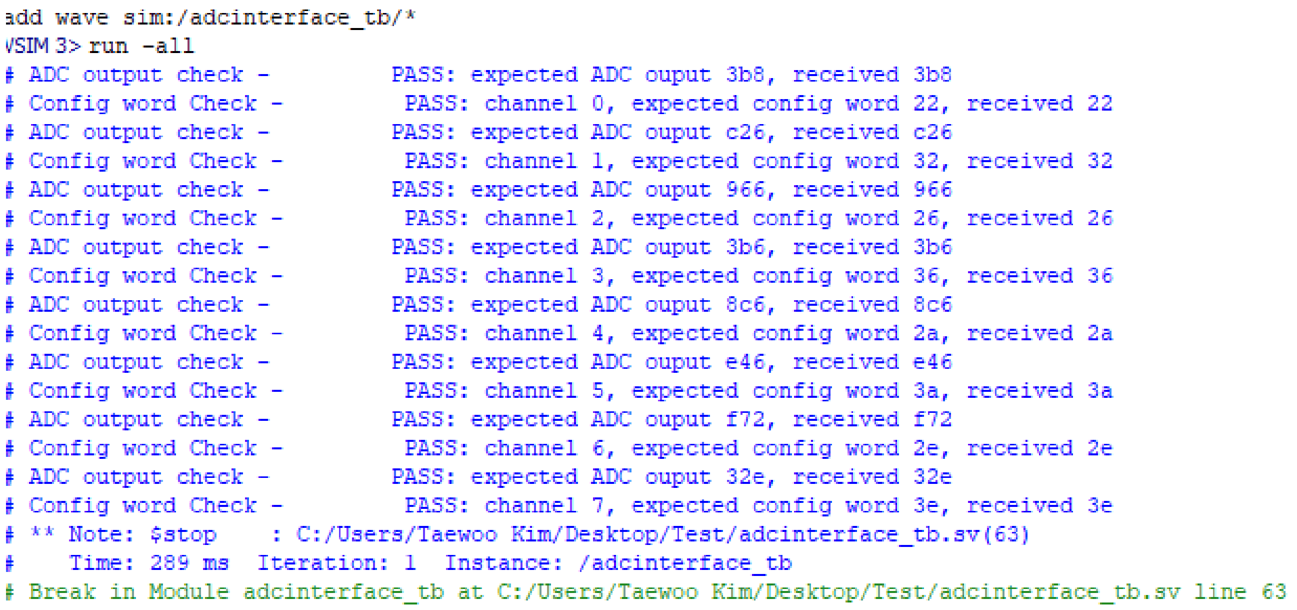
# ELEX 7660: Digital System Design

*Lab 4*

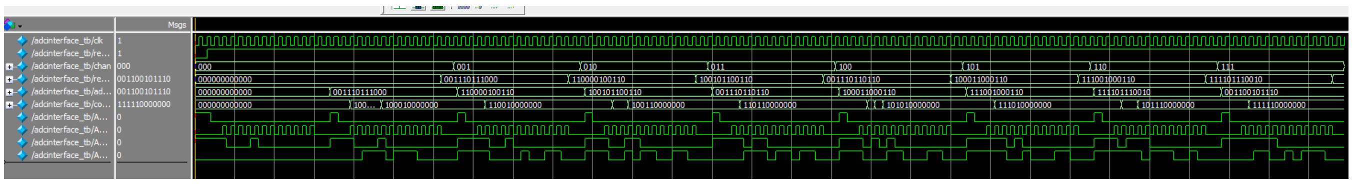
| | | |
|---|---|---|
| Student Name: Taewoo Kim | Student Number: A01284763 | Set: T |

# Table of Contents

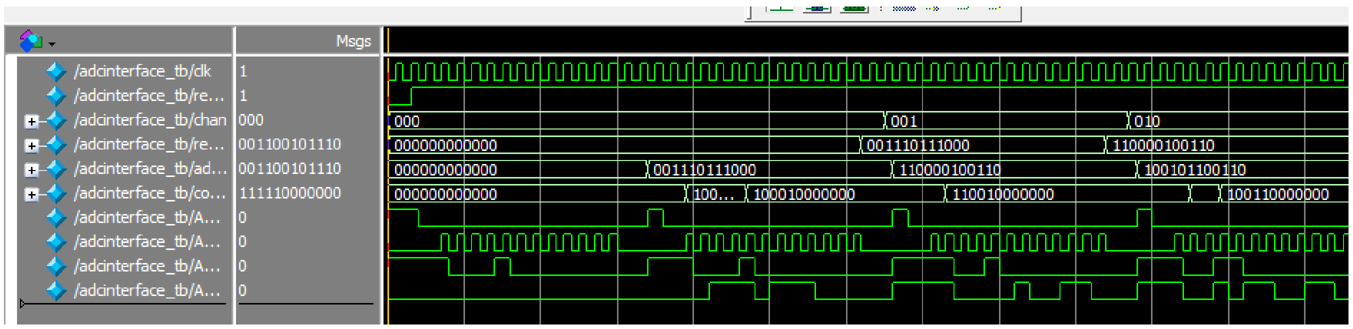
## 1 Screenshot of the simulations

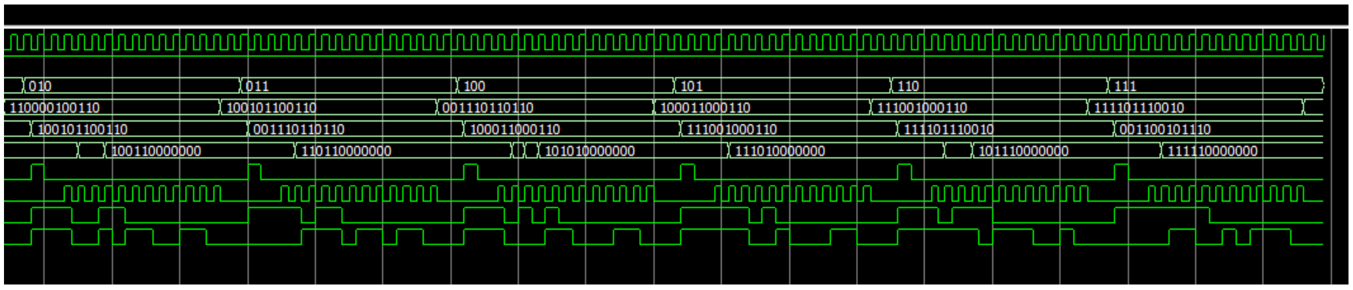


```
add wave sim:/adcinterface_tb/*
VSIM 3> run -all
# ADC output check -        PASS: expected ADC ouput 3b8, received 3b8
# Config word Check -        PASS: channel 0, expected config word 22, received 22
# ADC output check -        PASS: expected ADC ouput c26, received c26
# Config word Check -        PASS: channel 1, expected config word 32, received 32
# ADC output check -        PASS: expected ADC ouput 966, received 966
# Config word Check -        PASS: channel 2, expected config word 26, received 26
# ADC output check -        PASS: expected ADC ouput 3b6, received 3b6
# Config word Check -        PASS: channel 3, expected config word 36, received 36
# ADC output check -        PASS: expected ADC ouput 8c6, received 8c6
# Config word Check -        PASS: channel 4, expected config word 2a, received 2a
# ADC output check -        PASS: expected ADC ouput e46, received e46
# Config word Check -        PASS: channel 5, expected config word 3a, received 3a
# ADC output check -        PASS: expected ADC ouput f72, received f72
# Config word Check -        PASS: channel 6, expected config word 2e, received 2e
# ADC output check -        PASS: expected ADC ouput 32e, received 32e
# Config word Check -        PASS: channel 7, expected config word 3e, received 3e
# ** Note: $stop    : C:/Users/Taewoo Kim/Desktop/Test/adcinterface_tb.sv(63)
#    Time: 289 ms  Iteration: 1  Instance: /adcinterface_tb
# Break in Module adcinterface_tb at C:/Users/Taewoo Kim/Desktop/Test/adcinterface_tb.sv line 63
```



Zoomed in pictures:





## 2 Source code of the module

### 2.1 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_v
alues));

   // 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
```

## 2.2 adcinterface.sv code

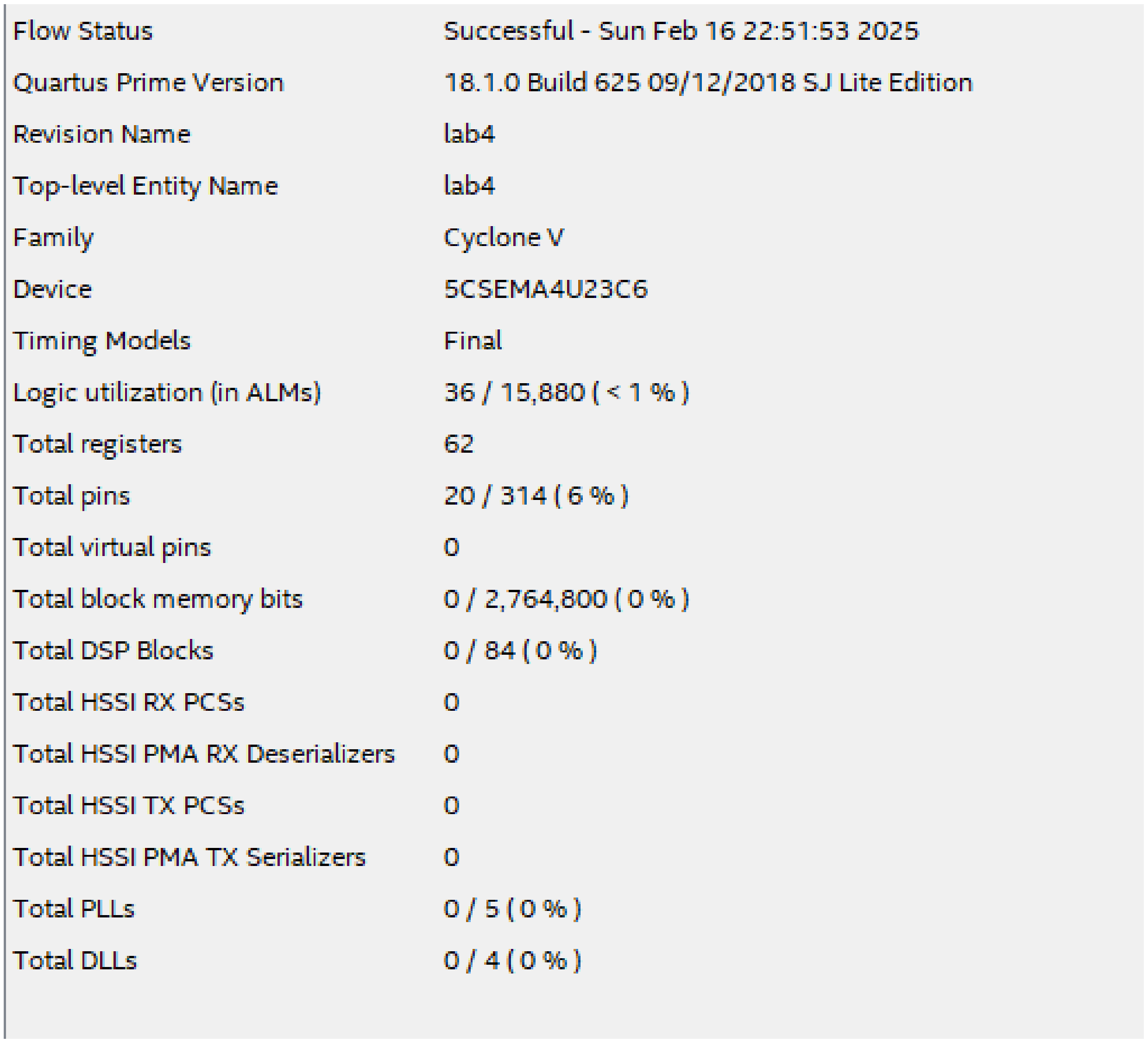
```
// File: adcinterface.sv
 // Description: adcinterface module to integrate ltc2308 adc
 // Author: Taewoo Kim
 // Date: 2025-02-10

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
```

## 2.3 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
```

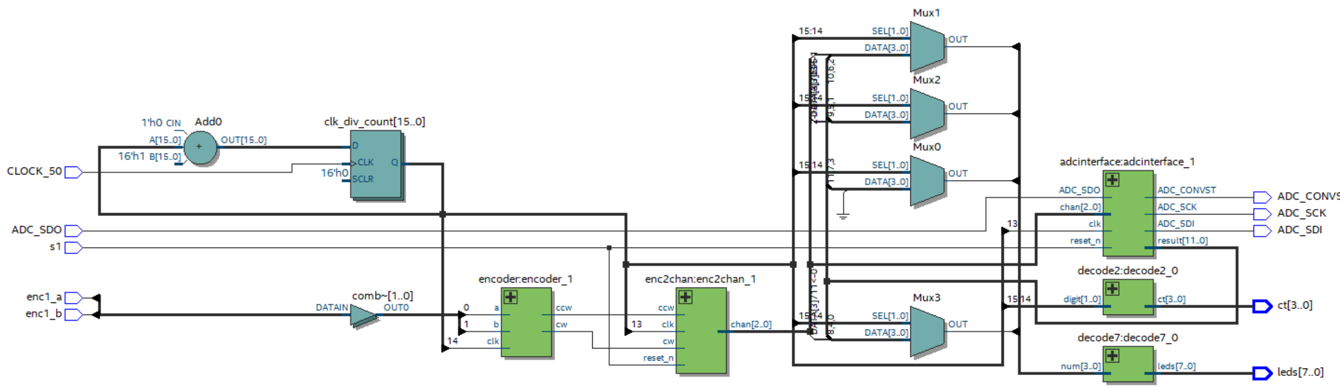
## 3 Quartus compilation report



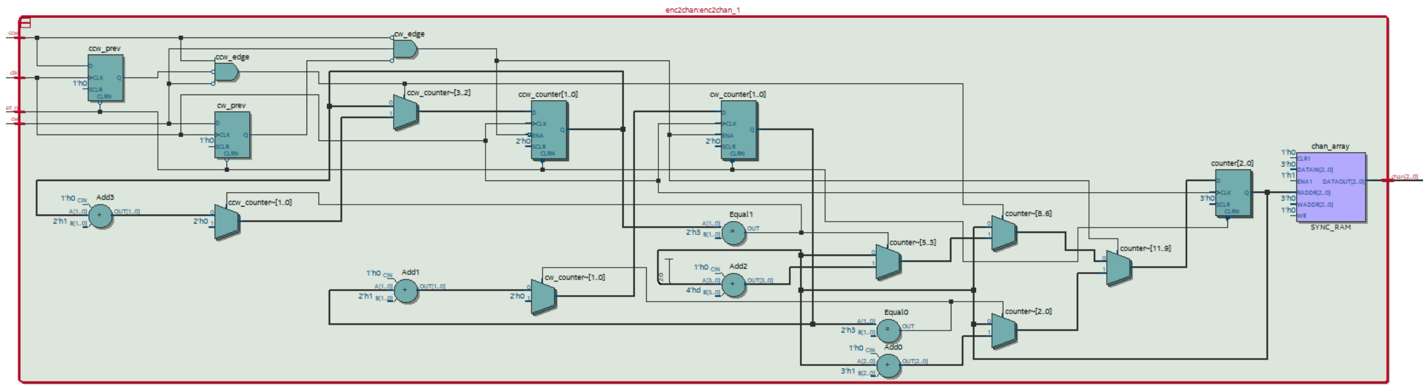
| | |
|---|---|
| Flow Status | Successful - Sun Feb 16 22:51:53 2025 |
| Quartus Prime Version | 18.1.0 Build 625 09/12/2018 SJ Lite Edition |
| Revision Name | lab4 |
| Top-level Entity Name | lab4 |
| Family | Cyclone V |
| Device | 5CSEMA4U23C6 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 36 / 15,880 ( < 1 % ) |
| Total registers | 62 |
| Total pins | 20 / 314 ( 6 % ) |
| 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

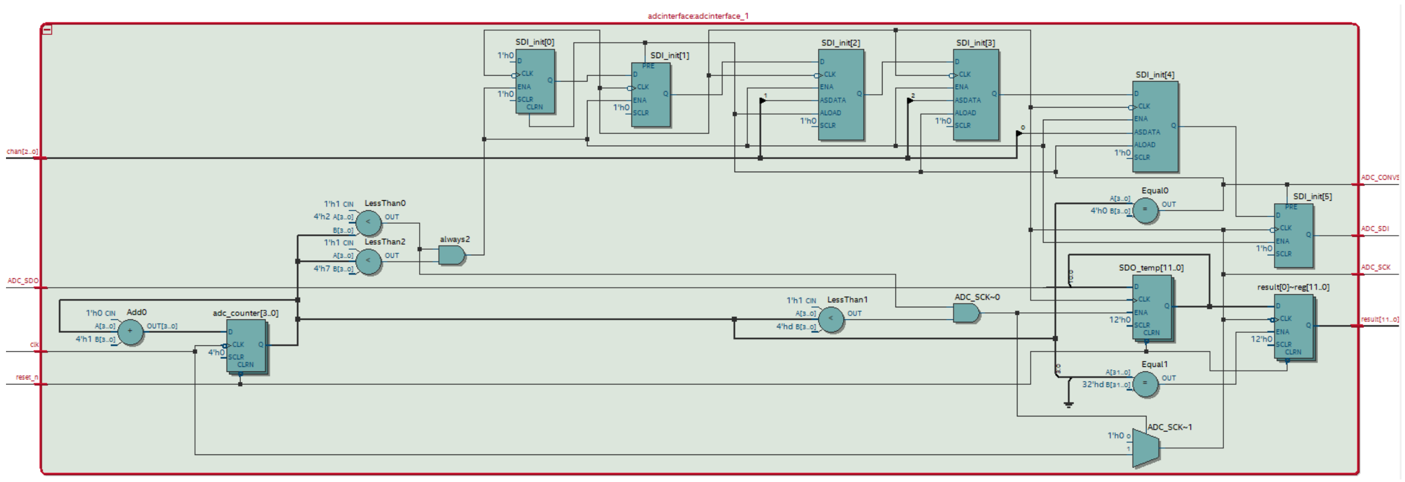
## 4.1 Overall view (lab4 module)



## 4.2 enc2chan.sv



## 4.3 adcinterface.sv


adcinterface:adcinterface_1
SDI_init[0]
SDI_init[1]
SDI_init[2]
SDI_init[3]
SDI_init[4]
SDI_init[5]
chan[2..0]
LessThan0
LessThan2
always2
ADC_SDO
Add0
adc_counter[3..0]
LessThan1
ADC_SCK~0
SDO_temp[11..0]
Equal0
Equal1
result[0]~reg[11..0]
ADC_SCK~1
ADC_CONVST
ADC_SDI
ADC_SCK
result[11..0]
clk
reset_n