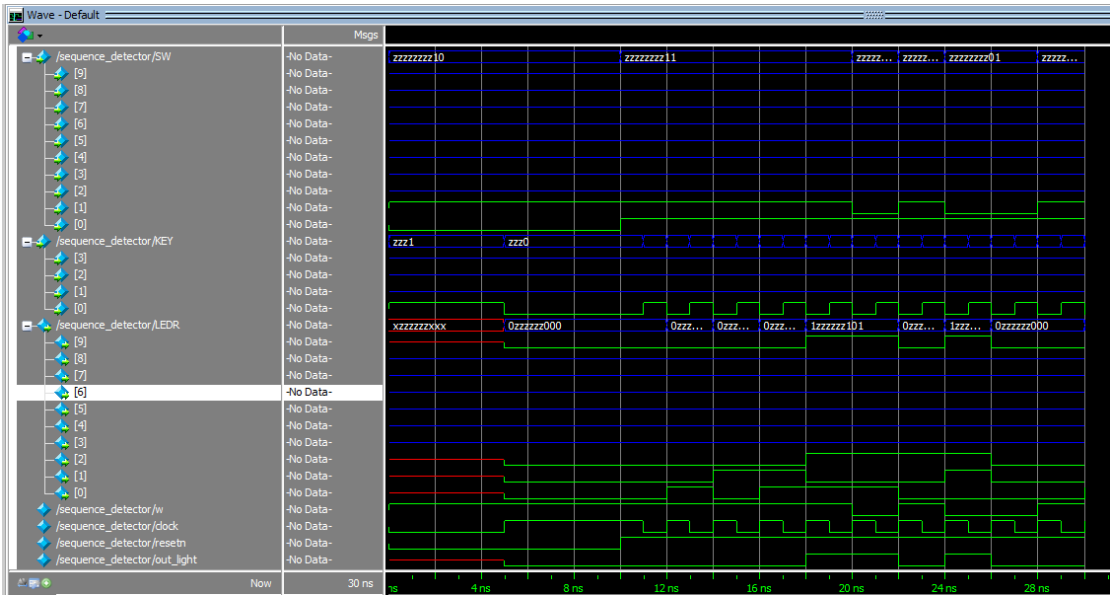
CSC258 Prelab

Part1

2. It is an active low synchronous reset. If reset is low, and after the key be released, the state should become to A.
3.

```
always @(*)
begin: state_table
    case (y_Q)
        A: begin
                if (!w) Y_D = A;
                else Y_D = B;
           end
        B: begin
                if(!w) Y_D = A;
                else Y_D = C;
           end
        C: begin
                if(!w) Y_D = E;
                else Y_D = D;
           end
        D: begin
                if(!w) Y_D = E;
                else Y_D = F;
           end
        E: begin
                if(!w) Y_D = A;
                else Y_D = G;
           end
        F: begin
                if(!w) Y_D = E;
                else Y_D = F;
           end
        G: begin
                if(!w) Y_D = A;
                else Y_D = C;
           end
        default: Y_D = A;
    endcase
end // state_table

// State Register (i.e., FFs)
always @(posedge clock)
begin: state_FFs
    if(resetn == 1'b0)
        y_Q <=  A; // Should set reset state to state A
    else
        y_Q <= Y_D;
end // State Register

// Output logic
// Set out_light to 1 to turn on LED when in relevant states
assign out_light = ((y_Q == G) || (y_Q == F));
```

4.


Wave - Default
Msgs
/sequence_detector/SW
[9]
[8]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
/sequence_detector/KEY
[3]
[2]
[1]
[0]
/sequence_detector/LEDR
[9]
[8]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
/sequence_detector/w
/sequence_detector/clock
/sequence_detector/resetn
/sequence_detector/out_light
-No Data-
zzzzzzzz10
zzzzzzzz11
zzzzz...
zzzzz...
zzzzzzzz01
zzzzz...
zzz1
zzz0
xzzzzzzxxx
0zzzzzz000
0zzz...
0zzz...
0zzz...
1zzzzzz101
0zzz...
1zzz...
0zzzzzz000
Now
30 ns
ns
4 ns
8 ns
12 ns
16 ns
20 ns
24 ns
28 ns

Part2:

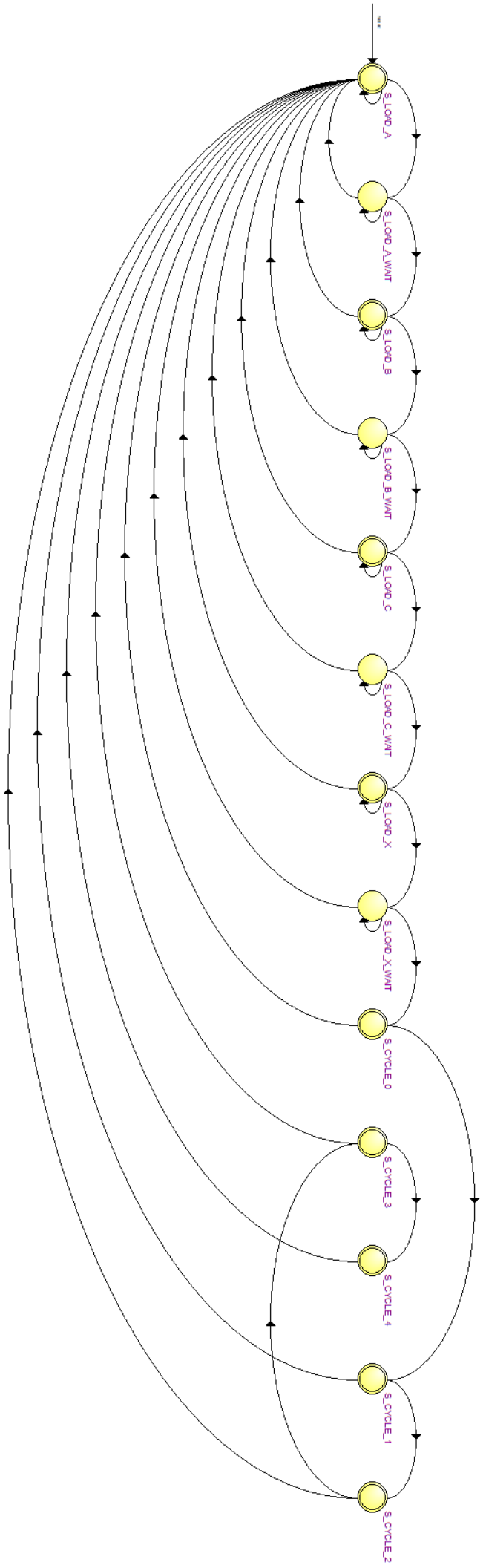
2.

High-level Step	Control Signals	Register A	Register B	Register C	Register X
Load data into A	ld_a = 1	A	0	0	0
Load data into B	ld_b = 1	A	B	0	0
Load data into C	ld_c = 1	A	B	C	0
Load data into X	ld_x = 1	A	B	C	X
multiply A & X, store result in A	ld_alu_out = 1'b1; ld_a = 1'b1; alu_select_a = 2'b11; alu_select_b = 2'b00; alu_op = 1'b1;	A * X	B	C	X
multiply A & X, store result in A	ld_alu_out = 1'b1; ld_a = 1'b1; alu_select_a = 2'b11; alu_select_b = 2'b00; alu_op = 1'b1;	A * X * X	B	C	X
multiply B & X, store result in B	ld_alu_out = 1'b1; ld_b = 1'b1; alu_select_a = 2'b01; alu_select_b = 2'b11; alu_op = 1'b1;	A*X*X	B*X	C	X
add B to A, store result in A	ld_alu_out = 1'b1; ld_a = 1'b1; alu_select_a = 2'b00; alu_select_b = 2'b01; alu_op = 1'b0;	A*X*X+B*X	B*X	C	X
add C to A, store result in result register	ld_r = 1'b1; alu_select_a = 2'b00; alu_select_b = 2'b10; alu_op = 1'b0;	A*X*X+B*X+C	B*X	C	X

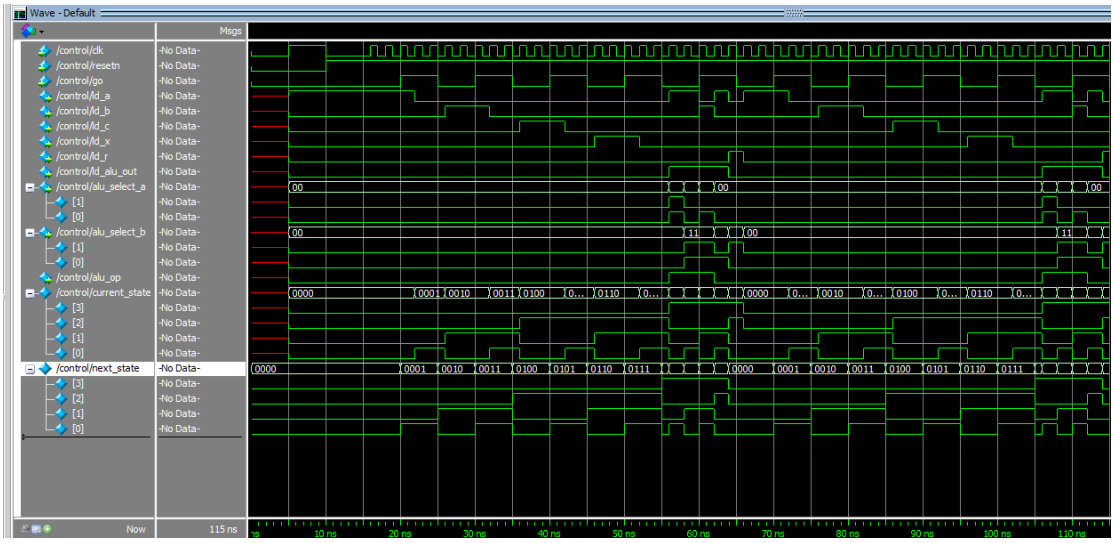
4.

```
module control(
    input clk,
    input resetn,
    input go,

    output reg  ld_a, ld_b, ld_c, ld_x, ld_r,
    output reg  ld_alu_out,
    output reg [1:0]  alu_select_a, alu_select_b,
    output reg alu_op
    );

    reg [3:0] current_state, next_state;

    localparam  S_LOAD_A        = 4'd0,
                S_LOAD_A_WAIT   = 4'd1,
                S_LOAD_B        = 4'd2,
                S_LOAD_B_WAIT   = 4'd3,
                S_LOAD_C        = 4'd4,
                S_LOAD_C_WAIT   = 4'd5,
                S_LOAD_X        = 4'd6,
                S_LOAD_X_WAIT   = 4'd7,
                S_CYCLE_0       = 4'd8,
                S_CYCLE_1       = 4'd9,
                S_CYCLE_2       = 4'd10,
                S_CYCLE_3       = 4'd11,
                S_CYCLE_4       = 4'd12;


    // Next state logic aka our state table
    always@(*)
    begin: state_table
            case (current_state)
                S_LOAD_A: next_state = go ? S_LOAD_A_WAIT : S_LOAD_A; // Loop in current state until value is input
                S_LOAD_A_WAIT: next_state = go ? S_LOAD_A_WAIT : S_LOAD_B; // Loop in current state until go signal goes low
                S_LOAD_B: next_state = go ? S_LOAD_B_WAIT : S_LOAD_B; // Loop in current state until value is input
                S_LOAD_B_WAIT: next_state = go ? S_LOAD_B_WAIT : S_LOAD_C; // Loop in current state until go signal goes low
                S_LOAD_C: next_state = go ? S_LOAD_C_WAIT : S_LOAD_C; // Loop in current state until value is input
                S_LOAD_C_WAIT: next_state = go ? S_LOAD_C_WAIT : S_LOAD_X; // Loop in current state until go signal goes low
                S_LOAD_X: next_state = go ? S_LOAD_X_WAIT : S_LOAD_X; // Loop in current state until value is input
                S_LOAD_X_WAIT: next_state = go ? S_LOAD_X_WAIT : S_CYCLE_0; // Loop in current state until go signal goes low
                S_CYCLE_0: next_state = S_CYCLE_1;
                S_CYCLE_1: next_state = S_CYCLE_2;
                S_CYCLE_2: next_state = S_CYCLE_3;
                S_CYCLE_3: next_state = S_CYCLE_4;
                S_CYCLE_4: next_state = S_LOAD_A; // we will be done our two operations, start over after
            default:     next_state = S_LOAD_A;
        endcase
    end // state_table


    // Output logic aka all of our datapath control signals
    always @(*)
    begin: enable_signals
        // By default make all our signals 0
        ld_alu_out = 1'b0;
        ld_a = 1'b0;
        ld_b = 1'b0;
        ld_c = 1'b0;
        ld_x = 1'b0;
        ld_r = 1'b0;
        alu_select_a = 2'b00;
        alu_select_b = 2'b00;
        alu_op       = 1'b0;

        case (current_state)
            S_LOAD_A: begin
                ld_a = 1'b1;
                end
            S_LOAD_B: begin
                ld_b = 1'b1;
                end
            S_LOAD_C: begin
                ld_c = 1'b1;
                end
            S_LOAD_X: begin
                ld_x = 1'b1;
                end
            S_CYCLE_0: begin // Do A <- A * x
                ld_alu_out = 1'b1; ld_a = 1'b1; // store result back into A
                alu_select_a = 2'b11; // Select register X
                alu_select_b = 2'b00; // Also select register A
                alu_op = 1'b1; // Do multiply operation
                end
            S_CYCLE_1: begin // Do A <- A * x * x
                ld_alu_out = 1'b1; ld_a = 1'b1; // store result back into A
                alu_select_a = 2'b00; // Select register A
                alu_select_b = 2'b11; // Also select register X
                alu_op = 1'b1; // Do multiply operation
                end
            S_CYCLE_2: begin // Do B <- B * x
                ld_alu_out = 1'b1; ld_b = 1'b1; // store result back into B
                alu_select_a = 2'b01; // Select register B
                alu_select_b = 2'b11; // Also select register X
                alu_op = 1'b1; // Do multiply operation
                end
            S_CYCLE_3: begin // Do A <- A * X * X + B * X
                ld_alu_out = 1'b1; ld_a = 1'b1; // store result back into A
                alu_select_a = 2'b00; // Select register A
                alu_select_b = 2'b01; // Also select register B
                alu_op = 1'b0; // Do Add operation
                end
            S_CYCLE_4: begin
                ld_r = 1'b1; // store result in result register
                alu_select_a = 2'b00; // Select register A
                alu_select_b = 2'b10; // Select register C
                alu_op = 1'b0; // Do Add operation
                end
            // default:    // don't need default since we already made sure all of our outputs
            // were assigned a value at the start of the always block
            endcase
    end // enable_signals
```

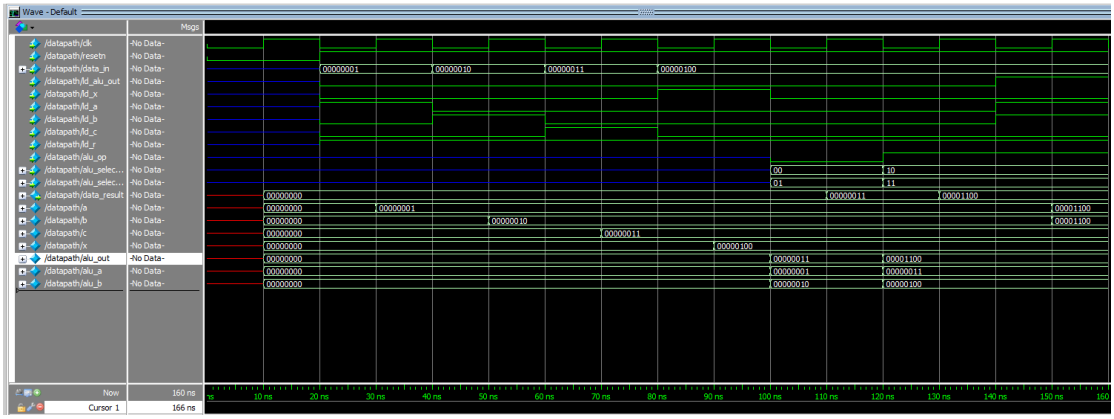
5.


S_LOAD_A
S_LOAD_A_WAIT
S_LOAD_B
S_LOAD_B_WAIT
S_LOAD_C
S_LOAD_C_WAIT
S_LOAD_X
S_LOAD_X_WAIT
S_CYCLE_0
S_CYCLE_3
S_CYCLE_4
S_CYCLE_1
S_CYCLE_2

6. Simulation for controller



Simulation for datapath



Simulation for whole program