Billy Koech and Nicolas Weinger

CS 141: Computing Hardware

Lab 2: ALU

Testing and Simulation methodology

The ALU is tested by iterating over all the possible OP codes and for each OP code passing different values of input X and Y. To begin, the testbench tests the functionality of all the logic gates and the adder by passing the standard case inputs listed below:

1. Two positive numbers: X = 2, Y = 3.
2. A maximum 32-bit number and a zero: X = 4294967295, Y = 0.

The testbench then tests for the following special cases:

1. Two negative numbers: X = -2, Y = -3
2. A positive and a negative number: X = 3, Y= -3
3. A maximum 32-bit number and 1: X = 4294967295, Y = 1
4. A big positive number and big negative number: X = 4294967295, Y = -4294967295;
5. Equality test: X=51, Y=51.

The ALU passed the tests by fulfilling the following expectations:

* The sum of a max 32-bit number and 1 triggered the overflow. The sum of two negative number also triggered the overflow.
* The logic gates followed the expected outputs as per their respective truth tables.
* The subtractor subtracted as specified and triggered the overflow for the difference of negative max 32-bit number and -1.

**Subtraction Implementation**

To implement the subtractor, we simply use the adder. For the Y bus, we invert all the bits before connecting it to the subtractor. We additionally set carry-in in order to do an accurate 2’s complement inversion of Y. The output is simply the difference of X and Y, (X – Y).

**SLT implementation**

The SLT is built by using the subtractor module and looking at the MSB of the output. This indicates whether the number is negative or not in the case where we have two positive or two negative inputs. In the other cases, we know which input is larger and hence do not have to look at the subtraction.

This logic is summarized in the truth table in the sketches. A sum-of-products of the MSBs of X, Y and the subtraction result gives the logic for the SLT output. It is checked by using previous test values and observing the anticpated outcome.

**SRL implementation**

The SRL is built by concatenating 32 zeros to the left of the input X to create a 64-bit bus. 32 MUXs are then connected to the 64 wires as per the sketch. This allows for the previously occupied bits to be set to zero when shifting occurs.

The output of the SRL is then connected to one input of a 2:1 mux. This mux connects the SRL to the output Z only if the select ‘shamt’ Y is less than or equal to 11111 (32base2). Otherwise it connects the mux to ground (zeros) thus when the shamt is greater than 11111 then the output is 32’b0.

The SRL is tested by checking whether shifting the Xs specified above shift by their corresponding Ys.

**SLL implementation**

The SLL is similar to the SRL except that it concatenates 32 zeros to the right of the input X to create a 64-bit bus. 32 MUXES are then connected to the 64 wires as per the sketch below. This allows for the previously occupied bits to be set to zero when shifting occurs.

The output of the SLL is then connected to one input of a 2:1 mux. This mux connects the SRL to the output Z only if the select ‘shamt’ Y is less than or equal to 11111 (32base2). Otherwise it connects the mux to ground (zeros) thus when the shamt is greater than 11111 then the output is 32’b0.

The SLL is tested by checking whether shifting the Xs specified above shift by their corresponding Ys.

**SRA implementation**

The SRA is implemented almost identically to the SRL. The difference lies in how the 64-bit bus is constructed. Instead of concatenating 32 zeros to the left of X, we concatenate 32 MSBs of X to the left of X. This allows for the previously occupied bits to be set to the MSB of X when shifting occurs.