

4×4 矩阵键盘扫描及消抖模块

简介：

矩阵键盘相较于传统的拨码开关能为用户提供更多的输入，比较适合需要多个用户指令输入的学生项目。4×4 矩阵键盘为一般机械结构，当用户按下按钮时，其不可避免地会产生电平抖动，因此在矩阵键盘的使用中，消抖操作是重点。

实现功能：

此模块实现 4×4 矩阵键盘的扫描及消抖操作，最终将用户按下的按钮信息通过 16bit 的独热码输出。输出的独热码可以通过转码（16bit 位宽较长，可以根据需要缩短，也可不做处理）或直接作为指令对其他模块进行控制。

实现原理：

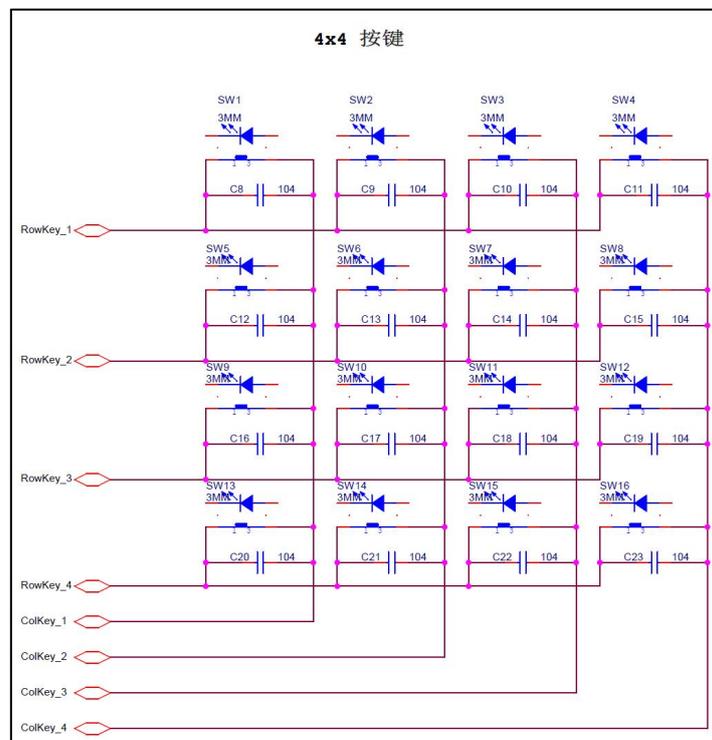


图 1：矩阵键盘硬件原理图

4×4 矩阵键盘只需要用到 8 个 I/O 口，通常将矩阵键盘的列线 col 拉高（设置为上拉电阻），在没有按键按下的情况下全部输出为高电平。若给某一行输入低电平并且按下了该行的某一个按键，那么该按键对应的列线就会被拉低。举例说明：给上图的 RowKey2 输入低电平，读取键盘列信号（注意 ColKey_4 为 4bit 列信号中的最高位，ColKey_1 为 4bit 列信号中的最低位）输出为 1011，则说明第二行和第三列对应的按键（SW7）被按下了。

键盘扫描模块中的几个要点作如下说明：第一，键扫的频率一般要高于 20Hz，过低会导致用户的“按下”操作扫描不到，但频率设置也不是越高越好，因为太高会浪费 FPGA 资源，此模块扫描频率设置在 13KHz 左右。第二，行扫描的方式



是通过一个 4 位宽的移位寄存器作移位操作实现的，如下：

```
row <= {row[2:0], row[3]}; //行扫描
```

其中，row（行电平信号）在复位时会赋初值，可选定任意一行初始为低电平，如 1110（图 1 中的 RowKey_1 初始为低电平），0111（图 1 中的 RowKey_4 初始为低电平）等都是可以的。第三，对于扫描之后得到的 16bit 的按键数据，没接触过矩阵键盘的读者可能会有一些的困惑，这里结合模块里面的源码作以下详细解释：

```
always@(posedge scan_clk2 or negedge rstn)begin
  if(!rstn)
    key <= 16'h0;
  else case(row)
    4'b1110 : key[3:0]   <= col   ; //列输入
    4'b1101 : key[7:4]   <= col   ;
    4'b1011 : key[11:8]  <= col   ;
    4'b0111 : key[15:12] <= col   ;
    default : key       <= 16'h0;
  endcase
end
```

现假设用户按下“9”键（图 1 中的 SW10，矩阵键盘硬件实物一般从 0 开始标号），很明显我们要扫描到“9”键必须是在第 3 行（RowKey_3）被拉低的时候。因此 key 这个寄存器赋值的过程为：key[0000_0000_0000_0000]（复位）→ key[0000_0000_0000_1111]（扫描到第 1 行）→ key[0000_0000_1111_1111]（扫描到第 2 行）→ key[0000_1101_1111_1111]（扫描到第 3 行）→ key[1111_1101_1111_1111]（扫描到第 4 行）（注意上述 key[****]的写法并不涉及任何程序语法，笔者只是想单纯表达随着 row 的变化，key 的值变化成什么了）。只要对应着“若给某一行输入低电平并且按下了该行的某一个按键，那么该按键对应的列线就会被拉低”的基本原理就很好理解了。

以上是对于矩阵扫描模块的说明，下面具体介绍一下键盘消抖的实现：

由于按键抖动的时间一般为 10~20 毫秒，而按键稳定的时间一般为数百毫秒，所以可以设置多个寄存器，延时读取按键的电平信号（具体操作是利用时钟分频制造一个周期为 20ms 的时钟，然后在这个时钟域下将 key_in 打 3 拍，也可以采用模块源码里面的方法：用一个计数器循环计数，每计数到 20ms 才向相应寄存器赋值），再综合判断按键是否按下。

消抖后的按键信息存于 key_deb，这里我们对这个 assign 语句做一下分析：

```
assign key_deb = (~key_reg0 & ~key_reg1 & ~key_reg2) | (~key_reg0 & ~key_reg1 & key_reg2);
```

首先，有两种不同的情况会被系统认为是用户按下了按键，结合 assign 语句的前半段（位或符号前面），第一种情况是三个寄存器都不发生抖动，这显然是按下的情况；结合 assign 语句的后半段（位或符号后面），按照机械键盘“抖动的时间为 10~20 毫秒，稳定的时间为数百毫秒”的特点，60ms 前的按键数值最可能发生抖动，因为此寄存器里面寄存的数据离“按下”操作时间最短。并且我们认为 20ms 前和 40ms 前的数据不会再发生抖动了。



我们还是以用户按下“9”键为例，具体说明消抖后的数据的由来：若“9”键按下，则进入到消抖模块的key_in的值为1111_1101_1111_1111（前面已经详细阐述过扫描后的按键数据怎么产生的，这里不赘述），在无任何抖动的情况下3个寄存器取反后按键数据都为0000_0010_0000_0000，输出结果为16进制的独热码0200。若60ms前的寄存器发生抖动，则按键数据(1111_1101_1111_1111)里面唯一的0也因为抖动变成了1，因此assign语句的后半段计算结果仍然为16进制的0200。

接口说明：

顶层：

Ports:

Name	Inout	Data Type	Datasize	Function
clk	input	wire	1	系统时钟
rstn	input	wire	1	系统复位
col	input	wire	4	列电平输入
row	output	wire	4	行电平输出
key_deb	output	wire	16	消抖后的按键

键盘扫描模块：

Ports:

Name	Inout	Data Type	Datasize	Function
clk	input	wire	1	系统时钟
rstn	input	wire	1	系统复位
col	input	wire	4	列电平输入
row	output	reg	4	行电平输出
key	output	reg	16	扫描到的按键信息

消抖模块：

Ports:

Name	Inout	Data Type	Datasize	Function
clk	input	wire	1	系统时钟
rstn	input	wire	1	系统复位
key_in	input	wire	16	扫描后的按键信息
key_deb	output	wire	16	消抖后的按键信息



模块引脚说明见表格，具体实现已经详细阐述过了，这里不再一一说明每个引脚的作用。

仿真：

此仿真假定键盘上“0”这个按钮按下，输出独热码结果为16进制的1：

