个 I/O 端口。

参数

outputstate：选择哪一个输出端口，第一或第二个
value：输出端口的数值

电路



程序

```
01 void Output4094_5(Byte outputstate,Byte value)
02 {
03    OE_PIN=1;              //P1.3=1,enable
04    if ( outputstate==1 )
05    {
06       DataTx8bit(value);
07       Strobe1( );
08    }
09    else if ( outputstate==2 )
10    {
11       DataTx8bit(value);
12       Strobe2( );
```

```
13      }
14      OE_PIN=0;              //P1.3=0,disable
15  }
16
17  void Strobe2(void)
18  {
19      STROBE2_PIN = 0;          //P1.4=="0"->"1"->"0"
20      ClkDelay( );
21
22      STROBE2_PIN = 1;
23      ClkDelay( );
24
25      STROBE2_PIN = 0;
26      ClkDelay( );
27  }
```

说明

当 I/O 的使用确实很多，可再扩充一个 IC4094 便可再多出一个输出端口，而微处理器本身只是多了一支 I/O 脚而已，由于无法作位寻址(直接设定或清除输出脚)，因此想要设定或清除某一个输出脚 Q1~Q8，必须将原来的数据值作 OR 门或 AND 门的逻辑运算。

行　号	说　　明
03	使能 IC4094，当 OE="1"为使能、OE="0"为除能不动作
04~08	当输出状态变量 outputstate=1，则做第一个输出端口
09~13	当输出状态变量 outputstate=2，则做第二个输出端口
14	OE="0"为除能，IC4094 不动作
17~27	第二个输出端口 STR 脚发送正脉冲，即 P1.4 做"0"->"1"->"0"的变化

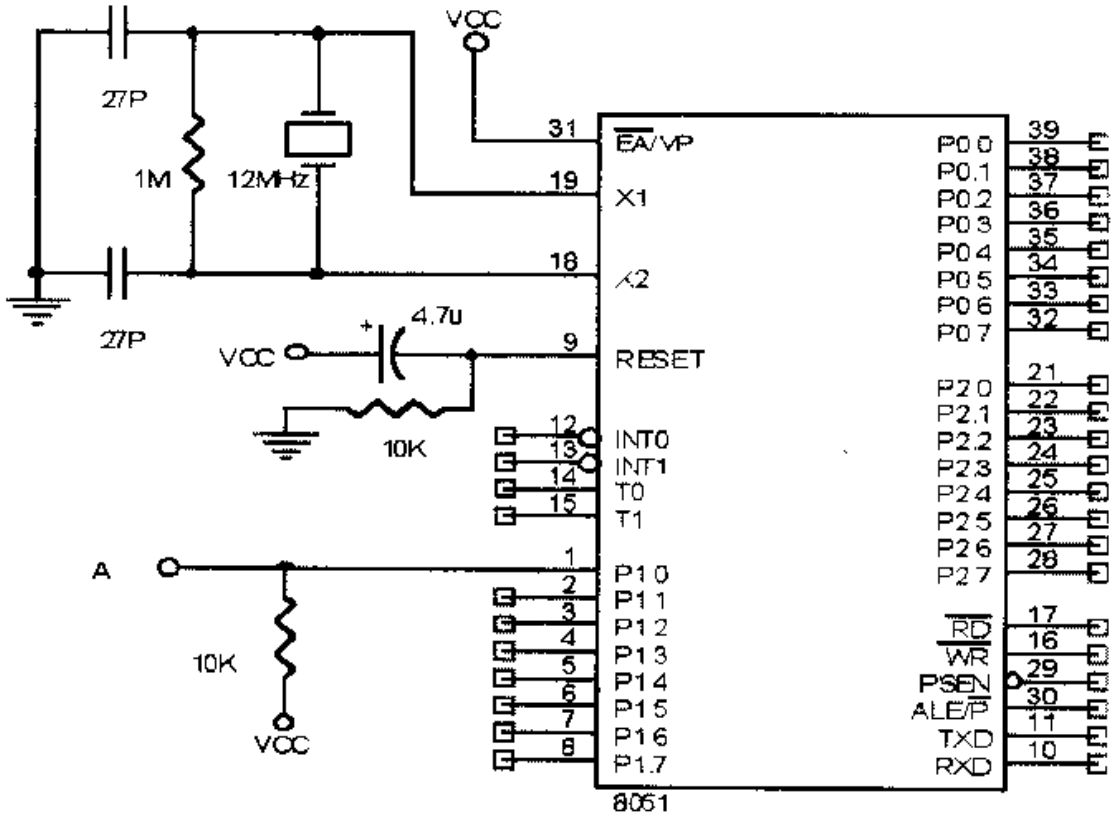
第 15 章　脉冲的应用

➢ OutPulse1(　)

目的

从 I/O 脚输出一个脉冲。

电路



程序

```
01 //duty cycle=1/5,one pulse
02 void OutPulse1(void)
03 {
```

```
04      P1_0=1;
05      DelayX1ms(1);
06      P1_0=0;
07      DelayX1ms(5);
08  }
```

说明

由微处理器产生一个正脉冲给外界外围的装置，最具有弹性而且简单，例如：产生一个正脉冲当作重置信号，或作为使能控制线，往往比利用硬件线路来达成而更简单。

行　号	说　　明
04,05	设定 I/O 脚输出 P1.0 为“1”，并延迟 1mS
06,07	设定 I/O 脚输出 P1.0 为“0”，并延迟 5mS，正常状态 P1.0 的输出保持在 0 电平

➢ OutPulse2(count)

目的

可以产生多个脉冲。

参数

count：产生几个脉冲的参数，范围从 0~255

程序

```
01 //pulse count generator
02 void OutPulse2(Byte count)
03 {
04     Byte  i;
05
06     for(i=0; i<count; i++)
07     {
08        P1_0=1;
09        DelayX1ms(1);
```

```
10        P1_0=0;
11        DelayX1ms(5);
12     }
13 }
```

说明

可控制个数的方波产生器，更具弹性来适合不同的需求。

行 号	说　明
06	产生方波的循环
08,11	输出一个正脉冲，即“0” -> “1” -> “0” 的变化，duty cycle 是 1/5。

➢ OutPulse3( )

目的

不断的产生脉冲，直到时间终了才停止。

程序

```
01 void OutPulse3(void)
02 {
03    PulseCount=TIME_5SEC;       //set pulse stop
04    TR1 = 1;                  //start timer
05
06    while( 1 )           //continue square generate
07    {
08       P1_0=1;
09       DelayX1ms(1);
10       P1_0=0;
11       DelayX1ms(5);
12       if ( PulseCount == 0 )   //if PulseCount=5s,then
                          stop generate square
13          break;          //exit while loop
14    }
```

```
15     TR1 = 0;                    //stop timer
16 }
17
18 void PulseOff_Timer1ISR (void) interrupt 3 using 2
19 {
20     TL1 = CLOCK_40MS & 0xff;  //timer=40ms,interrupt 1 time
21     TH1 = CLOCK_40MS >> 8;
22     TF1 = 0;
23
24     if ( PulseCount != 0 )
25        PulseCount--;          //after 40ms,then PulseCount-1
26 }
```

说明

按照时间来停止脉冲的输出，需要一个定时器来终止脉冲的产生。

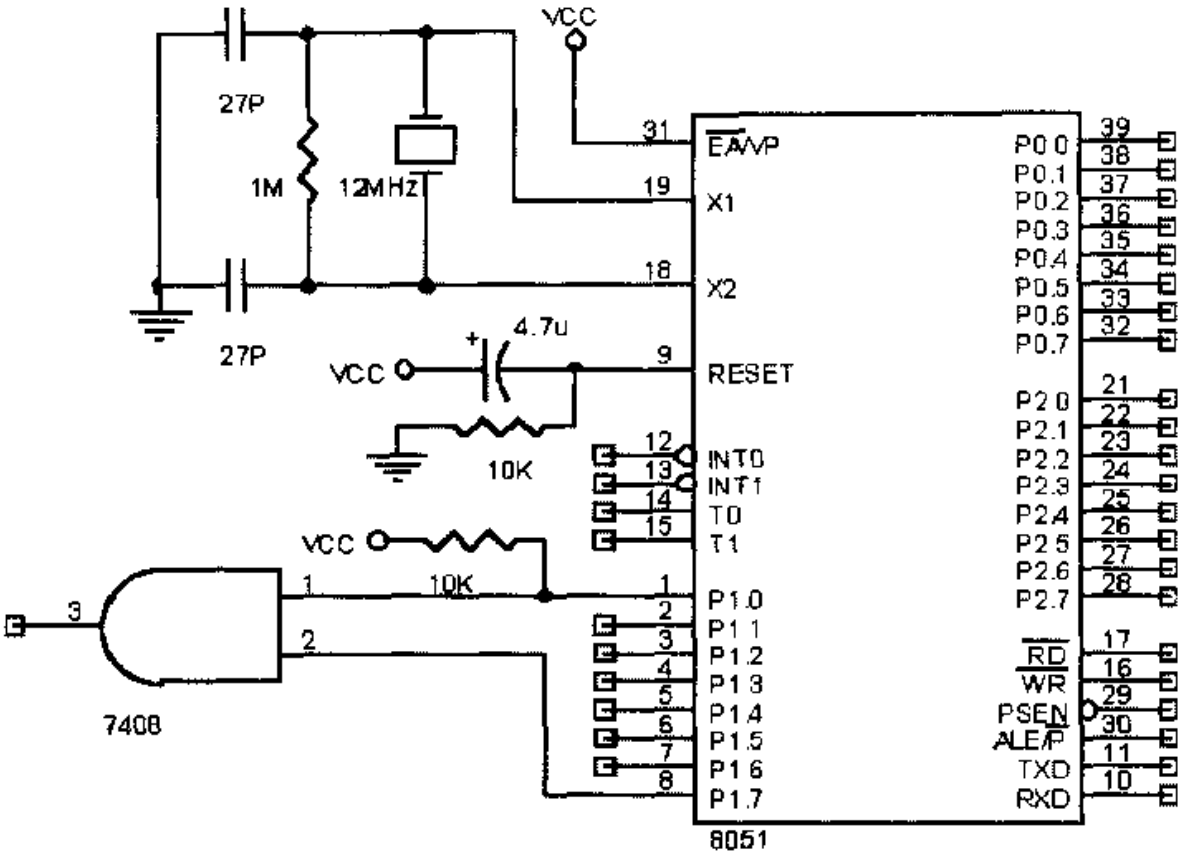
行　号	说　　明
03,04	设定脉冲停止时间为 5 秒，并开始计时
06	不断地重复执行循环来输出脉冲
08~11	通过 I/O 脚输出一个脉冲
12,13	计时的变量值等于 0，即时间终了，则跳出 while 循环，其中 break 指令是用于跳出循环
15	停止计时
18	计时中断例程
20~22	设定 TIMER1 为 40mS 就中断一次，并清除溢出标志
24,25	计时 5 秒的变量值，每 40mS 减 1，直到 0 为止

➢ OutPulse4( )

目的

利用 AND 门来停止脉冲的产生。

电路



程序

```
01 //AND gate
02 void OutPulse4(void)
03 {
04    while( 1 )           //continue square generate
05    {.
06       P1_7=1;           //AND gate,enable clk
07
08       P1_0=1;           //pulse output
09       DelayX1ms(1);
10       P1_0=0;
11       DelayX1ms(5);
12    }
13 }
14
```

```
15 void OutPulse5(void)
16 {
17     while( 1 )
18     {
19        P1_7=1;
20
21        P1_0=0;           //enable clk,inverse clk
22        DelayX1ms(1);
23        P1_0=1;
24        DelayX1ms(5);
25     }
26 }
```

说明

利用 AND 门来控制是否输出，是利用 AND 门的特性，只要一端为“0”电平，则不管另一端为何，其输出就一直为“0”电平，当一端为“1”电平，则输出就变成另一端的原始输入信号了。

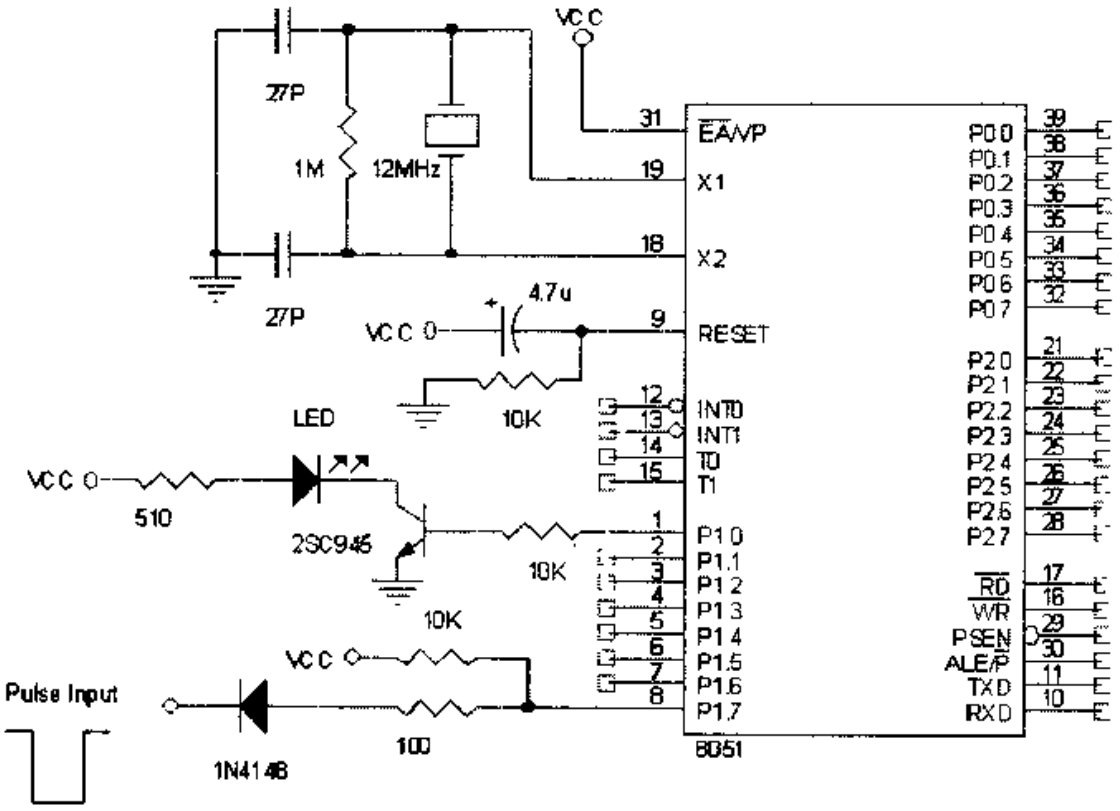
行 号	说　　明
06	AND 门的一端电平为“1”；即所谓使能控制线，如果电平为“0”则没有输出，即没有动作
08~11	AND 门的另一端输入信号由 P1.0 来产生正脉冲
15~26	产生负脉冲的程序，即电平“1”的时间长，电平“0”的时间短

➢ PulseDetect1(　)

目的

负脉冲信号的检测(一)

电路



程序

```
01 void PulseDetect1(void)
02 {
03    while( 1 )
04    {
05       Subroute1( );//null
06
07       Det_Pulse1( );   //repeat do route
08
09       Subroute2( );//null
10    }
11 }
12
13 void Det_Pulse1(void)
14 {
15    if ( P1_7==0 )      //detect:"0" level input
```

```
16    {
17        LedOn( );     //led on
18        DelayX1ms(1);
19        LedOff( );        //led off
20        DelayX1ms(5);
21    }
22 }
23
24
```

说明

当检测到负脉冲，即有一个信号电平是由"1"到"0"到"1"变化，LED 就闪烁一下。此程序的写法非常不合适，太草率了容易出错，如果负脉冲的“0”电平够长，则程序在行号 15 将会一直作判断且条件式也一直成立，则 LED 会不断闪烁，直到负脉冲恢复为“1”电平为止，因此，程序在执行时可能会让 LED 闪烁二下或闪烁三下，视负脉冲“0”电平的时间而定，如果负脉冲的信号一直为“0”电平，则 LED 也将会一直闪烁下去。

行 号	说　　明
01	主程序
05	其他子程序为空
07	检测负脉冲的程序，会一直执行
15	如果脉冲信号为“0”电平，则执行 LED 闪烁一下，又不断地判断是否为“0”电平会造成错误
17~20	LED 闪烁一下的程序

➢ PulseDetect2( )

目的

负脉冲信号的检测(二)。

程序

```
01 void PulseDetect2(void)
02 {
03    while( 1 )
```

```
04     {
05        Subroute1( );
06        Det_Pulse2( );
07        Subroute2( );
08     }
09 }
10
11 //not good design
12 void Det_Pulse2(void)
13 {
14     if ( P1_7==0 )
15     {
16        LedOn( );          //led on
17        DelayX1ms(1);
18        LedOff( );             //led off
19        DelayX1ms(5);
20        while ( P1_7==0 );        //wait until P1.7="1"
21     }
22 }
```

行 号	说 明
01~09	主程序，一般会有很多的其他子程序，行号 05，07 用于调用
14	输入信号为“0”电平则继续做下去，否则离开返回调用程序
16~19	LED 闪烁一下
20	如果脉冲信号为“0”电平则一直等待，直到输入信号回到“1”为止；但如果输入信号有异常一直处于“0”电平，则此程序将会一直执行，即“死机”。因此，编写程序也需要考虑异常程序处理，以避免造成错误，如下一个范例所示

➢ PulseDetect3( )

目的

负脉冲信号的检测(三)

程序

```
01 void PulseDetect3(void)
02 {
03    while( 1 )
04    {
05       Subroute1( );
06
07       Det_Pulse3( );
08
09       Subroute2( );
10    }
11 }
12
13 //good design
14 void Det_Pulse3(void)
15 {
16    if ( FgPulse==0 )
17    {
18       if ( P1_7==0 )
19       {
20          FgPulse=1;
21
22          LedOn( );          //led on
23          DelayX1ms(1);
24          LedOff( );     //led off
25          DelayX1ms(5);
26       }
27    }
28    else
29    {
30       if ( P1_7!=0 )        //detect normal:"1"
31          FgPulse=0;
32    }
33 }
```

说明

此程序可以改善前面二个范例的错误，在程序中多加了一个标志变量 FgPulse 决定程序的流程，请多利用标志变量的方法来达到程序的完整性及避免出错，这是非常重要的设计理念，请多注意与应用。

行 号	说 明
01~11	主程序
14	检测负脉冲的例程，完整且实用
16	FgPulse 为负脉冲来了的负边缘标志，因此不会管“0”电平有多长，当脉冲“0”电平时此标志为“1”，脉冲恢复为“1”电平时此标志清除为“0”，以准备下一次脉冲检测
18~26	负脉冲进来时则设定 FgPulse 为“1”，并使 LED 闪烁一下
28~32	LED 闪烁一下之后，即表示负脉冲已经被检测到。如果脉冲一直为“0”电平，则返回原调用程序，如果 P1.7 不等于 0 即脉冲恢复为“1”电平，则清除 FgPulse

➢ PulseGenerator( )

目的

产生 100 HZ 的方波 —— 使用延迟子程序和中断例程

程序

```
01 void PulseGenerator(void)
02 {
03    Byte  i;
04
05    SquareCount=0;
06    P1_0 = 0;        //initial P1.0="0"
07
08    while( 1 )
09    {
10       DelayPulse( );
11
12       i++;
13    }
```

```
14 }
15
16 //delay method
17 //because cpu execute time,100Hz change to 95 Hz
18 void DelayPulse(void)
19 {
20     Delay50uS(1);        //100hz square=10000uS
21     SquareCount++;       //duty cycle=10000us/2=5000us
22     if (SquareCount==100)   //50us*100=5000us
23     {
24       if ( P1_0==0 )
25       {
26         P1_0 = 1;        //P1.0="1"
27       SquareCount=0;     //reset SquareCount
28       }
29       else
30       {
31         P1_0 = 0;
32       SquareCount=0;
33       }
34     }
35 }
36
37 //timer method
38 //100hz square=10000uS,duty cycle=10000us/2=5000us
39 void Pulse_Timer1ISR (void) interrupt 3 using 2
40 {
41     if ( P1_0==0 )
42       P1_0 = 1;          //P1.0="1"
43     else
44       P1_0 = 0;
45
46     TL1 = CLOCK_5000us & 0xff;  //timer=5000us,interrupt 1
                                   time
47     TH1 = CLOCK_5000us >> 8;
```

```
48      TF1 = 0;
49  }
```

说明

利用编写程序的方法使 I/O 脚输出方波，而方波的频率、duty cycle 可以随系统的要求而作变更，确实比硬件电路来的更具弹性，但如果要产生数百 KHZ 的频率，用软件来实现较困难。以延迟一段时间后令计数一直加 1，并判别计数值的方法来产生方波，其精确度不够，因为要考虑其它指令执行的时间，要调整频率可修改 Delay50uS( )函数的计数值，另外利用计时中断的方法其频率的精确度非常高。

行 号	说　明
01~14	初始化方波计数器及输出为“0”电平并调用方波产生器，其中 i 变量只是调试时作为中断点使用并没有其他意义
20,21	每 Delay 50 uS 则计数变量加 1
22~34	如果计数值等于 100，即 50uS 乘以 100 等于 5000uS 其输出电平要转变 1 次，因此 5000uS 乘以 2 等于 10000uS 为 100Hz，并将计数值清除，以便下一周期重新计数，即重新产生下一周期
41~44	系统 5000uS 计时终了，产生中断的方法，将之前输出“0”电平的转变为“1”，如果之前输出为“1”就输出为“0”
46~48	重新载入计时值 5000uS 并将中断溢出标志清零

➢ PulseDuty1_Timer1ISR( )

目的

产生 100 HZ 的方波 —— 具有 1/2 Duty Cycle 的变化

程序

```
01 //100Hz,1/2 duty cycle extend
02 //extend "1" level duty,according DutyCycleValue
03 void PulseDuty1_Timer1ISR (void) interrupt 3 using 2
04 {
05     Word temp;
06
```

```
07      if ( P1_0==0 )
08      {
09         P1_0 = 1;
10
11         //temp=5000us-255*19=155us ,"1" is long
12         temp = CLOCK_5000us - DutyCycleValue * 19;
13      }
14      else
15      {
16         P1_0 = 0;
17
18         //temp=5000us+255*19=9845us,"0" is short
19         temp = CLOCK_5000us + DutyCycleValue * 19;
20      }
21
22      TL1 = temp & 0xff;        //set interrupt time
23      TH1 = temp >> 8;
24      TF1 = 0;
25  }
```

说明

此程序利用计时中断例程来产生 100 HZ 的方波，并按 DutyCycleValue 变量来变化 duty cycle，DutyCycleValue 数值越大则“1”电平的时间就越长，反之就越短，而当 DutyCycleValue 数值等于 0，则输出方波的“1”电平和“0”电平的时间就相同，即等于 1/2 duty cycle。

行 号	说　明
05	16 位的 LOCAL 计数变量
07~13	计时终了则将目前的输出反转，当之前为“0”电平则立即输出为“1”电平，并依据 DutyCycleValue 来产生“1”电平的时间做延时
14~20	当之前为“1”电平则立即输出为“0”电平，并开始计时“0”电平输出时间
22~24	将计时的时间变量载入计时器内，并将中断溢出标记清零

➢ PulseDuty2_Timer1ISR( )

目的

产生 100 HZ 的方波 —— 随意变化 Duty Cycle

程序

```
01 //100Hz,total duty cycle extend
02 //extend "1" level duty,according DutyCycleValue
03 void PulseDuty2_Timer1ISR (void) interrupt 3 using 2
04 {
05    Word temp;
06
07    if ( P1_0==0 )
08    {
09       P1_0 = 1;
10
11       //temp=10000us-255*39=55us ,"1" is long
12       temp =  65536 - DutyCycleValue * 39;
13    }
14    else
15    {
16       P1_0 = 0;
17
18       //temp=255*39=9945us,"0" is short
19       temp = 65536 -10000 + DutyCycleValue * 39;
20    }
21
22    TL1 = temp & 0xff;      //set interrupt time
23    TH1 = temp >> 8;
24    TF1 = 0;
25 }
```

说明

100HZ 的周期是 10000uS，而 DutyCycleValue 是一个 Byte 的公用变量，最

大值为 255，而 10000 除以 255 等于 39.21，取整数为 39，所以当 DutyCycleValue 为最大值时，仍然有 10000-255*39=55uS 时间的“1”电平，而微处理器 8051 的定时器是上数型，因此行号 12 是以 65536 减去待计时的时间的表达式作为实际计时的时间。

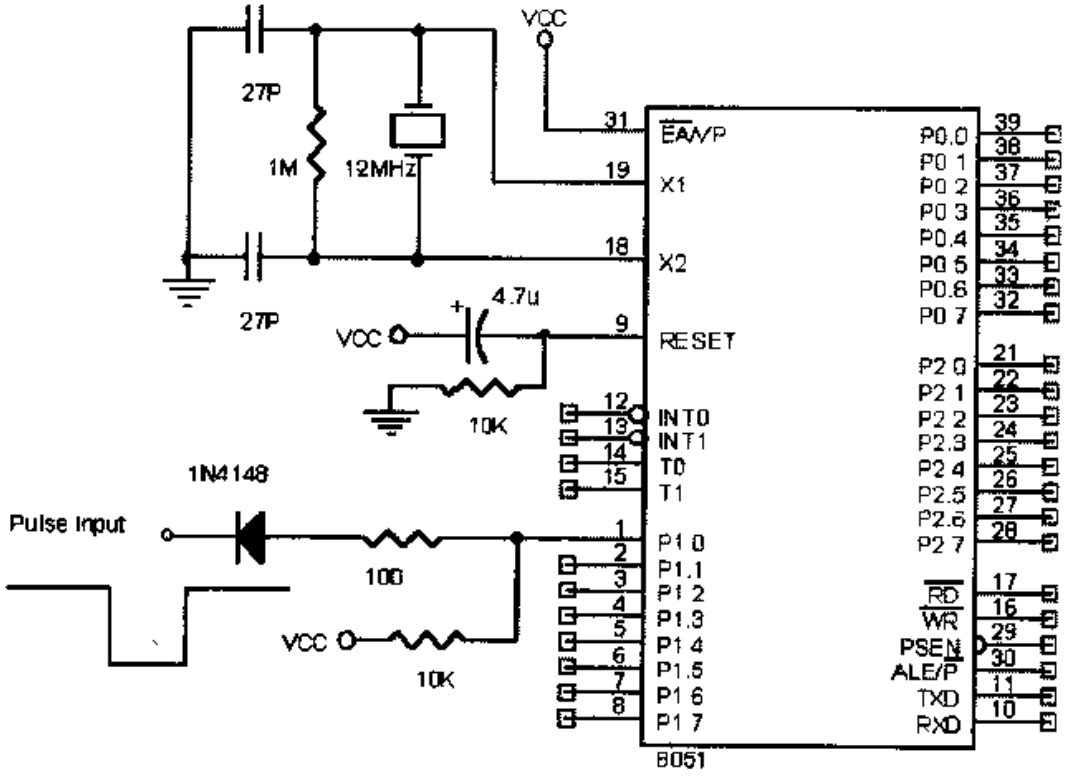
行　号	说　　明
05	16 位的 LOCAL 计数变量
07~13	如果 P1.0=“0”则输出“1”电平并开始计时其输出时间
14~20	如果 P1.0=“1”则输出“0”电平并开始计时其输出时间
22~24	将计时的时间变量载入计时器内，并将中断溢出标记清零

➢ CheckPulseCome(　)

目的

判别脉冲的方法。

电路



程序

```
01 void CheckPulseComc(void)
02 {
03    if ( FgPulse==0 )
04    {
05       if ( P1_0==0 )
06          FgPulse=1;     //pulse:"0" comming
07    }
08    else
09    {
10       if ( P1_0!=0 )
11          FgPulse=0;
12    }
13 }
```

说明

系统中如有异常或外围设备发生不同的状况，会以一脉冲告知主控制器也就是微处理器，因此微处理器必须随时待命检测，一旦发生就实时处理，此程序只是检测脉冲的应用，其脉冲发生的处理动作须依据系统需求而定。

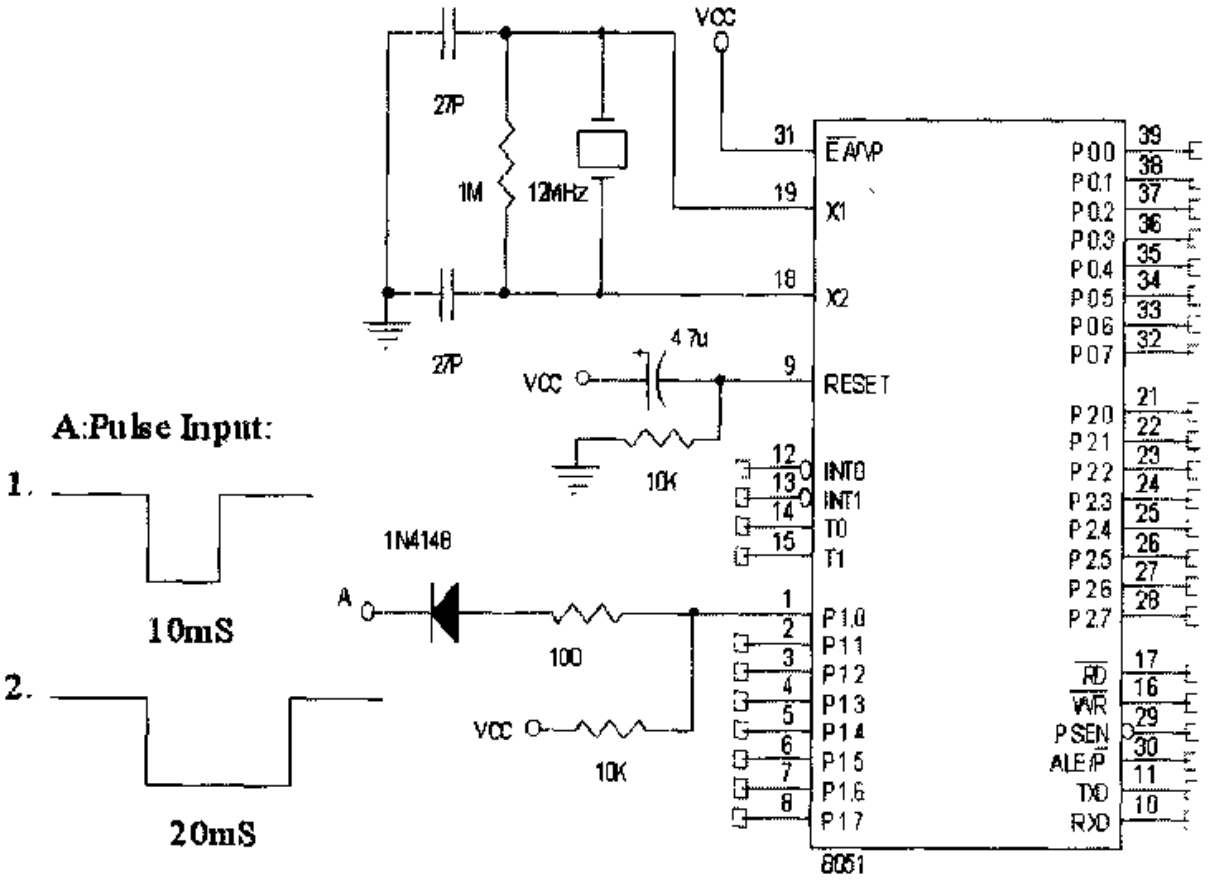
行 号	说 明
03	检测到脉冲的标志，如果恢复为“0”则可继续检测脉冲
05,06	输入信号如果为“0”电平，则设定脉冲标志 FgPulse
08~12	一旦脉冲标志已设定，则判断脉冲是否继续为“0”电平，当电平恢复为“1”则清除脉冲标志，否则离开并返回原调用程序

➢ CheckPulseWidth( )

目的

判别脉冲的宽度，而作不同的处理。

电路



程序

```
01 //"0" level short,then trigger warmming,FgPulseShort=1
02 //"0" level long ,then trigger on     ,FgPulseLong=1
03 void CheckPulseWidth(void)
04 {
05     FgPulseShort=0;
06     FgPulseLong =0;
07     DutyCount   =0;
08
09     while ( P1_0==0 )   //if P1.0="1",then don't do
10     {
11        DelayXlms(1);
12        if ( P1_0==0 )
```

```
13          DutyCount++;   //pulse:"0" comming
14      }
15
16      if ( ((0+3)<DutyCount) && (DutyCount<(10+3)) )
17        FgPulseShort=1;
18      else if ( ((10+3)<DutyCount) && (DutyCount<(20+3)) )
19        FgPulseLong=1;
20  }
```

说明

在实际应用中有时会依据脉冲时间的长短来表示不同的异常信号并需作不同的处理，例如：汽车被轻微碰撞则送一个短暂的脉冲至微处理器，而当被大力的碰撞则送一个较长时间的脉冲至微处理器，因此需要检测出脉冲时间的长短以决定触发源而作不同的处理动作。

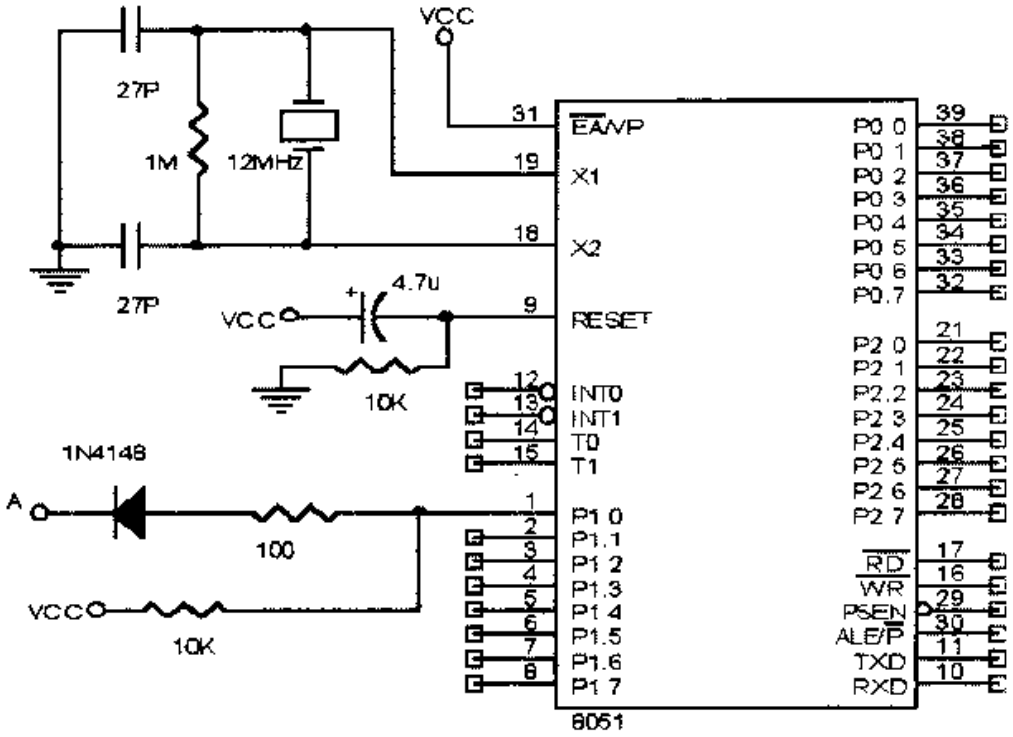
行 号	说　明
05~07	清除长、短脉冲及计数器
09~14	当电平为“0”时开始计数，并延迟一段时间再去检测是否继续为“0”电平，如果是“0”电平则不断累计计数器的数值，当电平恢复为“1”则判别计数器的数值就知道脉冲的长短了
16~19	计数器的数值介于 3~13 之间为短脉冲，当计数器的数值介于 32~23 自己之间为长脉冲，并设定对应的标志标量

➢ CheckPulseData( )

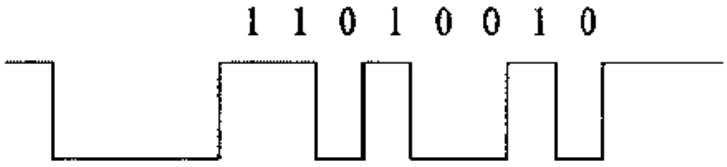
目的

将脉冲信号还原成数据。

电路



A:Pulse Input:



synchronous singal　　5mS

程序

```
01 //pulse width=10ms,delay 5ms check bit "1" or "0"
02 void CheckPulseData(void)
03 {
04    Byte  i,temp=0;
05
06    if ( P1_0==1 )
07       return;           //P1.0="1",normal
08
09    //synchronous singal
10    while ( P1_0==0 );      //wait while P1.0="0"
11
```

```
12      for(i=0; i<8; i++)
13      {
14          temp <<=1;
15          DelayX1ms(5);
16          if ( P1_0==1 )  temp |= 0x01;
17      }
18
19      PulseData=temp;
20  }
```

说明

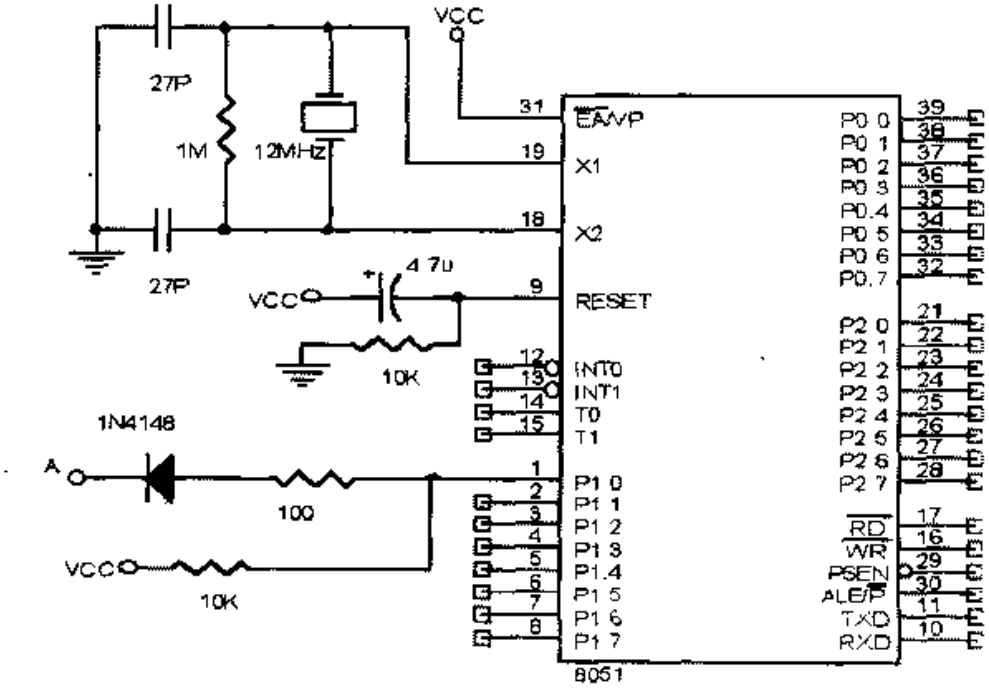
如果数据以脉冲形式来传送，必须还原成数据，由连续 8 个位即可组合成一个 Byte，之后即可作适当应用。为了正确地传送和接收，使用由“0”电平转态成“1”电平的同步信号可以更稳定，但使用固定时间来撷取位则脉冲“0”及“1”电平的宽度也须固定，否则会造成错误。

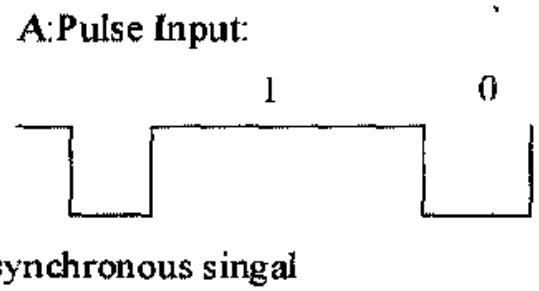
➢ CheckPulseHiLow( )

目的

检测“1”或“0”电平的时间长短 —— 有作 Timeout 超时处理

电路





程序

```
01 //check a hi-low pulse
02 void CheckPulseHiLow(void)
03 {
04    Byte  i,j;
05    Word  timeout;
06
07    LowPulseCount=0;        //initial
08    HiPulseCount =0;
09
10    //max 500ms,if excess that is timeout
11    for(timeout=0; timeout<500; timeout++);
12    {
13      //test "1" level,1ms count+1
14      for(j=0; j<20; i++)
15
16        if ( P1_0==0 ) goto TEST_LOW;
17        Delay50uS(1);      //"1" level,50us*20=1ms, then count+1
18      }
19      HiPulseCount++;
20    }
21    FgTimeout=1;            //represent timeout
22    return;
23
24 TEST_LOW:
25    //max 500ms,if excess that is timeout
26    for(timeout=0; timeout<500; timeout++);
27    {
```

```
28        //test "0" level,1ms count+1
29        for(j=0; j<20; i++)
30
31          if ( P1_0==1 ) goto TEST_END;
32          Delay50uS(1);       //"0" level,50us*20=1ms, then count+1
33        }
34        LowPulseCount++;
35      }
36      FgTimeout=1;              //represent timeout
37      return;
38
39  TEST_END:
40      FgTimeout=0;              //without timeout
41      return;
42  }
```

说明

将每个“1”电平的时间和“0”电平的时间分别计算出来，1 个周期代表 1 个位，如果“1”电平时间长而“0”电平的时间短即表示为“1”，如果“1”电平时间短而“0”电平的时间长即表示为“0”，用此方法撷取到的数据才会更稳定也更正确，此种方法称为译码，即依据编码格式的不同而使用程序技巧将它还原成原来的数据，程序中也利用 timeout 的技巧以避免死机，即检测时间超过一固定时间（例如 0.5 秒，以计数循环的次数来表示）都没反应，就设定时间超过标志并退出程序循环。

行 号	说　明
07,08	初始计数值
11	设定 timeout 时间，如果作了 500 次信号一直没有转态，表示“1”电平时间太长了，则设定时间超过标志
14~19	Delay50uS 乘以 20 次等于 1mS，即每经过 1mS 的时间 HiPulseCount 就累加 1，只要转态变成“0”电平则立即跳到“0”电平时间的检测
21,22	时间超过则设定标志并返回原调用程序
26	timeout 时间设定 500 次

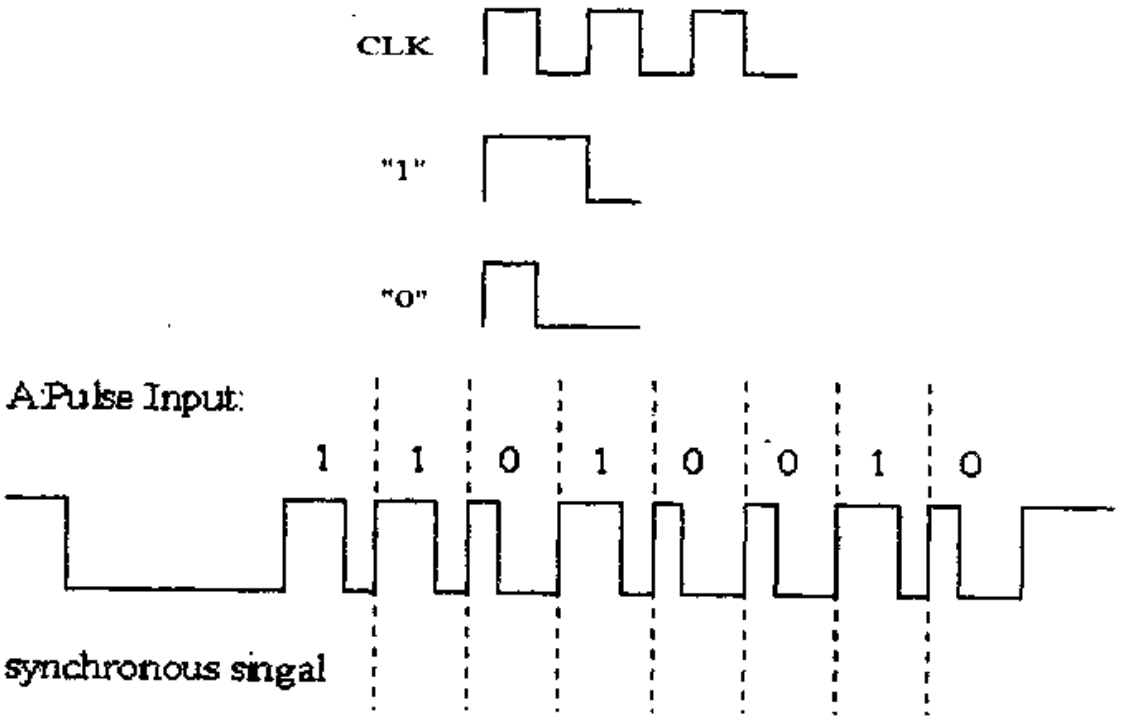
行　号	说　　明
29	将 1mS 又切分成更细的时间单位 50uS，因此需要作 20 次的循环，如果脉冲又转态为“1”电平，即结束此信号格式的判别
32	延迟 50uS
34	每经过 1mS 的脉冲信号还是为“0”电平的话，计数值 LowPulseCount 就累加 1
40,41	未超过时间就检测完毕，表示正常没有超时，因此标志设定为 0 并返回原调用程序

➢ PulseDecoder( )

目的

译码器的应用 —— 还回原来的数据

电路



程序

```
01 //"1":2 hi,1 low pulse width,count 100
02 //"0":1 hi,2 low pulse width,count 50
03 void PulseDecoder(void)
```

```
04 {
05    Byte  i,temp=0;
06
07    //synchronous singal
08    while ( P1_0==1 );      //wait while P1.0="1"
09    while ( P1_0==0 );      //wait while P1.0="0"
10    for(i=0; i<8; i++)      //1 byte
11    {
12       CheckPulseHiLow();
13       if ( FgTimeout==1 )  //represent timeout
14          return;
15
16       temp <<=1;
17       if ( ((HiPulseCount>85)&&(HiPulseCount<115))
18           && ((LowPulseCount>0)&&(LowPulseCount<65)) )
19          temp |= 0x01;
20       else if ( ((HiPulseCount>0)&&(HiPulseCount<65))
21            && ((LowPulseCount>85)&&(LowPulseCount<115)) )
22          temp |= 0x00;
23    }
24    PulseData=temp;
25 }
```

说明

此程序是上一个范例的应用，可作为编码器输出的还原用即所谓的译码。例如：密码或命令指令，如果需要更精确、安全，可将脉冲信号扩大为4个frame，1个frame有4个Byte的密码，紧接着才是真正需要的数据，总之，要使用微处理器破解脉冲信号，必须详细阅读编码的格式，才能使用程序技巧将它译码出来。

行 号	说　明
05	temp初始值为0
08,09	等待同步信号，当P1.0="1"就一直等待直到变成“0”，又当P1.0="0"时又继续等待直到变成“1”才开始接受
10	有8个周期"1"->"0"的变化

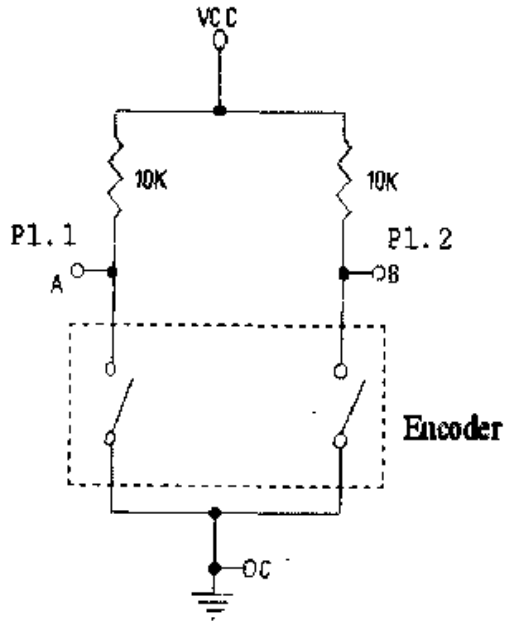
行 号	说　明
12	检查脉冲信号是“1”或“0”的例程
13,14	如果脉冲的周期变化太大，“1”电平或“0”电平的时间太长则离开返回原调用程序
16	每检测到 1 位，就将上一次检测的位向左移 1 位
17~22	如果“1”电平的数值介于 85~115 之间且“0”电平的数值介于 0~65 之间，则此脉冲格式被解成为“1”，又如果“1”电平的数值介于 0~65 之间且“0”电平的数值介于 85~115 之间，则此脉冲格式被解成为“0”
24	8 个位解出来之后即暂存至变量内，以便其它模组或例程存取

➢ EncoderProcess( )

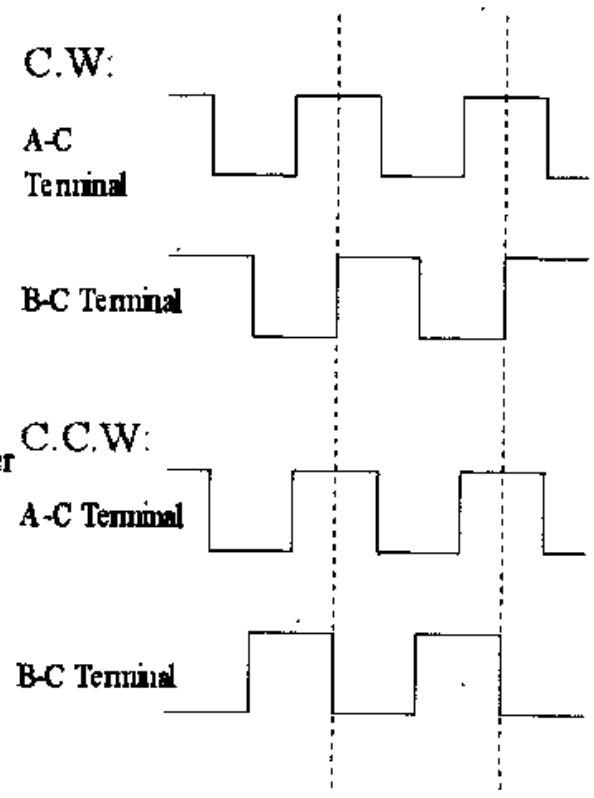
目的

解出 One-Knob 旋钮的旋转方向。

电路



图一　One-Knob 旋钮构造



图二　时序波形

程序

```
01 void EncoderProcess(void)
02 {
03    Byte  keytmp;
04
05    WheelNow= P1 & 0x06;    //P1.1, P1.2 as input i/o
06    keytmp  = WheelNow;
07    keytmp ^= WheelOld;
08    if ( keytmp==0 )   return;
09
10    if ( keytmp & 0x04 )
11    {
12      if ( WheelNow & 0x04 )
13         WheelLeft( );
14      else if ( WheelNow & 0x02 )
15         WheelLeft( );
16      else
17         WheelRight( );
18    }
19    else if ( keytmp & 0x06 )
20    {
21      if ( WheelNow==0x00 )
22         WheelLeft( );
23      else if ( WheelNow==0x02 )
24         WheelLeft( );
25      else
26           WheelRight( );
27    }
28
29    WheelOld=WheelNow;
30 }
31
32 void WheelLeft(void)
33 {
```

```
34      RightCount=0;
35      LeftCount++;
36      if ( LeftCount==2 )
37      {
38        EncoderCnt=2;
39        LeftCount =0;
40      }
41  }
42
43  void WheelRight(void)
44  {
45      LeftCount=0;
46      RightCount++;
47      if ( RightCount==2 )
48      {
49        EncoderCnt=1;
50        RightCount=0;
51      }
52  }
53
```

说明

图一是 one-knob 旋转钮的电路构造，one-knob 的中文名称为飞梭旋钮，可应用在 PC 监视器或 LCD 监视器。图二为转动飞梭旋钮的时序图，当顺时针转动及逆时针转动时，其 A、C 端和 B、C 端的波形及相位的差异如图上所示，而程序利用此特性，将上一次的状态值和目前的状态值作 XOR 运算来区分转动的方向。

行 号	说　明
05	将 A、B 两端即时的状态暂存至变量 WheelNow 内，而 A、B 两端正常的状态为“1”电平
06	将 WheelNow 存入 keytmp 变量内
07	将 keytmp 和上一次状态作 XOR 运算后再存入 keytmp
08	如果 keytmp 等于 0，即飞梭旋扭一直未作转动就返回原调用程序

行 号	说 明
10	两波形上一次的电平和这一次的电平作 XOR 运算之后，其 keytmp 等于 4，则执行括号内的语句
12~15	目前状态当 P1.2="1"且 P1.1="0"或者 P1.2="0"且 P1.1="1"，则是向左转动
16,17	否则，是向右转动
19	当 keytmp 等于 6，即作了 XOR 运算的结果数值为 6，XOR 运算的特性为相同位为“0”，不同位为“1”，例如：“0”与“0”或“1”与“1” 作 XOR 运算其结果为“0”；“0”与“1”或“1”与“0” 作 XOR 运算其结果为“1”
21~26	目前状态当 P1.2="0"且 P1.1="0"或者 P1.2="0"且 P1.1="1"，则是向左转动，否则就是向右转动了
29	将目前状态值存入 WheelOld，作为上一次的状态值
32	向左转动子程序
33~41	清除右计数变量，左计数变量的数值累加一并判断其数值等于 2，即确定向左转动便将 EncoderCnt 设定为 2，并将左计数变量归零
43	向右转动子程序
44~52	清除左计数变量，右计数变量的数值累加一并判断其数值等于 2，即确定向右转动便将 EncoderCnt 设定为 1，并将右计数变量归零

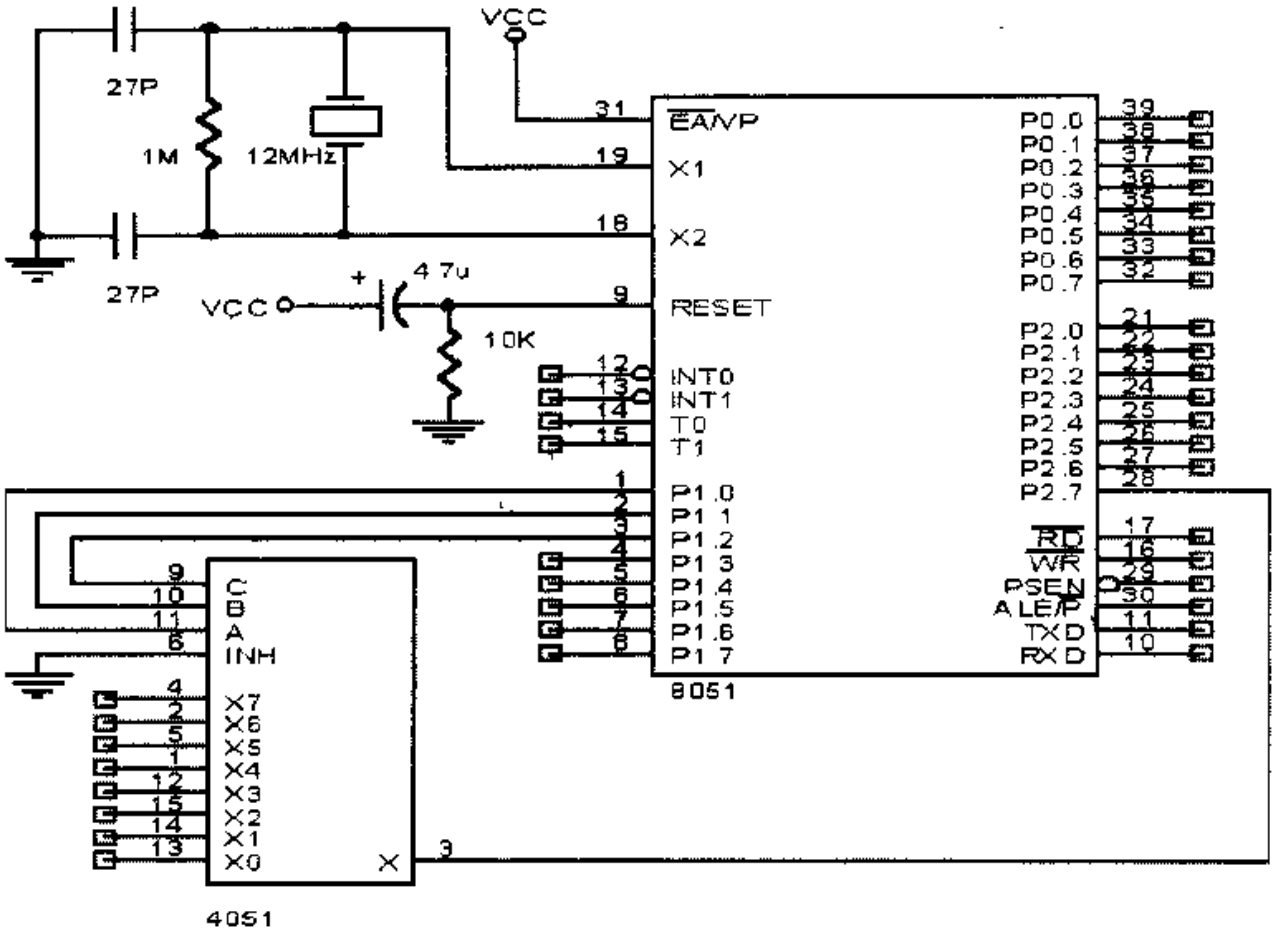
第 16 章　多任务器的应用

➢ Input4051_1(　)

目的

利用 IC4051 多任务器来扩充输入端口的应用。

电路



程序

```
01 //P1.0~P1.2 select input1~input8
02 //P2.7 is input
03 void Input4051_1(void)
04 {
05     P1_0=1;          //detect input2 (IC4051 X1,P14)
06     P1_1=0;
07     P1_2=0;
08     if ( P2_7==1 )       //"1":active
09       FgExtInput2=1;
10     else
11       FgExtInput2=0;
12 }
```

说明

当外部输入信号太多时会占用微处理器的 I/O 脚致使 I/O 脚不够用，可利用多任务器来代替 I/O 脚作为输入用，但是无法作为输出用也不能使用位寻址模式来检测输入，必须事先选择某一个输入端再检测。图中 IC4051 的 A、B、C 脚是作为选择 X0~X7 的输入端，可以想象为有 8 个输入端点 X0~X7 及一个共同端点 X 的开关，而 A、B、C 脚即是选择哪一个端点和共同端点相通，称为多任务器。

行 号	说　明
05~07	选择 X1 脚的端点作为输入端点
08~11	如果共同端点 X 即 P2.7="1"，也就是 X1 脚的信号电平为“1”则设定标记，否则，表示 X1 脚的信号电平为“0”则清除标记变数

➢ Input4051_2( )

目的

利用循环方法来检测输入端点 X0~X7

程序

```
01 //detect one of total input,priority front
02 void Input4051_2(void)
```

```
03 {
04     Byte  i;
05
06     for(i=0; i<8; i++)
07     {
08        P1=i & 0x07;
09        if ( P2_7==1 )
10        {
11           InputState=i+1;
12           break;     //exit for loop
13        }
14     }
15 }
```

说明

8 个外部输入源皆可独立互不影响，可以同时有很多输入皆为“1”电平或皆为“0”电平，或是“1”电平及“0”电平互相存在，此程序从 X0 开始检测是否为“1”电平，如果条件式成立则记录第几个输入端后并离开。此程序只是为了说明 if 与 break 连用的程序结构及思考逻辑，想要达到什么目的，而使用什么样的程序设计技巧。

行 号	说　　明
06	从 X0~X7 依次检测是否为“1”电平，共 8 个
08	多任务器有 A、B、C 作为选择用，所以和 0x07 作 AND 逻辑运算，其实不要和 0x07 做 AND 逻辑运算也可以，因为 i 会从 0 递增至 7 正好可以逐次选择 X0~X7，在程序中和 0x07 做 AND 逻辑运算代表着有更正确的观念
09~13	如果有任何一个输入端是“1”电平，则记录之并返回原调用程序，其中 break 是跳出 for 循环而不是跳出 if 语句

➢ Input4051_3(　)

目的

利用循环方法来检测输入端点 X7~X0。

程序

```
01 //detect one of total input,priority back
02 void Input4051_3(void)
03 {
04    Byte  i;
05
06    for(i=0; i<8; i++)
07    {
08       P1=i & 0x07;
09       if ( P2_7==1 )
10          InputState=i+1;
11    }
12 }
```

说明

此程序和上一个范例几乎雷同，只是少了 break 指令，如果 if 叙述的条件式成立则 for 循环也会继续作下去，因此形成以 X7~X0 的检测顺序，例如：如果 X6 和 X3 脚都有“1”电平的输入，则最后结果 InputState 将会等于 7 而不是等于 4。

行 号	说　明
06	从 X0~X7 依次检测是否为“1”电平，共 8 个
08	依次选择 X0~X7 的输入端
09,10	如果输入端为“1”电平则记录第几个输入，如果输入端为“0”电平则继续判断下一个输入，而达到以 X7~X0 为优先的顺序

➢ Input4051_4( )

目的

个别检测及记录任何一个输入端

程序

```
01 //total detect for sequent
02 void Input4051_4(void)
```

```
03 {
04     FgExtInput1=0;
05     FgExtInput2=0;
06     FgExtInput3=0;
07     FgExtInput4=0;
08     FgExtInput5=0;
09     FgExtInput6=0;
10     FgExtInput7=0;
11     FgExtInput8=0;
12
13     P1_0=0;
14     P1_1=0;
15     P1_2=0;
16     if ( P2_7==1 )
17        FgExtInput1=1;
18
19     P1_0=1;
20     P1_1=0;
21     P1_2=0;
22     if ( P2_7==1 )
23        FgExtInput2=1;
24
25     P1_0=0;
26     P1_1=1;
27     P1_2=0;
28     if ( P2_7==1 )
29        FgExtInput3=1;
30
31     P1_0=1;
32     P1_1=1;
33     P1_2=0;
34     if ( P2_7==1 )
35        FgExtInput4=1;
36
37     P1_0=0;
```

```
38     P1_1=0;
39     P1_2=1;
40     if ( P2_7==1 )
41        FgExtInput5=1;
42
43     P1_0=1;
44     P1_1=0;
45     P1_2=1;
46     if ( P2_7==1 )
47        FgExtInput6=1;
48
49     P1_0=0;
50     P1_1=1;
51     P1_2=1;
52     if ( P2_7==1 )
53        FgExtInput7=1;
54
55     P1_0=1;
56     P1_1=1;
57     P1_2=1;
58     if ( P2_7==1 )
59        FgExtInput8=1;
60 }
```

说明

IC4051 的 X0~X7 脚为独立的输入端，皆可分别检测、判断和存取用，在程序中先设定 P1.0~P1.2 即先选择哪一个脚 X0~X7 作为输入，然后再判断 P2.7 是否为“1”即输入端是否为“1”后，再记录其相对应的标记变量即可，依此方法来顺序判断 X0~X7 的输入。

行 号	说 明
04~11	清除 X0~X7 输入端的个别标记变数
13~17	检测 X0 输入端是否为“1”电平，如果是，则设定 FgExtInput1
19~23	检测 X1 输入端是否为“1”电平，如果是，则设定 FgExtInput2

行　号	说　　明
25~29	检测 X2 输入端是否为“1”电平，如果是，则设定 FgExtInput3
31~35	检测 X3 输入端是否为“1”电平，如果是，则设定 FgExtInput4
37~41	检测 X4 输入端是否为“1”电平，如果是，则设定 FgExtInput5
43~47	检测 X5 输入端是否为“1”电平，如果是，则设定 FgExtInput6
49~53	检测 X6 输入端是否为“1”电平，如果是，则设定 FgExtInput7
55~59	检测 X7 输入端是否为“1”电平，如果是，则设定 FgExtInput8

➢ Input4051_5( )

目的

各别检测及记录任何一个输入端，以 switch...case 的指令

程序

```
01 //total detect usage case loop
02 void Input4051_5(void)
03 {
04    Byte  i;
05
06    FgExtInput1=0;
07    FgExtInput2=0;
08    FgExtInput3=0;
09    FgExtInput4=0;
10    FgExtInput5=0;
11    FgExtInput6=0;
12    FgExtInput7=0;
13    FgExtInput8=0;
14
15    for(i=0; i<8; i++)
16    {
17       P1=i & 0x07;
18       if ( P2_7==1 )
```

```
19      {
20          switch( i )
21          {
22           case 0 :
23              FgExtInput1=1;
24              break;
25           case 1 :
26              FgExtInput2=1;
27              break;
28           case 2 :
29              FgExtInput3=1;
30              break;
31           case 3 :
32              FgExtInput4=1;
33              break;
34           case 4 :
35              FgExtInput5=1;
36              break;
37           case 5 :
38              FgExtInput6=1;
39              break;
40           case 6 :
41              FgExtInput7=1;
42              break;
43           case 7 :
44              FgExtInput8=1;
45              break;
46           default :
47              break;
48          }
49      }
50  }
51 }
```

说明

为了达到相同目的确有很多不同的方法和程序编写技巧，思考到底哪一种方法是既简洁又容易懂，程序维护简单又结构性严谨，也是软件设计人员要学习的方向与目标，此程序使用 switch...case 架构的方式，各位读者你们认为适合吗?

行 号	说　明
06~13	初始每个输入端的标记变数
15	从 X0~X7 作 8 次
17	由 P1 端口输出来选择 X0~X7
18	如果输入端为“1”电平，则才开始判断到底是由哪一个输入
20~45	依据 i 变数值来决定执行哪一个 CASE 底下的叙述，当 i=0 则执行 case 0 底下的叙述，当 I=1 则执行 case 1 底下的叙述并依此类推，设定相对标记并立即跳出 switch 循环
46,47	switch 循环的条件和 case 的条件值皆不符合，则执行 default 下的语句，此处为 break 指令即立即跳出 switch 循环

➢ Input4051_6( )

目的

各别检测及记录任何一个输入端，以 if...else 的指令

程序

```
01 //total detect usage if
02 void Input4051_6(void)
03 {
04     Byte  i;
05
06     FgExtInput1=0;
07     FgExtInput2=0;
08     FgExtInput3=0;
09     FgExtInput4=0;
10     FgExtInput5=0;
11     FgExtInput6=0;
```

```
12    FgExtInput7=0;
13    FgExtInput8=0;
14
15    for(i=0; i<8; i++)
16    {
17      P1=i & 0x07;
18      if ( P2_7==1 )
19      {
20        if (i==0)      FgExtInput1-1;
21        else  if (i==1)  FgExtInput2=1;
22        else  if (i==2)  FgExtInput3=1;
23        else  if (i==3)  FgExtInput4=1;
24        else  if (i==4)  FgExtInput5=1;
25        else  if (i==5)  FgExtInput6=1;
26        else  if (i==6)  FgExtInput7=1;
27        else  if (i==7)  FgExtInput8=1;
28      }
29    }
30 }
```

说明

这种 if...else 程序方式似乎比上一个范例 switch...case 的方法简洁有力，而且也比较容易体会程序的目的与功能。

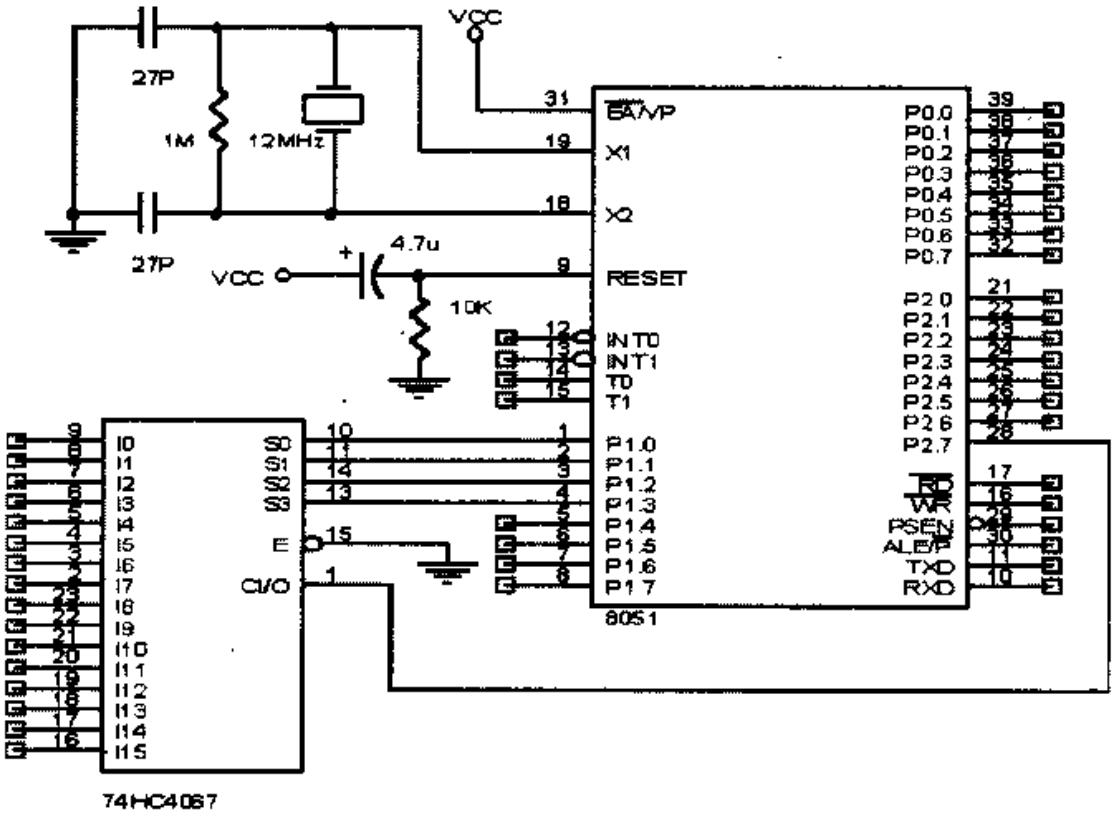
行 号	说　明
06~13	初始每个输入端的标记变数
15	从 X0~X7 作 8 次
17	由 P1 端口输出来选择 X0~X7
18	即输入端为“1”电平
19~28	开始判断 i 变数值，以决定相对应的标记变数

➢ Input4067_1(　)

目的

扩充外部为 16 个输入端 —— 顺序检测

电路



程序

```
01 //P1.0 ~ P1.3 select input1 ~ input16
02 //P2.7 is input
03 void Input4067_1(void)
04 {
05     FgExtInput1 =0;
06     FgExtInput2 =0;
07     FgExtInput3 =0;
08     FgExtInput4 =0;
09     FgExtInput5 =0;
```

```
10    FgExtInput6 =0;
11    FgExtInput7 =0;
12    FgExtInput8 =0;
13    FgExtInput9 =0;
14    FgExtInput10=0;
15    FgExtInput11=0;
16    FgExtInput12=0;
17    FgExtInput13=0;
18    FgExtInput14=0;
19    FgExtInput15=0;
20    FgExtInput16=0;
21
22    P1_0=0;
23    P1_1=0;
24    P1_2=0;
25    P1_3=0;
26    if ( P2_7==1 )  FgExtInput1=1;
27
28    P1_0=1;
29    P1_1=0;
30    P1_2=0;
31    P1_3=0;
32    if ( P2_7==1 )  FgExtInput2=1;
33
34    P1_0=0;
35    P1_1=1;
36    P1_2=0;
37    P1_3=0;
38    if ( P2_7==1 )  FgExtInput3=1;
39
40    P1_0=1;
41    P1_1=1;
42    P1_2=0;
43    P1_3=0;
44    if ( P2_7==1 )  FgExtInput4=1;
```

```
45
46      P1_0=0;
47      P1_1=0;
48      P1_2=1;
49      P1_3=0;
50      if ( P2_7==1 )  FgExtInput5=1;
51
52      P1_0=1;
53      P1_1=0;
54      P1_2=1;
55      P1_3=0;
56      if ( P2_7==1 )  FgExtInput6=1;
57
58      P1_0=0;
59      P1_1=1;
60      P1_2=1;
61      P1_3=0;
62      if ( P2_7==1 )  FgExtInput7=1;
63
64      P1_0=1;
65      P1_1=1;
66      P1_2=1;
67      P1_3=0;
68      if ( P2_7==1 )  FgExtInput8=1;
69
70      P1_0=0;
71      P1_1=0;
72      P1_2=0;
73      P1_3=1;
74      if ( P2_7==1 )  FgExtInput9=1;
75
76      P1_0=1;
77      P1_1=0;
78      P1_2=0;
79      P1_3=1;
```

```
80    if ( P2_7==1 )  FgExtInput10=1;
81
82    P1_0=0;
83    P1_1=1;
84    P1_2=0;
85    P1_3=1;
86    if ( P2_7==1 )  FgExtInput11=1;
87
88    P1_0=1;
89    P1_1=1;
90    P1_2=0;
91    P1_3=1;
92    if ( P2_7==1 )  FgExtInput12=1;
93
94    P1_0=0;
95    P1_1=0;
96    P1_2=1;
97    P1_3=1;
98    if ( P2_7==1 )  FgExtInput13=1;
99
100    P1_0=1;
101    P1_1=0;
102    P1_2=1;
103    P1_3=1;
104    if ( P2_7==1 )  FgExtInput14=1;
105
106    P1_0=0;
107    P1_1=1;
108    P1_2=1;
109    P1_3=1;
110    if ( P2_7==1 )  FgExtInput15=1;
111
112    P1_0=1;
113    P1_1=1;
114    P1_2=1;
```

```
115     P1_3=1;
116     if ( P2_7==1 )  FgExtInput16=1;
117
118     P1_0=0;         //reset P1
119     P1_1=0;
120     P1_2=0;
121     P1_3=0;
122 }
```

说明

当 IC4051 的 8 个输入端还不够使用时，例如：要检测 15Pin 或 25Pin 接点的接地电容时，就需要使用具有更多输入接点的组件了，如 IC74HC4067 就是一个 16 个输入端多任务器，其程序的检测方法类似于前面的范例，在此不再赘述。

➢ Input4067_2( )

目的

扩充外部为 16 个输入端 —— 循环式检测

程序

```
01 void Input4067_2(void)
02 {
03     Byte  i;
04
05     FgExtInput1 =0;
06     FgExtInput2 =0;
07     FgExtInput3 =0;
08     FgExtInput4 =0;
09     FgExtInput5 =0;
10     FgExtInput6 =0;
11     FgExtInput7 =0;
12     FgExtInput8 =0;
13     FgExtInput9 =0;
14     FgExtInput10=0;
```

```
15      FgExtInput11=0;
16      FgExtInput12=0;
17      FgExtInput13=0;
18      FgExtInput14=0;
19      FgExtInput15=0;
20      FgExtInput16=0;
21
22      for(i=0; i<16; i++)
23      {
24        P1=i & 0x0f;
25        if ( P2_7==1 )
26        {
27          if (i==0)      FgExtInput1 =1;
28          else  if (i==1 )   FgExtInput2 =1;
29          else  if (i==2 )   FgExtInput3 =1;
30          else  if (i==3 )   FgExtInput4 =1;
31          else  if (i==4 )   FgExtInput5 =1;
32          else  if (i==5 )   FgExtInput6 =1;
33          else  if (i==6 )   FgExtInput7 =1;
34          else  if (i==7 )   FgExtInput8 =1;
35          else  if (i==8 )   FgExtInput9 =1;
36          else  if (i==9 )   FgExtInput10=1;
37          else  if (i==10)   FgExtInput11=1;
38          else  if (i==11)   FgExtInput12=1;
39          else  if (i==12)   FgExtInput13=1;
40          else  if (i==13)   FgExtInput14=1;
41          else  if (i==14)   FgExtInput15=1;
42          else  if (i==15)   FgExtInput16=1;
43        }
44      }
45  }
```

说明

上一个范例具有 16 个输入而顺序个别作检测致使程序写得很冗长，不但不容易维护且又占用很多的程序内存空间，而本范例使用 for 循环则大大改善了程

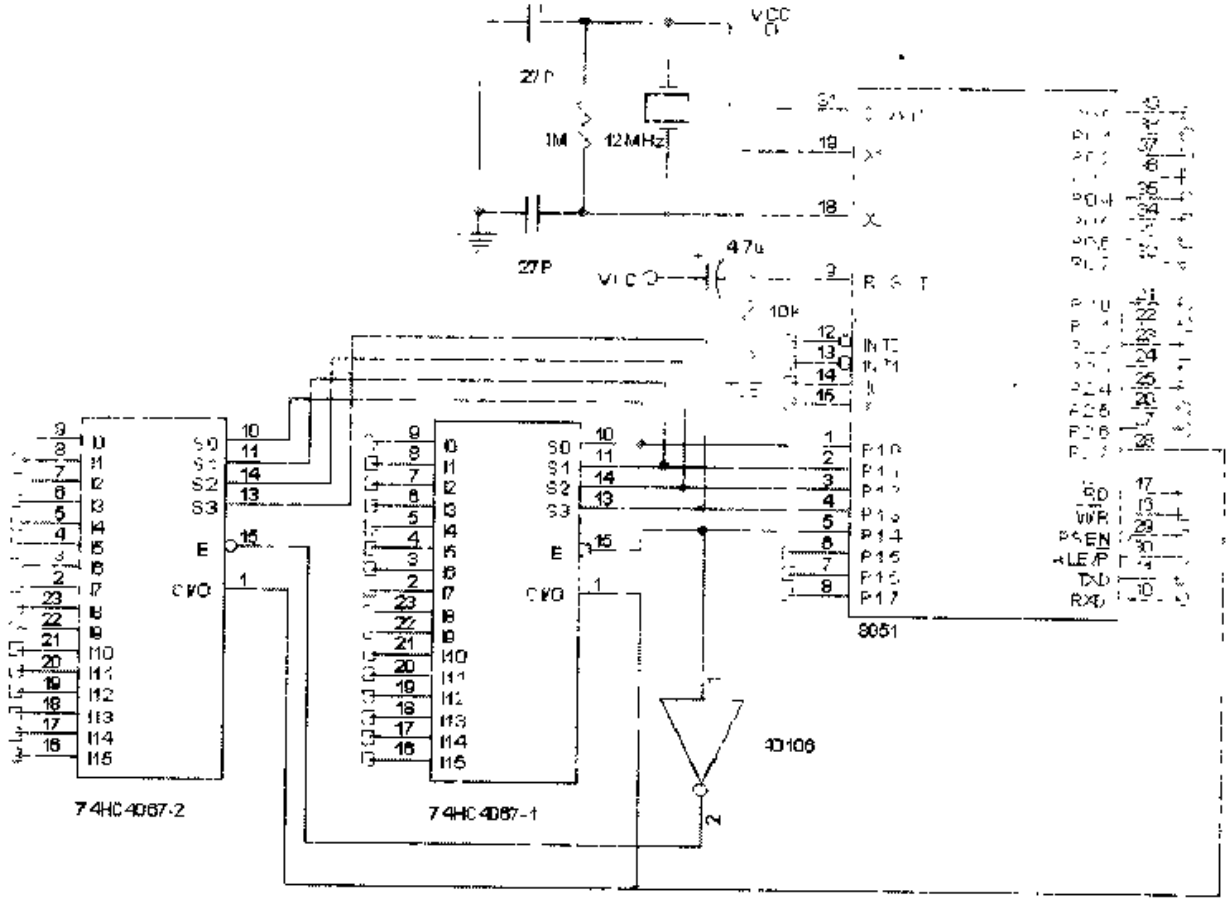
序的易读性与结构性，因此，编写程序时尽量使用循环，不管是几层都可使程序简化而变得简单明了又易维护。

➢ Input4067_3( )

目的

扩充外部为 32 个输入端 —— 循环式检测

电路



程序

```
01 void Input4067_3(void)
02 {
03     Byte  i;
04
05     for(i=0; i<32; i++)
06     {
```

```
07        P1=i & 0x1f;
08
09        TrmBuf[i]=0;
10        if ( P2_7==1 )
11           TrmBuf[i]=1;
12     }
13 }
```

说明

如果某些系统需要使用到更多的输入时，则需扩充至 2 个 74HC4067，而以一支 I/O 来当作选择线，当 P1.4="0"则 74HC4067-1 动作，而当 P1.4="1"经过反相器变为“0”则 74HC4067-2 动作，程序中使用数组方式来记录外部的输入状态，当外部的输入为“1”电平则数组内容为 0x01，当外部的输入为“0”电平则数组内容为 0x00，如果程序使用如上一个范例 if...else 的方法也会变得很冗长，在其它模块或例程也都必须一一的去检测相对应的标记，也会变得很冗长且无效率，使用了数组的方法程序是不是显得很简化、很舒服呢？

行 号	说　明
05	32 个外部输入，所以循环作 32 次
07	从第一个 74HC4067 开始选择输入端，直到第 32 个为止
09	相对应的阵列内容先清除为 0x00
10,11	如果输入端为“1”电平则设定相对应的阵列内容为 0x01

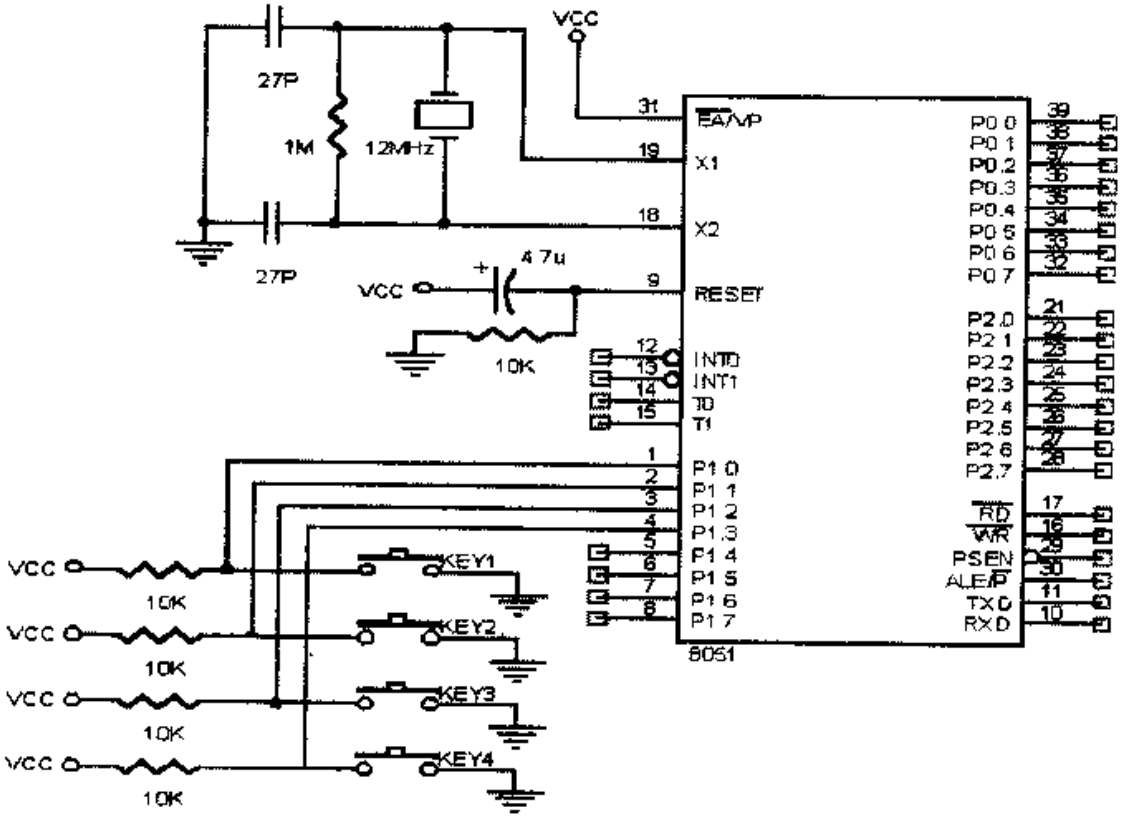
第 17 章　键盘操作的应用

➢ InputKey1(　)

目的

按键输入的检测 —— 单键的应用

电路



程序

```
01 //one key detect
02 void InputKey1(void)
03 {
```

```
04     Byte  keytmp;
05
06     keytmp = ~(P1) & 0x0f;  //"0":active
07     if ( keytmp==1)     KeyData = KEY1;
08     else if ( keytmp==2 )   KeyData = KEY2;
09     else if ( keytmp==4 )   KeyData = KEY3;
10     else if ( keytmp==8 )   KeyData = KEY4;
11 }
```

说明

检测按键的动作将会在系统中时常被应用到，按键如同是在下命令一样，按了不同的键就做不同的动作，完全视系统的功能及所定义的用途而做不同的响应，最简单的就是以个别的 I/O 来检测个别的开关是否按下，如上图所示，但同时间只能按下一个键，如同时按下复合键或很多键时，则会以 KEY1、KEY2、KEY3、KEY4 作为检测的顺序。

行 号	说　明
06	有 4 个按键分别由 P1.0~P1.3 来检测，当开关被按下时为“0”电平，因此要取反相和 0x0f 作 AND 运算
07	如果 P1.0="0"即 keytmp=1，则表示 KEY1 被按下并存入至变量内
08	如果 P1.1="0"即 keytmp=2，则表示 KEY2 被按下并存入至变量内
09	如果 P1.2="0"即 keytmp=4，则表示 KEY3 被按下并存入至变量内
10	如果 P1.3="0"即 keytmp=8，则表示 KEY4 被按下并存入至变量内

➢ InputKey2( )

目的

按键输入的检测 —— 复合键的应用(二)

程序

```
01 //double key detect
02 void InputKey2(void)
03 {
04     Byte  keytmp,keytmp1;
```

```
05
06      KeyData1 = 0;
07      KeyData2 = 0;
08      KeyData3 = 0;
09      KeyData4 = 0;
10      keytmp = ~(P1) & 0x0f;
11
12      keytmp1 = keytmp & 0x01;
13      if ( keytmp1==1)    KeyData1 = KEY1;
14
15      keytmp1 = keytmp & 0x02;
16      if ( keytmp1==2 )   KeyData2 = KEY2;
17
18      keytmp1 = keytmp & 0x04;
19      if ( keytmp1==4 )   KeyData3 = KEY3;
20
21      keytmp1 = keytmp & 0x08;
22      if ( keytmp1==8 )   KeyData4 = KEY4;
23
24      KeyData = KeyData1 + KeyData2 + KeyData3 + KeyData4;
25  }
```

说明

此范例程序的技巧可以应用于两个以上键同时按下，主要是以不同按键有不同的变量来记录按键的数值，最后再相加起来就可以判断出哪些键被按下了，即使 4 个键都被按下也可以检测出来，要注意的是当同时用手按复合键难免会有时间差，因此在原调用程序也需要作大约 20mS~50mS 的延迟，调用本例程后才能正确的检测出来。

行 号	说　　明
06~09	初始每个按键输入值
10	取得每个按键的数值，取反和和保留低 4 位及屏蔽高 4 位
12,13	如果 P1.0="0"即 keytmp1=1，则表示 KEY1 被按下并存入至 KeyData1
15,16	如果 P1.1="0"即 keytmp1=2，则表示 KEY2 被按下并存入至 KeyData2

行 号	说 明
18,19	如果 P1.2="0"即 keytmp1=4，则表示 KEY3 被按下并存入至 KeyData3
21,22	如果 P1.3="0"即 keytmp1=8，则表示 KEY4 被按下并存入至 KeyData4
24	将各别的输入值相加起来则可以检测出只有单键输入时，而要检测复合键则须判断个别的变量才可以；或是其所定义的按键数值为 KEY1=0x01、KEY2=0x02、KEY3=0x04、KEY4=0x08，如此的定义其相加起来的数值皆不会一样，因此无须分别判断各自输入的变量，才可以应用复合键的操作。

➢ InputKey3( )

目的

按键输入的检测 —— 复合键的应用(二)

程序

```
01 //modify follow coding
02 void InputKey3(void)
03 {
04     Byte  keytmp;
05
06     KeyData1 = 0;
07     KeyData2 = 0;
08     KeyData3 = 0;
09     KeyData4 = 0;
10     keytmp = ~(P1) & 0x0f;
11
12     if ( keytmp & 0x01 )    KeyData1 = KEY1;
13     if ( keytmp & 0x02 )    KeyData2 = KEY2;
14     if ( keytmp & 0x04 )    KeyData3 = KEY3;
15     if ( keytmp & 0x08 )    KeyData4 = KEY4;
16
17     KeyData = KeyData1 + KeyData2 + KeyData3 + KeyData4;
18 }
```

说明

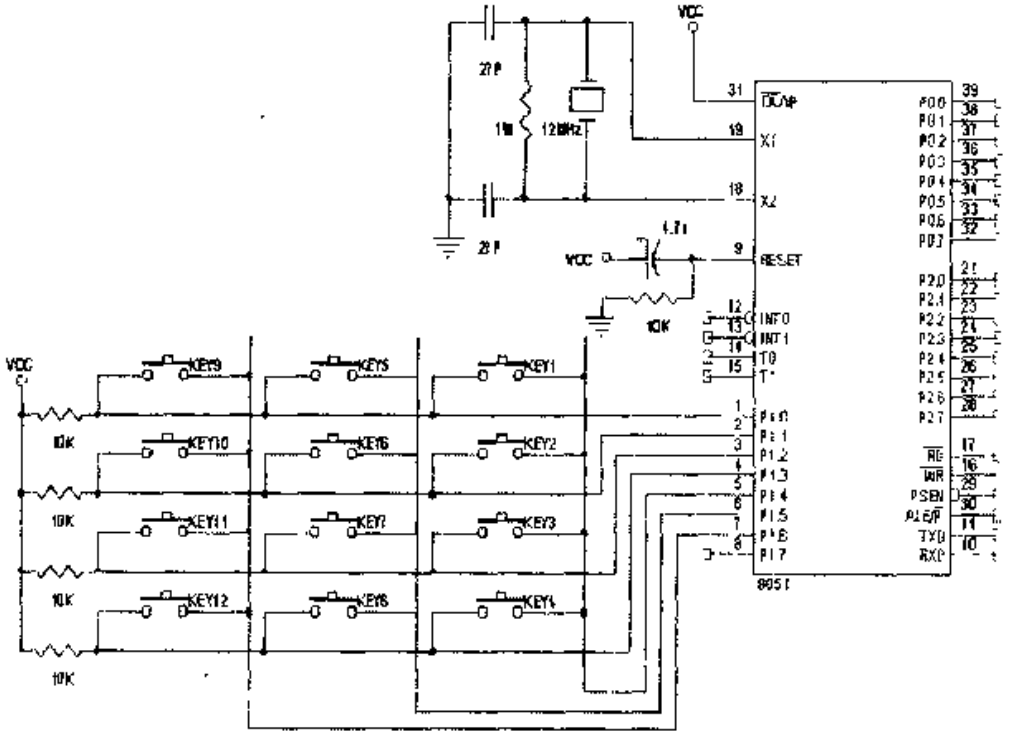
此程序和上一个范例的功能一样，只是少了一个 LOCAL 变量 keytmp1 并将上一个范例两列的判断式改写成一列，这样阅读程序比较容易懂又节省程序内存空间，请读者尽量以这种写法也尽量养成这样的思维。

➢ ScanKey1( )

目的

矩阵式键盘的扫描 —— 单键输入的应用

电路



程序

```
01 //one key detect
02 void ScanKey1(void)
03 {
04     Byte  keytmp;
05
06     COLUMN1_PIN = 0;     //P1.4,scan line1 "0":active
```

```
07    COLUMN2_PIN = 1;
08    COLUMN3_PIN = 1;
09    keytmp = ~(P1) & 0x0f;  //P1.0~P1.3
10    switch( keytmp )
11    {
12     case 1 :
13        KeyData = KEY1;
14        break;
15     case 2 :
16        KeyData = KEY2;
17        break;
18     case 4 :
19        KeyData = KEY3;
20        break;
21     case 8 :
22        KeyData = KEY4;
23        break;
24     default :
25        break;
26    }
27
28    COLUMN1_PIN = 1;
29    COLUMN2_PIN = 0;    //P1.5,scan line2 "0":active
30    COLUMN3_PIN = 1;
31    keytmp = ~(P1) & 0x0f;
32    switch( keytmp )
33    {
34     case 1 :
35        KeyData = KEY5;
36        break;
37     case 2 :
38        KeyData = KEY6;
39        break;
40     case 4 :
41        KeyData = KEY7;
```

```
42          break;
43       case 8 :
44          KeyData = KEY8;
45          break;
46       default :
47          break;
48      }
49
50      COLUMN1_PIN = 1;
51      COLUMN2_PIN = 1;
52      COLUMN3_PIN = 0;    //P1.6,scan line3 "0":active
53      keytmp = ~(P1) & 0x0f;
54      switch( keytmp )
55      {
56       case 1 :
57          KeyData = KEY9;
58          break;
59       case 2 :
60          KeyData = KEY10;
61          break;
62       case 4 :
63          KeyData = KEY11;
64          break;
65       case 8 :
66          KeyData = KEY12;
67          break;
68       default :
69          break;
70      }
71      COLUMN3_PIN = 1;    //scan line3 not active
72 }
```

说明

以一个按键就对应一个 I/O 脚的方法，其程序的写法就变得单纯得多，但如果有 12 个按键甚至更多，则微处理器的 I/O 势必会不够使用。因此，改成矩阵

式扫描的方式可以用较少的 I/O 而检测出更多的按键，例如：4 列×4 行的矩阵共享了 8 个 I/O，但可以检测出 4×4=16 个按键，如果使用一个按键就对应一个 I/O 的方式，16 个按键则需要 16 个 I/O，因此也就节省了 8 个 I/O 了；又如果需要检测 32 个按键则需要 8 列×4 行的矩阵，只用了 8+4=12 个 I/O，而使用对应方式需用 32 个 I/O 则节省了 20 个 I/O，由此可见，矩阵式键盘扫描是应用在需要很多个按键的系统上，而且时常被应用。

本范例的动作原理如下：

行扫描线为“0”电平动作，因此将第一行扫描线先设定为“0”电平，其余设为“1”电平，即可检测第一行的按键，只要按下键则相对应的“列”数据将会变成“0”电平，例如：P1.4="0"即检测第一行，当 KEY2 被按下则 P1.1="0"，其余 P1.0、P1.2、P1.3、都是“1”电平，既使按下 KEY5~KEY12 则 P1.0~P1.3 也不会变成“0”电平，因为相对应的扫描线没有动作即 P1.5 及 P1.6 都还是“1”电平，依此类推，再设定第二行扫描线 P1.5="0" 、第三行扫描线 P1.6="0"，并分别取得数据就可以检测出所有的按键了，如果都未按下任意键则按键变量值将为 0。

行 号	说 明
06~08	第一行扫描线先设定为“0”电平，即使能动作
09	取得数据，因为按键是“0”动作所以取反
10~26	依据取得的数据来判断哪一个键被按下并存入按键值
28~48	检测第二行上的按键并取得按键数据，如果有按下则存入按键值，如果都未按下则继续检测第三行
50~70	设定第三行扫描线动作即 P1.6="0"电平，依据按键值 keytmp 来执行 case 下的动作而取得按键数据
71	所有按键都检测完毕，则将扫描线除能即设定为“1”，如此的话，即使按键输入也不会动作

➢ ScanKey2( )

目的

矩阵式键盘的扫描 —— 复合键输入的应用

程序

```
01 void ScanKey2(void)
02 {
```

```
03    Byte  keytmp;
04
05    KeyData1 = 0;
06    COLUMN1_PIN = 0;
07    COLUMN2_PIN = 1;
08    COLUMN3_PIN = 1;
09    keytmp = ~(P1) & 0x0f;
10    switch( keytmp )
11    {
12     case 1 :
13        KeyData1 = KEY1;
14        break;
15     case 2 :
16        KeyData1 = KEY2;
17        break;
18     case 4 :
19        KeyData1 = KEY3;
20        break;
21     case 8 :
22        KeyData1 = KEY4;
23        break;
24     default :
25        break;
26    }
27
28    KeyData2 = 0;
29    COLUMN1_PIN = 1;
30    COLUMN2_PIN = 0;
31    COLUMN3_PIN = 1;
32    keytmp = ~(P1) & 0x0f;
33    switch( keytmp )
34    {
35     case 1 :
36        KeyData2 = KEY5;
37        break;
```

```
38      case 2 :
39         KeyData2 = KEY6;
40         break;
41      case 4 :
42         KeyData2 = KEY7;
43         break;
44      case 8 :
45         KeyData2 = KEY8;
46         break;
47      default :
48         break;
49    }
50
51    KeyData3 = 0;
52    COLUMN1_PIN = 1;
53    COLUMN2_PIN = 1;
54    COLUMN3_PIN = 0;
55    keytmp = ~(P1) & 0x0f;
56    switch( keytmp )
57    {
58     case 1 :
59        KeyData3 = KEY9;
60        break;
61     case 2 :
62        KeyData3 = KEY10;
63        break;
64     case 4 :
65        KeyData3 = KEY11;
66        break;
67     case 8 :
68        KeyData3 = KEY12;
69        break;
70    default :
71        break;
72    }
```

```
73      COLUMN3_PIN = 1;
74      KeyData = KeyData1 - KeyData2 + KeyData3;
75  }
```

说明

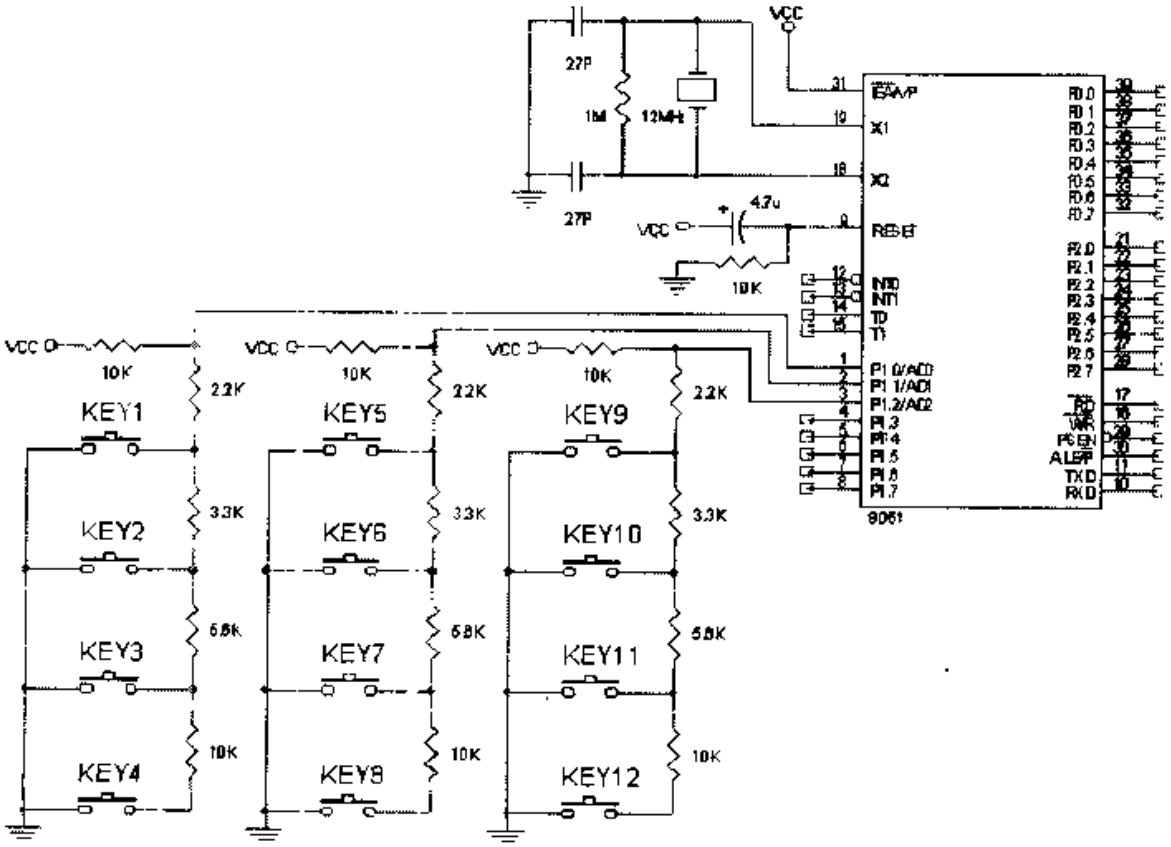
此程序可应用于复合键的操作，除了定义单键的功能外，也可以定义当二个键、三个键可以同时都按下而做不同的动作，程序中最主要是利用检测每一行的按键变量都分别独立再分别判断，由于 case 条件值的关系，此程序同一行只能按下一个键，而不同一行的按键才可以接受复合键的操作。

➢ GetKey1( )

目的

键盘的扫描 —— 利用 AD 转换器来检测单键输入的应用。

电路



程序

```
01 void GetKey1(void)
02 {
03    Byte  keytmp;
04
05    KeyData  = 0;
06    XFR_ADC=0x81;           //adc channel=1
07    DelayX1ms(2);
08    keytmp = XFR_ADC & 0x3f;    //6 bit ADC,2^6=0x3f
09    if ( keytmp < (12+4) )      KeyData = KEY1;
10    else if ( keytmp < (23+4) ) KeyData = KEY2;
11    else if ( keytmp < (34+4) )     KeyData = KEY3;
12    else if ( keytmp < (44+4) )     KeyData = KEY4;
13
14    XFR_ADC=0x82;           //adc channel=2
15    DelayX1ms(2);
16    keytmp = XFR_ADC & 0x3f;
17    if ( keytmp < (12+4) )      KeyData = KEY5;
18    else if ( keytmp < (23+4) )     KeyData = KEY6;
19    else if ( keytmp < (34+4) )     KeyData = KEY7;
20    else if ( keytmp < (44+4) )     KeyData = KEY8;
21
22    XFR_ADC=0x84;           //adc channel=3
23    DelayX1ms(2);
24    keytmp = XFR_ADC & 0x3f;
25    if ( keytmp < (12+4) )      KeyData = KEY9;
26    else if ( keytmp < (23+4) )     KeyData = KEY10;
27    else if ( keytmp < (34+4) )     KeyData = KEY11;
28    else if ( keytmp < (44+4) )     KeyData = KEY12;
29
30    XFR_ADC=0x00;           //disable adc
31 }
```

说明

如果有 16 个按键则使用矩阵式扫描方式也需要 4 列×4 行，共需要 8 个 I/O 感觉也占用很多个 I/O，如果必须限制按键电路板的尺寸不能太大而且又是单面板时，则布线(Layout)就显得比较麻烦难度也比较高，因此为了这种需求，也可以将电路设计修改成使用模拟至数字转换器(ADC)的方式，即利用按下不同键会有不同的直流电压值，再将电压转换成数值而判断其数值的不同就知道哪一个按键被按下了，一般 ADC 有分 4 位 0~15 的数值变化即 16 级、6 位 0~63 级的数值变化即 64 级及 8 位 0~255 的数值变化即 256 级阶，本范例程序是使用 6 位的 ADC，而 12 个按键使用了 3 个 ADC，每一个 ADC 负责 4 个按键的检测，其实也可以使用一个 ADC 负责 8 个按键的检测，只要每一个按键的电压值其微处理器内置的 ADC 可以检测出来即可，但尽可能的将每一个按键之间的电压值统一。

行　号	说　　明
05	设定按键数值为 0
06,07	选择 ADC 的频道 0 即 AD0，并使能 ADC 开始转换，延迟 2mS 以确保转换的时间充裕，否则如果时间太短则转换出来的数值会错误
08	将输入按键的直流电压转换成数值，因为是 6 位 ADC 所以要和 0x3f 作 AND 运算，屏蔽第 6、7 位，并存入 keytmp 变量内
09~12	判断 KEY1、KEY2、KEY3、KEY4 是否被按下，如是，则存入按键数值
14~16	选择 AD1 并开始转换，以 keytmp 来储存转换后的数值
17~20	判断 KEY5、KEY6、KEY7、KEY8 是否被按下
22~24	选择 AD2 并开始转换，以 keytmp 来储存转换后的数值
25~28	判断 KEY9、KEY10、KEY11、KEY12 是否被按下
30	除能 ADC 即不转换了

➢ GetKey2( )

目的

键盘的扫描 —— 利用 AD 转换器来检测复合键输入的应用

程序

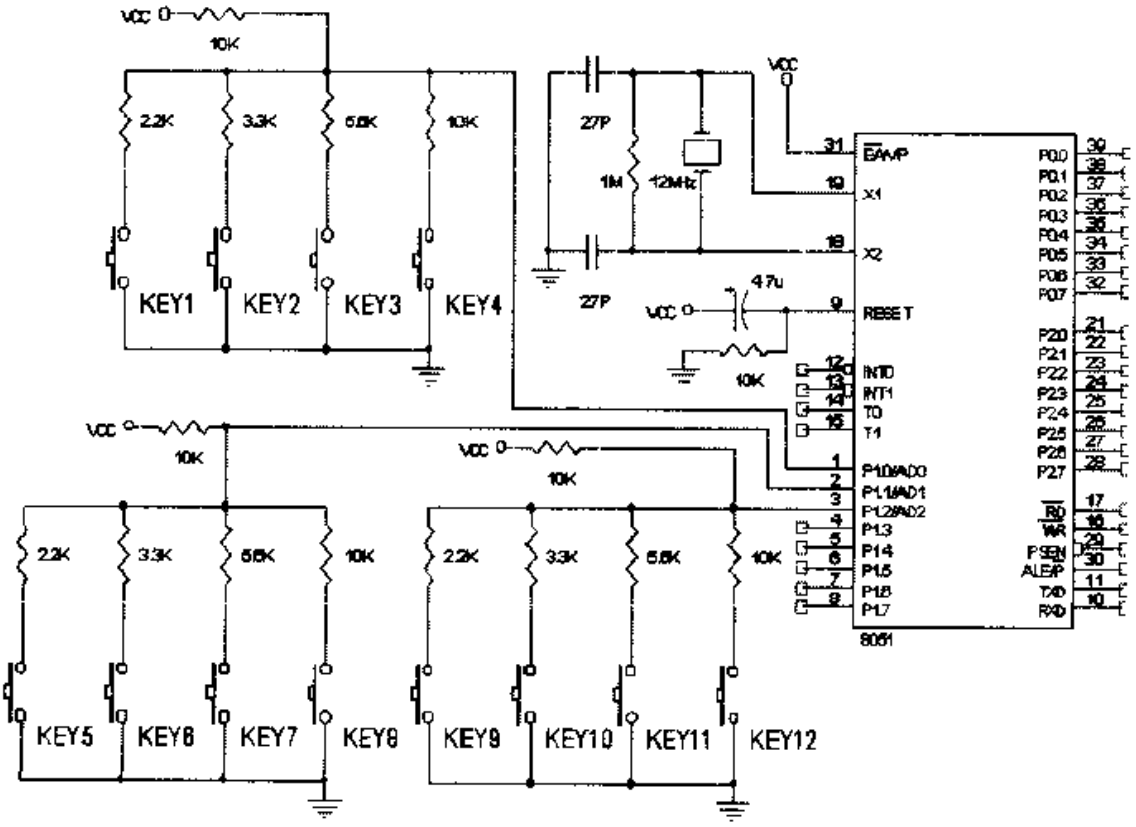
```
01 void GetKey2(void)
02 {
03    Byte  keytmp;
04
05    XFR_ADC=0x81;           //adc channel=1
06    DelayX1ms(2);           //wait AD convert complete
07    KeyData1 = 0;
08    keytmp = XFR_ADC & 0x3f;    //6 bit ADC,2^6=0x3f
09    if ( keytmp < (12+4) )      KeyData1 = KEY1;
10    else if ( keytmp < (23+4) )     KeyData1 = KEY2;
11    else if ( keytmp < (34+4) )     KeyData1 = KEY3;
12    else if ( keytmp < (44+4) )     KeyData1 = KEY4;
13
14    XFR_ADC=0x82;           //adc channel=2
15    DelayX1ms(2);
16    KeyData2 = 0;
17    keytmp = XFR_ADC & 0x3f;
18    if ( keytmp < (12+4) )      KeyData2 = KEY5;
19    else if ( keytmp < (23+4) )     KeyData2 = KEY6;
20    else if ( keytmp < (34+4) )     KeyData2 = KEY7;
21    else if ( keytmp < (44+4) )     KeyData2 = KEY8;
22
23    XFR_ADC=0x84;           //adc channel=3
24    DelayX1ms(2);
25    KeyData3 = 0;
26    keytmp = XFR_ADC & 0x3f;
27    if ( keytmp < (12+4) )      KeyData3 = KEY9;
28    else if ( keytmp < (23+4) )     KeyData3 = KEY10;
29    else if ( keytmp < (34+4) )     KeyData3 = KEY11;
30    else if ( keytmp < (44+4) )     KeyData3 = KEY12;
31
32    KeyData = KeyData1 + KeyData2 + KeyData3;
33    XFR_ADC=0x00;           //disable adc
```

```
34 }
```

说明

此程序是上一个范例应用的延伸，上一个范例同时间只能检测一个按键，如果二键以上同时被按下则只会检测出后面的按键，例如：KEY3 和 KEY11 同时都按下则只会检测出 KEY11，而本范例就能处理二键以上同时都被按下的程序。设计程序当然是要功能强、弹性大，日后修改程序也会变得较容易，不至于一开始没有考虑清楚想到什么就写什么，这样，一旦要增加新功能则原来的程序架构却无法达到要求，而必须重新设计而且浪费时间，程序内容的说明和上一个范例大同小异不再赘述，只要能掌握到原理则程序的写法可自由发挥。另外此程序是将每个 ADC 转换完成的数值，分别作判断以识别按键并将按键数值也分别储存，达到检测复合键的功能，但在同一个输入 ADC 的按键上也只能有一个键输入，即复合键的操作不可以在同一个 ADC 的按键上，如果要在同一个 ADC 的按键上可以同时作复合键输入，则电路图须改成并联电阻的连接方式，但要注意复合键输入所得到的并联电阻值不能和单键输入的电阻值太接近，以避免 ADC 所转换出的数值变化量太小，如下的电路图所示。

电路



➢ KeyCheck( )

目的

同时被按下指定键的应用

程序

```
01 //key detect,application
02 void InputKeyCheck(void)
03 {
04    InputKey2( );
05
06    FgDoubleKey = 0;
07    KeyData = NO_KEY;        //NO_KEY define=0
08
09    if ( (KeyData1==KEY1)&&(KeyData3==KEY3) )
10    {
11       FgDoubleKey = 1;
12       KeyData=DOUBLE_KEY;
13    }
14 }
15
16 void ScanKeyCheck(void)
17 {
18    ScanKey2( );
19
20    FgDoubleKey = 0;
21    KeyData = NO_KEY;        //NO_KEY define=0
22
23    if ( (KeyData1==KEY1)&&(KeyData3==KEY10) )
24    {
25       FgDoubleKey = 1;
26       KeyData=DOUBLE_KEY;
27    }
```

```
28 }
29
30 void GetKeyCheck(void)
31 {
32     GetKey2( );
33
34     FgDoubleKey = 0;
35     KeyData = NO_KEY;        //NO_KEY define=0
36
37     if ( (KeyData1==KEY1)&&(KeyData3==KEY10) )
38     {
39       FgDoubleKey = 1;
40       KeyData=DOUBLE_KEY;
41     }
42 }
```

说明

检测指定键是否按下只是按键应用中最基本的函数，本程序是示范当同时被按下指定键时，检测及动作的方法，分别以一个按键对应一个 I/O 的方式、键盘扫描的方式、利用 ADC 的方式。虽说是同时按下键但仍然有数 mS 的时间差，因此检测按键的程序或原调用程序，也要有处理按键时间差的程序架构，最简单为延迟数 mS 再检测即可，本章节 KeyProcess( )也有处理复合键的实用方法。

行 号	说　明
04	一个按键对应一个 I/O 方式的复合键检测函数
06,07	初始复合键标志及最后按键值
09~13	如果 KEY1 和 KEY3 同时被按下，就设定复合键标志，也将复合键的数值存入 KeyData 变量内
18	键盘扫描方式的复合键检测函数
23~27	如果 KEY1 和 KEY10 同时被按下，就设定复合键标志，也将复合键的数值存入 KeyData 变量内
32	利用 ADC 方式的复合键检测函数
37~41	也是检测当 KEY1 和 KEY10 同时都被按下时的动作

➢ KeyCountCheck( )

目的

按键次数的应用

程序

```
01 //press 20 times,then beep,beep
02 void KeyCountCheck1(void)
03 {
04    InputKey1( );
05    if ( KeyData==KEY1 )
06
07       KeyCount++;
08       if ( KeyCount<20 )
09       {
10          LedFlash5(1,50,50);   //flash 1 time,led on 500ms
11          Beep4(1,17,10);        //beep
12       }
13       else
14       {
15          Beep4(2,17,10);       //beep 2 times
16          KeyCount = 0;
17       }
18    }
19 }
20 void KeyCountCheck2(void)
21 {
22    InputKey1( );
23    if ( KeyData==KEY1 )
24
25       KeyCount++;
26       if ( KeyCount<20 )
27          BeepLed(1,17,10);     //simultaneously beep and
```

```
                    led on
28        else
29        {
30           Beep4(2,17,10);          //beep 2 times
31           KeyCount = 0;
32        }
33     }
34 }
```

说明

当同一个键按下的次数到了指定的数值时，例如：最大值、最小值或边限值，则作提示也是时常被广泛应用的，而简单的提示如蜂鸣器响叫、LED 闪烁等，或其它依据系统而作的处理。

行 号	说　　明
04	单键检测的程序
05	如果是按了 KEY1 的话
07	计数值加 1
08~17	每按一次按键则 LED 就闪烁一次后，蜂鸣器叫一声，如果按了 20 次表示到了最大值，则蜂鸣器响叫二声作为提示并清除计数值
27	此函数用于改善行号 10 和行号 11 顺序动作的缺点，将 LED 闪烁和蜂鸣器响叫同时作，而 LED 闪烁的速度取决于蜂鸣器响叫的时间，详细请参考第九章蜂鸣器的应用。

➢ KeyProcess( )

目的

处理按键的各种应用

程序

```
01 //usually key process notics
02 void KeyProcess(void)
03 {
```

```
04    GetKey2( );          //detect key
05
06    if ( KeyBuffer != KeyData )
07    {
08       KeyBuffer = KeyData; //update new key value
09       ScanKeyCounter = 20; //key debounce delay or simultaneously
press
10                                two key
11    }
12    else if( ScanKeyCounter != 0 )
13    {
14       ScanKeyCounter--;
15       DelayX1ms(1);
16    }
17    else if ( KeyBuffer != NO_KEY )
18    {
19       switch ( KeyBuffer )
20       {
21        case KEY1 :
22          if ( FgKEY1==0 )
23          {
24             FgKEY1 = 1;     //wait release key
25             Beep4(1,17,10);
26          }
27          break;
28        case KEY2 :
29          if ( FgKEY2==0 )
30          {
31           FgKEY2 = 1;
32           Beep4(1,17,10);
33           FgKEY2_ONOFF= !FgKEY2_ONOFF; //key toggle action inverse
35              if ( FgKEY2_ONOFF==1 )
36                 LedOn( );
37              else
38                 LedOff( );
```

```
39          }
40          break;
41        case KEY3 :
42          if ( (FgKEY3==0) && FgKEY2_ONOFF )        //if     KEY2
press,then
43                                  press KEY3 not action
44          {
45            FgKEY3 = 1;
46            Beep4(2,17,10); //beep.beep 2 times
47            return;      //exit switch loop
48          }
49          if ( FgKEY3==0 )
50          {
51          FgKEY3 = 1;
52          Beep4(1,17,10);
53          }
54          break;
55        case KEY4 :
56          if ( FgKEY4==0 )
57          {
58            FgKEY4 = 1;
59            Beep4(1,17,10);
60            KeyCountTimer = TIME_400MS;   //the first timer=400ms
61            UpdateValue = 0x01;      //press key,then value+1
62            SubProcess( );
63          }
64          else if ( KeyCountTimer==0 )
65          {
66            KeyCountTimer = TIME_80MS; //if    continue    press
key,then key
67                                      change fast speed
68            UpdateValue = 0x01;
69            SubProcess( );
70          }
71          break;
```

```
 72        case KEY5 :
 73          if ( FgKEY5==0 )
 74          {
 75            FgKEY5 = 1;
 76            Beep4(1,17,10);
 77            KeyCountTimer = TIME_400MS;    //timer=400ms
 78            UpdateValue = 0x01;          //first,value+1
 79            SubProcess( );
 80          }
 81          else if ( KeyCountTimer==0 )
 82          {
 83            KeyCountTimer = TIME_80MS; //if    continue    press
key,then key is
 84                                       //  fast speed
 85            CoarseAdjCount++;
 86            if ( CoarseAdjCount < 6 )  //count<6
 87              UpdateValue = 0x02;      //start adj,fine adjust
 88            else
 89              UpdateValue = 0x06;      //coarse adjust=3 * fine
 90            SubProcess( );
 91          }
 92          break;
 93        default :
 94          break;
 95       }
 96    }
 97    else        //no key press,then clear flag
 98    {
 99      KeyBuffer    = NO_KEY;
100      CoarseAdjCount = 0;
101      FgKEY1         = 0;
102       FgKEY2       = 0;
103       FgKEY3       = 0;
104       FgKEY4       = 0;
105       FgKEY2_ONOFF = 0;
```

```
106     }
107 }
```

说明

按键检测例程可以知道哪一个键被按下，是最基本的也是一定会用得到的，按了键之后的动作才是系统中所要的功能，范例中提供数种按键之后常用的动作模式。

行　号	说　　明
04	利用 ADC 方式的复合键检测函数
06~16	延迟按键的弹跳波，也可以做为检测复合键所须的时间差，即虽同时按下二或三键也会有快慢的分别，只是时间很短大约数 mS，其中 ScanKeyCounter 为延迟时间的计数值。当第一次按下键 KeyBuffer 和 KeyData 不相同，则记录此按键值并将计数值 ScanKeyCounter 为 20 即延迟 20mS，并返回原调用程序，原调用程序会继续调用本范例程序即 KeyProcess( )，如果一直按着键不放则程序便执行行号 12 括号下的动作，即延迟 1mS 一直到 ScanKeyCounter 等于 0，也就是等待 20mS 之后程序才会继续做下去
17	当按下键也一直按着，而且已经过了 20mS 之后程序才会到这里
19	依据所按下的键而执行不同的动作，而使用 switch...case 的叙述会更容易明确及区分各种按键的功能定义
21	case KEY1：一直按着仍然只做一次的功能
22~27	按下 KEY1 的动作，简单的蜂鸣器响叫 1 声，FgKEY1 设定为“1”；可以避免按键一直按着而蜂鸣器也一直响叫不停，即必须释放按键之后才能再接受按键的动作，而 break 是跳出 switch 循环
28	case KEY2：每按 1 次就转态的功能
31	设定 KEY2 只动作 1 次的标志，必须释放按键
33	FgKEY2_ONOFF 标志取反，如第一次为“1”再按一次就变成“0”，又再按一次就恢复为“1”，如此"1"->"0"->"1"->"0"--直循还下去
35~38	依据 FgKEY2_ONOFF 标志令 LED 亮或熄
41	case KEY3：按了 KEY2 让 FgKEY2_ONOFF 标志设定为“1”，就无法按 KEY3 的功能
42~48	如果按了 KEY2 使得 FgKEY2_ONOFF 标志设定为“1”，则即使按了 KEY3 也不会动作，而蜂鸣器响叫 2 声做为提示并返回原调用程序

行 号	说 明
49~53	KEY2 未按下使得 FgKEY2_ONOFF 标志为"0"，当按下 KEY3 时则蜂鸣器响叫 1 声并设定只做一次的标志
55	case KEY4：…直按着键就持续动作的功能
56~63	按了 KEY4 则蜂鸣器响叫一声，并设定计时 400mS 和单位数值等于 1，即 UpdateValue=1，且作 SubProcess( )的例程
64~70	如果 KEY4 一直按着不放，则第一次 400 mS 过了之后就做累加 1 的动作，之后改成 80 mS 过了之后就做累加 1 的动作，一般做为监视器画面尺寸的可视范围的调整之用，而不要按 1 次才调整 1 个单位，大部分监视器画面尺寸的可视范围的控制为 PWM，而一般 PWM(Pulse Width Modulation:脉冲宽度调变)输出为 255 级，如果要调整到最大单位 255 时或最小单位 0 时，岂不是要按 255 次，按到手痛！
72	case KEY5：一直按着有微调、粗调的功能
73~80	同行号 56~63 的功能
81~91	每经过 80mS 则 CoarseAdjCount 就累加 1，如果小于 6 则每次只加 2 即是微调，如果大于 6 以上则每经过 80mS 就累加 6 即是粗调
93,94	不是按 KEY1、KEY5 而是按了其它键则执行 default 指令，break 是跳出 switch 循环
98~106	当未按下任何键时，则清除变量值和标志变量恢复成起始状态

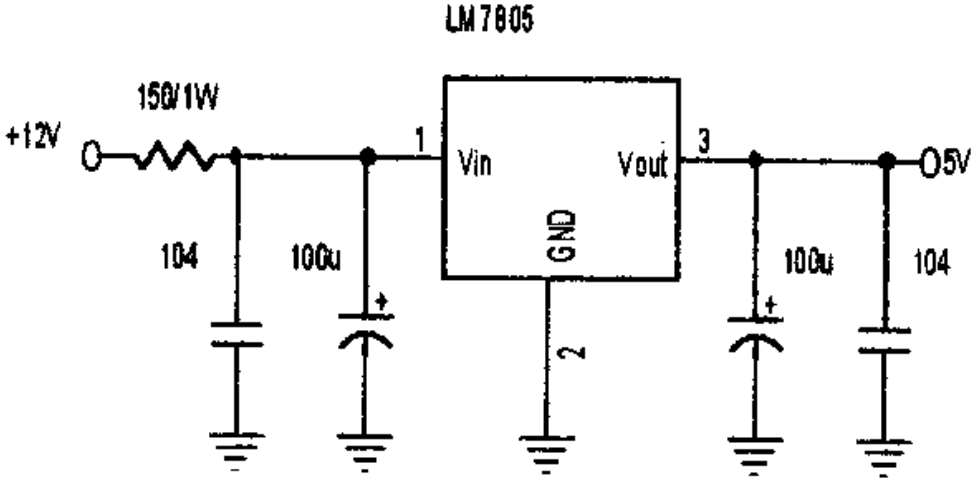
第 18 章　可控制电源电压的应用

➢ LM7805(　)

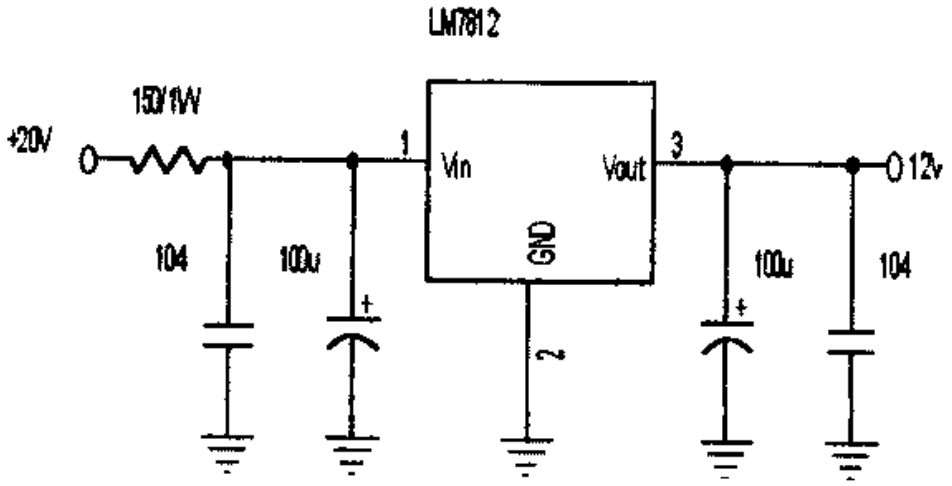
目的

固定输出电源电压的应用

电路



图一



图二

说明

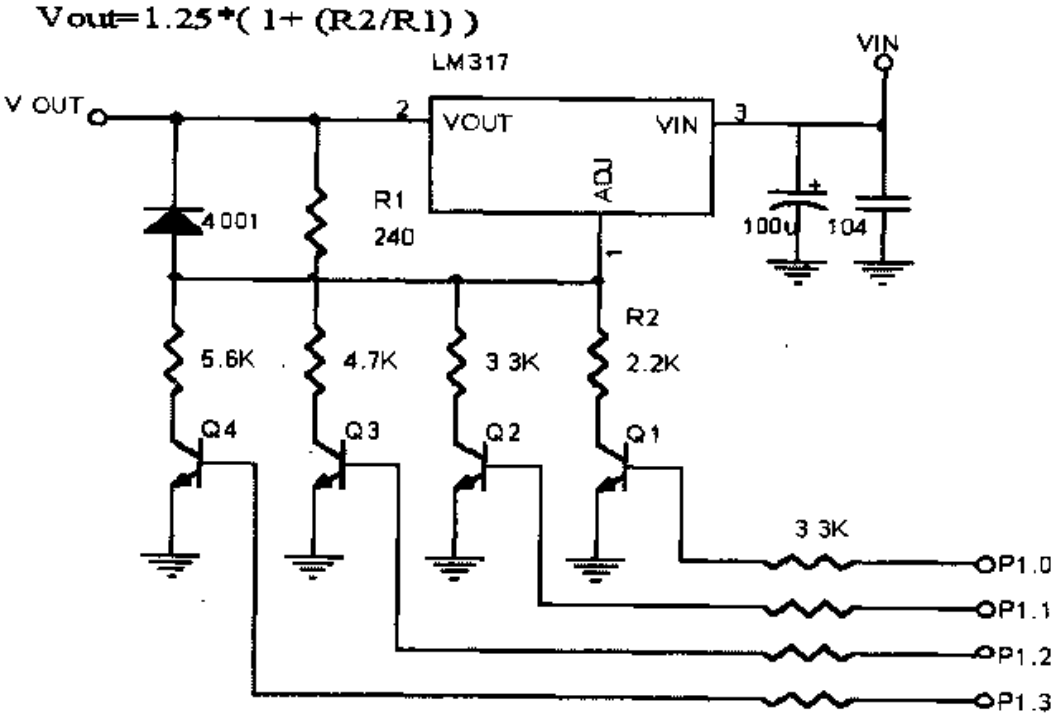
电子组件要正常动作都需要电源电压供给，一般常用的电源电压为+5V 或+12V，因为数字 IC（Integrated Circuit：集成电路）所供给的电压为+5V，而 CMOS IC 所供给的电压为+12V，其电路接法如图一、图二所示。

➢ LM317( )

目的

可控制电源电压的应用

电路



程序

```
01 void LM317_1(void)
02 {
03     P1_0=1;      //Q1 on,Vout=1.25*( 1+ (2.2K/240) )=12.7V
04     P1_1=0;
05     P1_2=0;
06     P1_3=0;
07 }
08
```

```
09 void LM317_2(void)
10 {
11     P1_0=0;     //Q3 on,Vout=1.25*( 1+ (4.7K/240) )=25.7V
12     P1_1=0;
13     P1_2=1;
14     P1_3=0;
15 }
16
17 void LM317_3(void)
18 {
19     P1_0=1;     //Q1,Q2 on,R=2.2*3.3/(2.2+3.3)=1.32K
20     P1_1=1;     //Vout=1.25*( 1+ (1.32K/240) )=8.125V
21     P1_2=0;
22     P1_3=0;
23 }
```

说明

上一个范例其输出为固定电压往往不敷所用，在实用上有时需要验证或测试而所设计的电子电路其需求的电源电压也许会有不一样，因此借着 I/O 来控制不同的输出电压，也是常用的方法而 LM317 正是可调整输出电压的组件。

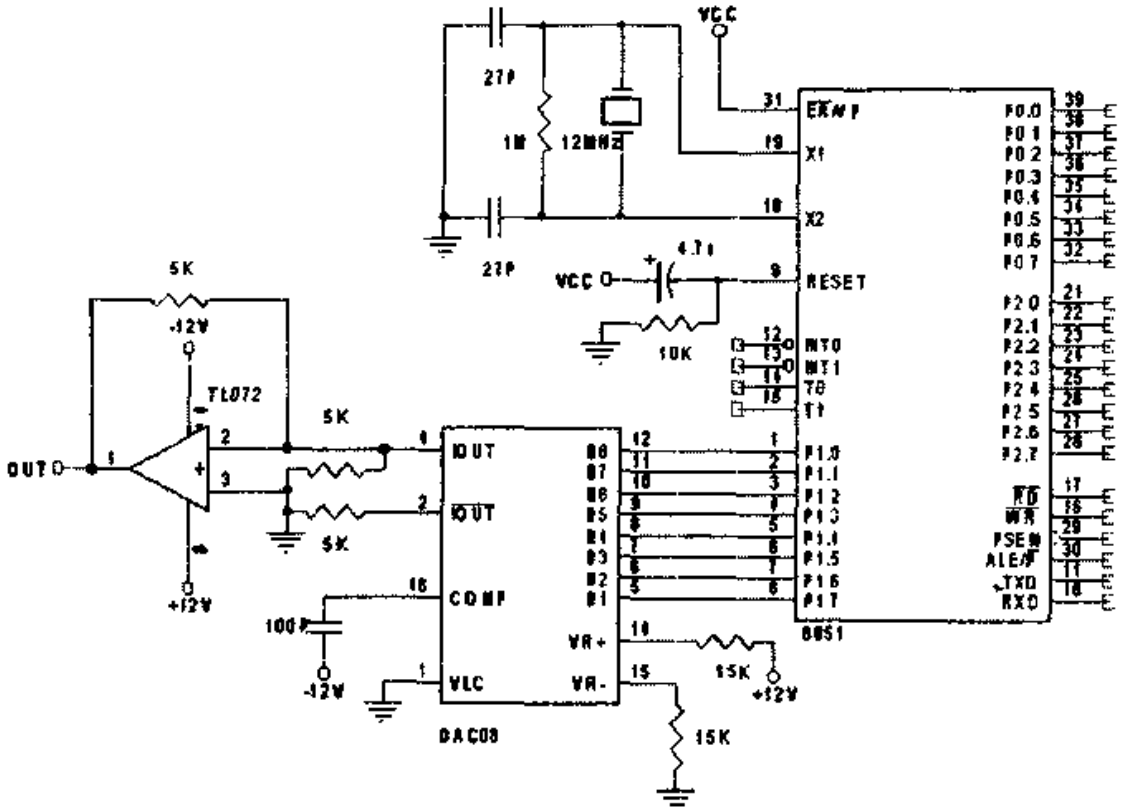
行 号	说　明
01~07	I/O P1.0="1"即 Q1 导通，则输出第一组电压，其输出电压的公式如行号 03 的注解所示
09~15	I/O P1.2="1"即 Q3 导通，则输出第三组电压，其输出电压的公式如行号 11 的注解所示
17~23	I/O P1.0="1"，P1.1="1"即 Q1、Q2 导通，其输出电压须先求出并联电阻后再代公式换算，如行号 19、20 的注解所示

➢ Dac08(　)

目的

利用数字至模拟转换 IC 来产生电源电压

电路



程序

```
01 // Dac08 digital to analog voltage output
02 // Vdd=2mA*5K=10V
03 void Dac08_1(void)
04 {
05     P1_0=1;      //volt=Vdd*((128+1)/255)
06     P1_7=1;
07 }
08
09 void Dac08_2(void)
10 {
11     P1=0x81;          //volt=Vdd*((128+1)/255)
12 }
```

说明

利用 DAC（Digital-to-Analog Converter，数字模拟转换器）其所能提供的电

压变化更为广泛，一般为 8 位的 DAC 所以总共有 255 段，如果输出最大电压为 10V，则每一段的电压为 10/255=39.2mV。

行 号	说　　明
03~07	设定 I/O P1.0="1"、P1.7="1"则换算成十进制为 129，因此输出电压为 10*(129/255)=5.05V
09~12	直接令 P1 端口的输出为十六进制 0x81，换算成十进制也是为 129，是另外一种程序写法，想要输出多少的电压则先换算出数值再直接由 P1 端口输出，就会产生预定的电源电压了

➢ SawTooth( )

目的

利用 DAC 产生锯齿波的应用

程序

```
01 void SawTooth1(void)
02 {
03    Byte  i=0x00;
04
05    while( 1 )
06
07       P1=i;     //P1 outport
08       i++;      //if i=0xff,then i+1=0
09    }
10 }
11
12 void SawTooth2(void)
13 {
14    Byte  i=0xff;
15
16    while( 1 )
17
18       P1=i;
```

```
19        i--;      //if i-0x00,then i-1=0xff
20    }
21 }
```

说明

DAC IC 所能控制的电压范围弹性很大，其输出的电压是依据所输入的数字信号而改变，因此只要改变输入的数值其输出也就跟着改变，因而可以创造各种波形即输出电压的变化。

行 号	说　明
01	往上走正斜率的锯齿波
03	初始值 i 等于 0
05	不断的重复执行循环，即不断的产生锯齿波
07,08	P1 端口的输出范围从 0~255 不断变化，因为每输出电压后 i 就累加一又继续输出，i 到了 255 加 1 后就变成 0，又从头开始形成了往上走的锯齿波，其锯齿波输出的频率可以在行号 07、08 之间插入延迟例程，但频率越低其波形较不流畅平滑，是因为延迟例程所致
12	往下走负斜率的锯齿波
18	P1 端口输出 i 的数值
19	减 1 后又存回 i 变数

➢ TriAngle( )

目的

利用 DAC 产生三角波的应用

程序

```
01 void TriAngle1(void)
02 {
03    Byte  i=0x00;
04
05    while( 1 )
06    {
```

```
07 PLUS:
08        P1=i;
09        i++;
10        if ( i!=0xff ) goto PLUS;
11
12 MINUS:
13        P1=i;
14        i--;
15        if ( i!=0x00 ) goto MINUS;
16     }
17 }
18
19 void TriAngle2(void)
20 {
21     Byte  i;
22
23     while( 1 )
24
25        for(i=0; i<255; i++)
26           P1=i;
27        for(i=255; i>0; i--)
28           P1=i;
29     }
30 }
31
32 void TriAngle3(void)
33 {
34     Byte  i;
35
36     while( 1 )
37
38        for(i=0; i<255; i++)
39           P1=i;
40        for(i=0; i<255; i++)
41           P1=~i; //P1=(255-i);
```

```
42      }
43  }
```

说明

三角波的原理是从0~255输出后，又从255~0的反向输出，而形成不断的增加输出电压，当达到了最大值后又开始不断的减少输出电压，当达到了最小值后又重复此循环动作而产生三角波。

行 号	说　明
05	不断重复三角波的输出波形
07~10	输出i数值后i就累加1，如果i还没有到最大值0xff则继续输出，则波形也就慢慢的往上爬升
12~15	当i到了最大值0xff后则P1端口先输出i数值并递减1，如果i还没有到最小值0x00则继续递减也继续输出，则波形也就慢慢的往下降了
19	三角波输出波形的第二种程序写法
25,26	使用for循环从0~255递增输出波形
27,28	使用for循环从255~0递减输出波形
32	三角波输出波形的第三种程序写法
40,41	使用for循环从0~255递增的方式，但将P1端口的输出却依据i的数值做反相，即将i的每一位反相

➢ Square( )

目的

利用DAC产生方波的应用

程序

```
01 void Square(void)
02 {
03     Byte  i;
04
05     while( 1 )
06
```

```
07        for(i=0; i<=255; i++)
08          P1=0x00;
09        for(i=0; i<=255; i++)
10          P1=0xff;
11      }
12  }
```

说明

产生方波的原理是输出最大电压后再输出最小电压，其间隔时间可任意调整，即形成方波的频率。

行 号	说　明
05	不断的产生方波循环
07	for 循环的条件判断式 i<=255 可控制方波的频率，其 i 数值越小频率越高，反之，数值越大则频率越低
08	输出为 0
09,10	输出最大电压，在此为 10V 并持续一段时间，如果要改变输出电压为 5V，则须将行号 10 的设定 P1=0xff 修正成 P1=0x80

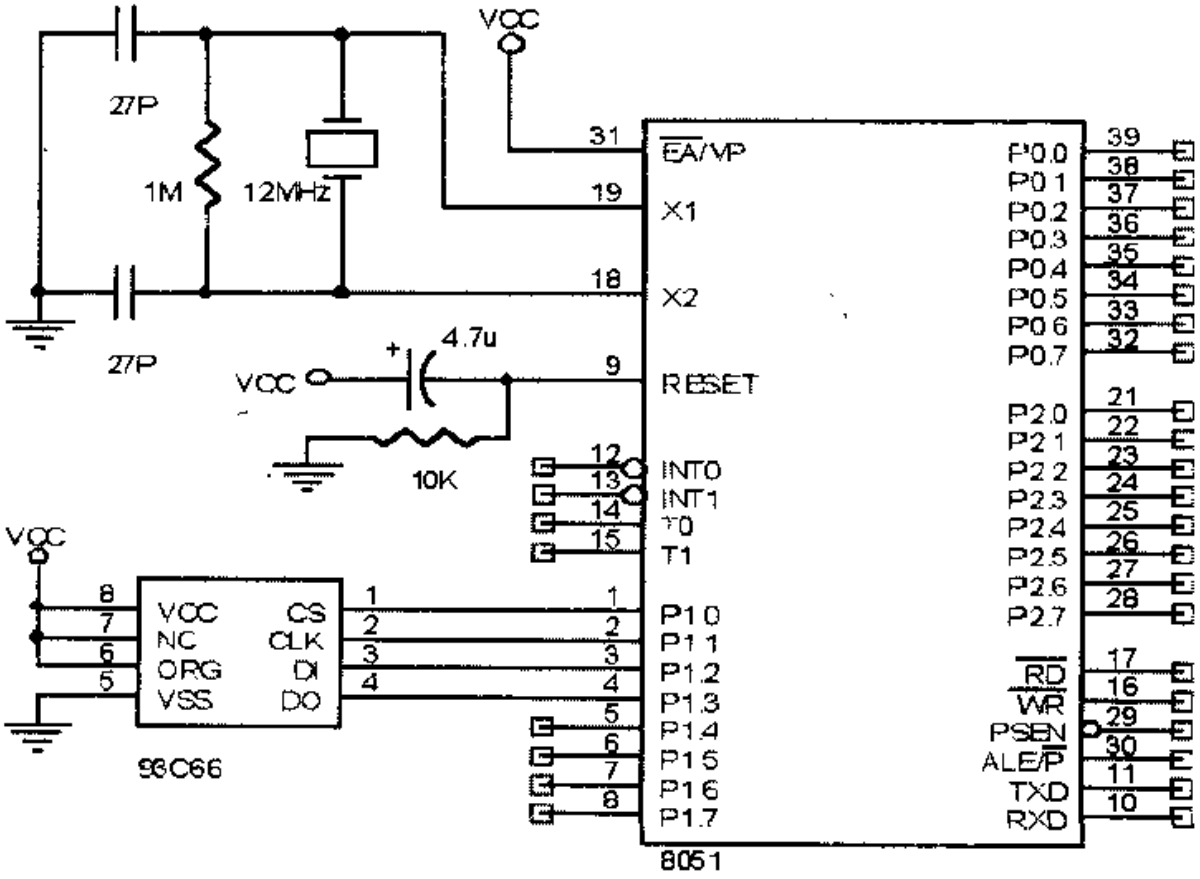
第 19 章　存储芯片 93C66 的应用

- PushEeprom93c66(　)

目的

将 128 个数据存入 EEPROM 93c66 内

电路



程序

```
01 //CS_PIN       =P1.0 , 93C66 PIN1:CS,chip select
02 //XSCK_PIN =P1.1 , 93C66 PIN2:CLK,clock
03 //XSDO_PIN =P1.2 , 93C66 PIN3:DI,data input
04 //XSDI_PIN =P1.3 , 93C66 PIN4:DO,data output
05
06 void PushEeprom93c66(void)
07 {
08     Byte  adr,value=0;
09
10     for(adr=0; adr<128; adr++)
11     {
12        CS_PIN=1;     //enable
13        EepWriteData(adr,value);
14        CS_PIN=0;
15        XSCK_PIN=0;
16        value++;
17     }
18 }
```

说明

EEPROM 93c66 可用来储存数据，其存储空间较少而写入或读取数据的方式为三线式并不是 ii2 bus 的协议，而 EEPROM 24c04、24c08、24c16 其控制的通信协议是 ii2 bus，如果将电源关掉，EEPROM 仍然可以保存正确的数据，也就是并不会因为失去了电源其内容就消失，因而常用于存储重要数据。

行　号	说　　明
01~04	93c66 接脚的定义与微电脑 I/O 脚的连接
06	此测试程序的作用为：第 0 地址写入 0，第 1 地址写入 1，第 2 地址写入 2，依此类推
08	初始数据值为 0
10	写入 128 个数据到 93c66 内
12	使能信号

行 号	说　明
13	依照所指定的地址，将数据写入93c66内
14	除能信号
15	clk送“0”电平
16	要写入的数值加1

➢ EepWriteData(adr,value)

目的

写数据至93c66内

程序

```
01 void EepWriteData(Byte adr,Byte value)
02 {
03     CS_PIN=1;
04     EwenROM( );
05     EraseROM(adr);
06     WriteROM(adr,value);
07     EwdsROM( );
08     CS_PIN=0;
09     XSCK_PIN=0;
10 }
11
12 void EwenROM(void)
13 {
14     CS_PIN=1;
15     WR_3wire(0x04);
16     WR_3wire(0xc0);
17     CS_PIN=0;
18     XbusDelay( );
19 }
20
```

```
21 void EraseROM(Byte adr)
22 {
23    CS_PIN=1;
24    WR_3wire(0x07);
25    WR_3wire(adr);
26    CS_PIN=0;
27    XSCK_PIN=0;
28    XSDO_PIN=0;
29    DelayX1ms(10);
30    if ( XSDI_PIN==0 )  //if always DO="0"
                  indicate is still in progress
31      ;
32 }
33
34 void EraseROM1(Byte adr)
35 {
36    Word  i;
37
38    CS_PIN=1;
39    WR_3wire(0x07);
40    WR_3wire(adr);
41    CS_PIN=0;
42    XSCK_PIN=0;
43    XSDO_PIN=0;
44    DelayX1ms(10);
45
46    FgTimeout=1;
47    for( i=0; i<1000; i++)      //timeout flag=1
48
49      if ( XSDI_PIN!=0 )
50      {
51        FgTimeout=0;
52        break;          //exit "for" loop
53      }
54    }
```

```
55 }
56
57 void WriteROM(Byte adr,Byte value)
58 {
59     CS_PIN=1;
60     WR_3wire(0x05);
61     WR_3wire(adr);
62     WR_3wire(value);
63     CS_PIN=0;
64     XSCK_PIN=0;
65     XSDO_PIN=0;
66     DelayX1ms(10);
67     if ( XSDI_PIN==0 )
68        ;
69 }
70
71 void EwdsROM(void)
72 {
73     CS_PIN=1;
74     WR_3wire(0x04);
75     WR_3wire(0x00);
76     CS_PIN=0;
77 }
78
79 void WR_3wire(Byte value)
80
81   Byte i;
82
83   for(i=0; i<8; i++)
84   {
85      if (value & 0x80)  XSDO_PIN=1;
86      else               XSDO_PIN=0;
87
88      value <<= 1;
89      XbusDelay( );
```

```
90
91      XSCK_PIN=1;
92      XbusDelay( );
93      XSCK_PIN=0;
94      XbusDelay( );
95    }
96 }
97
98 void XbusDelay(void)        //delay 50uS
99 {
100     Byte i;
101
102     for(i=0; i<10; i++)
103       ;
104 }
```

行 号	说　明
01	写入数据到 93c66 流程的主程序
03	使能
04	EWEN 例程
05	ERASE 例程
06	WRITE 例程
07	EWDS 例程
08,09	除能及清除 SCK
15	EWEN 的命令码为 0x04
16	EWEN 的地址为 0xC0
17,18	除能及延迟一段时间
24	ERASE 的命令码为 0x07
25	清除指定的地址
26~29	除能、清除 SCK 及 SDO 及延迟 10mS
30,31	如果 SDI="0"则一直等待，表示还在处理中，如果有异常使得 SDI="0" 则程序会在这里一直执行循环，即“死机”

行 号	说 明
46~54	为了避免程序一直执行循环，因而作了 timeout 的处理，即做固定常数的 for 循环，固定常数值要依据硬件的反应时间而定，一般以 0.5 秒或 1 秒为限等，如果 SDI 不等于“0”电平表示正常回应，则立即跳出 for 循环
60	WRITE 的命令码为 0x05
61,62	将数据写入指定的地址内
63~66	除能、清除 SCK 及 SDO 及延迟 10mS
67,68	如果 SDI="0"则一直等待，否则返回原调用程序
74	EWDS 的命令码为 0x04
75	EWDS 的地址为 0x00
79	SCK 和 SDO 的时序子程序
83	传送一个字节，要作 8 次
85,86	从数值的最高位开始传送，如果位等于“1”则设定 SDO，如果位等于“0”则清除 SDO
88~94	数值左移 1 位，而 SCK 作"1"->"0"的变化
98	短时间数 uS 的延迟子程序
102,103	执行 10 次的空指令

➢ PopEeprom93c66( )

目的

将 93c66 内的 128 个数据读出来

程序

```
01 void PopEeprom93c66(void)
02 {
03     Byte  adr;
04
05     for(adr=0; adr<128; adr++)
06     {
```

```
07        CS_PIN=1;          //cs,enable
08        Buffer=ReadROM(adr); //test
09        CS_PIN=0;
10        XSCK_PIN=0;
11    }
12 }
```

行 号	说　明
05	做 128 次的 for 循环
07	使能，设定 CS 脚为“1”电平
08	在指定的地址将 93c66 的内容读出并暂存至 Buffer 变量内
09	除能
10	清除 SCK

➢ ReadROM(adr)

目的

读出 93c66 地址的数据

程序

```
01 Byte ReadROM(Byte adr)
02 {
03    Byte value;
04
05    XSCK_PIN=0;
06    XSDO_PIN=0;
07    WR_3wire(0x06);
08    WR_3wire(adr);
09    XSCK_PIN=0;
10    XSDO_PIN=0;
11    value=ReceiveByte( );
```

```
12     return value;
13 }
14
15 Byte ReceiveByte(void)
16
17     Byte i;
18     Byte value=0;
19
20     // Receive byte (MSB first)
21     for(i=0; i<8; i++)
22
23       XSCK_PIN=1;
24       XbusDelay( );
25       XSCK_PIN=0;
26       value <<= 1;
27       if ( XSDI_PIN )  value |= 0x01;
28     }
29     XbusDelay( );
30     return value;
31 }
```

行 号	说　　明
05,06	SCK 和 SDO 清除为“0”
07	READ 的命令码为 0x06
08	传送指定的地址
09,10	SCK 和 SDO 清除为“0”
11,12	读出数据并将数据回复至原调用程序
15	读出一个数据的子程序
21	做 8 次，因为有 8 个位
22~28	SCK 作"1"->"0"的变化后，先左移此数据再判断 SDI 脚，如果 SDI="1" 则数据位设为“1”，否则，数据位清除为“0”
29,30	延迟一段时间后，回复读出的数据至原调用程序

第 20 章　IIC BUS 的应用

➢ IIC BUS 概念

IIC 是 Inter-Integrated Circuit 的缩写，也就是 IC 与 IC 之间沟通的总线，传统的并行总线因为采用平行的结构，所以 IC 之间接线较多，且需要译码电路，而显得更复杂。而具备 IIC BUS 功能的组件，其地址已经内置在组件中，只需要两条线(sda，scl)就能传送数据，其可靠度和安全性会更好，每个具有 IIC BUS 的 IC 均可视为独立的模块。

数字 IC 的 I/O 一般可分为 Totem-pole，Open collector 和 Tri-state，IIC 接口本身为 Open Drain 或 Open Collector 构造，因此需要外加电源及提升电阻才能运作，若直接连接是无法动作的。IIC BUS 包括两条线，一条为 SCL(Serial- Clock)，另一条为 SDA(Serial Data)，每一个连接在 IIC BUS 上的组件，不论 Micro-Processor、LCD driver、Memory 或其它组件均有其各别的地址，这些地址在组件出厂时就已经决定，各组件在 BUS 上的存取动作均通过地址进行。IIC BUS 也允许 multi-master 的操作模式，有 master 能力的组件，均可取得 BUS 的主控权。

由于 IIC BUS 具有这些特性对设计者而言有以下的优点：

1. 通常在产品设计之初，设计者常以模块图作为初步设计，IIC 电路和模块图很相近，一旦模块图完成后，采用 IIC 组件可立即变成实际电路，因此，可缩短设计开发的时间。

2. 无须设计 IC 的接口和译码电路，因为 IIC 组件已经将接口集成在 IC 内。

3. 数据传送的协议可以用软件规划，具有高度的弹性。

4. IC 不论从系统中删除或加入都不会影响原系统的各组件。

5. 由于是两线式，除错、维修变得很容易，可以简化很多线材的连接而且成本也较节省。

6. 不仅是硬件模块化，软件亦可写成模块化，可减少软件开发时间。

7. IIC 一定是 2 线式，若系统中某一 IC 改良或升级可以直接互换，并不影

响原系统或修改原系统的结构。

对于制造商方，也是有以下的优点：

1. 由于只使用两条线就能在 IC 之间传送数据，所以 IC 的接脚减少，PCB 电路板所需的面积可以减少，使得电路更节省空间和降低成本。

2. 不须使用地址译码 IC，可减少使用逻辑组件。

3. 若使用 SO 的包装，可节省更多的空间。

而 IIC 适用情况大致可归纳几点：

1. 系统包含微控制器及其它周边。

2. 希望将连接的成本降至最低。

3. 系统不需要很快的传输速度，IIC 传输速度最快为 400Kbit/Sec。

4. 可以应用于 Multi-Master 的系统。

➢ IIC 总线协议

【Definition of Start and Stop】

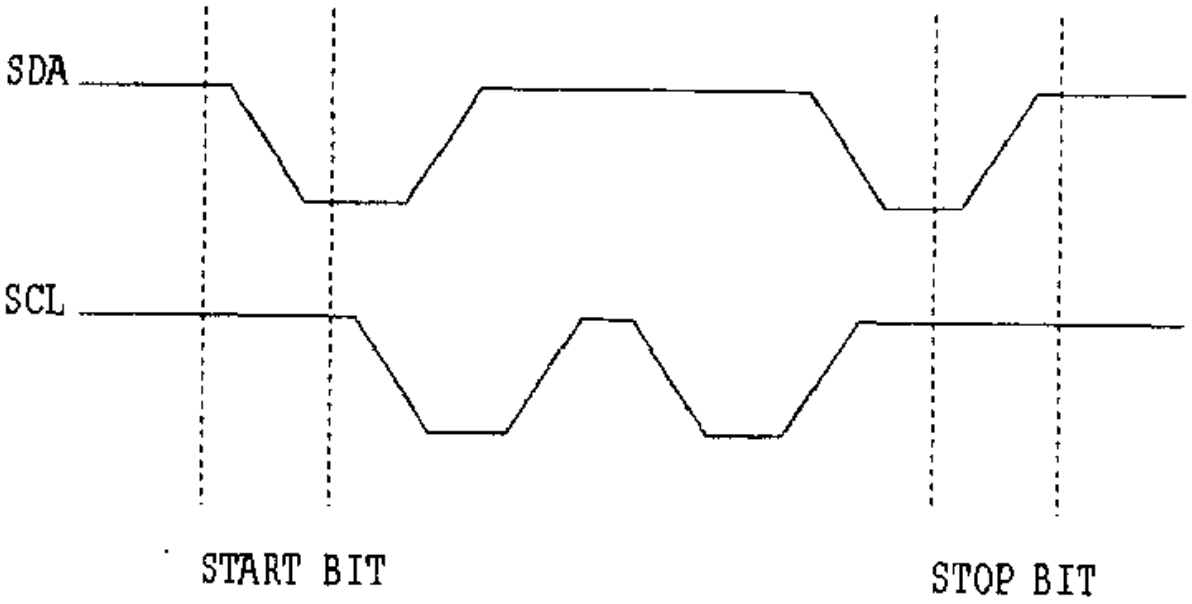


图 一

【Address Byte】

Address 7 bit

LSB=1,READ

LSB=0,WRITE

图 二

【Acknowledge Response from Receiver】

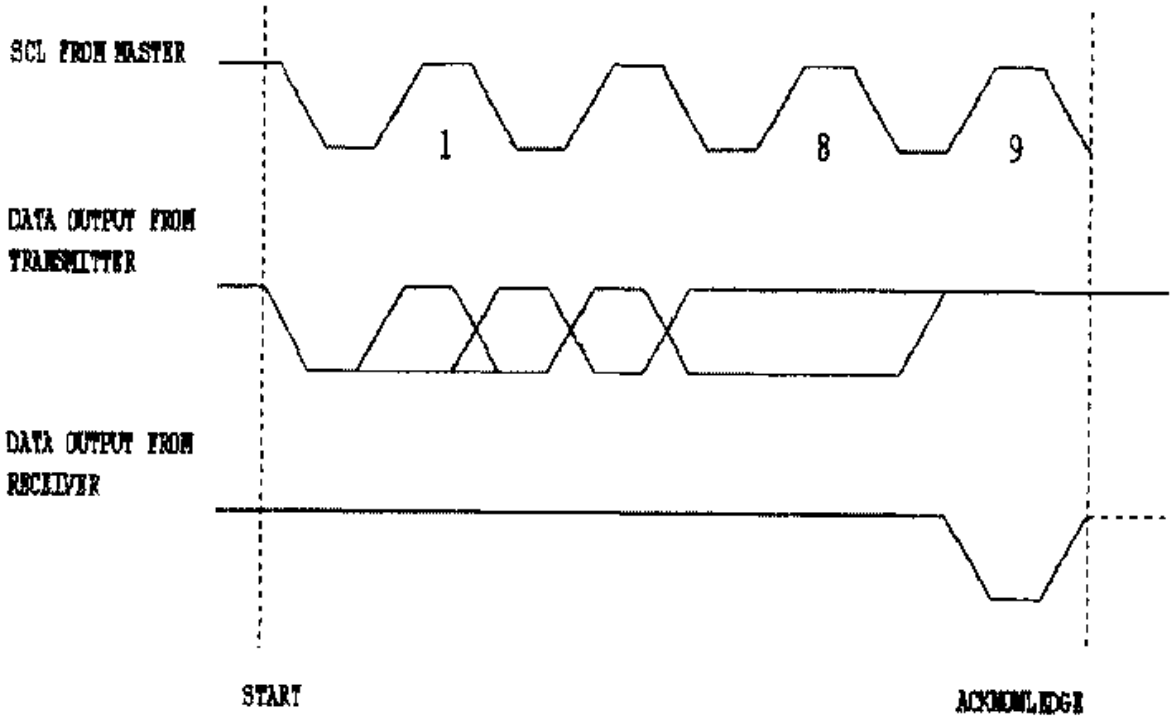


图 三

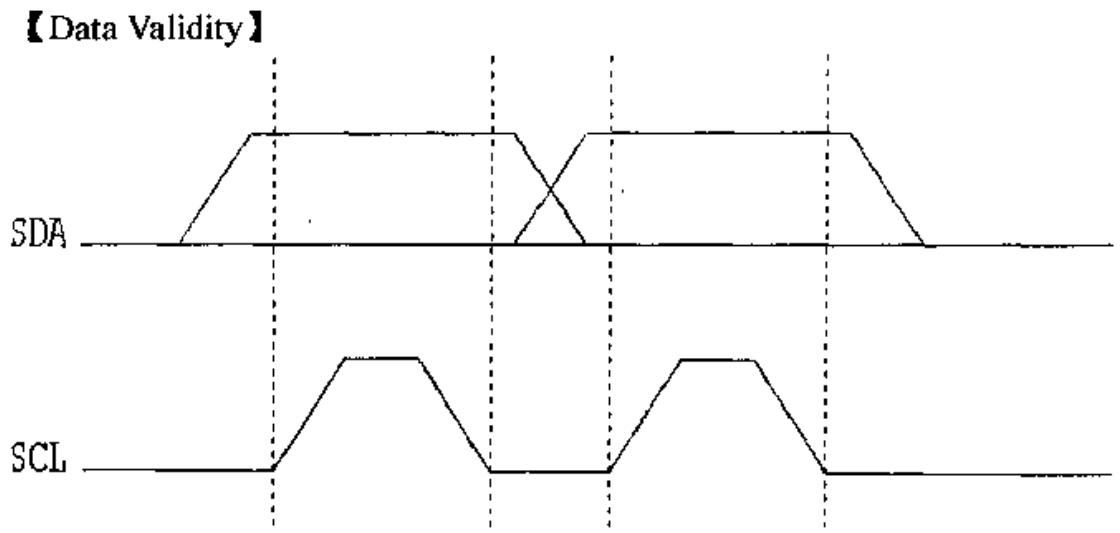


图 四

➢ 开始(Start)

master 端必须送出“开始”信号才能取得 IIC BUS 的控制权，当 IIC 没有动作时，而 scl、sda 都保持在高电平("1" level)，master 端先在 sda 送出低电平时("0" level)，经一小段时间后，再将 scl 变成低电平，如图 1。

➢ 地址(Address)

每一个送到 BUS 上的数据都必须是 8bit，以标准模式为例，地址具有 7 个 bit，master 先将 MSB 传送到 BUS 上，再依次传出 7 个 bit，此 7 个 bit 为地址，最后一个是 LSB，也就是 R/W，如图 2。

➢ 读/写(Read/Write)

紧接着地址的是读/写位，它只占一个位，高电平时是读取，低电平时是写入，如图 2。

➢ 确认(Acknowedge)

slave 端如果正确地收到地址和读/写后，发出确认的信号向 master 表示已经收到数据，此时 master 将 sda 释放成高电平，slave 确认时将 sda 拉至低电平，如果 slave 端未能正确地收到地址和读/写，则 slave 不动作，使 sda 维持在高电平，如图 3。

➢ 数据(Data)

数据和地址及读/写是一样的，只不过数据可能由 master 或 slave 送出，也可能由 master 或 slave 接收，而地址及读/写只由 master 送出、slave 接收。数据的意义随不同的组件可能有异，例如对 Serial EEPROM 而言可能为内存地址或内存的内容。另外 sda 若要变化，只有在 scl 为低电平时才可以，而 scl 在高电平时，其对应的 sda 状态就是其传送的位，如图 4。

➢ 停止(Stop)

master 完成和 slave 的动作后，先将 scl 释放至高电平，经一小段时间后，再将 sda 变成高电平，完成停止的动作，此时 IIC BUS 为 free 的状态，如图 1。

➢ IIC BUS 时序(Timming)

对于系统的正常运作和可靠度而言，时序是非常重要的，很多系统有不正常的问题大部分都是时序不够“安全”所致，也就是 SCL 和 SDA 之间延迟的时间不够。

如果以标准模式而言：

1. Stop 后须再等待至少 4.7 uS 的时间，才可再 Start。
2. Start 信号最少必须 4.0 uS 的时间。
3. SCL 的低电平最少必须维持 4.7 uS 的时间。
4. SCL 和 SDA 的上升时间最多不超过 1.0 uS 的时间。
5. SCL 已经降低电平后，SDA 至少必须维持 5.0 uS 的时间。
6. SCL 的高电平最少必须维持 4.0 uS 的时间。
7. SCL 和 SDA 的下降时间最多不超过 0.3 uS 的时间。
8. SDA 必须在 SCL 转变为高电平之前，SDA 至少必须领先 0.25 uS 的时间。
9. 连续 Start 时，SDA 必须在 SCL 转变为高电平之后，SDA 至少必须多维持 4.7 uS 的时间。
10. Stop 时， SCL 至少必须领先 4.0 uS 的时间。

其应用程序如下：

➢ I2cStart( )

目的

IIC BUS 的起始信号

程序

```
01 /*============================
02   I2c start condition
03  SDA high->low while SCL=high
04       _________
05   SCL         \_________
06       _______
07   SDA        \___________
08 ============================*/
09 void I2cStart(void)
10 {
11     ISDA=1;
12     ISCL=1;
13     I2cWait( );
14     ISDA=0;
15     I2cWait( );
16     ISCL=0;
17 }
```

行 号	说 明
11~13	SDA 和 SCL 先设为“1”电平，并延迟数 uS 的时间，时间参数须符合 IIC BUS 的时序
14	SDA 必须先清除“0”电平
15	再一次延迟数 uS 的时间后
16	SCL 才可以清除“0”电平

➢ I2cStop( )

目的

IIC BUS 的停止信号

程序

```
01 /*============================
02   I2c stop condition
03   SDA low->high while SCL=high
04              _________
05   SCL_______/
06                 _______
07   SDA_______/
08 ============================*/
09 void I2cStop(void)
10 {
11     ISDA=0;
12     I2cWait( );
13     ISCL=1;
14     I2cWait( );
15     ISDA=1;
16 }
```

行 号	说　明
11	SDA 清除“0”电平
12	延迟数 uS 的时间
13	SCL 必须先设定为“1”电平
14	延迟数 uS 的时间
15	SDA 才能再设定为“1”电平

➢ I2cWait( )

目的

延迟数 uS 的时间，以符合 IIC BUS 的时序

程序

```
01 //Wait for some time to get proper I2C timing
02 void I2cWait(void)
03 {
04     _nop_( );
05     _nop_( );
06 }
07
08 //Initialize I2C interface
09 //Release I2C BUS
10 void I2cInit(void)
11 {
12     ISDA=1;     //define I/O P1.1
13     ISCL=1;     //define I/O P1.0
14 }
```

行 号	说 明
04,05	以汇编语言 Assembler NOP 指令作为延迟，执行 NOP 指令只是要占用 CPU （Central Processing Unit）时间而并不作任何动作
10	IIC BUS 的初始化，SCL 和 SDA 正常状态都是要“1”电平
12	设定 SDA 为“1”电平
13	设定 SCL 为“1”电平

➢ I2cSentByte(bytedata)

目的

传送一个 Byte 数据至 slave (一)

参数

bytedata：要传送的数据

程序

```
01 //master transfer data to slave
02 //and return acknowledge bit
03 Bool I2cSentByte(Byte bytedata)
04 {
05    Byte i;
06    Bool ack;
07
08    ISDA=1;
09    for(i=0; i<8; i++)
10    {
11       //_crol_:include <intrins.h>
12       bytedata=_crol_(bytedata,1);
13       if (bytedata & 0x01)
14          ISDA=1;
15       else
16          ISDA=0;
17
18       ISCL=1;
19       I2cWait( );
20       ISCL=0;
21       I2cWait( );
22    }
23    ISDA=1;
24    I2cWait( );
25    ISCL=1;
26    I2cWait( );
27    ack=ISDA;
28    ISCL=0;
29    I2cWait( );
30    return ack;
```

```
31 }
```

行 号	说 明
06	ack 为“确认标志变量”
08	SDA 先设定“1”电平
09	要传送一个 Byte 数据，所以要作 8 次
12	向左移 1 位，即位 0 会移至位 1，而位 7 会移至位 0，依此类推，最后又存回原变量 bytedata 内，其中_crol_指令必须包含程序库 intrins.h 的文件
13~16	最高位已经移入最低位，因此要判断位 0，如果位 0 等于“1”则设定 SDA="1"，否则清除 SDA="0"
18~21	SCL 作"1"->"0"的变化
23~26	SDA 先设定为“1”，SCL 也设定为“1”，以让确认信号出现
27	读取确认信号
28,29	将 SCL 恢复为“0”电平
30	返回确认信号(ack)至原调用程序

➢ I2cSentByte1(bytedata)

目的

传送一个 Byte 数据至 slave (二)

参数

bytedata：要传送的数据

程序

```
01 //don't include <intrins.h>
02 Bool I2cSentByte1(Byte bytedata)
03 {
04     Byte i;
05     Bool ack;
06
07     //Transmit byte (MSB first)
```

```
08     for(i=0; i<8; i++)
09     {
10        if (bytedata & 0x80)
11           ISDA=1;
12        else
13           ISDA=0;
14
15        bytedata <<= 1;
16        I2cWait( );
17
18        ISCL=1;
19        I2cWait( );
20        ISCL=0;
21        I2cWait( );
22     }
23     ISDA=1;
24     I2cWait( );
25     ISCL=1;
26     I2cWait( );
27     ack=ISDA;
28     ISCL=0;
29     I2cWait( );
30     return ack;
31  }
```

说明

和上一个范例的差别是本程序不使用_crol_指令，所以也不需要包含链接库 intrins.h 的文件，其余部分都一样，其不同点如下说明：

行　号	说　　明
10~13	由最高位(位 7)开始传送，如果 bytedata 和 0x80 作 AND 逻辑运算等于 0x80，即表示位 7 等于“1”则条件式成立(非零就是真，零为假)，所以设定 SDA="1"，否则清除 SDA ="0"
15	每传送 1 位，则须将待传送的数据左移 1 位，以便继续测试下 1 位(由 bit7 开始至 bit0 结束)

➢ I2cReceiveByte( )

目的

从 slave 端读取一个数据

程序

```
01 // slave transfer data to master
02 Byte I2cReceiveByte(void)
03
04     Byte i;
05     Byte bytedata=0;
06
07     // Receive byte (MSB first)
08     for(i=0; i<8; i++)
09
10       ISCL=1;
11       I2cWait( );
12
13       bytedata <<= 1;
14       if ( ISDA )  bytedata |= 0x01;
15
16       ISCL=0;
17       I2cWait( );
18     }
19     return bytedata;
20 }
```

行 号	说　明
05	先初始化读取的数据变量
08	读取 8 个位，循环作 8 次
10,11	SCL 先设定为“1”，并等待一段时间
13	先将数据变量左移 1 位

行 号	说　　明
14	此时，如果 SDA="1"则将接收到的数据加 1(bit0 设为“1”)，否则，当左移一个位就会将 bit0 自动清除为 0
16,17	SCL 恢复为“0”电平并等待一段时间
19	返回此读取到的数据至原调用程序

➢ SendAcknowledge(ack)

目的

传送确认信号“1”或“0”电平至 slave 端

参数

ack：确认信号“1”或“0”电平

程序

```
01 //Master send acknowledge bit to slave
02 //acknowledge="0",non-acknowledge="1"
03 void SendAcknowledge(Bool ack)
04 {
05     ISDA=ack;
06     ISCL=1;
07     I2cWait( );
08     ISCL=0;
09 }
```

行 号	说　　明
05	设定 SDA 为要传送的确认信号
06~08	SCL 作"1"到"0"的变化

➢ I2cByteWrite(device,address,bytedata)

目的

将数据写入指定 slave 的地址内(一)

参数

device：slave 地址，每一个 IIC BUS 的组件(device)都有一个编号
address：sub 地址
bytedata：数据

程序

```
01 //do 1 times,and un-check acknowledge
02 void I2cByteWrite(Byte device,Byte address,Byte bytedata)
03
04     I2cStart( );
05     I2cSentByte(device);
06     I2cSentByte(address);
07     I2cSentByte(bytedata);
08     I2cStop( );
09     DelayX1ms(10);
10 }
```

说明

此程序只传送一次且不检测确认信号，如果 IIC BUS 受到干扰容易传送失误或不良，如果没有作 timeout 的程序结构处理，甚至会造成死机。

行 号	说　明
04	开始传送起始信号
05	传送 slave 地址
06	传送 address 地址，代表着 sub 地址或称为 IC 内部的地址
07	传送要写入的数据 bytedata
08	传送停止信号
09	延迟 10mS，IIC BUS 通信完成后则就会将数据(bytedata)写入至特定 IC(device 的 slave 地址)的内部地址(address)

➢ I2cByteWrite1(device,address,bytedata)

目的

将数据写入指定 slave 的地址内(二)

参数

device：slave 地址，每一个 IIC BUS 的组件(device)都有一个编号
address：sub 地址
bytedata：数据

程序

```
01 //retry,until acknowledge ok="0",if device bad,then do loop
02 void I2cByteWrite1(Byte device,Byte address,Byte bytedata)
03 {
04     Bool ack;
05
06     I2cStart( );
07
08 RESENT1:
09     I2cSentByte(device);
10     if (ack==1) goto  RESENT1;  //acknowledge error
11
12 RESENT2:
13     I2cSentByte(address);
14     if (ack==1) goto  RESENT2;
15
16 RESENT3:
17     I2cSentByte(bytedata);
18     if (ack==1) goto  RESENT3;
19
20     I2cStop( );
21     DelayX1ms(10);
22 }
```

说明

如上的程序范例，接收到的确认信号(ack)有错误，如果一直重新传送，不符合 IIC BUS 的协议，既使数据重传后正确，也已经不是正确的数据了。只要 IIC BUS 有错误就应该传送“停止信号”(Stop)后再从“起始信号”(Start)重新传送才可以确保数据的完整和正确，如下一个范例才是完整且实用的程序。

行 号	说 明
06	开始传送起始信号
08~10	传送1个字节（slave地址），如果传送错误则不断的重新传送这个字节
12~14	传送1个字节（address地址），如果传送错误则不断的重新传送这个字节
16~18	传送1位（要写入的数据bytedata），如果传送错误则不断的重新传送这1位
20	传送停止信号
21	延迟10mS

➢ I2cByteWrite2(device,address,bytedata)

目的

将数据写入指定slave的地址内(三)

参数

device：slave地址，每一个IIC BUS的组件(device)都有一个编号
address：sub地址
bytedata：数据

程序

```
01 //time out,and check acknowledge
02 void I2cByteWrite2(Byte device,Byte address,Byte bytedata)
03 {
04     Byte i;
05     Bool ack;
06
07     FgTimeout=1;
08     for( i=0; i<10; i++)       //time out,retry=10
09
10       I2cStart( );
11
12       ack=I2cSentByte(device);
13       if (ack==1)
```

```
14        {
15           I2cStop( );
16           continue;
17        }
18
19        ack=I2cSentByte(address);
20        if (ack==1)
21        {
22           I2cStop( );
23           continue;
24        }
25
26        ack=I2cSentByte(bytedata);
27        if (ack==1)
28        {
29           I2cStop( );
30           continue;
31        }
32
33        I2cStop( );
34        FgTimeout=0;
35        if ( ack==0 )  break;
36     }
37     DelayX1ms(10);
38  }
```

行　号	说　　明
07	先设定“时间超过标志”等于 1
08	重试 10 次，如果超过 10 次仍然有错误则离开并返回原调用程序，以避免程序不断执行循环，导致“死机”
10	开始传送起始信号
12~17	传送 1 字节（slave 地址），如果传送错误即“确认信号”等于“1”电平，就传送“停止信号”，然后继续从“起始信号”开始传送

行 号	说 明
19~24	传送 1 字节（address 地址），如果传送错误即“确认信号”等于“1”电平，就传送“停止信号”，然后继续从“起始信号”开始传送
26~31	传送 1 位（要写入的数据 bytedata），如果传送错误即“确认信号”等于“1”电平，就传送“停止信号”，然后继续从“起始信号”开始传送
33	传送成功则最后要传送停止信号
34	FgTimeout="0"，表示没有超时
35	如果“确认信号(ack)”等于“0”，表示传送成功则退出 for 循环
37	延迟 10mS，只为了 slave 是 EEPROM 时，因为 EEPROM 每写入一个 Byte 则必须延迟 10mS 才可以确保下一次写入的正确性

➢ I2cByteRead(device,address)

目的

从指定 slave 的地址内读取数据

参数

device：slave 地址，每一个 IIC BUS 的组件(device)都有一个编号
address：要读取数据的地址

程序

```
01 Byte I2cByteRead(Byte device,Byte address)
02 {
03     Byte bytedata;
04
05     I2cStart( );
06     I2cSentByte(device);
07     I2cSentByte(address);
08     I2cStart( );
09     I2cSentByte(device|0x01);
10     bytedata=I2cReceiveByte( );
11     SendAcknowledge(1);
12     I2cStop( );
```

```
13     return bytedata;
14 }
```

行 号	说　明
05	开始传送起始信号
06	传送 1 字节（slave 地址）
07	传送 1 字节（address 地址）
08	再一次的传送起始信号
09	传送“读取信号”，即 slave 地址加 1
10	从 slave 端读取一个数据并存入变量
11	Master 端送出确认的信号等于“1”，即 Acknowledge="1"
12	传送停止信号
13	返回读出值 bytedata 给原调用程序

➢ I2cSentData(bytecnt)

目的

将 TrmBuf 数组内的数据传送至 slave 端

参数

bytecnt：要传送数据的个数

程序

```
01 //master transfer count data to slave
02 Bool I2cSentData(Byte bytecnt)
03 {
04     Byte i;
05
06     for(i=0; i<bytecnt; i++)
07     {
08       if ( I2cSentByte (TrmBuf[i]) == FAILURE )
09          return FAILURE;
10     }
```

```
11     return SUCCESS;
12 }
```

行 号	说 明
06	要传送的数据有几个，则 for 循环就要作几次
07~10	将阵列内的数据 TrmBuf[i]依次传送出去，如果传送有错误则返回“失败信号(FAILURE)”给原调用程序
11	阵列内的数据已全部传送成功，则返回“成功信号(SUCCESS)”至原调用程序

➢ I2cReceiveData(bytecnt)

目的

从 slave 端读取一串数据并存入 TrmBuf 数组内

参数

bytecn：要接收数据的个数

程序

```
01 //slave transfer count data to master
02 Bool I2cReceiveData(Byte bytecnt)
03 {
04     Byte i;
05
06     for(i=0; i<bytecnt-1; i++)
07     {
08       TrmBuf[i]=I2cReceiveByte( );
09       SendAcknowledge(0);
10     }
11     TrmBuf[i]=I2cReceiveByte( );
12     SendAcknowledge(1);
13     I2cStop( );
14     return SUCCESS;
15 }
```

行 号	说　明
06	for 循环的次数等于 bytecnt 减 1，因为最后一笔的数据读取完毕后，则 master 端要送出“不要确认信号”，而其它读取到的数据要送出“确认信号”，此 master 端为微处理器
08	读取的数据存入至阵列内
09	送出“确认信号”
11	读取最后一笔的数据
12	送出确认信号等于“1”
13	传送停止信号
14	返回“成功信号(SUCCESS)”至原调用程序

➢ DataSetBit(device,addr,bitno)

目的

将指定 slave 地址的内部地址数据的第几个位强迫设定为“1”

参数

device：slave 地址，每一个 IIC BUS 的组件(device)都有一个编号
address：内部缓存器的地址
bitno：数据的第 0~7 位设定为“1”

程序

```
01 void DataSetBit(Byte device,Byte addr,Byte bitno)
02 {
03     Byte temp;
04
05     temp=I2cByteRead(device,addr);
06     temp |= (0x01<<bitno);
07     I2cByteWrite(device,addr,temp);
08 }
09 void DataSetBit1(Byte device,Byte addr,Byte bitno)
```

```
10 {
11     I2cByteWrite(device,addr,(I2cByteRead(device,addr) |
       (0x01<<bitno) ) );
12 }
13 void DataSetBit2(Byte device,Byte addr)
14 {
15     I2cByteWrite(device,addr,(I2cByteRead(device,addr) |
       0x80) );
16 }
```

行 号	说　明
03	暂存变量
05	先将指定 slave 的内部暂存器地址的内容读出并存入 temp 变量
06	强迫将 temp 变量内容的第 bitno 位设定为“1”后，并存入至 temp 变量内
07	存回到指定 slave 的内部暂存器内，即内部地址
11	不需要使用 temp 变量，直接读取并运算后就立即回存
15	将指定 slave 的内部暂存器地址的内容 bit7 设定为“1”

➢ DataClearBit(device,addr,bitno)

目的

将指定 slave 地址的内部地址的数据的第几位强制清除为“0”

参数

device：slave 地址，每一个 IIC BUS 的组件(device)都有一个编号
address：内部缓存器的地址
bitno：数据的第 0~7 位清除为"0"

程序

```
01 void DataClearBit(Byte device,Byte addr,Byte bitno)
02 {
03     Byte temp;
```

```
04
05      temp=I2cByteRead(device,addr);
06      temp &= ~(0x01<<bitno);
07      I2cByteWrite(device,addr,temp);
08  }
09  void DataClearBit1(Byte device,Byte addr,Byte bitno)
10  {
11      I2cByteWrite(device,addr,(I2cByteRead(device,addr) &
        ~(0x01<<bitno) ) );
12  }
13  void DataClearBit2(Byte device,Byte addr)
14  {
15      I2cByteWrite(device,addr,(I2cByteRead(device,addr) &
        0x7f) );
16  }
```

行 号	说　明
05	先将指定 slave 的内部暂存器地址的内容读出并存入至 temp 变量
06	将 temp 变量内容的第 bitno 位设定为“1”后，向相和原变量作 AND 运算即屏蔽此位后，再存入 temp 变量内
07	存回到指定 slave 的内部暂存器内，即内部地址
11	不需要使用 temp 变量，直接读取并运算后就立即回存，也就是将运算式直接当作引数来传递
15	将指定 slave 的内部暂存器地址的内容 bit7 清除为“0”

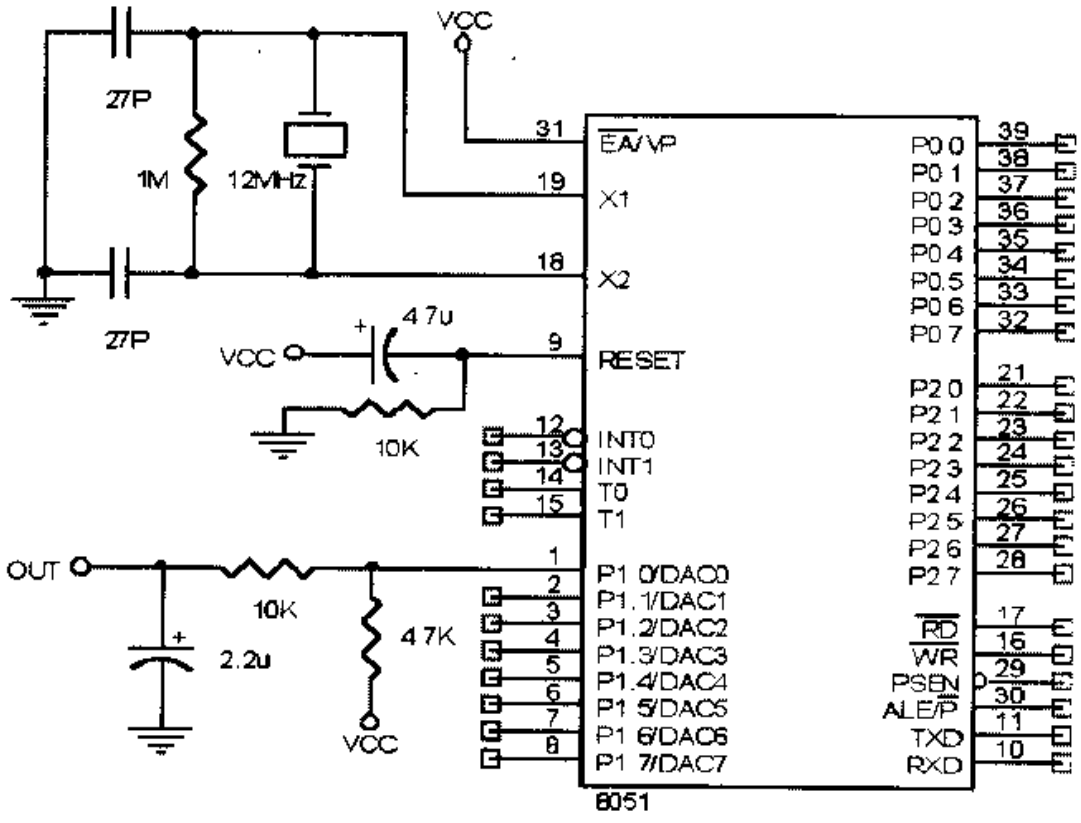
第 21 章　PWM IC 的应用

➢ PWM_Output(　)

目的

利用 PWM 作为输出“1”或“0”电平

电路



程序

```
01 //cpu internal PWM register output as Output
02 void PWM_OutputHI(void)
```

```
03 {
04     DAC0   = 0xff;        //output "1"
05 }
06
07 void PWM_OutputLOW(void)
08 {
09     DAC0   = 0x00;        //output "0"
10 }
11
12 void PWM_Output(Byte value) //could be set square frequance
13 {
14     DAC0   = value;       //output PWM:duty cycle=value/255
15 }
```

说明

PWM（Pulse Width Modulation：脉冲宽度调制）的特性就是以数值来产生脉冲宽的控制方式，通过改变方波的 duty-cycle "1"电平的长短，再经过电阻、电容（RC）电路转换成直流电压，因此只要 PWM 输出数值越大，则“1”电平的周期就越长，如果 PWM 输出最大值就变成输出“1”电平，PWM 输出最小值就变成输出“0”电平，因此当微处理器的输出脚不够使用时，可作为输出端口使用但不能作为输入端口，其 PWM 输出方波频率一般为 50~100KHZ，可以在内部缓存器内预先作设定使用。

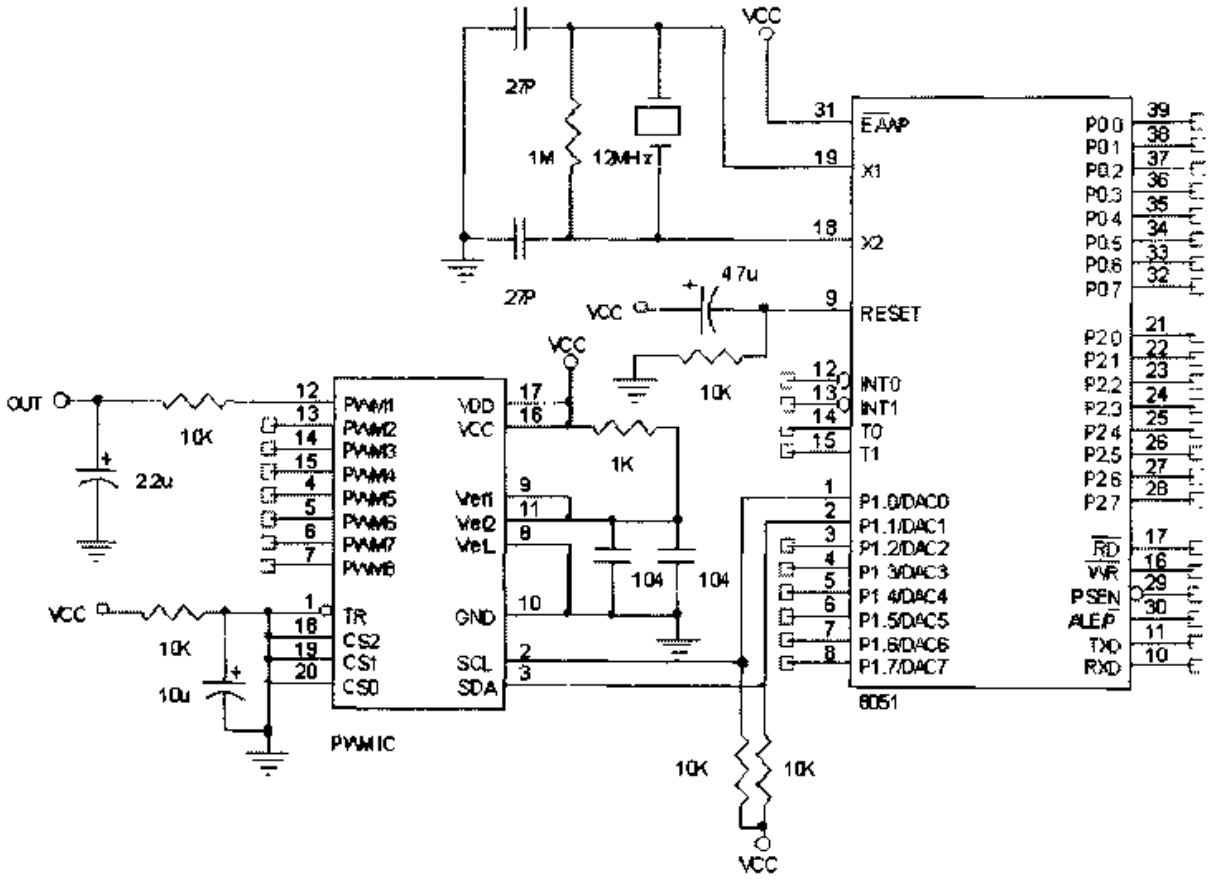
行 号	说　　明
04	将 PWM 输出最大值，即输出“1”电平
09	将 PWM 输出最小值，即输出“0”电平
14	依据参数形成脉冲宽调制输出

➢ TEST_DacOut (　)

目的

8 个输出的 PWM IC 的应用

电路



程序

```
01 //device=0x90,channel=0x01,value=0x80
02 void TEST_DacOut(void)
03
04     Byte device,channel,value;
05
06     device =0x90;    //device =0x90
07     channel=0x01;    //channel=0x01
08     value  =0x80;    //dac out=vdd * (0x80/0xff)
09     DacByteWrite(device,channel,value);
10 }
11
12 //DAC IC 8 channel,slave address=0x90
13 void DacByteWrite(Byte device,Byte subaddress,Byte
   bytedata)
```

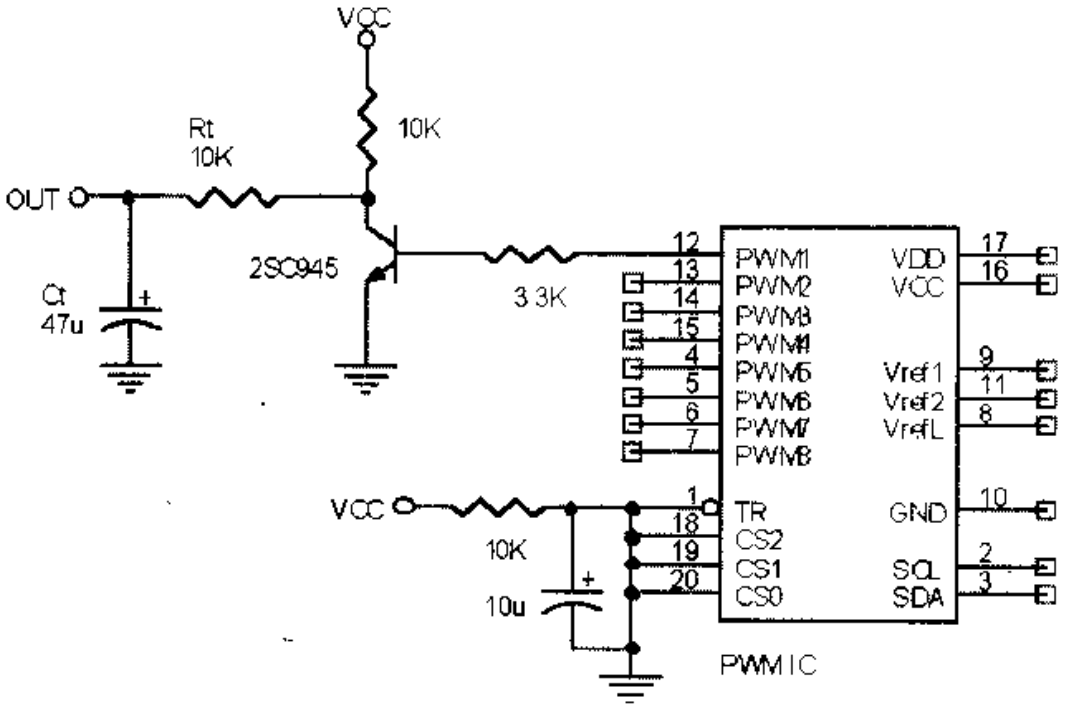
```
14
15      I2cStart( );
16      I2cSentByte(device);
17      I2cSentByte(subaddress);
18      I2cSentByte(bytedata);
19      I2cStop( );
20  }
```

说明

PWM IC 一般有 8 个输出，其控制方式是利用 IIC BUS 的通信协议，常应用在 PC monitor(PC 监视器)、LCD monitor(LCD 监视器)的画面几何调整，例如：水平大小(horizontal size)、水平位置(horizontal position)、垂直大小(vertical size)、垂直位置(vertical position)等，此程序在示范如何应用 PWM IC。

行　号	说　　明
06	这颗 PWM IC 的 slave 地址内定为 0x90，每一个 IIC BUS 的组件都有一个不同的地址，有一些 IC 的 slave 地址有开放 2 个，因此可以设定暂存器内容来选择其一，以区分其它组件的 slave 地址
07	选择第一个信道输出，如上图 PWM IC 第 12 脚
08	输出的数值为 0x80
09	调用 PWM 输出例程
13	PWM 输出例程
15	开始传送 IIC BUS 的起始信号
16~18	分别传送 slave 地址、第几个信道(subaddress)以及要输出的数值(bytedata)至 PWM IC 内
19	传送 IIC BUS 的停止信号

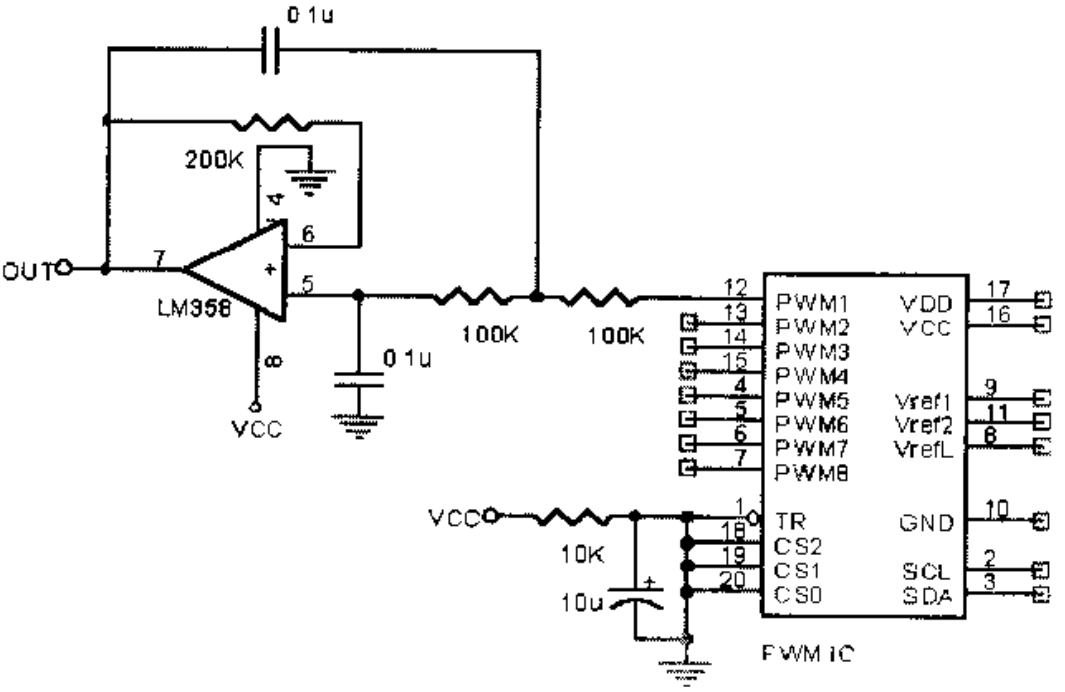
电路



说明

将 PWM 输出经过晶体管的反向输出以满足外围电路的需求，通过改变 Rt、Ct 可以限制直流电压，或者利用软件控制 PWM 输出的上限值和下限值。

电路



说明

R、C 充电将会有负载效应，即输出直流电压将会被外围电路拉低下来，如果用放大器高输入阻抗的特性并设计成滤波器的方式，即可解决负载效应，如上图电路所示。

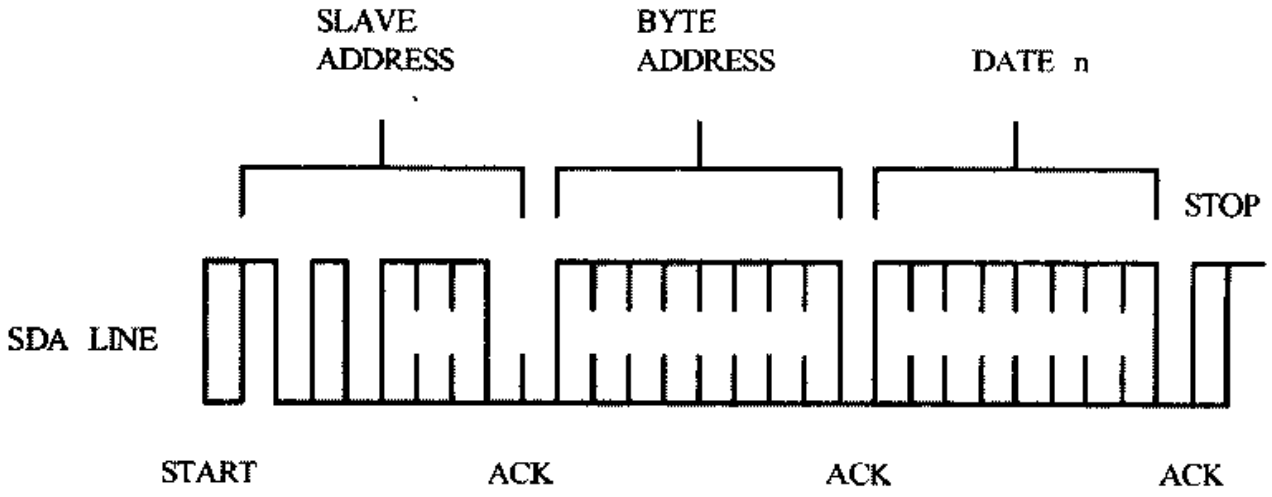
第 22 章　IC 24C08 的应用

➢ EEPROM 24c08 命令格式

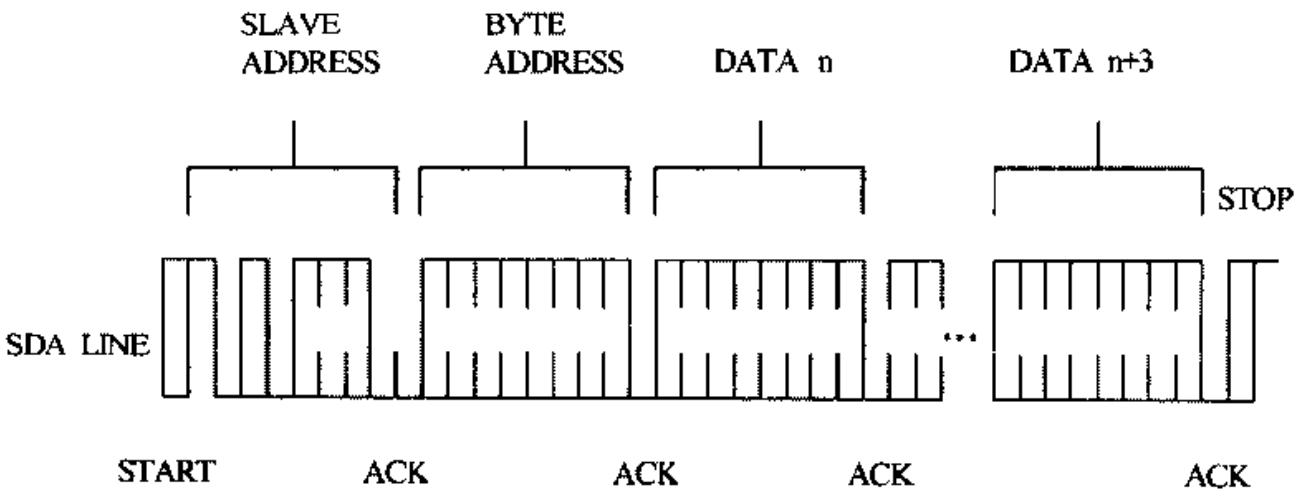
使用 EEPROM IC 最主要的目的是可以存储，如将上一次的状态全部存至 EEPROM 内，重新开机时，则可利用软件将 EEPROM 内的数据读出并立即执行相关动作。因为 EEPROM 24c08 内的数据并不会因为失去电源而破坏数据的完整性，其通信协议方式为 IIC BUS。

EEPROM 24c08 常用的命令格式，其时序图如下：

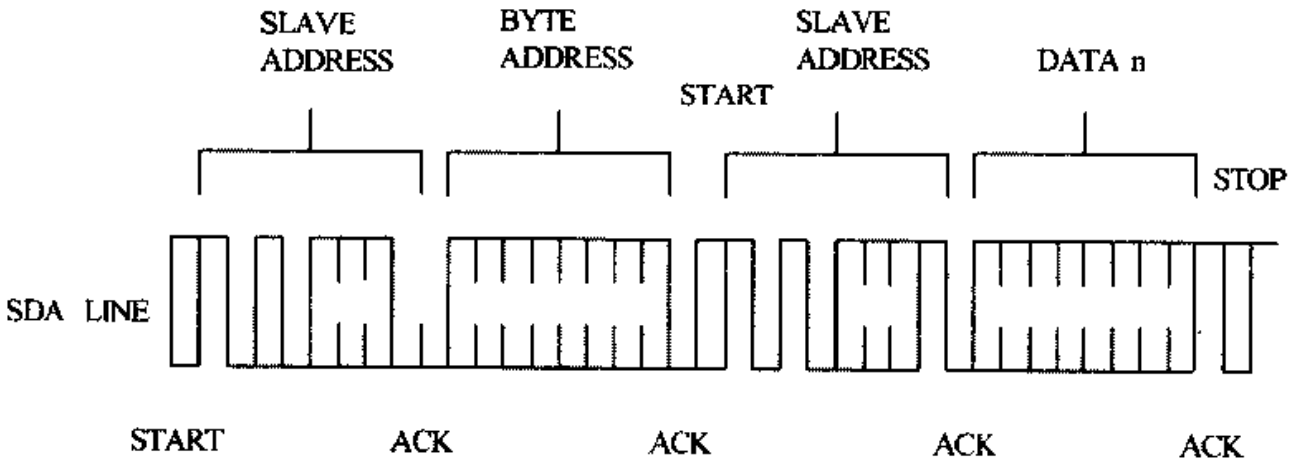
【Byte Write for Data】



【Page Write for Data】



【Random Read for Data】



其应用程序如下：

➢ EEPROMByteWrite0(bank,addr,value)

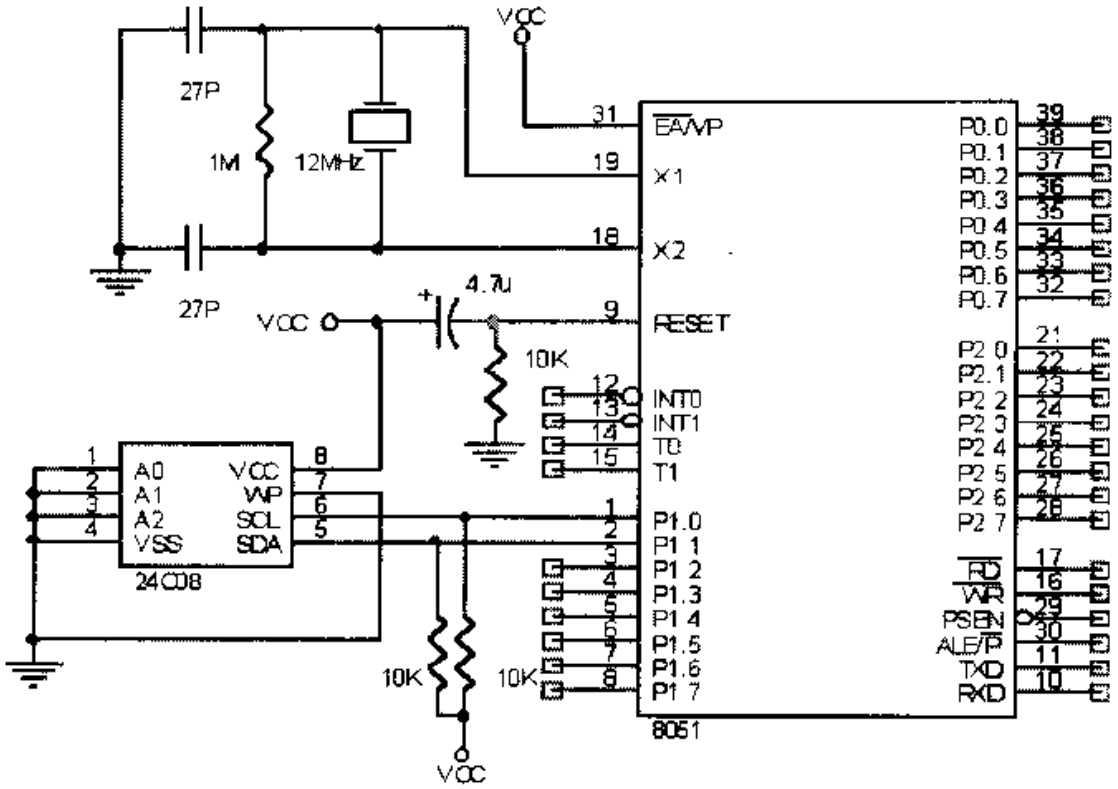
目的

将数据写入 EEPROM 24c08 的地址内(一)

参数

bank：页地址，范围从 0~3 即 bank0~bank3，有 4 页共 1024 Byte
addr：地址，范围从 0~255
value：要写入的数据，范围从 0~255

电路



程序

```
01 //Write byte data into EEPROM
02 void EEPROMByteWrite0(Byte bank,Byte addr,Byte value)
03 {
04     I2cByteWrite((0xA0|bank),addr,value);
05 }
06
07 //Write word data into EEPROM
08 void EEPROMWordWrite0(Byte bank,Byte addr,Word value)
09 {
10     I2cByteWrite((0xA0|bank),addr  ,value/256);
                                  // hi-byte
11     I2cByteWrite((0xA0|bank),addr+1,value-(value/256)*256);
                                  //low-byte
12 }
```

行 号	说　　明
04	将数据写入指定的 bank 和地址内，其中 I2cByteWrite( )例程请参考第二十章 IIC BUS 的应用
10	将 Word 数据的高字节写入地址内
11	将 Word 数据的低字节写入“地址+1”内

➢ EEPROMByteRead0(bank,addr)

目的

将地址内的数据读出来( )

参数

bank：即 bank0~bank3，有 4 页共 1024 Byte
addr：地址，范围从 0~255

程序

```
01 //Read byte data from EEPROM
02 Byte EEPROMByteRead0(Byte bank,Byte addr)
03 {
04    return ( I2cByteRead((0xA0|bank),addr) );
05 }
06
07 //Read word data from EEPROM
08 Word EEPROMWordRead0(Byte bank,Byte addr)
09 {
10  return
11  ( I2cByteRead((0xA0|bank),addr)*256+I2cByteRead
                        ((0xA0|bank),addr+1) );
12 }
```

行 号	说　　明
04	将指定 bank 和地址的数据读出来，并回复读出值至原调用程序，其中 I2cByteRead( )例程请参考第二十章 IIC BUS 的应用
10,11	将指定 bank 和地址的数据读出来当作是高位，而指定 bank 和下一个地址的数据也读出来当作是低位，并组合成 Word 数据回复至原调用程序

➢ EEPROMByteWrite(addr,bytedata)

目的

将数据写入EEPROM 24c08的地址内(二)

参数

addr：地址，范围从0~1023，由地址来换算相对应的bank
bytedata：要写入的数据，范围从0~255

程序

```
01 void EEPROMByteWrite(Word addr,Byte bytedata)
02 {
03     TrmBuf[ 1] = bytedata;
04     EEPROMWrite( addr,1 );
05 }
```

行 号	说 明
03	将数据存入数组变量内
04	依据数组变量将数据写入EEPROM的地址内，仅写入一个地址，其中EEPROMWrite( )函数有判断地址参数而将数据写入相对应的bank内

➢ EEPROMByteRead(addr)

目的

将地址内的数据读出来(二)

参数

addr：地址，范围从0~1023，由地址来换算相对应的bank

程序

```
01 Byte EEPROMByteRead(Word addr)
02 {
03     EEPROMRead( addr,1 );
04     return (TrmBuf[0]);
```

```
05 }
```

行 号	说　明
03	将指定地址内的数据读出来，而且仅读出一个地址的数据并将读出来的数据存入数组内，EEPROMRead( )函数会将各自读出的数据分别依次存入数组内
04	将数组变量 TrmBuf[0] 的内容，返回到原调用程序

➢ EEPROMPageWrite( )

目的

将数组变量 TrmBuf[ ] 内的一串数据，从指定的地址依次写入

程序

```
01 void EEPROMPageWrite(void)
02 {
03     TrmBuf[ 1] = Variable1;
04     TrmBuf[ 2] = Variable2;
05     TrmBuf[ 3] = Variable3;
06     TrmBuf[ 4] = Variable4;
07     TrmBuf[ 5] = Variable5;
08     TrmBuf[ 6] = Variable6;
09     TrmBuf[ 7] = Variable7;
10     TrmBuf[ 8] = (Byte)FgVariable1;
11     TrmBuf[ 9] = (Byte)FgVariable2;
12     TrmBuf[10] = (Byte)FgVariable3;
13
14     EEPROMWrite( START_ADDRESS,10 );
15 }
```

行 号	说　明
03~12	将变量值依次存入 TrmBuf[ ] 数组内，下标从 1 开始
14	从 START_ADDRESS 常数的地址连续写入 10 个数据，即将 TrmBuf[1]的内容写入 START_ADDRESS 常数的地址，而 TrmBuf[2] 的内容写入 START_ADDRESS+1 的地址，依此类推

➢ EEPROMPageRead( )

目的

从指定的地址连续读出一串数据，并依次存入变量内

程序

```
01 void EEPROMPageRead(void)
02 {
03    EEPROMRead( START_ADDRESS,10 );
04
05    Variable1  = TrmBuf[ 0];
06    Variable2  = TrmBuf[ 1];
07    Variable3  = TrmBuf[ 2];
08    Variable4  = TrmBuf[ 3];
09    Variable5  = TrmBuf[ 4];
10    Variable6  = TrmBuf[ 5];
11    Variable7  = TrmBuf[ 6];
12    FgVariable1= (Bool)TrmBuf[7];
13    FgVariable2= (Bool)TrmBuf[8];
14    FgVariable3= (Bool)TrmBuf[9];
15 }
```

行 号	说 明
03	从 START_ADDRESS 常数的地址连续读出 10 个数据，也就是将 START_ADDRESS+0～START_ADDRESS+9 地址内的数据读出来，并存入数组变量 TrmBuf[0]～TrmBuf[9]内
05~11	再将数组变量内的数据存入至指定的变量
12~14	Byte 数组变量作了 Bool 型态转换再存入标志变量

➢ EEPROMWrite(subaddress,count)

目的

将数组变量 TrmBuf[ ]内的一串数据连续写入地址内

参数

subaddress：地址，范围从 0~1023，由地址来换算相对应的 bank
count：连续写入数据的个数

程序

```
01 void EEPROMWrite(Word subaddress, Byte count)
02 {
03    Byte i;
04
05    if ( subaddress < 256 )
06    {
07       SlvAdr = 0xA0;            //bank 0,256byte
08    }
09    else if ( subaddress < 512 )
10    {
11       SlvAdr = 0xA2;            //bank 1
12       subaddress = subaddress - 256;
13    }
14    else if ( subaddress < 768 )
15    {
16       SlvAdr = 0xA4;            //bank 2
17       subaddress = subaddress - 512;
18    }
19    else
20    {
21       SlvAdr = 0xA6;            //bank 3
22       subaddress = subaddress - 768;
23    }
24
25    for( i=1; i<=count; i++)
26    {
27       TrmBuf[0] = (Byte)subaddress;
28       TrmBuf[1] = TrmBuf[i];
29       ByteCnt = 2;
```

```
30        SendEEPROMData( );
31        subaddress++;
32     }
33
34     DelayXlms(3);
35  }
```

行 号	说 明
05~08	当地址介于 0~255 就写入 bank0
09~13	当地址介于 256~511 就写入 bank1，并将地址减去 256 作为 bank1 的实际地址
14~18	当地址介于 512~767 就写入 bank2，并将地址减去 512 作为 bank2 的实际地址
19~23	当地址介于 768~1023 就写入 bank3，并将地址减去 768 作为 bank3 的实际地址
25	从数组 1 的内容开始写入，并连续写入参数 count 的个数
27~31	将数据写入指定的地址内，一次只做一个地址的写入动作，（如果连续写入多个数据要注意跨列的限制，即 1 页(bank)有 16 列，1 列有 16 Byte，总共 256 Byte，例如：从第 1 列的地址 0x0a 连续写入 8 个数据，则在第 2 列的地址 0x00、0x01 将无法正确的写入数据，而会写入至同列(第 1 列)的地址 0x00、0x01 内而造成错误，因此一般以每次写入一个地址就不会造成上述问题，即所谓的跨列写入）写入完成后地址就加 1，以便下一个数组变量内的数据可以写入下一个地址
34	延迟 3mS

➢ EEPROMRead(subaddress,count)

目的

从指定地址连续读出一串数据并存入至数组变量 TrmBuf[ ] 内

参数

subaddress：地址，范围从 0~1023，由地址来换算相对应的 bank

count：连续读出数据的个数

程序

```
01 void EEPROMRead(Word subaddress,Byte count)
02 {
03    if ( subaddress < 256 )
04    {
05       SlvAdr = 0xA0;            //bank 0,256byte
06    }
07    else if ( subaddress < 512 )
08    {
09       SlvAdr = 0xA2;            //bank 1
10       subaddress = subaddress - 256;
11    }
12    else if ( subaddress < 768 )
13    {
14       SlvAdr = 0xA4;            //bank 2
15       subaddress = subaddress - 512;
16    }
17    else
18    {
19       SlvAdr = 0xA6;            //bank 3
20       subaddress = subaddress - 768;
21    }
22
23    TrmBuf[0] = (Byte)subaddress;
24    ByteCnt   = 0x01;
25    SendData( );
26    ByteCnt   = count;
27    RcvData( );
28 }
```

行 号	说　明
03~06	当地址介于 0~255 就从 bank0 读出
07~11	当地址介于 256~511 就将地址减去 256 后当成实际地址，并从 bank1 读出
12~16	当地址介于 512~767 就将地址减去 512 后当成实际地址，并从 bank2 读出

行 号	说　明
17~21	当地址介于 768~1023 就将地址减去 768 后当成实际地址，并从 bank3 读出
23	先传送读取的起始地址
24	传送起始地址，因此只有一个字节
25	传送例程
26~27	连续读出 count 个数的数据，并存入至数组变量 TrmBuf[ ]内

➢ SendEEPROMData( )

目的

将数组变量 TrmBuf[ ] 内的一串数据传送至 EEPROM 内

程序

```
01 Bool SendEEPROMData(void)
02 {
03    Byte  i,j;
04    Bool  no_ack = 1;
05
06    for( i=0; i<10; i++)              //retry 10 times
07    {
08       if ( GoMaster(SlvAdr) != SUCCESS )
09       {
10          SendStop( );
11          DelayX1ms (10);            //delay 10mS
12          continue;
13       }
14       no_ack = 0;
15       for(j=0; j<ByteCnt; j++)
16       {
17          if ( SendByte( TrmBuf[j] ) != SUCCESS )
18          {
19             no_ack = 1;
```

```
20              break;
21            }
22        }
23        SendStop( );
24        if ( no_ack==0 )  break;
25        DelayX1ms(10);              //delay 10mS
26    }
27    return  no_ack;
28 }
```

行 号	说　明
06	IIC BUS 在传输过程有错误时则重试 10 次，即作 timeout 超过时间的处理，也就是当 IIC BUS 被干扰严重或 IIC BUS 组件没有连接，程序也不会死掉，也可避免因瞬间干扰而造成 IIC BUS 无法正常运作
08~13	检测传送起始信号是否正确，如果传送失败则传送停止信号并延迟 10mS 后再重传，直到重试 10 次为止
14	初始“确认信号”为正常
15~22	连续传送几个字节，每传送一个数据就验证是否传送成功，如果传送失败则设定“确认信号”为失败，并跳离 for 循环，继续做行号 23、25 即传送停止信号并延迟 10mS 后再重传，直到重试 10 次为止
23	传送成功或传送失败都必须作传送停止信号的动作
24	如果传送成功则退出重试 10 次的循环
25	如果传送失败则延迟 10mS 后再重传，直到重试 10 次为止
27	回复标志变量(no_ack)至原调用程序

➢ SendData( )

目的

将数组变量 TrmBuf[ ] 内的一串数据传送至 IIC BUS 的组件内

程序

```
01 Bool SendData(void)
02 {
```

```
03    Byte  i,j;
04    Bool  no_ack = 1;
05
06    for( i=0; i<10; i++)    //time out,retry 10 times
07    {
08      if ( GoMaster(SlvAdr) != SUCCESS )
09      {
10        SendStop( );
11        continue;
12      }
13      no_ack = 0;
14      for(j=0; j<ByteCnt; j++)
15      {
16        if ( SendByte( TrmBuf[j] ) != SUCCESS )
17        {
18          no_ack = 1;
19          break;
20        }
21      }
22      SendStop( );
23      if ( no_ack==0 )  break;
24    }
25    return  no_ack;
26 }
```

说明

本范例和上一个范例 SendEEPROMData( )例程大致一样，不再赘述，需注意的是 SendEEPROMData( )例程，每写入 EEPROM 一次则至少需延迟 10mS 以上，而本程序可适用其它非 EEPROM 的 IIC BUS 的组件。

➢ RcvData( )

目的

从 IIC BUS 的组件读取一串数据至数组变量 TrmBuf[ ]内

程序

```
01 Bool RcvData(void)
02 {
03    Byte  i,j,bval;
04
05    if ( GoMaster(SlvAdr+1) != SUCCESS )
06    {
07       SendStop( );
08       return  FAILURE;
09    }
10
11    for(i=0; i<ByteCnt; i++)
12    {
13       bval = 0;
14       for(j=0; j<8; j++)
15       {
16          SCL_PIN = 1;
17          I2cDelay( );
18
19          bval = bval << 1;
20          if ( SDA_PIN )  bval |= 0x01;
21
22          SCL_PIN = 0;
23          I2cDelay( );
24       }
25       if ( i==(ByteCnt-1) )    SDA_PIN = 1;
26       else                     SDA_PIN = 0;
27
28       I2cDelay( );
29
30       SCL_PIN = 1;
31       I2cDelay( );
32
33       SCL_PIN = 0;
```

```
34        I2cDelay( );
35
36        SDA_PIN = 1;
37        I2cDelay( );
38
39        TrmBuf[i] = bval;
40    }
41
42    SendStop( );
43
44    return  SUCCESS;
45 }
```

行 号	说　明
05~09	如果传送 IIC BUS 的“起始信号”和“slave+1 的地址信号”造成错误，则传送“停止信号”后，须将“传送失败(FAILURE) ”回复至原调用程序
11	连续接收数据存入数组变量 TrmBuf[ ]内
13	接收数据变量的初始值为 0
14~24	接收 8 位，SCL 和 SDA 的时序
25,26	如果已经接收最后一笔数据，则确认信号送“1”电平即 SDA="1"，否则确认信号送“0”电平即 SDA="0"
28~37	SCL 和 SDA 的时序
39	将接收到的数据依次存入数组变量 TrmBuf[ ]内
42	全部的数据个数都接收完毕则传送停止信号
44	回复“成功信号(SUCCESS)”至原调用程序

➢ GoMaster(slaveaddr)

目的

传送起始信号和 slave 地址

参数

slaveaddr：IIC BUS 组件的 slave 地址

程序

```
01 Bool GoMaster(Byte slaveaddr)
02 {
03     if ( (SCL_PIN==0) || (SDA_PIN==0) )
04        return FAILURE;
05
06     SDA_PIN = 0;
07     I2cDelay( );
08
09     SCL_PIN = 0;
10     I2cDelay( );
11
12     SendByte(slaveaddr);
13
14     return SUCCESS;
15 }
```

行 号	说　明
03,04	如果 SCL 或 SDA 任一信号有“0”电平存在，表示有错误或在使用中，则回复“失败信号(FAILURE)”至原调用程序，正常 SCL 和 SDA 信号应该保持在“1” 电平，而且两者都需要接提升电阻至+5V
06,07	SDA 信号先拉至“0”电平，并持续数 uS 的时间
09,10	SCL 信号再拉至“0”电平，也持续数 uS 的时间
12	传送 slave 的地址
14	回复“成功信号(SUCCESS)”至原调用程序

➢ SendByte(value)

目的

传送一个字节

参数

value：要传送的数值数据

程序

```
01 Bool SendByte(Byte value)
02 {
03    Byte  i;
04    Bool  no_ack = 0;
05
06    for(i=0; i<8; i++)
07    {
08       if ( value & 0x80 )  SDA_PIN = 1;
09       else                 SDA_PIN = 0;
10
11       value <<= 1;
12
13       I2cDelay( );
14       SCL_PIN = 1;
15       I2cDelay( );
16       SCL_PIN = 0;
17       I2cDelay( );
18    }
19
20    SDA_PIN = 1;
21    I2cDelay( );
22    SCL_PIN = 1;
23    I2cDelay( );
24
25    if ( SDA_PIN == 1 )  no_ack = 1;
26
27    SCL_PIN = 0;
28    I2cDelay( );
29
30    return  no_ack;
```

```
31 }
```

行 号	说　明
06	传送 8 位，因此循环作 8 次
08,09	从最高位开始传送，如果此位等于“1”则 SDA 信号送“1”电平，此位等于“0”则 SDA 信号送“0”电平
11	因为是从最高位开始传送，所以传送 1 位后则往左移 1 位
13~17	SCL 作“1”->“0”的变化
20,21	8 位全部传送完毕后则 SDA 先设定为“1”电平，并持续数 uS 的时间
22,23	SCL 也设定为“1”电平，并持续数 Us 的时间
25	在 SDA 上的信号即为“确认信号”，如果 SDA=“1”则表示“无确认信号”，即设定 no_ack = 1
27,28	SCL 再清除为“0”电平，并持续数 uS 的时间
30	回复“无确认信号的标志(no_ack)”至原调用程序，其中 no_ack=“0”表示有确认信号，no_ack=“1”表示无确认信号

➢ SendStop( )

目的

传送停止信号

程序

```
01 void SendStop(void)
02 {
03     SDA_PIN = 0;
04     SCL_PIN = 0;
05     I2cDelay( );
06
07     SCL_PIN = 1;
08     I2cDelay( );
09
10     SDA_PIN = 1;
11     I2cDelay( );
```

```
12 }
13
14 void I2cDelay(void)
15 {
16     _nop_( );
17     _nop_( );
18     _nop_( );
19     _nop_( );
20 }
```

行 号	说　明
03~05	SDA 和 SCL 信号都清除为“0”电平，并持续数 uS 的时间
07,08	SCL 先设定“1”电平，并持续数 uS 的时间
10,11	SDA 才能设定“1”电平，并持续数 uS 的时间
14	IIC BUS 延迟数 uS 的时间
16~19	如同 8051 组合语言的 NOP 指令

➢ DdcChecksum(adr)

目的

作校验的应用

参数

adr：索引地址，范围从 0~128

程序

```
01 Byte DdcChecksum(Byte adr)
02 {
03     Byte  i,temp=0;
04
05     for( i=adr; i<adr+127; i++) //total 128 bytes
06     {
07        EEPROMRead(i,1); //set adr,read 1 byte to  TrmBuf[0]
08        temp+=TrmBuf[0];
```

```
09      }
10      temp=(~temp)+1;      //Checksum =0xff-(total sum)+1
11
12      TrmBuf[ 1] = temp;  //last byte is Checksum byte
13      EEPROMWrite( adr+127,1 );
14      return temp;
15  }
```

行 号	说　明
03	初始 LOCAL 变量
05	从索引地址开始连续做 127 次
07,08	从 EEPROM 内的地址一次读出一个数据，并将数据一直累加起来
10	127 个数据最后累加的结果取反相后再加 1，并存入 temp 变量
12,13	将 temp 变量写入至 EEPROM 24c08 从索引地址开始的第 128 个的地址内
14	也回复此校验的数值至原调用程序

第 23 章　IC 24C32 的应用

➢ EEPROM24c32WriteByte_1(addr,value)

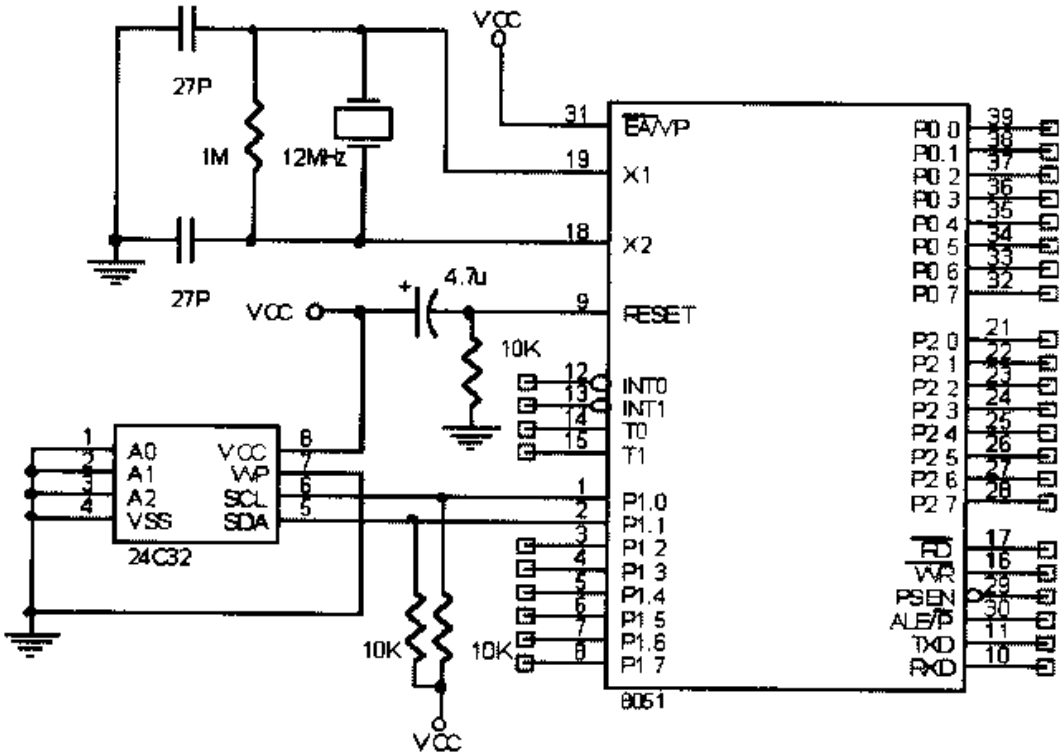
目的

将数据写入至 EEPROM 24c32 内

参数

addr：0~4096 的地址
value：要写入的数据

电路



程序

```
01 //24c32 x 1
```

```
02 void EEPROM24c32WriteByte_1(Word addr,Byte value)
03 {
04      I2cStart( );
05      I2cSentByte(0xA0);
06
07      //Send Address
08      I2cSentByte (addr >> 8);   // Send HIBYTE address
09      I2cSentByte (addr & 0xff);// Send LOBYTE address
10      I2cSentByte (value);
11      I2cStop( );
12 }
```

说明

当有大量的数据要作储存，则 EEPROM 24c32 可以达到需求，EEPROM 24c32 共有 32K bit、4K Byte 共有 16 页的数据，因此需要使用两个 Byte 来表示地址，也就是先设定地址再将数据写入或读出，其格式不同于 EEPROM 24c04、24c08 或 24c16，特别在此章节作范例说明。

行　号	说　　明
04	开始传送起始信号
05	传送一个字节代表 slave 地址，其地址等于 0xA0
08,09	先传送地址高字节，再传送地址低字节，即完成寻址
10	再将数据写入所指定的地址内
15	传送停止信号

➢ EEPROM24c32WriteByte_2(addr,value)

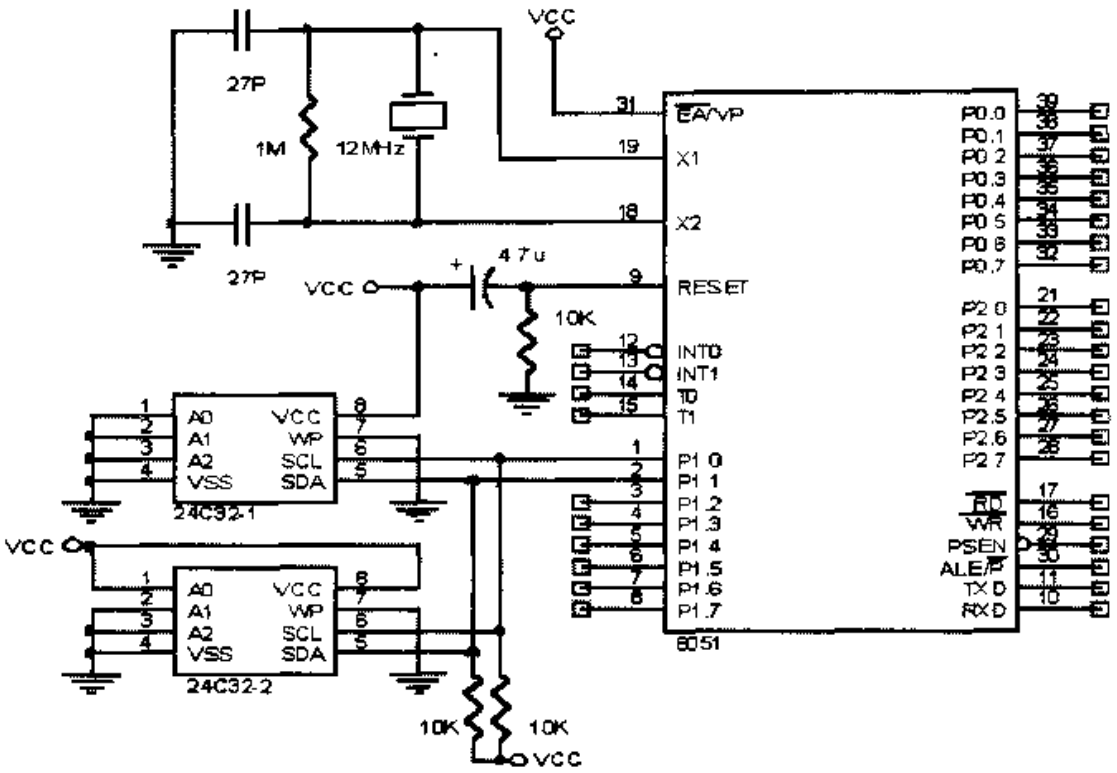
目的

将数据写入二个 EEPROM 24c32 内

参数

addr：0~0x1000 为第一个 24c32；0x8000~0x9000 为第二个 24c32
value：要写入的数据

电路



程序

```
01 //24c32 x 2
02 void EEPROM24c32WriteByte_2(Word addr,Byte value)
03 {
04     I2cStart( );
05
06     if (addr < 0x8000)
07        I2cSentByte(0xA0);
08     else
09        I2cSentByte(0xA2);
10
11     //Send Address
12     I2cSentByte ((Byte) ((addr >> 8) & 0x7f));
                    // Send HIBYTE address
13     I2cSentByte ((Byte) addr);// Send LOBYTE address
14     I2cSentByte (value);
15     I2cStop( );
16 }
```

行　号	说　　明
04	开始传送起始信号
06~09	如果地址小于 0x8000 则将数据写入第一个 EEPROM，其 slave 地址等于 0xA0；当地址大于或等于 0x8000 则将数据写入第二个 EEPROM，其 slave 地址等于 0xA2
12,13	要屏蔽最高位即 bit7 而换算成实际要写入的地址，先写入高字节的地址再写入低字节的地址
14	再将数据写入所指定的地址内
15	传送停止信号

➢ EEPROM24c32WriteMulti_1(addr,count)

目的

将数组变量 TrmBuf[ ] 内的一串数据，写入 EEPROM 24c32 内

参数

addr：0~4096 的地址
count：要写入数据的个数

程序

```
01 //24c32 x 1
02 void EEPROM24c32WriteMulti_1(Word addr,Byte count)
03 {
04     Byte i;
05
06     I2cStart( );
07     I2cSentByte(0xA0);
08
09     //Send Address
10     I2cSentByte (addr >> 8);  // Send HIBYTE address
11     I2cSentByte (addr & 0xff);// Send LOBYTE address
12
13     for(i=0;i<COUNT;I++)< pre>
14        I2cSentByte (TrmBuf[i]);
15     I2cStop( );
```

```
16 }
```

行 号	说 明
06	开始传送起始信号
07	传送一个字节代表着 slave 地址，其地址等于 0xA0
10	先传送高字节的地址
11	再传送低字节的地址
13,14	连续传送数据，即将数组变量 TrmBuf[ ] 内的数据依序写入地址内
15	传送停止信号

➢ EEPROM24c32WriteMulti_2(addr,count)

目的

将数组变量 TrmBuf[ ] 内的一串数据，写入二个 EEPROM 24c32 内

参数

addr：0~0x1000 为第一个 24c32；0x8000~0x9000 为第二个 24c32
value：要写入的数据

程序

```
01 //24c32 x 2
02 void EEPROM24c32WriteMulti_2(Word addr,Byte count)
03 {
04     Byte i;
05
06     I2cStart( );
07     if (addr < 0x8000)
08        I2cSentByte(0xA0);
09     else
10        I2cSentByte(0xA2);
11     //Send Address
12     I2cSentByte ((Byte) ((addr >> 8) & 0x7f));
```

```
13      I2cSentByte ((Byte) addr);
14      for(i=0;i<COUNT;I++)< pre>
15         I2cSentByte (TrmBuf[i]);
16      I2cStop( );
17  }
```

行　号	说　　明
06	开始传送起始信号
07~10	如果地址小于 0x8000 则将数据写入至第一个 EEPROM，其 slave 地址等于 0xA0；当地址大于或等于 0x8000 则将数据写入至第二个 EEPROM，其 slave 地址等于 0xA2
12	先传送高字节的地址
13	再传送低字节的地址
14~16	将数组变量 TrmBuf[ ] 内的数据依序写入至地址内并传送停止信号

➢ EEPROM24c32ReadByte_1(addr)

目的

将 EEPROM 24c32 内的地址的数据读出来

参数

addr：从指定的地址读出数据，范围从 0~0x1000

程序

```
01 //24c32 x 1
02 Byte EEPROM24c32ReadByte_1(Word addr)
03 {
04      Byte temp;
05
06      I2cStart( );
07      I2cSentByte(0xA0);
08      I2cSentByte (addr >> 8);  // Send HIBYTE address
09      I2cSentByte (addr & 0xff);// Send LOBYTE address
10
```

```
11      I2cStart( );
12      I2cSentByte(0xA1);
13      temp = I2cReceiveByte( );
14      SendAcknowledge(1);
15      I2cStop( );
16      return temp;
17  }
```

行 号	说 明
06	开始传送起始信号
07	传送一个字节代表着 slave 地址，其地址等于 0xA0
08,09	先传送高字节的地址，再传送低字节的地址
11	再一次传送起始信号
12	传送 0xA1 的地址，即“读出”的 slave 地址
13	接收从 EEPROM 读出来的数据，并暂存至 temp 变量
14	传送确认信号等于“1”电位
15	传送停止信号
16	回复读出值 temp 至原调用程序

➢ EEPROM24c32ReadByte_2(addr)

目的

将二个 EEPROM 24c32 内的地址的数据读出来

参数

addr：第一个 EEPROM 的地址，范围从 0~0x1000
第二个 EEPROM 的地址，范围从 0x8000~0x9000

程序

```
01 //24c32 x 2
02 Byte EEPROM24c32ReadByte_2(Word addr)
03 {
04      Byte temp;
05
```

```
06      //Reading from the second EEPROM if MSB is set. (Write
      Address)
07      //Otherwise reading from the first. (Write Address)
08      if (addr & 0x8000) temp = 0xA2;
09      else   temp = 0xA0;
10
11      I2cStart( );
12      I2cSentByte(temp);
13      I2cSentByte( (Byte) ((addr >> 8) & 0x7f) );
14      I2cSentByte( (Byte) addr );
15
16      I2cStart( );
17      I2cSentByte(temp|0x01);
18      temp = I2cReceiveByte( );
19      SendAcknowledge(1);
20      I2cStop( );
21      return temp;
22  }
```

行　号	说　　明
08~09	如果地址大于或等于 0x8000 表示要从第二个 EEPROM 读出数据，其 slave 地址等于 0xA2；否则从第一个 EEPROM 读出数据，其 slave 地址等于 0xA0
11	开始传送起始信号
12	依据 temp 变量来传送 slave 地址，即选择第一个 EEPROM 或第二个 EEPROM
13,14	先传送高字节的地址，再传送低字节的地址，即寻址完成
16	再一次传送起始信号
17	传送“读出”数据即 slave 地址加 1，如果是第一个 EEPROM 此地址是等于 0xA1；如果是第二个 EEPROM 此地址是等于 0xA3
18	接收从 EEPROM 读出来的数据，并暂存至 temp 变量
19	传送确认信号等于“1”电位
20	传送停止信号
21	回复读出值 temp 至原调用程序

➢ EEPROM24c32ReadWord_1(addr)

目的

将 EEPROM 24c32 内的地址的 Word 数据读出来

参数

addr：从指定的地址读出数据，范围从 0~0x1000

程序

```
01 //24c32 x 1
02 Word EEPROM24c32ReadWord_1(Word addr)
03 {
04     Byte temp;
05     Word value;
06
07     I2cStart( );
08     I2cSentByte(0xA0);
09     I2cSentByte (addr >> 8);  // Send HIBYTE address
10     I2cSentByte (addr & 0xff);// Send LOBYTE address
11
12     I2cStart( );
13     I2cSentByte(0xA1);
14     value = I2cReceiveByte( );// hi-byte
15     SendAcknowledge(0);
16     value <<= 8;
17
18     temp = I2cReceiveByte( ); //low-byte
19     value |= temp;
20     SendAcknowledge(1);
21     I2cStop( );
22     return value;
23 }
```

行 号	说 明
07	开始传送起始信号
08	传送一个字节代表着 slave 地址，其地址等于 0xA0

行　号	说　　明
09,10	先传送高字节的地址，再传送低字节的地址，即寻址完成
12	再一次传送起始信号
13	传送“读出”地址等于 0xA1
14	读出指定地址的数据，作为高字节
15	传送确认信号为 “0” 电位
16	移入高字节
18,19	再读出下一地址的数据即指定地址加 1，作为低字节，高、低字节将之合起来即为 Word 数据
20	传送确认信号等于“1” 电位
21	传送停止信号
22	回复读出值 value 至原调用程序

➢ EEPROM24c32ReadWord_2(addr)

目的

将二个 EEPROM 24c32 内的地址的 Word 数据读出来

参数

addr：第一个 EEPROM 的地址，范围从 0~0x1000
　　第二个 EEPROM 的地址，范围从 0x8000~0x9000

程序

```
01 //24c32 x 2
02 Word EEPROM24c32ReadWord_2(Word addr)
03 {
04     Byte temp;
05     Word value;
06
07     I2cStart( );
08
09     if (addr < 0x8000)
10        I2cSentByte(0xA0);
```

```
11      else
12         I2cSentByte(0xA2);
13
14      I2cSentByte( (Byte) ((addr >> 8) & 0x7f) );
15      I2cSentByte( (Byte) addr );
16
17      I2cStart( );
18      if (addr < 0x8000)
19         I2cSentByte(0xA0|0x01);
20      else
21         I2cSentByte(0xA2|0x01);
22
23      value = I2cReceiveByte( );
24      SendAcknowledge(0);
25      value <<= 8;
26
27      temp = I2cReceiveByte( );
28      value |= temp;
29      SendAcknowledge(1);
30      I2cStop( );
31      return value;
32  }
```

行 号	说 明
07	开始传送起始信号
09~12	如果地址小于 0x8000 则传送 0xA0 的地址；否则传送 0xA2 的地址
14,15	传送指定的地址
17	再一次传送起始信号
18~21	如果地址小于 0x8000 则送出读出地址等于 0xA1，表示从第一个 EEPROM 内读出数据；否则送出读出地址等于 0xA3，表示从第二个 EEPROM 内读出数据
23	读出指定地址的数据，作为高字节
24	传送确认信号为“0”电位
25	移入高字节

行　号	说　　明
27,28	再读出下一地址的数据即指定地址加 1，作为低字节，高、低字节将之合起来即为 Word 数据
29	传送确认信号等于“1” 电位
30	传送停止信号
31	回复读出值 value 至原调用程序

➢ EEPROM24c32ReadMulti_1(addr,count)

目的

从 EEPROM 24c32 读出一串数据并存入数组变量 TrmBuf[ ] 内

参数

addr：地址，范围从 0~0x1000
count：连续读出数据的个数

程序

```
01 //24c32 x 1
02 void EEPROM24c32ReadMulti_1(Word addr,Byte count)
03 {
04     Byte i;
05
06     I2cStart( );
07     I2cSentByte(0xA0);
08     I2cSentByte (addr >> 8);  // Send HIBYTE address
09     I2cSentByte (addr & 0xff);// Send LOBYTE address
10
11     I2cStart( );
12     I2cSentByte(0xA1);
13
14     for(i=0;i<COUNT;I++)< pre>
15     {
16        TrmBuf[i] = I2cReceiveByte( );
17        if ( i != (count - 1) )  SendAcknowledge(0);
18     }
19     SendAcknowledge(1);
```

```
20      I2cStop( );
21  }
```

行 号	说 明
06	开始传送起始信号
07	传送一个字节代表着 slave 地址，其地址等于 0xA0
08,09	先传送高字节的地址，再传送低字节的地址
11	再一次传送起始信号
12	传送读出地址 0xA1
14	读出一串数据，总共有 count 个数
16	将读出的数据依序存入数组变量 TrmBuf[ ] 内
17	还没有读完最后一笔数据，则送出确认信号为“0”电位
19	如果已经全部读出则送出确认信号为“1”电位
20	传送停止信号

➢ EEPROM24c32ReadMulti_2(addr,count)

目的

从二个 EEPROM 24c32 读出一串数据并存入数组变量 TrmBuf[ ] 内

参数

addr： 第一个 EEPROM 的地址，范围从 0~0x1000
第二个 EEPROM 的地址，范围从 0x8000~0x9000
count： 连续读出数据的个数

程序

```
01 //24c32 x 2
02 void EEPROM24c32ReadMulti_2(Word addr,Byte count)
03 {
04      Byte i;
05
06      I2cStart( );
07
08      if (addr < 0x8000)
```

```
09          I2cSentByte(0xA0);
10      else
11          I2cSentByte(0xA2);
12      I2cSentByte( (Byte) ((addr >> 8) & 0x7f) );
13      I2cSentByte( (Byte) addr );
14
15      I2cStart( );
16      if (addr < 0x8000)
17          I2cSentByte(0xA0|0x01);
18      else
19          I2cSentByte(0xA2|0x01);
20
21      for(i=0;i<COUNT;I++)< pre>
22      {
23          TrmBuf[i] = I2cReceiveByte( );
24          if ( i != (count - 1) )  SendAcknowledge(0);
25      }
26      SendAcknowledge(1);
27      I2cStop( );
28  }
```

行　号	说　　明
06	开始传送起始信号
08~11	如果地址小于 0x8000 则传送 0xA0 的地址；否则传送 0xA2 的地址
12,13	传送指定的地址
15	再一次传送起始信号
16~19	如果地址小于 0x8000 则送出读出地址等于 0xA1，表示从第一个 EEPROM 内读出数据；否则送出读出地址等于 0xA3，表示从第二个 EEPROM 内读出数据
21	读出一串数据，总共有 count 个数
23	将读出的数据依序存入数组变量 TrmBuf[ ] 内
24	还没有读完最后一笔数据，则送出确认信号为“0”电位
26	如果已经全部读出则送出确认信号为“1”电位
27	传送停止信号

第 24 章　OSD IC 的应用

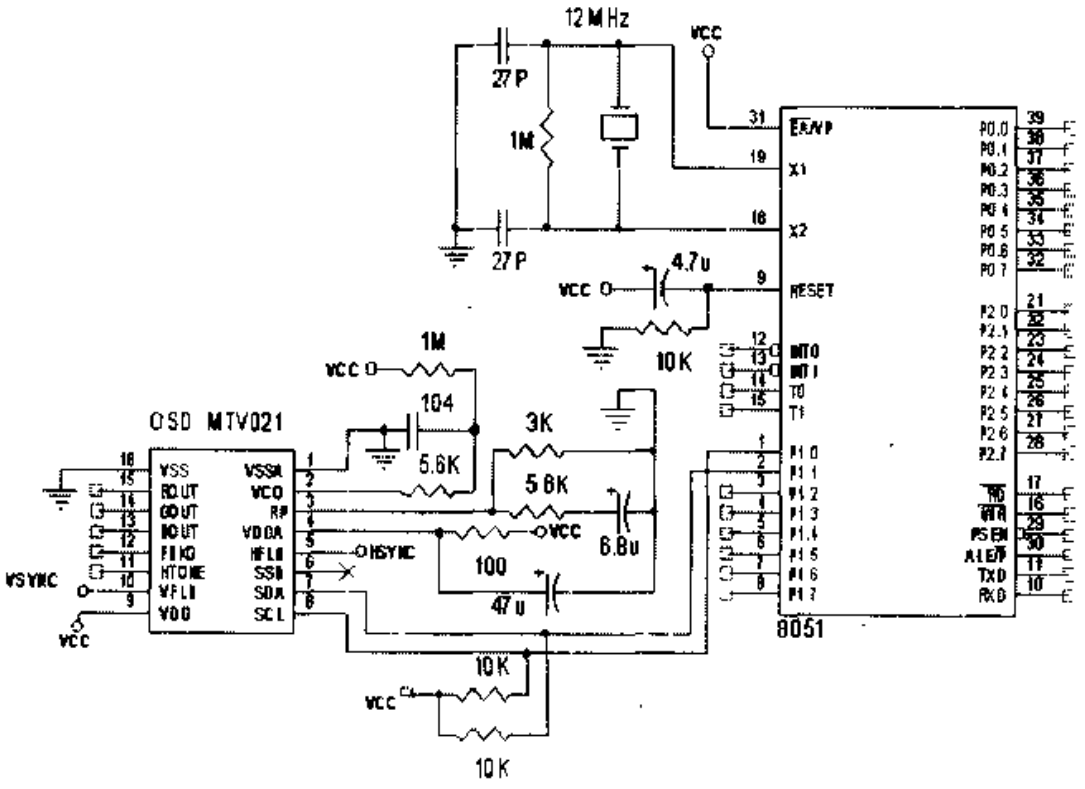
OSD 的全文为 On-Screen Display，也就是说在监视器上显示彩色字符的方式，市面上有许多 OSD IC，其通信协议的方式为 IIC BUS，一般应用在有屏幕装置的商品上，例如：PC monitor、LCD monitor、CCTV、TV、LCD TV 等，OSD IC 内部已经有规划好的内定字符(也可以依据须求而重新设计字符但须开光罩)，使用者必须在指定的“列”、“行”将字符显示出来并规划其排列格式，和配合按键操作而达到亲切的使用者接口，以下为常用 OSD IC 接口的应用程序。

➢ OsdStart(　)

目的

以 OSD IC 当作 slave，传送 IIC BUS 的起始信号

电路



程序

```
01 void OsdStart(void)
02 {
03     OSD_SDA=1;
04     OSD_SCL=1;
05     I2cWait( );
06     OSD_SDA=0;
07     I2cWait( );
08     OSD_SCL=0;
09 }
```

行　号	说　　明
03	SDA 设定为“1”电位
04	SCL 设定为“1”电位
05	延迟数 uS 的时间，配合 IIC BUS 的时序
06	SDA 必须先清除“0”电位
07	再一次延迟数 uS 的时间后
08	SCL 才可以清除“0”电位

➢ OsdStop( )

目的

传送停止信号

程序

```
01 void OsdStop(void)
02 {
03     OSD_SDA=0;
04     I2cWait( );
05     OSD_SCL=1;
06     I2cWait( );
07     OSD_SDA=1;
```

```
08 }
```

行　号	说　　明
03	SDA 清除“0”电位
04	延迟数 uS 的时间
05	SCL 必须先设定为“1”电位
06	延迟数 uS 的时间
07	SDA 才能再设定为“1”电位

➢ OsdSentByte(bytedata)

目的

传送一个字节(Byte)的数据

参数

bytedata：要传送的数据

程序

```
01 //master transfer data to slave,return acknowledge bit
02 Bool OsdSentByte(Byte bytedata)
03 {
04     Byte i;
05     Bool ack;
06
07     for(i=0; i<8; i++)
08     {
09       bytedata=_crol_(bytedata,1);
10
11       if ( bytedata & 0x01 )
12         OSD_SDA=1;
13       else
14         OSD_SDA=0;
15
```

```
16        OSD_SCL=1;
17        I2cWait( );
18        OSD_SCL=0;
19        I2cWait( );
20      }
21      OSD_SDA=1;
22      I2cWait( );
23      OSD_SCL=1;
24      I2cWait( );
25      ack=OSD_SDA;
26      OSD_SCL=0;
27      I2cWait( );
28      return ack;
29  }
```

行　号	说　　明
05	ack 为“确认标志变量”
07	要传送一个 Byte 数据，所以要作 8 次
09	向左移 1 位，即第 0 位会移至第 1 位，而第 7 位会移至第 0 位，依此类推，最后又存回原变量 bytedata 内，其中_crol_指令必须包含程序库 intrins.h 的文件
11~14	最高位已经移到最低位，因此要判断第 0 位，如果第 0 位等于“1”则设定 SDA="1"，否则清除 SDA="0"
16~19	SCL 作"1"->"0"的变化
21~24	SDA 先设定为“1”，SCL 也设定为“1”，以便确认信号出现
25	读取确认信号
26,27	将 SCL 恢复为“0”电位
28	回复确认信号(ack)至原调用程序

➢ OsdReceiveByte( )

目的

读取一个字节(Byte)的数据

程序

```
01 Byte  OsdReceiveByte(void)
02 {
03   Byte i;
04   Byte bytedata=0;
05
06   for(i=0; i<8; i++)
07
08      OSD_SCL=1;
09      I2cWait( );
10
11      bytedata <<= 1;
12      if ( OSD_SDA )  bytedata |= 0x01;
13
14      OSD_SCL=0;
15      I2cWait( );
16   }
17   return bytedata;
18 }
```

行　号	说　　明
04	初始化读取的数据变量
06	读取 8 个位元，循环作 8 次
08,09	SCL 先设定为“1”，并等待一段时间
11	先将数据变量左移一个位
12	此时，如果 SDA="1"则将接收到的数据加 1(bit0 设为“1”)，否则，当左移 1 位元就会将 bit0 自动清除为 0

行　号	说　明
14,15	SCL 恢复为“0”电位并等待一段时间
17	回复此读取到的数据至原调用程序

➢ OsdFormatA_0(row,col,value)

目的

在指定的列、行地址内写入数值数据(一)，没有作超时处理

参数

row：列地址
col：行地址
value：要写入的数据

程序

```
01 //do 1 times,and un-check acknowledge
02 void OsdFormatA_0(Byte row,Byte col,Byte value)
03 {
04     OsdStart( );
05     OsdSentByte(DEVICE_ADR);
06     OsdSentByte(row);
07     OsdSentByte(col);
08     OsdSentByte(value);
09     OsdStop( );
10 }
```

行　号	说　明
04	开始传送起始信号
05	传送 slave 的地址，一般 OSD IC 的 slave 地址为 0x7A
06	传送列地址
07	传送行地址
08	传送要写入的数据
09	最后再传送停止信号，而完成 IIC BUS 的通信流程

➢ OsdFormatA(row,col,value)

目的

在指定的行、列地址内写入数值数据(二)，有作超时处理

参数

row：列地址
col：行地址
value：要写入的数据

程序

```
01 //time out,and check acknowledge
02 void OsdFormatA(Byte row,Byte col,Byte value)
03 {
04     TrmBuf[0] = row;
05     TrmBuf[1] = col;
06     TrmBuf[2] = value;
07     SlvAdr  = DEVICE_ADR;
08     ByteCnt = 3;
09     SendData( );
10 }
```

行　号	说　　明
04~06	先将列、行地址及数值先存入数组变量 TrmBuf[ ] 内，再依序传送出去
07	传送 slave 的地址，一般 OSD IC 的 slave 地址为 0x7A
08	要传送 3 个字节，从数组 0 开始传送
09	传送常式，在 EEPROM 24c08 章节里有说明

➢ OsdFrameControl (vertd,hord,height, width, rowspace)

目的

设定 OSD 字符显示的参数

参数

vertd：显示 OSD 字符的垂直位置
hord：显示 OSD 字符的水平位置
height：OSD 字符的高度
width：OSD 字符的宽度
rowspace：OSD 字符“列”与“列”之间的距离

程序

```
01 void OsdFrameControl(Byte vertd,Byte hord,Byte height,Byte width,
Byte rowspace)
02
03 {
04     OsdFormatA(15|DISPLAY_ROW, 12, vertd);
05     OsdFormatA(15|DISPLAY_ROW, 13, hord);
06     OsdFormatA(15|DISPLAY_ROW, 14, height);
07     OsdFormatA(15|DISPLAY_ROW, 15, width);
08     OsdFormatA(15|DISPLAY_ROW, 16, rowspace);
09 }
```

行　号	说　　明
04	在第 15 列、第 12 行设定参数，即显示字符的垂直位置
05	在第 15 列、第 13 行设定参数，即显示字符的水平位置
06	在第 15 列、第 14 行设定参数，即显示字符的高度
07	在第 15 列、第 15 行设定参数，即显示字符的宽度
08	在第 15 列、第 16 行设定参数，即显示字符的列距

➢ OsdLocationSet(vertical,horizontal)

目的

设定 OSD 字符的垂直和水平位置

参数

vertical：显示 OSD 字符的垂直位置

horizontal：显示 OSD 字符的水平位置

程序

```
01 void OsdLocationSet(Byte vertical,Byte horizontal)
02 {
03     OsdFormatA(15|DISPLAY_ROW, 12, vertical);
04     OsdFormatA(15|DISPLAY_ROW, 13, horizontal);
05 }
```

行　号	说　　明
03	在第 15 列、第 12 行设定垂直位置的参数
04	在第 15 列、第 13 行设定水平位置的参数

➢ OsdRamClear( )

目的

清除 OSD 的所有字符和窗口，即设为空白

程序

```
01 void OsdRamClear(void)
02 {
03     OsdFormatA(15|ATTRIBUTE_ROW, 17, 0x06);
04     DelayX1ms(1);
05     OsdFormatA(15|ATTRIBUTE_ROW, 17, 0x00);
06 }
```

行　号	说　　明
03	第 15 列、第 17 行暂存器的第 1 位和第 2 位设定为“1”
04	延迟 1mS 的时间
05	第 15 列、第 17 行暂存器的第 1 位和第 2 位再恢复为“0”，就可以清除字符和窗口了

➢ OsdEnable(yes)

目的

使能或除能 OSD，即是否显示 OSD

参数

yes："1"表示可以显示 OSD，"0"表示无法显示 OSD

程序

```
01 void OsdEnable(Bool yes)
02 {
03    if (yes==1)
04       OsdFormatA(15|DISPLAY_ROW, 17, 0x80);
05    else
06       OsdFormatA(15|DISPLAY_ROW, 17, 0x00);
07 }
```

行　号	说　　明
03,04	如果引数 yes="1"则第 15 列、第 17 行暂存器的位元 7 设定为"1"，即可以显示 OSD
05,06	如果引数 yes="0"则第 15 列、第 17 行暂存器的位元 7 清除为"0"，即无法显示 OSD

➢ OsdOpenUp( )

目的

显示 OSD 时具有飞溅效果的功能

程序

```
01 void OsdOpenUp(void)
02 {
03    OsdFormatA(15|DISPLAY_ROW, 17, 0x80|0x10);
04 }
```

行 号	说 明
03	第15列、第17行暂存器的第7位和第4位设定为“1”，其中第7位为OSD显示是否、第4位为当OSD显示时具有飞溅效果

➢ OsdNormal( )

目的

OSD作正常显示，即不具有飞溅效果的功能

程序

```
01 void OsdNormal(void)
02 {
03     OsdFormatA(15|DISPLAY_ROW, 17, 0x80|0x00);
04 }
```

行 号	说 明
03	第15列、第17行暂存器的位元7设定为“1”，即使能OSD显示；但位元4清除为“0”，即不具有飞溅效果的功能

➢ OsdResetFont( )

目的

将OSD所有字符设定为正常宽度、高度及清除为空白

程序

```
01 void OsdResetFont(void)
02 {
03     Byte i,j;
04
05     //all character SPACE,BLACK
06     for( i=0; i<15; i++ )
07        for( j=0; j<30; j++ )
08           OsdPrintIcon(i,j,_SPACE,BLACK);          //_SPACE=space
font
```

```
09
10      //every char CHS,CWS=0
11      for(i=0; i<15; i++)
12        OsdFormatA(i|DISPLAY_ROW, 30, 0);
13  }
```

行　号	说　　明
06	第 0~14 列
07	每一列的第 0~29 行的字符
08	字符属性设定为黑色空白
11	第 0~14 列
12	每一列的字符宽度、高度为正常，即不要双倍的宽度和高度

➢ OsdClearRow(start,end,color)

目的

清除列上的每一个字符(· ·)

参数

start：起始列
end：结束列
color：颜色属性

程序

```
01 void OsdClearRow(Byte start,Byte end,Byte color)
02 {
03    Byte i,j;
04
05    for( i=start; i<=end; i++ )
06    {
07      TrmBuf[0]  = i|ATTRIBUTE_ROW;
08      TrmBuf[1]  = 0|ATTRIBUTE_COLUMN;
09      for( j=0; j<30; j++ )
```

```
10          TrmBuf[j+2] = color;
11        SlvAdr  = DEVICE_ADR;
12        ByteCnt = 32;
13        SendData( );
14
15        TrmBuf[0] = i|DISPLAY_ROW;
16        TrmBuf[1] = 0|DISPLAY_COLUMN;
17        for( j=0; j<30; j++ )
18          TrmBuf[j+2] = _SPACE;
19        SlvAdr  = DEVICE_ADR;
20        ByteCnt = 32;
21        SendData( );
22      }
23 }
```

行　号	说　　明
05	起始列至结束列上的字符皆清除为空白
07,08	设定属性列和属性行
09,10	设定每一字符的颜色属性
11~13	OSD 的 slave 地址等于 0x7A，总共要传送 32 个字节并依据数组变量 TrmBuf[ ] 内的数据，传送至 OSD 的暂存器内
15,16	设定显示列和显示行
17~21	列上每一字符写入空白的字符，并传送至 OSD 的暂存器内

➢ OsdClearRow1(start,end,color)

目的

清除列上的每一个字符(二)

参数

start：起始列
end：结束列
color：颜色属性

程序

```
01 void OsdClearRow1(Byte start,Byte end,Byte color)
02 {
03    Byte i,j;
04
05    for( i=start; i<=end; i++ )
06    {
07       OsdStart( );
08       OsdSentByte(DEVICE_ADR);
09       OsdSentByte(i|ATTRIBUTE_ROW);
10       OsdSentByte(0|ATTRIBUTE_COLUMN);
11       for( j=0; j<30; j++ )
12          OsdSentByte(color);
13       OsdStop( );
14
15       OsdStart( );
16       OsdSentByte(DEVICE_ADR);
17       OsdSentByte(i|DISPLAY_ROW);
18       OsdSentByte(0|DISPLAY_COLUMN);
19       for( j=0; j<30; j++ )
20          OsdSentByte(_SPACE);
21       OsdStop( );
22       DelayX10ms(1);
23    }
24 }
```

行　号	说　　明
05	起始列至结束列上的字符都清除为空白
07	开始传送起始信号
08~10	传送 OSD 的 slave 地址等于 0x7A，设定属性列和属性行
11,12	列上的每一字符先设定颜色属性
13	传送停止信号

行 号	说 明
15	再一次传送起始信号
16~18	传送 OSD 的 slave 地址等于 0x7A，设定显示列和显示行
19,20	列上每一字符写入空白的字符，并传送至 OSD 的暂存器内
21	传送停止信号
22	延迟 10mS

➢ OsdPrintIcon(row,col,icon,color)

目的

在指定的列与行显示一个字符

参数

row：列
col：行
icon：字符
color：字符的颜色属性

程序

```
01 void OsdPrintIcon(Byte row,Byte col,Byte icon,Byte color)
02 {
03     OsdFormatA(row|ATTRIBUTE_ROW, col, color);
04     OsdFormatA(row|DISPLAY_ROW, col, icon);
05 }
```

行 号	说 明
03	先设定字符的颜色属性
04	再将字符显示出来

➢ OsdStringAdr0(*string,sel)

目的

计算字符串的起始位置

参数

*string：字符串表
sel：字符串表上的哪一列

程序

```
01 void OsdStringAdr0(Byte RDATA *string,Byte sel)
02 {
03    Byte DATA  j;
04
05    for(j=0; j<sel; j++)
06    {
07       while( *string != _EOF )
08       {
09          string++;
10       }
11       string++;
12    }
13    DataPointer = string;
14 }
```

行　号	说　　明
05	从字串表开始并跳过 sel 列
07~10	字串表上的每一列字符其结束符号为_EOF，定义为 0xff，因此当字串上的字符不是_EOF 则将字串指针 string 加 1，当字串上的字符是_EOF 表示此列已经结束
11	此列已经结束则字串指针也要跳过_EOF 这个字符位置，因此字串指针 string 还要加 1
13	循环作完后就指到要显示的列地址，即要从字串表上的第几列开始显示，而将此地址存入 DataPointer 指针变量内

➢ OsdStringAdr(*string,total,sel,fglanguage)

目的

计算含有多国语言之字符串表的起始位置，

参数

*string：字符串表
total：同一种语言上的字符串表总共有几列
sel：字符串表上的哪一列
fglanguage："1" 有含多国语言、"0" 无多国语言

程序

```
01 //multi-language string start pointer
02 void OsdStringAdr(Byte RDATA *string,Byte total,Byte
   sel,Bool fglanguage)
03 {
04    Byte DATA  i,j;
05
06    if ( fglanguage )
07    {
08       for(i=0; i<OsdLanguage; i++)
09       {
10          for(j=0; j<total; j++)
11          {
12             while( *string != _EOF )
13
14                string++;
15             }
16             string++;
17          }
18       }
19    }
20
21    for(j=0; j<sel; j++)
```

```
22    {
23      while( *string != _EOF )
24      {
25        string++;
26      }
27      string++;
28    }
29    DataPointer = string;
30 }
```

行　号	说　　明
06	如果有含多国语言，则需跳过其它语言 OSD 的字符
08	多国语言的索引变量值，例如：英文，其 OsdLanguage=1、法文，其 OsdLanguage=2，依此类推，因此要跳过整段字符
10	每一种语言要显示的字符共有 total 列，因此也要全部跳过这些字符
12~16	开始判断每一列字符是否到结尾了，如果不是，则 string 指针就一直加 1，如果已经到结尾了，则将结束符号_EOF 这个字符位置也跳过，而字串指针 string 还要加 1
21	string 指针已经指到了要显示字符语言的地址了，则开始计算显示第几列的地址
23~27	字串表上的每一列字符其结束符号为_EOF，定义为 0xff，因此当字串上的字符不是_EOF 则将字串指针 string 加 1，当字串上的字符是_EOF 表示此列已经结束，而字串指针 string 还要加结束符号_EOF，因此 string 还要加 1
29	要显示字串的起始点存入 DataPointer 指针变量内

➢ OsdPrintString(row,col,color,*string)

目的

在指定的列、行上将字符串显示出来(一)

参数

row：列地址
col：行地址
color：字符的颜色属性

*string：字符串的起始点

程序

```
01 void OsdPrintString(Byte row,Byte col,Byte color,Byte
   *string)
02 {
03     Byte j;
04     Byte RDATA *temp;
05
06     temp=string;
07     j=2;
08     TrmBuf[0]  = row|ATTRIBUTE_ROW;
09     TrmBuf[1]  = col|ATTRIBUTE_COLUMN;
10     while( (*string)!=_EOF )
11     {
12       TrmBuf[j++] = color;
13       string++;
14     }
15     SlvAdr  = DEVICE_ADR;
16     ByteCnt = j;
17     SendData( );
18
19     string=temp;
20     j=2;
21     TrmBuf[0]  = row|DISPLAY_ROW;
22     TrmBuf[1]  = col|DISPLAY_COLUMN;
23     while( (*string)!=_EOF )
24     {
25       TrmBuf[j++] = *string;
26       string++;
27     }
28     SlvAdr  = DEVICE_ADR;
29     ByteCnt = j;
30     SendData( );
31 }
```

行　号	说　　明
06	将显示字串的起始地址暂存至 temp 变量
07	颜色属性从数组 2 开始储存
08,09	数组内存入属性列和属性行
10~14	字串未到结尾字符_EOF，则将每一字符的颜色属性存入至数组内
15~17	设定好 slave 地址并将数组内的字符数据传送出去
19	返回字串的起始地址
21,22	设定显示列和显示行
23~30	将字串指针变量 string 的内容，即要显示的字符存入至数组内直到结尾字符_EOF 出现为止，并将数组内的字符数据全部传送出去

➢ OsdPrintString1(row,col,color,*string)

目的

在指定的列、行上将字符串显示出来(二)

参数

row：列地址
col：行地址
color：字符的颜色属性
*string：字符串的起始点

程序

```
01 void OsdPrintString1(Byte row,Byte col,Byte color,Byte RDATA
*string)
02 {
03    Byte RDATA *temp;
04
05    temp=string;
06    OsdStart( );
07    OsdSentByte(DEVICE_ADR);
```

```
08    OsdSentByte(row|ATTRIBUTE_ROW);
09    OsdSentByte(col|ATTRIBUTE_COLUMN);
10    while((*string)!=_EOF)
11    {
12      OsdSentByte(color);
13      string++;
14    }
15    OsdStop( );
16
17    string=temp;
18    OsdStart( );
19    OsdSentByte(DEVICE_ADR);
20    OsdSentByte(row|DISPLAY_ROW);
21    OsdSentByte(col|DISPLAY_COLUMN);
22    while((*string)!=_EOF)
23    {
24      OsdSentByte(*string);
25      string++;
26    }
27    OsdStop( );
28    DelayX1ms(10);
29 }
```

行　号	说　　明
05	将显示字串的起始地址暂存至 temp 变量
06~09	开始传送起始信号，传送 OSD 的 slave 地址，设定属性列和属性行
10~15	字串上的字符不是_EOF 则将每一字符的颜色属性传送出去，每传送一个 Byte 则将字串指针 string 加 1，整列字串传送完毕后则传送停止信号
17	返回字串的起始地址
18~21	传送起始信号、OSD 的 slave 地址、设定显示列和显示行
22~26	字串上的字符不是_EOF 则将每一个字符传送出去，每传送一个 Byte 则将字串指针 string 加 1
27	当字串传送完毕后则传送停止信号
28	延迟 10mS

➢ OsdDisableWindow1(sub_window)

目的

不显示 OSD 的背景区域(WINDOW)

参数

sub_window：第几个 window，介于 0~3 之间

程序

```
01 void OsdDisableWindow1(Byte sub_window)
02 {
03     Byte temp;
04     Byte row_start=0;
05     Byte row_end=0;
06     Byte column_start=0;
07     Byte column_end=0;
08     Byte attribute=0;
09
10     temp=row_start;
11     temp<<=4;
12     temp|=row_end;
13     OsdFormatA(15|DISPLAY_ROW, 0+(sub_window-1)*3, temp);
14
15     temp=column_start;
16     temp<<=3;
17     temp&=0x00;  //disable window
18     OsdFormatA(15|DISPLAY_ROW, 1+(sub_window-1)*3, temp);
19
20     temp=column_end;
21     temp<<=3;
22     temp|=attribute;
23     OsdFormatA(15|DISPLAY_ROW, 2+(sub_window-1)*3, temp);
24 }
25
```

```
26 void OsdDisableWindow2(Byte sub_window)
27 {
28     OsdFormatA(15|DISPLAY_ROW,1+(sub_window-1)*3,0);
29 }
```

行 号	说 明
04~08	初始化起始列、结束列，起始行、结束行及背景颜色属性
10~13	起始列移入高 4 位并加上结束列低 4 位，即 but4~bit7 是起始列，而 but0~bit3 是结束列，并写入至指定的暂存器内
15~18	起始行向左移 3 位并除能 window 的显示，写入至指定的暂存器内
20~23	结束行也向左移 3 位并加入属性后，也写入至指定的暂存器内
26	另外一种除能 window 的简单方法
28	只要将特定暂存器的某一个位元清除为 0 即可

➢ OsdSetWindow(sub_window,row_start,row_end,column_start,column_end,attribute)

目的

开启一个 OSD 的背景区域(WINDOW)

参数

sub_window：第几个 window，介于 0~3 之间
row_start：起始列
row_end：结束列
column_start：起始行
column_end：结束行
attribute：背景颜色属性

程序

```
01 void OsdSetWindow(Byte sub_window,Byte row_start,Byte
   row_end, Byte column_start,Byte column_end,Byte attribute)
02
```

```
03 {
04     Byte temp;
05
06     temp=row_start;
07     temp<<=4;
08     temp|=row_end;
09     OsdFormatA(15|DISPLAY_ROW, 0+(sub_window-1)*3, temp);
10
11     temp=column_start;
12     temp<<=3;
13     temp|=0x04;  //disable shadow,enable window display
14     OsdFormatA(15|DISPLAY_ROW, 1+(sub_window-1)*3, temp);
15
16     temp=column_end;
17     temp<<=3;
18     temp|=attribute;
19     OsdFormatA(15|DISPLAY_ROW, 2+(sub_window-1)*3, temp);
20 }
```

行　号	说　　明
06~09	起始列移入高 4 位并加上结束列低 4 位，即 but4~bit7 是起始列，而 but0~bit3 是结束列，并写入至指定的暂存器内
11~14	起始行向左移 3 位并使能 window 的显示，写入至指定的暂存器内
16~19	结束行也向左移 3 位并加入背景颜色属性后，也一并写入至指定的暂存器内

➢ OsdBarHandle(row,col,color)

目的

显示调整大小用的 BAR 字符(一)

参数

row：列地址
col：行地址

color：BAR 字符的颜色属性

程序

```
01 Byte  RDATA OSDM_BAR_H[  ] = { _HBar0,_HBar1,_HBar2,_HBar3,
   _HBar4,_HBar5,_HBar6 };
02
03 // 10 bar * 6=60
04 void OsdBarHandle(Byte row,Byte col,Byte color)
05 {
06    Byte  i,j;
07    Byte  BarMaxValue,BarMaxCount = 10;
08
09    OsdClearRow(row,row,BLACK);
10
11    TrmBuf[0]  = row|ATTRIBUTE_ROW;
12    TrmBuf[1]  = col|ATTRIBUTE_COLUMN;
13    for( j=0; j<(BarMaxCount+2); j++ )
14       TrmBuf[j+2] = color;
15    SlvAdr  = DEVICE_ADR;
16    ByteCnt = BarMaxCount+4;
17    SendData( );
18
19    BarMaxValue = BarMaxCount * 6;
20    UpdateValue = (Byte)( ( (CurrentValue-
      OSDMinValue)*BarMaxValue )/(OSDMaxValue-OSDMinValue) );
21
23    TrmBuf[0]  = row|DISPLAY_ROW;
24    TrmBuf[1]  = col|DISPLAY_COLUMN;
25
26    // Bar Left
27    TrmBuf[2]  = _HBarL;
28
29    // how many full bar
30    j = UpdateValue / 6;
31
```

```
32    for( i=3; i<j+3; i++ )
33      TrmBuf[i] = OSDM_BAR_H[6];
34
35    // not full part
36    j = UpdateValue % 6;
37    TrmBuf[i++] = OSDM_BAR_H[j];
38
39    // no scale part:space
40    for( ; i<(BarMaxCount+3); i++ )
41      TrmBuf[i] = OSDM_BAR_H[0];
42
43    // Bar Right
44    TrmBuf[BarMaxCount+3] = _HBarR;
45
46    SlvAdr  = DEVICE_ADR;
47    ByteCnt = BarMaxCount+4;
48    SendData( );
49 }
```

行　号	说　　明
01	将渐进式 bar 的参数定义出来，以便取表使用
07	bar 的长度为 10 格
09	首先将要显示 bar 的这一列清除为空白
11~17	先设定每一个 bar 区段的颜色属性
19	bar 的最大值为 10*6=60
20	将目前值转换为 bar 的数值
23,24	从要显示 bar 位置的列、行开始
27	bar 的最左边字符
30	计算有多少个满格数
32,33	将满格的字符取表后存入至数组变量 TrmBuf[ ] 内
36,37	余数就是非满格的字符，并取表后也存入至数组变量 TrmBuf[ ] 内
40,41	剩下的是空白 bar，也要存入至数组变量 TrmBuf[ ] 内

行 号	说 明
44	将 bar 的最右边字符，也存入至数组变量 TrmBuf[ ] 内
46	OSD 的 slave 地址等于 0x7A
47	总共要写入的个数
48	写入至 OSD 暂存器内，则就会显示出横向 bar 的样子。一般使用在调整时其数值越大则 bar 越往右填满，反之，则空白 bar 就越多

➢ OsdBarHandle1(row,col,color)

目的

显示调整大小用的 BAR 字符(二)

参数

row：列地址
col：行地址
color： BAR 字符的颜色属性

程序

```
01 // 10 bar * 6=60
02 void OsdBarHandle1(Byte row,Byte col,Byte color)
03 {
04     Byte  i,j;
05     Byte  BarMaxValue,BarMaxCount = 10;
06
07     OsdClearRow(row,row,BLACK);
08
09     OsdStart( );
10     OsdSentByte(DEVICE_ADR);
11     OsdSentByte(row|ATTRIBUTE_ROW);
12     OsdSentByte(col|ATTRIBUTE_COLUMN);
13     for( j=0; j<(BarMaxCount+2); j++ )
14         OsdSentByte(color);
15     OsdStop( );
16
```

```
17    BarMaxValue = BarMaxCount * 6;
18    UpdateValue = (Byte)( ( (CurrentValue-
      OSDMinValue)*BarMaxValue )/(OSDMaxValue-OSDMinValue) );
19
21    TrmBuf[0]  = row|DISPLAY_ROW;
22    TrmBuf[1]  = col|DISPLAY_COLUMN;
23
24     TrmBuf[2]  = _HBarL;
25
26    j = UpdateValue / 6;
27
28    for( i=3; i<j+3; i++ )
29       TrmBuf[i] = OSDM_BAR_H[6];
30
31    j = UpdateValue % 6;
32    TrmBuf[i++] = OSDM_BAR_H[j];
33
34    for( ; i<(BarMaxCount+3); i++ )
35       TrmBuf[i] = OSDM_BAR_H[0];
36
37    TrmBuf[BarMaxCount+3] = _HBarR;
38
39    SlvAdr  = DEVICE_ADR;
40    ByteCnt = BarMaxCount+4;
41    SendData( );
42 }
```

说明

行号 09 至行号 15 不同于上一个范例的写法，其余皆一样而其目的、功能也都一样，在此列出参考不再赘述。

行　号	说　　明
09	开始传送起始信号
10	传送 OSD 的 slave 地址等于 0x7A

行　号	说　　明
11,12	设定属性列和属性行
13,14	bar字符先设定颜色属性
15	传送停止信号

➢ OsdDisplayValue(row,col,color)

目的

显示0~100的数值

参数

row：列地址
col：行地址
color：数值的颜色属性

程序

```
01 Byte  RDATA OSDM_DEC_TABLE[  ] = { _0,_1,_2,_3,_4,_5,_6,_7,_8,_9 };
02
03 // 4 digital
04 void OsdDisplayValue(Byte row,Byte col,Byte color)
05 {
06    Byte i;
07    Bool sign;
08
09    TrmBuf[0]  = row|ATTRIBUTE_ROW;
10    TrmBuf[1]  = col|ATTRIBUTE_COLUMN;
11    for( i=0; i<4; i++ )
12       TrmBuf[i+2] = color;
13    SlvAdr  = DEVICE_ADR;
14    ByteCnt = 7;
15    SendData( );
16
17    CurrentValue = (Byte)( ( (CurrentValue-OSDMinValue)*100
       )/ (OSDMaxValue-OSDMinValue) );
18
```

```
19    for(i=0; i<4; i++)
20    {
21      UpdateValue = CurrentValue % 10;
22      CurrentValue = CurrentValue / 10;
23      TrmBuf[10+i] = OSDM_DEC_TABLE[UpdateValue];
24    }
25
26    UpdateValue = 4;
27    sign = 1;
28    for(i=0; i<4; i++)
29    {
30      if ( sign )
31      {
32        if ( TrmBuf[13-i] != _0 )
33        {
34          sign = 0;
35          TrmBuf[UpdateValue++] = TrmBuf[13-i];
36        }
37      }
38      else
39        TrmBuf[UpdateValue++] = TrmBuf[13-i];
40    }
41    if ( sign )
42      TrmBuf[UpdateValue++] = _0;
43
44    UpdateValue -= 4; //subtrast 4 BaseIndex TrmBuf(4):data
45    NextColumn = col+UpdateValue;  //display icon,next colume
                       position
46
47    SlvAdr    = DEVICE_ADR;
48    ByteCnt   = UpdateValue + 2;
49    TrmBuf[0] = row|DISPLAY_ROW;
50    TrmBuf[1] = col|DISPLAY_COLUMN;
51    for(i=1; i<=UpdateValue; i++)
52      TrmBuf[i+1] = TrmBuf[i+3];
```

```
53     SendData( );
54 }
55
56 //test column position
57 void TEST_VALUE(Byte value)
58 {
59     CurrentValue = value;
60     OsdDisplayValue(1,12,CYAN);
61     OsdPrintIcon(1,NextColumn++,_S ,BLUE);
62     OsdPrintIcon(1,NextColumn++,_E ,BLUE);
63     OsdPrintIcon(1,NextColumn++,_C ,BLUE);
64 }
```

行　号	说　　明
01	数字 0~9 的参数表
09,10	先设定属性列和属性行
11,12	有 4 位数并设定颜色属性
13~15	OSD 的 slave 地址，并写入至 OSD 暂存器内
17	将目前值转换成介于 0~100 之间
19~24	由个位数开始将每一个数字计算出来并取表存入至数组变量 TrmBuf[ ] 内
26,27	设定初始值
28~40	从千位数开始判断是否为零，如果为 0 则将不显示数字 0 出来，依此类推，再判断百位数、十位数、个位数，如果有任 1 位数不为零则不再判断了
41,42	如果 4 个位数都为零，则只显示一个 0 来表示
44,45	数字显示完毕则计算出下一个可以显示字符的行地址
47,48	slave 地址和要写入的个数
49~53	将数组内的数字数据写入至 OSD 的暂存器内
57	应用范例
59	将引数直接存入至目前值 CurrentValue 变量内
60	在第 1 列、第 12 行的地址显示出青色的数值(0~100)
61~63	显示数值后，紧接著显示蓝色的英文字，分别为 S、E、C 字符

➢ OsdDisplayCount(count)

目的

显示多国语言字符串及数值的应用

参数

count：要显示的数值

程序

```
01 Byte  RDATA OSDCOUNT_DISPLAY[  ] =
02 {
03   //English language font
04   _E,_D,_O,_G,__,_C,_O,_U,_N,_T,_COLON,_EOF,
05   //France language font
06   _F,_D,_O,_G,__,_C,_O,_U,_N,_T,_COLON,_EOF,
07   //Germance language font
08   _G,_D,_O,_G,__,_C,_O,_U,_N,_T,_COLON,_EOF,
09   //Italy language font
10   _I,_D,_O,_G,__,_C,_O,_U,_N,_T,_COLON,_EOF,
11   //Spanish language font
12   _S,_D,_O,_G,__,_C,_O,_U,_N,_T,_COLON,_EOF
13 };
14 void OsdDisplayCount(Byte count)
15 {
16    OsdEnable(1);
17    OsdNormal( );
18    OsdLocationSet(0x40,0x50);
19
20    OsdSetWindow( 1,0,2,0,14,BLUE);
21    OsdStringAdr(OSDCOUNT_DISPLAY,1,0,1);
22    OsdPrintString(1,1,RED,DataPointer);
23    CurrentValue = count;
24    OsdDisplayValue(1,11,WHITE);
25 }
```

行　号	说　　明
01	多国语言字符表的标题名称
02~13	分别为英文、法文、德文、意大利文、西班牙文的字符表，请依照个别相对应的字符作修正，在此仅作范例说明用
14	将计数器的数值以 OSD 方式显示在屏幕上
16,17	使能 OSD 功能并以正常的方式显示
18	将 OSD 定位在垂直为 0x40、水平为 0x50 的位置上
20	设定 window1，从第 0 列~第 2 列，第 0 行~第 14 行显示出一个背景色为蓝色的视窗(window)
21	设定要显示字串的地址，OSDCOUNT_DISPLAY 为字符表的标题名称、每一种语言的总字串有 1 列(total)、要显示第 0 列的字串(sel)、具有多国语言(fglanguage)
22	将字串在第 1 列、第 1 行以红色显示出来
23,24	将计数器的数值在第 1 列、第 11 行以白色显示出来

附录 A 头文件

```
#ifndef     GLOBAL_H
#define     GLOBAL_H

//input
extern Bool   FgP1_0;
extern Bool   FgP1_1;
extern Bool   FgP1_2;
extern Bool   FgP1_3;
extern Bool   FgP1_4;
extern Bool   FgP1_5;
extern Bool   FgP1_6;
extern Bool   FgP1_7;
extern Bool   FgP2_0;
extern Bool   FgExtInput1;
extern Bool   FgExtInput2;
extern Bool   FgExtInput3;
extern Bool   FgExtInput4;
extern Bool   FgExtInput5;
extern Bool   FgExtInput6;
extern Bool   FgExtInput7;
extern Bool   FgExtInput8;
extern Bool   FgExtInput9;
extern Bool   FgExtInput10;
extern Bool   FgExtInput11;
extern Bool   FgExtInput12;
extern Bool   FgExtInput13;
extern Bool   FgExtInput14;
extern Bool   FgExtInput15;
```

```
extern Bool   FgExtInput16;
extern Byte   IDATA    TypeState;
extern Byte   IDATA    InputState;

//led
extern Bool   FgExtOut1;
extern Bool   FgExtOut2;
extern Bool   FgExtOut3;
extern Bool   FgExtOut4;
extern Bool   FgExtOut5;
extern Bool   FgExtOut6;
extern Bool   FgExtOut7;
extern Bool   FgExtOut8;
extern Bool   FgLedFlag;
extern Bool   FgLed2On;
extern Bool   FgLed3On;
extern Bool   FgLed2Off;
extern Bool   FgLed1Off;
extern Bool   FgChange;
extern Byte   IDATA    LedCount;
extern Byte   IDATA    OutputState;
extern Byte   IDATA    OutPutData1;

//utility
extern Bool   FgBeepOff;
extern Bool   FgAlarmOff;
extern Bool   FgExtFreq1;
extern Bool   FgExtFreq2;
extern Bool   FgExtFreq3;
extern Bool   FgExtFreq4;
extern Bool   Fgdirection;
extern Byte   IDATA    BcdData1;
extern Byte   IDATA    BcdData2;
extern Byte   IDATA    BcdData3;
extern Byte   IDATA    BcdData4;
```

```
extern Byte   IDATA    DisplayState;
extern Byte   IDATA    BcdVarible;
extern Byte   IDATA    UpdateValue;
extern Byte   IDATA    *ByteVariablePtr;
extern Word   IDATA    *WordVariablePtr;
extern Word   IDATA    MaxValue;
extern Word   IDATA    MinValue;
extern Word   IDATA    BcdVarible1;

//pulse
extern Bool   FgPulse,
extern Bool   FgPulseShort;
extern Bool   FgPulseLong;
extern Bool   FgTimeout;
extern Byte   IDATA    DutyCount;
extern Byte   IDATA    DutyCycleValue;
extern Byte   IDATA    PulseData;
extern Byte   IDATA    LowPulseCount;
extern Byte   IDATA    HiPulseCount;
extern Byte   IDATA    WheelNow;
extern Byte   IDATA    WheelOld;
extern Byte   IDATA    RightCount;
extern Byte   IDATA    LeftCount;
extern Byte   IDATA    EncoderCnt;
extern Byte   IDATA    SquareCount;

//timer
extern Byte   IDATA      T40msTimer;
extern Word   IDATA      LedOffCount;
extern Word   IDATA      Timer40msCount;
extern Word   IDATA      PulseCount;

//key
extern Bool   FgDoubleKey;
extern Bool   FgKEY1;
```

```
extern Bool   FgKEY2;
extern Bool   FgKEY3;
extern Bool   FgKEY4;
extern Bool   FgKEY5;
extern Bool   FgKEY2_ONOFF;
extern Byte   IDATA      KeyData;
extern Byte   IDATA      KeyData1;
extern Byte   IDATA      KeyData2;
extern Byte   IDATA      KeyData3:
extern Byte   IDATA      KeyData4;
extern Byte   IDATA      KeyCount;
extern Byte   IDATA      ScanKeyCounter;
extern Byte   IDATA      KeyBuffer;
extern Byte   IDATA      KeyCountTimer;
extern Byte   IDATA      CoarseAdjCount;

//iic
extern Byte   IDATA   ByteCnt;
extern Byte   IDATA   SlvAdr;

//eeprom
extern Bool   FgVariable1;
extern Bool   FgVariable2;
extern Bool   FgVariable3;
extern Byte   IDATA   TrmBuf[34];
extern Byte   IDATA   Variable1;
extern Byte   IDATA   Variable2;
extern Byte   IDATA   Variable3;
extern Byte   IDATA   Variable4;
extern Byte   IDATA   Variable5;
extern Byte   IDATA   Variable6;
extern Byte   IDATA   Variable7;

//93c66
extern Byte   IDATA   Buffer;
```

```
//osd
extern Byte   IDATA    OsdLanguage;
extern Byte   RDATA   *DataPointer;
extern Byte   IDATA    CurrentValue;
extern Byte   IDATA    OSDMinValue;
extern Byte   IDATA    OSDMaxValue;
extern Byte   IDATA    CurrentValue;
extern Byte   IDATA    DogCount;
extern Byte   IDATA    NextColumn;

//external ram
extern Byte   PDATA    XFR_ADC;
extern Byte   PDATA    DAC0;

#endif



#ifndef      _POWERON_H
#define      _POWERON_H

extern Byte RDATA EEPROM_DEFAULT_TABLE1[][6];
extern Byte RDATA EEPROM_DEFAULT_TABLE2[];

extern void PowerOnInitial(void);
extern void InitialCpu(void);
extern void InitialCpuIO(void);
extern void InitialEeprom(void);
extern void InitialVariable(void);

#endif
```

```
#ifndef     _DELAY_H
#define     _DELAY_H

extern void DelayX1ms(Word count);
extern void DelayX1ms1(Word count);
extern void DelayX1ms2(Byte count);
extern void DelayX10ms(Word count);
extern void DelayX10ms1(Word count);
extern void Delay50uS(Byte count);
extern void ShortDelay(Byte count);
extern void Timer40msDelay(Byte count);
extern void Timer1ISR_40ms(void);

#endif



#ifndef     _INPUT_H
#define     _INPUT_H

extern void  Led_1(void);
extern void  LedOn(void);
extern void  LedOff(void);

extern void  Input1(void);
extern void  Input2(void);
extern void  Input3(void);
extern void  Input4(void);

#endif



#ifndef     _TIMER_H
#define     _TIMER_H
```

```
extern void InitialMain(void);
extern void CountMain1(void);
extern void CountMain2(void);
extern void CountMain3(void);
extern void Int0Main1(void);

#endif



#ifndef     _UTILITY_H
#define     _UTILITY_H

extern void ByteVariableAdd1(void);
extern void ByteVariableAdd2(void);
extern void ByteVariableSub(void);
extern void ByteProcess(void);

extern void WordVariableAdd1(void);
extern void WordVariableAdd2(void);
extern void WordVariableSub1(void);
extern void WordVariableSub2(void);
extern void WordProcess(void);

extern void Hex2Bcd1(Byte value);
extern void Hex2Bcd2(Word value);
extern void Hex2Bcd3(Word value);
extern void Value255_100(Byte value);
extern void Value100_128a(Byte value);
extern void Value100_128b(Byte value);

extern void RamClear(void);
extern void ZeroContinue(Byte counter);

#endif
```

```
#ifndef    _LED_H
#define    _LED_H

extern void  LedFlash0(void);
extern void  LedFlash1(void);
extern void  LedFlash2(void);
extern void  LedFlash3(void);
extern void  LedFlash4(Byte ontime,Byte offtime);
extern void  LedFlash5(Byte count,Byte ontime,Byte offtime);
extern void  LedFlash6(Byte count,Byte ontime,Byte offtime);
extern void  LedFlashGetkey(Byte count,Byte ontime,Byte offtime);

extern void  LedMain1(void);
extern void  LedMain2(void);
extern void  LedOff_Timer1ISR(void);

extern void  LedTimming(void);
extern void  Led1On(void);
extern void  Led1Off(void);
extern void  Led2On(void);
extern void  Led2Off(void);
extern void  Led3On(void);
extern void  Led3Off(void);
extern void  Led4On(void);
extern void  Led4Off(void);

extern void  LedMain3(void);
extern void  LedDelayFlash(void);

extern void  LedMain4(void);
extern void  LedTimerFlash(void);
extern void  Led_Timer1ISR(void);
```

```
extern void   LedMain5(void);

extern void   Led_1Flash(void);
extern void   Led_2Flash(void);
extern void   Led_3Flash(void);

#endif
#ifndef       _BEEP_H
#define       _BEEP_H

extern void Beep1(void);
extern void Beep2(Byte tone);
extern void Beep3(Byte soundlong,Byte tone);
extern void Beep4(Word count,Byte soundlong,Byte tone);

extern void BeepGetkey(Word count,Byte soundlong,Byte tone);
extern void Alarm1(Byte soundlong,Byte tone);
extern void Alarm2(Word count,Byte soundlong,Byte tone);
extern void AlarmGetkey(Word count,Byte soundlong,Byte tone);
extern void BeepLed(Word count,Byte soundlong,Byte tone);

extern void HardWareBeep1(void);
extern void HardWareBeep2(void);
extern void HardWareBeep3(void);

#endif



#ifndef       _MUSIC_H
#define       _MUSIC_H

extern Byte   RDATA SOUND_LONG[];
extern Byte   RDATA SOUND_TONE[];
extern Byte   RDATA MUSIC_SOUNDLONG[];
```

```
extern Byte   RDATA MUSIC_SOUNDTONE[];
extern Byte   RDATA MUSIC_SOUNDLONG1[];
extern Byte   RDATA MUSIC_SOUNDTONE1[];
extern Byte   RDATA MUSIC_SOUNDTONE2[];
extern Byte   RDATA MUSIC_SOUNDLONG3[];
extern Byte   RDATA MUSIC_SOUNDTONE3[];

extern void Sound(void);
extern void Music1(void);
extern void Music2(void);
extern void Music3(void);
extern void Music4(Byte number);
#endif



#ifndef       _BCD_H
#define       _BCD_H

extern void BcdDisplay1(void);
extern void BcdDisplay2(void);
extern void BcdDisplay3(void);
extern void BcdDisplay4(void);

#endif



#ifndef       _DOT5X7_H
#define       _DOT5X7_H

extern Byte   RDATA DISPLAY_TABLE0[];
extern Byte   RDATA DISPLAY_TABLE1[];
extern Byte   RDATA DISPLAY_TABLE2[];
extern Byte   RDATA DISPLAY_TABLE3[];
```

```
extern Byte   RDATA DISPLAY_TABLE4[];
extern Byte   RDATA DISPLAY_TABLE5[];
extern Byte   RDATA DISPLAY_TABLE6[];
extern Byte   RDATA DISPLAY_TABLE7[];
extern Byte   RDATA DISPLAY_TABLE8[];
extern Byte   RDATA DISPLAY_TABLE9[];
extern Byte   RDATA DISPLAY_TABLE10[];
extern Byte   RDATA DISPLAY_TABLE11[][5];

extern void Dot5x7_Display1(void);
extern void Dot5x7_Display2(void);
extern void Dot5x7_Display3(void);
extern void Dot5x7_Display4(void);
extern void Dot5x7_Display5(void);
extern void Dot5x7_Display6(void);

#endif
#ifndef      _IC74138_H
#define      _IC74138_H

extern void   Output74138_1(void);
extern void   Output74138_2(void);
extern void   Output74138_3(void);
extern void   Output74138_4(void);

#endif



#ifndef      _IC4094_H
#define      _IC4094_H

extern void   Output4094_1(Byte value);
extern void   Output4094_2(void);
extern void   Output4094_3(void);
```

```
extern void   Output4094_4(void);
extern void   Output4094_5(Byte outputstate,Byte value);
extern void   DataTx8bit(Byte value);
extern void   Strobe1(void);
extern void   Strobe2(void);
extern void   ClkDelay(void);

#endif



#ifndef      _PULSE_H
#define      _PULSE_H

extern void OutPulse1(void);
extern void OutPulse2(Byte count);
extern void OutPulse3(void);
extern void PulseOff_Timer1ISR(void);
extern void OutPulse4(void);
extern void OutPulse5(void);
extern void PulseDetect1(void);
extern void Det_Pulse1(void);
extern void Subroute1(void);
extern void Subroute2(void);
extern void PulseDetect2(void);
extern void Det_Pulse2(void);
extern void PulseDetect3(void);
extern void Det_Pulse3(void);

extern void PulseGenerator(void);
extern void DelayPulse(void);
extern void Pulse_Timer1ISR (void);
extern void PulseDuty1_Timer1ISR (void);
extern void PulseDuty2_Timer1ISR (void);
```

```
extern void CheckPulseCome(void);
extern void CheckPulseWidth(void);
extern void CheckPulseData1(void);
extern void CheckPulseData2(void);

extern void EncoderProcess(void);
extern void WheelLeft(void);
extern void WheelRight(void);

#endif



#ifndef     _IC4051_H
#define     _IC4051_H

extern void  Input4051_1(void);
extern void  Input4051_2(void);
extern void  Input4051_3(void);
extern void  Input4051_4(void);

extern void  Input4051_5(void);
extern void  Input4051_6(void);

extern void  Input4067_1(void);
extern void  Input4067_2(void);
extern void  Input4067_3(void);
#endif



#ifndef     _KEY_H
#define     _KEY_H

extern void  InputKey1(void);
```

```
extern void   InputKey2(void);
extern void   InputKey3(void);
extern void   ScanKey1(void);
extern void   ScanKey2(void);
extern void   GetKey1(void);
extern void   GetKey2(void);

extern void   InputKeyCheck(void);
extern void   ScanKeyCheck(void);
extern void   GetKeyCheck(void);

extern void   KeyCountCheck1(void);
extern void   KeyCountCheck2(void);

extern void   KeyProcess(void);
extern void   SubProcess(void);

#endif



#ifndef      _DAC08_H
#define      _DAC08_H

extern void LM317_1(void);
extern void LM317_2(void);
extern void LM317_3(void);

extern void DAC08_1(void);
extern void DAC08_2(void);

extern void SawTooth1(void);
extern void SawTooth2(void);
extern void TriAngle1(void);
extern void TriAngle2(void);
```

```
extern void TriAngle3(void);
extern void Square(void);

#endif



#ifndef     _EEP93C66_H
#define     _EEP93C66_H

extern void PushEeprom93c66(void);
extern void EepWriteData(Byte adr,Byte value);
extern void EwenROM(void);
extern void EraseROM(Byte adr);
extern void EraseROM1(Byte adr);
extern void WriteROM(Byte adr,Byte value);
extern void EwdsROM(void);
extern void WR_3wire(Byte value);
extern void XbusDelay(void);

extern void PopEeprom93c66(void);
extern Byte ReadROM(Byte adr);
extern Byte ReceiveByte(void);

#endif



#ifndef     _IIC_H
#define     _IIC_H

extern void I2cInit(void);
extern void I2cWait(void);

extern void I2cStart(void);
```

```
extern void I2cStop(void);
extern Bool I2cSentByte(Byte bytedata);
extern Bool I2cSentByte1(Byte bytedata);
extern Byte I2cReceiveByte(void);
extern void SendAcknowledge(Bool ack);
extern void I2cByteWrite(Byte device,Byte address,Byte bytedata);
extern void I2cByteWrite1(Byte device,Byte address,Byte bytedata);
extern void I2cByteWrite2(Byte device,Byte address,Byte bytedata);

extern Byte I2cByteRead(Byte device,Byte address);
extern Bool I2cSentData(Byte bytecnt);
extern Bool I2cReceiveData(Byte bytecnt);

extern void DataSetBit(Byte device,Byte addr,Byte bitno);
extern void DataSetBit1(Byte device,Byte addr,Byte bitno);
extern void DataSetBit2(Byte device,Byte addr);
extern void DataClearBit(Byte device,Byte addr,Byte bitno);
extern void DataClearBit1(Byte device,Byte addr,Byte bitno);
extern void DataClearBit2(Byte device,Byte addr);

#endif



#ifndef     _PWM_H
#define     _PWM_H

extern void   PWM_OutputHI(void);
extern void   PWM_OutputLOW(void);
extern void   PWM_Output(Byte value);
extern void   DacByteWrite(Byte device,Byte subaddress,Byte bytedata);
extern void   TEST_DacOut(void);

#endif
```

```
#ifndef     _EEP24C08_H
#define     _EEP24C08_H

extern void EepromByteWrite0(Byte bank,Byte addr,Byte value);
extern void EepromWordWrite0(Byte bank,Byte addr,Word value);
extern Byte EepromByteRead0(Byte bank,Byte addr);
extern Word EepromWordRead0(Byte bank,Byte addr);

extern void EepromByteWrite(Word addr,Byte bytedata);
extern Byte EepromByteRead(Word addr);
extern void EcpromPageWrite(void);
extern void EepromPageRead(void);
extern void EepromWrite(Word SubAddress, Byte count);
extern void EepromRead(Word SubAddress,Byte count);
extern Bool SendData(void);
extern Bool SendEepromData(void);
extern Bool RcvData(void);

extern Byte DdcChecksum(Byte adr);

#endif



#ifndef     _EEP24C32_H
#define     _EEP24C32_H

extern void Eeprom24c32WriteByte_1(Word addr,Byte value);
extern void Eeprom24c32WriteByte_2(Word addr,Byte value);

extern void Eeprom24c32WriteMulti_1(Word addr,Byte count);
extern void Eeprom24c32WriteMulti_2(Word addr,Byte count);
```

```
extern Byte Eeprom24c32ReadByte_1(Word addr);
extern Byte Eeprom24c32ReadByte_2(Word addr);

extern Word Eeprom24c32ReadWord_1(Word addr);
extern Word Eeprom24c32ReadWord_2(Word addr);

extern void Eeprom24c32ReadMulti_1(Word addr,Byte count);

extern void Eeprom24c32ReadMulti_2(Word addr,Byte count);

#endif
#ifndef     _OSD_H
#define     _OSD_H

extern Byte RDATA OSDM_DEC_TABLE[];
extern Byte RDATA OSDM_BAR_H[];
extern Byte RDATA OSDCOUNT_DISPLAY[];
extern Byte RDATA TEST_DISPLAY[];

extern void OsdStart(void);
extern void OsdStop(void);
extern Bool OsdSentByte(Byte bytedata);
extern Byte OsdReceiveByte(void);
extern void OsdFormatA_0(Byte row,Byte col,Byte value);
extern void OsdFormatA(Byte row,Byte col,Byte value);
extern void OsdFrameControl(Byte vertd,Byte hord,Byte height,Byte width,Byte rowspace);
extern void OsdLocationSet(Byte vertical,Byte horizontal);
extern void OsdRamClear(void);
extern void OsdEnable(Bool yes);
extern void OsdResetFont(void);
extern void OsdOpenUp(void);
extern void OsdNormal(void);
extern void OsdClearRow(Byte start,Byte end,Byte color);
extern void OsdClearRow1(Byte start,Byte end,Byte color);
extern void OsdPrintIcon(Byte row,Byte col,Byte icon,Byte color);
```

```
extern void OsdStringAdr0(Byte RDATA *string,Byte sel);
extern void OsdStringAdr(Byte RDATA *string,Byte total,Byte sel,Bool fglanguage);

extern void OsdPrintString(Byte row,Byte col,Byte color,Byte RDATA *string);
extern void OsdPrintString1(Byte row,Byte col,Byte color,Byte RDATA *string);

extern void OsdDisableWindow1(Byte sub_window);
extern void OsdDisableWindow2(Byte sub_window);
extern void OsdSetWindow(Byte sub_window,Byte row_start,Byte row_end,Byte column_start,Byte column_end,Byte attribute);
extern void OsdBarHandle(Byte row,Byte col,Byte color);
extern void OsdBarHandle1(Byte row,Byte col,Byte color);
extern void OsdDisplayValue(Byte row,Byte col,Byte color);
extern void TEST_VALUE(Byte value);

extern void OsdDisplayCount(Byte count);

#endif
```

附录 B　汇编程序

```
Main:
    LCALL  PowerOnInitial
?C0001:
    LCALL  KeyProcess
    LCALL  SubRoutine11
    LCALL  SubRoutine22
    JB     FgLedFlag,?C0003
    MOV    R0,#LedOffCount+01H
    MOV    A,@R0
    DEC    R0
    ORL    A,@R0
    JNZ    ?C0003
    SETB   FgLedFlag
    LCALL  LedOff
?C0003:
    LCALL  SubRoutine33
    SJMP   ?C0001
    RET

SubRoutine11:
    RET

SubRoutine22:
    RET

SubRoutine33:
    RET


PowerOnInitial:
    LCALL  InitialCpu
    LCALL  InitialCpuIO
    MOV    R7,#0FEH
    MOV    R6,#01H
    MOV    R5,#02H
    LCALL  _EepromRead
    MOV    R0,#TmBuf
    MOV    A,@R0
    CJNE   A,#055H,?C0002
    INC    R0
    MOV    A,@R0
    XRL    A,#0AAH
?C0006:
    JZ     ?C0001
?C0002:
    LCALL  InitialEeprom
?C0001:
    LCALL  InitialVariable
    RET


InitialCpu:
    CLR    A
    MOV    IE,A
    MOV    PSW,A
    MOV    IP,#0BH
    MOV    TMOD,#015H
    CLR    TR0
    SETB   TR1
    SETB   IT0
    MOV    TL0,#0FFH
    MOV    TH0,#0FFH
    MOV    TL1,#0C0H
    MOV    TH1,#063H
    SETB   EX0
    SETB   ET1
    SETB   ET0
    SETB   EA
    RET


InitialCpuIO:
    CLR    P1_0
    SETB   P1_1
    CLR    P1_2
    CLR    P1_3
    SETB   P1_4
    SETB   P1_5
    CLR    P1_6
    SETB   P1_7
    RET
InitialEeprom:
    CLR    A
    MOV    i?340,A

    MOV    A,i?340
```

```
    MOV     B,#06H
    MUL     AB
    ADD     A,#LOW
(EPROM_DEFAULT_TABLE1)
    MOV     DPL,A
    CLR     A
    ADDC    A,#HIGH
(EEPROM_DEFAULT_TABLE1)
    MOV     R7,DPL
    MOV     DataPointer,A
    MOV     DataPointer+01H,R7
    CLR     A
    MOV     R7,A
?C0009:
    MOV     A,DataPointer+01H
    ADD     A,R7
    MOV     DPL,A
    CLR     A
    ADDC    A,DataPointer
    MOV     DPH,A
    CLR     A
    MOVC    A,@A+DPTR
    MOV     R6,A
    MOV     A,#TrmBuf+01H
    ADD     A,R7
    MOV     R0,A
    MOV     A,R6
    MOV     @R0,A
    INC     R7
    CJNE    R7,#06H,?C0009
?C0010:
    MOV     A,i?340
    MOV     B,#06H
    MUL     AB
    MOV     R7,A
    MOV     R6,B
    MOV     R5,#06H
    LCALL   _EepromWrite
    INC     i?340
    MOV     A,i?340
    CLR     C
    SUBB    A,#03H
    JC      ?C0006
?C0007:
    CLR     A
    MOV     i?340,A
?C0012:
    MOV     A,i?340
    MOV     DPTR,#EEPROM_DEFAULT_TABLE2
    MOVC    A,@A+DPTR
    MOV     R7,A
    MOV     A,#TrmBuf+01H
    ADD     A,i?340
    MOV     R0,A
    MOV     A,R7
    MOV     @R0,A
    INC     i?340
    MOV     A,i?340
    CJNE    A,#0CH,?C0012
?C0013:
    MOV     R7,#080H
    MOV     R6,#00H
    MOV     R5,#0CH
    LCALL   EepromWrite
    MOV     R0,#TrmBuf+01H
    MOV     @R0,#055H
    INC     R0
    MOV     @R0,#0AAH
    MOV     R7,#0FEH
    MOV     R6,#01H
    MOV     R5,#02H
    LCALL   EepromWrite
    RET


InitialVariable:
    CLR     A
    MOV     R0,#LedCount
    MOV     @R0,A
    MOV     R0,#OutputState
    MOV     @R0,#0FFH
    MOV     R0,#DutyCount
    MOV     @R0,A
    MOV     R0,#KeyData
    MOV     @R0,A
    CLR     FgKEY1
    CLR     FgKEY2
    CLR     FgKEY3
    CLR     FgKEY4
    CLR     FgKEY5
    RET


EEPROM_DEFAULT_TABLE1:
    DB      011H,012H,013H,014H,015H,016H
    DB      081H,082H,083H,084H,085H,086H
    DB      0C1H,0C2H,0C3H,0C4H,0C5H,0C6H


EEPROM_DEFAULT_TABLE2:
    DB      080H,000H,00FH,010H,010H,000H
    DB      001H,000H,000H,000H,000H,000H


_DelayX1ms:
    CLR     A
    MOV     R5,A
    MOV     R4,A
```

```
?C0001:
    CLR     C
    MOV     A,R5
    SUBB    A,R7
    MOV     A,R4
    SUBB    A,R6
    JNC     ?C0007
    CLR     A
    MOV     R3,A
    MOV     R2,A
?C0004:
    INC     R3
    CJNE    R3,#00H,?C0062
    INC     R2
?C0062:
    MOV     A,R3
    XRL     A,#078H
    ORL     A,R2
    JNZ     ?C0004
?C0003:
    INC     R5
    CJNE    R5,#00H,?C0063
    INC     R4
?C0063:
    SJMP    ?C0001
?C0007:
    RET

_DelayX1ms1:
?C0008:
    MOV     A,R7
    DEC     R7
    XCH     A,R2
    MOV     A,R6
    XCH     A,R2
    JNZ     ?C0064
    DEC     R6
?C0064:
    ORL     A,R2
    JZ      ?C0013
    CLR     A
    MOV     R4,A
    MOV     R5,A
?C0010:
    CLR     C
    MOV     A,R5
    SUBB    A,#048H
    MOV     A,R4
    SUBB    A,#00H
    JNC     ?C0008
    INC     R5
    CJNE    R5,#00H,?C0065
    INC     R4
?C0065:
    SJMP    ?C0010
?C0013:
    RET

_DelayX1ms2:
    CLR     A
    MOV     R6,A
?C0014:
    MOV     A,R6
    CLR     C
    SUBB    A,R7
    JNC     ?C0023
    CLR     A
    MOV     R5,A
?C0017:
    CLR     A
    MOV     R4,A
?C0020:
    INC     R4
    CJNE    R4,#078H,?C0020
?C0019:
    INC     R5
    CJNE    R5,#028H,?C0017
?C0016:
    INC     R6
    SJMP    ?C0014
?C0023:
    RET

_DelayX10ms:
    MOV     count?349,R6
    MOV     count?349+01H,R7
    CLR     A
    MOV     R7,A
    MOV     R6,A
?C0024:
    CLR     C
    MOV     A,R7
    SUBB    A,count?349+01H
    MOV     A,R6
    SUBB    A,count?349
    JNC     ?C0033
    CLR     A
    MOV     R5,A
    MOV     R4,A
?C0027:
    CLR     A
    MOV     R3,A
    MOV     R2,A
?C0030:
    INC     R3
```

```
        CJNE    R3,#00H,?C0066
        INC     R2
?C0066:
        MOV     A,R3
        XRL     A,#078H
        ORL     A,R2
        JNZ     ?C0030
?C0029:
        INC     R5
        CJNE    R5,#00H,?C0067
        INC     R4
?C0067:
        MOV     A,R5
        XRL     A,#0AH
        ORL     A,R4
        JNZ     ?C0027
?C0026:
        INC     R7
        CJNE    R7,#00H,?C0068
        INC     R6
?C0068:
        SJMP    ?C0024
?C0033:
        RET


_DelayX10ms1:
?C0034:
        MOV     A,R7
        DEC     R7
        XCH     A,R2
        MOV     A,R6
        XCH     A,R2
        JNZ     ?C0069
        DEC     R6
?C0069:
        ORL     A,R2
        JZ      ?C0042
        CLR     A
        MOV     R4,A
        MOV     R5,A
?C0036:
        CLR     C
        MOV     A,R5
        SUBB    A,#0AH
        MOV     A,R4
        SUBB    A,#00H
        JNC     ?C0034
        CLR     A
        MOV     R3,A
        MOV     R2,A
?C0039:
        INC     R3
        CJNE    R3,#00H,?C0070
        INC     R2
?C0070:
        MOV     A,R3
        XRL     A,#048H
        ORL     A,R2
        JNZ     ?C0039
?C0038:
        INC     R5
        CJNE    R5,#00H,?C0071
        INC     R4
?C0071:
        SJMP    ?C0036
?C0042:
        RET


_Delay50uS:
        CLR     A
        MOV     R6,A
?C0043:
        MOV     A,R6
        CLR     C
        SUBB    A,R7
        JNC     ?C0049
        CLR     A
        MOV     R5,A
?C0046:
        INC     R5
        CJNE    R5,#06H,?C0046
?C0045:
        INC     R6
        SJMP    ?C0043
?C0049:
        RET


_ShortDelay:
        CLR     A
        MOV     R6,A
?C0050:
        MOV     A,R6
        CLR     C
        SUBB    A,R7
        JNC     ?C0056
        CLR     A
        MOV     R5,A
?C0053:
        NOP
        INC     R5
        MOV     A,R5
        SETB    C
        SUBB    A,#08CH
        JC      ?C0053
```

```
?C0052:
    INC     R6
    SJMP    ?C0050
?C0056:
    RET


_Timer40msDelay:
    MOV     R0,#T40msTimer
    MOV     A,R7
    MOV     @R0,A
?C0057:
    MOV     R0,#T40msTimer
    MOV     A,@R0
    JNZ     ?C0057
?C0059:
    RET


Timer1ISR_40ms:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW,#010H
    MOV     TL1,#0C0H
    MOV     TH1,#063H
    CLR     TF1
    MOV     R0,#T40msTimer
    MOV     A,@R0
    JZ      ?C0061
    DEC     @R0
?C0061:
    POP     PSW
    POP     ACC
    RETI


Led_1:
    CLR     P1_0
    RET


LedOn:
    SETB    P1_0
    RET


LedOff:
    CLR     P1_0
    RET


Input1:
    JB      P1_0,?C0004
    SETB    FgP1_0
    MOV     R0,#TypeState
    MOV     @R0,#0FFH
    RET
?C0004:
    CLR     FgP1_0
    CLR     A
    MOV     R0,#TypeState
    MOV     @R0,A
    RET
Input2:
    JNB     P1_0,?C0007
    SETB    FgP1_0
    RET
?C0007:
    CLR     FgP1_0
    RET


Input3:
    MOV     A,P1
    CPL     A
    ANL     A,#0FH
    MOV     R7,A
    CLR     FgP1_0
    CLR     FgP1_1
    JNB     ACC.0,?C0010
    SETB    FgP1_0
?C0010:
    MOV     A,R7
    JNB     ACC.1,?C0011
    SETB    FgP1_1
?C0011:
    CLR     A
    MOV     R0,#TypeState
    MOV     @R0,A
    MOV     A,R7
    ANL     A,#0CH
    MOV     R6,A
    CJNE    R6,#04H,?C0012
    MOV     @R0,#01H
    RET
?C0012:
    CJNE    R6,#08H,?C0015
    MOV     R0,#TypeState
    MOV     @R0,#02H
?C0015:
    RET


Input4:
    JB      P1_0,?C0016
    SETB    FgP1_0
    SJMP    ?C0017
?C0016:
```

```
    CLR     FgP1_0
?C0017:
    JB      P2_0,?C0018
    SETB    FgP2_0
    RET
?C0018:
    CLR     FgP2_0
    RET


Input5:
    MOV     A,P1
    CPL     A
    MOV     R7,A
    CLR     FgP1_0
    CLR     FgP1_1
    CLR     FgP1_2
    CLR     FgP1_3
    CLR     FgP1_4
    CLR     FgP1_5
    CLR     FgP1_6
    CLR     FgP1_7
    JNB     ACC.0,?C0021
    SETB    FgP1_0
?C0021:
    MOV     A,R7
    JNB     ACC.1,?C0022
    SETB    FgP1_1
?C0022:
    MOV     A,R7
    JNB     ACC.2,?C0023
    SETB    FgP1_2
?C0023:
    MOV     A,R7
    JNB     ACC.3,?C0024
    SETB    FgP1_3
?C0024:
    MOV     A,R7
    JNB     ACC.4,?C0025
    SETB    FgP1_4
?C0025:
    MOV     A,R7
    JNB     ACC.5,?C0026
    SETB    FgP1_5
?C0026:
    MOV     A,R7
    JNB     ACC.6,?C0027
    SETB    FgP1_6
?C0027:
    MOV     A,R7
    JNB     ACC.7,?C0029
    SETB    FgP1_7
?C0029:
    RET


InitialMain:
    CLR     A
    MOV     IE,A
    MOV     IP,#0BH
    MOV     TMOD,#015H
    SETB    TR0
    SETB    TR1
    SETB    IT0
    MOV     TL1,#0C0H
    MOV     TH1,#063H
    MOV     TL0,#0FFH
    MOV     TH0,#0FFH
    SETB    EX0
    SETB    ET1
    SETB    ET0
    SETB    EA
    RET


CountMain1:
    CLR     A
    MOV     IE,A
    MOV     IP,#0BH
    MOV     TMOD,#015H
    MOV     TL0,#0FFH
    MOV     TH0,#0FFH
    SETB    ET0
    SETB    EA
    SETB    TR0
    MOV     R0,#PulseCount
    MOV     @R0,A
    INC     R0
    MOV     @R0,A
    RET


Timer0ISR_1:
    PUSH    ACC
    PUSH    B
    PUSH    DPH
    PUSH    DPL
    PUSH    PSW
    MOV     PSW,#018H
    LCALL   LedOn
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   LedOff
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    MOV     TL0,#0FFH
```

```
    MOV   TH0,#0FFH
    CLR   TF0
    POP   PSW
    POP   DPL
    POP   DPH
    POP   B
    POP   ACC
    RETI

Timer0ISR_2:
    PUSH  ACC
    PUSH  B
    PUSH  DPH
    PUSH  DPL
    PUSH  PSW
    MOV   PSW,#018H
    MOV   R0,#PulseCount+01H
    INC   @R0
    MOV   A,@R0
    JNZ   ?C0022
    DEC   R0
    INC   @R0
?C0022:
    MOV   R0,#PulseCount+01H
    MOV   A,@R0
    XRL   A,#03H
    DEC   R0
    ORL   A,@R0
    JNZ   ?C0004
    MOV   @R0,A
    INC   R0
    MOV   @R0,A
    LCALL LedOn
    MOV   R7,#032H
    MOV   R6,#00H
    LCALL _DelayX10ms
    LCALL LedOff
    MOV   R7,#032H
    MOV   R6,#00H
    LCALL _DelayX10ms
?C0004:
    MOV   TL0,#0FFH
    MOV   TH0,#0FFH
    CLR   TF0
    POP   PSW
    POP   DPL
    POP   DPH
    POP   B
    POP   ACC
    RETI

CountMain2:
    CLR   A
    MOV   IE,A
    MOV   IP,#0BH
    MOV   TMOD,#015H
    MOV   TL0,#0FDH
    MOV   TH0,#0FFH
    SETB  ET0
    SETB  EA
    SETB  TR0
    RET

Timer0ISR_3:
    PUSH  ACC
    PUSH  B
    PUSH  DPH
    PUSH  DPL
    PUSH  PSW
    MOV   PSW,#018H
    LCALL LedOn
    MOV   R7,#032H
    MOV   R6,#00H
    LCALL _DelayX10ms
    LCALL LedOff
    MOV   R7,#032H
    MOV   R6,#00H
    LCALL _DelayX10ms
    MOV   TL0,#0FDH
    MOV   TH0,#0FFH
    CLR   TF0
    POP   PSW
    POP   DPL
    POP   DPH
    POP   B
    POP   ACC
    RETI

CountMain3:
    CLR   A
    MOV   IE,A
    MOV   IP,#0BH
    MOV   TMOD,#015H
    MOV   TL0,#0FFH
    MOV   TH0,#0FFH
    SETB  ET0
    SETB  EA
    SETB  TR0
    MOV   R0,#PulseCount
    MOV   @R0,A
    INC   R0
    MOV   @R0,A
    RET
```

```
Timer0ISR_4:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW,#018H
    MOV     R0,#PulseCount+01H
    INC     @R0
    MOV     A,@R0
    JNZ     ?C0023
    DEC     R0
    INC     @R0
?C0023:
    MOV     TL0,#0FFH
    MOV     TH0,#0FFH
    CLR     TF0
    POP     PSW
    POP     ACC
    RETI

Int0Main1:
    CLR     A
    MOV     IE,A
    MOV     IP,#0BH
    SETB    IT0
    SETB    EX0
    SETB    EA
    RET

One_INT0ISR:
    PUSH    ACC
    PUSH    B
    PUSH    DPH
    PUSH    DPL
    PUSH    PSW
    MOV     PSW,#018H
    CLR     EX0
    LCALL   LedOn
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   LedOff
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SETB    EX0
    CLR     IE0
    POP     PSW
    POP     DPL
    POP     DPH
    POP     B
    POP     ACC
    RETI

More_INT0ISR:
    PUSH    ACC
    PUSH    B
    PUSH    DPH
    PUSH    DPL
    PUSH    PSW
    MOV     PSW,#018H
    CLR     EX0
    JNB     P1_0,?C0012
    LCALL   Led1On
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   Led1Off
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0013
?C0012:
    JNB     P1_1,?C0014
    LCALL   Led2On
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   Led2Off
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0013
?C0014:
    JNB     P1_2,?C0016
    LCALL   Led3On
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   Led3Off
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0013
?C0016:
    JNB     P1_3,?C0013
    LCALL   Led4On
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   Led4Off
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
?C0013:
    SETB    EX0
```

```
    CLR     IE0
    POP     PSW
    POP     DPL
    POP     DPH
    POP     B
    POP     ACC
    RETI


Timer1ISR_1:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW,#010H
    MOV     TL1,#0C0H
    MOV     TH1,#063H
    CLR     TF1
    MOV     R0,#Timer40msCount+01H
    INC     @R0
    MOV     A,@R0
    JNZ     ?C0024
    DEC     R0
    INC     @R0
?C0024:
    MOV     R0,#LedOffCount+01H
    MOV     A,@R0
    DEC     R0
    ORL     A,@R0
    JZ      ?C0025
    INC     R0
    MOV     A,@R0
    DEC     @R0
    JNZ     ?C0025
    DEC     R0
    DEC     @R0
?C0025:
    POP     PSW
    POP     ACC
    RETI


UnSignVar:
    CLR     A
    MOV     R7,A
?C0001:
    INC     R7
    SJMP    ?C0001
    RET


SignVar:
    CLR     A
    MOV     R7,A
?C0004:
    INC     R7
    SJMP    ?C0004
    RET


ByteVariableAdd1:
    MOV     R0,ByteVariablePtr
    MOV     A,@R0
    MOV     R7,A
    MOV     R0,#UpdateValue
    MOV     A,@R0
    MOV     R6,A
    MOV     R5,A
    MOV     A,R7
    ADD     A,R5
    MOV     R5,A
    CLR     A
    RLC     A
    MOV     R4,A
    MOV     R0,#MaxValue+01H
    MOV     A,@R0
    MOV     R1,A
    MOV     R3,A
    CLR     C
    MOV     A,R5
    SUBB    A,R3
    MOV     A,#080H
    MOV     R0,A
    XRL     A,R4
    SUBB    A,R0
    JNC     ?C0007
    MOV     A,R6
    ADD     A,R7
    MOV     R7,A
    SJMP    ?C0008
?C0007:
    MOV     A,R1
    MOV     R7,A
?C0008:
    MOV     R0,ByteVariablePtr
    MOV     A,R7
    MOV     @R0,A
    RET


ByteVariableAdd2:
    MOV     R0,ByteVariablePtr
    MOV     A,@R0
    MOV     R7,A
    MOV     R6,#00H
    MOV     R0,#UpdateValue
    MOV     A,@R0
    MOV     R1,A
    ADD     A,R7
```

```
    MOV     R5,A
    CLR     A
    RLC     A
    MOV     R4,A
    MOV     R0,#MaxValue
    MOV     A,@R0
    MOV     R2,A
    INC     R0
    MOV     A,@R0
    MOV     R3,A
    CLR     C
    MOV     A,R5
    SUBB    A,R3
    MOV     A,R2
    XRL     A,#080H
    MOV     R0,A
    MOV     A,R4
    XRL     A,#080H
    SUBB    A,R0
    JNC     ?C0010
    MOV     A,R1
    ADD     A,R7
    MOV     R7,A
    CLR     A
    RLC     A
    MOV     R6,A
    SJMP    ?C0011
?C0010:
    XCH     A,R6
    MOV     A,R2
    XCH     A,R6
    XCH     A,R7
    MOV     A,R3
    XCH     A,R7
?C0011:
    MOV     A,R7
    MOV     R0,ByteVariablePtr
    MOV     @R0,A
    RET

ByteVariableSub:
    MOV     R0,ByteVariablePtr
    MOV     A,@R0
    MOV     R7,A
    MOV     R6,#00H
    MOV     R0,#UpdateValue
    MOV     A,@R0
    MOV     R1,A
    MOV     R5,A
    CLR     C
    MOV     A,R7
    SUBB    A,R5
    MOV     R5,A
    MOV     A,R6
    SUBB    A,#00H
    MOV     R4,A
    MOV     R0,#MinValue
    MOV     A,@R0
    MOV     R2,A
    INC     R0
    MOV     A,@R0
    MOV     R3,A
    SETB    C
    MOV     A,R5
    SUBB    A,R3
    MOV     A,R2
    XRL     A,#080H
    MOV     R0,A
    MOV     A,R4
    XRL     A,#080H
    SUBB    A,R0
    JC      ?C0013
    MOV     A,R1
    MOV     R5,A
    CLR     C
    MOV     A,R7
    SUBB    A,R5
    MOV     R7,A
    MOV     A,R6
    SUBB    A,#00H
    MOV     R6,A
    SJMP    ?C0014
?C0013:
    XCH     A,R6
    MOV     A,R2
    XCH     A,R6
    XCH     A,R7
    MOV     A,R3
    XCH     A,R7
?C0014:
    MOV     A,R7
    MOV     R0,ByteVariablePtr
    MOV     @R0,A
    RET

ByteProcess:
    MOV     R0,ByteVariablePtr
    MOV     A,@R0
    MOV     R7,A
    MOV     R6,#00H
    JNB     Fgdirection,?C0016
    MOV     R0,#UpdateValue
    MOV     A,@R0
    MOV     R1,A
    MOV     R5,A
    CLR     C
```

```
    MOV   A,R7
    SUBB  A,R5
    MOV   R5,A
    MOV   A,R6
    SUBB  A,#00H
    MOV   R4,A
    MOV   R0,#MinValue
    MOV   A,@R0
    MOV   R2,A
    INC   R0
    MOV   A,@R0
    MOV   R3,A
    SETB  C
    MOV   A,R5
    SUBB  A,R3
    MOV   A,R2
    XRL   A,#080H
    MOV   R0,A
    MOV   A,R4
    XRL   A,#080H
    SUBB  A,R0
    JC    ?C0017
    MOV   A,R1
    MOV   R5,A
    CLR   C
    MOV   A,R7
    SUBB  A,R5
    MOV   R7,A
    MOV   A,R6
    SUBB  A,#00H
    MOV   R6,A
    SJMP  ?C0019
?C0017:
    XCH   A,R6
    MOV   A,R2
    XCH   A,R6
    XCH   A,R7
    MOV   A,R3
    XCH   A,R7
    SJMP  ?C0019
?C0016:
    MOV   R0,#UpdateValue
    MOV   A,@R0
    MOV   R1,A
    ADD   A,R7
    MOV   R5,A
    CLR   A
    ADDC  A,R6
    MOV   R4,A
    MOV   R0,#MaxValue
    MOV   A,@R0
    MOV   R2,A
    INC   R0
    MOV   A,@R0
    MOV   R3,A
    CLR   C
    MOV   A,R5
    SUBB  A,R3
    MOV   A,R2
    XRL   A,#080H
    MOV   R0,A
    MOV   A,R4
    XRL   A,#080H
    SUBB  A,R0
    JNC   ?C0020
    MOV   A,R1
    ADD   A,R7
    MOV   R7,A
    CLR   A
    ADDC  A,R6
    MOV   R6,A
    SJMP  ?C0019
?C0020:
    XCH   A,R6
    MOV   A,R2
    XCH   A,R6
    XCH   A,R7
    MOV   A,R3
    XCH   A,R7
?C0019:
    MOV   A,R7
    MOV   R0,ByteVariablePtr
    MOV   @R0,A
    RET

WordVariableAdd1:
    MOV   R0,WordVariablePtr
    MOV   A,@R0
    MOV   R6,A
    INC   R0
    MOV   A,@R0
    MOV   R7,A
    MOV   R0,#UpdateValue
    MOV   A,@R0
    MOV   R1,A
    ADD   A,R7
    MOV   R5,A
    CLR   A
    ADDC  A,R6
    MOV   R4,A
    MOV   R0,#MaxValue
    MOV   A,@R0
    MOV   R2,A
    INC   R0
    MOV   A,@R0
    MOV   R3,A
    CLR   C
```

```
    MOV     A,R5
    SUBB    A,R3
    MOV     A,R4
    SUBB    A,R2
    JNC     ?C0023
    MOV     A,R1
    ADD     A,R7
    MOV     R7,A
    CLR     A
    ADDC    A,R6


    MOV     R6,A
    SJMP    ?C0024
?C0023:
    XCH     A,R6
    MOV     A,R2
    XCH     A,R6
    XCH     A,R7
    MOV     A,R3
    XCH     A,R7
?C0024:
    MOV     R0,WordVariablePtr
    MOV     A,R6
    MOV     @R0,A
    INC     R0
    MOV     A,R7
    MOV     @R0,A
    RET


WordVariableAdd2:
    MOV     R0,WordVariablePtr
    MOV     A,@R0
    MOV     R6,A
    INC     R0
    MOV     A,@R0
    MOV     R7,A
    MOV     R0,#UpdateValue
    MOV     A,@R0
    MOV     R1,A
    ADD     A,R7
    MOV     R5,A
    CLR     A
    ADDC    A,R6
    MOV     R4,A
    MOV     R0,#MaxValue
    MOV     A,@R0
    MOV     R2,A
    INC     R0
    MOV     A,@R0
    MOV     R3,A
    CLR     C
    MOV     A,R5
    SUBB    A,R3
    MOV     A,R2
    XRL     A,#080H
    MOV     R0,A
    MOV     A,R4
    XRL     A,#080H
    SUBB    A,R0
    JNC     ?C0026
    MOV     A,R1
    ADD     A,R7
    MOV     R7,A
    CLR     A
    ADDC    A,R6
    MOV     R6,A
    SJMP    ?C0027
?C0026:
    XCH     A,R6
    MOV     A,R2
    XCH     A,R6
    XCH     A,R7
    MOV     A,R3
    XCH     A,R7
?C0027:
    MOV     R0,WordVariablePtr
    MOV     A,R6
    MOV     @R0,A
    INC     R0
    MOV     A,R7
    MOV     @R0,A
    RET


WordVariableSub1:
    MOV     R0,WordVariablePtr
    MOV     A,@R0
    MOV     R6,A
    INC     R0
    MOV     A,@R0
    MOV     R7,A
    MOV     R0,#UpdateValue
    MOV     A,@R0
    MOV     R1,A
    MOV     R5,A
    CLR     C
    MOV     A,R7
    SUBB    A,R5
    MOV     R5,A
    MOV     A,R6
    SUBB    A,#00H
    MOV     R4,A
    MOV     R0,#MinValue
    MOV     A,@R0
    MOV     R2,A
    INC     R0
```

```
    MOV   A,@R0
    MOV   R3,A
    SETB  C
    MOV   A,R5
    SUBB  A,R3
    MOV   A,R4
    SUBB  A,R2
    JC    ?C0029
    MOV   A,R1
    MOV   R5,A
    CLR   C
    MOV   A,R7
    SUBB  A,R5
    MOV   R7,A
    MOV   A,R6
    SUBB  A,#00H
    MOV   R6,A
    SJMP  ?C0030
?C0029:
    XCH   A,R6
    MOV   A,R2
    XCH   A,R6
    XCH   A,R7
    MOV   A,R3
    XCH   A,R7
?C0030:
    MOV   R0,WordVariablePtr
    MOV   A,R6
    MOV   @R0,A
    INC   R0
    MOV   A,R7
    MOV   @R0,A
    RET

WordVariableSub2:
    MOV   R0,WordVariablePtr
    MOV   A,@R0
    MOV   R6,A
    INC   R0
    MOV   A,@R0
    MOV   R7,A
    MOV   R0,#UpdateValue
    MOV   A,@R0
    MOV   R1,A
    MOV   R5,A
    CLR   C
    MOV   A,R7
    SUBB  A,R5
    MOV   R5,A
    MOV   A,R6
    SUBB  A,#00H
    MOV   R4,A
    MOV   R0,#MinValue
    MOV   A,@R0
    MOV   R2,A
    INC   R0
    MOV   A,@R0
    MOV   R3,A
    SETB  C
    MOV   A,R5
    SUBB  A,R3
    MOV   A,R2
    XRL   A,#080H
    MOV   R0,A
    MOV   A,R4
    XRL   A,#080H
    SUBB  A,R0
    JC    ?C0032
    MOV   A,R1
    MOV   R5,A
    CLR   C
    MOV   A,R7
    SUBB  A,R5
    MOV   R7,A
    MOV   A,R6
    SUBB  A,#00H
    MOV   R6,A
    SJMP  ?C0033
?C0032:
    XCH   A,R6
    MOV   A,R2
    XCH   A,R6
    XCH   A,R7
    MOV   A,R3
    XCH   A,R7
?C0033:
    MOV   R0,WordVariablePtr
    MOV   A,R6
    MOV   @R0,A
    INC   R0
    MOV   A,R7
    MOV   @R0,A
    RET

WordProcess:
    MOV   R0,WordVariablePtr
    MOV   A,@R0
    MOV   R6,A
    INC   R0
    MOV   A,@R0
    MOV   R7,A
    JNB   Fgdirection,?C0035
    MOV   R0,#UpdateValue
    MOV   A,@R0
    MOV   R1,A
    MOV   R5,A
```

```
    CLR     C
    MOV     A,R7
    SUBB    A,R5
    MOV     R5,A
    MOV     A,R6
    SUBB    A,#00H
    MOV     R4,A
    MOV     R0,#MinValue
    MOV     A,@R0
    MOV     R2,A
    INC     R0
    MOV     A,@R0
    MOV     R3,A
    SETB    C
    MOV     A,R5
    SUBB    A,R3
    MOV     A,R2
    XRL     A,#080H
    MOV     R0,A
    MOV     A,R4
    XRL     A,#080H
    SUBB    A,R0
    JC      ?C0036
    MOV     A,R1
    MOV     R5,A
    CLR     C
    MOV     A,R7
    SUBB    A,R5
    MOV     R7,A
    MOV     A,R6
    SUBB    A,#00H
    MOV     R6,A
    SJMP    ?C0038
?C0036:
    XCH     A,R6
    MOV     A,R2
    XCH     A,R6
    XCH     A,R7
    MOV     A,R3
    XCH     A,R7
    SJMP    ?C0038
?C0035:
    MOV     R0,#UpdateValue
    MOV     A,@R0
    MOV     R1,A
    ADD     A,R7
    MOV     R5,A
    CLR     A
    ADDC    A,R6
    MOV     R4,A
    MOV     R0,#MaxValue
    MOV     A,@R0
    MOV     R2,A
    INC     R0
    MOV     A,@R0
    MOV     R3,A
    CLR     C
    MOV     A,R5
    SUBB    A,R3
    MOV     A,R2
    XRL     A,#080H
    MOV     R0,A
    MOV     A,R4
    XRL     A,#080H
    SUBB    A,R0
    JNC     ?C0039
    MOV     A,R1
    ADD     A,R7
    MOV     R7,A
    CLR     A
    ADDC    A,R6
    MOV     R6,A
    SJMP    ?C0038
?C0039:
    XCH     A,R6
    MOV     A,R2
    XCH     A,R6
    XCH     A,R7
    MOV     A,R3
    XCH     A,R7
?C0038:
    MOV     R0,WordVariablePtr
    MOV     A,R6
    MOV     @R0,A
    INC     R0
    MOV     A,R7
    MOV     @R0,A
    RET


_Hex2Bcd1:
    MOV     A,R7
    MOV     B,#0AH
    DIV     AB
    MOV     R6,A
    MOV     B,#0AH
    MUL     AB
    MOV     R5,A
    CLR     C
    MOV     A,R7
    SUBB    A,R5
    MOV     R5,A
    MOV     R7,A
    MOV     R0,#KeyData2
    MOV     A,R6
    MOV     @R0,A
    MOV     R0,#KeyData1
    MOV     A,R5
```

```
    MOV     @R0,A
    RET

_Hex2Bcd2:
    XCH     A,R3
    MOV     A,R7
    XCH     A,R3
    XCH     A,R2
    MOV     A,R6
    XCH     A,R2
    MOV     R4,#00H
    MOV     R5,#064H
    LCALL   ?C?UIDIV
    MOV     number3?1255,R7
    MOV     A,R7
    MOV     B,#064H
    MUL     AB
    MOV     R7,A
    CLR     C
    MOV     A,R3
    SUBB    A,R7
    MOV     R7,A
    MOV     A,R2
    SUBB    A,B
    MOV     R6,A
    MOV     R2,A
    XCH     A,R3
    MOV     A,R7
    XCH     A,R3
    MOV     R4,#00H
    MOV     R5,#0AH
    LCALL   ?C?UIDIV
    XCH     A,R1
    MOV     A,R7
    XCH     A,R1
    MOV     A,R1
    MOV     B,#0AH
    MUL     AB
    MOV     R7,A
    CLR     C
    MOV     A,R3
    SUBB    A,R7
    MOV     R7,A
    MOV     A,R2
    SUBB    A,B
    MOV     R2,A
    XCH     A,R3
    MOV     A,R7
    XCH     A,R3
    MOV     R0,#KeyData3
    MOV     @R0,number3?1255
    MOV     R0,#KeyData2
    MOV     A,R1
    MOV     @R0,A
    MOV     R0,#KeyData1
    MOV     A,R7
    MOV     @R0,A
    RET

_Hex2Bcd3:
    MOV     value?1358,R6
    MOV     value?1358+01H,R7
    MOV     R4,#03H
    MOV     R5,#0E8H
    LCALL   ?C?UIDIV
    MOV     number4?1359,R7

    MOV     R6,#00H
    MOV     R4,#03H
    MOV     R5,#0E8H
    LCALL   ?C?IMUL
    CLR     C
    MOV     A,value?1358+01H
    SUBB    A,R7
    MOV     R7,A
    MOV     A,value?1358
    SUBB    A,R6
    MOV     R6,A
    MOV     value?1358,A
    MOV     value?1358+01H,R7
    MOV     R4,#00H
    MOV     R5,#064H
    LCALL   ?C?UIDIV

    XCH     A,R2
    MOV     A,R7
    XCH     A,R2
    MOV     A,R2
    MOV     B,#064H
    MUL     AB
    MOV     R7,A
    CLR     C
    MOV     A,value?1358+01H
    SUBB    A,R7
    MOV     R7,A
    MOV     A,value?1358
    SUBB    A,B
    MOV     R6,A
    MOV     value?1358,A
    MOV     value?1358+01H,R7
    MOV     R4,#00H
    MOV     R5,#0AH
    LCALL   ?C?UIDIV

    XCH     A,R3
    MOV     A,R7
```

```
    XCH     A,R3
    MOV     A,R3
    MOV     B,#0AH
    MUL     AB
    MOV     R7,A
    CLR     C
    MOV     A,value?1358+01H
    SUBB    A,R7
    MOV     R7,A
    MOV     A,value?1358
    SUBB    A,B
    MOV     value?1358,A
    MOV     value?1358+01H,R7
    MOV     R0,#KeyData4
    MOV     @R0,number4?1359
    MOV     R0,#KeyData3
    MOV     A,R2
    MOV     @R0,A
    MOV     R0,#KeyData2
    MOV     A,R3
    MOV     @R0,A
    MOV     R0,#KeyData1
    MOV     A,R7
    MOV     @R0,A
    RET

_Value255_100:
    MOV     R0,#CurrentValue
    MOV     A,R7
    MOV     @R0,A
    CLR     A
    MOV     R0,#MinValue
    MOV     @R0,A
    INC     R0
    MOV     @R0,A
    MOV     R0,#MaxValue
    MOV     @R0,A
    INC     R0
    MOV     @R0,#0FFH
    CLR     C
    MOV     A,R7
    SUBB    A,#00H
    MOV     R7,A
    CLR     A
    SUBB    A,#00H
    MOV     R6,A
    MOV     R4,#00H
    MOV     R5,#064H
    LCALL   ?C?IMUL
    CLR     C
    MOV     A,#0FFH
    SUBB    A,#00H
    MOV     R5,A
    CLR     A
    SUBB    A,#00H
    MOV     R4,A
    LCALL   ?C?UIDIV
    MOV     R0,#CurrentValue
    MOV     A,R7
    MOV     @R0,A
    RET

_Value100_128a:
    MOV     R0,#CurrentValue
    MOV     A,R7
    MOV     @R0,A
    CLR     A
    MOV     R0,#MinValue
    MOV     @R0,A
    INC     R0
    MOV     @R0,A
    MOV     R0,#MaxValue
    MOV     @R0,A
    INC     R0
    MOV     @R0,#064H
    CLR     C
    MOV     A,R7
    SUBB    A,#00H
    MOV     R7,A
    CLR     A
    SUBB    A,#00H
    MOV     R6,A
    MOV     A,R7
    MOV     R0,#07H
?C0049:
    CLR     C
    RLC     A
    XCH     A,R6
    RLC     A
    XCH     A,R6
    DJNZ    R0,?C0049
    MOV     R7,A
    CLR     C
    MOV     A,#064H
    SUBB    A,#00H
    MOV     R5,A
    CLR     A
    SUBB    A,#00H
    MOV     R4,A
    LCALL   ?C?UIDIV
    MOV     R0,#CurrentValue
    MOV     A,R7
    MOV     @R0,A
    RET
```

```
_Value100_128b:
    MOV     A,R7
    SETB    C
    SUBB    A,#064H
    JC      ?C0047
    MOV     R7,#064H
?C0047:
    MOV     R0,#CurrentValue
    MOV     A,R7

    MOV     @R0,A
    CLR     A
    MOV     R0,#MinValue
    MOV     @R0,A
    INC     R0
    MOV     @R0,A
    MOV     R0,#MaxValue
    MOV     @R0,A
    INC     R0
    MOV     @R0,#064H

    MOV     R0,#CurrentValue
    MOV     A,@R0
    CLR     C
    SUBB    A,#00H
    MOV     R7,A
    CLR     A
    SUBB    A,#00H
    MOV     R6,A
    MOV     A,R7
    MOV     R0,#07H
?C0050:
    CLR     C
    RLC     A
    XCH     A,R6
    RLC     A
    XCH     A,R6
    DJNZ    R0,?C0050
    MOV     R7,A
    CLR     C
    MOV     A,#064H
    SUBB    A,#00H
    MOV     R5,A
    CLR     A
    SUBB    A,#00H
    MOV     R4,A
    LCALL   ?C?UIDIV
    MOV     R0,#CurrentValue
    MOV     A,R7
    MOV     @R0,A
    RET

?RamClear:
    MOV     R7,#LOW (address)
    CLR     A
    MOV     R6,A
?C0001:
    XCH     A,R0

    MOV     A,R7
    XCH     A,R0
    CLR     A
    MOV     @R0,A
    INC     R7
    INC     R6
    CJNE    R6,#0E0H,?C0001
    RET


_ZeroContinue:
    CLR     A
    MOV     R6,A
?C0005:
    MOV     A,#TrmBuf
    ADD     A,R7
    MOV     R0,A
    MOV     A,@R0
    JNZ     ?C0010
    INC     R7
    MOV     A,R7
    SETB    C
    SUBB    A,#04H
    JC      ?C0007
    MOV     R7,#01H
?C0007:
    INC     R6
    CJNE    R6,#04H,?C0005
?C0010:
    RET


LedFlash0:
    CLR     P1_0
    RET


LedFlash1:
    SETB    P1_0
    RET


LedFlash2:
    LCALL   LedOn
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   LedOff
```

```
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    RET

_LedFlash3:
?C0004:
    LCALL   LedOn
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   LedOff
    MOV     R7,#032H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0004
    RET

_LedFlash4:
    MOV     ontime?440,R7
    MOV     offtime?441,R5
    LCALL   LedOn
    MOV     R7,ontime?440
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   LedOff
    MOV     R7,offtime?441
    MOV     R6,#00H
    LCALL   _DelayX10ms
    RET

_LedFlash5:
    MOV     count?542,R7
    MOV     ontime?543,R5
    MOV     offtime?544,R3
    CLR     A
    MOV     i?545,A
?C0008:
    MOV     A,i?545
    CLR     C
    SUBB    A,count?542
    JNC     ?C0011
    LCALL   LedOn
    MOV     R7,ontime?543
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   LedOff
    MOV     R7,offtime?544
    MOV     R6,#00H
    LCALL   _DelayX10ms
    INC     i?545
    SJMP    ?C0008
?C0011:
    RET

_LedFlash6:
    MOV     count?646,R7
    MOV     ontime?647,R5
    MOV     offtime?648,R3
    CLR     TR1
    CLR     A
    MOV     i?649,A
?C0012:
    MOV     A,i?649
    CLR     C
    SUBB    A,count?646
    JNC     ?C0013
    LCALL   LedOn
    MOV     R7,ontime?647
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   LedOff
    MOV     R7,offtime?648
    MOV     R6,#00H
    LCALL   _DelayX10ms
    INC     i?649
    SJMP    ?C0012
?C0013:
    SETB    TR1
    RET

_LedFlashGetkey:
    MOV     count?750,R7
    MOV     ontime?751,R5
    MOV     offtime?752,R3
    CLR     TR1
    CLR     A
    MOV     i?753,A
?C0016:
    MOV     A,i?753
    CLR     C
    SUBB    A,count?750
    JNC     ?C0017
    LCALL   LedOn
    MOV     R7,ontime?751
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   GetKey2
    MOV     R0,#KeyData
    MOV     A,@R0
    JNZ     ?C0017
?C0019:
    LCALL   LedOff
```

```
    MOV     R7,offtime?752
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   GetKey2
    MOV     R0,#KeyData
    MOV     A,@R0
    JNZ     ?C0017
?C0018:
    INC     i?753
    SJMP    ?C0016
?C0017:
    LCALL   LedOff
    SETB    TR1
    RET


LedMain1:
    LCALL   Subroute1
    LCALL   LedOn
    MOV     R7,#0F4H
    MOV     R6,#01H
    LCALL   _DelayX10ms
    LCALL   LedOff
    LCALL   Subroute2
    RET


LedMain2:
    CLR     A
    MOV     IE,A
    MOV     IP,#0BH
    MOV     TMOD,#015H
    MOV     TL1,#0C0H
    MOV     TH1,#063H
    SETB    ET1
    SETB    EA
    LCALL   LedOn
    CLR     FgLedFlag
    MOV     R0,#LedOffCount
    MOV     @R0,#00H
    INC     R0
    MOV     @R0,#07DH
    SETB    TR1
?C0023:
    LCALL   Subroute1
    JB      FgLedFlag,?C0025
    MOV     R0,#LedOffCount+01H
    MOV     A,@R0
    DEC     R0
    ORL     A,@R0
    JNZ     ?C0025
    SETB    FgLedFlag
    LCALL   LedOff
?C0025:
    LCALL   Subroute2
    SJMP    ?C0023
    RET


LedOff_Timer1ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW,#010H
    MOV     TL1,#0C0H
    MOV     TH1,#063H
    CLR     TF1
    MOV     R0,#Timer40msCount+01H
    INC     @R0
    MOV     A,@R0
    JNZ     ?C0106
    DEC     R0
    INC     @R0
?C0106:
    MOV     R0,#LedOffCount+01H
    MOV     A,@R0
    DEC     R0
    ORL     A,@R0
    JZ      ?C0107
    INC     R0
    MOV     A,@R0
    DEC     @R0
    JNZ     ?C0107
    DEC     R0
    DEC     @R0
?C0107:
    POP     PSW
    POP     ACC
    RETI


LedTimming:
    CLR     A
    MOV     IE,A
    MOV     IP,#0BH
    MOV     TMOD,#015H
    MOV     TL1,#0C0H
    MOV     TH1,#063H
    SETB    ET1
    SETB    EA
    LCALL   Led1On
    CLR     FgLed2On
    CLR     FgChange
    MOV     R0,#LedOffCount
    MOV     @R0,#0011
    INC     R0
    MOV     @R0,#07DH
    SETB    TR1
?C0030:
```

```
    JB      FgChange,?C0032
    JB      FgLed2On,?C0033
    MOV     R0,#LedOffCount+01H
    MOV     A,@R0
    DEC     R0
    ORL     A,@R0
    JNZ     ?C0030
    LCALL   Led2On
    SETB    FgLed2On
    CLR     FgLed3On
    MOV     R0,#LedOffCount
    MOV     @R0,#00H
    INC     R0
    MOV     @R0,#04BH
    SJMP    ?C0030
?C0033:
    JB      FgLed3On,?C0030

    MOV     R0,#LedOffCount+01H
    MOV     A,@R0
    DEC     R0
    ORL     A,@R0
    JNZ     ?C0030
    LCALL   Led3On
    SETB    FgLed3On
    CLR     FgLed2Off
    MOV     R7,#0F4H
    MOV     R6,#01H
    LCALL   _DelayX10ms
    SETB    FgChange
    LCALL   Led3Off
    CLR     FgLed2Off
    SETB    FgLed1Off
    MOV     R0,#LedOffCount
    MOV     @R0,#00H
    INC     R0
    MOV     @R0,#032H
    SJMP    ?C0030
?C0032:
    JNB     FgChange,?C0030
    JB      FgLed2Off,?C0040
    MOV     R0,#LedOffCount+01H
    MOV     A,@R0
    DEC     R0
    ORL     A,@R0
    JNZ     ?C0030
    LCALL   Led2Off
    SETB    FgLed2Off
    CLR     FgLed1Off
    MOV     R0,#LedOffCount
    MOV     @R0,#00H
    INC     R0
    MOV     @R0,#019H
    SJMP    ?C0030

?C0040:
    JB      FgLed1Off,?C0030
    MOV     R0,#LedOffCount+01H
    MOV     A,@R0
    DEC     R0
    ORL     A,@R0
    JNZ     ?C0030
    LCALL   Led1Off
    SETB    FgLed1Off
    CLR     FgChange
    SJMP    ?C0030
    RET


Led1On:
    SETB    P1_4
    RET

Led1Off:
    CLR     P1_4
    RET

Led2On:
    SETB    P1_5
    RET

Led2Off:
    CLR     P1_5
    RET

Led3On:
    SETB    P1_6
    RET

Led3Off:
    CLR     P1_6
    RET

Led4On:
    SETB    P1_7
    RET

Led4Off:
    CLR     P1_7
    RET

LedMain3:
    CLR     A
    MOV     R0,#LedCount
```

```
    MOV   @R0,A
?C0054:
    LCALL Subroute1
    LCALL LedDelayFlash
    MOV   R7,#028H
    MOV   R6,#00H
    LCALL _DelayX1ms
    MOV   R0,#LedCount
    INC   @R0
    LCALL Subroute2
    SJMP  ?C0054
    RET

LedDelayFlash:
    MOV   R0,#LedCount
    MOV   A,@R0
    CLR   C
    SUBB  A,#01AH
    JNC   ?C0057
    LCALL LedOn
    RET
?C0057:
    MOV   R0,#LedCount
    MOV   A,@R0
    CLR   C
    SUBB  A,#032H
    JNC   ?C0059
    LCALL LedOff
    RET
?C0059:
    CLR   A
    MOV   R0,#LedCount
    MOV   @R0,A
    RET

LedMain4:
    CLR   A
    MOV   IE,A
    MOV   IP,#0BH
    MOV   TMOD,#015H
    MOV   TL1,#0C0H
    MOV   TH1,#063H
    SETB  ET1
    SETB  EA
    SETB  TR1
    MOV   R0,#LedCount
    MOV   @R0,A
?C0062:
    LCALL Subroute1
    LCALL LedTimerFlash
    LCALL Subroute2
    SJMP  ?C0062
    RET

LedTimerFlash:
    MOV   R0,#LedCount
    MOV   A,@R0
    CLR   C
    SUBB  A,#01AH
    JNC   ?C0065
    LCALL LedOn
    RET
?C0065:
    MOV   R0,#LedCount
    MOV   A,@R0
    CLR   C
    SUBB  A,#032H
    JNC   ?C0067
    LCALL LedOff
    RET
?C0067:
    CLR   A
    MOV   R0,#LedCount
    MOV   @R0,A
    RET

Led_Timer1ISR:
    PUSH  PSW
    MOV   PSW,#010H
    MOV   TL1,#0C0H
    MOV   TH1,#063H
    CLR   TF1
    MOV   R0,#LedCount
    INC   @R0
    POP   PSW
    RETI

LedMain5:
?C0071:
    JB    P2_0,?C0073
    LCALL Led_1Flash
    SJMP  ?C0071
?C0073:
    JB    P2_1,?C0075
    LCALL Led_2Flash
    SJMP  ?C0071
?C0075:
    JB    P2_2,?C0071
    LCALL Led_3Flash
    SJMP  ?C0071
    RET
```

```
Led_1Flash:
    MOV     R0,#LedCount
    MOV     A,@R0
    CLR     C
    SUBB    A,#06H
    JNC     ?C0079
    LCALL   LedOn
    RET
?C0079:
    MOV     R0,#LedCount
    MOV     A,@R0
    CLR     C
    SUBB    A,#03AH
    JNC     ?C0081
    LCALL   LedOff
    RET
?C0081:
    CLR     A
    MOV     R0,#LedCount
    MOV     @R0,A
    RET


Led_2Flash:
    MOV     R0,#LedCount
    MOV     A,@R0
    CLR     C
    SUBB    A,#06H
    JNC     ?C0084
    LCALL   LedOn
    RET
?C0084:
    MOV     R0,#LedCount
    MOV     A,@R0
    CLR     C
    SUBB    A,#0BH
    JNC     ?C0086
    LCALL   LedOff
    RET
?C0086:
    MOV     R0,#LedCount
    MOV     A,@R0
    CLR     C
    SUBB    A,#010H
    JNC     ?C0088
    LCALL   LedOn
    RET
?C0088:
    MOV     R0,#LedCount
    MOV     A,@R0
    CLR     C
    SUBB    A,#044H
    JNC     ?C0090
    LCALL   LedOff
    RET
?C0090:
    CLR     A
    MOV     R0,#LedCount
    MOV     @R0,A
    RET


Led_3Flash:
    MOV     R0,#LedCount
    MOV     A,@R0
    CLR     C
    SUBB    A,#06H
    JNC     ?C0093
    LCALL   LedOn
    RET
?C0093:
    MOV     R0,#LedCount
    MOV     A,@R0
    CLR     C
    SUBB    A,#0BH
    JNC     ?C0095
    LCALL   LedOff
    RET
?C0095:
    MOV     R0,#LedCount
    MOV     A,@R0
    CLR     C
    SUBB    A,#010H
    JNC     ?C0097
    LCALL   LedOn
    RET
?C0097:
    MOV     R0,#LedCount
    MOV     A,@R0
    CLR     C
    SUBB    A,#015H
    JNC     ?C0099
    LCALL   LedOff
    RET
?C0099:
    MOV     R0,#LedCount
    MOV     A,@R0
    CLR     C
    SUBB    A,#01AH
    JNC     ?C0101
    LCALL   LedOn
    RET
?C0101:
    MOV     R0,#LedCount
    MOV     A,@R0
    CLR     C
    SUBB    A,#04EH
    JNC     ?C0103
```

```
    LCALL  LedOff
    RET
?C0103:
    CLR    A
    MOV    R0,#LedCount
    MOV    @R0,A
    RET
Beep0:
    CLR    P1_0
    RET
Beep1:
    CLR    A
    MOV    R7,A
    MOV    R6,A
?C0002:
    CLR    A
    MOV    R5,A
    MOV    R4,A
?C0005:
    CLR    P1_0
    INC    R5
    CJNE   R5,#00H,?C0135
    INC    R4
?C0135:
    MOV    A,R5
    XRL    A,#032H
    ORL    A,R4
    JNZ    ?C0005
?C0006:
    CLR    A
    MOV    R4,A
    MOV    R5,A
?C0008:
    SETB   P1_0
    INC    R5
    CJNE   R5,#00H,?C0136
    INC    R4
?C0136:
    MOV    A,R5
    XRL    A,#032H
    ORL    A,R4
    JNZ    ?C0008
?C0004:
    INC    R7
    CJNE   R7,#00H,?C0137
    INC    R6
?C0137:
    MOV    A,R7
    XRL    A,#011H
    ORL    A,R6
    JNZ    ?C0002
?C0011:
    RET
_Beep2:
    MOV    A,R7
    MOV    R5,A
    MOV    R4,#00H
    MOV    R6,#03H
    MOV    R7,#0E8H
    LCALL  ?C?UIDIV
    MOV    A,R6
    CLR    C
    RRC    A
    MOV    R6,A
    MOV    A,R7
    RRC    A
    MOV    R7,A
    CLR    A
    MOV    R5,A
    MOV    R4,A
?C0012:
    CLR    A
    MOV    R3,A
    MOV    R2,A
?C0015:
    CLR    C
    MOV    A,R3
    SUBB   A,R7
    MOV    A,R2
    SUBB   A,R6
    JNC    ?C0016
    CLR    P1_0
    INC    R3
    CJNE   R3,#00H,?C0138
    INC    R2
?C0138:
    SJMP   ?C0015
?C0016:
    CLR    A
    MOV    R2,A
    MOV    R3,A
?C0018:
    CLR    C
    MOV    A,R3
    SUBB   A,R7
    MOV    A,R2
    SUBB   A,R6
    JNC    ?C0014
    SETB   P1_0
    INC    R3
    CJNE   R3,#00H,?C0139
    INC    R2
?C0139:
    SJMP   ?C0018
?C0014:
    INC    R5
```

```
    CJNE    R5,#00H,?C0140
    INC     R4
?C0140:
    MOV     A,R5
    XRL     A,#01H
    ORL     A,R4
    JNZ     ?C0012
?C0021:
    RET


_Beep3:
    MOV     soundlong?346,R7
    MOV     A,R5
    MOV     R4,#00H
    MOV     R6,#03H
    MOV     R7,#0E8H
    LCALL   ?C?UIDIV
    MOV     A,R6
    CLR     C
    RRC     A
    MOV     R6,A
    MOV     A,R7
    RRC     A
    MOV     R7,A
    CLR     A
    MOV     R5,A
    MOV     R4,A
?C0022:
    CLR     C
    MOV     A,R5
    SUBB    A,soundlong?346
    MOV     A,R4
    SUBB    A,#00H
    JNC     ?C0031
    CLR     A
    MOV     R3,A
    MOV     R2,A
?C0025:
    CLR     C
    MOV     A,R3
    SUBB    A,R7
    MOV     A,R2
    SUBB    A,R6
    JNC     ?C0026
    CLR     P1_0
    INC     R3
    CJNE    R3,#00H,?C0141
    INC     R2
?C0141:
    SJMP    ?C0025
?C0026:
    CLR     A
    MOV     R2,A
    MOV     R3,A
?C0028:
    CLR     C
    MOV     A,R3
    SUBB    A,R7
    MOV     A,R2
    SUBB    A,R6
    JNC     ?C0024
    SETB    P1_0
    INC     R3
    CJNE    R3,#00H,?C0142
    INC     R2
?C0142:
    SJMP    ?C0028
?C0024:
    INC     R5
    CJNE    R5,#00H,?C0143
    INC     R4
?C0143:
    SJMP    ?C0022
?C0031:
    RET


_Beep4:
    MOV     count?451,R6
    MOV     count?451+01H,R7
    MOV     soundlong?452,R5
    MOV     A,R3
    MOV     R5,A
    MOV     R4,#00H
    MOV     R6,#03H
    MOV     R7,#0E8H
    LCALL   ?C?UIDIV
    MOV     A,R6
    CLR     C
    RRC     A
    MOV     SpFreq?457,A
    MOV     A,R7
    RRC     A
    MOV     SpFreq?457+01H,A
    CLR     A
    MOV     i?454,A
    MOV     i?454+01H,A
?C0032:
    CLR     C
    MOV     A,i?454+01H
    SUBB    A,count?451+01H
    MOV     A,i?454
    SUBB    A,count?451
    JNC     ?C0044
    CLR     A
    MOV     R7,A
    MOV     R6,A
```

```
?C0035:
    CLR     C
    MOV     A,R7
    SUBB    A,soundlong?452
    MOV     A,R6
    SUBB    A,#00H
    JNC     ?C0036
    CLR     A
    MOV     R5,A
    MOV     R4,A
?C0038:
    CLR     C
    MOV     A,R5
    SUBB    A,SpFreq?457+01H
    MOV     A,R4
    SUBB    A,SpFreq?457
    JNC     ?C0039
    CLR     P1_0
    INC     R5
    CJNE    R5,#00H,?C0144
    INC     R4
?C0144:
    SJMP    ?C0038
?C0039:
    CLR     A
    MOV     R4,A
    MOV     R5,A
?C0041:
    CLR     C
    MOV     A,R5
    SUBB    A,SpFreq?457+01H
    MOV     A,R4
    SUBB    A,SpFreq?457
    JNC     ?C0037
    SETB    P1_0
    INC     R5
    CJNE    R5,#00H,?C0145
    INC     R4
?C0145:
    SJMP    ?C0041
?C0037:
    INC     R7
    CJNE    R7,#00H,?C0146
    INC     R6
?C0146:
    SJMP    ?C0035
?C0036:
    MOV     R7,#0CH
    MOV     R6,#00H
    LCALL   _DelayX10ms
    INC     i?454+01H
    MOV     A,i?454+01H
    JNZ     ?C0147
    INC     i?454
?C0147:
    SJMP    ?C0032
?C0044:
    RET

_BeepGetkey:
    MOV     count?558,R6
    MOV     count?558+01H,R7
    MOV     soundlong?559,R5
    CLR     FgBeepOff
    MOV     A,R3
    MOV     R5,A
    MOV     R4,#00H
    MOV     R6,#03H
    MOV     R7,#0E8H
    LCALL   ?C?UIDIV
    MOV     A,R6
    CLR     C
    RRC     A
    MOV     SpFreq?564,A
    MOV     A,R7
    RRC     A
    MOV     SpFreq?564+01H,A
    CLR     A
    MOV     i?561,A
    MOV     i?561+01H,A
?C0045:
    CLR     C
    MOV     A,i?561+01H
    SUBB    A,count?558+01H
    MOV     A,i?561
    SUBB    A,count?558
    JNC     ?C0059
    CLR     A
    MOV     R7,A
    MOV     R6,A
?C0048:
    CLR     C
    MOV     A,R7
    SUBB    A,soundlong?559
    MOV     A,R6
    SUBB    A,#00H
    JNC     ?C0049
    CLR     A
    MOV     R5,A
    MOV     R4,A
?C0051:
    CLR     C
    MOV     A,R5
    SUBB    A,SpFreq?564+01H
    MOV     A,R4
    SUBB    A,SpFreq?564
    JNC     ?C0052
```

```
    CLR     P1_0
    INC     R5
    CJNE    R5,#00H,?C0148
    INC     R4
?C0148:
    SJMP    ?C0051
?C0052:
    CLR     A
    MOV     R4,A
    MOV     R5,A
?C0054:
    CLR     C
    MOV     A,R5
    SUBB    A,SpFreq?564+01H
    MOV     A,R4
    SUBB    A,SpFreq?564
    JNC     ?C0050
    SETB    P1_0
    INC     R5
    CJNE    R5,#00H,?C0149
    INC     R4
?C0149:
    SJMP    ?C0054
?C0050:
    INC     R7
    CJNE    R7,#00H,?C0150
    INC     R6
?C0150:
    SJMP    ?C0048
?C0049:
    MOV     R7,#0CH
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   GetKey2
    MOV     R0,#KeyData
    MOV     A,@R0
    MOV     R7,A
    XRL     A,#01H
    JZ      ?C0058
    MOV     A,R7
    CJNE    A,#0AH,?C0047
?C0058:
    SETB    FgBeepOff
    RET
?C0047:
    INC     i?561+01H
    MOV     A,i?561+01H
    JNZ     ?C0151
    INC     i?561
?C0151:
    SJMP    ?C0045
?C0059:
    RET

_Alarm1:
    MOV     soundlong?665,R7
    MOV     A,R5
    MOV     R4,#00H
    MOV     R6,#03H
    MOV     R7,#0E8H
    LCALL   ?C?UIDIV
    MOV     A,R6
    CLR     C
    RRC     A
    MOV     SpFreq?669,A
    MOV     A,R7
    RRC     A
    MOV     SpFreq?669+01H,A
?C0060:
    CLR     A
    MOV     R7,A
    MOV     R6,A
?C0062:
    CLR     C
    MOV     A,R7
    SUBB    A,soundlong?665
    MOV     A,R6
    SUBB    A,#00H
    JNC     ?C0063
    CLR     A
    MOV     R5,A
    MOV     R4,A
?C0065:
    CLR     C
    MOV     A,R5
    SUBB    A,SpFreq?669+01H
    MOV     A,R4
    SUBB    A,SpFreq?669
    JNC     ?C0066
    CLR     P1_0
    INC     R5
    CJNE    R5,#00H,?C0152
    INC     R4
?C0152:
    SJMP    ?C0065
?C0066:
    CLR     A
    MOV     R4,A
    MOV     R5,A
?C0068:
    CLR     C
    MOV     A,R5
    SUBB    A,SpFreq?669+01H
    MOV     A,R4
    SUBB    A,SpFreq?669
    JNC     ?C0064
    SETB    P1_0
```

```
    INC     R5
    CJNE    R5,#00H,?C0153
    INC     R4
?C0153:
    SJMP    ?C0068
?C0064:
    INC     R7
    CJNE    R7,#00H,?C0154
    INC     R6
?C0154:
    SJMP    ?C0062
?C0063:
    MOV     R7,#0CH
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0060
    RET

Alarm2:
    MOV     count?770,R6
    MOV     count?770+01H,R7
    MOV     soundlong?771,R5
    MOV     A,R3
    MOV     R5,A
    MOV     R4,#00H
    MOV     R6,#03H
    MOV     R7,#0E8H
    LCALL   ?C?UIDIV
    MOV     A,R6
    CLR     C
    RRC     A
    MOV     SpFreq?776,A
    MOV     A,R7
    RRC     A
    MOV     SpFreq?776+01H,A
?C0072:
    CLR     A
    MOV     j?774,A
    MOV     j?774+01H,A
?C0074:
    CLR     C
    MOV     A,j?774+01H
    SUBB    A,count?770+01H
    MOV     A,j?774
    SUBB    A,count?770
    JNC     ?C0075
    CLR     A
    MOV     R7,A
    MOV     R6,A
?C0077:
    CLR     C
    MOV     A,R7
    SUBB    A,soundlong?771
    MOV     A,R6
    SUBB    A,#00H
    JNC     ?C0078
    CLR     A
    MOV     R5,A
    MOV     R4,A
?C0080:
    CLR     C
    MOV     A,R5
    SUBB    A,SpFreq?776+01H
    MOV     A,R4
    SUBB    A,SpFreq?776
    JNC     ?C0081
    CLR     P1_0
    INC     R5
    CJNE    R5,#00H,?C0155
    INC     R4
?C0155:
    SJMP    ?C0080
?C0081:
    CLR     A
    MOV     R4,A
    MOV     R5,A
?C0083:
    CLR     C
    MOV     A,R5
    SUBB    A,SpFreq?776+01H
    MOV     A,R4
    SUBB    A,SpFreq?776
    JNC     ?C0079
    SETB    P1_0
    INC     R5
    CJNE    R5,#00H,?C0156
    INC     R4
?C0156:
    SJMP    ?C0083
?C0079:
    INC     R7
    CJNE    R7,#00H,?C0157
    INC     R6
?C0157:
    SJMP    ?C0077
?C0078:
    MOV     R7,#0CH
    MOV     R6,#00H
    LCALL   _DelayX10ms
    INC     j?774+01H
    MOV     A,j?774+01H
    JNZ     ?C0158
    INC     j?774
?C0158:
    SJMP    ?C0074
?C0075:
    MOV     R7,#064H
```

```
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0072
    RET

_AlarmGetkey:
    MOV     count?877,R6
    MOV     count?877+01H,R7
    MOV     soundlong?878,R5
    CLR     FgBeepOff
    CLR     FgAlarmOff
    MOV     A,R3
    MOV     R5,A
    MOV     R4,#00H
    MOV     R6,#03H
    MOV     R7,#0E8H
    LCALL   ?C?UIDIV
    MOV     A,R6
    CLR     C
    RRC     A
    MOV     SpFreq?883,A
    MOV     A,R7
    RRC     A
    MOV     SpFreq?883+01H,A
?C0087:
    CLR     A
    MOV     j?881,A
    MOV     j?881+01H,A
?C0089:
    CLR     C
    MOV     A,j?881+01H
    SUBB    A,count?877+01H
    MOV     A,j?881
    SUBB    A,count?877
    JNC     ?C0090
    CLR     A
    MOV     R7,A
    MOV     R6,A
?C0092:
    CLR     C
    MOV     A,R7
    SUBB    A,soundlong?878
    MOV     A,R6
    SUBB    A,#00H
    JNC     ?C0093
    CLR     A
    MOV     R5,A
    MOV     R4,A
?C0095:
    CLR     C
    MOV     A,R5
    SUBB    A,SpFreq?883+01H
    MOV     A,R4
    SUBB    A,SpFreq?883
    JNC     ?C0096
    CLR     P1_0
    INC     R5
    CJNE    R5,#00H,?C0159
    INC     R4
?C0159:
    SJMP    ?C0095
?C0096:
    CLR     A
    MOV     R4,A
    MOV     R5,A
?C0098:
    CLR     C
    MOV     A,R5
    SUBB    A,SpFreq?883+01H
    MOV     A,R4
    SUBB    A,SpFreq?883
    JNC     ?C0094
    SETB    P1_0
    INC     R5
    CJNE    R5,#00H,?C0160
    INC     R4
?C0160:
    SJMP    ?C0098
?C0094:
    INC     R7
    CJNE    R7,#00H,?C0161
    INC     R6
?C0161:
    SJMP    ?C0092
?C0093:
    MOV     R7,#0CH
    MOV     R6,#00H
    LCALL   _DelayX10ms
    LCALL   GetKey2
    MOV     R0,#KeyData
    MOV     A,@R0
    MOV     R7,A
    XRL     A,#01H
    JZ      ?C0102
    MOV     A,R7
    CJNE    A,#0AH,?C0091
?C0102:
    SETB    FgBeepOff
    SJMP    ?C0090
?C0091:
    INC     j?881+01H
    MOV     A,j?881+01H
    JNZ     ?C0162

    INC     j?881
?C0162:
    SJMP    ?C0089
```

```
?C0090:
    JNB     FgBeepOff,?C0103
    SETB    FgAlarmOff
    RET
?C0103:
    MOV     R7,#064H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    SJMP    ?C0087
?C0104:
    RET

_BeepLed:
    MOV     count?984,R6
    MOV     count?984+01H,R7
    MOV     soundlong?985,R5
    MOV     A,R3
    MOV     R5,A
    MOV     R4,#00H
    MOV     R6,#03H
    MOV     R7,#0E8H
    LCALL   ?C?UIDIV
    MOV     A,R6
    CLR     C
    RRC     A
    MOV     SpFreq?990,A
    MOV     A,R7
    RRC     A
    MOV     SpFreq?990+01H,A
    CLR     A
    MOV     i?987,A
    MOV     i?987+01H,A
?C0105:
    CLR     C
    MOV     A,i?987+01H
    SUBB    A,count?984+01H
    MOV     A,i?987
    SUBB    A,count?984
    JNC     ?C0117
    SETB    P1_1
    CLR     A
    MOV     R7,A
    MOV     R6,A
?C0108:
    CLR     C
    MOV     A,R7
    SUBB    A,soundlong?985
    MOV     A,R6
    SUBB    A,#00H
    JNC     ?C0109
    CLR     A
    MOV     R5,A
    MOV     R4,A
?C0111:
    CLR     C
    MOV     A,R5
    SUBB    A,SpFreq?990+01H
    MOV     A,R4
    SUBB    A,SpFreq?990
    JNC     ?C0112
    CLR     P1_0
    INC     R5
    CJNE    R5,#00H,?C0163
    INC     R4
?C0163:
    SJMP    ?C0111
?C0112:
    CLR     A
    MOV     R4,A
    MOV     R5,A
?C0114:
    CLR     C
    MOV     A,R5
    SUBB    A,SpFreq?990+01H
    MOV     A,R4
    SUBB    A,SpFreq?990
    JNC     ?C0110
    SETB    P1_0
    INC     R5
    CJNE    R5,#00H,?C0164
    INC     R4
?C0164:
    SJMP    ?C0114
?C0110:
    INC     R7
    CJNE    R7,#00H,?C0165
    INC     R6
?C0165:
    SJMP    ?C0108
?C0109:
    CLR     P1_1
    MOV     R7,#0CH
    MOV     R6,#00H
    LCALL   _DelayX10ms
    INC     i?987+01H
    MOV     A,i?987+01H
    JNZ     ?C0166
    INC     i?987
?C0166:
    SJMP    ?C0105
?C0117:
    RET

HardWareBeep1:
    SETB    P1_0
    RET
```

```
HardWareBeep2:
    JNB     FgExtFreq1,?C0119
    SETB    P1_0
    CLR     P1_1
    CLR     P1_2
    CLR     P1_3
    RET
?C0119:
    JNB     FgExtFreq2,?C0121
    CLR     P1_0
    SETB    P1_1
    CLR     P1_2
    CLR     P1_3
    RET
?C0121:
    JNB     FgExtFreq3,?C0123
    CLR     P1_0
    CLR     P1_1
    SETB    P1_2
    CLR     P1_3
    RET
?C0123:
    JNB     FgExtFreq4,?C0126
    CLR     P1_0
    CLR     P1_1
    CLR     P1_2
    SETB    P1_3
?C0126:
    RET


HardWareBeep3:
    CLR     P1_0
    CLR     P1_1
    CLR     P1_2
    CLR     P1_3
    JNB     FgExtFreq1,?C0127
    SETB    P1_0
    RET
?C0127:
    JNB     FgExtFreq2,?C0129
    SETB    P1_1
    RET
?C0129:
    JNB     FgExtFreq3,?C0131
    SETB    P1_2
    RET
?C0131:
    JNB     FgExtFreq4,?C0134
    SETB    P1_3
?C0134:
    RET
```

```
Sound:
    CLR     A
    MOV     i?040,A
?C0001:
    MOV     A,i?040
    MOV     DPTR,#SOUND_LONG
    MOVC    A,@A+DPTR
    MOV     R7,A

    MOV     A,i?040
    MOV     DPTR,#SOUND_TONE
    MOVC    A,@A+DPTR
    MOV     R6,A
    CLR     A
    MOV     R5,A
?C0004:
    MOV     A,R5
    CLR     C
    SUBB    A,R7
    JNC     ?C0005
    CLR     A
    MOV     R4,A
?C0007:
    CLR     A
    MOV     R3,A
?C0010:
    MOV     A,R6
    MOV     B,#03H
    DIV     AB
    MOV     R2,A
    MOV     A,R3
    CLR     C
    SUBB    A,R2
    JNC     ?C0011
    CLR     P1_0
    INC     R3
    SJMP    ?C0010
?C0011:
    CLR     A
    MOV     R3,A
?C0013:
    MOV     A,R6
    MOV     B,#03H
    DIV     AB
    MOV     R2,A
    MOV     A,R3
    CLR     C
    SUBB    A,R2
    JNC     ?C0009
    SETB    P1_0
    INC     R3
    SJMP    ?C0013
```

```
?C0009:
    INC     R4
    CJNE    R4,#08H,?C0007
?C0006:
    INC     R5
    SJMP    ?C0004
?C0005:
    MOV     R7,#06H
    LCALL   _Delay50uS
    INC     i?040
    MOV     A,i?040
    CLR     C
    SUBB    A,#0FH
    JC      ?C0001
    RET


Music1:
    CLR     A
    MOV     i?146,A
?C0017:
    MOV     A,i?146
    MOV     DPTR,#MUSIC_SOUNDLONG
    MOVC    A,@A+DPTR
    MOV     SoundLong?149,A
    MOV     A,i?146
    MOV     DPTR,#MUSIC_SOUNDTONE
    MOVC    A,@A+DPTR
    MOV     R1,A
    CLR     A
    MOV     R7,A
?C0020:
    MOV     A,R7
    CLR     C
    SUBB    A,SoundLong?149
    JNC     ?C0021
    CLR     A
    MOV     R6,A
?C0023:
    CLR     A
    MOV     R5,A
    MOV     R4,A
?C0026:
    MOV     A,R1
    MOV     R3,A
    CLR     C
    MOV     A,R5
    SUBB    A,R3
    MOV     A,R4
    SUBB    A,#00H
    JNC     ?C0027
    CLR     P1_0
    INC     R5
    CJNE    R5,#00H,?C0091
    INC     R4
?C0091:
    SJMP    ?C0026
?C0027:
    CLR     A
    MOV     R4,A
    MOV     R5,A
?C0029:
    MOV     A,R1
    MOV     R3,A
    CLR     C
    MOV     A,R5
    SUBB    A,R3
    MOV     A,R4
    SUBB    A,#00H
    JNC     ?C0025
    SETB    P1_0
    INC     R5
    CJNE    R5,#00H,?C0092
    INC     R4
?C0092:
    SJMP    ?C0029
?C0025:
    INC     R6
    CJNE    R6,#0CH,?C0023
?C0022:
    INC     R7
    SJMP    ?C0020
?C0021:
    MOV     R7,#06H
    LCALL   _Delay50uS
    INC     i?146
    MOV     A,i?146
    CLR     C
    SUBB    A,#01FH
    JC      ?C0017
    RET


Music2:
    CLR     A
    MOV     i?252,A
?C0035:
    MOV     A,i?252
    MOV     DPTR,#MUSIC_SOUNDLONG1
    MOVC    A,@A+DPTR
    MOV     SoundLong?255,A
    MOV     A,i?252
    MOV     DPTR,#MUSIC_SOUNDTONE1
    MOVC    A,@A+DPTR
    MOV     R1,A
    INC     i?252
    CLR     A
    MOV     R7,A
```

```
?C0036:
    MOV     A,R7
    CLR     C
    SUBB    A,SoundLong?255
    JNC     ?C0037
    CLR     A
    MOV     R6,A
?C0039:
    CLR     A
    MOV     R5,A
    MOV     R4,A
?C0042:
    MOV     A,R1
    MOV     R3,A
    CLR     C
    MOV     A,R5
    SUBB    A,R3
    MOV     A,R4
    SUBB    A,#00H
    JNC     ?C0043
    CLR     P1_0
    INC     R5
    CJNE    R5,#00H,?C0093
    INC     R4
?C0093:
    SJMP    ?C0042
?C0043:
    CLR     A
    MOV     R4,A
    MOV     R5,A
?C0045:
    MOV     A,R1
    MOV     R3,A
    CLR     C
    MOV     A,R5
    SUBB    A,R3
    MOV     A,R4
    SUBB    A,#00H
    JNC     ?C0041
    SETB    P1_0
    INC     R5
    CJNE    R5,#00H,?C0094
    INC     R4
?C0094:
    SJMP    ?C0045
?C0041:
    INC     R6
    CJNE    R6,#0CH,?C0039
?C0038:
    INC     R7
    SJMP    ?C0036
?C0037:
    MOV     R7,#06H
    LCALL   _Delay50uS
    MOV     A,i?252
    MOV     DPTR,#MUSIC_SOUNDTONE1
    MOVC    A,@A+DPTR
    JNZ     ?C0035
    RET

Music3:
    CLR     A
    MOV     i?358,A
?C0049:
    MOV     A,i?358
    MOV     DPTR,#MUSIC_SOUNDLONG
    MOVC    A,@A+DPTR
    MOV     R1,A
    MOV     A,i?358
    ADD     A,ACC
    ADD     A,#LOW (MUSIC_SOUNDTONE2)
    MOV     DPL,A
    CLR     A
    ADDC    A,#HIGH (MUSIC_SOUNDTONE2)
    MOV     DPH,A
    CLR     A
    MOVC    A,@A+DPTR
    MOV     R7,#00H
    MOV     R6,A
    MOV     A,i?358
    ADD     A,ACC
    ADD     A,#LOW
(MUSIC_SOUNDTONE2+01H)
    MOV     DPL,A
    CLR     A
    ADDC    A,#HIGH
(MUSIC_SOUNDTONE2+01H)
    MOV     DPH,A
    CLR     A
    MOVC    A,@A+DPTR
    ADD     A,R7
    MOV     R7,A
    CLR     A
    ADDC    A,R6
    MOV     R6,A
    CLR     A
    MOV     R5,A
?C0052:
    MOV     A,R5
    CLR     C
    SUBB    A,R1
    JNC     ?C0053
    CLR     A
    MOV     R4,A
?C0055:
    CLR     A
    MOV     R3,A
```

```
    MOV   R2,A
?C0058:
    CLR   C
    MOV   A,R3
    SUBB  A,R7
    MOV   A,R2
    SUBB  A,R6
    JNC   ?C0059
    CLR   P1_0
    INC   R3
    CJNE  R3,#00H,?C0095
    INC   R2
?C0095:
    SJMP  ?C0058
?C0059:
    CLR   A
    MOV   R2,A
    MOV   R3,A
?C0061:
    CLR   C
    MOV   A,R3
    SUBB  A,R7
    MOV   A,R2
    SUBB  A,R6
    JNC   ?C0057
    SETB  P1_0
    INC   R3
    CJNE  R3,#00H,?C0096
    INC   R2
?C0096:
    SJMP  ?C0061
?C0057:
    INC   R4
    CJNE  R4,#0CH,?C0055
?C0054:
    INC   R5
    SJMP  ?C0052
?C0053:
    MOV   R7,#061H
    LCALL _Delay50uS
    INC   i?358
    MOV   A,i?358
    CLR   C
    SUBB  A,#01FH
    JC    ?C0049
    RET

_Music4:
    CLR   A
    MOV   i?469,A
    MOV   i?469+01H,A
    MOV   j?470,A
    MOV   j?470+01H,A
    MOV   k?465,A
?C0065:
    MOV   A,R7
    DEC   A
    MOV   R6,A
    MOV   A,k?465
    CLR   C
    SUBB  A,R6
    JNC   ?C0066
?C0068:
    MOV   A,i?469+01H
    MOV   DPTR,#MUSIC_SOUNDLONG3
    MOVC  A,@A+DPTR
    JZ    ?C0069
    INC   i?469+01H
    MOV   A,i?469+01H
    JNZ   ?C0097
    INC   i?469
?C0097:
    SJMP  ?C0068
?C0069:
    INC   i?469+01H
    MOV   A,i?469+01H
    JNZ   ?C0098
    INC   i?469
?C0098:
    INC   k?465
    SJMP  ?C0065
?C0066:
    CLR   A
    MOV   k?465,A
?C0070:
    MOV   A,R7
    DEC   A
    MOV   R6,A
    MOV   A,k?465
    CLR   C
    SUBB  A,R6
    JNC   ?C0077
?C0073:
    MOV   A,j?470+01H
    MOV   DPTR,#MUSIC_SOUNDTONE3
    MOVC  A,@A+DPTR
    JZ    ?C0074
    INC   j?470+01H
    MOV   A,j?470+01H
    JNZ   ?C0099
    INC   j?470
?C0099:
    SJMP  ?C0073
?C0074:
    INC   j?470+01H
    MOV   A,j?470+01H
    JNZ   ?C0100
```

```
    INC     j?470
?C0100:
    INC     k?465
    SJMP    ?C0070
?C0077:
    MOV     A,i?469+01H
    MOV     DPTR,#MUSIC_SOUNDLONG3
    MOVC    A,@A+DPTR
    MOV     R1,A
    MOV     A,j?470+01H
    MOV     DPTR,#MUSIC_SOUNDTONE3
    MOVC    A,@A+DPTR
    MOV     R7,A
    INC     i?469+01H
    MOV     A,i?469+01H
    JNZ     ?C0101
    INC     j?469
?C0101:
    INC     j?470+01H
    MOV     A,j?470+01H
    JNZ     ?C0102
    INC     j?470
?C0102:
    CLR     A
    MOV     R6,A
?C0078:
    MOV     A,R6
    CLR     C
    SUBB    A,R1
    JNC     ?C0079
    CLR     A
    MOV     k?465,A
?C0081:
    CLR     A
    MOV     R5,A
    MOV     R4,A
?C0084:
    MOV     A,R7
    CLR     C
    RRC     A
    MOV     R3,A
    CLR     C
    MOV     A,R5
    SUBB    A,R3
    MOV     A,R4
    SUBB    A,#00H
    JNC     ?C0085
    CLR     P1_0
    INC     R5
    CJNE    R5,#00H,?C0103
    INC     R4
?C0103:
    SJMP    ?C0084
?C0085:
    CLR     A
    MOV     R4,A
    MOV     R5,A
?C0087:
    MOV     A,R7
    CLR     C
    RRC     A
    MOV     R3,A
    CLR     C
    MOV     A,R5
    SUBB    A,R3
    MOV     A,R4
    SUBB    A,#00H
    JNC     ?C0083
    SETB    P1_0
    INC     R5
    CJNE    R5,#00H,?C0104
    INC     R4
?C0104:
    SJMP    ?C0087
?C0083:
    INC     k?465
    MOV     A,k?465
    CJNE    A,#0CH,?C0081
?C0080:
    INC     R6
    SJMP    ?C0078
?C0079:
    MOV     R7,#06H
    LCALL   _Delay50uS
    MOV     A,i?469+01H
    MOV     DPTR,#MUSIC_SOUNDLONG3
    MOVC    A,@A+DPTR
    JNZ     ?C0077
    MOV     A,j?470+01H
    MOV     DPTR,#MUSIC_SOUNDTONE3
    MOVC    A,@A+DPTR
    JNZ     ?C0077
    RET


SOUND_LONG:
    DB      004H,004H,004H,004H,004H
    DB      004H,004H,004H,004H,004H
    DB      004H,004H,004H,004H,004H


SOUND_TONE:
    DB      064H,05BH,053H,04DH,047H
    DB      043H,03EH,03BH,037H,034H
    DB      037H,03EH,047H,053H,064H


MUSIC_SOUNDLONG:
```

```
    DB      006H,006H,009H,003H,006H
    DB      006H,00CH,006H,006H,006H
    DB      006H,006H,006H,00CH,006H
    DB      006H,009H,003H,006H,006H
    DB      009H,003H,006H,003H,003H
    DB      006H,003H,003H,006H,006H
    DB      009H

MUSIC_SOUNDTONE:
    DB      078H,078H,06AH,05FH,050H
    DB      050H,05FH,050H,050H,047H
    DB      03FH,03CH,03CH,050H,03CH
    DB      03CH,047H,050H,05FH,047H
    DB      050H,05FH,078H,06AH,05FH
    DB      050H,047H,050H,05FH,06AH
    DB      078H
MUSIC_SOUNDLONG1:
    DB      006H,006H,009H,003H,006H
    DB      006H,00CH,006H,006H,006H
    DB      006H,006H,006H,00CH,006H
    DB      006H,009H,003H,006H,006H
    DB      009H,003H,006H,003H,003H
    DB      006H,003H,003H,006H,006H
    DB      009H,000H

MUSIC_SOUNDTONE1:
    DB      078H,078H,06AH,05FH,050H
    DB      050H,05FH,050H,050H,047H
    DB      03FH,03CH,03CH,050H,03CH
    DB      03CH,047H,050H,05FH,047H
    DB      050H,05FH,078H,06AH,05FH
    DB      050H,047H,050H,05FH,06AH
    DB      078H,000H

MUSIC_SOUNDTONE2:
    DB      000H,078H,000H,078H,000H,06AH
    DB      000H,05FH,000H,050H,000H,050H
    DB      000H,05FH,000H,050H,000H,050H
    DB      000H,047H,000H,03FH,000H,03CH
    DB      000H,03CH,000H,050H,000H,03CH
    DB      000H,03CH,000H,047H,000H,050H
    DB      000H,05FH,000H,047H,000H,050H
    DB      000H,05FH,000H,078H,000H,06AH
    DB      000H,05FH,000H,050H,000H,047H
    DB      000H,050H,000H,05FH,000H,06AH
    DB      000H,078H

MUSIC_SOUNDLONG3:
    DB      006H,006H,009H,003H,006H
    DB      006H,00CH,006H,006H,006H
    DB      006H,006H,006H,00CH,006H
    DB      006H,009H,003H,006H,006H
    DB      009H,003H,006H,003H,003H
    DB      006H,003H,003H,006H,006H
    DB      009H,000H
    DB      00CH,006H,006H,00CH,006H
    DB      006H,006H,00CH,006H,018H
    DB      00CH,006H,006H,00CH,006H
    DB      006H,006H,00CH,006H,018H
    DB      00CH,006H,006H,006H,006H
    DB      006H,006H,006H,00CH,006H
    DB      018H,00CH,00CH,00CH,006H
    DB      006H,006H,00CH,006H,018H
    DB      000H

MUSIC_SOUNDTONE3:
    DB      0EFH,0EFH,0D4H,0BDH,09FH
    DB      09FH,0BDH,09FH,09FH,08EH
    DB      07EH,078H,078H,09FH,078H
    DB      078H,08EH,09FH,0BDH,08EH
    DB      09FH,0BDH,0FFH,0D4H,0BDH
    DB      09FH,08EH,09FH,0BDH,0D4H
    DB      0EFH,000H
    DB      08EH,09FH,0BDH,08EH,09FH
    DB      0BDH,08EH,08EH,09FH,08EH
    DB      08EH,09FH,0BDH,08EH,09FH
    DB      0BDH,0D4H,0D4H,0EFH,0D4H
    DB      0BDH,0BDH,09FH,08EH,078H
    DB      08EH,09FH,0BDH,0BDH,09FH
    DB      0EFH,0BDH,0BDH,0BDH,0BDH
    DB      0BDH,08EH,08EH,09FH,08EH
    DB      000H
BcdDisplay1:
    CLR     A
    MOV     P1,A
?C0001:
    MOV     R0,#BcdVarible
    MOV     A,@R0
    ANL     A,#0FH
    MOV     P1,A
    SETB    P1_4
    CLR     P1_5
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#BcdVarible
    MOV     A,@R0
    ANL     A,#0F0H
    MOV     temp2040,A
    SWAP    A
    ANL     A,#0FH
    MOV     P1,A
    CLR     P1_4
```

```
    SETB    P1_5
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    SJMP    ?C0001
    RET

BcdDisplay2:
    CLR     A
    MOV     P1,A
?C0004:
    MOV     R0,#BcdVarible1+01H
    MOV     A,@R0
    MOV     temp1?141,A
    DEC     R0
    MOV     A,@R0
    MOV     temp2?142,A
    MOV     A,temp1?141
    ANL     A,#0FH
    MOV     P1,A
    SETB    P1_4
    CLR     P1_5
    CLR     P1_6
    CLR     P1_7
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     A,temp1?141
    ANL     A,#0F0H
    MOV     R7,A
    SWAP    A
    ANL     A,#0FH
    MOV     P1,A
    CLR     P1_4
    SETB    P1_5
    CLR     P1_6
    CLR     P1_7
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     A,temp2?142
    ANL     A,#0FH
    MOV     P1,A
    CLR     P1_4
    CLR     P1_5
    SETB    P1_6
    CLR     P1_7
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     A,temp2?142
    ANL     A,#0F0H
    MOV     R7,A
    SWAP    A
    ANL     A,#0FH
    MOV     P1,A
    CLR     P1_4
    CLR     P1_5
    CLR     P1_6
    SETB    P1_7
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    SJMP    ?C0004
    RET

BcdDisplay3:
?C0007:
    MOV     R0,#BcdData1
    MOV     A,@R0
    ANL     A,#0FH
    MOV     R7,A
    MOV     @R0,A
    SWAP    A
    ANL     A,#0F0H
    ORL     A,#01H
    MOV     temp?243,A
    MOV     R7,A
    LCALL   _Output4094_1
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#BcdData2
    MOV     A,@R0
    ANL     A,#0FH
    MOV     R7,A
    MOV     @R0,A
    SWAP    A
    ANL     A,#0F0H
    ORL     A,#02H
    MOV     temp?243,A
    MOV     R7,A
    LCALL   _Output4094_1
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#BcdData3
    MOV     A,@R0
    ANL     A,#0FH
    MOV     R7,A
    MOV     @R0,A
    SWAP    A
    ANL     A,#0F0H
    ORL     A,#04H
    MOV     temp?243,A
    MOV     R7,A
```

```
    LCALL   _Output4094_1
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#BcdData4
    MOV     A,@R0
    ANL     A,#0FH
    MOV     R7,A
    MOV     @R0,A
    SWAP    A
    ANL     A,#0F0H
    ORL     A,#08H
    MOV     temp?243,A
    MOV     R7,A
    LCALL   _Output4094_1
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    SJMP    ?C0007
    RET

BcdDisplay4:
    MOV     R0,#BcdVarible
    MOV     A,@R0
    MOV     P1,A
    RET

Dot5x7_Display1:
?C0001:
    MOV     P2,#039H
    SETB    P1_0
    CLR     P1_1
    CLR     P1_2
    CLR     P1_3
    CLR     P1_4
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     P2,#01EH
    CLR     P1_0
    SETB    P1_1
    CLR     P1_2
    CLR     P1_3
    CLR     P1_4
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     P2,#02EH
    CLR     P1_0
    CLR     P1_1
    SETB    P1_2
    CLR     P1_3
    CLR     P1_4
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     P2,#036H
    CLR     P1_0
    CLR     P1_1
    CLR     P1_2
    SETB    P1_3
    CLR     P1_4
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     P2,#039H
    CLR     P1_0
    CLR     P1_1
    CLR     P1_2
    CLR     P1_3
    SETB    P1_4
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    SJMP    ?C0001
    RET

Dot5x7_Display2:
?C0004:
    CLR     A
    MOV     i?140,A
?C0006:
    MOV     A,i?140
    MOV     DPTR,#DISPLAY_TABLE2
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?140
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0089
?C0088:
    CLR     C
    RLC     A
?C0089:
    DJNZ    R0,?C0088
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?140
    MOV     A,i?140
```

```
    CLR     C
    SUBB    A,#05H
    JC      ?C0006
    SJMP    ?C0004
    RET

Dot5x7_Display3:
?C0010:
    MOV     R0,#DisplayState
    MOV     A,@R0
    JNZ     ?C0012
    MOV     i?241,A
?C0013:
    MOV     A,i?241
    CLR     C
    SUBB    A,#05H
    JNC     ?C0010
    MOV     A,i?241
    MOV     DPTR,#DISPLAY_TABLE0
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?241
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0091
?C0090:
    CLR     C
    RLC     A
?C0091:
    DJNZ    R0,?C0090
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?241
    SJMP    ?C0013
?C0012:
    MOV     R0,#DisplayState
    MOV     A,@R0
    XRL     A,#01H
    JNZ     ?C0017
    MOV     i?241,A
?C0018:
    MOV     A,i?241
    CLR     C
    SUBB    A,#05H
    JNC     ?C0010
    MOV     A,i?241
    MOV     DPTR,#DISPLAY_TABLE1
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?241
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0093
?C0092:
    CLR     C
    RLC     A
?C0093:
    DJNZ    R0,?C0092
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?241
    SJMP    ?C0018
?C0017:
    MOV     R0,#DisplayState
    MOV     A,@R0
    XRL     A,#02H
    JNZ     ?C0022
    MOV     i?241,A
?C0023:
    MOV     A,i?241
    CLR     C
    SUBB    A,#05H
    JNC     ?C0010
    MOV     A,i?241
    MOV     DPTR,#DISPLAY_TABLE2
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?241
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0095
?C0094:
    CLR     C
    RLC     A
?C0095:
    DJNZ    R0,?C0094
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?241
    SJMP    ?C0023
?C0022:
    MOV     R0,#DisplayState
```

```
    MOV     A,@R0
    XRL     A,#03H
    JNZ     ?C0027
    MOV     i?241,A
?C0028:
    MOV     A,i?241
    CLR     C
    SUBB    A,#05H


    JC      $ + 5H
    LJMP    ?C0010
    MOV     A,i?241
    MOV     DPTR,#DISPLAY_TABLE3
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?241
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0097
?C0096:
    CLR     C
    RLC     A
?C0097:
    DJNZ    R0,?C0096
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?241
    SJMP    ?C0028
?C0027:
    MOV     R0,#DisplayState
    MOV     A,@R0
    XRL     A,#04H
    JNZ     ?C0032
    MOV     i?241,A
?C0033:
    MOV     A,i?241
    CLR     C
    SUBB    A,#05H
    JC      $ + 5H
    LJMP    ?C0010
    MOV     A,i?241
    MOV     DPTR,#DISPLAY_TABLE4
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?241
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0099
?C0098:
    CLR     C
    RLC     A
?C0099:
    DJNZ    R0,?C0098
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?241
    SJMP    ?C0033
?C0032:
    MOV     R0,#DisplayState
    MOV     A,@R0
    XRL     A,#05H
    JNZ     ?C0037
    MOV     i?241,A
?C0038:
    MOV     A,i?241
    CLR     C
    SUBB    A,#05H
    JC      $ + 5H
    LJMP    ?C0010
    MOV     A,i?241
    MOV     DPTR,#DISPLAY_TABLE5
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?241
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0101
?C0100:
    CLR     C
    RLC     A
?C0101:
    DJNZ    R0,?C0100
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?241
    SJMP    ?C0038
?C0037:
    MOV     R0,#DisplayState
    MOV     A,@R0
    XRL     A,#06H
    JNZ     ?C0042
    MOV     i?241,A
```

```
?C0043:
    MOV     A,i?241
    CLR     C
    SUBB    A,#05H
    JC      $ + 5H
    LJMP    ?C0010
    MOV     A,i?241
    MOV     DPTR,#DISPLAY_TABLE6
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?241
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0103
?C0102:
    CLR     C
    RLC     A
?C0103:
    DJNZ    R0,?C0102
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?241
    SJMP    ?C0043
?C0042:
    MOV     R0,#DisplayState
    MOV     A,@R0
    XRL     A,#07H
    JNZ     ?C0047
    MOV     i?241,A
?C0048:
    MOV     A,i?241
    CLR     C
    SUBB    A,#05H
    JC      $ + 5H
    LJMP    ?C0010
    MOV     A,i?241
    MOV     DPTR,#DISPLAY_TABLE7
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?241
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0105
?C0104:
    CLR     C
    RLC     A
?C0105:
    DJNZ    R0,?C0104
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?241
    SJMP    ?C0048
?C0047:
    MOV     R0,#DisplayState
    MOV     A,@R0
    XRL     A,#08H
    JNZ     ?C0052
    MOV     i?241,A
?C0053:
    MOV     A,i?241
    CLR     C
    SUBB    A,#05H
    JC      $ + 5H
    LJMP    ?C0010
    MOV     A,i?241
    MOV     DPTR,#DISPLAY_TABLE8
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?241
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0107
?C0106:
    CLR     C
    RLC     A
?C0107:
    DJNZ    R0,?C0106
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?241
    SJMP    ?C0053
?C0052:
    MOV     R0,#DisplayState
    MOV     A,@R0
    XRL     A,#09H
    JZ      $ + 5H
    LJMP    ?C0010
    MOV     i?241,A
?C0058:
    MOV     A,i?241
    CLR     C
    SUBB    A,#05H
    JC      $ + 5H
```

```
    LJMP    ?C0010
    MOV     A,i?241
    MOV     DPTR,#DISPLAY_TABLE9
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?241
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0109
?C0108:
    CLR     C
    RLC     A
?C0109:
    DJNZ    R0,?C0108
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?241
    SJMP    ?C0058
    RET

Dot5x7_Display4:
    MOV     R0,#DisplayState
    MOV     A,@R0
    CLR     C
    SUBB    A,#0AH
    JNC     ?C0068
?C0063:
    CLR     A
    MOV     i?342,A
?C0065:
    MOV     R0,#DisplayState
    MOV     A,@R0
    MOV     B,#05H
    MUL     AB
    MOV     j?343,A
    ADD     A,i?342
    MOV     DPTR,#DISPLAY_TABLE10
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?342
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0111
?C0110:
    CLR     C
    RLC     A
?C0111:
    DJNZ    R0,?C0110
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?342
    MOV     A,i?342
    CLR     C
    SUBB    A,#05H
    JC      ?C0065
    SJMP    ?C0063
?C0068:
    RET

Dot5x7_Display5:
    MOV     R0,#DisplayState
    MOV     A,@R0
    CLR     C
    SUBB    A,#0AH
    JNC     ?C0075
?C0070:
    CLR     A
    MOV     i?444,A
?C0072:
    MOV     R0,#DisplayState
    MOV     A,@R0
    MOV     B,#05H
    MUL     AB
    ADD     A,#LOW (DISPLAY_TABLE11)
    MOV     DPL,A
    CLR     A
    ADDC    A,#HIGH (DISPLAY_TABLE11)
    MOV     DPH,A
    MOV     A,DPL
    ADD     A,i?444
    MOV     DPL,A
    CLR     A
    ADDC    A,DPH
    MOV     DPH,A
    CLR     A
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?444
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0113
?C0112:
```

```
    CLR     C
    RLC     A
?C0113:
    DJNZ    R0,?C0112
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?444
    MOV     A,i?444
    CLR     C
    SUBB    A,#05H
    JC      ?C0072
    SJMP    ?C0070
?C0075:
    RET


Dot5x7_Display6:
?C0076:
    CLR     A
    MOV     j?546,A
?C0078:
    CLR     A
    MOV     count?547,A
    MOV     count?547+01H,A
?C0081:
    CLR     A
    MOV     i?545,A
?C0084:
    MOV     A,j?546
    MOV     B,#05H
    MUL     AB
    ADD     A,#LOW (DISPLAY_TABLE11)
    MOV     DPL,A
    CLR     A
    ADDC    A,#HIGH (DISPLAY_TABLE11)
    MOV     DPH,A
    MOV     A,DPL
    ADD     A,i?545
    MOV     DPL,A
    CLR     A
    ADDC    A,DPH
    MOV     DPH,A
    CLR     A
    MOVC    A,@A+DPTR
    MOV     P2,A
    MOV     R7,i?545
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0115
?C0114:
    CLR     C
    RLC     A
?C0115:
    DJNZ    R0,?C0114
    MOV     P1,A
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?545
    MOV     A,i?545
    CLR     C
    SUBB    A,#05H
    JC      ?C0084
?C0083:
    INC     count?547+01H
    MOV     A,count?547+01H
    JNZ     ?C0116
    INC     count?547
?C0116:
    CLR     C
    MOV     A,count?547+01H
    SUBB    A,#042H
    MOV     A,count?547
    SUBB    A,#00H
    JC      ?C0081
?C0082:
    MOV     P2,#0FFH
    CLR     A
    MOV     P1,A
    MOV     R7,#0C8H
    MOV     R6,A
    LCALL   _DelayX10ms
    INC     j?546
    MOV     A,j?546
    CLR     C
    SUBB    A,#0AH
    JC      ?C0078
    SJMP    ?C0076
    RET


DISPLAY_TABLE0:
    DB      041H,03EH,03EH,03EH,041H


DISPLAY_TABLE1:
    DB      03FH,03DH,000H,03FH,03FH


DISPLAY_TABLE2:
    DB      039H,01EH,02EH,036H,039H
```

```
DISPLAY_TABLE3:
    DB      05DH,036H,036H.036H.049H

DISPLAY_TABLE4:
    DB      067H,06BH,06DH.000H,06FH
DISPLAY_TABLE5:
    DB      030H,036H.036H.036H,041H

DISPLAY_TABLE6:
    DB      000H,036H.036H,036H.006H

DISPLAY_TABLE7:
    DB      07CH,07EH,00EH,076H,078H

DISPLAY_TABLE8:
    DB      041H,036H,036H,036H,041H

DISPLAY_TABLE9:
    DB      030H,036H,036H.036H,000H

DISPLAY_TABLE10:
    DB      041H,03EH,03EH,03EH,041H
    DB      03FH,03DH,000H,03FH,03FH
    DB      039H,01EH,02EH,036H,039H
    DB      05DH,036H,036H,036H,049H
    DB      067H,06BH,06DH.000H,06FH
    DB      030H.036H,036H,036H,041H
    DB      000H.036H,036H.036H,006H
    DB      07CH,07EH,00EH,076H,078H
    DB      041H,036H.036H,036H,041H
    DB      030H.036H,036H.036H.000H

DISPLAY_TABLE11:
    DB      041H,03EH,03EH.03EH,041H
    DB      03FH,03DH,000H,03FH,03FH
    DB      039H,01EH.02EH,036H,039H
    DB      05DH.036H,036H,036H,049H
    DB      067H,06BH.06DH.000H,06FH
    DB      030H.036H.036H.036H,041H
    DB      000H,036H.036H.036H.006H
    DB      07CH,07EH,00EH,076H,078H
    DB      041H,036H.036H,036H.041H
    DB      030H.036H,036H,036H.000H

Output74138_1:
    SETB    P1_3
    SETB    P1_0
    CLR     P1_1
    CLR     P1_2
    RET

Output74138_2:
    SETB    P1_3
    JNB     FgExtOut1,?C0002
    CLR     P1_0
    CLR     P1_1
    CLR     P1_2
    RET
?C0002:
    JNB     FgExtOut2,?C0004
    SETB    P1_0
    CLR     P1_1
    CLR     P1_2
    RET
?C0004:
    JNB     FgExtOut3,?C0005
    CLR     P1_0
    SETB    P1_1
    CLR     P1_2
    RET
?C0005:
    JNB     FgExtOut4,?C0006
    SETB    P1_0
    SETB    P1_1
    CLR     P1_2
    RET
?C0006:
    JNB     FgExtOut5,?C0007
    CLR     P1_0
    CLR     P1_1
    SETB    P1_2
    RET
?C0007:
    JNB     FgExtOut6,?C0008
    SETB    P1_0
    CLR     P1_1
    SETB    P1_2
    RET
?C0008:
    JNB     FgExtOut7,?C0009
    CLR     P1_0
    SETB    P1_1
    SETB    P1_2
    RET
?C0009:
    JNB     FgExtOut8,?C0010
    SETB    P1_0
    SETB    P1_1
    SETB    P1_2
```

```
    RET
?C0010:
    CLR     P1_3
    RET

Output74138_3:
    SETB    P1_3
    JNB     FgExtOut1,?C0011
    CLR     P1_0
    CLR     P1_1
    CLR     P1_2
    RET
?C0011:
    JNB     FgExtOut2,?C0013
    SETB    P1_0
    CLR     P1_1
    CLR     P1_2
    RET
?C0013:
    JNB     FgExtOut3,?C0015
    CLR     P1_0
    SETB    P1_1
    CLR     P1_2
    RET
?C0015:
    JNB     FgExtOut4,?C0017
    SETB    P1_0
    SETB    P1_1
    CLR     P1_2
    RET
?C0017:
    JNB     FgExtOut5,?C0019
    CLR     P1_0
    CLR     P1_1
    SETB    P1_2
    RET
?C0019:
    JNB     FgExtOut6,?C0021
    SETB    P1_0
    CLR     P1_1
    SETB    P1_2
    RET
?C0021:
    JNB     FgExtOut7,?C0023
    CLR     P1_0
    SETB    P1_1
    SETB    P1_2
    RET
?C0023:
    JNB     FgExtOut8,?C0025
    SETB    P1_0
    SETB    P1_1
    SETB    P1_2
    RET
?C0025:
    CLR     P1_3
    RET

Output74138_4:
    SETB    P1_3
    MOV     R0,#OutputState
    MOV     A,@R0
    LCALL   ?C?CCASE
    DW      ?C0029
    DB      01H
    DW      ?C0030
    DB      02H
    DW      ?C0031
    DB      03H
    DW      ?C0032
    DB      04H
    DW      ?C0033
    DB      05H
    DW      ?C0034
    DB      06H
    DW      ?C0035
    DB      07H
    DW      ?C0036
    DB      08H
    DW      00H
    DW      ?C0037
?C0029:
    CLR     P1_0
    CLR     P1_1
    CLR     P1_2
    RET
?C0030:
    SETB    P1_0
    CLR     P1_1
    CLR     P1_2
    RET
?C0031:
    CLR     P1_0
    SETB    P1_1
    CLR     P1_2
    RET
?C0032:
    SETB    P1_0
    SETB    P1_1
    CLR     P1_2
    RET
?C0033:
    CLR     P1_0
    CLR     P1_1
```

```
    SETB    P1_2
    RET
?C0034:
    SETB    P1_0
    CLR     P1_1
    SETB    P1_2
    RET
?C0035:
    CLR     P1_0
    SETB    P1_1
    SETB    P1_2
    RET
?C0036:
    SETB    P1_0
    SETB    P1_1
    SETB    P1_2
    RET
?C0037:
    CLR     P1_3
    RET


_Output4094_1:
    SETB    OE_PIN
    LCALL   _DataTx8bit
    LCALL   Strobe1
    CLR     OE_PIN
    RET


Output4094_2:
    SETB    OE_PIN
    MOV     R0,#OutPutData1
    MOV     A,@R0
    ORL     A,#02H
    MOV     R7,A
    MOV     @R0,A
    LCALL   _DataTx8bit
    LCALL   Strobe1
    CLR     OE_PIN
    RET


Output4094_3:
    SETB    OE_PIN
    MOV     R0,#OutPutData1
    MOV     A,@R0
    ANL     A,#0FDH
    MOV     R7,A
    MOV     @R0,A
    LCALL   _DataTx8bit
    LCALL   Strobe1
    CLR     OE_PIN
    RET
```

```
Output4094_4:
    SETB    OE_PIN
    MOV     R0,#OutPutData1
    MOV     A,@R0
    ORL     A,#02H
    MOV     @R0,A
    ANL     A,#0FBH
    MOV     R7,A
    MOV     @R0,A
    LCALL   _DataTx8bit
    LCALL   Strobe1
    CLR     OE_PIN
    RET


_DataTx8bit:
    MOV     value?441,R7
    CLR     A
    MOV     i?442,A
?C0005:
    MOV     A,value?441
    JNB     ACC.7,?C0008
    SETB    DATA_PIN
    SJMP    ?C0009
?C0008:
    CLR     DATA_PIN
?C0009:
    MOV     A,value?441
    ADD     A,ACC
    MOV     value?441,A
    LCALL   ClkDelay
    SETB    CLK_PIN
    LCALL   ClkDelay
    CLR     CLK_PIN
    LCALL   ClkDelay
    INC     i?442
    MOV     A,i?442
    CLR     C
    SUBB    A,#08H
    JC      ?C0005
    RET


Strobe1:
    CLR     STROBE1_PIN
    LCALL   ClkDelay
    SETB    STROBE1_PIN
    LCALL   ClkDelay
    CLR     STROBE1_PIN
    LCALL   ClkDelay
    RET
```

```
ClkDelay:
    CLR     A
    MOV     R7,A
?C0012:
    INC     R7
    CJNE    R7,#06H,?C0012
    RET


_Output4094_5:
    MOV     outputstate?744,R7
    MOV     value?745,R5
    SETB    OE_PIN
    MOV     A,outputstate?744
    CJNE    A,#01H,?C0016
    MOV     R7,value?745
    LCALL   _DataTx8bit
    LCALL   Strobe1
    SJMP    ?C0017
?C0016:
    MOV     A,outputstate?744
    CJNE    A,#02H,?C0017
    MOV     R7,value?745
    LCALL   _DataTx8bit
    LCALL   Strobe2
?C0017:
    CLR     OE_PIN
    RET


Strobe2:
    CLR     STROBE2_PIN
    LCALL   ClkDelay
    SETB    STROBE2_PIN
    LCALL   ClkDelay
    CLR     STROBE2_PIN
    LCALL   ClkDelay
    RET


OutPulse1:
    SETB    P1_0
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    CLR     P1_0
    MOV     R7,#05H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    RET


_OutPulse2:
    MOV     count?140,R7
    CLR     A
    MOV     i?141,A
?C0002:
    MOV     A,i?141
    CLR     C
    SUBB    A,count?140
    JNC     ?C0005
    SETB    P1_0
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    CLR     P1_0
    MOV     R7,#05H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    INC     i?141
    SJMP    ?C0002
?C0005:
    RET


OutPulse3:
    MOV     R0,#PulseCount
    MOV     @R0,#00H
    INC     R0
    MOV     @R0,#07DH
    SETB    TR1
?C0006:
    SETB    P1_0
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    CLR     P1_0
    MOV     R7,#05H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#PulseCount+01H
    MOV     A,@R0
    DEC     R0
    ORL     A,@R0
    JNZ     ?C0006
?C0007:
    CLR     TR1
    RET


PulseOff_Timer1ISR:
    PUSH    ACC
    PUSH    PSW
    MOV     PSW,#01001
    MOV     TL1,#0C0H
    MOV     TH1,#063H
    CLR     TF1
```

```
    MOV     R0,#PulseCount+01H
    MOV     A,@R0
    DEC     R0
    ORL     A,@R0
    JZ      ?C0011
    INC     R0
    MOV     A,@R0
    DEC     @R0
    JNZ     ?C0011
    DEC     R0
    DEC     @R0
?C0011:
    POP     PSW
    POP     ACC
    RETI

OutPulse4:
?C0012:
    SETB    P1_7
    SETB    P1_0
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    CLR     P1_0
    MOV     R7,#05H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    SJMP    ?C0012
    RET

OutPulse5:
?C0015:
    SETB    P1_7
    CLR     P1_0
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    SETB    P1_0
    MOV     R7,#05H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    SJMP    ?C0015
    RET

PulseDetect1:
?C0018:
    LCALL   Subroute1
    LCALL   Det_Pulse1
    LCALL   Subroute2
    SJMP    ?C0018
    RET

Det_Pulse1:
    JB      P1_7,?C0022
    LCALL   LedOn
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    LCALL   LedOff
    MOV     R7,#05H
    MOV     R6,#00H
    LCALL   _DelayX1ms
?C0022:
    RET

Subroute1:
    RET

Subroute2:
    RET

PulseDetect2:
?C0025:
    LCALL   Subroute1
    LCALL   Det_Pulse2
    LCALL   Subroute2
    SJMP    ?C0025
    RET

Det_Pulse2:
    JB      P1_7,?C0031
    LCALL   LedOn
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    LCALL   LedOff
    MOV     R7,#05H
    MOV     R6,#00H
    LCALL   _DelayX1ms
?C0029:
    JNB     P1_7,?C0029
?C0031:
    RET

PulseDetect3:
?C0032:
    LCALL   Subroute1
    LCALL   Det_Pulse3
    LCALL   Subroute2
```

```
    SJMP    ?C0032
    RET

Det_Pulse3:
    JB      FgPulse,?C0035
    JB      P1_7,?C0039
    SETB    FgPulse
    LCALL   LedOn
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    LCALL   LedOff
    MOV     R7,#05H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    RET
?C0035:
    JNB     P1_7,?C0039
    CLR     FgPulse
?C0039:
    RET

PulseGenerator:
    CLR     A
    MOV     R0,#SquareCount
    MOV     @R0,A
    CLR     P1_0
?C0040:
    LCALL   DelayPulse
    INC     i?1442
    SJMP    ?C0040
    RET

DelayPulse:
    MOV     R7,#01H
    LCALL   _Delay50uS
    MOV     R0,#SquareCount
    INC     @R0
    MOV     A,@R0
    CJNE    A,#064H,?C0046
    JB      P1_0,?C0044
    SETB    P1_0
    CLR     A
    MOV     @R0,A
    RET
?C0044:
    CLR     P1_0
    CLR     A
    MOV     R0,#SquareCount
    MOV     @R0,A
?C0046:
    RET

Pulse_Timer1ISR:
    JB      P1_0,?C0047
    SETB    P1_0
    SJMP    ?C0048
?C0047:
    CLR     P1_0
?C0048:
    MOV     TL1,#078H
    MOV     TH1,#0ECH
    CLR     TF1
    RETI

PulseDuty1_Timer1ISR:
    PUSH    ACC
    PUSH    B
    PUSH    PSW
    MOV     PSW,#010H
    JB      P1_0,?C0050
    SETB    P1_0
    MOV     R0,#DutyCycleValue
    MOV     A,@R0
    MOV     B,#013H
    MUL     AB
    MOV     R7,A
    CLR     C
    MOV     A,#078H
    SUBB    A,R7
    MOV     R5,A
    MOV     A,#0ECH
    SUBB    A,B
    MOV     R4,A
    SJMP    ?C0051
?C0050:
    CLR     P1_0
    MOV     R0,#DutyCycleValue
    MOV     A,@R0
    MOV     B,#013H
    MUL     AB
    ADD     A,#078H
    MOV     R5,A
    MOV     A,B
    ADDC    A,#0ECH
    MOV     R4,A
?C0051:
    MOV     A,R5
    MOV     TL1,A
    MOV     A,R4
    MOV     TH1,A
    CLR     TF1
    POP     PSW
    POP     B
```

```
    POP     ACC
    RETI

PulseDuty2_Timer1ISR:
    PUSH    ACC
    PUSH    B
    PUSH    PSW
    MOV     PSW,#010H
    JB      P1_0,?C0053
    SETB    P1_0
    MOV     R0,#DutyCycleValue
    MOV     A,@R0
    MOV     B,#027H
    MUL     AB
    MOV     R7,A
    CLR     C
    CLR     A
    SUBB    A,R7
    MOV     R5,A
    CLR     A
    SUBB    A,B
    MOV     R4,A
    SJMP    ?C0054
?C0053:
    CLR     P1_0
    MOV     R0,#DutyCycleValue
    MOV     A,@R0
    MOV     B,#027H
    MUL     AB
    ADD     A,#0F0H
    MOV     R5,A
    MOV     A,B
    ADDC    A,#0D8H
    MOV     R4,A
?C0054:
    MOV     A,R5
    MOV     TL1,A
    MOV     A,R4
    MOV     TH1,A
    CLR     TF1
    POP     PSW
    POP     B
    POP     ACC
    RETI

CheckPulseComc:
    JB      FgPulse,?C0056
    JB      P1_0,?C0060
    SETB    FgPulse
    RET
?C0056:
    JNB     P1_0,?C0060
    CLR     FgPulse
?C0060:
    RET

CheckPulseWidth:
    CLR     FgPulseShort
    CLR     FgPulseLong
    CLR     A
    MOV     R0,#DutyCount
    MOV     @R0,A
?C0061:
    JB      P1_0,?C0062
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    JB      P1_0,?C0061
    MOV     R0,#DutyCount
    INC     @R0
    SJMP    ?C0061
?C0062:
    MOV     R0,#DutyCount
    MOV     A,@R0
    MOV     R7,A
    MOV     A,#03H
    CLR     C
    SUBB    A,R7
    JNC     ?C0064
    MOV     A,R7
    CLR     C
    SUBB    A,#0DH
    JNC     ?C0064
    SETB    FgPulseShort
    RET
?C0064:
    MOV     A,#0DH
    CLR     C
    SUBB    A,R7
    JNC     ?C0067
    MOV     A,R7
    CLR     C
    SUBB    A,#017H
    JNC     ?C0067
    SETB    FgPulseLong
?C0067:
    RET

CheckPulseData:
    CLR     A
    MOV     temp?2146,A
    JB      P1_0,?C0069
?C0070:
    JNB     P1_0,?C0070
```

```
?C0071:
    CLR     A
    MOV     i?2145,A
?C0072:
    MOV     A,temp?2146
    ADD     A,ACC
    MOV     temp?2146,A
    MOV     R7,#05H
    MOV     R6,#00H
    LCALL   _DelayXms
    JNB     P1.0,?C0074
    ORL     temp?2146,#01H
?C0074:
    INC     i?2145
    MOV     A,i?2145
    CLR     C
    SUBB    A,#08H
    JC      ?C0072
?C0073:
    MOV     R0,#PulseData
    MOV     @R0,temp?2146
?C0069:
    RET

CheckPulseHiLow:
    CLR     A
    MOV     R0,#LowPulseCount
    MOV     @R0,A
    MOV     R0,#HiPulseCount
    MOV     @R0,A
    MOV     timeout?2249,A
    MOV     timeout?2249+01H,A
?C0076:
    INC     timeout?2249+01H
    MOV     A,timeout?2249+01H
    JNZ     ?C0107
    INC     timeout?2249
?C0107:
    MOV     A,timeout?2249+01H
    XRL     A,#0F4H
    JNZ     ?C0108
    MOV     A,timeout?2249
    XRL     A,#01H
?C0108:
    JNZ     ?C0076
?C0077
    CLR     A
    MOV     j?2248,A
?C0079:
    MOV     A,j?2248
    CLR     C
    SUBB    A,#014H
    JNC     ?C0080
    JNB     P1.0,?TEST_LOW
?C0082:
    MOV     R7,#01H
    LCALL   _Delay50uS
    INC     i?2247
    SJMP    ?C0079
?C0080:
    MOV     R0,#HiPulseCount
    INC     @R0
    SETB    FgTimeout
    RET
?TEST_LOW:
    CLR     A
    MOV     timeout?2249,A
    MOV     timeout?2249+01H,A
?C0085:
    INC     timeout?2249+01H
    MOV     A,timeout?2249+01H
    JNZ     ?C0109
    INC     timeout?2249
?C0109:
    MOV     A,timeout?2249+01H
    XRL     A,#0F4H
    JNZ     ?C0110
    MOV     A,timeout?2249
    XRL     A,#01H
?C0110:
    JNZ     ?C0085
?C0086:
    CLR     A
    MOV     j?2248,A
?C0088:
    MOV     A,j?2248
    CLR     C
    SUBB    A,#014H
    JNC     ?C0089
    JB      P1.0,?TEST_END
?C0091:
    MOV     R7,#01H
    LCALL   _Delay50uS
    INC     i?2247
    SJMP    ?C0088
?C0089:
    MOV     R0,#LowPulseCount
    INC     @R0
    SETB    FgTimeout
    RET
?TEST_END:
    CLR     FgTimeout
    RET

PulseDecoder:
    CLR     A
```

```
    MOV     temp?2351,A
?C0093:
    JB      P1_0,?C0093
?C0095:
    JNB     P1_0,?C0095
?C0096:
    CLR     A
    MOV     i?2350,A
?C0097:
    LCALL   CheckPulseHiLow
    JB      FgTimeout,?C0101
?C0100:
    MOV     A,temp?2351
    ADD     A,ACC
    MOV     temp?2351,A
    MOV     R0,#HiPulseCount
    MOV     A,@R0
    MOV     R7,A
    SETB    C
    SUBB    A,#055H
    JC      ?C0102
    MOV     A,R7
    CLR     C
    SUBB    A,#073H
    JNC     ?C0102
    MOV     R0,#LowPulseCount
    MOV     A,@R0
    MOV     R6,A
    SETB    C
    SUBB    A,#00H
    JC      ?C0102
    MOV     A,R6
    CLR     C
    SUBB    A,#041H
    JNC     ?C0102
    ORL     temp?2351,#01H
    SJMP    ?C0099
?C0102:
    MOV     A,R7
    SETB    C
    SUBB    A,#00H
    JC      ?C0099
    MOV     A,R7
    CLR     C
    SUBB    A,#041H
    JNC     ?C0099
    MOV     R0,#LowPulseCount
    MOV     A,@R0
    MOV     R7,A
    SETB    C
    SUBB    A,#055H
    JC      ?C0099
    MOV     A,R7
    CLR     C
    SUBB    A,#073H
    JNC     ?C0099
?C0099:
    INC     i?2350
    MOV     A,i?2350
    CLR     C
    SUBB    A,#08H
    JC      ?C0097
?C0098:
    MOV     R0,#PulseData
    MOV     @R0,temp?2351
?C0101:
    RET

EncoderProcess:
    MOV     A,P1
    ANL     A,#06H
    MOV     R0,#WheelNow
    MOV     @R0,A
    MOV     R7,A
    MOV     keytmp?040,A
    MOV     R0,#WheelOld
    MOV     A,@R0
    XRL     keytmp?040,A
    MOV     A,keytmp?040
    JZ      ?C0002
?C0001:
    MOV     A,keytmp?040
    JNB     ACC.2,?C0003
    MOV     A,R7
    JNB     ACC.2,?C0004
    LCALL   WheelLeft
    SJMP    ?C0008
?C0004:
    MOV     R0,#WheelNow
    MOV     A,@R0
    JNB     ACC.1,?C0006
    LCALL   WheelLeft
    SJMP    ?C0008
?C0006:
    LCALL   WheelRight
    SJMP    ?C0008
?C0003:
    MOV     A,keytmp?040
    ANL     A,#06H
    JZ      ?C0008
    MOV     R0,#WheelNow
    MOV     A,@R0
    JNZ     ?C0010
    LCALL   WheelLeft
    SJMP    ?C0008
?C0010:
    MOV     R0,#WheelNow
```

```
    MOV     A,@R0
    CJNE    A,#02H,?C0012
    LCALL   WheelLeft
    SJMP    ?C0008
?C0012:
    LCALL   WheelRight
?C0008:
    MOV     R0,#WheelNow
    MOV     A,@R0
    MOV     R0,#WheelOld
    MOV     @R0,A
?C0002:
    RET


WheelLeft:
    CLR     A
    MOV     R0,#RightCount
    MOV     @R0,A
    MOV     R0,#LeftCount
    INC     @R0
    MOV     A,@R0
    CJNE    A,#02H,?C0015
    MOV     R0,#EncoderCnt
    MOV     @R0,#02H
    CLR     A
    MOV     R0,#LeftCount
    MOV     @R0,A
?C0015:
    RET


WheelRight:
    CLR     A
    MOV     R0,#LeftCount
    MOV     @R0,A
    MOV     R0,#RightCount


    INC     @R0
    MOV     A,@R0
    CJNE    A,#02H,?C0017
    MOV     R0,#EncoderCnt
    MOV     @R0,#01H
    CLR     A
    MOV     R0,#RightCount
    MOV     @R0,A
?C0017:
    RET


Input4051_1:
    SETB    P1_0
    CLR     P1_1
    CLR     P1_2
    JNB     P2_7,?C0001
    SETB    FgExtInput2
    RET
?C0001:
    CLR     FgExtInput2
    RET


Input4051_2:
    CLR     A
    MOV     R7,A
?C0004:
    MOV     A,R7
    ANL     A,#07H
    MOV     P1,A
    JNB     P2_7,?C0006
    MOV     A,R7
    INC     A
    MOV     R0,#InputState
    MOV     @R0,A
    RET
?C0006:
    INC     R7
    CJNE    R7,#08H,?C0004
    RET


Input4051_3:
    CLR     A
    MOV     R7,A
?C0009:
    MOV     A,R7
    ANL     A,#07H
    MOV     P1,A
    JNB     P2_7,?C0011
    MOV     A,R7
    INC     A
    MOV     R0,#InputState
    MOV     @R0,A
?C0011:
    INC     R7
    CJNE    R7,#08H,?C0009
    RET


Input4051_4:
    CLR     FgExtInput1
    CLR     FgExtInput2
    CLR     FgExtInput3
    CLR     FgExtInput4
    CLR     FgExtInput5
    CLR     FgExtInput6
    CLR     FgExtInput7
```

```
    CLR     FgExtInput8
    CLR     P1_0
    CLR     P1_1
    CLR     P1_2
    JNB     P2_7,?C0014
    SETB    FgExtInput1
?C0014:
    SETB    P1_0
    CLR     P1_1
    CLR     P1_2
    JNB     P2_7,?C0015
    SETB    FgExtInput2
?C0015:
    CLR     P1_0
    SETB    P1_1
    CLR     P1_2
    JNB     P2_7,?C0016
    SETB    FgExtInput3
?C0016:
    SETB    P1_0
    SETB    P1_1
    CLR     P1_2
    JNB     P2_7,?C0017
    SETB    FgExtInput4
?C0017:
    CLR     P1_0
    CLR     P1_1
    SETB    P1_2
    JNB     P2_7,?C0018
    SETB    FgExtInput5
?C0018:
    SETB    P1_0
    CLR     P1_1
    SETB    P1_2
    JNB     P2_7,?C0019
    SETB    FgExtInput6
?C0019:
    CLR     P1_0
    SETB    P1_1
    SETB    P1_2
    JNB     P2_7,?C0020
    SETB    FgExtInput7
?C0020:
    SETB    P1_0
    SETB    P1_1
    SETB    P1_2
    JNB     P2_7,?C0022
    SETB    FgExtInput8
?C0022:
    RET

Input4051_5:
    CLR     FgExtInput1
    CLR     FgExtInput2
    CLR     FgExtInput3
    CLR     FgExtInput4
    CLR     FgExtInput5
    CLR     FgExtInput6
    CLR     FgExtInput7
    CLR     FgExtInput8
    CLR     A
    MOV     R7,A
?C0023:
    MOV     A,R7
    ANL     A,#07H
    MOV     P1,A
    JNB     P2_7,?C0025
    MOV     A,R7
    LCALL   ?C?CCASE
    DW      ?C0028
    DB      00H
    DW      ?C0029
    DB      01H
    DW      ?C0030
    DB      02H
    DW      ?C0031
    DB      03H
    DW      ?C0032
    DB      04H
    DW      ?C0033
    DB      05H
    DW      ?C0034
    DB      06H
    DW      ?C0035
    DB      07H
    DW      00H
    DW      ?C0025
?C0028:
    SETB    FgExtInput1
    SJMP    ?C0025
?C0029:
    SETB    FgExtInput2
    SJMP    ?C0025
?C0030:
    SETB    FgExtInput3
    SJMP    ?C0025
?C0031:
    SETB    FgExtInput4
    SJMP    ?C0025
?C0032:
    SETB    FgExtInput5
    SJMP    ?C0025
?C0033:
    SETB    FgExtInput6
    SJMP    ?C0025
?C0034:
    SETB    FgExtInput7
```

```
    SJMP    ?C0025
?C0035:
    SETB    FgExtInput8
?C0025:
    INC     R7
    CJNE    R7,#08H,?C0023
    RET

Input4051_6:
    CLR     FgExtInput1
    CLR     FgExtInput2
    CLR     FgExtInput3
    CLR     FgExtInput4
    CLR     FgExtInput5
    CLR     FgExtInput6
    CLR     FgExtInput7
    CLR     FgExtInput8
    CLR     A
    MOV     R7,A
?C0038:
    MOV     A,R7
    ANL     A,#07H
    MOV     P1,A
    JNB     P2_7,?C0040
    MOV     A,R7
    JNZ     ?C0042
    SETB    FgExtInput1
    SJMP    ?C0040
?C0042:
    CJNE    R7,#01H,?C0044
    SETB    FgExtInput2
    SJMP    ?C0040
?C0044:
    CJNE    R7,#02H,?C0046
    SETB    FgExtInput3
    SJMP    ?C0040
?C0046:
    CJNE    R7,#03H,?C0048
    SETB    FgExtInput4
    SJMP    ?C0040
?C0048:
    CJNE    R7,#04H,?C0050
    SETB    FgExtInput5
    SJMP    ?C0040
?C0050:
    CJNE    R7,#05H,?C0052
    SETB    FgExtInput6
    SJMP    ?C0040
?C0052:
    CJNE    R7,#06H,?C0054
    SETB    FgExtInput7
    SJMP    ?C0040
?C0054:
    CJNE    R7,#07H,?C0040
    SETB    FgExtInput8
?C0040:
    INC     R7
    CJNE    R7,#08H,?C0038
    RET

Input4067_1:
    CLR     FgExtInput1
    CLR     FgExtInput2
    CLR     FgExtInput3
    CLR     FgExtInput4
    CLR     FgExtInput5
    CLR     FgExtInput6
    CLR     FgExtInput7
    CLR     FgExtInput8
    CLR     FgExtInput9
    CLR     FgExtInput10
    CLR     FgExtInput11
    CLR     FgExtInput12
    CLR     FgExtInput13
    CLR     FgExtInput14
    CLR     FgExtInput15
    CLR     FgExtInput16
    CLR     P1_0
    CLR     P1_1
    CLR     P1_2
    CLR     P1_3
    JNB     P2_7,?C0058
    SETB    FgExtInput1
?C0058:
    SETB    P1_0
    CLR     P1_1
    CLR     P1_2
    CLR     P1_3
    JNB     P2_7,?C0059
    SETB    FgExtInput2
?C0059:
    CLR     P1_0
    SETB    P1_1
    CLR     P1_2
    CLR     P1_3
    JNB     P2_7,?C0060
    SETB    FgExtInput3
?C0060:
    SETB    P1_0
    SETB    P1_1
    CLR     P1_2
    CLR     P1_3
    JNB     P2_7,?C0061
    SETB    FgExtInput4
?C0061:
    CLR     P1_0
```

```
    CLR     P1_1
    SETB    P1_2
    CLR     P1_3
    JNB     P2_7,?C0062
    SETB    FgExtInput5
?C0062:
    SETB    P1_0
    CLR     P1_1
    SETB    P1_2
    CLR     P1_3
    JNB     P2_7,?C0063
    SETB    FgExtInput6
?C0063:
    CLR     P1_0
    SETB    P1_1
    SETB    P1_2
    CLR     P1_3
    JNB     P2_7,?C0064
    SETB    FgExtInput7
?C0064:
    SETB    P1_0
    SETB    P1_1
    SETB    P1_2
    CLR     P1_3
    JNB     P2_7,?C0065
    SETB    FgExtInput8
?C0065:
    CLR     P1_0
    CLR     P1_1
    CLR     P1_2
    SETB    P1_3
    JNB     P2_7,?C0066
    SETB    FgExtInput9
?C0066:
    SETB    P1_0
    CLR     P1_1
    CLR     P1_2
    SETB    P1_3
    JNB     P2_7,?C0067
    SETB    FgExtInput10
?C0067:
    CLR     P1_0
    SETB    P1_1
    CLR     P1_2
    SETB    P1_3
    JNB     P2_7,?C0068
    SETB    FgExtInput11
?C0068:
    SETB    P1_0
    SETB    P1_1
    CLR     P1_2
    SETB    P1_3
    JNB     P2_7,?C0069
    SETB    FgExtInput12
?C0069:
    CLR     P1_0
    CLR     P1_1
    SETB    P1_2
    SETB    P1_3
    JNB     P2_7,?C0070
    SETB    FgExtInput13
?C0070:
    SETB    P1_0
    CLR     P1_1
    SETB    P1_2
    SETB    P1_3
    JNB     P2_7,?C0071
    SETB    FgExtInput14
?C0071:
    CLR     P1_0
    SETB    P1_1
    SETB    P1_2
    SETB    P1_3
    JNB     P2_7,?C0072
    SETB    FgExtInput15
?C0072:
    SETB    P1_0
    SETB    P1_1
    SETB    P1_2
    SETB    P1_3
    JNB     P2_7,?C0073
    SETB    FgExtInput16
?C0073:
    CLR     P1_0
    CLR     P1_1
    CLR     P1_2
    CLR     P1_3
    RET


Input4067_2:
    CLR     FgExtInput1
    CLR     FgExtInput2
    CLR     FgExtInput3
    CLR     FgExtInput4
    CLR     FgExtInput5
    CLR     FgExtInput6
    CLR     FgExtInput7
    CLR     FgExtInput8
    CLR     FgExtInput9
    CLR     FgExtInput10
    CLR     FgExtInput11
    CLR     FgExtInput12
    CLR     FgExtInput13
    CLR     FgExtInput14
    CLR     FgExtInput15
    CLR     FgExtInput16
    CLR     A
```

```
    MOV     R7,A
?C0075:
    MOV     A,R7
    ANL     A,#0F1H
    MOV     P1,A
    JNB     P2_7,?C0077
    MOV     A,R7
    JNZ     ?C0079
    SETB    FgExtInput1
    SJMP    ?C0077
?C0079:
    CJNE    R7,#01H,?C0081
    SETB    FgExtInput2
    SJMP    ?C0077
?C0081:
    CJNE    R7,#02H,?C0083
    SETB    FgExtInput3
    SJMP    ?C0077
?C0083:
    CJNE    R7,#03H,?C0085
    SETB    FgExtInput4
    SJMP    ?C0077
?C0085:
    CJNE    R7,#04H,?C0087
    SETB    FgExtInput5
    SJMP    ?C0077
?C0087:
    CJNE    R7,#05H,?C0089
    SETB    FgExtInput6
    SJMP    ?C0077
?C0089:
    CJNE    R7,#06H,?C0091
    SETB    FgExtInput7
    SJMP    ?C0077
?C0091:
    CJNE    R7,#07H,?C0093
    SETB    FgExtInput8
    SJMP    ?C0077
?C0093:
    CJNE    R7,#08H,?C0095
    SETB    FgExtInput9
    SJMP    ?C0077
?C0095:
    CJNE    R7,#09H,?C0097
    SETB    FgExtInput10
    SJMP    ?C0077
?C0097:
    CJNE    R7,#0AH,?C0099
    SETB    FgExtInput11
    SJMP    ?C0077
?C0099:
    CJNE    R7,#0BH,?C0101
    SETB    FgExtInput12
    SJMP    ?C0077
?C0101:
    CJNE    R7,#0CH,?C0103
    SETB    FgExtInput13
    SJMP    ?C0077
?C0103:
    CJNE    R7,#0DH,?C0105
    SETB    FgExtInput14
    SJMP    ?C0077
?C0105:
    CJNE    R7,#0EH,?C0107
    SETB    FgExtInput15
    SJMP    ?C0077
?C0107:
    CJNE    R7,#0FH,?C0077
    SETB    FgExtInput16
?C0077:
    INC     R7
    MOV     A,R7
    XRL     A,#010H
    JNZ     ?C0075
    RET
Input4067_3:
    CLR     A
    MOV     R7,A
?C0111:
    MOV     A,R7
    ANL     A,#01FH
    MOV     P1,A
    MOV     A,#FrmBuf
    ADD     A,R7
    MOV     R0,A
    CLR     A
    MOV     @R0,A
    JNB     P2_7,?C0113
    MOV     A,#FrmBuf
    ADD     A,R7
    MOV     R0,A
    MOV     @R0,#01H
?C0113:
    INC     R7
    CJNE    R7,#020H,?C0111
    RET


InputKey1:
    MOV     A,P1
    CPL     A
    ANL     A,#0FH
    MOV     R7,A
    CJNE    R7,#01H,?C0001
    MOV     R0,#KeyData
    MOV     @R0,#01H
    RET
?C0001:
```

```
    CJNE    R7,#02H,?C0003
    MOV     R0,#KeyData
    MOV     @R0,#02H
    RET
?C0003:
    CJNE    R7,#04H,?C0005
    MOV     R0,#KeyData
    MOV     @R0,#03H
    RET
?C0005:
    CJNE    R7,#08H,?C0008
    MOV     R0,#KeyData
    MOV     @R0,#04H
?C0008:
    RET

InputKey2:
    CLR     A
    MOV     R0,#KeyData1
    MOV     @R0,A
    MOV     R0,#KeyData2
    MOV     @R0,A
    MOV     R0,#KeyData3
    MOV     @R0,A
    MOV     R0,#KeyData4
    MOV     @R0,A
    MOV     A,P1
    CPL     A
    ANL     A,#0FH
    MOV     R7,A
    ANL     A,#01H
    MOV     R6,A
    CJNE    R6,#01H,?C0009
    MOV     R0,#KeyData1
    MOV     @R0,#01H
?C0009:
    MOV     A,R7
    ANL     A,#02H
    MOV     R6,A
    CJNE    R6,#02H,?C0010
    MOV     R0,#KeyData2
    MOV     @R0,#02H
?C0010:
    MOV     A,R7
    ANL     A,#04H
    MOV     R6,A
    CJNE    R6,#04H,?C0011
    MOV     R0,#KeyData3
    MOV     @R0,#03H
?C0011:
    MOV     A,R7
    ANL     A,#08H
    MOV     R6,A
    CJNE    R6,#08H,?C0012
    MOV     R0,#KeyData4
    MOV     @R0,#04H
?C0012:
    MOV     R1,#KeyData1
    MOV     A,@R1
    MOV     R0,#KeyData2
    ADD     A,@R0
    MOV     R0,#KeyData3
    ADD     A,@R0
    MOV     R0,#KeyData4
    ADD     A,@R0
    MOV     R1,#KeyData
    MOV     @R1,A
    RET

InputKey3:
    CLR     A
    MOV     R0,#KeyData1
    MOV     @R0,A
    MOV     R0,#KeyData2
    MOV     @R0,A
    MOV     R0,#KeyData3
    MOV     @R0,A
    MOV     R0,#KeyData4
    MOV     @R0,A
    MOV     A,P1
    CPL     A
    ANL     A,#0FH
    MOV     R7,A
    JNB     ACC.0,?C0014
    MOV     R0,#KeyData1
    MOV     @R0,#01H
?C0014:
    MOV     A,R7
    JNB     ACC.1,?C0015
    MOV     R0,#KeyData2
    MOV     @R0,#02H
?C0015:
    MOV     A,R7
    JNB     ACC.2,?C0016
    MOV     R0,#KeyData3
    MOV     @R0,#03H
?C0016:
    MOV     A,R7
    JNB     ACC.3,?C0017
    MOV     R0,#KeyData4
    MOV     @R0,#04H
?C0017:
    MOV     R1,#KeyData1
    MOV     A,@R1
    MOV     R0,#KeyData2
    ADD     A,@R0
```

```
    MOV     R0,#KeyData3
    ADD     A,@R0
    4MOV    R0,#KeyData4
    ADD     A,@R0
    MOV     R1,#KeyData
    MOV     @R1,A
    RET


ScanKey1:
    CLR     COLUMN1_PIN
    SETB    COLUMN2_PIN
    SETB    COLUMN3_PIN
    MOV     A,P1
    CPL     A
    ANL     A,#0FH
    MOV     R7,A
    ADD     A,#0FEH
    JZ      ?C0021
    ADD     A,#0FEH
    JZ      ?C0022
    ADD     A,#0FCH
    JZ      ?C0023
    ADD     A,#07H
    JNZ     ?C0019
?C0020:
    MOV     R0,#KeyData
    MOV     @R0,#01H
    SJMP    ?C0019
?C0021:
    MOV     R0,#KeyData
    MOV     @R0,#02H
    SJMP    ?C0019
?C0022:
    MOV     R0,#KeyData
    MOV     @R0,#03H
    SJMP    ?C0019
?C0023:
    MOV     R0,#KeyData
    MOV     @R0,#04H
?C0019:
    SETB    COLUMN1_PIN
    CLR     COLUMN2_PIN
    SETB    COLUMN3_PIN
    MOV     A,P1
    CPL     A
    ANL     A,#0FH
    MOV     R7,A
    ADD     A,#0FEH
    JZ      ?C0027
    ADD     A,#0FEH
    JZ      ?C0028
    ADD     A,#0FCH
    JZ      ?C0029
    ADD     A,#07H
    JNZ     ?C0025
?C0026:
    MOV     R0,#KeyData
    MOV     @R0,#05H
    SJMP    ?C0025
?C0027:
    MOV     R0,#KeyData
    MOV     @R0,#06H
    SJMP    ?C0025
?C0028:
    MOV     R0,#KeyData
    MOV     @R0,#07H
    SJMP    ?C0025
?C0029:
    MOV     R0,#KeyData
    MOV     @R0,#08H
?C0025:
    SETB    COLUMN1_PIN
    SETB    COLUMN2_PIN
    CLR     COLUMN3_PIN
    MOV     A,P1
    CPL     A
    ANL     A,#0FH
    MOV     R7,A
    ADD     A,#0FEH
    JZ      ?C0033
    ADD     A,#0FEH
    JZ      ?C0034
    ADD     A,#0FCH
    JZ      ?C0035
    ADD     A,#07H
    JNZ     ?C0031
?C0032:
    MOV     R0,#KeyData
    MOV     @R0,#09H
    SJMP    ?C0031
?C0033:
    MOV     R0,#KeyData
    MOV     @R0,#0AH
    SJMP    ?C0031
?C0034:
    MOV     R0,#KeyData
    MOV     @R0,#0BH
    SJMP    ?C0031
?C0035:
    MOV     R0,#KeyData
    MOV     @R0,#0CH
?C0031:
    SETB    COLUMN3_PIN
    RET


ScanKey2:
```

```
    CLR     A
    MOV     R0,#KeyData1
    MOV     @R0,A
    CLR     COLUMN1_PIN
    SETB    COLUMN2_PIN
    SETB    COLUMN3_PIN
    MOV     A,P1
    CPL     A
    ANL     A,#0FH
    MOV     R7,A
    ADD     A,#0FEH
    JZ      ?C0040
    ADD     A,#0FEH
    JZ      ?C0041
    ADD     A,#0FCH
    JZ      ?C0042
    ADD     A,#07H
    JNZ     ?C0038
?C0039:
    MOV     R0,#KeyData1
    MOV     @R0,#01H
    SJMP    ?C0038
?C0040:
    MOV     R0,#KeyData1
    MOV     @R0,#02H
    SJMP    ?C0038
?C0041:
    MOV     R0,#KeyData1
    MOV     @R0,#03H
    SJMP    ?C0038
?C0042:
    MOV     R0,#KeyData1
    MOV     @R0,#04H
?C0038:
    CLR     A
    MOV     R0,#KeyData2
    MOV     @R0,A
    SETB    COLUMN1_PIN
    CLR     COLUMN2_PIN
    SETB    COLUMN3_PIN
    MOV     A,P1
    CPL     A
    ANL     A,#0FH
    MOV     R7,A
    ADD     A,#0FEH
    JZ      ?C0046
    ADD     A,#0FEH
    JZ      ?C0047
    ADD     A,#0FCH
    JZ      ?C0048
    ADD     A,#07H
    JNZ     ?C0044
?C0045:
    MOV     R0,#KeyData2
    MOV     @R0,#05H
    SJMP    ?C0044
?C0046:
    MOV     R0,#KeyData2
    MOV     @R0,#06H
    SJMP    ?C0044
?C0047:
    MOV     R0,#KeyData2
    MOV     @R0,#07H
    SJMP    ?C0044
?C0048:
    MOV     R0,#KeyData2
    MOV     @R0,#08H
?C0044:
    CLR     A
    MOV     R0,#KeyData3
    MOV     @R0,A
    SETB    COLUMN1_PIN
    SETB    COLUMN2_PIN
    CLR     COLUMN3_PIN
    MOV     A,P1
    CPL     A
    ANL     A,#0FH
    MOV     R7,A
    ADD     A,#0FEH
    JZ      ?C0052
    ADD     A,#0FEH
    JZ      ?C0053
    ADD     A,#0FCH
    JZ      ?C0054
    ADD     A,#07H
    JNZ     ?C0050
?C0051:
    MOV     R0,#KeyData3
    MOV     @R0,#09H
    SJMP    ?C0050
?C0052:
    MOV     R0,#KeyData3
    MOV     @R0,#0AH
    SJMP    ?C0050
?C0053:
    MOV     R0,#KeyData3
    MOV     @R0,#0BH
    SJMP    ?C0050
?C0054:
    MOV     R0,#KeyData3
    MOV     @R0,#0CH
?C0050:
    SETB    COLUMN3_PIN
    MOV     R1,#KeyData1
    MOV     A,@R1
    MOV     R0,#KeyData2
    ADD     A,@R0
    MOV     R0,#KeyData3
```

```
    ADD     A,@R0
    MOV     R1,#KeyData
    MOV     @R1,A
    RET

GetKey1:
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#081H
    MOVX    @R0,A
    MOV     R7,#02H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#LOW (XFR_ADC)
    MOVX    A,@R0
    ANL     A,#03FH
    MOV     keytmp?546,A
    CLR     C
    SUBB    A,#010H
    JNC     ?C0057
    MOV     R0,#KeyData
    MOV     @R0,#01H
    SJMP    ?C0058
?C0057
    MOV     A,keytmp?546
    CLR     C
    SUBB    A,#01BH
    JNC     ?C0059
    MOV     R0,#KeyData
    MOV     @R0,#02H
    SJMP    ?C0058
?C0059:
    MOV     A,keytmp?546
    CLR     C
    SUBB    A,#026H
    JNC     ?C0061
    MOV     R0,#KeyData
    MOV     @R0,#03H
    SJMP    ?C0058
?C0061:
    MOV     A,keytmp?546
    CLR     C
    SUBB    A,#030H
    JNC     ?C0058
    MOV     R0,#KeyData
    MOV     @R0,#04H
?C0058:
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#082H
    MOVX    @R0,A
    MOV     R7,#02H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#LOW (XFR_ADC)
    MOVX    A,@R0
    ANL     A,#03FH
    MOV     keytmp?546,A
    CLR     C
    SUBB    A,#010H
    JNC     ?C0064
    MOV     R0,#KeyData
    MOV     @R0,#05H
    SJMP    ?C0065
?C0064:
    MOV     A,keytmp?546
    CLR     C
    SUBB    A,#01BH
    JNC     ?C0066
    MOV     R0,#KeyData
    MOV     @R0,#06H
    SJMP    ?C0065
?C0066:
    MOV     A,keytmp?546
    CLR     C
    SUBB    A,#026H
    JNC     ?C0068
    MOV     R0,#KeyData
    MOV     @R0,#07H
    SJMP    ?C0065
?C0068:
    MOV     A,keytmp?546
    CLR     C
    SUBB    A,#030H
    JNC     ?C0065
    MOV     R0,#KeyData
    MOV     @R0,#08H
?C0065:
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#084H
    MOVX    @R0,A
    MOV     R7,#02H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R0,#LOW (XFR_ADC)
    MOVX    A,@R0
    ANL     A,#03FH
    MOV     keytmp?546,A
    CLR     C
    SUBB    A,#010H
    JNC     ?C0071
    MOV     R0,#KeyData
    MOV     @R0,#09H
    SJMP    ?C0072
?C0071:
    MOV     A,keytmp?546
```

```
    CLR   C
    SUBB  A,#01BH
    JNC   ?C0073
    MOV   R0,#KeyData
    MOV   @R0,#0AH
    SJMP  ?C0072
?C0073:
    MOV   A,keytmp?546
    CLR   C
    SUBB  A,#026H
    JNC   ?C0075
    MOV   R0,#KeyData
    MOV   @R0,#0BH
    SJMP  ?C0072
?C0075:
    MOV   A,keytmp?546
    CLR   C
    SUBB  A,#030H
    JNC   ?C0072
    MOV   R0,#KeyData
    MOV   @R0,#0CH
?C0072:
    CLR   A
    MOV   R0,#LOW (XFR_ADC)
    MOVX  @R0,A
    RET


GetKey2:
    MOV   R0,#LOW (XFR_ADC)
    MOV   A,#081H
    MOVX  @R0,A
    MOV   R7,#02H
    MOV   R6,#00H
    LCALL _DelayX1ms
    CLR   A
    MOV   R0,#KeyData1
    MOV   @R0,A
    MOV   R0,#LOW (XFR_ADC)
    MOVX  A,@R0
    ANL   A,#03FH
    MOV   keytmp?647,A
    CLR   C
    SUBB  A,#010H
    JNC   ?C0079
    MOV   R0,#KeyData1
    MOV   @R0,#01H
    SJMP  ?C0080
?C0079:
    MOV   A,keytmp?647
    CLR   C
    SUBB  A,#01BH
    JNC   ?C0081
    MOV   R0,#KeyData1
    MOV   @R0,#02H
    SJMP  ?C0080
?C0081:
    MOV   A,keytmp?647
    CLR   C
    SUBB  A,#026H
    JNC   ?C0083
    MOV   R0,#KeyData1
    MOV   @R0,#03H
    SJMP  ?C0080
?C0083:
    MOV   A,keytmp?647
    CLR   C
    SUBB  A,#030H
    JNC   ?C0080
    MOV   R0,#KeyData1
    MOV   @R0,#04H
?C0080:
    MOV   R0,#LOW (XFR_ADC)
    MOV   A,#082H
    MOVX  @R0,A
    MOV   R7,#02H
    MOV   R6,#00H
    LCALL _DelayX1ms
    CLR   A
    MOV   R0,#KeyData2
    MOV   @R0,A
    MOV   R0,#LOW (XFR_ADC)
    MOVX  A,@R0
    ANL   A,#03FH
    MOV   keytmp?647,A
    CLR   C
    SUBB  A,#010H
    JNC   ?C0086
    MOV   R0,#KeyData2
    MOV   @R0,#05H
    SJMP  ?C0087
?C0086:
    MOV   A,keytmp?647
    CLR   C
    SUBB  A,#01BH
    JNC   ?C0088
    MOV   R0,#KeyData2
    MOV   @R0,#06H
    SJMP  ?C0087
?C0088:
    MOV   A,keytmp?647
    CLR   C
    SUBB  A,#026H
    JNC   ?C0090
    MOV   R0,#KeyData2
    MOV   @R0,#07H
    SJMP  ?C0087
?C0090:
```

```
    MOV     A,keytmp?647
    CLR     C
    SUBB    A,#030H
    INC     ?C0087
    MOV     R0,#KeyData2
    MOV     @R0,#08H
?C0087:
    MOV     R0,#LOW (XFR_ADC)
    MOV     A,#084H
    MOVX    @R0,A
    MOV     R7,#02H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    CLR     A
    MOV     R0,#KeyData3
    MOV     @R0,A
    MOV     R0,#LOW (XFR_ADC)
    MOVX    A,@R0
    ANL     A,#03FH
    MOV     keytmp?647,A
    CLR     C
    SUBB    A,#010H
    JNC     ?C0093
    MOV     R0,#KeyData3
    MOV     @R0,#09H
    SJMP    ?C0094
?C0093:
    MOV     A,keytmp?647
    CLR     C
    SUBB    A,#01BH
    INC     ?C0095
    MOV     R0,#KeyData3
    MOV     @R0,#0AH
    SJMP    ?C0094
?C0095:
    MOV     A,keytmp?647
    CLR     C
    SUBB    A,#026H
    JNC     ?C0097
    MOV     R0,#KeyData3
    MOV     @R0,#0BH
    SJMP    ?C0094
?C0097:
    MOV     A,keytmp?647
    CLR     C
    SUBB    A,#030H
    JNC     ?C0094
    MOV     R0,#KeyData3
    MOV     @R0,#0CH
?C0094:
    MOV     R1,#KeyData1
    MOV     A,@R1
    MOV     R0,#KeyData2
    ADD     A,@R0
    MOV     R0,#KeyData3
    ADD     A,@R0
    MOV     R1,#KeyData
    MOV     @R1,A
    CLR     A
    MOV     R0,#LOW (XFR_ADC)
    MOVX    @R0,A
    RET

InputKeyCheck:
    LCALL   InputKey2
    CLR     FgDoubleKey
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
    MOV     R0,#KeyData1
    MOV     A,@R0
    CJNE    A,#01H,?C0102
    MOV     R0,#KeyData3
    MOV     A,@R0
    CJNE    A,#03H,?C0102
    SETB    FgDoubleKey
    MOV     R0,#KeyData
    MOV     @R0,#063H
?C0102:
    RET

ScanKeyCheck:
    LCALL   ScanKey2
    CLR     FgDoubleKey
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
    MOV     R0,#KeyData1
    MOV     A,@R0
    CJNE    A,#011H,?C0104
    MOV     R0,#KeyData3
    MOV     A,@R0
    CJNE    A,#0AH,?C0104
    SETB    FgDoubleKey
    MOV     R0,#KeyData
    MOV     @R0,#063H
?C0104:
    RET

GetKeyCheck:
    LCALL   GetKey2
    CLR     FgDoubleKey
    CLR     A
    MOV     R0,#KeyData
    MOV     @R0,A
```

```
    MOV     R0,#KeyData1
    MOV     A,@R0
    CJNE    A,#01H,?C0106
    MOV     R0,#KeyData3
    MOV     A,@R0
    CJNE    A,#0AH,?C0106
    SETB    FgDoubleKey
    MOV     R0,#KeyData
    MOV     @R0,#063H
?C0106:
    RET

KeyCountCheck1:
    LCALL   InputKey1
    MOV     R0,#KeyData
    MOV     A,@R0
    XRL     A,#01H
    JNZ     ?C0110
    MOV     R0,#KeyCount
    INC     @R0
    MOV     A,@R0
    CLR     C
    SUBB    A,#014H
    JNC     ?C0108
    MOV     R7,#01H
    MOV     R5,#032H
    MOV     R3,#032H
    LCALL   _LedFlash5
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep4
    RET
?C0108:
    MOV     R7,#02H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep4
    CLR     A
    MOV     R0,#KeyCount
    MOV     @R0,A
?C0110:
    RET

KeyCountCheck2:
    LCALL   InputKey1
    MOV     R0,#KeyData
    MOV     A,@R0
    XRL     A,#01H
    JNZ     ?C0114
    MOV     R0,#KeyCount
    INC     @R0
    MOV     A,@R0
    CLR     C
    SUBB    A,#014H
    JNC     ?C0112
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep4
    RET
?C0112:
    MOV     R7,#02H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep4
    CLR     A
    MOV     R0,#KeyCount
    MOV     @R0,A
?C0114:
    RET

KeyProcess:
    LCALL   GetKey2
    MOV     R0,#KeyData
    MOV     A,@R0
    MOV     R7,A
    MOV     R0,#KeyBuffer
    XRL     A,@R0
    JZ      ?C0115
    MOV     A,R7
    MOV     @R0,A
    MOV     R0,#ScanKeyCounter
    MOV     @R0,#014H
    RET
?C0115:
    MOV     R0,#ScanKeyCounter
    MOV     A,@R0
    JZ      ?C0117
    DEC     @R0
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    RET
?C0117:
    MOV     R0,#KeyBuffer
    MOV     A,@R0
    MOV     R7,A
    JNZ     $ + 5H
    LJMP    ?C0119
    ADD     A,#0FEH
```

```
    JZ      ?C0123
    DEC     A
    JZ      ?C0127
    DEC     A
    JZ      ?C0131
    DEC     A
    JNZ     $ + 5H
    LJMP    ?C0135
    ADD     A,#04H
    JZ      $ + 5H
    LJMP    ?C0129
?C0121:
    JNB     FgKEY1,$ + 6H
    LJMP    ?C0129
    SETB    FgKEY1
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep4
    RET
?C0123:
    JNB     FgKEY2,$ + 6H
    LJMP    ?C0129
    SETB    FgKEY2
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep4
    CPL     FgKEY2_ONOFF
    JNB     FgKEY2_ONOFF,?C0125
    LCALL   LedOn
    RET
?C0125:
    LCALL   LedOff
    RET
?C0127:
    JB      FgKEY3,?C0128
    JNB     FgKEY2_ONOFF,?C0128
    SETB    FgKEY3
    MOV     R7,#02H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep4
    RET
?C0128:
    JNB     FgKEY3,$ + 6H
    LJMP    ?C0129
    SETB    FgKEY3
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep4
    RET
?C0131:
    JB      FgKEY4,?C0132
    SETB    FgKEY4
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep4
    MOV     R0,#KeyCountTimer
    MOV     @R0,#0AH
    LCALL   SubProcess
    RET
?C0132:
    MOV     R0,#KeyCountTimer
    MOV     A,@R0
    JNZ     ?C0129
    MOV     @R0,#02H
    MOV     R0,#UpdateValue
    MOV     @R0,#01H
    LCALL   SubProcess
    RET
?C0135:
    JB      FgKEY5,?C0136
    SETB    FgKEY5
    MOV     R7,#01H
    MOV     R6,#00H
    MOV     R5,#011H
    MOV     R3,#0AH
    LCALL   _Beep4
    MOV     R0,#KeyCountTimer
    MOV     @R0,#0AH
    MOV     R0,#UpdateValue
    MOV     @R0,#01H
    LCALL   SubProcess
    RET
?C0136:
    MOV     R0,#KeyCountTimer
    MOV     A,@R0
    JNZ     ?C0129
    MOV     @R0,#02H
    MOV     R0,#CoarseAdjCount
    INC     @R0
    MOV     A,@R0
    CLR     C
    SUBB    A,#06H
    JNC     ?C0139
    MOV     R0,#UpdateValue
    MOV     @R0,#02H
    SJMP    ?C0140
?C0139:
    MOV     R0,#UpdateValue
```

```
    MOV     @R0,#06H
?C0140:
    LCALL   SubProcess
    RET
?C0119:
    CLR     A
    MOV     R0,#KeyBuffer
    MOV     @R0,A
    MOV     R0,#CoarseAdjCount
    MOV     @R0,A
    CLR     FgKEY1
    CLR     FgKEY2
    CLR     FgKEY3
    CLR     FgKEY4
    CLR     FgKEY2_ONOFF
?C0129:
    RET


SubProcess:
    RET


LM317_1:
    SETB    P1_0
    CLR     P1_1
    CLR     P1_2
    CLR     P1_3
    RET


LM317_2:
    CLR     P1_0
    CLR     P1_1
    SETB    P1_2
    CLR     P1_3
    RET


LM317_3:
    SETB    P1_0
    SETB    P1_1
    CLR     P1_2
    CLR     P1_3
    RET


DAC08_1:
    SETB    P1_0
    SETB    P1_7
    RET


DAC08_2:
    MOV     P1,#081H
    RET


SawTooth1:
    CLR     A
    MOV     R7,A
?C0006:
    MOV     P1,R7
    INC     R7
    SJMP    ?C0006
    RET


SawTooth2:
    MOV     R7,#0FFH
?C0009:
    MOV     P1,R7
    DEC     R7
    SJMP    ?C0009
    RET


TriAngle1:
    CLR     A
    MOV     R7,A
?PLUS:
    MOV     P1,R7
    INC     R7
    CJNE    R7,#0FFH,?PLUS
?MINUS:
    MOV     P1,R7
    DEC     R7
    MOV     A,R7
    JZ      ?PLUS
    SJMP    ?MINUS
    RET


TriAngle2:
?C0019:
    CLR     A
    MOV     R7,A
?C0021:
    MOV     P1,R7
    INC     R7
    CJNE    R7,#0FFH,?C0021
?C0022:
    MOV     R7,#0FFH
?C0024:
    MOV     A,R7
    SETB    C
    SUBB    A,#00H
    JC      ?C0019
```

```
    MOV     P1,R7
    DEC     R7
    SJMP    ?C0024
    RET

TriAngle3:
?C0028:
    CLR     A
    MOV     R7,A
?C0030:
    MOV     P1,R7
    INC     R7
    CJNE    R7,#0FFH,?C0030
?C0031:
    CLR     A
    MOV     R7,A
?C0033:
    MOV     A,R7
    CLR     C
    SUBB    A,#0FFH
    JNC     ?C0028
    MOV     A,R7
    CPL     A
    MOV     P1,A
    INC     R7
    SJMP    ?C0033
    RET

Square:
?C0037:
    CLR     A
    MOV     R7,A
?C0039:
    CLR     A
    MOV     P1,A
    INC     R7
    MOV     A,R7
    SETB    C
    SUBB    A,#0FFH
    JC      ?C0039
?C0040:
    CLR     A
    MOV     R7,A
?C0042:
    MOV     A,R7
    SETB    C
    SUBB    A,#0FFH
    JNC     ?C0037
    MOV     P1,#0FFH
    INC     R7
    SJMP    ?C0042
    RET

PushEeprom93c66:
    CLR     A
    MOV     value?041,A
    MOV     adr?040,A
?C0001:
    SETB    CS_PIN
    MOV     R7,adr?040
    MOV     R5,value?041
    LCALL   _EepWriteData
    CLR     CS_PIN
    CLR     XSCK_PIN
    INC     value?041
    INC     adr?040
    MOV     A,adr?040
    CLR     C
    SUBB    A,#08011
    JC      ?C0001
    RET

EepWriteData:
    MOV     adr?142,R7
    MOV     value?143,R5
    SETB    CS_PIN
    LCALL   EwenROM
    MOV     R7,adr?142
    LCALL   EraseROM
    MOV     R7,adr?142
    MOV     R5,value?143
    LCALL   WriteROM
    LCALL   EwdsROM
    CLR     CS_PIN
    CLR     XSCK_PIN
    RET

EwenROM:
    SETB    CS_PIN
    MOV     R7,#04H
    LCALL   WR_3wire
    MOV     R7,#0C0H
    LCALL   _WR_3wire
    CLR     CS_PIN
    LCALL   XbusDelay
    RET

_EraseROM:
    MOV     adr?344,R7
    SETB    CS_PIN
    MOV     R7,#07H
    LCALL   _WR_3wire
```

```
    MOV     R7,adr?344
    LCALL   _WR_3wire
    CLR     CS_PIN
    CLR     XSCK_PIN
    CLR     XSDO_PIN
    MOV     R7,#0AH
    MOV     R6,#00H
    LCALL   _DelayX1ms
    RET

_EraseROM1:
    MOV     adr?445,R7
    SETB    CS_PIN
    MOV     R7,#07H
    LCALL   _WR_3wire
    MOV     R7,adr?445
    LCALL   _WR_3wire
    CLR     CS_PIN
    CLR     XSCK_PIN
    CLR     XSDO_PIN
    MOV     R7,#0AH
    MOV     R6,#00H
    LCALL   _DelayX1ms
    SETB    FgTimeout
    CLR     A
    MOV     R7,A
    MOV     R6,A
?C0009:
    JNB     XSDI_PIN,?C0011
    CLR     FgTimeout
    RET
?C0011:
    INC     R7
    CJNE    R7,#00H,?C0037
    INC     R6
?C0037:
    CJNE    R6,#03H,?C0009
    CJNE    R7,#0E8H,?C0009
?C0013:
    RET

_WriteROM:
    MOV     adr?547,R7
    MOV     value?548,R5
    SETB    CS_PIN
    MOV     R7,#05H
    LCALL   _WR_3wire
    MOV     R7,adr?547
    LCALL   _WR_3wire
    MOV     R7,value?548
    LCALL   _WR_3wire
    CLR     CS_PIN
    CLR     XSCK_PIN
    CLR     XSDO_PIN
    MOV     R7,#0AH
    MOV     R6,#00H
    LCALL   _DelayX1ms
    RET

EwdsROM:
    SETB    CS_PIN
    MOV     R7,#04H
    LCALL   _WR_3wire
    CLR     A
    MOV     R7,A
    LCALL   _WR_3wire
    CLR     CS_PIN
    RET

_WR_3wire:
    MOV     value?749,R7
    CLR     A
    MOV     i?750,A
?C0017:
    MOV     A,value?749
    JNB     ACC.7,?C0020
    SETB    XSDO_PIN
    SJMP    ?C0021
?C0020:
    CLR     XSDO_PIN
?C0021:
    MOV     A,value?749
    ADD     A,ACC
    MOV     value?749,A
    LCALL   XbusDelay
    SETB    XSCK_PIN
    LCALL   XbusDelay
    CLR     XSCK_PIN
    LCALL   XbusDelay
    INC     i?750
    MOV     A,i?750
    CLR     C
    SUBB    A,#08H
    JC      ?C0017
    RET

XbusDelay:
    CLR     A
    MOV     R7,A
?C0023:
    INC     R7
    CJNE    R7,#0AH,?C0023
```

```
    RET


PopEeprom93c66:
    CLR     A
    MOV     adr?952,A
?C0027:
    SETB    CS_PIN
    MOV     R7,adr?952
    LCALL   _ReadROM
    MOV     R0,#Buffer
    MOV     A,R7
    MOV     @R0,A
    CLR     CS_PIN
    CLR     XSCK_PIN
    INC     adr?952
    MOV     A,adr?952
    CLR     C
    SUBB    A,#080H
    JC      ?C0027
    RET


_ReadROM:
    MOV     adr?1053,R7
    CLR     XSCK_PIN
    CLR     XSDO_PIN
    MOV     R7,#06H
    LCALL   _WR_3wire
    MOV     R7,adr?1053
    LCALL   _WR_3wire
    CLR     XSCK_PIN
    CLR     XSDO_PIN
    LCALL   ReceiveByte
    RET


ReceiveByte:
    CLR     A
    MOV     R6,A
    MOV     R5,A
?C0032:
    SETB    XSCK_PIN
    LCALL   XbusDelay
    CLR     XSCK_PIN
    MOV     A,R6
    ADD     A,ACC
    MOV     R6,A
    JNB     XSDI_PIN,?C0034
    XCH     A,R6
    ORL     A,#01H
    XCH     A,R6
?C0034:
    INC     R5
    CJNE    R5,#08H,?C0032
    LCALL   XbusDelay
    XCH     A,R7
    MOV     A,R6
    XCH     A,R7
    RET


I2cWait:
    NOP
    NOP
    RET


I2cInit:
    SETB    ISDA
    SETB    ISCL
    RET


I2cStart:
    SETB    ISDA
    SETB    ISCL
    LCALL   I2cWait
    CLR     ISDA
    LCALL   I2cWait
    CLR     ISCL
    RET


I2cStop:
    CLR     ISDA
    LCALL   I2cWait
    SETB    ISCL
    LCALL   I2cWait
    SETB    ISDA
    RET


_I2cSentByte:
    XCH     A,R6
    MOV     A,R7
    XCH     A,R6
    SETB    ISDA
    CLR     A
    MOV     R5,A
?C0005:
    MOV     R0,#011H
    XCH     A,R7
    MOV     A,R6
    XCH     A,R7
    MOV     A,R7
    INC     R0
    SJMP    ?C0056
```

```
?C0055:
    RL      A
?C0056:
    DJNZ    R0,?C0055
    MOV     R6,A
    JNB     ACC.0,?C0008
    SETB    ISDA
    SJMP    ?C0009
?C0008:
    CLR     ISDA
?C0009:
    SETB    ISCL
    LCALL   I2cWait
    CLR     ISCL
    LCALL   I2cWait
    INC     R5
    CJNE    R5,#08H,?C0005
?C0006:
    SETB    ISDA
    LCALL   I2cWait
    SETB    ISCL
    LCALL   I2cWait
    MOV     C,ISDA
    MOV     ack?442,C
    CLR     ISCL
    LCALL   I2cWait
    MOV     C,ack?442
    RET
I2cSentByte1:
    CLR     A
    MOV     R6,A
?C0011:
    MOV     A,R7
    JNB     ACC.7,?C0014
    SETB    ISDA
    SJMP    ?C0015
?C0014:
    CLR     ISDA
?C0015:
    MOV     A,R7
    ADD     A,ACC
    MOV     R7,A
    LCALL   I2cWait
    SETB    ISCL
    LCALL   I2cWait
    CLR     ISCL
    LCALL   I2cWait
    INC     R6
    CJNE    R6,#08H,?C0011
    SETB    ISDA
    LCALL   I2cWait
    SETB    ISCL
    LCALL   I2cWait
    MOV     C,ISDA
    MOV     ack?545,C
    CLR     ISCL
    LCALL   I2cWait
    MOV     C,ack?545
    RET

I2cReceiveByte:
    CLR     A
    MOV     R7,A
    MOV     R6,A
?C0017:
    SETB    ISCL
    LCALL   I2cWait
    MOV     A,R7
    ADD     A,ACC
    MOV     R7,A
    JNB     ISDA,?C0020
    XCH     A,R7
    ORL     A,#01H
    XCH     A,R7
?C0020:
    CLR     ISCL
    LCALL   I2cWait
    INC     R6
    CJNE    R6,#08H,?C0017
    RET

SendAcknowledge:
    MOV     C,ack?748
    MOV     ISDA,C
    SETB    ISCL
    LCALL   I2cWait
    CLR     ISCL
    RET

_I2cByteWrite:
    XCH     A,R6
    MOV     A,R7
    XCH     A,R6
    XCH     A,R4
    MOV     A,R5
    XCH     A,R4
    LCALL   I2cStart
    LCALL   _I2cSentByte
    XCH     A,R7
    MOV     A,R4
    XCH     A,R7
    LCALL   _I2cSentByte
    XCH     A,R7
    MOV     A,R3
    XCH     A,R7
```

```
    LCALL  _I2cSentByte
    LCALL  I2cStop
    MOV    R7,#0AH
    MOV    R6,#00H
    LCALL  _DelayX1ms
    RET


_I2cByteWrite1:
    XCH    A,R4
    MOV    A,R7
    XCH    A,R4
    XCH    A,R2
    MOV    A,R5
    XCH    A,R2
    LCALL  I2cStart
?RESENT1:
    XCH    A,R7
    MOV    A,R4
    XCH    A,R7
    LCALL  _I2cSentByte
    JB     ack?955,?RESENT1
?RESEN12:
    XCH    A,R7
    MOV    A,R2
    XCH    A,R7
    LCALL  _I2cSentByte
    JB     ack?955,?RESENT2
?RESENT3:
    XCH    A,R7
    MOV    A,R3
    XCH    A,R7
    LCALL  _I2cSentByte
    JB     ack?955,?RESENT3
    LCALL  I2cStop
    MOV    R7,#0AH
    MOV    R6,#00H
    LCALL  _DelayX1ms
    RET

_I2cByteWrite2:
    XCH    A,R4
    MOV    A,R7
    XCH    A,R4
    XCH    A,R2
    MOV    A,R5
    XCH    A,R2
    SETB   Fg1imeout
    CLR    A
    MOV    R1,A
?C0031:
    LCALL  I2cStart
    XCH    A,R7
    MOV    A,R4
    XCH    A,R7
    LCALL  _I2cSentByte
    MOV    ack?1060,C
    JNB    ack?1060,?C0034
    LCALL  I2cStop
    SJMP   ?C0033
?C0034:
    XCH    A,R7
    MOV    A,R2
    XCH    A,R7
    LCALL  I2cSentByte
    MOV    ack?1060,C
    JNB    ack?1060,?C0035
    LCALL  I2cStop
    SJMP   ?C0033
?C0035:
    XCH    A,R7
    MOV    A,R3
    XCH    A,R7
    LCALL  _I2cSentByte
    MOV    ack?1060,C
    JNB    ack?1060,?C0036
    LCALL  I2cStop
    SJMP   ?C0033
?C0036:
    LCALL  I2cStop
    CLR    FgTimeout
    JNB    ack?1060,?C0032
?C0033:
    INC    R1
    CJNE   R1,#0AH,?C0031
?C0032:
    MOV    R7,#0AH
    MOV    R6,#00H
    LCALL  _DelayX1ms
    RET


_I2cByteRead:
    XCH    A,R4
    MOV    A,R7
    XCH    A,R4

    XCH    A,R3
    MOV    A,R5
    XCH    A,R3
    LCALL  I2cStart
    LCALL  _I2cSentByte
    XCH    A,R7
    MOV    A,R3
    XCH    A,R7
    LCALL  _I2cSentByte
    LCALL  I2cStart
    MOV    A,R4
```

```
    ORL     A,#01H
    MOV     R7,A
    LCALL   _I2cSentByte
    LCALL   I2cReceiveByte
    SETB    ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
    LCALL   I2cStop
    RET


_I2cSentData:
    XCH     A,R4
    MOV     A,R7
    XCH     A,R4
    CLR     A
    MOV     R3,A
?C0040:
    MOV     A,R3
    CLR     C
    SUBB    A,R4
    JNC     ?C0041
    MOV     A,#TrmBuf
    ADD     A,R3
    MOV     R0,A
    MOV     A,@R0
    MOV     R7,A
    LCALL   _I2cSentByte
    JNC     ?C0042
    SETB    C
    RET
?C0042:
    INC     R3
    SJMP    ?C0040
?C0041:
    CLR     C
    RET


_I2cReceiveData:
    XCH     A,R5
    MOV     A,R7
    XCH     A,R5
    CLR     A
    MOV     R4,A
?C0045:
    MOV     A,R5
    DEC     A
    MOV     R7,A
    MOV     A,R4
    CLR     C
    SUBB    A,R7
    JNC     ?C0046
    LCALL   I2cReceiveByte
    MOV     A,#TrmBuf
    ADD     A,R4
    MOV     R0,A
    MOV     A,R7
    MOV     @R0,A
    CLR     ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
    INC     R4
    SJMP    ?C0045
?C0046:
    LCALL   I2cReceiveByte
    MOV     A,#TrmBuf
    ADD     A,R4
    MOV     R0,A
    MOV     A,R7
    MOV     @R0,A
    SETB    ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
    LCALL   I2cStop
    CLR     C
    RET


_DataSetBit:
    MOV     bitno?1470,R3
    XCH     A,R2
    MOV     A,R7
    XCH     A,R2
    XCH     A,R1
    MOV     A,R5
    XCH     A,R1
    LCALL   _I2cByteRead
    XCH     A,R3
    MOV     A,R7
    XCH     A,R3
    MOV     R7,bitno?1470
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0058
?C0057:
    CLR     C
    RLC     A
?C0058:
    DJNZ    R0,?C0057
    XCH     A,R3
    ORL     A,R3
    XCH     A,R3
    XCH     A,R7
    MOV     A,R2
    XCH     A,R7
    XCH     A,R5
    MOV     A,R1
```

```
    XCH     A,R5
    LCALL   _I2cByteWrite
    RET

_DataSetBit1:
    MOV     bitno?1574,R3
    XCH     A,R2
    MOV     A,R7
    XCH     A,R2

    XCH     A,R1
    MOV     A,R5
    XCH     A,R1
    LCALL   _I2cByteRead
    MOV     R6,bitno?1574
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R6
    XCH     A,R0
    INC     R0
    SJMP    ?C0060
?C0059:
    CLR     C
    RLC     A
?C0060:
    DJNZ    R0,?C0059
    ORL     A,R7
    MOV     R3,A
    XCH     A,R7
    MOV     A,R2
    XCH     A,R7
    XCH     A,R5
    MOV     A,R1
    XCH     A,R5
    LCALL   _I2cByteWrite
    RET

_DataSetBit2:
    XCH     A,R2
    MOV     A,R7
    XCH     A,R2
    XCH     A,R1
    MOV     A,R5
    XCH     A,R1
    LCALL   _I2cByteRead
    MOV     A,R7
    ORL     A,#080H
    MOV     R3,A
    XCH     A,R7
    MOV     A,R2
    XCH     A,R7
    XCH     A,R5
    MOV     A,R1
    XCH     A,R5
    LCALL   _I2cByteWrite
    RET

_DataClearBit:
    MOV     bitno?1779,R3
    XCH     A,R2
    MOV     A,R7
    XCH     A,R2
    XCH     A,R1
    MOV     A,R5
    XCH     A,R1
    LCALL   _I2cByteRead
    XCH     A,R3
    MOV     A,R7
    XCH     A,R3
    MOV     R7,bitno?1779
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R7
    XCH     A,R0
    INC     R0
    SJMP    ?C0062
?C0061:
    CLR     C
    RLC     A
?C0062:
    DJNZ    R0,?C0061
    CPL     A
    XCH     A,R3
    ANL     A,R3
    XCH     A,R3
    XCH     A,R7
    MOV     A,R2
    XCH     A,R7
    XCH     A,R5
    MOV     A,R1
    XCH     A,R5
    LCALL   _I2cByteWrite
    RET

_DataClearBit1:
    MOV     bitno?1883,R3
    XCH     A,R2
    MOV     A,R7
    XCH     A,R2
    XCH     A,R1
    MOV     A,R5
    XCH     A,R1
    LCALL   _I2cByteRead
    MOV     R6,bitno?1883
```

```
    MOV     A,#01H
    XCH     A,R0
    MOV     A,R6
    XCH     A,R0
    INC     R0
    SJMP    ?C0064
?C0063:
    CLR     C
    RLC     A
?C0064:
    DJNZ    R0,?C0063
    CPL     A
    ANL     A,R7
    MOV     R3,A
    XCH     A,R7
    MOV     A,R2
    XCH     A,R7
    XCH     A,R5
    MOV     A,R1
    XCH     A,R5
    LCALL   _I2cByteWrite
    RET

_DataClearBit2:
    XCH     A,R2
    MOV     A,R7
    XCH     A,R2
    XCH     A,R1
    MOV     A,R5
    XCH     A,R1
    LCALL   _I2cByteRead
    MOV     A,R7
    ANL     A,#07FH
    MOV     R3,A
    XCH     A,R7
    MOV     A,R2
    XCH     A,R7
    XCH     A,R5
    MOV     A,R1
    XCH     A,R5
    LCALL   _I2cByteWrite
    RET

PWM_OutputHI:
    MOV     R0,#LOW (DAC0)
    MOV     A,#0FFH
    MOVX    @R0,A
    RET

PWM_OutputLOW:
    CLR     A
    MOV     R0,#LOW (DAC0)
    MOVX    @R0,A
    RET

_PWM_Output:
    MOV     R0,#LOW (DAC0)
    MOV     A,R7
    MOVX    @R0,A
    RET

TEST_DacOut:
    MOV     R7,#090H
    MOV     R5,#01H
    MOV     R3,#080H
    LCALL   _DacByteWrite
    RET

_DacByteWrite:
    MOV     device?444,R7
    MOV     subaddress?445,R5
    MOV     bytedata?446,R3
    LCALL   I2cStart
    MOV     R7,device?444
    LCALL   _I2cSentByte
    MOV     R7,subaddress?445
    LCALL   _I2cSentByte
    MOV     R7,bytedata?446
    LCALL   _I2cSentByte
    LCALL   I2cStop
    RET

_EepromByteWrite0:
    MOV     A,R7
    ORL     A,#0A0H
    MOV     R7,A
    LCALL   _I2cByteWrite
    RET

_EepromWordWrite0:
    MOV     bank?143,R7
    MOV     addr?144,R5
    MOV     value?145,R2
    MOV     value?145+01H,R3
    MOV     A,bank?143
    ORL     A,#0A0H
    MOV     R7,A
    MOV     A,value?145
    MOV     R3,A
    LCALL   _I2cByteWrite
    MOV     A,bank?143
```

```
    ORL     A,#0A0H
    MOV     R7,A
    MOV     A,addr?144
    INC     A
    MOV     R5,A
    MOV     R3,value?145+01H
    LCALL   _I2cByteWrite
    RET


_EepromByteRead0:
    MOV     A,R7
    ORL     A,#0A0H
    MOV     R7,A
    LCALL   _I2cByteRead
    RET


_EepromWordRead0:
    MOV     bank?348,R7
    MOV     addr?349,R5
    MOV     A,bank?348
    ORL     A,#0A0H
    MOV     R7,A
    LCALL   _I2cByteRead
    MOV     A,R7
    MOV     R7,#00H
    MOV     R6,A
    MOV     A,R6
    PUSH    ACC
    MOV     A,R7
    PUSH    ACC
    MOV     A,bank?348
    ORL     A,#0A0H
    MOV     R7,A
    MOV     A,addr?349
    INC     A
    MOV     R5,A
    LCALL   _I2cByteRead
    MOV     A,R7
    MOV     R5,A
    POP     ACC
    ADD     A,R5
    MOV     R7,A
    POP     ACC
    ADDC    A,#00H
    MOV     R6,A
    RET


_EepromByteWrite:
    MOV     R0,#TrmBuf+01H
    MOV     A,R5
    MOV     @R0,A
    MOV     R5,#01H
    LCALL   _EepromWrite
    RET


_EepromByteRead:
    MOV     R5,#01H
    LCALL   _EepromRead
    MOV     R0,#TrmBuf
    MOV     A,@R0
    MOV     R7,A
    RET


EepromPageWrite:
    MOV     R0,#Variable1
    MOV     A,@R0
    MOV     R0,#TrmBuf+01H
    MOV     @R0,A
    MOV     R0,#Variable2
    MOV     A,@R0
    MOV     R0,#TrmBuf+02H
    MOV     @R0,A
    MOV     R0,#Variable3
    MOV     A,@R0
    MOV     R0,#TrmBuf+03H
    MOV     @R0,A
    MOV     R0,#Variable4
    MOV     A,@R0
    MOV     R0,#TrmBuf+04H
    MOV     @R0,A
    MOV     R0,#Variable5
    MOV     A,@R0
    MOV     R0,#TrmBuf+05H
    MOV     @R0,A
    MOV     R0,#Variable6
    MOV     A,@R0
    MOV     R0,#TrmBuf+06H
    MOV     @R0,A
    MOV     R0,#Variable7
    MOV     A,@R0
    MOV     R0,#TrmBuf+07H
    MOV     @R0,A
    MOV     C,FgVariable1
    CLR     A
    RLC     A
    INC     R0
    MOV     @R0,A
    MOV     C,FgVariable2
    CLR     A
    RLC     A
    INC     R0
    MOV     @R0,A
    MOV     C,FgVariable3
```

```
    CLR     A
    RLC     A
    INC     R0
    MOV     @R0,A
    MOV     R7,#050H
    MOV     R6,#00H
    MOV     R5,#0AH
    LCALL   _EepromWrite
    RET

EepromPageRead:
    MOV     R7,#050H
    MOV     R6,#00H
    MOV     R5,#0AH
    LCALL   _EepromRead
    MOV     R0,#TrmBuf
    MOV     A,@R0
    MOV     R0,#Variable1
    MOV     @R0,A
    MOV     R0,#TrmBuf+01H
    MOV     A,@R0
    MOV     R0,#Variable2
    MOV     @R0,A
    MOV     R0,#TrmBuf+02H
    MOV     A,@R0
    MOV     R0,#Variable3
    MOV     @R0,A
    MOV     R0,#TrmBuf+03H
    MOV     A,@R0
    MOV     R0,#Variable4
    MOV     @R0,A
    MOV     R0,#TrmBuf+04H
    MOV     A,@R0
    MOV     R0,#Variable5
    MOV     @R0,A
    MOV     R0,#TrmBuf+05H
    MOV     A,@R0
    MOV     R0,#Variable6
    MOV     @R0,A
    MOV     R0,#TrmBuf+06H
    MOV     A,@R0
    MOV     R0,#Variable7
    MOV     @R0,A
    MOV     R0,#TrmBuf+07H
    MOV     A,@R0
    ADD     A,#0FFH
    MOV     FgVariable1,C
    INC     R0
    MOV     A,@R0
    ADD     A,#0FFH
    MOV     FgVariable2,C
    INC     R0
    MOV     A,@R0
    ADD     A,#0FFH
    MOV     FgVariable3,C
    RET
_EepromWrite:
    MOV     subaddress?853,R6
    MOV     subaddress?853+01H,R7
    MOV     count?854,R5
    CLR     C
    MOV     A,subaddress?853
    SUBB    A,#01H
    JNC     ?C0009
    MOV     R0,#SlvAdr
    MOV     @R0,#0A0H
    SJMP    ?C0010
?C0009:
    CLR     C
    MOV     A,subaddress?853
    SUBB    A,#02H
    JNC     ?C0011
    MOV     R0,#SlvAdr
    MOV     @R0,#0A2H
    CLR     A
    ADD     A,subaddress?853+01H
    MOV     subaddress?853+01H,A
    MOV     A,#0FFH
    ADDC    A,subaddress?853
    MOV     subaddress?853,A
    SJMP    ?C0010
?C0011:
    CLR     C
    MOV     A,subaddress?853
    SUBB    A,#03H
    JNC     ?C0013
    MOV     R0,#SlvAdr
    MOV     @R0,#0A4H
    CLR     A
    ADD     A,subaddress?853+01H
    MOV     subaddress?853+01H,A
    MOV     A,#0FEH
    ADDC    A,subaddress?853
    MOV     subaddress?853,A
    SJMP    ?C0010
?C0013:
    MOV     R0,#SlvAdr
    MOV     @R0,#0A6H
    CLR     A
    ADD     A,subaddress?853+01H
    MOV     subaddress?853+01H,A
    MOV     A,#0FDH
    ADDC    A,subaddress?853
    MOV     subaddress?853,A
?C0010:
    MOV     i?855,#01H
?C0015:
```

```
    MOV     A,i?855
    SETB    C
    SUBB    A,count?854
    JNC     ?C0016
    MOV     R0,#TrmBuf
    MOV     A,subaddress?853+01H
    MOV     @R0,A
    MOV     A,#TrmBuf
    ADD     A,i?855
    MOV     R0,A
    MOV     A,@R0
    MOV     R0,#TrmBuf+01H
    MOV     @R0,A
    MOV     R0,#ByteCnt
    MOV     @R0,#02H
    LCALL   SendEepromData
    INC     subaddress?853+01H
    MOV     A,subaddress?853+01H
    JNZ     ?C0073
    INC     subaddress?853
?C0073:
    INC     i?855
    SJMP    ?C0015
?C0016:
    MOV     R7,#03H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    RET
_EepromRead:
    MOV     count?957,R5
    CLR     C
    MOV     A,R6
    SUBB    A,#01H
    JNC     ?C0019
    MOV     R0,#SlvAdr
    MOV     @R0,#0A0H
    SJMP    ?C0020
?C0019:
    CLR     C
    MOV     A,R6
    SUBB    A,#02H
    JNC     ?C0021
    MOV     R0,#SlvAdr
    MOV     @R0,#0A2H
    CLR     A
    ADD     A,R7
    MOV     R7,A
    MOV     A,#0FFH
    ADDC    A,R6
    MOV     R6,A
    SJMP    ?C0020
?C0021:
    CLR     C
    MOV     A,R6
    SUBB    A,#03H
    JNC     ?C0023
    MOV     R0,#SlvAdr
    MOV     @R0,#0A4H
    CLR     A
    ADD     A,R7
    MOV     R7,A
    MOV     A,#0FEH
    ADDC    A,R6
    MOV     R6,A
    SJMP    ?C0020
?C0023:
    MOV     R0,#SlvAdr
    MOV     @R0,#0A6H
    CLR     A
    ADD     A,R7
    MOV     R7,A
    MOV     A,#0FDH
    ADDC    A,R6
    MOV     R6,A
?C0020:
    MOV     R0,#TrmBuf
    MOV     A,R7
    MOV     @R0,A
    MOV     R0,#ByteCnt
    MOV     @R0,#01H
    LCALL   SendData
    MOV     R0,#ByteCnt
    MOV     @R0,count?957
    LCALL   RcvData
    RET


SendEepromData:
    SETB    no_ack?1060
    CLR     A
    MOV     i?1058,A
?C0026:
    MOV     R0,#SlvAdr
    MOV     A,@R0
    MOV     R7,A
    LCALL   _GoMaster
    JNC     ?C0029
    LCALL   SendStop
    MOV     R7,#0AH
    MOV     R6,#00H
    LCALL   _DelayX1ms
    SJMP    ?C0028
?C0029:
    CLR     no_ack?1060
    CLR     A
    MOV     j?1059,A
?C0030:
    MOV     A,j?1059
```

```
    CLR     C
    MOV     R0,#ByteCnt
    SUBB    A,@R0
    JNC     ?C0031
    MOV     A,#TrmBuf
    ADD     A,j?1059
    MOV     R0,A
    MOV     A,@R0
    MOV     R7,A
    LCALL   _SendByte
    JNC     ?C0032
    SETB    no_ack?1060
    SJMP    ?C0031
?C0032:
    INC     j?1059
    SJMP    ?C0030
?C0031:
    LCALL   SendStop
    JNB     no_ack?1060,?C0027
?C0034:
    MOV     R7,#0AH
    MOV     R6,#00H
    LCALL   _DelayX1ms
?C0028:
    INC     i?1058
    MOV     A,i?1058
    CLR     C
    SUBB    A,#0AH
    JC      ?C0026
?C0027:
    MOV     C,no_ack?1060
    RET


SendData:
    SETB    no_ack?1163
    CLR     A
    MOV     i?1161,A
?C0036:
    MOV     R0,#SlvAdr
    MOV     A,@R0
    MOV     R7,A
    LCALL   _GoMaster
    JNC     ?C0039
    LCALL   SendStop
    SJMP    ?C0038
?C0039:
    CLR     no_ack?1163
    CLR     A
    MOV     j?1162,A
?C0040:
    MOV     A,j?1162
    CLR     C
    MOV     R0,#ByteCnt
    SUBB    A,@R0
    JNC     ?C0041
    MOV     A,#TrmBuf
    ADD     A,j?1162
    MOV     R0,A
    MOV     A,@R0
    MOV     R7,A
    LCALL   _SendByte
    JNC     ?C0042
    SETB    no_ack?1163
    SJMP    ?C0041
?C0042:
    INC     j?1162
    SJMP    ?C0040
?C0041:
    LCALL   SendStop
    JNB     no_ack?1163,?C0037
?C0038:
    INC     i?1161
    MOV     A,i?1161
    CLR     C
    SUBB    A,#0AH
    JC      ?C0036
?C0037:
    MOV     C,no_ack?1163
    RET


RcvData:
    MOV     R0,#SlvAdr
    MOV     A,@R0
    INC     A
    MOV     R7,A
    LCALL   _GoMaster
    JNC     ?C0046
    LCALL   SendStop
    SETB    C
    RET
?C0046:
    CLR     A
    MOV     i?1264,A
?C0048:
    MOV     A,i?1264
    CLR     C
    MOV     R0,#ByteCnt
    SUBB    A,@R0
    JNC     ?C0049
    CLR     A
    MOV     bval?1266,A
    MOV     j?1265,A
?C0051:
    SETB    SCL_PIN
    LCALL   I2cDelay
    MOV     A,bval?1266
```

```
    ADD     A,ACC
    MOV     bval?1266,A
    JNB     SDA_PIN,?C0054
    ORL     bval?1266,#01H
?C0054:
    CLR     SCL_PIN
    LCALL   I2cDelay
    INC     j?1265
    MOV     A,j?1265
    CLR     C
    SUBB    A,#08H
    JC      ?C0051
?C0052:
    MOV     R0,#ByteCnt
    MOV     A,@R0
    DEC     A
    XRL     A,i?1264
    JNZ     ?C0055
    SETB    SDA_PIN
    SJMP    ?C0056
?C0055:
    CLR     SDA_PIN
?C0056:
    LCALL   I2cDelay
    SETB    SCL_PIN
    LCALL   I2cDelay
    CLR     SCL_PIN
    LCALL   I2cDelay
    SETB    SDA_PIN
    LCALL   I2cDelay
    MOV     A,#TrmBuf
    ADD     A,i?1264
    MOV     R0,A
    MOV     @R0,bval?1266
    INC     i?1264
    SJMP    ?C0048
?C0049:
    LCALL   SendStop
    CLR     C
    RET


_GoMaster:
    MOV     slaveaddr?1367,R7
    JNB     SCL_PIN,?C0058
    JB      SDA_PIN,?C0057
?C0058:
    SETB    C
    RET
?C0057:
    CLR     SDA_PIN
    LCALL   I2cDelay
    CLR     SCL_PIN
    LCALL   I2cDelay
    MOV     R7,slaveaddr?1367
    LCALL   _SendByte
    CLR     C
    RET


SendByte:
    MOV     value?1468,R7
    CLR     no_ack?1470
    CLR     A
    MOV     i?1469,A
?C0060:
    MOV     A,value?1468
    JNB     ACC.7,?C0063
    SETB    SDA_PIN
    SJMP    ?C0064
?C0063:
    CLR     SDA_PIN
?C0064:
    MOV     A,value?1468
    ADD     A,ACC
    MOV     value?1468,A
    LCALL   I2cDelay
    SETB    SCL_PIN
    LCALL   I2cDelay
    CLR     SCL_PIN
    LCALL   I2cDelay
    INC     i?1469
    MOV     A,i?1469
    CLR     C
    SUBB    A,#08H
    JC      ?C0060
?C0061:
    SETB    SDA_PIN
    LCALL   I2cDelay
    SETB    SCL_PIN
    LCALL   I2cDelay
    JNB     SDA_PIN,?C0065
    SETB    no_ack?1470
?C0065:
    CLR     SCL_PIN
    LCALL   I2cDelay
    MOV     C,no_ack?1470
    RET


SendStop:
    CLR     SDA_PIN
    CLR     SCL_PIN
    LCALL   I2cDelay
    SETB    SCL_PIN
    LCALL   I2cDelay
    SETB    SDA_PIN
    LCALL   I2cDelay
```

```
    RET


I2cDelay:
    NOP
    NOP
    NOP
    NOP
    RET


_DdcChecksum:
    MOV     adr?1771,R7
    CLR     A
    MOV     temp?1773,A
    MOV     i?1772,adr?1771
?C0069:
    MOV     A,adr?1771
    ADD     A,#07FH
    MOV     R7,A
    CLR     A
    RLC     A
    MOV     R6,A
    CLR     C
    MOV     A,i?1772
    SUBB    A,R7
    MOV     A,R6
    XRL     A,#080H
    MOV     R0,A
    MOV     A,#080H
    SUBB    A,R0
    JNC     ?C0070
    MOV     R7,i?1772
    MOV     R6,#00H
    MOV     R5,#01H
    LCALL   _EepromRead
    MOV     R0,#TrmBuf
    MOV     A,@R0
    ADD     A,temp?1773
    MOV     temp?1773,A
    INC     i?1772
    SJMP    ?C0069
?C0070:
    MOV     A,temp?1773
    CPL     A
    INC     A
    MOV     temp?1773,A
    MOV     R0,#TrmBuf+01H
    MOV     @R0,A
    MOV     A,adr?1771
    ADD     A,#07FH
    MOV     R7,A
    CLR     A
    RLC     A
    MOV     R6,A
    MOV     R5,#01H
    LCALL   _EepromWrite
    MOV     R7,temp?1773
    RET


_Eeprom24c32WriteByte_1:
    MOV     addr?040,R6
    MOV     addr?040+01H,R7
    MOV     value?041,R5
    LCALL   I2cStart
    MOV     R7,#0A0H
    LCALL   _I2cSentByte
    MOV     A,addr?040
    MOV     R7,A
    LCALL   _I2cSentByte
    MOV     R7,addr?040+01H
    MOV     A,addr?040+01H
    LCALL   _I2cSentByte
    MOV     R7,value?041
    LCALL   _I2cSentByte
    LCALL   I2cStop
    RET


_Eeprom24c32WriteByte_2:
    MOV     addr?142,R6
    MOV     addr?142+01H,R7
    MOV     value?143,R5
    LCALL   I2cStart
    CLR     C
    MOV     A,addr?142
    SUBB    A,#080H
    JNC     ?C0002
    MOV     R7,#0A0H
    LCALL   _I2cSentByte
    SJMP    ?C0003
?C0002:
    MOV     R7,#0A2H
    LCALL   _I2cSentByte
?C0003:
    MOV     A,addr?142
    ANL     A,#07FH
    MOV     R7,A
    LCALL   _I2cSentByte
    MOV     R7,addr?142+01H
    LCALL   _I2cSentByte
    MOV     R7,value?143
    LCALL   _I2cSentByte
    LCALL   I2cStop
    RET
```

```
_Eeprom24c32WriteMulti_1:
    MOV     addr?244,R6
    MOV     addr?244+01H,R7
    MOV     count?245,R5
    LCALL   I2cStart
    MOV     R7,#0A0H
    LCALL   _I2cSentByte
    MOV     A,addr?244
    MOV     R7,A
    LCALL   _I2cSentByte
    MOV     R7,addr?244+01H
    MOV     A,addr?244+01H
    LCALL   _I2cSentByte
    CLR     A
    MOV     i?246,A
?C0005:
    MOV     A,i?246
    CLR     C
    SUBB    A,count?245
    JNC     ?C0006
    MOV     A,#TrmBuf
    ADD     A,i?246
    MOV     R0,A
    MOV     A,@R0
    MOV     R7,A
    LCALL   _I2cSentByte
    INC     i?246
    SJMP    ?C0005
?C0006:
    LCALL   I2cStop
    RET

_Eeprom24c32WriteMulti_2:
    MOV     addr?347,R6
    MOV     addr?347+01H,R7
    MOV     count?348,R5
    LCALL   I2cStart
    CLR     C
    MOV     A,addr?347
    SUBB    A,#080H
    JNC     ?C0009
    MOV     R7,#0A0H
    LCALL   _I2cSentByte
    SJMP    ?C0010
?C0009:
    MOV     R7,#0A2H
    LCALL   _I2cSentByte
?C0010:
    MOV     A,addr?347
    ANL     A,#07FH
    MOV     R7,A
    LCALL   _I2cSentByte
    MOV     R7,addr?347+01H
    LCALL   _I2cSentByte
    CLR     A
    MOV     i?349,A
?C0011:
    MOV     A,i?349
    CLR     C
    SUBB    A,count?348
    JNC     ?C0012
    MOV     A,#TrmBuf
    ADD     A,i?349
    MOV     R0,A
    MOV     A,@R0
    MOV     R7,A
    LCALL   _I2cSentByte
    INC     i?349
    SJMP    ?C0011
?C0012:
    LCALL   I2cStop
    RET

_Eeprom24c32ReadByte_1:
    MOV     addr?450,R6
    MOV     addr?450+01H,R7
    LCALL   I2cStart
    MOV     R7,#0A0H
    LCALL   _I2cSentByte
    MOV     A,addr?450
    MOV     R7,A
    LCALL   _I2cSentByte
    MOV     R7,addr?450+01H
    MOV     A,addr?450+01H
    LCALL   _I2cSentByte
    LCALL   I2cStart
    MOV     R7,#0A1H
    LCALL   _I2cSentByte
    LCALL   I2cReceiveByte
    MOV     temp?451,R7
    SETB    ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
    LCALL   I2cStop
    MOV     R7,temp?451
?C0015:
    RET

_Eeprom24c32ReadByte_2:
    MOV     addr?552,R6
    MOV     addr?552+01H,R7
    MOV     A,addr?552
    JNB     ACC.7,?C0016
    MOV     temp?553,#0A2H
    SJMP    ?C0017
?C0016:
    MOV     temp?553,#0A0H
```

```
?C0017:
    LCALL   I2cStart
    MOV     R7,temp?553
    LCALL   _I2cSentByte
    MOV     A,addr?552
    ANL     A,#07FH
    MOV     R7,A
    LCALL   _I2cSentByte
    MOV     R7,addr?552+01H
    LCALL   _I2cSentByte
    LCALL   I2cStart
    MOV     A,temp?553
    ORL     A,#01H
    MOV     R7,A
    LCALL   _I2cSentByte
    LCALL   I2cReceiveByte
    MOV     temp?553,R7
    SETB    ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
    LCALL   I2cStop
    MOV     R7,temp?553
?C0018:
    RET


_Eeprom24c32ReadWord_1:
    MOV     addr?654,R6
    MOV     addr?654+01H,R7
    LCALL   I2cStart
    MOV     R7,#0A0H
    LCALL   _I2cSentByte
    MOV     A,addr?654
    MOV     R7,A
    LCALL   _I2cSentByte
    MOV     R7,addr?654+01H
    MOV     A,addr?654+01H
    LCALL   _I2cSentByte
    LCALL   I2cStart
    MOV     R7,#0A1H
    LCALL   _I2cSentByte
    LCALL   I2cReceiveByte
    MOV     value?656,#00H
    MOV     value?656+01H,R7
    CLR     ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
    MOV     A,value?656+01H
    MOV     value?656+01H,#00H
    MOV     value?656,A
    LCALL   I2cReceiveByte
    MOV     A,R7
    ORL     value?656+01H,A
    CLR     A
    SETB    ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
    LCALL   I2cStop
    MOV     R6,value?656
    MOV     R7,value?656+01H
?C0019:
    RET


_Eeprom24c32ReadWord_2:
    MOV     addr?757,R6
    MOV     addr?757+01H,R7
    LCALL   I2cStart
    CLR     C
    MOV     A,addr?757
    SUBB    A,#080H
    JNC     ?C0020
    MOV     R7,#0A0H
    LCALL   _I2cSentByte
    SJMP    ?C0021
?C0020:
    MOV     R7,#0A2H
    LCALL   _I2cSentByte
?C0021:
    MOV     A,addr?757
    ANL     A,#07FH
    MOV     R7,A
    LCALL   I2cSentByte
    MOV     R7,addr?757+01H
    LCALL   I2cSentByte
    LCALL   I2cStart
    CLR     C
    MOV     A,addr?757
    SUBB    A,#080H
    JNC     ?C0022
    MOV     R7,#0A1H
    LCALL   _I2cSentByte
    SJMP    ?C0023
?C0022:
    MOV     R7,#0A3H
    LCALL   _I2cSentByte
?C0023:
    LCALL   I2cReceiveByte
    MOV     value?759,#00H
    MOV     value?759+01H,R7
    CLR     ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
    MOV     A,value?759+01H
    MOV     value?759+01H,#00H
    MOV     value?759,A
    LCALL   I2cReceiveByte
    MOV     A,R7
    ORL     value?759+01H,A
    CLR     A
    SETB    ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
```

```
    LCALL   I2cStop
    MOV     R6,value?759
    MOV     R7,value?759+01H
    RET


_Eeprom24c32ReadMulti_1:
    MOV     addr?860,R6
    MOV     addr?860+01H,R7
    MOV     count?861,R5
    LCALL   I2cStart
    MOV     R7,#0A0H
    LCALL   _I2cSentByte
    MOV     A,addr?860
    MOV     R7,A
    LCALL   _I2cSentByte
    MOV     R7,addr?860+01H
    MOV     A,addr?860+01H
    LCALL   _I2cSentByte
    LCALL   I2cStart
    MOV     R7,#0A1H
    LCALL   _I2cSentByte
    CLR     A
    MOV     i?862,A
?C0025:
    MOV     A,i?862
    CLR     C
    SUBB    A,count?861
    JNC     ?C0026
    LCALL   I2cReceiveByte
    MOV     A,#FrmBuf
    ADD     A,i?862
    MOV     R0,A
    MOV     A,R7
    MOV     @R0,A
    MOV     A,count?861
    DEC     A
    XRL     A,i?862
    JZ      ?C0027
    CLR     ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
?C0027:
    INC     i?862
    SJMP    ?C0025
?C0026:
    SETB    ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
    LCALL   I2cStop
    RET
_Eeprom24c32ReadMulti_2:
    MOV     addr?963,R6
    MOV     addr?963+01H,R7
    MOV     count?964,R5
    LCALL   I2cStart
    CLR     C
    MOV     A,addr?963
    SUBB    A,#080H
    JNC     ?C0030
    MOV     R7,#0A0H
    LCALL   _I2cSentByte
    SJMP    ?C0031
?C0030:
    MOV     R7,#0A2H
    LCALL   _I2cSentByte
?C0031:
    MOV     A,addr?963
    ANL     A,#07FH
    MOV     R7,A
    LCALL   _I2cSentByte
    MOV     R7,addr?963+01H
    LCALL   _I2cSentByte
    LCALL   I2cStart
    CLR     C
    MOV     A,addr?963
    SUBB    A,#080H
    JNC     ?C0032
    MOV     R7,#0A1H
    LCALL   _I2cSentByte
    SJMP    ?C0033
?C0032:
    MOV     R7,#0A3H
    LCALL   _I2cSentByte
?C0033:
    CLR     A
    MOV     i?965,A
?C0034:
    MOV     A,i?965
    CLR     C
    SUBB    A,count?964
    JNC     ?C0035
    LCALL   I2cReceiveByte
    MOV     A,#FrmBuf
    ADD     A,i?965
    MOV     R0,A
    MOV     A,R7
    MOV     @R0,A
    MOV     A,count?964
    DEC     A
    XRL     A,i?965
    JZ      ?C0036
    CLR     ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
?C0036:
    INC     i?965
    SJMP    ?C0034
?C0035:
    SETB    ?SendAcknowledge?BIT
    LCALL   SendAcknowledge
```

```
    LCALL  I2cStop
    RET


OsdStart:
    SETB   OSD_SDA
    SETB   OSD_SCL
    LCALL  I2cWait
    CLR    OSD_SDA
    LCALL  I2cWait
    CLR    OSD_SCL
    RET


OsdStop:
    CLR    OSD_SDA
    LCALL  I2cWait
    SETB   OSD_SCL
    LCALL  I2cWait
    SETB   OSD_SDA
    RET


_OsdSentByte:
    MOV    bytedata?240,R7
    CLR    A
    MOV    i?241,A
?C0003:
    MOV    R7,bytedata?240
    MOV    R0,#01H
    MOV    A,R7
    INC    R0
    SJMP   ?C0137
?C0136:
    RL     A
?C0137:
    DJNZ   R0,?C0136
    MOV    bytedata?240,A
    JNB    ACC.0,?C0006
    SETB   OSD_SDA
    SJMP   ?C0007
?C0006:
    CLR    OSD_SDA
?C0007:
    SETB   OSD_SCL
    LCALL  I2cWait
    CLR    OSD_SCL
    LCALL  I2cWait
    INC    i?241
    MOV    A,i?241
    CLR    C
    SUBB   A,#08H
    JC     ?C0003
?C0004:
    SETB   OSD_SDA
    LCALL  I2cWait
    SETB   OSD_SCL
    LCALL  I2cWait
    MOV    C,OSD_SDA
    MOV    ack?242,C
    CLR    OSD_SCL
    LCALL  I2cWait
    MOV    C,ack?242
?C0008:
    RET


OsdReceiveByte:
    CLR    A
    MOV    bytedata?344,A
    MOV    i?343,A
?C0009:
    SETB   OSD_SCL
    LCALL  I2cWait
    MOV    A,bytedata?344
    ADD    A,ACC
    MOV    bytedata?344,A
    JNB    OSD_SDA,?C0012
    ORL    bytedata?344,#01H
?C0012:
    CLR    OSD_SCL
    LCALL  I2cWait
    INC    i?343
    MOV    A,i?343
    CLR    C
    SUBB   A,#08H
    JC     ?C0009
?C0010:
    MOV    R7,bytedata?344
?C0013:
    RET


_OsdFormatA_0:
    MOV    row?445,R7
    MOV    col?446,R5
    MOV    value?447,R3
    LCALL  OsdStart
    MOV    R7,#07AH
    LCALL  _OsdSentByte
    MOV    R7,row?445
    LCALL  _OsdSentByte
    MOV    R7,col?446
    LCALL  _OsdSentByte
    MOV    R7,value?447
    LCALL  _OsdSentByte
    LCALL  OsdStop
    RET
```

```
_OsdFormatA:
    MOV     R0,#TrmBuf
    MOV     A,R7
    MOV     @R0,A
    INC     R0
    MOV     A,R5
    MOV     @R0,A
    INC     R0
    MOV     A,R3
    MOV     @R0,A
    MOV     R0,#SlvAdr
    MOV     @R0,#07AH
    MOV     R0,#ByteCnt
    MOV     @R0,#03H
    LCALL   SendData
    RET

_OsdFrameControl:
    MOV     hord?652,R5
    MOV     height?653,R3
    XCH     A,R6
    MOV     A,R7
    XCH     A,R6
    MOV     R7,#08FH
    MOV     R5,#0CH
    XCH     A,R3
    MOV     A,R6
    XCH     A,R3
    LCALL   _OsdFormatA
    MOV     R7,#08FH
    MOV     R5,#0DH
    MOV     R3,hord?652
    LCALL   _OsdFormatA
    MOV     R7,#08FH
    MOV     R5,#0EH
    MOV     R3,height?653
    LCALL   _OsdFormatA
    MOV     R7,#08FH
    MOV     R5,#0FH
    MOV     R3,width?654
    LCALL   _OsdFormatA
    MOV     R7,#08FH
    MOV     R5,#010H
    MOV     R3,rowspace?655
    LCALL   _OsdFormatA
    RET

_OsdLocationSet:
    MOV     horizontal?757,R5
    XCH     A,R3
    MOV     A,R7
    XCH     A,R3
    MOV     R7,#08FH
    MOV     R5,#0CH
    LCALL   _OsdFormatA
    MOV     R7,#08FH
    MOV     R5,#0DH
    MOV     R3,horizontal?757
    LCALL   _OsdFormatA
    RET

OsdRamClear:
    MOV     R7,#0AFH
    MOV     R5,#011H
    MOV     R3,#06H
    LCALL   _OsdFormatA
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX1ms
    MOV     R7,#0AFH
    MOV     R5,#011H
    CLR     A
    MOV     R3,A
    LCALL   _OsdFormatA
    RET

OsdEnable:
    JNB     yes?958,?C0019
    MOV     R7,#08FH
    MOV     R5,#011H
    MOV     R3,#080H
    LCALL   _OsdFormatA
    RET
?C0019:
    MOV     R7,#08FH
    MOV     R5,#011H
    CLR     A
    MOV     R3,A
    LCALL   _OsdFormatA
?C0021:
    RET

OsdOpenUp:
    MOV     R7,#08FH
    MOV     R5,#011H
    MOV     R3,#090H
    LCALL   _OsdFormatA
    RET

OsdNormal:
```

```
    MOV     R7,#08FH
    MOV     R5,#011H
    MOV     R3,#080H
    LCALL   _OsdFormatA
    RET


OsdResetFont:
    CLR     A
    MOV     i?1259,A
?C0024:
    CLR     A
    MOV     j?1260,A
?C0027:
    MOV     R7,i?1259
    MOV     R5,j?1260
    CLR     A
    MOV     R3,A
    MOV     ?_OsdPrintIcon?BYTE+03H,A
    LCALL   _OsdPrintIcon
    INC     j?1260
    MOV     A,j?1260
    CLR     C
    SUBB    A,#01EH
    JC      ?C0027
?C0026:
    INC     i?1259
    MOV     A,i?1259
    CLR     C
    SUBB    A,#0FH
    JC      ?C0024
?C0025:
    CLR     A
    MOV     i?1259,A
?C0030:
    MOV     A,i?1259
    ORL     A,#080H
    MOV     R7,A
    MOV     R5,#01EH
    CLR     A
    MOV     R3,A
    LCALL   _OsdFormatA
    INC     i?1259
    MOV     A,i?1259
    CLR     C
    SUBB    A,#0FH
    JC      ?C0030
?C0033:
    RET


_OsdClearRow:
    MOV     end?1362,R5
    MOV     color?1363,R3
    MOV     i?1364,R7
?C0034:
    MOV     A,i?1364
    SETB    C
    SUBB    A,end?1362
    JNC     ?C0043
    MOV     A,i?1364
    ORL     A,#0A0H
    MOV     R0,#TrmBuf
    MOV     @R0,A
    INC     R0
    MOV     @R0,#040H
    CLR     A
    MOV     j?1365,A
?C0037:
    MOV     A,#TrmBuf+02H
    ADD     A,j?1365
    MOV     R0,A
    MOV     @R0,color?1363
    INC     j?1365
    MOV     A,j?1365
    CJNE    A,#01EH,?C0037
?C0038:
    MOV     R0,#SlvAdr
    MOV     @R0,#07AH
    MOV     R0,#ByteCnt
    MOV     @R0,#020H
    LCALL   SendData
    MOV     A,i?1364
    ORL     A,#080H
    MOV     R0,#TrmBuf
    MOV     @R0,A
    INC     R0
    MOV     @R0,#040H
    CLR     A
    MOV     j?1365,A
?C0040:
    MOV     A,#TrmBuf+02H
    ADD     A,j?1365
    MOV     R0,A
    CLR     A
    MOV     @R0,A
    INC     j?1365
    MOV     A,j?1365
    CJNE    A,#01EH,?C0040
?C0041:
    MOV     R0,#SlvAdr
    MOV     @R0,#07AH
    MOV     R0,#ByteCnt
    MOV     @R0,#020H
    LCALL   SendData
    INC     i?1364
    SJMP    ?C0034
?C0043:
```

```
    RET


_OsdClearRow1:
    MOV     end?1467,R5
    MOV     color?1468,R3
    MOV     i?1469,R7
?C0044:
    MOV     A,i?1469
    SETB    C
    SUBB    A,end?1467
    JNC     ?C0053
    LCALL   OsdStart
    MOV     R7,#07AH
    LCALL   _OsdSentByte
    MOV     A,i?1469
    ORL     A,#0A0H
    MOV     R7,A
    LCALL   _OsdSentByte
    MOV     R7,#040H
    LCALL   _OsdSentByte
    CLR     A
    MOV     j?1470,A
?C0047:
    MOV     R7,color?1468
    LCALL   _OsdSentByte
    INC     j?1470
    MOV     A,j?1470
    CLR     C
    SUBB    A,#01EH
    JC      ?C0047
?C0048:
    LCALL   OsdStop
    LCALL   OsdStart
    MOV     R7,#07AH
    LCALL   _OsdSentByte
    MOV     A,i?1469
    ORL     A,#080H
    MOV     R7,A
    LCALL   _OsdSentByte
    MOV     R7,#040H
    LCALL   _OsdSentByte
    CLR     A
    MOV     j?1470,A
?C0050:
    CLR     A
    MOV     R7,A
    LCALL   _OsdSentByte
    INC     j?1470
    MOV     A,j?1470
    CLR     C
    SUBB    A,#01EH
    JC      ?C0050
?C0051:
    LCALL   OsdStop
    MOV     R7,#01H
    MOV     R6,#00H
    LCALL   _DelayX10ms
    INC     i?1469
    SJMP    ?C0044
?C0053:
    RET


_OsdPrintIcon:
    MOV     row?1571,R7
    MOV     col?1572,R5
    MOV     icon?1573,R3
    MOV     A,row?1571
    ORL     A,#0A0H
    MOV     R7,A
    MOV     R3,color?1574
    LCALL   _OsdFormatA
    MOV     A,row?1571
    ORL     A,#080H
    MOV     R7,A
    MOV     R5,col?1572
    MOV     R3,icon?1573
    LCALL   _OsdFormatA
    RET


_OsdStringAdr0:
    CLR     A
    MOV     R4,A
?C0055:
    MOV     A,R4
    CLR     C
    SUBB    A,R5
    JNC     ?C0056
?C0058:
    MOV     DPL,R7
    MOV     DPH,R6
    CLR     A
    MOVC    A,@A+DPTR
    CPL     A
    JZ      ?C0059
    INC     R7
    CJNE    R7,#00H,?C0138
    INC     R6
?C0138:
    SJMP    ?C0058
?C0059:
    INC     R7
    CJNE    R7,#00H,?C0139
    INC     R6
?C0139:
    INC     R4
```

```
    SJMP    ?C0055
?C0056:
    MOV     DataPointer,R6
    MOV     DataPointer+01H,R7
    RET


_OsdStringAdr:
    MOV     string?1778,R6
    MOV     string?1778+01H,R7
    XCH     A,R2
    MOV     A,R5
    XCH     A,R2
    JNB     fglanguage?1781. ?C0061
    CLR     A
    MOV     R5,A
?C0062:
    MOV     R0,#OsdLanguage
    MOV     A,@R0
    MOV     R7,A
    MOV     A,R5
    CLR     C
    SUBB    A,R7
    JNC     ?C0061
    CLR     A
    MOV     R4,A
?C0065:
    MOV     A,R4
    CLR     C
    SUBB    A,R2
    JNC     ?C0064
?C0068:
    MOV     DPL,string?1778+01H
    MOV     DPH,string?1778
    CLR     A
    MOVC    A,@A+DPTR
    CPL     A
    JZ      ?C0069
    INC     string?1778+01H
    MOV     A,string?1778+01H
    JNZ     ?C0140
    INC     string?1778
?C0140:
    SJMP    ?C0068
?C0069:
    INC     string?1778+01H
    MOV     A,string?1778+01H
    JNZ     ?C0141
    INC     string?1778
?C0141:
    INC     R4
    SJMP    ?C0065
?C0064:
    INC     R5
    SJMP    ?C0062
?C0061:
    CLR     A
    MOV     R4,A
?C0070:
    MOV     A,R4
    CLR     C
    SUBB    A,R3
    JNC     ?C0071
?C0073:
    MOV     DPL,string?1778+01H
    MOV     DPH,string?1778
    CLR     A
    MOVC    A,@A+DPTR
    CPL     A
    JZ      ?C0074
    INC     string?1778+01H
    MOV     A,string?1778+01H
    JNZ     ?C0142
    INC     string?1778
?C0142:
    SJMP    ?C0073
?C0074:
    INC     string?1778+01H
    MOV     A,string?1778+01H
    JNZ     ?C0143
    INC     string?1778
?C0143:
    INC     R4
    SJMP    ?C0070
?C0071:
    MOV     DataPointer, string?1778
    MOV     DataPointer+01H, string?1778+01H
    RET


_OsdPrintString:
    MOV     row?1884,R7
    MOV     col?1885,R5
    MOV     temp?1889,string ?1887
    MOV     temp?1889+01H, string?1887+01H
    MOV     j?1888,#02H
    MOV     A,row?1884
    ORL     A,#0A0H
    MOV     R0,#TrnBuf
    MOV     @R0,A
    MOV     A,col?1885
    ORL     A,#040H
    INC     R0
    MOV     @R0,A
?C0076:
    MOV     DPL,string?1887+01H
    MOV     DPH,string?1887
    CLR     A
```

```
    MOVC    A,@A+DPTR
    CPL     A
    JZ      ?C0077
    MOV     R7,j?1888
    INC     j?1888
    MOV     A,#TrmBuf
    ADD     A,R7
    MOV     R0,A
    MOV     A,R3
    MOV     @R0,A
    INC     string?1887+01H
    MOV     A,string?1887+01H
    JNZ     ?C0144
    INC     string?1887
?C0144:
    SJMP    ?C0076
?C0077:
    MOV     R0,#SlvAdr
    MOV     @R0,#07AH
    MOV     R0,#ByteCnt
    MOV     @R0,j?1888
    LCALL   SendData
    MOV     string?1887,temp ?1889
    MOV     string?1887+01H, temp?1889+01H
    MOV     j?1888,#02H
    MOV     A,row?1884
    ORL     A,#080H
    MOV     R0,#TrmBuf
    MOV     @R0,A
    MOV     A,col?1885
    ORL     A,#040H
    INC     R0
    MOV     @R0,A
?C0078:
    MOV     DPL,string?1887+01H
    MOV     DPH,string?1887
    CLR     A
    MOVC    A,@A+DPTR
    MOV     R7,A
    CPL     A
    JZ      ?C0079
    MOV     R6,j?1888
    INC     j?1888
    MOV     A,#TrmBuf
    ADD     A,R6
    MOV     R0,A
    MOV     A,R7
    MOV     @R0,A
    INC     string?1887+01H
    MOV     A,string?1887+01H
    JNZ     ?C0145
    INC     string?1887
?C0145:
    SJMP    ?C0078
?C0079:
    MOV     R0,#SlvAdr
    MOV     @R0,#07AH
    MOV     R0,#ByteCnt
    MOV     @R0,j?1888
    LCALL   SendData
    RET


_OsdPrintString1:
    MOV     row?1990,R7
    MOV     col?1991,R5
    MOV     color?1992,R3
    MOV     temp?1994,string ?1993
    MOV     temp?1994+01H, string?1993+01H
    LCALL   OsdStart
    MOV     R7,#07AH
    LCALL   _OsdSentByte
    MOV     A,row?1990
    ORL     A,#0A0H
    MOV     R7,A
    LCALL   _OsdSentByte
    MOV     A,col?1991
    ORL     A,#040H
    MOV     R7,A
    LCALL   _OsdSentByte
?C0081:
    MOV     DPL,string?1993+01H
    MOV     DPH,string?1993
    CLR     A
    MOVC    A,@A+DPTR
    CPL     A
    JZ      ?C0082
    MOV     R7,color?1992
    LCALL   _OsdSentByte
    INC     string?1993+01H
    MOV     A,string?1993+01H
    JNZ     ?C0146
    INC     string?1993
?C0146:
    SJMP    ?C0081
?C0082:
    LCALL   OsdStop
    MOV     string?1993, temp?1994
    MOV     string?1993+01H, temp?1994+01H
    LCALL   OsdStart
    MOV     R7,#07AH
    LCALL   _OsdSentByte
    MOV     A,row?1990
    ORL     A,#080H
    MOV     R7,A
    LCALL   _OsdSentByte
    MOV     A,col?1991
    ORL     A,#040H
```

```
    MOV     R7,A
    LCALL   _OsdSentByte
?C0083:
    MOV     DPL,string?1993+01H
    MOV     DPH,string?1993
    CLR     A
    MOVC    A,@A+DPTR
    MOV     R7,A
    CPL     A
    JZ      ?C0084
    LCALL   _OsdSentByte
    INC     string?1993+01H
    MOV     A,string?1993+01H
    JNZ     ?C0147
    INC     string?1993
?C0147:
    SJMP    ?C0083
?C0084:
    LCALL   OsdStop
    MOV     R7,#0AH
    MOV     R6,#00H
    LCALL   _DelayXms
    RET

_OsdDisableWindow1:
    MOV     sub_window?22100,R7
    CLR     A
    MOV     column_start?22104,A
    MOV     column_end?22105,A
    MOV     attribute?22106,A
    MOV     temp?22101,A
    MOV     A,temp?22101
    SWAP    A
    ANL     A,#0F0H
    MOV     temp?22101,A
    CLR     A
    MOV     R7,#08FH
    MOV     A,sub_window?22100
    DEC     A
    MOV     B,#03H
    MUL     AB
    MOV     R5,A
    MOV     R3,temp?22101
    LCALL   _OsdFormatA
    MOV     temp?22101,column_start?22104
    MOV     A,temp?22101
    RLC     A
    RLC     A
    RLC     A
    ANL     A,#0F8H
    MOV     temp?22101,A
    MOV     temp?22101,#00H
    MOV     R7,#08FH
    MOV     A,sub_window?22100
    MOV     B,#03H
    MUL     AB
    ADD     A,#0FEH
    MOV     R5,A
    MOV     R3,temp?22101
    LCALL   _OsdFormatA
    MOV     temp?22101,column_end?22105
    MOV     A,temp?22101
    RLC     A
    RLC     A
    RLC     A
    ANL     A,#0F8H
    MOV     temp?22101,A
    MOV     A,attribute?22106
    ORL     temp?22101,A
    MOV     R7,#08FH
    MOV     A,sub_window?22100
    MOV     B,#03H
    MUL     AB
    DEC     A
    MOV     R5,A
    MOV     R3,temp?22101
    LCALL   _OsdFormatA
    RET

_OsdDisableWindow2:
    XCH     A,R6
    MOV     A,R7
    XCH     A,R6
    MOV     R7,#08FH
    MOV     A,R6
    MOV     B,#03H
    MUL     AB
    ADD     A,#0FDH
    MOV     R5,A
    CLR     A
    MOV     R3,A
    LCALL   _OsdFormatA
    RET

_OsdSetWindow:
    MOV     sub_window?24108,R7
    MOV     temp?24114,R5
    MOV     A,temp?24114
    SWAP    A
    ANL     A,#0F0H
    MOV     temp?24114,A
    MOV     A,R3
    ORL     temp?24114,A
    MOV     R7,#08FH
    MOV     A,sub_window?24108
```

```
    DEC     A
    MOV     B,#03H
    MUL     AB
    MOV     R5,A
    MOV     R3,temp?24114
    LCALL   _OsdFormatA
    MOV     temp?24114.column _start?24111
    MOV     A,temp?24114
    RLC     A
    RLC     A
    RLC     A
    ANL     A,#0F8H
    MOV     temp?24114,A
    ORL     temp?24114,#04H
    MOV     R7,#08FH
    MOV     A,sub_window?24108
    MOV     B,#03H
    MUL     AB
    ADD     A,#0FEH
    MOV     R5,A
    MOV     R3,temp?24114
    LCALL   _OsdFormatA
    MOV     temp?24114.column _end?24112
    MOV     A,temp?24114
    RLC     A
    RLC     A
    RLC     A
    ANL     A,#0F8H
    MOV     temp?24114,A
    MOV     A,attribute?24113
    ORL     temp?24114,A
    MOV     R7,#08FH
    MOV     A,sub_window?24108
    MOV     B,#03H
    MUL     AB
    DEC     A
    MOV     R5,A
    MOV     R3,temp?24114
    LCALL   _OsdFormatA
    RET

_OsdBarHandle:
    MOV     row?25115,R7
    MOV     col?25116,R5
    MOV     color?25117,R3
    MOV     BarMaxCount?25121 ,#0AH
    MOV     R5,row?25115
    CLR     A
    MOV     R3,A
    LCALL   _OsdClearRow
    MOV     A,row?25115
    ORL     A,#0A0H
    MOV     R0,#TrmBuf
    MOV     @R0,A
    MOV     A,col?25116
    ORL     A,#040H
    INC     R0
    MOV     @R0,A
    CLR     A
    MOV     j?25119,A
?C0097:
    MOV     A,BarMaxCount?25121
    ADD     A,#02H
    MOV     R7,A
    CLR     A
    RLC     A
    MOV     R6,A
    CLR     C
    MOV     A,j?25119
    SUBB    A,R7
    MOV     A,R6
    XRL     A,#080H
    MOV     R0,A
    MOV     A,#080H
    SUBB    A,R0
    JNC     ?C0098
    MOV     A,#TrmBuf+02H
    ADD     A,j?25119
    MOV     R0,A
    MOV     @R0,color?25117
    INC     j?25119
    SJMP    ?C0097
?C0098:
    MOV     R0,#SlvAdr
    MOV     @R0,#07AH
    MOV     A,BarMaxCount?25121
    ADD     A,#04H
    MOV     R0,#ByteCnt
    MOV     @R0,A
    LCALL   SendData
    MOV     A,BarMaxCount?25121
    MOV     B,#06H
    MUL     AB
    MOV     R7,A
    MOV     R0,#OSDMinValue
    MOV     A,@R0
    MOV     R6,A
    CLR     C
    MOV     R0,#CurrentValue
    MOV     A,@R0
    SUBB    A,R6
    MOV     B,R7
    MUL     AB
    MOV     R7,A
    CLR     C
    MOV     R0,#OSDMaxValue
    MOV     A,@R0
```

```
    SUBB    A,R6
    MOV     R6,A
    MOV     A,R7
    MOV     B,R6
    DIV     AB
    MOV     R7,A
    MOV     R0,#UpdateValue
    MOV     @R0,A
    MOV     A,row?25115
    ORL     A,#080H
    MOV     R0,#TrmBuf
    MOV     @R0,A
    MOV     A,col?25116
    ORL     A,#040H
    INC     R0
    MOV     @R0,A
    INC     R0
    MOV     @R0,#06FH
    MOV     A,R7
    MOV     B,#06H
    DIV     AB
    MOV     j?25119,A
    MOV     R7,#03H
?C0100:
    MOV     A,j?25119
    ADD     A,#03H
    MOV     R5,A
    CLR     A
    RLC     A
    MOV     R4,A
    MOV     A,R7
    CLR     C
    SUBB    A,R5
    MOV     A,R4
    XRL     A,#080H
    MOV     R0,A
    MOV     A,#080H
    SUBB    A,R0
    JNC     ?C0101
    MOV     DPTR,#OSDM_BAR_H+06H
    CLR     A
    MOVC    A,@A+DPTR
    MOV     R6,A
    MOV     A,#TrmBuf
    ADD     A,R7
    MOV     R0,A
    MOV     A,R6
    MOV     @R0,A
    INC     R7
    SJMP    ?C0100
?C0101:
    MOV     R0,#UpdateValue
    MOV     A,@R0
    MOV     B,#06H
    DIV     AB
    MOV     j?25119,B
    MOV     A,j?25119
    MOV     DPTR,#OSDM_BAR_H
    MOVC    A,@A+DPTR
    MOV     R6,A
    XCH     A,R5
    MOV     A,R7
    XCH     A,R5
    INC     R7
    MOV     A,#TrmBuf
    ADD     A,R5
    MOV     R0,A
    MOV     A,R6
    MOV     @R0,A
?C0103:
    MOV     A,BarMaxCount?25121
    ADD     A,#03H
    MOV     R5,A
    CLR     A
    RLC     A
    MOV     R4,A
    MOV     A,R7
    CLR     C
    SUBB    A,R5
    MOV     A,R4
    XRL     A,#080H
    MOV     R0,A
    MOV     A,#080H
    SUBB    A,R0
    JNC     ?C0104
    MOV     DPTR,#OSDM_BAR_H
    CLR     A
    MOVC    A,@A+DPTR
    MOV     R6,A
    MOV     A,#TrmBuf
    ADD     A,R7
    MOV     R0,A
    MOV     A,R6
    MOV     @R0,A
    INC     R7
    SJMP    ?C0103
?C0104:
    MOV     A,#TrmBuf+03H
    ADD     A,BarMaxCount ?25121
    MOV     R0,A
    MOV     @R0,#077H
    MOV     R0,#SlvAdr
    MOV     @R0,#07AH
    MOV     A,BarMaxCount ?25121
    ADD     A,#04H
    MOV     R0,#ByteCnt
    MOV     @R0,A
    LCALL   SendData
```

```
    RET

_OsdBarHandle1:
    MOV     row?26122,R7
    MOV     col?26123,R5
    MOV     color?26124,R3
    MOV     BarMaxCount?26128 ,#0AH
    MOV     R5,row?26122
    CLR     A
    MOV     R3,A
    LCALL   _OsdClearRow
    LCALL   OsdStart
    MOV     R7,#07AH
    LCALL   _OsdSentByte
    MOV     A,row?26122
    ORL     A,#0A0H
    MOV     R7,A
    LCALL   _OsdSentByte
    MOV     A,col?26123
    ORL     A,#040H
    MOV     R7,A
    LCALL   _OsdSentByte
    CLR     A
    MOV     j?26126,A
?C0107:
    MOV     A,BarMaxCount ?26128
    ADD     A,#02H
    MOV     R7,A
    CLR     A
    RLC     A
    MOV     R6,A
    CLR     C
    MOV     A,j?26126
    SUBB    A,R7
    MOV     A,R6
    XRL     A,#080H
    MOV     R0,A
    MOV     A,#080H
    SUBB    A,R0
    JNC     ?C0108
    MOV     R7,color?26124
    LCALL   _OsdSentByte
    INC     j?26126
    SJMP    ?C0107
?C0108:
    LCALL   OsdStop
    MOV     A,BarMaxCount ?26128
    MOV     B,#06H
    MUL     AB
    MOV     R7,A
    MOV     R0,#OSDMinValue
    MOV     A,@R0
    MOV     R6,A
    CLR     C
    MOV     R0,#CurrentValue
    MOV     A,@R0
    SUBB    A,R6
    MOV     B,R7
    MUL     AB
    MOV     R7,A
    CLR     C
    MOV     R0,#OSDMaxValue
    MOV     A,@R0
    SUBB    A,R6
    MOV     R6,A
    MOV     A,R7
    MOV     B,R6
    DIV     AB
    MOV     R7,A
    MOV     R0,#UpdateValue
    MOV     @R0,A
    MOV     A,row?26122
    ORL     A,#080H
    MOV     R0,#TrmBuf
    MOV     @R0,A
    MOV     A,col?26123
    ORL     A,#040H
    INC     R0
    MOV     @R0,A
    INC     R0
    MOV     @R0,#06FH
    MOV     A,R7
    MOV     B,#06H
    DIV     AB
    MOV     j?26126,A
    MOV     R7,#03H
?C0110:
    MOV     A,j?26126
    ADD     A,#03H
    MOV     R5,A
    CLR     A
    RLC     A
    MOV     R4,A
    MOV     A,R7
    CLR     C
    SUBB    A,R5
    MOV     A,R4
    XRL     A,#080H
    MOV     R0,A
    MOV     A,#080H
    SUBB    A,R0
    JNC     ?C0111
    MOV     DPTR,#OSDM_BAR_H+06H
    CLR     A
    MOVC    A,@A+DPTR
    MOV     R6,A
    MOV     A,#TrmBuf
```

```
    ADD     A,R7
    MOV     R0,A
    MOV     A,R6
    MOV     @R0,A
    INC     R7
    SJMP    ?C0110
?C0111:
    MOV     R0,#UpdateValue
    MOV     A,@R0
    MOV     B,#06H
    DIV     AB
    MOV     j?26126,B
    MOV     A,j?26126
    MOV     DPTR,#OSDM_BAR_H
    MOVC    A,@A+DPTR
    MOV     R6,A
    XCH     A,R5
    MOV     A,R7
    XCH     A,R5
    INC     R7
    MOV     A,#TrmBuf
    ADD     A,R5
    MOV     R0,A
    MOV     A,R6
    MOV     @R0,A
?C0113:
    MOV     A,BarMaxCount ?26128
    ADD     A,#03H
    MOV     R5,A
    CLR     A
    RLC     A
    MOV     R4,A
    MOV     A,R7
    CLR     C
    SUBB    A,R5
    MOV     A,R4
    XRL     A,#080H
    MOV     R0,A
    MOV     A,#080H
    SUBB    A,R0
    JNC     ?C0114
    MOV     DPTR,#OSDM_BAR_H
    CLR     A
    MOVC    A,@A+DPTR
    MOV     R6,A
    MOV     A,#TrmBuf
    ADD     A,R7
    MOV     R0,A
    MOV     A,R6
    MOV     @R0,A
    INC     R7
    SJMP    ?C0113
?C0114:
    MOV     A,#TrmBuf+03H
    ADD     A,BarMaxCount ?26128
    MOV     R0,A
    MOV     @R0,#077H
    MOV     R0,#SlvAdr
    MOV     @R0,#07AH
    MOV     A,BarMaxCount ?26128
    ADD     A,#04H
    MOV     R0,#ByteCnt
    MOV     @R0,A
    LCALL   SendData
    RET

_OsdDisplayValue:
    MOV     row?27129,R7
    MOV     col?27130,R5
    MOV     A,row?27129
    ORL     A,#0A0H
    MOV     R0,#TrmBuf
    MOV     @R0,A
    MOV     A,col?27130
    ORL     A,#040H
    INC     R0
    MOV     @R0,A
    CLR     A
    MOV     i?27132,A
?C0117:
    MOV     A,#TrmBuf+02H
    ADD     A,i?27132
    MOV     R0,A
    MOV     A,R3
    MOV     @R0,A
    INC     i?27132
    MOV     A,i?27132
    CJNE    A,#04H,?C0117
?C0118:
    MOV     R0,#SlvAdr
    MOV     @R0,#07AH
    MOV     R0,#ByteCnt
    MOV     @R0,#07H
    LCALL   SendData
    MOV     R0,#OSDMinValue
    MOV     A,@R0
    MOV     R7,A
    CLR     C
    MOV     R0,#CurrentValue
    MOV     A,@R0
    SUBB    A,R7
    MOV     B,#064H
    MUL     AB
    MOV     R6,A
    CLR     C
    MOV     R0,#OSDMaxValue
    MOV     A,@R0
```

```
    SUBB    A,R7
    MOV     R7,A
    MOV     A,R6
    MOV     B,R7
    DIV     AB
    MOV     R0,#CurrentValue
    MOV     @R0,A
    CLR     A
    MOV     i?27132,A
?C0120:
    MOV     R0,#CurrentValue
    MOV     A,@R0
    MOV     R7,A
    MOV     B,#0AH
    DIV     AB
    MOV     R0,#UpdateValue
    MOV     A,B
    MOV     @R0,A
    MOV     A,R7
    MOV     B,#0AH
    DIV     AB
    MOV     R0,#CurrentValue
    MOV     @R0,A
    MOV     R0,#UpdateValue
    MOV     A,@R0
    MOV     DPTR,#OSDM_DEC_TABLE
    MOVC    A,@A+DPTR
    MOV     R7,A
    MOV     A,#TrmBuf+0AH
    ADD     A,i?27132
    MOV     R0,A
    MOV     A,R7
    MOV     @R0,A
    INC     i?27132
    MOV     A,i?27132
    CJNE    A,#04H,?C0120
?C0121:
    MOV     R0,#UpdateValue
    MOV     @R0,#04H
    SETB    sign?27133
    CLR     A
    MOV     i?27132,A
?C0123:
    JNB     sign?27133,?C0126
    CLR     C
    MOV     A,#0DH
    SUBB    A,i?27132
    ADD     A,#TrmBuf
    MOV     R0,A
    MOV     A,@R0
    MOV     R7,A
    XRL     A,#04FH
    JZ      ?C0125
    CLR     sign?27133
    MOV     R0,#UpdateValue
    MOV     A,@R0
    INC     @R0
    ADD     A,#TrmBuf
    MOV     R0,A
    MOV     A,R7
    MOV     @R0,A
    SJMP    ?C0125
?C0126:
    CLR     C
    MOV     A,#0DH
    SUBB    A,i?27132
    ADD     A,#TrmBuf
    MOV     R0,A
    MOV     A,@R0
    MOV     R7,A
    MOV     R0,#UpdateValue
    MOV     A,@R0
    INC     @R0
    ADD     A,#TrmBuf
    MOV     R0,A
    MOV     A,R7
    MOV     @R0,A
?C0125:
    INC     i?27132
    MOV     A,i?27132
    CJNE    A,#04H,?C0123
?C0124:
    JNB     sign?27133,?C0129
    MOV     R0,#UpdateValue
    MOV     A,@R0
    INC     @R0
    ADD     A,#TrmBuf
    MOV     R0,A
    MOV     @R0,#04FH
?C0129:
    MOV     R0,#UpdateValue
    MOV     A,#0FCH
    ADD     A,@R0
    MOV     @R0,A
    MOV     R7,A
    ADD     A,col?27130
    MOV     R0,#NextColumn
    MOV     @R0,A
    MOV     R0,#SlvAdr
    MOV     @R0,#07AH
    MOV     A,R7
    ADD     A,#02H
    MOV     R0,#ByteCnt
    MOV     @R0,A
    MOV     A,row?27129
    ORL     A,#080H
    MOV     R0,#TrmBuf
    MOV     @R0,A
```

```
    MOV     A,col?27130
    ORL     A,#040H
    INC     R0
    MOV     @R0,A
    MOV     i?27132,#01H
?C0130:
    MOV     A,i?27132
    SETB    C
    MOV     R0,#UpdateValue
    SUBB    A,@R0
    JNC     ?C0131
    MOV     A,#TrmBuf+03H
    ADD     A,i?27132
    MOV     R0,A
    MOV     A,@R0
    MOV     R7,A
    MOV     A,#TrmBuf+01H
    ADD     A,i?27132
    MOV     R0,A
    MOV     A,R7
    MOV     @R0,A
    INC     i?27132
    SJMP    ?C0130
?C0131:
    LCALL   SendData
    RET


_TEST_VALUE:
    MOV     R0,#CurrentValue
    MOV     A,R7
    MOV     @R0,A
    MOV     R7,#01H
    MOV     R5,#0CH
    MOV     R3,#03H
    LCALL   _OsdDisplayValue
    MOV     R7,#01H
    MOV     R0,#NextColumn
    MOV     A,@R0
    INC     @R0
    MOV     R5,A
    MOV     R3,#053H
    MOV     ?_OsdPrintIcon?BYTE+03H,#01H
    LCALL   _OsdPrintIcon
    MOV     R7,#01H
    MOV     R0,#NextColumn
    MOV     A,@R0
    INC     @R0
    MOV     R5,A
    MOV     R3,#045H
    MOV     ?_OsdPrintIcon?BYTE+03H,#01H
    LCALL   _OsdPrintIcon
    MOV     R7,#01H
    MOV     R0,#NextColumn
    MOV     A,@R0
    INC     @R0
    MOV     R5,A
    MOV     R3,#043H
    MOV     ?_OsdPrintIcon ?BYTE+03H,#01H
    LCALL   _OsdPrintIcon
    RET


_OsdDisplayCount:
    MOV     count?29135,R7
    SETB    ?OsdEnable?BIT
    LCALL   OsdEnable
    LCALL   OsdNormal
    MOV     R7,#040H
    MOV     R5,#050H
    LCALL   _OsdLocationSet
    MOV     R7,#01H
    CLR     A
    MOV     R5,A
    MOV     R3,#02H
    MOV     ?_OsdSetWindow ?BYTE+03H,A
    MOV     ?_OsdSetWindow ?BYTE+04H,#0EH
    MOV     ?_OsdSetWindow ?BYTE+05H,#01H
    LCALL   _OsdSetWindow
    MOV     R6,#HIGH (OSDCOUNT_DISPLAY)
    MOV     R7,#LOW (OSDCOUNT_DISPLAY)
    MOV     R5,#01H
    CLR     A
    MOV     R3,A
    SETB    ?_OsdStringAdr ?BIT
    LCALL   _OsdStringAdr
    MOV     R7,#01H
    MOV     R5,#01H
    MOV     R3,#04H
    MOV     OsdPrintString? BYTE+03H,DataPointer
    MOV
    OsdPrintString ?BYTE+04H,DataPointer+01H
    LCALL   _OsdPrintString
    MOV     R0,#CurrentValue
    MOV     @R0,count?29135
    MOV     R7,#01H
    MOV     R5,#0BH
    MOV     R3,#07H
    LCALL   _OsdDisplayValue
    RET


OSDM_BAR_H:
    DB      076H,075H,074H,073H,072H
    DB      071H,070H


OSDM_DEC_TABLE:
```

```
    DB      04FH,031H,032H,033H,034H
    DB      035H,036H,037H,038H,039H

OSDCOUNT_DISPLAY:
    DB      045H,044H,04FH,047H,000H ,043H
    DB      04FH,055H,04EH,054H,03AH ,0FFH
    DB      046H,044H,04FH,047H,000H ,043H
    DB      04FH,055H,04EH,054H,03AH ,0FFH
    DB      047H,044H,04FH,047H,000H ,043H
    DB      04FH,055H,04EH,054H,03AH ,0FFH
    DB      049H,044H,04FH,047H,000H ,043H
    DB      04FH,055H,04EH,054H,03AH ,0FFH
    DB      053H,044H,04FH,047H,000H ,043H
    DB      04FH,055H,04EH,054H,03AH ,0FFH
```

[General Information]
书名=8051单片机C语言彻底应用
作者=
页数=499
SS号=10441508
出版日期=

Powered by XiaoGuo’s publishing Studio

QQ:8204136

Website: www.mcuzone.com

2005