Prelab for Lab7

Xiaoyu Zhou, 1004081147

**Part I**

9. Simulation

图片包含 屏幕截图

描述已自动生成

10. Instantiate the ram32x4 module into a top-level Verilog modul

module top (SW, KEY, HEX0, HEX2, HEX4, HEX5);

input [9:0] SW;

input [3:0] KEY;

output [6:0] HEX0;

output [6:0] HEX2;

output [6:0] HEX4;

output [6:0] HEX5;

wire out;

kk u1(

.address(SW[8:4]),

.clock(KEY[0]),

.data(SW[3:0]),

.wren(SW[9]),

.q(out)

);

hex dataa(

.in2(SW[3]),

.in3(SW[2]),

.in4(SW[1]),

.in5(SW[0]),

.o2(HEX2[0]),

.o3(HEX2[1]),

.o4(HEX2[2]),

.o5(HEX2[3]),

.o6(HEX2[4]),

.o7(HEX2[5]),

.o8(HEX2[6])

);

hex outtt(

.in2(out[3]),

.in3(out[2]),

.in4(out[1]),

.in5(out[0]),

.o2(HEX0[0]),

.o3(HEX0[1]),

.o4(HEX0[2]),

.o5(HEX0[3]),

.o6(HEX0[4]),

.o7(HEX0[5]),

.o8(HEX0[6])

);

hex addd1(

.in2(SW[7]),

.in3(SW[6]),

.in4(SW[5]),

.in5(SW[4]),

.o2(HEX4[0]),

.o3(HEX4[1]),

.o4(HEX4[2]),

.o5(HEX4[3]),

.o6(HEX4[4]),

.o7(HEX4[5]),

.o8(HEX4[6])

);

hex addd2(

.in2(0),

.in3(0),

.in4(0),

.in5(SW[8]),

.o2(HEX5[0]),

.o3(HEX5[1]),

.o4(HEX5[2]),

.o5(HEX5[3]),

.o6(HEX5[4]),

.o7(HEX5[5]),

.o8(HEX5[6])

.o7(HEX5[5]),

.o8(HEX5[6])

);

endmodule

// synopsys translate\_off

`timescale 1 ps / 1 ps

// synopsys translate\_on

module kk (

address,

clock,

data,

wren,

q);

input [4:0] address;

input clock;

input [3:0] data;

input wren;

output [3:0] q;

`ifndef ALTERA\_RESERVED\_QIS

// synopsys translate\_off

`endif

tri1 clock;

`ifndef ALTERA\_RESERVED\_QIS

// synopsys translate\_on

`endif

wire [3:0] sub\_wire0;

wire [3:0] q = sub\_wire0[3:0];

altsyncram altsyncram\_component (

.address\_a (address),

.clock0 (clock),

.data\_a (data),

.wren\_a (wren),

.q\_a (sub\_wire0),

.aclr0 (1'b0),

.aclr1 (1'b0),

.address\_b (1'b1),

.addressstall\_a (1'b0),

.addressstall\_b (1'b0),

.byteena\_a (1'b1),

.byteena\_b (1'b1),

.clock1 (1'b1),

.clocken0 (1'b1),

.clocken1 (1'b1),

.clocken2 (1'b1),

.clocken3 (1'b1),

.data\_b (1'b1),

.eccstatus (),

.q\_b (),

.rden\_a (1'b1),

.rden\_b (1'b1),

.wren\_b (1'b0));

defparam

altsyncram\_component.clock\_enable\_input\_a = "BYPASS",

altsyncram\_component.clock\_enable\_output\_a = "BYPASS",

altsyncram\_component.intended\_device\_family = "Cyclone V",

altsyncram\_component.lpm\_hint = "ENABLE\_RUNTIME\_MOD=NO",

altsyncram\_component.lpm\_type = "altsyncram",

altsyncram\_component.numwords\_a = 32,

altsyncram\_component.operation\_mode = "SINGLE\_PORT",

altsyncram\_component.outdata\_aclr\_a = "NONE",

altsyncram\_component.outdata\_reg\_a = "CLOCK0",

altsyncram\_component.power\_up\_uninitialized = "FALSE",

altsyncram\_component.read\_during\_write\_mode\_port\_a = "NEW\_DATA\_NO\_NBE\_READ",

altsyncram\_component.widthad\_a = 5,

altsyncram\_component.width\_a = 4,

altsyncram\_component.width\_byteena\_a = 1;

endmodule

module hex(in2, in3, in4, in5, o2, o3, o4, o5, o6, o7, o8);

input in2;

input in3;

input in4;

input in5;

output o2;

output o3;

output o4;

output o5;

output o6;

output o7;

output o8;

hex0 u1(

.c3(in2),

.c2(in3),

.c1(in4),

.c0(in5),

.out1(o2)

);

hex1 u2(

.c3(in2),

.c2(in3),

.c1(in4),

.c0(in5),

.out2(o3)

);

hex2 u3(

.c3(in2),

.c2(in3),

.c1(in4),

.c0(in5),

.out3(o4)

);

hex3 u4(

.c3(in2),

.c2(in3),

.c1(in4),

.c0(in5),

.out4(o5)

);

hex4 u5(

.c3(in2),

.c2(in3),

.c1(in4),

.c0(in5),

.out5(o6)

);

hex5 u6(

.c3(in2),

.c2(in3),

.c1(in4),

.c0(in5),

.out6(o7)

);

hex6 u7(

.c3(in2),

.c2(in3),

.c1(in4),

.c0(in5),

.out7(o8)

);

endmodule

module hex0(c3, c2, c1, c0, out1);

input c3; //selected when s is 0

input c2; //selected when s is 1

input c1; //select signal

input c0;

output out1; //output

assign out1 = ~c3 & c2 & ~c1 & ~c0 | c3 & c2 & ~c1 & c0 | c3 & ~c2 & c1 & c0 | ~c3 & ~c2 & ~c1 & c0;

endmodule

module hex1(c3, c2, c1, c0, out2);

input c3; //selected when s is 0

input c2; //selected when s is 1

input c1; //select signal

input c0;

output out2; //output

assign out2 = c2 & c1 & ~c0 | c3 & c1 & c0 | ~c3 & c2 & ~c1 & c0 | c3 & c2 & ~c0;

endmodule

module hex2(c3, c2, c1, c0, out3);

input c3; //selected when s is 0

input c2; //selected when s is 1

input c1; //select signal

input c0;

output out3; //output

assign out3 = ~c3 & ~c2 & c1 & ~c0 | c3 & c2 & c1 | c3 & c2 & ~c0;

endmodule

module hex3(c3, c2, c1, c0, out4);

input c3; //selected when s is 0

input c2; //selected when s is 1

input c1; //select signal

input c0;

output out4; //output

assign out4 = ~c3 & c2 & ~c1 & ~c0 | ~c2 & ~c1 & c0 | c2 & c1 & c0 | c3 & ~c2 & c1 & ~c0;

endmodule

module hex4(c3, c2, c1, c0, out5);

input c3; //selected when s is 0

input c2; //selected when s is 1

input c1; //select signal

input c0;

output out5; //output

assign out5 = ~c3 & c2 & ~c1 | ~c2 & ~c1 & c0 | ~c3 & c0;

endmodule

module hex5(c3, c2, c1, c0, out6);

input c3; //selected when s is 0

input c2; //selected when s is 1

input c1; //select signal

input c0;

output out6; //output

assign out6 = ~c3 & ~c2 & c0 | ~c3 & ~c2 & c1 | ~c3 & c1 & c0 | c3 & c2 & ~c1 & c0;

endmodule

图片包含 文字

描述已自动生成module hex6(c3, c2, c1, c0, out7);

input c3; //selected when s is 0

input c2; //selected when s is 1

input c1; //select signal

input c0;

output out7; //output

assign out7 = ~c3 & ~c2 & ~c1 | c3 & c2 & ~c1 & ~c0 | ~c3 & c2 & c1 & c0;

endmodule

11. Schematic.

**Part II**

1. Verilog:

module datapath(clk, temp\_x, temp\_y, temp\_c, reset\_n, go, color\_in, position, x\_out, y\_out, color\_out);

input clk, temp\_x, temp\_y, temp\_c, reset\_n, go;

input [2:0] color\_in;

input [6:0] position;

output [7:0] x\_out;

output [6:0] y\_out;

output [2:0] color\_out;

reg [2:0] count\_x, count\_y;

reg [7:0] real\_x;

reg [6:0] real\_y;

reg [2:0] color;

// registors for x, y and color

always @(posedge clk) begin

if (!reset\_n) begin

real\_x <= 8'b0;

real\_y <= 7'b0;

color <= 3'b0;

end

else begin

if (temp\_x)

real\_x <= {1'b0, position};

if (temp\_y)

real\_y <= position;

if (temp\_c)

color <= color\_in;

end

end

// counter for x

always @(posedge clk) begin

if (!reset\_n)

count\_x <= 2'b00;

else if (go) begin

if (count\_x == 2'b11)

count\_x <= 2'b00;

else begin

count\_x <= count\_x + 1'b1;

end

end

end

// counter for y

always @(posedge clk) begin

if (!reset\_n)

count\_y <= 2'b00;

else if (go && (count\_x == 2'b11)) begin

if (count\_y != 2'b11)

count\_y <= count\_y + 1'b1;

else

count\_y <= 2'b00;

end

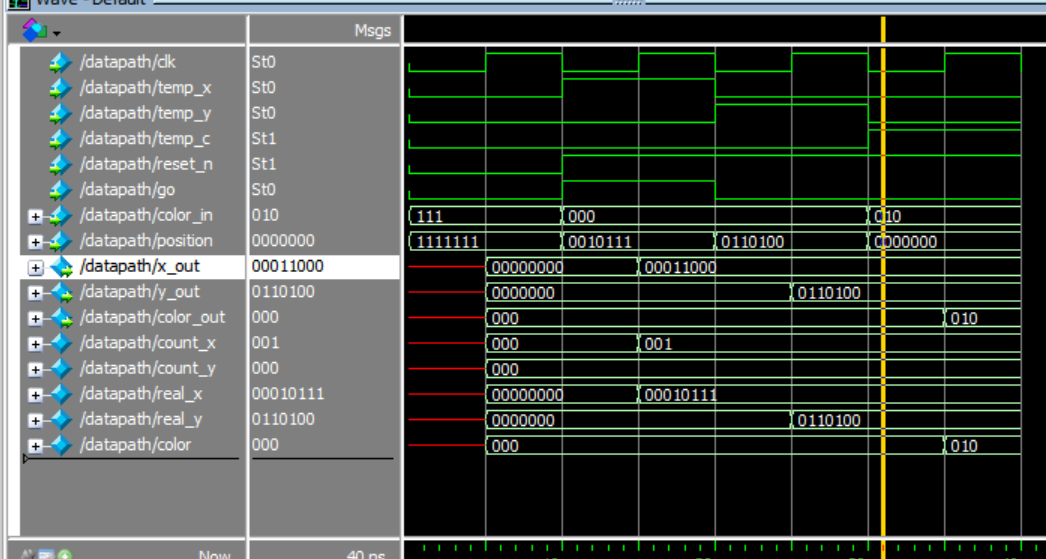
end

assign x\_out = real\_x + count\_x;

assign y\_out = real\_y + count\_y;

assign color\_out = color;

endmodule

Simulation:

State Diagram

图片包含 文字

描述已自动生成

State Table

图片包含 文字

描述已自动生成

Verilog

module control(clk, reset\_n, ld, draw, temp\_x, temp\_y, temp\_c, go);

input clk, reset\_n, ld, draw;

output reg temp\_x, temp\_y, temp\_c, go;

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

localparam

Load\_x = 3'd0,

Load\_x\_wait = 3'd1,

Load\_y = 3'd2,

Load\_y\_wait = 3'd3,

Load\_c = 3'd4,

Load\_c\_wait = 3'd5,

Draw = 3'd6;

always @(\*) begin

case (current\_state)

Load\_x: next\_state = ld ? Load\_x\_wait : Load\_x;

Load\_x\_wait: next\_state = ld ? Load\_x\_wait : Load\_y;

Load\_y: next\_state = ld ? Load\_y\_wait : Load\_y;

Load\_y\_wait: next\_state = ld ? Load\_y\_wait : Load\_c;

Load\_c: next\_state = draw ? Load\_c\_wait : Load\_c;

Load\_c\_wait: next\_state = draw ? Load\_c\_wait : Draw;

Draw: next\_state = ld ? Load\_x : Draw;

endcase

end

always @(\*) begin

temp\_x = 1'b0;

temp\_y = 1'b0;

temp\_c = 1'b0;

go = 1'b0;

case (current\_state)

Load\_x: begin

temp\_x = 1;

end

Load\_y: begin

temp\_y = 1;

end

Load\_c: begin

temp\_c = 1;

end

Draw: begin

go = 1;

end

endcase

end

always @(posedge clk) begin

if (!reset\_n)

current\_state <= Load\_x;

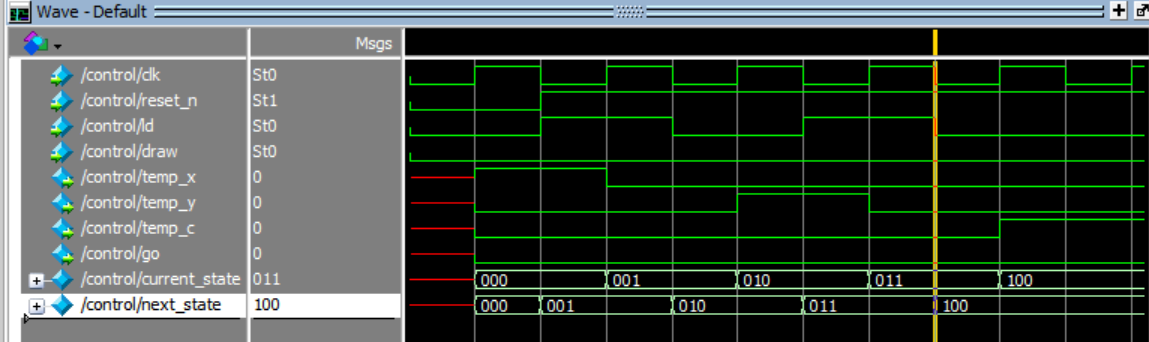
else

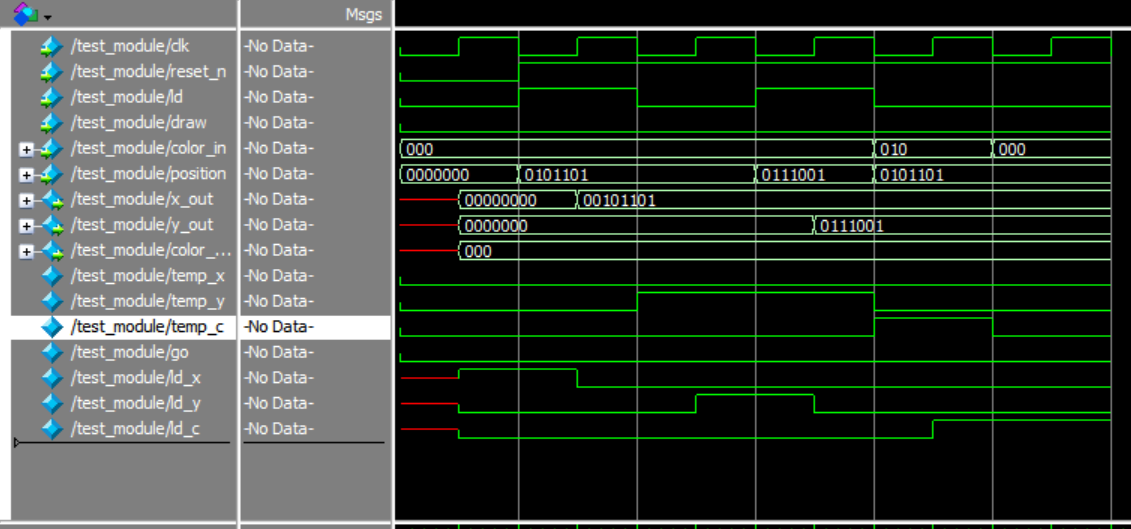
current\_state <= next\_state;

end

endmodule

Simulation



1. Simulation

**Part III**

Verilog

module datapath(enable,clock,ld\_c,colour,reset\_n,X,Y,colour\_out);

input enable,clock,reset\_n,ld\_c;

input [2:0] colour;

output[7:0] X,Y;

output[2:0] colour\_out;

wire enable\_fc;

wire enable\_xy;

wire[3:0] c1;

wire signal\_x,signal\_y;

wire[7:0] x\_in,y\_in;

wire[2:0] colour\_1;

x\_counter x\_c(

.x\_in(x\_in),

.clock(clock),

.reset\_n(reset\_n),

.enable(enable\_xy),

.x\_out(x\_in)

);

y\_counter y\_c(

.y\_in(y\_in),

.clock(clock),

.reset\_n(reset\_n),

.enable(enable\_xy),

.y\_out(y\_in)

);

delay\_counter dc1(

.clock(clock),

.reset\_n(reset\_n),

.enable(enable),

.enable\_fc(enable\_fc)

);

frame\_counter fc2(

.clock(clock),

.reset\_n(reset\_n),

.enable(enable\_fc),

.enable\_xy(enable\_xy),

.colour\_1(colour\_1),

.colour(colour)

);

draw dr1(x\_in,y\_in,colour\_1,ld\_c,clock,reset\_n,enable,X,Y,colour\_out);

endmodule

Simulation

图片包含 时钟

描述已自动生成

2.

State Diagram

图片包含 文字

描述已自动生成

Verilog

module control(clock,reset\_n,go,enable,ld\_c,plot);

input clock,reset\_n,go;

output reg enable,ld\_c,plot;

reg [3:0] current\_state, next\_state;

localparam S\_LOAD\_C = 4'd0,

S\_LOAD\_C\_WAIT = 4'd1,

S\_CYCLE\_0 = 4'd2;

always@(\*)

begin: state\_table

case (current\_state)

S\_LOAD\_C: next\_state = go ? S\_LOAD\_C\_WAIT : S\_LOAD\_C;

S\_LOAD\_C\_WAIT: next\_state = go ? S\_LOAD\_C\_WAIT : S\_CYCLE\_0;

S\_CYCLE\_0: next\_state = S\_CYCLE\_0;

default: next\_state = S\_LOAD\_C;

endcase

end

always@(\*)

begin: enable\_signals

// By default make all our signals 0

ld\_c = 1'b0;

enable = 1'b0;

plot = 1'b0;

case(current\_state)

S\_LOAD\_C:begin

end

S\_CYCLE\_0:begin

ld\_c = 1'b1;

enable = 1'b1;

plot = 1'b1;

end

endcase

end

always@(posedge clock)

begin: state\_FFs

if(!reset\_n)

current\_state <= S\_LOAD\_C;

else

current\_state <= next\_state;

end

endmodule

Simulation

图片包含 设备, 仪表, 时钟

描述已自动生成