Nicholas Scott AKA “White Cheddar”

A01255181 | Jan 30th, 2024

ELEX 7660

ass1

Contents

[Problem 1 2](#_Toc157613972)

[‘encoder.sv’ 2](#_Toc157613973)

[‘encoder\_tb.sv’ 3](#_Toc157613974)

[Problem 2 6](#_Toc157613975)

[‘shiftReg.sv’ 6](#_Toc157613976)

[‘shiftReg\_tb.sv’ 9](#_Toc157613977)

**Table of Figures**

[Figure 1 - 'encoder.sv' Compilation Report 3](#_Toc157613984)

[Figure 2 - 'encoder.sv' RTL Netlist 3](#_Toc157613985)

[Figure 3 - 'encoder\_tb' Simulation Waveforms 5](#_Toc157613986)

[Figure 4 - 'encoder.sv' Simulation Transcript 6](#_Toc157613987)

[Figure 5 - 'shiftReg.sv' Compilation Report 7](#_Toc157613988)

[Figure 6 - 'shiftReg.sv' RTL Netlist 8](#_Toc157613989)

[Figure 7 - 'shiftReg\_tb.sv' Simulation Waveforms 11](#_Toc157613990)

[Figure 8 - 'shiftReg\_tb.sv' Simulation Transcript 11](#_Toc157613991)

# Problem 1

I’m so excited.

## ‘encoder.sv’

I wrote the following code for ‘encoder.sv’:

// ELEX 7660 Ass1

// Nicholas Scott AKA "White Cheddar"

// Jan. 30th, 2024

// Instructor: Sweet Bobby T

module encoder (output logic [1:0] y,

                output logic valid,

                input logic [3:0] a);

    always\_comb begin

        // prepare for a case statement

        case (a)

            0 : begin

                y = 0;

                valid = 0;

            end

            1 : begin

                y = 0;

                valid = 1;

            end

            2, 3 : begin

                y = 1;

                valid = 1;

            end

            4, 5, 6, 7 : begin

                y = 2;

                valid = 1;

            end

            8, 9, 10, 11, 12, 13, 14, 15 : begin

                y = 3;

                valid = 1;

            end

        endcase

    end

endmodule

The code is very straightforward… I couldn’t think of any meaningful comments to add. This code yielded the following compilation report and RTL Netlist:

A screenshot of a computer

Description automatically generated

Figure - 'encoder.sv' Compilation Report

A diagram of a computer program

Description automatically generated

Figure - 'encoder.sv' RTL Netlist

Now for the fun part…

## ‘encoder\_tb.sv’

I implemented the following code for the encoder test bench:

// ELEX 7660 Ass1

// Nicholas Scott AKA "White Cheddar"

// Jan. 30th, 2024

// Instructor: Sweet Bobby T

`timescale 1ms/1ms

function logic check\_value (int expected\_value, int actual\_value); // your handy lil function

    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 encoder\_tb ;

    // expected values, in sequence from a = 0 to a = 15

    int yExpectedValues [0:15] = '{0,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3};

    int validExpectedValues [0:15] = '{0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};

    // encoder outputs

    logic [1:0] y;

    logic valid;

    logic [3:0] a = 0; // encoder input

    logic tb\_fail = 0;

    logic clk = 1;

    encoder dut (.\*);

    initial begin

        for (int i = 0; i <=15; i++) begin

            a = i; // increment a from 0 to 15

            repeat(1) @(negedge clk); // allow some time for output to stabilize

            // check both outputs

            tb\_fail |= check\_value (yExpectedValues[i], y);

            tb\_fail |= check\_value (validExpectedValues[i], valid);

            repeat(1) @(negedge clk); // allow some time again

        end

        if (tb\_fail)

            $display("DUMBASS!") ;

        else

            $display("YEEEEEEAH BOIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII") ;

        $stop;

    end

    // 1 Hz clock

    always

        #1000ms clk = ~clk ;

endmodule

When simulated, this test bench yielded the following waveforms and simulation transcript. Not the *y* output changing as expected, *tb\_fail* remaining logic low, and the yeeeeeeah boiiiiiiiiiiiiiiiii.

A screen shot of a computer

Description automatically generated

Figure - 'encoder\_tb' Simulation Waveforms

A screenshot of a computer

Description automatically generated

Figure - 'encoder.sv' Simulation Transcript

# Problem 2

I then did the next thing.

## ‘shiftReg.sv’

I wrote the following code for the shift register:

// ELEX 7660 Ass1

// Nicholas Scott AKA "White Cheddar"

// Jan. 30th, 2024

// Instructor: Sweet Bobby T

module shiftReg (output logic [7:0] q,

                input logic [7:0] a, input logic [1:0] s,

                input logic shiftIn, clk, reset\_n);

    always\_ff @( posedge clk or negedge reset\_n ) begin

        if(!reset\_n)

            q <= a; // what I assume you want the reset to do

        else begin // if active low reset not pressed

            case(s)

                0 : q <= a; // follow a

                1 : begin

                    q >>= 1; // make room for shiftIn variable

                    q[7] <= shiftIn;    // get in there

                end

                2 : begin

                    q <<= 1; // make some room on the right side

                    q[0] <= shiftIn; // shift that thang in there

                end

                3 : q <= q; // maintain current state

            endcase

        end

    end

endmodule

I threw that shit in Quartus and compiled its ass. I got the following compilation report and RTL netlist as evidence:

A screenshot of a computer

Description automatically generated

Figure - 'shiftReg.sv' Compilation Report

A diagram of a computer

Description automatically generated

Figure - 'shiftReg.sv' RTL Netlist

## ‘shiftReg\_tb.sv’

I also made a cute little test bench.

// ELEX 7660 Ass1

// Nicholas Scott AKA "White Cheddar"

// Jan. 30th, 2024

// Instructor: Sweet Bobby T

`timescale 1ms/1ms

function logic check\_value (int expected\_value, int actual\_value); // your handy lil function

    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 shiftReg\_tb;

    logic [7:0] q;

    logic [7:0] a = 'b00001000; // almost palindrominc

    logic [1:0] s;

    logic reset\_n = 1;

    logic clk = 1;

    logic shiftIn;

    logic tb\_fail = 0;

    // sequence of expected outputs

    logic [7:0] qExpectedValues[0:6] = '{'b00001000, 'b00000100,'b10000010,'b00000100,'b00001001,'b00001001,'b00001000};

    shiftReg dut (.\*);

    initial begin

        // set q to a

        s = 0;

        repeat(1) @(negedge clk);

        tb\_fail |= check\_value (qExpectedValues[0], q);

        // right shift in a 0

        s = 1;

        shiftIn = 0;

        repeat(1) @(negedge clk);

        tb\_fail |= check\_value (qExpectedValues[1], q);

        // right shift in a 1

        s = 1;

        shiftIn = 1;

        repeat(1) @(negedge clk);

        tb\_fail |= check\_value (qExpectedValues[2], q);

        // left shift in a 0

        s = 2;

        shiftIn = 0;

        repeat(1) @(negedge clk);

        tb\_fail |= check\_value (qExpectedValues[3], q);

        // left shift in a 1

        s = 2;

        shiftIn = 1;

        repeat(1) @(negedge clk);

        tb\_fail |= check\_value (qExpectedValues[4], q);

        // maintain q

        s = 3;

        repeat(1) @(negedge clk);

        tb\_fail |= check\_value (qExpectedValues[5], q);

        // reset

        reset\_n = 0;

        repeat(1) @(negedge clk);

        tb\_fail |= check\_value (qExpectedValues[6], q);

        if (tb\_fail)

            $display("DUMBASS!") ;

        else

            $display("YEEEEEEAH BOIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII") ;

        $stop;

    end

    // 1 Hz clock

    always

        #1000ms clk = ~clk ;

endmodule

It ended up being a bit long but it did the trick. I slapped it into ModelSim and got the following results:

A screen shot of a computer

Description automatically generated

Figure - 'shiftReg\_tb.sv' Simulation Waveforms

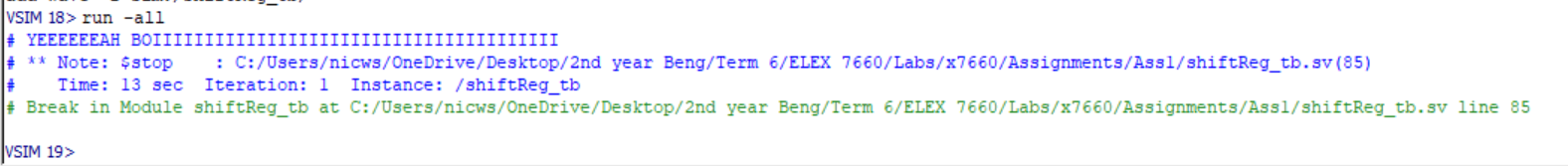


Figure - 'shiftReg\_tb.sv' Simulation Transcript