实验二 基本功能模块程序设计

**一、实验目的**

1. 掌握Verilog语言框架，编程及调试的方法。
2. 熟悉Verilog的基本语法。
3. 掌握Verilog中时序模块电路的设计方法。
4. 熟悉Verilog中层次结构的设计方法。
5. 掌握Logisim的使用。

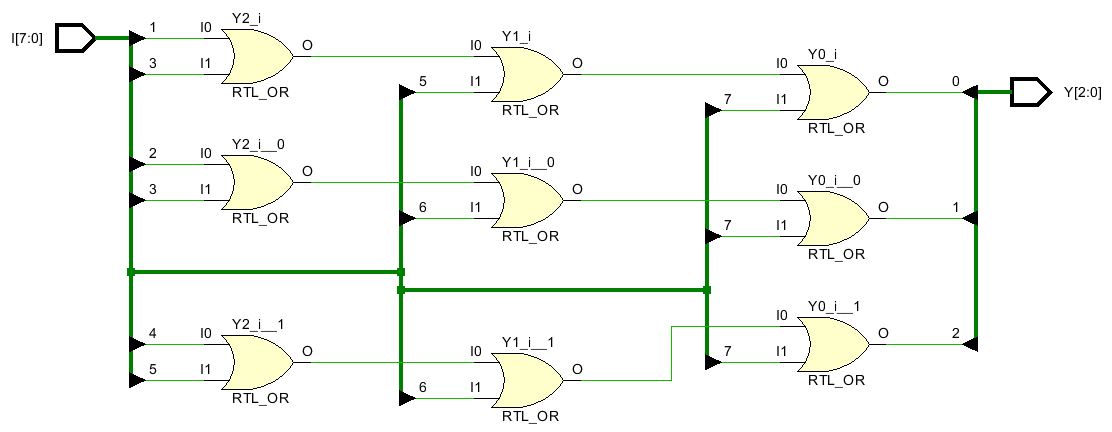
**二、实验内容（使用Logisim或Vivado实现）**

1. 完成编码器、译码器等功能。
2. 完成一个触发器电路模块（使能端、复位）的设计。
3. 完成寄存器和移位寄存器（循环移位）电路模块的设计。
4. 完成各种数字（个人学号末两位）进制的计数器。
5. 掌握以上电路的程序结构和风格。（logisim可忽略）
6. 观察和分析仿真波形，注重输入输出之间的时序关系。（logisim可忽略）

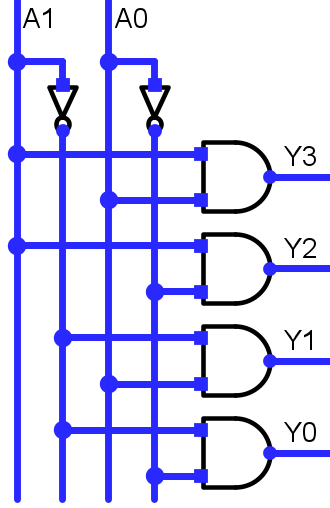
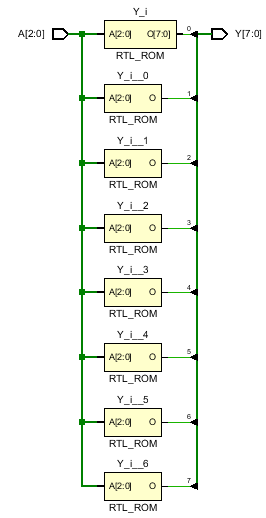
**三、实验要求**

1. 画出模块的电路图。

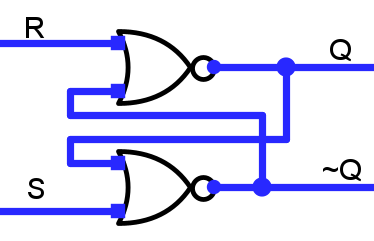
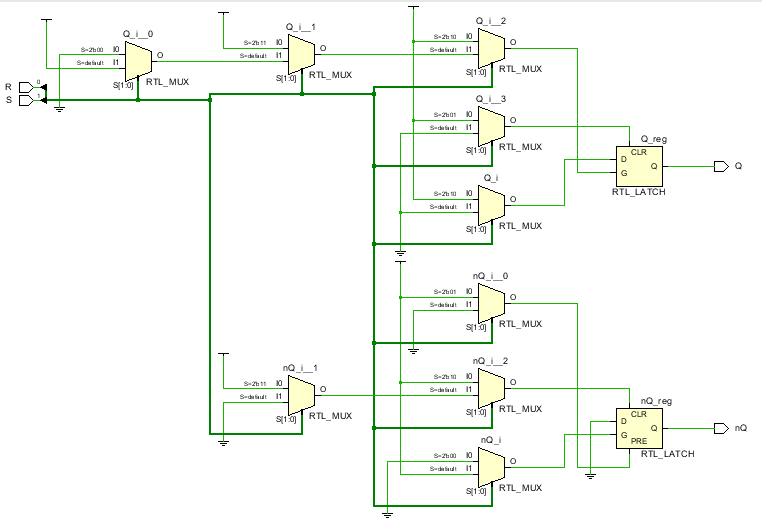
1.1 8-3编码器



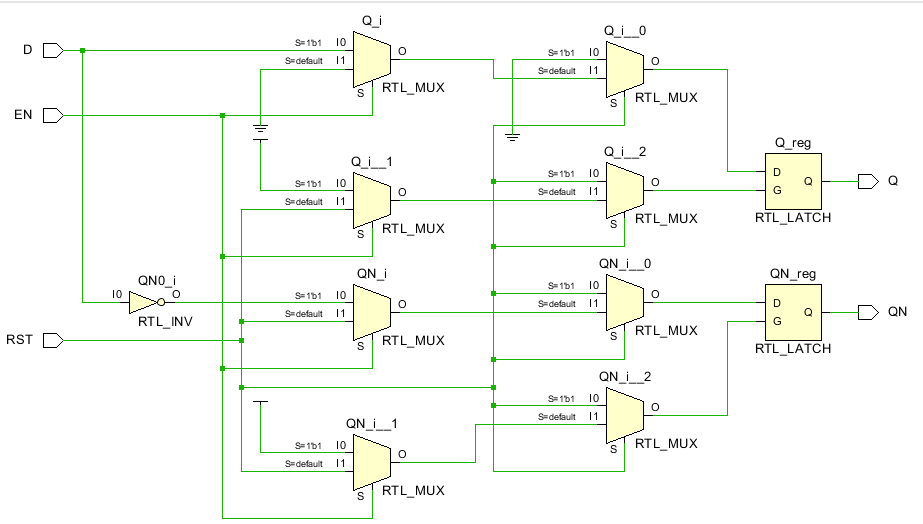
* 1. 3-8译码器

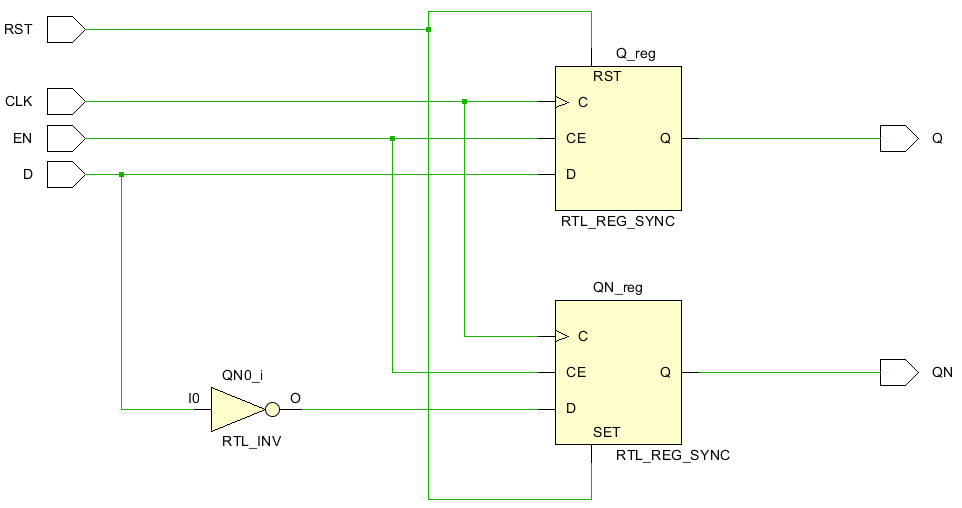
* 1. SR锁存器

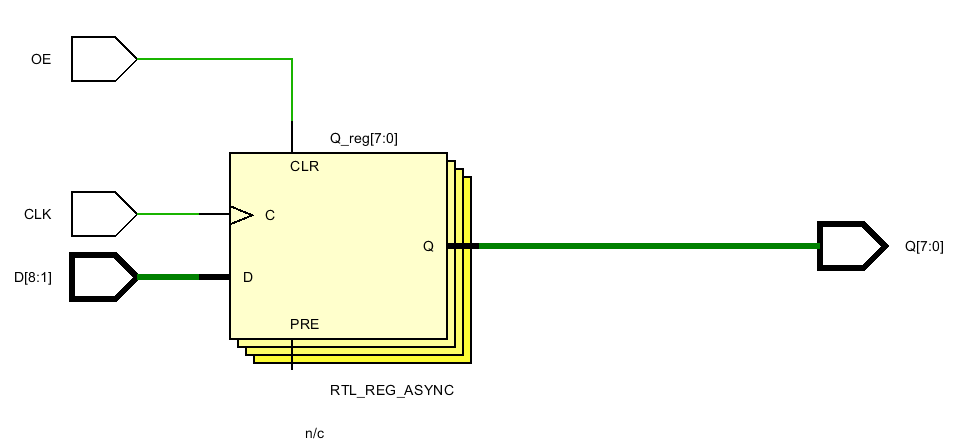
* 1. D锁存器



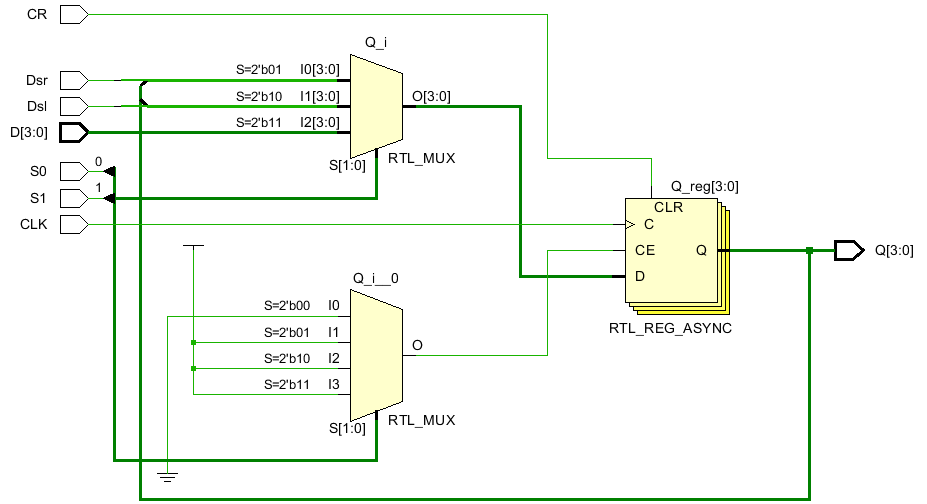
* 1. D触发器



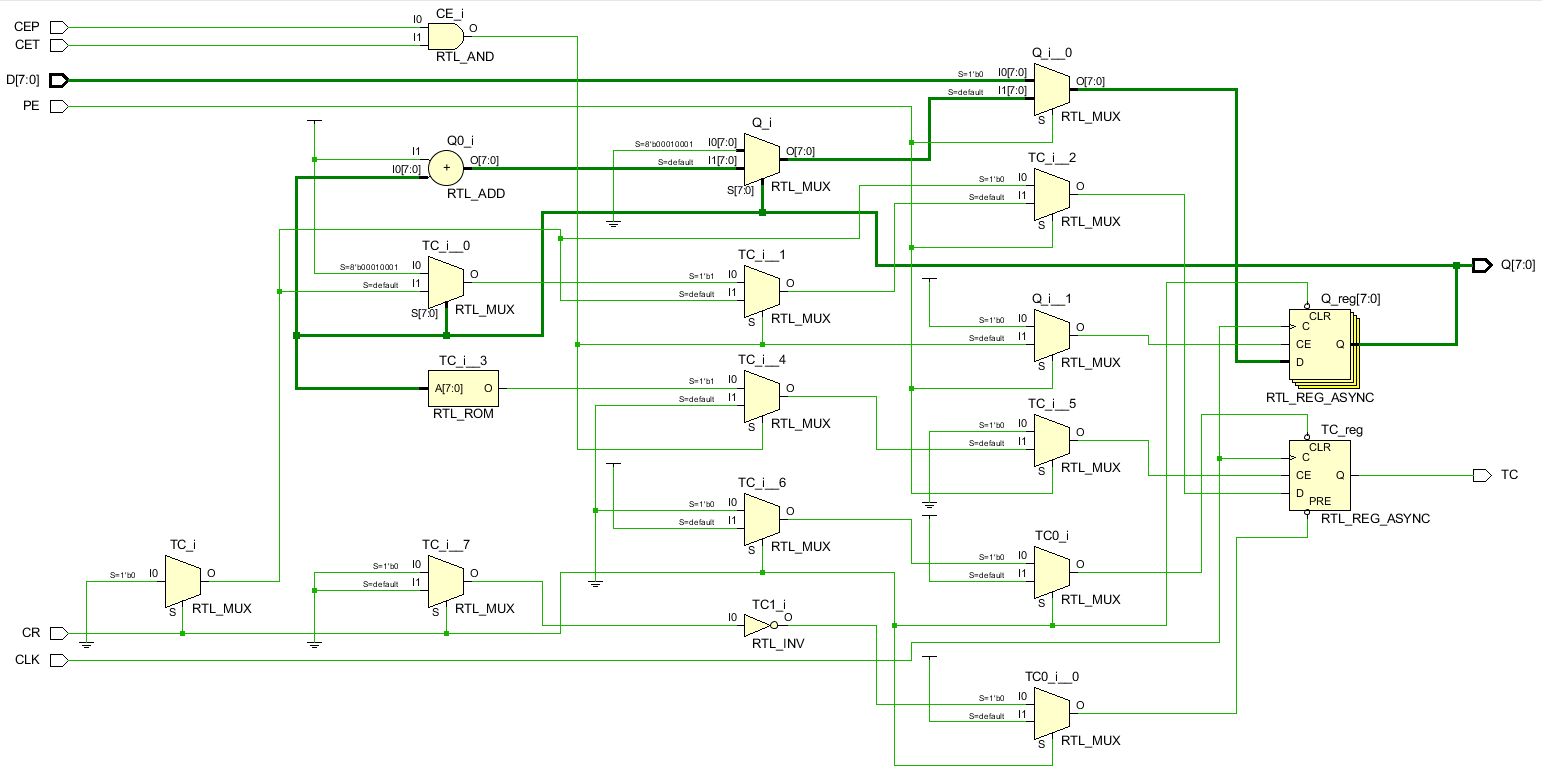
* 1. 寄存器



* 1. 移位寄存器

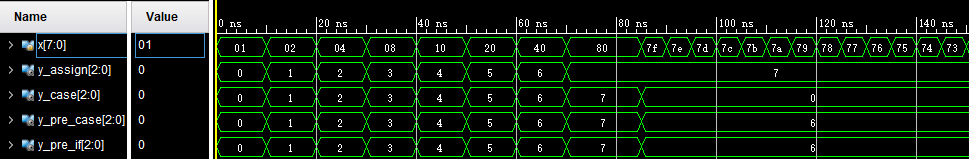


* 1. 十八进制的计数器



1. 分析电路的仿真波形/真值表/逻辑表达式。

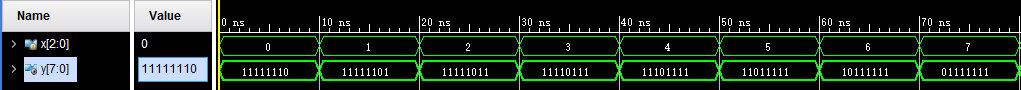
2.1 8-3编码器

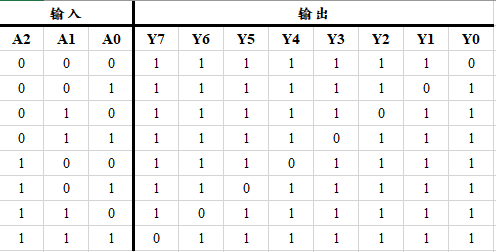




X作为八位输入分别对应I7~I0,并且某一位输入1会使Y输出结果为其编号。故仿真信号中依次让I0~I7为1，分别用四种语法格式得输出Y值，均正常。之后让x从1000 0000b=128d开始逐次减一，因为得Y值方式不同，所以产生了不同的结果。在assign方式中Y = {I[4]|I[5]|I[6]|I[7],I[2]|I[3]|I[6]|I[7],I[1]|I[3]|I[5]|I[7]}，x从1000 0000减一后为0111 1111根据此公式此时为111b = 7d。在case方法中，只写了I0~I7的情况，且定义此外情况均为Y = 000b = 0。在pre\_case中，仅检测对应Ix位是否为一，高位是否全为0，不关心地位的状态，故为0111 1111满足01XX XXXX的情况，故输出Y为6。在pre\_if中，由于if执行的先后顺序，从高位开始判断，故在I6 = 1是不考虑低位情况，输出Y为6。

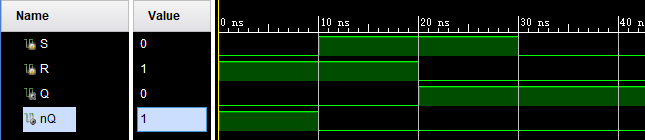
* 1. 3-8译码器





X三位输出使A的值对应八位输出I7~I0，使对应位为0，其余均为1。在仿真文件中使x每10ns自增一，对应y输出为八位二进制数。

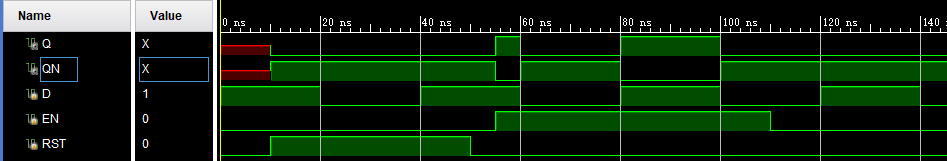
* 1. SR锁存器





当输入都为0时，Q与~Q保持先前的状态，输入都为1，Q与~Q置零，输入不同时，Q与S保持一致，~Q与R保持一致。

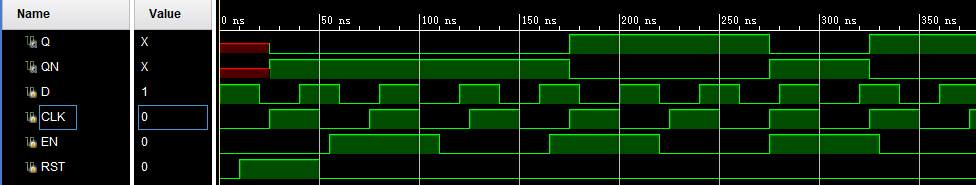
* 1. D锁存器



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| D | EN | RST | Q | QN |
| d | d | 1 | 0 | 1 |
| d | 0 | 0 | Qpre | QNpre |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

当RST复位键有效时，将Q置0，~Q置1。RST复位键无效时，EN使能端无效时，Q与QN保持先前的状态。RST复位键无效时，EN使能端有效时，Q与D保持一致，QN与~D保持一致。

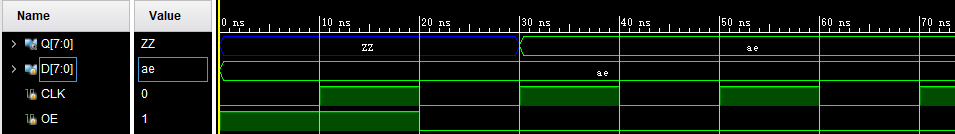
* 1. D触发器



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D | CLK | EN | RST | Q | QN |
| d | d | d | 1 | 0 | 1 |
| d | d | 0 | 0 | Qpre | QNpre |
| 0 | ↑ | 1 | 0 | 0 | 1 |
| 1 | ↑ | 1 | 0 | 1 | 0 |
| d | x | 1 | 0 | Qpre | QNpre |

基本与D锁存器一致，不过在原先RST复位键无效时，EN使能端有效时，CLK时钟信号出现上升沿，Q才和D保持一致，QN与~D保持一致，CLK其他状态时，Q与QN保持先前的状态。

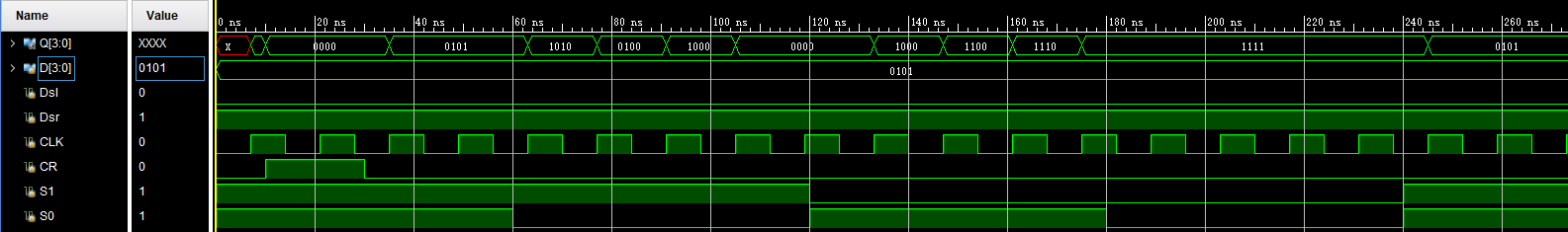
* 1. 寄存器



|  |  |  |
| --- | --- | --- |
| CLK | OE | Q |
| x | 1 | ZZ |
| ↑ | 0 | <=D |
| x | 0 | Qpre |

在OE信号有效时，让Q为高阻态。在OE信号无效，当CLK时钟信号出现上升沿时，改变输出Q为预先设定的D值，并在CLK其他信号时保持。

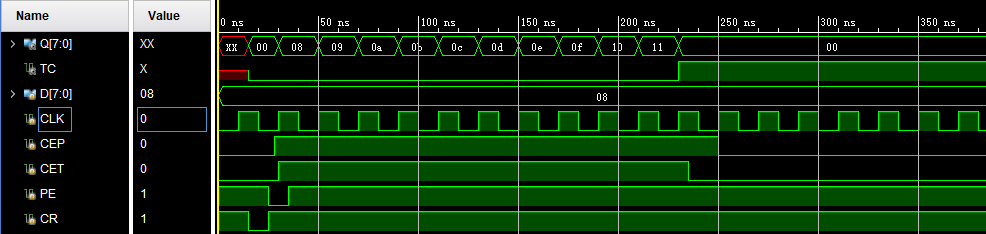
* 1. 移位寄存器



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLK | ~CR | S1 S0 | 功能 | Q值 |
| X | 0 | X | 清零 | Q = 0 |
| X | 1 | 00 | 保持 | Q <=Q |
| ↑ | 1 | 01 | 右移 | Q[N-1:0]<={Dsr,Q[N-1:1]} |
| ↑ | 1 | 10 | 左移 | Q[N-1:0]<={Q[N-2:0],Dsl} |
| ↑ | 1 | 11 | 并行输入 | Q <= D |

见真值表。CR信号有效时清零输出Q。在CR信号无效，CLK出现上升沿时，根据S1与S0值做不同操作，为00时保持Q值不变，为01时让Q值右移一位，去除最低位，将最高位置1，为10时让Q值左移一位，去除最高位，将最低位置0，为11时改变Q为D值。

* 1. 十八进制的计数器



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CP | ~CR | ~PE | CEP CET | 功能 | Q值 |
| X | 1 | X | XX | 异步清零 | Q = 0 |
| ↑ | 0 | 1 | XX | 同步输入 | Q <= D |
| ↑ | 0 | 0 | 11 | 计数 | Q <= Q+1 |
| X | 0 | 0 | 0X | 保持 | Q <= Q |
| X | 0 | 0 | X0 | 保持 | Q <= Q |

见真值表。CR信号无效时清零输出Q。在CR信号有效，PE信号无效时，改变Q为D值。在CR信号,PE信号，CEP信号，CET信号有效时，CLK出现上升沿时，Q值自增一。其他状态保持Q值不变。

1. 记录设计和调试过程。

根据对应模块要求设计，并写出方便观察各种情况的仿真波形的仿真文件。

**四、实验代码及结果**

1. 8-3编码器

encoder83\_assign.v

module encoder83\_assign(I, Y);

input I;

output Y;

wire [7:0] I;

reg[3:1] Y;

always @(I) begin

assign Y = {I[4]|I[5]|I[6]|I[7],I[2]|I[3]|I[6]|I[7],I[1]|I[3]|I[5]|I[7]};

end

endmodule

encoder83\_case.v

module encoder83\_case(I, Y);

input I;

output Y;

wire [7:0] I;

reg[3:1] Y;

always @(I) begin

case(I)

8'b0000\_0001: Y = 3'b000;

8'b0000\_0010: Y = 3'b001;

8'b0000\_0100: Y = 3'b010;

8'b0000\_1000: Y = 3'b011;

8'b0001\_0000: Y = 3'b100;

8'b0010\_0000: Y = 3'b101;

8'b0100\_0000: Y = 3'b110;

8'b1000\_0000: Y = 3'b111;

default: Y = 3'b000;

endcase

end

endmodule

pre\_encoder83\_case.v

module pre\_encoder83\_case(I, Y);

input I;

output Y;

wire [7:0] I;

reg[3:1] Y;

always @(I) begin

casex(I)

8'b0000\_0001: Y = 3'b000;

8'b0000\_001X: Y = 3'b001;

8'b0000\_01XX: Y = 3'b010;

8'b0000\_1XXX: Y = 3'b011;

8'b0001\_XXXX: Y = 3'b100;

8'b001X\_XXXX: Y = 3'b101;

8'b01XX\_XXXX: Y = 3'b110;

8'b1XXX\_XXXX: Y = 3'b111;

default: Y = 3'b000;

endcase

end

endmodule

pre\_encoder83\_if.v

module pre\_encoder83\_if(I, Y);

input I;

output Y;

wire [7:0] I;

reg[3:1] Y;

always @(\*) begin

if(I[7] == 1) Y = 3'b111;

else if(I[6] == 1) Y = 3'b110;

else if(I[5] == 1) Y = 3'b101;

else if(I[4] == 1) Y = 3'b100;

else if(I[3] == 1) Y = 3'b011;

else if(I[2] == 1) Y = 3'b010;

else if(I[1] == 1) Y = 3'b001;

else if(I[0] == 1) Y = 3'b000;

else Y = 3'b000;

end

endmodule

sim\_ encoder83.v

module sim\_encoder83();

reg[7:0] x;

wire[2:0] y\_assign, y\_case, y\_pre\_case, y\_pre\_if;

integer i;

initial begin

x = 1;

for(i = 0; i < 7; i = i + 1) #10 x = x \* 2;

#10 x = 128;

while(x > 0) #5 x = x - 1;

end

encoder83\_assign encoder83\_assign\_1(x, y\_assign);

encoder83\_case encoder83\_case\_1(x, y\_case);

pre\_encoder83\_case pre\_encoder83\_case\_1(.I(x), .Y(y\_pre\_case));

pre\_encoder83\_if pre\_encoder83\_if\_1(.I(x), .Y(y\_pre\_if));

endmodule

2. 3-8译码器

decoder38.v

module decoder38(A, Y);

input[2:0] A;

output reg[7:0] Y;

integer i;

always @(A) begin

Y = 8'b1111\_1111;

for(i = 0; i <= 7; i = i + 1)

if(A == i)

Y[i] = 0;

else

Y[i] = 1;

end

endmodule

sim\_decoder38.v

module sim\_decoder38();

reg[2:0] x;

wire[7:0] y;

initial begin

for(x = 0; x <= 7; x = x + 1) #10;

end

decoder38 decoder38\_1(x, y);

endmodule

1. SR锁存器

sr.v

module sr(S, R, Q, nQ);

input S, R;

output reg Q, nQ;

always @(S, R) begin

if({S, R} == 2'b01) {Q, nQ} = 2'b01;

else if({S, R} == 2'b10) {Q, nQ} = 2'b10;

else if({S, R} == 2'b11) {Q, nQ} = 2'b00;

else if({S, R} == 2'b00) ;

else {Q, nQ} = 2'b00;

end

endmodule

sim\_sr.v

module sim\_sr();

reg S, R;

wire Q, nQ;

initial begin

{S, R} = 2'b01;

#10 {S, R} = 2'b11;

#10 {S, R} = 2'b10;

#10 {S, R} = 2'b00;

end

sr sr\_1(S, R, Q, nQ);

endmodule

1. D锁存器

D\_latch.v

module D\_latch(Q, QN, D, EN, RST);

output reg Q, QN;

input D, EN, RST;

always @(D, EN, RST) begin

if(RST) begin

Q = 0;

QN = 1;

end

else if(EN) begin

Q <= D;

QN <= ~D;

end

end

endmodule

sim\_D\_latch.v

module sim\_D\_latch();

reg D, EN, RST;

wire Q, QN;

initial begin

D = 1;EN = 0; RST = 0;

fork

forever #20 D = ~D;

begin #10 RST = 1; #40 RST = 0; end

begin #55 EN = 1; #55 EN = 0; end

join

end

D\_latch D\_latch\_1(Q, QN, D, EN, RST);

endmodule

1. D触发器

D\_ff.v

module D\_ff(Q, QN, D, EN, RST, CLK);

output reg Q, QN;

input D, EN, RST, CLK;

always @(posedge CLK) begin

if(RST) begin

Q <= 0;

QN <= 1;

end

else if(EN) begin

Q <= D;

QN <= ~D;

end

end

endmodule

sim\_D\_ff.v

module sim\_D\_ff();

reg D, EN, RST, CLK;

wire Q, QN;

initial begin

D = 1;EN = 0; RST = 0;CLK = 0;

fork

forever #20 D = ~D;

forever #25 CLK = ~CLK;

begin #10 RST = 1; #40 RST = 0; end

forever #55 EN = ~EN;

join

end

D\_ff D\_ff\_1(Q, QN, D, EN, RST, CLK);

Endmodule

1. 寄存器

register.v

module register(Q, D, OE, CLK);

parameter N = 8;

output reg[N-1:0] Q;

input [N:1] D;

input OE, CLK;

always @(posedge OE or posedge CLK)

if(OE) Q <= 8'bzzzz\_zzzz;

else Q <= D;

endmodule

sim\_register.v

module sim\_register();

reg CLK, OE;

reg[7:0] D;

wire[7:0] Q;

initial begin

OE = 1;CLK = 0;D = 8'b1010\_1110;

fork

#20 OE = 0;

forever #10 CLK = ~CLK;

join

end

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

endmodule

1. 移位寄存器

shift\_register.v

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

parameter N = 4;

input S1, S0, Dsl, Dsr, 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

sim\_ shift\_register.v

module sim\_shift\_register();

parameter N = 4;

reg S1, S0, Dsl, Dsr, CLK, CR;

reg[N-1:0] D;

wire[N-1:0] Q;

initial begin

S0 = 1;S1 = 1;CR = 0;CLK = 0;Dsl = 0;Dsr = 1;D = 4'b0101;

fork

forever #7 CLK = ~CLK;

begin #10 CR = 1;#20 CR = 0; end

forever #60 S0 = ~S0;

forever #120 S1 = ~S1;

join

end

shift\_register shift\_register\_1(S1, S0, D, Dsl, Dsr, Q, CLK, CR);

endmodule

1. 十八进制的计数器

counter74x181.v

module counter74x181(CEP, CET, PE, CLK, CR, D, TC, Q);

parameter N = 8;

parameter M = 18;

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

sim\_counter74x181.v

module sim\_counter74x181();

parameter N = 8;

parameter M =16;

reg CEP, CET, PE, CLK, CR;

reg[N-1:0] D;

wire TC;

wire[N-1:0] Q;

initial begin

CEP = 0; CET = 0;PE = 1;CLK = 0;CR = 1;D = 8'b0000\_1000;

fork

forever #10 CLK = ~CLK;

#28 CEP = 1;#250 CEP = 0;

#30 CET = 1;#235 CET = 0;

#25 PE = 0;#35 PE = 1;

#15 CR = 0;#25 CR = 1;

join

end

counter74x181 counter74x181\_1(CEP, CET, PE, CLK, CR, D, TC, Q);

endmodule

**五、调试和心得体会**

合适的仿真文件方便观察各种情况，合适的仿真波形对于帮助找到错误非常重要。

复习熟悉了fork…join，begin…end的语法。其中begin…end为串行，fork…join为并行，可嵌套使用，例如在并行中想两次改变某一输入的值，可以在fork…join中嵌套begin #10 D=1； #10 D=0； end，也可以直接写#10 D=1； #20 D=0；。

研究了case，if，casex，assign等语句的用法和区别。