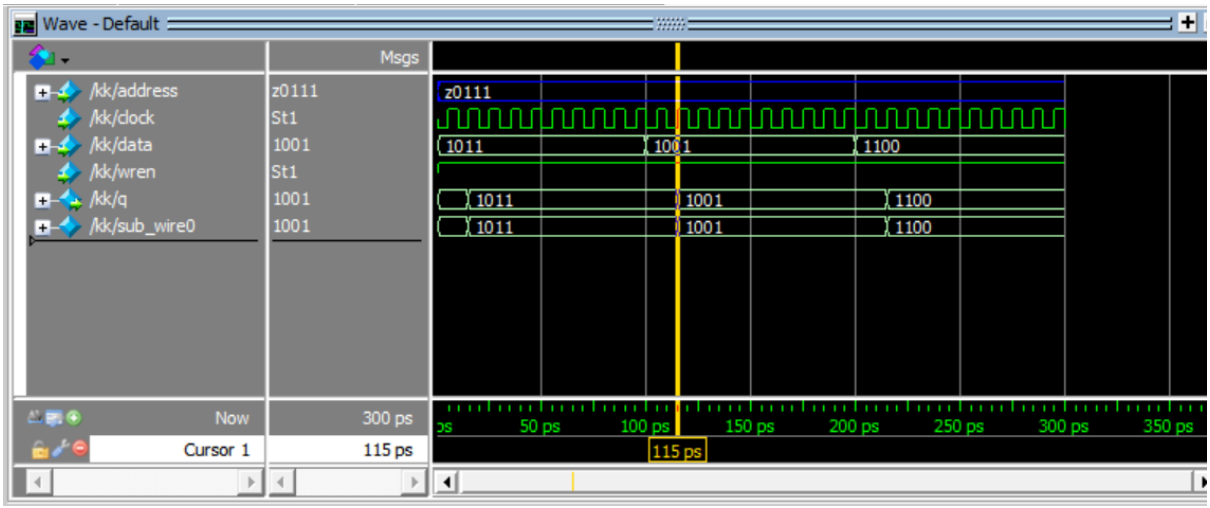
Prelab for Lab7

Xiaoyu Zhou, 1004081147

Part I

9. Simulation



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

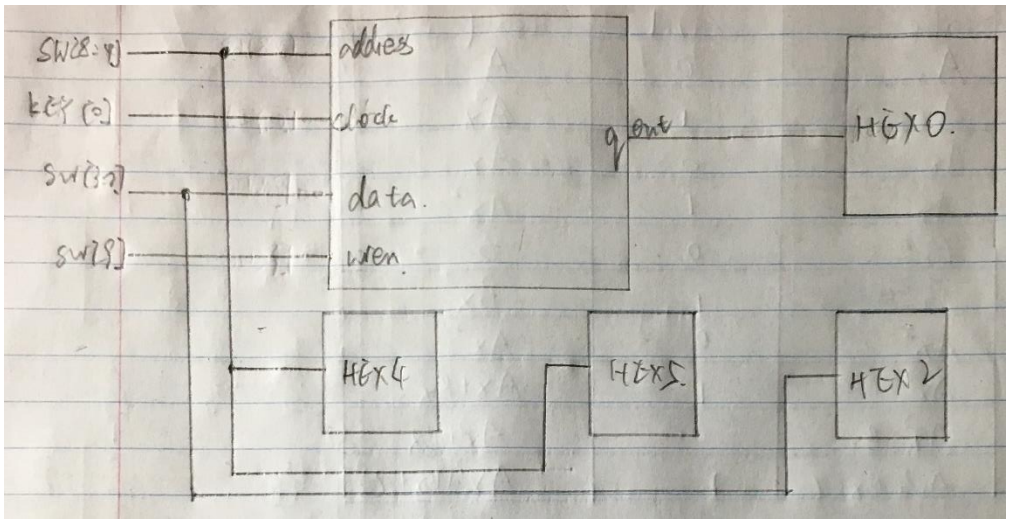
```
module top (SW, KEY, HEX0, HEX2, HEX4, HEX5);
	input [9:0] SW;
		input [3:0] KEY;
		output [6:0] HEX0;
		output [6:0] HEX2;
		output [6:0] HEX4;
		output [6:0] HEX5;
		wire out;

		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

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.

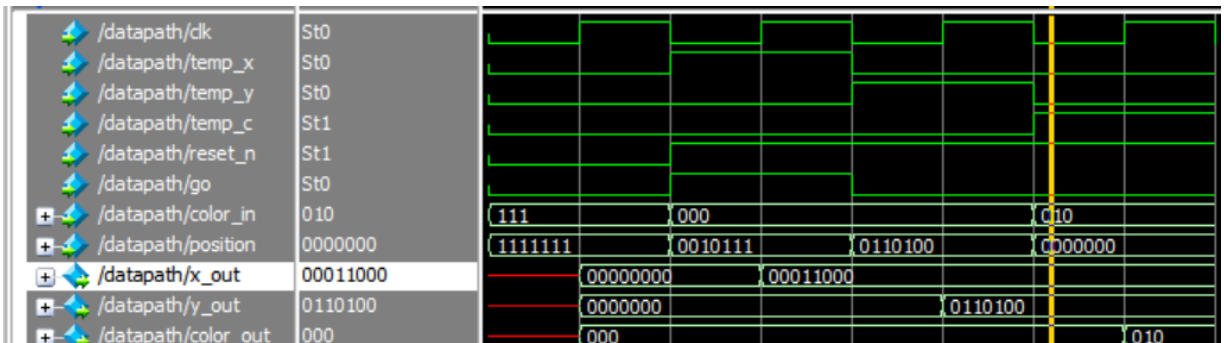


Part II

1. Verilog:

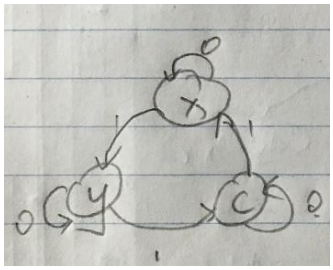
```
module datapath(clk, temp_x, temp_y, temp_c, reset_n, go, color_in, position, x_out, y_out, color_out);

	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:

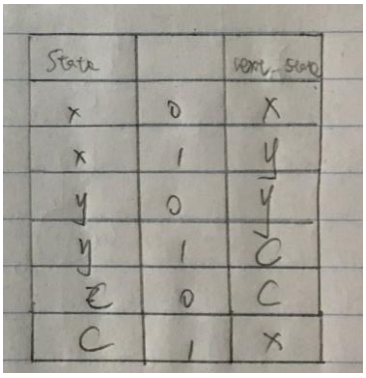


2.

State Diagram



State Table

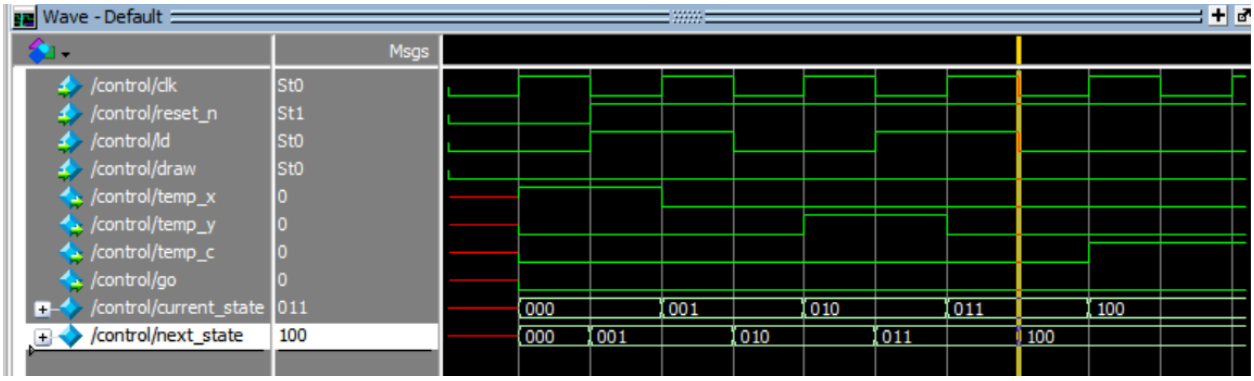


Verilog

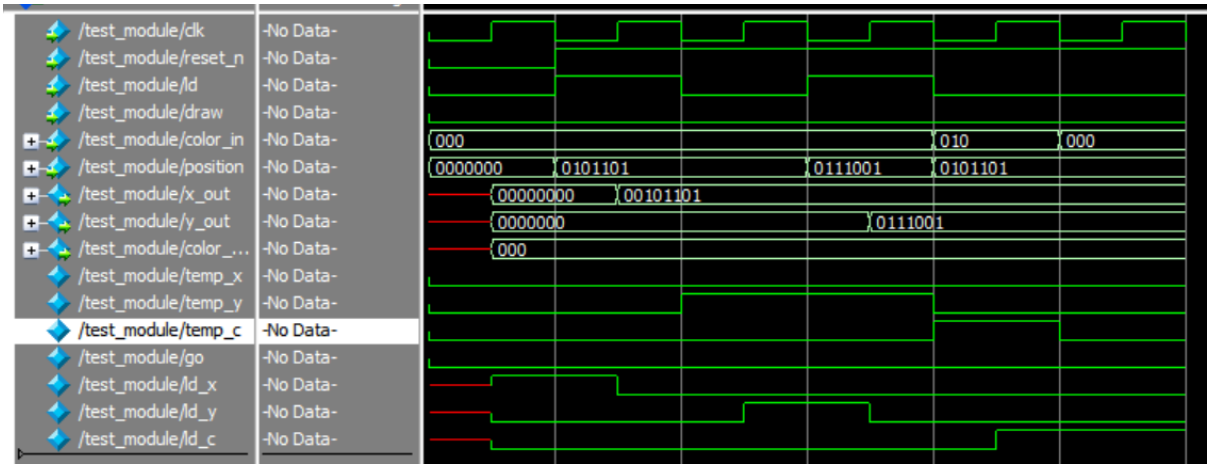
```
module control(clk, reset_n, ld, draw, temp_x, temp_y, temp_c, go);
	input clk, reset_n, ld, draw;
	output reg temp_x, temp_y, temp_c, go;
	reg [2:0] current_state, next_state;

	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



3. Simulation

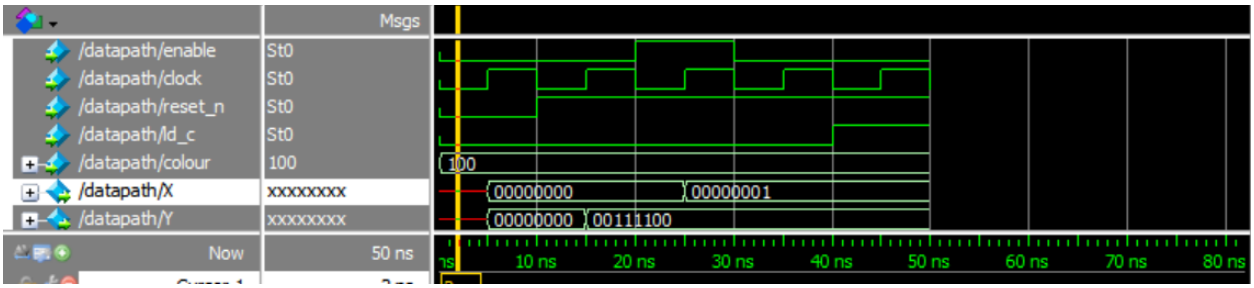


Part III

Verilog

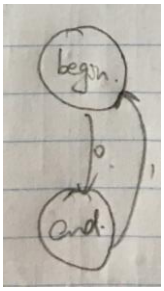
```
module datapath(enable,clock,ld_c,colour,reset_n,X,Y,colour_out);
	input enable,clock,reset_n,ld_c;
	input [2:0] colour;
	output[7:0] X,Y;
	output[2:0] colour_out;

	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



2.

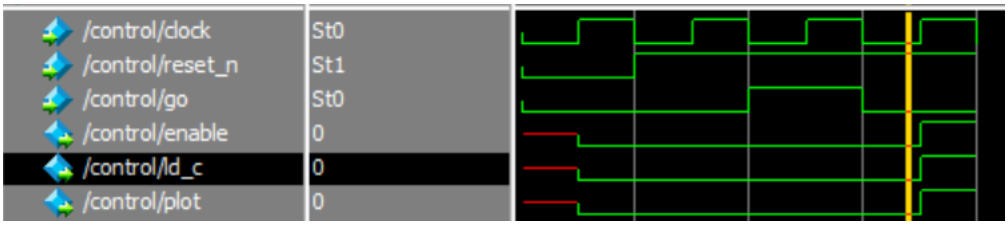
State Diagram



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


/control/clock
St0
/control/reset_n
St1
/control/go
St0
/control/enable
0
/control/ld_c
0
/control/plot
0