|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **学 期** | 2020-2021学年第1学期 | | | | **实验日期** | | 2020.12.8 | | |
| **学 院** | 信息学部 | | | | **专 业** | | 计算机类 | | |
| **班 级** | 190741 | | **学 号** | | 19030419 | | **姓 名** | | 王昕 |
| **组 号** | 24 | | **学 号** | | 19074104 | | **姓 名** | | 王佳玮 |
| **评 阅 内 容** | | | | | | | | | |
| **实验预习** | | **实验原理** | | **详细设计** | | **实验结果** | | **成 绩** | |
|  | |  | |  | |  | |  | |
|  | | | | | | | | | |
| **题 目** | 实验11：键盘扫描电路设计 | | | | | | | | |

**一．实验目的**

1. 理解键盘扫描电路工作原理。
2. 掌握键盘扫描的控制方式。

**二．实验任务及要求**

任务：实现4 × 4键盘识别和连续输入

要求：

1. 系统时钟为50 MHz，使用之前设计的分频器模块。
2. 实现4 × 4键盘识别。
3. 将按键显示在数码管上。
4. 连续输入8位数字，不满8位时最高位之前的数码管无显示。

**三．实验原理**

图1为实验台4 × 4键盘硬件结构示意图，图中列线通过电阻接3.3 V。当键盘上没有键按下时，所有的行线和列线断开，列线I\_SWC0 ~ I\_SWC3呈高电平；当键盘上某一个键闭合时，该键所对应的行线和列线短路。例如，6号键按下时，I\_SWC2与O\_SWR1短路，此时I\_SWC2电平由O\_SWR1电平决定。如果把列线接到FPGA的输入端口，行线接到FPGA的输出端口，在FPGA的控制下，使行线O\_SWR0为低电平，其余三根行线O\_SWR1、O\_SWR2、O\_SWR3都为高电平。然后通过FPGA输入端口读列线的状态，如果I\_SWC0、I\_SWC1、I\_SWC2、I\_SWC3都为高电平，则O\_SWR0这一行没有键闭合；如果读出的行线状态不全为高电平，则低电平的列线和O\_SWR0相交键处于闭合状态。如果O\_SWR0这一行没有键闭合，接着使行线O\_SWR1为低电平，其余行线为高电平。用同样的方法检查O\_SWR1这一行，以此类推，最后令行线O\_SWR3为低电平，其余的行线为高电平，检查O\_SWR3这一行是否有键闭合。即循环向O\_SWR0、O\_SWR1、O\_SWR2、O\_SWR3输出0111、1011、1101和1110，通过读入I\_SWC0、I\_SWC1、I\_SWC2和I\_SWC3的状态，就可以判断哪个键按键按下。

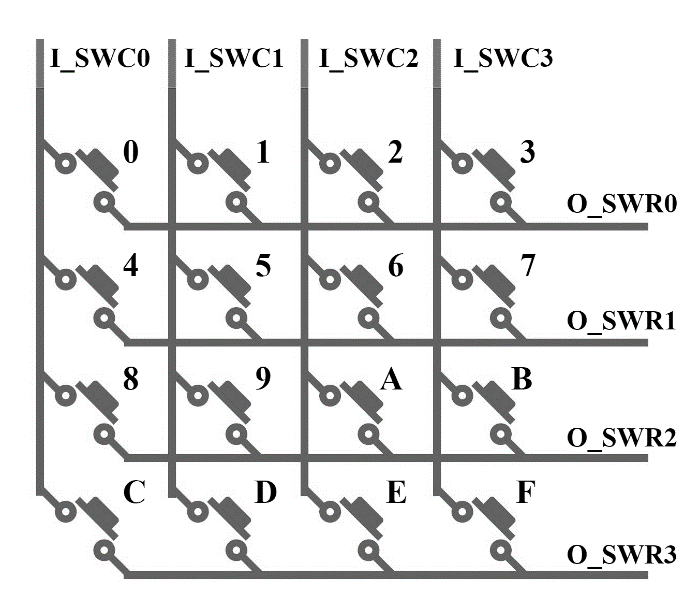


图1　4 × 4键盘硬件结构示意图

**四．详细设计**

1. 电路实现

根据要求采用相应的Verilog HDL代码来实现。本实验需要用键盘扫描电路作为整个电路的基础，并以此扩展延伸。扫描电路的代码逻辑与实验原理描述相同，定义4位输入变量SWR和SWC作为行扫描和列扫描的变量，16位的keys作为键盘扫描的输出变量。初始化SWR为1110，电路从最低一行开始扫描，定义4位寄存器型变量keys3、keys2、keys1和keys0分别对应SWR3 – SWR0，并将这四个变量连接成16位的keys输出变量作为扫描的结果（每个按键是否按下）。在always块描述每当时钟信号clk为上升沿时，SWR左移循环计数，从第一行到第四行再回到第一行循环扫描，在case块中将每行扫描时所检测到的列SWC对应位取反，并将第一行到第四行的取反结果依次赋值给keys0 ~ keys3，最终再将keys0 ~ keys3组合成输出变量keys，从而扫描出哪些按键被按下。

键盘扫描电路Verilog HDL代码：

module frequency\_divider\_074124(clk\_50mhz,reset,clk\_1hz,clk\_2hz,clk\_50hz,clk\_1khz);

input reset,clk\_50mhz;

output clk\_1hz,clk\_2hz,clk\_50hz,clk\_1khz;

reg clk\_1hz,clk\_2hz,clk\_50hz,clk\_1khz;

reg[31:0]cnt\_1hz,cnt\_2hz,cnt\_50hz,cnt\_1khz;

parameter N\_1hz=50\_000\_000;

parameter N\_2hz=25\_000\_000;

parameter N\_50hz=1\_000\_000;

parameter N\_1khz=50\_000;

//parameter N\_1hz=50;

//parameter N\_2hz=24;

//parameter N\_50hz=10;

//parameter N\_1khz=3;

always@(posedge clk\_50mhz)

begin

if(!reset)

begin

cnt\_1hz<=1'b0;

cnt\_2hz<=1'b0;

cnt\_50hz<=1'b0;

cnt\_1khz<=1'b0;

clk\_1hz<=1'b0;

clk\_2hz<=1'b0;

clk\_50hz<=1'b0;

clk\_1khz<=1'b0;

end

//1hz

if(cnt\_1hz<N\_1hz/2-1)

cnt\_1hz<=cnt\_1hz+1'b1;

else

begin

cnt\_1hz<=1'b0;

clk\_1hz<=~clk\_1hz;

end

//2hz

if(cnt\_2hz<N\_2hz/2-1)

cnt\_2hz<=cnt\_2hz+1'b1;

else

begin

cnt\_2hz<=1'b0;

clk\_2hz<=~clk\_2hz;

end

//50hz

if(cnt\_50hz<N\_50hz/2-1)

cnt\_50hz<=cnt\_50hz+1'b1;

else

begin

cnt\_50hz<=1'b0;

clk\_50hz<=~clk\_50hz;

end

//1khz

if(cnt\_1khz<N\_1khz/2-1)

cnt\_1khz<=cnt\_1khz+1'b1;

else

begin

cnt\_1khz<=1'b0;

clk\_1khz<=~clk\_1khz;

end

end

Endmodule

module jpsm\_074124 (clk,en,swc,key,swr,flag);

input clk,en;

input [3:0]swc;

output [3:0]key,swr;

output flag;

reg [3:0]key,swr;

reg flag;

always @(posedge clk)

begin

if(~en) swr=4'b1110;

else

begin

case({swr,swc})

8'b1110\_1110:{key,flag}=5'b00001;

8'b1110\_1101:{key,flag}=5'b00011;

8'b1110\_1011:{key,flag}=5'b00101;

8'b1110\_0111:{key,flag}=5'b00111;

8'b1101\_1110:{key,flag}=5'b01001;

8'b1101\_1101:{key,flag}=5'b01011;

8'b1101\_1011:{key,flag}=5'b01101;

8'b1101\_0111:{key,flag}=5'b01111;

8'b1011\_1110:{key,flag}=5'b10001;

8'b1011\_1101:{key,flag}=5'b10011;

8'b1011\_1011:{key,flag}=5'b10101;

8'b1011\_0111:{key,flag}=5'b10111;

8'b0111\_1110:{key,flag}=5'b11001;

8'b0111\_1101:{key,flag}=5'b11011;

8'b0111\_1011:{key,flag}=5'b11101;

8'b0111\_0111:{key,flag}=5'b11111;

default:{key,flag}={key,1'b0};

endcase

case(swr)

4'b1110:swr=4'b1101;

4'b1101:swr=4'b1011;

4'b1011:swr=4'b0111;

4'b0111:swr=4'b1110;

default:swr=4'b1110;

endcase

end

end

Endmodule

module dtsm\_074124(clk,ds,select);

input clk;

output [7:0]ds;

output [2:0]select;

reg [7:0]ds;

reg [2:0]select;

reg [2:0]state;

parameter s0=0,s1=1,s2=2,s3=3,s4=4,s5=5,s6=6,s7=7;

always @ (state)

begin

case(state)

s0: {select[2:0],ds[7:0]}=11'b000\_1111\_1110;

s1: {select[2:0],ds[7:0]}=11'b001\_1111\_1101;

s2: {select[2:0],ds[7:0]}=11'b010\_1111\_1011;

s3: {select[2:0],ds[7:0]}=11'b011\_1111\_0111;

s4: {select[2:0],ds[7:0]}=11'b100\_1110\_1111;

s5: {select[2:0],ds[7:0]}=11'b101\_1101\_1111;

s6: {select[2:0],ds[7:0]}=11'b110\_1011\_1111;

s7: {select[2:0],ds[7:0]}=11'b111\_0111\_1111;

default: {select[2:0],ds[7:0]}=11'b000\_1111\_1111;

endcase

end

always @ (posedge clk)

begin

case(state)

s0:state<=s1;

s1:state<=s2;

s2:state<=s3;

s3:state<=s4;

s4:state<=s5;

s5:state<=s6;

s6:state<=s7;

s7:state<=s0;

default:state<=s0;

endcase

end

endmodule

module multiplexers\_074124(out,en,sel,in1,in2,in3,in4,in5,in6,in7,in8);

input en;

input [2:0]sel;

input [3:0]in1,in2,in3,in4,in5,in6,in7,in8;

output [3:0]out;

reg [3:0]out;

always @(sel or in8 or in7 or in6 or in5 or in4 or in3 or in2 or in1 or en)

begin

if(~en)

out=4'b1111;

Else

case(sel)

3'b000:out=in1;

3'b001:out=in2;

3'b010:out=in3;

3'b011:out=in4;

3'b100:out=in5;

3'b101:out=in6;

3'b110:out=in7;

3'b111:out=in8;

default:out=4'bx;

endcase

end

Endmodule

module led\_g074124 (data,a,b,c,d,e,f,g);

input [3:0]data;

output a,b,c,d,e,f,g;

reg a,b,c,d,e,f,g;

always @(data)

begin

case(data)

4'b0000:{a,b,c,d,e,f,g}=7'b1111110;

4'b0001:{a,b,c,d,e,f,g}=7'b0110000;

4'b0010:{a,b,c,d,e,f,g}=7'b1101101;

4'b0011:{a,b,c,d,e,f,g}=7'b1111001;

4'b0100:{a,b,c,d,e,f,g}=7'b0110011;

4'b0101:{a,b,c,d,e,f,g}=7'b1011011;

4'b0110:{a,b,c,d,e,f,g}=7'b1011111;

4'b0111:{a,b,c,d,e,f,g}=7'b1110000;

4'b1000:{a,b,c,d,e,f,g}=7'b1111111;

4'b1001:{a,b,c,d,e,f,g}=7'b1111011

4'b1010:{a,b,c,d,e,f,g}=7'b1110111;//A

4'b1011:{a,b,c,d,e,f,g}=7'b0011111;//b

4'b1100:{a,b,c,d,e,f,g}=7'b1001110;//C

4'b1101:{a,b,c,d,e,f,g}=7'b0111101;//d

4'b1110:{a,b,c,d,e,f,g}=7'b1001111;//E

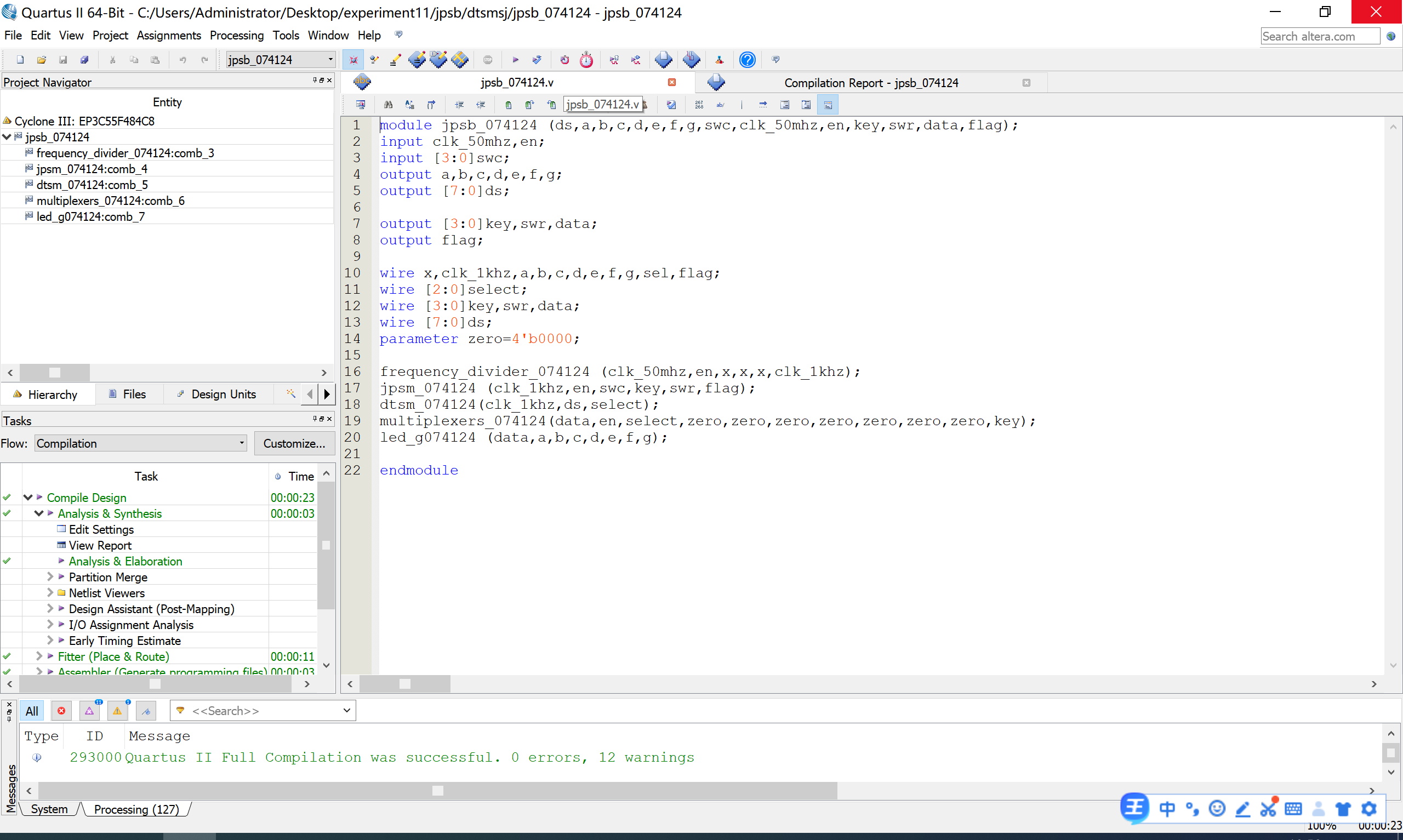
4'b1111:{a,b,c,d,e,f,g}=7'b1000111;//F

default:{a,b,c,d,e,f,g}=7'b0000000;

endcase

end

Endmodule

顶层：

拓展：

module shift\_register\_8\_074124(clk,clrn,dsl,dsr,s0,s1,a,b,c,d,e,f,g,h,qa,qb,qc,qd,qe,qf,qg,qh);

input clk,clrn,s0,s1;

input [3:0]a,b,c,d,e,f,g,h,dsl,dsr;

output [3:0]qa,qb,qc,qd,qe,qf,qg,qh;

wire [3:0]qa,qb,qc,qd,qe,qf,qg,qh;

shift\_register\_074124 (clk,clrn,qd,dsr,s0,s1,e,f,g,h,qe,qf,qg,qh);

shift\_register\_074124 (clk,clrn,dsl,qe,s0,s1,a,b,c,d,qa,qb,qc,qd);

Endmodule

module shift\_register\_074124(clk,clrn,dsl,dsr,s0,s1,a,b,c,d,qa,qb,qc,qd);

input clk,clrn,s0,s1;

input [3:0]a,b,c,d,dsl,dsr;

output [3:0]qa,qb,qc,qd;

reg [3:0]qa,qb,qc,qd;

always @ (posedge clk or negedge clrn)

begin

if(clrn==0) {qd,qc,qb,qa}=4'b0000\_0000\_0000\_0000;

else

case({s1,s0})

2'b00:{qd,qc,qb,qa}<={qd,qc,qb,qa};

2'b11:{qd,qc,qb,qa}<={d,c,b,a};

2'b01:{qd,qc,qb,qa}<={dsr,qd,qc,qb};

2'b10:{qd,qc,qb,qa}<={qc,qb,qa,dsl};

endcase

end

Endmodule

module multiplexers\_074124(out,en,sel,in1,in2,in3,in4,in5,in6,in7,in8);

input en;

input [2:0]sel;

input [3:0]in1,in2,in3,in4,in5,in6,in7,in8;

output [3:0]out;

reg [3:0]out;

always @(sel or in8 or in7 or in6 or in5 or in4 or in3 or in2 or in1 or en)

begin

if(~en)

out=4'b1111;

else

case(sel)

3'b000:out=in1;

3'b001:out=in2;

3'b010:out=in3;

3'b011:out=in4;

3'b100:out=in5;

3'b101:out=in6;

3'b110:out=in7;

3'b111:out=in8;

default:out=4'bx;

endcase

end

Endmodule

module led\_g074124 (data,a,b,c,d,e,f,g);

input [3:0]data;

output a,b,c,d,e,f,g;

reg a,b,c,d,e,f,g;

always @(data)

begin

case(data)

4'b0000:{a,b,c,d,e,f,g}=7'b1111110;

4'b0001:{a,b,c,d,e,f,g}=7'b0110000;

4'b0010:{a,b,c,d,e,f,g}=7'b1101101;

4'b0011:{a,b,c,d,e,f,g}=7'b1111001;

4'b0100:{a,b,c,d,e,f,g}=7'b0110011;

4'b0101:{a,b,c,d,e,f,g}=7'b1011011;

4'b0110:{a,b,c,d,e,f,g}=7'b1011111;

4'b0111:{a,b,c,d,e,f,g}=7'b1110000;

4'b1000:{a,b,c,d,e,f,g}=7'b1111111;

4'b1001:{a,b,c,d,e,f,g}=7'b1111011;

4'b1010:{a,b,c,d,e,f,g}=7'b1110111;//A

4'b1011:{a,b,c,d,e,f,g}=7'b0011111;//b

4'b1100:{a,b,c,d,e,f,g}=7'b1001110;//C

4'b1101:{a,b,c,d,e,f,g}=7'b0111101;//d

4'b1110:{a,b,c,d,e,f,g}=7'b1001111;//E

4'b1111:{a,b,c,d,e,f,g}=7'b1000111;//F

default:{a,b,c,d,e,f,g}=7'b0000000;

endcase

end

Endmodule

module led\_g074124 (data,a,b,c,d,e,f,g);

input [3:0]data;

output a,b,c,d,e,f,g;

reg a,b,c,d,e,f,g;

always @(data)

begin

case(data)

4'b0000:{a,b,c,d,e,f,g}=7'b1111110;

4'b0001:{a,b,c,d,e,f,g}=7'b0110000;

4'b0010:{a,b,c,d,e,f,g}=7'b1101101;

4'b0011:{a,b,c,d,e,f,g}=7'b1111001;

4'b0100:{a,b,c,d,e,f,g}=7'b0110011;

4'b0101:{a,b,c,d,e,f,g}=7'b1011011;

4'b0110:{a,b,c,d,e,f,g}=7'b1011111;

4'b0111:{a,b,c,d,e,f,g}=7'b1110000;

4'b1000:{a,b,c,d,e,f,g}=7'b1111111;

4'b1001:{a,b,c,d,e,f,g}=7'b1111011;

4'b1010:{a,b,c,d,e,f,g}=7'b1110111;//A

4'b1011:{a,b,c,d,e,f,g}=7'b0011111;//b

4'b1100:{a,b,c,d,e,f,g}=7'b1001110;//C

4'b1101:{a,b,c,d,e,f,g}=7'b0111101;//d

4'b1110:{a,b,c,d,e,f,g}=7'b1001111;//E

4'b1111:{a,b,c,d,e,f,g}=7'b1000111;//F

default:{a,b,c,d,e,f,g}=7'b0000000;

endcase

end

Endmodule

module jpsm\_074124 (clk\_1khz,en,swc,key,swr,flag);

input clk\_1khz,en;

input [3:0]swc;

output [3:0]key,swr;

output flag;

reg [3:0]key,swr;

reg flag;

always @(posedge clk\_1khz)

begin

if(~en) swr=4'b1110;

else

begin

case({swr,swc})

8'b1110\_1110:{key,flag}=5'b00001;

8'b1110\_1101:{key,flag}=5'b00011;

8'b1110\_1011:{key,flag}=5'b00101;

8'b1110\_0111:{key,flag}=5'b00111;

8'b1101\_1110:{key,flag}=5'b01001;

8'b1101\_1101:{key,flag}=5'b01011;

8'b1101\_1011:{key,flag}=5'b01101;

8'b1101\_0111:{key,flag}=5'b01111;

8'b1011\_1110:{key,flag}=5'b10001;

8'b1011\_1101:{key,flag}=5'b10011;

8'b1011\_1011:{key,flag}=5'b10101;

8'b1011\_0111:{key,flag}=5'b10111;

8'b0111\_1110:{key,flag}=5'b11001;

8'b0111\_1101:{key,flag}=5'b11011;

8'b0111\_1011:{key,flag}=5'b11101;

8'b0111\_0111:{key,flag}=5'b11111;

default:{key,flag}={key,1'b0};

endcase

case(swr)

4'b1110:swr=4'b1101;

4'b1101:swr=4'b1011;

4'b1011:swr=4'b0111;

4'b0111:swr=4'b1110;

default:swr=4'b1110;

endcase

end

end

Endmodule

module frequency\_divider\_074124(clk\_50mhz,reset,clk\_1hz,clk\_2hz,clk\_50hz,clk\_1khz);

input reset,clk\_50mhz;

output clk\_1hz,clk\_2hz,clk\_50hz,clk\_1khz;

reg clk\_1hz,clk\_2hz,clk\_50hz,clk\_1khz;

reg[31:0]cnt\_1hz,cnt\_2hz,cnt\_50hz,cnt\_1khz;

parameter N\_1hz=50\_000\_000;

parameter N\_2hz=25\_000\_000;

parameter N\_50hz=1\_000\_000;

parameter N\_1khz=50\_000;

//parameter N\_1hz=50;

//parameter N\_2hz=24;

//parameter N\_50hz=10;

//parameter N\_1khz=6;

always@(posedge clk\_50mhz)

begin

if(!reset)

begin

cnt\_1hz<=1'b0;

cnt\_2hz<=1'b0;

cnt\_50hz<=1'b0;

cnt\_1khz<=1'b0;

clk\_1hz<=1'b0;

clk\_2hz<=1'b0;

clk\_50hz<=1'b0;

clk\_1khz<=1'b0;

end

//1hz

if(cnt\_1hz<N\_1hz/2-1)

cnt\_1hz<=cnt\_1hz+1'b1;

else

begin

cnt\_1hz<=1'b0;

clk\_1hz<=~clk\_1hz;

end

//2hz

if(cnt\_2hz<N\_2hz/2-1)

cnt\_2hz<=cnt\_2hz+1'b1;

else

begin

cnt\_2hz<=1'b0;

clk\_2hz<=~clk\_2hz;

end

//50hz

if(cnt\_50hz<N\_50hz/2-1)

cnt\_50hz<=cnt\_50hz+1'b1;

else

begin

cnt\_50hz<=1'b0;

clk\_50hz<=~clk\_50hz;

end

//1khz

if(cnt\_1khz<N\_1khz/2-1)

cnt\_1khz<=cnt\_1khz+1'b1;

else

begin

cnt\_1khz<=1'b0;

clk\_1khz<=~clk\_1khz;

end

end

Endmodule

module dtsm\_074124(clk,ds,select);

input clk;

output [7:0]ds;

output [2:0]select;

reg [7:0]ds;

reg [2:0]select;

reg [2:0]state;

parameter s0=0,s1=1,s2=2,s3=3,s4=4,s5=5,s6=6,s7=7;

always @ (state)

begin

case(state)

s0: {select[2:0],ds[7:0]}=11'b000\_1111\_1110;

s1: {select[2:0],ds[7:0]}=11'b001\_1111\_1101;

s2: {select[2:0],ds[7:0]}=11'b010\_1111\_1011;

s3: {select[2:0],ds[7:0]}=11'b011\_1111\_0111;

s4: {select[2:0],ds[7:0]}=11'b100\_1110\_1111;

s5: {select[2:0],ds[7:0]}=11'b101\_1101\_1111;

s6: {select[2:0],ds[7:0]}=11'b110\_1011\_1111;

s7: {select[2:0],ds[7:0]}=11'b111\_0111\_1111;

default: {select[2:0],ds[7:0]}=11'b000\_1111\_1111;

endcase

end

always @ (posedge clk)

begin

case(state)

s0:state<=s1;

s1:state<=s2;

s2:state<=s3;

s3:state<=s4;

s4:state<=s5;

s5:state<=s6;

s6:state<=s7;

s7:state<=s0;

default:state<=s0;

endcase

end

Endmodule

module decoder\_074124 (in,out0,out1);

input [3:0]in;

output [3:0]out0,out1;

reg [3:0]out0,out1;

always @ (in)

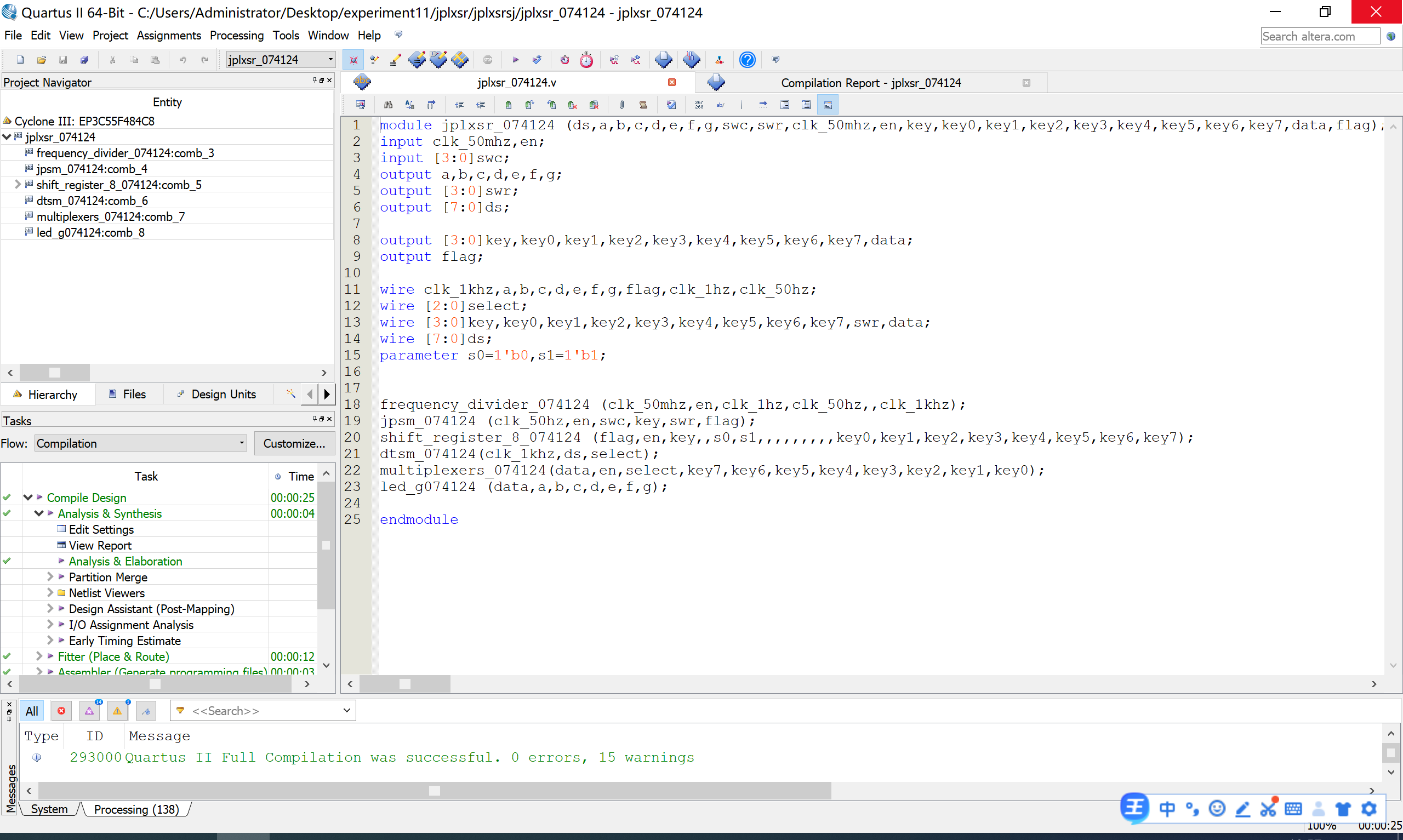
begin

out0=in%10;

out1=in/10;

end

Endmodule

顶层：

1. 仿真

键盘扫描电路仿真的波形结果如图3，clk规律变化，SWR从1110到0111变化循环扫描，手动给SWC赋值，模拟实验中在一段时间内可能按下的按键情况。SWR一轮循环走完后，输出经过扫描输出键值，如下图第一次完全扫描后，keys[3]变成高电平，说明按键按下键值为3。第二次、第三次同理。需要注意，列线的低电平到输出键值的波形变成高电平前会有一个周期的延时，原因在于clk信号为上升沿的一瞬间检测的是前一时刻的电平。

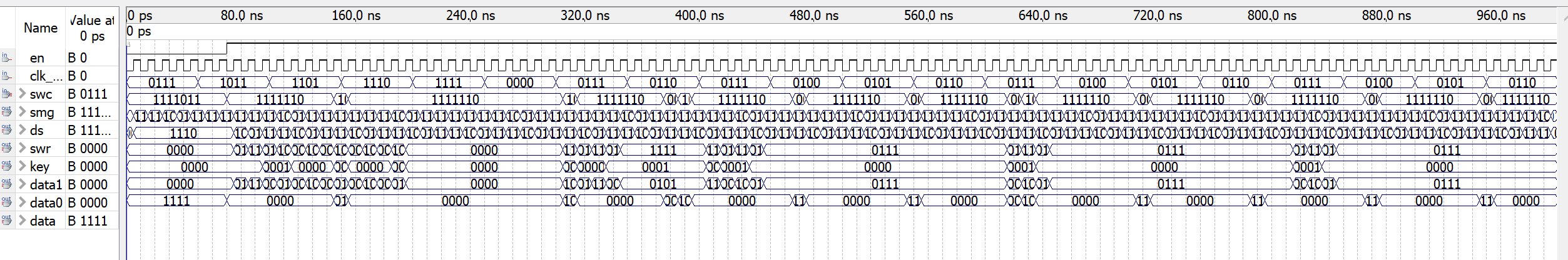


图3　键盘扫描电路仿真波形

拓展实验为每次输入一个数字，该数字都会出现在led灯的最右边，同时led灯上的数字全部左移。

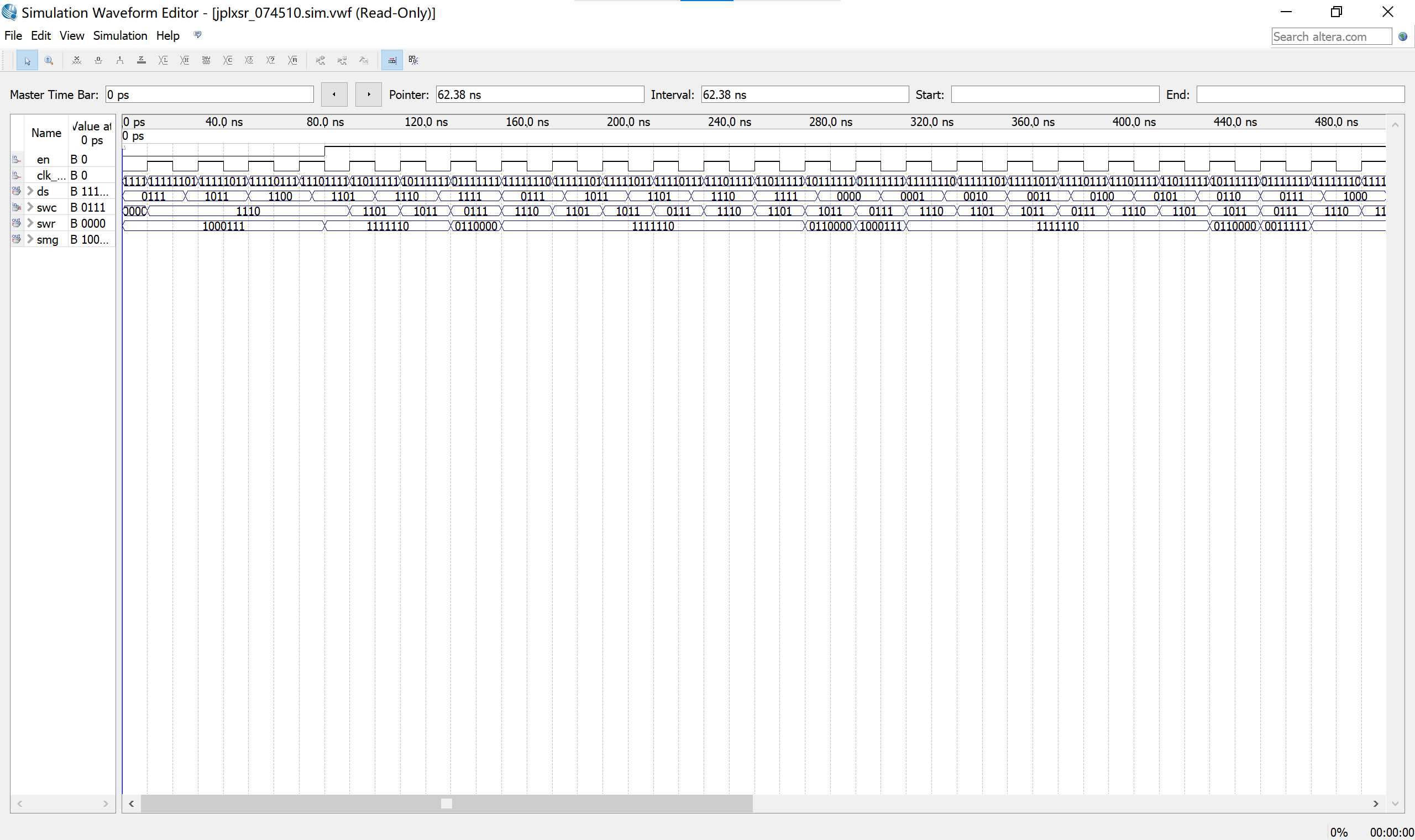
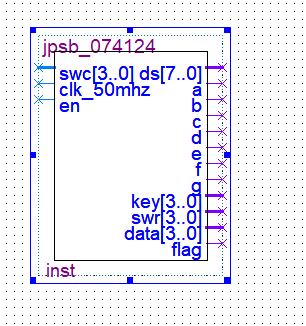


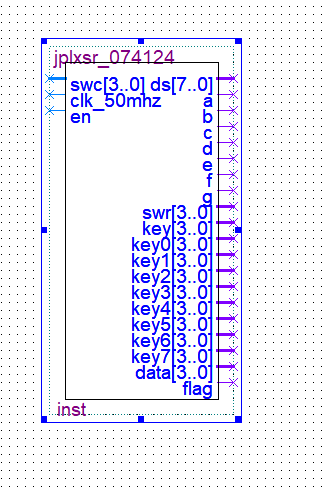
图4　键盘连续输入

1. 封装图



输入端：clk 时钟信  
SWC[3..0] 列线

En:使能端



**五．实验结果**

（1）引脚分配

本实验需要用到4 × 4键盘，四行分别对应平台端口SWR3 ~ SWR0，四列分别对应平台端口SWC3 ~ SWC0。八个数码管的位选分配到引脚DS8 ~ DS1。数码管的段选信号和小数点分配到LA ~ LH。时钟信号T1。

表1　键盘连续输入实验所选平台端口及对应引脚编号

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **平台端口** | SWR3 | SWR2 | SWR1 | SWR0 |
| **引脚编号** | PIN\_C4 | PIN\_A16 | PIN\_A15 | PIN\_A14 |
| **节点名称** | SWR[3] | SWR[2] | SWR[1] | SWR[0] |
| **平台端口** | DS8 | DS7 | DS6 | DS5 | DS4 | DS3 | DS2 |
| **引脚编号** | V16 | AA17 | U22 | V22 | W22 | Y22 | Y21 |
| **节点名称** | ds[7] | ds[6] | ds[5] | ds[4] | ds[3] | ds[2] | ds[1] |
| **平台端口** | LA | LB | LC | LD | LE | LF | LG |
| **引脚编号** | PIN\_AA20 | PIN\_W20 | PIN\_R21 | PIN\_P21 | PIN\_N21 | PIN\_N20 | PIN\_M21 |
| **节点名称** | a | b | c | d | e | f | g |
| **平台端口** | SWC3 | SWC2 | SWC1 | SWC0 | DS1 | LH | T1 |
| **引脚编号** | PIN\_A13 | PIN\_F9 | PIN\_D10 | PIN\_B10 | AB20 | M19 | PIN\_T1 |
| **节点名称** | SWC[3] | SWC[2] | SWC[1] | SWC[0] | ds[0] | h | Clk\_50mhz |

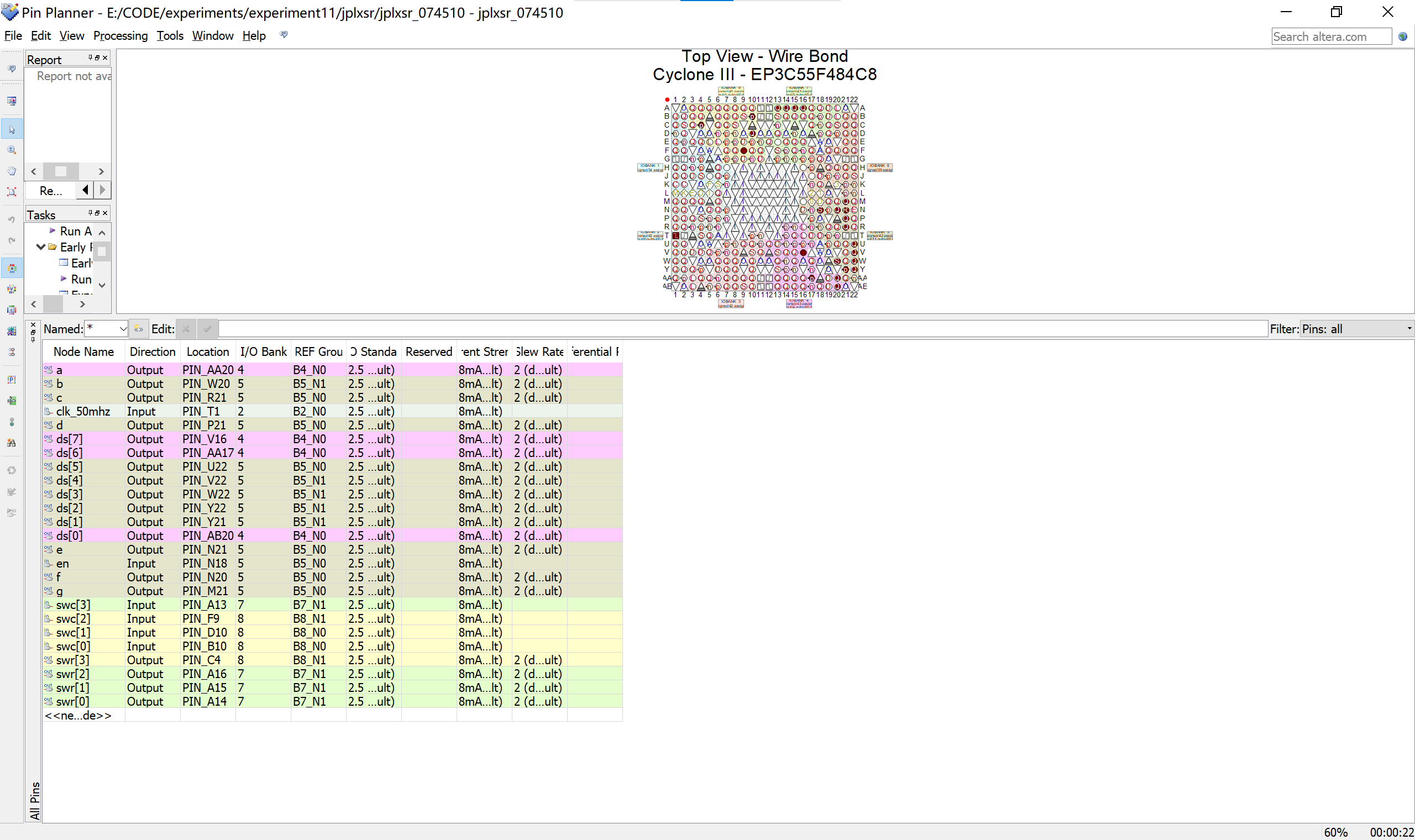


图5　引脚分配图

1. 实验现象

基础：下载后，在键盘上按下按键，当按下的是第一行第一列的按键时，数码管上显示数字1，按照实验箱上的数字提示，每按下一个按键，数码管上显示的数字都是相应的十六进制数，分别为1，2，3，4，5，6，7，8，9，a，b，c，d，e，f，g；按下按键之后，数据会保持，知道按下下一个按键，数据改变。

拓展：下载后，在键盘上按下按键，按下按键后，对应数码管的最右边将显示此次按下的键值，上次按下的键值则往左移位，从而实现了键盘的连续输入，每次输入的数字都会到最右边的数码管上，实现移位；同时未满八位时，数码管显示无效。

附图：顶层模块

