****

EDA实验报告

|  |  |
| --- | --- |
| 题目 | PS/2接口的矩阵键盘 |
| 姓名 | 王瀚雷 |
| 班级 | 电科1802 |
| 学号 | 8206180209 |

物理与电子学院

2020年11月

实验平台说明

|  |  |
| --- | --- |
| **特征** | **说明** |
| 操作系统版本和系统类型 | Windows 10 home 64bit |
| EDA软件名称和版本 | Quatus Prime Lite 20.1 |
| **DE1-SoC**开发板编号 | 1711503S |

自查清单

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Part？/Lab？** | 题目分析 | 原理分析 | 代码解读 | 设计验证 | 总结讨论 |
| **Part 1** | **√** | **√** | **√** | **√** | **√** |
| **Part 2** | **√** | **√** | **√** | **√** | **√** |
| **Part 3** | **√** | **√** | **√** | **√** | **√** |
| **Part 4** | **√** | **√** | **√** | **√** | **√** |
| **Part 5** | **√** | **√** | **√** | **√** | **√** |
| **Part 6** | **√** | **√** | **√** | **√** | **√** |
| **Part 7** | **√** | **√** | **√** | **√** | **√** |

目录

[PART 1 矩阵键盘扫描 5](#_Toc56330067)

[题目分析 5](#_Toc56330068)

[原理分析 5](#_Toc56330069)

[代码解读 5](#_Toc56330070)

[设计验证 6](#_Toc56330071)

[总结 6](#_Toc56330072)

[PART 2 LED点阵扫描 7](#_Toc56330073)

[题目分析 7](#_Toc56330074)

[原理分析 7](#_Toc56330075)

[代码解读 7](#_Toc56330076)

[设计验证 8](#_Toc56330077)

[总结 8](#_Toc56330078)

[PART 3 从PS/2键盘读取键值 9](#_Toc56330079)

[题目分析 9](#_Toc56330080)

[原理分析 9](#_Toc56330081)

[代码解读 9](#_Toc56330082)

[设计验证 10](#_Toc56330083)

[总结 12](#_Toc56330084)

[PART 4 模拟PS/2键盘 13](#_Toc56330085)

[题目分析 13](#_Toc56330086)

[原理分析 13](#_Toc56330087)

[代码解读 13](#_Toc56330088)

[设计验证 16](#_Toc56330089)

[总结 17](#_Toc56330090)

[PART 5 FIFO 18](#_Toc56330091)

[题目分析 18](#_Toc56330092)

[原理分析 18](#_Toc56330093)

[代码解读 18](#_Toc56330094)

[设计验证 20](#_Toc56330095)

[总结 21](#_Toc56330096)

[PART 6 发送扫描码 22](#_Toc56330097)

[题目分析 22](#_Toc56330098)

[原理分析 22](#_Toc56330099)

[代码解读 22](#_Toc56330100)

[设计验证 24](#_Toc56330101)

[总结 24](#_Toc56330102)

[PART 7 点亮LED 25](#_Toc56330103)

[题目分析 25](#_Toc56330104)

[原理分析 25](#_Toc56330105)

[代码解读 25](#_Toc56330106)

[设计验证 25](#_Toc56330107)

[总结 25](#_Toc56330108)

[PART 8 最小系统 26](#_Toc56330109)

[题目分析 26](#_Toc56330110)

[原理分析 26](#_Toc56330111)

[代码解读 26](#_Toc56330112)

[设计验证 26](#_Toc56330113)

[总结 26](#_Toc56330114)

# PART 1 矩阵键盘扫描

## 题目分析

设计系统读取4\*4矩阵键盘键值，并显示到开发板的数码管上。（要求显示的键值稳定）

## 原理分析

矩阵键盘采用列扫描法得到键值，然后通过译码器将其转化为数码管显示的段码。

#### **代码解读**

    assign GPIO[35:32] = COL;

    assign ROW         = GPIO[31:28];

    assign HEX0        = value;

    //assign LEDR      = {COL,ROW};

    divider#(10000) div(CLOCK\_50, clk);

    always @ (posedge clk)

    begin

        state <= state + 1;

        case(state)

            0: COL      <= 4'b1110;

            1: COL      <= 4'b1101;

            2: COL      <= 4'b1011;

            3: COL      <= 4'b0111;

            default:COL <= 4'b1111;

        endcase

    end

    always @ (posedge clk)

    begin

        case({COL,ROW})

            8'b1110\_0111 : value = 7'b100\_0000;

            8'b1110\_1011 : value = 7'b111\_1001;

            8'b1110\_1101 : value = 7'b010\_0100;

            8'b1110\_1110 : value = 7'b011\_0000;

            8'b1101\_0111 : value = 7'b001\_1001;

            8'b1101\_1011 : value = 7'b001\_0010;

            8'b1101\_1101 : value = 7'b000\_0010;

            8'b1101\_1110 : value = 7'b111\_1000;

            8'b1011\_0111 : value = 7'b000\_0000;

            8'b1011\_1011 : value = 7'b001\_0000;

            8'b1011\_1101 : value = 7'b000\_1000;

            8'b1011\_1110 : value = 7'b000\_0011;

            8'b0111\_0111 : value = 7'b100\_0110;

            8'b0111\_1011 : value = 7'b010\_0001;

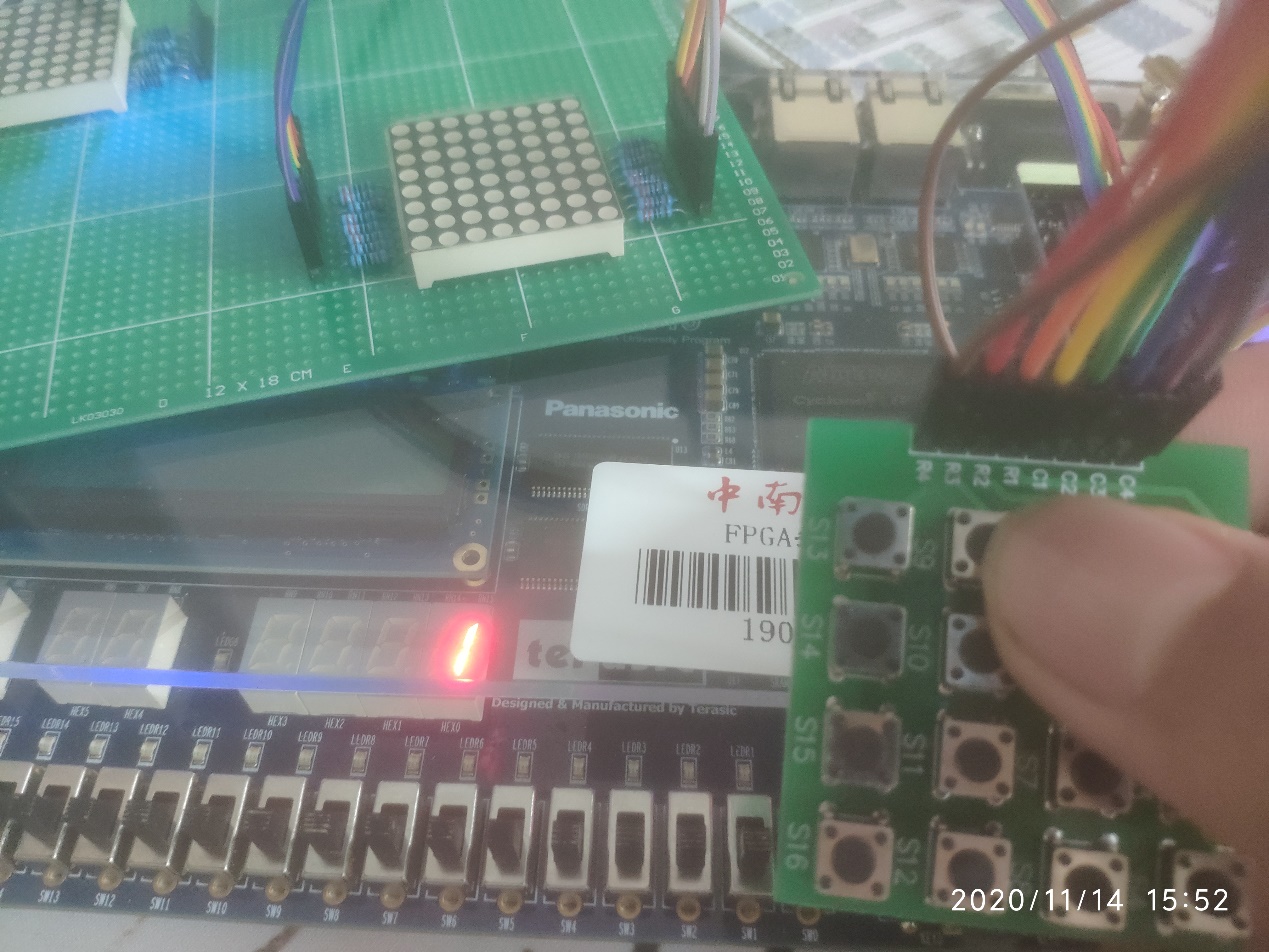
            8'b0111\_1101 : value = 7'b000\_0110;

            8'b0111\_1110 : value = 7'b000\_1110;

        endcase

    end

## **设计验证**



图表 1按下第二个键

## 总结

过于简单无需总结。

# PART 2 LED点阵扫描

## 题目分析

用LED点阵滚动显示图案。

## 原理分析

使用mif文件将图案存储到RAM中，col计数器用来扫描，row计数器用来提供滚动显示的偏置。借助电脑把图案转换成ram里的数据格式输入到mif文件中。

#### **代码解读**

    reg [63:0] counter;

    reg [2:0] col, row;

    (\* ram\_init\_file = "img.mif" \*)

    reg[17:10] memory[7:0];

    always @(posedge CLOCK\_50)

        counter <= counter + 1;

    always @(posedge counter[15])

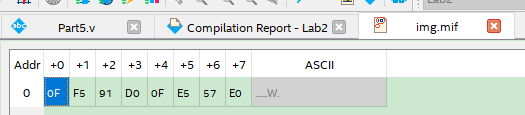
        col <= col + 1;

    always @(posedge counter[24])

        row <= row + 1;

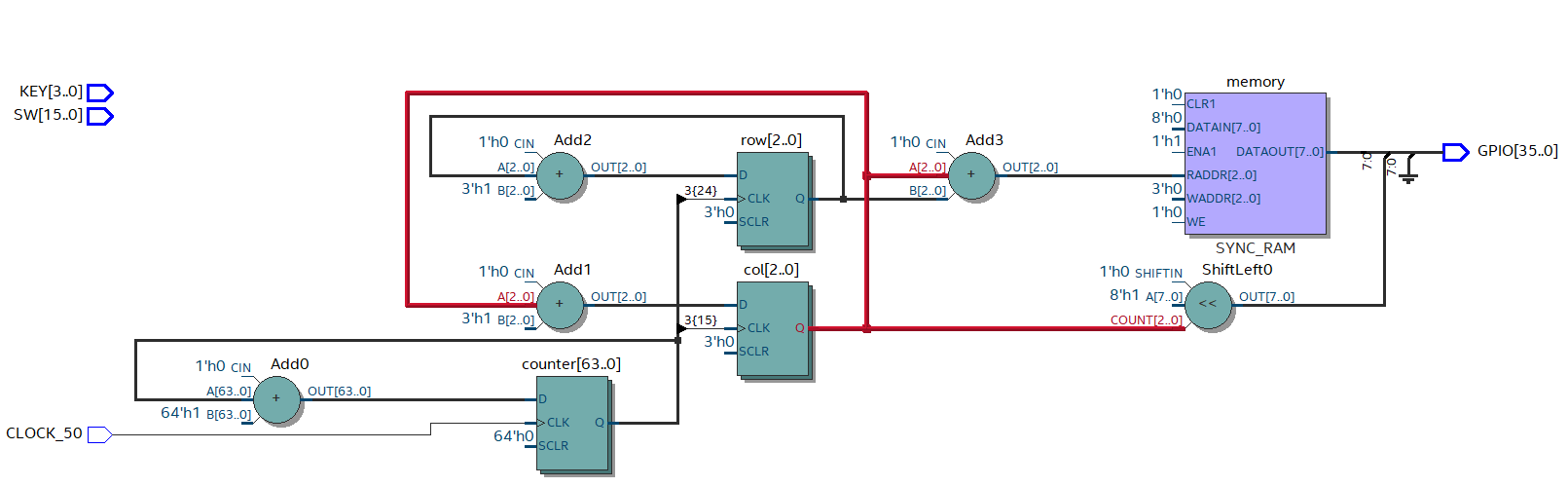
    assign GPIO[7:0]   = 8'b00000001 << col;

    assign GPIO[17:10] = memory[col + row];

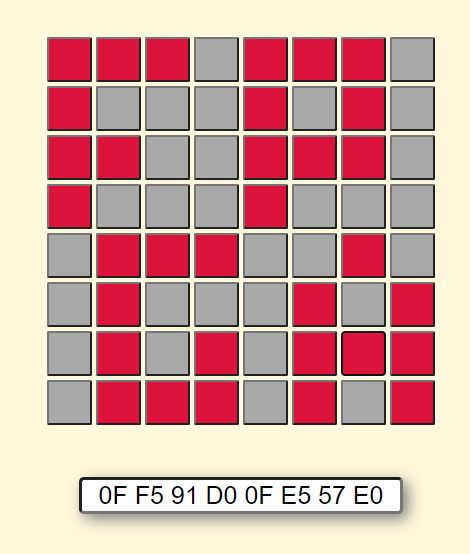


图表 2 显示在点阵上的数据

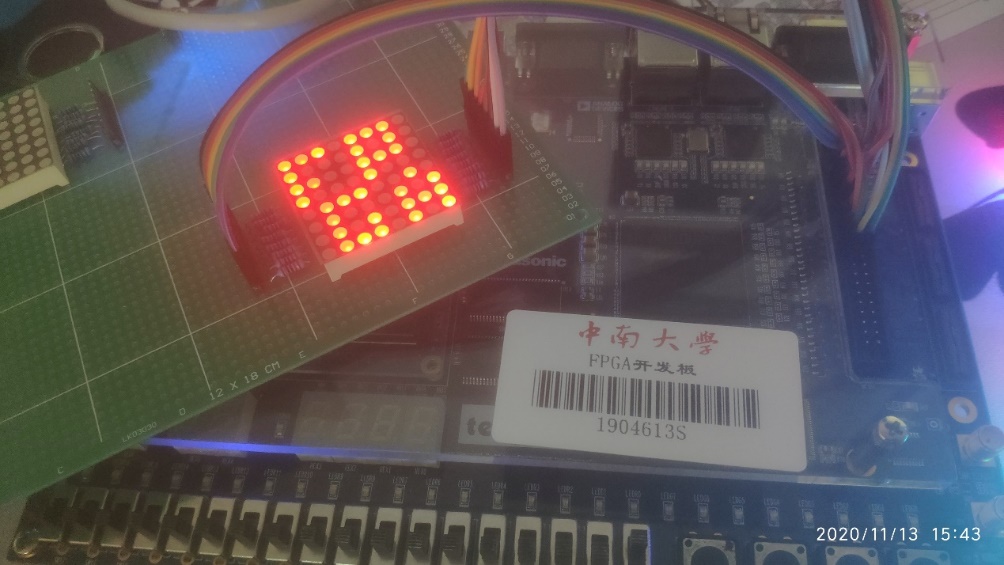
## **设计验证**



图表 3 RTL



图表 4 设计显示图案



图表 5点阵显示的“FPGA”

## 总结

通过增加RAM位数即可扩展显示内容，实现滚动显示长图。

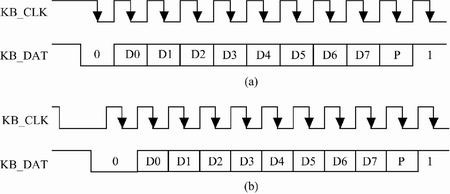
# PART 3 从PS/2键盘读取键值

## 题目分析

将PS/2键盘插入开发板，用数码管显示按下了哪个键。

## 原理分析

|  |  |
| --- | --- |
| 1个起始位 | 总是逻辑0 |
| 8个数据位 | （LSB）低位在前 |
| 1个奇偶校验位 | 奇校验 |
| 1个停止位 | 总是逻辑1 |
| 1个应答位 | 仅用在主机对设备的通讯中 |



图表 6键盘接口时序(a) 发送；(b) 接收

#### **代码解读**

    always @ (posedge clk or negedge rst)

    begin //接收数据的相应处理，这里只对1byte的键值进行处

        if (!rst) begin

            key\_f0      <= 1'b0;

            ps2\_state\_r <= 1'b0;

        end

        else if (num == 4'd10&&neg\_ps2k\_clk)

        begin   //刚传送完?个字节数

            if (temp\_data == 8'hf0)

                key\_f0 <= 1'b1;

            else

            begin

                if (!key\_f0)

                begin //说明有键按下

                    ps2\_state\_r <= 1'b1;

                    ps2\_byte\_r  <= temp\_data; //锁存当前键

                end

                else

                begin

                    ps2\_state\_r <= 1'b0;

                    key\_f0      <= 1'b0;

                end

            end

        end

    end

    reg ps2k\_clk\_r0,ps2k\_clk\_r1,ps2k\_clk\_r2; //ps2k\_clk状太寄存器

    //wire pos\_ps2k\_clk;    // ps2k\_clk上升沿标志位

    wire neg\_ps2k\_clk;   // ps2k\_clk下降沿标志位

    always @ (posedge clk or negedge rst) begin

        if (!rst) begin

            ps2k\_clk\_r0 <= 1'b0;

            ps2k\_clk\_r1 <= 1'b0;

            ps2k\_clk\_r2 <= 1'b0;

        end

        else begin                        //锁存状态，进行滤波

            ps2k\_clk\_r0 <= ps2k\_clk;

            ps2k\_clk\_r1 <= ps2k\_clk\_r0;

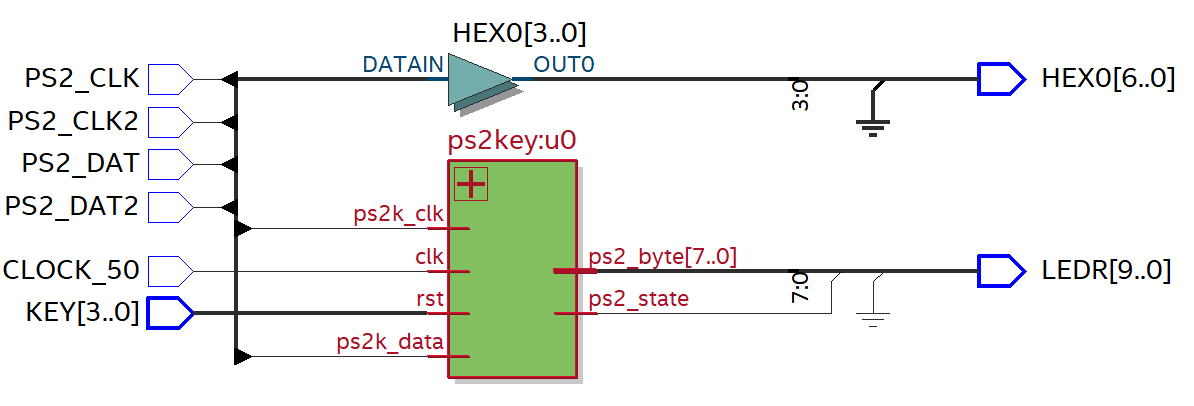
            ps2k\_clk\_r2 <= ps2k\_clk\_r1;

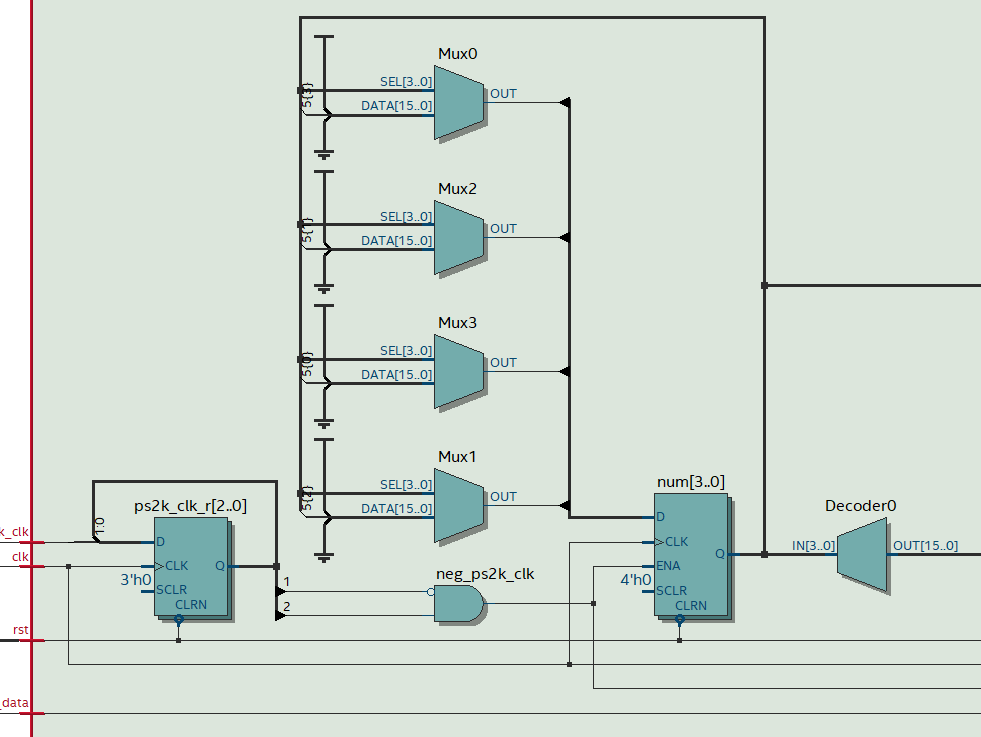
        end

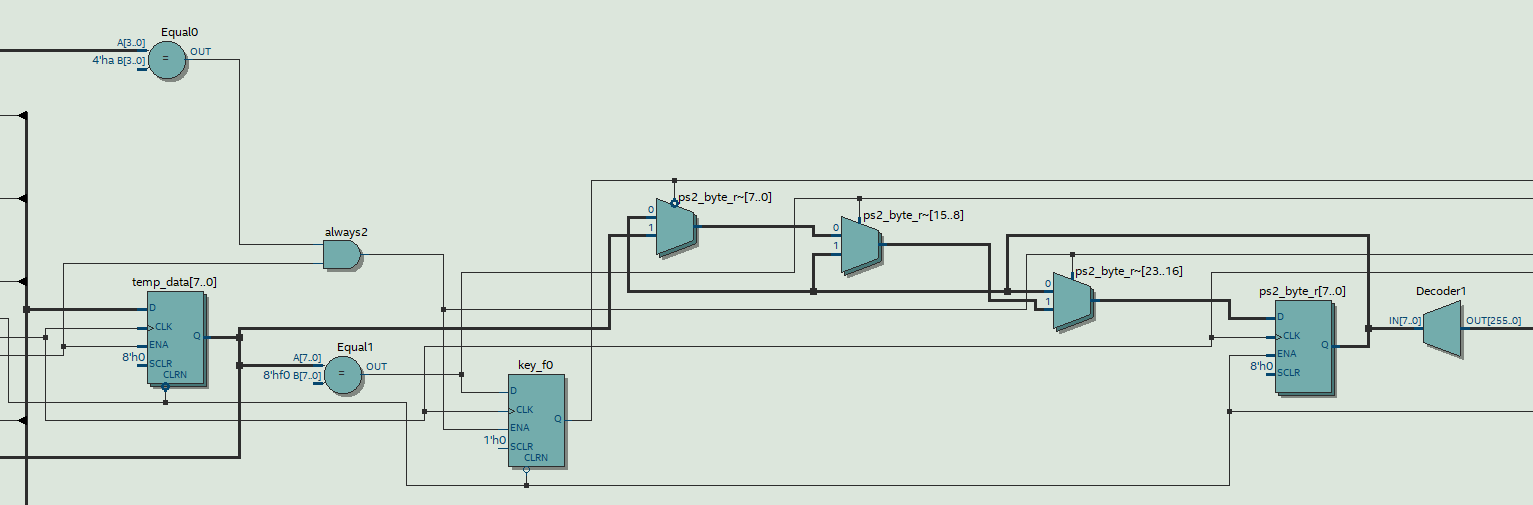
    end

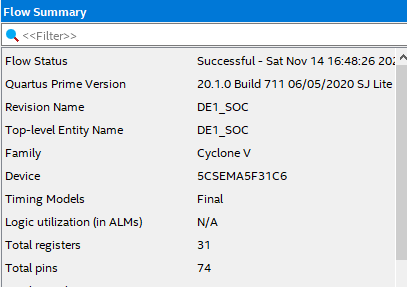
    assign neg\_ps2k\_clk = ~ps2k\_clk\_r1 & ps2k\_clk\_r2;   //下降沿

## **设计验证**



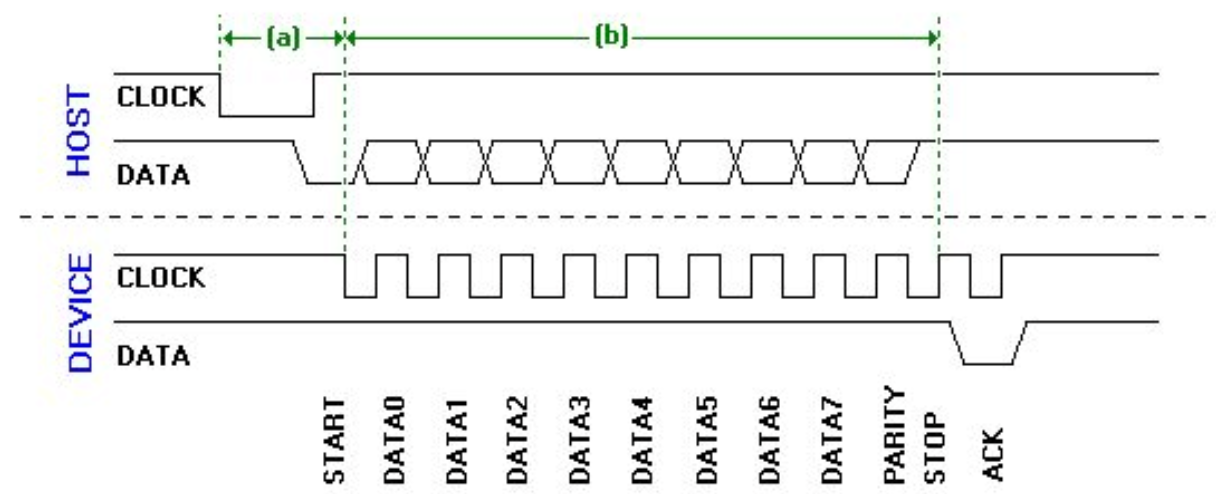




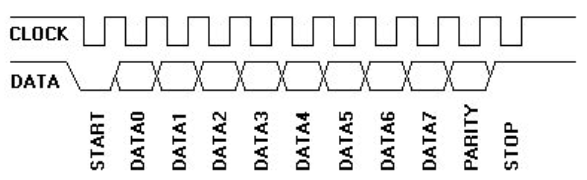


## 总结

同大多数有线通信协议一样，PS/2也是分主机（host）和从机（device）的，本部分是用FPGA作为主机来读取键盘设备，下一部分则是使用FPGA来模拟一个键盘给电脑发送。由于时钟都是设备产生的，所以从机的逻辑比主机要复杂得多。



图表 7 主机到设备



图表 8 设备到主机

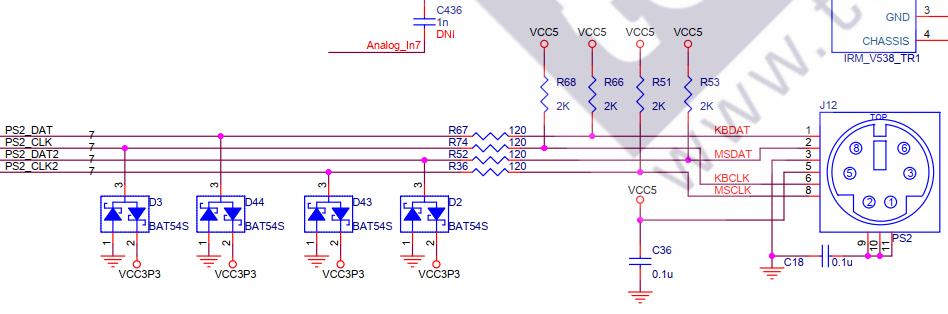
# PART 4 模拟PS/2键盘

## 题目分析

使用FPGA模拟一个PS/2键盘，向电脑发送键码。

## 原理分析

由于PS/2总线是开集输出的，所以CLK和DAT引脚必须由三态门驱动。



图表 9 开发板上的PS2接口

#### **代码解读**

module keyboard(inout tri PS2\_CLK,

                PS2\_DAT,

                input CLOCK\_50,

                start\_tx,

                reset,

                input[7:0] tx\_buf,

                output[4:0] state);

    localparam start = 5'b000001,

                data = 5'b000010,

              parity = 5'b000100,

                stop = 5'b001000,

               delay = 6'b010000,

                idel = 5'b100000;

    reg clk\_out, dat\_out;

    wire clk\_in, dat\_in;

    wire txen, rxen;

    wire clock, half\_clk;

    reg[2:0] bit\_cnt;

    reg[1:0] bit\_state;

    reg[4:0] curr\_state, next\_state;

    divider #(24)(.clk\_in(CLOCK\_50), .clk\_out(clock));  // 1us

    divider #(9)(.clk\_in(clock), .clk\_out(half\_clk));  // 20us

    assign state = curr\_state;

    ps2\_bus ps2\_bus(

    .PS2\_CLK(PS2\_CLK),

    .PS2\_DAT(PS2\_DAT),

    .clock(clock),

    .clk\_in(clk\_in),

    .clk\_out(clk\_out),

    .dat\_in(dat\_in),

    .dat\_out(dat\_out)

    );

    always @ (posedge clk\_out)

        if (curr\_state != data)

            bit\_cnt <= 0;

        else

            bit\_cnt <= bit\_cnt + 1;

    always @ (posedge clock)

        if (reset)

            curr\_state <= idel;

        else

            curr\_state <= next\_state;

    always @ \*

    case (curr\_state)

        idel:

            if (start\_tx)

                next\_state <= start;

            else

                next\_state <= idel;

        start:

            if (bit\_state > 2)

                next\_state <= data;

            else

                next\_state <= start;

        data:

            if (bit\_cnt == 7)

                next\_state <= parity;

            else

                next\_state <= data;

        parity:

            if (bit\_state > 2)

                next\_state <= stop;

            else

                next\_state <= parity;

        stop:

            if (bit\_state > 2)

                next\_state <= delay;

            else

                next\_state <= stop;

        delay:

            next\_state <= delay;

        default:

            next\_state <= idel;

    endcase

    always @ (posedge half\_clk)

        begin

            if (reset | bit\_state > 2)

                bit\_state <= 0;

            else

                bit\_state <= bit\_state + 1;

            if (start\_tx)

                case (bit\_state)

                    1: clk\_out <= 0;

                    3: clk\_out <= 1;

                    default: clk\_out <= clk\_out;

                endcase

            else clk\_out <= 1;

            if (bit\_state)

                dat\_out <= dat\_out;

            else if (start\_tx)

                case (curr\_state)

                    idel: dat\_out <= 1;

                    start: dat\_out <= 0;

                    data: dat\_out <= tx\_buf[bit\_cnt];

                    parity: dat\_out <= !(^tx\_buf);

                    stop: dat\_out <= 1;

                    delay: dat\_out <= 1;

                    default: dat\_out <= dat\_out;

                endcase

            else dat\_out <= 1;

        end

endmodule

module ps2\_bus(

  inout tri PS2\_CLK, PS2\_DAT,

  input clock, clk\_out, dat\_out,

  output reg clk\_in, dat\_in

);

assign PS2\_CLK = clk\_out ? 1'bZ : 1'b0;

assign PS2\_DAT = dat\_out ? 1'bZ : 1'b0;

always @ (posedge clock)

  begin

   clk\_in <= PS2\_CLK;

   dat\_in <= PS2\_DAT;

  end

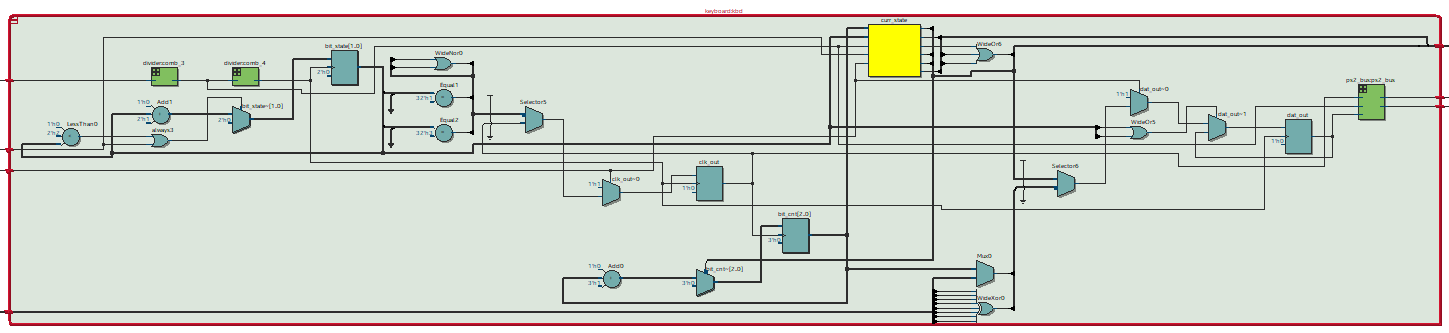
endmodule

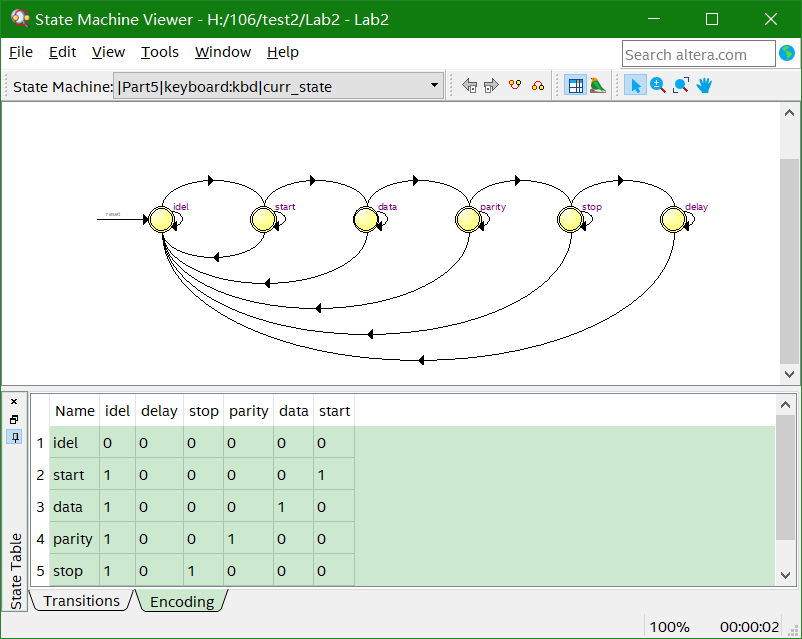
## **设计验证**

使用双踪示波器观察PS/2总线的波形：



图表 10 发送一字节的波形





图表 11 发送FSM

## 总结

为了让电脑识别到键盘，需要在启动时不断发送0xaa，直到主机向设备写指令。设备读取成功后也要不断发送ack直到成功。

状态机必须使用局部变量。

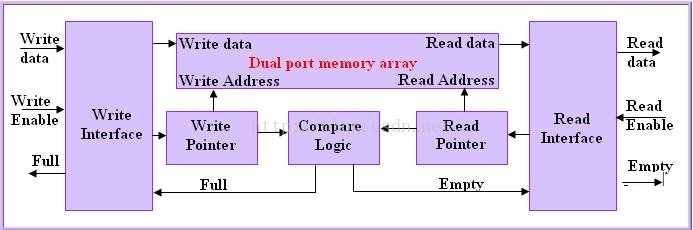
# PART 5 FIFO

## 题目分析

PS/2串口发送扫描码是需要时间的，但用户可以同时按下多个键，因此每次扫描都要先将动作存入FIFO。

## 原理分析

简单来说，FIFO就是一个双口RAM加上两个读写指针。一般FIFO使用循环指针(计数溢出自动归零)。一般可以称写指针为头head，读指针为尾tail。初始化时，读写指针指向同一数据地址。



图表 12 FIFO

#### **代码解读**

`define BUF\_WIDTH 4 *//地址宽度为3+1，*

`define BUF\_SIZE 8 *//数据个数，FIFO深度*

module fifo\_counter( clk,rst\_n,buf\_in,buf\_out,wr\_en,rd\_en,buf\_empty,buf\_full,fifo\_cnt);

input clk,rst\_n;

input wr\_en,rd\_en;

input [7:0] buf\_in; *// data input to be pushed to buffer*

output reg [7:0] buf\_out; *// port to output the data using pop.*

output wire buf\_empty,buf\_full; *// buffer empty and full indication*

output reg [`BUF\_WIDTH-1:0] fifo\_cnt; *// number of data pushed in to buffer*

*//写入数据等于8时，满*

reg [`BUF\_WIDTH-2:0] rd\_ptr,wr\_ptr; *//这个很重要，数据指针3位宽度，0-7索引，8个数据深度，循环指针0-7-0-7*

reg [7:0] buf\_mem[0:`BUF\_SIZE-1];

*//判断空满*

assign buf\_empty = (fifo\_cnt == 0); *//buf\_empty若是reg类型则错，不能使用assign持续赋值*

assign buf\_full = (fifo\_cnt == `BUF\_SIZE);

always @(posedge clk or negedge rst\_n)begin

if(!rst\_n)

fifo\_cnt <= 0;

else if((!buf\_full&&wr\_en)&&(!buf\_empty&&rd\_en)) *//同时读写，数量不变*

fifo\_cnt <= fifo\_cnt;

else if(!buf\_full && wr\_en) *//写数据*

fifo\_cnt <= fifo\_cnt + 1;

else if(!buf\_empty && rd\_en) *//读数据*

fifo\_cnt <= fifo\_cnt-1;

else

fifo\_cnt <= fifo\_cnt;

end

always @(posedge clk or negedge rst\_n) begin *//读数据*

if(!rst\_n)

buf\_out <= 0;

if(rd\_en && !buf\_empty)

buf\_out <= buf\_mem[rd\_ptr];

end

always @(posedge clk) begin

if(wr\_en && !buf\_full)

buf\_mem[wr\_ptr] <= buf\_in;

end

always @(posedge clk or negedge rst\_n) begin

if(!rst\_n) begin

wr\_ptr <= 0;

rd\_ptr <= 0;

end

else begin

if(!buf\_full && wr\_en)

wr\_ptr <= wr\_ptr + 1;

if(!buf\_empty && rd\_en)

rd\_ptr <= rd\_ptr + 1;

end

end

endmodule

**top：**

//=======================================================

// REG/WIRE declarations

//=======================================================

wire[3:0] cnt

//=======================================================

// Structural coding

//=======================================================

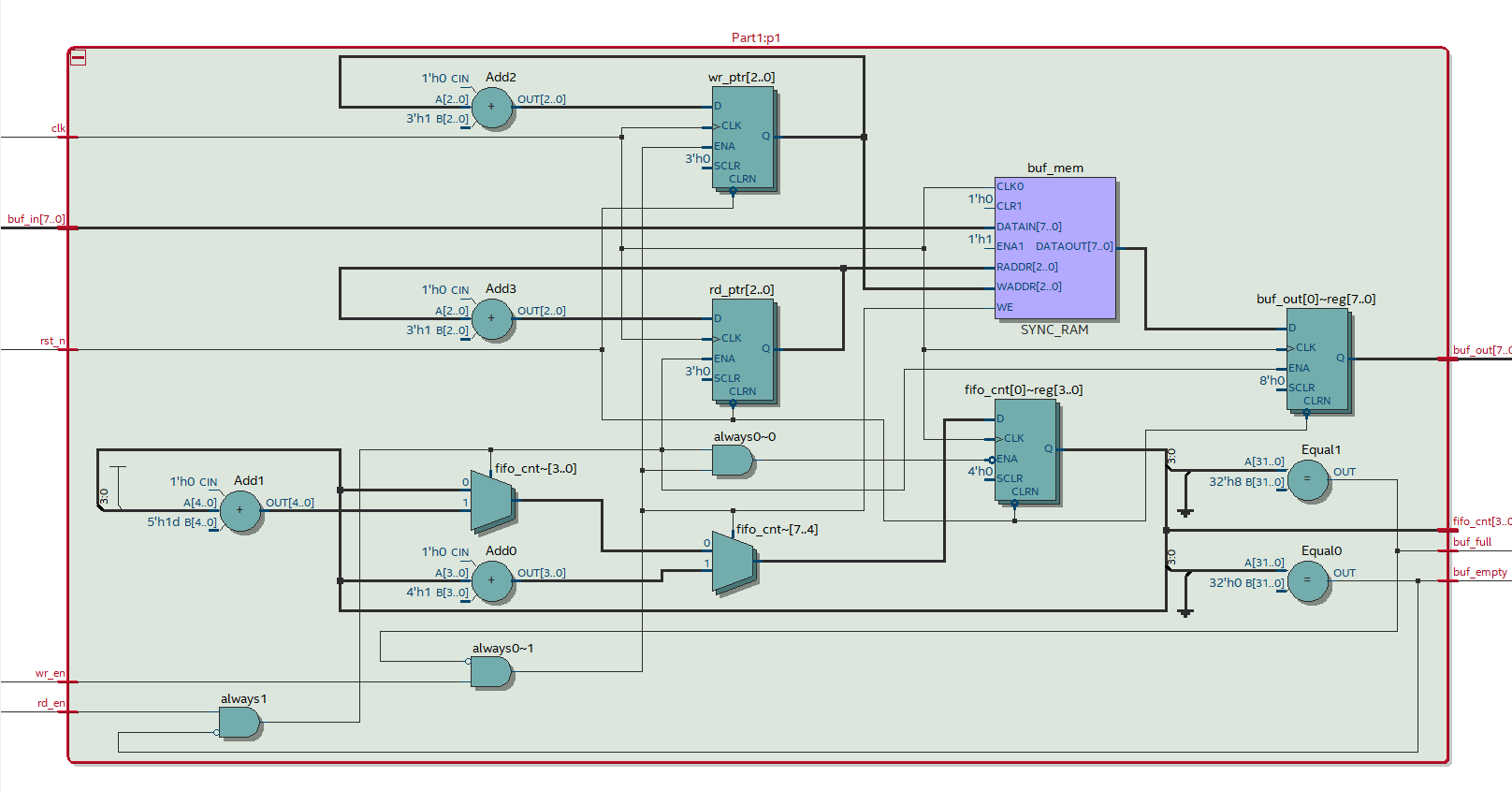
Part1 p1(KEY[0], KEY[3], SW[7:0], LEDR[7:0], SW[9], SW[8], LEDR[9], LEDR[8], cnt);

seg7 u0(CLOCK\_50, cnt, HEX0);

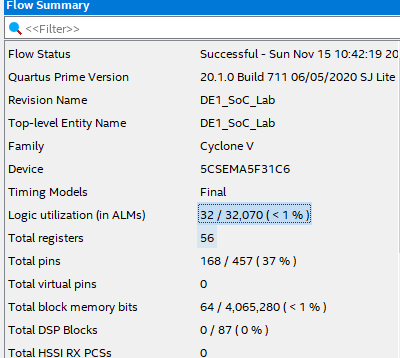
assign {HEX1, HEX2, HEX3, HEX4, HEX5} = ~35'h0;

## **设计验证**

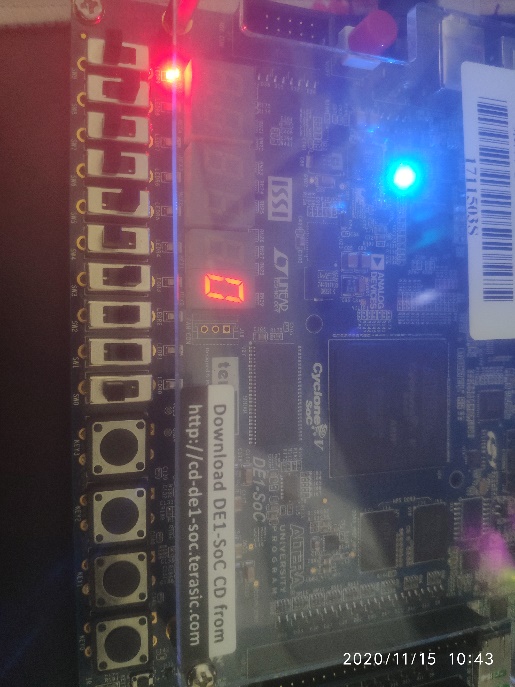
DE1-SoC开发板共有10个开关，使用其中8个开关作为FIFO输入、1个读使能1个写使能，输出连接到8个LED，使用按钮作为clk。



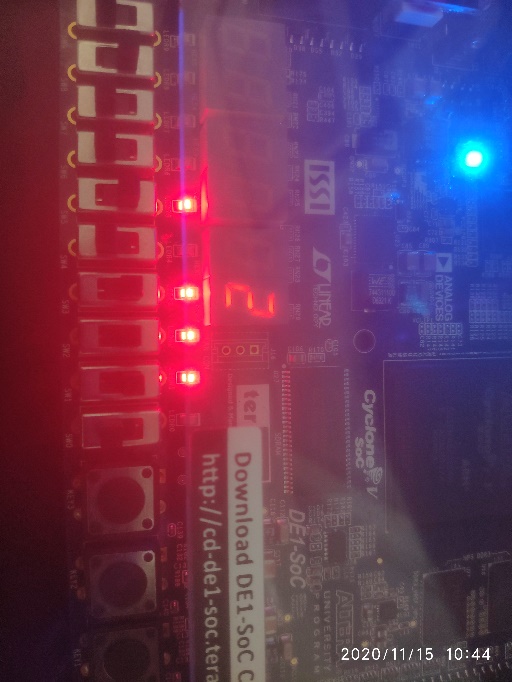
图表 13 rtl



图表 14



图表 15 复位后



图表 16 读取后

## 总结

同样的数字电路，有许多不同风格的写法。

# PART 6 发送扫描码

## 题目分析

将矩阵扫描得到的结果通过PS/2发送到电脑。

## 原理分析

考虑到每个按键对应一至八个扫描码，放入ram是比较方便的选择。若矩阵键盘有6\*23个键，则ram空间应至少为6\*23\*8 bytes（约1KB）。若扫描码不足8字节，应在空位补零。

RAM的地址线可使用{COL，ROW}的方式拼接。

每次扫描到有键按下后立即存入FIFO，而PS/2发送完成后立即从FIFO读取下一个键值。每次发送一字节后递增在RAM的指针，PS/2的发送状态机在检测到RAM数据为0后跳转到空闲状态。

#### **代码解读**

module matrix(

    input clk\_50,

    input[4:0] COL,

    output reg[4:0] ROW,

    output reg key\_down

);

reg[2:0] col\_cnt, row\_cnt;

wire clock;

divider #(999) divider(clk\_50, clock);

always @ (posedge clock)

    begin

        col\_cnt <= col\_cnt + 1;

          if (!col\_cnt)

                row\_cnt <= row\_cnt + 1;

            else

                row\_cnt <= row\_cnt;

            if (COL[col\_cnt])

                key\_down <= 1;

            else

                key\_down <= 0;

        case(row\_cnt)

            0: ROW      <= 4'b1110;

            1: ROW      <= 4'b1101;

            2: ROW      <= 4'b1011;

            3: ROW      <= 4'b0111;

            default:ROW <= 4'b1111;

        endcase

    end

endmodule

**top：**

wire key\_down, txen;

wire[7:0] key\_value;

keyboard kbd(

  .PS2\_CLK(PS2\_KBCLK),

  .PS2\_DAT(PS2\_KBDAT),

  .CLOCK\_50(CLOCK\_50),

  .start\_tx(txen),

  .reset(!KEY[0]),

  .tx\_buf(key\_value),

  .state(LEDR[4:0]));

matrix mat(

    .COL(GPIO[3:0]),

    .ROW(GPIO[7:4]),

    .clk\_50(CLOCK\_50),

    .key\_down(key\_down));

fifo fifo(

    .clk(CLOCK\_50),

    .rst\_n(!KEY[0]),

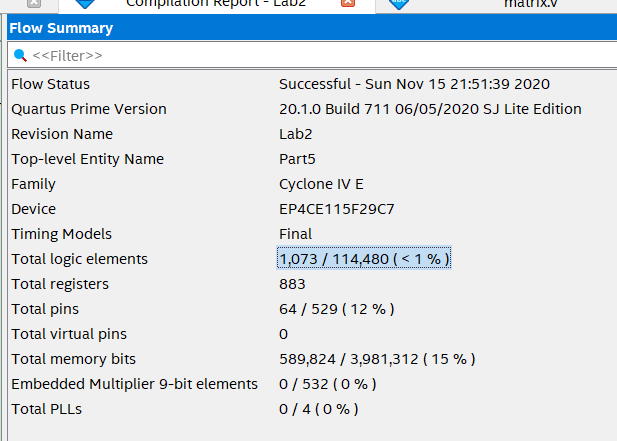
    .buf\_in(GPIO[7:0]),

    .buf\_out(key\_value),

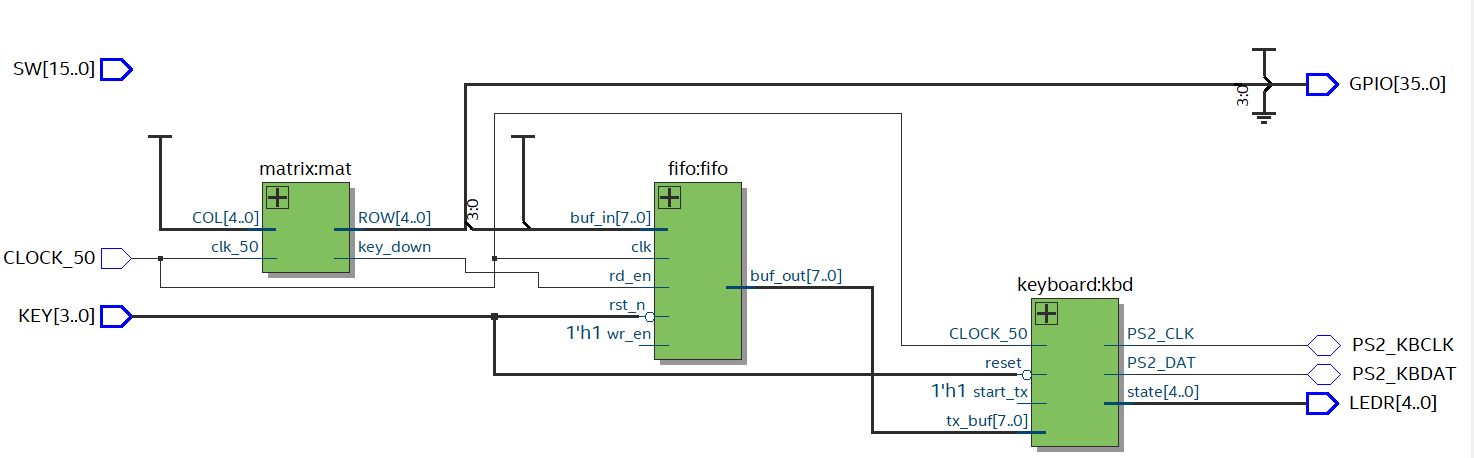
    .wr\_en(txen),

    .rd\_en(key\_down));

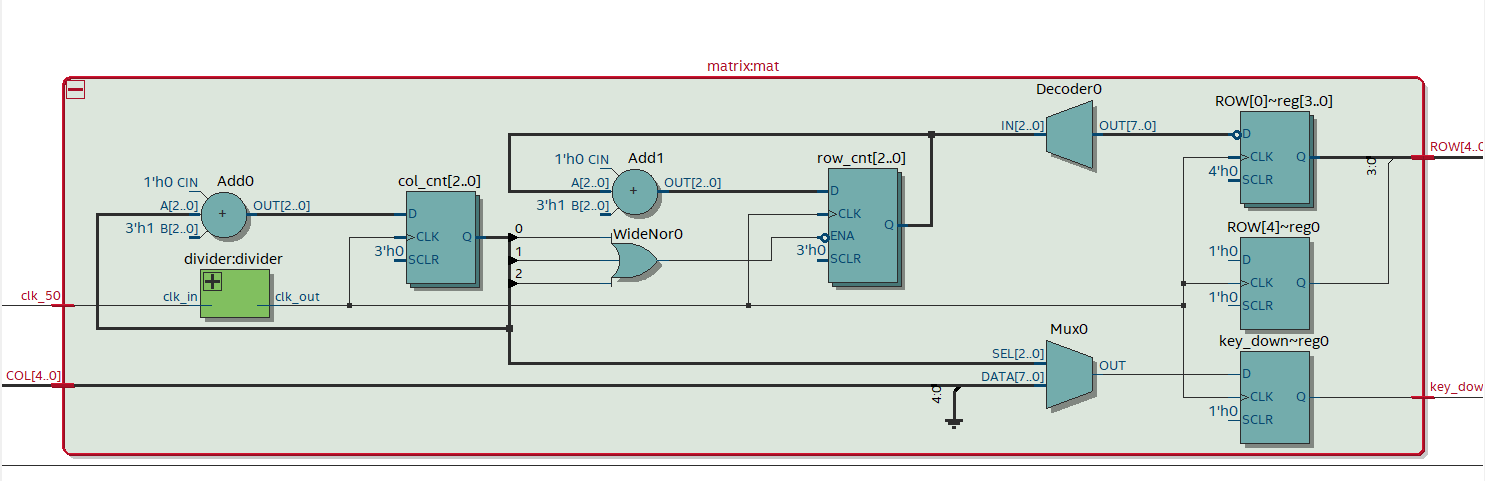
## **设计验证**



图表 17



图表 18 前几部分的综合



图表 19 矩阵扫描

## 总结

很遗憾，面对串行的PS/2总线，FPGA无法发挥并行计算的优势. ☹

# PART 7 点亮LED

## 题目分析

从主机读取CapsLock和NumLock并通过板载LED实时显示。

## 原理分析

主机识别到键盘后就会发送LED状态。

#### **代码解读**

## **设计验证**

## 总结

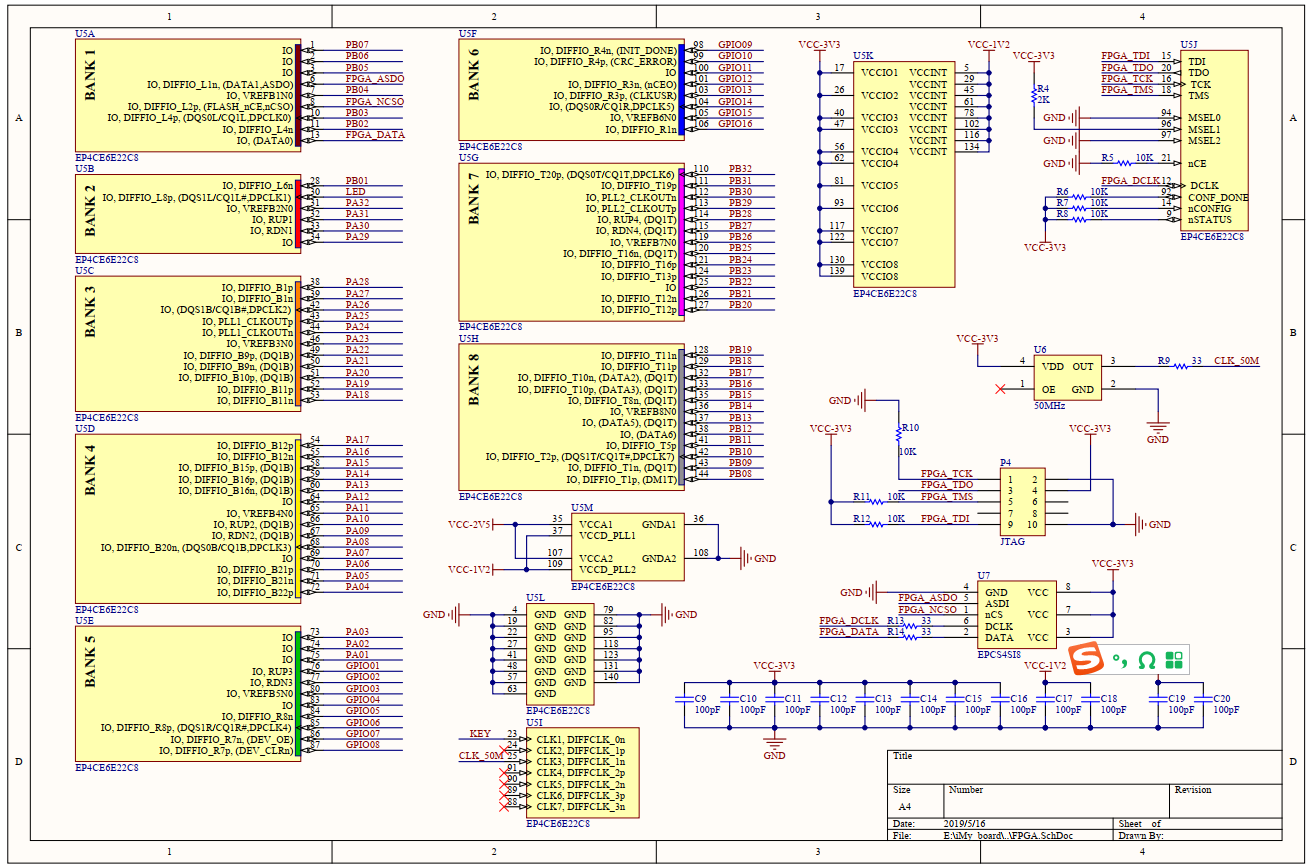
正在努力调试中……

# PART 8 最小系统

## 题目分析

最终本键盘选则拥有6K逻辑单元和144引脚的EP4CE6E22C8N来实现。

## 原理分析



图表 20 Cyclone IV E最小系统原理图

#### **代码解读**

无

## **设计验证**

还没画完……

## 总结