实验二 时序逻辑设计

**一、实验目的**

1 掌握Verilog语言和Vivado、Logisim开发平台的使用；

2 掌握基础时序逻辑电路的设计和测试方法。

**二、实验内容（使用Logisim或Vivado实现）**

1 锁存器、触发器的设计与测试

2 寄存器、计数器的设计与测试

3 状态机的设计与测试

**三、实验要求**

1 掌握Vivado或Logisim开发工具的使用，掌握以上电路的设计和测试方法；

2 记录设计和调试过程（Verilog代码/电路图/表达式/真值表，Vivado仿真结果，Logisim验证结果等）；

3 分析Vivado仿真波形/Logism验证结果，注重输入输出之间的对应关系。

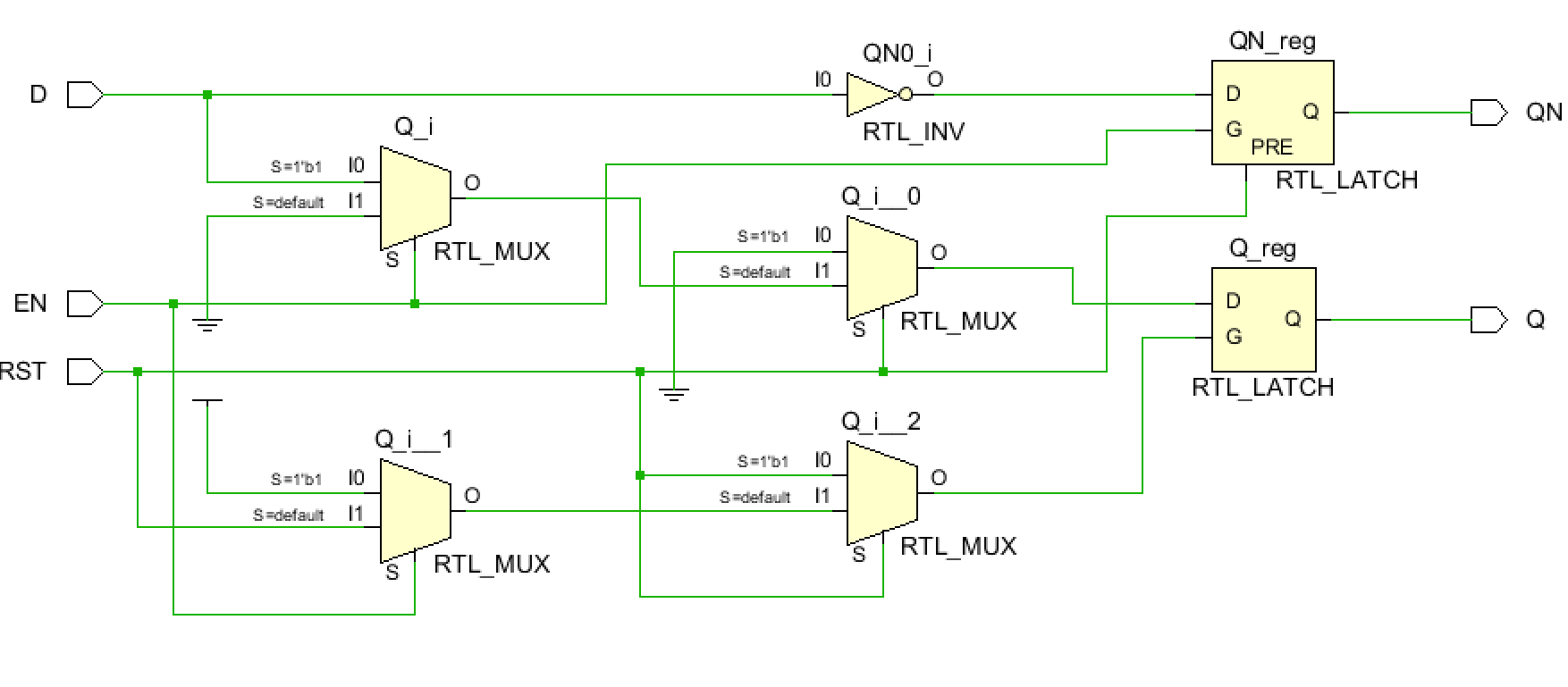
**四、实验过程及分析**

1、D锁存器的设计

(1)设计文件：

module  D\_latch(Q, QN, D, EN ,RST);  
 output reg Q,QNQ;  
 input D;  
 input EN,RST;  
 always@(EN,RST,D) begin  
 if(RST) begin  
 Q=0;  
 QN=1;  
 end  
 else if(EN) begin  
 Q<=D;  
 QN=~D;  
 end  
 end  
 endmodule

逻辑图：



(2)仿真文件：

module sim\_D\_latch();  
    reg D,EN,RST;  
    wire Q,QN;  
      
    initial begin  
        RST=0;EN=0;D=0;  
        fork  
            #10 RST=1;  
            #20 RST=0;EN=1;D=1;  
            #30 D=0;  
            #40 D=1;  
            #40 EN=0;D=0;  
            #50 RST=1;  
        join  
     end  
D\_latch D\_latch1(Q,QN,D,EN,RST);  
endmodule

(3)时序图：

 (4)代码分析

设计思路为，检测RST,EN,D其中有一个信号有效时开始更新状态。

如RST有效不管其他信号如何则置Q为0，~Q为1；

如果EN信号，则把Q更新为D，QN=~Q；

若EN无效，则不操作。

仿真文件遍历所有情况即可。

在时序图中可以观察到当RST无效，EN有效时，Q=D；

当RST有效时，无论EN有效无效，都使得Q=D；

RST变回无效时候，再去考虑EN，如果有效才进行操作Q=D。

2、SR锁存器

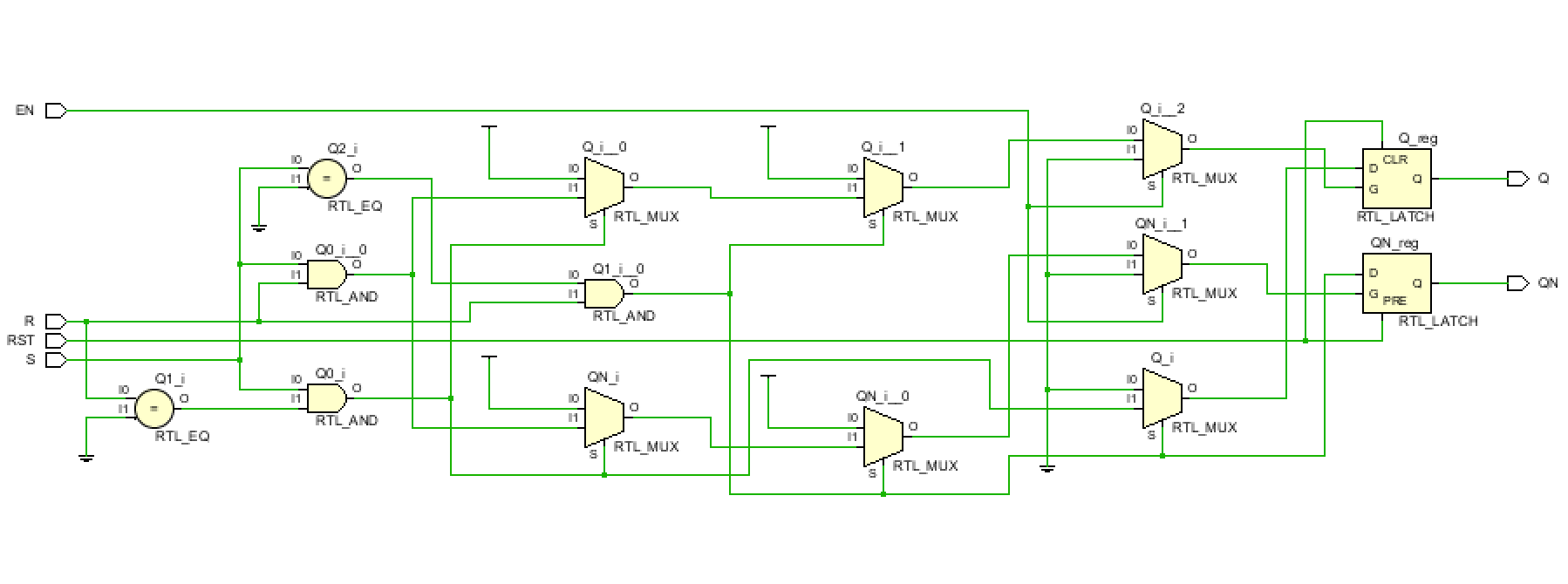
(1)设计文件：

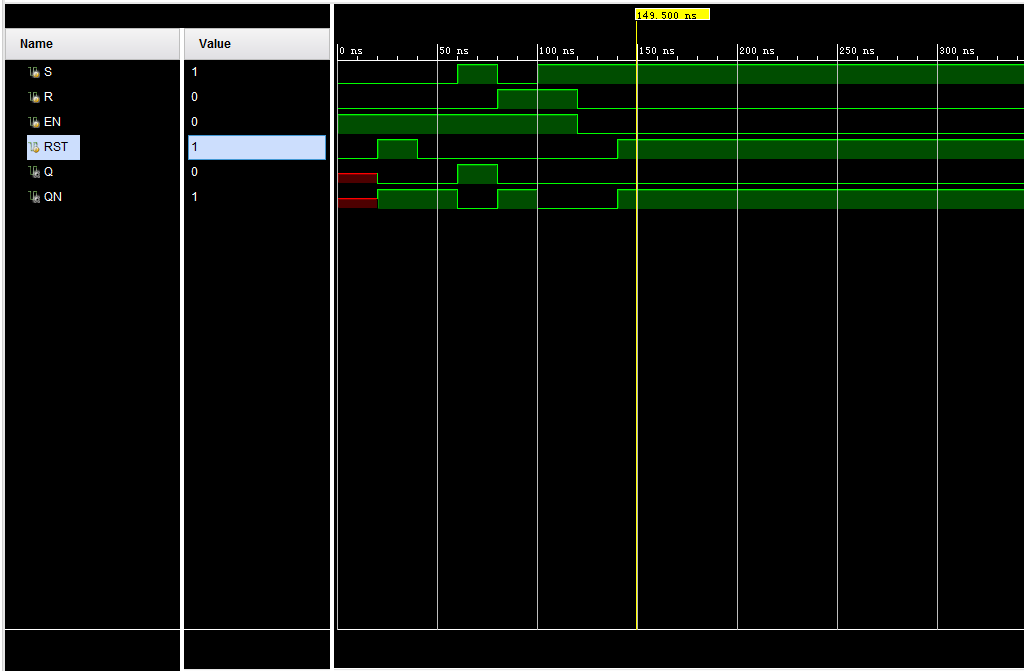
module SR\_latch(Q, QN, S,R, EN ,RST);  
output reg Q,QN;  
input S,R;  
input EN,RST;  
  
always@(EN ,RST,S,R) begin  
if(RST) begin  
Q=0;  
QN=1;  
end  
  
else if(EN) begin  
if(S==0&&R==1)  
 begin  
        Q=0;  
       QN=1;  
  end  
    
  else if(S==1&&R==0)  
  begin  
          Q=1;  
          QN=0;  
   end  
     
   else if(S==1&&R==1)  
      begin  
          Q=0;  
          QN=0;  
       end  
end  
end  
endmodule

(2)仿真文件：

module sim\_SR\_latch();  
    reg S,R,EN,RST;  
    wire Q,QN;  
      
    initial begin  
        RST=0;EN=1;S=0;R=0;  
            #20 RST=1;  
            #20 RST=0;  
            #20 S=1;  
            #20 R=1;S=0;  
            #20 S=1;  
            #20 EN=0;R=0;  
            #20 RST=1;  
        
     end  
SR\_latch SR\_latch1(Q,QN,S,R,EN,RST);  
endmodule

逻辑图：

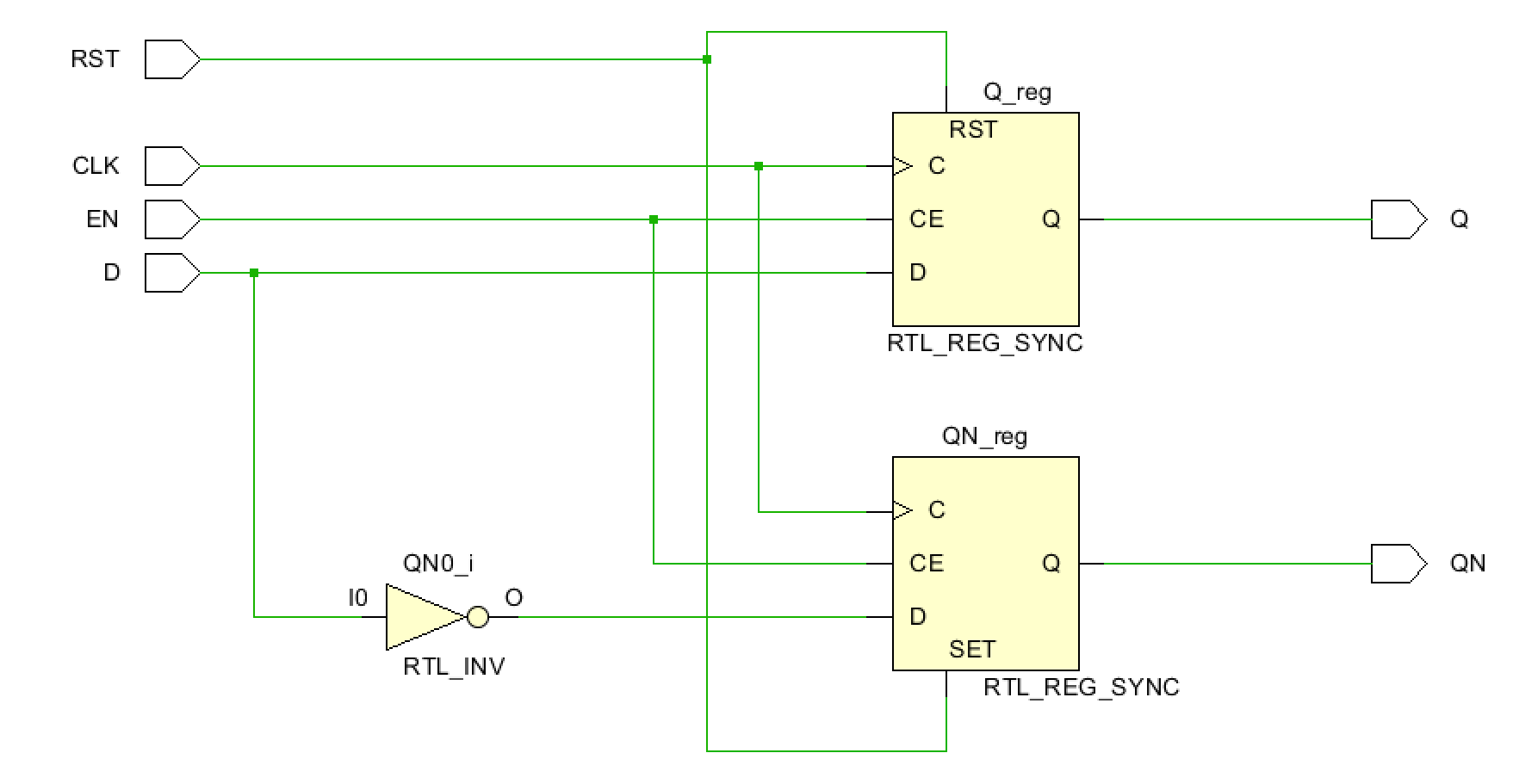
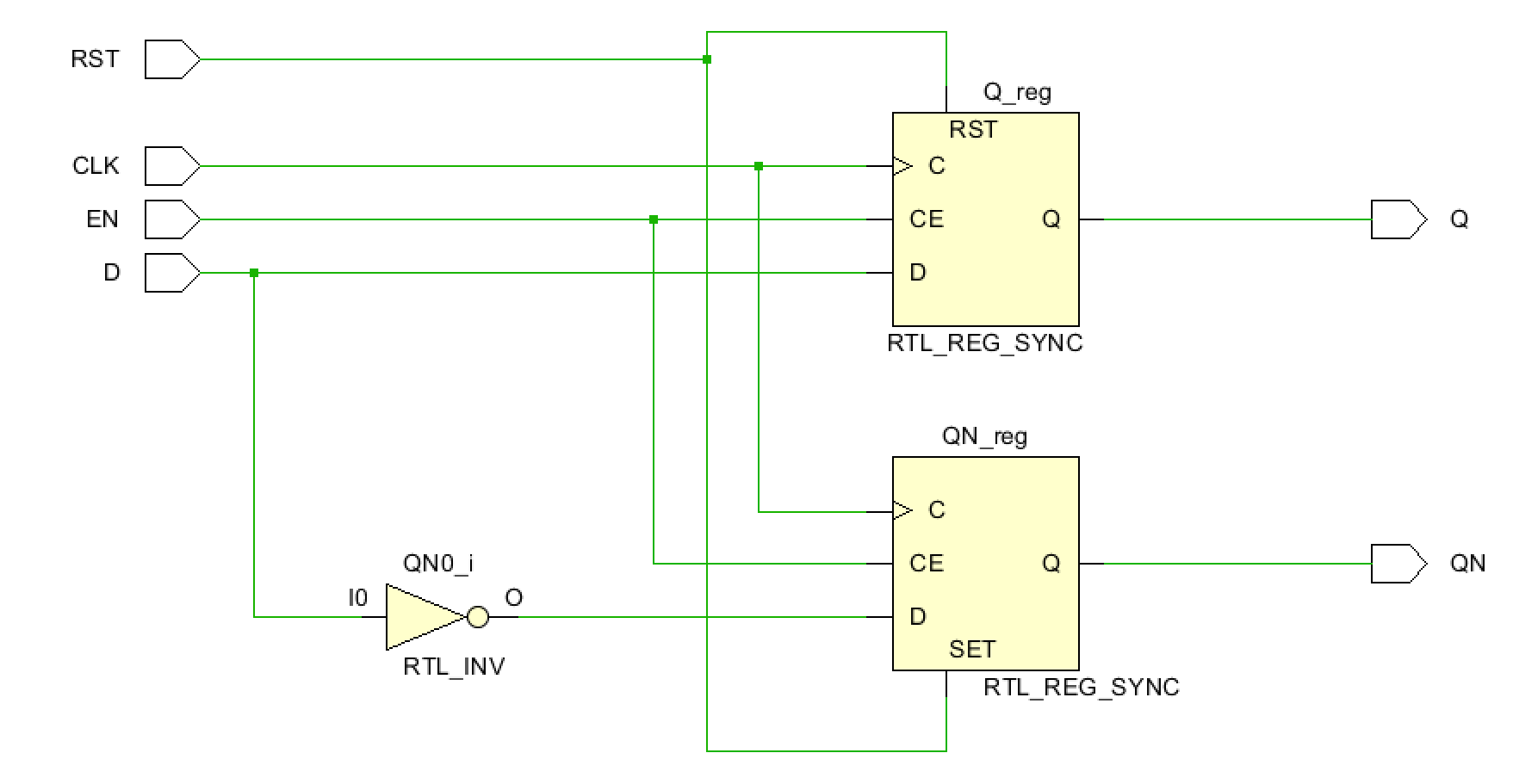


1. 时序图：
2. 代码分析：基本原则设计原则与D锁存器相同，只需要根据输出Q与输入信号S、R信号的真值表改变对应逻辑关系即可。

时序图中略有不同的是，最初把RST置为无效,EN置为有效，希望通过输入信号提供初始状态，但初始把S、R信号都置为0，表示状态不变，所以导致输出Q与QN初始时没有稳定状态，具体表现为在时序图中最初Q与QN的信号位于高、低电平之间且为红色。

1. D触发器
   1. 设计文件

module D\_ff(Q,QN,D,EN,RST,CLK);  
  
    output reg Q,QN;  
    input D;  
    input EN,RST,CLK;  
    always @(posedge CLK) begin  
          
        if(RST)  
          begin  
               Q<=1'b0;  
               QN<=1'b1;  
           end  
          else if(EN)  
            begin  
              Q<=D;  
              QN<=~D;  
            end  
         
    end  
endmodule

逻辑图

（2）仿真文件

* 1. module sim\_D\_ff();  
         reg D,EN,RST,CLK;  
         wire Q,QN;  
           
         initial   
           begin CLK=0;D=0;RST=0;EN=0;  
             fork  
                 repeat(200) #10 CLK=~CLK;  
                 repeat(100) #20 D=~D;  
                 repeat (50) #40 EN=~EN;;  
                 repeat(2) #160 RST=~RST;  
              join  
         end  
     D\_ff D\_ff1(Q,QN,D,EN,RST,CLK);  
     endmodule
  2. 时序图



* 1. 代码分析

触发器可以看成只有在始终上升沿才会生效的锁存器。所以在设计文件中使用always@(postage CLK),表示只有在始终上升沿才对输出变量进行更新。

仿真文件主要遍历所有功能，主要验证是否实现以下几个功能：

只有在时钟上升沿才能完成状态改变；

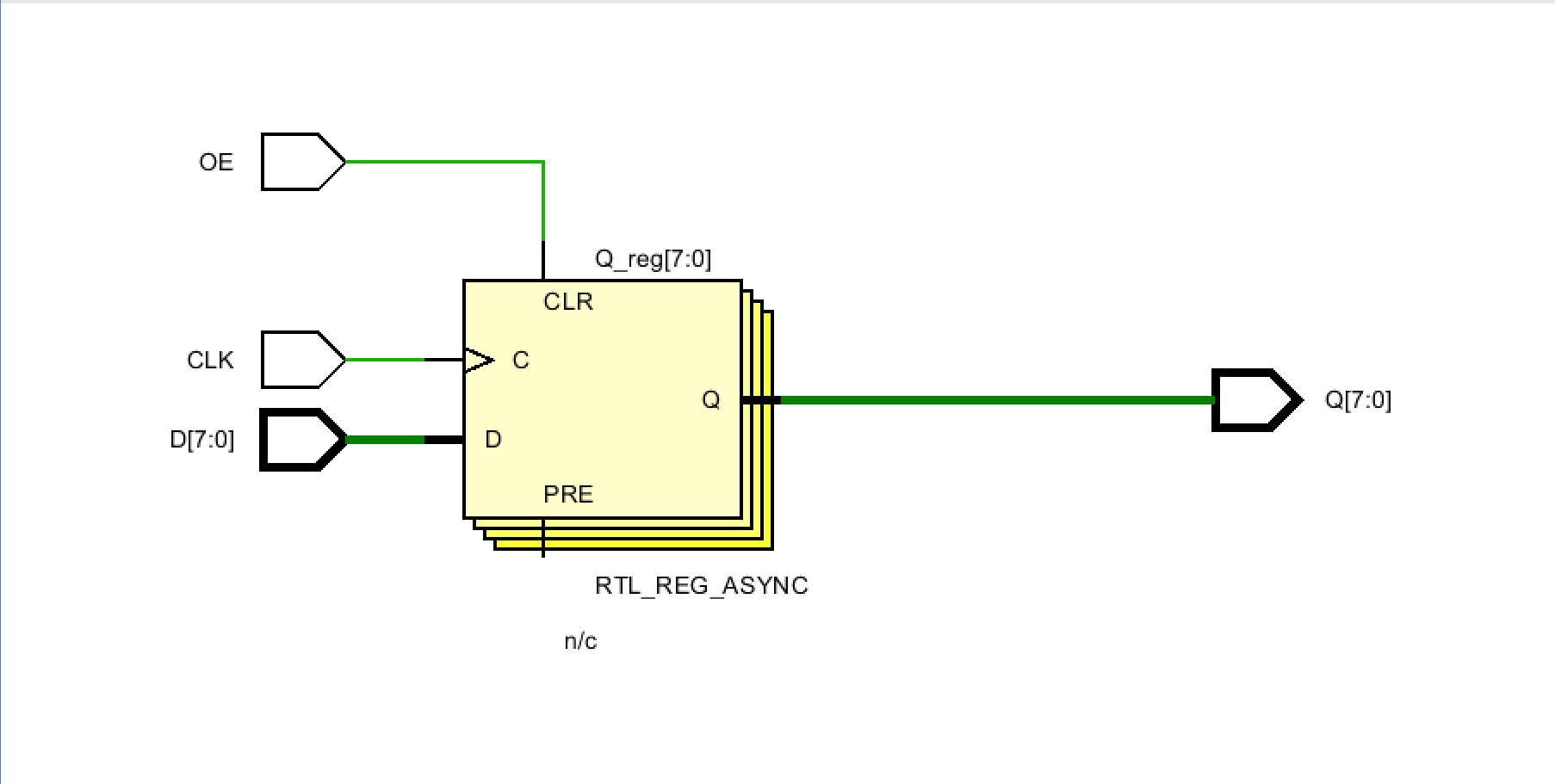
RST信号优先级高于EN/D信号；

EN信号有效时才能实现Q=D；

经过时序图验证满足以上功能。

1. 寄存器
   1. 设计文件

module register(Q, D, OE, CLK );  
   parameter N = 8;  
   output reg [N-1:0] Q;  
   input [N-1:0] D;  
   input OE, CLK;  
     
   always @ (posedge CLK or posedge OE)  
    if(OE) Q <= 8'bzzzz\_zzzz;  
    else Q <= D;  
     
endmodule

逻辑图：

* 1. 仿真文件

module sim\_registor();

reg OE,CLK;

reg [7:0] D;

wire [7:0] Q;

initial begin

D=55;CLK=0;OE=1;

fork

repeat(10) #10 CLK=~CLK;

#20 OE=~OE;

#50 D=44;

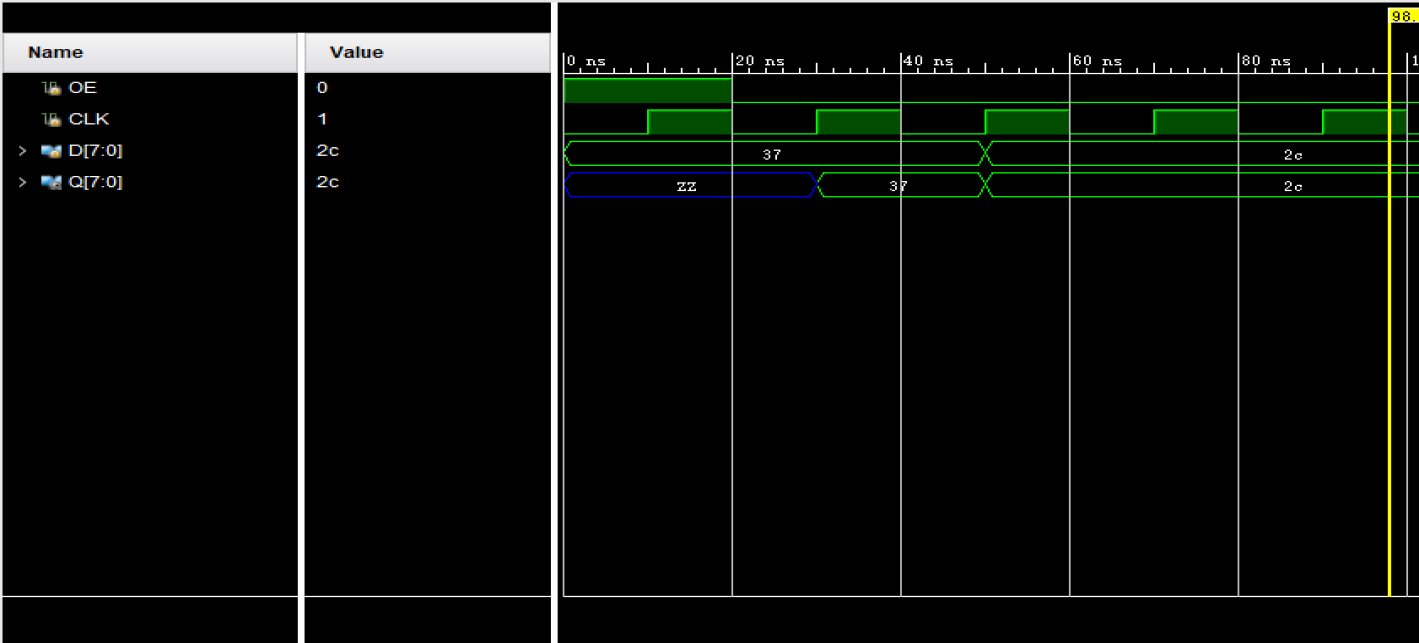
join

end

register registor\_1(Q, D, OE, CLK );

endmodule

* 1. 时序图



* 1. 代码分析

寄存器实现与触发器很相似，必须要在始终上升沿时才能将输入D同步给输出Q，只是输入位数和输出位数发生了变化。此外，增加了是使能无效端信号OE，使输出信号为高阻态。

时序图要验证的功能如下：

只有在时钟上升沿能完成输出信号Q的更新。

OE有效能使输出为无效高阻态。

OE无效时，每个时钟上升沿时，会进行Q=D的操作。

由时序图可知，上述功能满足。

1. 移位寄存器
   1. 设计文件

module shift\_register(S1, S0, D, Dsl, Dsr, Q, CLK, CR);

parameter N = 4;

input S1, S0;

input Dsl, Dsr;

input CLK, CR;

input [N-1:0] D;

output [N-1:0] Q;

reg [N-1:0] Q;

always @ (posedge CLK or posedge CR)

if(CR)

Q <= 0;

else

case({S1, S0})

2'b00: Q <= Q;

2'b01: Q <= {Dsr, Q[N-1:1]};

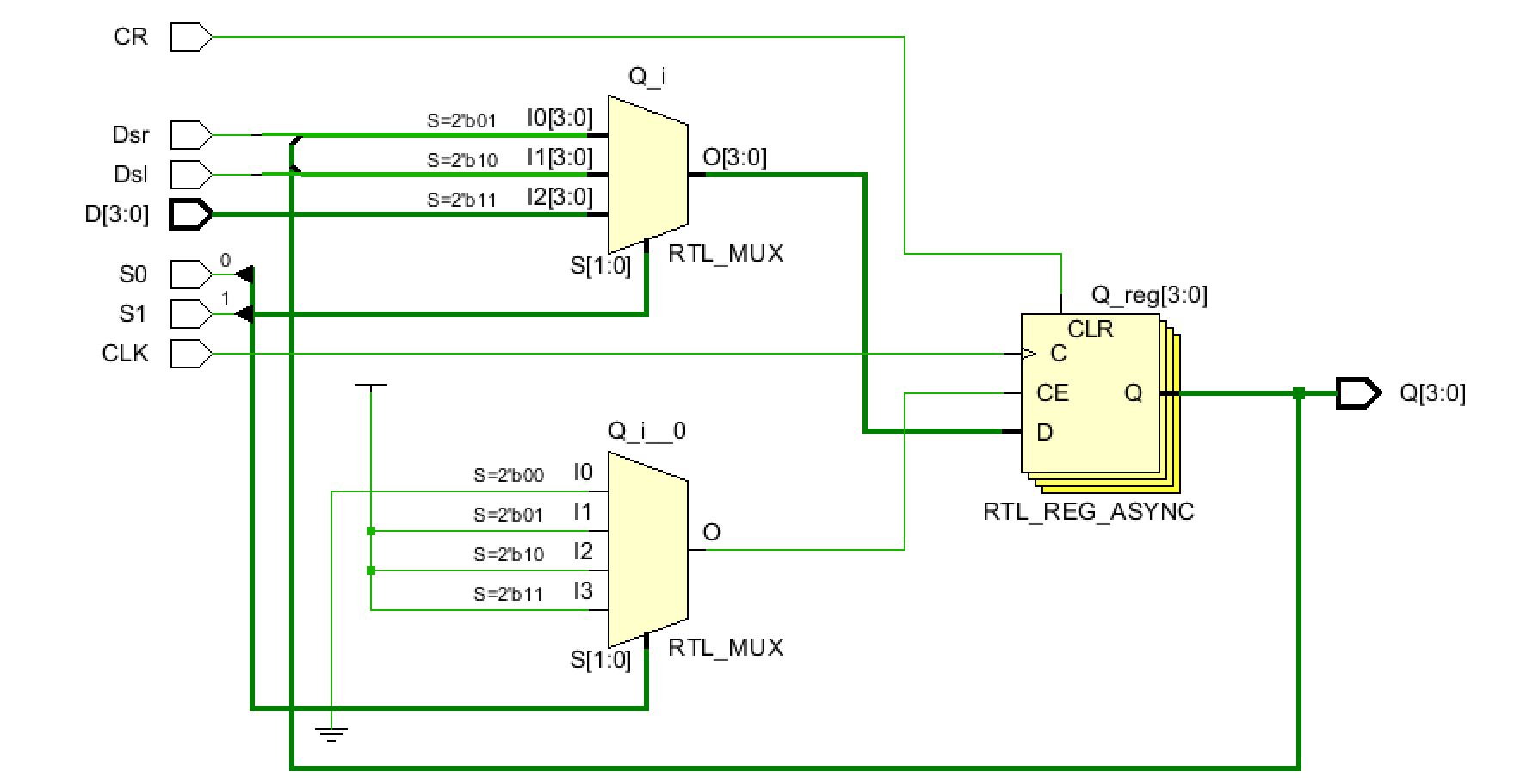
2'b10: Q <= {Q[N-2:0], Dsl};

2'b11: Q <= D;

endcase

Endmodule

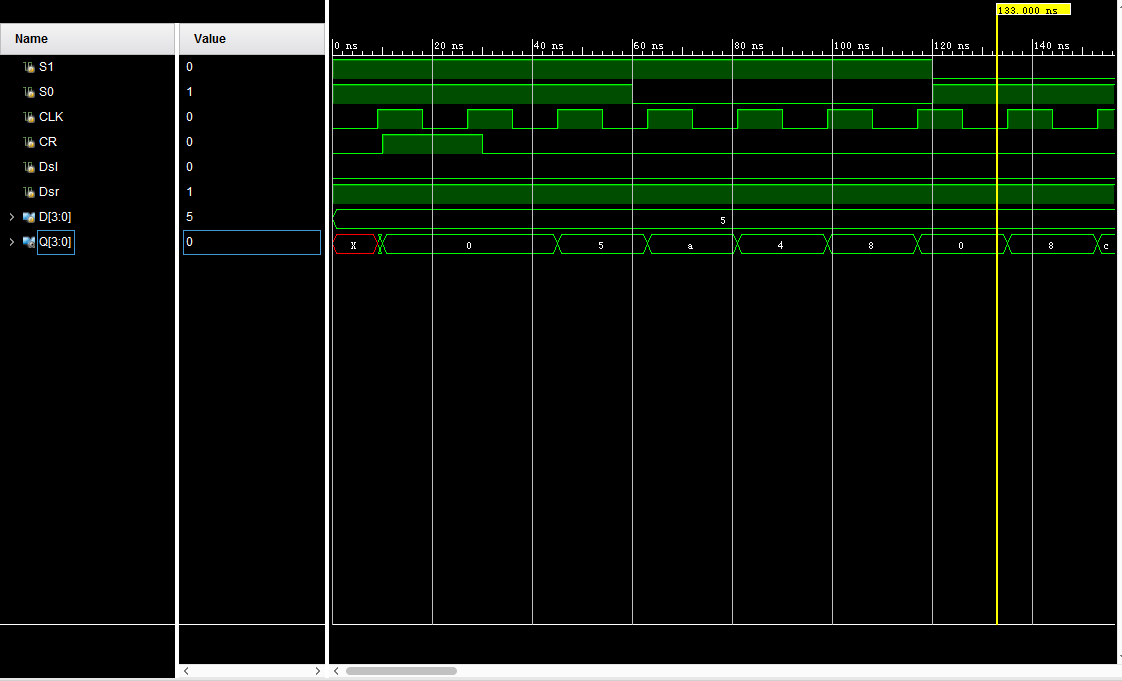
逻辑图：



* 1. 仿真文件

module sim\_shift\_register ();  
  
   reg S1, S0,CLK,CR,Dsl,Dsr;  
   reg[3:0] D;  
   wire [3:0] Q;  
   shift\_register shift\_register\_1(S1, S0, D, Dsl, Dsr, Q, CLK, CR);  
    initial begin     
        D=4'b0101;Dsl=0;Dsr=1;CLK=0;CR=0;S1=1;S0=1;  
        fork   
            repeat(100) #9 CLK=~CLK;  
            #10 CR=~CR;  
            #30 CR=~CR;  
            repeat(10)#120 S1=~S1;  
            repeat(10)#60  S0=~S0;  
        join  
end  
endmodule

* 1. 时序图



* 1. 代码分析

移位寄存器主要是在寄存器的基础上增加了选择信号S1、S0，通过（S1,S0)不同组合来控制寄存器的功能。其他原理与普通寄存器相同

由电路时序图可知：

CR信号可以将将输出信号置零且优先级高于其他信号。

(S1,S0)=(1,0)时左移1位，并由Dsl(0)信号补全最低位：

5=4’b0101 左移且用0补齐最低位：1010=4’b1010=10=a

后续同理变为8、0；

(S1,S0)=(0,1)时右移1位，并由Dsr(1)信号补全最高位

0=4’b0000 右移且用1补全最高位：1000=8

后续同理

1. 69进制计数器（学号后2位）
   1. 设计文件

module counter\_N(CEP, CET, PE, CLK, CR, D, TC, Q);

parameter N = 8;

parameter M = 69;

input CEP, CET, PE, CLK, CR;

input [N-1:0] D;

output reg TC;

output reg [N-1:0] Q;

wire CE;

assign CE = CEP & CET;

always @ (posedge CLK, negedge CR)

if(~CR) begin Q <= 0; TC = 0; end

else if(~PE) Q <= D;

else if(CE) begin

if(Q == M-1) begin

TC <= 1;

Q <= 0;

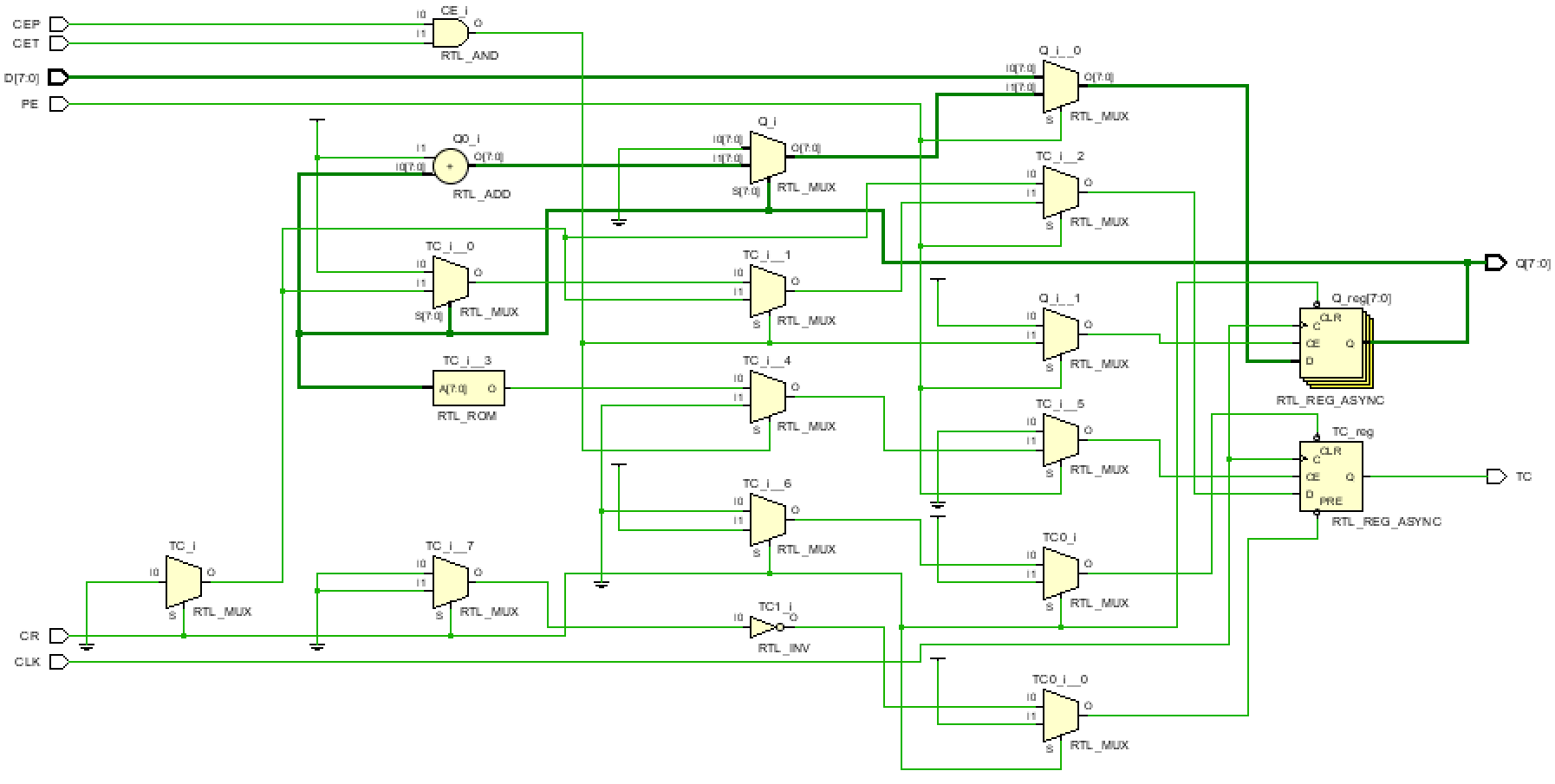
end

else Q <= Q +1;

end

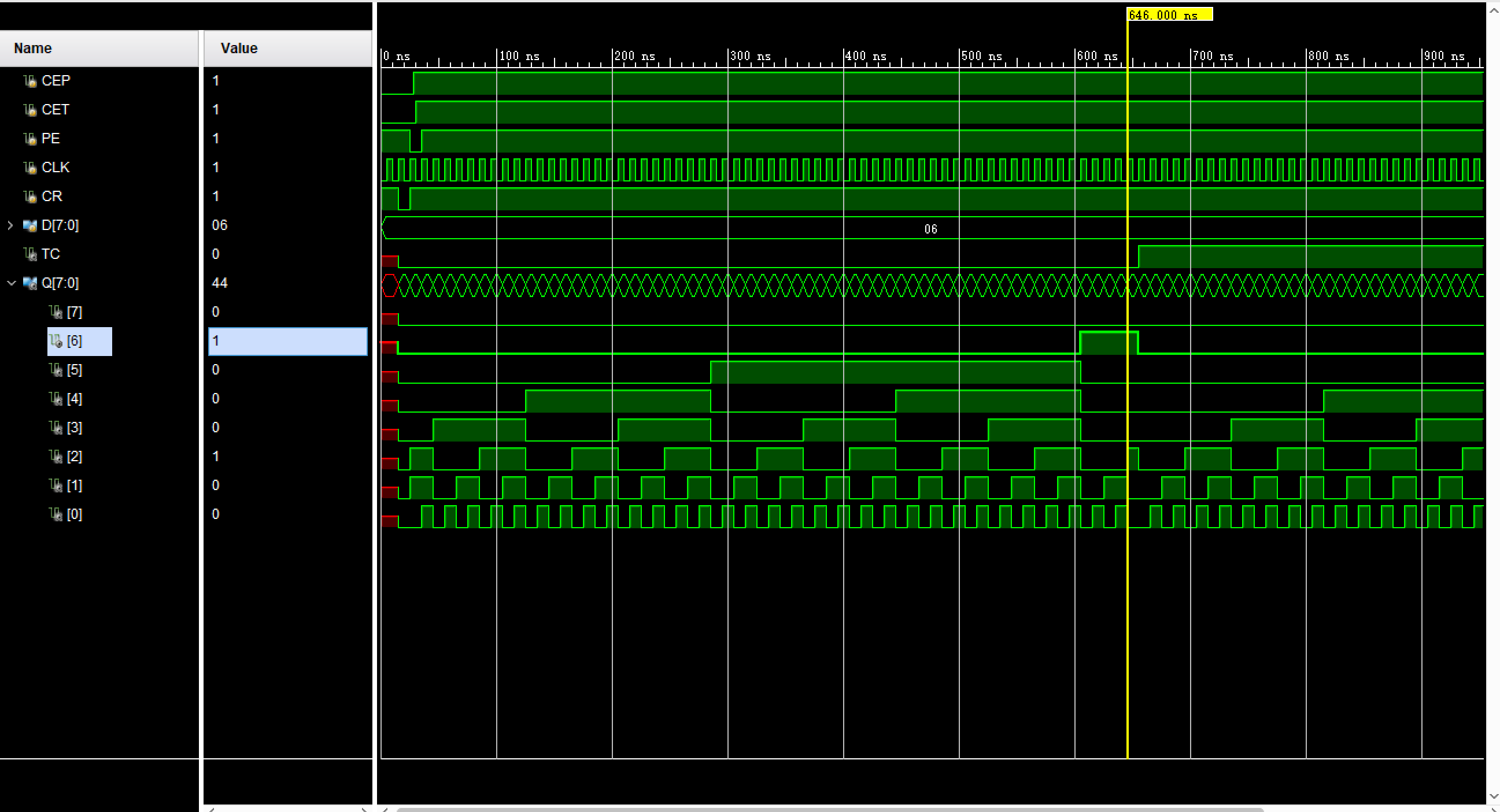
else Q <= Q;

endmodule

逻辑图：

* 1. 仿真文件

module sim\_counterN();  
reg CEP,CET,PE,CLK,CR;  
reg[7:0] D;  
wire TC;  
wire[7:0] Q;  
  
initial begin  
D=6;CLK=0;CEP=0;CET=0;PE=1;CR=1;  
fork  
repeat(1000)#5 CLK=~CLK;  
#28 CEP=~CEP;  
#30 CET=~CET;  
#25 PE=~PE;  
#35 PE=~PE;  
#15 CR=~CR;  
#25 CR=~CR;  
join  
end  
counter\_N counter69(CEP, CET, PE, CLK, CR, D, TC, Q);  
endmodule

* 1. 时序图
  2. 代码分析

大部分功能实现与前面已有模块基本相同，不再赘述，主要描述自增功能的实现和进位的实现。

设计当控制信号CE=CEP&CET==1时进入计数模式；

使用8位数Q计数，2^8=256，可以实现256进制内的所有计数器。

设置进制数为M，当Q<M-1时，每经过一个CLK上升沿，Q=Q+1实现自增，以此计数。

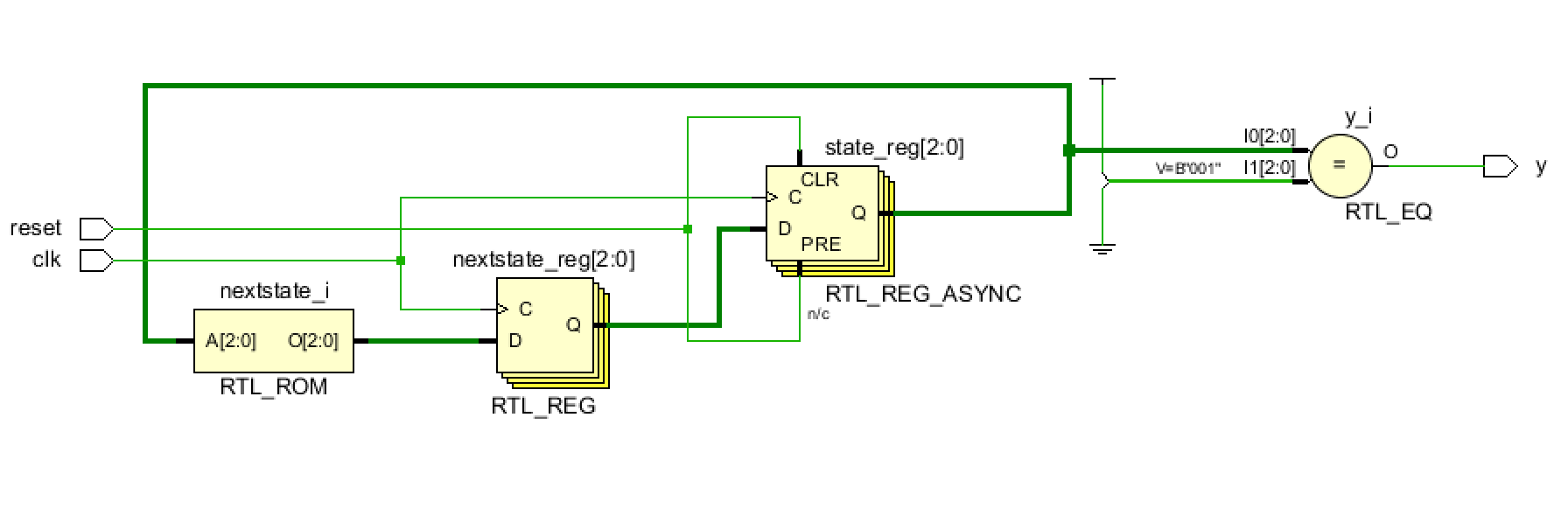
当Q=M-1时，迎来下一个始终上升沿时，由于从0开始计数，实际已计数M次，需要进位，置TC=1，且需把Q清零，使得Q=0；

时序图为实现的69进制计数器，可以观察到在16进表示下的44（即10进制下68），当来临下一个时钟信号，Q随机变为0，表示已经过了一个计数周期，TC=1表示进位。

1. 状态机1
   1. 设计文件

module FSM\_case(input clk,  
                input reset,  
                output y );  
    reg[2:0] state, nextstate;  
    always @(posedge clk, posedge reset)  
        if(reset) state = 2'b001;  
        else state = nextstate;  
    always@(posedge clk)  
    case(state)  
         'b001: nextstate = 'b010;  
         'b010: nextstate = 'b100;  
         'b100: nextstate = 'b001;  
         default: nextstate = 'b001;  
     endcase  
     assign y = (state == 'b001);  
endmodule

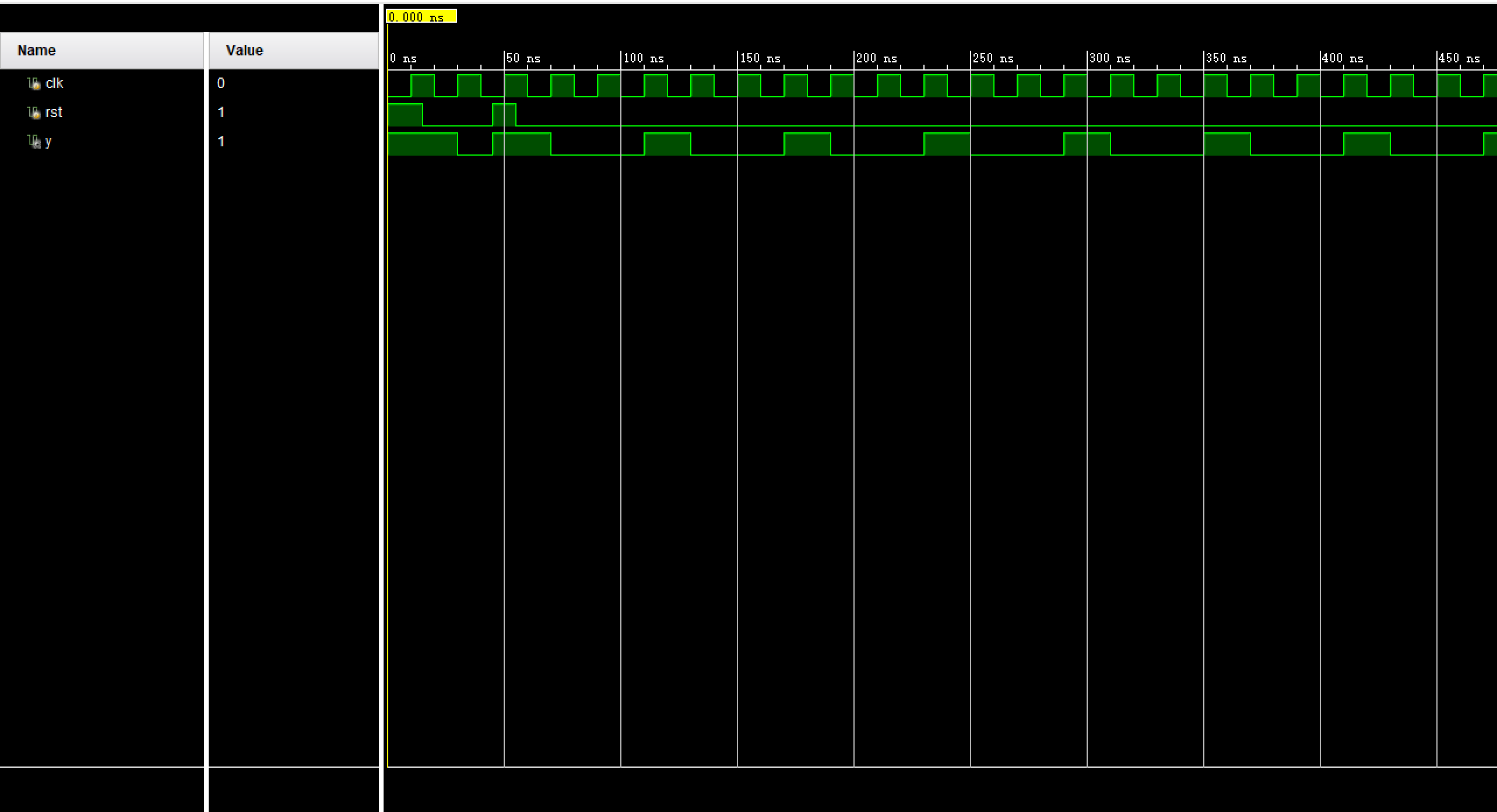
逻辑图：



* 1. 仿真文件

module sim\_FSM\_case();  
  
reg clk,rst;  
wire y;  
  
initial begin  
  
clk=0;rst=1;  
    fork  
   repeat(100) #10 clk=~clk;  
     
   #15   rst=0;  
   #45  rst=1;  
   #55  rst=0;   
      
    join  
end  
FSM\_case fsmcase1(clk,rst,y);  
endmodule

* 1. 时序图



* 1. 代码分析

状态机一要实现的功能：

RST信号将状态机置为输出状态。

状态机由三个状态组成，初始状态输出高电平，其他状态输出低电平。

每个时钟信号上升沿状态机发生一次状态改变。

使用always@(postage CLK)，使得每次检测到始终信号上升沿进行操作。

使用always@(rst),只要rst信号有效，则直接设置状态机的状态。

1. 状态机2
   1. 设计文件

module FSM\_cf(input clk,

input reset,

input a,

output y);

reg[1:0] state, nextstate;

always @(posedge clk, posedge reset)

if(reset) state = 2'b00;

else state = nextstate;

always @(posedge clk)

case(state)

'b00: if(a) nextstate = 'b00;

else nextstate = 'b01;

'b01: if(a) nextstate = 'b10;

else nextstate = 'b01;

'b10: if(a) nextstate = 'b00;

else nextstate = 'b01;

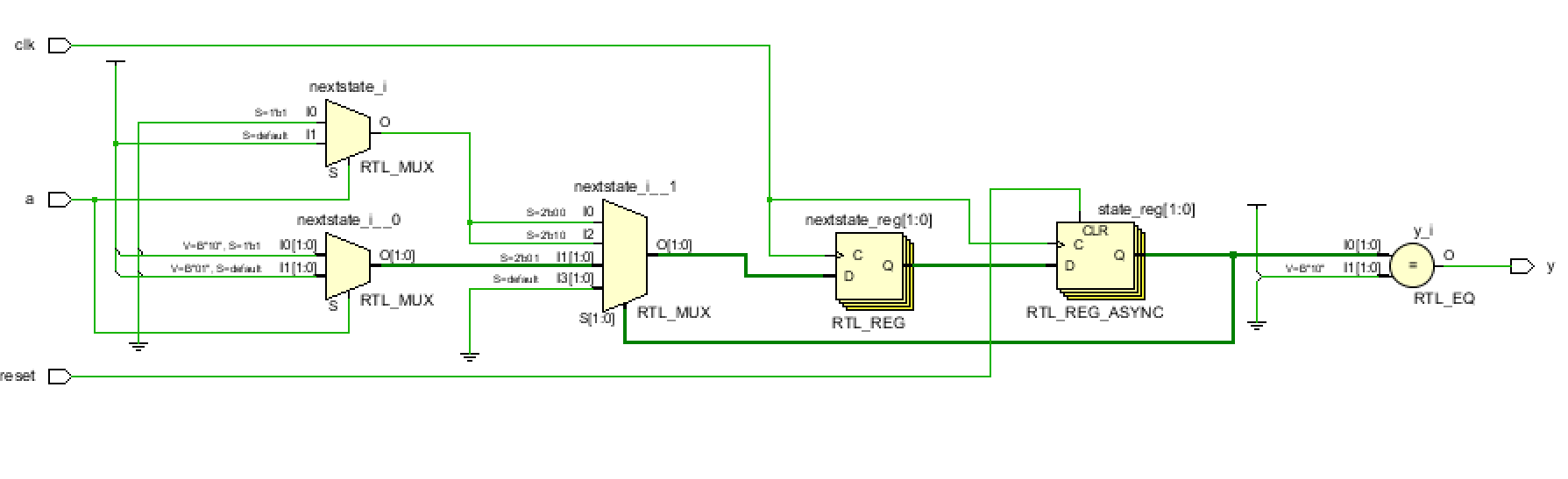
default: nextstate = 'b00;

endcase

assign y = (state == 'b10);

endmodule

逻辑图：



* 1. 仿真文件

module sim\_FSM\_cf();

reg clk,reset,a;

wire y;

initial begin

clk=0;reset=1;a=1;

fork

repeat(100) #10 clk=~clk;

#15 reset=0;

#25 a=~a;;

#45 a=~a;

#75 a=~a;

#85 a=~a;

join

end

FSM\_cf fsmcase1(clk,reset,a,y);

endmodule

* 1. 时序图
  2. 代码分析

本模块主要新增功能为引入临时变量nextstate存储器，用来存储下一个要进入的状态。每当检测到一个时钟上升沿，根据现有状态和输入信号a来确定nextstate，而输出y只与当前状态相关。

发现的问题：注意到本设计文件存在一些问题，状态的更新总落后一步，相当于一个时钟上升沿和此时输入信号所确定的下一个状态，总要等到下一个时钟上升沿才会赋值给state完成状态更新，修改后的代码如下：

module FSM\_cf(input clk,

input reset,

input a,

output y);

reg[1:0] state;

always @(posedge clk, posedge reset)

if(reset) state = 2'b00;

// else state = nextstate;

always @(posedge clk)

if(~reset) begin

case(state)

'b00: if(a) state = 'b00;

else state = 'b01;

'b01: if(a) state = 'b10;

else state = 'b01;

'b10: if(a) state = 'b00;

else state = 'b01;

default: state = 'b00;

endcase

end

assign y = (state == 'b10);

endmodule

设计上直接舍弃掉nextstate临时变量，直接根据此时的状态和输入信号a对state完成更新。

**五、调试和心得体会**

本次实验中由于对串行begin end和并行fork join语句和阻塞赋值和非阻塞赋值的僚机额不够充分全面，混乱使用，导致代码运行出错，找到老师现场指导后了解到了错误原因所在，后对串行并行、阻塞赋值和非阻塞赋值重新做了完整了解。

本次实验的调试和设计过程让我意识到了仿真文件的重要性，之前很多仿真文件都是对照着提供的课程PPT的时序图设计仿真文件，这次尝试自己设计仿真文件，发现仿真文件既要时间上分配合理，并且一定要在有限的时间里组合所有可能的输入，来验证所有的输入信号情况是否符合我们设计的条件，不能有遗漏，如果仿真文件设计不当，即使设计文件设计无误，一些功能也无法在电路时序图中体现出来。