



ELEX 7660: Digital System Design

Lab 3

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

Table of Contents

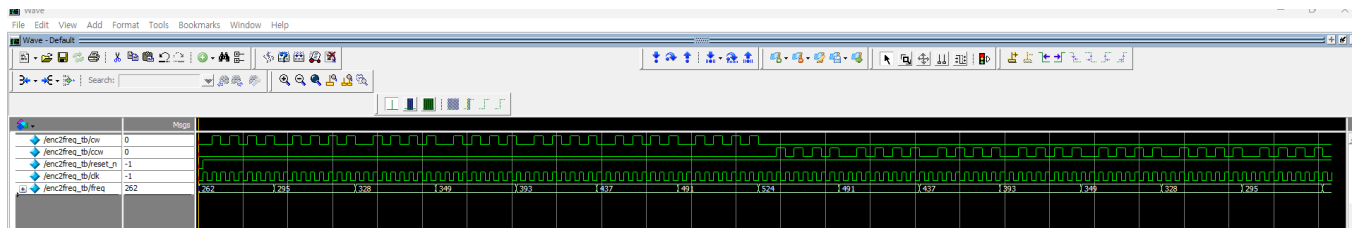
1	Screenshot of the simulations.....	3
2	Source code of the module	3
2.1	Lab3.sv code	3
2.2	enc2freq.sv code.....	5
2.3	enc2freq_tb.sv	6
2.4	tongen.sv	8
3	Quartus compilation report	9
4	RTL Netlist.....	9
4.1	Overall view	9
4.2	enc2freq.sv	10
4.3	tongen.sv	10
4.4	decode2.sv.....	10
4.5	decode7.sv.....	11
4.6	encoder.sv	12

1 Screenshot of the simulations

```

VSIM 3> run -all
# [PASS] Initial frequency correct:      262
# [PASS] CW step      1: Frequency correct (      295)
# [PASS] CW step      2: Frequency correct (      328)
# [PASS] CW step      3: Frequency correct (      349)
# [PASS] CW step      4: Frequency correct (      393)
# [PASS] CW step      5: Frequency correct (      437)
# [PASS] CW step      6: Frequency correct (      491)
# [PASS] CW step      7: Frequency correct (     524)
# [PASS] CCW step      6: Frequency correct (      491)
# [PASS] CCW step      5: Frequency correct (      437)
# [PASS] CCW step      4: Frequency correct (      393)
# [PASS] CCW step      3: Frequency correct (      349)
# [PASS] CCW step      2: Frequency correct (      328)
# [PASS] CCW step      1: Frequency correct (      295)
# [PASS] CCW step      0: Frequency correct (      262)
# Testbench completed.
# ** Note: $stop      : C:/Users/Taewoo Kim/Desktop/Test/enc2freq_tb.sv(70)
#   Time: 1265 ps  Iteration: 1  Instance: /enc2freq_tb
# Break in Module enc2freq_tb at C:/Users/Taewoo Kim/Desktop/Test/enc2freq_tb.sv line 70

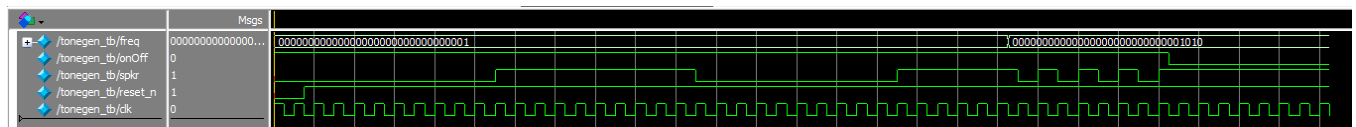
```



```

ModelSim> vsim -gui work.tonegen_tb
# vsim -gui work.tonegen_tb
# Start time: 22:02:15 on Feb 08, 2025
# Loading sv_std.std
# Loading work.tonegen_tb
# Loading work.tonegen
add wave sim:/tonegen_tb/*
VSIM 7> run -all
# ** Note: $stop      : C:/Users/Taewoo Kim/Desktop/Test/tonegen_tb.sv(40)
#   Time: 2625 ms  Iteration: 1  Instance: /tonegen_tb
# Break in Module tonegen_tb at C:/Users/Taewoo Kim/Desktop/Test/tonegen_tb.sv line 40

```



2 Source code of the module

2.1 Lab3.sv code

```

// File: lab3.sv
// Description: ELEX 7660 lab3 top-level module.
//           We can scale the buzzer to make different frequency sounds
//           and 7-segment display the corresponding frequency.
//           display module.
// Author: Taewoo Kim
// Date: 2025-02-02

```

```

module lab3 ( input logic CLOCK_50,          // 50 MHz clock
              input logic s1,
              input logic s2,
              (* 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
              output logic spkr ) ;        // speaker

  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 [31:0] desired_freq;
  logic enc1_cw, enc1_ccw;

  // 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));
  // instantiate encoder to bcd
  enc2freq enc2freq_1
(.cw(enc1_cw), .ccw(enc1_ccw), .freq(desired_freq), .reset_n(s1), .clk(CLOCK_50));
  tonegen tonegen_1
( .freq(desired_freq), .onOff(s2), .spkr(spkr), .reset_n(s1), .clk(CLOCK_50));

  // 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 = desired_freq[3:0]; // if digit is 0
      2'b01: disp_digit = desired_freq[7:4]; // if digit is 1
      2'b10: disp_digit = desired_freq[11:8]; // if digit is 2
    endcase
  end

```

```

        2'b11: disp_digit = desired_freq[15:12]; // if digit is 3
        default: disp_digit = 4'b0000;         // Default or error value
    endcase
end

endmodule

```

2.2 enc2freq.sv code

```

// File: enc2freq.sv
// Description: enc2freq module using cw or ccw generate the desired freq scale
// Author: Taewoo Kim
// Date: 2025-02-02

module enc2freq ( input logic cw, ccw,          // input cw and ccw
                  output logic [31:0] freq,     // desired frequency
                  input logic reset_n, clk );    // reset and clock

    // array of frequency scale
    logic [31:0] freq_array [7:0] = '{524, 491, 437, 393, 349, 328, 295, 262};

    // Count until 3
    localparam PULSE_COUNTER = 2'b11;

    // initialize the logic count (0~3 and wrap back)
    logic [1:0] cw_counter = 0, ccw_counter = 0;
    logic [2:0] 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, negedge reset_n) begin
        if (~reset_n) begin
            counter <= 0;
        end
        // CW handling check if it's mutually exclusive
        else 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_counter <= 0;
                counter <= counter + 1;
            // if it didn't reached, send 0 for cw_ready and count up the counter
            end else begin
                cw_counter <= cw_counter + 1;
            end
        end
    end
end

```

```

    // CCW handling (similar) check if it's mutually exclusive
    else 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_counter <= 0;
            counter <= counter - 1;
        // if it didn't reached, send 0 for ccw_ready and count up the counter
        end else begin
            ccw_counter <= ccw_counter + 1;
        end
    end
end

// assign the desired frequency
assign freq = freq_array[counter];

endmodule

```

2.3 enc2freq_tb.sv

```

// File: enc2freq_tb.sv
// Description: This is simple testbench to test enc2freq module
// Author: Taewoo Kim, ChatGPT (got some help)
// Date: 2025-02-02

module enc2freq_tb();
    // Testbench signals
    logic cw, ccw, reset_n, clk; // Clockwise, counter-clockwise, reset, and clock
    signals
    logic [31:0] freq; // Output frequency signal

    // Instantiate the DUT (Device Under Test)
    enc2freq dut ( .cw(cw), .ccw(ccw), .freq(freq), .reset_n(reset_n), .clk(clk) );

    // Clock generation
    always #5 clk = ~clk; // 10 ns period (100 MHz)

    // Frequency array (expected values)
    logic [31:0] expected_freq [7:0] = '{524, 491, 437, 393, 349, 328, 295, 262};

    // Task to generate pulses, chatGPT generated send_pulses task.
    task send_pulses(input logic dir);
        integer i;
        for (i = 0; i < 4; i = i + 1) begin // Generate 4 pulses per call
            @(posedge clk); // Wait for a clock edge
            if (dir) cw = 1; else ccw = 1; // Set appropriate direction signal
            @(posedge clk);
        end
    endtask
endmodule

```

```

        cw = 0; ccw = 0; // Reset signals
    end
endtask

// Test sequence
initial begin
    // Initialize signals
    clk = 0;
    reset_n = 0;
    cw = 0;
    ccw = 0;
    @(posedge clk);
    reset_n = 1; // Release reset after one clock cycle

    // Verify initial frequency
    if (freq != expected_freq[0]) // Check if initial frequency matches
expected value
        $display("[FAIL] Initial frequency incorrect: Expected %d, Got %d",
expected_freq[0], freq);
    else
        $display("[PASS] Initial frequency correct: %d", freq);

    // Test CW direction
    for (int i = 1; i < 8; i++) begin
        send_pulses(1); // Send pulses in CW direction
        @(posedge clk); // Wait for one clock cycle to allow frequency update
        if (freq != expected_freq[i])
            $display("[FAIL] CW step %d: Expected %d, Got %d", i,
expected_freq[i], freq);
        else
            $display("[PASS] CW step %d: Frequency correct (%d)", i, freq);
    end

    // Test CCW direction
    for (int i = 6; i >= 0; i--) begin
        send_pulses(0); // Send pulses in CCW direction
        @(posedge clk); // Wait for one clock cycle to allow frequency update
        if (freq != expected_freq[i])
            $display("[FAIL] CCW step %d: Expected %d, Got %d", i,
expected_freq[i], freq);
        else
            $display("[PASS] CCW step %d: Frequency correct (%d)", i, freq);
    end

    // End simulation
    $display("Testbench completed.");
    $stop; // Halt simulation
end

```

```
endmodule
```

2.4 tongen.sv

```
// File: tongen.sv
// Description: simple tongen module to create the tone using
//             input as desired frequency.
// Author: Taewoo Kim
// Date: 2025-02-02

module tonegen #(parameter FCLK = 50000000)           // clock frequency, Hz
    ( input logic [31:0] freq,                        // frequency to output on speaker
      input logic onOff,                             // 1 -> generate output, 0-> no output
      output logic spkr,                             // speaker output
      input logic reset_n, clk);                     // reset and clock

    // initialize the counter and desired_freq variable
    logic [31:0] count;
    logic [31:0] desired_freq;

    // shift by 1 to multiply 2 to freq without using multiplication
    assign desired_freq = freq << 1;

    // using always_ff to count to get desired cycles
    always_ff @( posedge clk, negedge reset_n ) begin

        // rest the spkr and counter
        if(~reset_n) begin
            spkr <= 0;
            count <= 0;
        end

        // if tonegen is on start count until desired cycle
        else if (onOff) begin

            // count the counter using desired freq
            count <= count + desired_freq;
            // once it reached desired cycle toggle spkr to create tone
            if(count >= (FCLK-desired_freq)) begin
                spkr <= ~spkr;
                count <= 0;
            end

        end


    end

end

end
```

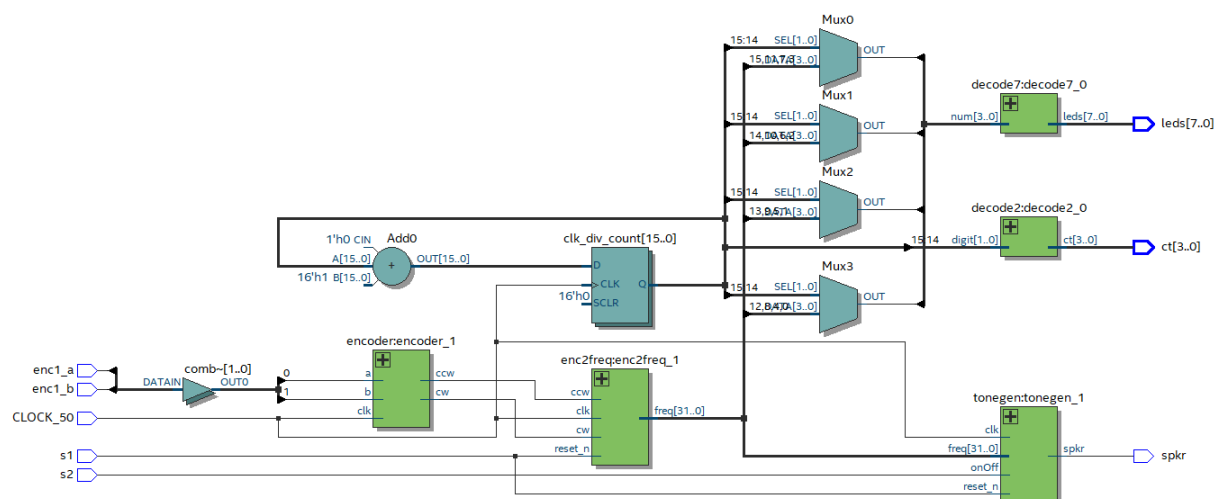

endmodule

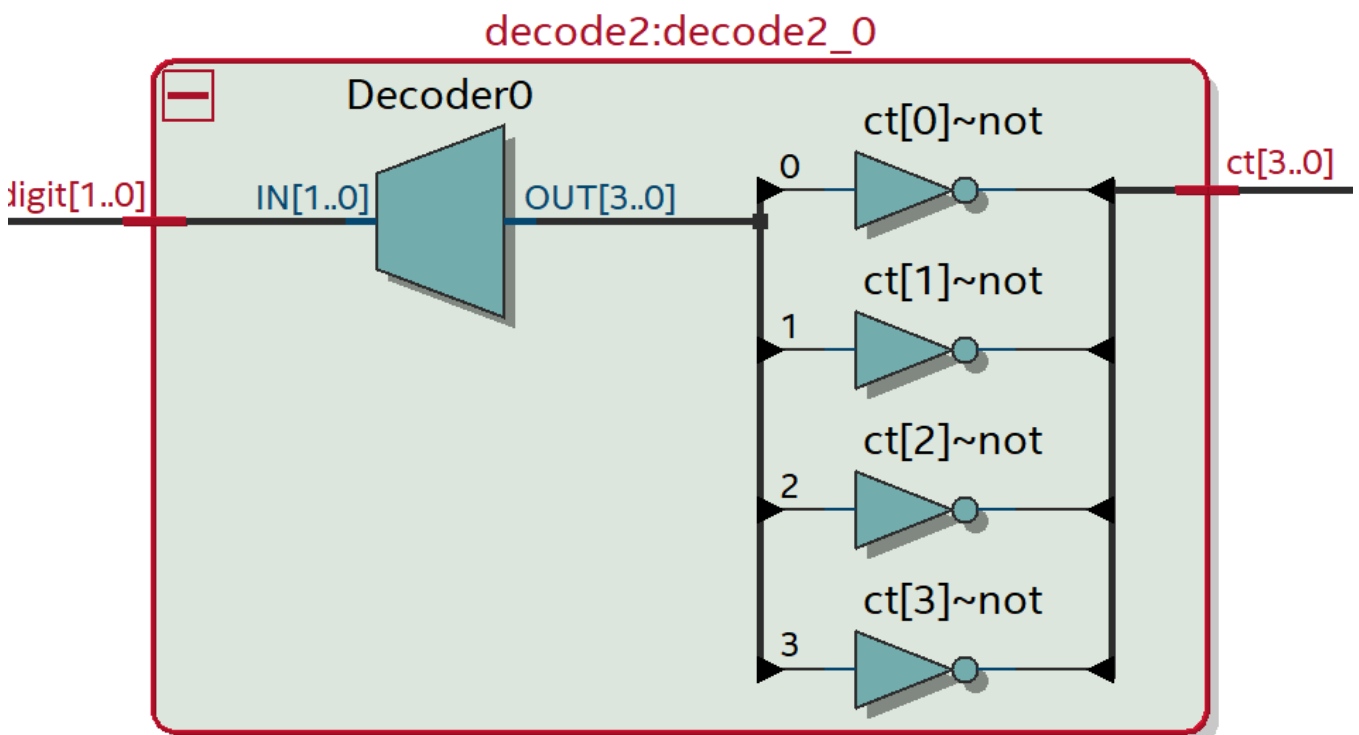
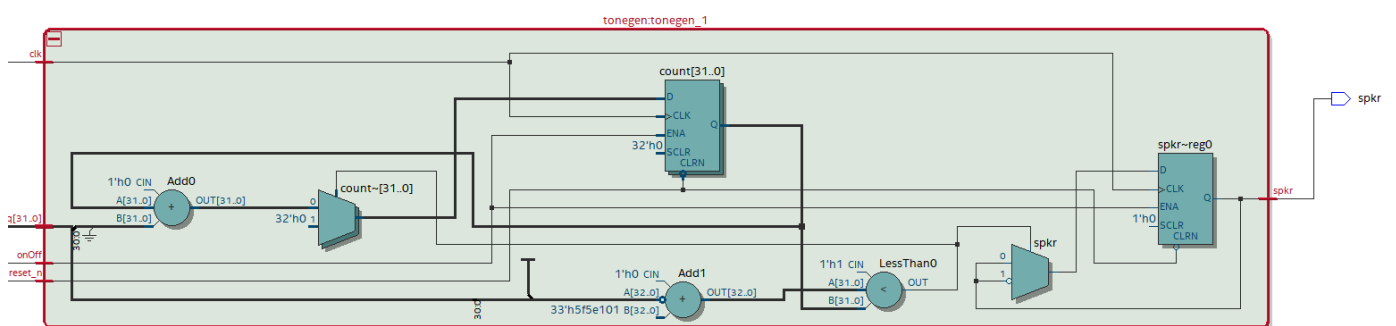
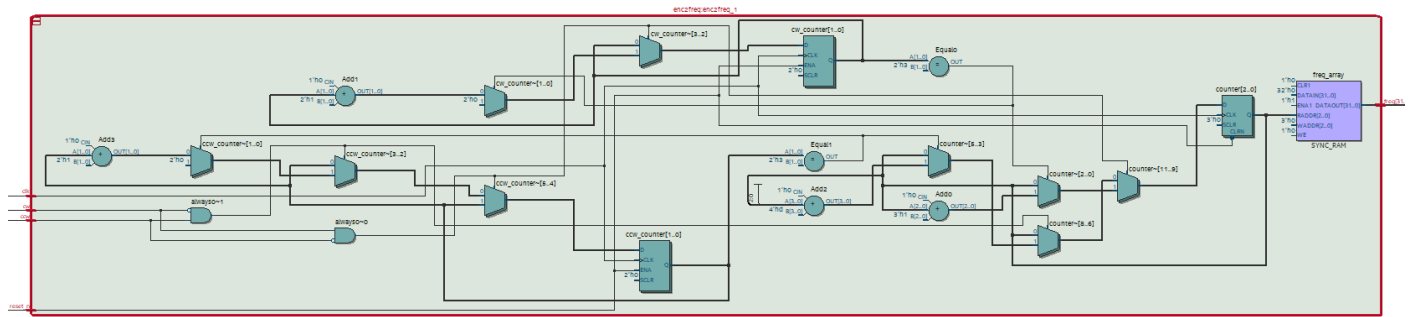
3 Quartus compilation report

Flow Summary	
 <<Filter>>	
Flow Status	Successful - Sat Feb 08 21:26:41 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	lab3
Top-level Entity Name	lab3
Family	Cyclone V
Device	5CSEMA4U23C6
Timing Models	Final
Logic utilization (in ALMs)	83 / 15,880 (< 1 %)
Total registers	63
Total pins	18 / 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 %)

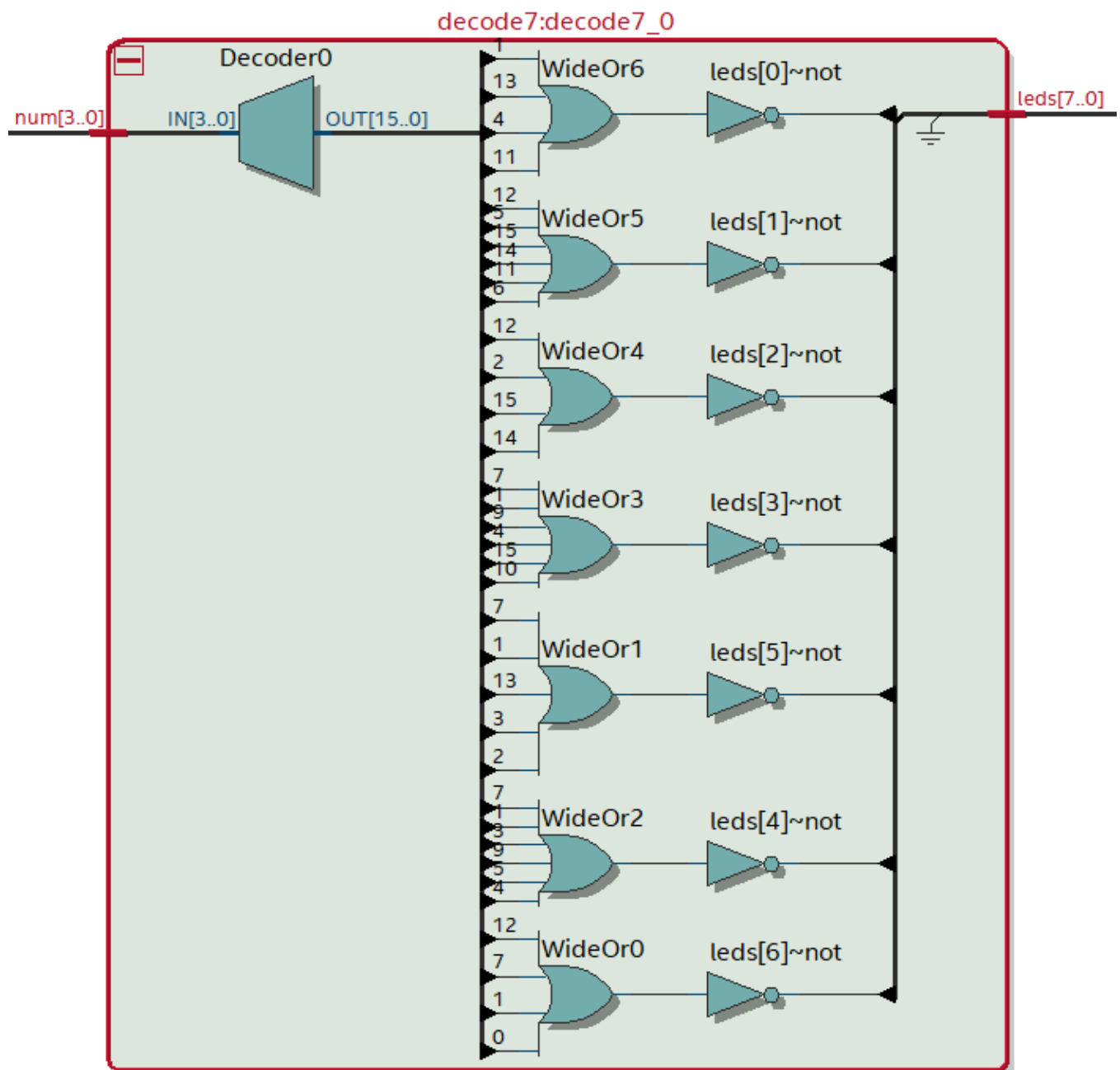
4 RTL Netlist

4.1 Overall view





4.5 decode7.sv



4.6 encoder.sv

