ECE472 Lab 2

Ashtyn Moehlenhoff
Torben Rasmussen

Overview:

For this lab, we created a 16-bit ALU to handle 5 different opcodes for: 'AND', 'OR', 'add', 'subtract' and 'set on less than'. We were asked to use carry-look-ahead addition and set the zero/overflow flags correctly.

In order to implement this ALU using carry-look-ahead addition, we created an ALU Slice which takes 4-bit operands, a and b, a carry in bit, the 3-bit opcode, and 'Less'. This slice uses the CLA from lab 1 to add the two 4-bit operands. The slice also finds the result of and'ing and or'ing the four bit operands together. Now that we have all the possible results, we use a 4-to-1 MUX to get the correct output (based on the opcode).

In order to get the 'set on less than' operation to work, we had to create a specialized MSB ALU Slice that has an output, 'set', which is set if a < b (so if the result of a-b is negative, meaning the msb=1, then 'set'=1).

We combine these slices into a 16-bit ALU which is a combination of 3 ALU slices and 1 of the specialized MSB slices. The specialized MSB slice passes the 'set' bit into the first ALU slice, so that if a < b (and opcode==111), the output from the ALU will be 0x0001.

In order to get the zero flag working, we simply check if the output from the ALU is zero. If the output is zero, the flag is set. In order to get the overflow flag working correctly, we check if operands a and b are both positive but we get a negative output, the overflow flag is set. If both operands are negative but we get a positive output, the overflow is set. If one of the operands is positive and another is negative, there is no chance of overflow, so the overflow flag isn't set.

In order to check that our 16-bit ALU is working, we tested each operation with different inputs. You can see the results of our tests in the rest of this document.

Estimated time: ~6hrs

Most valuable: learning more verilog syntax, working with opcodes and mux's.

Least valuable: implementing CLA was repetitive and not very helpful.

Improvements: none

Test 'AND':

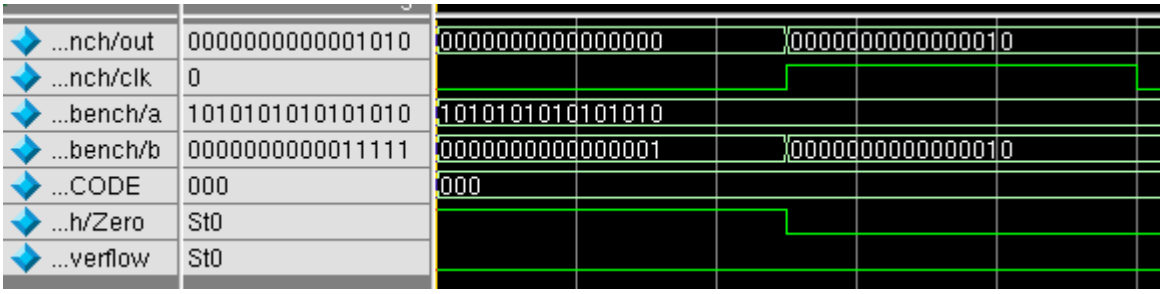
Case 1: a = 16'b0101010101010101; b = 16'b1010101010101010; expected = 16'b0000000000000000

You can see that the output is as we expected, which means this test case passes. Also, the zero flag has been set because the output is all zeros, which is correct!

Case 2: a = 16'b1010101010101010; b = 16'b1010101010101010; expected = 16'b1010101010101010

You can see that the output is as we expected, which means this test case passes. The output is no longer zeros, so the zero flag not being set is correct!

Case 3: a = 16'b1010101010101010; b = 16'b0000000000000001; (and b=b+1 on posedge)



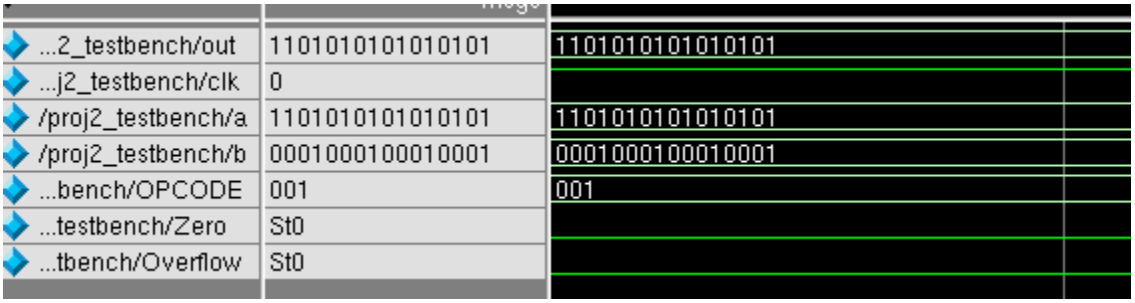
Now you can see that the results for some type of "random" input is correct, and the zero flag is still being set correctly. In all of these cases, the overflow flag is zero. This is because there is no overflow when you're doing an AND operation.

Test ‘OR’

Case 1: a = 16'b0000000000000000; b = 16'b0000000000000000; expected = 16’b0000000000000000

The output is as expected, and the zero flag is set because the output is all zeros.

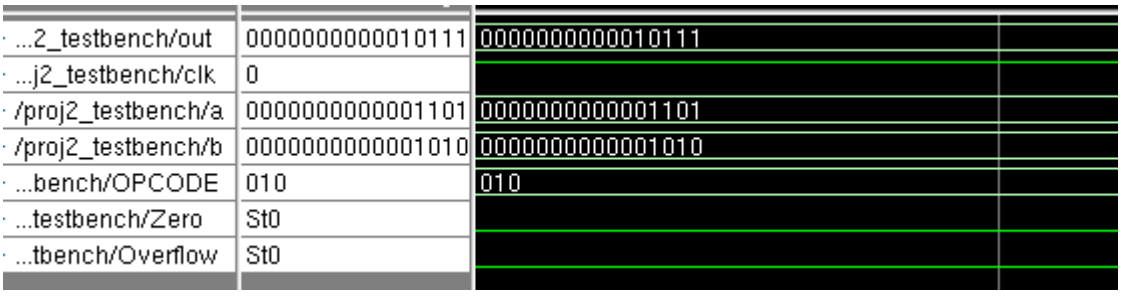
Case 2: a = 16'b1101010101010101; b = 16'b0001000100010001; expected = 16’b1101010101010101



For these more inputs, you can see that the output is as expected and the zero flag is no longer set.

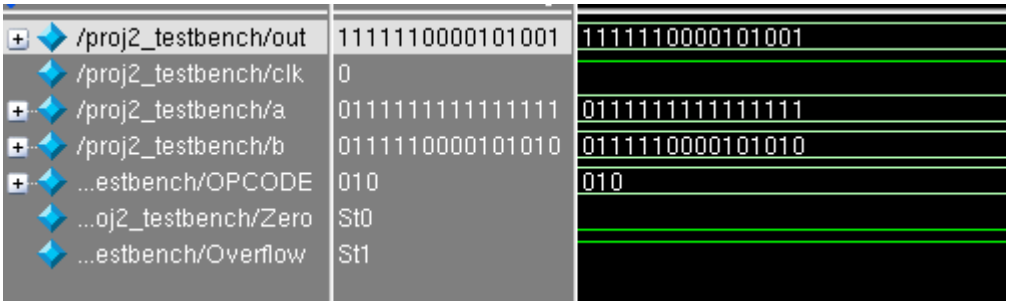
Test ‘add’:

Case 1: (test a simple carry) a = 16'b0000000000001101; b = 16'b0000000000001010; expected = 16’b0000000000010111



For these inputs the output is as expected. The zero and overflow flags aren’t set, which is correct.

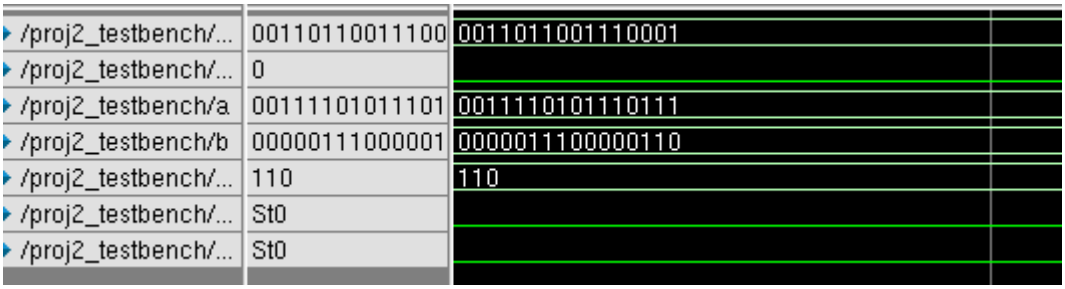
Case2: (test overflow) a = 16'b0111111111111111; b = 16'b0111110000101010;
expected = 16'b1111110000101001



For these inputs the overflow flag should be set because we're adding two positive numbers and the result is a negative number.

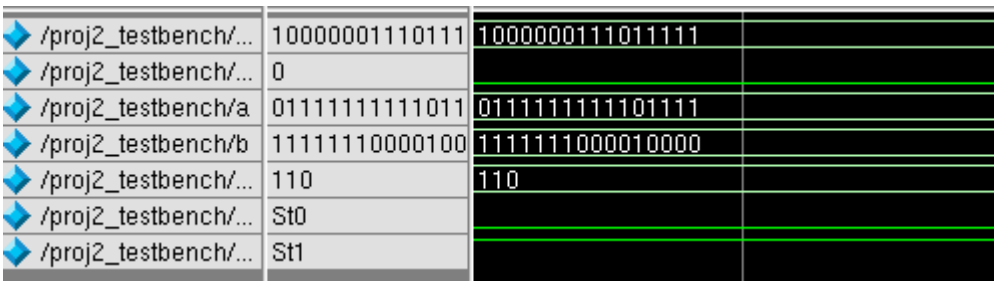
Test 'subtract':

Case 1: (with borrowing) a = 16'b0011110101110111; b = 16'b0000011100000110;
expected =16'b0011011001110001



The output is as expected, success!

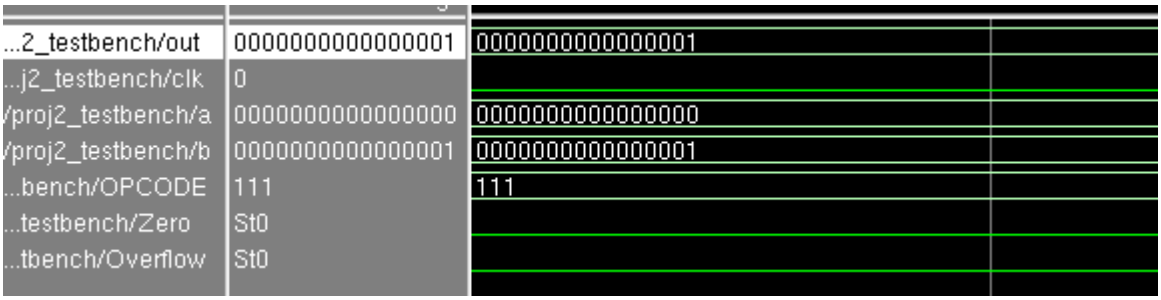
Case 2: (overflow) a = 16'b0111111111101111; b=16'b1111111000010000;
expected=16'b1000000111011111 (overflow set).



Here, because we are subtracting a negative number from a positive number, we should get a positive result. The fact that we get a negative result means we have overflow!

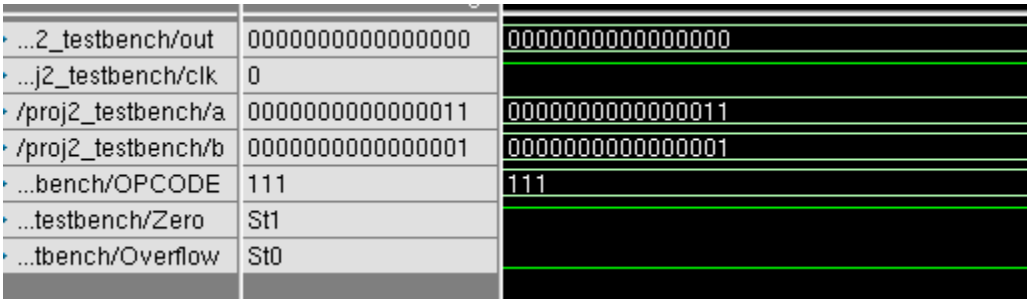
Test ‘set on less than’:

Case 1: (a<b) a = 16'b0000000000000000; b = 16'b0000000000000001;
expected = 16’b0000000000000001



Because a is less than b, the output should be 0x0001 and it is!

Case 2: (a>b) a = 16'b0000000000000011; b = 16'b0000000000000001;
expected = 16’b0000000000000000



Because a is greater than b, the output should be 0x0000 and it is! Also in this case the zero flag is set, which is correct.