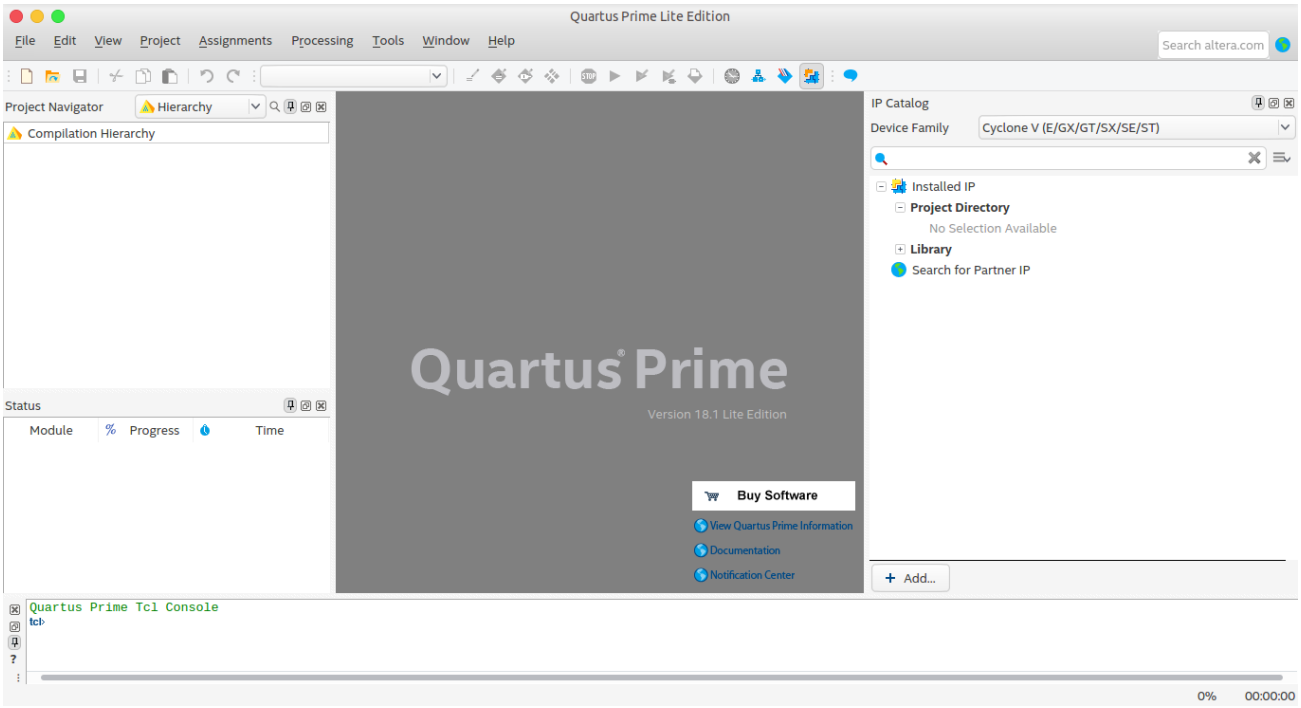
CSC258H1: Pre-lab Exercise for Lab 7

Wang, Weiqing

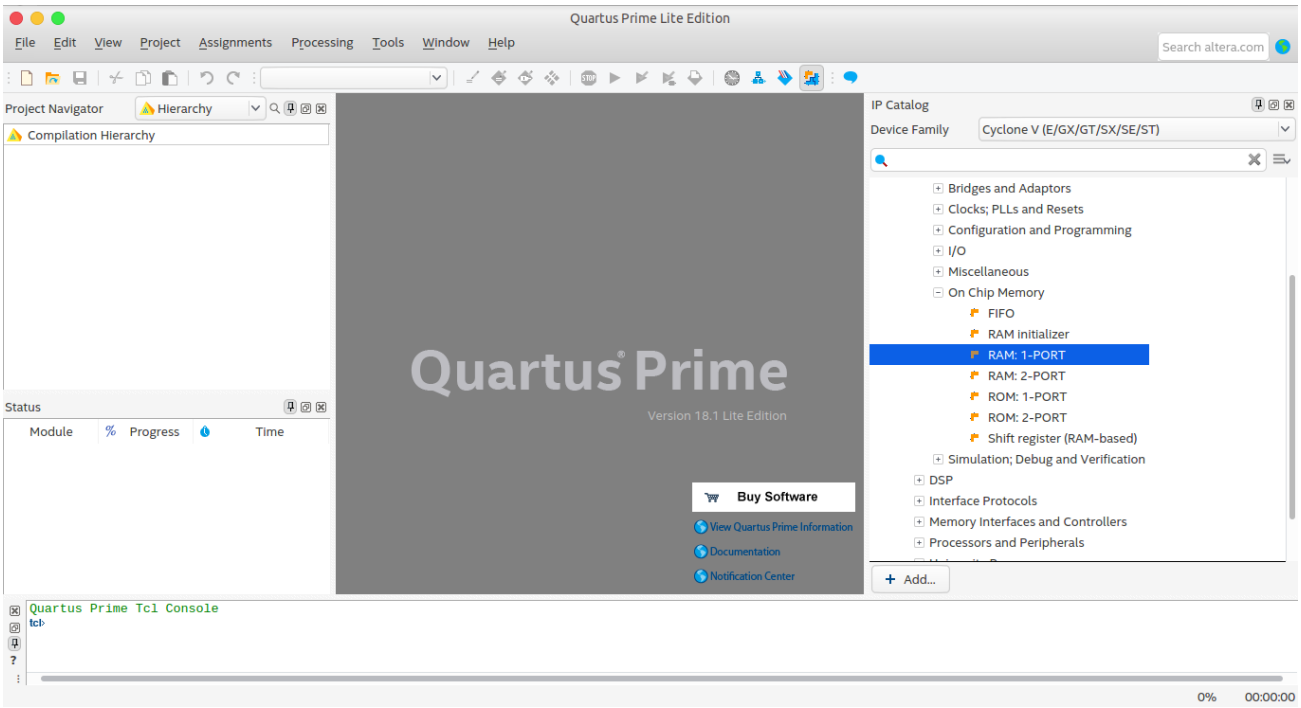
Due: March 11th, 2018 before 12 a.m.

1 Part I

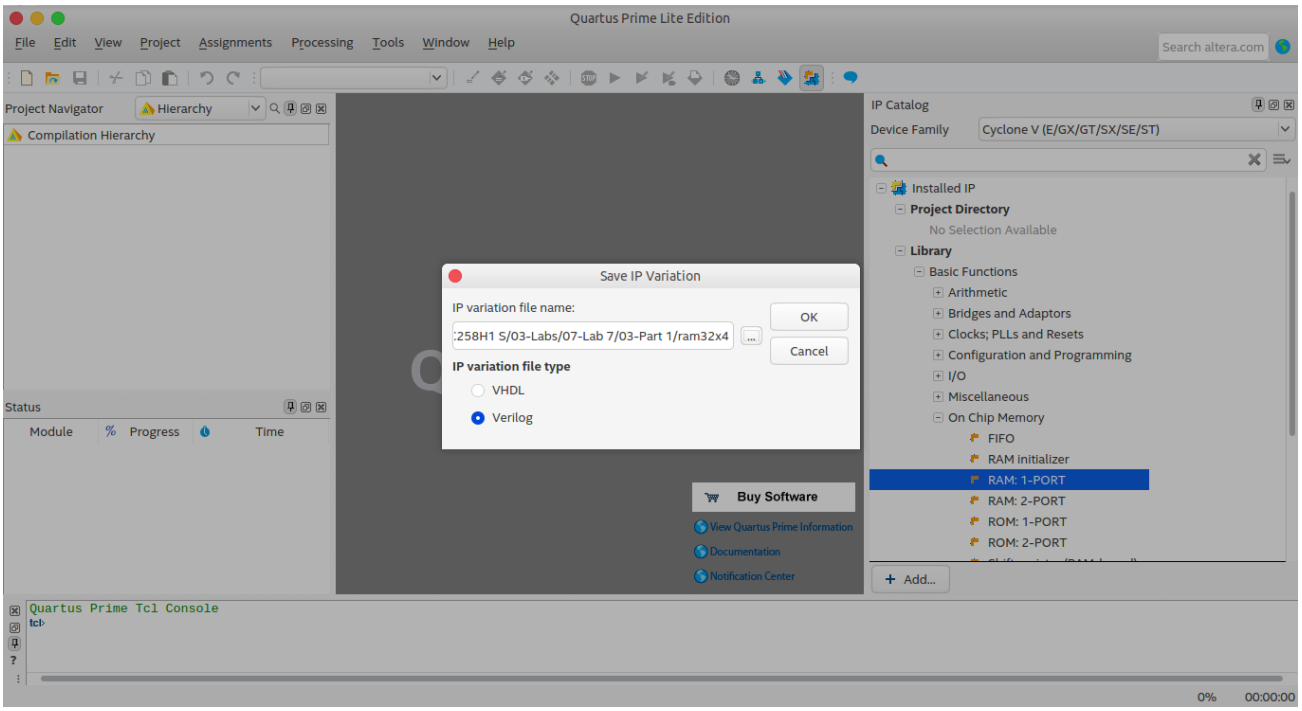
1. You will now create a memory module that you can include into your design. First, select Tools → IP Catalog. Note that the IP Catalog usually shows up as a pane on the right side of the Quartus Window, and may have already been there by default.



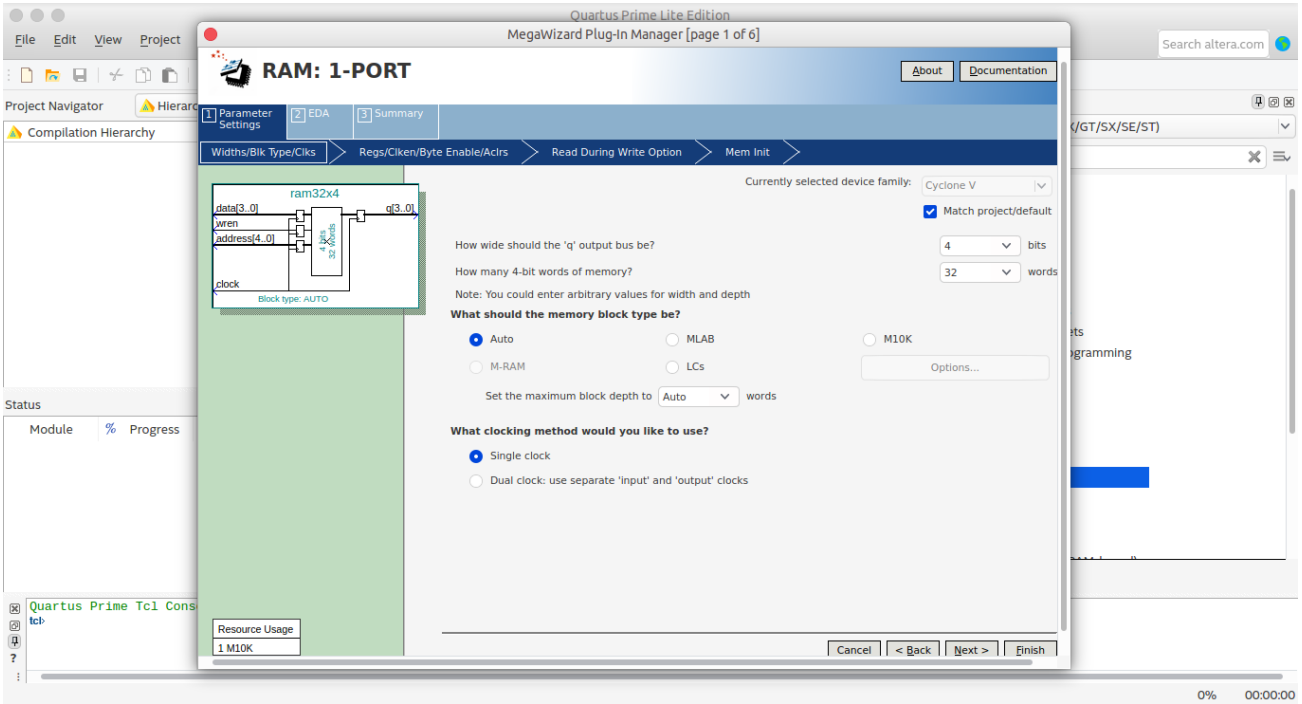
2. In the IP Catalog pane, expand Installed IP → Library → Basic Functions → On Chip Memory, then double-click on *RAM:1-PORT*.



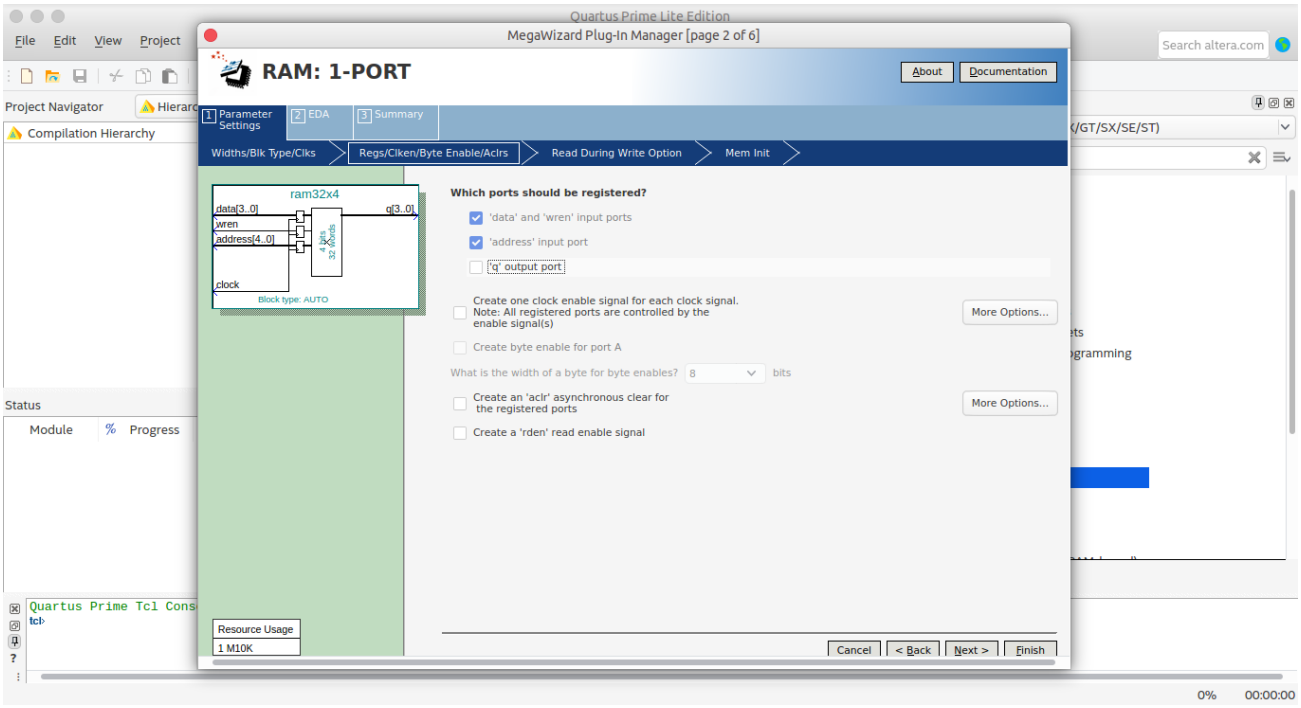
3. Browse to the folder or directory where you want to build your project. This is where the file for the memory module will be created. Call the file `ram32x4.v`. Choose the IP variation to be Verilog and click OK.



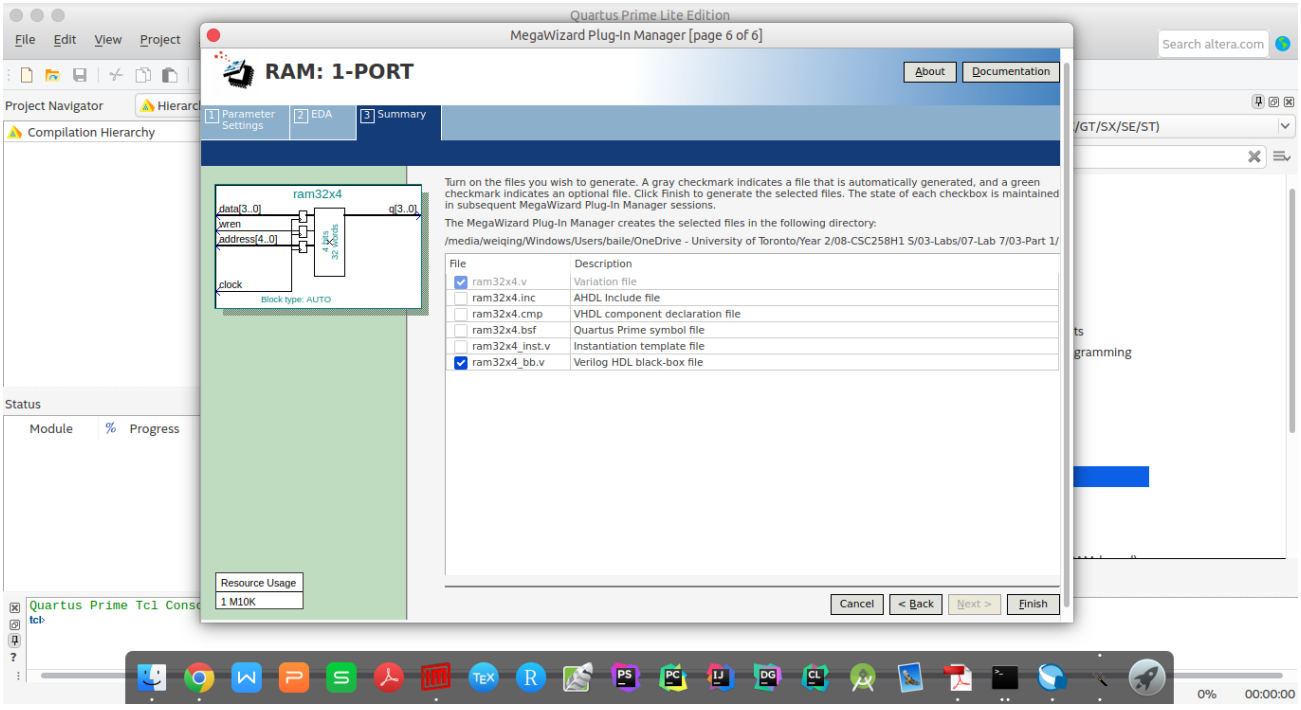
4. Select a 4-bit wide (width of 'q' output bus) memory with 32 words. Leave the memory block type as *Auto* and use a *Single Clock*. Observe how the memory block diagram on the left side of the window changes automatically as you change different parameters. Pay particular attention to the fact that 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.



5. **Deselect** q as a registered port.



6. Click Finish and Finish again to generate the new Verilog file, `ram32x4.v`.



7. 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.

Open ram32x4.v Save

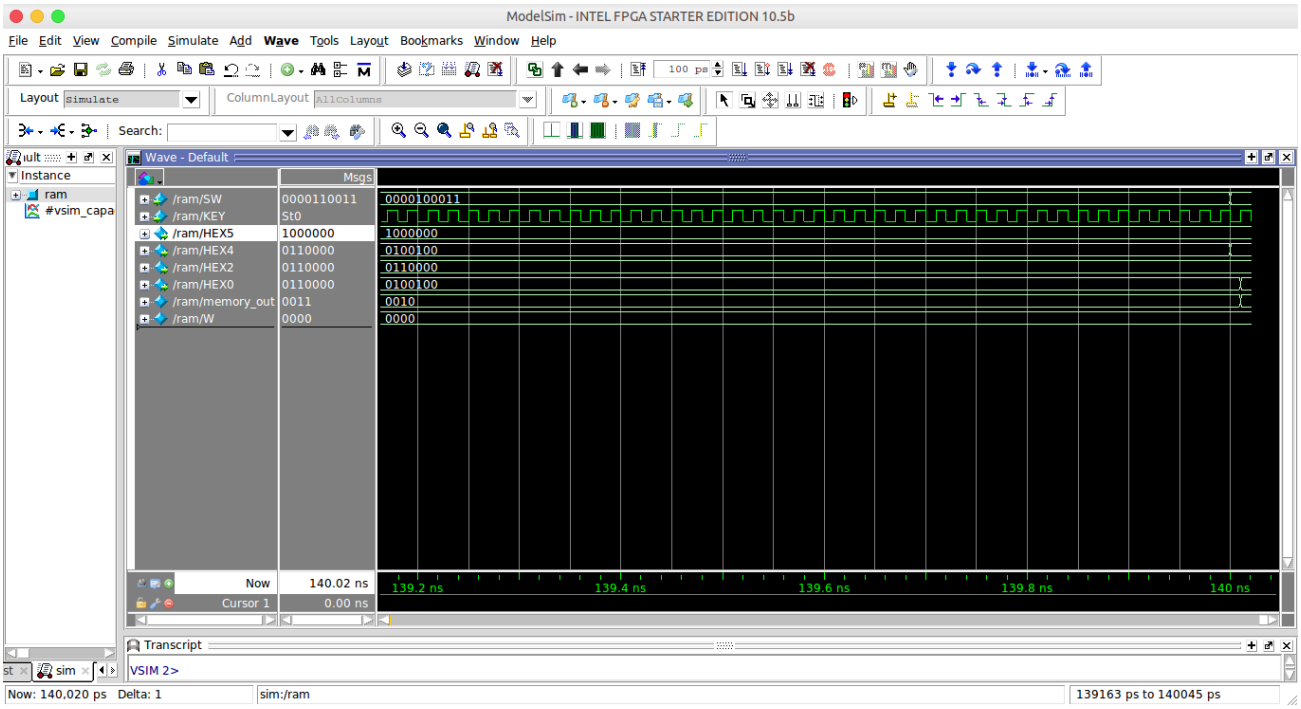
```
// megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: ram32x4.v
// Megafunction Name(s):
//                     altsyncram
//
// Simulation Library Files(s):
//                     altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 18.1.0 Build 625 09/12/2018 SJ Lite Edition
// ************************************************************


//Copyright (C) 2018  Intel Corporation. All rights reserved.
//Your use of Intel Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel FPGA IP License Agreement, or other applicable license
//agreement, including, without limitation, that your use is for
//the sole purpose of programming logic devices manufactured by
//Intel and sold by Intel or its authorized distributors.  Please
//refer to the applicable agreement for further details.
```

Verilog Tab Width: 8 Ln 1, Col 1 INS

8. 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.



9. 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.

```
1 // megafunction wizard: %RAM: 1-PORT%
```

```
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: ram32x4.v
// Megafunction Name(s):
//          altsyncram
//
// Simulation Library Files(s):
//          altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 17.0.0 Build 595 04/25/2017 SJ Lite Edition
// ************************************************************


//Copyright (C) 2017  Intel Corporation. All rights reserved.
//Your use of Intel Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
//that your use is for the sole purpose of programming logic
//devices manufactured by Intel and sold by Intel or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on

module ram(
        input [9:0] SW,
        input [0:0] KEY,
        output [6:0] HEX5,
        output [6:0] HEX4,
        output [6:0] HEX2,
        output [6:0] HEX0
            );

    wire [3:0] memory_out;
    ram32x4 r0(
        .address(SW[8:4]),
        .clock(KEY[0]),
        .data(SW[3:0]),
        .wren(SW[9]),
        .q(memory_out[3:0]));

```

```
58      //input data
59      hex_decoder H2(
60          .hex_digit(SW[3:0]),
61          .segments(HEX2)
62          );
63
64      //output of memory
65      hex_decoder H0(
66          .hex_digit(memory_out[3:0]),
67          .segments(HEX0)
68          );
69
70      //address
71      wire [3:0] W;
72      assign W[3:0] = {3'b000, SW[8]};
73
74      hex_decoder H5(
75          .hex_digit(W[3:0]),
76          .segments(HEX5)
77          );
78      hex_decoder H4(
79          .hex_digit(SW[7:4]),
80          .segments(HEX4)
81          );
82  endmodule
83
84
85  module ram32x4 (
86      address,
87      clock,
88      data,
89      wren,
90      q);
91
92      input    [4:0]  address;
93      input           clock;
94      input    [3:0]  data;
95      input           wren;
96      output   [3:0]  q;
97  `ifndef ALTERA_RESERVED_QIS
98  // synopsys translate_off
99  `endif
100     tri1            clock;
101 `ifndef ALTERA_RESERVED_QIS
102 // synopsys translate_on
103 `endif
104
105     wire [3:0] sub_wire0;
106     wire [3:0] q = sub_wire0[3:0];
107
108     altsyncram  altsyncram_component (
109                 .address_a (address),
110                 .clock0 (clock),
111                 .data_a (data),
112                 .wren_a (wren),
113                 .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 = "UNREGISTERED",
        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

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
```

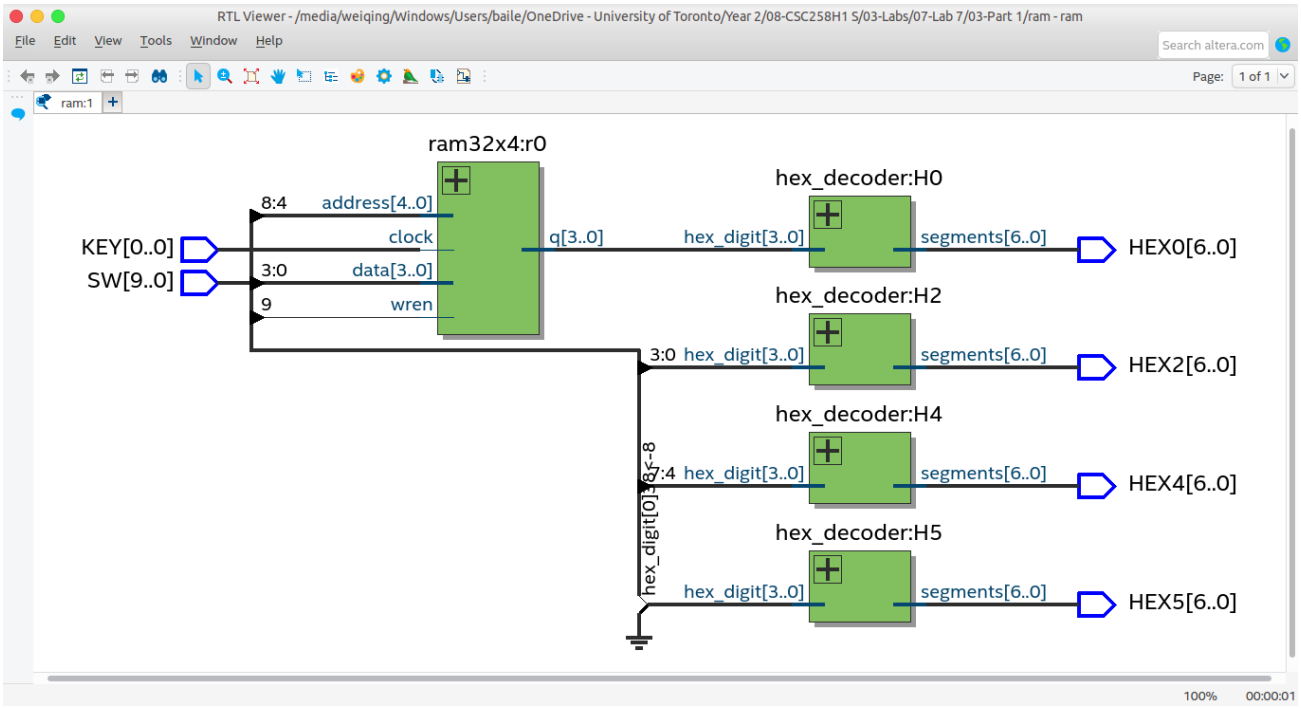
```
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING ""
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "32"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "5"
// Retrieval info: PRIVATE: WidthData NUMERIC "4"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "32"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING "NEW_DATA_NO_NBE_REA
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "5"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "4"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 5 0 INPUT NODEFVAL "address[4..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 4 0 INPUT NODEFVAL "data[3..0]"
// Retrieval info: USED_PORT: q 0 0 4 0 OUTPUT NODEFVAL "q[3..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 5 0 address 0 0 5 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 4 0 data 0 0 4 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 4 0 @q_a 0 0 4 0
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram32x4_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf


module hex_decoder(hex_digit, segments);
    input [3:0] hex_digit;
    output reg [6:0] segments;

    always @(*)
```

```
226         case (hex_digit)
227             4'h0: segments = 7'b100_0000;
228             4'h1: segments = 7'b111_1001;
229             4'h2: segments = 7'b010_0100;
230             4'h3: segments = 7'b011_0000;
231             4'h4: segments = 7'b001_1001;
232             4'h5: segments = 7'b001_0010;
233             4'h6: segments = 7'b000_0010;
234             4'h7: segments = 7'b111_1000;
235             4'h8: segments = 7'b000_0000;
236             4'h9: segments = 7'b001_1000;
237             4'hA: segments = 7'b000_1000;
238             4'hB: segments = 7'b000_0011;
239             4'hC: segments = 7'b100_0110;
240             4'hD: segments = 7'b010_0001;
241             4'hE: segments = 7'b000_0110;
242             4'hF: segments = 7'b000_1110;
243             default: segments = 7'h7f;
244         endcase
245 endmodule
```

10. Draw a schematic describing the circuit as part of your preparation.



2 Part II

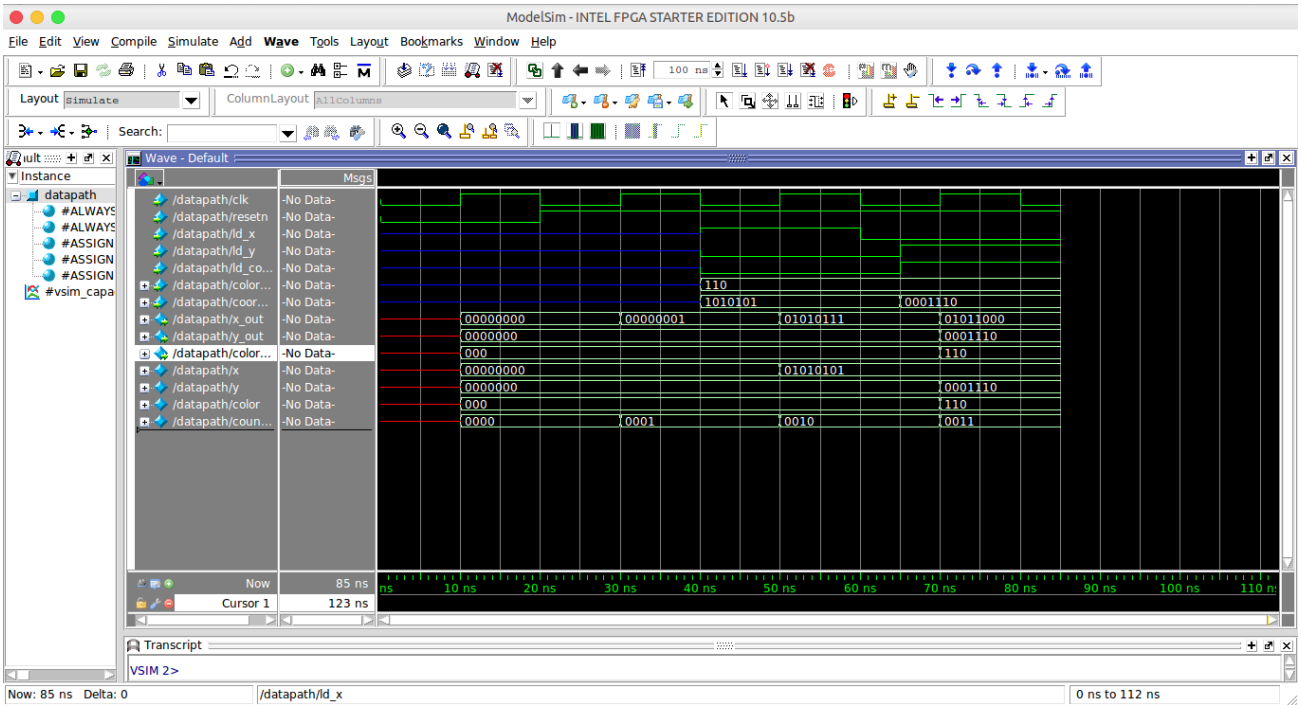
1. Design (draw the schematic and write Verilog code by adding to the provided skeleton code in `lab7-part2.zip`) and simulate a datapath that implements the required functionality. Note that the provided skeleton code incorporates the VGA adapter. Your simulation for this part should only include the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

```
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
```



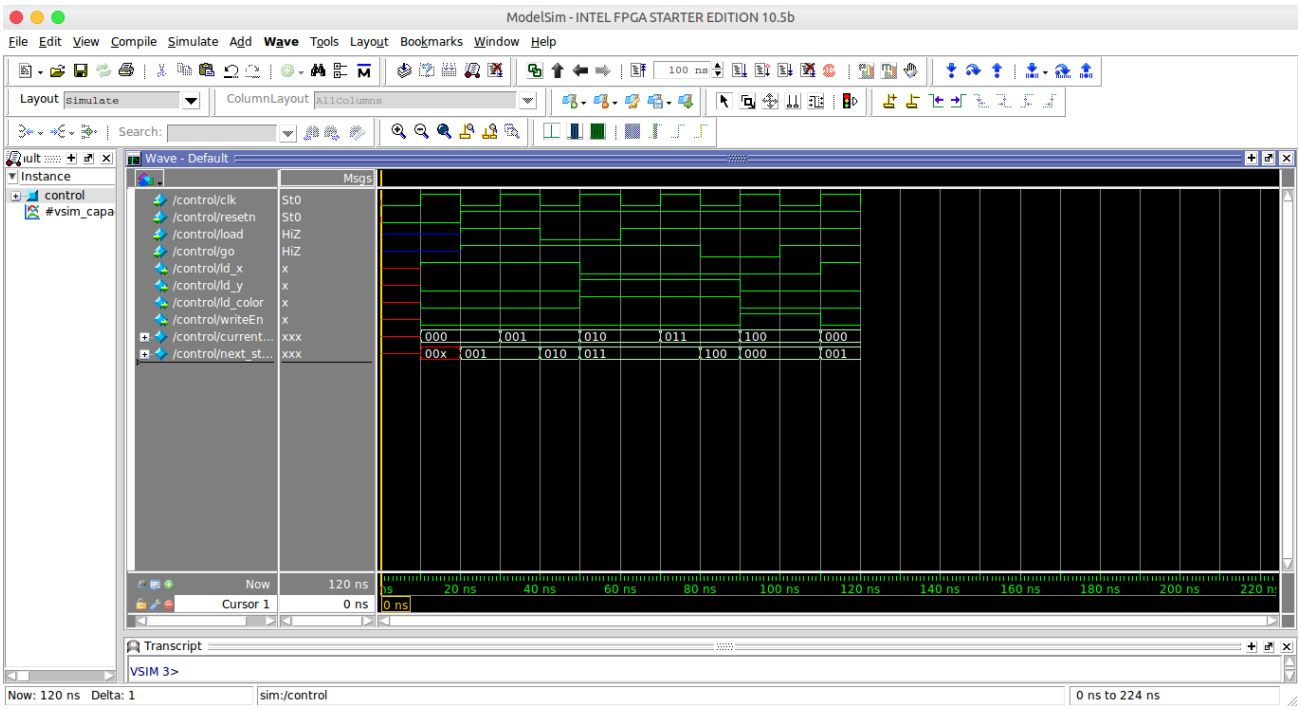
2. Design (draw a state diagram, state table, and write Verilog code by adding to the provided skeleton code) to implement and simulate an FSM that controls the implemented datapath. Your simulation for this part should only include the FSM.

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

```
    //output logic  aka output of datapath control signals
    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.

```
module combination(
```

```
    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
```