**Part1**

In the part1 of the lab, we complied the Verilog code programed in the prelab, a d flip-flop, a four to one multiplexer, a four-bit counter, and a d latch. And simulated every module.

**D-flip-flop**

module flipflop (input d, clk, reset, enable, output reg q);

always @(posedge clk or negedge reset)

begin

if (~reset)

q <= 1'b0;

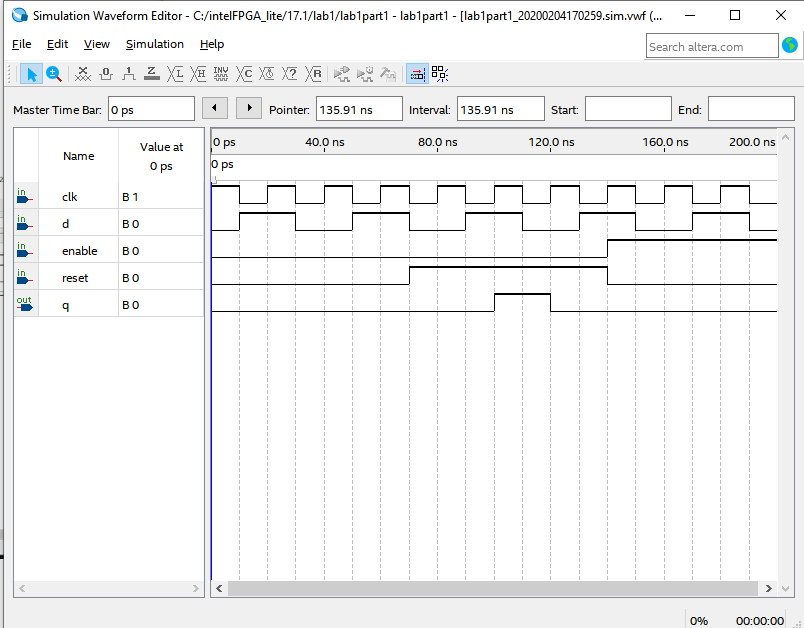
else begin

if (~enable)

q <= d;

end

end

endmodule 

**Four to one multiplexer**

module mux41(input [3:0] d0, d1, d2, d3,

input [1:0] sel,

output reg [3:0] Q);

always @(sel or d0 or d1 or d2 or d3)

begin

case (sel)

2'b00: Q <= d0;

2'b01: Q <= d1;

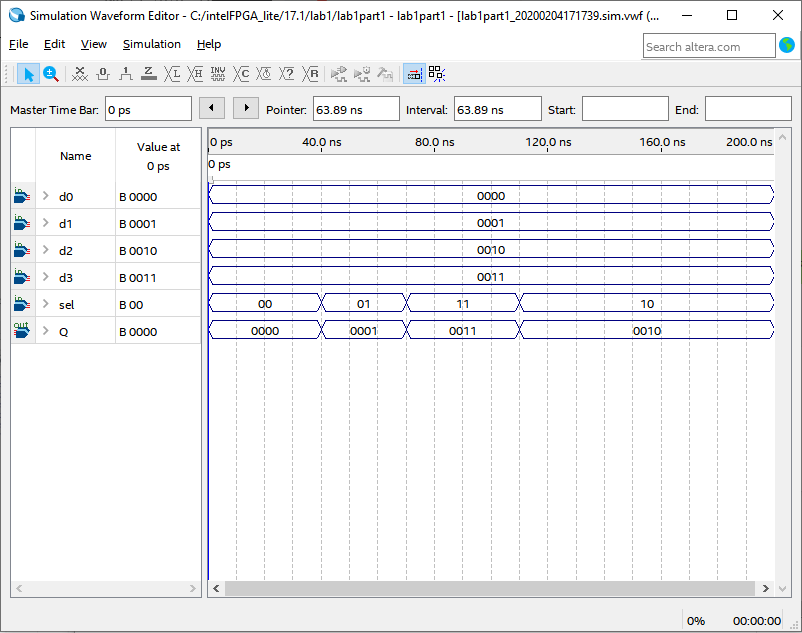
2'b10: Q <= d2;

2'b11: Q <= d3;

endcase

end

endmodule



**Four-bit counter**

module counter(reset, enable, q); //UP counter

input reset, enable;

output [3:0] q;

reg [3:0] data;

always @(posedge enable or posedge reset)

begin

if (reset) //synchronous active high reset

data <= 4'b0000;

else

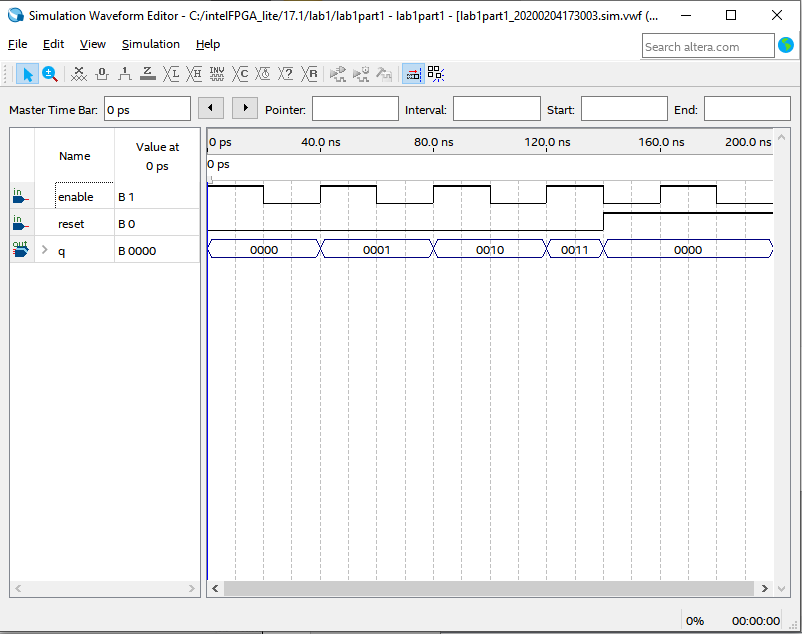
if (enable) //synchronous active high enable

data <= data + 1'b1;

end

assign q = data;

endmodule



**D-latch**

module dlatch1(

data,

enable,

reset,

q

);

input data, enable, reset;

output q;

reg q;

always @ (\*)

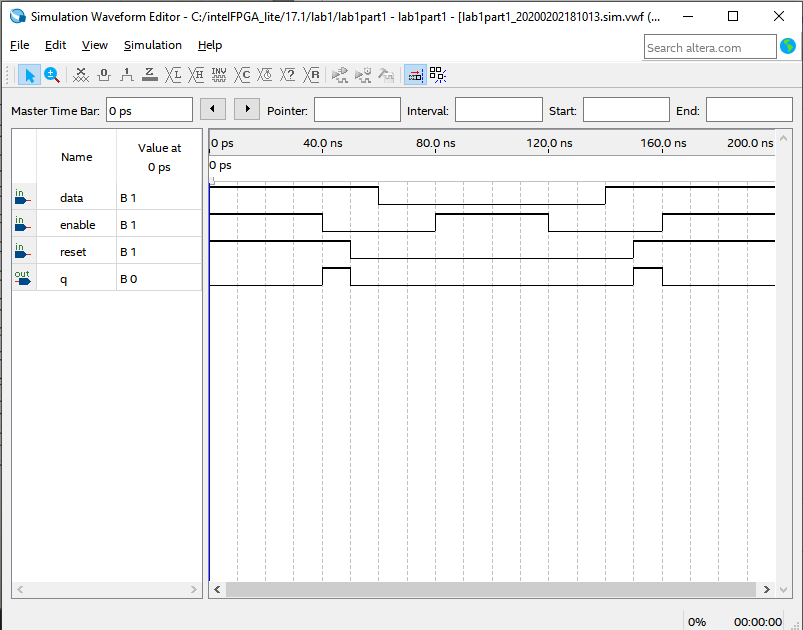
if(~reset | enable)begin

q <= 1'b0;

end else if (~enable) begin

q <= data;

end

endmodule

**Part 2**

In the part2 of the lab, we used the seven-segment Verilog code that we programed in the prelab and complied the result. The result matched with expectation.

module lab1part2(input [3:0] x, output [6:0] hex\_LEDs);

reg [6:0] reg\_LEDs;

assign hex\_LEDs[6:0] = reg\_LEDs[6:0];

always @ (\*)

begin

case(x)

4'b0000: reg\_LEDs[6:0]=7'b1000000;

4'b0001: reg\_LEDs[6:0]=7'b1111001;

4'b0010: reg\_LEDs[6:0]=7'b0100100;

4'b0011: reg\_LEDs[6:0]=7'b0110000;

4'b0100: reg\_LEDs[6:0]=7'b0011001;

4'b0101: reg\_LEDs[6:0]=7'b0010010;

4'b0110: reg\_LEDs[6:0]=7'b0000010;

4'b0111: reg\_LEDs[6:0]=7'b1111000;

4'b1000: reg\_LEDs[6:0]=7'b0000000;

4'b1001: reg\_LEDs[6:0]=7'b0010000;

4'b1010: reg\_LEDs[6:0]=7'b0011000;

4'b1011: reg\_LEDs[6:0]=7'b1100011;

4'b1100: reg\_LEDs[6:0]=7'b0001000;

4'b1101: reg\_LEDs[6:0]=7'b0101011;

4'b1110: reg\_LEDs[6:0]=7'b0010001;

4'b1111: reg\_LEDs[6:0]=7'b1111111;

endcase

end

endmodule

**Part 3**

In the part3 of the lab, we combined modules from part1 and part2 to create a new project that can display seven-segment decoder, a 4-bit counter that counts KEY3 been pressed, and a 30-bit counter that counts the pulses of CLOCK\_50 of the DE1-SOC board.

module total ( input [9:0] SW,

input [3:2]KEY,

input CLOCK\_50,

output [6:0] HEX0);

wire [3:0] m\_00, m\_01, m\_10, m\_11, y;

mode\_00 (SW [3:0], m\_00);

mode\_01 (KEY [2], KEY [3], m\_01);

mode\_10 (KEY[2], CLOCK\_50, m\_10);

mode\_11 (m\_11);

mux\_4to1\_case\_sw9sw8 (m\_00, m\_01, m\_10, m\_11, SW [9:8], y);

seven\_seg\_decoder(y, HEX0);

endmodule

module mode\_00 (input [3:0] SW, output reg [3:0] y);

always @ (SW)

begin

y <= SW;

end

endmodule

module mode\_01 (input KEY2, input KEY3, output [3:0] y);

reg [3:0] counter;

initial

begin

counter <= 4'b0000;

end

always @ (posedge KEY3)

begin

if (~KEY2)

begin

counter <= 4'b0000;

end

// else if (y == 4'b1111)

// begin

// y <= 4'b0000;

// end

else

begin

counter <= counter + 4'b0001;

end

end

assign y = counter;

endmodule

module mode\_10 (input KEY2, input CLK\_50, output [3:0] y);

reg [29:0] counter;

always @ (posedge CLK\_50)

begin

if (~KEY2)

begin

counter <= 30'b000000000000000000000000000000;

end

else if (counter == 30'b111111111111111111111111111111)

begin

counter <= 30'b000000000000000000000000000000;

end

else

begin

counter <= counter + 1'b1;

end

end

// always @ (negedge KEY0)

// begin

// counter <= 30'b000000000000000000000000000000;

// end

assign y = counter [9:6];

endmodule

module mode\_11 (output reg [3:0] y);

always @ (\*)

begin

y <= 4'b1111;

end

endmodule

module mux\_4to1\_case\_sw9sw8 ( input [3:0] mod\_00,

input [3:0] mod\_01,

input [3:0] mod\_10,

input [3:0] mod\_11,

input [1:0] s,

output reg [3:0] y);

always @ (mod\_00 or mod\_01 or mod\_10 or mod\_11 or s)

begin

case (s)

2'b00 : y <= mod\_00;

2'b01 : y <= mod\_01;

2'b10 : y <= mod\_10;

2'b11 : y <= mod\_11;

endcase

end

endmodule

module seven\_seg\_decoder(input [3:0] x, output [6:0] hex\_LEDs);

reg [6:0] reg\_LEDs;

assign hex\_LEDs[6:0] = reg\_LEDs[6:0];

always @ (\*)

begin

case(x)

4'b0000: reg\_LEDs[6:0]=7'b1000000;

4'b0001: reg\_LEDs[6:0]=7'b1111001;

4'b0010: reg\_LEDs[6:0]=7'b0100100;

4'b0011: reg\_LEDs[6:0]=7'b0110000;

4'b0100: reg\_LEDs[6:0]=7'b0011001;

4'b0101: reg\_LEDs[6:0]=7'b0010010;

4'b0110: reg\_LEDs[6:0]=7'b0000010;

4'b0111: reg\_LEDs[6:0]=7'b1111000;

4'b1000: reg\_LEDs[6:0]=7'b0000000;

4'b1001: reg\_LEDs[6:0]=7'b0010000;

4'b1010: reg\_LEDs[6:0]=7'b0011000;

4'b1011: reg\_LEDs[6:0]=7'b1100011;

4'b1100: reg\_LEDs[6:0]=7'b0001000;

4'b1101: reg\_LEDs[6:0]=7'b0101011;

4'b1110: reg\_LEDs[6:0]=7'b0010001;

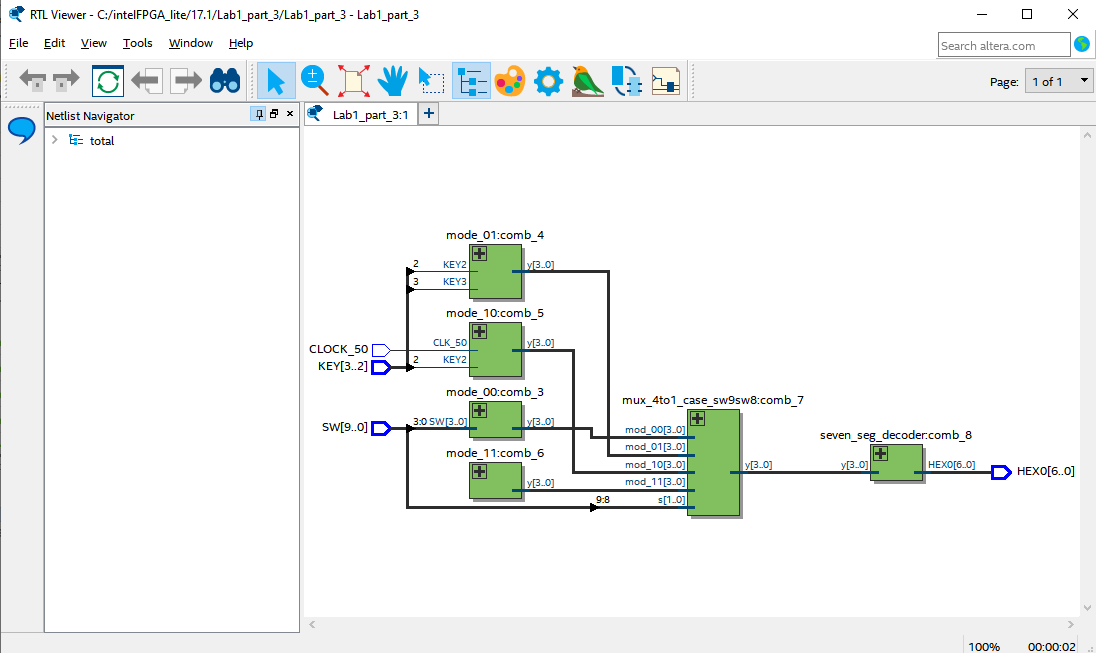
4'b1111: reg\_LEDs[6:0]=7'b1111111;

endcase

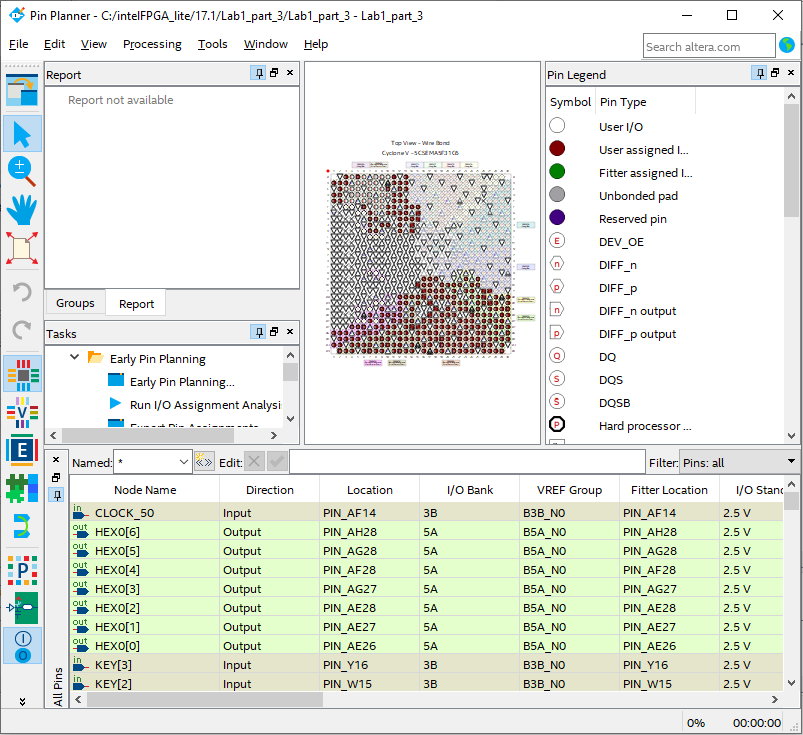
end

endmodule

RTL Viewer



Pin planer screen shot



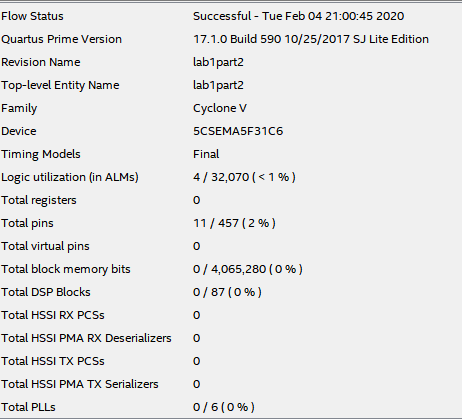
**Report questions**

1. The DE1-SOC.qsf file that I imported into my design is for pin assignment, the file helps me find the 16-bit input pin name.
2. For part2:

The total number of logic elements used by my circuit is 4.

The total number of registers is 0.

The total number of pins is 11.

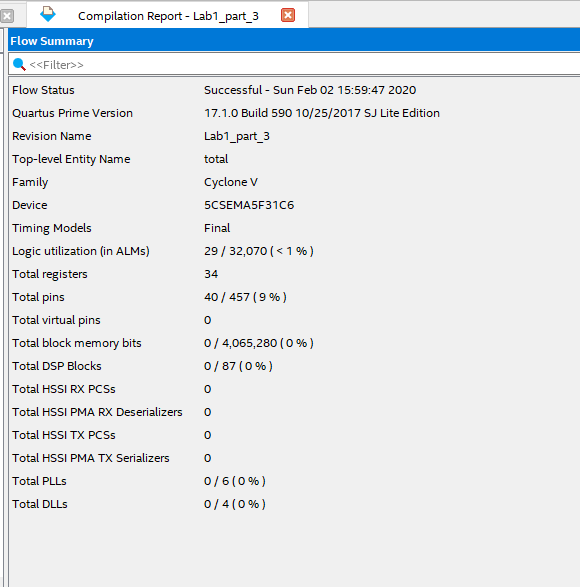


For part3:

The total number of logic elements used by my circuit is 29.

The total number of registers is 34.

The total number of pins is 20.



The maximum number of logic elements that can fit on the FPGA is 32070.