Lab 2 - Program Counter

Jon Johnston and Justin Roessler

February 4, 2019

1 Executive Summary

The purpose of this lab was to make a 64-bit adder and a 64-bit mux. These modules will be implemented into our ARM processor. Also two test benches were created to test each module. The adder will update our Program Counter with each clock cycle and the mux will be used to set the Program Counter to the incremented PC or to a different address based on the control line.

Also, this lab helped us regather our knowledge with verilog and git. Tasks such as creating a new file, resetting the top module ,and updating files were a good refresher on using these programs.

2 Test Report

To verify operation of these two modules, this lab requires two separate test benches.

1. Adder Test Bench
2. Mux Test Bench

2.1 Adder Test Bench

The adder test bench contains:

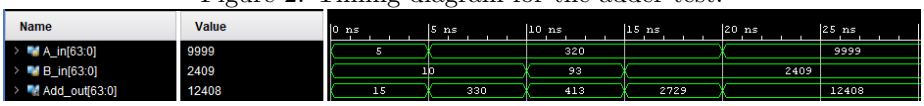
1. Inputs
   (a) A_in - the first 64-bit number
   (b) B_in - the second 64-bit number
2. Outputs
   (a) Add_out - the 64-bit sum of the two input numbers

The adder test bench sets the values of two input numbers, a and b, and sends the numbers into the adder module. The adder module adds these two numbers together and gives us our result. Correct operation is verified by comparing the Simulation Results with the Expected Results Table. After analyzing the results, the adder works as expected.

Figure 1: Expected Results of the adder test.

Time(ns)	0-5	5-10	10-15	15-20	20+
A_in	5	320	320	320	9999
B_in	10	10	93	2409	2409
Add_out	15	330	413	2729	12408

Figure 2: Timing diagram for the adder test.



2.2 Mux Test Bench

The mux test bench contains:

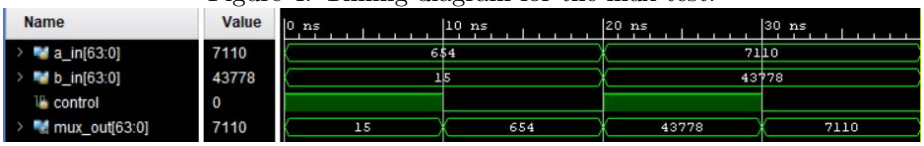
1. Inputs
   (a) a_in - the first 64-bit mux input, which is passed to mux_out when the control line is set to 0
   (b) b_in - the second 64-bit mux input, which is passed to mux_out when the control line is set to 1
   (c) control - the 1-bit control input to the mux which determines whether a or b will be passed to mux_out
2. Outputs
   (a) mux_out - the 64-bit output of the mux

The mux_test bench sets the values of a, b, and the control line and verifies that mux_out is set to the correct value. The mux module takes the inputs and gives us a when control is 0 and b when control is 1. Based on 4 cases conducted in the mux_test module, correct operation is verified by comparing the Simulation Results with the Expected Results Table. After analyzing the results, the mux works as expected.

Figure 3: Expected Results of the mux test.

Time (ns)	0 to 10	10 to 20	20 to 30	30 to 40
a_in	654	654	7110	7110
b_in	15	15	43778	43778
control	1	0	1	0
mux_out	15	654	43778	7110

Figure 4: Timing diagram for the mux test.



3 Code Appendix

Listing 1: Verilog code for testing the adder.

```
`include "definitions.vh"

module adder_test;

    reg [`WORD-1:0] A_in, B_in;
    wire [`WORD-1:0] Add_out;

    adder add(
        .a_in(A_in),
        .b_in(B_in),
        .add_out(Add_out)
        );

initial
begin
    A_in = 5;
    B_in = 10; #5;
    A_in = 320; #5;
    B_in = 93; #5;
    B_in = 2409; #5;
    A_in = 9999; #5;
```

```
end
endmodule
```

Listing 2: Verilog code for the adder.

```
`include "definitions.vh"

module adder(
    input [`WORD-1:0] a_in,
    input [`WORD-1:0] b_in,
    output [`WORD-1:0] add_out
    );

    // add_out is the sum of a_in and b_in
    assign add_out = a_in + b_in;

endmodule
```

Listing 3: Verilog code for testing the mux.

```
`include "definitions.vh"

module mux_test;  //Test for 2way mux
 //a_in == in0
 //b_in == in1
 //When control == 0 it selects a_in
 //When control == 1 it selects b_in

reg [`WORD-1:0]a_in, b_in;
reg control;
wire [`WORD-1:0] mux_out;
mux MUX (.a_in(a_in), .b_in(b_in), .control(control), .mux_out(mux_out));
initial
    begin
            a_in= 654;
            b_in= 15;
            control= 1; #10;

            a_in= 654;
            b_in= 15;
            control= 0; #10;

            a_in= 7110;
            b_in= 43778;
            control= 1; #10;
```

```
            a_in= 7110;
            b_in= 43778;
            control= 0; #10;
            $finish;
    end

endmodule
```

Listing 4: Verilog code for the mux.

```
`include "definitions.vh"

module mux#(
    parameter SIZE=64)(
    input [SIZE-1:0] a_in,
    input [SIZE-1:0] b_in,
    input control,
    output [SIZE-1:0] mux_out
    );

    assign mux_out = (control) ? b_in : a_in;//control is your selector
    //When control == 0 it selects a_in
    //When control == 1 it selects b_in

endmodule
```