**Lab1 – Part 1**

One bit data width D flip-flop

Graphical user interface, application, table, Excel

Description automatically generated

One bit data width D flip-flop with active low synchronous reset

Diagram

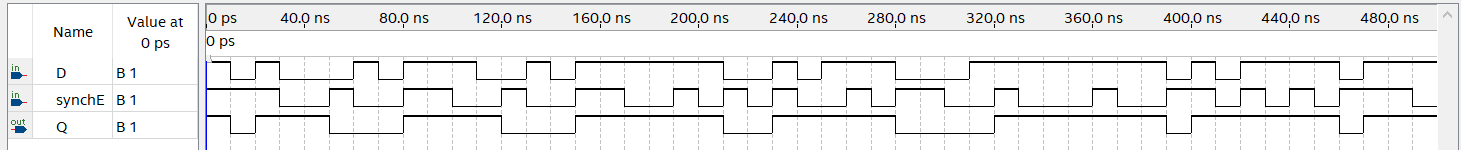
Description automatically generated

One bit data width D flip-flop with active low synchronous reset and active low enable

Diagram

Description automatically generated

D latch with synchronous enable control



4-to-1 multiplexer

A picture containing diagram

Description automatically generated

4-bit counter with reset and enable controls

A picture containing timeline

Description automatically generated

**Lab1 – Part 3**

A screenshot of a computer

Description automatically generated with medium confidence

**Figure 1: RTL Viewer of Part 3 Compilation – Overview of the Project**

Diagram, schematic

Description automatically generated

Figure : RTL View of Seven-Segment Decoder

Diagram, schematic

Description automatically generated

Figure : RTL View of the Multiplexer

Diagram, schematic

Description automatically generated

Figure : RTL View of the Counter Clock

Chart, scatter chart

Description automatically generated

**Figure 5: Assignment -> Pin Planner**

**Questions**

In Part 1, we compiled and simulated the modules built during the prelab. We ensured to use various test cases through randomization and observed the output.

In Part 2, we used the multiplexer and switches 0 to 3 to display a specific value on the seven-segment display.

In Part 3, we created a top level module to bring together several other modules, and passed them all through a multiplexer to control what was being output through the seven segment decoder module.

1. The DE1-SoC.qsf file is to map switch, button, key and pin assignments to the corresponding inputs and outputs on the board. These assignments can then be referenced within Verilog to be used as required.
2. Compilation report numbers:
   1. Total number of logic elements: 24
   2. Total number of registers: 36
   3. Total number of pins: 22
   4. Maximum number of logic elements: 32070

**Code**

**Part 2:**

module lab1part2 (input [3:0] SW, output [6:0] HEX0);

  seven\_seg\_decoder decoder(.x(SW), .hex\_LEDs(HEX0));

endmodule

seven\_seg\_decoder Module provided in prelab

**Part 3:**

module lab1part3 (input [9:0] SW, input [3:0] KEY, input CLOCK\_50, output [6:0] HEX0);

  wire [3:0] b\_cnt;

  wire [29:0] c\_cnt;

  wire [3:0] disp\_in;

  seven\_seg\_decoder decoder (.x(disp\_in), .hex\_LEDs(HEX0));

  counter\_button button (.key(KEY[3]), .reset(KEY[0]), .out(b\_cnt));

  counter\_clock clock (.clk(CLOCK\_50), .reset(KEY[0]), .out(c\_cnt));

  multiplexer multiplex (.a(SW[3:0]), .b(b\_cnt), .c(c\_cnt[29:26]),

                         .sel(SW[9:8]), .out(disp\_in));

endmodule

module counter\_clock (input clk, reset, output reg[WIDTH-1:0] out);

  parameter WIDTH = 30;

  //2's power of 26 is 67,108,864.

  // that is 1 second (2^26/50e6 = 1.34), if count is using CLOCK 50

  always@(posedge clk or negedge reset) begin

    if(!reset) begin

      out <= 30'b0;

    end else begin

      out <= out + 1'b1;

    end

  end

endmodule

module counter\_button (input key, reset, output reg [3:0] out);

  always @(negedge key or negedge reset) begin

    if (!reset)

      out <= 4'b0;

    else

      out <= out + 1'b1;

  end

endmodule

module multiplexer (input [3:0] a, b, c, input [1:0] sel, output reg [3:0] out);

  always @(a or b or c or sel)  begin

    case (sel)

      2'b00: out <= a;        // Decoder

      2'b01: out <= b;        // Button Counter

      2'b10: out <= c;        // Clock Counter

      2'b11: out <= 4'b1111;  // OFF

    endcase

  end

endmodule

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

  reg [6:0] reg\_LEDs;

  assign hex\_LEDs[0]= ~x[3] & ~x[2] & ~x[1] & x[0] |

                ~x[3] & x[2] & ~x[1] & ~x[0] |

                x[3] & x[2] & x[0] |

                x[3] & x[1] & x[0];   /\* expression for segment 0 \*/

  assign hex\_LEDs[1]=   x[2] & ~x[1] & x[0] |

                x[3] & x[2] & x[0] |

                x[2] & x[1] & ~x[0] |

                x[3] & x[1] & ~x[0];    /\* expression for segment 1 \*/

                assign hex\_LEDs[6:2]=reg\_LEDs[6:2];

  always @(\*)

  begin

  case (x)

    4'b0000: reg\_LEDs[6:2]=5'b10000; //7’b1000000 decimal 0

    4'b0001: reg\_LEDs[6:2]=5'b11110; //7’b1111001 decimal 1

    4'b0010: reg\_LEDs[6:2]=5'b01001; //7’b0100100 decimal 2

    4'b0011: reg\_LEDs[6:2]=5'b01100; //7’b0110000 decimal 3

    4'b0100: reg\_LEDs[6:2]=5'b00110; //7’b0011001 decimal 4

    4'b0101: reg\_LEDs[6:2]=5'b00100; //7’b0100100 decimal 5

    4'b0110: reg\_LEDs[6:2]=5'b00000; //7’b0000010 decimal 6

    4'b0111: reg\_LEDs[6:2]=5'b11110; //7’b1111000 decimal 7

    4'b1000: reg\_LEDs[6:2]=5'b00000; //7’b0000000 decimal 8

    4'b1001: reg\_LEDs[6:2]=5'b00100; //7’b0010000 decimal 9

    4'b1010: reg\_LEDs[6:2]=5'b00100; //7’b0010010 decimal S

    4'b1011: reg\_LEDs[6:2]=5'b00010; //7’b0001001 decimal H

    4'b1100: reg\_LEDs[6:2]=5'b00010; //7’b0001000 decimal A

    4'b1101: reg\_LEDs[6:2]=5'b01010; //7’b0101011 decimal N

    4'b1110: reg\_LEDs[6:2]=5'b10000; //7’b1000010 decimal G

    4'b1111: reg\_LEDs[6:2]=5'b11111; //7’b1111111 ALL OFF

  endcase

  end

endmodule