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.

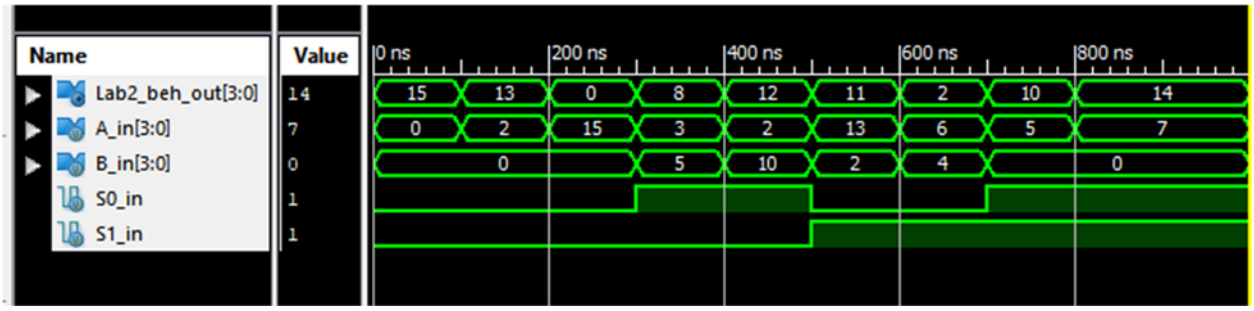
Problem 1: Verilog Test Bench

```
// 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.

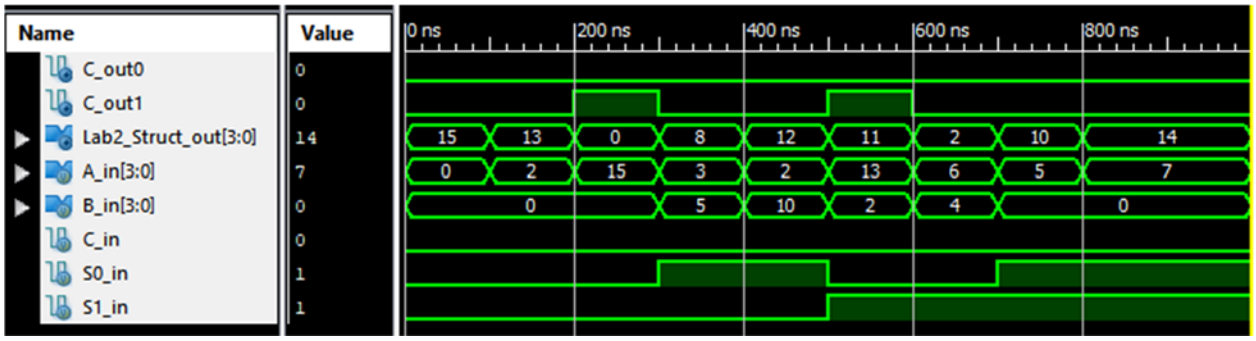
Problem 2: Verilog Test Bench

```
// 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.