ELEX 7660: Digital System Design

Lab 2

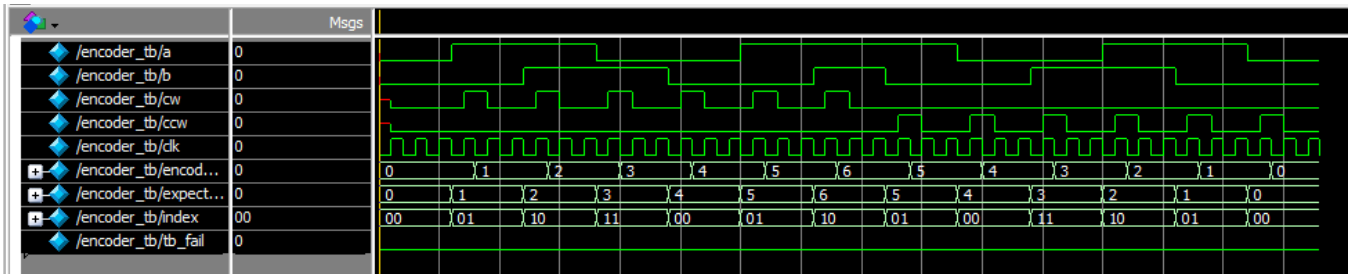
Student Name: Taewoo Kim	Student Number: A01284763	Set: T

Table of Contents

1 Screenshot of the simulations

```
VSIM 3> run -all
# Lab 2 Encoder Simulation *** PASSED ***
# ** Note: $stop    : C:/Users/Taewoo Kim/Desktop/Test/encoder_tb.sv(74)
#    Time: 1560 ns  Iteration: 1  Instance: /encoder_tb
# Break in Module encoder_tb at C:/Users/Taewoo Kim/Desktop/Test/encoder_tb.sv line 74

VSIM 4>
```



2 Source code of the module

2.1 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
```

2.2 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
```

2.3 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
```

3 Quartus compilation report

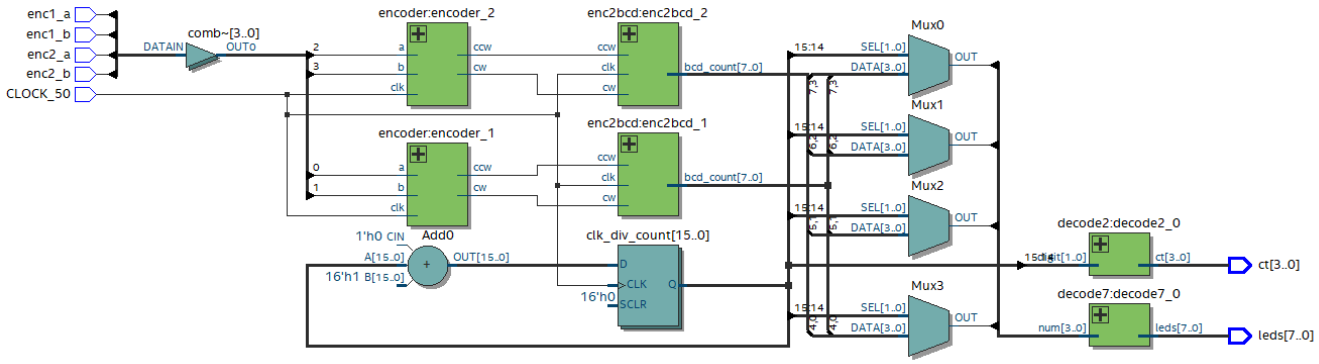
Flow Summary

<<Filter>>

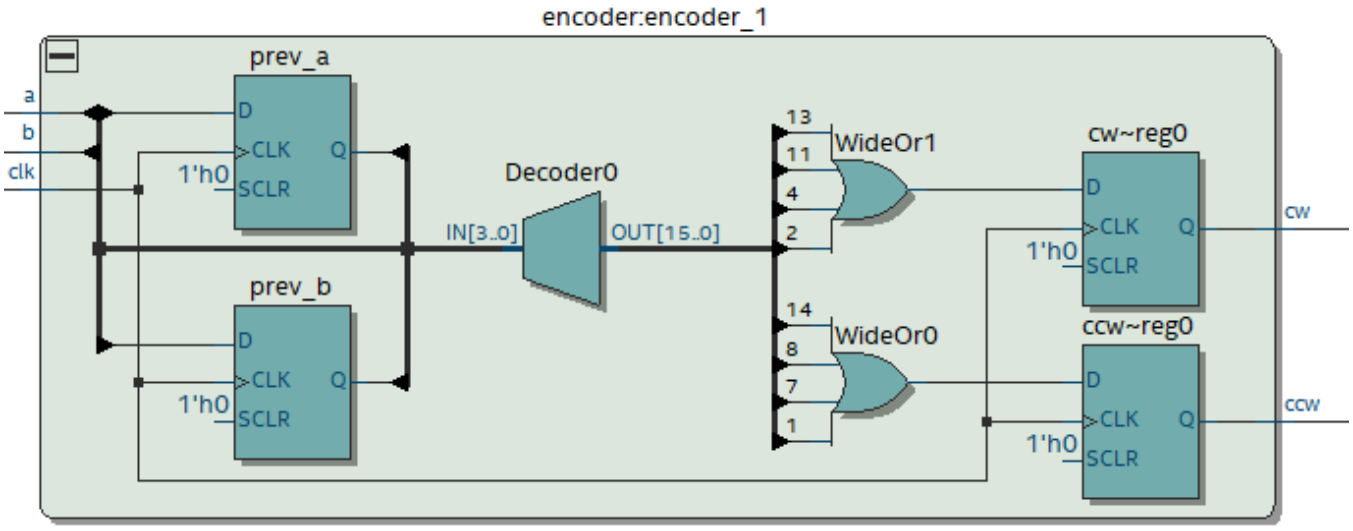
Flow Status	Successful - Sun Feb 02 18:59:18 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	lab2
Top-level Entity Name	lab2
Family	Cyclone V
Device	5CSEMA4U23C6
Timing Models	Final
Logic utilization (in ALMs)	39 / 15,880 (< 1 %)
Total registers	67
Total pins	17 / 314 (5 %)
Total virtual pins	0
Total block memory bits	0 / 2,764,800 (0 %)
Total DSP Blocks	0 / 84 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 5 (0 %)
Total DLLs	0 / 4 (0 %)

4 RTL Netlist

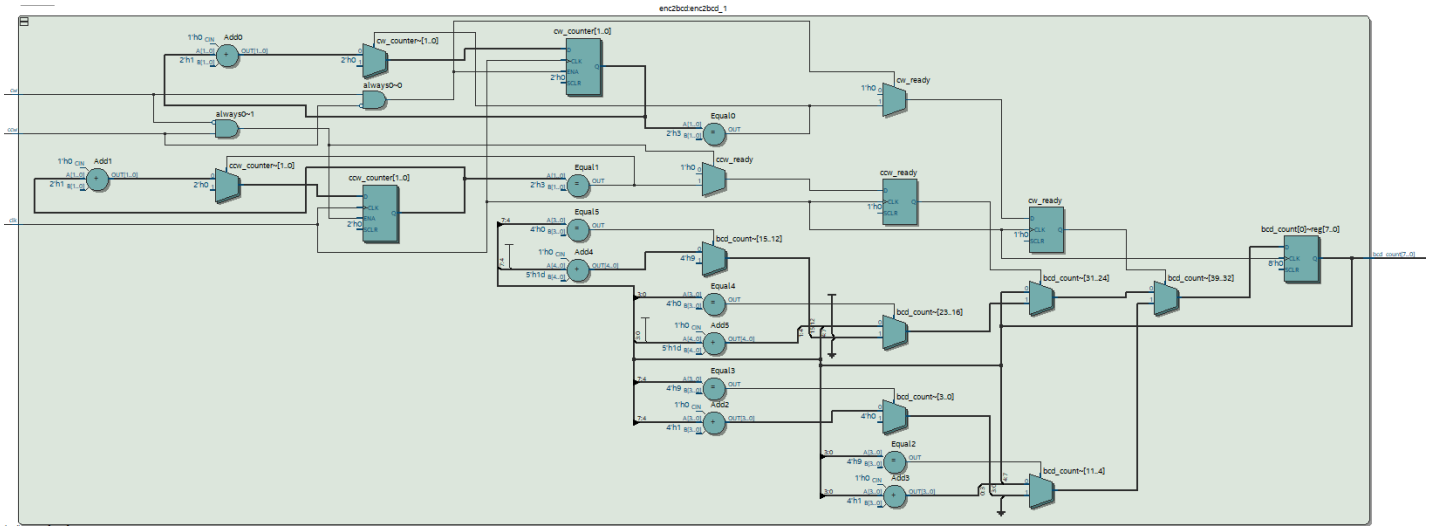
4.1 Overall view



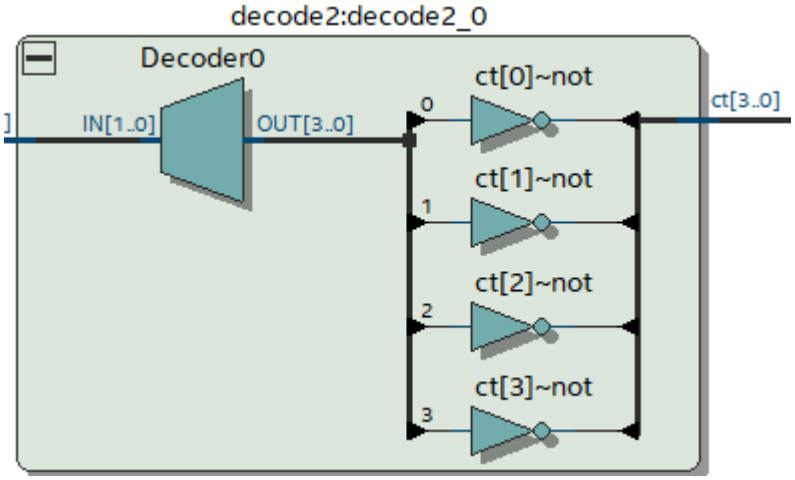
4.2 encoder



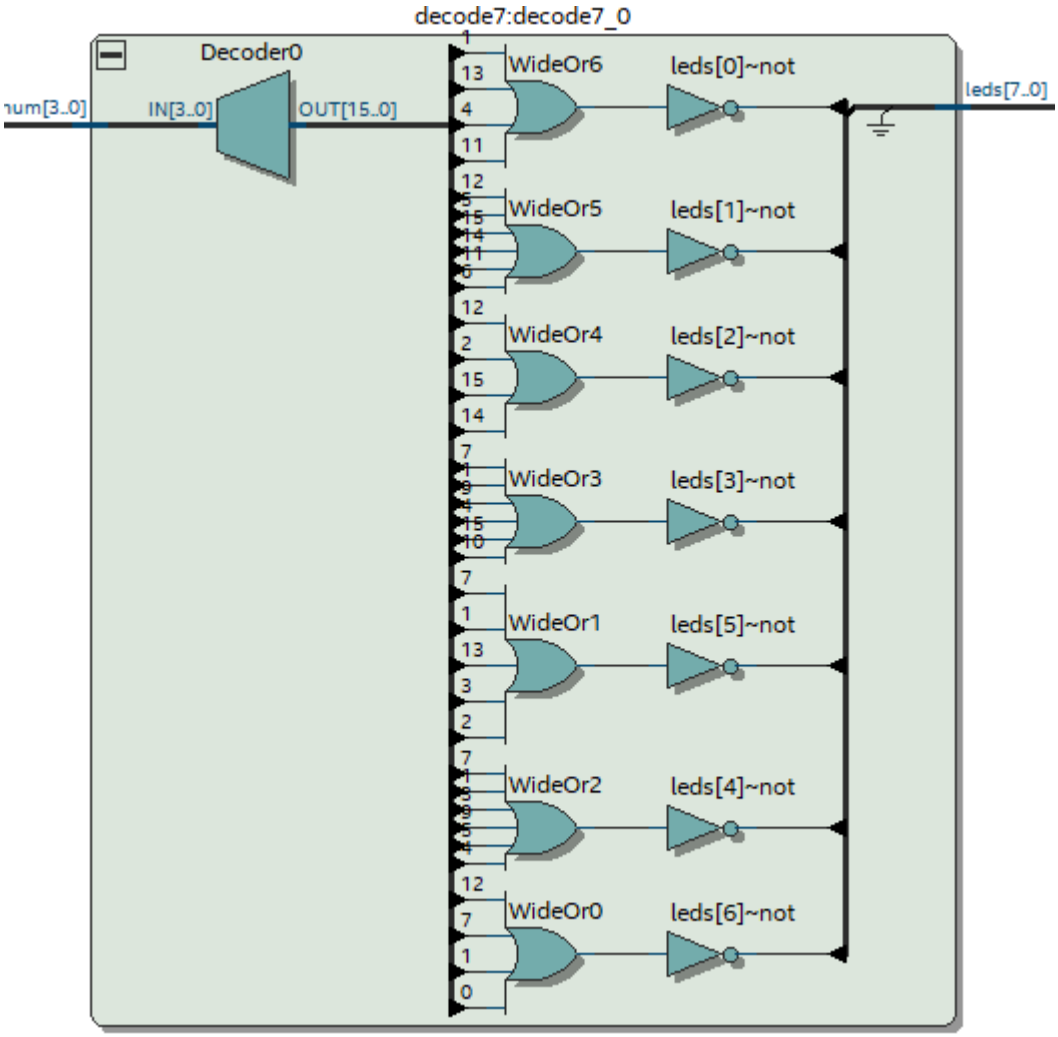
4.3 enc2bcd



4.4 decode2



4.5 decode7


decode7:decode7_0
Decoder0
num[3..0]
IN[3..0]
OUT[15..0]
WideOr6
WideOr5
WideOr4
WideOr3
WideOr1
WideOr2
WideOr0
leds[0]~not
leds[1]~not
leds[2]~not
leds[3]~not
leds[5]~not
leds[4]~not
leds[6]~not
leds[7..0]