MSP430程序库<七>按键

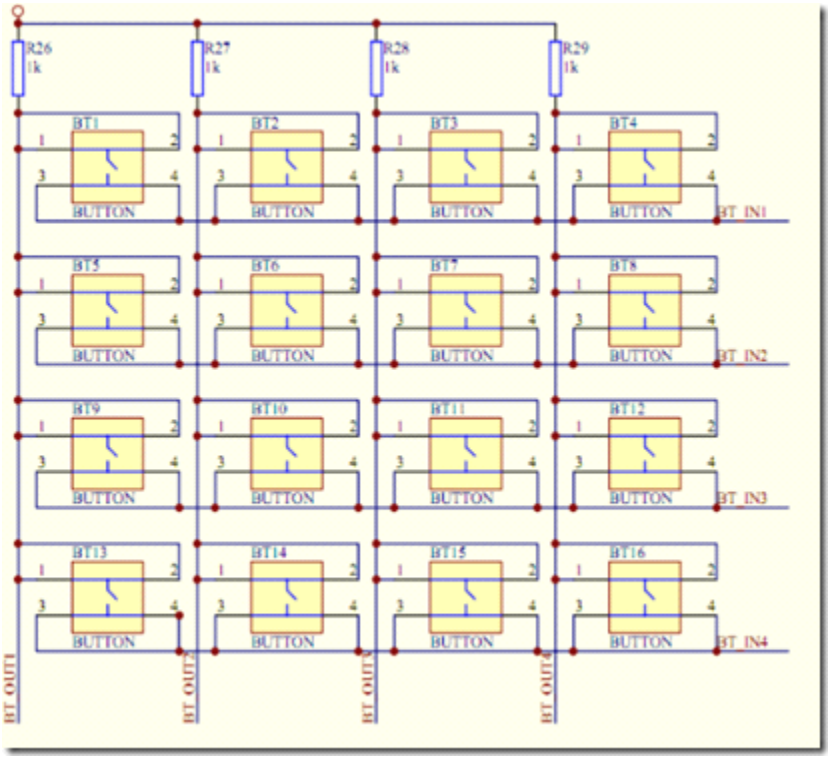
按键是单片机系统最常用的输入设备之一；几乎是只要需要交互输入，就必须有键盘。这篇博客实现了一个通用的键盘程序，只要提供一个读取键值的函数(底层键值)，程序将完成消抖、存入队列等一些列处理。同时本程序提供最常用的4*4矩阵键盘的程序，和4个按键的程序。

1.硬件介绍：

本文主要实现了一个键盘的通用框架，可以很方便的改为不同的键盘函数，这里实现了两种按键4个单独按键和4*4行列扫描的键盘。

4个按键的是这样的：四个按键分别一端接地，另一端接上拉电阻后输入单片机的P1.0-P1.3口；这样，按键按下时，单片机接到低电平，松开时单片机输入信号有上拉电阻固定为高电平。

4*4的按键：行输入信号配有桑拉电阻，无按键时默认电平高电平；列扫描信号线直接接到按键列线；读键时，列扫描信号由单片机给出低电平信号(按列逐列扫描)，读取行信号，从而判断具体是哪个按键；电路图大概如下：



图中，IN 是键盘的列扫描线，OUT 是键盘的输出的行信号线。扫描是也可以按行扫描，这时 IN 是行扫描线，OUT 的按键输出的列信号线。我的程序是按列扫描的（行列扫描原理一样，只是行列进行了交换）。

这里，同时实现了4*4按键的 scanf 函数的移植，同时，加入了之前实现的液晶的 printf 函数的移植，搭建了一个可以交互输入输出的完整的一个系统；液晶的 printf 又加入了函数，实现了退格；可以在输入错误数字的时候退格重新输入。

2.程序实现：

先说一下程序的结构，程序实现了一个循环队列，用来存放已按下的键值，可以保存最新的

四个按键，可以防止按键丢失；程序使用的是中断的方式进行按键，每16ms(用的是看门狗的间隔中断)读一次按键，进行判断键值是否有效，有效则放入队列，等待读取。

循环队列的实现:用数组实现，为判断队满，数组的最后一个元素不用于存储键码值：

```
/**********************宏定义**********************/
#defineKeySize       4              //键码值队列
#defineLength        KeySize+1    //队列数组元素个数
/**************************************************/

/**********************键值队列**********************/
//可 KeySize(Length-1)个键码循环队列占用一个元素空间
   charKey[Length];
```

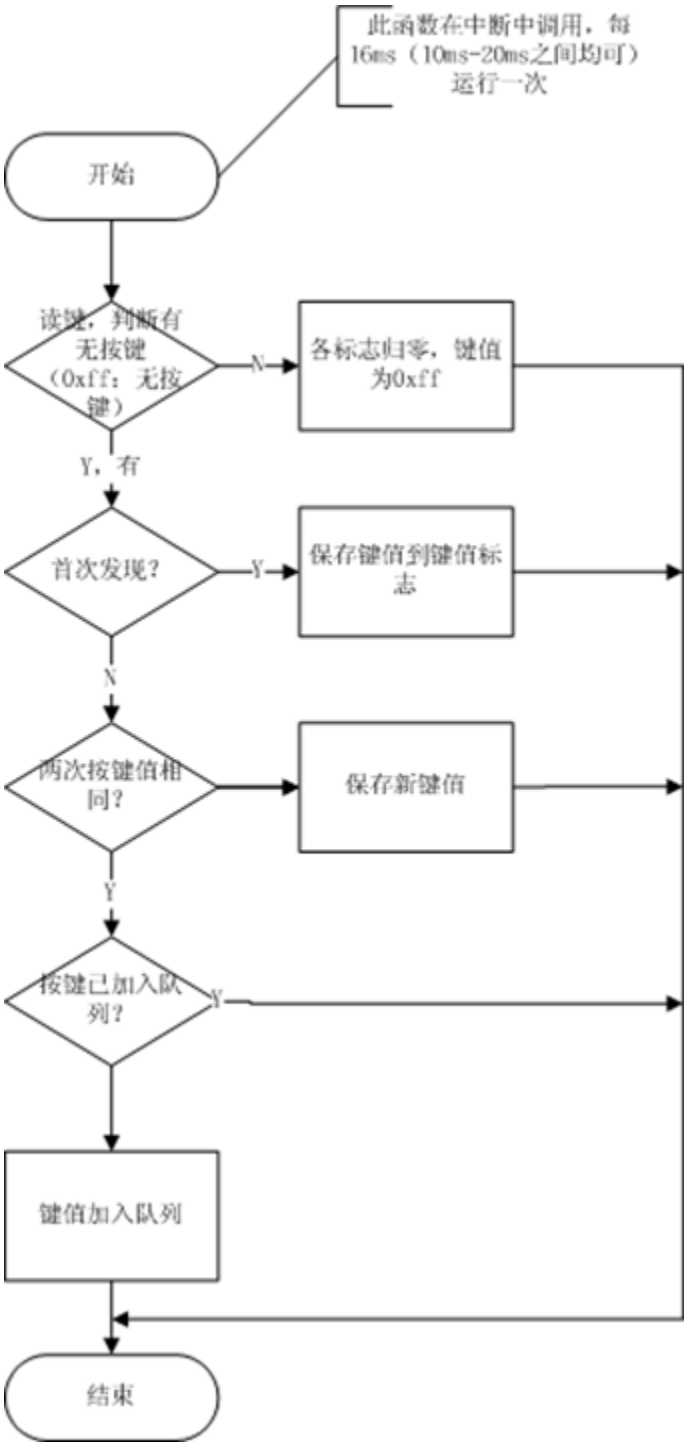
入队函数：入队时，队满则出队一个，以保存最新的四个按键。

```
   voidAddKeyCode(charkeyCode)
   {
    if((rear+1)%Length==front)        //队满
      {
        front=(front+1)%Length;       //出队一个
      }
       Key[rear] = keyCode;
       rear=(rear+1)%Length;
   }
```

出队函数：出队函数即是读取按键的函数，以供其他需要的地方调用。

```
   charReadKey()
   {
       chartemp;
    //if(rear==front) return '\0';      //无按键
      while(rear==front);
       temp = Key[front];
       front=(front+1)%Length;
       returntemp;
   }
```

KeyProcess:这个函数即是键盘处理函数，需要被每10ms-20ms 的时间调用一次的函数，在这里把它放入了看门狗定时器16ms 的中断中;函数流程图和函数内容如下：




```
voidKeyProcess()
{
  static charkeyValue = 0xff;     //按键标识，键值
 static charaddedFlag = 0;        //加入队列标志
    charkeyVal = GetKey();
  if(keyVal==0xff)                  //无按键
    {
          keyValue = 0xff;
          addedFlag = 0;
```

```
        return;
    }
  if(keyValue==0xff)                //之前状态无按键
    {
        keyValue = keyVal;
        return;
    }
  if(keyValue!=keyVal)              //和前次按键不同
    {
      keyValue = keyVal;            //保存新按键值
       return;
    }
  if(addedFlag==1)                  //已加入队列
    {
        return;
    }
    addedFlag = 1;
    AddKeyCode(KeyCode[keyVal]);
}
```

这个函数完成按键的判断，并和上次的比较，从而判断是否是有效按键，再根据是否已经入队保存，去判断是否要保存，入队列保存按键。

这个函数需要每10ms-20ms 中断运行一次：

```
#pragmavector=WDT_VECTOR
__interrupt voidWDT_ISR()
{
    KeyProcess();
}
```

这是430看门狗的间隔定时中断，设置的是每16ms 中断一次：

```
  WDTCTL=WDT_ADLY_16;    //看门狗内部定时器模式16ms
IE1 |= WDTIE;            //允许看门狗中断
```

KeyProcess 里调用了 GetKey 函数，这个函数需要用户提供，以满足特殊的按键需求，这里提供了两个实例：4个按键和4*4矩阵键盘。

4个按键的 getkey 函数：

```
charGetKey()
{
    if((P1IN&0X0F)==0x0E)
    {
        return0;
    }
    if((P1IN&0X0F)==0x0D)
    {
```

```
        return1;
    }
    if((P1IN&0X0F)==0x0B)
    {
        return2;
    }
    if((P1IN&0X0F)==0x07)
    {
        return3;
    }
    return0xff;
}
```

这里根据每个按键，输出按键原始键值，没有按键则输出0xff；当自己提供 getkey 函数时，也需要这样，无按键时返回0xff

把对应原始键值翻译成所需键码，用数组 KeyCode：

```
charKeyCode[] = "0123";    /*4个按键时*/
```

这里把它转化成 ASCII 码输出，需要的话可以自行更改。

4*4矩阵键盘：getkey：

```
charGetKey()
{
  P1DIR |= 0XF0;                    //高四位输出
    for(inti=0;i<4;i++)
    {
        P1OUT = 0XEF << i;
        for(intj=0;j<4;j++)
        {
            if((P1IN&(0x01<<j))==0)
            {
                return(i+4*j);
            }
        }
    }
    return0xff;
}
```

这里是按列扫描，可以随意改成其他扫描方式，只要获取原始键值即可，无按键是须返回0xff。

KeyCode，翻译成 ASCII 码：

```
charKeyCode[] = "0123456789ABCDEF"
```

到这里，正常的键盘程序结束，调用时只需加入 Key.c，包含 Key.h 即可使用，先调用 KeyInit 后，就可以正常的读键了。这里不再细说。

scanf 移植：scanf 移植时，需要的是 ASCII 码字符型设备，利用 ASCII 码输入数据还必须要有回车键，只有这样，才能用 scanf 输入数据，这里为了输入数据错误时，可以退格修改，按键还有一个退格键。

键盘结构：

1	2	3	退格
4	5	6	保留
7	8	9	保留
保留	0	保留	回车

保留键用字符'\0',回车'\n'退格'\b'

所以：KeyCode：

```
charKeyCode[] = "123\b456\000789\0\0000\0\r"; /* 4*4,scanf 移植*/
```

在字符串里，\0后面是数字时，必须用'\000'否则，c 语言编译器认为\0和后面的数字组合为一个字符。

scanf 的移植，需要实现 getchar 函数，这里和之前的 getchar 函数类似，把它放到了 Getchar.c 文件里，内容如下：

```
#include<stdio.h>
#include"Key.h"

#defineLINE_LENGTH 20            //行缓冲区大小，决定每行最多输入的字符数

/*标准终端设备中，特殊 ASCII 码定义，请勿修改*/
#defineInBACKSP 0x08              //ASCII  <--  (退格键)
#defineInDELETE 0x7F              //ASCII <DEL> (DEL 键)
#defineInEOL '\r'                //ASCII <CR>  (回车键)
#defineInLF '\n'                  //ASCII <LF>  (回车)
#defineInSKIP '\3'                //ASCII control-C
#defineInEOF '\x1A'               //ASCII control-Z

#defineOutDELETE "\x8 \x8"       //VT100 backspace and clear
#defineOutSKIP "^C\n"            //^C and new line
#defineOutEOF "^Z"               //^Z and return EOF

intgetchar()
{
 static charinBuffer[LINE_LENGTH + 2];        //Where to put chars
static charptr;                              //Pointer in buffer
   charc;
```

```
while(1)
{
 if(inBuffer[ptr])                          //如果缓冲区有字符
    return(inBuffer[ptr++]);                //则逐个返回字符
ptr = 0;                                    //直到发送完毕，缓冲区指针归零
while(1)                                    //缓冲区没有字符，则等待字符输入
   {
     c = ReadKey();                          //等待接收一个字符==移植时关
键
    if(c == InEOF && !ptr)                  //==EOF==  Ctrl+Z
    {                                       //只有在未入其他字符时才有效
        printf(OutEOF);                     //终端显示 EOF 符
        returnEOF;                          //返回 EOF（-1）
      }
     if(c==InDELETE || c==InBACKSP)        //==退格或删除键==
      {
        if(ptr)                             //缓冲区有值
          {
            ptr--;                          //从缓冲区移除一个字符
           printf(OutDELETE);               //同时显示也删掉一个字符
          }
       }
     else if(c == InSKIP)                   //==取消键 Ctrl+C ==
      {
        printf(OutSKIP);                    //终端显示跳至下一行
       ptr = LINE_LENGTH + 1;               //==0 结束符==
          break;
       }
     else if(c == InEOL||c == InLF)         //== '\r' 回车=='\n'回车
      {
        putchar(inBuffer[ptr++] = '\n');//终端换行
       inBuffer[ptr] = 0;                   //末尾添加结束符（NULL）
       ptr = 0;                             //指针清空
          break;
       }
     else if(ptr < LINE_LENGTH)             //== 正常字符 ==
      {
        if(c >= ' ')                        //删除 0x20以下字符
          {
            //存入缓冲区
              putchar(inBuffer[ptr++] = c);
           }
       }
     else                                   //缓冲区已满
```

```
            {
              putchar('\7');                    //== 0x07 蜂鸣符，PC 回响一声
            }
        }
    }
}
```

这里是支持退格等键的详细函数。

如果不需要支持退格，可以简化为：

```
intgetchar()
{
    returnReadKey();
}
```

要实现 scanf 调用，还需要设置，详细设置参考：MSP430程序库<四>printf 和 scanf 函数移植；需要把库设置为 CLIB；在 Option-general option-library configuration 里面。

这样，键盘的 scanf 移植完成，需要使用时，只需加入对 stdio.h 文件的包含，然后完成键盘的初始化即可。

3.使用示例：

这里，示例实现的是键盘和液晶的简单交互；键盘输入数据，液晶正常显示；就像 c 语言调试时键盘和屏幕一样；当然没有那个丰富啦。

液晶的部分，用的是原来实现的程序，在这里，为了支持输入错误时退格，对原来的 printf 函数加入了退格支持。具体参考：MSP430程序库<四>printf 和 scanf 函数移植(已经更新)。

项目中接入液晶的 c 程序文件和 printf 的程序文件（Lcd12864.c、Printf.c），加入 Lcd12864.h 的文件包含；初始化液晶后，就可用 printf 向液晶输出要显示的内容了。

键盘：加入 Key.c，包含 Key.h，加入 Getchar.c，程序中初始化键盘；然后设置所用的 lib 为 CLIB，具体设置见：MSP430程序库<四>printf 和 scanf 函数移植。之后就可以用键盘和液晶完成和430单片机简单的交互了。

详细参考示例工程和 main.c。

```
#include<msp430x16x.h>
#include<stdio.h>
#include"Lcd12864.h"
#include"Key.h"

longa;
voidmain( void)
{
  // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
     ClkInit();
     LcdInit();
```

```
    KeyInit();
    _EINT();
    while(1)
    {
        printf("请输入数字：");
        scanf("%ld",&a);
        printf("输入的数字是：%ld",a);
        _NOP();
    }
}
```

这样，就可以用键盘向单片机输入数据，同时利用液晶可以很容易的知道数据输入的是否有问题。

键盘的程序库就到这里。

相关文章及附件下载：http://www.ideyi.org/bbs/article_1077_371614.html