新型的按键扫描程序

以下假设你懂 C 语言，因为纯粹的 C 语言描述，所以和处理器平台无关，你可以在 MCS-51，AVR，PIC，甚至是 ARM 平台上面测试这个程序性能。当然，我自己也是在多个项目用过，效果非常好的。好了，工程人员的习惯，废话就应该少说，开始吧。

核心算法:

```
unsigned char Trg;
unsigned char Cont;
void KeyRead( void )
{
   unsigned char ReadData = PINB^0xff;     // 1
   Trg   = ReadData & (ReadData ^ Cont);    // 2
   Cont  = ReadData;                        // 3
}
```

完了。有没有一种不可思议的感觉？当然，没有想懂之前会那样，想懂之后就会惊叹于这算法的精妙！！

下面是程序解释:

Trg(triger)代表的是触发，Cont(continue)代表的是连续按下。

1:读 PORTB 的端口数据，取反，然后送到 ReadData 临时变量里面保存起来。

2:算法 1，用来计算触发变量的。一个位与操作，一个异或操作，我想学过 C 语言都应该懂吧？Trg 为全局变量，其它程序可以直接引用。

3:算法 2，用来计算连续变量。

看到这里，有种“知其然，不知其所以然”的感觉吧？代码很简单，但是它到底是怎么样实现我们的目的的呢？好，下面就让我们绕开云雾看青天吧。

我们最常用的按键接法如下：AVR 是有内部上拉功能的，但是为了说明问题，我是特意用外部上拉电阻。那么，按键没有按下的时候，读端口数据为 1，如果按键按下，那么端口读到 0。下面就看看具体几种情况之下，这算法是怎么一回事。

(1)没有按键的时候

端口为 0xff，ReadData 读端口并且取反，很显然，就是 0x00 了。

Trg = ReadData & (ReadData ^ Cont); (初始状态下，Cont 也是为 0 的)很简单的数学计算，因为 ReadData 为 0，则它和任何数“相与”，结果也是为 0 的。

Cont = ReadData; 保存 Cont 其实就是等于 ReadData，为 0；

结果就是:

ReadData = 0；

Trg = 0；

Cont = 0；

(2)第一次 PB0 按下的情况
端口数据为 0xfe，ReadData 读端口并且取反，很显然，就是 0x01 了。
Trg = ReadData & (ReadData ^ Cont); 因为这是第一次按下，所以 Cont 是上次的值，应为为 0。那么这个式子的值也不难算，也就是 Trg = 0x01 & (0x01^0x00) = 0x01
Cont = ReadData = 0x01；
结果就是:
ReadData = 0x01；
Trg = 0x01；Trg 只会在这个时候对应位的值为 1，其它时候都为 0
Cont = 0x01；

(3)PB0 按着不松(长按键)的情况
端口数据为 0xfe，ReadData 读端口并且取反是 0x01 了。
Trg = ReadData & (ReadData ^ Cont); 因为这是连续按下，所以 Cont 是上次的值，应为为 0x01。那么这个式子就变成了 Trg = 0x01 & (0x01^0x01) = 0x00
Cont = ReadData = 0x01；
结果就是:
ReadData = 0x01；
Trg = 0x00；
Cont = 0x01；
因为现在按键是长按着，所以 MCU 会每个一定时间(20ms 左右)不断的执行这个函数，那么下次执行的时候情况会是怎么样的呢？
ReadData = 0x01；这个不会变，因为按键没有松开
Trg = ReadData & (ReadData ^ Cont) = 0x01 & (0x01 ^ 0x01) = 0 ，只要按键没有松开，这个 Trg 值永远为 0 ！！！
Cont = 0x01；只要按键没有松开，这个值永远是 0x01！！

(4)按键松开的情况
端口数据为 0xff，ReadData 读端口并且取反是 0x00 了。
Trg = ReadData & (ReadData ^ Cont) = 0x00 & (0x00^0x01) = 0x00
Cont = ReadData = 0x00；
结果就是:
ReadData = 0x00；
Trg = 0x00；
Cont = 0x00；

很显然，这个回到了初始状态，也就是没有按键按下的状态。
总结一下，不知道想懂了没有？其实很简单，答案如下：
Trg 表示的就是触发的意思，也就是跳变，只要有按键按下(电平从 1 到 0 的跳变)，那么 Trg 在对应按键的位上面会置一，我们用了 PB0 则 Trg 的值为 0x01，类似，如果我们 PB7 按下的话，Trg 的值就应该为 0x80 ，这个很好理解，还有，最关键的地方，Trg 的值每次按下只会出现一次，然后立刻被清除，完全不需要人工去干预。所以按键功能处理程序不会重复执行，省下了一大堆的条件判断，这个可是精粹哦！！Cont 代表的是长按键，如果 PB0 按着不放，那么 Cont 的值就为 0x01，相对应，PB7 按着不放，那么 Cont 的值应该

为 0x80，同样很好理解。
如果还是想不懂的话，可以自己演算一下那两个表达式，应该不难理解的。因为有了这个支持，那么按键处理就变得很爽了，下面看应用：

应用一:一次触发的按键处理
假设 PB0 为蜂鸣器按键，按一下，蜂鸣器 beep 的响一声。这个很简单，但是大家以前是怎么做的呢？对比一下看谁的方便？

```
#define  KEY_BEEP  0x01
void  KeyProc(void)
{
   if  (Trg  &  KEY_BEEP)   //  如果按下的是 KEY_BEEP
     {
       Beep();            //  执行蜂鸣器处理函数
      }
}
```

怎么样？够和谐不？记得前面解释说 Trg 的精粹是什么？精粹就是只会出现一次。所以你按下按键的话，Trg & KEY_BEEP 为“真”的情况只会出现一次，所以处理起来非常的方便，蜂鸣器也不会没事乱叫，hoho～～～或者你会认为这个处理简单，没有问题，我们继续。

应用 2:长按键的处理
项目中经常会遇到一些要求，例如：一个按键如果短按一下执行功能 A，如果长按 2 秒不放的话会执行功能 B，又或者是要求 3 秒按着不放，计数连加什么什么的功能，很实际。不知道大家以前是怎么做的呢？我承认以前做的很郁闷。但是看我们这里怎么处理吧，或许你会大吃一惊，原来程序可以这么简单,这里举个简单例子，为了只是说明原理，PB0 是模式按键，短按则切换模式，PB1 就是加，如果长按的话则连加（玩过电子表吧？没错，就是那个！）

```
#define  KEY_MODE  0x01    //  模式按键
#define  KEY_PLUS  0x02    //  加
void  KeyProc(void)
{
   if  (Trg  &  KEY_MODE)   //  如果按下的是 KEY_MODE，而且你常按这按键也没有用，
     {              //它是不会执行第二次的哦 ，  必须先松开再按下
       Mode++;       //  模式寄存器加 1，当然，这里只是演示，你可以执行你想
                  //  执行的任何代码
     }
   if  (Cont  &  KEY_PLUS)       //  如果“加”按键被按着不放
     {
       cnt_plus++;          //  计时
       if  (cnt_plus  >  100)   //  20ms*100  =  2S  如果时间到
         {
           Func();        //  你需要的执行的程序
         }
```

```
    }
}
```

不知道各位感觉如何？我觉得还是挺简单的完成了任务，当然，作为演示用代码。

应用 3:点触型按键和开关型按键的混合使用
点触形按键估计用的最多，特别是单片机。开关型其实也很常见，例如家里的电灯，那些按下就不松开，除非关。这是两种按键形式的处理原理也没啥特别，但是你有没有想过，如果一个系统里面这两种按键是怎么处理的？我想起了我以前的处理，分开两个非常类似的处理程序，现在看起来真的是笨的不行了，但是也没有办法啊，结构决定了程序。不过现在好了，用上面介绍的办法，很轻松就可以搞定。原理么？可能你也会想到，对于点触开关，按照上面的办法处理一次按下和长按，对于开关型，我们只需要处理 Cont 就 OK 了，为什么？很简单嘛，把它当成是一个长按键，这样就找到了共同点，屏蔽了所有的细节。程序就不给了，完全就是应用 2 的内容，在这里提为了就是说明原理～～

好了，这个好用的按键处理算是说完了。可能会有朋友会问，为什么不说延时消抖问题？哈哈，被看穿了。果然不能偷懒。下面谈谈这个问题，顺便也就非常简单的谈谈我自己用时间片轮办法，以及是如何消抖的。延时消抖的办法是非常传统，也就是 第一次判断有按键，延时一定的时间(一般习惯是 20ms)再读端口，如果两次读到的数据一样，说明了是真正的按键，而不是抖动，则进入按键处理程序。
当然，不要跟我说你 delay(20)那样去死循环去，真是那样的话，我衷心的建议你先放下手上所有的东西，好好的去了解一下操作系统的分时工作原理，大概知道思想就可以，不需要详细看原理，否则你永远逃不出“菜鸟”这个圈子。当然我也是菜鸟。我的意思是，真正的单片机入门，是从学会处理多任务开始的，这个也是学校程序跟公司程序的最大差别。当然，本文不是专门说这个的，所以也不献丑了。

我的主程序架构是这样的：

```
volatile unsigned char Intrcnt;
void InterruptHandle()    // 中断服务程序
{
  Intrcnt++;              // 1ms 中断 1 次，可变
 }
void main(void)
{
   SysInit();
   while(1)                // 每 20ms 执行一次大循环
    {
      KeyRead();           // 将每个子程序都扫描一遍
      KeyProc();
      Func1();
      Funt2();
      …
      …
      while(1)
```

```
        {
          if (Intrcnt>20)     // 一直在等，直到 20ms 时间到
            {
              Intrcnt="0";
              break;          // 返回主循环
            }
        }
      }
}
```

貌似扯远了，回到我们刚才的问题，也就是怎么做按键消抖处理。我们将读按键的程序放在了主循环，也就是说，每 20ms 我们会执行一次 KeyRead()函数来得到新的 Trg 和 Cont 值。好了，下面是我的消抖部分,很简单,基本架构如上，我自己比较喜欢的，一直在用。当然，和这个配合，每个子程序必须执行时间不长，更加不能死循环，一般采用有限状态机的办法来实现，具体参考其它资料咯。

懂得基本原理之后，至于怎么用就大家慢慢思考了，我想也难不到聪明的工程师们。例如还有一些处理，怎么判断按键释放？很简单，Trg 和 Cont 都为 0 则肯定已经释放了。在这个基础上再增加一个按键释放检测功能，程序如下：

```
volatile unsigned char Trg;
volatile unsigned char Cont;
volatile unsigned char Release;
// 再增加新功能！
void KeyRead( void )
{
  unsigned char ReadData = PINB^0xff;      // 1  读键值
  Trg = ReadData & (ReadData ^ Cont);      // 2  得到按下触发值
  Release= (ReadData ^ Trg ^ Cont);        // 3  得到释放触发值
  Cont = ReadData;                         // 4  得到所有未释放的键值
}
```