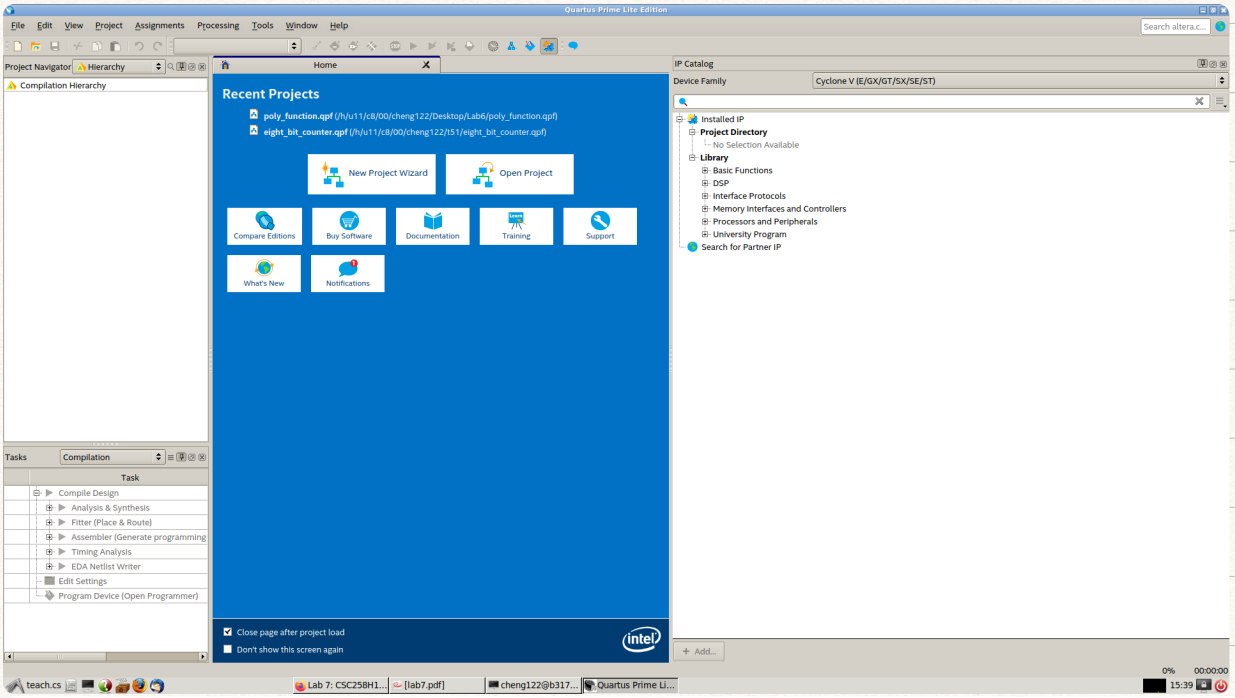
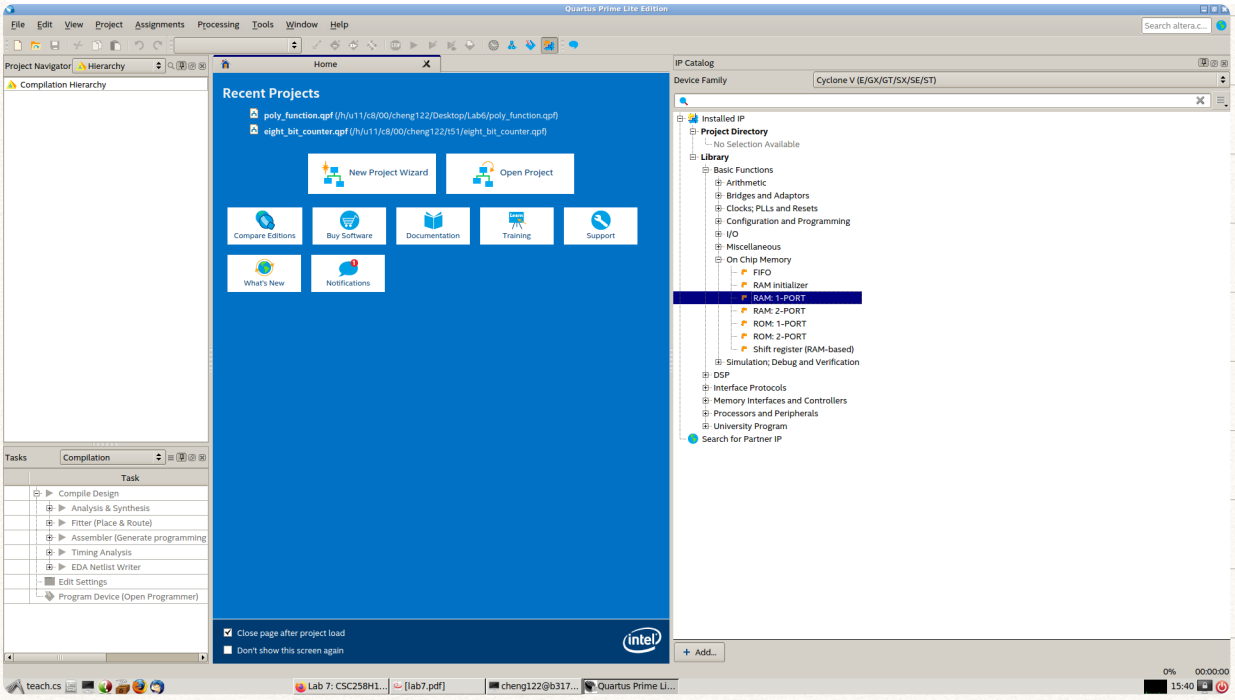
Prelab 7

Part I

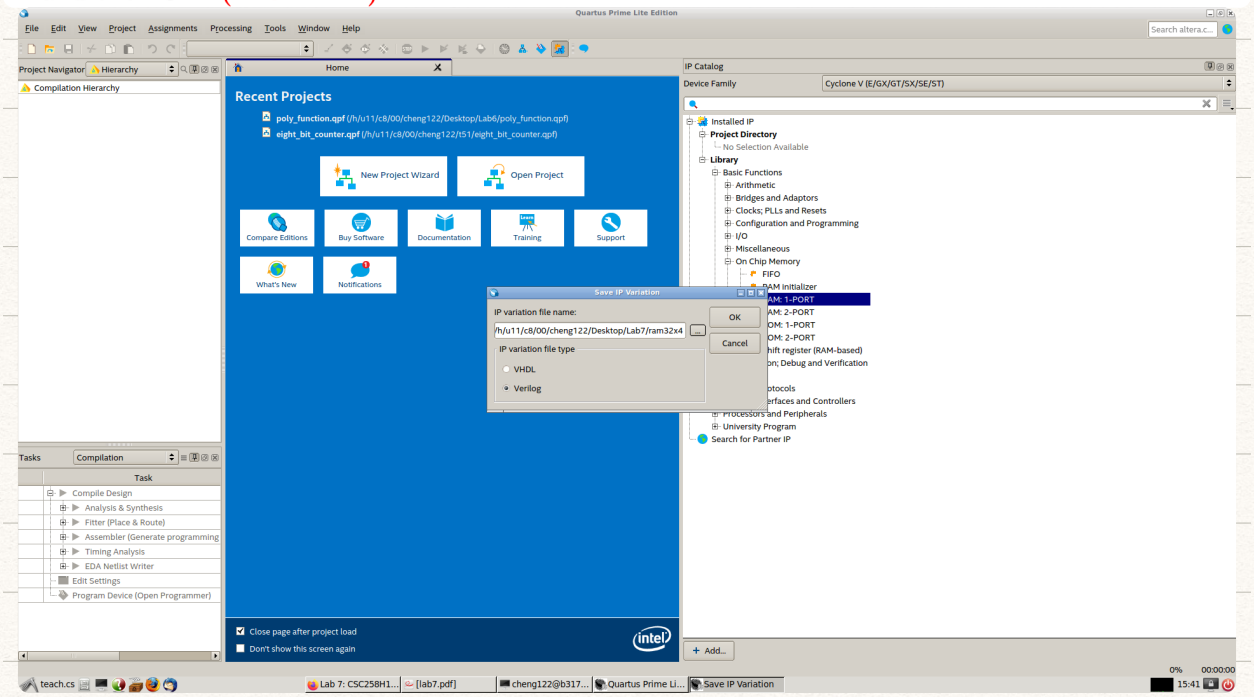
2. You will now create a memory module that you can include into your design. First, select Tools−>IP Catalog. Note that the IP Catalog usually shows up as a pane on the right side of the Quartus Window, and may have already been there by default. **(PRELAB)**



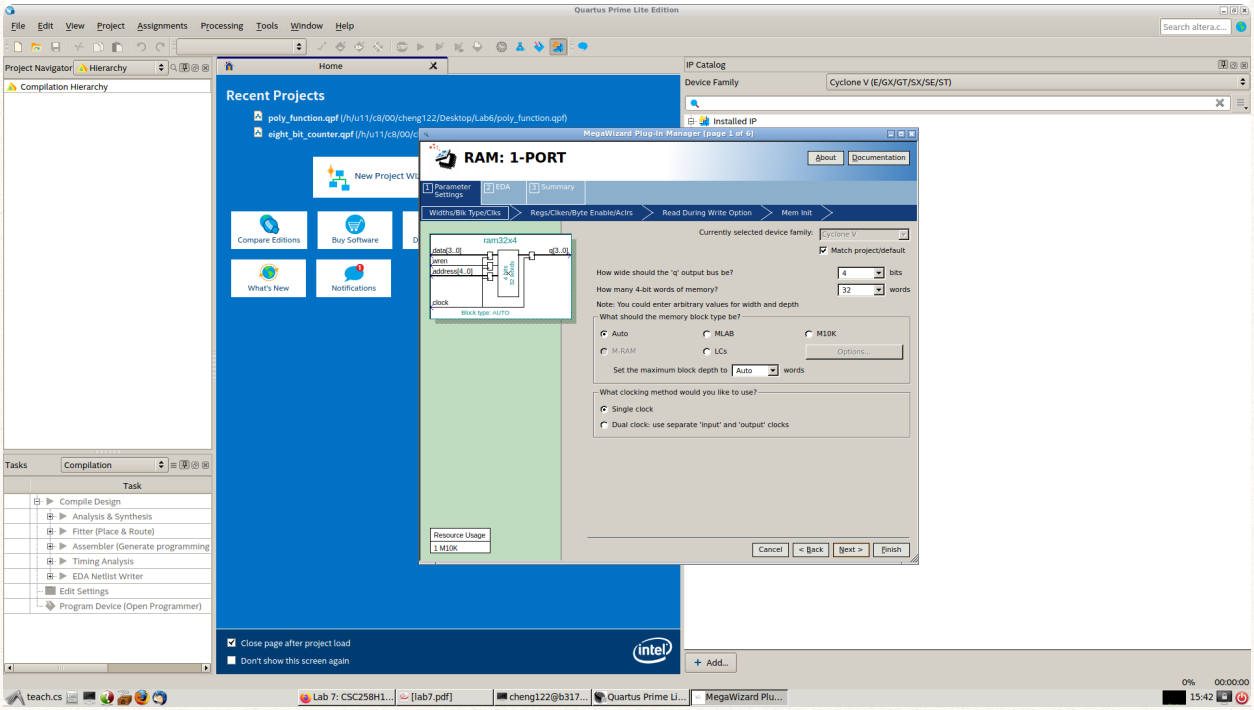
3. In the IP Catalog pane, expand Installed IP−>Library−>Basic Functions−>On Chip Memory, then double-click on *RAM:1-PORT*. **(PRELAB)**



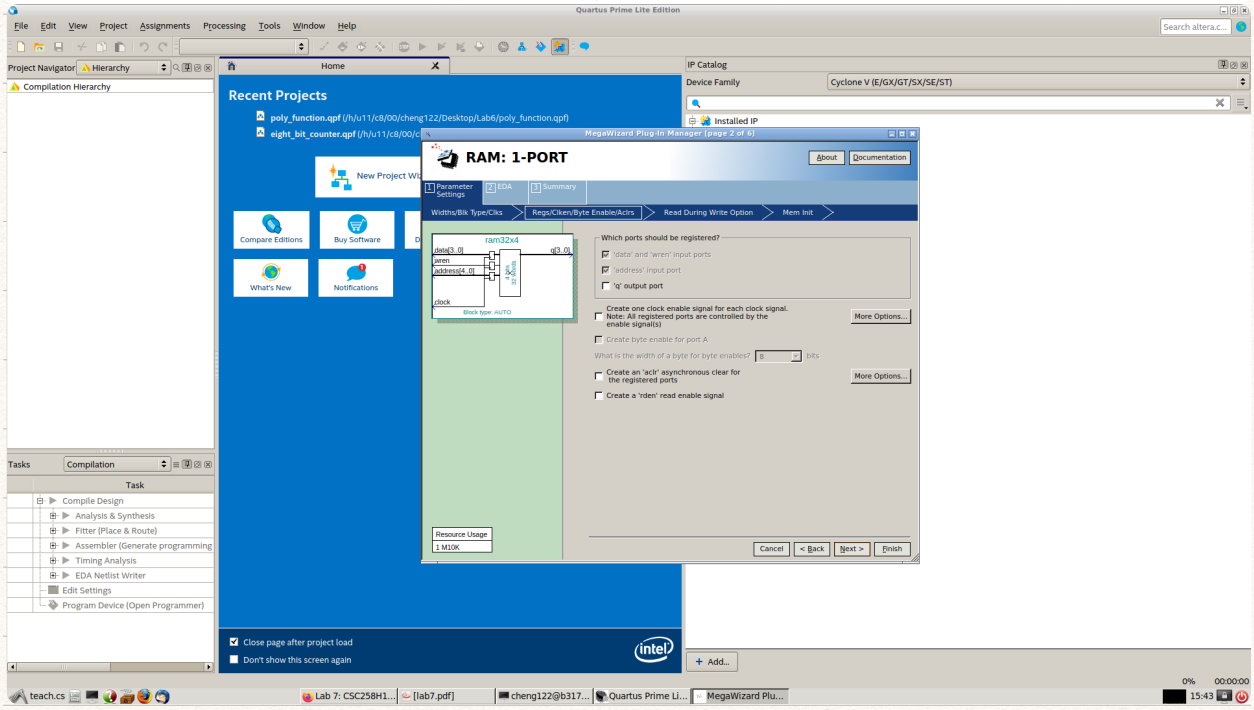
4. Browse to the folder or directory where you want to build your project. This is where the file for the memory module will be created. Call the file `ram32x4.v`. Choose the IP variation to be Verilog and click OK. **(PRELAB)**



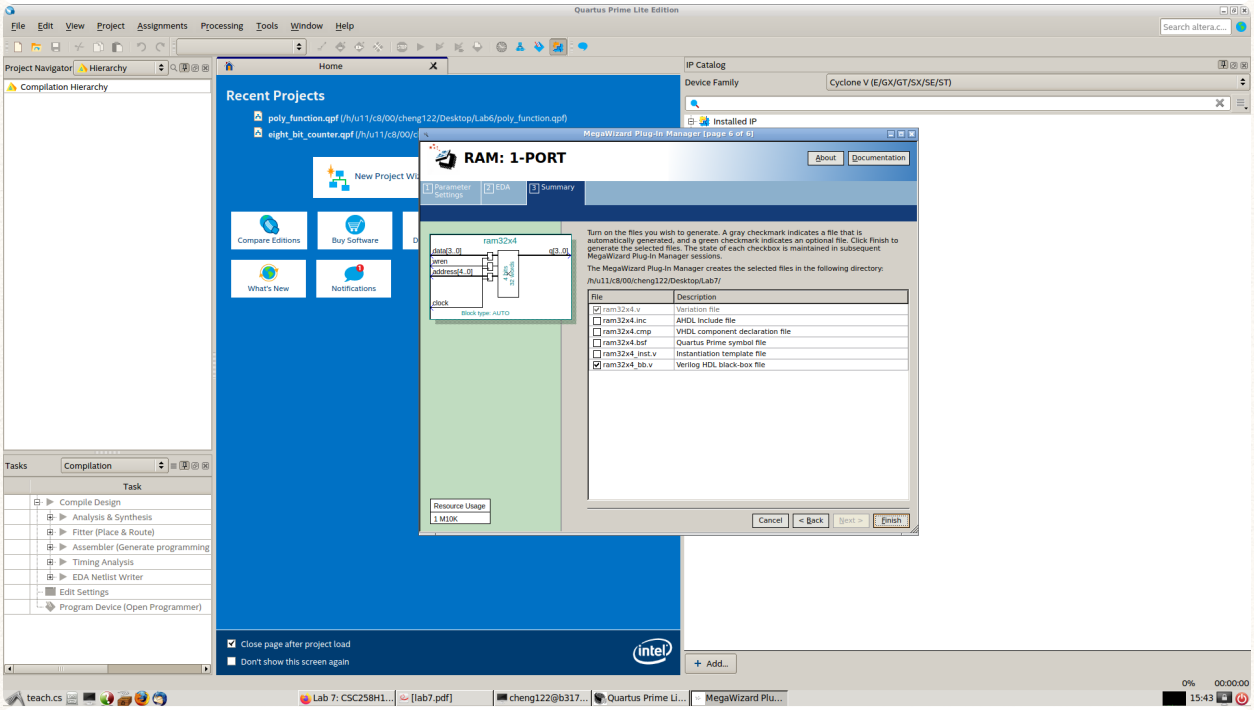
5. Select a 4-bit wide (width of 'q' output bus) memory with 32 words. Leave the memory block type as *Auto* and use a *Single Clock*. Observe how the memory block diagram on the left side of the window changes automatically as you change different parameters. Pay particular attention to the fact that registers that hold the address, data and write enable signals are included in the block diagram. This means they will be part of the memory module, and you do **NOT** need to add extra registers outside the memory module. Click Next. **(PRELAB)**



6. **Deselect** q as a registered port. **(PRELAB)**



7. Click Finish and Finish again to generate the new Verilog file, `ram32x4.v`. **(PRELAB)**

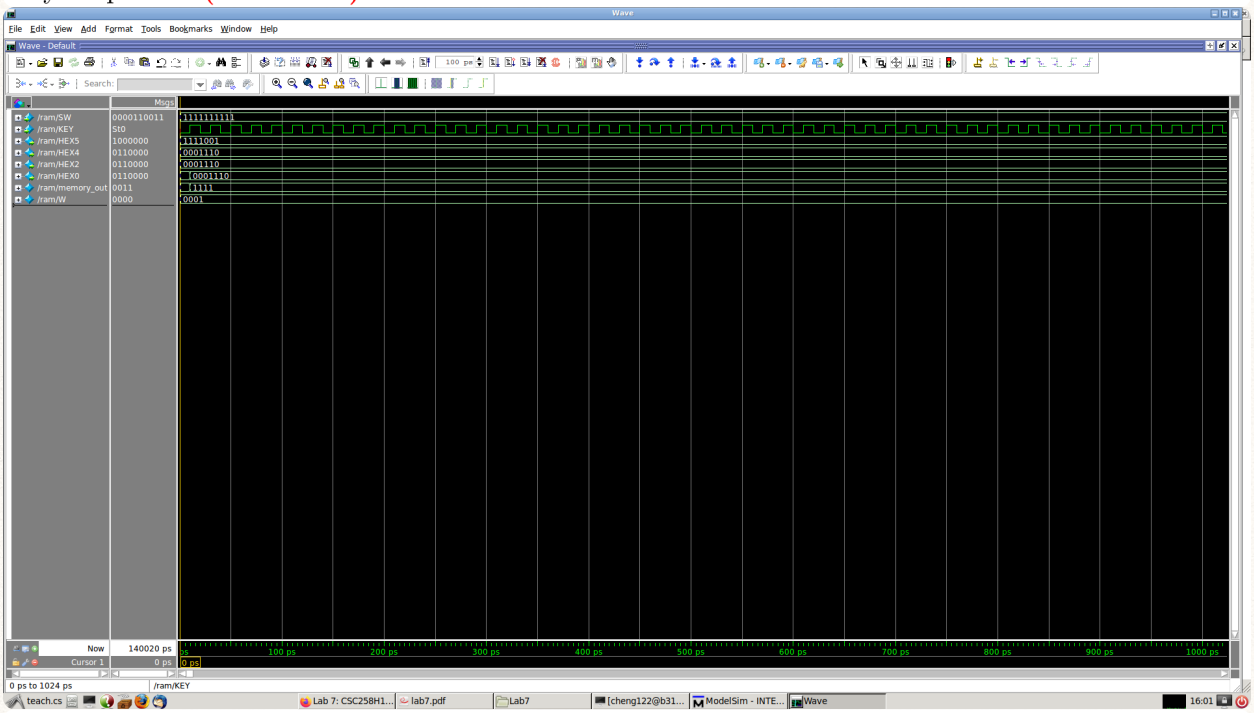


8. Examine the newly created Verilog file. Observe that it declares a module with the required ports as shown in Figure 1. You can now instantiate the module into any design. **(PRELAB)**

see ram32x4.v

I've examined the code

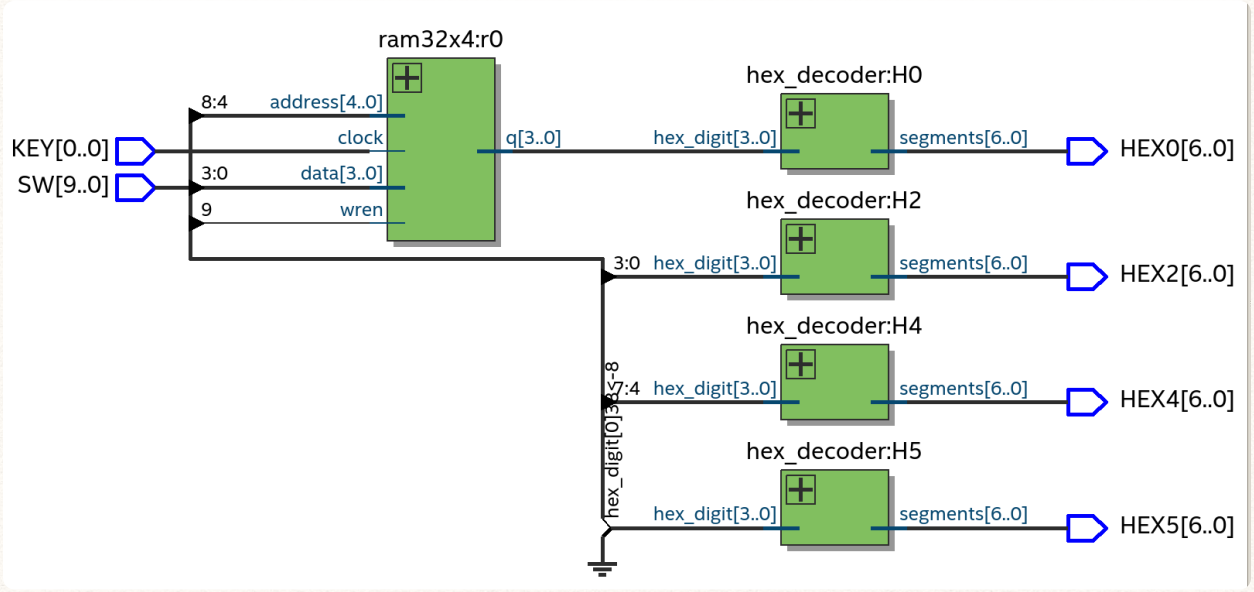
9. Simulate the created memory module using ModelSim for a variety of input settings, ensuring the output waveforms are correct. Show that you can read and write to the memory at several locations. You must include screenshots to show that you performed these simulations as part of your prelab. **(PRELAB)**



10. Instantiate the *ram32x4* module into a top-level Verilog module that connects to the inputs and outputs in the following way: Connect SW[3:0] to the data inputs, SW[8:4] to the address inputs, SW[9] to the Write Enable input and use KEY[0] as the clock input. Show the address on HEX5 and HEX4, the input data on HEX2 and the data output of the memory on HEX0. **(PRELAB)**

see ram.v

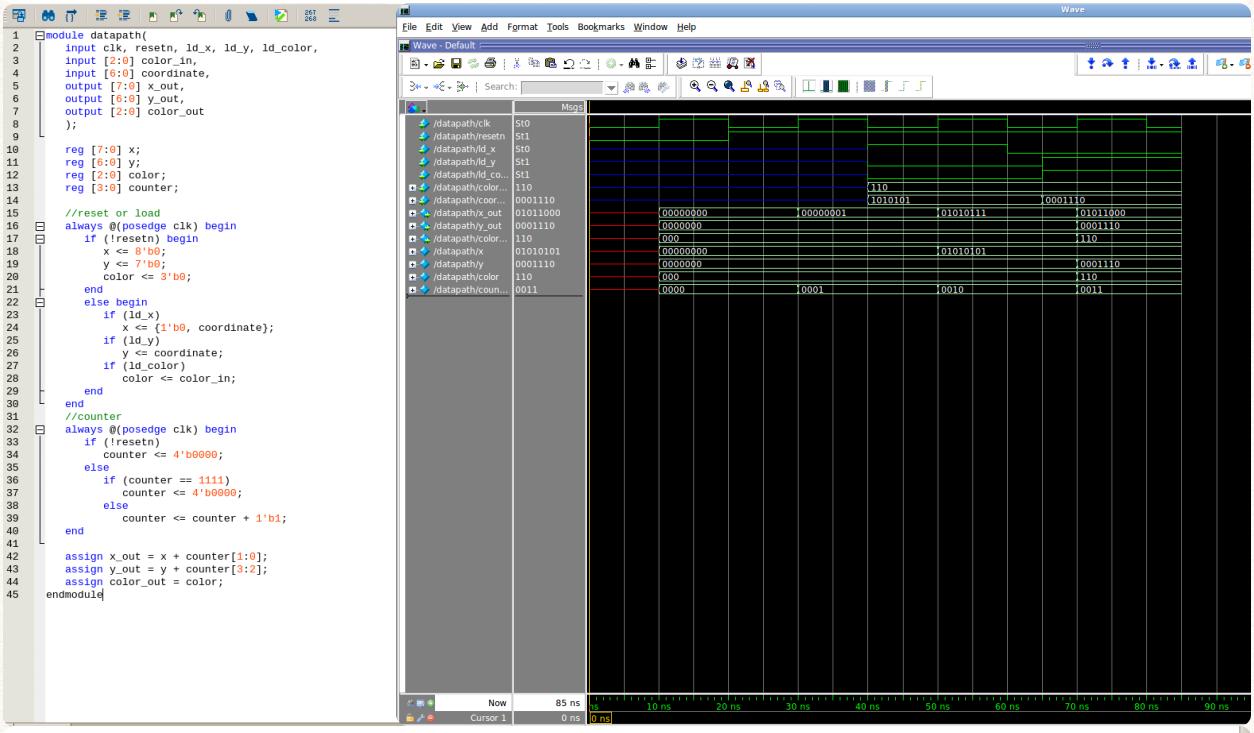
11. Draw a schematic describing the circuit as part of your preparation. **(PRELAB)**

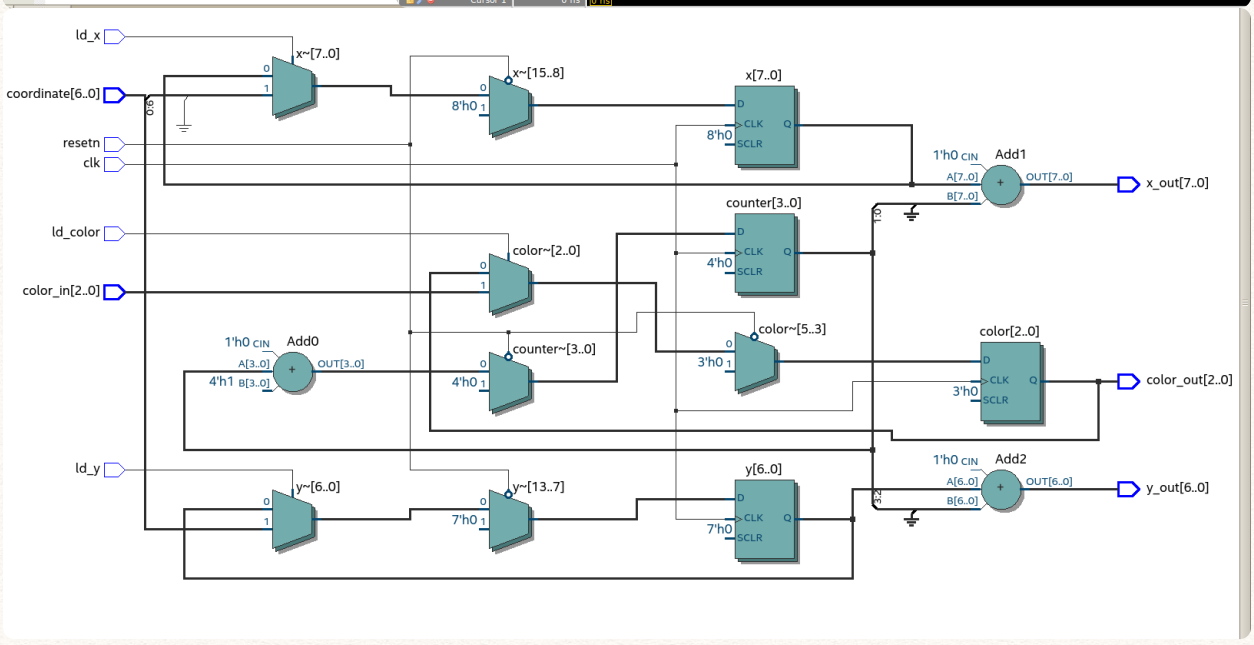


Part II

1. Design (draw the schematic and write Verilog code by adding to the provided skeleton code in `lab7-part2.zip`) and simulate a datapath that implements the required functionality. Note that the provided skeleton code incorporates the VGA adapter. Your simulation for this part should only include the datapath only, and not the FSM or the VGA adapter. Include schematics, Verilog code, and screenshots of simulation output in your prelab. **(PRELAB)**

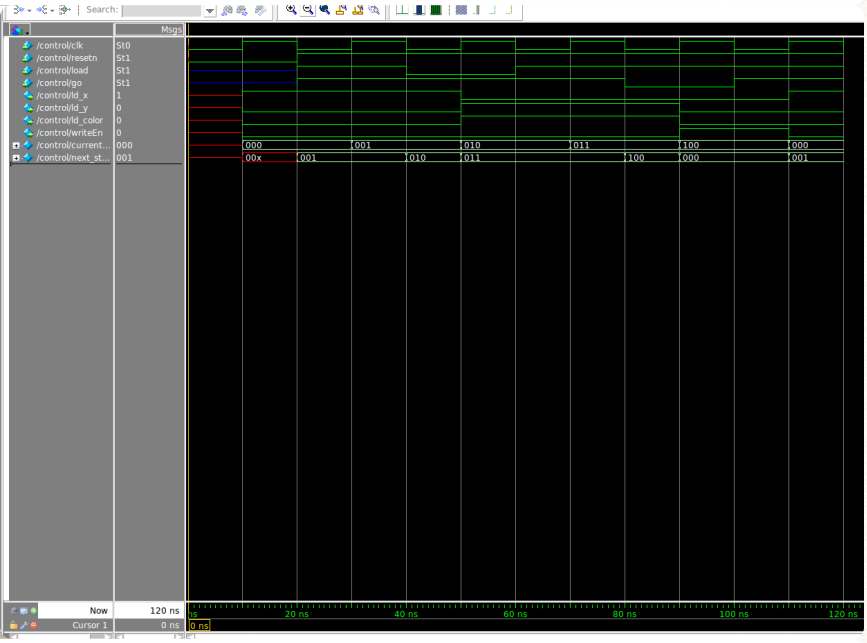
```
module datapath(
    input clk, resetn, ld_x, ld_y, ld_color,
    input [2:0] color_in,
    input [6:0] coordinate,
    output [7:0] x_out,
    output [6:0] y_out,
    output [2:0] color_out
    );

    reg [7:0] x;
    reg [6:0] y;
    reg [2:0] color;
    reg [3:0] counter;

    //reset or load
    always @(posedge clk) begin
        if (!resetn) begin
            x <= 8'b0;
            y <= 7'b0;
            color <= 3'b0;
        end
        else begin
            if (ld_x)
                x <= {1'b0, coordinate};
            if (ld_y)
                y <= coordinate;
            if (ld_color)
                color <= color_in;
        end
    end
    //counter
    always @(posedge clk) begin
        if (!resetn)
            counter <= 4'b0000;
        else
            if (counter == 1111)
                counter <= 4'b0000;
            else
                counter <= counter + 1'b1;
    end

    assign x_out = x + counter[1:0];
    assign y_out = y + counter[3:2];
    assign color_out = color;
endmodule
```

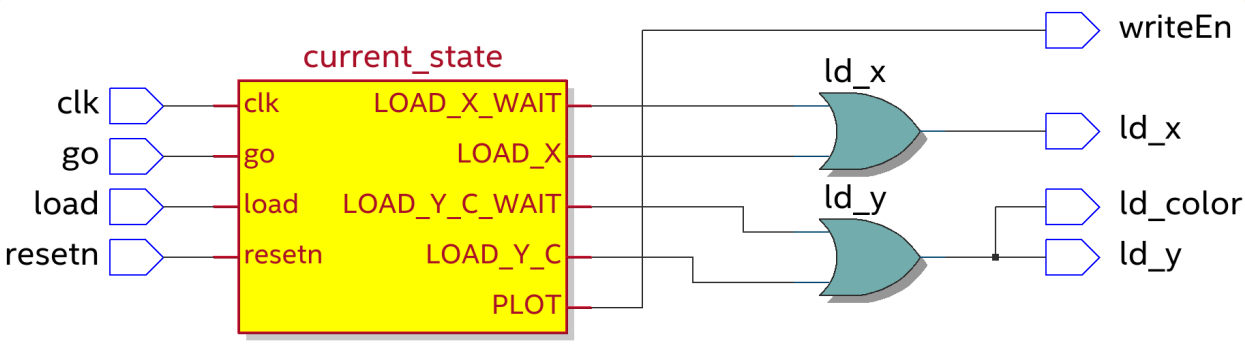


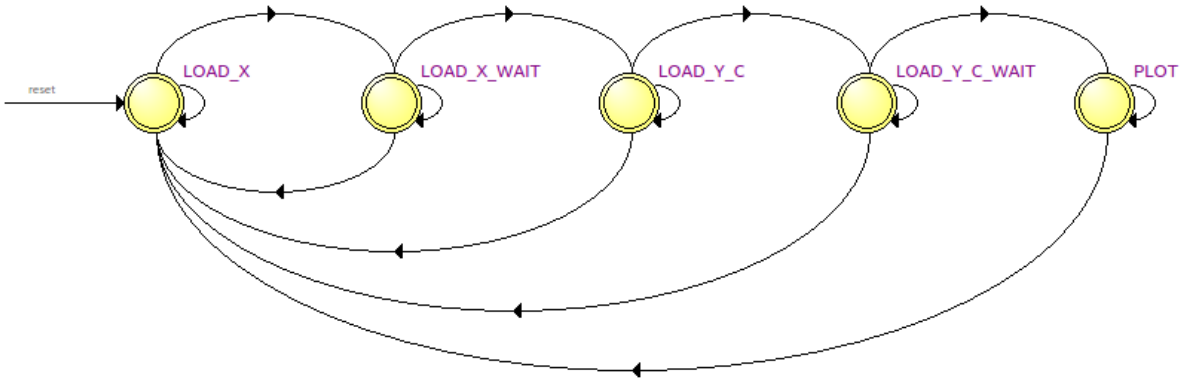


2. Design (draw a state diagram, state table, and write Verilog code by adding to the provided skeleton code) to implement and simulate an FSM that controls the implemented datapath. Your simulation for this part should only include the FSM. **(PRELAB)**

```
module control(
    input clk, resetn, load, go,
    output reg ld_x, ld_y, ld_color, writeEn);
    reg [2:0] current_state, next_state;

    localparam  LOAD_X = 3'b000,
                LOAD_X_WAIT = 3'b001,
                LOAD_Y_C = 3'b010,
                LOAD_Y_C_WAIT = 3'b011,
                PLOT = 3'b100;

    always @(posedge clk) begin   //reset
        if (!resetn)
            current_state <= LOAD_X;
        else
            current_state <= next_state;
    end

    always @(*) //state table
    begin: state_table
        case (current_state)
            LOAD_X: next_state = load ? LOAD_X_WAIT : LOAD_X;
            LOAD_X_WAIT: next_state = load ? LOAD_X_WAIT : LOAD_Y_C
            LOAD_Y_C: next_state = go ? LOAD_Y_C_WAIT : LOAD_Y_C;
            LOAD_Y_C_WAIT: next_state = go ? LOAD_Y_C_WAIT : PLOT;
            PLOT: next_state = load ? LOAD_X : PLOT;
            default: next_state = LOAD_X;
        endcase
    end

    always @(*)
    begin
        ld_x = 1'b0;
        ld_y = 1'b0;
        ld_color = 1'b0;
        writeEn = 0;
        case (current_state)
            LOAD_X: ld_x = 1;
            LOAD_X_WAIT: ld_x = 1;
            LOAD_Y_C:
                begin
                    ld_x = 0;
                    ld_y = 1;
                    ld_color = 1;
                end
            LOAD_Y_C_WAIT:
                begin
                    ld_x = 0;
                    ld_y = 1;
                    ld_color = 1;
                end
            PLOT: writeEn = 1;
        endcase
    end
endmodule
```







3. Since the VGA adapter connects to an external circuit whose details you may not fully understand, it is not straightforward to simulate your entire top-level design. You should simulate the combination of the FSM and datapath circuits (and not including the VGA adapter). You should then verify that the outputs from the datapath (inputs to the VGA adapter) are correct. That is, your prelab should include a simulation that demonstrates that all of the pixel writes to the vga_adapter module work as anticipated. **(PRELAB)**

```
module combo(
    input clk, resetn, load, go,
    input [2:0] color_in,
    input [6:0] coordinate,
    output [7:0] x_out,
    output [6:0] y_out,
    output [2:0] color_out
    );

    wire ld_x, ld_y, ld_color, writeEn;

    control c0(
        .clk(clk),
        .resetn(resetn),
        .load(load),
        .go(go),
        .ld_x(ld_x),
        .ld_y(ld_y),
        .ld_color(ld_color),
        .writeEn(writeEn)
    );

    datapath d0(
        .clk(clk),
        .color_in(color_in[2:0]),
        .resetn(resetn),
        .ld_x(ld_x),
        .ld_y(ld_y),
        .ld_color(ld_color),
        .coordinate(coordinate[6:0]),
        .x_out(x_out),
        .y_out(x_out),
        .color_out(color_out)
    );

endmodule
```