

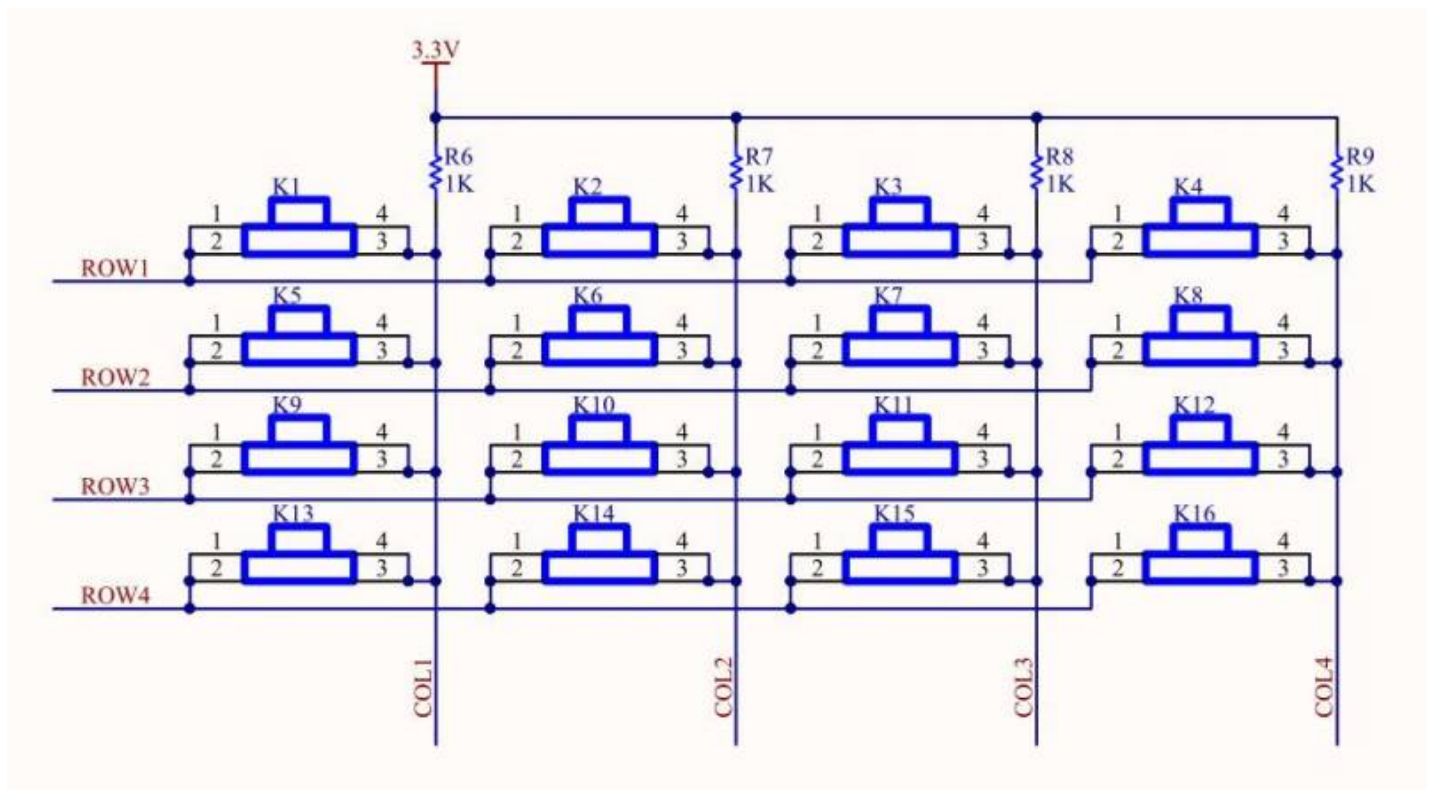
Matrix button driver based on STEP FPGA

In this section, we will use FPGA to drive the 4×4 matrix keyboard on the backplane.

====Hardware description=====

When there are a large number of keys in the keyboard, in order to reduce the occupation of the I/O ports, the keys are usually arranged in a matrix form, and the row and column lines are respectively connected to the two ends of the key switch, so that we can pass 4 row lines and 4 column lines (a total of 8 I/O ports) connect 16 buttons, and the more the buttons, the more obvious the advantage.

FPGA drives the matrix button module. First of all, let's understand the hardware connection of the matrix buttons: the picture above is the hardware circuit diagram of the 4×4 matrix buttons. You can see 4 row lines (ROW1, ROW2, ROW3, ROW4) and 4 column lines (COL1, COL2, COL3, COL4), and the column line is connected to the VCC voltage (3.3V) through a pull-up resistor. For matrix buttons:



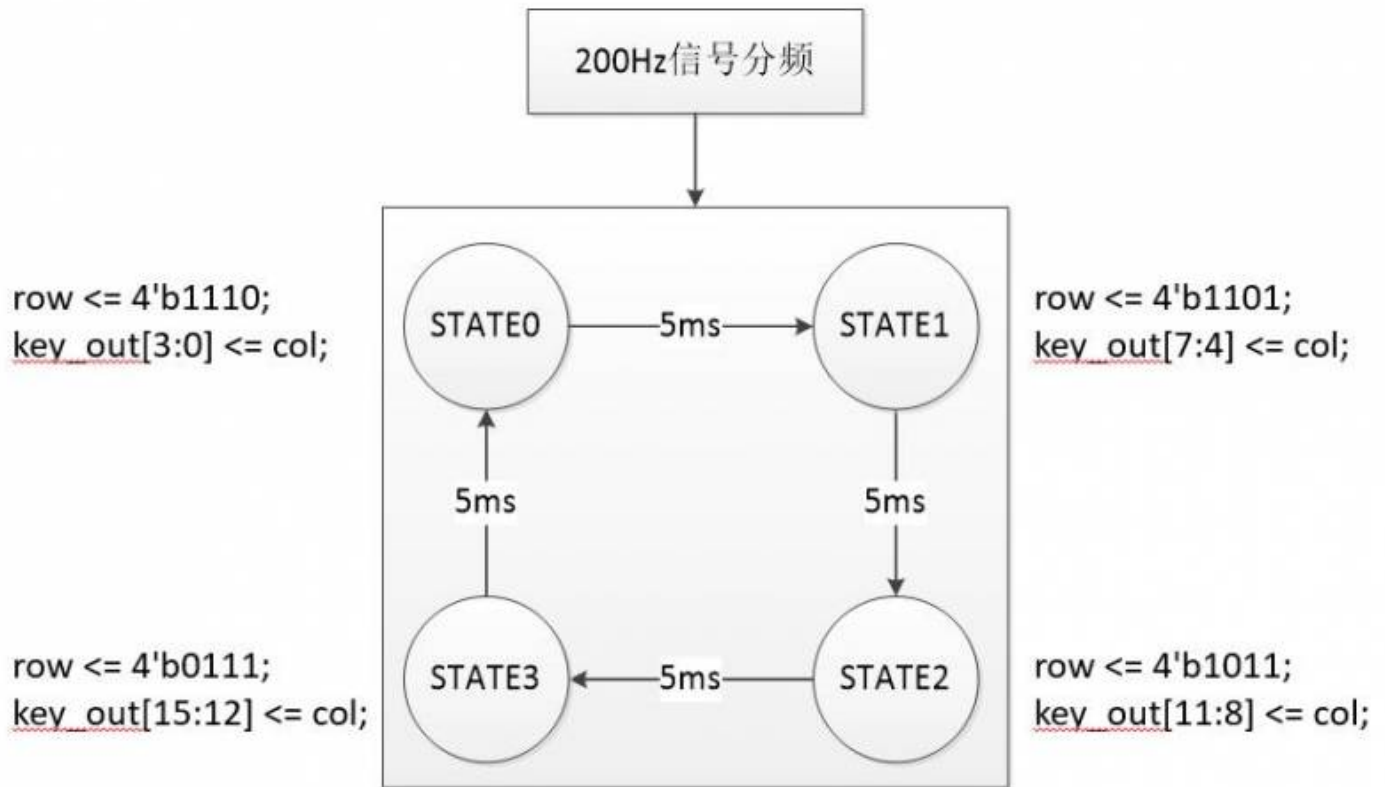
1. The 4 row lines are input, which are pulled up or down by FPGA control,
2. The output of the 4 column lines is determined by the input of the 4 row lines and the state of the buttons, and output to the FPGA

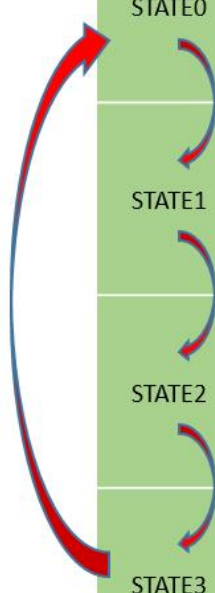
When at a certain moment, FPGA controls 4 row lines respectively ROW1=0, ROW2=1, ROW3=1, ROW4=1,

- For K1, K2, K3, K4 buttons: when pressed, the corresponding 4 column lines output COL1=0, COL2=0, COL3=0, COL4=0, and the 4 column lines output COL1=1, COL2=1 when not pressed, COL3=1, COL4=1,
- For the keys between K5~~~K16: Whether pressed or not, the corresponding 4 column lines output COL1=1, COL2=1, COL3=1, COL4=1,

Through the above description: only K1, K2, K3, K4 buttons are pressed at this moment, will cause the 4 column lines to output COL1=0, COL2=0, COL3=0, COL4=0, otherwise COL1=1, COL2=1, COL3=1, COL4=1, on the contrary, when FPGA detects a low level signal in the column lines (COL1, COL2, COL3, COL4), the corresponding K1, K2, K3, K4 buttons should be Pressed down.

According to the scanning method, it is divided into 4 moments, which correspond to one of the 4 row lines being pulled down, and the 4 moments are cycled in turn. This completes all the scanning and detection of the matrix buttons. We use these 4 in the program The time corresponds to the 4 states of the state machine. As for the cycle period, according to our basic tutorial, the instability time of button jitter is within 10ms, so the period of sampling the same button is greater than 10ms, which also takes 20ms. 20ms time corresponds to 4 states, and a state transition is performed every 5ms.





状态	行	列	按键
STATE0	ROW1=0 第1行输出低电平 其余为高电平	判断第1列电平：COL1==0/1	K1按下/松开
		判断第2列电平：COL2==0/1	K2按下/松开
		判断第3列电平：COL3==0/1	K3按下/松开
		判断第4列电平：COL4==0/1	K4按下/松开
STATE1	ROW2=0 第2行输出低电平 其余为高电平	判断第1列电平：COL1==0/1	K5按下/松开
		判断第2列电平：COL2==0/1	K6按下/松开
		判断第3列电平：COL3==0/1	K7按下/松开
		判断第4列电平：COL4==0/1	K8按下/松开
STATE2	ROW3=0 第3行输出低电平 其余为高电平	判断第1列电平：COL1==0/1	K9按下/松开
		判断第2列电平：COL2==0/1	K10按下/松开
		判断第3列电平：COL3==0/1	K11按下/松开
		判断第4列电平：COL4==0/1	K12按下/松开
STATE3	ROW4=0 第4行输出低电平 其余为高电平	判断第1列电平：COL1==0/1	K13按下/松开
		判断第2列电平：COL2==0/1	K14按下/松开
		判断第3列电平：COL3==0/1	K15按下/松开
		判断第4列电平：COL4==0/1	K16按下/松开

====Verilog code=====

```
// -----  
// >>>>>>>>>>>>>>>>>>>>> COPYRIGHT NOTICE <<<<<<<<<<<<<<<<<<<<<  
// -----  
// Module: Array_KeyBoard  
//  
// Author: Step  
//  
// Description: Array_KeyBoard  
//  
// Web: www.stepfapga.com  
//  
// -----  
// Code Revision History:  
// -----  
// Version: |Mod. Date: |Changes Made:  
// V1.0 |2015/11/11 |Initial ver  
// -----  
module Array_KeyBoard #  
(  
    parameter                NUM_FOR_200HZ = 60000 //Define the counting range of  
counter cnt, which can be changed          during instantiation )  
  
(  
    input clk_in ,            //system clock  
    input                    rst_n_in ,           //system reset, low effective  
    input [ 3 : 0 ] col ,      / / Matrix button column interface  
    output reg [ 3 : 0 ] row ,   // Matrix button row interface  
    output reg [ 15 : 0 ] key_out // Signal after debounce  
) ;  
/*  
Due to the use of 4x4 matrix buttons, realized by scanning method, so here is realized by state machine, which is divided into 4 states  
In a certain state time, the corresponding 4 buttons are equivalent to independent buttons, and can be sampled according to the periodic sampling method of independent buttons  
During periodic sampling, sampling is performed every 20ms, which corresponds to the state machine looping every 20ms, and each state corresponds to 5ms time  
If you don't understand the principle of matrix buttons, please go to understand the principle of matrix buttons  
*/  
    localparam STATE0 = 2'b00 ;  
    localparam STATE1 = 2'b01 ;  
    localparam STATE2 = 2'b10 ;  
    localparam STATE3 = 2'b11 ;  
  
    //Counter frequency division realizes 5ms periodic signal clk_200hz  
    reg [ 15 : 0 ] cnt ;  
    reg clk_200hz ;  
    always @ ( posedge clk_in or negedge rst_n_in ) begin
```

```

        if ( ! Rst_n_in ) begin                                //Counter cnt is cleared when reset, c
lk_200hz signal starts The starting level is low
            cnt <= 1 6'd0 ;
            clk_200hz <= 1'b0 ;
        end else begin
            if ( cnt >= ( ( NUM_FOR_200HZ>> . 1 ) - . 1 ) ) the begin // dig
ital logic corresponds to a shift right except 2
                CNT <= . 1 6'd0 ;
                clk_200hz <= ~ clk_200hz ;        // clk_200hz signal is negated
            End the else the begin
                CNT <= CNT + 1'b1 ;
                clk_200hz <= clk_200hz ;
            end
        end
    end

    reg          [ 1 : 0 ]          c_state ;
    //The state machine cycles through 4 states according to the clk_200hz signal, and eac
h state is valid for a single line of the row interface of the matrix buttons
    always @ ( posedge clk_200hz or negedge rst_n_in ) begin
        if ( ! rst_n_in ) begin
            c_state <= STATE0 ;
            row <= 4'b1110 ;
        end else begin
            case ( c_state )
                STATE0 : begin c_state <= STATE1 ; row<= 4'b1101 ; End
Row // output matrix in a state key and a corresponding jump c_state state
                STATE1 : the begin c_state <= STATE2 ; Row <= 4'b1011 ; End
                STATE2 : the begin c_state <= STATE3 and ; Row <= . 4 'b0111
; end

                STATE3 : begin c_state <= STATE0 ; row <= 4'b1110 ; end
                default : begin c_state <= STATE0 ;row <= 4'b1110 ; end
            endcase
        end
    end

    //Because a single row in each state is valid, the state of the corresponding 4 button
s is obtained by sampling the level state of the column interface, and then looping
    always @ ( negedge clk_200hz or negedge rst_n_in ) begin
        if ( ! Rst_n_in ) begin
            key_out <= 1 6 'hffff ;
        end else begin
            case ( c_state )
                STATE0 : key_out [ 3 : 0 ] <= col ;           //Collect the
column data of the current state and assign it to the corresponding register bit
                STATE1 : key_out [ 7 : 4 ] <= col ;
                STATE2 : key_out [ 11 : 8 ] <= col ;
                STATE3 : key_out [ 15 : 12 ] <= col ;
                default : key_out <= 1 6'hffff ;
            endcase
        end
    end
end

```

```
endmodule
```

====Summary====

This section mainly explains the working principle and software design of the matrix buttons. You need to create your own project while mastering it, and generate the FPGA configuration file loading test through the entire design process. If you are not familiar with the use of Diamond software, please refer to here: [Use of Diamond](#) .

====Related Information====

Use STEP-MXO2 second generation matrix key procedures: subsequent download connection will be updated
using the STEP-MAX10 matrix key procedures: subsequent download link will be updated