A blue square with white text

Description automatically generated

ELEX 7660: Digital System Design

Lab 2

|  |  |  |
| --- | --- | --- |
| Student Name: Taewoo Kim | Student Number: A01284763 | Set: T |

Table of Contents

[1 Screenshot of the simulations 3](#_Toc189415731)

[2 Source code of the module 3](#_Toc189415732)

[2.1 Lab2.sv code 3](#_Toc189415733)

[2.2 encoder.sv code 4](#_Toc189415734)

[2.3 en2bcd.sv 6](#_Toc189415735)

[3 Quartus compilation report 8](#_Toc189415736)

[4 RTL Netlist 8](#_Toc189415737)

[4.1 Overall view 8](#_Toc189415738)

[4.2 encoder 9](#_Toc189415739)

[4.3 enc2bcd 9](#_Toc189415740)

[4.4 decode2 9](#_Toc189415741)

[4.5 decode7 10](#_Toc189415742)

# Screenshot of the simulations

A computer code with text

Description automatically generated

A screenshot of a video game

Description automatically generated

# Source code of the module

## Lab2.sv code

|  |
| --- |
| // File: lab2.sv  // Description: ELEX 7660 lab2 top-level module.  Counts up the  //              digits of decimal number 00~99 on each side of encoder of 4 digit 7-segment  //              display module.  // Author: Robert Trost  // Updated by: Taewoo Kim  // Date: 2025-02-02  module lab2 ( input logic CLOCK\_50,       // 50 MHz clock                (\* altera\_attribute = "-name WEAK\_PULL\_UP\_RESISTOR ON" \*)                input logic enc1\_a, enc1\_b, //Encoder 1 pins                (\* altera\_attribute = "-name WEAK\_PULL\_UP\_RESISTOR ON" \*) input logic                enc2\_a, enc2\_b,             //Encoder 2 pins                output logic [7:0] leds,    // 7-seg LED enables                output logic [3:0] ct ) ;   // digit cathodes     logic [1:0] digit;  // select digit to display     logic [3:0] disp\_digit;  // current digit of count to display     logic [15:0] clk\_div\_count; // count used to divide clock     logic [7:0] enc1\_count, enc2\_count; // count used to track encoder movement and to display     logic enc1\_cw, enc1\_ccw, enc2\_cw, enc2\_ccw;  // encoder module outputs     logic enc1\_a\_db, enc1\_b\_db, enc2\_a\_db, enc2\_b\_db;     // instantiate modules to implement design     decode2 decode2\_0 (.digit,.ct) ;     decode7 decode7\_0 (.num(disp\_digit),.leds) ;    // instantiate encoders     encoder encoder\_1 (.clk(CLOCK\_50), .a(enc1\_a), .b(enc1\_b), .cw(enc1\_cw), .ccw(enc1\_ccw));     encoder encoder\_2 (.clk(CLOCK\_50), .a(enc2\_a), .b(enc2\_b), .cw(enc2\_cw), .ccw(enc2\_ccw));    // instantiate encoder to bcd     enc2bcd enc2bcd\_1 (.clk(CLOCK\_50), .cw(enc1\_cw), .ccw(enc1\_ccw), .bcd\_count(enc1\_count));     enc2bcd enc2bcd\_2 (.clk(CLOCK\_50), .cw(enc2\_cw), .ccw(enc2\_ccw), .bcd\_count(enc2\_count));       // use count to divide clock and generate a 2 bit digit counter to determine which digit to display     always\_ff @(posedge CLOCK\_50)       clk\_div\_count <= clk\_div\_count + 1'b1 ;    // assign the top two bits of count to select digit to display    assign digit = clk\_div\_count[15:14];    // Select digit to display (disp\_digit)    // Left two digits (3,2) display encoder 1 hex count and right two digits (1,0) display encoder 2 hex count    always\_comb begin        // according to enc1\_counts or enc2\_counts value set disp\_digit accordingly to set the leds       case (digit)          2'b00: disp\_digit = enc2\_count[3:0]; // if digit is 0          2'b01: disp\_digit = enc2\_count[7:4]; // if digit is 1          2'b10: disp\_digit = enc1\_count[3:0]; // if digit is 2          2'b11: disp\_digit = enc1\_count[7:4]; // if digit is 3          default: disp\_digit = 4'b0000;       // Default or error value       endcase    end  endmodule |

## encoder.sv code

|  |
| --- |
| // encoder.sv  // Description: This is simple module to control encoder  //              by checking the states in a pulse.  // author: Taewoo Kim  // date: Jan 26, 2025  module encoder( input logic a, b, clk, // input: swithc a, b and clk                  output logic cw, ccw); // output: direction of the encoder      // Initialize the prev\_a & prev\_b also define cw\_next & ccw\_next      logic prev\_a = 0;      logic prev\_b = 0;      logic cw\_next, ccw\_next;      // Patterns that's been used for the rotary encoder      /\* Prev a, a, Prev b, b              0, 1,      0, 0 : CW              1, 1,      0, 1 : CW              1, 0,      1, 1 : CW              0, 0,      1, 0 : CW              0, 0,      0, 1 : CCW              0, 1,      1, 1 : CCW              1, 1,      1, 0 : CCW              1, 0,      0, 0 : CCW  \*/        // Using the above patterns implement the conditions with case statement      always\_comb begin          // Initialize outputs to default values           cw\_next  = 1'b0;           ccw\_next = 1'b0;           // Check the pattern          case ({prev\_a, a, prev\_b, b})              4'b0010, 4'b1011, 4'b1101, 4'b0100: cw\_next = 1'b1; // states for the CW              4'b0001, 4'b0111, 4'b1110, 4'b1000: ccw\_next  = 1'b1; // states for the CCW          endcase      end      // In rising edge the set desired outputs      always\_ff @( posedge clk ) begin          prev\_a <= a;        // save a value to prev\_a          prev\_b <= b;        // save b value to prev\_b          ccw <= ccw\_next;    // set ccw          cw <= cw\_next;      // set cw      end  endmodule |

## en2bcd.sv

|  |
| --- |
| // en2bcd.sv  // This is simple module to control encoder to count up once in single detent (count one for four pulse.)  // and convert hex to 00~99  // author: Taewoo Kim  // date: Jan 26, 2025  module enc2bcd (input logic clk, cw, ccw, output logic [7:0] bcd\_count);      // Count until 3      localparam PULSE\_COUNTER = 2'b11;      // initalize the logic count (0~3 and wrap back)      logic [1:0] cw\_counter = 0, ccw\_counter = 0;      logic cw\_ready, ccw\_ready;      // using if statement to check if counter is ready and if it is send out the ccw/cw\_ready signal.      always\_ff @(posedge clk) begin        // CW handling check if it's mutually exclusive        if (cw && !ccw) begin          // once it reaches the third state (0~3) send cw\_ready and reset the counter           if (cw\_counter == PULSE\_COUNTER) begin              cw\_ready <= 1;              cw\_counter <= 0;          // if it didn't reached, send 0 for cw\_ready and count up the counter           end else begin              cw\_counter <= cw\_counter + 1;              cw\_ready <= 0;           end        end else cw\_ready <= 0;          // CCW handling (similar) check if it's mutually exclusive        if (ccw && !cw) begin          // once it reaches the third state (0~3) send ccw\_ready and reset the counter           if (ccw\_counter == PULSE\_COUNTER) begin              ccw\_ready <= 1;              ccw\_counter <= 0;          // if it didn't reached, send 0 for ccw\_ready and count up the counter           end else begin              ccw\_counter <= ccw\_counter + 1;              ccw\_ready <= 0;           end        end else ccw\_ready <= 0;      end      // encoder counts: (increment when cw\_ready = 1, decrement when ccw\_ready = 1)      always\_ff @(posedge clk)  begin          // cw activated          if (cw\_ready) begin              // check if first digit is 9, if it wrap back to 0 and check second digit is 9              // if it was 9 wrap back to 0 else +1              if(bcd\_count[3:0] == 4'h09) begin                  bcd\_count[3:0] <= 4'h00;                  bcd\_count[7:4] <= (bcd\_count[7:4] == 4'h09) ? 4'h00 : bcd\_count[7:4] + 1;              end else bcd\_count[3:0] <= bcd\_count[3:0] + 1'b1; // cw: count up          // ccw activated          end else if (ccw\_ready) begin              // if first digit is 0 than set to 9 (wrap back)              // if second digit was 0 wrap back to 0 otherwise -1              if(bcd\_count[3:0] == 4'h00) begin                  bcd\_count[3:0] <= 4'h09;                  bcd\_count[7:4] <= (bcd\_count[7:4] == 4'h00) ? 4'h09 : bcd\_count[7:4] - 1'b1;              end else bcd\_count[3:0] <= bcd\_count[3:0] - 1'b1; // ccw: count up          end      end    endmodule |

# Quartus compilation report

A screenshot of a computer program

Description automatically generated

# RTL Netlist

## Overall view

A diagram of a computer

Description automatically generated

## encoder

A computer screen shot of a computer code

Description automatically generated

## enc2bcd

A computer screen shot of a diagram

Description automatically generated

## decode2

A diagram of a computer program

Description automatically generated

## decode7

A screenshot of a computer program

Description automatically generated