A blue square with white text

Description automatically generated

ELEX 7660: Digital System Design

Lab 4

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

Table of Contents

[1 Screenshot of the simulations 3](#_Toc190639000)

[2 Source code of the module 4](#_Toc190639001)

[2.1 Lab4.sv code 4](#_Toc190639002)

[2.2 adcinterface.sv code 5](#_Toc190639003)

[2.3 enc2chan.sv 8](#_Toc190639004)

[3 Quartus compilation report 10](#_Toc190639005)

[4 RTL Netlist 11](#_Toc190639006)

[4.1 Overall view (lab4 module) 11](#_Toc190639007)

[4.2 enc2chan.sv 11](#_Toc190639008)

[4.3 adcinterface.sv 11](#_Toc190639009)

# Screenshot of the simulations

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Zoomed in pictures:

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

# Source code of the module

## Lab4.sv code

|  |
| --- |
| // File: lab4.sv  // Description: ELEX 7660 lab4 top-level module.  //              We can use enc to select analog channel  //              and use adc to convert analog to digital bits and display on  //              7-segments  // Author: Taewoo Kim  // Date: 2025-02-10  module lab4 (           input logic CLOCK\_50,           // 50 MHz clock input           (\* altera\_attribute = "-name WEAK\_PULL\_UP\_RESISTOR ON" \*)           input logic enc1\_a, enc1\_b,     // Encoder 1 signals (A and B channels)           input logic s1,             // Pushbuttons for reset (s1) and turning On/Off           input logic ADC\_SDO,           output logic [7:0] leds,        // 7-segment LED display output           output logic ADC\_CONVST, ADC\_SCK, ADC\_SDI,           output logic [3:0] ct           // Digit cathodes for the 7-segment display           );     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 enc1\_cw, enc1\_ccw; // enc1 cw/ccw variables     logic [11:0] adc\_values; // variable to store adc vals     logic [2:0] channel\_values; // variable to store channel vals    // instantiate modules to implement design    decode2 decode2\_0 (.digit, .ct);    // instantiate decode 7 to display digit on leds    decode7 decode7\_0 (.num(disp\_digit), .leds);      // instantiate encoders     encoder encoder\_1 (.clk(digit), .a(enc1\_a), .b(enc1\_b), .cw(enc1\_cw), .ccw(enc1\_ccw));    // instantiate encoder to channel     enc2chan enc2chan\_1 (.clk(clk\_div\_count[13]), .reset\_n(s1), .cw(enc1\_cw), .ccw(enc1\_ccw), .chan(channel\_values));       // instantiate adc interface module     adcinterface adcinterface\_1 (.clk(clk\_div\_count[13]), .reset\_n(s1), .chan(channel\_values), .result(adc\_values), .ADC\_SDO, .ADC\_CONVST, .ADC\_SCK, .ADC\_SDI);       // 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 = adc\_values[3:0];    // display adc values digit 0            2'b01: disp\_digit = adc\_values[7:4];    // display adc values digit 1            2'b10: disp\_digit = adc\_values[11:8];   // display adc values digit 2         2'b11: disp\_digit = {1'b0, channel\_values}; // display # of analog channel         default: disp\_digit = 4'd0;       endcase    end  endmodule |

## adcinterface.sv code

|  |
| --- |
| // File: adcinterface.sv   // Description: adcinterface module to integrate ltc2308 adc   // Author: Taewoo Kim   // Date: 2025-02-10  module adcinterface(                          input logic clk, reset\_n,   // clock and reset signals: 'clk' for timing, 'reset\_n' active low reset.                          input logic [2:0] chan,     // ADC channel to sample, 3 bits wide to select channel and configuration bits.                          output logic [11:0] result, // 12-bit ADC result output from the ADC conversion.                            // ltc2308 signals                          output logic ADC\_CONVST, ADC\_SCK, ADC\_SDI,                          input logic ADC\_SDO                          );      // patterns:      //      // ADC\_CONVST   Output  Conversion Start – Triggers ADC conversion.      // ADC\_SCK      Output  Serial Clock – Clocks data in/out (like SPI).      // ADC\_SDI      Output  Serial Data Input – Sends config commands to ADC.      // ADC\_SDO      Input   Serial Data Output – Receives 12-bit ADC result.      /\*          adc\_counter ADC\_CONVST  ADC\_SCK     Event          0           1           0           Start ADC conversion (Pulse CONVST).          1           0           0           Hold CONVST low for 1 cycle.          2-13        0           clk         Toggle SCK for 12 cycles (data transfer).          14-15       0           0           Wait before restart.      \*/      logic [3:0] adc\_counter;  // 4-bit counter to keep track of the ADC state sequence.      logic [5:0] SDI\_init;     // 6-bit register used to hold and shift out configuration bits to the ADC.      logic [11:0] SDO\_temp;    // Temporary 12-bit register to hold serial data received from ADC before finalizing.      // Combinational block: defines outputs ADC\_SDI, ADC\_CONVST, and ADC\_SCK based on adc\_counter and clock.      always\_comb begin          ADC\_SDI = SDI\_init[5];   // Drive ADC\_SDI with the most significant bit of the configuration register.          ADC\_CONVST = (adc\_counter == 4'd0) ? 1'b1 : 1'b0;  // Generate a high pulse on ADC\_CONVST when adc\_counter is 0, triggering a conversion.          // For ADC\_SCK, during states 2 to 13, output the system clock to drive ADC\_SCK; otherwise, hold it low.          ADC\_SCK = (adc\_counter >= 2 && adc\_counter <= 13) ? clk : 1'b0;      end      // Sequential block: Negative edge of clock or negative reset, updating the adc\_counter.      always\_ff @(negedge clk, negedge reset\_n) begin          if(~reset\_n) begin              adc\_counter <= 4'd0;  // On reset (active low), initialize adc\_counter to 0.          end else adc\_counter <= adc\_counter + 1'b1;  // Otherwise, increment the adc\_counter by 1 on every negative edge of clk.      end      /\*          S/D = SINGLE-ENDED/DIFFERENTIAL BIT          O/S = ODD/SIGN BIT          S1 = ADDRESS SELECT BIT 1          S0 = ADDRESS SELECT BIT 0          UNI = UNIPOLAR/BIPOLAR BIT          SLP = SLEEP MODE BIT      \*/      // Sequential block: Sets up the configuration bits for the ADC (SDI\_init) based on the ADC\_CONVST and adc\_counter.      always\_ff @(posedge ADC\_CONVST, negedge ADC\_SCK) begin          if (ADC\_CONVST) begin              // On the rising edge of ADC\_CONVST, load SDI\_init with the configuration bits:              // Bit breakdown:              //   - Single-Ended = 1,              //   - Odd bit is taken from chan[0],              //   - Next two bits (S1:S0) from chan[2:1],              //   - Unipolar = 1,              //   - Sleep = 0.              SDI\_init <= { 1'b1, chan[0], chan[2:1], 1'b1, 1'b0 };          end          else if (adc\_counter >= 4'd2 && adc\_counter <= 4'd7) begin              // Between adc\_counter values 2 and 7, shift SDI\_init left by one bit.              // The left shift moves the current MSB out and introduces a 0 at the LSB.              SDI\_init <= {SDI\_init[4:0], 1'b0};          end      end      // Sequential block: Captures ADC\_SDO data into SDO\_temp on the positive edge of ADC\_SCK or negative reset.      always\_ff @(posedge ADC\_SCK, negedge reset\_n) begin          if (~reset\_n) begin              SDO\_temp <= 12'b0;  // On reset, clear the temporary data register.          end          else if (adc\_counter >= 2 && adc\_counter <= 13) begin              // During the data transfer phase (adc\_counter from 2 to 13), shift in ADC\_SDO bit by bit.              // The previous 11 bits in SDO\_temp are shifted left, and the new bit is concatenated at LSB.              SDO\_temp <= {SDO\_temp[10:0], ADC\_SDO};          end      end      // Sequential block: At the negative edge of ADC\_SCK, finalize and store the conversion result.      always\_ff @(negedge ADC\_SCK, negedge reset\_n) begin          if (~reset\_n) begin              result <= 12'b0;  // On reset, clear the final ADC result.          end          else if (adc\_counter == 13) begin              result <= SDO\_temp;  // When adc\_counter reaches 13, assign the captured 12-bit data to result.          end      end  endmodule |

## enc2chan.sv

|  |
| --- |
| // File: enc2chan.sv  // Description: Module using cw or ccw inputs to generate the desired channel scale  // Author: Taewoo Kim  // Date: 2025-02-10  module enc2chan (      input  logic       cw, ccw,       // Encoder signals: cw and ccw      output logic [2:0] chan,          // Desired channel output      input  logic       reset\_n, clk   // Asynchronous reset and clock  );      // Array of channel scales (frequency levels)      logic [2:0] chan\_array [7:0] = '{3'b111, 3'b110, 3'b101, 3'b100, 3'b011, 3'b010, 3'b001, 3'b000};      // Count threshold (pulse counter)      localparam logic [1:0] PULSE\_COUNTER = 2'b11;      // Pulse counters for cw and ccw signals      logic [1:0] cw\_counter, ccw\_counter;      // Main channel index counter      logic [2:0] counter;      // Registers to store previous states for edge detection      logic cw\_prev, ccw\_prev;      // Edge detection: capture previous cw and ccw values      always\_ff @(posedge clk, negedge reset\_n) begin          // initialize cw\_prev and ccw\_prev          if (!reset\_n) begin              cw\_prev  <= 1'b0;              ccw\_prev <= 1'b0;          // every clock edge store cw/ccw to cw\_prev/ccw\_prev          end else begin              cw\_prev  <= cw;              ccw\_prev <= ccw;          end      end      // Generate one-shot pulses on the rising edge of cw or ccw,      // and ensure they are mutually exclusive      logic cw\_edge, ccw\_edge;      assign cw\_edge  = (cw && !cw\_prev) && !ccw;      assign ccw\_edge = (ccw && !ccw\_prev) && !cw;      // Update counters on detected rising edges      always\_ff @(posedge clk, negedge reset\_n) begin          if (!reset\_n) begin              cw\_counter   <= 2'b0;              ccw\_counter  <= 2'b0;              counter      <= 3'b0;          end else begin              // CW handling: if a rising edge is detected on cw              if (cw\_edge) begin                  if (cw\_counter == PULSE\_COUNTER) begin                      cw\_counter <= 2'b0;                      counter    <= counter + 1; // Increment channel index                  end else begin                      cw\_counter <= cw\_counter + 1;                  end              end              // CCW handling: if a rising edge is detected on ccw              else if (ccw\_edge) begin                  if (ccw\_counter == PULSE\_COUNTER) begin                      ccw\_counter <= 2'b0;                      counter     <= counter - 1; // Decrement channel index                  end else begin                      ccw\_counter <= ccw\_counter + 1;                  end              end              // Optionally, you might reset the counters if no edge is detected:              else begin                  cw\_counter  <= cw\_counter;                  ccw\_counter <= ccw\_counter;              end          end      end      // Assign the channel output based on the channel index      assign chan = chan\_array[counter];  endmodule |

# Quartus compilation report

A screenshot of a computer program

AI-generated content may be incorrect.

# RTL Netlist

## Overall view (lab4 module)

A diagram of a computer diagram

AI-generated content may be incorrect.

## enc2chan.sv

A diagram of a circuit

AI-generated content may be incorrect.

## adcinterface.sv

A diagram of a computer

AI-generated content may be incorrect.