Teo Patrosio
Lab Assignment #1
EE-126/COMP-46: Computer Engineering w/lab
Professor: Mark Hempstead TA:Parnian Mokri
Tufts University, Fall 2021

Introduction + Purpose

In this lab, I will show my implementation of the AND2, MUX5, MUX64, SignExtend, ShiftLeft2, and 32-bit program counter (PC). These pieces specifically are supposed to be the building blocks of the long term ultimate goal of designing a pipelined version of an ARM processor.

Components built

AND2

The first component built was the AND2, which performs the and operation with two inputs. This component was reused from Lab0. Here there are four different test cases, visible in Figure 1.

- The first (from 0ns to 50 ns) the two inputs, sig0 and sig1 are set to 0, as expected the output of the and2 gate resulted in a zero.
- The second (from 50ns to 100 ns) the two inputs, sig0 and sig1 are set to 0 and 1, as expected the output of the and2 gate resulted in a zero.
- The third (from 100ns to 150 ns) the two inputs, sig0 and sig1 are set to 1 and 0 as expected the output of the and2 gate resulted in a zero.
- The last (from 150ns to 200 ns) the two inputs, sig0 and sig1 are set to 1 and 1, as expected the output of the and2 gate resulted in a one.

I used a behavioral modeling type due to the clarity of seeing the AND2 outputs on the waveforms.

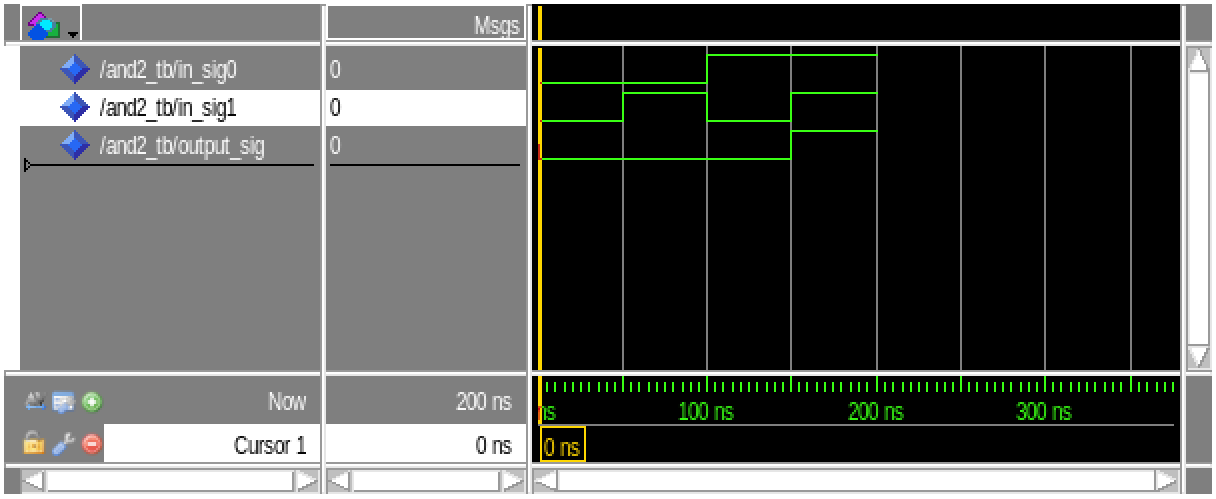

Figure 1: Waveforms for AND2 component

MUX5

The second component, moving up in complexity was two-input 5-bit mux. To test the Mux5, I preset the two mux options as b00000 and b11111. As the select stitches from being set to 1 to 0, the output changes from being b11111 (mux input 1) to b00000 (mux input 0). I again used a behavioral model so the change over time as the set value changed would be clearly visible in Figure 2.

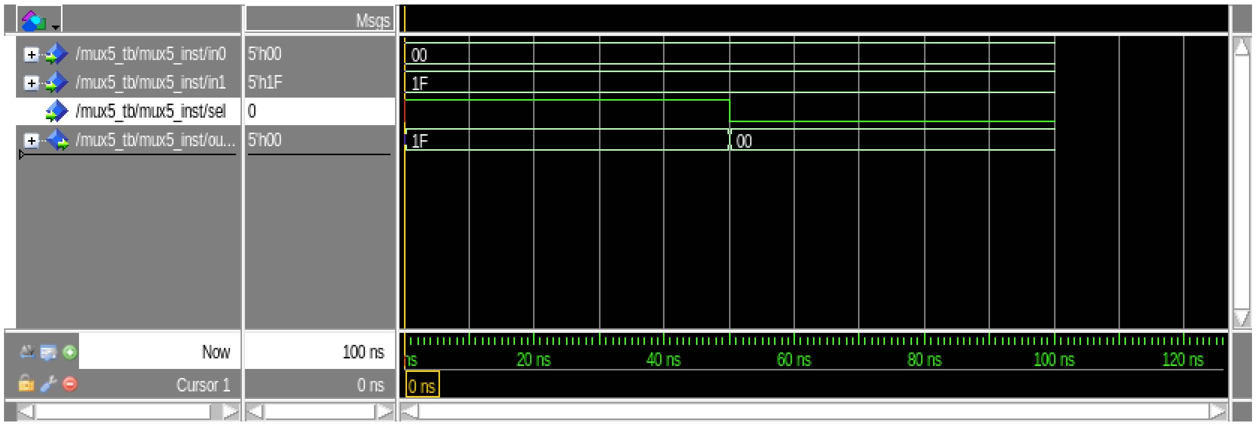

Figure 2: Waveforms for MUX5 component

MUX64

The third component, moving up in complexity was two-input 32-bit mux. To test the Mux5, I preset the two mux options as b000000000000000000000000000000000000 and b1111111111111111111111111111111111111111. As the select stitches from being set to 1 to 0, the output changes from being b1111111111111111111111111111111111111111 (mux input 1) to b000000000000000000000000000000000000(mux input 0). I again used a behavioral model so the change over time as the set value changed would be clearly visible in Figure 3.

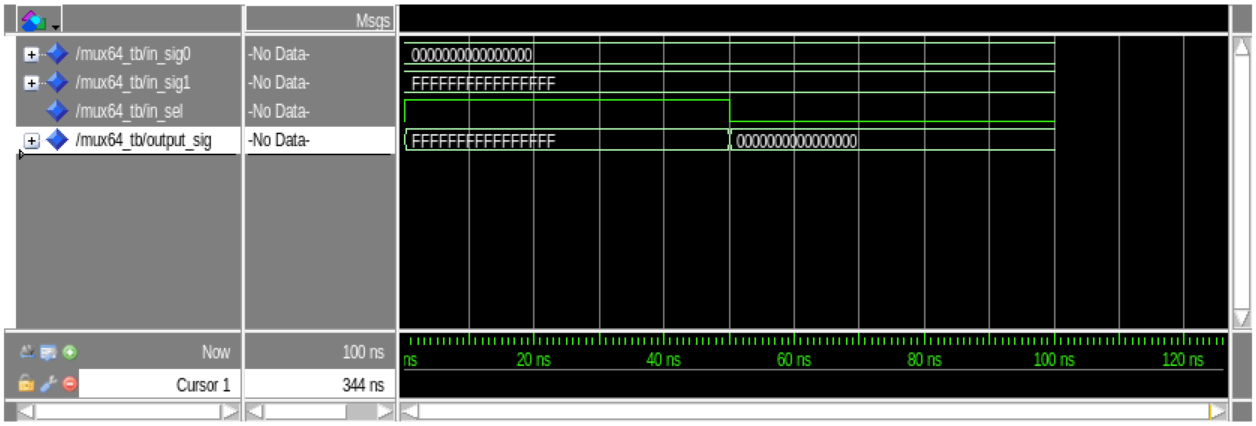

Figure 3: Waveforms for MUX32 component

SignExtend

The fourth component is a sign extension for 32-bit to 64-bit integers. This is required due to how two's complement scales with large negative numbers. In figure 4, four different test cases are shown and described below.

- The first test case (from 0 to 50 ns) shows an input of 0x00000000 to 0x0000000000000000. Since this is a positive number, its signed representation does not affect higher digits.
- The second test case (from 50 to 100 ns) shows an input of 0x00000001 to 0x0000000000000001. Since this is also a positive number, its signed representation does not affect higher digits.
- The third test case (from 100 to 150 ns) shows an input of 0x80000000 (b10000000000000000000000000000000) which is a negative number so "a negative sign" needs to be propagated through the rest of the number resulting in an 0xFFFFFFFF80000000.
- The last test case (from 150 to 200 ns) shows an input of 0x7FFFFFFF, which is 2^{31} - 1, the highest positive number. Since this is a positive number, the positive value was able to be distinguished and extended like the first two cases.

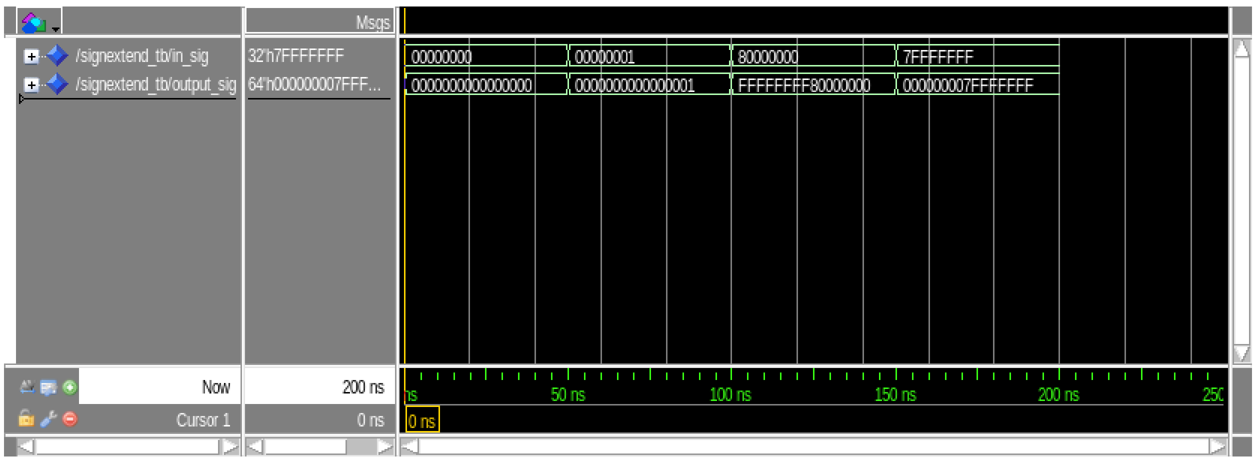

Figure 4: Waveforms for SignExtend

This was tested through a behavioral model so one input could be tested at a time and checked. (A concurrent model could also have been used testing multiple signextend modules at once.)

ShiftLeft2

The Shift Left module takes a 64 bit input and shifts it left by 2 bits. This component was tested through two simple test cases listed below.

- The first test case was an input of 63 zeros and a 1 on the position of the lsb (right). This was expected then to have an output of b0000000000000000000000000000000000000000000000000000000000000100, which can be seen in Figure 5.
- The second test was an input of also 63 zeros and a 1, but this time the 1 was on the position of msb (right). This bit was expected to then be lost as the input is shifted left. This results can be seen as b0000000000000000000000000000000000000000000000000000000000000000.

Figure 5: Waveforms for ShiftLeft2

PC

The last module that was made was called the Program counter that is based on setting the address out as address in, on a 32-bit rising-edge if a write-enable parameter is set. There is also asynchronous reset, which resets the address out to zero.

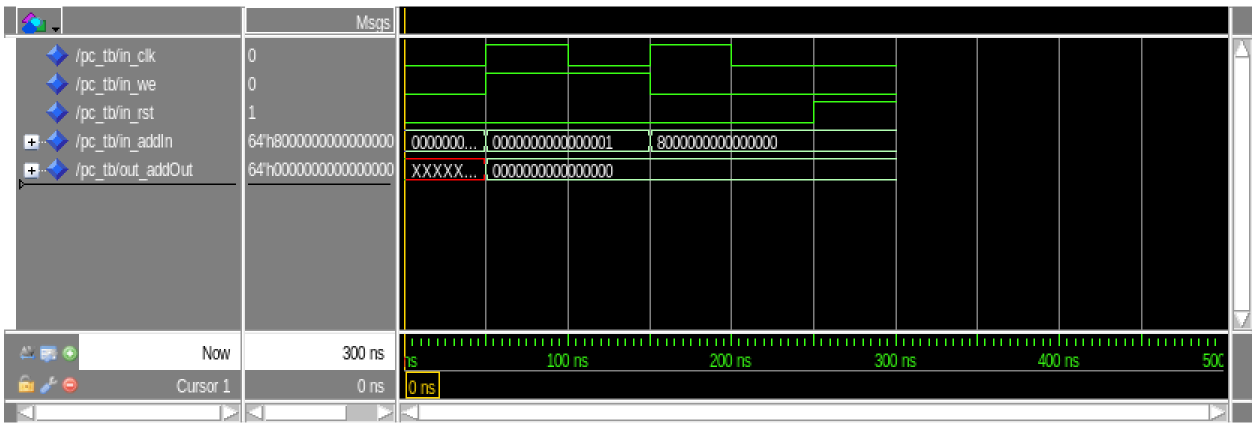

Figure 6: Waveforms for PC

For the PC test bench, I tested first that the address Out wouldn't be set until a rising edge which can be seen at 50 ns in the top and bottom signal. Then at 150 ns I tested that setting the write enable as false will not actually make the out address change. This appeared to work. Finally I pause the clock and test the asynchronous.

The PC needs to be tested in a process-based way to see how signals change over time (specifically the rising clock edge).