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

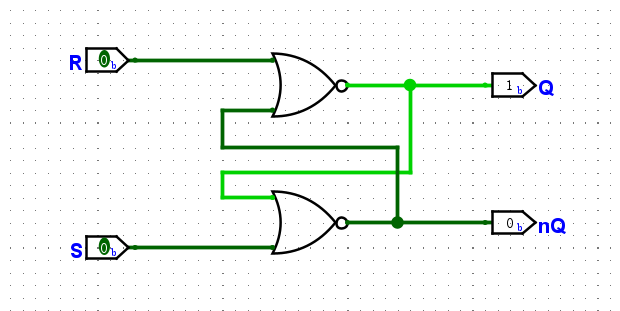
**三、实验要求**

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**

SR锁存器：图形用户界面, 图示, 应用程序, 表格, Excel

描述已自动生成

SR锁存器是一种基本的数字电路，用于存储一个比特的数据。它由两个输入端（S和R）和两个输出端（Q和Q'）组成，其中Q和Q'是互补输出。SR锁存器的原理是通过控制输入信号S和R的状态来控制输出信号Q和Q'的状态。

SR锁存器的逻辑表达式如下：

Q = S'Q' + R'Q

Q' = RQ + S'Q'

其中，S和R分别代表设置和重置输入信号，Q和Q'分别代表输出信号。SR锁存器可以通过两个NOR门实现。

SR锁存器的工作原理如下：

当S和R均为0时，SR锁存器处于保持状态，输出Q和Q'保持不变。

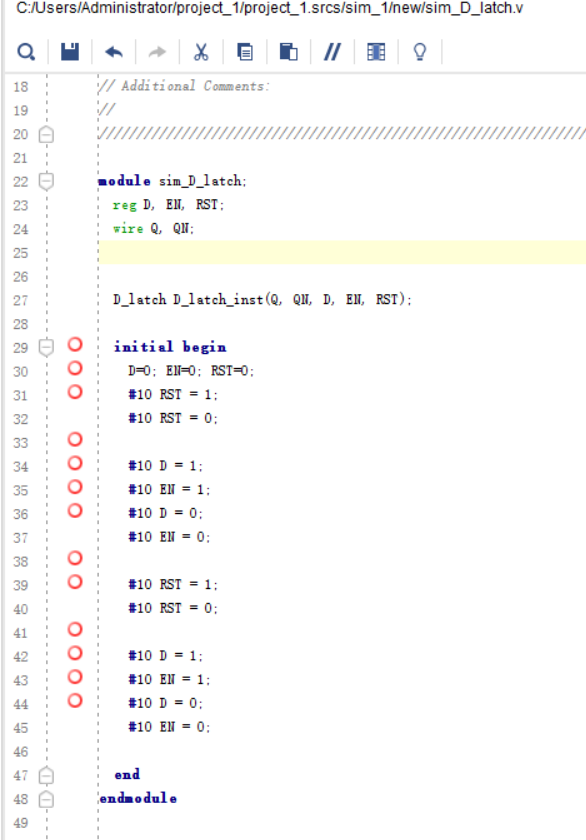
当S为1，R为0时，SR锁存器处于设置状态，输出Q为1，Q'为0。

当S为0，R为1时，SR锁存器处于重置状态，输出Q为0，Q'为1。

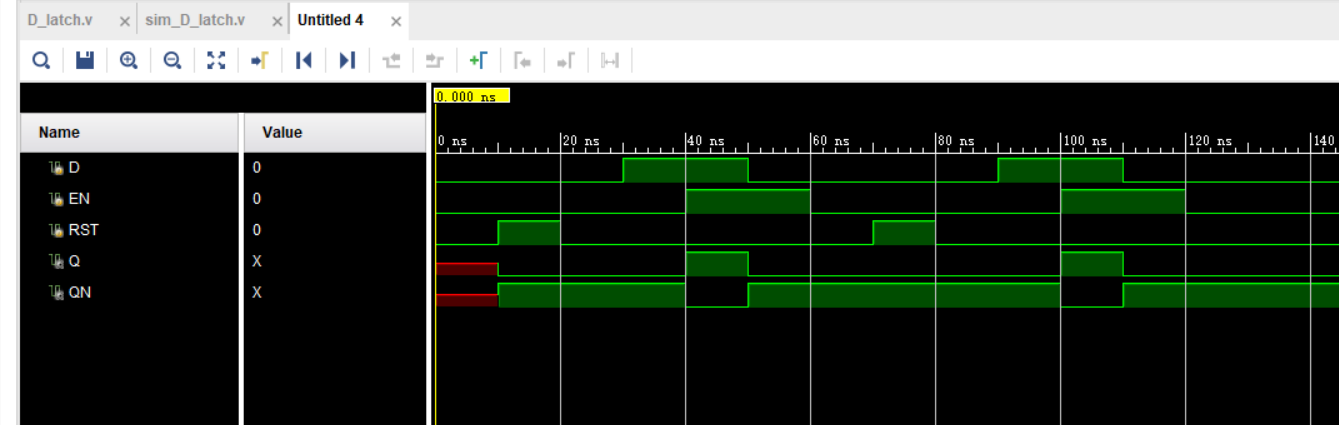
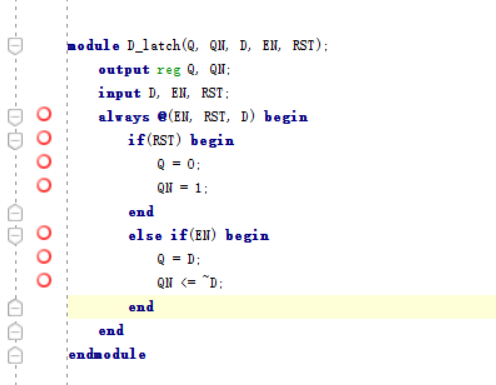
当S和R均为1时，SR锁存器处于不稳定状态，输出Q和Q'的状态无法确定。

总之，SR锁存器是一种基本的数字电路，用于存储一个比特的数据。它的原理是通过控制输入信号S和R的状态来控制输出信号Q和Q'的状态。

如上图，SR锁存器可以通过两个NOR门来实现，真值表和时序图印证了上述原理逻辑。



锁存器：



D锁存器是一种基本的数字电路，用于存储一个比特的数据。它由一个输入端（D）和两个输出端（Q和QN）组成，其中Q和QN是互补输出。

程序的含义如下：

module D\_latch(Q, QN, D, EN, RST);：定义了一个名为D\_latch的模块，该模块有四个输入输出端口，分别是Q、QN、D、EN和RST。

output reg Q, QN;：定义了两个输出信号Q和QN，使用reg类型表示，表示可以在always块中进行赋值。

input D, EN, RST;：定义了三个输入信号D、EN和RST。

always @(EN, RST, D) begin：always块表示在输入信号EN、RST和D变化时执行以下逻辑。

if(RST) begin：如果RST信号为1，表示复位状态，则执行以下逻辑。

Q = 0;：将输出信号Q设置为0。

QN = 1;：将输出信号QN设置为1。

else if(EN) begin：如果EN信号为1，表示使能状态，则执行以下逻辑。

Q = D;：将输出信号Q设置为输入信号D的值。

QN <= 'D;：将输出信号QN设置为输入信号D的互补值。

以上代码实现了D锁存器的功能。当输入信号RST为1时，D锁存器处于复位状态，输出信号Q为0，输出信号QN为1。当输入信号EN为1时，D锁存器处于使能状态，输出信号Q为输入信号D的值，输出信号QN为输入信号D的互补值。在其他情况下，D锁存器保持上一状态不变。

module sim\_D\_latch;：定义了一个名为sim\_D\_latch的模块。

reg D, EN, RST;：定义了三个寄存器类型的变量D、EN和RST，用于模拟输入信号。

wire Q, QN;：定义了两个线类型的变量Q和QN，用于模拟输出信号。

D\_latch D\_latch\_inst(Q, QN, D, EN, RST);：实例化了一个D\_latch模块，将输入输出信号连接到对应的变量。

initial begin：initial块表示仿真开始时执行以下逻辑。

#10 RST = 1;：在仿真时间10个时间单位后，将RST信号置为1，表示进行复位。

#10 RST = 0;：在仿真时间10个时间单位后，将RST信号置为0，表示结束复位。

#10 D = 1;：在仿真时间10个时间单位后，将输入信号D置为1。

#10 EN = 1;：在仿真时间10个时间单位后，将输入信号EN置为1。

#10 D = 0;：在仿真时间10个时间单位后，将输入信号D置为0。

#10 EN = 0;：在仿真时间10个时间单位后，将输入信号EN置为0。

#10 RST = 1;：在仿真时间10个时间单位后，将RST信号置为1，再次进行复位。

#10 RST = 0;：在仿真时间10个时间单位后，将RST信号置为0，表示结束复位。

#10 D = 1;：在仿真时间10个时间单位后，将输入信号D置为1。

#10 EN = 1;：在仿真时间10个时间单位后，将输入信号EN置为1。

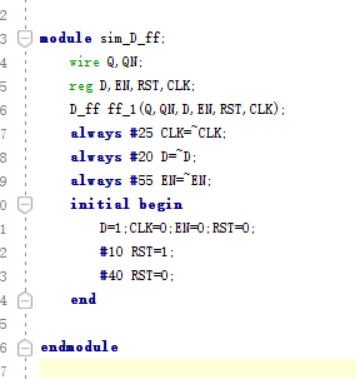
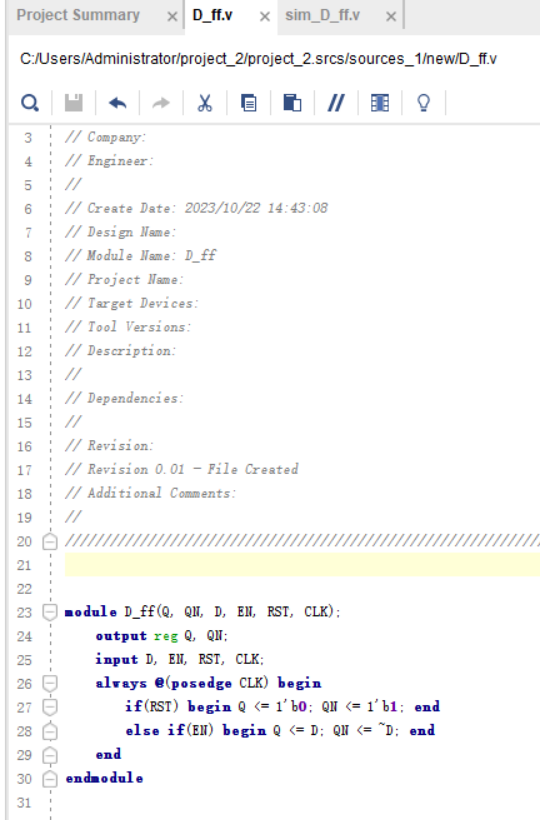
#10 D = 0;：在仿真时间10个时间单位后，将输入信号D置为0。

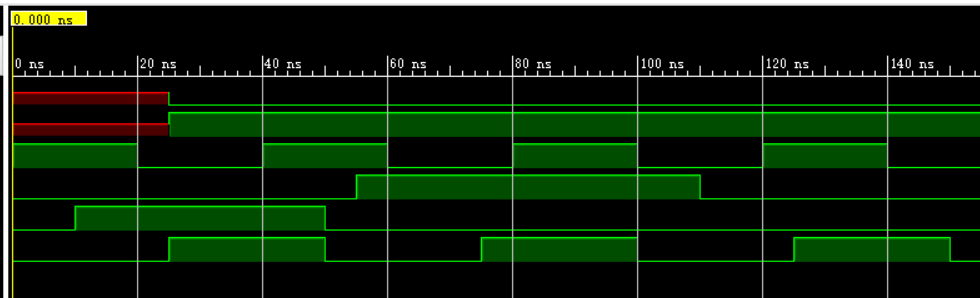
#10 EN = 0;：在仿真时间10个时间单位后，将输入信号EN置为0。

以上仿真程序通过改变输入信号的值和时间来测试D锁存器的功能。

通过时序图可以看出D，EN，RST三个信号的变化随着程序的控制进行。导致了Q和QN的变化。时序图验证了D锁存器的以下功能：当输入信号RST为1时，D锁存器处于复位状态，输出信号Q为0，输出信号QN为1。当输入信号EN为1时，D锁存器处于使能状态，输出信号Q为输入信号D的值，输出信号QN为输入信号D的互补值。在其他情况下，D锁存器保持上一状态不变。

触发器：





定义了一个D触发器模块（D\_ff），它具有以下输入和输出：

output reg Q, QN;：输出信号Q和QN，使用寄存器类型（reg）声明，表示D触发器的输出。

input D, EN, RST, CLK;：输入信号D、使能信号EN、复位信号RST和时钟信号CLK，表示D触发器的输入。

always @(posedge CLK) begin：使用always块来定义时钟边沿触发的行为。

if(RST) begin Q <= 1'b0; QN <= 1'b1; end：如果复位信号RST为1，则将输出信号Q设置为0，输出信号QN设置为1。

else if(EN) begin Q <= D; QN <= ~D; end：否则，如果使能信号EN为1，则将输出信号Q设置为输入信号D的值，输出信号QN设置为输入信号D的互补值。

以上程序实现了D触发器的功能。在每个时钟上升沿时，根据输入信号的状态，更新输出信号的值。当复位信号RST为1时，D触发器处于复位状态，输出信号Q为0，输出信号QN为1。当使能信号EN为1时，D触发器处于使能状态，输出信号Q为输入信号D的值，输出信号QN为输入信号D的互补值。在其他情况下，D触发器保持上一状态不变。

该模块包含了一个D触发器（D\_ff）的实例化。

wire Q, QN;：定义了两个输出信号Q和QN，使用wire类型声明。

reg D, EN, RST, CLK;：定义了四个输入信号D、EN、RST和CLK，使用reg类型声明。

D\_ff ff\_1(Q, QN, D, EN, RST, CLK);：实例化了一个D触发器（D\_ff）模块，将输入输出信号连接到实例化模块的对应端口。

always #25 CLK = ~CLK;：使用always块来定义一个时钟信号的周期性翻转。每隔25个时间单位，时钟信号CLK的值取反。

always #20 D = ~D;：使用always块来定义一个输入信号D的周期性翻转。每隔20个时间单位，输入信号D的值取反。

always #55 EN = ~EN;：使用always块来定义一个输入信号EN的周期性翻转。每隔55个时间单位，输入信号EN的值取反。

initial begin：initial块表示仿真开始时执行以下逻辑。

D = 1; CLK = 0; EN = 0; RST = 0;：在仿真开始时，将输入信号D置为1，时钟信号CLK置为0，使能信号EN置为0，复位信号RST置为0。

#10 RST = 1;：在仿真时间10个时间单位后，将复位信号RST置为1，表示进行复位操作。

#40 RST = 0;：在仿真时间40个时间单位后，将复位信号RST置为0，表示结束复位操作。

以上仿真文件通过改变输入信号的值和时间来测试D触发器的功能。通过观察输出信号Q和QN的变化，可以验证D触发器是否按照预期工作。

时序图分析：

在仿真开始时，D触发器处于复位状态，输出信号Q为0，输出信号QN为1。

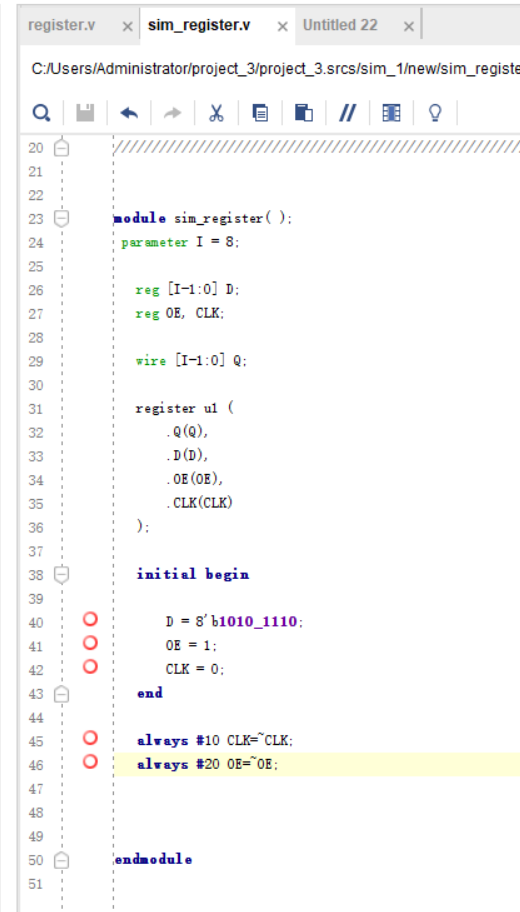
在仿真时间10个时间单位后，复位信号RST被置为1，D触发器进入复位状态，输出信号Q为0，输出信号QN为1。

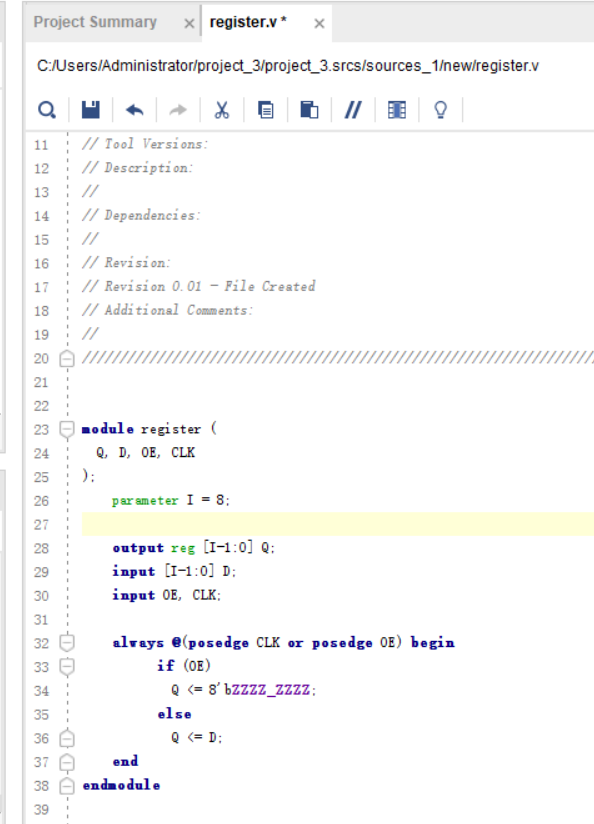
在仿真时间40个时间单位后，复位信号RST被置为0，D触发器退出复位状态。

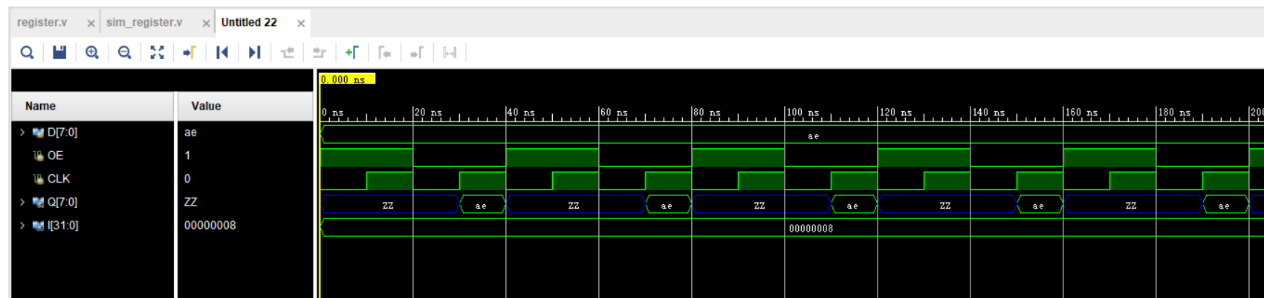
在仿真时间55个时间单位后，使能信号EN被置为1，D触发器进入使能状态，根据输入信号D的值，输出信号Q和QN会相应改变。

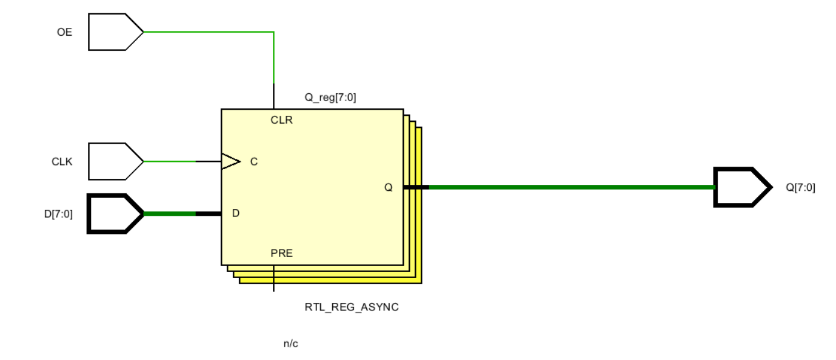
在仿真时间75个时间单位后，输入信号D被翻转，D触发器在下一个时钟上升沿时根据新的输入信号D的值更新输出信号Q和QN。

在仿真时间80个时间单位后，时钟信号CLK被翻转，D触发器在下一个时钟上升沿时根据当前输入信号D的值更新输出信号Q和QN。

****寄存器：

****

****

****

定义了一个名为register的模块，该模块实现了一个带有输出使能的寄存器。

parameter I = 8;：定义了一个参数I，表示寄存器的位宽，默认为8。

output reg [I-1:0] Q;：定义了一个输出信号Q，使用reg类型声明，位宽为I-1到0。

input [I-1:0] D;：定义了一个输入信号D，位宽为I-1到0。

input OE, CLK;：定义了两个输入信号OE和CLK，分别表示输出使能和时钟。

always @(posedge CLK or posedge OE) begin：使用always块来定义一个时钟上升沿或输出使能上升沿触发的逻辑。

if (OE)：如果输出使能信号OE为1，执行以下逻辑。

Q <= 8'bZZZZ\_ZZZZ;：将输出信号Q的值置为8个高阻态（Z）。

else：否则，执行以下逻辑。

Q <= D;：将输出信号Q的值赋为输入信号D的值。

以上程序实现了一个带有输出使能的寄存器。当输出使能信号OE为1时，输出信号Q的值被置为高阻态（Z），表示输出无效。当输出使能信号OE为0时，输出信号Q的值被赋为输入信号D的值，表示输出有效。在时钟上升沿或输出使能上升沿触发时，根据输出使能信号OE的值来更新输出信号Q的值。

定义了一个名为sim\_register的模块，该模块用于对寄存器进行仿真。

parameter I = 8;：定义了一个参数I，表示寄存器的位宽，默认为8。

reg [I-1:0] D;：定义了一个寄存器的输入信号D，位宽为I-1到0。

reg OE, CLK;：定义了两个寄存器的输入信号OE和CLK，分别表示输出使能和时钟。

wire [I-1:0] Q;：定义了一个寄存器的输出信号Q，位宽为I-1到0。

register u1 (.Q(Q), .D(D), .OE(OE), .CLK(CLK));：实例化了一个名为u1的寄存器模块，并将输入输出信号连接到模块的对应端口。

initial begin：使用initial块来初始化寄存器的输入信号。

D = 8'b1010\_1110;：将输入信号D的值初始化为8位二进制数1010\_1110。

OE = 1;：将输出使能信号OE的值初始化为1。

CLK = 0;：将时钟信号CLK的值初始化为0。

always #10 CLK=~CLK;：使用always块来定义一个每隔10个时间单位翻转一次时钟信号CLK的逻辑。

always #20 0E=~OE;：使用always块来定义一个每隔20个时间单位翻转一次输出使能信号OE的逻辑。

以上仿真程序实例化了一个寄存器模块，并通过初始化输入信号的值和周期性翻转时钟和输出使能信号的值来对寄存器进行仿真。

时序图：

初始状态下，输入信号D为8'b1010\_1110，输出使能信号OE为1，时钟信号CLK为0。

经过10个时间单位后，时钟信号CLK翻转为1。

在时钟信号上升沿触发时，寄存器模块会根据输入信号D和输出使能信号OE的值来更新输出信号Q的值。由于输出使能信号OE为1，输出信号Q应该被置为高阻态（Z）。

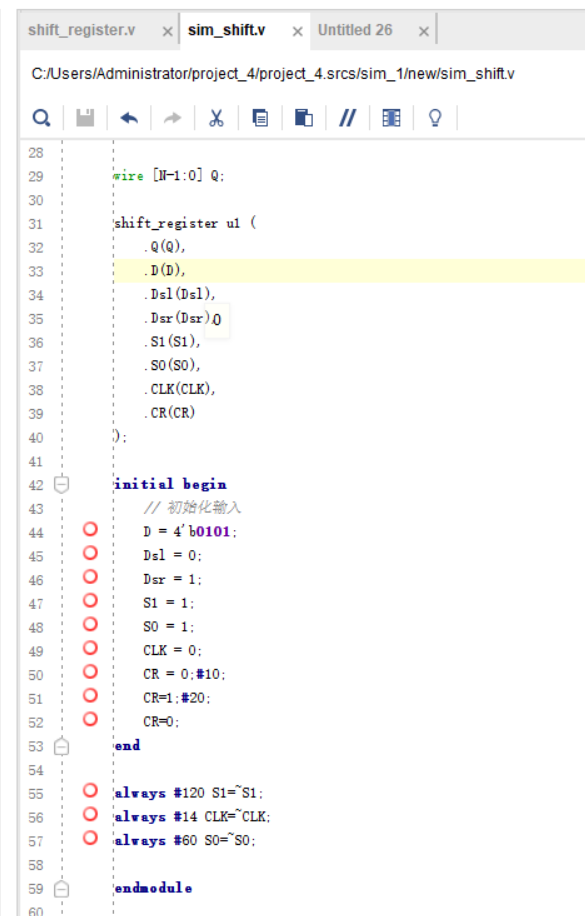
经过20个时间单位后，输出使能信号OE翻转为0。

在输出使能信号OE为0时，寄存器模块会根据输入信号D的值来更新输出信号Q的值。根据输入信号D的值为8'b1010\_1110，输出信号Q应该被赋为8'b1010\_1110。

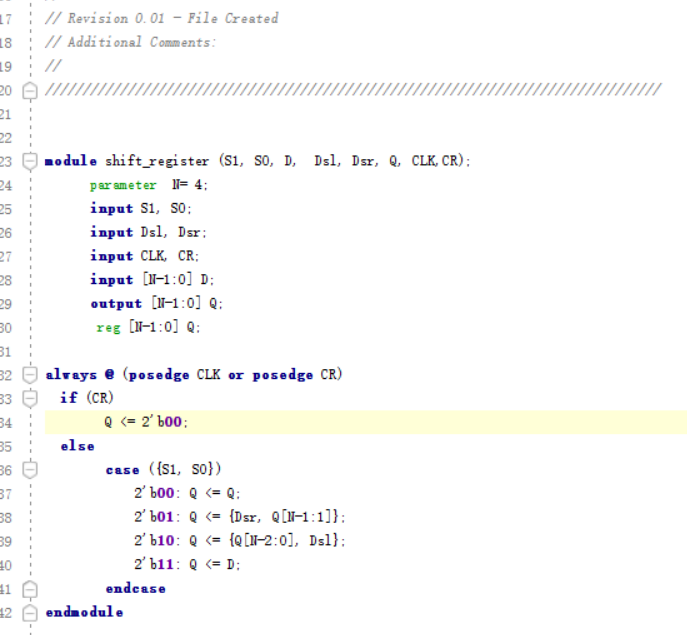
经过10个时间单位后，时钟信号CLK翻转为0。

经过10个时间单位后，时钟信号CLK翻转为1。

在时钟信号上升沿触发时，寄存器模块会根据输入信号D和输出使能信号OE的值来更新输出信号Q的值。由于输出使能信号OE为0，输出信号Q应该保持为上一次更新的值，即8'b1010\_1110。

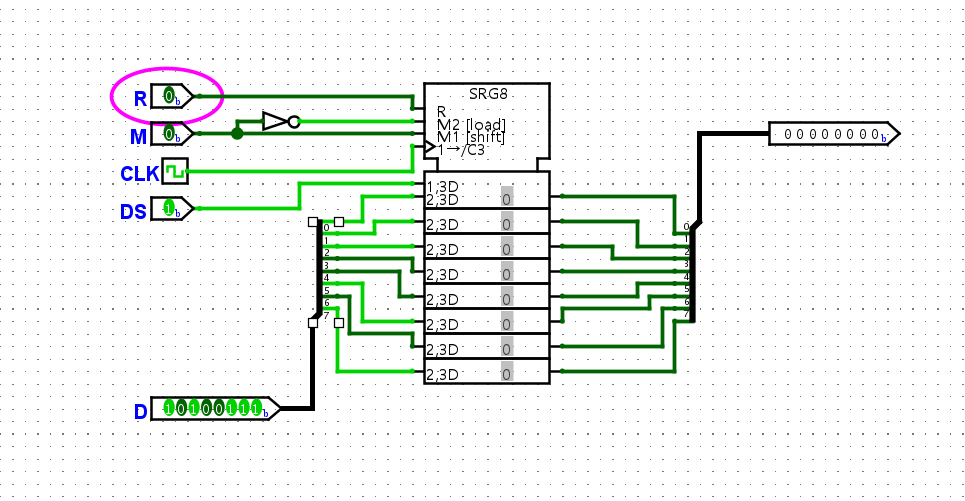


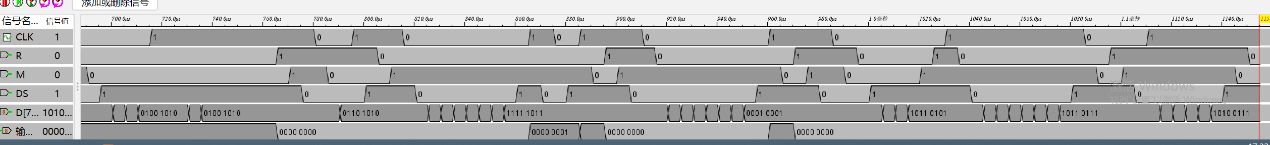
移位寄存器:



电脑萤幕画面

中度可信度描述已自动生成





一个带有异步清零功能的移位寄存器模块。下面是对程序的详细分析：

module shift\_register (S1, S0, D, Dsl, Dsr, Q, CLK,CR);：定义了一个名为shift\_register的模块，该模块具有输入信号S1、S0、D、Dsl、Dsr、CLK和CR，以及输出信号Q。

parameter N= 4;：定义了一个参数N，表示寄存器的位宽，默认为4。

input S1, S0;：定义了两个输入信号S1和S0，用于选择移位寄存器的操作模式。

input Dsl, Dsr;：定义了两个输入信号Dsl和Dsr，用于输入数据时的移位方向。

input CLK, CR;：定义了两个输入信号CLK和CR，分别表示时钟和异步清零信号。

input [N-1:0] D;：定义了一个输入信号D，用于输入数据。

output [N-1:0] Q;：定义了一个输出信号Q，表示寄存器的输出数据。

reg [N-1:0] Q;：定义了一个寄存器的内部变量Q，用于存储寄存器的数据。

always @ (posedge CLK or posedge CR)：使用always块来定义一个时钟触发的逻辑，当时钟上升沿或异步清零信号CR上升沿时，执行下面的逻辑。

if (CR)：如果异步清零信号CR为1，则执行下面的逻辑。

Q <= 2'b00;：将寄存器的数据Q置为2'b00，即异步清零。

else：如果异步清零信号CR为0，则执行下面的逻辑。

case ({S1, S0})：根据输入信号S1和S0的值进行选择。

2'b00: Q <= Q;：当S1和S0为2'b00时，寄存器的数据Q保持不变。

2'b01: Q <= {Dsr, Q[N-1:1]};：当S1和S0为2'b01时，将输入信号Dsr和寄存器的数据Q[N-1:1]连接起来，作为新的寄存器数据Q。

2'b10: Q <= {Q[N-2:0], Dsl};：当S1和S0为2'b10时，将寄存器的数据Q[N-2:0]和输入信号Dsl连接起来，作为新的寄存器数据Q。

2'b11: Q <= D;：当S1和S0为2'b11时，将输入信号D作为新的寄存器数据Q。

module sim\_shift;：定义了一个名为sim\_shift的模块。

parameter N = 4;：定义了一个参数N，表示寄存器的位宽，默认为4。

reg [N-1:0] D;：定义了一个寄存器变量D，用于存储输入数据。

reg Dsl, Dsr, S1, S0, CLK, CR;：定义了一些寄存器变量，用于控制和输入。

wire [N-1:0] Q;：定义了一个连线Q，用于连接到移位寄存器的输出。

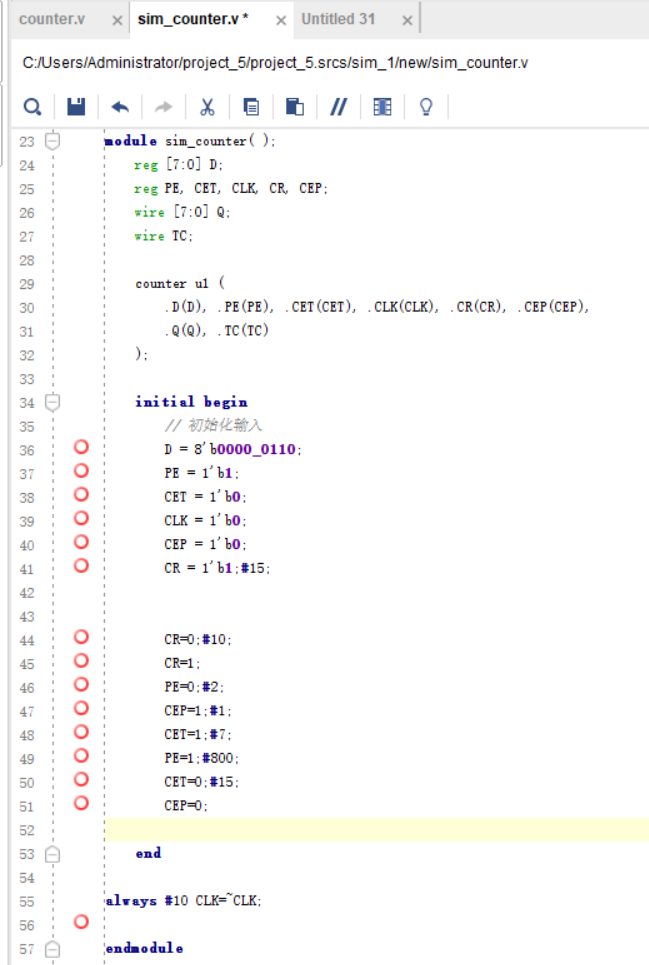
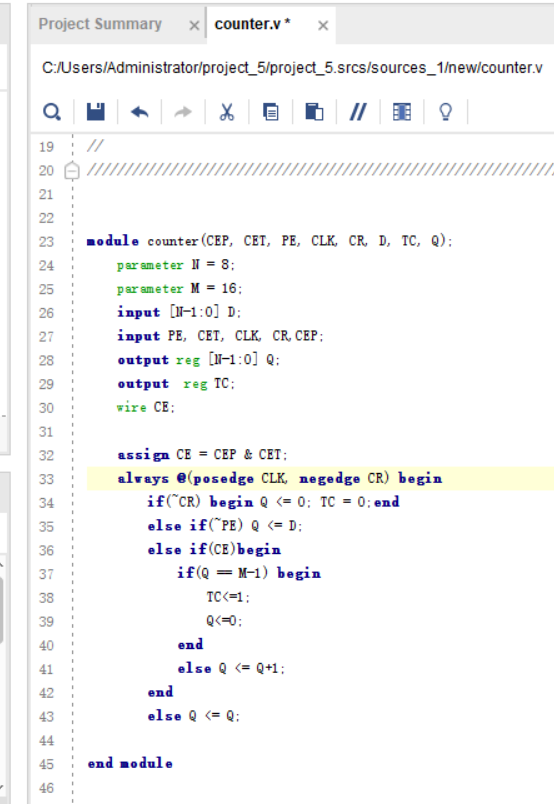
shift\_register u1 ( .Q(Q), .D(D), .Dsl(Dsl), .Dsr(Dsr), .S1(S1), .S0(S0), .CLK(CLK), .CR(CR) );：实例化了一个移位寄存器模块u1，并将输入和输出信号连接到模块的对应端口。

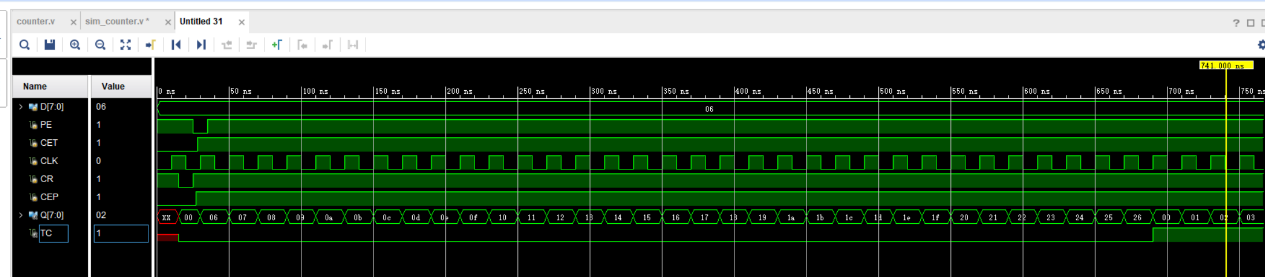
initial begin：初始化块，用于初始化输入信号。D = 4'b0101;：将输入数据D初始化为4'b0101。Dsl = 0;：将输入信号Dsl初始化为0。Dsr = 1;：将输入信号Dsr初始化为1。S1 = 1;：将输入信号S1初始化为1。S0 = 1;：将输入信号S0初始化为1。CLK = 0;：将输入信号CLK初始化为0。CR = 0; #10;：将输入信号CR初始化为0，并延迟10个时间单位。CR=1;#20;：将输入信号CR设置为1，并延迟20个时间单位。CR=0;：将输入信号CR设置为0。

always #120 S1=~S1;：每隔120个时间单位，取反一次S1的值。always #14 CLK=~CLK;：每隔14个时间单位，取反一次CLK的值。always #60 S0=~S0;：每隔60个时间单位，取反一次S0的值。

当CR信号为1时，移位寄存器的值将被异步清零，即Q的值被设置为2'b00。当CR信号为0时，根据S1和S0的值选择不同的操作模式：当S1和S0都为0时，Q的值保持不变。当S1为0，S0为1时，将Dsr的值作为最低位，将Q的高位向右移动一位，得到新的Q的值。当S1为1，S0为0时，将Dsl的值作为最高位，将Q的低位向左移动一位，得到新的Q的值。当S1和S0都为1时，将D的值直接赋给Q。

计数器：





一个16进制的8位计数器模块，具有异步清零功能和可编程上限值。它有以下输入和输出：

输入：

D：输入信号，用于在PE信号为1时，将D的值赋给计数器的当前值Q。

PE：计数器使能信号，当PE为1时，将D的值赋给Q。

CET：计数器计数使能信号，当CET和CEP同时为1时，计数器开始计数。

CEP：计数器计数使能信号，当CET和CEP同时为1时，计数器开始计数。

CLK：时钟信号，用于触发计数器的操作。

CR：异步清零信号，当CR为1时，计数器的值被清零。

输出：

TC：计数器的进位信号，当计数器从最大值M-1增加到0时，TC被置为1。

Q：计数器的输出信号，即当前计数器的值。

计数器的功能如下：

当CR信号为1时，计数器的值被清零，即Q的值被设置为0，TC被置为0。

当PE信号为1时，将D的值赋给Q，即计数器的值被设置为D的值。

当CET和CEP同时为1时，计数器开始计数。根据时钟信号的上升沿触发计数器的操作。

在计数器开始计数的情况下，当计数器的值Q等于M-1时，表示计数器已经达到了上限值，此时TC被置为1，Q被清零。

在计数器开始计数的情况下，当计数器的值Q小于M-1时，计数器的值Q被加1。

当PE、CET和CEP都为0时，计数器的值Q保持不变。

**将M改为39就变成了39进制的计数器：**

输入信号：D、PE、CET、CLK、CR、CEP

输出信号：Q、TC

使用counter模块实例化一个计数器u1，并将输入输出信号连接到该实例。

初始化块：

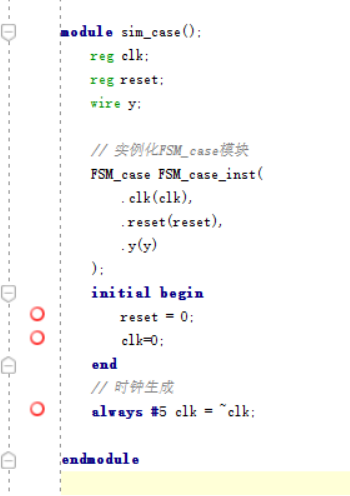
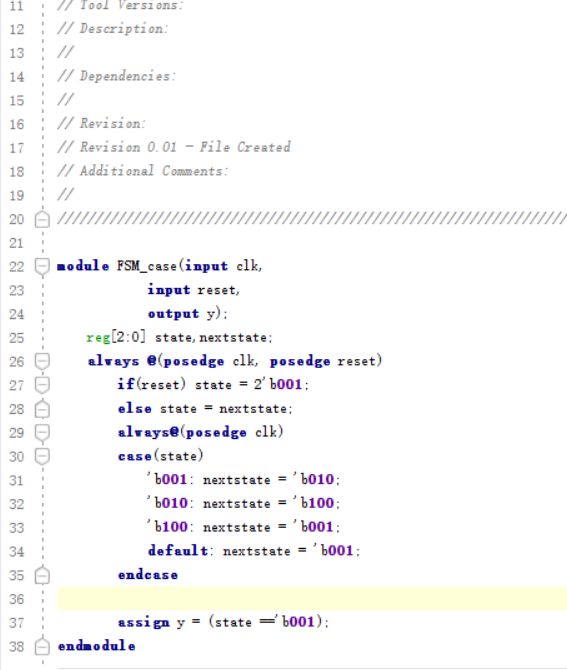
设置D的值为8'b0000\_0110，即6的二进制表示。设置PE为1，表示使能计数器。设置CET为0，表示不使能计数器的计数功能。设置CLK为0，表示时钟信号初始值为0。设置CEP为0，表示不使能计数器的计数功能。设置CR为1，表示清零计数器。延迟15个时间单位。设置CR为0，表示停止清零计数器。延迟10个时间单位。设置CR为1，表示再次清零计数器。设置PE为0，表示禁用计数器。延迟2个时间单位。设置CEP为1，表示使能计数器的计数功能。延迟1个时间单位。设置CET为1，表示使能计数器的计数功能。延迟7个时间单位。

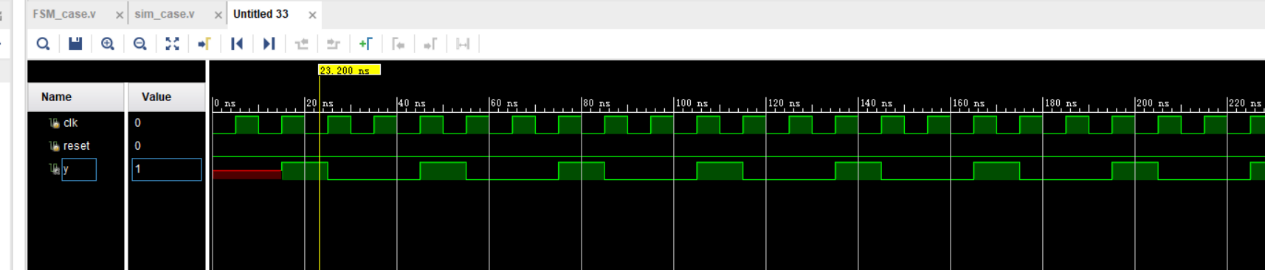
设置PE为1，表示使能计数器。延迟800个时间单位。设置CET为0，表示禁用计数器的计数功能。延迟15个时间单位。设置CEP为0，表示禁用计数器的计数功能。

使用always #10 CLK=~CLK的语句，每隔10个时间单位，将时钟信号CLK取反。

**最终可以通过时序图输出可以看到，在CLK时钟信号的脉冲下，Q数组的数字在缓慢加1，最终加到16进制的26，即是代表了十进制的38，说明计数器的正确性，39进制的计数器的实现成功了，计数器的技术范围是十进制的0-38,16进制下的00-26；**

状态机：



****

实现了一个有限状态机，根据输入的时钟信号和复位信号，以及当前状态state，计算出下一个状态nextstate，并根据当前状态判断输出信号y的值。

输入信号：clk、reset

输出信号：y

内部变量：state、nextstate

状态寄存器：

在always块中，使用posedge clk和 posedge reset触发条件，根据reset信号的值，将state的值初始化为2'b001或将state的值赋为nextstate的值。

状态转移逻辑：

在always块中，使用posedge clk触发条件，根据当前状态state，使用case语句计算下一个状态nextstate的值：

当state为2'b001时，nextstate被赋值为2'b010。

当state为2'b010时，nextstate被赋值为2'b100。

当state为2'b100时，nextstate被赋值为2'b001。

当state为其他值时，nextstate被赋值为2'b001。

输出逻辑：

使用assign语句，根据当前状态state判断输出信号y的值：

当state为2'b001时，y被赋值为1'b1。

当state为其他值时，y被赋值为1'b0。

仿真程序实现了对FSM\_case模块的仿真测试。

输入信号：clk、reset输出信号：y

实例化FSM\_case模块，并连接其输入输出信号。

使用always #5 clk = ~clk的语句，每隔5个时间单位，将时钟信号clk取反。

测试序列：

在initial块中，初始化reset为0，clk为0。

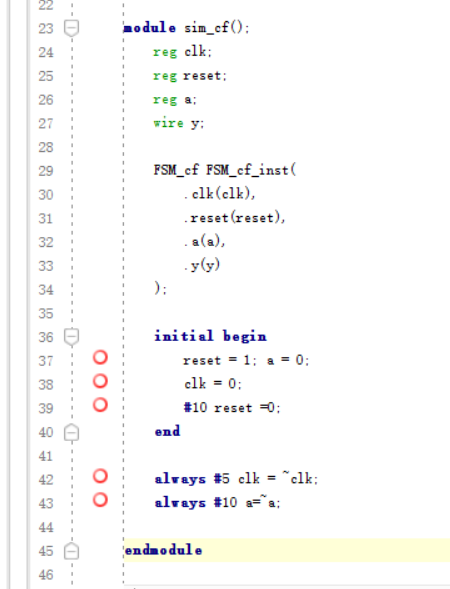
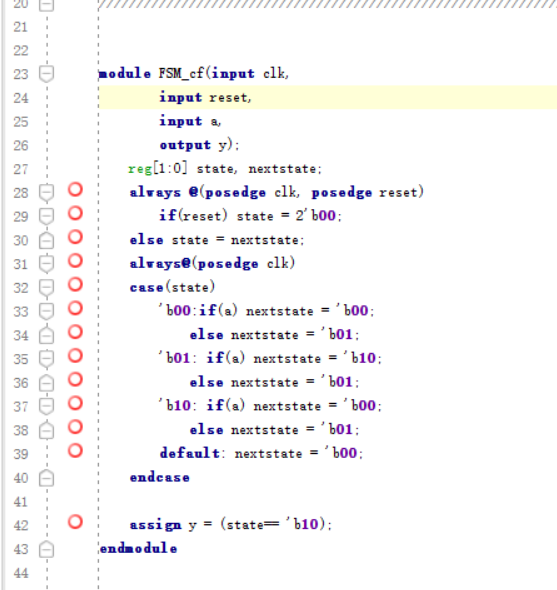
时序分析：

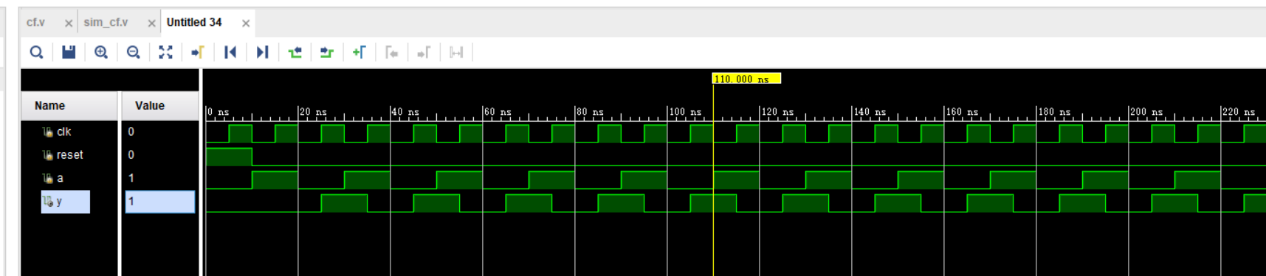
根据状态机代码的逻辑，初始状态state为2'b001，输出信号y为1'b1。然后，每当时钟信号clk上升沿到来时，状态机会根据当前状态state计算下一个状态nextstate。根据状态转移逻辑，状态会按照'b001 -> 'b010 -> 'b100 -> 'b001 -> 'b010 -> 'b100 -> ... 的顺序循环变化。

根据仿真文件中的时钟信号生成逻辑，时钟信号clk每隔5个时间单位取反一次。因此，我们可以预测输出信号y的变化情况如下：

初始状态：state = 2'b001，y = 1'b1 时钟上升沿：state = 2'b010，y = 1'b0 时钟上升沿：state = 2'b100，y = 1'b0 时钟上升沿：state = 2'b001，y = 1'b1 时钟上升沿：state = 2'b010，y = 1'b0 时钟上升沿：state = 2'b100，y = 1'b0 ...

根据状态机的循环特性，输出信号y会周期性地在0和1之间变化。

****

****

它有一个时钟输入clk，一个复位输入reset，一个输入信号a，一个输出信号y。首先定义了两个寄存器state和nextstate，用于存储当前状态和下一个状态。然后使用always块来描述状态转移逻辑。当时钟信号clk上升沿到来或复位信号reset上升沿到来时，根据reset的值来更新state的值。如果reset为1，表示复位，将state设置为2'b00；否则，将state设置为nextstate的值。

接着使用always块和case语句来描述状态转移逻辑。根据当前状态state的值，确定下一个状态nextstate的值。具体逻辑如下：

当state为2'b00时，如果输入信号a为1，则下一个状态为2'b00；否则，下一个状态为2'b01。

当state为2'b01时，如果输入信号a为1，则下一个状态为2'b10；否则，下一个状态为2'b01。

当state为2'b10时，如果输入信号a为1，则下一个状态为2'b00；否则，下一个状态为2'b01。

其他情况下，下一个状态为2'b00。

最后，使用assign语句将输出信号y赋值为(state == 2'b10)的结果。当且仅当当前状态为2'b10时，输出信号y为1；否则，输出信号y为0。

总结来说，以上程序实现了一个带有复位功能的有限状态机。根据输入信号a的值，状态在0、1和2之间循环切换，并且输出信号y在状态为2时为1，其他状态为0。

名为sim\_cf的模块，首先定义了一个时钟输入clk，一个复位输入reset，一个输入信号a，一个输出信号y。然后实例化了一个FSM\_cf模块，并将clk、reset、a、y连接到对应的端口。接着使用两个always块来描述时钟和输入信号的行为。

第一个always块中，每隔5个时间单位，时钟信号clk取反一次，即实现了时钟的周期性变化。

第二个always块中，每隔10个时间单位，输入信号a取反一次，即实现了输入信号的周期性变化。

最后，在initial块中初始化了reset和a的值为0，clk的值为0。然后等待10个时间单位后，将reset的值设为0。

**时序：**

时钟信号clk每隔5个时间单位变化一次，即0、5、10、15、20...依次循环。

输入信号a每隔10个时间单位变化一次，即0、10、20、30...依次循环。

在仿真开始时，reset和a的值都被初始化为0，clk的值也为0。

在10个时间单位后，reset的值被设为0，状态机开始运行。

根据状态机的状态转移逻辑，当输入信号a为0时，状态机的状态会在0和1之间循环切换；当输入信号a为1时，状态机的状态会在1和2之间循环切换。

当状态机的状态为2时，输出信号y的值为1；否则，输出信号y的值为0。

**五、调试和心得体会**

首先，我使用Logisim实现了SR锁存器。SR锁存器是一种基本的存储器元件，可以通过设置和复位输入来存储一个比特。在Logisim中，我通过使用RS触发器来实现SR锁存器，其中R和S输入分别对应设置和复位输入。通过这个实验，我更加深入地理解了SR锁存器的工作原理和应用场景。

接下来，我使用Verilog编写代码实现了D锁存器。D锁存器是一种存储器元件，可以存储一个比特的数据。在Verilog中，我可以使用一个D触发器来实现D锁存器。通过将输入数据传递到D触发器的D输入，并使用时钟信号来控制触发器的工作，我可以实现D锁存器。这个实验让我更加熟悉了Verilog语言和D锁存器的原理。

然后，我使用Verilog实现了D触发器。D触发器是一种存储器元件，可以存储一个比特的数据，并在时钟上升沿触发。通过使用Verilog中的always块和posedge关键字，我可以实现D触发器的行为。这个实验加深了我对D触发器的理解和Verilog语言的应用。

接着，我使用Verilog实现了寄存器。寄存器是一种存储多个比特的存储器元件。我可以使用Verilog中的reg类型和always块来实现寄存器。通过在always块中根据时钟信号和复位信号的状态来更新寄存器的值，我可以实现寄存器的功能。这个实验让我更加熟悉了寄存器的原理和Verilog语言的使用。

然后，我使用Verilog实现了移位寄存器。移位寄存器是一种特殊的寄存器，可以将存储的数据进行位移操作。我可以使用Verilog中的shift操作符和for循环来实现移位寄存器。通过这个实验，我更深入地理解了移位寄存器的原理和Verilog语言的应用。

接下来，我使用Verilog实现了39进制计数器。这是一种特殊的计数器，可以在每个时钟周期内按照39进制进行计数。为了实现39进制计数器，我设计了一个6位的计数器，通过递增和重置操作来实现计数。这个实验让我更加深入地理解了39进制计数器的原理和Verilog语言的使用。

最后，我使用Verilog实现了一个检测二进制序列中的01序列的状态机。状态机是一种用于描述系统行为的模型。通过使用Verilog中的always块和case语句，我可以实现一个简单的状态机，用于检测二进制序列中的01序列。这个实验加深了我对状态机的理解和Verilog语言的应用。

通过这些实验，我不仅加深了对计算机组成原理的理论知识的理解，还学会了如何将这些理论应用到实际操作中。这些实验经验对我今后的学习和研究都将有很大的帮助。我相信，通过这些实验，我可以更加深入地理解计算机硬件的工作原理和实现方法。