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。

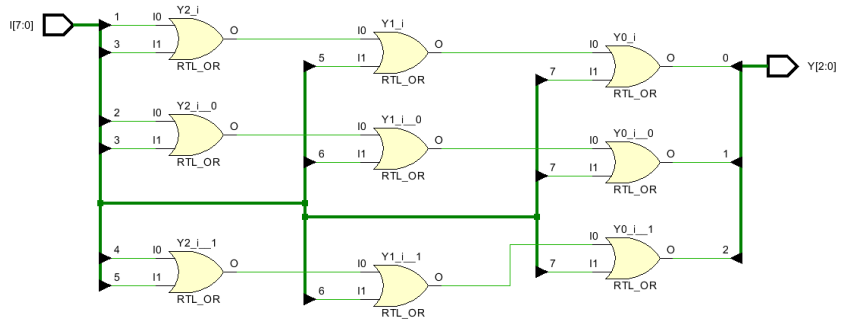
二、实验内容（使用 Logisim 或 Vivado 实现）

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）

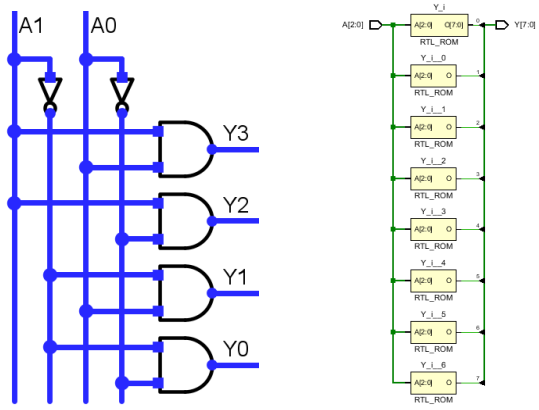
三、实验要求

1. 画出模块的电路图。

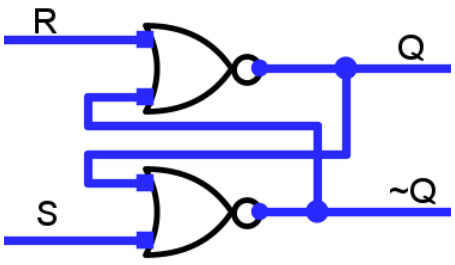
1.1 8-3 编码器

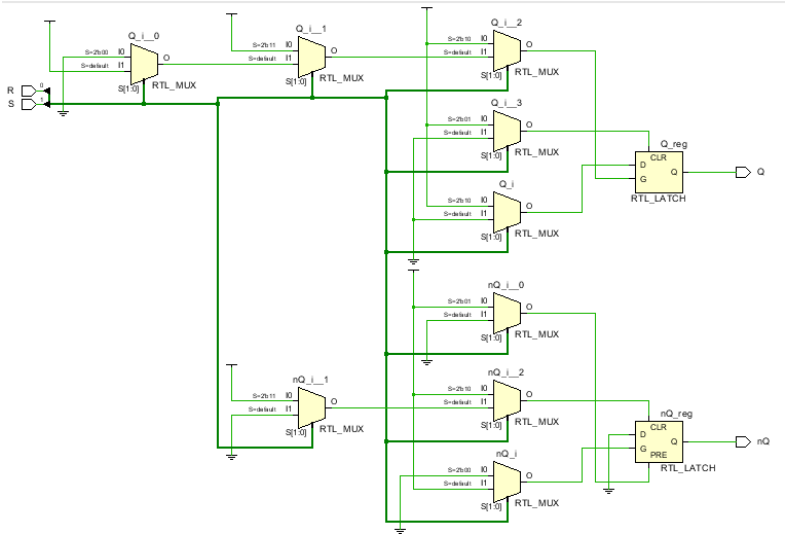


1.2 3-8 译码器

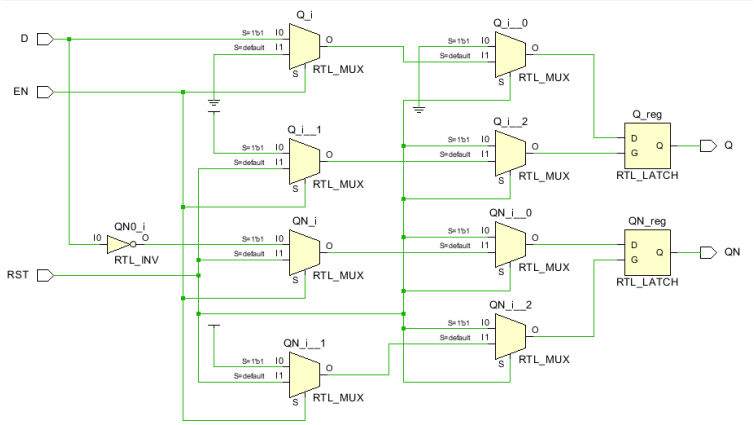


1.3 SR 锁存器

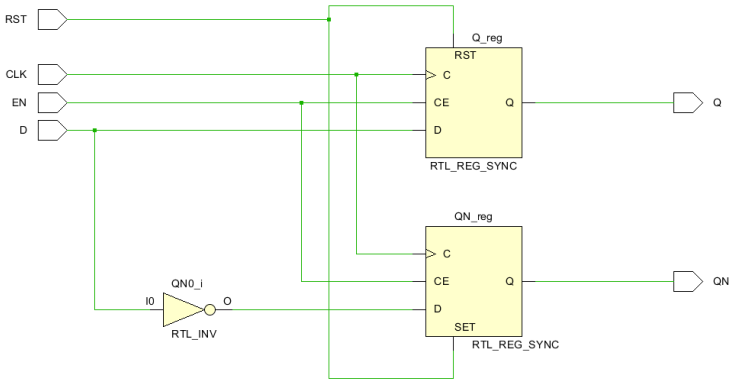




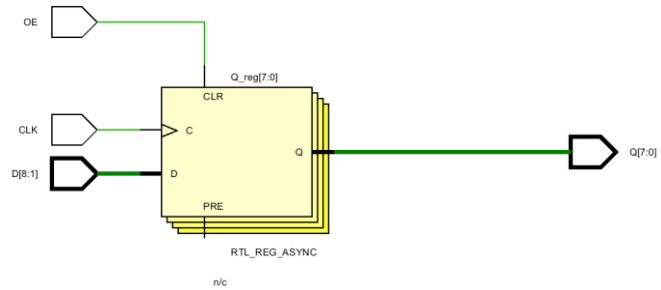
1.4 D 锁存器



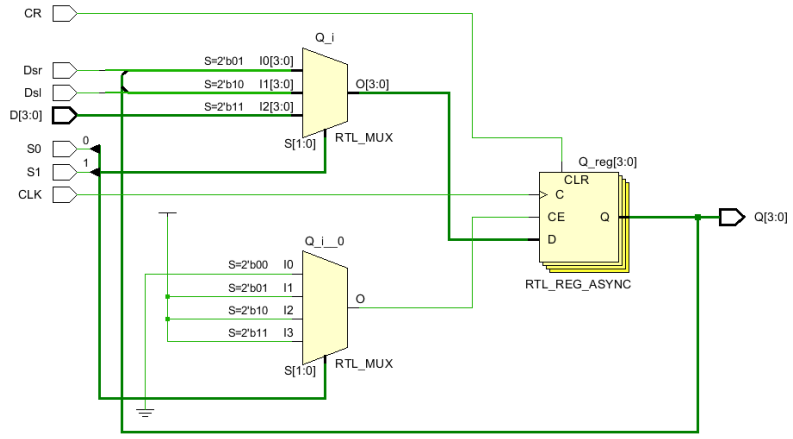
1.5 D 触发器



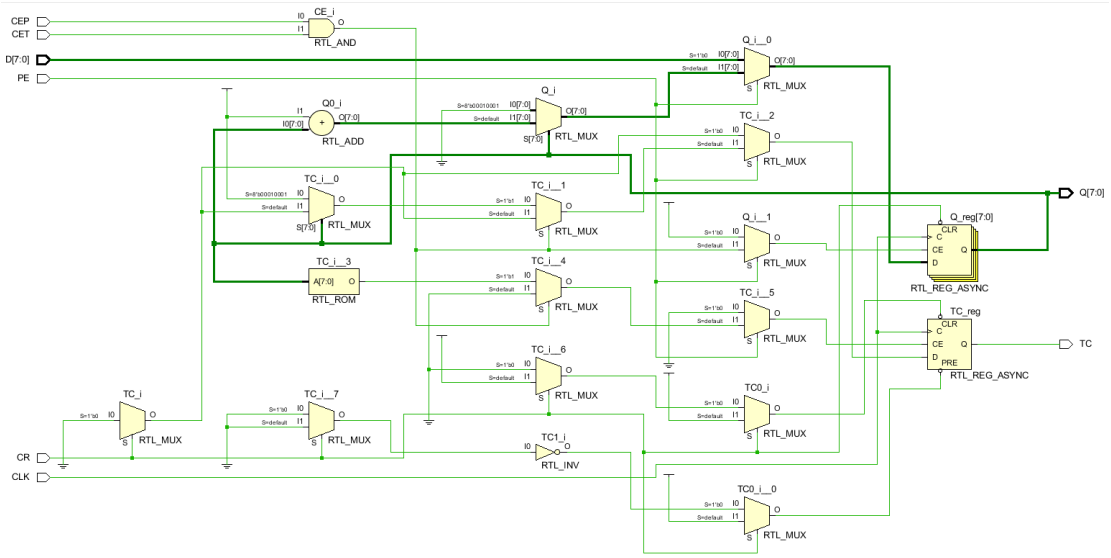
1.6 寄存器



1.7 移位寄存器

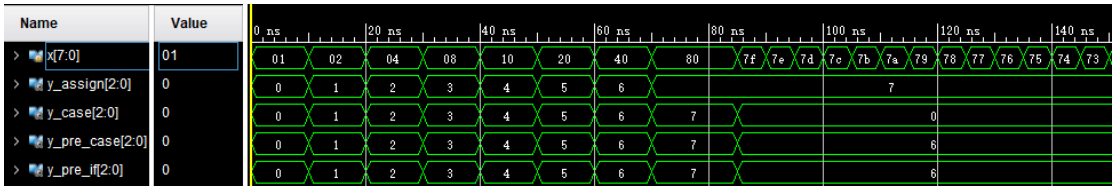


1.8 十八进制的计数器



2. 分析电路的仿真波形/真值表/逻辑表达式。

2.1 8-3 编码器

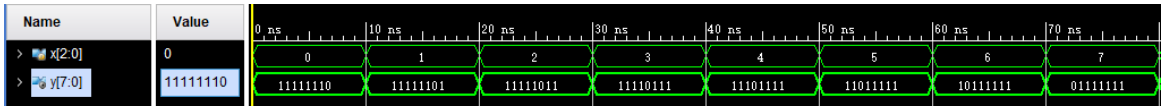


输入								输出		
I7	I6	I5	I4	I3	I2	I1	I0	Y2	Y1	Y0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

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。

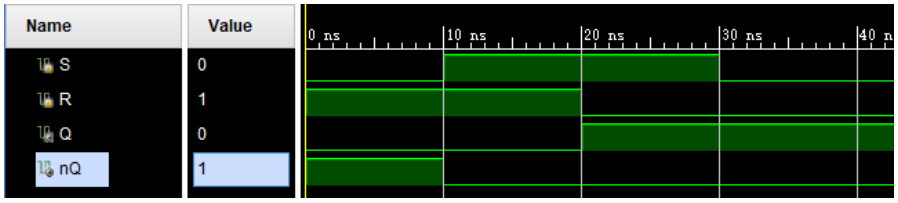
2.2 3-8 译码器



输入			输出							
A2	A1	A0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1

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。

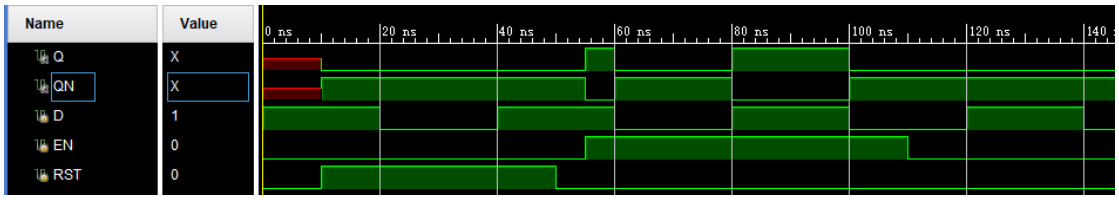
2.3 SR 锁存器



情况	S	R	Q	~Q
IV	0	0	Q_{prev}	$\sim Q_{prev}$
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0

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。

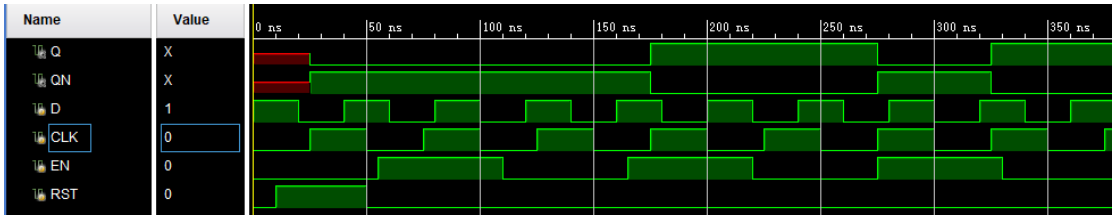
2.4 D 锁存器



D	EN	RST	Q	QN
d	d	1	0	1
d	0	0	Q_{pre}	QN_{pre}
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。

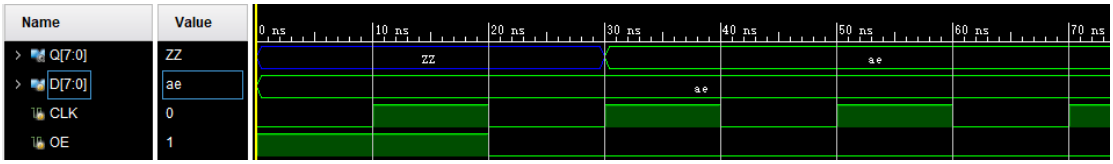
2.5 D 触发器



D	CLK	EN	RST	Q	QN
d	d	d	1	0	1
d	d	0	0	Q_{pre}	QN_{pre}
0	↑	1	0	0	1
1	↑	1	0	1	0
d	x	1	0	Q_{pre}	QN_{pre}

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。

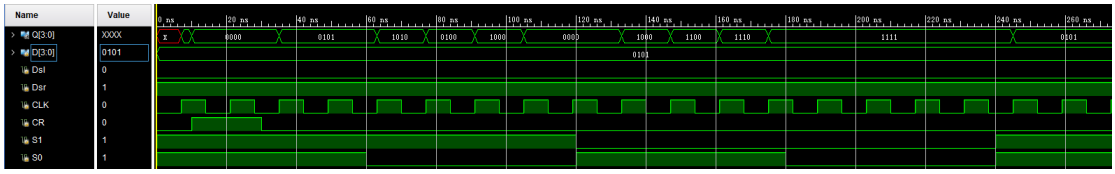
2.6 寄存器



CLK	OE	Q
x	1	ZZ
↑	0	<=D
x	0	Q_{pre}

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。

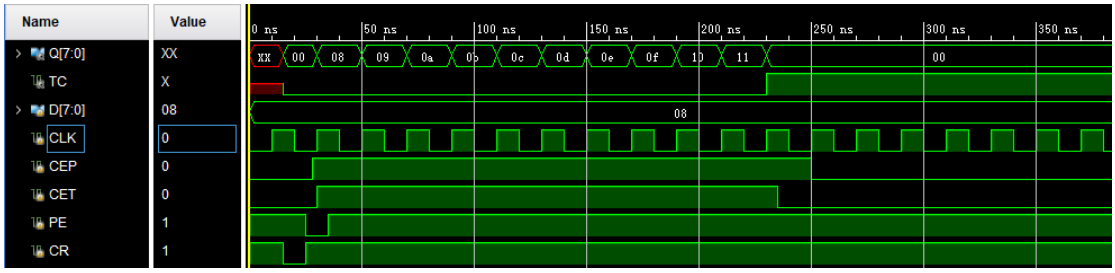
2.7 移位寄存器



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。

2.8 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。

3. 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
```

3. 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
```

4. 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
```

5. 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
```

6. 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
```

7. 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
```

8. 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。