单片机基础知识

单片机的外部结构：

1、 DIP40 双列直插；

2、 P0，P1，P2，P3 四个 8 位准双向 I/O 引脚；（作为 I/O 输入时，要先输出高电平）

3、 电源 VCC（PIN40）和地线 GND（PIN20）；

4、 高电平复位 RESET（PIN9）；（10uF 电容接 VCC 与 RESET，即可实现上电复位）

5、 内置振荡电路，外部只要接晶体至 X1（PIN18）和 X0（PIN19）；（频率为主频的 12 倍）

6、 程序配置 EA（PIN31）接高电平 VCC；（运行单片机内部 ROM 中的程序）

7、 P3 支持第二功能：RXD、TXD、INT0、INT1、T0、T1

单片机内部 I/O 部件：(所为学习单片机，实际上就是编程控制以下 I/O 部件，完成指定任务)

1、 四个 8 位通用 I/O 端口，对应引脚 P0、P1、P2 和 P3；

2、 两个 16 位定时计数器；（TMOD，TCON，TL0，TH0，TL1，TH1）

3、 一个串行通信接口；（SCON，SBUF）

4、 一个中断控制器；（IE，IP）

针对 AT89C52 单片机，头文件 AT89x52.h 给出了 SFR 特殊功能寄存器所有端口的定义。教科书的 160 页给出了针对 MCS51 系列单片机的 C 语言扩展变量类型。

C 语言编程基础：

1、 十六进制表示字节 0x5a：二进制为 01011010B；0x6E 为 01101110。

2、 如果将一个 16 位二进数赋给一个 8 位的字节变量，则自动截断为低 8 位，而丢掉高 8 位。

3、 ++var 表示对变量 var 先增一；var—表示对变量后减一。

4、 x |= 0x0f;表示为 x = x | 0x0f;

5、 TMOD = (TMOD & 0xf0) | 0x05;表示给变量 TMOD 的低四位赋值 0x5，而不改变 TMOD 的高四位。

6、 While(1); 表示无限执行该语句，即死循环。语句后的分号表示空循环体，也就是{;}

第一章　　单片机最小应用系统：

单片机最小系统的硬件原理接线图：

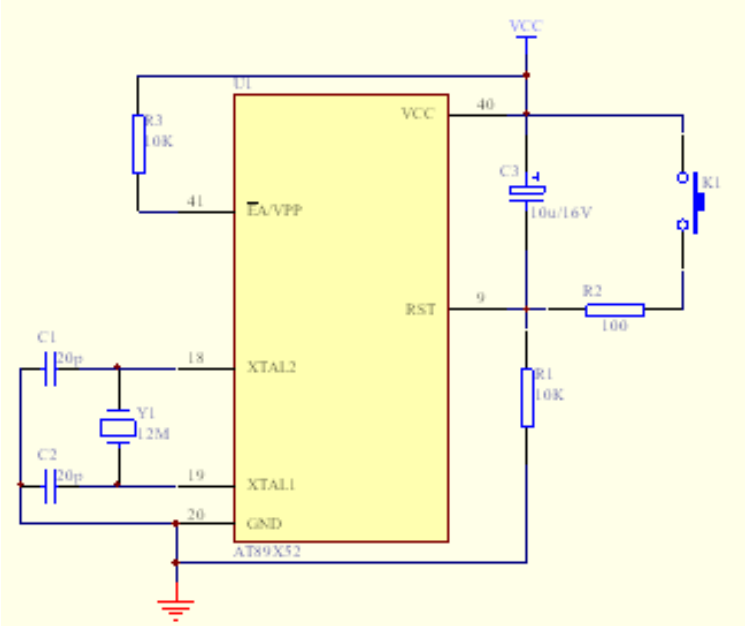
1、 接电源：VCC（PIN40）、GND（PIN20）。加接退耦电容 0.1uF

2、 接晶体：X1（PIN18）、X2（PIN19）。注意标出晶体频率（选用 12MHz），还有辅助电容 30pF

3、 接复位：RES（PIN9）。接上电复位电路，以及手动复位电路，分析复位工作原理

4、 接配置：EA（PIN31）。说明原因。

具体接法如下图所示：



第二章　　基本 I/O 口的应用。

例 1：用 P1 口输出一倍频方波。

```
#include <reg52.h>      //reg52.h 为包含 51 资源的库文件
void main ( void )
{
    while (1==1)
    {
        ++P1;          //使 P1 口加一完成一倍频方波，
    }
}
```

注意：*P0 的每个引脚要输出高电平时，必须外接上拉电阻（如 4K7）至 VCC 电源。*

例 2：用 P1 口输出一倍频方波,要求能用万用表测出方波。

其实，只需要在上面的程序中添加延时程序即可。

```
#include <reg52.h>
void main ( void )
{
    unsigned int i,j;
    while (1==1)
    {
        ++P1;
        for (i=0;i<1000;i++)
        for(j=0;j<1000;j++);  //该循环是一个大概的延时，具体时间要看汇编语言的指令才能判断。
    }
}
```

例 3：要求从 P1 口输出一方波，要求 P1.7 变化的最快，P1.0 变化的最慢。

```
#include <reg52.h>
void main ( void )
```

```
{
    unsigned char m,n;          //定义两个中间变量完成交换过程
    unsigned int i,j;
    while (1)
    {
        n = 0;
        ++m;
        n|=(m<<7)&0x80;         //将第 0 位的值送至第 7 位
        n|=(m<<5)&0x40;         //将第 1 位的值送至第 6 位
        n|=(m<<3)&0x20;         //将第 2 位的值送至第 5 位
        n|=(m<<1)&0x10;         //将第 3 位的值送至第 4 位
        n|=(m>>1)&0x08;         //将第 4 位的值送至第 3 位
        n|=(m>>3)&0x04;         //将第 5 位的值送至第 2 位
        n|=(m>>5)&0x02;         //将第 6 位的值送至第 1 位
        n|=(m>>7)&0x01;         //将第 7 位的值送至第 0 位
        P1 = n;
        for(i=0;i<1000;i++)
            for(j=0;j<1000;j++);
    }
}
```

注意：*一个字节的 8 位 D7、D6 至 D0，分别输出到 P3.7、P3.6 至 P3.0，比如 P3=0x0f，则 P3.7、P3.6、P3.5、P3.4 四个引脚都输出低电平，而 P3.3、P3.2、P3.1、P3.0 四个引脚都输出高电平。同样，输入一个端口 P2，即是将 P2.7、P2.6 至 P2.0，读入到一个字节的 8 位 D7、D6 至 D0。*

第三章　　显示驱动

数码管的接法和驱动原理

一支七段数码管实际由 8 个发光二极管构成，其中 7 个组形构成数字 8 的七段笔画，所以称为七段数码管，而余下的 1 个发光二极管作为小数点。作为习惯，分别给 8 个发光二极管标上记号：a,b,c,d,e,f,g,h。对应 8 的顶上一画，按顺时针方向排，中间一画为 g，小数点为 h。

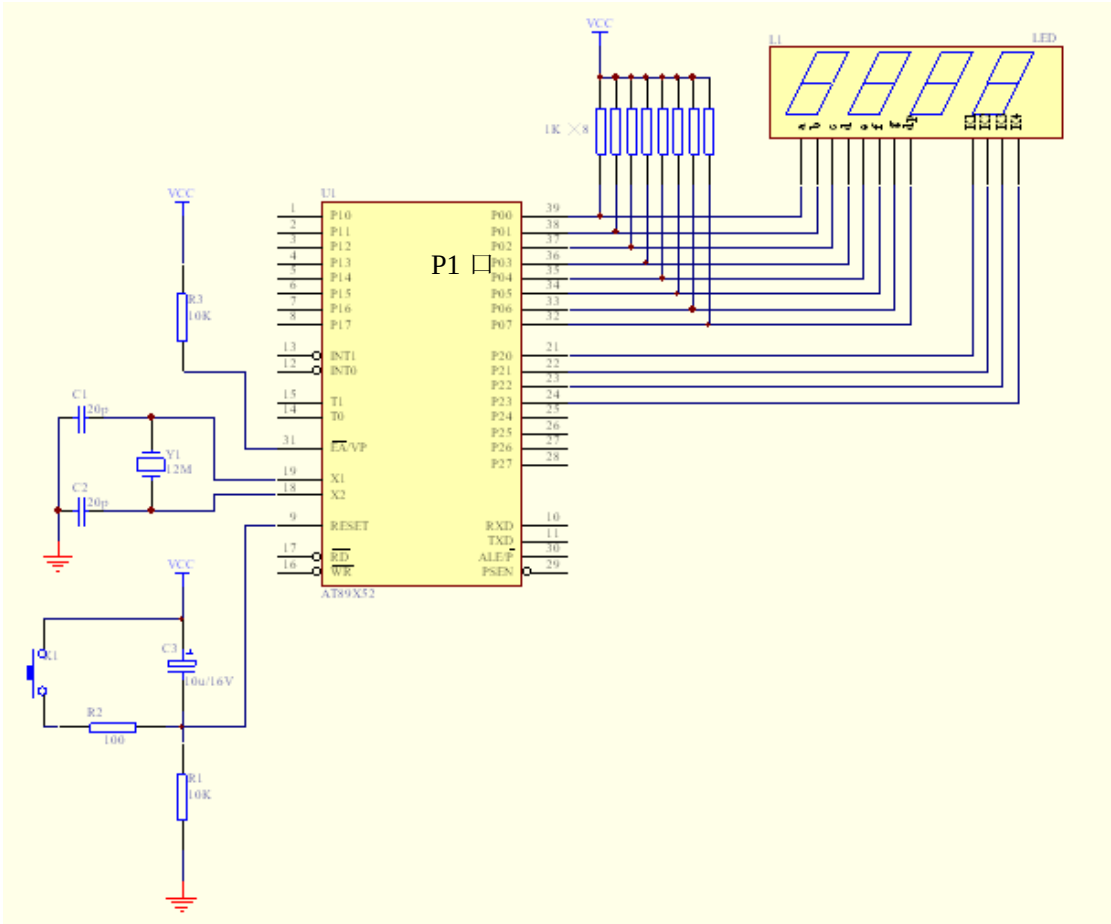
我们通常又将各二极与一个字节的 8 位对应，a(D0),b(D1),c(D2),d(D3),e(D4),f(D5),g(D6),h(D7)，相应 8 个发光二极管正好与单片机一个端口 Pn 的 8 个引脚连接，这样单片机就可以通过引脚输出高低电平控制 8 个发光二极的亮与灭，从而显示各种数字和符号；对应字节，引脚接法为：a(Pn.0)，b(Pn.1)，c(Pn.2)，d(Pn.3)，e(Pn.4)，f(Pn.5)，g(Pn.6)，h(Pn.7)。

如果将 8 个发光二极管的负极（阴极）内接在一起，作为数码管的一个引脚，这种数码管则被称为共阴数码管，共同的引脚则称为共阴极，8 个正极则为段极。否则，如果是将正极（阳极）内接在一起引出的，则称为共阳数码管，共同的引脚则称为共阳极，8 个负极则为段极。

以单支共阴数码管为例，可将段极接到某端口 Pn，共阴极接 GND，则可编写出对应十六进制码的七段码表字节数据如下图：

显示字符	共阴极段选码	共阳极段选码	显示字符	共阴极段选码	共阳极段选码
0	3FH	C0H	C	39H	C6H
1	06H	F9H	D	5EH	A1H
2	5BH	A4H	E	79H	86H
3	4FH	B0H	F	71H	84H
4	66H	99H	P	73H	82H
5	6DH	92H	U	3EH	C1H
6	7DH	82H	r	31H	CEH
7	07H	F8H	y	6EH	91H
8	7FH	80H	8	FFH	00H
9	6FH	90H	“灭”	00H	FFH
A	77H	88H			⋮
B	7CH	83H			

动态显示的电路连接如下图所示：



下面，我们编程在数码管上显示出“1 2 3 4”。程序如下：

```
#include <reg52.h>

Code unsigned char Seg7Code[16]=      //用十六进数作为数组下标，可直接取得对应的七段编码字节
```

```
// 0    1    2    3    4    5    6    7    8    9    A    b    C    d    E    F
{0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71};

void main ( void )
{
    unsigned int i;
    while (1)
    {
        P2 |= 0x0f;              //消隐，让数码管开始处于不亮的状态
        P0 = LedCode[1];         //将“1”的代码送出
        P2 &= 0xfe;              //选中第一个数码管
        for（i=0;i<1000;i++）;
        P2 |= 0x0f;
        P0 = LedCode[2];
        P2 &= 0xfd;
        for（i=0;i<1000;i++）;
        P2 |= 0x0f;
        P0 = LedCode[3];
        P2 &= 0xfb;
        for（i=0;i<1000;i++）;
        P2 |= 0x0f;
        P0 = LedCode[4];
        P2 &= 0xf7;
        for（i=0;i<1000;i++）;
    }
}
```

关于 DRIVER

编写 DRIVER 的目的是让程序能适应更多的场合，让我们的使用更加方便，大家可以把一些自己编过的有用的程序做成 DRIVER 便于自己以后的使用。

下面介绍显示的驱动程序：

首先，定义一个头文档 <LedDriver.H>，描述可用函数，如下：

```
#ifndef _ LedDriver_H _          //防止重复引用该文档，如果没有定义过符号 _KEY_H_，则编译下面语句
#define _ LedDriver_H _          //只要引用过一次，即 #include <key.h>，则定义符号 _KEY_H_
void LedPrint ( unsigned char Dat ) //数据缓冲区间，完成移位功能
void LedWork ( void )            //送数到显示数码管
#endif
```

然后，定义函数体文档 LedDriver.C，如下：

```
#include <reg52.h>
#include “LedDriver.h”
```

```
Code unsigned char LedCode[16]=        //Code 是表示这个数组的存储空间
{0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71};
unsigned char DisBuf[4];
void LedPrint (unsigned char Dat)
{
    DisBuf[0] = DisBuf[1];    //每次用后一个数冲掉前一个数，便于扩展显示位数
    DisBuf[1] = DisBuf[2];
    DisBuf[2] = DisBuf[3];
    DisBuf[3] = Dat;
}

void LedWork ( void )
{
    static unsigned char i = 0;      //static 表示静态变量，指变量的赋值只在第一次定义的时候赋
    P2 |= 0x0f;
    P0 = LedCode[DisBuf[i]];
    Switch( i )                      //选择数据送到哪个管子
    {
        case 0: P2_0 = 0; break;
        case 1: P2_1 = 0; break;
        case 2: P2_2 = 0; break;
        case 3: P2_3 = 0; break;
    }
    if (++i>=4)      i = 0;          //判断四位数是否都已经送完
    for (m=0;m<1000;m++);            //延时
}
```

这样 DRIVER 的程序就编好了，我们以后用的时候直接调用函数就可以了。

主程序可以编写如下：

```
#include <reg52.h>
#include “LedDriver.h”
void mian ( void )
{
    LedPrint( 1 );          //调用函数，把想显示的数据送如缓存
    LedPrint( 2 );
    LedPrint( 3 );
    LedPrint( 4 );
    While( 1 )
    {
        LedWork( );
    }
}
```

下面介绍一个例子供大家参考。

显示“12345678”

P1 端口接 8 联共阴数码管 SLED8 的段极：P1.7 接段 h,…，P1.0 接段 a

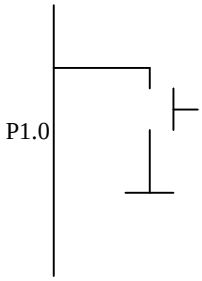
P2 端口接 8 联共阴数码管 SLED8 的段极：P2.7 接左边的共阴极，…，P2.0 接右边的共阴极

方案说明：晶振频率 fosc=12MHz，数码管采用动态刷新方式显示，在 1ms 定时断服务程序中实现

```
#include <reg52.h>
unsigned char DisBuf[8];     //全局显示缓冲区，DisBuf[0]对应右 SLED，DisBuf[7]对应左 SLED，
void DisplayBrush( void )
{       code unsigned char cathode[8]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};       //阴极控制码
        code unsigned char Seg7Code[16]=       //用十六进数作为数组下标，可直接取得对应的七段编码字节
        {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};
        static unsigned char i=0; //  （0≤i≤7）  循环刷新显示，由于是静态变量，此赋值只做一次。
        P2 = 0xff;          //显示消隐，以免下一段码值显示在前一支 SLED
        P1 = Seg7Code[ DisBuf[i] ];//从显示缓冲区取出原始数据，查表变为七段码后送出显示
        P2 = cathode[ i ];              //将对应阴极置低，显示
        if( ++i >= 8 ) i=0;             //指向下一个数码管和相应数据
}
void Timer0IntRoute( void ) interrupt 1
{
        TL0 = -1000;            //由于 TL0 只有 8bits，所以将（-1000）低 8 位赋给 TL0
        TH0 = (-1000)>>8;       //取（-1000）的高 8 位赋给 TH0，重新定时 1ms
        DisplayBrush();
}
void Timer0Init( void )
{
        TMOD=(TMOD & 0xf0) | 0x01;  //初始化，定时器 T0，工作方式 1
        TL0 = -1000;     //定时 1ms
        TH0 = (-1000)>>8;
        TR0 = 1;                //允许 T0 开始计数
        ET0 = 1;                //允许 T0 计数溢出时产生中断请求
}
void Display( unsigned char index, unsigned char dataValue )
{
         DisBuf[ index ] = dataValue;
}
void main( void )
{
        unsigned char i;
        for( i=0; i<8; i++ ){ Display(i, 8-i); } //DisBuf[0]为右，DisBuf[0]为左
        Timer0Init();
        EA = 1;                 //允许 CPU 响应中断请求
        While(1);
}
```

第四章　　键盘驱动

单片机 I/O 口作为输入的前提是必须首先输出一个高电平。




```
char Kbhit ( void )
{
 P1_0 = 1;
 if   (P1_9 = = 0 )     return ( 1 );
 else return ( 0 );
)
```

下面我们对上面的程序作个改进：

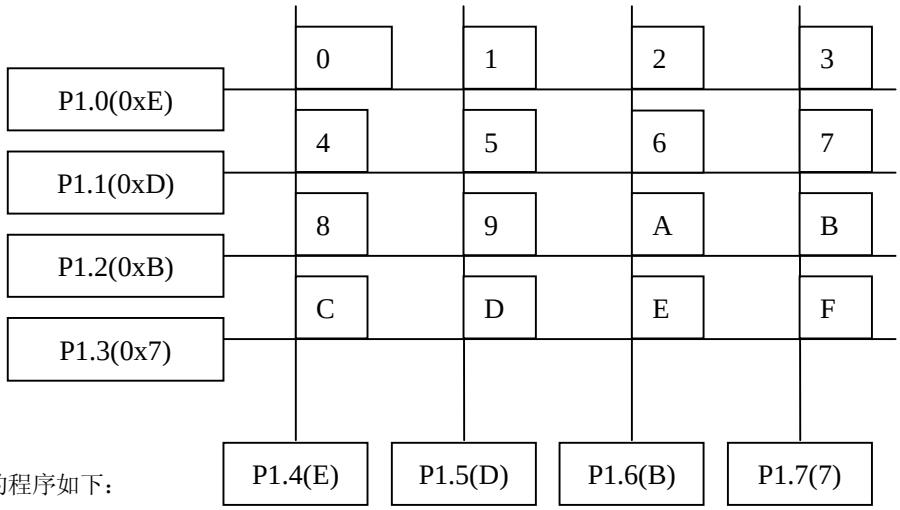
```
char Kbhit ( void )
{
    P1 = 0xff;
    if ((P1^0xff) != 0)     return ( 1 );
}
```

一般来说，按键的时候会有抖动，我们可以用加延时的办法来去除抖动。即：

```
P1 =0xff;
if ((P1^0xff )!= 0)
延时 20ms;
if ((P1^0xff) !=0)     return (1);
```

4X4 按键。

由 P1 端口的高 4 位和低 4 位构成 4X4 的矩阵键盘，本程序只认为单键操作为合法，同时按多键时无效。



取键值的程序如下：

```
unsigned char getch ( void )
{
    unsigned char X,Y,Z;
    P1 = 0xf0;
    X = P1;
    P1 = 0x0f;
```

```
    Y = P1;
    Z =X | Y;
    switch ( Z )
    {
        case 0xee: return ( 0 );
        case 0xde: return ( 1 );
        case 0xbe: return ( 2 );
        case 0x7e: return ( 3 );
        case 0xed: return ( 4 );
        case 0xdd: return ( 5 );
        case 0xbd: return ( 6 );
        case 0x7d: return ( 7 );
        case 0xeb: return ( 8 );
        case 0xdb: return ( 9 );
        case 0xbb: return ( 10 );
        case 0x7b: return ( 11 );
        case 0xe7: return ( 12 );
        case 0xd7: return ( 13 );
        case 0xb7: return ( 14 );
        case 0x77: return ( 15 );
      }
}
```

判断有无键按下的程序：

```
char Kbhit ( void )
{
    P1 = 0xf0;
    if (P1 = = 0xf0) return ( 0 );
    else return ( 1 );
}
```

下面是键盘的 Driver 程序：

首先我们还是来写 KeyDriver.h 这个程序：

```
#ifndef _KeyDriver_h_
#define _KeyDriver_h_
char Khbit ( void );
char Getch ( void );
#endif
```

接着，我们来写 KeyDriver.c 程序

```
#include <reg52.h>
#include “KeyDriver.h”

char Kbhit ( void )
{
    P1 = 0xf0;
```

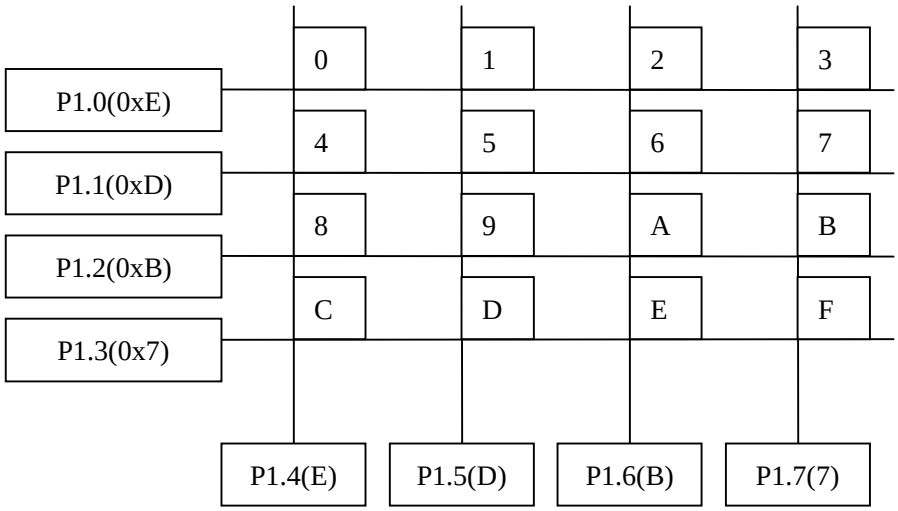
```
    if (P1 = = 0xf0) return ( 0 );
    else return ( 1 );
}

  unsigned char getch ( void )
  {
    unsigned char X,Y,Z;
    P1 = 0xf0;
    X = P1;
    P1 = 0x0f;
    Y = P1;
    Z =X | Y;
    switch ( Z )
    {
        case 0xee: return ( 0 );
        case 0xde: return ( 1 );
        case 0xbe: return ( 2 );
        case 0x7e: return ( 3 );
        case 0xed: return ( 4 );
        case 0xdd: return ( 5 );
        case 0xbd: return ( 6 );
        case 0x7d: return ( 7 );
        case 0xeb: return ( 8 );
        case 0xdb: return ( 9 );
        case 0xbb: return ( 10 );
        case 0x7b: return ( 11 );
        case 0xe7: return ( 12 );
        case 0xd7: return ( 13 );
        case 0xb7: return ( 14 );
        case 0x77: return ( 15 );
    }
}
```

按键显示程序如下：

```
#include < reg52.h >
#include “LedDriver.h”
#include “KeyDriver.h”
void main ( void )
{
    unsigned char i;
    for (i=1;i<5;i++)
    {   LedPrint ( i );}
    while ( 1 )
    {
```

```
            if (Kbhit( ))
            { LedPrint ( Getch( ));}
            LedWork ( );
        }
}
```

下面是另一个键盘值的算法，供大家参考。



定义一个头文档 <KEY.H>，描述可用函数，如下：

```
#ifndef _KEY_H_              //防止重复引用该文档，如果没有定义过符号 _KEY_H_，则编译下面语句
#define _KEY_H_              //只要引用过一次，即 #include <key.h>，则定义符号 _KEY_H_
unsigned char keyHit( void );       //如果按键，则返回非 0 ，否则返回 0
unsigned char keyGet( void );       //读取按键值，如果没有按键则等待到按键为止
void keyPut( unsigned char ucKeyVal ); //保存按键值 ucKeyVal 到按键缓冲队列末
void keyBack( unsigned char ucKeyVal );      //退回键值 ucKeyVal 到按键缓冲队列首
#endif
```

定义函数体文档 KEY.C，如下：

```
＃include “key.h”
#define KeyBufSize    16    //定义按键缓冲队列字节数
unsigned char KeyBuf[ KeyBufSize ];  //定义一个无符号字符数组作为按键缓冲队列。该队列为先进
                                     //先出，循环存取，下标从 0 到 KeyBufSize-1
unsigned char KeyBufWp=0;       //作为数组下标变量，记录存入位置
unsigned char KeyBufRp=0; //作为数组下标变量，记录读出位置
//如果存入位置与读出位置相同，则表明队列中无按键数据
unsigned char keyHit( void )
{      if( KeyBufWp == KeyBufRp ) return( 0 ); else return( 1 ); }
```

```
unsigned char keyGet( void )
{    unsigned char retVal;  //暂存读出键值
     while( keyHit()==0 );  //等待按键，因为函数 keyHit()的返回值为 0 表示无按键
     retVal = KeyBuf[ KeyBufRp ];    //从数组中读出键值
     if( ++KeyBufRp >= KeyBufSize ) KeyBufRp=0;    //读位置加 1，超出队列则循环回初始位置
     return( retVal );
}

void keyPut( unsigned char ucKeyVal )
{    KeyBuf[ KeyBufWp ] = ucKeyVal;      //键值存入数组
     if( ++KeyBufWp >= KeyBufSize ) KeyBufWp=0;  //存入位置加 1，超出队列则循环回初始位置
}
/*********************************************************************************************
由于某种原因，读出的按键，没有用，但其它任务要用该按键，但传送又不方便。此时可以退回按键队列。
就如取错了信件，有必要退回一样
*********************************************************************************************/
void keyBack( unsigned char ucKeyVal )
{
     /*
     如果 KeyBufRp=0; 减 1 后则为 FFH，大于 KeyBufSize，即从数组头退回到数组尾。或者由于干扰使
     得 KeyBufRp 超出队列位置，也要调整回到正常位置，
     */
     if( --KeyBufRp >= KeyBufSize ) KeyBufRp=KeyBufSize-1;
     KeyBuf[ KeyBufRp ] = ucKeyVal;//回存键值
}
```

```
#include <at89x52.h>
#include “KEY.H”
unsigned char keyScan( void )      //返回 0 表示无按键，或无效按键，其它值为按键编码值
{    code unsigned char keyCode[16]=
     /0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF
     { 0,   0,   0,   0,   0,   0,   0,   1,   0,   0,   0,   2,   0,   3,   4,   0 };
     unsigned char x, y, retVal;
     P1=0x0f;                    //低四位输入，高四位输出 0
     x=P1&0x0f;                  //P1 输入后，清高四位，作为 X 值
     P1=0xf0;                    //高四位输入，低四位输出 0
     y=(P1 >> 4) & 0x0f;   //P1 输入后移位到低四位，并清高四位，作为 Y 值
     retVal = keyCode[x]*4 + keyCode[y];   //根据本公式倒算按键编码
     if( retVal==0 ) return(0); else return( retVal-4 );
}
//比如按键‘1’，得 X=0x7，Y=0x7，算得 retVal= 5，所以返回函数值 1。
//双如按键‘7’，得 X=0xb，Y=0xd，算得 retVal=11，所以返回函数值 7。
```

```
void main( void )
{
    TMOD = （TMOD & 0xf0 ） | 0x01; //不改变 T1 的工作方式，T0 为定时器方式 1
    TL0 = -20000;                   //计数周期为 20000 个主频脉，自动取低 8 位
    TH0 = (-20000)>>8;              //右移 8 位，实际上是取高 8 位
    TR0=1;                          //允许 T0 开始计数
    ET0=1;                          //允许 T0 计数溢出时产生中断请求
    EA=1;                           //允许 CPU 响应中断请求
    while( 1 ) //永远为真，即死循环
    {
        if( keyHit() != 0 )     //如果队列中有按键
            P2=Seg7Code[ keyGet() ];  //从队列中取出按键值，并显示在数码管上
    }
}
void timer0int( void ) interrupt 1   //20ms；T0 的中断号为 1
{   static unsigned char sts=0;
    TL0 = -20000;                   //方式 1 为软件重载
    TH0 = (-20000)>>8;              //右移 8 位，实际上是取高 8 位
    P1_0 = 1;         //作为输入引脚，必须先输出高电平
    switch( sts )
    {
        case 0: if( keyScan()!=0 ) sts=1; break;  //按键则转入状态 1
        case 1:
            if( keyScan()==0 ) sts=0;           //假按错，或干扰，回状态 0
            else{ sts=2; keyPut( keyScan() ); }//确实按键，键值入队列，并转状态 2
            break;
        case 2: if(keyScan()==0 ) sts=3; break;  //如果松键，则转状态 3
        case 3:
            if( keyScan()!=0 ) sts=2;           //假松键，回状态 2
            else sts=0;                     //真松键，回状态 0，等待下一次按键过程
    }
}
```

第五章　　中断系统应用

对于 51 系列单片机的中断资源在本课件中就不再多加描述 ，同学们可以参考书上的一些资料，主要在这里是介绍它的应用。

序号	中断源	中断控制位（允许否）	优先控制位	中断状态	其他
0	X0	EX0	PX0	IE0	INT0
1	Timer0	ET0	PT0	TF0	T0
2	X1	EX1	PX1	IE1	INT1

3	Timer1	ET1	PT1	TF1	T1
4	UART	ES	PS	RXD/TXD	RI/TI
5	Timer2	ET2	PT2	TF2	T2
		EA			

完成以下程序设计(初始化)：

要求：1、将串口中断的级别设置为最高；

2、INT0 工作于边沿模式，INT1 工作于电平模式，这两个中断都是从外部输入；

3、允许 T1 定时器中断。

```
#include <reg52.h>
void main ( void )
{
    EA = 0;
    PS = 1; PT1 = 0; PT0 = 0; PX0 = 0; PX1 = 0;    //设置串口的中断级别最高
    INT1 = 1;   INT0 = 1;       //设置外部输入中断
    IT0 = 1; IT1 = 0;       //设置 INT0 工作于边沿模式，INT1 工作于电平模式
    ET1 = 1;            //允许定时器 1 中断
    EX0 = 1; EX1 = 1;     //允许外部中断 0、1 工作
    ES = 1;          //允许串口中断
    EA = 1;           //开中断
    while ( 1 );
}
```

下面的程序为中断的具体应用，主要是针对 T2 定时器的中断。

```
#include <REG52.h>
void main( void )
{
    EA = 0;     //disable interrupt for system
    C_T2 = 0;         //time
    CP_RL2 = 0;           //Reload
    RCAP2L = -1000;                 //low 8 bits
    RCAP2H = (-1000)>>8;      //high 8 bits
    TL2 = RCAP2L;         //first load to T2
    TH2  = RCAP2H;
    TR2 = 1;                        //start count
    ET2 = 1;    //enable Timer2 interrupt
    EA    = 1;  //open interrupt for system
    while( 1 ){;}
}

void Timer2Int( void ) interrupt 5
{
    TF2 = 0;
```

```
    P1 ^= 0xff;
}
```

下面的程序是将按键和显示放在中断服务程序中进行处理。程序内容为上课时的例子 test2。

clock.h 文件编写如下：

```
#ifndef _clock_h_
#define _clock_h_

#define SysClock        3686400

struct sClock
{
    unsigned char flag;
    unsigned long second;  //2^32 seconds for 136 years
    unsigned int ms;
};

void ClockOpen( void );
struct sClock * ClockGet( void );
//void ClockSet( struct sClock *ptr );
void ClockCall( void );

extern struct sClock gClock;

#endif
```

---

clock.c 文件编写如下：

```
#include <reg52.h>
#include "clock.h"
#include "LedDriver.h"
#include "KeyDriver.h"

void ClockCall_ms( void )
{
    LedTimeCall();
    KeyTimeCall();
}

void ClockOpen( void )                  //初始化 Timer2 产生 1ms 定时中断
{
    gClock.ms=0;
    gClock.second=0;

    CP_RL2 = 0;                                     //重载模式
```

```
    C_T2 = 0;                                   //定时器方式
    RCAP2H = (-(SysClock/1000)) >> 8;           //重载值高 8 位
    RCAP2L = (-(SysClock/1000)) & 0x00ff;       //重载值低 8 位
    TR2  = 1;                                   //允许定时计数
    ET2 = 1;                                    //允许 Timer2 中断
}

void T2int( void ) interrupt 5
{
    TF2 = 0;                            //clear interrupt status
    ClockCall_ms();
}
```

KeyDriver.h 文件编写如下：

```
#ifndef _KeyDriver_H_
#define _KeyDriver_H_

#define     KeyBufSize    4
char kbhit( void );
char getch( void );
void KeyBufIn( char dat );
void KeyTimeCall( void );

#endif
```

KeyDriver.c 文件编写如下：

```
#include <reg52.h>
#include "KeyDriver.h"

unsigned char KeyBufWp=0;
unsigned char KeyBufRp=0;
unsigned char KeyBuf[KeyBufSize];

char kbhit( void ){     return( KeyBufWp - KeyBufRp ); }

char getch( void )
{
    char ret;
    ret = KeyBuf[ KeyBufRp ];
    if( ++KeyBufRp >= KeyBufSize ) KeyBufRp=0;
    return( ret );
}

void KeyBufIn( char dat )
{
```

```
    KeyBuf[ KeyBufWp ] = dat;
    if( ++KeyBufWp >= KeyBufSize ) KeyBufWp=0;
}

void KeyTimeCall( void )
{
    code char KeyCode[]={
    /* 0     1     2     3     4     5     6     7  8     9     AB     C     DE     F   */
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff,     //0
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //1
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //2
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //3
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //4
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //5
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //6
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0x0F, 0xff,0xff,0xff,0x0B, 0xff,0x07,0x03,0xff, //7

    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //8
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //9
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //A
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0x0E, 0xff,0xff,0xff,0x0A, 0xff,0x06,0x02,0xff, //B
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //C
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0x0D, 0xff,0xff,0xff,0x09, 0xff,0x05,0x01,0xff, //D
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0x0C, 0xff,0xff,0xff,0x08, 0xff,0x04,0x00,0xff, //E
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff    //F
    };
    unsigned char KeyScan;
    static unsigned char KeyScanCode=0;
    static unsigned char sts=0;
    static unsigned char ms=20;

    if( --ms ) return;
    ms=20;
    P1=0x0f;
    KeyScan=P1;
    P1=0xf0;
    KeyScan|=P1;
    switch( sts )
    {
        case 0:
            if( KeyScan != 0xff )                    //可能有按键
            { sts=1; KeyScanCode=KeyScan; }
            break;
```

```
            case 1:
                if( KeyScanCode == KeyScan )          //去抖后确为键按下
                {
                    sts = 2;
                    KeyBufIn( KeyCode[ KeyScan ] );      //返回键值
                }
                else                                      //否则认为是干扰，重新检测
                    sts = 0;
                break;

            case 2:
                if( KeyScanCode != KeyScan )          //检测松开按键
                {
                    if( KeyScan == 0xff ) sts=3;
                }
                break;
            //按键超过 1 秒认为是连续按键
            //其后每 0.2 秒一次键，直到松开为止
            //要处理组合按键（即 0.1 秒后确认读键，保证所有组合键到位
            //还可能保持不松开全部按键的情况下，转换按其它键组合
            //
            case 3:
                if( KeyScan == 0xff ) sts=0;       //去抖后确为松开按键
                else sts=2;                               //是干扰
                break;
        }
}
```

LedDriver.h 文件编写如下：

```
#ifndef _LedDriver_H_
#define _LedDriver_H_

/*
显示数据为一个字节，由两部分组成，高三位为属性，低五位为值
    BIT7：为小数点
    BIT6：为闪烁位
    BIT5：保留
*/
#define CharAtr_POINT     0x80
#define CharAtr_FLASH     0x40

#define Char_0        0
#define Char_1        1
#define Char_2        2
```

```
#define Char_3      3
#define Char_4      4
#define Char_5      5
#define Char_6      6
#define Char_7      7
#define Char_8      8
#define Char_9      9
#define Char_a      10
#define Char_b      11
#define Char_c      12
#define Char_d      13
#define Char_e      14
#define Char_f      15
#define Char_N      16      //singned -
#define Char_H      17
#define Char_L      18
#define Char_P      19
#define Char_o      20


extern unsigned char DisBuf[];

#define LedPutchar(bitN, Dat ) {DisBuf[bitN]=Dat;}
void LedPrint(unsigned char);
void LedTimeCall( void );


#endif
```

LedDriver.c 文件编写如下：

```
#include <reg52.h>
#include "LedDriver.h"

code unsigned char LedHexCode[]=
    {
        //0     1     2     3     4     5     6     7
        0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07,
        //8     9     a     b     c     d     e     f
        0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71,
        //-     H     L     P     o
        0x40, 0x76, 0x38, 0x73,   0x5c,

    };

unsigned char DisBuf[4];
```

```
void LedPrint( unsigned char dat )
{
    DisBuf[0] = DisBuf[1];
    DisBuf[1] = DisBuf[2];
    DisBuf[2] = DisBuf[3];
    DisBuf[3] = dat;
}

void LedTimeCall( void )
{
    static unsigned char index=0;
    P2 |= 0x0f;
    P0 = LedHexCode[ DisBuf[index] ];
    P2 &= (1<<index)^0xff;
    if( ++index == 4 ) index=0;
}
```

main.c 文件编写如下：

```
程序功能描述: 多彩的世界,变化无穷

实现方法: 控制单片机左右两排发光二极管
注意事项:运行本程序时,拨码开关 SW1. SW2 全部拨到'OFF'位置(即左边).
        如果 LED6、LED7 和 LED8 不工作，按一下 S2 和 S3 即可
工作方式控制：按键 K1、K2、K3 和 K4 可做出不同的显示。
*/

#include <reg52.h>     //该头文档描述单片机所有特殊功能寄存器的称名,程序中可直接使用,比喻'P1'
#include "LedDriver.h"
#include "KeyDriver.h"
#include "clock.h"

unsigned char function=0;

void main( void )              //一个工程项目必须有一个 main 函数,并且只能有一个 main 函数
{
    char keyVal=0;

    EA = 0;
    ClockOpen();

    LedPrint(Char_6);
    LedPrint(Char_o);
    LedPrint(Char_o);
    LedPrint(Char_d);
```

```
    EA = 1;

    while( 1 )
    {
        if( kbhit() )       //如果有键按下返回非 0 值
        {
            keyVal=getch();  //K1---K16 返回的键值分别为 0---15
            LedPrint( keyVal );
        }
    }

}
```

/*

1. 参考任一个显不方式的模块，增加一种显示方式对应键 K4，左右两排发光二极管交替亮灭
2. 每个按键 Ki(i=1...16)对应一个发光二极管 LEDi，按相应的键 Ki，则对应的灯 LEDi 亮，
   再按，则灭，交替工作。
3. 你现在可以做一下十字路的交通灯管制系统了，做产品就这么容量 ^=^

下面的程序是 test3。只有主程序部分于上面的 test2 有不同，现将 main.c 写在下面供大家参考。

```
#include <reg52.h>     //该头文档描述单片机所有特殊功能寄存器的称名,程序中可直接使用,比喻'P1'
#include "LedDriver.h"
#include "KeyDriver.h"
#include "clock.h"

unsigned char function=0;

void main( void )              //一个工程项目必须有一个 main 函数,并且只能有一个 main 函数
{
    char keyVal=0;

    if( INT0==0 ) function=1;
    if( INT1==0 ) function=2;
    if(     T0==0 ) function=3;
    if(     T1==0 ) function=4;

    EA = 0;
    ClockOpen();

    LedPrint(Char_6);
    LedPrint(Char_o);
    LedPrint(Char_o);
    LedPrint(Char_d);
    EA = 1;
```

```
    switch( function )
    {
        case 0:
            while( 1 )
            {
                if( kbhit() )       //如果有键按下返回非 0 值
                {
                    keyVal=getch();  //K1---K16 返回的键值分别为 0---15
                    LedPrint( keyVal );
                }
            }

        case 1:
            while( 1 )
            {
                unsigned int old, new;
                unsigned char minute, second;
                new = ClockGet()->second;
                if( old != new )
                {
                    minute = (new % 3600) / 60;
                    second = new % 60 ;
                    LedPrint( minute / 10 );
                    LedPrint( (minute % 10 ) | CharAtr_POINT );
                    LedPrint( second / 10 );
                    LedPrint( second % 10 );
                }
                old = new;
            }

        default: function=0;
    }
}
```

第六章　　计数器/定时器的应用

对于 T0，T1 定时器，主要的控制寄存器为 TMOD、TCON，我们可以通过设置这些寄存器的值来改变定时器的工作情况。

例 1：

设置 Timer1 工作于计数模式，工作于方式 2 状态，要求每 16 个脉冲中断一次。

```
#include <reg52.h>
```

```
void main ( void )
{
    EA = 1;
    TMOD = ( TMOD&0XF0 ) | 0x60;
       //设置定时器 1 工作于方式 2，计数模式，并且不改变定时器 0 的工作状态。
    T1 = 1;    //设置 P3.5 为输入状态
    ET1 = 1;  //允许定时器 1 中断
    TH1 = -16;   TL1 = TH1;      //给定时器送初值
    TR1 = 1;       //开启定时器 1（使计数开始）
    EA = 1;      //允许中断
    while (1);

void Timer1_int( void ) interrupt 3       //定时器 1 的中断号是 3
{
    TF1 = 0;    //对于定时器 0 和定时器 1 可以不用写这句，因为硬件会自动对 TF1 进行清零
    TXD =! TXD;
}
```

例 2：完成下面的程序：

要求：1、Timer0 工作在方式 2，作为定时器使用，受门控，每 100 个脉冲中断一次，中断服务程序对 RXD 取反；

2、Timer1 工作在方式 1，作为计数器使用，不受门控，每 4567 个脉冲中断一次，中断后取反 TXD。

程序如下：

```
#include <reg52.h>
void main ( void )
{
    EA = 0;
    TMOD = 0X5A;        //设置好两个定时器的工作情况
    TH0 = -100;  TL0 = TH0;       //给定时器 0 置初值
    ET0 = 1;
    TH1 = (-4567)>>8;    TL1 = -4567;
    ET1 = 1;    T1 = 1;    INT0 = 1;
    TR0 = 1;
    EA = 1;
    while ( 1 );
}

void Timer0_int ( void ) interrupt 1
{
    RXD =! RXD;
}
void Timer1_int ( void ) interrupt 3
{
    TXD =! TXD;
```

```
    TH1 = (-4567)>>8;
    TL1 = -4567;
}
```

例 3：晶体 f_{osc} = 12M,12 分频，用 T0 或者 T1，每毫秒运行一次函数 TimerCall(),定时精度与晶体相同。

```
#include <reg52.h>
void main ( void )
{
    EA = 0;
    TMOD = ( TMOD&0X0F ) | 0X2F;     //因为要求与晶体的时间一致，所以必须采用方式 2
    TH1 = -200;  TL1 = TH1;
    TR1 = 1;
    EA = 1;
    while ( 1 );
}
void Timer0_int ( void ) interrupt 3
{
    static unsigned char TimerC = 4;
    if ( --TimerC == 0)
        TimerCall ();
        TimerC = 4;
}
```

低频频率计的设计

LedDriver.h 如下：

```
*************************************************
#ifndef _LedDriver_H_
#define _LedDriver_H_

/*
显示数据为一个字节，由两部分组成，高三位为属性，低五位为值
    BIT7：为小数点
    BIT6：为闪烁位
    BIT5：保留
*/
#define CharAtr_POINT  0x80
#define CharAtr_FLASH  0x40

#define Char_0         0
#define Char_1         1
#define Char_2         2
#define Char_3         3
#define Char_4         4
#define Char_5         5
```

```
#define Char_6          6
#define Char_7          7
#define Char_8          8
#define Char_9          9
#define Char_a          10
#define Char_b          11
#define Char_c          12
#define Char_d          13
#define Char_e          14
#define Char_f          15
#define Char_N          16        //singned -
#define Char_H          17
#define Char_L          18
#define Char_P          19
#define Char_o          20

extern unsigned char DisBuf[];

#define LedPutchar(bitN, Dat ) {DisBuf[bitN]=Dat;}
void LedPrint(unsigned char);
void LedTimeCall( void );

#endif
```

KeyDriver.h 如下：

```
************************************************
#ifndef _KeyDriver_H_
#define _KeyDriver_H_

#define  KeyBufSize  4
char kbhit( void );
char getch( void );
void KeyBufIn( char dat );
void KeyTimeCall( void );

#endif
```

Clock.h 如下：

```
************************************************
#ifndef _clock_h_
#define _clock_h_
#define SysClock   3686400

struct sClock
{
    unsigned char flag;
```

```
    unsigned long second;   //2^32 seconds for 136 years
    unsigned int ms;
}
void ClockOpen( void );
struct sClock * ClockGet( void );
//void ClockSet( struct sClock *ptr );
void ClockCall( void );
extern struct sClock gClock;

#endif
```

LedDriver.c 如下：

```
*********************************************************
#include <reg52.h>
#include "LedDriver.h"

code unsigned char LedHexCode[]=
    {
        //0     1     2     3     4     5     6     7
        0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07,
        //8     9     a     b     c     d     e     f
        0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71,
        //-     H     L     P     o
        0x40, 0x76, 0x38, 0x73, 0x5c,

    };

unsigned char DisBuf[4];

void LedPrint( unsigned char dat )
{
    DisBuf[0] = DisBuf[1];
    DisBuf[1] = DisBuf[2];
    DisBuf[2] = DisBuf[3];
    DisBuf[3] = dat;
}

void LedTimeCall( void )
{
    static unsigned char index=0;
    P2 |= 0x0f;
    P0 = LedHexCode[ DisBuf[index] & 0x1f ] | (DisBuf[index]&0x80);
    P2 &= (1<<index)^0xff;
    if( ++index == 4 ) index=0;
}
```

KeyDriver.c 如下：

```
************************************************
#include <reg52.h>
#include "KeyDriver.h"

unsigned char KeyBufWp=0;
unsigned char KeyBufRp=0;
unsigned char KeyBuf[KeyBufSize];

char kbhit( void ){    return( KeyBufWp - KeyBufRp ); }

char getch( void )
{
    char ret;
    ret = KeyBuf[ KeyBufRp ];
    if( ++KeyBufRp >= KeyBufSize ) KeyBufRp=0;
    return( ret );
}

void KeyBufIn( char dat )
{
    KeyBuf[ KeyBufWp ] = dat;
    if( ++KeyBufWp >= KeyBufSize ) KeyBufWp=0;
}

void KeyTimeCall( void )
{
    code char KeyCode[]={
    /* 0   1   2   3    4   5   6   7 8   9   A B    C   D E   F  */
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //0
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //1
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //2
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //3
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //4
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //5
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //6
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0x0F, 0xff,0xff,0xff,0x0B, 0xff,0x07,0x03,0xff, //7
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //8
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //9
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //A
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0x0E, 0xff,0xff,0xff,0x0A, 0xff,0x06,0x02,0xff, //B
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, //C
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0x0D, 0xff,0xff,0xff,0x09, 0xff,0x05,0x01,0xff, //D
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0x0C, 0xff,0xff,0xff,0x08, 0xff,0x04,0x00,0xff, //E
```

```
    0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff  //F
    };
    unsigned char KeyScan;
    static unsigned char KeyScanCode=0;
    static unsigned char sts=0;
    static unsigned char ms=20;

    if( --ms ) return; ms=20;
    P1=0x0f; KeyScan=P1; P1=0xf0; KeyScan|=P1; P1=0xff;
    switch( sts )
    {
        case 0:
            if( KeyScan != 0xff )                //可能有按键
            { sts=1; KeyScanCode=KeyScan; }
            break;

        case 1:
            if( KeyScanCode == KeyScan )         //去抖后确为键按下
            {
                sts = 2;
                KeyBufIn( KeyCode[ KeyScan ] );      //返回键值
            }
            else                                 //否则认为是干扰，重新检测
                sts = 0;
            break;

        case 2:
            if( KeyScanCode != KeyScan )         //检测松开按键
            {
                if( KeyScan == 0xff ) sts=3;
            }
            break;
        //按键超过 1 秒认为是连续按键
        //其后每 0.2 秒一次键，直到松开为止
        //要处理组合按键（即 0.1 秒后确认读键，保证所有组合键到位
        //还可能保持不松开全部按键的情况下，转换按其它键组合
        //
        case 3:
            if( KeyScan == 0xff ) sts=0;         //去抖后确为松开按键
            else sts=2;                              //是干扰
            break;
    }
}
```

clock.c 如下：

```
************************************************
#include <reg52.h>
#include "clock.h"
#include "LedDriver.h"
#include "KeyDriver.h"

struct sClock gClock;

void ClockCall_ms( void )
{
    LedTimeCall();
    KeyTimeCall();
}

void ClockCall_second( void )
{
}


void ClockOpen( void )              //初始化 Timer2 产生 1ms 定时中断
{
    gClock.ms=0;
    gClock.second=0;

    CP_RL2 = 0;                                 //重载模式
    C_T2 = 0;                                   //定时器方式
    RCAP2H = (-(SysClock/1000)) >> 8;           //重载值高 8 位
    RCAP2L = (-(SysClock/1000)) & 0x00ff;//重载值低 8 位
    TR2  = 1;                                   //允许定时计数
    ET2 = 1;                                    //允许 Timer2 中断
}

void T2int( void ) interrupt 5
{
    TF2 = 0;                    //clear interrupt status
    ++gClock.ms;
    ClockCall_ms();
    if( gClock.ms >= 1000 )
    {
        gClock.ms = 0;
        ++gClock.second;
        ClockCall_second();
    }
}
```

```
struct sClock * ClockGet( void )
{
    return( &gClock );
}

/*
void ClockSet( struct sClock *ptr )
{
}
*/
```

FrequencyCounter.c 如下：

```
*************************************************
#include <reg52.h> //该头文档描述单片机所有特殊功能寄存器的称名,程序中可直接使用,比喻'P1'
#include "LedDriver.h"
#include "KeyDriver.h"
#include "clock.h"

//信号除输入到 T1（P3.5）外，还输入到 INT1（P3.3）。

#define Stime 18432
#define S0        200

unsigned int us100;             //对 100us 时间间隔单位计数，即有多少个 100us。
unsigned char Second;
unsigned int K64;               //对 64K 单位计数，即有多少个 64K
unsigned char oldT0;
unsigned int oldus, oldK64, oldT1;
unsigned long fcy;              //存放频率值，单位为 Hz
bit HighLow=1;                  //1：表示信号超过 1KHz；0：表示信号低于 1KHz。

void InitialHigh( void )
{
    IE=0; IP=0; HighLow=1;
    TMOD = (TMOD & 0xf0) | 0x02;
    TH0=-200; TL0=TH0; PX0=1; T0=1; ET0=1;

    TMOD = (TMOD & 0x0f) | 0x50;
    TH1=0; TL1=0; T1=1; ET1=1;

    us100=0; Second=0; K64=0;
    oldK64=0; oldT1=0;
    TCON |= 0x50;      //同时置 TR0=1; TR1=1;
    ET2 = 1;
```

```
    EA = 1;
}
void InitialLow( void )
{
    IE=0; IP=0; HighLow=0;
    TMOD = (TMOD & 0xf0) | 0x02;
    TH0=-200; TL0=TH0; ET0=1; TR0=1;

    INT1 = 1; IT1=1; EX1=1;
    us100=0; Second=0; K64=0;
    oldK64=0; oldT1=0;
    ET2 = 1;
    EA = 1;
}

void T0intr( void ) interrupt 1
{
    if( HighLow==0 ){ ++us100; return; }

    if( ++us100 >= 18432 )
    {
        unsigned int tmp1, tmp2;
        TR1=0;
        tmp1=(TH1<<8) + (TL1);
        tmp2=K64;
        TR1=1;
        fcy=((tmp2-oldK64)<<16) + (tmp1-oldT1);
        oldK64=tmp2; oldT1=tmp1;
        Second++;
        us100=0;
    }
}

void T1intr( void ) interrupt 3 {    ++K64;    }

void X1intr( void ) interrupt 2
{
    static unsigned char sts=0;
    switch( sts )
    {
        case 0: sts = 1; break;
        case 1: oldT0=TL0; oldus=us100; sts=2; break;
        case 2:
            {
```

```
            unsigned char tmp1, tmp2;
            TR0=0; tmp1=TL0; tmp2=us100; TR0=1;
            fcy = 1000000L/( (tmp2-oldus)*100L + (256-tmp1)/2 );
            Second ++;
            }
            sts = 0;
            break;
    }
}
void FrequencyCounter( void )
{
    if( HighLow==1) InitialHigh(); else InitialLow();
    while(1)
    {
        if( Second != 0 )
        {
            Second = 0;
            //display fcy  引用前面的数码管驱动程序，注意下面对 T0 中断服务程序的修改
            {
                unsigned char i;
                unsigned char dbit[4];
                unsigned int tmp;
                if( fcy<1000000L )
                {
                    if( fcy < 10000L ){ i=0; tmp=fcy; }
                    else{ i=1; tmp=fcy/10L; }
                }
                else
                {
                    if( fcy < 10000000L ){ i=2; tmp=fcy/100L; }
                    else{ i=3; tmp=fcy/1000L; }
                }

                dbit[0]=tmp/1000;  tmp %= 1000;
                dbit[1]=tmp/100;   tmp %= 100;
                dbit[2]=tmp/10;        tmp %= 10;
                dbit[3]=tmp;
                dbit[i] |= CharAtr_POINT;

                for( i=0; i<4; i++ ) LedPrint( dbit[i] );
            }
            /*
            if( HighLow==1 )
            {    if( fcy<1000L ){ InitialLow();}  }
```

```
            else
            {    if( fcy>1000L ){ InitialHigh();} }
            */
        }
    }
}
//修改 T0 的中断服务程序，让它在完成时标的功能时，同时完成数码管显示刷新


/*
void T0intr( void ) interrupt 1
{
    if( HighLow==0 ) ++us100;
    else
    if( ++us100 >= 10000 )
    {
        unsigned int tmp1, tmp2;
        TR1=0; tmp1=(TH1<<8) + (TL1); tmp2=K64; TR1=1;
        fcy=((tmp2-oldK64)<<16) + (tmp1-oldT1);
        oldK64=tmp1; oldT1=tmp2;
        Second++;
        us100=0;
    }
}
*/
```

speaker.c 如下：

```
************************************************
#include <reg52.h> //该头文档描述单片机所有特殊功能寄存器的称名,程序中可直接使用,比喻'P1'
#include "LedDriver.h"


#define SL1   1
#define SL2   2
#define SL3   3
#define SL4   4
#define SL5   5
#define SL6   6
#define SL7   7
#define SM1   8
#define SM2   9
#define SM3   10
#define SM4   11
#define SM5   12
#define SM6   13
```

```
#define SM7   14
#define SH1   15
#define SH2   16
#define SH3   17
#define SH4   18
#define SH5   19
#define SH6   20
#define SH7   21
#define ST1   22

#define SpeakerBit 0xdf

void Midi( unsigned char, unsigned char );
void SpeakerOpen( void ){ P2 &= SpeakerBit; }
void SpeakerClose( void ){ P2 |= SpeakerBit^0xff; }

code unsigned char sound[]=
{    SL6,6, SL5,2, SL6,2, SM3,6, SM2,6, SM1,1, SM2,1, SM3,4, SL6,4,
     SL7,10, SM1,2, SM2,6, SL7,2, SM1,2, SM2,2, SM3,8,
     SL6,6, SL5,2, SL6,2, SM3,6, SM2,6, SM1,1, SM2,1, SM3,4, SL6,4,
     SL7,6, SM1,2, SM2,8, SM2,4, SM1,2, SL7,2, SL6,6, SM1,2, SL7,10,
     SM1,2,SM2,4, SM2,8,SM1,4,SL7,4,SL6,16,0};
// 1    2    3    4    5    6    7    1
//100, 112, 126, 133, 150, 168, 189, 200
void SpeakerMidi( void )              //一个工程项目必须有一个 main 函数,并且只能有一个 main 函
数
{
     unsigned int i;
     TMOD = ( TMOD & 0xf0 ) | 0x01;
     TR0 = 1;
     EA = 0;
     while(1)
     {
          while(sound[i] != 0)
          {
               Midi( sound[i], sound[i+1] );
               i += 2;
          }
          i=0;
     }
}

void wait( unsigned int time )
{
```

```
    time = -(time*8);
    TH0 = time >> 8;
    TL0 = time;
    TF0 = 0;
    while( TF0 == 0 );
}

void Midi( unsigned char sound, unsigned char time )          //一个工程项目必须有一个
main 函数,并且只能有一个 main 函数
{
    char loop;
    code unsigned int pn[]={    8*105,
                            4*200, 4*178, 4*159, 4*150, 4*133, 4*118, 4*105,
                            2*200, 2*178, 2*159, 2*150, 2*133, 2*118, 2*105,
                            1*200, 1*178, 1*159, 1*150, 1*133, 1*118, 1*105, 100};

    for( loop=0; loop<time; ++loop)
    {
        unsigned int lp, len;
        len = 30000/pn[sound];
        for( lp=0; lp<len; lp++ )
        {
            SpeakerOpen();
            wait( pn[sound] );
            SpeakerClose();
            wait( pn[sound] );
        }
    }
}
```

main.c 如下：

```
*************************************************
#include <reg52.h> //该头文档描述单片机所有特殊功能寄存器的称名,程序中可直接使用,比喻'P1'
#include "LedDriver.h"
#include "KeyDriver.h"
#include "clock.h"

extern void FrequencyCounter( void );
extern void SpeakerMidi( void );

unsigned char function=0;

void main( void )          //一个工程项目必须有一个 main 函数,并且只能有一个 main 函数
{
    char keyVal=0;
```

```
    if( INT0==0 ) function=1;
    if( INT1==0 ) function=2;
    if(   T0==0 ) function=3;
    if(   T1==0 ) function=4;

    EA = 0;
    ClockOpen();


    LedPrint(Char_6);
    LedPrint(Char_o);
    LedPrint(Char_o);
    LedPrint(Char_d);
    EA = 1;

    if( function==0 ) FrequencyCounter();
    if( function==1 ) SpeakerMidi();

    while( 1 ){ ; }
}
```

第七章　　串行接口应用

首先我们来看对于串口所对应的初始化程序。

```
#include <reg52.h>

void main ( void )
{
    EA = 1;
    TMOD = (TMOD & 0X0F) | 0X20;     //串口工作在方式 1
    TH1 = -22118400L/12/32/9600;     //求当波特率是 9600 时定时器的初值
    TR1 = 1;
    SCON = 0X42;
    while ( 1 )
    {
        if (TI == 1){SBUF = 'A';TI = 0;}
    }
}
```

下面我们来完成一个串口收到什么数据就发送什么数据的程序。

```
#include <reg52.h>
```

```
void main ( void )
{
    unsigned char ch;
    EA = 0;
    TMOD = (TMOD & 0X0F) | 0X20;          //串口工作在方式 1
    TH1 = -22118400L/12/32/9600;          //求当波特率是 9600 时定时器的初值
    TR1 = 1;
    SCON = 0X52;
    while ( 1 )
    {
        if(RI==1)
        {
            ch = SBUF;
            RI = 0;
            while (TI==0);
            SBUF = ch;
            TI = 0;
        }
    }
}
```

例：编写如下的程序：单片机收到小写字母，把它变成大写字母发送出去；收到大写字母，变成小写字母发送出去，其他内容不做变化。

```
#include <reg52.h>

void main ( void )
{
    unsigned char ch;
    EA = 0;
    TMOD = (TMOD & 0X0F) | 0X20;     //串口工作在方式 1
    TH1 = -22118400L/12/32/9600;     //求当波特率是 9600 时定时器的初值
    TR1 = 1;
    SCON = 0X52;
    PCON |= 0X80;   //波特率加倍
    while ( 1 )
    {
        if (RI)
        {
            ch = SBUF;
            RI = 0;
            if (ch >= 'A' && ch <= 'Z')    //判断是否是大写字母，如果是则变成小写
            {ch = ch - 'A' + 'a';}
            else if (ch >= 'a' && ch <= 'z')      //判断是否是小写字母，如果是则变成大写
            {ch = ch - 'a' + 'A';}
```

```
            while ( TI == 0)
            SBUF = ch;          //将处理好的数据发送出去
            TI = 0;
        }
    }
}
```

下面我们也将串口的部分做成一个 Driver，把串口当成一个设备进行使用。

UartDriver.h 如下:

```
#ifndef _UART_DRV_H_
#define _UART_DRV_H_

#define RBUFSIZE 4      //设定接受数据缓冲区的大小
#define TBUFSIZE 8      //设定发送数据缓冲区的大小

char UartOpen ( unsigned int bps );         //返回 0,则表示成功
char UartClose ( void );    //关闭串口
char UartRead ( void );     //从接受缓冲区里读数
char UartWrite ( unsigned char Dat );//向发送缓冲区里写数
char UartEmpty ( void );    //返回 1,表示接受数据缓冲区里无新的数据

void UartCall_ms ( void );  //每毫秒中断一次,检查发送缓冲区里是否有数等待发出
char UartString ( char * ); //开机友好提示

#endif
**************************************************
```

UartDriver.c 如下：

```
#include <reg52.h>
#include "UartDriver.h"

unsigned char RBuf[RBUFSIZE];    //定义一个接收数据缓冲区
unsigned char TBuf[TBUFSIZE];    //定义一个发送数据缓冲区

unsigned char RBufRead = 0;      //定义一个接收数据缓冲区读指针
unsigned char RBufWrite = 0;     //定义一个接收数据缓冲区写指针

unsigned char TBufRead = 0;      //定义一个接收数据缓冲区读指针
unsigned char TBufWrite = 0;     //定义一个接收数据缓冲区写指针

bit OldTI = 0;          //定义一个位变量,用于检查发送中断的情况

char UartOpen ( unsigned int bps )
{
    static bit Open = 0;    //定义一个位变量,用于检查串口的工作情况
```

```
    if (Open)return ( -1 );          //如果 Open 为 1,则说明串口无法打开,返回错误代码-1
    Open = 1;
    TMOD = ( TMOD & 0X0F ) | 0X20;   //设置定时器 T1 的工作方式为方式 2
    TH1 = -22118400L/12/32/bps*2;    //设置在波特率为 bps 时定时器 T1 的初值
    PCON |= 0X80;       //使波特率加倍
    TL1 = TH1;
    TR1 = 1; //启动 T1 计数器
    SM0=0;SM1=1;SM2=1;REN=1;RI=0;TI=1;    //PCON = 0X72;  设置串行通信工作在方式 1,允许接收
    PS = 0;ES = 1;
    return ( 0 );
}

char UartClose ( void )
{
    return ( 0 );
}

char UartEmpty ( void )
{
    if ( RBufRead != RBufWrite )      return ( 0 );
    //判断读写指针是否相等,如果不相等说明接受数据缓冲区里有新的数据未读出,即缓冲区不为空
    else return ( 1 );
}

char UartRead ( void )
{
    unsigned char ch;
    ch = RBuf[RBufRead];
    if (++RBufRead >= RBUFSIZE)    RBufRead = 0;
    return (ch);
}

char UartWrite ( unsigned char Dat )
{
    if (( (TBufWrite + 1)%TBUFSIZE ) == TBufRead )     return ( -1 );
        //检查发送缓冲区是否已经满,如果满了，返回失败值-1
    TBuf[TBufWrite] = Dat;
    if (++TBufWrite >= TBUFSIZE)   TBufWrite = 0;
    return ( 0 );
}

void UartCall_ms ( void )
{
    if ( OldTI )
```

```
    {
        if ( TBufRead != TBufWrite )
        {
            OldTI = 0;
            TI = 1;
        }
    }
}

char UartString (unsigned char *str)
{
    while (*str != 0)
    {
        while (UartWrite (*str) != 0);
        ++str;
    }
    return ( 0 );
}


void Uart_int ( void ) interrupt 4
{
    if (RI)
    {
        RBuf[RBufWrite] = SBUF;
        RI = 0;
        if (++RBufWrite >= RBUFSIZE)   RBufWrite = 0;
    }
    if (TI)
    {
        TI = 0;
        if (TBufWrite == TBufRead)    OldTI = 1;  //表示此时并没有数需要送出
        else
        {
            SBUF = TBuf[TBufRead];
            if (++TBufRead >= TBUFSIZE)   TBufRead = 0;
        }
    }
}
```

相应的主程序如下：

```
#include <reg52.h>
#include "UartDriver.h"

void main ( void )
```

```
{
    unsigned char str[]="This is a test program.\n\r";
    unsigned char s0[]="How are you!\n\r";
    unsigned char s1[]="Hello!\n\r";
    C_T2 = 0;  CP_RL2 = 0;          //本条指令即以下均为设置定时器 2 产生 1ms 定时
    RCAP2L = (-1000)&0X00FF;
    RCAP2H = (-1000)>>8;
    TL2 = RCAP2L;   TH2 = RCAP2H;
    TR2 = 1;
    ET2 = 1;
    UartOpen(9600);         //开始初始化串口
    EA = 1;
    UartString(str);   //开机友好提示
    UartString(s1);
    UartString(s0);
    while ( 1 );
}

void Timer2_int ( void ) interrupt 5
{
    TF2 = 0;
    UartCall_ms();
}
```