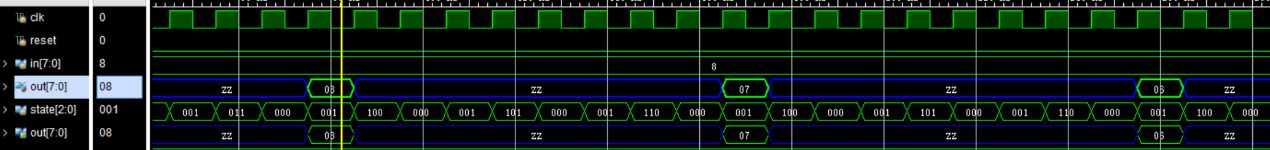
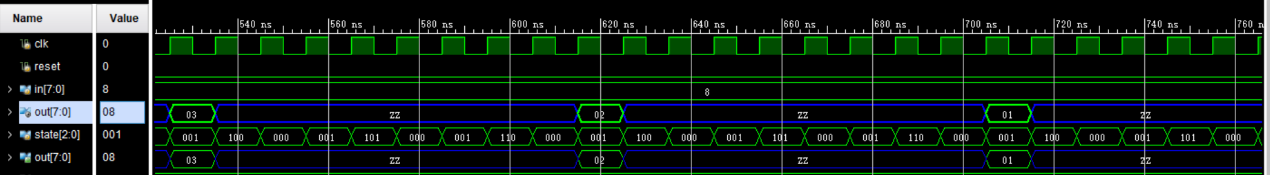
**实验报告 Lab 7**

一、代码（见附页）

二、仿真

****

**附页** 代码

module EC1(

input clk,

input reset,

input [7:0] in,

output [7:0] out

);

wire clk\_2s;

wire IRload;

wire PCload;

wire INmux;

wire Aload;

wire JNZmux;

wire OE;

wire [2:0] IR1;

wire a\_0;

clk\_div div1(

.clk(clk),

.reset(~reset),

.clk\_2s(clk\_2s)

);

Control control\_unit (

.clk(clk\_2s),

.reset(~reset),

.a\_0(a\_0),

.IR1(IR1),

.IRload(IRload),

.PCload(PCload),

.INmux(INmux),

.Aload(Aload),

.JNZmux(JNZmux),

.OE(OE)

);

Datapath datapath (

.IRload(IRload),

.JNZmux(JNZmux),

.PCload(PCload),

.INmux(INmux),

.Aload(Aload),

.reset(~reset),

.clk(clk\_2s),

.in(in),

.OE(OE),

.IR1(IR1),

.a\_0(a\_0),

.out(out)

);

endmodule

module Control(

input clk,

input reset,

input a\_0,

input [2:0]IR1,

output reg IRload,

output reg PCload,

output reg INmux,

output reg Aload,

output reg JNZmux,

output reg OE

);

parameter Fetch = 0;

parameter Decode = 1;

parameter Input = 3;

parameter Output = 4;

parameter Dec = 5;

parameter JNZ = 6;

parameter Halt = 7;

reg [2:0] state, next\_state;

always@(\*) begin

if (reset) begin // 统一为高电平复位

next\_state = Fetch;

end else begin

case (state)

Fetch: next\_state = Decode ;

Decode:

if (IR1==3'b000||IR1==3'b001||IR1==3'b010)

next\_state = Fetch ;

else if (IR1==3'b011)

next\_state = Input ;

else if (IR1==3'b100)

next\_state = Output ;

else if (IR1==3'b101)

next\_state = Dec ;

else if (IR1==3'b110)

next\_state = JNZ ;

else if (IR1==3'b111)

next\_state = Halt ;

Input: next\_state = Fetch ;

Output: next\_state = Fetch ;

Dec: next\_state = Fetch ;

JNZ: next\_state = Fetch ;

Halt: next\_state = Halt ;

default:next\_state = Fetch;

endcase

end

end

// 状态寄存器更新逻辑

always@(posedge clk) begin

if (reset) begin

state <= Fetch;

end else begin

state <= next\_state;

end

end

always@(\*)begin

case (next\_state)

Fetch: begin

IRload = 1;

PCload = 1;

INmux = 0;

Aload = 0;

JNZmux = 0;

OE = 0;

end

Decode: begin

IRload = 0;

PCload = 0;

INmux = 0;

Aload = 0;

JNZmux = 0;

OE = 0;

end

Input: begin

IRload = 0;

PCload = 0;

INmux = 1;

Aload = 1;

JNZmux = 0;

OE = 0;

end

Output: begin

IRload = 0;

PCload = 0;

INmux = 0;

Aload = 0;

JNZmux = 0;

OE = 1;

end

Dec: begin

IRload = 0;

PCload = 0;

INmux = 0;

Aload = 1;

JNZmux = 0;

OE = 0;

end

JNZ: begin

IRload = 0;

PCload = a\_0;

INmux = 0;

Aload = 0;

JNZmux = 1;

OE = 0;

end

Halt: begin

IRload = 0;

PCload = 0;

INmux = 0;

Aload = 0;

JNZmux = 0;

OE = 0;

end

endcase

end

endmodule

module Datapath(

input IRload,

input JNZmux,

input PCload,

input INmux,

input Aload,

input reset,

input clk,

input [7:0]in,

input OE,

output [2:0]IR1,

output a\_0,

output [7:0]out

);

wire [7:0]A1;

wire [3:0]A2;

wire [3:0]A3;

wire [3:0]A4;

wire [3:0]A5;

wire [7:0]B1;

wire [7:0]B2;

wire [7:0]B3;

Reg\_IR #(.N(8)) Reg\_IR1(.in(A1),.clk(clk),.reset(reset),.load(IRload),.IR0(A2),.IR1(IR1));

MUX #(.N(4)) MUX1(.A(A3),.B(A2),.S(JNZmux),.out(A4));

Reg #(.N(4)) Reg\_PC(.in(A4),.clk(clk),.reset(reset),.load(PCload),.out(A5));

incre #(.N(4)) incre1(.in(A5),.out(A3));

ROM ROM1(.in(A5),.out(A1));

MUX #(.N(8)) MUX2(.A(B1),.B(in),.S(INmux),.out(B2));

Reg #(.N(8)) Reg\_A(.in(B2),.clk(clk),.reset(reset),.load(Aload),.out(B3));

decre #(.N(8)) decre1(.in(B3),.out(B1));

OE #(.N(8)) OE1(.in(B3),.en(OE),.out(out));

assign a\_0=B3[7]|B3[6]|B3[5]|B3[4]|B3[3]|B3[2]|B3[1]|B3[0];

endmodule  
  
module Reg\_IR

#(parameter N = 8)(

input [N-1:0]in,

input clk,

input reset,

input load,

output [3:0]IR0,

output [2:0]IR1

);

reg [N-1:0]out;

always @(posedge clk or posedge reset) begin

if (reset)

out <= 0;

else if (load)

out <= in;

end

assign IR0 = out[3:0];

assign IR1 = out[7:5];

endmodule

module Reg

#(parameter N = 8)(

input [N-1:0]in,

input clk,

input reset,

input load,

output reg [N-1:0]out

);

always @(posedge clk or posedge reset) begin

if (reset)

out <= 0;

else if (load)

out <= in;

end

endmodule

module incre

#(parameter N = 8)(

input [N-1:0]in,

output [N-1:0]out

);

assign out=in+1;

endmodule

module ROM(

input [3:0]in,

output reg[7:0]out

);

always @(\*)begin

case (in)

4'b0000: out <= 8'b01100000;

4'b0001: out <= 8'b10000000;

4'b0010: out <= 8'b10100000;

4'b0011: out <= 8'b11000001;

4'b0100: out <= 8'b11111111;

default: out <= 8'b00000000;

endcase

end

endmodule