详解矩阵按键扫描

> ## 屋脊雀工作室编写
>
> 立志提供能用的源码例程
>
> ## 版权所有 翻版必究
>
> 最后更新时间:2018.03.15

按键输入是人接交互的一个重要输入途径。键盘，就是一个最常见的矩阵按键。
在其他电子消费产品中，常见的是4*4矩阵按键。
用户按下按键后，程序是如何知道用户按下哪个键呢？这就是我们码农存在的价值。我们会编写一段叫做矩阵按键扫描的程序。如何扫描？我们从头说起。

单键扫描

单键，并不是指一个按键，而是指一个IO口控制一个按键，原理图如下：

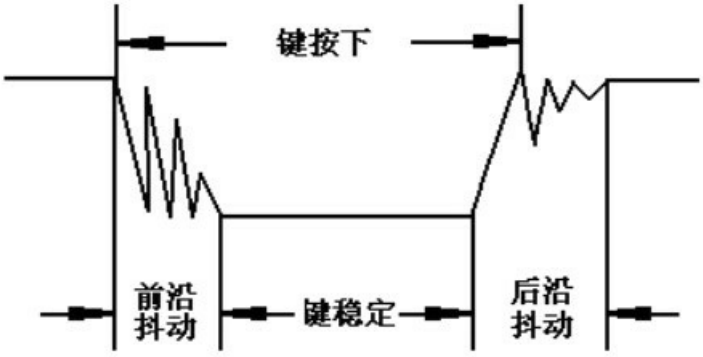
PA0这个IO口接到按键的一端，按键另外一端接到地。
当按键按下，两端短路，IO口就接到地，就是低电平。
那没按下时，IO口啥都没接，为啥是高电平呢？因为IO口在芯片内部可以配置连接一个内部上拉电阻。如果使用了没有内部上拉电阻的IO，就只能在外部接一个电阻将IO口上拉到高电平。

按键扫描方式

首先我们要记住的以下常识：

1. 芯片跑得很快。
   从一个IO口读取输入电平，只是一瞬间的事情。
2. 手可能会抖动。
3. 机械按键可能会抖动。

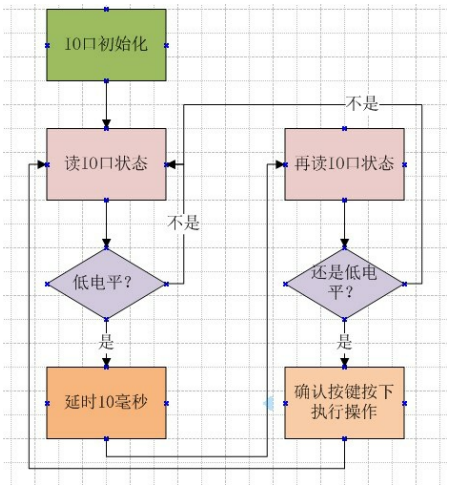
用示波器分析操作按键的波形，很可能如下图：



很明显，如果当我们在抖动的时候读IO状态，得到的值将是随机的。因此在按键扫描中最重要的一个内容就是去抖动。
去抖动的原理很简单：

> 间隔多读几次，连续相同则认为状态是可靠的。

通常大家看到的教程，按键流程如下：



这样的单键扫描流程通常有如下问题：

> 1. 去抖动通过硬延时实现。
> 2. 只对按下去抖动，没有对松开去抖动。

硬延时在真正的开发中，是绝对不建议使用的，无乱是扫描还是其他功能。
至于按下去抖动松开去抖动，像锁定开关，会有抖动，通常的锅仔，由于机械结构做的好，很少见抖动了。不过还是建议加上去抖动。

那么如何优化掉硬延时呢？
在复杂一点的单片机系统中，常用的是轮询模式。轮询模式的代码模式大概如下：

```
main(void)
{

        初始化

        while(1)
        {
                扫描按键();
                扫描串口();
                ........
        }

}
```

在轮询模式下，驱动常用的一种编码手段就是步骤拆分。什么叫步骤拆分？
假设轮询按键扫描的间隔是2毫秒（2毫秒执行一次扫描按键）。那么我们就在按键扫描里面增加一个防抖计数和一个步骤计数。

```
scan_key(void)
{
        第一步，检测按键是否按下

        第二步，判断防抖计数，
                记到5次，就到10毫秒了。
                再判断按键有没有按下

        回到第一步
}
```

程序流程如上面的伪代码。每次进入scan_key这个函数，只会执行一个步骤。这样，陷入scan_key的时间将会很短，只需要几个us。

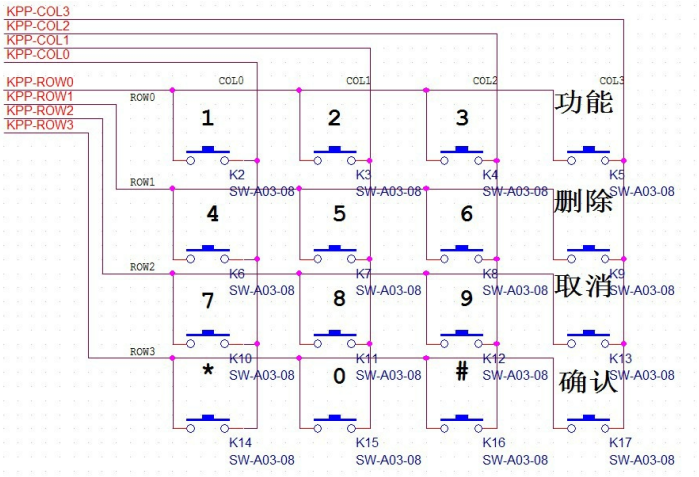
在while中需要轮询的其他功能就可以很快得到执行。
当然，其他轮询的功能也不能陷入太久。

对于第二个问题：松开没有去抖动。
我们的解决方法并不是松开的时候也增加去抖动，而是将松开和按下合并处理。在扫描的时候，不区分松开和按下，我们判断的是按键是否有变化。经过去抖动扫描得到稳定的状态后，再根据状态判断是按下还是松开。

对于单键扫描的程序，在此就不展开了，大家可以根据原理自己尝试编写。

矩阵按键的扫描原理

单键模式有一个很大缺陷，当需要较多的按键时，需要的IO口就多。例如16个按键的时候，就需要16个IO口。
矩阵按键在IO口方面就很有优势了。下面的接法就叫做矩阵按键。



> 在8根IO上要串上限流电阻，上图没体现。

从图上可以看出，只需要8跟IO口，就可以实现16个按键的输入。原理是什么呢？我们通过分析扫描方法解释原理。
矩阵按键扫描通常有两种方法：交叉扫描、逐行扫描。
交叉扫描速度快，程序简单，扫描结果通过查表获取。但是缺陷也多。
逐行扫描需要轮询所有行，程序也稍微复杂，但是可以识别键盘的多种状态。
通常我们说的按键扫描都是用逐行扫描。
就像字面意思说的，逐行逐行扫描。
例如上面原理图，

> KPP-ROW0输出0，其他ROW输出1，读取4根COL IO的状态，就可得到第一行四个按键1、2、3、功能的状态了。（当然，还需要去抖动）。
> 然后KPP-ROW1输入0，其他ROW输出1，读4根COL IO的状态。。。。。。
> 。。。。
> 不断重复从ROW0到ROW3，这就是逐行扫描。

这就叫做矩阵按键逐行扫描。
上面仅仅说明原理，真正的逐行扫描当然没那么简单。下面我们就用一个完整的逐行扫描说明程序应该如何写。

```
/**
 *@brief:      dev_keypad_init
 *@details:    初始化矩阵按键IO口
 *@param[in]   void
 *@param[out]  无
 *@retval:
 */
s32 dev_keypad_init(void)
{
        /*
        c:PF8-PF11    当做输输入
```

```
        r:PF12-PF15  当做输出
        */

        GPIO_InitTypeDef  GPIO_InitStructure;

        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);

        /* r */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
        GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
        GPIO_Init(GPIOF, &GPIO_InitStructure);

        /* c */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9|GPIO_Pin_10|GPIO_Pin_11;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
        GPIO_Init(GPIOF, &GPIO_InitStructure);

        GPIO_SetBits(GPIOF, GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15);

        u8 i;
        for(i = 0; i< KEY_PAD_ROW_NUM; i++)
        {
                KeyPadCtrl[i].dec = 0;
                KeyPadCtrl[i].oldsta = KEYPAD_INIT_STA_MASK;
                KeyPadCtrl[i].newsta = KEYPAD_INIT_STA_MASK;
        }

}
/**
 *@brief:      dev_keypad_scan
 *@details:    按键扫描，在定时器或者任务中定时执行
 *@param[in]   void
 *@param[out]  无
 *@retval:
 */
s32 dev_keypad_scan(void)
{
        u16 ColSta;
        u8 chgbit;
        static u8 scanrow = 0;
        u8 keyvalue;

        if(DevKeypadGd == -1)
                return -1;

        /*读输入的状态，如果不是连续IO，先拼成连续IO*/
        ColSta = GPIO_ReadInputData(GPIOF);
        ColSta = (ColSta>>8)&KEYPAD_INIT_STA_MASK;

        /*记录新状态，新状态必须是连续稳定，否则重新计数*/
        if(ColSta != KeyPadCtrl[scanrow].newsta)
        {
                KeyPadCtrl[scanrow].newsta = ColSta;
                KeyPadCtrl[scanrow].dec = 0;
        }

        /*如新状态与旧状态有变化，进行扫描判断*/
        if(ColSta != KeyPadCtrl[scanrow].oldsta)
        {
                uart_printf(" chg--");
                KeyPadCtrl[scanrow].dec++;
                if(KeyPadCtrl[scanrow].dec >= KEY_PAD_DEC_TIME)//大于防抖次数
                {
                        /*确定有变化*/
```

```
80                          KeyPadCtrl[scanrow].dec = 0;
81                          /*新旧对比，找出变化位*/
82                          chgbit = KeyPadCtrl[scanrow].oldsta^KeyPadCtrl[scanrow].newsta;
83                          uart_printf("row:%d, chage bit:%02x\r\n",scanrow,chgbit);
84
85                          /*根据变化的位，求出变化的按键位置*/
86                          u8 i;
87                          for(i=0;i<KEY_PAD_COL_NUM;i++)
88                          {
89                                  if((chgbit & (0x01<<i))!=0)
90                                  {
91                                          keyvalue =      scanrow*KEY_PAD_COL_NUM+i;
92                                          /*添加通断（按下松开）标志*/
93                                          if((KeyPadCtrl[scanrow].newsta&(0x01<<i)) == 0)
94                                          {
95                                                  uart_printf("press\r\n");
96                                          }
97                                          else
98                                          {
99                                                  uart_printf("rel\r\n");
100                                                 keyvalue += KEYPAD_PR_MASK;
101                                         }
102                                         /**/
103                                         KeyPadBuff[KeyPadBuffW] =keyvalue+1;//+1，调整到1开始，不从0开始
104                                         KeyPadBuffW++;
105                                         if(KeyPadBuffW>=KEYPAD_BUFF_SIZE)
106                                                 KeyPadBuffW = 0;
107                                 }
108                         }
109
110                         KeyPadCtrl[scanrow].oldsta = KeyPadCtrl[scanrow].newsta;
111
112                 }
113         }
114
115         /*将下一行的IO输出0*/
116         scanrow++;
117         if(scanrow >= KEY_PAD_ROW_NUM)
118                 scanrow = 0;
119
120         GPIO_SetBits(GPIOF, GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15);
121
122         switch(scanrow)
123         {
124                 case 0:
125                         GPIO_ResetBits(GPIOF, GPIO_Pin_12);
126                         break;
127                 case 1:
128                         GPIO_ResetBits(GPIOF, GPIO_Pin_13);
129                         break;
130                 case 2:
131                         GPIO_ResetBits(GPIOF, GPIO_Pin_14);
132                         break;
133                 case 3:
134                         GPIO_ResetBits(GPIOF, GPIO_Pin_15);
135                         break;
136         }
137
138 }
```

1. dev_keypad_init初始化函数，完成IO口初始化，并初始化扫描过程用到的变量。
2. dev_keypad_scan就是扫描函数，这个函数可以放到定时器，也可以放在main函数的 while状态轮询，轮询间隔会影响防抖效果。
3. 62/63行，读取4根col的状态，如果你的IO不是连续的，最好在这里拼成连续的，方便下面处理。
   估计有同学会问，怎么没有输出对应的ROW就读COL状态？
   前面我们说过步骤拆分，在此就是将“对应ROW输出0电平”，“读COL”，拆分为两个步骤。上一次退出扫描函数的时候，

将对应ROW输出0，等下一次进入扫描的时候才读取COL。为什么？
我们一直强调，CPU很快，如果你对应的ROW输出0，然后立马读COL，间隔很短，**IO口电平变化可是要时间的**。
有同学又会说了，我知道，IO口电平变化需要可能几百纳秒，那我输出0后，延时1us，再读，应该可以读到真正电平了吧？硬延时1us，通常也是可以接受的。
是的，很多情况下我们也可以这么做，但是，会有意外。
原因就是，电平变化时间跟CPU本身性能有关外，还跟外部硬件有关，例如PCB板材，按键材料，PCB走线，甚至是环境温度湿度都会影响电平变化。
当年因为换了PCB板厂，发到东北的产品按键按键无效的事情我不会跟你说的。

> IO电平变化时间由ns上升到us。
> 我们在上一次就将对应ROW输出0，下一次轮询的时候读COL状态，间隔通常是ms级的，可以避免上面问题。

4. 66-70行，意思就是不仅仅是跟上一次的稳定状态不一样（有变化），而且需要在防抖过程中多次读取的状态一样，不一样就重新开始去抖动计数。
5. 73行到113行就是去抖动跟按键识别。
6. 73-78，状态变化，而且连续变化次数达到去抖动计数，我们就认为是一个稳定的变化了。
7. 82，新旧状态异或，找出变化的位。（这样处理的好处就是，当同一行的两个按键同时按下时，我们都可以识别）
8. 87 行这个for循环的意思就是，每一个col的变化我们都要识别。
9. 91行，识别到变化按键的物理位置编码。
   我们提供的是位置编码，能不能提供功能编码？例如，第一个键是按键1，最后一个键是确认。可以，不建议，而且是非常不建议。
   功能是谁的定义？谁关心？应用关心，那就让它去管，反正我按键扫描就告诉你，第一个按键按下，至于是1还是确认，你自己定。
   如果扫描给出的键值是功能键值，那就麻烦了，因为这个按键是什么功能，只要客户说一声，然后，改一下丝印。怎么的？你还想改底层驱动？
10. 93-101行，判断是按下还是松开，并在键值上添加标志位。
11. 103-106，将键值填入缓冲。关于这个缓冲，又是一个驱动设计的要点。
    很多人写的按键扫描，都是直接通过return返回键值。我就问你，当一个系统比较复杂的时候，合适吗？
    明显不合适，假如一个系统，有两个菜单，每个菜单都需要按键功能，难道这两个菜单都调用扫描函数？
    这种现象叫什么？叫做耦合：两个不同模块扯在一起。
    耦合，是大忌；解耦合才是正道。哈哈。
    解耦合的一个手段就是数据缓冲。
    扫描模块扫到键值，丢入缓冲，至于谁要这个按键？I Don't care!
    再提供一个函数用于读缓冲内的键值。谁想要谁读，按键扫描程序自己运行到天荒地老。
12. 115-135行，就是将下一ROW输出0。

一个按键矩阵按键扫描流程就是这样。
怎么用这个按键扫描呢？
这样用：

```
main()
{
                dev_keypad_init();
                dev_keypad_open();
                其他初始化


                while(1)
                {
                        dev_keypad_scan();
                        其他轮询任务
                        ...。。。
                        应用处理
                        dev_keypad_read(&key,1);
                        读到按键就处理。
                }
}
```

问题

这个矩阵按键的硬件设计是有缺陷的，会产生幽灵键。
当你按下124三个按键，按键5即使没按下，程序扫描出来的结果也是按下。
要解决这个问题，需要在所有按键上增加二极管。
通常我们会放弃三键按下这种情况，通过软件判断，将三件按下时产生幽灵键的情况抛弃。
具体可以参考文档《矩阵式键盘的先天缺陷与解决方案.docx》

更多

1. 其他更多信息请参考附件源码。
   这个源码是在STM32F407芯片上的。
   官网www.wujique.com，最新资料源码可从官网下载。
2. 如果你觉得这个文档或是源码有用，可以发一封邮件给我表示感谢！
   让我知道时间没白费。
   或者对这份代码有优化。
   请联系：code@wujique.com
3. 如果你是一个在校学生，看了本文档跟源码，觉得得到了收益，欢迎打赏1元钱，我们就知道，写的东西有人看。
   如果你是一个新手码农，通过本文档跟源码，学会了按键扫描，并且用在了自己的工作上，欢迎打赏n元，我们就知道，写的代码真有人用。
   如果你是一个资深码农，觉得文档讲的不错，可以给小弟们学习学习；代码也不错，可以在项目中使用，欢迎打赏nn元，“我CAO, 我们还是有点水平的嘛”。
   (可发送邮件到code@wujique.com获取最新二维码，支付宝用户名“*文婷”）。

---

end