CDA 4203L 001

Lab #1

January 19, 2013, Mark Little

## Purpose and Objectives:

This lab is intended to give students the opportunity to practice circuit design and analysis using Verilog software. Students are required to create four ALUs to perform four functions; inverting, adding, subtracting, and doubling. To create these ALUs students must use Behavioral Verilog, Structured Verilog, Simple Structural Verilog, and IP Core.

## Design:

All four parts of this lab share five common elements but with slight variations, especially in the case of the first part. They all use an inverter, adder, subtractor, doubler, and a multiplexer.

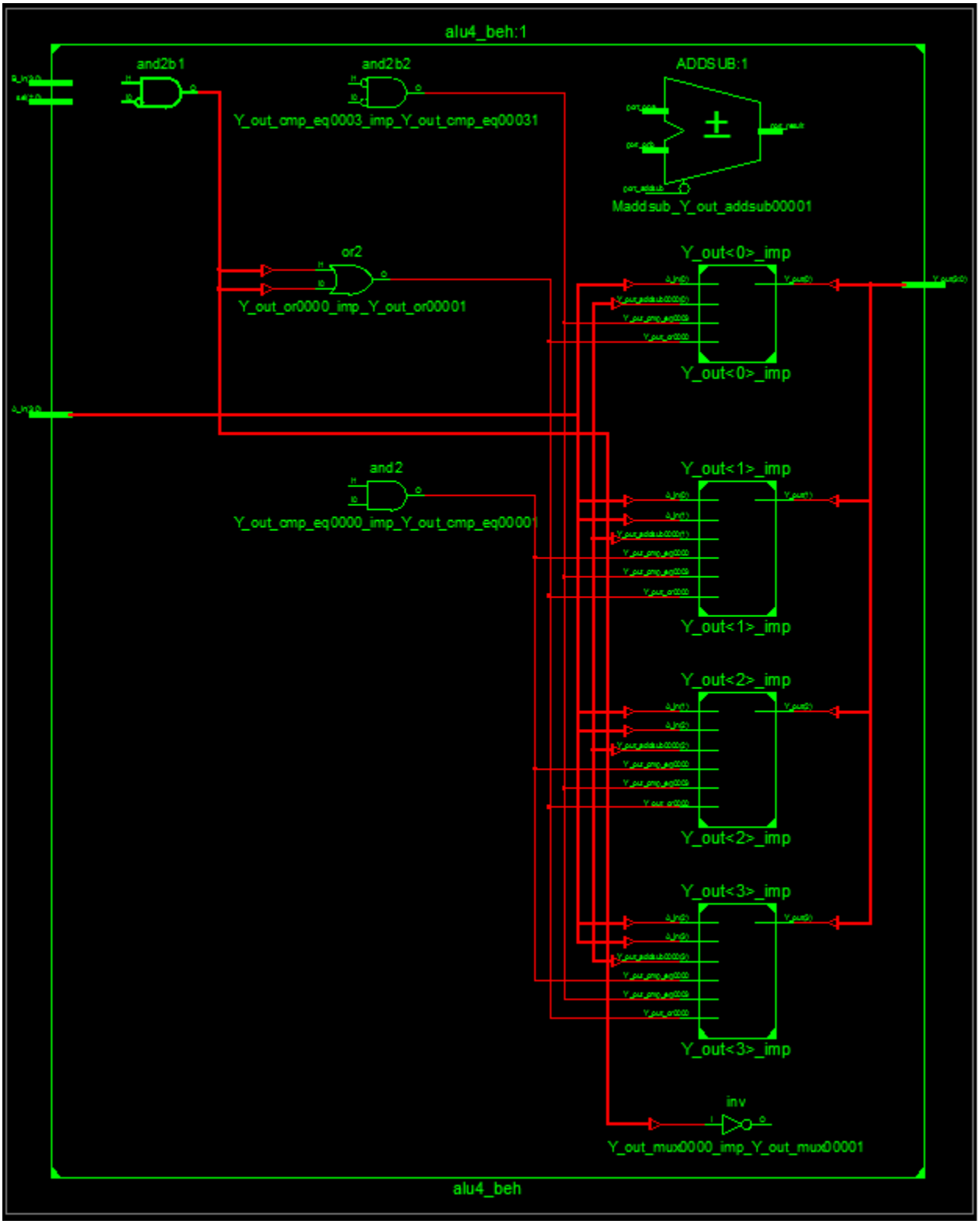
The first part was implemented by using Behavioral Verilog, the functions are carried out using arithmetic and bitwise operators for the inverting, adding, subtracting, and doubling functions, the ALU's functions are then selected using a multiplexer. The 4-bit ALU is capable of handling I/O ranging from 0 to 1111.

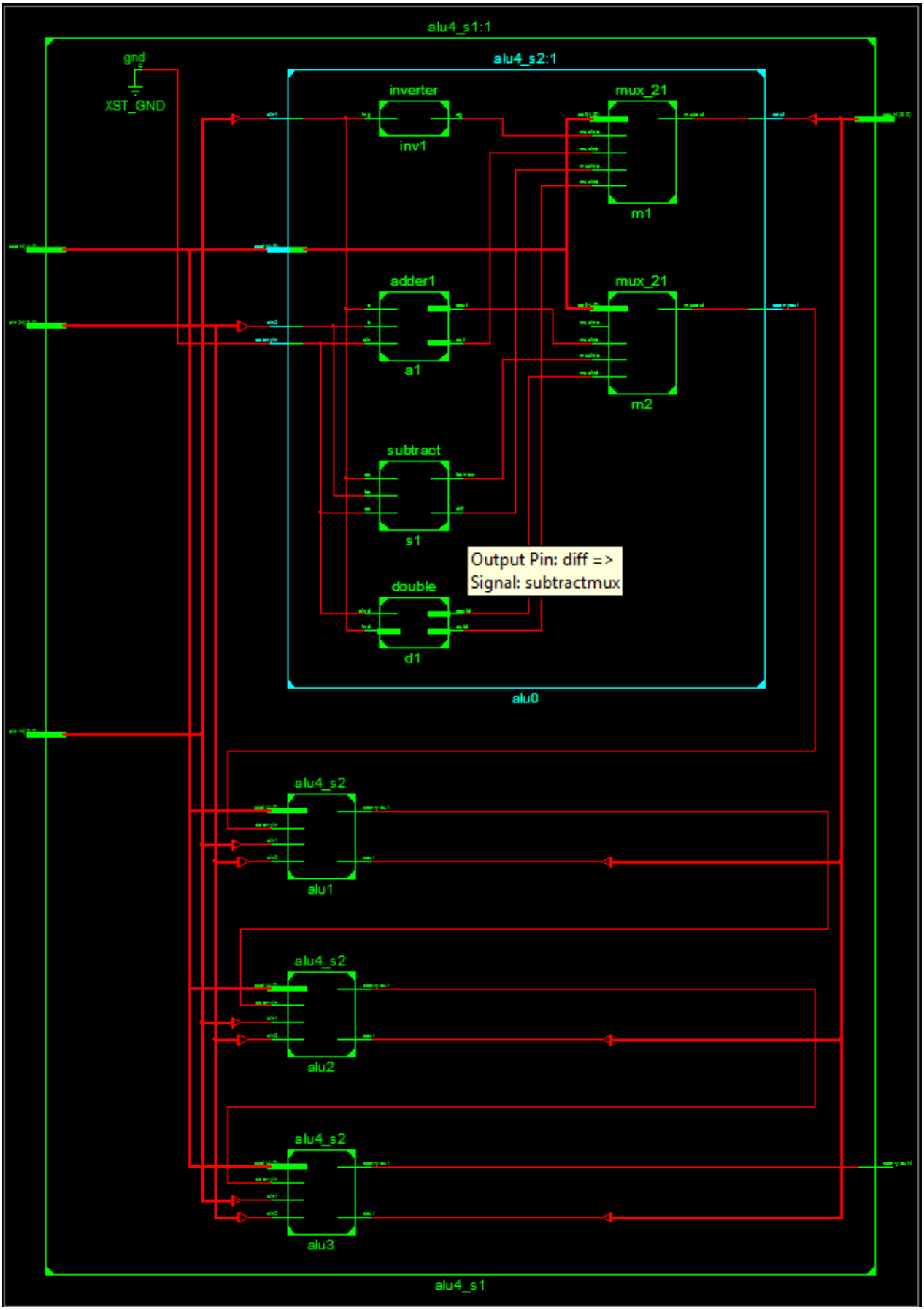
The second part was implemented by using 1-bit behavioral models of an inverter, adder, subtractor, doubler, and multiplexer. These components were then combined to create a 1-bit ALU which was used to create a 4-bit ALU. The functions are carried out using arithmetic and bitwise operators for the inverting, adding, subtracting, and doubling functions. The 4-bit ALU is capable of handling inputs ranging from 0 to 1111 and outputs ranging from 0 to 11111 through the use of a carry-out bit.

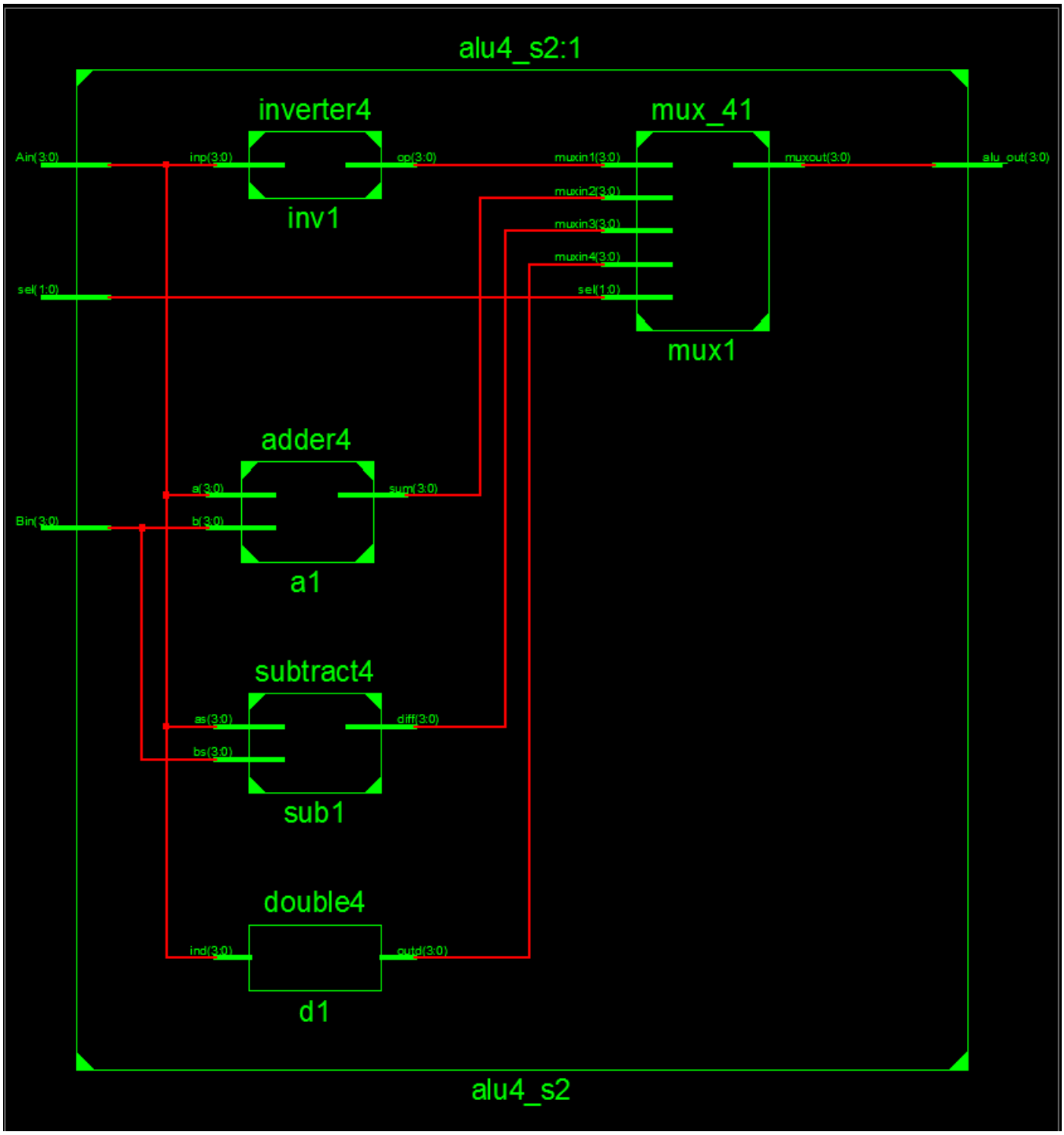
The third part was implemented by using 4-bit models of an inverter, adder, subtractor, doubler, and multiplexer. The functions are carried out using arithmetic and bitwise operators for the inverting, adding, subtracting, and doubling functions. The 4-bit components were combined structurally to create a 4-bit ALU capable of handling I/O ranging from 0 to 1111.

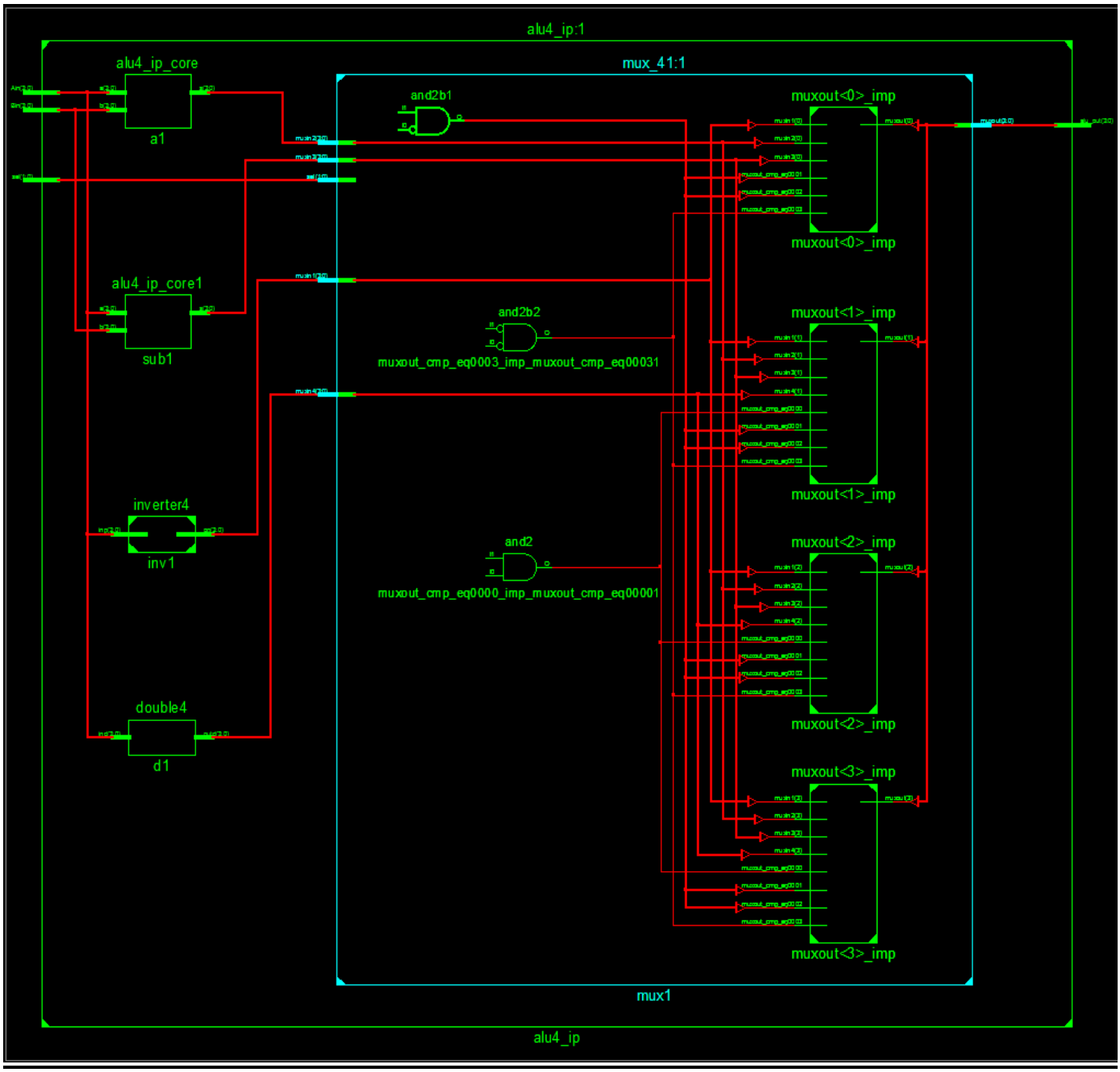
The fourth part was implemented by using a multiplexer, and an IP Core design for the adder and subtractor, while the inverter, and doubler components used arithmetic and bitwise operators. This ALU is capable of handling I/O ranging from 0 to 1111.

Here are schematic views of part 1, part 2, part 3 and part 4 respectively;



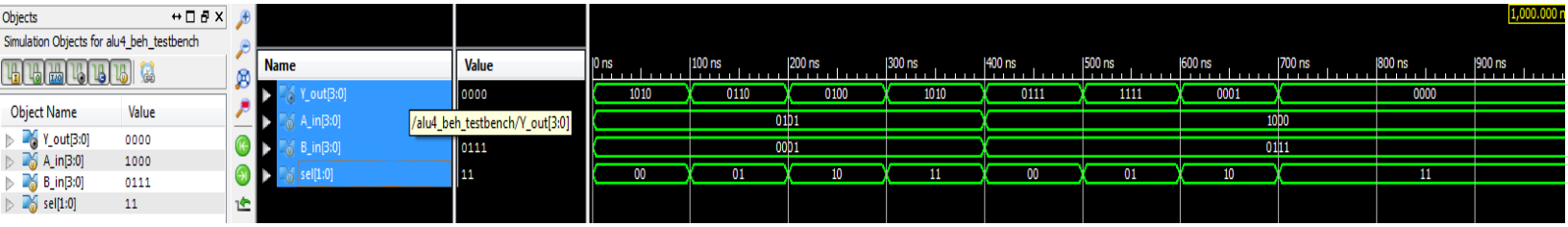

alu4_s1:1
gnd
XST_GND
alu4_s2:1
inverter
inv1
mux_21
m1
adder1
a1
mux_21
m2
subtract
s1
Output Pin: diff =>
Signal: subtractmux
double
d1
alu0
alu4_s2
alu1
alu4_s2
alu2
alu4_s2
alu3
alu4_s1

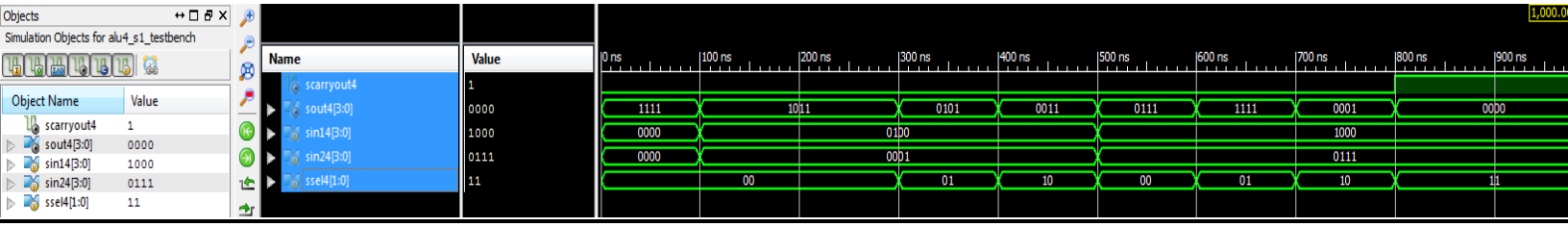

alu4_s2:1
inverter4
mux_41
Ain(3:0)
inp(3:0)
op(3:0)
muxin1(3:0)
muxout(3:0)
alu_out(3:0)
inv1
muxin2(3:0)
muxin3(3:0)
muxin4(3:0)
sel(1:0)
sel(1:0)
mux1
adder4
a(3:0)
sum(3:0)
Bin(3:0)
b(3:0)
a1
subtract4
as(3:0)
diff(3:0)
bs(3:0)
sub1
double4
ind(3:0)
outd(3:0)
d1
alu4_s2

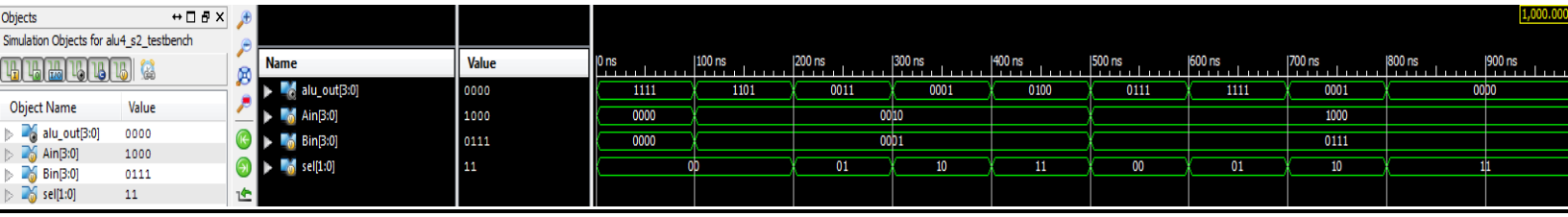

alu4_ip:1
alu4_ip_core
a1
mux_41:1
and2b1
muxout<0>_imp
muxout<0>_imp
alu4_ip_core1
sub1
and2b2
muxout_cmp_eq0003_imp_muxout_cmp_eq00031
muxout<1>_imp
muxout<1>_imp
inverter4
inv1
and2
muxout_cmp_eq0000_imp_muxout_cmp_eq00001
muxout<2>_imp
muxout<2>_imp
double4
d1
muxout<3>_imp
muxout<3>_imp
mux1
alu4_ip

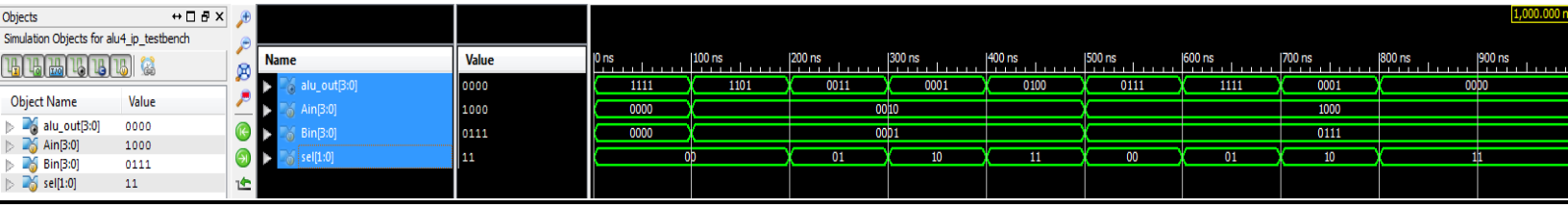
## Testing:

The testing process for part 1 through part 4 remained very similar. Multiple inputs were tested while varying the select bits to test the various functions. The waveforms for part 1, part 2, part 3, and part 4 are shown respectively;




## Results:

Part 1, 3, and 4 will give the same results for their outputs even though they are each implemented in different ways, there isn't much exciting going on there, but, with part 2 there is some variation. Part 2 contains a carry out bit as a way of accommodating 5-bit output values. The carry out bit takes the place of the MSB, as can be seen in the waveform for doubling the sin14 input of 1000, the carry out bit is set to 1 so the output is 10000. When the output does not require 5 bits the carry out bit is a 0.

**Conclusion:**

This lab provided me with the opportunity to experiment with Verilog far more than I was expecting for the first lab. It was interesting to see how many different ways hardware can be programmed to perform the same functions using Verilog code. Using the versatility of Verilog code I was able to create multiple ALU's to perform the same functions but using different approaches. Experimenting with a carry out bit proved to be challenging but the TA's provided good direction and ideas for implementing the required designs.