ELEX 7660

LAB 3 – TONE GENERATOR

Nicholas Scott AKA "White Cheddar"
A01255181 | JAN 28TH, 2024

Contents

Table of Figures

Prelab

I did a prelab that you probably wont' check.

tonegen.sv

I started with coding the tonegen.sv module. I implemented the following code:

```
// ELEX 7660 Lab 3
// Nicholas Scott AKA "White Cheddar"
// January 27th, 2024
// Instructor: Sweet Bobby T

module tonegen
  #( parameter FCLK )           // 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

    logic [31:0] count;
    logic prevOnOff;
    logic mute = 0;

    always_ff @( posedge clk) begin

        // latch and debounce for mute button
        prevOnOff <= onOff;
        if (!onOff && prevOnOff)
            mute <= ~mute;

        if (reset_n) begin                 // if  active-low reset not pressed
            count <= count + (freq<<1); // count up by twice the tone frequency
            // if count reaches clock frequency minus one increment
            if (count >= (FCLK - (freq<<1))) begin
                if (!mute)             // if mute is not pressed
                    spkr <= ~spkr;     // toggle speaker every 1/2 period
                count <= 0;            // reset count
            end
        end
        else begin  // reset speaker and count if reset is pressed
            spkr <= 0;
            count <= 0;
        end
    end
endmodule
```

This code yielded the following simulation waveforms. The cursors clearly show a period of 1 second for the 1 Hz portion of the testbench.

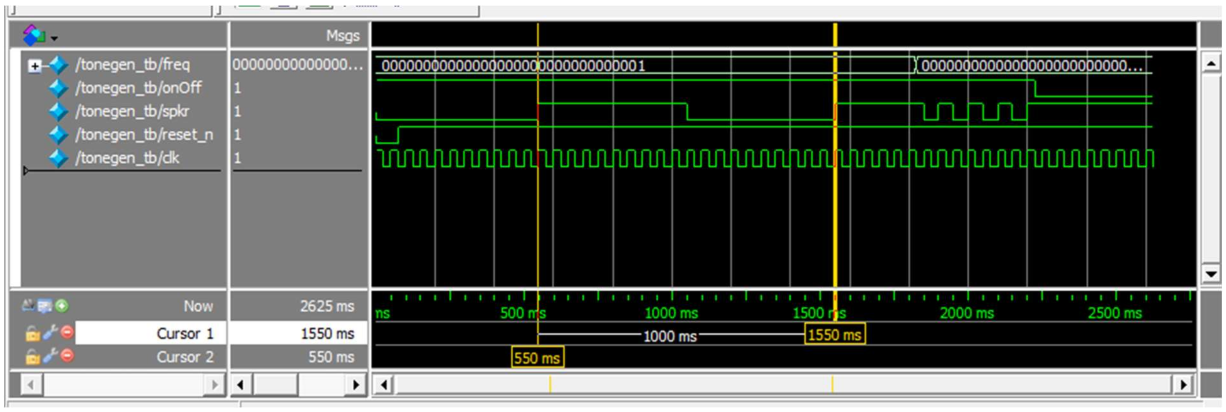


Figure 1 - 'tonegen.sv' Simulation Waveforms

This part of the lab was easy compared to the next part.

enc2freq.sv

I then wrote the following code for the enc2freq.sv module:

```
// ELEX 7660 Lab 3
// Nicholas Scott AKA "White Cheddar"
// January 28th, 2024
// Instructor: Sweet Bobby T

module enc2freq
   ( input logic cw, ccw,        // outputs from lab 2 encoder module
     output logic [31:0] freq,   // desired frequency
     input logic reset_n, clk); // reset and clock

    logic [7:0] countup = 0;
    logic [7:0] countdown = 0;

    always_ff @(posedge clk) begin
        if (reset_n) begin  // if active-low reset  is not pressed
            if (cw)
                countup <= countup + 1; // Count up for cw movement
            else if (ccw)
                countdown <= countdown + 1; // Count "down" for ccw movement

            if (countup >= 4) begin // after 4 cw pulses
                case (freq) // increment to next frequency of C Major scale
                    0 : freq <= 262;
                    262 : freq <= 295;
                    295 : freq <= 328;
                    328 : freq <= 349;
                    349 : freq <= 393;
                    393 : freq <= 437;
                    437 : freq <= 491;
                    491 : freq <= 524;
                    524 : freq <= 0; // cw rollover
                    default : freq <= 0;
                endcase
                countup <= 0;
            end
            else if (countdown >= 4) begin // after 4 ccw pulses
                case (freq) // decrement to last frequency of C Major scale
                    0 : freq <= 524;    // ccw rollover
                    524 : freq <= 491;
                    491 : freq <= 437;
                    437 : freq <= 393;
                    393 : freq <= 349;
                    349 : freq <= 328;
                    328 : freq <= 295;
                    295 : freq <= 262;
                    262 : freq <= 0;
                    default : freq <= 0;
                endcase
                countdown <= 0;
            end
        end
        else // set frequency to zero if reset is pressed
            freq <= 0;
    end
endmodule
```

That part wasn't too bad either. I like how the lab document just nonchalantly asks us to code a self-checking test bench, as if it's not by far the most difficult part of this lab.

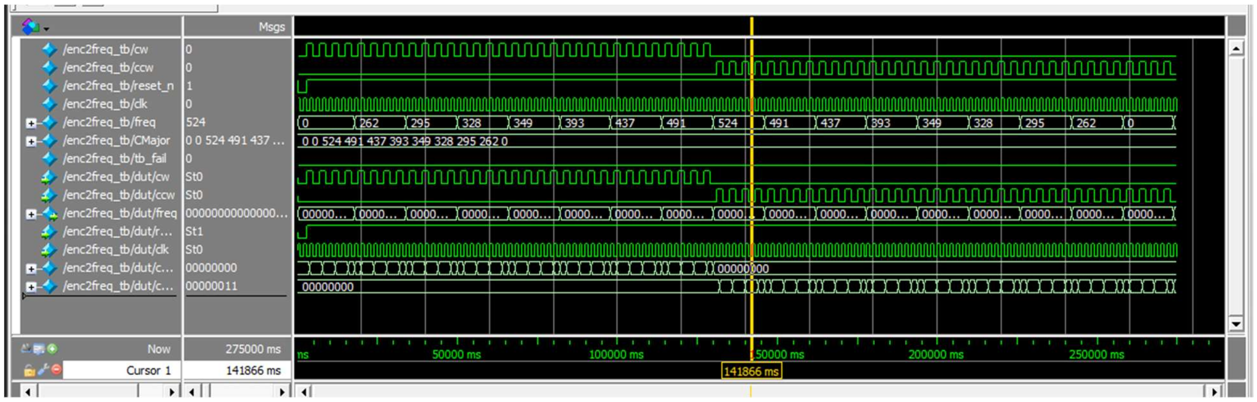
enc2freq_tb.sv

After enduring great pain, I implemented the following code for the test bench:

```
// ELEX 7660 Lab 3
// Nicholas Scott AKA "White Cheddar"
// January 28th, 2024
// Instructor: Sweet Bobby T

`timescale 1ms/1ms

function logic check_value (int expected_value, int actual_value);
```

```
    if (expected_value != actual_value) begin
        $display("FAIL: expected value is %d => actual value is %d",
expected_value, actual_value) ;
        check_value = 1;
    end else
        check_value = 0;

endfunction

module enc2freq_tb ;

    logic cw = 0;
    logic ccw = 0;
    logic reset_n = 1;
    logic clk = 1;
    logic [31:0] freq;

    logic [10:0][10:0] CMajor = '{0,0,524,491,437,393,349,328,295,262,0};
    logic tb_fail = 0;

    enc2freq dut (.*);

    initial begin

        // reset
        reset_n = 0;
        repeat(2) @(negedge clk) ;
        reset_n = 1;

        // ensure frequency starts at 0 after reset is pressed
        tb_fail = check_value (CMajor[0], freq);

        // simulate full cw rotation
        for (int i = 1 ; i<=8 ; i++) begin

            // 4 cw pulses
            for (int i = 0; i<4; i++)begin
                cw = 1;
                repeat(1) @(negedge clk);
                cw = 0;
                repeat(1) @(negedge clk);
            end

            tb_fail |= check_value (CMajor[i], freq);
        end

        // simulate full ccw rotation
        for (int i = 8 ; i>=0 ; i--) begin

            tb_fail |= check_value (CMajor[i], freq);

            // 4 ccw pulses
            for (int i = 0; i<4; i++)begin
                ccw = 1;
                repeat(1) @(negedge clk);
                ccw = 0;
                repeat(1) @(negedge clk);
            end
        end

        if (tb_fail)
            $display("DUMBASS!") ;
        else
            $display("YEEEEEEAH BOIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII") ;

        $stop;
    end
    // 1 Hz clock
    always
        #1000ms clk = ~clk ;
endmodule
```

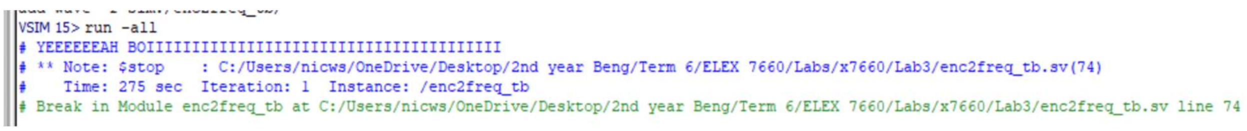
This code yielded the following waveforms:



```
                              Msgs
/enc2freq_tb/cw               0
/enc2freq_tb/ccw              0
/enc2freq_tb/reset_n          1
/enc2freq_tb/clk              0
/enc2freq_tb/freq             524
/enc2freq_tb/CMajor           0 0 524 491 437 ...
/enc2freq_tb/tb_fail          0
/enc2freq_tb/dut/cw           St0
/enc2freq_tb/dut/ccw          St0
/enc2freq_tb/dut/freq         00000000000000...
/enc2freq_tb/dut/r...         St1
/enc2freq_tb/dut/clk          St0
/enc2freq_tb/dut/c...         00000000
/enc2freq_tb/dut/c...         00000011

Now                           275000 ms
Cursor 1                      141866 ms
```

Figure 2 - 'enc2freq_tb' Simulation Waveforms

The *freq* variable can be seen incrementing up and back again, and the *tb_fail* variable is logic low. We got ourselves a tight situation here. The simulation transcript agrees:



```
VSIM 15> run -all
# YEEEEEEAH BOIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
# ** Note: $stop    : C:/Users/nicws/OneDrive/Desktop/2nd year Beng/Term 6/ELEX 7660/Labs/x7660/Lab3/enc2freq_tb.sv(74)
#    Time: 275 sec  Iteration: 1  Instance: /enc2freq_tb
# Break in Module enc2freq_tb at C:/Users/nicws/OneDrive/Desktop/2nd year Beng/Term 6/ELEX 7660/Labs/x7660/Lab3/enc2freq_tb.sv line 74
```

Figure 3 - 'enc2freq_tb' Simulation Transcript

That was one heck of a prelab.

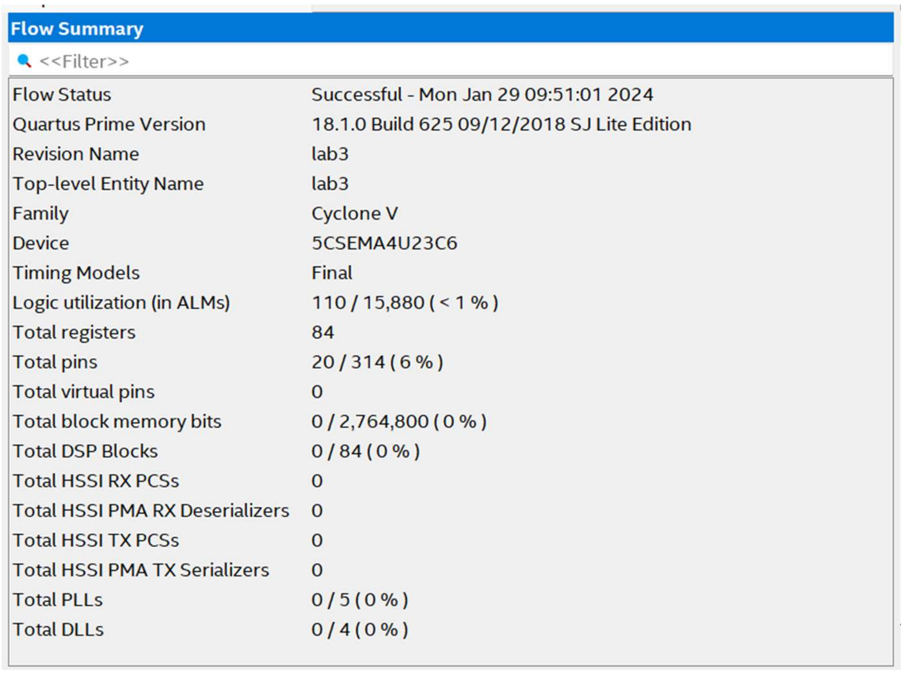
Lab Activities

I then wrote the following code for the top-level module lab3.sv:

```
// ELEX 7660 Lab 3
// Nicholas Scott AKA "White Cheddar"
// January 27th, 2024
// Instructor: Sweet Bobby T

module lab3 ( 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
              input logic s1, s2,         // reset and onOff pushbuttons
              output logic [7:0] leds,    // 7-seg LED enables
              output logic [3:0] ct,      // digit cathodes
              output logic spkr ) ;       // tone output

   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] freq;   // tone frequency

   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

   // instantiate modules to implement design
   decode2 decode2_0 (.digit,.ct) ;
   decode7 decode7_0 (.num(disp_digit),.leds) ;
   encoder encoder_1 (.clk(CLOCK_50), .a(enc1_a), .b(enc1_b), .cw(enc1_cw),
.ccw(enc1_ccw));
   enc2freq enc2freq_1 (.clk(CLOCK_50), .cw(enc1_cw), .ccw(enc1_ccw),
.freq(freq), .reset_n(s1));
   tonegen #(50000000) tonegen_1 (.clk(CLOCK_50), .freq(freq), .spkr(spkr),
.onOff(s2), .reset_n(s1));
```

```
   // 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)
  // each digit displays one nibble of tone frequency in hexidecimal
  always_comb begin

    case (digit)
      0 : disp_digit = freq[3:0]; // Set digit 0 to encoder 2 count LSN (Least 
Significant Nibble)
      1 : disp_digit = freq[7:4]; // Set digit 1 to encoder 2 count MSN
      2 : disp_digit = freq[11:8]; // Set digit 2 to encoder 1 count LSN
      3 : disp_digit = freq[15:12]; // Set digit 3 to encoder 1 count MSN
    endcase

  end

endmodule
```

I then plopped my code into Quartus and compiled it up. Behold the compilation report and netlists:


Flow Summary

<<Filter>>

Flow Status	Successful - Mon Jan 29 09:51:01 2024
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)	110 / 15,880 (< 1 %)
Total registers	84
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 %)

Figure 4 - Lab 3 Compilation Report

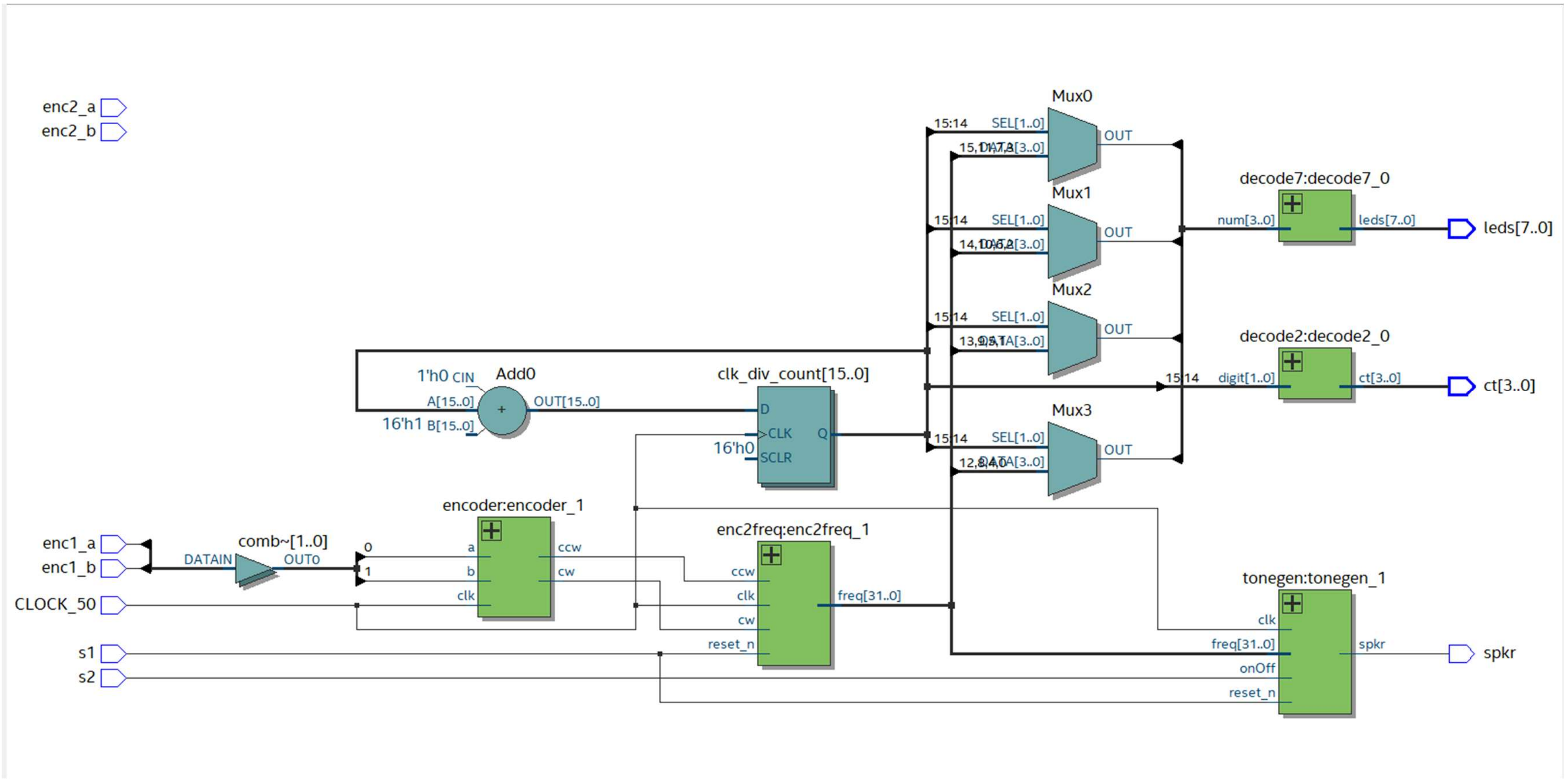


Figure 5 - 'lab3.sv' RTL Netlist

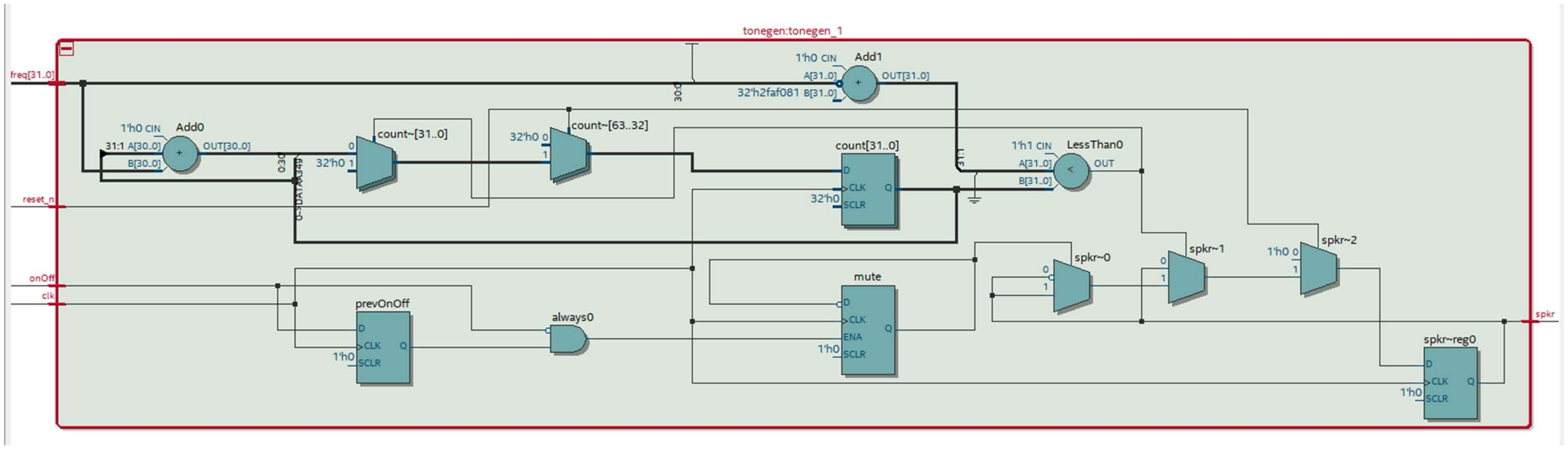


Figure 6 - 'tonegen.sv' RTL Netlist

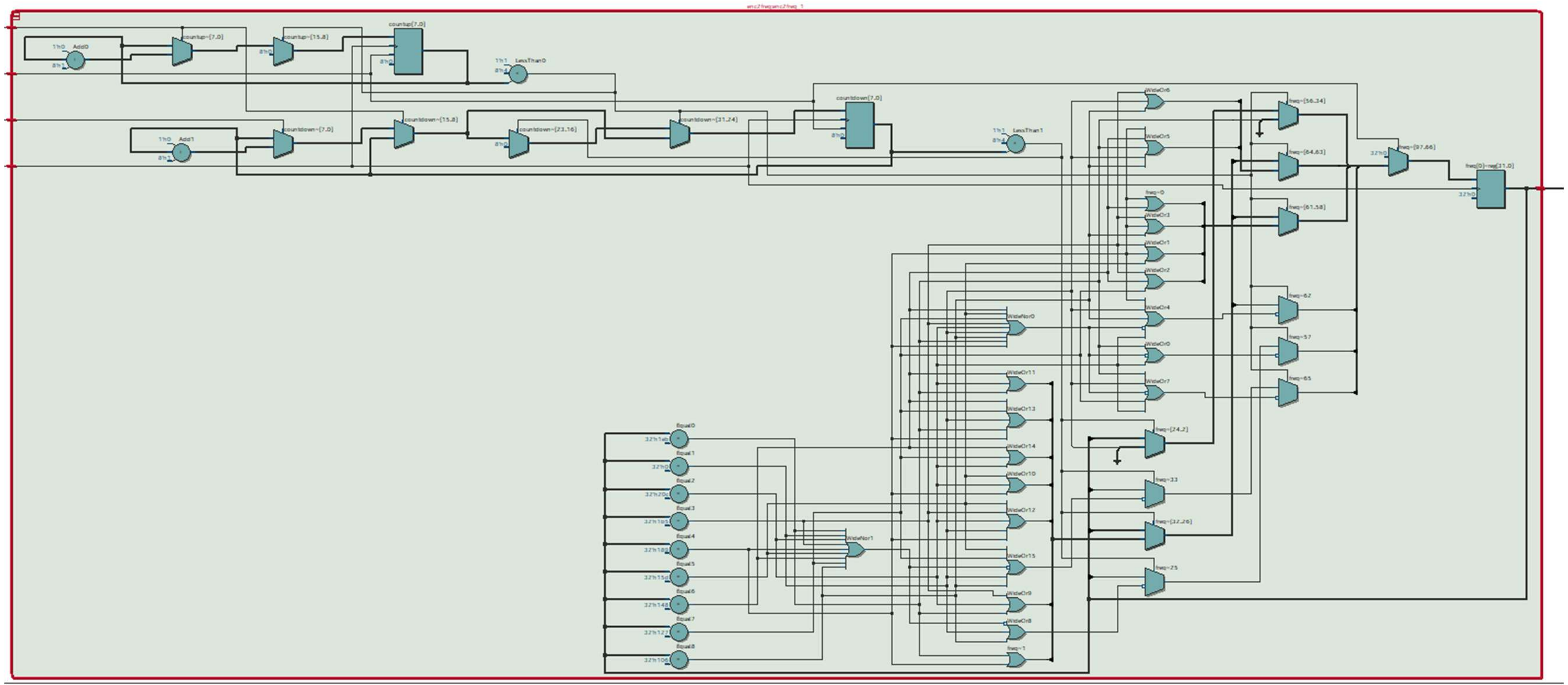


Figure 7 - 'enc2freq' RTL Netlist