**北京科技大学 计算机与通信工程学院**

**硬件描述语言实验报告**

**专 业**： 计算机科学与工程

**班 级**： 计1503

**学生姓名**： 唐誉源

**学 号**： 41503302

**指导教师**： 齐 悦

**实验地点**： 机电楼304

**实验时间**： 2017 年 3月 2 日~ 3 月 14日

**实验成绩**：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**一、实验目的与实验要求**

**1 实验目的**

（1）熟悉常用的EDA设计工具、设计流程，掌握前端设计要点；

（2）通过简单的逻辑功能模块的设计，加深对硬件描述语言的理解和对逻辑设计的认识；

（3）理解逻辑综合的概念，熟悉Verilog的编码风格，掌握可综合的逻辑设计；

（4）训练组合逻辑设计、时序逻辑设计的能力，以及基本的设计验证能力。

**2 实验要求**

（1）在有限的实验课内学时可能难以较好完成所有实验内容，因此将实验板分组下发，要求在实验课之外抽时间完成剩余实验内容；

（2）对于每个实验都需要完成逻辑电路设计以及TestBench模块的编写，能够根据前仿真的波形图给粗结果分析；针对需要板子上验证的题目，需进一步给出相关控制逻辑的设计及板级验证说明。

（3）完成的每个实验需要在实验课内经指导教师或助教现场检查、解释波形及验证逻辑，回答指导教师提出的问题，以确保实验完成的质量；

**二、实验设备（环境）及要求**

实验室提供Windows 7操作系统下的Xilinx开发平台vivado2015.4环境，以及EGO开发板。

**三、实验内容、步骤与结果分析**

**1 实验1**

**1.1 实验内容**

设计一个周期为40个时间单位的时钟信号，其占空比为25％。使用always和initial块进行设计。将其在仿真0时刻的值初始化为0。

**1.2 实验步骤**

将电路模块置空

测试模块编辑HDL代码为：

`timescale 1ns / 1ps

module task1\_tb();

task1 u0();

reg clk;

initial begin

clk=0; //仿真0时刻的值初始化为0

end

always

begin

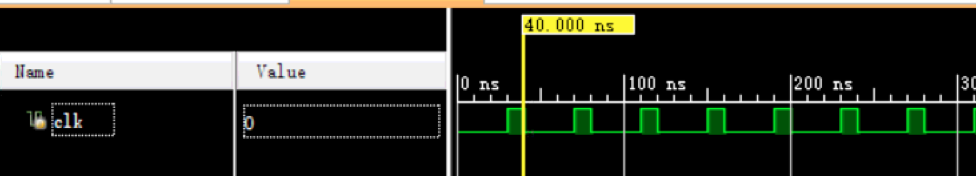
#30 clk=1; //占空比为25％

#10 clk=0;

end

endmodule

**1.3 结果分析**

****

时钟clk仿真结果图

可以观察到clk仿真0时刻的初始值为0，占空比为25%，且一个周期为40个时间单位。激励满足题目要求。

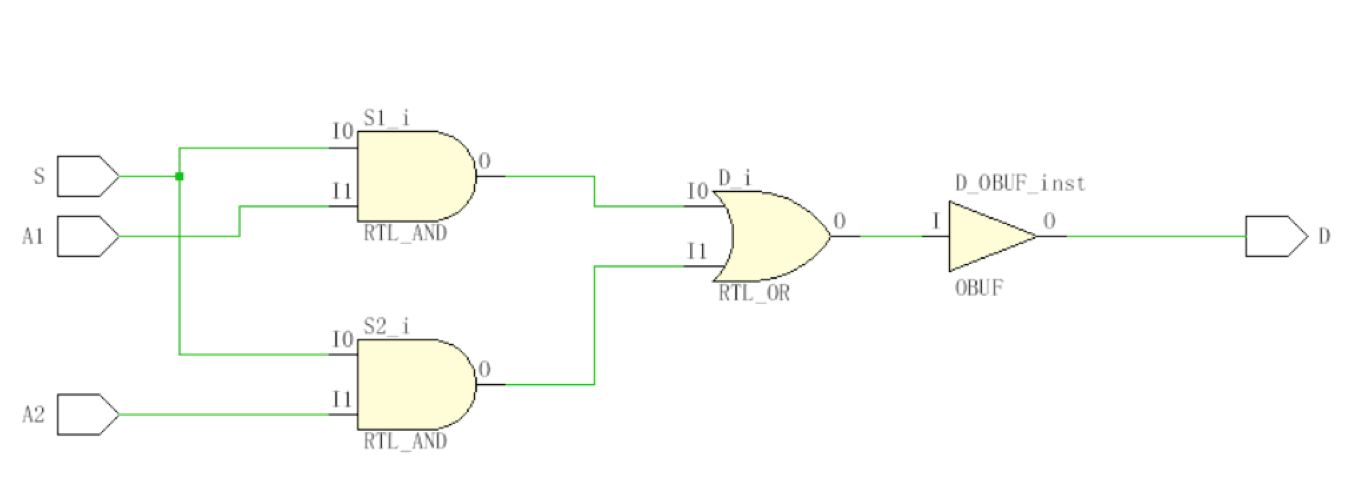
**2 实验2**

**2.1 实验内容**

设计一个1位的二选一多路选择器分别实现使用门级描述，实现使用逻辑表达式和实现使用if条件语句实现。

**2.2 实验步骤**

* 门级描述



RTL级原理图

电路模块代码：

module mux2to1(S,A1,A2,D);

input S,A1,A2;

output D;

wire n\_S,S1,S2;

not(n\_S,S);

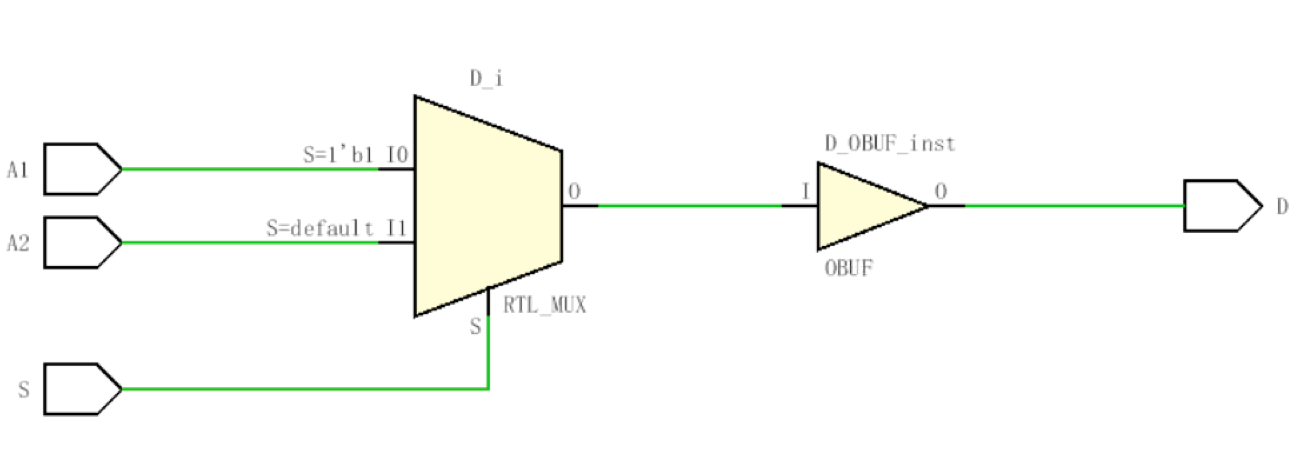
and(S1,S,A1);

and(S2,S,A2);

or(D,S1,S2);

endmodule

* + 使用逻辑表达式实现



RTL级原理图

电路模块代码：

module mux2to1(S,A1,A2,D);

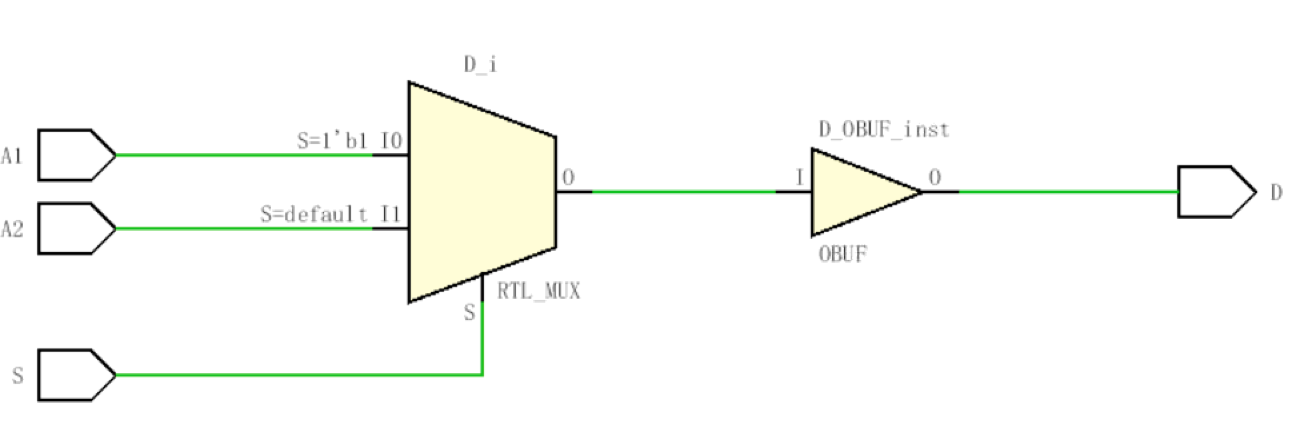
input S,A1,A2;

output D;

assign D=S?A1:A2;

endmodule

* + 使用if条件语句实现



RTL级原理图

电路模块代码：

module mux2to1(S,A1,A2,D);

input S,A1,A2;

output reg D;

always @(S or A1 or A2)

begin

if(S==1) D=A1;

else D=A2;

end

endmodule

**2.3 结果分析**

**测试模块代码编写为：**

`timescale 1ns / 1ps

module mux2to1\_tb();

reg A1,A2,S;

task1 u0(S,A1,A2,D);

initial begin

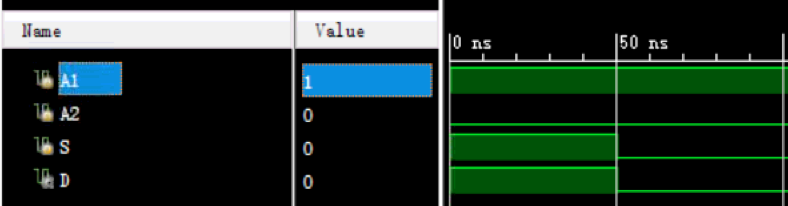
A1=1;A2=0;

S=1;

#50 S=0;

#50 $stop;

end

endmodule

**二选一多路选择器仿真结果图**

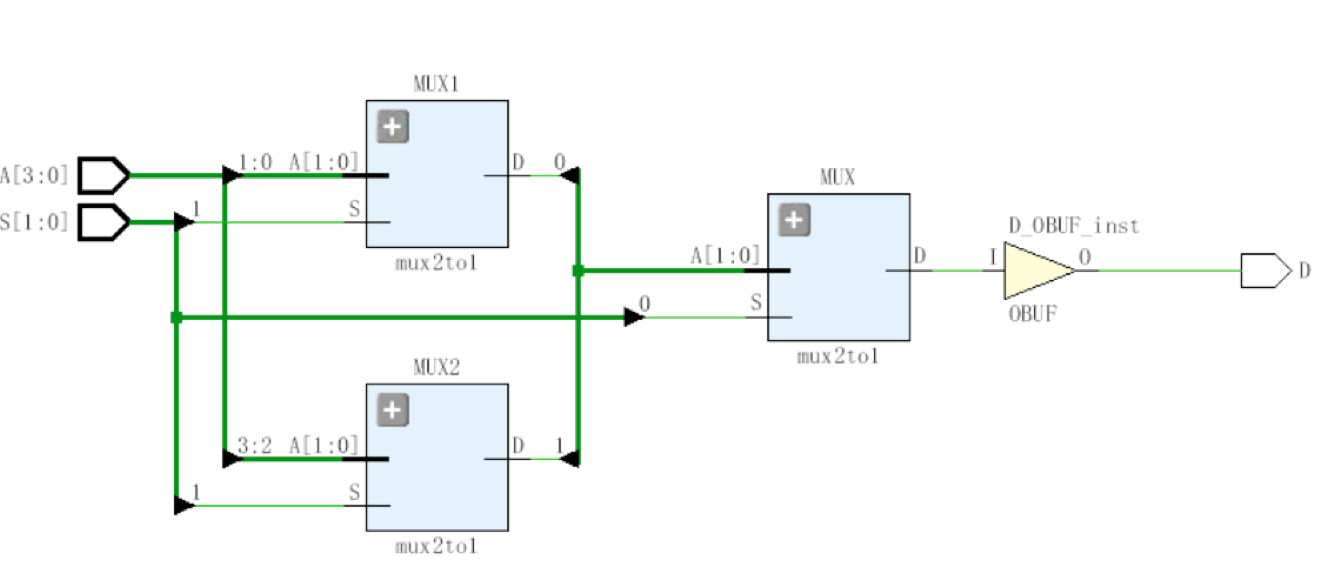
**三个不同方法实现的电路模块由于其功能相同，testbench设计相同，所得仿真结果也相同，在此就展示一张图。**

**由仿真结果图可观察出：当A1始终为1，A2始终为0，当S为1的时候，D为A1的值；当S为0的时候，D为A2的值。可看出编写的电路模块满足了二选一多路选择器的功能。**

**3 实验3**

**3.1 实验内容**

设计一个1位的四选一多路选择器并使用二选一多路选择器模块进行搭建。

**3.2 实验步骤**

四选一多路选择器RTL级原理图

电路模块代码：

module mux4to1(S,A,D);

input [1:0]S;

input [3:0]A;

output D;

wire [1:0]W;

mux2to1 MUX1(S[1],A[1:0],W[0]);

mux2to1 MUX2(S[1],A[3:2],W[1]);

mux2to1 MUX(S[0],W[1:0],D);

endmodule

**3.3 结果分析**

测试模块设计：

module mux4to1\_tb;

reg [1:0]s;

reg [3:0]a;

wire d;

mux4to1 U0(.S(s),.A(a),.D(d));

initial begin

a=0101; //为A[3:0]赋初值

s=00;//设初始S的值为00

#25 s=10;//25个时间单位之后，S的值变为01

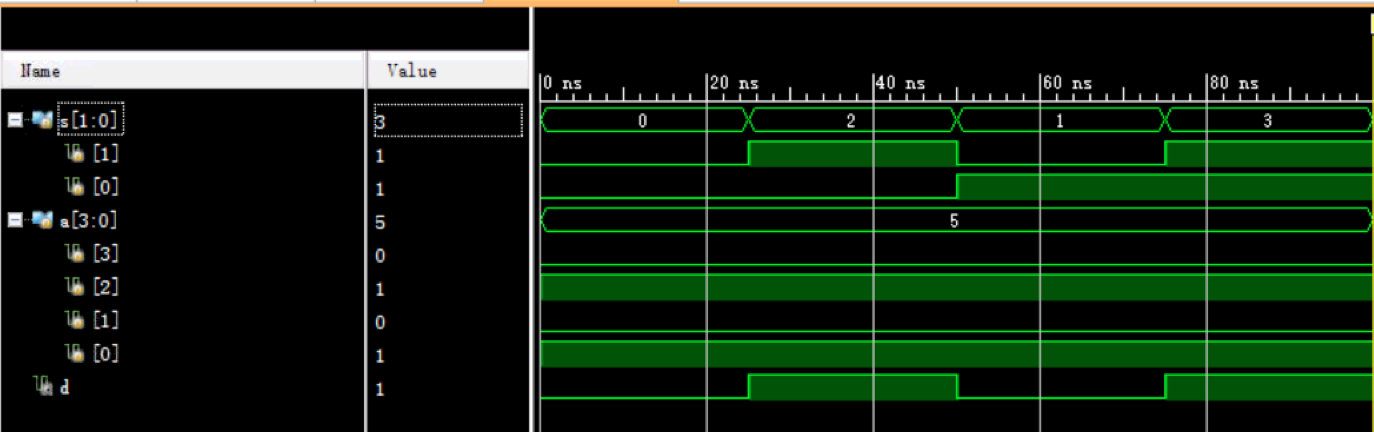
#25 s=01;//25个时间单位之后，S的值变为10

#25 s=11;//25个时间单位之后，S的值变为11

#25 $stop;

end

endmodule

四选一多路选择器仿真波形

在测试文件中，我们设置A的值为0101，S的值随时间变化分别为00，10，01，11，分别选择了A[0]，A[1]，A[2]，A[3]。由图d的值随时间变化分别为0，1，0，1。所以电路模块满足四选一多路选择器基本功能。

**4 实验4**

**4.1 实验内容**

设计一个4位二选一多路选择器，使用case语句实现。

**4.2 实验步骤**

电路模块设计代码：

module mux4to1(S,A,D);

input [1:0]S;

input [3:0]A;

output reg D;

always @(\*)

case(S)

2’b00:D=A[0];

2’b01:D=A[1];

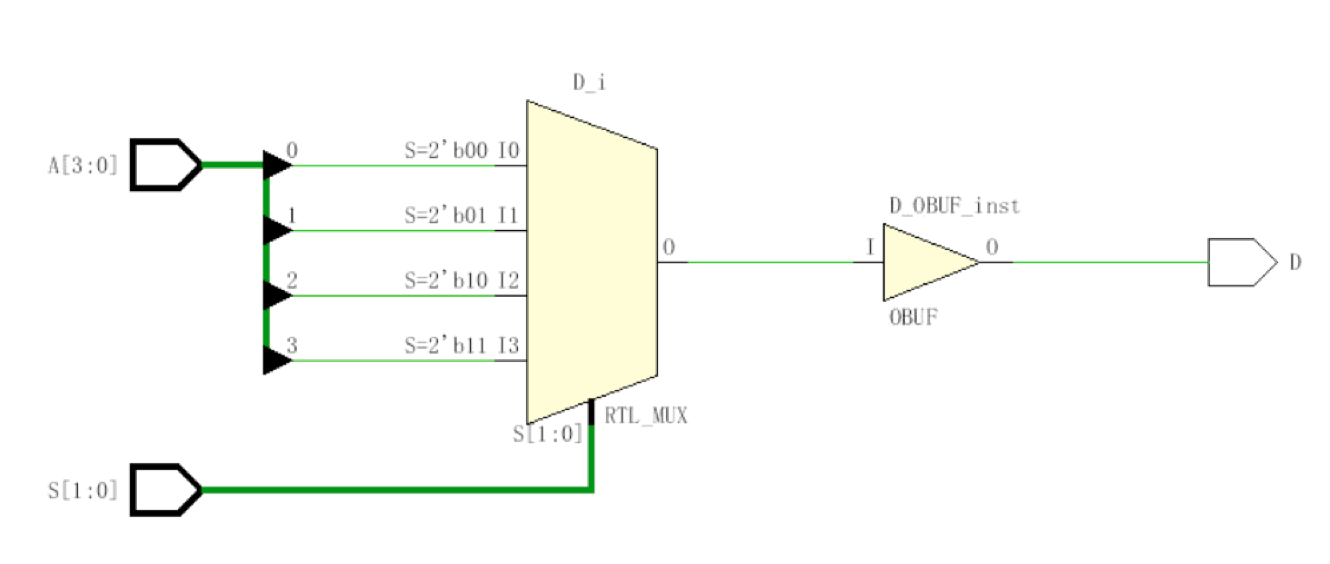
2’b10:D=A[2];

2’b11:D=A[3];

default:D=’bz;

endcase

endmodule

RTL原理图

**4.3 结果分析**

测试模块代码：

module mux4to1\_tb;

reg [1:0]S;

reg [3:0]A;

wire D;

mux4to1 MUX(S,A,D);

initial begin

A=0101;

S=00;

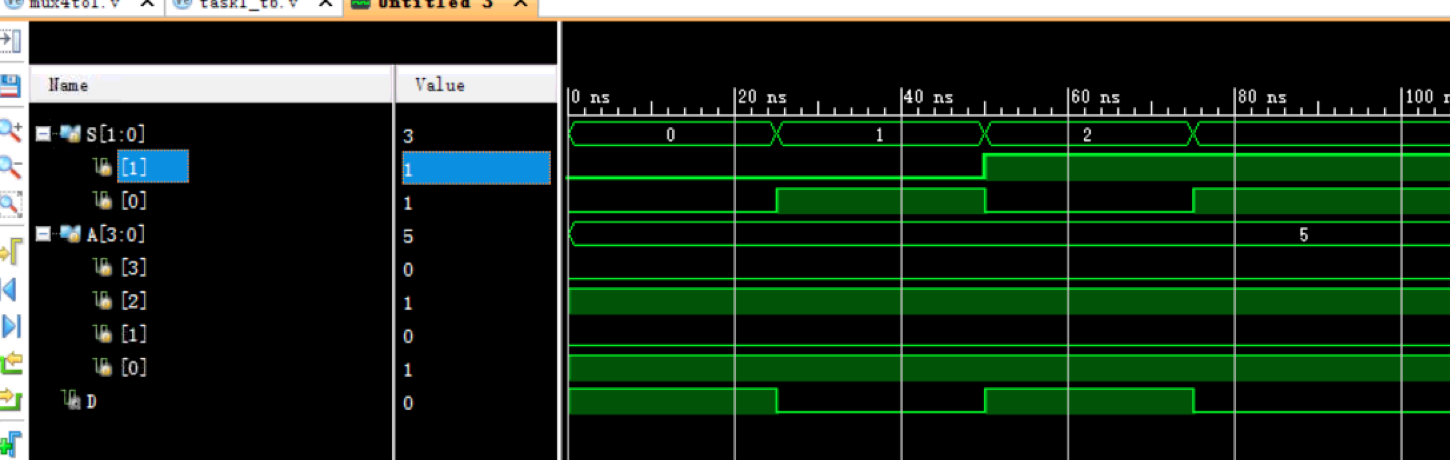
#25 S=01;

#25 S=10;

#25 S=11;

end

endmodule

仿真波形截图

由方针波形可以看出编写的HDL语言满足了4位二选一多路选择器的基本功能。S的值分别为0、1、2、3时，选择器分别选择了A0、A1、A2、A3，即D的值分别为1、0、1、0。

**5 实验5**

**5.1 实验内容**

|  |  |
| --- | --- |
| Select信号 | ALU功能 |
| 3’b000 | out=a |
| 3’b001 | out=a+b |
| 3’b010 | out=a-b |
| 3’b011 | out=a/b |
| 3’b100 | out=a%b |
| 3’b101 | out=a\*b |
| 3’b110 | out=a<<1 |
| 3’b111 | out=a>>1 |

使用case语句设计八功能的算术运算单元（ALU），其输入信号a和b均为4位，输入功能选择信号select为3位，输出信号out为5位。算术运算单元ALU所执行的操作与select信号有关，其关系如下：

**5.2 实验步骤**

电路模块设计代码：

module ALU(out, select, a, b);

input [2:0]select;

input [3:0]a;

input [3:0]b;

output reg [4:0]out;

always @(a or b or select)

case(select)

3'b000: out=a;

3'b001: out=a+b;

3'b010: out=a-b;

3'b011: out=a/b;

3'b100: out=a%b;

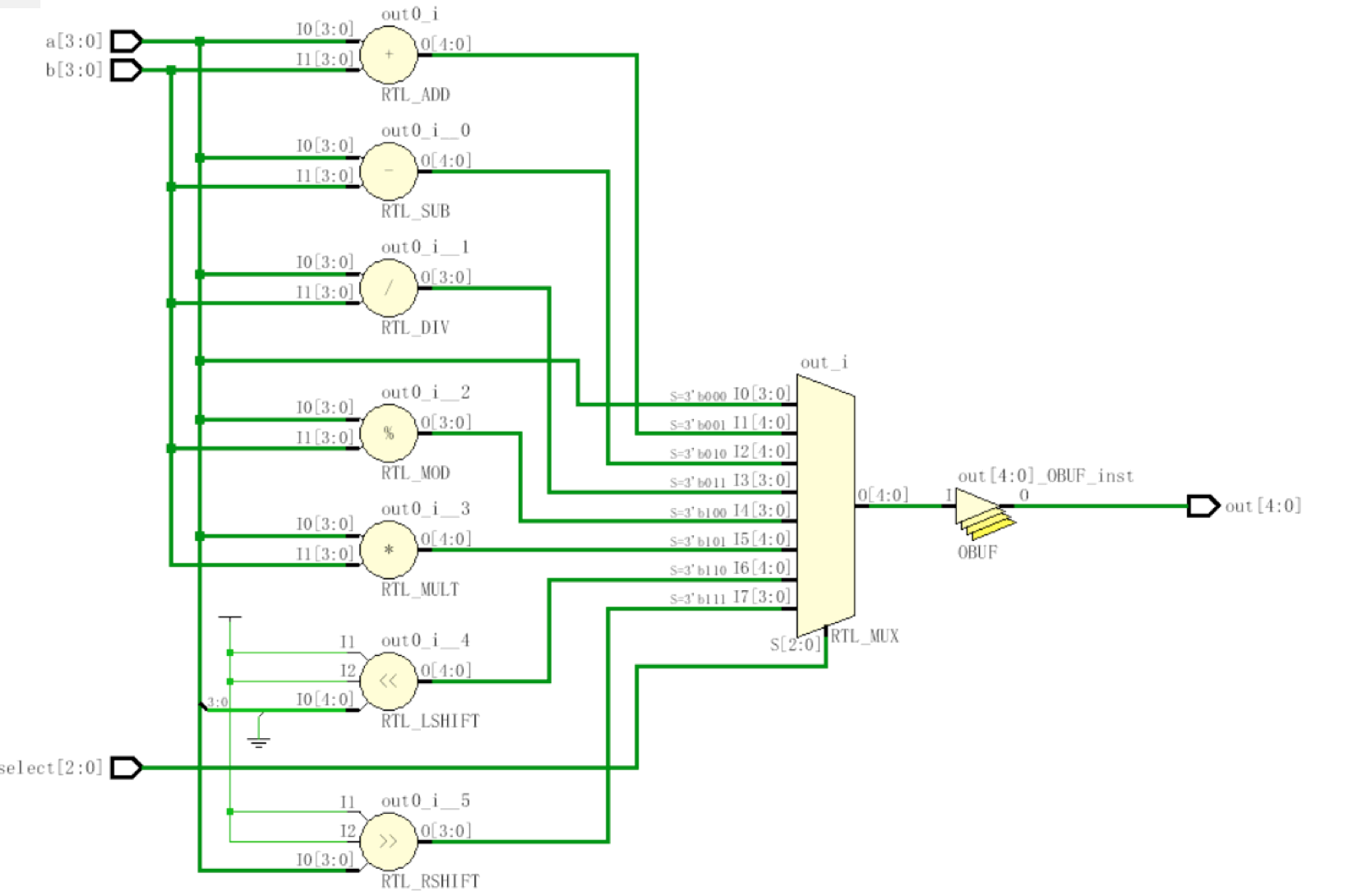
3'b101: out=a\*b;

3'b110: out=a<<1;

3'b111: out=a>>1;

endcase

endmodule



ALU的RTL级原理图

**5.3 结果分析**

测试文件设计代码：

module ALU\_tb;

reg [2:0]select;

reg [3:0]a;

reg [3:0]b;

wire [4:0]out;

ALU alu1(out, select, a, b);

initial begin

a=01;b=1000;

select=000;

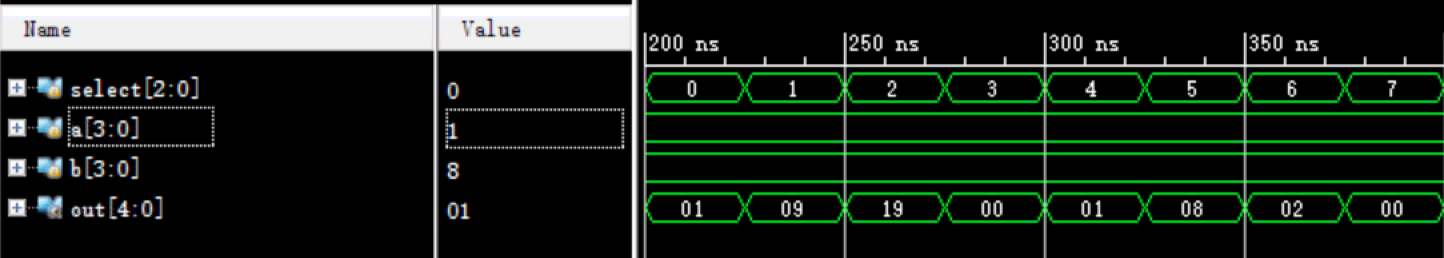
end

always begin

#25 select=select+1;

end

endmodule



仿真波形截图

在select=0时，out与a的值相同为1；select=1时，out为a和b的和，9；当select=2时out=a-b，select=2时out应为a-b的值-7，由于out是reg类型的没有负数，所以用补码的形式表示即19（16），select=3时，out=1=a/b的值0；select=4时，out=a%b=8；select=5时，out=a\*b=8；select=6时，out为a左移变为2；select=7时，out为a右移，用零补位，为0。

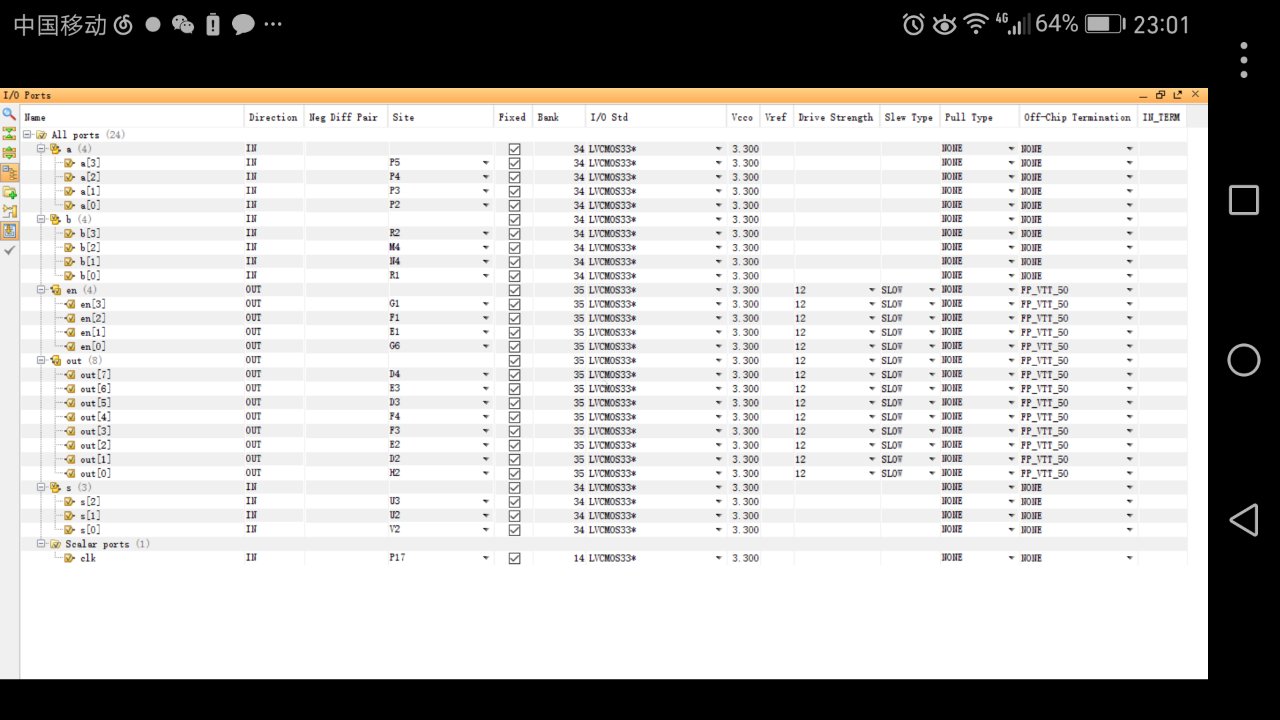
**6 实验6**

**6.1 实验内容**

对实验5进行板级验证;通过板子上的码拨开关输入操作数，自行设计拨码开关的控制逻辑，也可利用板子上的其它控制资源（如按钮等);将运算结果显示到数码管上。

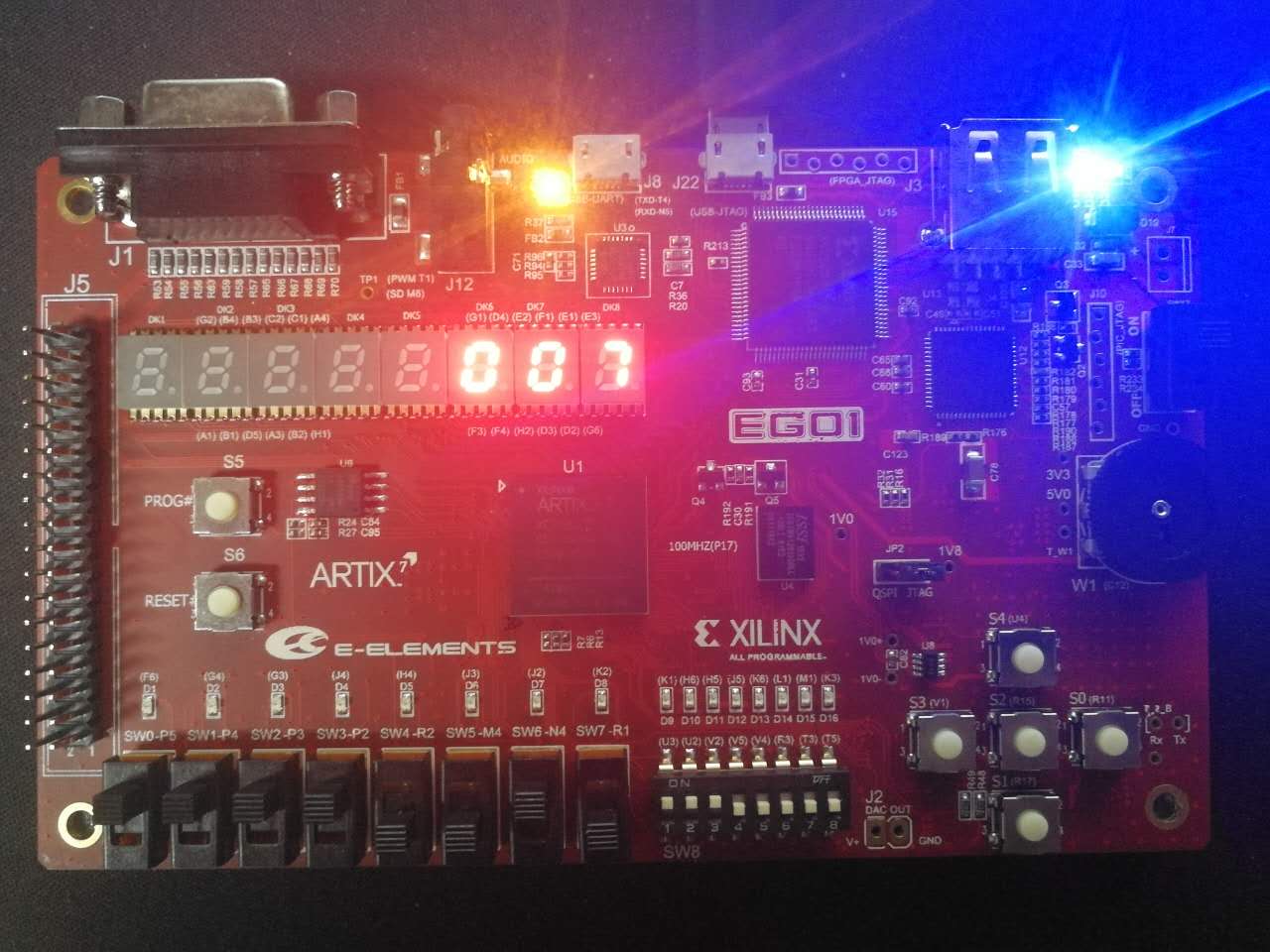
**6.2 实验步骤**

在详细学习了如何应用板子上的数码管之后，我设置了如下图的管脚约束。将select的控制设置在右下一排带有指示灯的开关键的左边三位。而输入端a、b分别设置成左下方带有指示灯的开关键的左边四位和右边四位（分别都是最后为LSB）。在设置完管脚、综合之后，将文件下载到FPGA的板子上，打开开关，调节不同的select键和a、b值之后，就可以看到数码管响应的计算结果了。



管脚约束示意图

**6.3 结果分析**

板级验证示意图

连接并装载好FPGA板之后，打开开关，调节不同的select键和a、b值，数码管就可以显示出响应的结果。在此只粘贴一张示意图。图中显示的是a为15时，左移一位，等到的结果是7，显示在了数码管上。

**7 实验1**

**7.1 实验内容**

带有清零和置位端的D触发器：门级描述，行为级描述。对比两种方法的波形图。

**7.2 实验步骤**

**门级电路描述代码:**

module d1(R,S,D,clk,out);

input R,S,D,clk;

output out;

wire w1,w2,S,w3;

nand nd1(w1,D,clk,R),

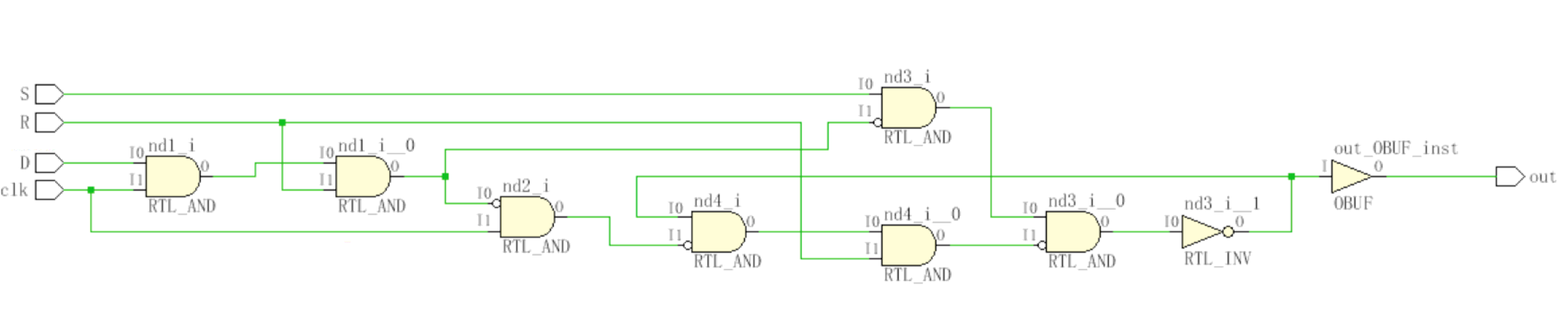
nd2(w2,w1,clk),

nd3(out,S,w1,w3),

nd4(w3,out,w2,R);

not no1(S\_n,S);

endmodule



门级描述RTL级设计图

门级描述的电路中的out是由clk电平控制，在clk为1的时候改变，在clk为0的时候保持原状态。异步置数和异步清零端。

行为级电路描述代码:

module d2(

input d,

input r,

input s,

input clk,

output q

);

reg q;

//异步清零和置数端

//低电平触发

always @(posedge clk or negedge r or negedge s)

begin

if(r==0) q=0;

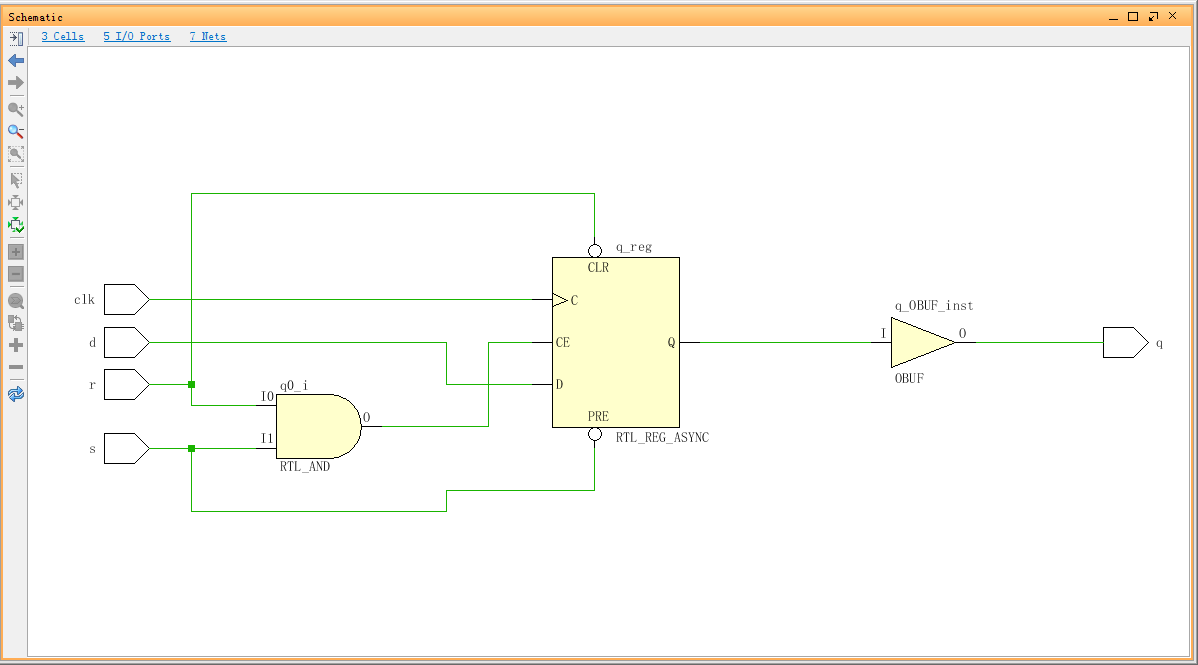
else if (s==0) q=1;

else if (r==1&&s==1) q=d;

end

endmodule

行为级描述中的q（即out）是由边沿触发的，clk上升沿或置数端和清零端的下降沿触发的。



行为级描述RTL级设计图

**7.3 结果分析**

门级描述测试代码：

module d1\_tb;

reg d,r,s,clk;

wire out;

d1 U1(r,s,d,clk,out);

initial begin

clk=0; end

always begin

#3 clk=~clk; end

initial begin

r=1;s=1;d=0;

#10 d=1;

#10 r=0;

#10 r=1;s=0;

#10 $stop;

end

endmodule

行为级描述测试代码：

module d2\_tb;

reg d,r,s;

wire q;

reg clk;

d2 U2(d,r,s,clk,q);

initial begin

clk=0; end

always begin

#3 clk=~clk; end

initial begin

r=1;s=1;d=0;

#10 d=1;

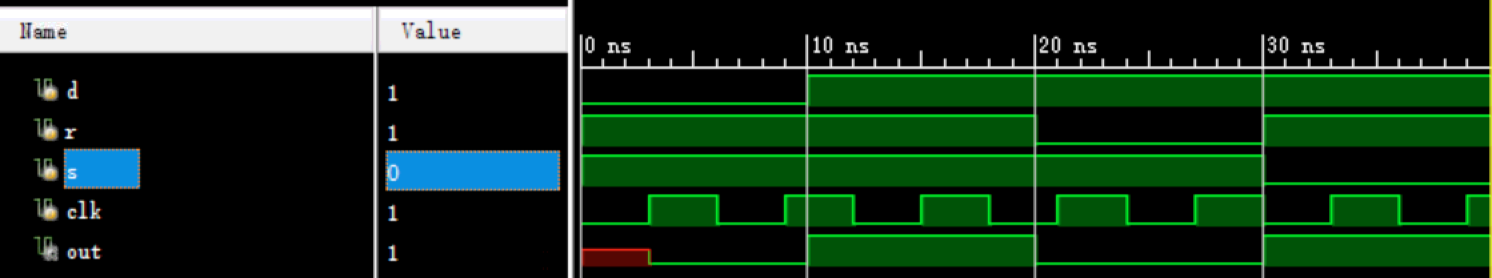
#10 r=0;

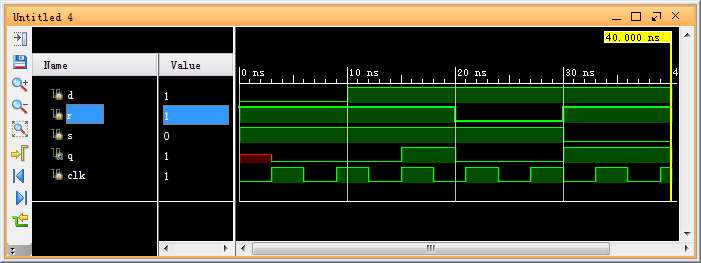
#10 r=1;s=0;

#10 $stop;

end

endmodule

****门级描述仿真示意图



行为级描述仿真示意图

对比两个不同描述方法的仿真图，可以演出在10ns-20ns处的输出端有所不同，在门级描述中，由于是clk电平触发，所以在10ns的时候由于clk为1，所以输出变为1；而在行为描述中，由于是clk边沿触发，在10ns的时候不是clk的边沿，所以等到了15ns时，clk上升沿时才使输出变为1.

**8 实验2**

**8.1 实验内容**

实现一个8位二进制计数器，编写电路模块及激励模块，波形图截屏验证

**8.2 实验步骤**

电路模块**代码：**

module counter(

input rset,

input clk,

output reg [7:0]out);

always@(posedge clk or negedge rset)

begin

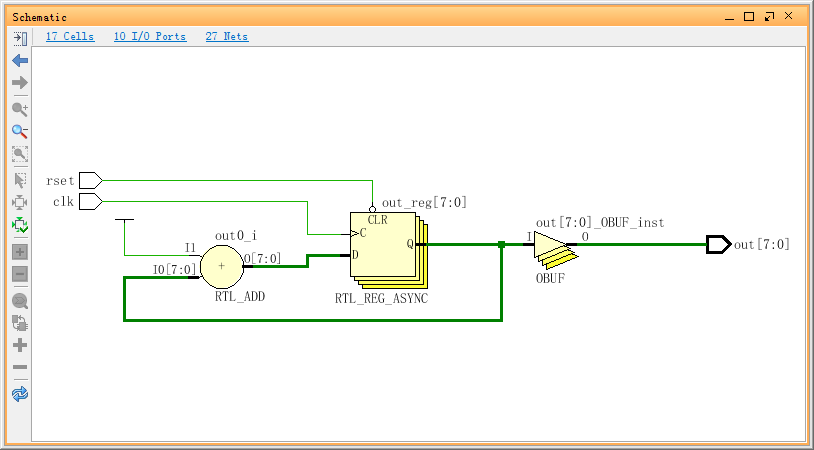
if(rset==0)out=0;

else

out=out+1;

end

endmodule



RTL级设计图

**8.3 结果分析**

module counter\_tb();

reg clk;

reg rset;

wire [7:0]out;

counter U0(rset,clk,out);

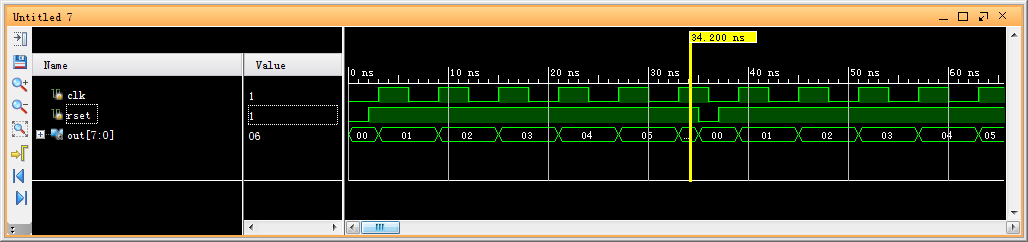
initial begin

clk=0;rset=0;#2 rset=1;#33 rset=0;#2 rset=1;end

always begin

#3 clk=~clk; end

endmodule



八位计数器波形仿真图

由图可以看出输出从零开始逐个增加，电路模块和测试模块的编写基本实现了所要求的基本功能。

我还增加了置零功能，在遇到置数端下降沿时重新开始计数。

**9 实验3**

**9.1 实验内容**

实现一个8位二进制计数器，编写电路模块及激励模块，波形图截屏验证。

**9.2 实验步骤**

module counter(

input rset,

input clk,

output reg [7:0]out);

always@(posedge clk or negedge rset)

begin

if(rset==0)out=0;

else

out=out+1;

end

endmodule

由于连在板子上的计数器时钟频率过高，肉眼不可识别，所以需要增大时钟周期，使现实更加清晰。在这里使用了二分之一频率。

**9.3 结果分析**

仿真模块设计代码：

`timescale 1ns / 1ps

module counter\_tb();

reg switch;

reg clk;

wire [7:0] out;

counter o1(.switch(switch),.clk(clk),.out(out));

initial begin

clk = 0;

switch = 1;

end

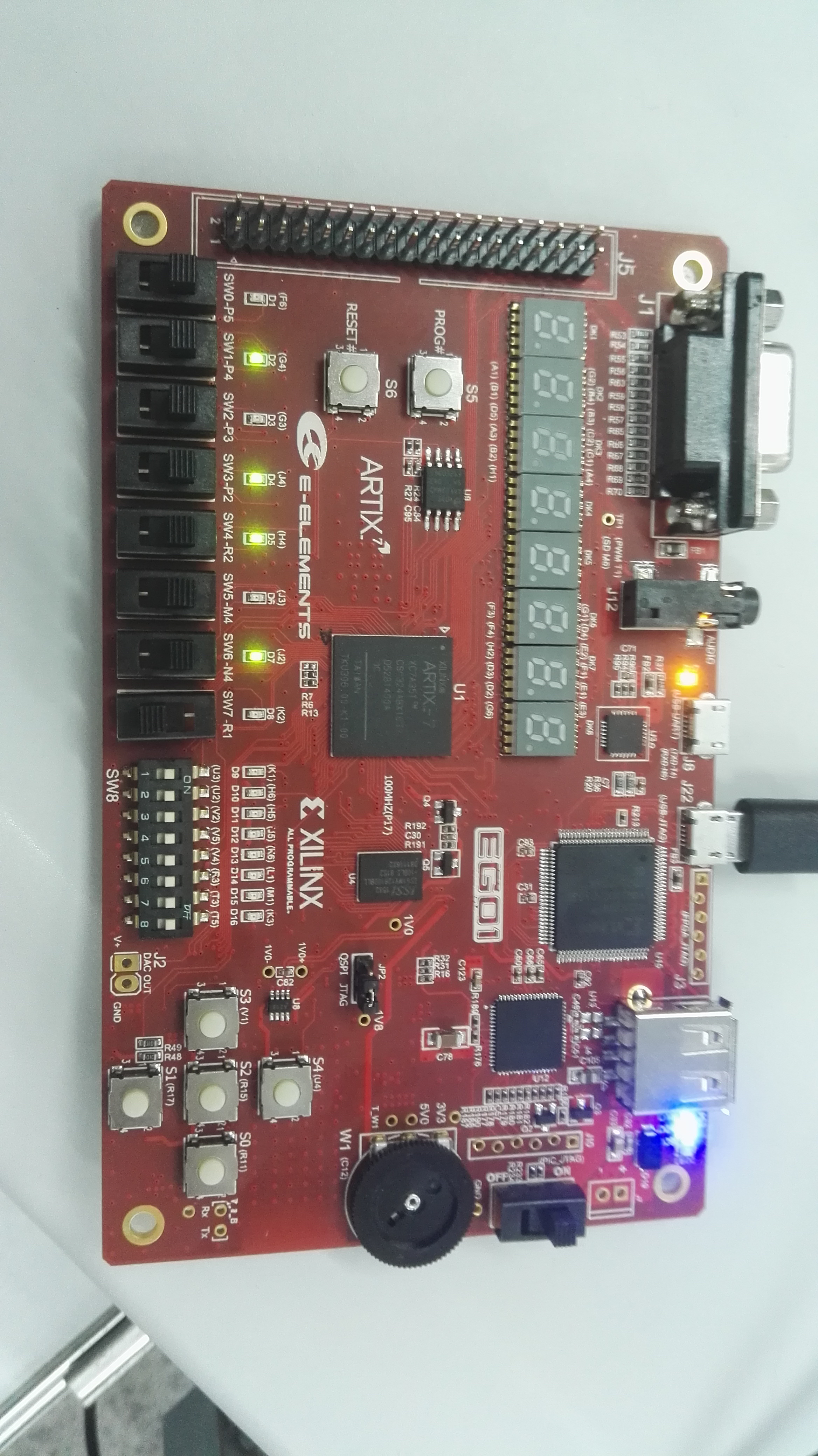
always begin

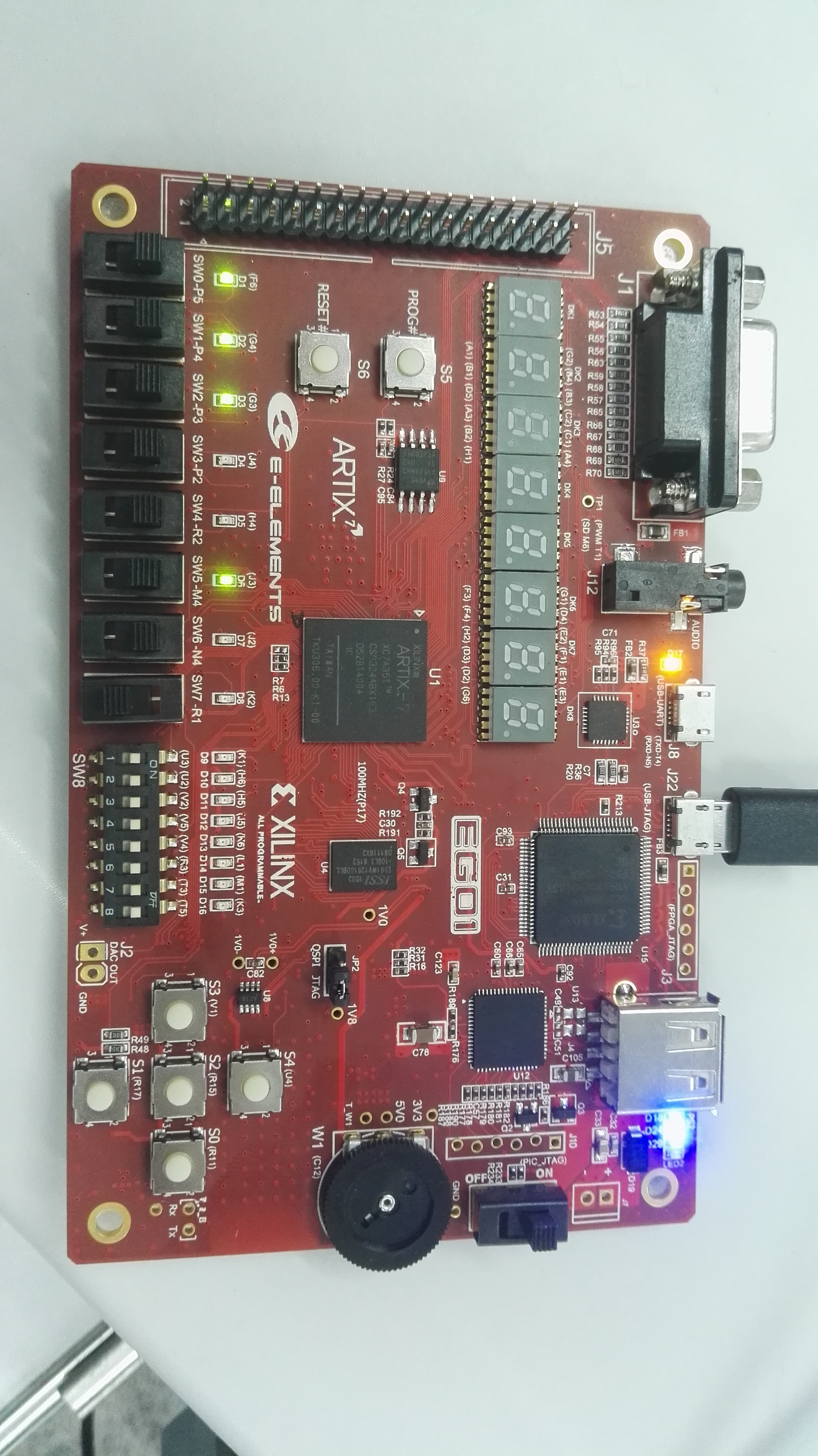
if (switch == 1) #10 clk = ~clk;

else clk = 0;

end

endmodule

counter板级验证1

counter板级验证2

我是利用流水灯来指示计数器的计数的。由于计数器技术过程是动态过程，所以只截了两张图作为展示参考。在第八个实验的基础上，实验九的主要改进在于时钟频率的减少和管脚的约束。板级验证中，计数器可以依次加一并显示出来，在停止计数键有效时，暂停计数。

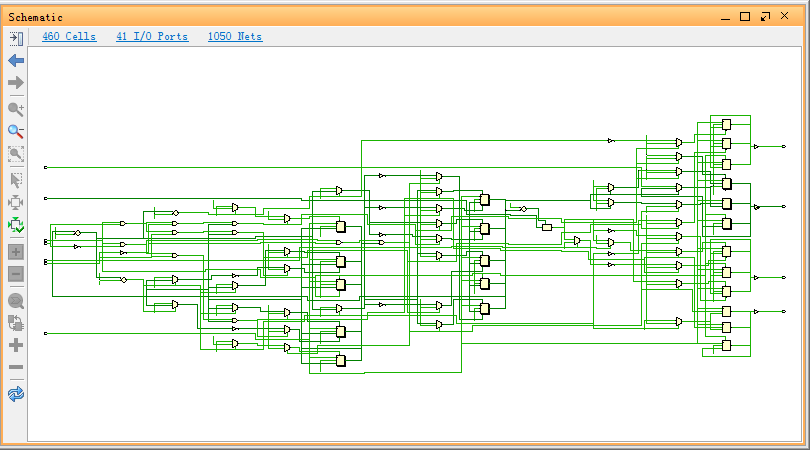
**10 实验3**

**10.1 实验内容**

设计一个数据串并转换电路：

* 一端是串行数据
* 另一端是并行数据
* 并行输入/出端可配置成8位/16位两种情况
* 自行设计相关控制信号
* 完成电路逻辑设计
* 编写测试激励
* 波形仿真

**10.2 实验步骤**

****

**RTL**级电路图

电路模块设计代码：

module interface(

input data\_in1,

output reg data\_out1,

input [15:0]data\_in2,

output reg [15:0]data\_out2,

input r\_w,ctrl\_w,reset,clk,write\_en,

output reg data\_ready,receive\_ready

);

integer i=0;//设置全局变量

integer j=0;//设置全局变量

always @(posedge clk or negedge reset)//每次改变r\_w,ctrl\_w等状态时必须先重置

begin

if(reset==0)

begin

data\_out1=0;

data\_out2=0;

data\_ready=0;

receive\_ready=0;

j=1;i=0;

end

end

always@(posedge clk or negedge reset)//在时钟上升沿时串入的每一位导入到相应的输出位

begin

if(reset&&r\_w&&ctrl\_w&&j)//r\_w为1串入并出，ctrl\_w为1为16位

begin

data\_out2[i]=data\_in1;

i=i+1;

if(i==16) begin

data\_ready=1;i=0;j=0;//当并出接受全部串入的值时，示意外部可以输出

end

end

end

always@(posedge clk or negedge reset)//在时钟上升沿时串入的每一位导入到相应的输出位

begin

if(reset&&r\_w&&!ctrl\_w&&j)//r\_w为1串入并出，ctrl\_w为0为8位

begin

data\_out2[i]=data\_in1;

i=i+1;

data\_out2[15:8]=2'h00;

if(i==8) begin

data\_ready=1;i=0;j=0;//当并出接受全部串入的值时，示意外部可以输出

end

end

end

always@(posedge clk or negedge reset)

begin

if(reset&&!r\_w&&ctrl\_w&&j)//r\_w为0串出并入，ctrl\_w为1为16位

begin

receive\_ready=1;//示意外部可以接受数据

if(write\_en)

data\_out1=data\_in2[15-i];//左移先输出MSB

i=i+1;

if(i==16)

begin

i=0;j=0;receive\_ready=0;

end

end

end

always@(posedge clk or negedge reset)

begin

if(reset&&!r\_w&&!ctrl\_w&&j)//r\_w为0串出并入，ctrl\_w为0为8位

begin

receive\_ready=1;//示意外部可以接受数据

if(write\_en)

if(i<8)

data\_out1=0;//先输出8位0

if(7<i<16)

data\_out1=data\_in2[15-i];//左移先输出MSB

i=i+1;

if(i==16)

begin

i=0;receive\_ready=0;j=0;

end

end

end

endmodule

**10.3 结果分析**

测试模块代码：

`timescale 1ns / 1ps

module interface\_tb;

reg data\_in1;

wire data\_out1;

reg [15:0]data\_in2;

wire [15:0]data\_out2;

reg r\_w,ctrl\_w,reset,clk,write\_en;

wire data\_ready,receive\_ready;

interface u0( data\_in1, data\_out1, data\_in2, data\_out2, r\_w,ctrl\_w, reset,clk, write\_en, data\_ready, receive\_ready);

initial

begin

reset=0;clk=0;r\_w=1;ctrl\_w=1;write\_en=0;//先检验串入并出16位

data\_in1=0;

#1 reset=1;//置零

#3 data\_in1=1;

#2 data\_in1=1;

#2 data\_in1=1;

#2 data\_in1=0;

#2 data\_in1=1;

#2 data\_in1=1;

#2 data\_in1=0;

#2 data\_in1=0;

#2 data\_in1=1;

#2 data\_in1=1;

#2 data\_in1=1;

#2 data\_in1=0;

#2 data\_in1=1;

#2 data\_in1=1;

#2 data\_in1=0;

#2 ctrl\_w=0;reset=0;

#1 reset=1;//置零

#1 data\_in1=0;

#2 data\_in1=1;

#2 data\_in1=0;

#2 data\_in1=1;

#2 data\_in1=1;

#2 data\_in1=0;

#2 data\_in1=1;

#2 data\_in1=1;

#2 reset=0;r\_w=0;ctrl\_w=1;

#2 reset=1;//置零

data\_in2=16'b1010101010111001;

write\_en=1;

#40 ctrl\_w=0;reset=0;//置零

#1 reset=1;

data\_in2=8'b10101010;

write\_en=1;

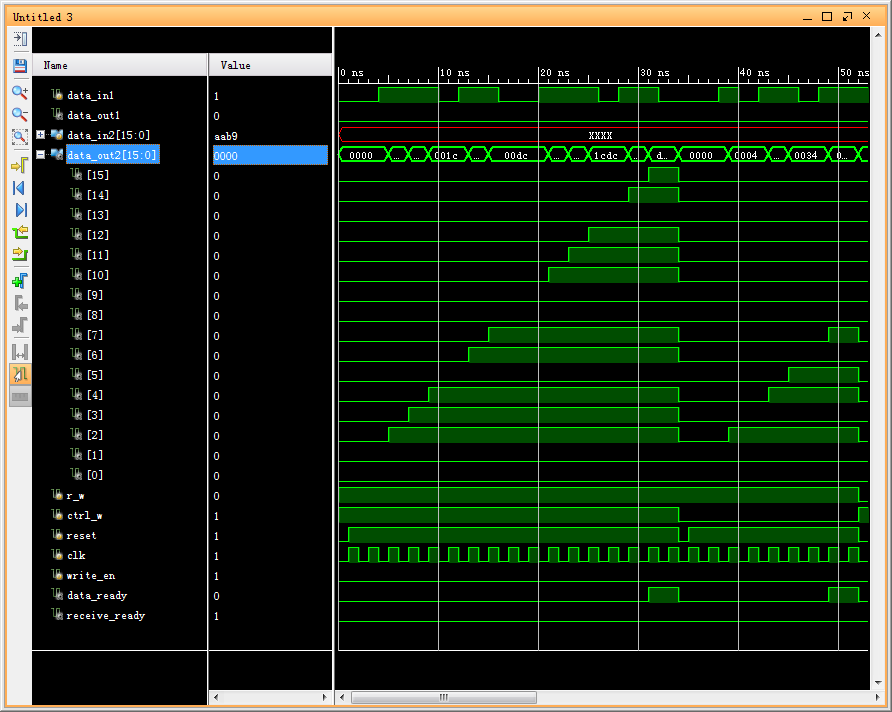
#40 $finish;

end

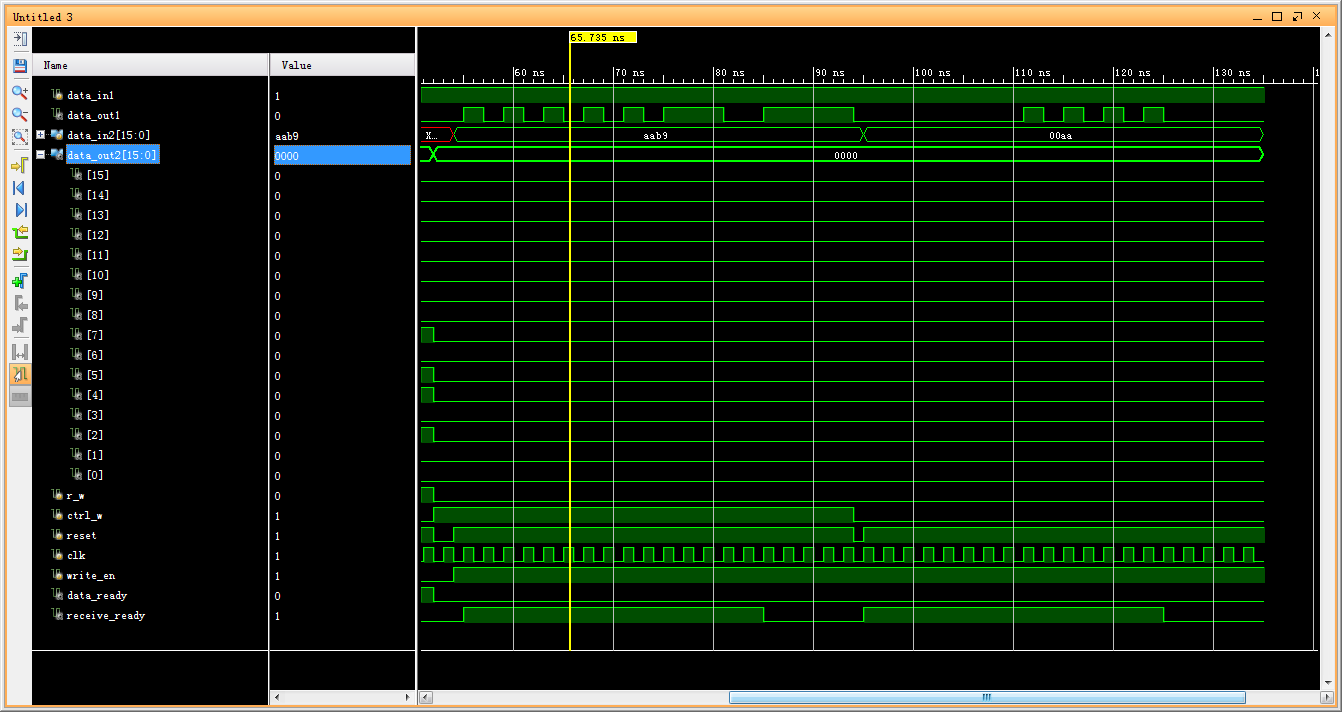
always #1 clk=~clk;

endmodule

在测试激励中设置的主要是时钟频率，串入和并入的数据等。由于在串出和并出设置的是由全局变量控制的循环，所以在变化格式时需要先进行清零操作。

****

**串入并出功能电路仿真示意图**

如图中实现的是串入并出功能的仿真，可以看出，data\_out2在每一个时钟周期被赋入一位串入端的值，并且在data\_ready为高电平时完成全部赋值，可以输出。****

**串出并入功能电路仿真示意图**

如图中实现的是并入串出功能的仿真，由于左移输出，所以先输出MSB。从图中可以看出串行输出输出了并行输入的值，实现了串出并入的功能。

**四、结论与讨论**

在整个Verilog实验中，设置了几个对比行为级描述和门级描述的区别的实验。例如在实验二中，用if条件句、逻辑表达式和门级分别描述了二选一多路选通器。在他们的RTL设计图中，由if条件句和逻辑表达式描述的电路的RTL设计相同，由二选一选择器构成主要部分，而门级描述的RTL级设计有更为精细的门级电路。虽然门级描述在实现上更为方便，但易与出错，而且设计层面上，设计者不关注底层实现，所以设计时应多使用行为级描述从而确保系统整体的功能。

在实验十中，我设置了一个循环：每次在时钟上升沿时，将串入端赋值给下一个并出端。刚开始我是在一个大的always模块设置一个for循环，但都满足不了每次在时钟上升沿开始的这一条件，所以循环必须单独设置一个always模块。这样设置就导致必须设置好循环开始和结束的条件，于是设置了全局变量并控制好全局变量的改变。

以上是我印象较为深刻实验的总结。

在完成实验时，由于初学Veilog，对语言还不是较为熟练，用了大量的时间去调试程序。到了后期，基本不会出现语法上的错误，效率得到大量的提高。在计算机学科学习上，实践具有很重要的意义，在大量练习中，我们学习到的知识得到了巩固，学科思维得到了训练。

**五、教师评审**

|  |  |
| --- | --- |
| **教师评语** | **实验成绩（25分）** |
| 签名：  日期： |  |