CDA 4203L Sec 001

Computer System Design Lab

Lab 2 Report

Verilog Based ALU Design

|  |  |
| --- | --- |
| Today’s Date: | 2/9/18 |
| Your Name: | Boyang Wu |
| Your U Number: | U95035892 |
| No. of Hours Spent: | 10 |
| Exercise Difficulty:  (Easy, Average, Hard) | Average |
| Any Other Feedback: | A bit trickier in the structural part since I had a bug in the 4-to-1 mux with incorrect S0/S0\_in values. This took about 2 hours to figure out. Once that was fixed everything worked |

Problem 1: ALU Behavioral Verilog

module Lab2\_beh(

input wire [3:0] A\_in,

input wire [3:0] B\_in,

input wire S0\_in,

input wire S1\_in,

output reg [3:0] Lab2\_beh\_out);

always @(\*) begin

if(!S1\_in & !S0\_in)

Lab2\_beh\_out = ~A\_in;

if(!S1\_in & S0\_in)

Lab2\_beh\_out = A\_in + B\_in;

if(S1\_in & !S0\_in)

Lab2\_beh\_out = A\_in - B\_in;

if(S1\_in & S0\_in)

Lab2\_beh\_out = A\_in + A\_in;

end

endmodule

//Easy code. Just use if statements for every condition listed in the table.

Problem 1: Verilog Test Bench

// Initialize Inputs

A\_in = 0;

B\_in = 0;

S0\_in = 0;

S1\_in = 0;

//Invert A = 2

#100

S1\_in = 0;

S0\_in = 0;

A\_in = 2;

B\_in = 0;

//Invert A = 15

#100

S1\_in = 0;

S0\_in = 0;

A\_in = 15;

B\_in = 0;

//Add 3 + 5

#100

S1\_in = 0;

S0\_in = 1;

A\_in = 3;

B\_in = 5;

//Add 2 + 10

#100

S1\_in = 0;

S0\_in = 1;

A\_in = 2;

B\_in = 10;

//Subtract 13 - 2

#100

S1\_in = 1;

S0\_in = 0;

A\_in = 13;

B\_in = 2;

//Subtract 6 - 4

#100

S1\_in = 1;

S0\_in = 0;

A\_in = 6;

B\_in = 4;

//Double A = 3

#100

S1\_in = 1;

S0\_in = 1;

A\_in = 5;

B\_in = 0;

//Double A = 7

#100

S1\_in = 1;

S0\_in = 1;

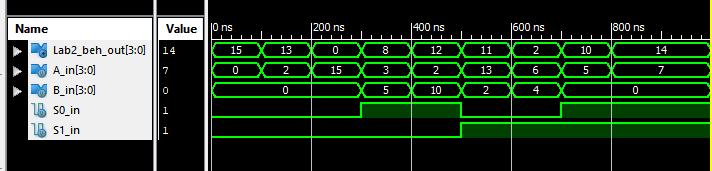
A\_in = 7;

B\_in = 0;

//Same as from Lab 1 other than change from //Subtract 14-2 to //Subtract 13-2 to make the waveform a bit easier to read.

Problem 1: Simulation Waveforms (add as many pages as you need).

Include *atleast* two test vectors per function. For example, demonstrate through the waveforms, that the ALU performs inversion correctly on two inputs, say, 0010 and 1111.



//Goes through everything in the test bench. 3 total inverts (including start), followed by 2 adds, 2 subtracts, then 2 doubles.

Problem 2: Behavioral Components (insert 1 page per component)

module mux\_21\_4bit(mux\_in0, mux\_in1, sel, mux\_out);

//Declare inputs/outputs

input sel;

input [3:0] mux\_in0;

input [3:0] mux\_in1;

output [3:0] mux\_out;

reg [3:0] mux\_out;

always @ (\*) begin

if (sel == 0) begin

mux\_out = mux\_in0;

end

else begin

mux\_out = mux\_in1;

end

end

endmodule

//Simple 4-bit 2-to-1 mux derived from the 1-bit mux in the tutorial

module mux\_41\_4bit(A\_in, B\_in, C\_in, D\_in, S0\_in, S1\_in, Mux\_out);

input [3:0] A\_in;

input [3:0] B\_in;

input [3:0] C\_in;

input [3:0] D\_in;

input S0\_in;

input S1\_in;

output [3:0] Mux\_out;

wire [3:0] Mux1\_out;

wire [3:0] Mux2\_out;

mux\_21\_4bit Mux1 (.mux\_in0(A\_in[3:0]),

.mux\_in1(B\_in[3:0]),

.sel(S0\_in),

.mux\_out(Mux1\_out[3:0]));

mux\_21\_4bit Mux2 (.mux\_in0(C\_in[3:0]),

.mux\_in1(D\_in[3:0]),

.sel(S0\_in),

.mux\_out(Mux2\_out[3:0]));

mux\_21\_4bit Mux3 (.mux\_in0(Mux1\_out[3:0]),

.mux\_in1(Mux2\_out[3:0]),

.sel(S1\_in),

.mux\_out(Mux\_out[3:0]));

endmodule

//I realize the lab said only 2-to-1 mux, but it is not exactly feasible to keep using 2-to-1 muxes in the final structural Verilog code, so I made a structural 4-to-1 mux first (it is still structural at least) to make things a bit cleaner.

module inverter4bit(I\_in, I\_out);

input [3:0] I\_in;

output [3:0] I\_out;

inverter Inv1 (.inp(I\_in[3]), .op(I\_out[3]));

inverter Inv2 (.inp(I\_in[2]), .op(I\_out[2]));

inverter Inv3 (.inp(I\_in[1]), .op(I\_out[1]));

inverter Inv4 (.inp(I\_in[0]), .op(I\_out[0]));

endmodule

module inverter (inp, op);

input inp;

output op;

assign op = ~inp;

endmodule

//Made the 4-bit inverter out of the 1-bit inverter from the tutorial. It does the job.

Problem 2: ALU Structural Verilog (Use as many pages as needed.)

module Lab2\_Struct(A\_in, B\_in, C\_in, S0\_in, S1\_in, C\_out0, C\_out1, Lab2\_Struct\_out);

input [3:0] A\_in;

input [3:0] B\_in;

input C\_in;

input S0\_in;

input S1\_in;

output C\_out0;

output C\_out1;

output [3:0] Lab2\_Struct\_out;

wire [3:0] Inv\_out;

wire [3:0] Add\_out;

wire [3:0] Dbl\_out;

wire [3:0] Sub\_out;

wire [3:0] Sub1\_out;

wire [3:0] Sub2\_out;

inverter4bit Inv(.I\_in(A\_in[3:0]),

.I\_out(Inv\_out[3:0]));

fulladd4bit Add(.A\_in(A\_in[3:0]),

.B\_in(B\_in[3:0]),

.C\_in(C\_in),

.C\_out(C\_out0),

.S\_out(Add\_out[3:0]));

inverter4bit Inv1(.I\_in(B\_in[3:0]),

.I\_out(Sub\_out[3:0]));

assign Sub1\_out[3:0] = Sub\_out[3:0] + 1;

fulladd4bit Sub(.A\_in(A\_in[3:0]),

.B\_in(Sub1\_out[3:0]),

.C\_in(C\_in),

.C\_out(),

.S\_out(Sub2\_out[3:0]));

fulladd4bit Dbl(.A\_in(A\_in[3:0]),

.B\_in(A\_in[3:0]),

.C\_in(C\_in),

.C\_out(C\_out1),

.S\_out(Dbl\_out[3:0]));

mux\_41\_4bit Lab\_out(.A\_in(Inv\_out[3:0]),

.B\_in(Add\_out[3:0]),

.C\_in(Sub2\_out[3:0]),

.D\_in(Dbl\_out[3:0]),

.S0\_in(S0\_in),

.S1\_in(S1\_in),

.Mux\_out(Lab2\_Struct\_out[3:0]));

endmodule

//This is a bit messy, so I will explain it a bit.

The first inverter4bit called “Inv” is for ~A in the truth table.

Next, we get the first fulladd4bit called “Add” which is A + B in the truth table.

Next, we have an inverter called “Inv1”, with 1 added to the output (Sub\_out), then the output put into an adder called “Sub”. This is the 2’s complement of B added to A, which in turn gives us a 4-bit full subtractor (A – B in truth table)

Next, we have another fulladd4bit “Dbl” to add A to A (2\*A in truth table).

Finally, the 4-to-1 mux made earlier (in structural Verilog) is used to switch between inputs.

Problem 2: Verilog Test Bench

// Initialize Inputs

A\_in = 0;

B\_in = 0;

S0\_in = 0;

S1\_in = 0;

//Invert A = 2

#100

S1\_in = 0;

S0\_in = 0;

A\_in = 2;

B\_in = 0;

//Invert A = 15

#100

S1\_in = 0;

S0\_in = 0;

A\_in = 15;

B\_in = 0;

//Add 3 + 5

#100

S1\_in = 0;

S0\_in = 1;

A\_in = 3;

B\_in = 5;

//Add 2 + 10

#100

S1\_in = 0;

S0\_in = 1;

A\_in = 2;

B\_in = 10;

//Subtract 13 - 2

#100

S1\_in = 1;

S0\_in = 0;

A\_in = 13;

B\_in = 2;

//Subtract 6 - 4

#100

S1\_in = 1;

S0\_in = 0;

A\_in = 6;

B\_in = 4;

//Double A = 3

#100

S1\_in = 1;

S0\_in = 1;

A\_in = 5;

B\_in = 0;

//Double A = 7

#100

S1\_in = 1;

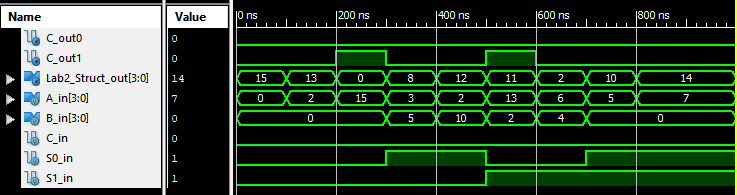
S0\_in = 1;

A\_in = 7;

B\_in = 0;

//Same as before. No changes. Problem 2: Simulation Results

Include *atleast* two test vectors per function. For example, demonstrate through the waveforms, that the ALU performs inversion correctly on two inputs, say, 0010 and 1111.



//Same as before, although waveform isn’t organized the same (not sure how to change that). And for reference, I’ll also include the waveform from Lab 1 which was perfect to reaffirm that this lab is correct.

