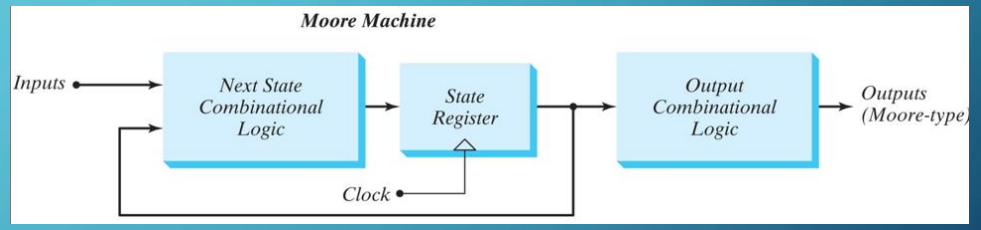
DIGITAL DESIGN

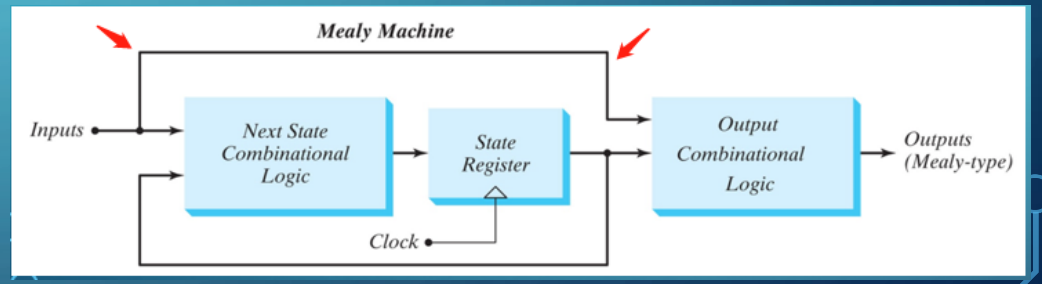
LAB15 VERILOG-SUMMARY(2)-SEQUENTIAL CIRCUIT

WANGW6@SUSTECH.EDU.CN

LAB15

- Verilog summary(2)-Seqential Circuit
  - initial state, reset
  - procedure assignemnt
    - blocking vs non-blocking
  - 组合逻辑环带来的问题
  - 两段式替代一段式。。。
  - multiple-driver
  - shift operator
- 开发板的相关操作





INITIAL STATE(1)

```
module stateTest(
input clk,output reg y);
always @(posedge clk)
    y <= ~y;
endmodule
```

```
module shiftOpSim( );
reg clk;
wire sy;
stateTest u1(.clk(clk),.y(sy));
initial begin
        clk = 1'b0;
        #20 $finish();
end
initial
forever #5 clk = !clk;
endmodule
```

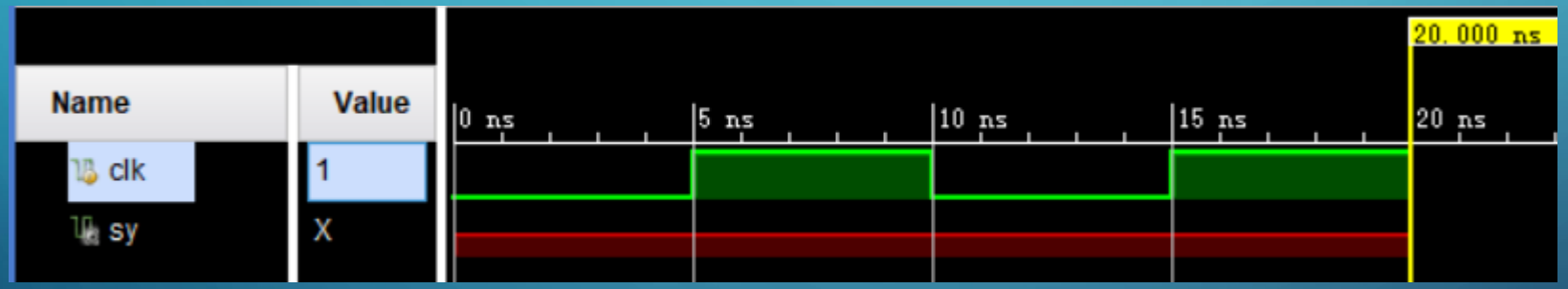
这个代码有什么问题

尝试修改这个代码的问题

INITIAL STATE(2)

```
module stateTest(
input clk,output reg y);
always @(posedge clk)
    y <= ~y;
endmodule
```

输出一直是不定态！不符合设计预期




```
module shiftOpSim( );
reg clk;
wire sy;
stateTest u1(.clk(clk),.y(sy));
initial begin
clk = 1'b0;
#20 $finish();
end
initial
forever #5 clk = !clk;
endmodule
```

INITIAL STATE(3)

同步

```
module stateTest(input
clk,output reg y);
always @(posedge clk)
    y <= ~y;
endmodule
```

异步

```
module stateTest(input clk,output reg y);
always @(posedge clk)
    y <= ~y;
endmodule
initial y<=0;
```

尝试多种解决方案，哪种合适？

```
module stateTest(input clk,output reg y);
always @(posedge clk) begin
    y <=0;
    y <= ~y;
end
endmodule
```

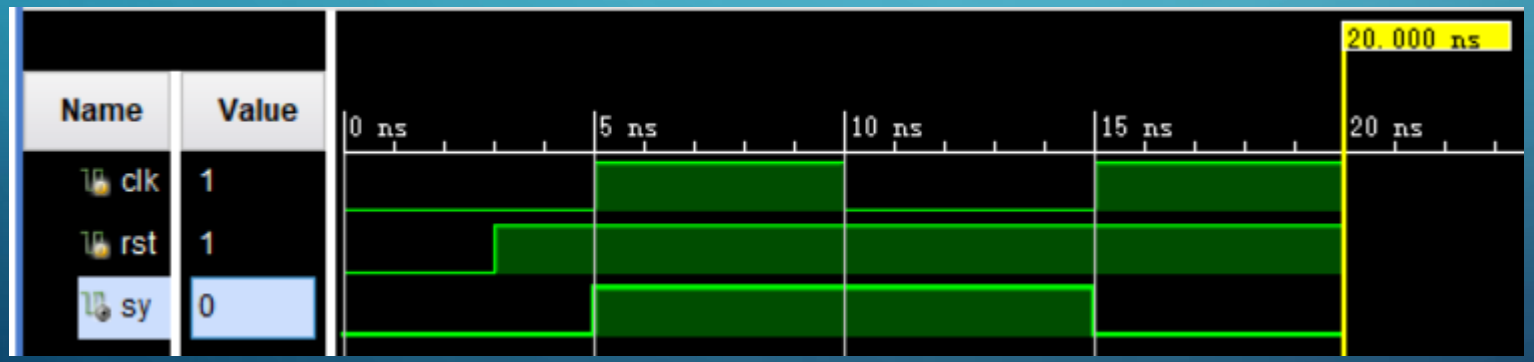
```
module stateTest(input clk,output reg y);
reg y=0;
always @(posedge clk) begin
    y <= ~y;
end
endmodule
```

```
module stateTest(input clk,rst, output reg y);
always @(posedge clk) begin
   if(rst) y<=0;
   else y <= ~y;
end
endmodule
```

RESET(1)

```
module stateTest(input
clk,output reg y);
always @(posedge clk)
    y <= ~y;
endmodule
```

需要增加一个复位信号
根据右侧的testbench和下方的波形图，
请问应该增加一个什么样的复位信号：
高电平有效？低电平有效？
同步复位？异步复位？




```
module shiftOpSim( );
reg clk;
wire sy;
stateTest u1(.clk(clk),.y(sy));
initial fork
        clk = 1'b0;
        rst = 1’b0;
        #3 rst = 1’b1;
        #20 $finish();
join
initial
        forever #5 clk = !clk;
endmodule
```

RESET(2)

```
module shiftOp(input clk,rst, output reg y);
always @(posedge clk) begin
   if(rst) y<=0;
   else y <= ~y;
end
endmodule
```

```
module shiftOp(input clk,rst, output reg y);
always @(posedge clk, posedge rst) begin
   if(rst) y<=0;
   else y <= ~y;
end
endmodule
```

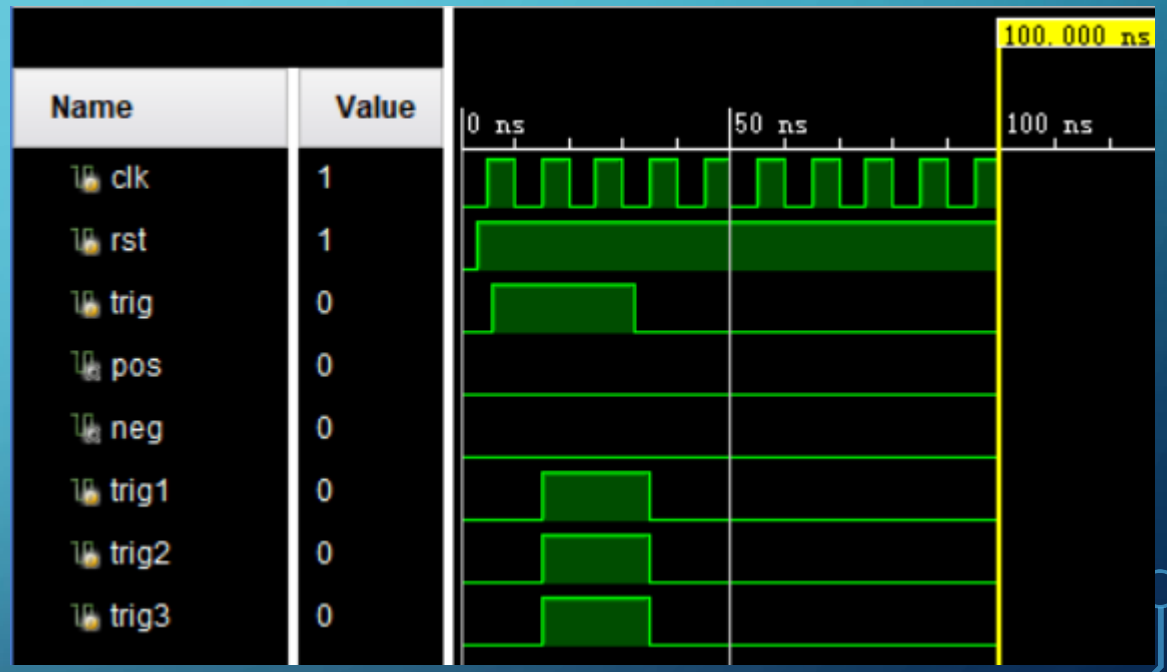
```
module shiftOp(input clk,rst, output reg y);
always @(posedge clk) begin
   if(!rst) y<=0;
   else y <= ~y;
end
endmodule
```

```
module shiftOp(input clk,rst, output reg y);
always @(posedge clk, negege rst) begin
   if(!rst) y<=0;
   else y <= ~y;
end
endmodule
```

```
module shiftOpSim( );
reg clk;
wire sy;
shiftOp u1(.clk(clk),.y(sy));
initial fork
clk = 1'b0;
rst = 1’b0;
#3 rst = 1’b1;
#20 $finish();
join
initial
forever #5 clk = !clk;
endmodule
```

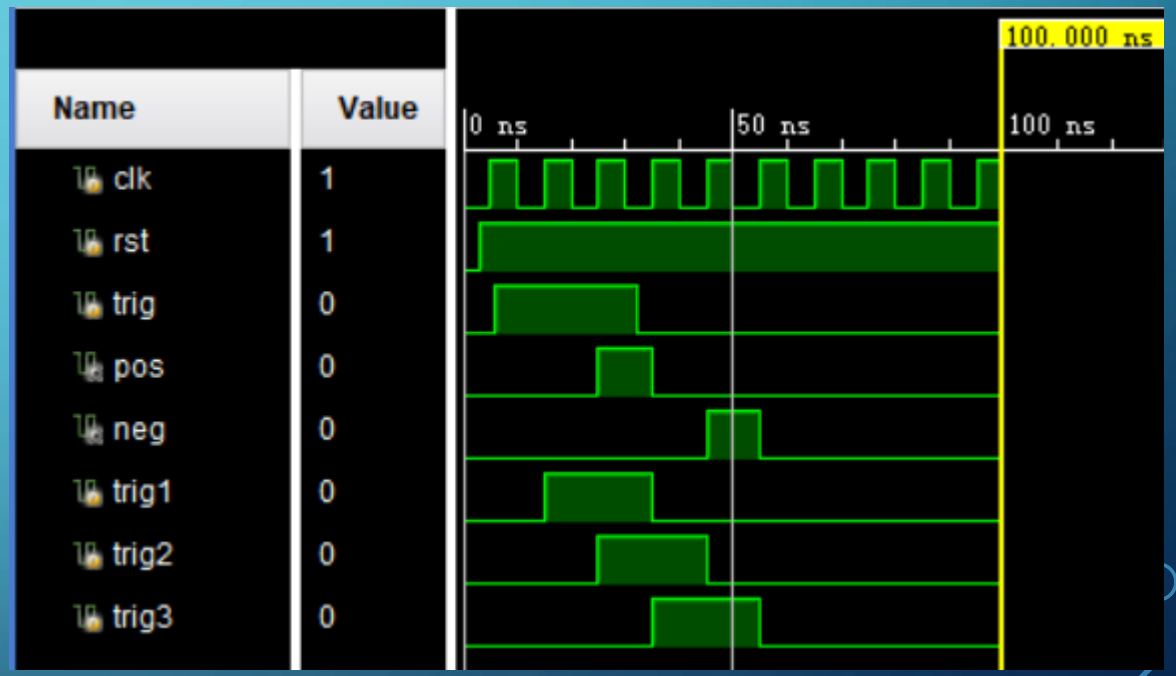
BLOCKING VS NON-BLOCKING ASSIGNMENT

```
module trigTest(input clk,rst,trig,output pos,neg);
reg trig1,trig2,trig3;
always @(posedge clk, negedge rst)
   if(!rst)
      {trig1,trig2,trig3} = 3'b000;
   else begin
      trig1 = trig;
      trig2 = trig1;
      trig3 = trig2;
   end
assign pos = (~trig3) & trig2;
assign neg = trig3 & (~trig2);
endmodule
```



BLOCKING VS NON-BLOCKING ASSIGNMENT

```
module trigTest(input clk,rst,trig, output pos,neg);
reg trig1,trig2,trig3;
always @(posedge clk, negedge rst)
   if(!rst)
      {trig1,trig2,trig3} <= 3'b000;
   else begin
      trig1 <= trig;
      trig2 <= trig1;
      trig3 <= trig2;
   end
assign pos = (~trig3) & trig2;
assign neg = trig3 & (~trig2);
endmodule
```



BLOCKING VS NON-BLOCKING ASSIGNMENT

```
module trigTest(input clk,rst,trig,output pos,neg);
reg trig1,trig2,trig3;
always @(posedge clk, negedge rst)
   if(!rst)
      {trig1,trig2,trig3} <= 3'b000;
   else begin
      trig3 <= trig2;
      trig2 <= trig1;
      trig1 <= trig;
   end
assign pos = ~trig3&trig2;
assign neg = trig3&(!trig2);
endmodule
```

```
module trigTest(input clk,rst,trig,output pos,neg);
reg trig1,trig2,trig3;
always @(posedge clk, negedge rst)
   if(!rst)
      {trig1,trig2,trig3} = 3'b000;
   else begin
      trig3 = trig2;
      trig2 = trig1;
      trig1 = trig;
   end
assign pos = ~trig3&trig2;
assign neg = trig3&(!trig2);
endmodule
```


请问，调换了 else分支中的trig1，trig2，trig3的次序后是否会影响执行结果？

BLOCKING VS NON-BLOCKING ASSIGNMENT

```
module trigTest(input clk,rst,trig, output pos,neg);
reg trig1,trig2,trig3;
always @(posedge clk, negedge rst)
   if(!rst)
      trig1<= 1'b0;
   else begin
      trig1 <= trig;
   end
```

```
always @(posedge clk, negedge rst)
   if(!rst)
      trig2<= 1'b0;
   else begin
      trig2 <= trig1;
   end
```

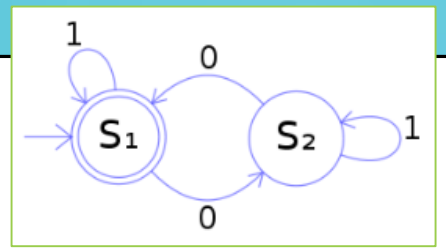
```
always @(posedge clk, negedge
rst)
   if(!rst)
      trig3<= 1'b0;
   else begin
      trig3 <= trig2;
   end
```

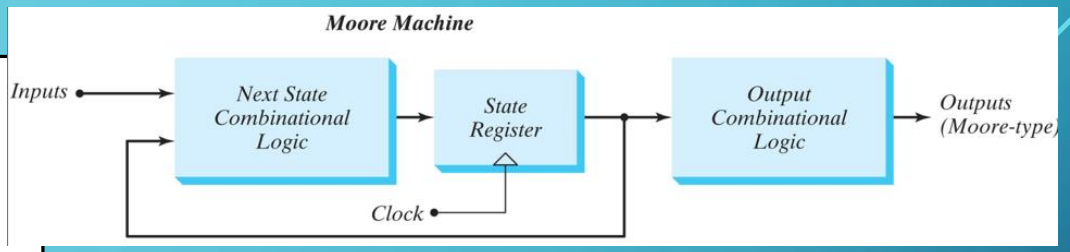
```
assign pos = (~trig3) & trig2;
assign neg = trig3 & (~trig2);
endmodule
```

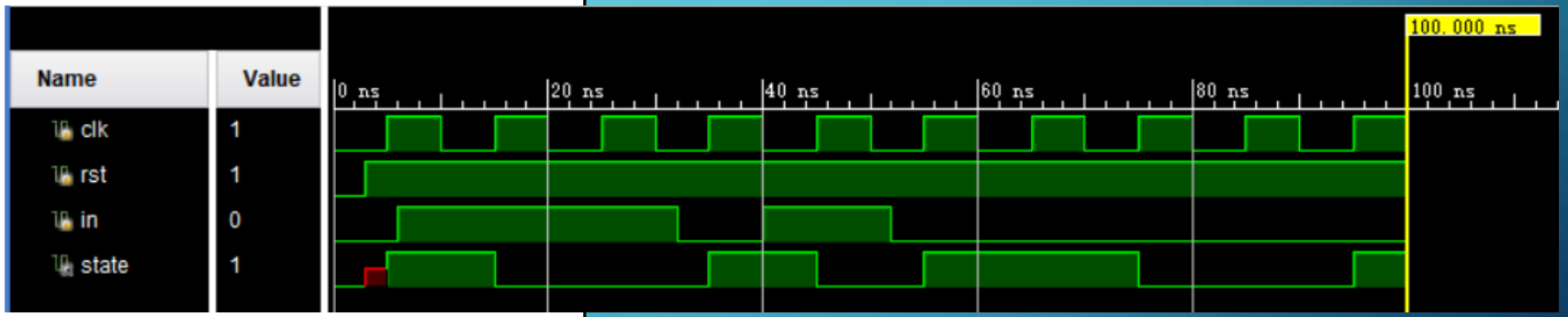
推荐的做法：
在一个always里只对一个信号赋值
如果要对多个信号赋值，为每个信号分派一个独立的always语句块

MIXED SENSITIVE LIST- NOT SUGGESTED !!

```
module fsmTest(input clk,rst,in,output reg state);
reg next_state;
parameter s1=1'b0,s2=1'b1;
always @(posedge clk, rst, in)
   if(!rst)
      state <= s1;
   else begin
      state <= next_state;
      case(state)
            s1: if(in) next_state = s1; else next_state = s2;
            s2: if(in) next_state = s2; else next_state = s1;
        endcase
   end
endmodule
```



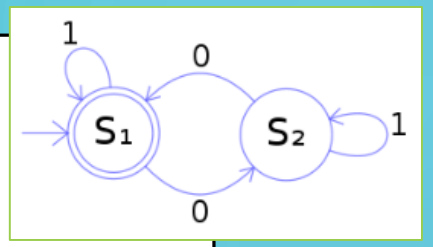


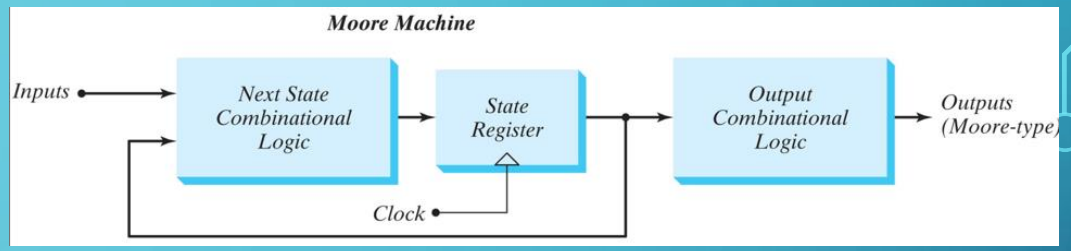


这种做法不推荐，原因
1. 敏感列表边沿信号和电平信号混杂
2. 同一个always中阻塞赋值和非阻塞赋值混杂

DEVIDE THE MIXED SENSITIVE LIST

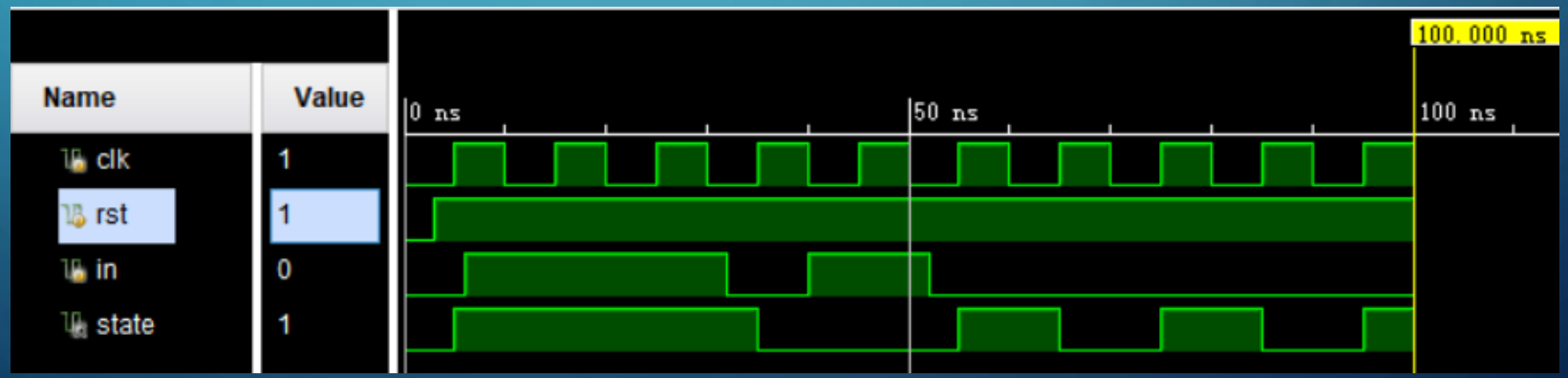
```
module trigTest(input clk,rst,in,output state);
reg next_state;
parameter s1=1'b0,s2=1'b1;
always @(posedge clk, negedge rst)
   if(!rst)
      state <= s1;
   else
      state <= next_state;
```



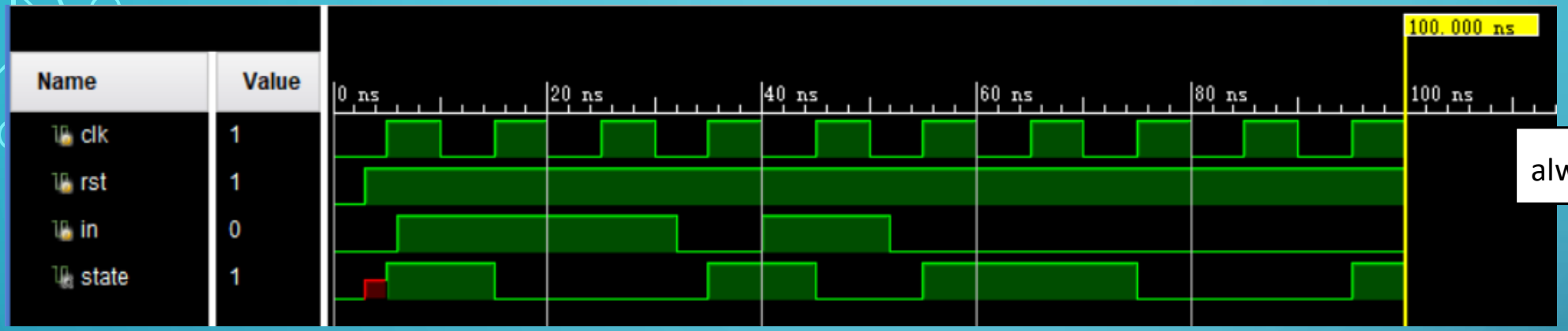


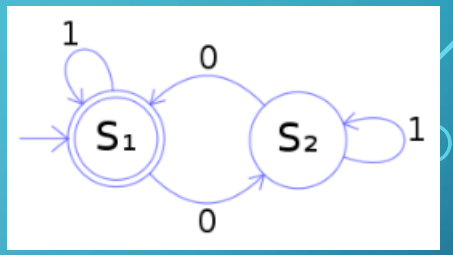

```
always@(*)
      case(state)
            s1: if(in) next_state = s1; else next_state = s2;
            s2: if(in) next_state = s2; else next_state = s1;
      endcase
```

正确的做法：应该将组合逻辑与时序逻辑分开实现



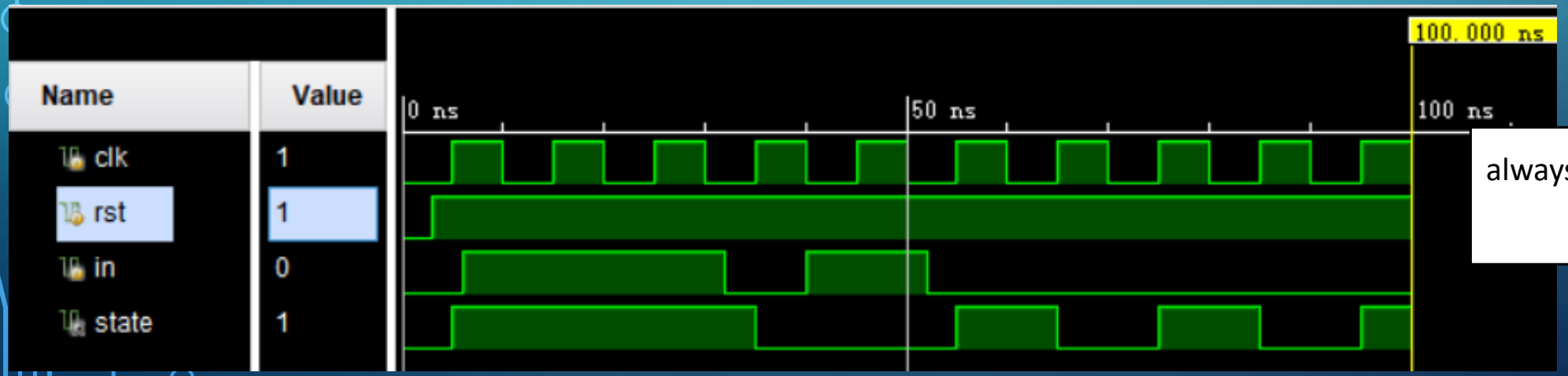
WHY NO SUGGEST THE MIXED SENSITIVE LIST





always @(posedge clk, rst, in)

下方的做法才能实现设计预期



always @(posedge clk, negedge rst)

always @(*)

MULTIPY DRIVER

- 模块例化和端口绑定，明确数据流，必要的时候增加多路选择器

如何避免MULTIPY-DRIVER

- 一个信号在多个模块中做赋值，如何避免multipy-driver

```
always@ *
   if(条件b)
      a = yyy;

always@ *
   if（条件c）
      a = xxx;
```

```
always@ *
   if（条件c）
      t条件c = ccc;

always@ *
   if（条件b）
      t条件b = bbb;
```

```
always@ *
   case（{t条件b，t条件c}）
      bbbccc: a = xxx;
      ....
      endcase
```

LOGIC SHIFT VS ARITHMETIC SHIFT(1)

```
module shiftTest( );
reg [3:0] y;
initial begin
    #1 y= 4'b1001 <<3;      //
    #1 y= 4'b1001 <<<3;     //
    #1 y= 4'b1001 >>3;      //
    #1 y= 4'b1001 >>>3;     //
    #1 y= 4'sb1001 >>3;     //
    #1 y= 4'sb1001 >>>3;    //
    $finish();
end
endmodule
```

operator	<<	>>	<<<	>>>
operation	Logic Shift Left	Logic Shift Right	Arithmetic Shift Left	Arithmetic Shift Right

4’b1001 : unsigned int
4’sb1001 : signed int

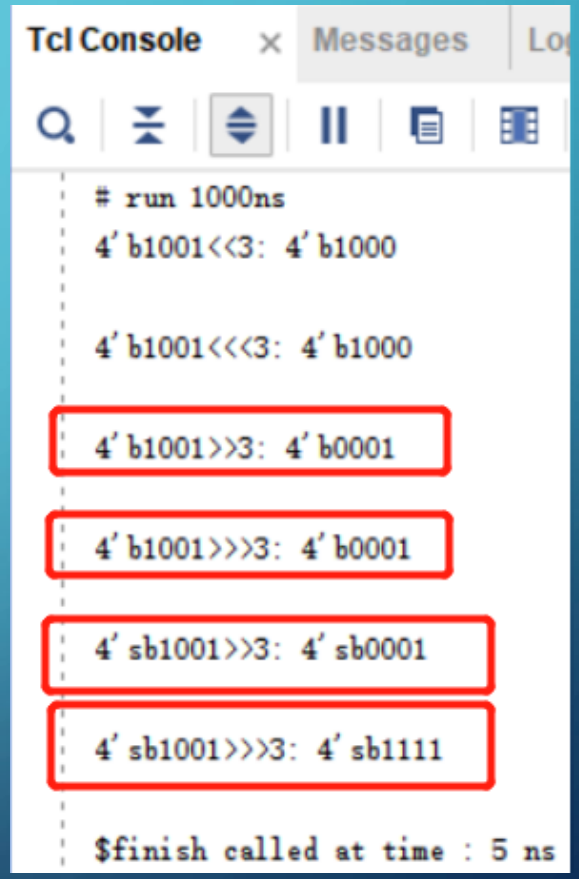
Q1. 请问对 4‘b1001 进行逻辑左移和算数左移时有什么样的差异？

Q2. 请问对 4‘sb1001 进行逻辑右移和算数右移时有什么样的差异？

LOGIC SHIFT VS ARITHMETIC SHIFT(2)

```
module shiftTest( );
reg [3:0] y;
initial begin
            y= 4'b1001 <<3;      $display("4'b1001<<3: 4'b%4b\n",y);
   #1       y= 4'b1001 <<<3;     $display("4'b1001<<<3: 4'b%4b\n",y);
   #1       y= 4'b1001 >>3;      $display("4'b1001>>3: 4'b%4b\n",y);
   #1       y= 4'b1001 >>>3;     $display("4'b1001>>>3: 4'b%4b\n",y);
   #1       y= 4'sb1001 >>3;     $display("4'sb1001>>3: 4'sb%4b\n",y);
   #1       y= 4'sb1001 >>>3;    $display("4'sb1001>>>3: 4'sb%4b\n",y);
   $finish();
end
endmodule
```

有符号数 算术右移 用原始数最高位填充



开发板的相关问题

- 1. 以下哪个文件修改了需要重新生成bitstream文件
  - 项目文件（芯片类型做了修改）、设计文件、仿真文件、约束文件
- 2. 关闭开发板的次序是
- tcl窗口中执行 disconnect_hw_server，关闭硬件管理器，关闭开发板电源，断开物理连接
- 3. 如果报xx错误，应该如何处理（修改约束文件只是权宜之计
- 4. bitstream文件更新后必须重新烧写到开发板的fpaga芯片才能生效？ （是）
- 5. 什么情况下烧写到fpga芯片上的bitstream文件会失效：关闭vivado工程（no），关闭开发板电源（yes），断开开发板与pc的连接（yes）
- 6. 如果要查看顶层模块的内部结构，可以通过什么方式查看：1）rtl分析 yes 2）综合 yes 3）实现 no
- 7. vivado 中的默认仿真时间不可以做修改 （false）
- 8. vivado 工程的综合、实现、生成bitstream文件都是基于当前的top设计模块以及active的约束集中的约束文件（true）