A cat with two heads on a coat of arms

Description automatically generated

bcit

Lab 4 – ADC Interface

Nicholas Scott AKA “White Cheddar”

A01255181 | Feb. 10th, 2024

Contents

[‘adcinterface.sv’ 2](#_Toc158475371)

[‘enc2chan’ 5](#_Toc158475372)

[‘lab4.sv’ 6](#_Toc158475373)

[RTL Netlists 8](#_Toc158475374)

**Table of Figures**

[Figure 1 - 'adcinterface.sv' Simulation Waveforms 4](#_Toc158475377)

[Figure 2 - 'adcinterface.sv' Simulation Transcript 5](#_Toc158475378)

[Figure 3 - 'lab4.sv' Compilation Report 8](#_Toc158475379)

[Figure 4 - 'adcinterface.sv' RTL Netlist 9](#_Toc158475380)

[Figure 5 - 'lab4.sv' RTL Netlist 0](#_Toc158475381)

[Figure 6 - 'enc2chan.sv' RTL Netlist 1](#_Toc158475382)

“Ninety-five percent of people who pass my test bench get it working the first time.”

* *Sweet Bobby T*

# ‘adcinterface.sv’

After much hardship, I settled on the following code for my ADC interface module:

// ELEX 7660 Lab 4

// Nicholas Scott AKA "White Cheddar"

// A01255181

// Feb. 4th, 2024

// Instructor: Sweet Bobby T

module adcinterface(

    input logic clk, reset\_n,   // clock and reset

    input logic [2:0] chan,     // ADC channel to sample

    output logic [11:0] result, // ADC result

    // ltc2308 signals

    output logic ADC\_CONVST, ADC\_SCK, ADC\_SDI,

    input logic ADC\_SDO

);

    logic [5:0] word; // word which tells ADC which channel we want

    logic [3:0] count; // a count bit that counts

    logic [11:0] tempResult; // a temp variable to store result

    always\_comb begin

        // activate ADC clock for 12 clock cycles

        ADC\_SCK = ((count >= 2) && (count <= 13)) ? clk : 1'b0;

        // Set config word based on chan input

        case (chan)

            0 : word = 6'b100010;

            1 : word = 6'b110010;

            2 : word = 6'b100110;

            3 : word = 6'b110110;

            4 : word = 6'b101010;

            5 : word = 6'b111010;

            6 : word = 6'b101110;

            7 : word = 6'b111110;

        endcase

    end

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

        if (~reset\_n)

            count <= 0;

        else

            count <= count + 1;  // automatically rolls over at 16

    end

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

        if (~reset\_n) begin

            ADC\_CONVST <= 1;

            ADC\_SDI <= 0;

        end

        else begin

            case (count)

                0 : ADC\_CONVST <= 0; // conversion start signal

                // send config word one bit at a time

                1 : ADC\_SDI <= word[5];

                2 : ADC\_SDI <= word[4];

                3 : ADC\_SDI <= word[3];

                4 : ADC\_SDI <= word[2];

                5 : ADC\_SDI <= word[1];

                6 : ADC\_SDI <= word[0];

                15 : ADC\_CONVST <= 1; // conversion end

                default : ADC\_SDI <= 0;

            endcase

        end

    end

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

        if(~reset\_n)

            result <= 0;

        else

            case (count)

                // receive ADC result one bit at a time

                2 : tempResult[11] <= ADC\_SDO;

                3 : tempResult[10] <= ADC\_SDO;

                4 : tempResult[9] <= ADC\_SDO;

                5 : tempResult[8] <= ADC\_SDO;

                6 : tempResult[7] <= ADC\_SDO;

                7 : tempResult[6] <= ADC\_SDO;

                8 : tempResult[5] <= ADC\_SDO;

                9 : tempResult[4] <= ADC\_SDO;

                10 : tempResult[3] <= ADC\_SDO;

                11 : tempResult[2] <= ADC\_SDO;

                12 : tempResult[1] <= ADC\_SDO;

                13 : tempResult[0] <= ADC\_SDO;

                14 : result <= tempResult; // assign result

                default: result <= result;

            endcase

endmodule

This code yielded the following simulation waveforms and transcript:

A screenshot of a computer

Description automatically generated

Figure 1 - 'adcinterface.sv' Simulation Waveforms

A close-up of a computer screen

Description automatically generated

Figure 2 - 'adcinterface.sv' Simulation Transcript

# ‘enc2chan’

This code is basically the same as my ‘enc2freq.sv’ code, with the frequencies replaced with channels instead.

// ELEX 7660 Lab 4

// Nicholas Scott AKA "White Cheddar"

// A01255181

// Feb. 6th, 2024

// Instructor: Sweet Bobby T

module enc2chan

   ( input logic cw, ccw,       // outputs from lab 2 encoder module

     output logic [2:0] chan,  // desired channel

     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 (chan) // increment to next channel

                    0 : chan <= 1;

                    1 : chan <= 2;

                    2 : chan <= 3;

                    3 : chan <= 4;

                    4 : chan <= 5;

                    5 : chan <= 6;

                    6 : chan <= 7;

                    7 : chan <= 0; // cw rollover

                    default : chan <= 0;

                endcase

                countup <= 0;

            end

            else if (countdown >= 4) begin // after 4 ccw pulses

                case (chan) // decrement to last channel

                    0 : chan <= 7;    // ccw rollover

                    7 : chan <= 6;

                    6 : chan <= 5;

                    5 : chan <= 4;

                    4 : chan <= 3;

                    3 : chan <= 2;

                    2 : chan <= 1;

                    1 : chan <= 0;

                    default : chan <= 0;

                endcase

                countdown <= 0;

            end

        end

        else // set chan to zero if reset is pressed

            chan <= 0;

    end

endmodule

You didn’t ask us to write a test bench for this. Besides, I know it will work because ‘enc2freq.sv’ worked.

# ‘lab4.sv’

This module looks a lot like ‘lab3.sv’:

// ELEX 7660 Lab 4

// Nicholas Scott AKA "White Cheddar"

// A01255181

// Feb. 6th, 2024

// Instructor: Sweet Bobby T

module lab4 ( input logic ADC\_SDO,        // ADC input

              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 ADC\_CONVST, ADC\_SCK, ADC\_SDI // ADC outputs

              );

   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 [2:0] chan;  // channel

   logic [11:0] result; // ADC result

   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));

   enc2chan enc2chan\_1 (.clk(CLOCK\_50), .cw(enc1\_cw), .ccw(enc1\_ccw), .chan(chan), .reset\_n(s1));

   adcinterface adcinterface\_1 (.clk(clk\_div\_count[15]), .reset\_n(s1), .chan(chan), .result, .ADC\_CONVST, .ADC\_SCK, .ADC\_SDI, .ADC\_SDO);

   // 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)

  // the right three digits display the ADC output in hexadecimal

  // the leftmost digit displays the channel

  always\_comb begin

    case (digit)

      0 : disp\_digit = result[3:0]; // result LSN (Least Significant Nibble)

      1 : disp\_digit = result[7:4]; // the middle nibble

      2 : disp\_digit = result[11:8]; // result MSN

      3 : disp\_digit = chan; // Set digit 3 channel

    endcase

  end

endmodule

This code yielded the following compilation report:

A screenshot of a computer

Description automatically generated

Figure 3 - 'lab4.sv' Compilation Report

# My Issue

The reason I couldn’t demo on the first week is because I was defining my ‘word’ variable incorrectly. For some reason, ModelSim did not complain about it, but Quartus didn’t like it. I found out that:

A screenshot of a computer

Description automatically generated

The code on the left worked in ModelSim, but not Quartus. I suspect that ModelSim was smart enough to realize that I was concatenating single-bit logic, but Quartus assumed a width of 32 bits for each of the values, and thus the leading 0’s caused ‘word[5:0]’ to be all 0’s, and hence I was permanently stuck on channel 0 and 1 differential mode. Very silly of me.

# RTL Netlists

This whole design yielded the following RTL Netlists:

A computer screen shot of a diagram

Description automatically generated

Figure 4 - 'adcinterface.sv' RTL Netlist

A diagram of a computer

Description automatically generated

Figure 5 - 'lab4.sv' RTL Netlist

A computer screen shot of a circuit board

Description automatically generated

Figure 6 - 'enc2chan.sv' RTL Netlist