Nicholas Scott AKA “White Cheddar”

A01255181 | Jan 28th, 2024

ELEX 7660

Lab 3 – Tone Generator

Contents

[Prelab 2](#_Toc157416475)

[tonegen.sv 2](#_Toc157416476)

[enc2freq.sv 3](#_Toc157416477)

[enc2freq\_tb.sv 4](#_Toc157416478)

[Lab Activities 7](#_Toc157416479)

**Table of Figures**

[Figure 1 - 'tonegen.sv' Simulation Waveforms 3](#_Toc157416483)

[Figure 2 - 'enc2freq\_tb' Simulation Waveforms 6](#_Toc157416484)

[Figure 3 - 'enc2freq\_tb' Simulation Transcript 7](#_Toc157416485)

[Figure 4 - Lab 3 Compilation Report 9](#_Toc157416486)

[Figure 5 - 'lab3.sv' RTL Netlist 0](#_Toc157416487)

[Figure 6 - 'tonegen.sv' RTL Netlist 1](#_Toc157416488)

[Figure 7 - 'enc2freq' RTL Netlist 2](#_Toc157416489)

# 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.

A screen shot of a computer

Description automatically generated

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 BOIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII") ;

        $stop;

    end

    // 1 Hz clock

    always

        #1000ms clk = ~clk ;

endmodule

This code yielded the following waveforms:

A screen shot of a computer

Description automatically generated

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:

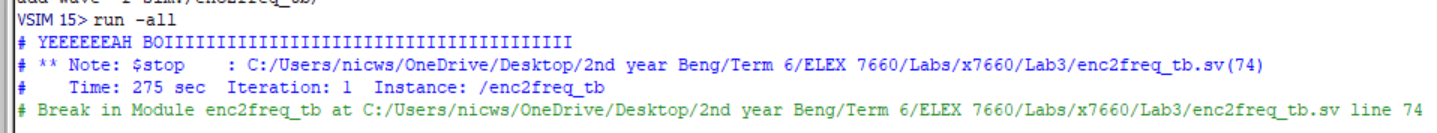


Figure 3 - 'enc2freq\_tb' Simulation Transcript

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:

A screenshot of a computer

Description automatically generated

Figure 4 - Lab 3 Compilation Report

A diagram of a computer circuit

Description automatically generated

Figure 5 - 'lab3.sv' RTL Netlist

A diagram of a machine

Description automatically generated

Figure 6 - 'tonegen.sv' RTL Netlist

A diagram of a circuit board

Description automatically generated

Figure 7 - 'enc2freq' RTL Netlist