

# 简捷的键盘扫描去抖方法（1）

今越电子制作 刘泽民

[www.jyotech.com](http://www.jyotech.com)

实现简洁高效的键盘处理对于提高单片机系统的可靠性及速度有重要意义。键盘处理的重要环节是扫描和去抖，不少书籍在介绍这些内容时往往只介绍查询和循环延迟的方法，这使初学者比较容易理解，但在实际中通常不会使用这些方法，因为那样的话效率太低了。这里我们介绍一种采用定时扫描和计数去抖的键盘处理方法，它不仅使程序非常简单高效，而且便于灵活扩展实现处理更复杂的键盘要求。本文先介绍它的基本算法。

## 算法流程图

图 1 为该算法的流程图，它用到下面的变量和常数。

### 变量

KscnaBuf： 键盘扫描码缓存

KScan： 键盘扫描码

KCount： 去抖计数器

KReady： 按键有效标志

### 常量

KD\_val： 去抖计数长度

算法的基本过程如下：CPU 以一定时间间隔周期地执行此键盘扫描处理程序。先是对键盘作扫描，获得反映键盘状态的键盘扫描码；然后对扫描码进行前后对比和定时计数，实现去抖；去抖后置位 KReady 标志，通知键盘分析程序已检测到有效按键。

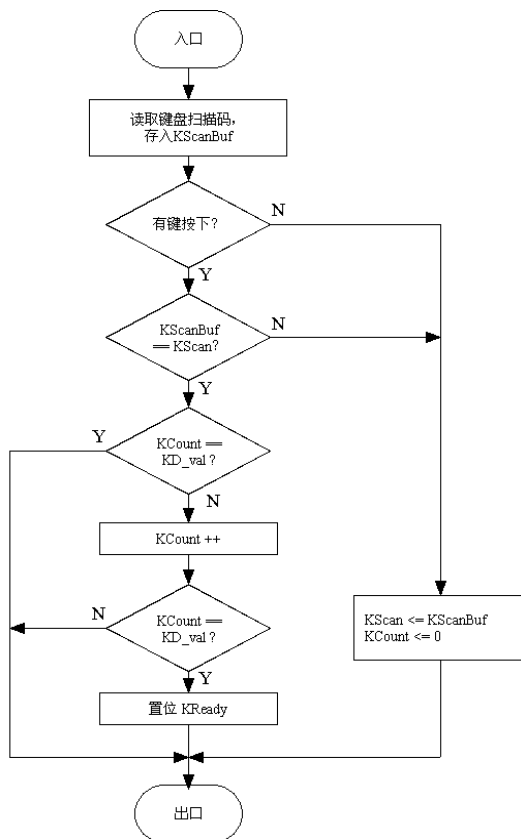


图 1

## 关于键盘扫描码

键盘扫描码反映读取键盘时刻的键盘动作状态。对键盘扫描码的基本要求是它要对每一个单键作唯一编码，如果要使用多键组合则要能对每一种组合进行唯一编码。通常能做到对任意单键或双键作唯一编码即能满足大多数场合的要求。

具体键盘扫描码的编码方法随采用电路的不同而不同，这里以图 2 中的 4 X 4 键盘为例说明。在该电路中，D4 - D7 是输出，D0 - D3 是输入。在读取键盘时程序分别将 D4，D5，D6 和 D7 单独地置成低电平，然后依次读取 D0 - D3，即可获得所有按键的状态。我们可以将 D0 - D3 与 D4 - D7 合在一起，作为按键的编码。例如，当 A 键按下时，得到的编码是 10111110；当 B 键按下时，得到的编码是 01110111。为了反映整个键盘的状态，可以将 D0 - D3 不全为 1 的编码作逻辑与，其结果作为键盘扫描码。可以验证，这样设计的扫描码能够唯一地标识任意单键或双键的动作状态。

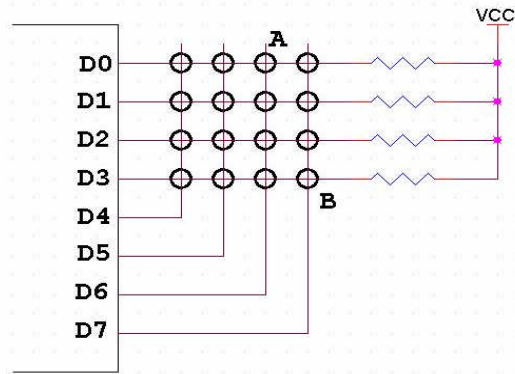


图 2

### 去抖的实现

我们知道，当一个按键按下和释放时，对应信号线电平的变化有一个不稳定期，即所谓“抖动”，这是因为多数开关的闭合和断开都有一个过程，并不是即刻实现的。在读取键盘状态是必须避开这个不稳定期，以免造误判，这样一个做法叫做“去抖”。

图 3 是一按键的动作波形，箭头表示定时扫描时读取键盘的时刻。可以看出，如果当在一段时间内都连续多次读到同一个非空(即有键按下)的扫描码，我们可以认为这时按键已处于稳定状态，这时得到的扫描码就代表了一个键盘动作，抖动的影响已经被剔除。按照这个想法，我们只需设置一计数器，当每次读到的非空扫描码与上一次的相同，就将计数器加 1；而当一旦读到不同的扫描码或空码则立即将计数器清 0。如果计数器的值达到某预定值 N 时，则表示连续 N 次读到同一扫描码，于是可认为已经读到了有效的按键。

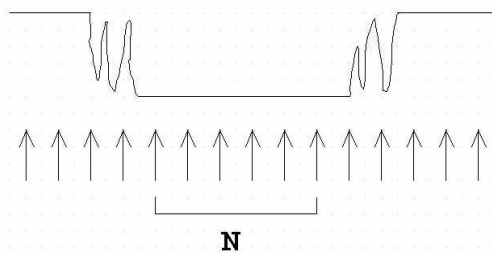


图 3

设键盘扫描的间隔是 T，则去抖的时间就是  $N * T$ 。在实际中可以根据按键的特性选取适当的 N 和 T 值，以达到可靠的去抖效果。

### 应用说明及示例

键盘扫描程序通常用定时中断周期地启动执行。不过一般不宜将整个键盘扫描程序全部放在中断响应程序里，因为那样会降低系统对其他中断响应的性能。通常可以把它放在程序的主循环里（这里只是针对前后台系统讨论，若是多任务系统可以将其设计成一个任务），由定时中断程序通过一个标志周期地启动其执行。当然，如果系统对中断响应性能要求不高，也可以将整个键盘扫描程序放在中断响应程序里。

在系统启动时，需要对键盘扫描程序作初始化，即将 KReady 和 KDcnt 清 0。扫描获得的有效键盘动作通过变量 KScan 和标志 KReady 传递给键盘分析执行程序。

下面应用示例程序。注意这不是完整的程序，若要直接使用则需要加上变量和常量定义，并根据所用的编译器写中断响应函数。

```

// -- 中断程序
    ... ..
DoKeyScan = 1;
    ... ..

// -- 键盘扫描及去抖程序
void KeyScan(void)
{
// -- 扫描键盘，获得扫描码

    ... ..
// -- 去抖处理
if((KScanBuf == NoKey) || (KScanBuf != KScan)) {
    KScan = KScanBuf;
    KCount = 0;
    }
else {
    if(KCount == KD_val) return;    // 避免重复
    KCount++;
    If(KCount == KD_val) KReady = 1;
    }
}

// -- 主程序
int    main()
{
// -- 初始化
KReady = 0;
KCount = 0;

    ... ..
while ( 1 ) {

    ... ..
    if (DoKeyScan == 1) {
        // -- 作键盘扫描去抖
        DoKeyScan = 0;
        KeyScan( );
    }
}

```

```
if (KReady == 1) {  
    // -- 作键盘分析处理  
    KReady = 0;  
  
    ... ..  
}  
  
... ..  
}  
}
```

该算法稍作修改，还可以灵活实现其他特殊的键盘要求，如重复、长按、释放时动作等，对此我们将另觅篇幅介绍。

( 2008.1.15 )