

**计算机与信息学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 计算机组成与体系结构实验 | | | | |
| 实验编号： | Project13 | | | | |
| 实验名称： | 寄存器设计与验证 | | | | |
| 实验人员： | 学号 | 18111303044 | | | |
| 姓名 | 邵一波 | | | |
| 班级 | 2018级计算机类1班 | | | |
| 实验日期： | 2019-12-2 | | | | |
| 实验室： | 学苑南楼2幢202室 | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

# 寄存器设计与验证

# 一、实验目的

## 1.掌握寄存器的建模方法与验证技术。

## 2.掌握寄存器组的建模与验证技术。

# 二、实验工具

## 1.Xilinx Vivado 2014.2软件。

## 2.Windows系统PC机。

# 三、实验要求

## 1.利用Verilog HDL对指令寄存器、指令计数器、数据缓冲器建模与验证。

## 2. 利用Verilog HDL对寄存器文件（通用寄存器）建模与验证。

# 四、实验内容

## 1.指令计数器PC

#### (1)基本描述

PC是指令计数器，主要功能是完成输出当前指令地址并保存下一条指令地址。复位后，PC指向0x0000\_3000，此处为第一条指令的地址。



图1 PC模块

#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| [31:2]NPC | I | 下条指令的地址 |
| PCWr | I | PC写使能  1：允许NPC写入PC内部寄存器  0：禁止NPC写入PC内部寄存器 |
| clk | I | 时钟信号 |
| rst | I | 复位信号。  1：复位  0：无效 |
| [31:2]PC | O | 30位指令存储器地址(最低2位省略) |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | if rst=1，PC🠄0x0000\_3000。 |
| 2 | 保存NPC并输出 | Clk上升沿。  if rst=0,PC🠄 NPC。 |

#### (4)模块描述

module PC( clk, rst, PCWr, NPC, PC );

input clk;

input rst;

input PCWr;

input [31:2] NPC;

output [31:2] PC;

reg [31:2] PC;

reg [1:0] tmp;

always @(posedge clk or posedge rst) begin

if ( rst )

{PC, tmp} <= 32'h0000\_3000;

else if ( PCWr )

PC <= NPC;

end // end always

endmodule

module test();

reg clk;

reg rst;

reg PCWr;

reg [31:2]NPC;

wire [31:2]PC;

PC u0(.clk(clk),.rst(rst),.PCWr(PCWr),.NPC(NPC),.PC(PC));

initial begin

PCWr=1;

clk=1;

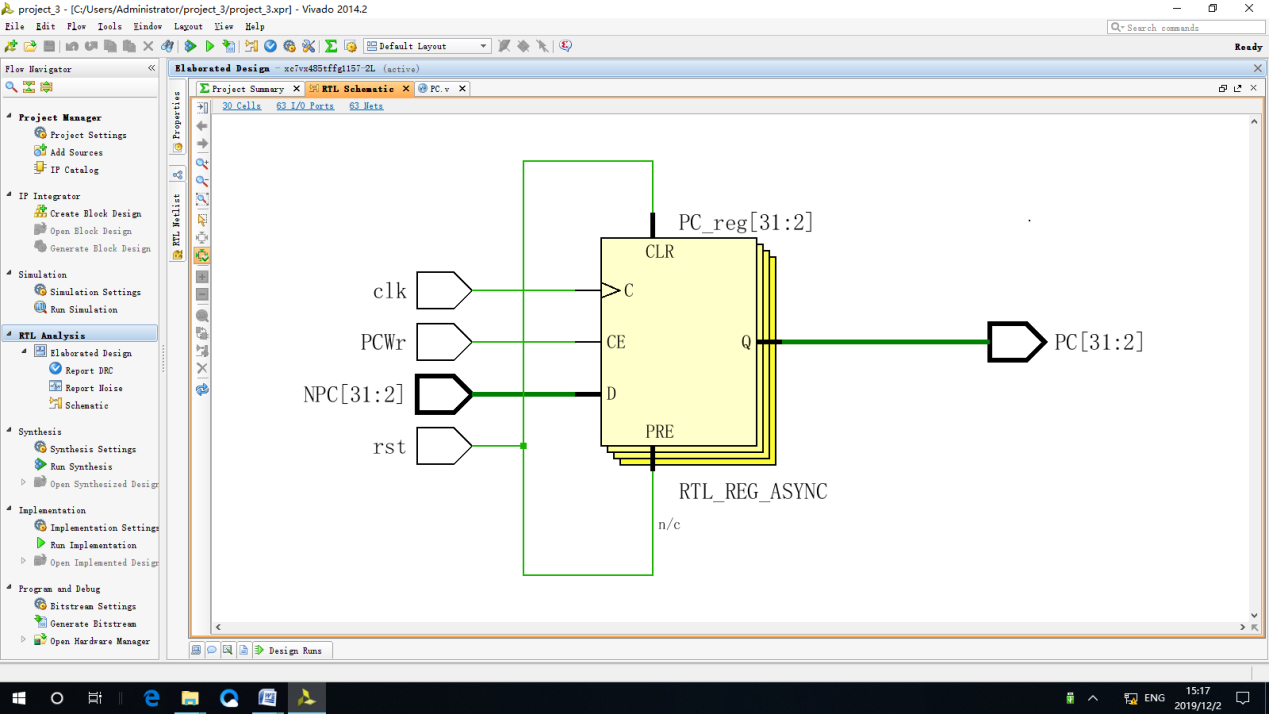
rst=1;

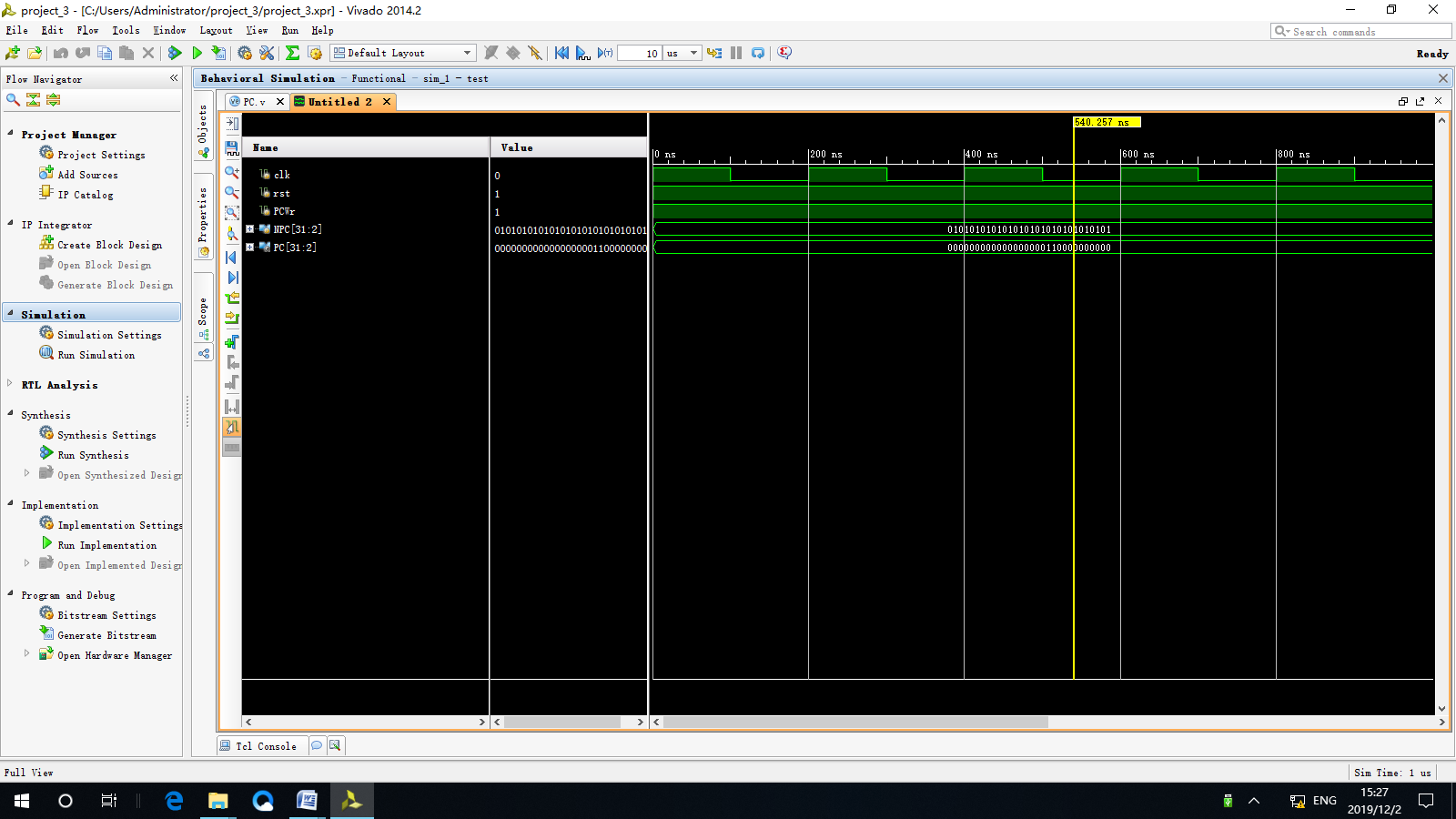
NPC=30'b010101010101010101010101010101;

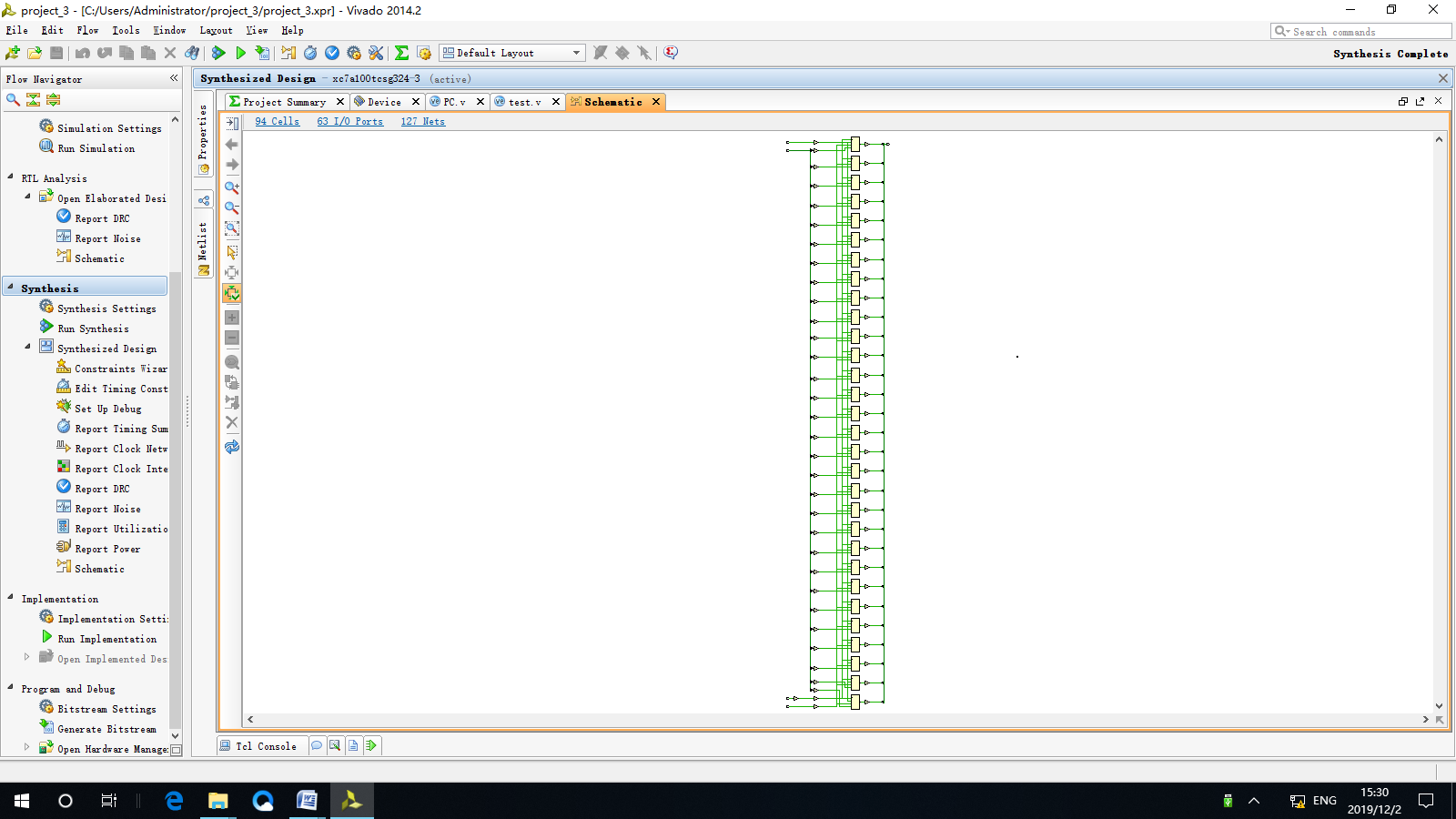
end

always #100 clk=~clk;

endmodule







## 2.下条指令计数器NPC

#### (1)基本描述

NPC是下条指令计数产生器，主要功能是计算下一条指令地址，NPCOp[1:0]决定如何计算NPC，与PC配合使用。严格说来，NPC不是一个时序逻辑电路。



图2 NPC模块

#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC[31:2] | I | 30位本条指令的地址(最低2位省略) |
| Imm[25:0] | I | 立即数(偏移量) |
| NPCOp[1:0] | I | 计算方式 |
| NPC[31:2] | O | 30位下一条指令地址(最低2位省略) |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算下一条指令地址 | |  |  |  | | --- | --- | --- | | NPCOp | 功能 | 操作 | | 00 | 顺序地址 | NPC 🠄 PC + 1 | | 01 | 计算B指令转移地址 | NPC 🠄 PC +{sign\_ext(imm16) } | | 10 | 计算J类指令转移地址 | NPC 🠄{PC[31:28], imm26} | | 11 | ... | ... | |

#### (4)模块描述

`include "ctrl\_encode\_def.v"

module NPC( PC, NPCOp, IMM, NPC );

input [31:2] PC;

input [1:0] NPCOp;

input [25:0] IMM;

output [31:2] NPC;

reg [31:2] NPC;

always @(\*) begin

case (NPCOp)

`NPC\_PLUS4: NPC = PC + 1;

`NPC\_BRANCH: NPC = PC + 1+{{14{IMM[15]}}, IMM[15:0]};

`NPC\_JUMP: NPC = {PC[31:28], IMM[25:0]};

default: ;

endcase

end // end always

endmodule

module test();

reg [31:2] PC;

reg [1:0] NPCOp;

reg [25:0] IMM;

wire [31:2] NPC;

NPC u0(.PC(PC),.NPCOp(NPCOp),.IMM(IMM),.NPC(NPC));

initial begin

NPCOp=2'b00;

PC=30'b010101010101010101010101010101;

IMM=26'b01010101010101010101010101;

#100;

NPCOp=2'b01;

PC=30'b010101010101010101010101010101;

IMM=26'b01010101010101010101010101;

#100;

NPCOp=2'b10;

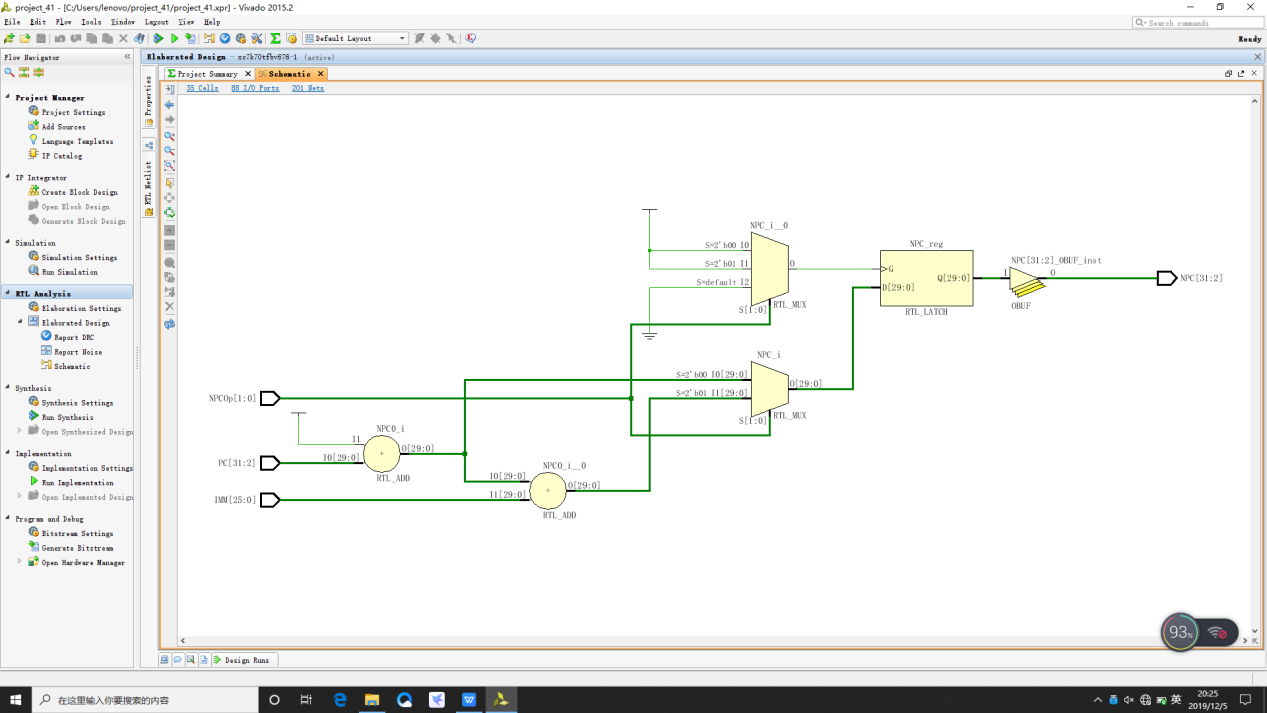
PC=30'b010101010101010101010101010101;

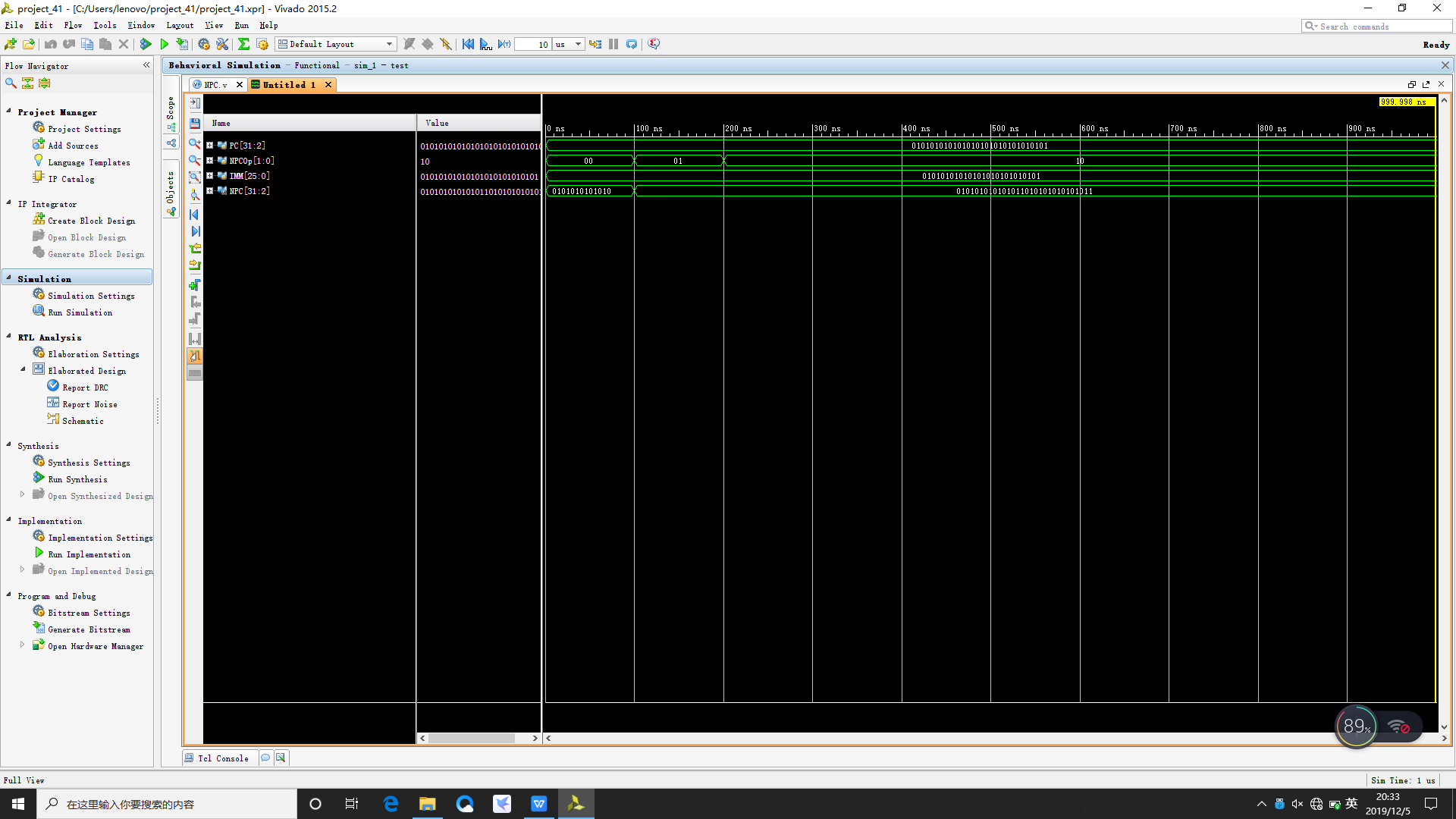
IMM=26'b01010101010101010101010101;

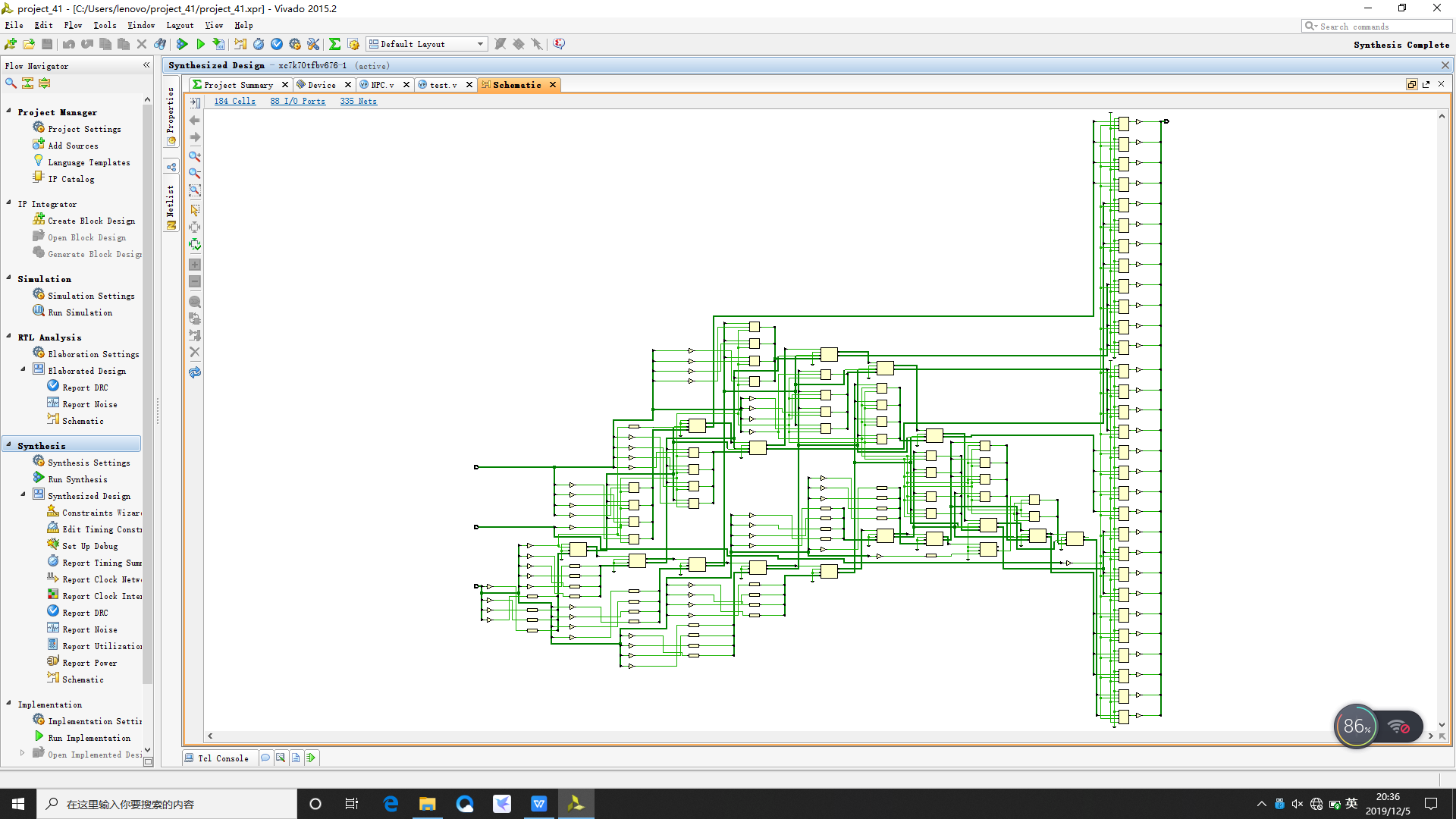
#100;

end

endmodule







## 3.指令寄存器IR

#### (1)基本描述

IR主要功能是完成对来自IM的指令的缓冲。



图3 IR模块

#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| [31:0] im\_dout; | I | 指令输入 |
| IRWr | I | 写使能信号 |
| rst | I | 复位信号 |
| clk | I | 时钟信号 |
| [31:0] instr; | O | 指令输出 |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | if rst=1, instr🠄0 |
| 2 | 缓冲 | if rst=0, instr🠄im\_dout |

#### (4)模块描述

module IR (clk, rst, IRWr, im\_dout, instr);

input clk;

input rst;

input IRWr;

input [31:0] im\_dout;

output [31:0] instr;

reg [31:0] instr;

always @(posedge clk or posedge rst) begin

if ( rst )

instr<= 0;

else if (IRWr)

instr<= im\_dout;

end // end always

endmodule

module test();

reg clk;

reg rst;

reg IRWr;

reg [31:0]im\_dout;

wire [31:0]instr;

IR u0(.clk(clk),.rst(rst),.IRWr(IRWr),.im\_dout(im\_dout),.instr(instr));

initial begin

IRWr=1;

clk=1;

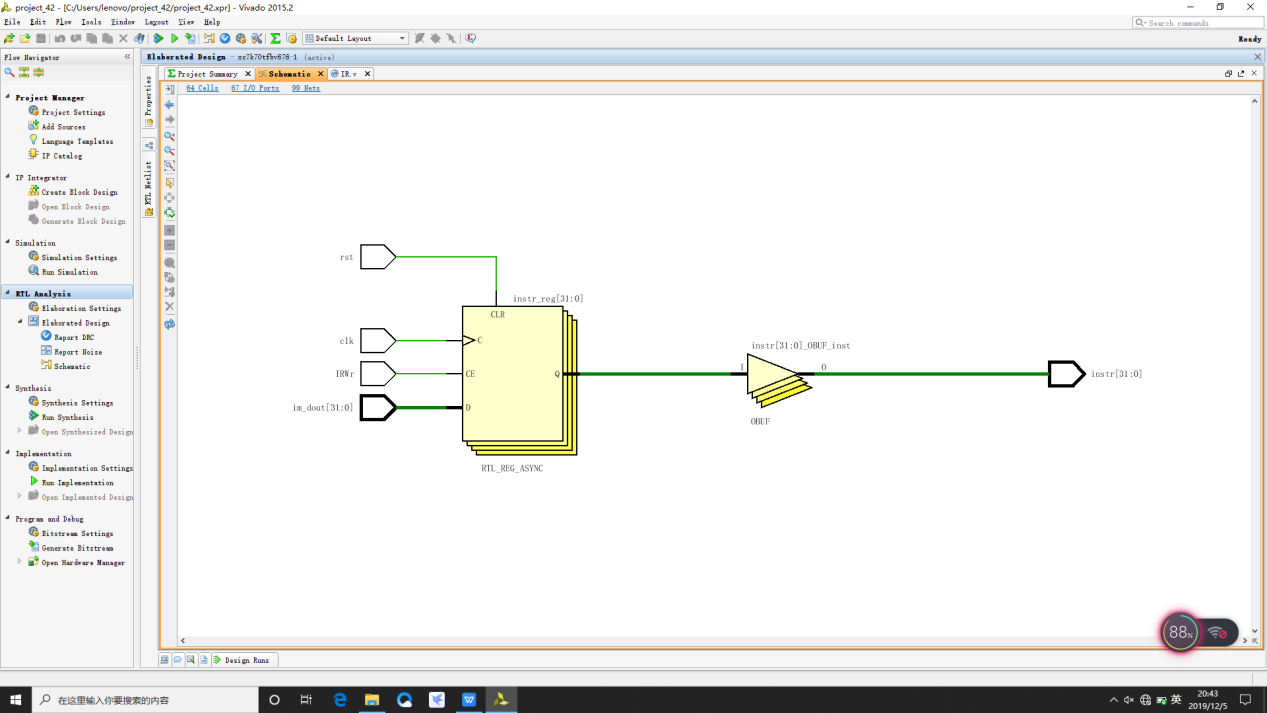
rst=1;

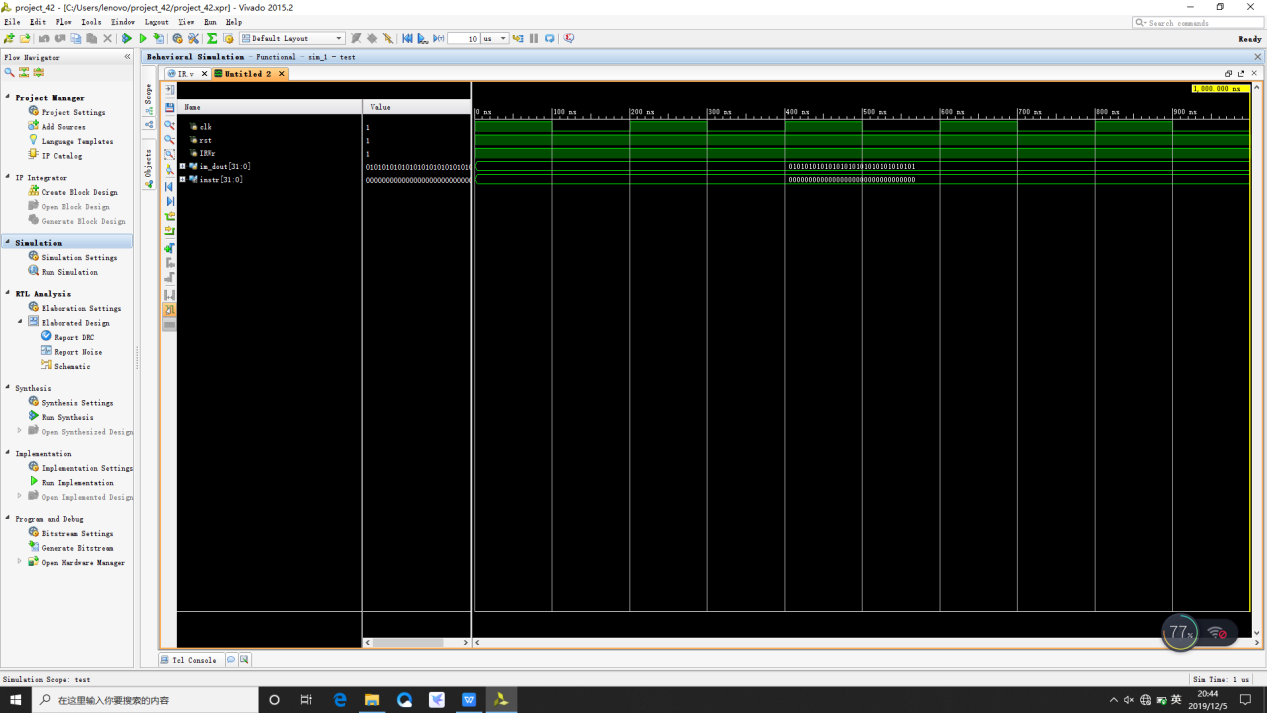
im\_dout=32'b01010101010101010101010101010101;

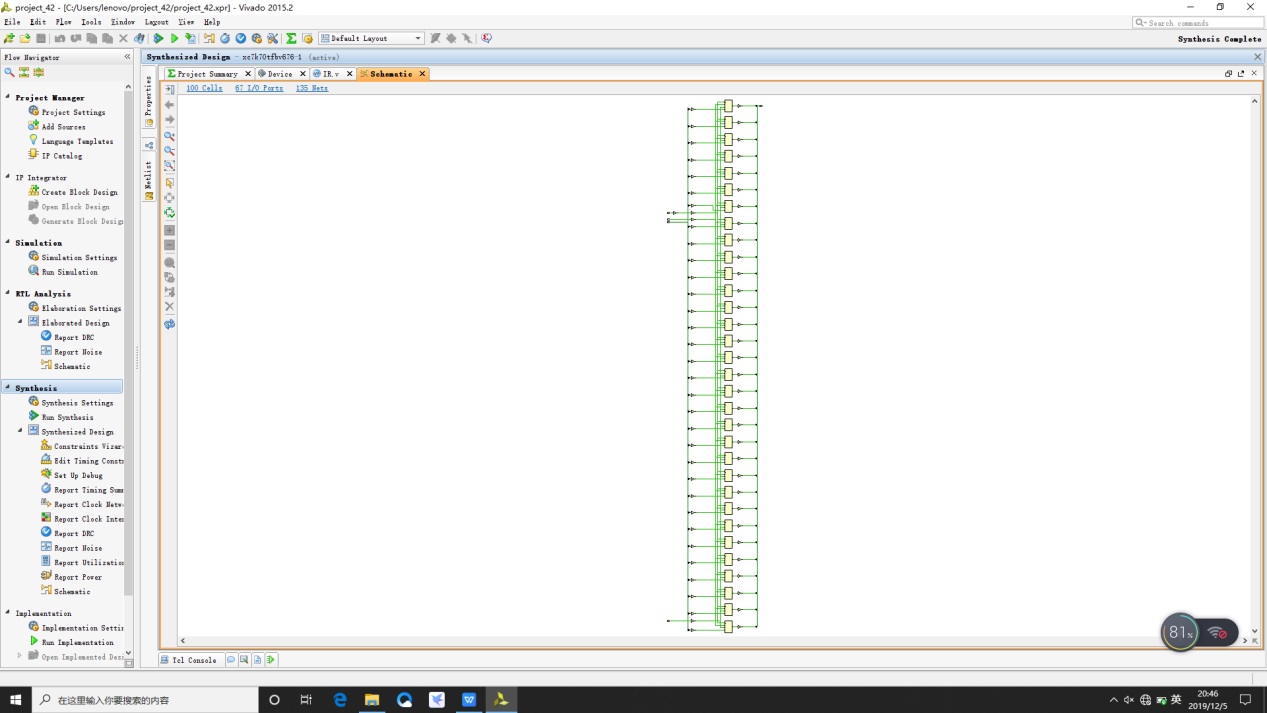
end

always #100 clk=~clk;

endmodule







## 4.数据缓冲器flopr

#### (1)基本描述

存储器数据寄存器MDR、存储器地址寄存器MAR、暂存器A、暂存器B、ALU运算结果寄存器ALUOut是由flopr模块实例化，主要功能是数据缓冲，由带复位的D触发器构成。



图4 flopr模块

#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| [WIDTH-1:0] d | I | 输入数据 |
| rst | I | 复位信号 |
| clk | I | 时钟信号 |
| [WIDTH-1:0] q | O | 输出数据 |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 数据复位 | if rst=0, q🠄d |
| 2 | 数据缓冲 | if rst=1, q🠄0 |

#### (4)模块描述

module flopr #(parameter WIDTH = 8)

(clk, rst, d, q);

input clk;

input rst;

input [WIDTH-1:0] d;

output [WIDTH-1:0] q;

reg [WIDTH-1:0] q\_r;

always @(posedge clk or posedge rst) begin

if ( rst )

q\_r<= 0;

else

q\_r<= d;

end // end always

assign q = q\_r;

endmodule

module test();

parameter WIDTH = 8;

reg clk;

reg rst;

reg [WIDTH-1:0] d;

wire [WIDTH-1:0] q;

flopr u0(.clk(clk),.rst(rst),.d(d),.q(q));

initial begin

rst=1;

clk=1;

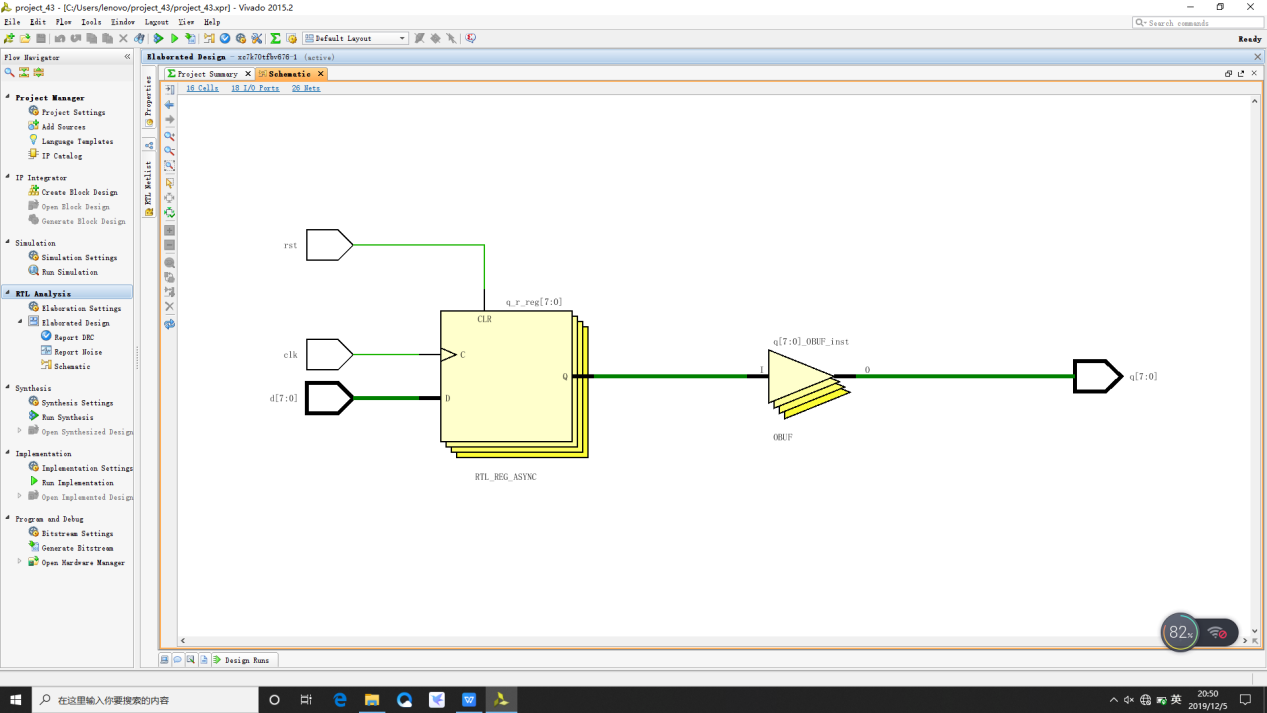
d=8'b01010101;

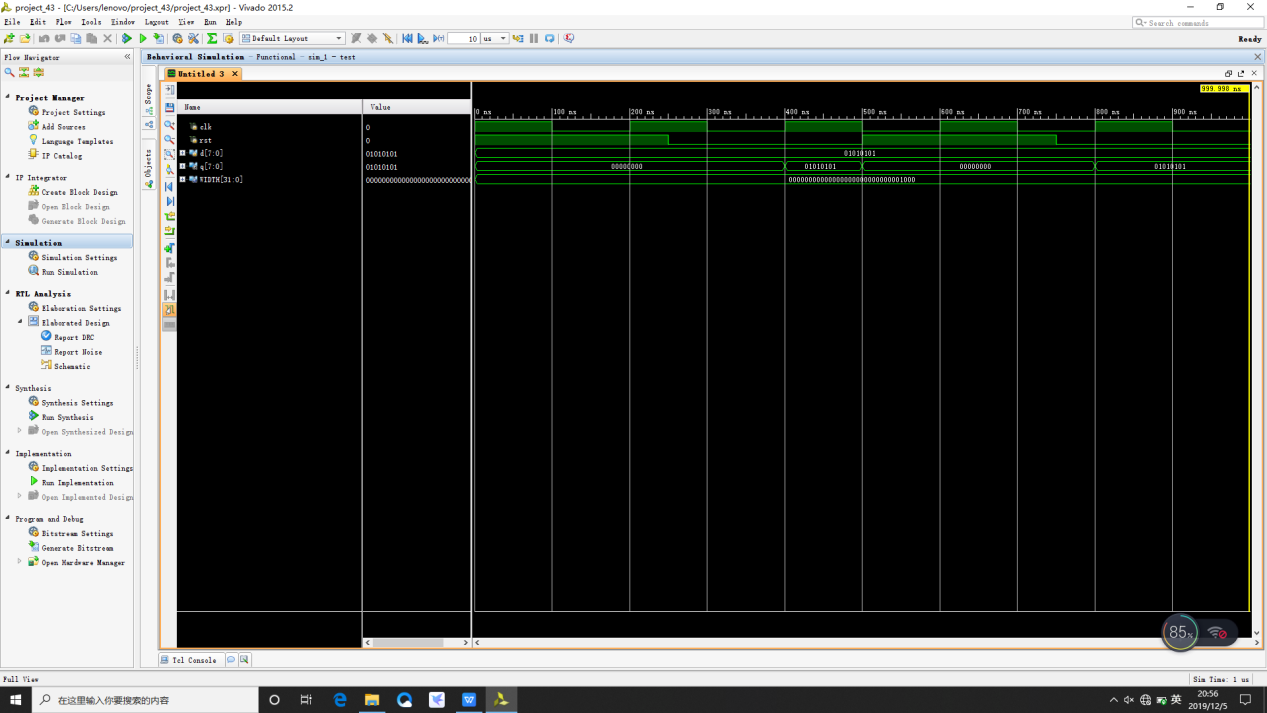
end

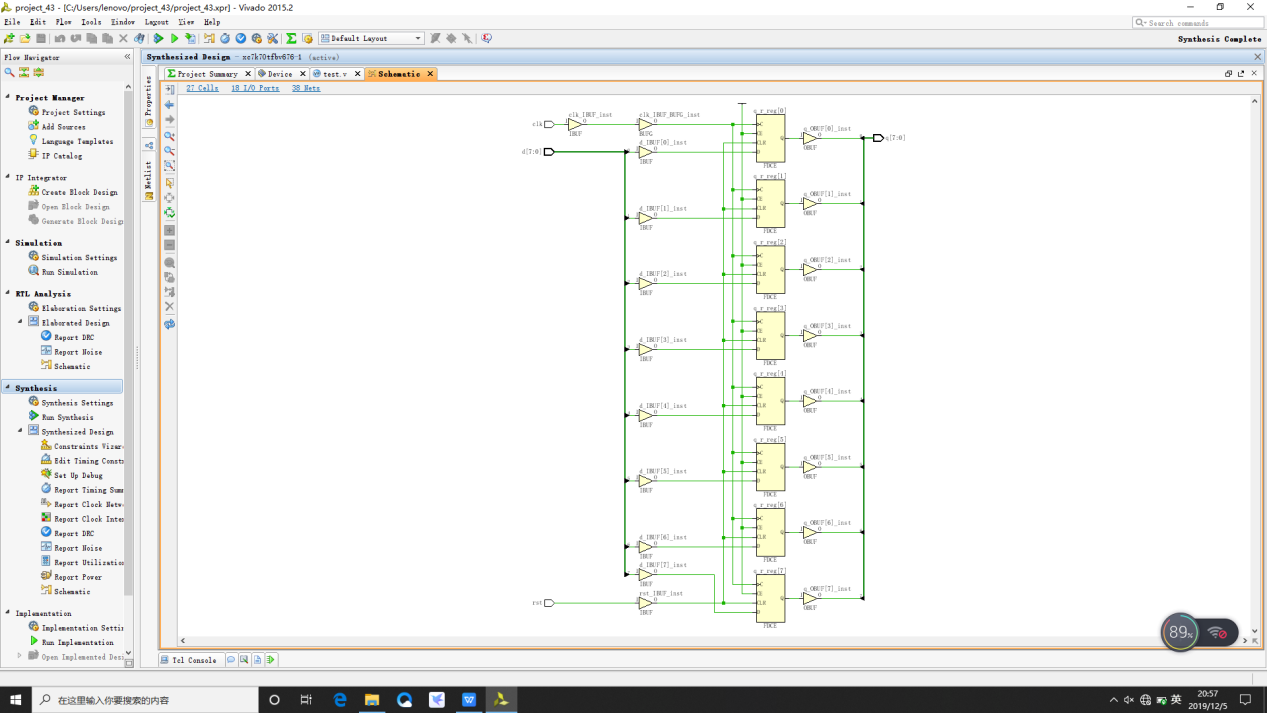
always #100 clk=~clk;

always #250 rst=~rst;

endmodule







## 5.寄存器文件RF

#### (1)基本描述

RF主要功能是保存寄存器文件，并支持对通用寄存器的访问。



图5 RF模块

#### (2)模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A1 [4:0] | I | 需要读的寄存器1的地址 |
| A2 [4:0] | I | 需要读的寄存器2的地址 |
| A3 [4:0] | I | 需要写的寄存器的地址 |
| WD [31:0] | I | 需要写的寄存器的数据 |
| RFWr | I | 寄存器写使能端  0：寄存器不写  1：寄存器写 |
| clk | I | 时钟信号 |
| RD1 [31:0] | O | 需要读的寄存器1的数据 |
| RD2 [31:0] | O | 需要读的寄存器2的数据 |

#### (3)功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读取通用寄存器 | 根据输入的RS、RT域的值，输出相应通用寄存器所存储的数据。  RD1🠄RF[A1]；RD2🠄RF[A2] |
| 2 | 写回通用寄存器 | 当RF写使能有效时，将待写的数据写入给定地址的通用寄存器中。  Clk上升沿时  if (RFWr) then  RF[A3]🠄WD |

#### (4)模块描述

`define DEBUG 1

module RF( A1, A2, A3, WD, clk, RFWr, RD1, RD2 );

input [4:0] A1, A2, A3;

input [31:0] WD;

input clk;

input RFWr;

output [31:0] RD1, RD2;

reg [31:0] rf;

integer i;

initial begin

for (i=0; i<32; i=i+1)

rf[i] = 0;

end

always @(posedge clk) begin

if (RFWr)

rf[A3] <= WD;

`ifdef DEBUG

$display("R[00-07]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", 0, rf[1], rf[2], rf[3], rf[4], rf[5], rf[6], rf[7]);

$display("R[08-15]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[8], rf[9], rf[10], rf[11], rf[12], rf[13], rf[14], rf[15]);

$display("R[16-23]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[16], rf[17], rf[18], rf[19], rf[20], rf[21], rf[22], rf[23]);

$display("R[24-31]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[24], rf[25], rf[26], rf[27], rf[28], rf[29], rf[30], rf[31]);

`endif

end // end always

assign RD1 = (A1 == 0) ? 32'd0 :rf[A1];

assign RD2 = (A2 == 0) ? 32'd0 :rf[A2];

endmodule

**【试用Verilog HDL建模方法对以上各模块进行建模，编写testbench进行仿真，并进行RTL分析、综合】**

`timescale 1ns / 1ps

module test();

reg [4:0]A1;

reg [4:0]A2;

reg [4:0]A3;

reg [31:0]WD;

reg clk;

reg RFWr;

wire [31:0]RD1;

wire [31:0]RD2;

RF u0(.A1(A1),.A2(A2),.A3(A3),.WD(WD),.clk(clk),.RFWr(RFWr),.RD1(RD1),.RD2(RD2));

initial

begin

RFWr=1;

clk=1;

A1=4'b0000;

A2=4'b0000;

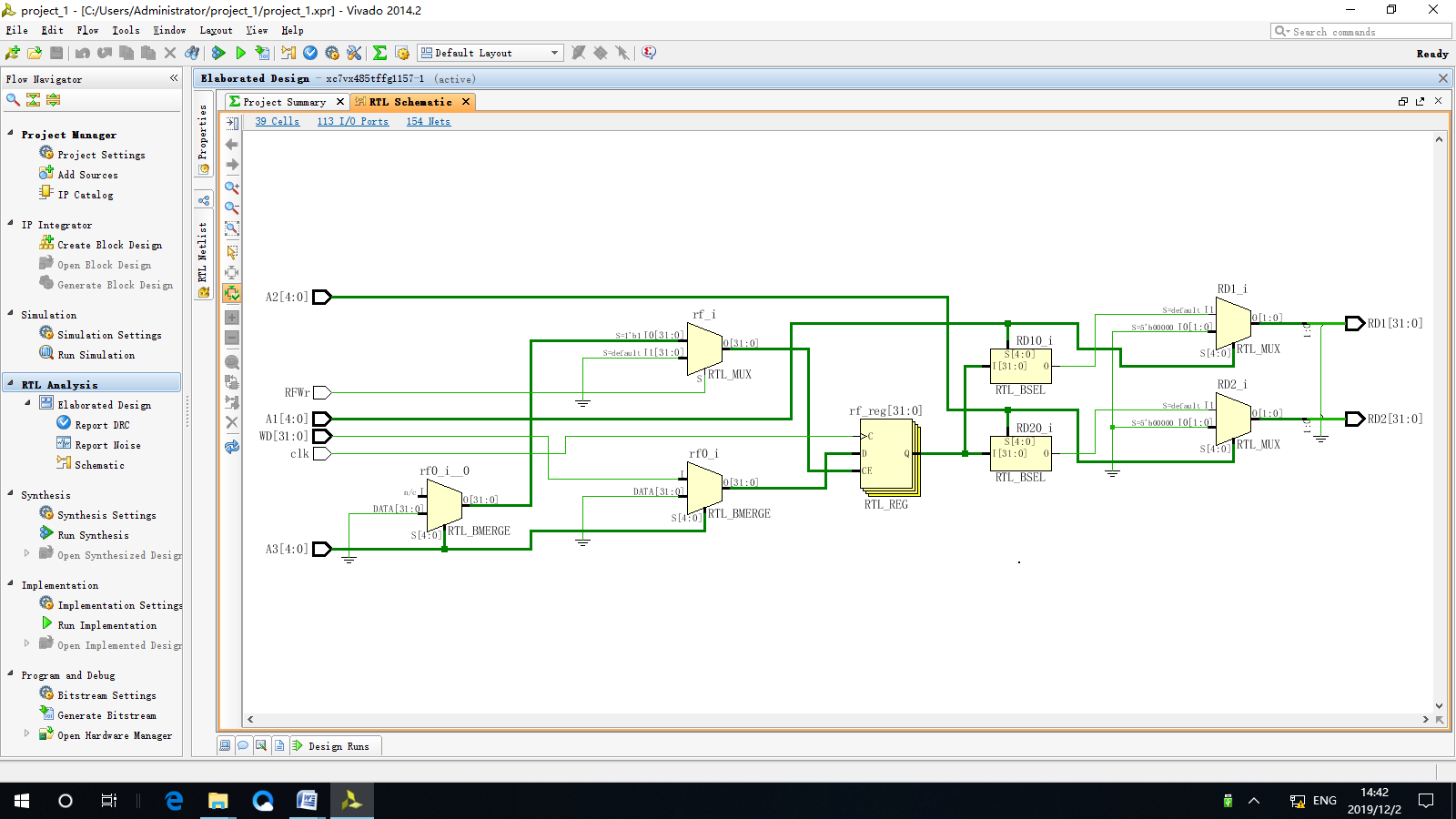
A3=4'b0000;

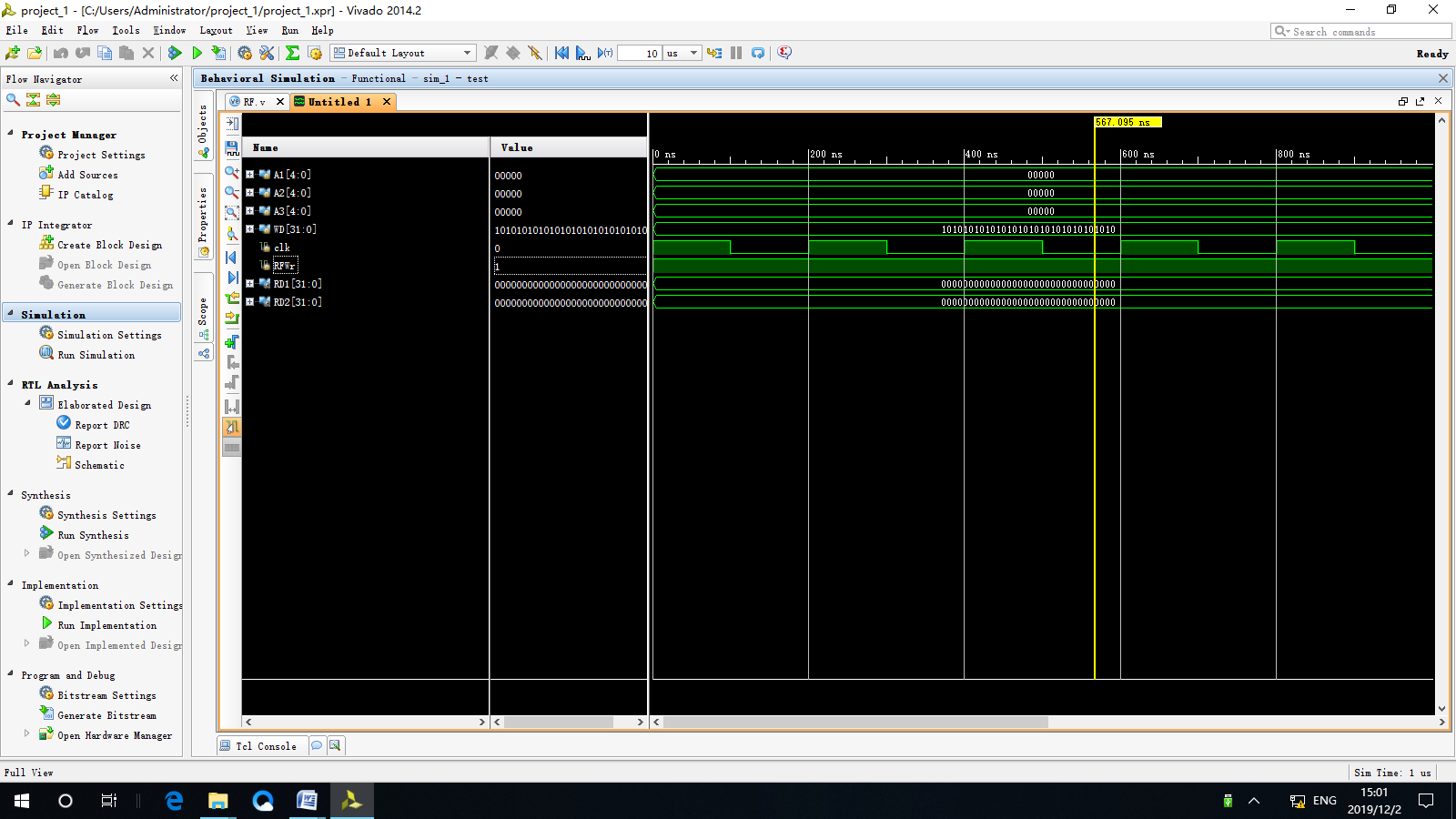
WD=32'b10101010101010101010101010101010;

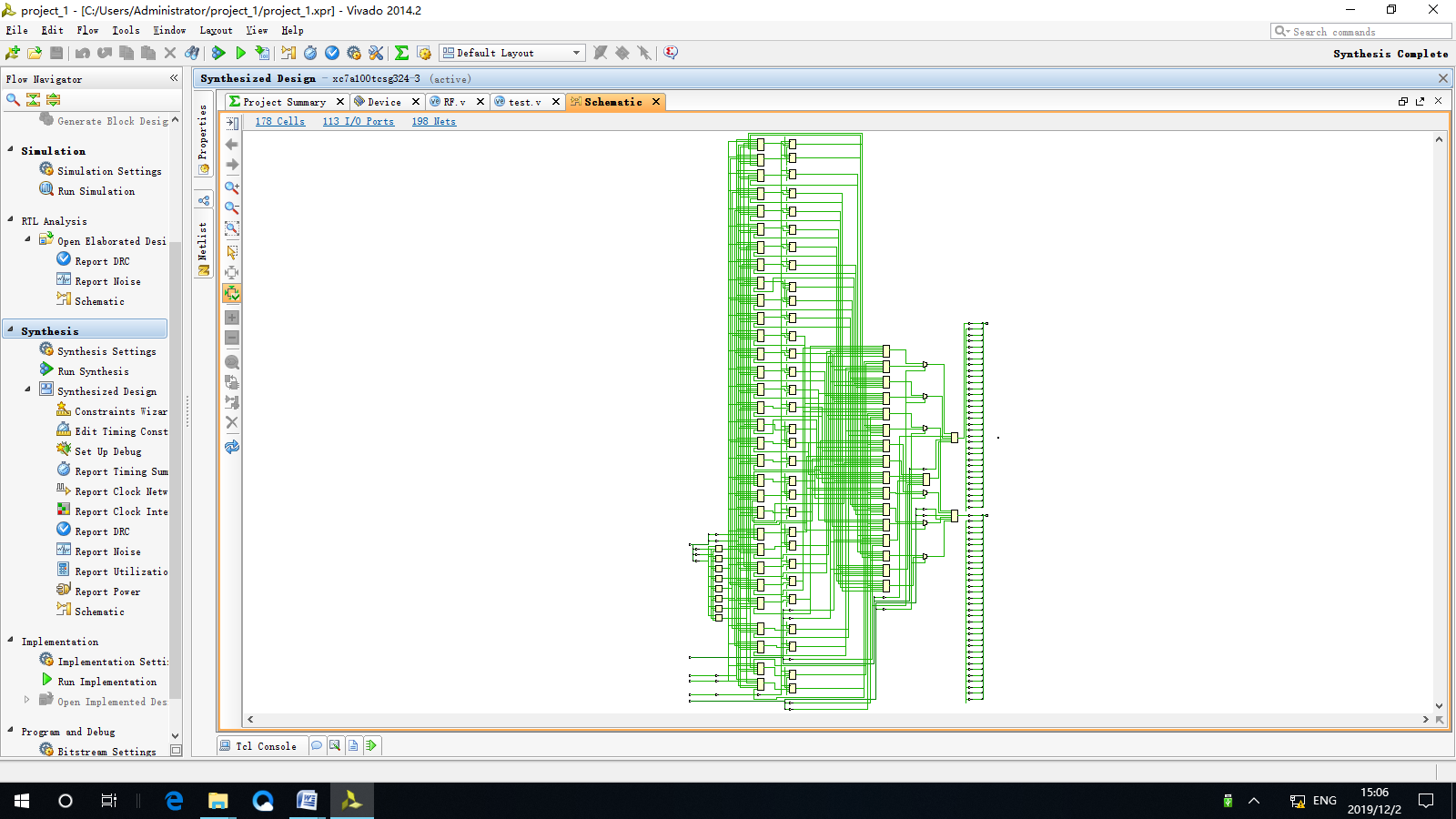
end

always #100 clk<=~clk;

endmodule







# 五、实验思考

## 1.D触发器是什么？除了D触发器还有哪几种触发器？如何用Verilog HDL门级建模方式建立边沿D触发器的模型？

图6 同步D触发器



图7 边沿D触发器

**【答】**

D触发器是一个具有记忆功能的，具有两个稳定状态的信息存储器件，是构成多种时序电路的最基本逻辑单元，也是数字逻辑电路中一种重要的单元电路。

RS触发器，JK触发器，T触发器

module D(

input D,CLK,

output Q,Qn

);

wire g118,g86,g75,g65,Qm,Qmn,g42,g31,CLKn;

not g11(g118,D);

nand g7(g75,D,CLK);

nand g8(g86,g118,CLK);

nand g5(Qm,g65,g75);

nand g6(g65,Qm,g86);

not g9(CLKn,CLK);

not g10(Qmn,Qm);

nand g3(g31,Qm,CLKn);

nand g4(g42,Qmn,CLKn);

nand g1(Q,g31,Qn);

nand g2(Qn,Q,g42);

endmodule

## 2.如何用D触发器实现数据寄存器和移位寄存器？

**【答】**

module yiwei(q,d,clk);

input d,clk;

output q;

wire d;

wire [7:0]q;

D u2(q[0],d,clk);

D u3(q[1],q[0],clk);

D u4(q[2],q[1],clk);

D u5(q[3],q[2],clk);

D u6(q[4],q[3],clk);

D u7(q[5],q[4],clk);

D u8(q[6],q[5],clk);

D u9(q[7],q[6],clk);

endmodule

module D(q,d,clk);

input d,clk;

output q;

reg q;

always @(posedge clk)

begin

q<=d;

end

endmodule