John Gangemi
U 6871-4612

Computer Systems Design - Lab 1

Introduction

The purpose of this lab assignment was to provide students with a quick introduction to the Xilinx ISE Design Suite and Verilog, a hardware description language loosely based on the C programming construct. Through the creation of a 4-bit ALU fundamental concepts were reinforced and a distinction between behavioral and structural Verilog was seeded. Verification of circuit design using timing diagrams from the ISim software extension of ISE was shown to be a powerful tool in testing and debugging designs.

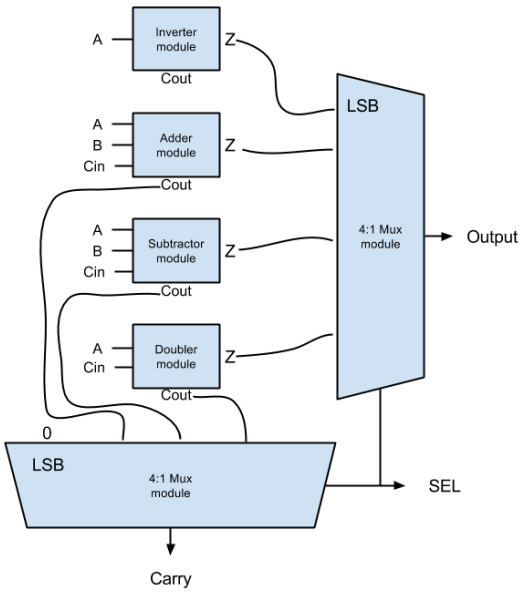
Design

Part 1

This behavioral design was implemented using a single module, thus the behavior of the ALU given by the Function Table could be described in a simple ‘case’ statement were the control variable is derived from the 2-bit select signal (S). Arithmetic, Shift, and Bit-wise operators allowed for relatively easy assignment to the ALU’s main output.

Part 2

For this structural design based off 1-bit components it was necessary to sketch a basic block diagram with hope that it might reduce logical errors and increase productivity. Each 1-bit behavioral component (module) was connected in such a manner to form a 1-bit ALU.



By connecting the carry-out and carry-in of four 1-bit ALU modules a 4-bit ALU module was created.

Part 3
The design for this simple structural module follows the same logic as developed in *part 2*, however minus the need for 1-bit components. 4-bit components (modules) of an inverter, adder, subtractor, and multiplexer were all created and implemented in a 4-bit ALU module.

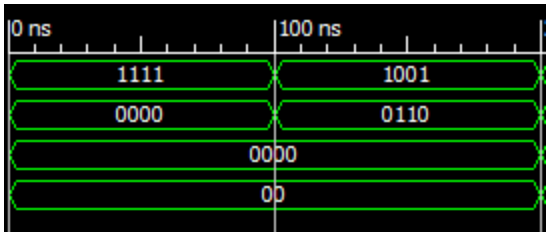
Part 4
IP Core Generation was the easiest methodology for developing more-comprehensive modules but it also took the longest to implement as load times for the program are rather long (10-15 seconds). Given the predefined HDL instantiations when creating an IP core it was easy to make the 'wire' interconnects between components and avoid long batches of code.

Testing
All four implementations of the 4-bit ALU used the same test vectors to increase the probability of accurate data. For each possible configuration of the select signals S0 & S1, of which there are four, two different input vectors were used. While this is not an exhaustive set of input vectors it does attempt to show overflow for addition and subtraction.

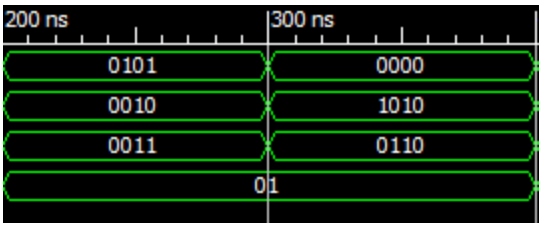
The ISim program was very useful in debugging Verilog code and figuring out how to organize test benches to show consistent changes at regular time intervals.

Results
Three of the four ALU designs show accurate data for all functions, whereas the structural design from *part 2* shows error in the ALU output when observing the subtraction function. The initial design for the subtraction module in *part 2* was to assign the module's output to '(a-b) + cin' were cin will always be zero under the assumption that all arithmetic operations in Verilog are done so using 2's complement. When this method did not produce the necessary results a series of small changes to the code with repetitive tests concluded that the most basic implementation of subtraction was by far the most reliable, that is too invert all bits for input B and then carry-in a '1' where basic addition can be performed on the two binary numbers resulting in 2's complement for the answer.

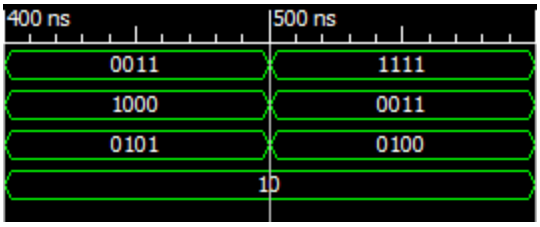
alu4_beh results:
First row is ALU output, Second row is input A, Third row is input B, Fourth row is select line



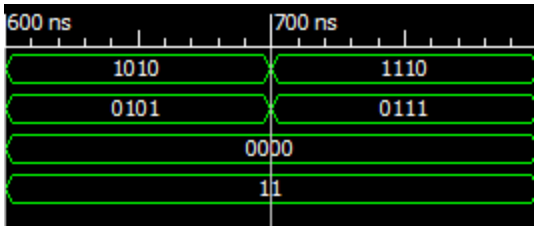
This image shows the inverting function on input A (second row)



This image shows the addition function, notice overflow from (10 + 6)

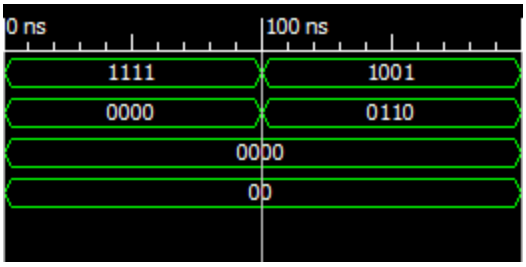


This image shows the subtraction function

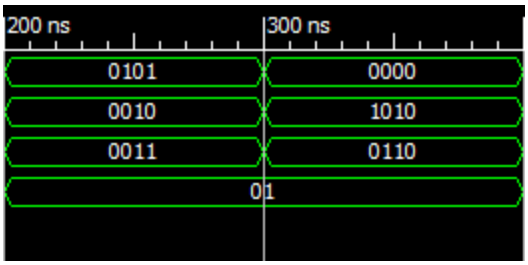


This image shows the doubling of input A (second row)

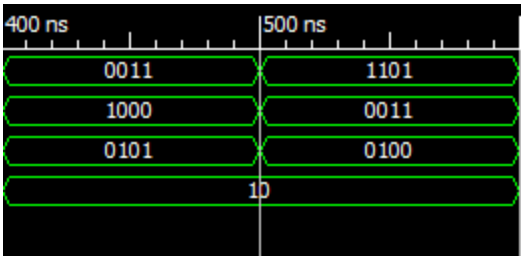
alu4_s1 results:
First row is ALU output, Second row is input A, Third row is input B, Fourth row is select line



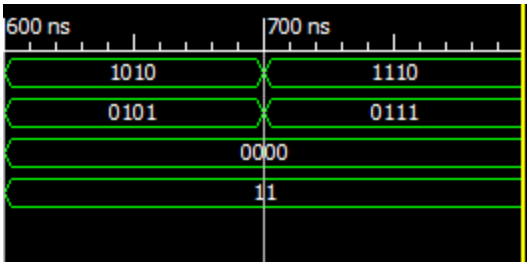
This image shows the inverting function on input A (second row)



This image shows the addition function, notice overflow from (10 + 6)



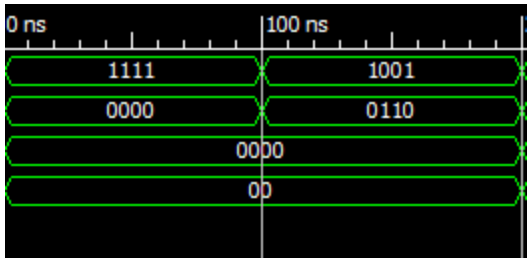
This image shows the subtraction function, notice that for 0011 - 0100 the result should be 1111 and not 1101, could not find solution...most likely the implementation is wrong



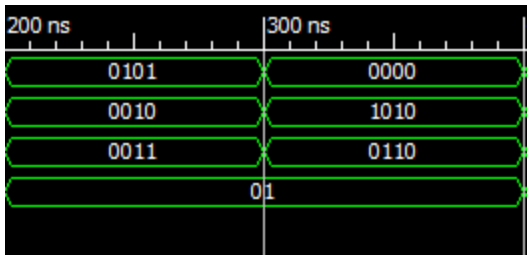
This image shows the doubling of input A (second row)

alu4_s2 results:

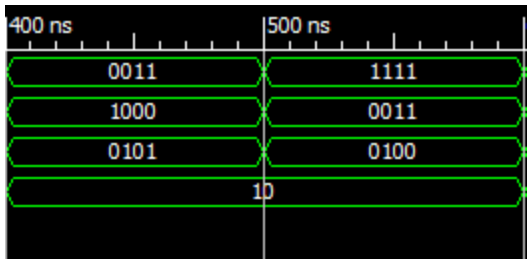
First row is ALU output, Second row is input A, Third row is input B, Fourth row is select line



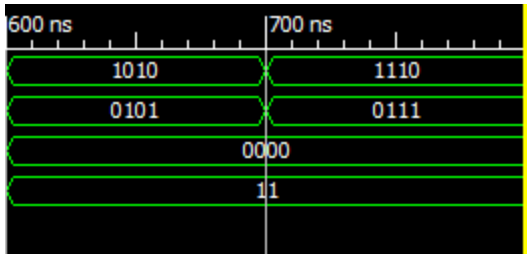
This image shows the inverting function on input A (second row)



This image shows the addition function, notice overflow from (10 + 6)



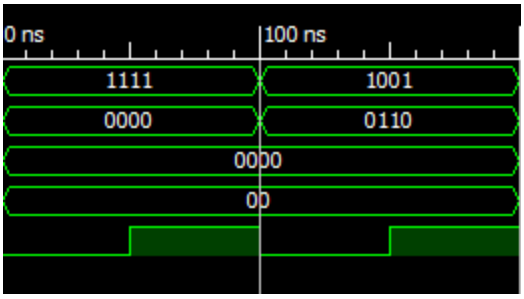
This image shows the subtraction function



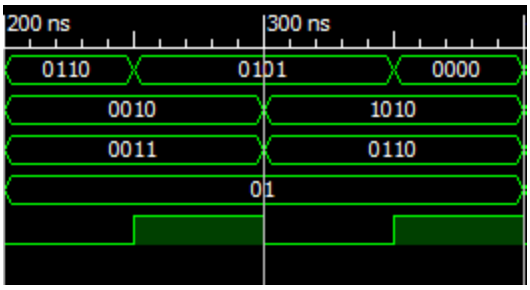
This image shows the doubling of input A (second row)

alu4_ip results:

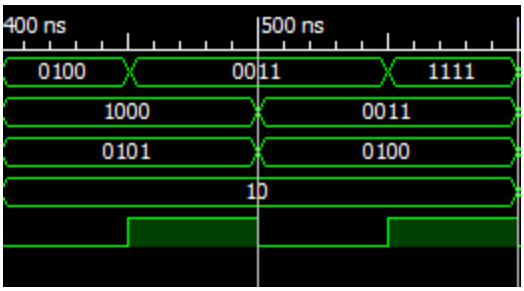
First row is ALU output, Second row is input A, Third row is input B, Fourth row is select line, Fifth row is clock



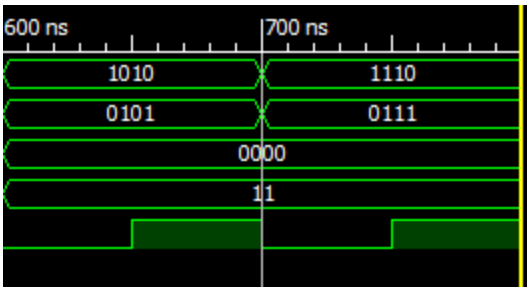
This image shows the inverting function on input A (second row)



This image shows the addition function, notice overflow from (10 + 6)



This image shows the subtraction function



This image shows the doubling of input A (second row)

Conclusion

While this was not overly difficult it was challenging occasionally especially when introduced to all new software/syntax for the first time. However, it does not seem to be an arduous task to learn the ‘ins and outs’ of Verilog. In fact, this lab assignment could have been finished days ago if it were not for *part 2*, the 1-bit Subtractor module caused a lot of headaches. While its disappointing to not have it fully functional it is really irrelevant to scope of this assignment as it was shown there is a multitude of ways to design circuits.