FPGA 矩阵键盘驱动设计与验证

（本实验针对小梅哥团队出品的任意一款 FPGA 开发板均可通用）

本文由西安网友张昭贡献，小梅哥补充修订，特此感谢

实验简介

本章我们主要是来实现用 verilog 语言实现矩阵键盘的驱动。提到矩阵键盘，许多读者应该不会陌生，因为学过单片机、ARM 的人都应该知道并用 C 语言做过矩阵键盘的驱动。本章作者将用 verilog 语言来写一个矩阵键盘(4*4)的驱动，并用 LED 灯的状态指示按键的状态。课后会留有和本章内容有关的作业，希望所有读者能独立完成作业。接下来就让我们一块走进今天的学习内容“矩阵键盘学习之旅”。

矩阵键盘诞生的背景

相信许多人经常用到矩阵键盘但是却不知道“它”的由来。由于最早的 MCU(即单片机)其 IO 口相对较少，而且用到按键过多的话，就会占用过多的 IO。人们为了解决这个问题就引入了“矩阵键盘”。在矩阵键盘中，每条行线和列线在交叉处都不是直接连同，而是通过一个按键直接相连，这样以来一个 4*4 的矩阵键盘只需要 8 根控制线就可以完成 16 个按键的控制。下面就是常见的两类 4*4 矩阵键盘：

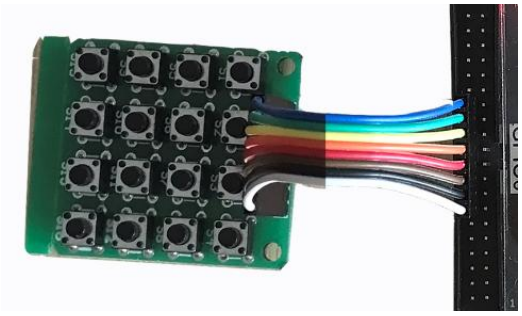


图 1.PCBA 型矩阵键盘

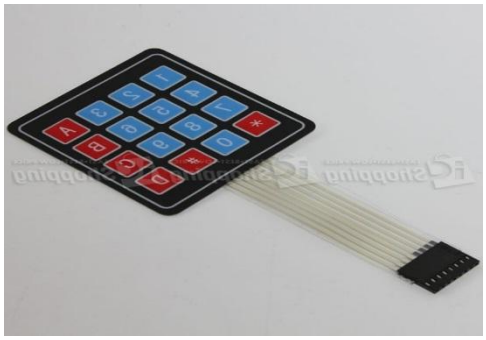


图 2.薄膜型矩阵键盘

两种矩阵键盘虽然形式不同，但是原理和功能完全相同，使用方式也完全相同，不过 PCBA 形式的那种手感相对要好一些。

矩阵键盘工作原理

要说起矩阵键盘是如何工作的？就得从它的硬件原理图来理解它工作机理。下图为一个典型的矩阵键盘应用电路：

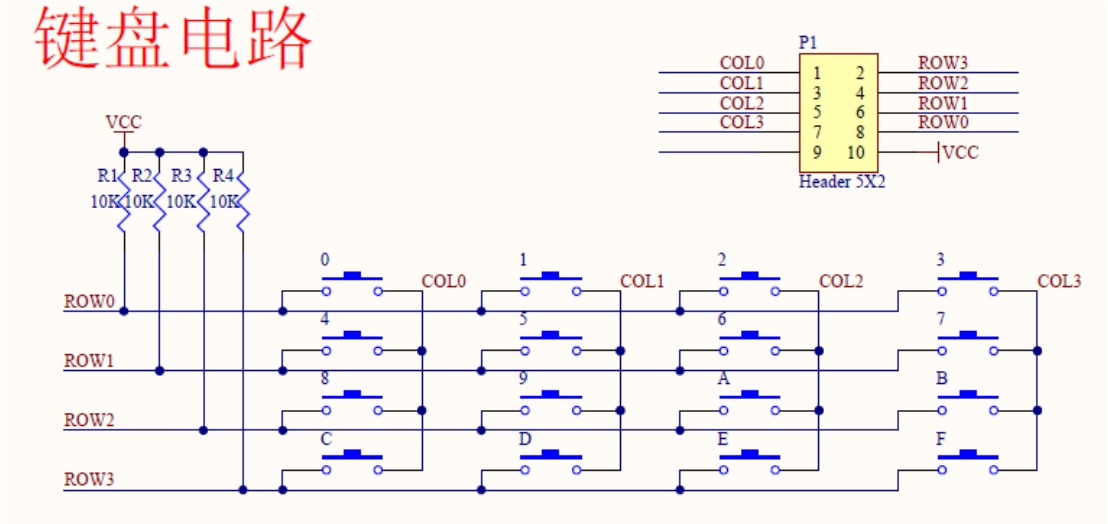
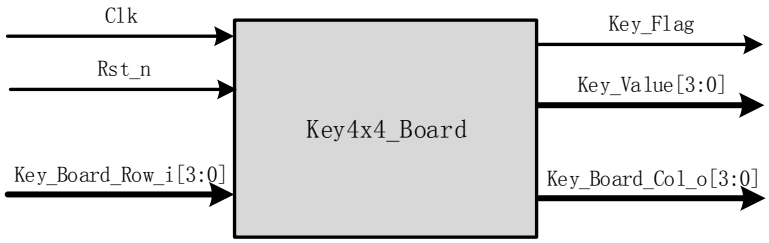


图　矩阵键盘典型电路图

矩阵键盘有两组，共 8 根信号线，其中 COL 将每一列的四个按键的一端连接起来，而 ROW 则将每一行的 4 个按键的一端连接起来，通过 4 行 4 列的 8 根信号线，总共能够管理 16 个按键，这也是矩阵键盘相较于普通独立按键的优势所在，即用较少的 IO 得到较多的按键输入。

通常我们检测矩阵键盘中某一按键是否被按下，采用的方法是列扫描法。图中一共有 8 条控制线，4 条行控制线(ROW),4 条列控制线（COL），4 根行控制线（ROW）被通过上拉电阻拉到了高电平，因此，如果没有按键被按下的情况下，使用 FPGA 的 IO 坚持 ROW 信号上的电平，应该默认都是高电平。那我们如何去检测某一个按键被按下，并且还能找到其具体的位置呢？思路其实并不复杂，假如我们让 COL0=0，然后去读取 ROW 的值，如果 4 个 ROW 信号的电平全部为高，则表明当前列并无按键按下，则切换到扫描下一列，再去读取 4 个 ROW 信号的值，根据读到的 ROW 值判断当前是哪一个按键被按下。例如当扫描第三列的时候，即 COL=4’b1011 时，检测到 ROW=4’b1011,则说明“A”被按下。

总体框架图



端口列表如下：

端口类型	端口名	描述
Input	Clk	系统时钟 50MHz
Input	Rst_n	系统复位，低有效
Input	Key_Board_Row_i	矩阵键盘行控制线，按键未按下为高电平
Output	Key_Board_Col_o	矩阵键盘列控制线，驱动列信号为低，实现

		按键状态扫描，
Output	Key_Value	输出键盘键值
Output	Key_Flag	按键检查成功标志信号，每当按键检测成功，产生一个时钟周期的高脉冲

状态转移图

现在作者就来给大家讲解今天的重点，状态转移图的设计。可能会有读者说，这个东西有什么用处？给大家举个例子，“如何去建筑一座楼房”。在建造前肯定是工程师设计建筑的建造图纸，然后工人们按照图纸进行分工建造房屋。今天作者讲的状态转移图就像是工程师设计图纸没有图纸，直接编代码，肯定会来回修改多次。下面进入正题：

矩阵键盘的状态转移图如下所示：

IDLE：主要是用来检测是否有按键被按下，无按键按下时，行控制线 Key_Board_Row_i 等于 15，有按键按下时 Key_Board_Row_i 不等于 15,则进入 P_FILTER 状态;

P_FILTER: 主要是进行按键消抖检测，判断是否有按键真实被按下，如果有按键真实被按下则跳转至下一个状态 READ_ROW_P 获取行状态的值，否则不跳转;

READ_ROW_P：读取行状态并将存入的寄存器中 Key_Board_Row_r(ps:存入寄存器是因为按键按下的时间比较短暂，所以要将其状态缓存起来)，并将第 0 列拉低，进入 SCAN_C0 状态进行行扫描;

SCAN_C0: 进行列扫描，来判断该列是否有按键被按下，判断行输入是否等于 15，如果不等于则说明该列有按键被按下，否则将第 1 列拉低，状态跳转到 SCAN_C1 状态进行行扫描;

SCAN_C1: 进行列扫描，来判断该列是否有按键被按下，判断行输入是否等于 15，如果不等于则说明该列有按键被按下，否则将第 2 列拉低，状态跳转到 SCAN_C3 状态进行行扫描;

SCAN_C2: 进行列扫描，来判断该列是否有按键被按下，判断行输入是否等于 15，如果不等于则说明该列有按键被按下，否则将第 3 列拉低，状态跳转到 SCAN_C3 状态进行行扫描;

SCAN_C3: 进行列扫描，来判断该列是否有按键被按下，判断行输入是否等于 15，如果不等于则说明该列有按键被按下，否则状态跳转到 PRESS_RESULT 状态输出扫描结果;

PRESS_RESULT: 输出扫描结果。这块给大家教一种比较特别的方法来确定整个键盘上只有一个按键被按下：4 位行控制线如果 4 位行线值加起来如果等于 3 则说明有一行被按下:

(Key_Board_Row_i[0]+ Key_Board_Row_i[1]+ Key_Board_Row_i[2]+ Key_Board_Row_i[3]=3),同样如果 4 位列控制线值加起来如果等于 1 则说明有一列被按下:

(Col_Tmp [0]+ Col_Tmp [1]+ Col_Tmp [2]+ Col_Tmp [3]=1)

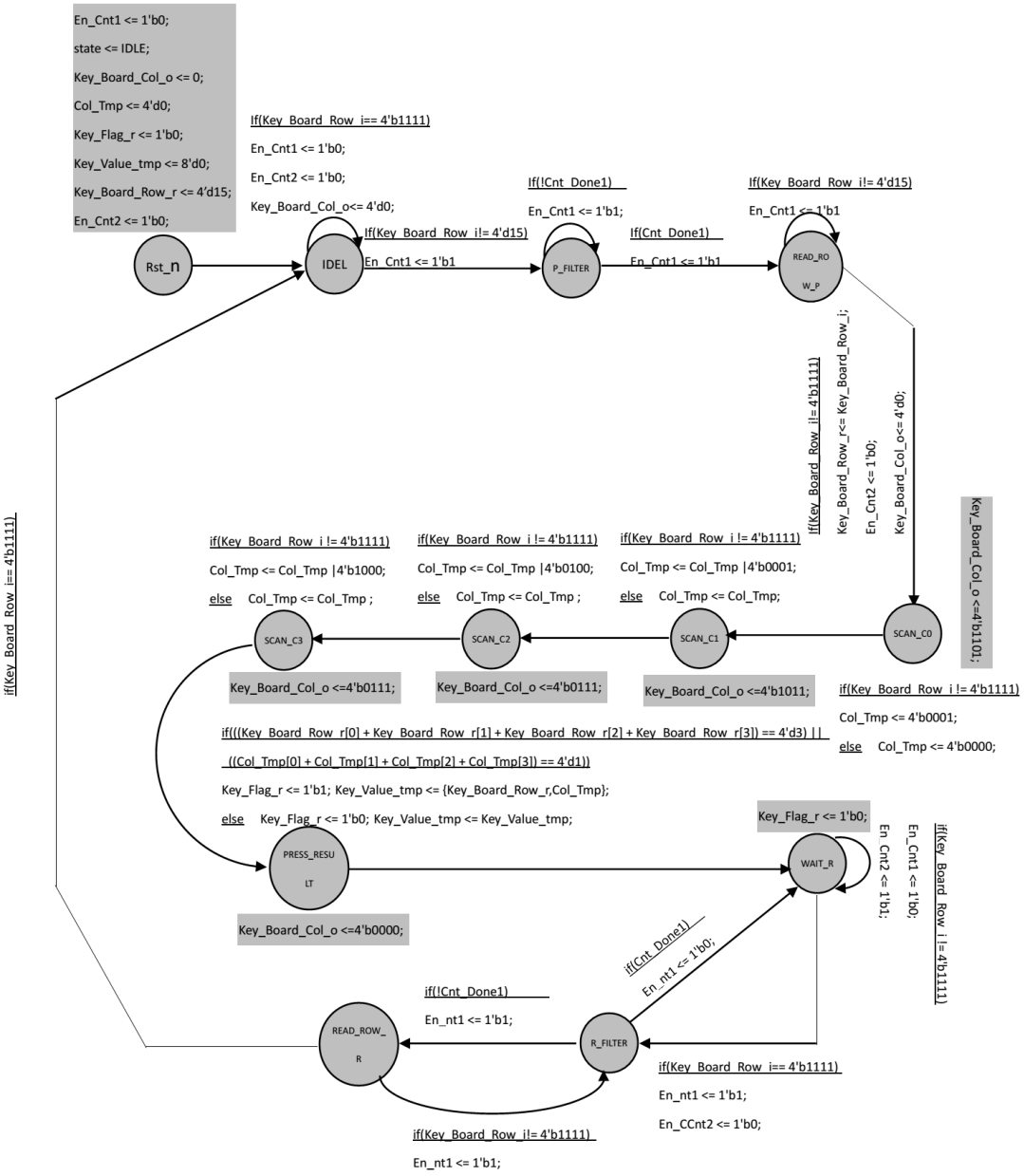
通过这两个参数的限定，确保了只有一个按键被按下。如果不满足，则不产生检测成功标志信号。状态跳转到 WAIT_R 等待按键释放状态

WAIT_R: 等待按键释放，如果 Key_Board_Row_i=15 则说明按键被释放，状态跳转到释放消抖状态 R_FILTER

R_FILTER：检测按键是否完全被释放如果释放则跳转到 READ_ROW_R 状态读取按键行输入状态，读取行的状态。

READ_ROW_R：读取按键行输入状态。

通过这些状态，我们把一个按键按下，到释放的过程完美的进行了一次检测，最终将输出结果用 LED 灯的状态表示。接下来，就是照着下面的状态转移图敲写程序。这下读者们可以自由选择语言编程，Verilog 或者 VHDL 随意选，然后去盖起自己的矩阵键盘“大厦”。



矩阵键盘扫描逻辑代码

下面是整个矩阵键盘驱动的程序部分：

```
module
Key4x4_Board(Clk,Rst_n,Key_Board_Row_i,Key_Board_Col_o,Key_Flag,Key_Value);


    input Clk;  //模块工作时钟，默认为 50M
    input Rst_n;    //模块复位，低电平复位


    input [3:0]Key_Board_Row_i; //矩阵键盘行输入
    output reg [3:0]Key_Board_Col_o;    //矩阵键盘列输出
```

```
    output reg Key_Flag;    //按键检查成功标志信号，每次检测成功输出一个时钟周期的高脉冲
    output reg[3:0]Key_Value;   //键值

    reg [19:0] counter1;    //延时计数器
    reg En_Cnt1;    //延时计数器使能控制信号，当进入延时消抖阶段时为高
    reg Cnt_Done1;  //延时计数器延时完成（计满 20ms）信号

    reg [25:0] counter2;    //连按间隔计数器
    reg En_Cnt2;    //连按间隔计数器使能控制信号，当进入连按间隔阶段时为高
    reg Cnt_Done2;  //连按间隔计数器延时完成（计满 200ms）信号

    reg [10:0]state;    //状态寄存器，存储系统的状态
    reg [3:0]Key_Board_Row_r;//矩阵键盘行输入寄存器，存储行状态
    reg [3:0]Col_Tmp;   //
    reg [7:0]Key_Value_tmp;//矩阵键盘扫描结果
    reg Key_Flag_r; //按键检查成功标志信号

    localparam
        IDLE            = 11'b00000000001,  //空闲态，无按键按下
        P_FILTER        = 11'b00000000010,  //按下消抖状态
        READ_ROW_P      = 11'b00000000100,  //读取按下时矩阵键盘行状态
        SCAN_C0         = 11'b00000001000,  //扫描矩阵键盘第 0 列（Col0）
        SCAN_C1         = 11'b00000010000,  //扫描矩阵键盘第 1 列（Col1）
        SCAN_C2         = 11'b00000100000,  //扫描矩阵键盘第 2 列（Col2）
        SCAN_C3         = 11'b00001000000,  //扫描矩阵键盘第 3 列（Col3）
        PRESS_RESULT    = 11'b00010000000,  //获得扫描结果
        WAIT_R          = 11'b00100000000,  //等待释放信号到来
        R_FILTER        = 11'b01000000000,  //释放消抖
        READ_ROW_R      = 11'b10000000000;  //读取释放时矩阵键盘行状态

//  延时计数器，计数以延时 20ms
    always@(posedge Clk or negedge Rst_n)
    if(!Rst_n)
        counter1 <= 20'd0;
    else if(En_Cnt1)begin
        if(counter1 == 20'd999999)
            counter1 <= 20'd0;
        else
            counter1 <= counter1 + 1'b1;
    end
    else
        counter1 <= 20'd0;
```

```
//产生延时完成标志信号，（当延时 20ms 时产生一个计数完成信号 Cnt_Done1）
    always@(posedge Clk or negedge Rst_n)
    if(!Rst_n)
        Cnt_Done1 <= 1'b0;
    else if(counter1 == 20'd999999)
        Cnt_Done1 <= 1'b1;
    else
        Cnt_Done1 <= 1'b0;


//  连按间隔计数器，计数以延时 20ms
    always@(posedge Clk or negedge Rst_n)
    if(!Rst_n)
        counter2 <= 26'd49_999_999; //初始时设定间隔为 1s
    else if(En_Cnt2)begin
        if(counter2 == 26'd0)
            counter2 <= 26'd999999; //启动间隔后每 20ms 发出一次间隔标志
        else
            counter2 <= counter2 - 1'b1;
    end
    else
        counter2 <= 26'd49_999_999;

//产生连按间隔完成标志信号，（当延时 20ms 时产生一个计数完成信号 Cnt_Done2）
    always@(posedge Clk or negedge Rst_n)
    if(!Rst_n)
        Cnt_Done2 <= 1'b0;
    else if(counter2 == 26'd0)
        Cnt_Done2 <= 1'b1;
    else
        Cnt_Done2 <= 1'b0;

//矩阵键盘扫描状态机
    always@(posedge Clk or negedge Rst_n)
    if(!Rst_n)begin
        En_Cnt1 <= 1'b0;
        state <= IDLE;

//默认需要让列输出为 0，这样，当有按键按下时，行列接通，行输入才能读到有低电平
        Key_Board_Col_o <= 4'b0000;
        Col_Tmp <= 4'd0;
        Key_Flag_r <= 1'b0;
        Key_Value_tmp <= 8'd0;
        Key_Board_Row_r <= 4'b1111;
```

```
    end
    else begin
        case(state)
//空闲态，持续判断是否有按键被按下（按键按下时 Key_Board_Row_i 将不为全 1）
        IDLE:
                if(Key_Board_Row_i != 4'b1111)begin //有按键按下
                    En_Cnt1 <= 1'b1;    //启动延时计数器
                    state <= P_FILTER;  //跳转到按下消抖状态
                end
                else begin
                    En_Cnt1 <= 1'b0;
                    Key_Board_Col_o <= 4'b0000;
                    state <= IDLE;
                end

            P_FILTER:   //按下消抖状态
                if(Cnt_Done1) begin //20ms 延时完成
                    En_Cnt1 <= 1'b0;    //停止延时计数器
                    state <= READ_ROW_P;    //跳转到读取矩阵键盘行输入状态
                end
                else begin  //20ms 延时还没有完成
                    En_Cnt1 <= 1'b1;    //保持延时计数器继续计数
                    state <= P_FILTER;
                end

            READ_ROW_P: //读取矩阵键盘行输入状态
                if(Key_Board_Row_i != 4'b1111)begin //行输入不全为 1，表明确实有按键按下
                    Key_Board_Row_r <= Key_Board_Row_i; //将当期行输入状态存入行状态寄存
器
                    state <= SCAN_C0;       //跳转到扫描矩阵键盘第 0 列（Col0）状态
                    Key_Board_Col_o <= 4'b1110; //将第 0 列列输出置 0
                end
                else begin  //若行输入恢复为了全 1，则表明此次为抖动，不再执行扫描，跳回空闲
态
                    state <= IDLE;
                    Key_Board_Col_o <= 4'b0000;
                end

            SCAN_C0:    //扫描矩阵键盘第 0 列（Col0）状态
                begin
                    state <= SCAN_C1;   //下一个状态为扫描矩阵键盘第 1 列（Col1）状态
                    Key_Board_Col_o <= 4'b1101; //扫描第 1 列对应的行输出为 1101
                    if(Key_Board_Row_i != 4'b1111)  //行输入不全为 1，表明为当前列（第 0
列）有按键按下
```

```
                    Col_Tmp <= 4'b0001;//将列值存入列寄存器中
                else
                    Col_Tmp <= 4'b0000;
            end

        SCAN_C1:    //扫描矩阵键盘第 1 列（Col1）状态
            begin
                state <= SCAN_C2;   //下一个状态为扫描矩阵键盘第 2 列（Col2）状态
                Key_Board_Col_o <= 4'b1011; //扫描第 1 列对应的行输出为 1011
                if(Key_Board_Row_i != 4'b1111)  //行输入不全为 1，表明为当前列（第 1
列）有按键按下
                    Col_Tmp <= Col_Tmp | 4'b0010;//将列值存入列寄存器中
                else
                    Col_Tmp <= Col_Tmp;
            end

        SCAN_C2:    //扫描矩阵键盘第 2 列（Col2）状态
            begin
                state <= SCAN_C3;   //下一个状态为扫描矩阵键盘第 3 列（Col3）状态
                Key_Board_Col_o <= 4'b0111;//扫描第 1 列对应的行输出为 0111
                if(Key_Board_Row_i != 4'b1111)//行输入不全为 1，表明为当前列（第 2
列）有按键按下
                    Col_Tmp <= Col_Tmp | 4'b0100;//将列值存入列寄存器中
                else
                    Col_Tmp <= Col_Tmp;
            end

        SCAN_C3:    //扫描矩阵键盘第 3 列（Col3）状态
            begin
                state <= PRESS_RESULT;  //下一个状态为得到扫描结果状态
                if(Key_Board_Row_i != 4'b1111)//行输入不全为 1，表明为当前列（第 3
列）有按键按下
                    Col_Tmp <= Col_Tmp | 4'b1000;//将列值存入列寄存器中
                else
                    Col_Tmp <= Col_Tmp;
            end

        PRESS_RESULT:   //得到扫描结果状态
            begin
                state <= WAIT_R;    //下一个状态为等待按键释放状态

        //让列输出全 0，这样，当有按键按下时，行列接通，行输入才能读到有低电平
                Key_Board_Col_o <= 4'b0000;
```

```
                    /*4 位行输入值相加为 3，表明有且只有一个行输入为 0，既保证只有一行中有按键被
按下
                    *4 位列扫描结果相加为 1，表明有且只有列中有按键被按下，通过这两个条件保证了
一次
                    *按键中只有一个按键被按下时才检测有效*/
                    if(((Key_Board_Row_r[0] + Key_Board_Row_r[1] +
                           Key_Board_Row_r[2] + Key_Board_Row_r[3]) == 4'd3) &&
                           ((Col_Tmp[0] + Col_Tmp[1] + Col_Tmp[2] + Col_Tmp[3]) ==
4'd1))begin
                        Key_Flag_r <= 1'b1;//产生检测成功标志信号
                        Key_Value_tmp <= {Key_Board_Row_r,Col_Tmp};//将行列结果组合得
到按键扫描结果
                    end
                    else begin  //若不满足只有一个按键被按下，则不产生检测成功标志信号
                        Key_Flag_r <= 1'b0;
                        Key_Value_tmp <= Key_Value_tmp;
                    end
                end

            WAIT_R://等待按键释放状态
                begin
                    Key_Flag_r <= 1'b0;//清零检测成功标志信号
                    if(Key_Board_Row_i == 4'b1111)begin//无按键按下,表明按键被释放
                        En_Cnt1 <= 1'b1;//启动延时计数器
                        state <= R_FILTER;//进入释放消抖状态
                        En_Cnt2 <= 1'b0;    //停止连按间隔计数器
                    end
                    else begin
                        state <= WAIT_R;
                        En_Cnt1 <= 1'b0;
                        En_Cnt2 <= 1'b1;//启动连按间隔计数器
                    end
                end

            R_FILTER:   //释放消抖状态
                    if(Cnt_Done1) begin //延时时间到
                        En_Cnt1 <= 1'b0;
                        state <= READ_ROW_R;//跳转入读取按键行输入状态（按键释放阶段）
                    end
                    else begin
                        En_Cnt1 <= 1'b1;
                        state <= R_FILTER;
                    end
```

```
            READ_ROW_R://读取按键行输入状态（按键释放阶段）
                if(Key_Board_Row_i == 4'b1111)  //无按键按下，表明确实已经稳定的释放了
                    state <= IDLE;  //跳转回空闲状态
                else begin  //否则表明此次释放为抖动，回到释放消抖状态继续等待
                    En_Cnt1 <= 1'b1;
                    state <= R_FILTER;
                end
            default:;
        endcase
    end

//将扫描结果译码输出
    always@(posedge Clk or negedge Rst_n)
    if(!Rst_n)begin
        Key_Flag <= 1'd0;
        Key_Value <= 4'd0;
    end
    else begin

    //按键检查成功输出标志为扫描成功与连按间隔的逻辑或
        Key_Flag <= Key_Flag_r | Cnt_Done2;
        case(Key_Value_tmp)
            8'b1110_0001 : Key_Value <= 4'd0;   //第 0 行，第 0 列
            8'b1110_0010 :  Key_Value <= 4'd1;  //第 0 行，第 1 列
            8'b1110_0100 : Key_Value <= 4'd2;   //第 0 行，第 2 列
            8'b1110_1000 : Key_Value <= 4'd3;   //第 0 行，第 3 列

            8'b1101_0001 : Key_Value <= 4'd4;   //第 1 行，第 0 列
            8'b1101_0010 : Key_Value <= 4'd5;   //第 1 行，第 1 列
            8'b1101_0100 : Key_Value <= 4'd6;   //第 1 行，第 2 列
            8'b1101_1000 : Key_Value <= 4'd7;   //第 1 行，第 3 列

            8'b1011_0001 : Key_Value <= 4'd8;   //第 2 行，第 0 列
            8'b1011_0010 : Key_Value <= 4'd9;   //第 2 行，第 1 列
            8'b1011_0100 : Key_Value <= 4'd10;  //第 2 行，第 2 列
            8'b1011_1000 : Key_Value <= 4'd11;  //第 2 行，第 3 列

            8'b0111_0001 : Key_Value <= 4'd12;  //第 3 行，第 0 列
            8'b0111_0010 : Key_Value <= 4'd13;  //第 3 行，第 1 列
            8'b0111_0100 : Key_Value <= 4'd14;  //第 3 行，第 2 列
            8'b0111_1000 : Key_Value <= 4'd15;  //第 3 行，第 3 列
            default:begin Key_Value <= Key_Value;Key_Flag <= Key_Flag;end
        endcase
    end
```

```
endmodule
```

以上就是程序部分，主要部分状态机部分的代码完全是按照，状态及设计部分的状态转移图来实现的。可能刚开始让大家，先设计状态转移图后敲代码比较难，但是还是希望大家从刚开始就养成一个好的习惯，这样日积月累自己独立思考能力也就上去了。

仿真验证

初学的学员，不喜欢写测试文件、害怕写测试文件、忽视仿真的重要性，而是喜欢直接下板进行板及验证。以下问题普遍存在于各大学员中：

1. 刚开始学 FPGA，按照单片机的设计思路，过分追求下板验证效果，不懂得仿真验证才是 FPGA 设计中最为关键的环节；
2. 不会写测试脚本文件，就算勉强能够运行起来仿真，对于仿真的结果波形也不懂得如何分析，不会通过波形去发现潜在的问题。
3. 眼高手低，简单的测试脚本写起来感觉没意思纯属是在浪费时间，总想着到了复杂的系统时候再说，但是复杂的系统要考虑的问题太多，没有经过简单系统的练习，复杂系统就不知道如何去编写 testbench 文件了，也不知道查看仿真波形的技巧，困难重重，最终只得放弃。

在学习党的过程中，直接下板看现象，忽视仿真，后果的严重性暂时不明显，最多也就耽误时间，学不会而已。但是换一个思路，如果我们是在设计航天飞机，那么你设计的航天飞机难道也要每改一次就让其飞一飞试试，不行再改吗？一次航天的升空成功，是经过了千千万万次理论论证和模拟实验的结果，所以，仿真验证作为理论验证和协助分析查找问题的工具，是每一个学习和使用 fpga 的工程师必备技能。本例中，我们将花费一定的篇幅讲解测试脚本的编写，对于矩阵键盘，我们采用 testbench + 矩阵键盘仿真模型的方式进行仿真。下图为测试脚本和被测试设计的关系图。

由上图知道本节内容的测试脚本由两部分组成：被测试部分和仿真模型；被测试部分我们已将前面已经讲完了，下面给大家讲解仿真模型如何编写.

```
`timescale 1ns/1ns
//矩阵键盘的仿真模型
module Key_Board_model(Key_Col,Key_Row);

    input [3:0] Key_Col;        //矩阵键盘列输入
```

```
    output reg [3:0]Key_Row;    //矩阵键盘行输出

    reg [15:0] myrand;
    reg [3:0] Key_Row_r;        //行寄存器
    reg key_row_sel;            //行选通信号
    reg [1:0]now_col,now_row;   //当前行与当前列

    initial begin               //进行初始化
        now_col = 0;
        now_row = 0;
        key_row_sel = 0;
        myrand = 0;
    end

    initial begin
        Key_Row_r = 4'b1111;     //初始状态行全为 1
        #50000000;
        press_key(0, 0);         //第一行
        press_key(0, 1);
        press_key(0,2);
        press_key(0,3);

        press_key(1, 0);         //第二行
        press_key(1, 1);
        press_key(1,2);
        press_key(1,3);

        press_key(2, 0);         //第三行
        press_key(2, 1);
        press_key(2,2);
        press_key(2,3);

        press_key(3, 0);         //第四行
        press_key(3, 1);
        press_key(3,2);
        press_key(3,3);
        $stop;
    end

    task press_key;
        input [1:0]row,col;
        begin
            key_row_sel = 0;
            Key_Row_r = 4'b1111;
```

```
            Key_Row_r[row] = 0;
            now_row = row;
            repeat(20)begin     //重复 20 次，随机产生按键按下时 20 个不同
的行状态
                myrand = {$random} % 65536;
                #myrand Key_Row_r[row] = ~Key_Row_r[row];
            end
            key_row_sel = 1;    //将行选择信号设置为有效状态
            now_col = col;     //获取当前列状态
            #22000000;
            key_row_sel = 0;
            Key_Row_r = 4'b1111;
            repeat(20)begin     //重复 20 次，随机产生按键松开时 20 个不同
行状态
                myrand = {$random} % 65536;
                #myrand Key_Row_r[row] = ~Key_Row_r[row];
            end
            Key_Row_r = 4'b1111;
            #22000000;
        end
    endtask

    always@(*)
    if(key_row_sel)          //行选择信号有效
        case(now_row)        //根据当前行输出，键盘的行信号
            2'd0:Key_Row = {1'b1,1'b1,1'b1,Key_Col[now_col]};
            2'd1:Key_Row = {1'b1,1'b1,Key_Col[now_col],1'b1};
            2'd2:Key_Row = {1'b1,Key_Col[now_col],1'b1,1'b1};
            2'd3:Key_Row = {Key_Col[now_col],1'b1,1'b1,1'b1};
        endcase
    else                     //保持上一个行状态
        Key_Row = Key_Row_r;

endmodule
```

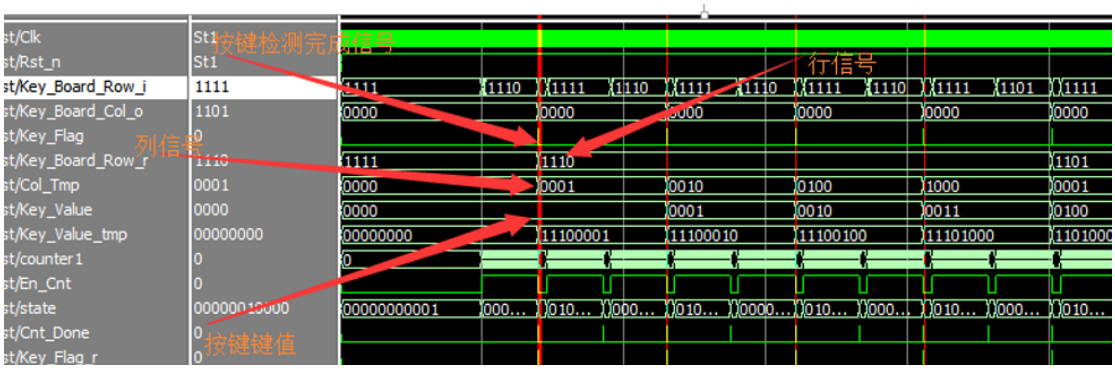
以上就是本节内容的仿真模型代码部分，可能刚开始大家不习惯这么做觉得太繁琐了；但是这样写的结果大大提高了你的仿真正确性，而且模拟的行状态比你用延迟那样让某一行有效要高效的多准确性高得多。下面贴上整个仿真的测试顶层代码:

```
`timescale 1ns/1ns

module Key_Board_tb;

    reg Clk;
    reg Rst_n;
```

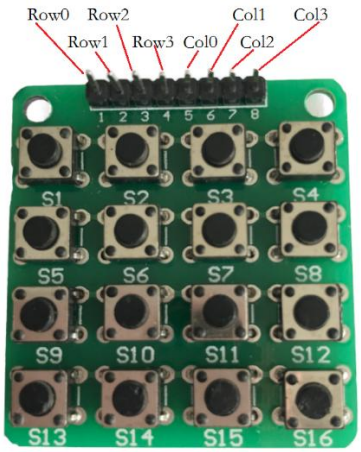
```
    wire [3:0]Key_Row;
    wire [3:0]Key_Col;

    wire Key_Flag;
    wire [3:0]Key_Value;

    Key4x4_Board Key4x4_Board _inst(
        .Clk(Clk),
        .Rst_n(Rst_n),
        .Key_Board_Row_i(Key_Row),
        .Key_Board_Col_o(Key_Col),
        .Key_Flag(Key_Flag),
        .Key_Value(Key_Value)
    );

    Key_Board_model Key_Board_model_inst(Key_Col,Key_Row);

    initial Clk = 1;
    always #10 Clk = ~Clk;

    initial begin
        Rst_n = 0;
        #200 Rst_n = 1;
    end


endmodule
```

经过上面的编写整个工程的所有代码都给大家呈现出来了，接下来我们可以进行仿真了

下面是仿真结果图：



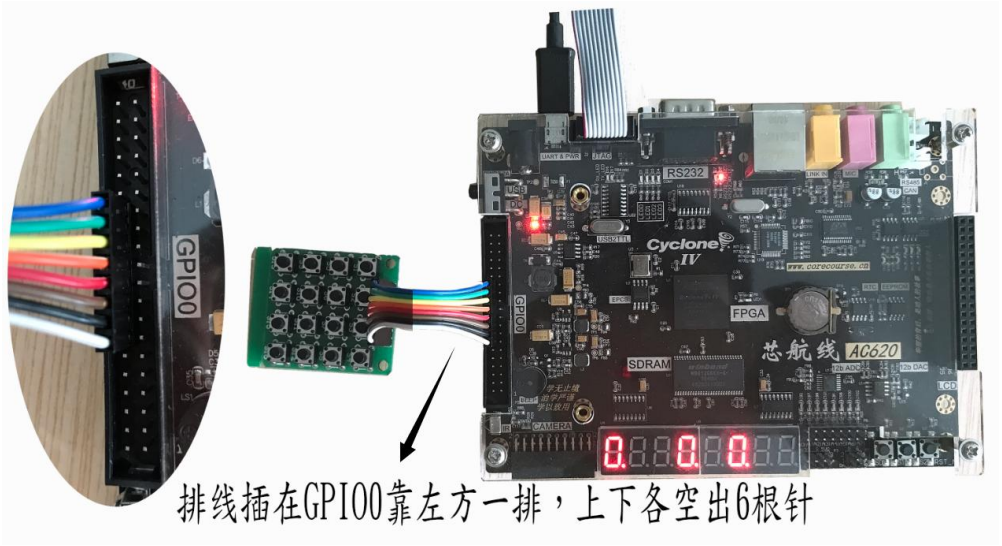
通过观察可以见到当按键检测完成信号 Key_Flag 有效时,Key_Board_Row_r 寄存器中行值为 1110，此时的列值 Col_Tmp 为 0001,检测结果 Key_Value_tmp 为:1110_0001,通过译码结果 Key_Value 为第一行第一列所以键值为“0000”，同理可以分析其他键值情况。

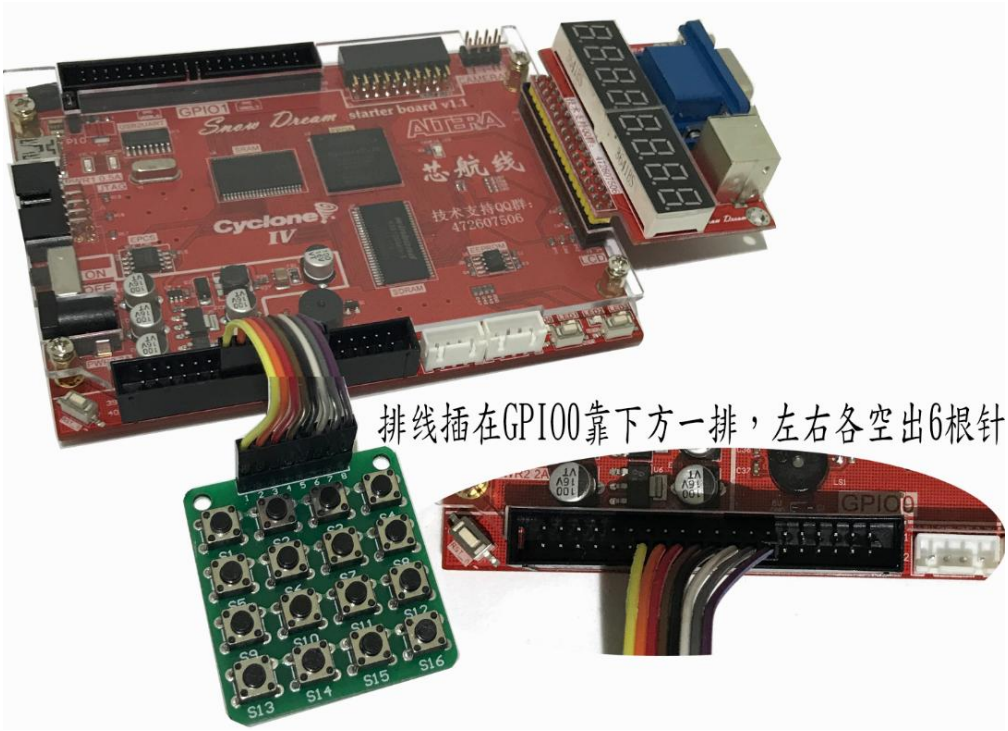


板级调试

硬件介绍：

小梅哥团队出品的 AC620 开发板、Starter 开发板、AC601 评估板、AC6102 以及后续新推出的开发板都支持通过外接的方式连接矩阵键盘，我们也提供了能够很好的兼容这些板卡的矩阵键盘，如图所示。





另外，在之前介绍矩阵键盘检测原理的时候讲解过，对于 ROW 信号，其与 FPGA 的连接管脚上默认是连接了有上拉电阻的，但是市面上很多矩阵键盘默认都是没有提供上拉电阻的，我们开发板选配的矩阵键盘默认也是没有提供上拉电阻的，那么是不是没有上拉电阻就不能完成实验了呢？实际上，我们可以使用 FPGA 片上的 IO 弱上拉电阻来代替外部的上拉电阻。

FPGA 的片上弱上拉电阻设置

Cyclone IV E FPGA 的通用输入输出管脚都支持内部弱上拉电阻，但是时钟输入脚不支持。所以，当需要上拉电阻的信号（如本例中的矩阵键盘 Row 信号和 IIC 协议中的 SDA、SCL 信号）连接到了 FPGA 的通用输入输出管脚上，在一些要求不高的场合，就可以使用片上上拉电阻来为这些信号设置上拉了。以 AC620 和 Starter 开发板连接矩阵键盘为例，矩阵键盘通过排线连接到了两个板卡各自的 GPIO0 接口上，如上图所示，Row0~Row3、Col0~Col3 对应连接到的 FPGA 管脚为

信号名	GPIO0 上的功能脚	AC620 管脚	Starter 管脚
Row[0]	GPIO0-25	PIN_D3	PIN_E5
Row[1]	GPIO0-23	PIN_D1	PIN_F5
Row[2]	GPIO0-21	PIN_F5	PIN_J6
Row[3]	GPIO0-19	PIN_G2	PIN_K6
Col[0]	GPIO0-17	PIN_F3	PIN_L4
Col[1]	GPIO0-15	PIN_J6	PIN_N3
Col[2]	GPIO0-13	PIN_L4	PIN_P3
Col[3]	GPIO0-11	PIN_L3	PIN_M6

分配引脚并设置 row 上拉电阻详细方式

1．如下图所示，在菜单 Assignments 中选择 Pin Planner，也可以直接点击面板上引脚分配的图标；

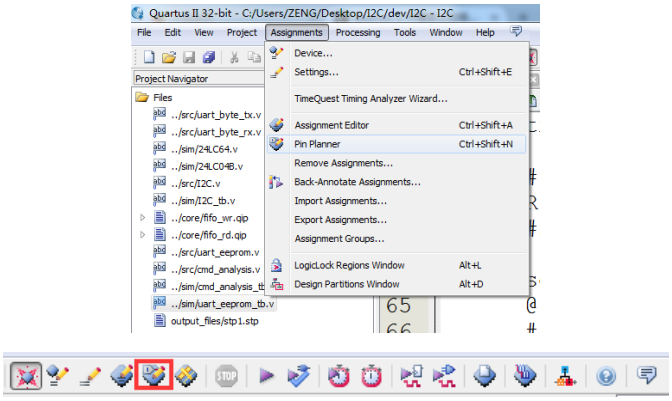


图　进入引脚分配界面选项

2.进入引脚分配的界面之后，按照上面给出的矩阵键盘与 AC620 板卡的连接关系以及引脚分配情况，完成引脚分配工作，Starter 板卡用户请按照 Starter 板引脚分配关系分配，并将 Key_Value 的 4 位信号分别连接到 4 位 LED 上，以方便通过 LED 的亮灭值确定按键的值。

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard
Clk	Input	PIN_E1	1	B1_N0	3.3-V LVTTL
Key_Board_Col_o[0]	Output	PIN_F3	1	B1_N0	3.3-V LVTTL
Key_Board_Col_o[1]	Output	PIN_J6	2	B2_N0	3.3-V LVTTL
Key_Board_Col_o[2]	Output	PIN_L4	2	B2_N0	3.3-V LVTTL
Key_Board_Col_o[3]	Output	PIN_L3	2	B2_N0	3.3-V LVTTL
Key_Board_Row_i[0]	Input	PIN_D3	8	B8_N0	3.3-V LVTTL
Key_Board_Row_i[1]	Input	PIN_D1	1	B1_N0	3.3-V LVTTL
Key_Board_Row_i[2]	Input	PIN_F5	1	B1_N0	3.3-V LVTTL
Key_Board_Row_i[3]	Input	PIN_G2	1	B1_N0	3.3-V LVTTL
Key_Flag	Output				3.3-V LVTTL
Rst_n	Input	PIN_E16	6	B6_N0	3.3-V LVTTL
r_Key_Value[0]	Output	PIN_A2	8	B8_N0	3.3-V LVTTL
r_Key_Value[1]	Output	PIN_B3	8	B8_N0	3.3-V LVTTL
r_Key_Value[2]	Output	PIN_A4	8	B8_N0	3.3-V LVTTL
r_Key_Value[3]	Output	PIN_A3	8	B8_N0	3.3-V LVTTL

3.在弹出的 Pin Planner 界面的 All Pins 区域里任意位置点击鼠标右键，找到 Customize Columns 并点击进入，如下图所示；

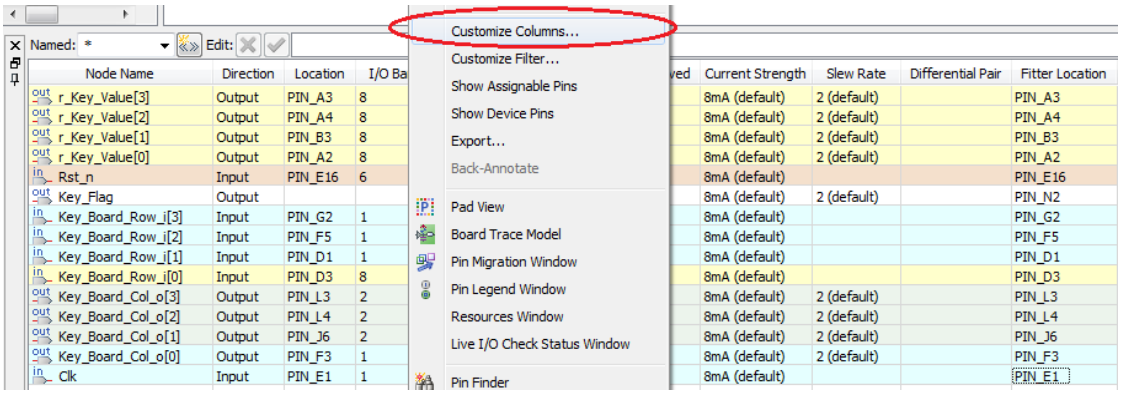


图 引脚配置工具选项界面

4．在弹出的 Customize Columns 对话框的左列表框选择 Weak Pull-Up Resistor，如图下图所示，再点击和大于号（>）一样的图标，这样把 Weak Pull-Up Resistor 添加到右列表框，最后点击 OK。

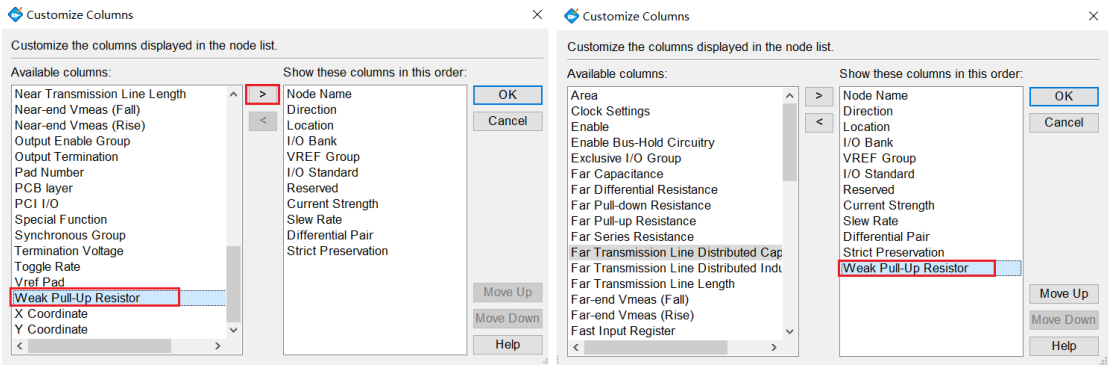
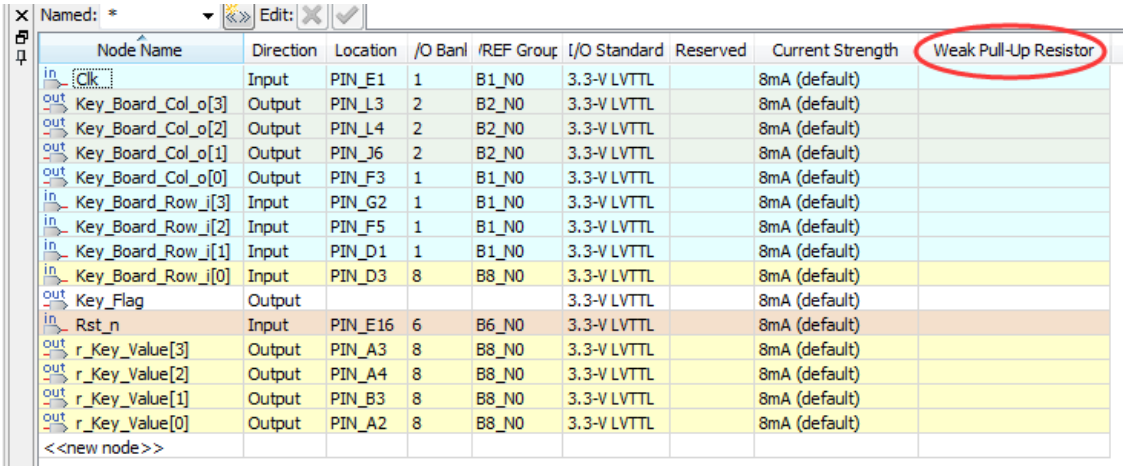


图 Customize Columns 设置界面

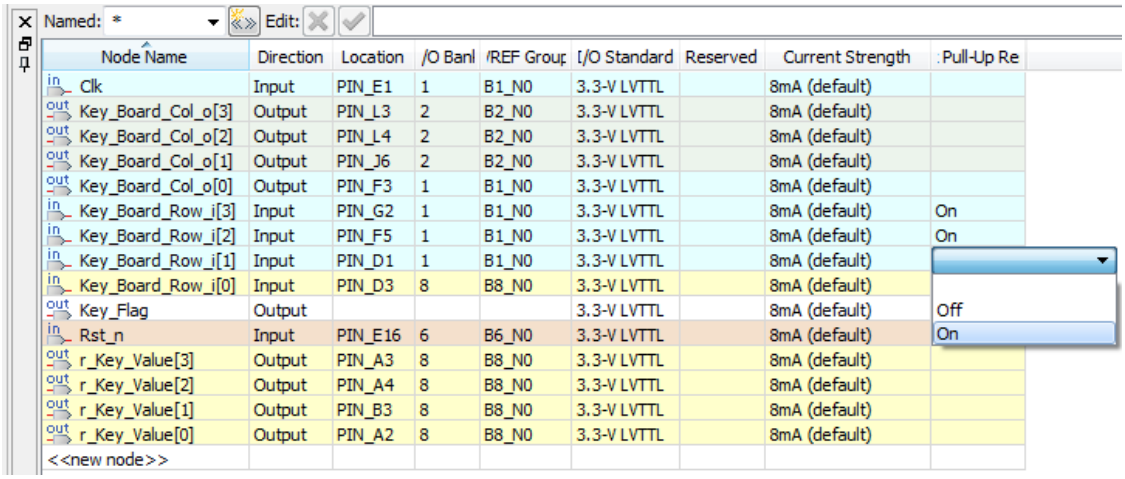
5．经过步骤 4 后，在引脚分配界面就会多出一个 Weak Pull-Up Resistor 列，如下图所示：



Node Name	Direction	Location	/O Banl	/REF Group	I/O Standard	Reserved	Current Strength	Weak Pull-Up Resistor
Clk	Input	PIN_E1	1	B1_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Col_o[3]	Output	PIN_L3	2	B2_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Col_o[2]	Output	PIN_L4	2	B2_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Col_o[1]	Output	PIN_J6	2	B2_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Col_o[0]	Output	PIN_F3	1	B1_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Row_i[3]	Input	PIN_G2	1	B1_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Row_i[2]	Input	PIN_F5	1	B1_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Row_i[1]	Input	PIN_D1	1	B1_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Row_i[0]	Input	PIN_D3	8	B8_N0	3.3-V LVTTL		8mA (default)	
Key_Flag	Output				3.3-V LVTTL		8mA (default)	
Rst_n	Input	PIN_E16	6	B6_N0	3.3-V LVTTL		8mA (default)	
r_Key_Value[3]	Output	PIN_A3	8	B8_N0	3.3-V LVTTL		8mA (default)	
r_Key_Value[2]	Output	PIN_A4	8	B8_N0	3.3-V LVTTL		8mA (default)	
r_Key_Value[1]	Output	PIN_B3	8	B8_N0	3.3-V LVTTL		8mA (default)	
r_Key_Value[0]	Output	PIN_A2	8	B8_N0	3.3-V LVTTL		8mA (default)	
<<new node>>								

图　添加了 WeakPull-Up Resistor 选项的引脚分配界面

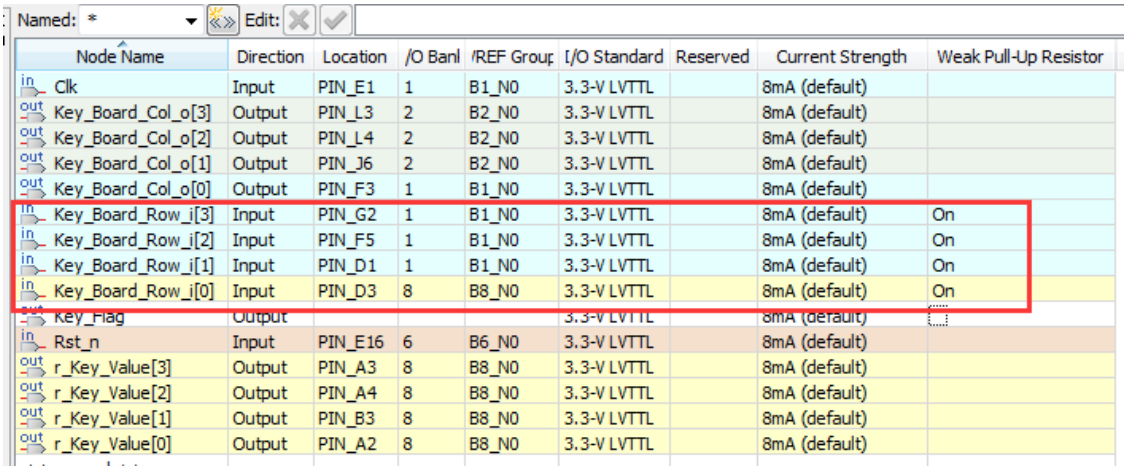
6.再把需要上拉的 Row0~Row3 对应 WeakPull-Up Resistor 的位置双击鼠标左键，就会弹出一个 Off/On 的选项，选上 On 就可以了。



Node Name	Direction	Location	/O Banl	/REF Group	I/O Standard	Reserved	Current Strength	Pull-Up Re
Clk	Input	PIN_E1	1	B1_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Col_o[3]	Output	PIN_L3	2	B2_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Col_o[2]	Output	PIN_L4	2	B2_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Col_o[1]	Output	PIN_J6	2	B2_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Col_o[0]	Output	PIN_F3	1	B1_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Row_i[3]	Input	PIN_G2	1	B1_N0	3.3-V LVTTL		8mA (default)	On
Key_Board_Row_i[2]	Input	PIN_F5	1	B1_N0	3.3-V LVTTL		8mA (default)	On
Key_Board_Row_i[1]	Input	PIN_D1	1	B1_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Row_i[0]	Input	PIN_D3	8	B8_N0	3.3-V LVTTL		8mA (default)	
Key_Flag	Output				3.3-V LVTTL		8mA (default)	Off
Rst_n	Input	PIN_E16	6	B6_N0	3.3-V LVTTL		8mA (default)	On
r_Key_Value[3]	Output	PIN_A3	8	B8_N0	3.3-V LVTTL		8mA (default)	
r_Key_Value[2]	Output	PIN_A4	8	B8_N0	3.3-V LVTTL		8mA (default)	
r_Key_Value[1]	Output	PIN_B3	8	B8_N0	3.3-V LVTTL		8mA (default)	
r_Key_Value[0]	Output	PIN_A2	8	B8_N0	3.3-V LVTTL		8mA (default)	
<<new node>>								

图　为 Row0~Row3 设置上拉电阻

7.设置完毕后。按照 AC620 开发板背面标注的功能对应的引脚编号输入引脚分配表中以完成管脚分配，如图下图所示。

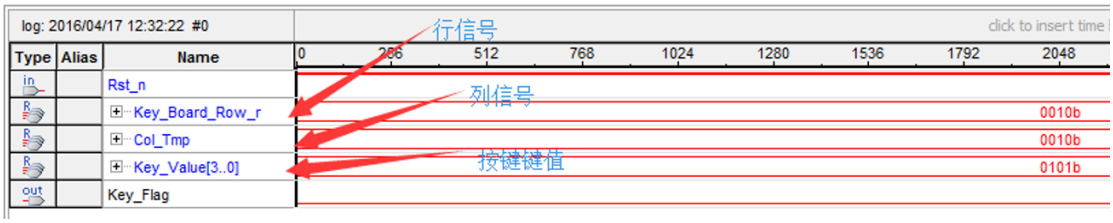


Node Name	Direction	Location	/O Banl	/REF Group	I/O Standard	Reserved	Current Strength	Weak Pull-Up Resistor
Clk	Input	PIN_E1	1	B1_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Col_o[3]	Output	PIN_L3	2	B2_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Col_o[2]	Output	PIN_L4	2	B2_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Col_o[1]	Output	PIN_J6	2	B2_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Col_o[0]	Output	PIN_F3	1	B1_N0	3.3-V LVTTL		8mA (default)	
Key_Board_Row_i[3]	Input	PIN_G2	1	B1_N0	3.3-V LVTTL		8mA (default)	On
Key_Board_Row_i[2]	Input	PIN_F5	1	B1_N0	3.3-V LVTTL		8mA (default)	On
Key_Board_Row_i[1]	Input	PIN_D1	1	B1_N0	3.3-V LVTTL		8mA (default)	On
Key_Board_Row_i[0]	Input	PIN_D3	8	B8_N0	3.3-V LVTTL		8mA (default)	On
Key_Flag	Output				3.3-V LVTTL		8mA (default)	
Rst_n	Input	PIN_E16	6	B6_N0	3.3-V LVTTL		8mA (default)	
r_Key_Value[3]	Output	PIN_A3	8	B8_N0	3.3-V LVTTL		8mA (default)	
r_Key_Value[2]	Output	PIN_A4	8	B8_N0	3.3-V LVTTL		8mA (default)	
r_Key_Value[1]	Output	PIN_B3	8	B8_N0	3.3-V LVTTL		8mA (default)	
r_Key_Value[0]	Output	PIN_A2	8	B8_N0	3.3-V LVTTL		8mA (default)	

图　引脚分配详情

下面我们再通过另开一种方法也是工作中比较常用的一种验证方法，嵌入式逻辑分析仪，来检测本设计在板上运行时的工作情况。

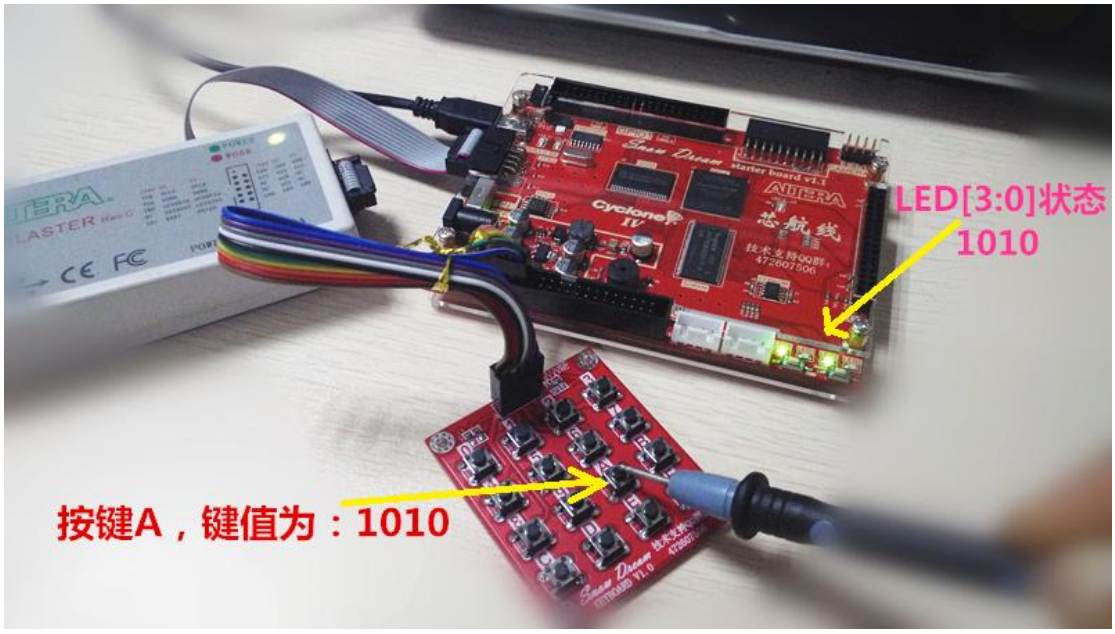
经过上述描述，硬件的环境搭建完毕，然后使用 Quartus ii 里面 SignalTap 进行在线调试，抓取实时数据分析波形。



观察逻辑分析仪抓取到的信号，可以见到当完成一次检测时，Key_Board_Row_r 寄存器中行值为 0010b，此时的列值 Col_Tmp 为 0010b,检测结果 Key_Value_tmp 为 0010_0010b,通过译码结果 Key_Value 为第一行第一列所以键值为“0101”，同理可以分析其他键值情况。

目标板验证

下图为使用另一款矩阵键盘连接 Starter 开发板时最终的实验效果图，其他矩阵键盘按照本文描述的方式连接，效果与此类似。



其中，4 个 LED 灯从左到右依次为 LED0 到 LED3，状态为亮灭亮灭 ，对应的 4 位驱动 IO 逻辑由低到高为 0101，与按键值从低到高的 0101 一致

总结

本设计实例通过状态机的设计思想实现了矩阵键盘的扫描以及消抖工作，同时介绍了在 FPGA 上设置 IO 口的片上弱上拉电阻的方法，该方法尤其适用于做 IIC 接口时，在总线上没有上拉电阻的情况下临时顶替。本矩阵键盘消抖模块具有一定的实用性，用户可自行用于自己设计的系统中。

课后作业

作业一：

将矩阵键盘的键值显示到数码管上。

作业二:

制作简单的计算器

设计要求：

1、 完成整体架构框图设计；

2、 完成状态转移图的设计；

3、 自己独立完成成个工程代码的编写；

4、 使用 ModelSim 对设计进行仿真验证。

功能要求：

1、 使用矩阵键盘、数码管；

2、 最起码能够实现 100000 以内的加减乘除运算；

温馨提示：

1. 由于计算只用到 0~9 这几个键，那么剩下的 A~F 可以用作符号键。
2. 要将计算结果正确显示到数码管上就需要用二进制转 BCD。

六月飞鱼

芯航线电子工作室

2016 年 4 月 16 日