Lab 2: Program Counter Incrementer

Matthew Carrano and Breana Leal

January 23, 2018

1 Introduction

The purpose behind Lab 2 is to address the ARM's counting and selection capabilities. Counting introduced the assembly of an Incrementer, the Incrementer uses an adder to allow the program counter to jump to the next instruction in memory. Selection engaged a Mux, a mux allows for the ARM to decide where to jump in memory, either sequentially or to begin branching, based on given inputs.

The lab reiterates the use of Verilog modules, testbenches, and simulations for code debugging.

2 Interface and Implementation

The incrementer is comprised of an adder. The adder module has two inputs, Ain and Bin, and one output add-out each of [WORD-1:0] length. In Listing 1, the assign statement adds both inputs and assigns the value to the output (add-out). Verilog comprehends higher level programming compared to assembly and does not require gate level details, thus the addition sign is best to use for implementation. The mux has two inputs (Ain and Bin) of [SIZE-1:0] length, a single bit selector input, and an output (mux-out) of [SIZE-1:0] length. SIZE is a defined parameter within the mux module of length 8. The assign statement assigns the output (mux-out) a value (Ain or Bin) based on the controlling input (control). Again, Verilog engages the use of the symbols to avoid binary gate logic.

Listing 1: Verilog code for an Adder.

```
`timescale 1ns / 1ps
`include "definitions.vh"

module adder(
    input [`WORD-1:0] Ain,
    input [`WORD-1:0] Bin,
    output [`WORD-1:0] add_out
    );
    assign add_out = Ain + Bin;

endmodule
```

Listing 2: Verilog code for a Mux.

```
`include "definitions.vh"

module mux#(
    parameter SIZE=8)(
    input [SIZE-1:0] Ain,
    input [SIZE-1:0] Bin,
    input control,
    output [SIZE-1:0] mux_out
    );
    assign mux_out = control?Bin:Ain;
endmodule
```

3 Test Bench Design

Listing 3 breaks down cases where 4 is continuously being added to the output after waiting a period of 200ns. Four is passed in for each case to create consistency and following convention. The program counter, therefore will have 4 as the program counter advances. There are two reg (in1 and in2) and one wire (out). The mux's testbench has its three reg inputs and one output wire. The mux has a defined parameter of 64 bits. Listing 4 demonstrates four cases when the selector value varies. in1 corresponds to Ain and is selected when con is set to 0. Inversely, in2 corresponds to Bin and is selected when the con is set to 1. Each test is separated by 200ns.

Listing 3: Verilog code for testing an Adder.

```
`timescale 1ns / 1ps
`include "definitions.vh"
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/22/2018 10:32:02 AM
// Design Name:
// Module Name: adder_test
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////


module adder_test;

    reg[`WORD - 1:0] in1;
    reg[`WORD - 1:0] in2;
    wire[`WORD - 1:0] out;

    adder UUT(
        .Ain(in1),
        .Bin(in2),
        .add_out(out)
        );

initial begin
    in1 <= 4;
    in2 <= 0;
        #200;
    in1 <= 4;
    in2 <= out;
        #200;
    in1 <= 4;
    in2 <= out;
        #200;
    in1 <= 4;
    in2 <= out;
        #200;
    in1 <= 4;
    in2 <= out;

end
endmodule
```

Listing 4: Verilog code for testing a Mux.

```
`timescale 1ns / 1ps
`include "definitions.vh"
```

```
module mux_test;

    reg[`WORD-1:0] in1;
    reg[`WORD-1:0] in2;
    reg con;
    wire[`WORD-1:0] out;

mux#(64) UUT(
    .Ain(in1),
    .Bin(in2),
    .control(con),
    .mux_out(out)
    );

 initial begin

 in1 <= 5;
 in2 <= 8;
 con <= 0;
 #200;
 con <= 1;
 #200
 con <= 0;
 #200
 in1 <= 23455;
 in2 <= 1234567898;
 con <= 1;

 end

endmodule
```

4 Simulation

Figure 1 below is the simulation for the adder testbench. in1 maintains a constant value while in2 is counting by 4s. As expected the output values are incrementing by 4. This result verifies the adder is working. Figure 2 is the mux testbench results. The first 600 ns have 5 and 8 as in1 and in2 but the out differs based on the con value. When con is set to 0 out is 5 until the time reaches 200ns and con's value changes to 1, also changing out to 8. Cases 1-3 demonstrates the mux's assign statement successes in being able to distinguish between inputs. The remaining time displays the same concept with alternate input values.

Figure 1: Timing diagram for Adder Test.

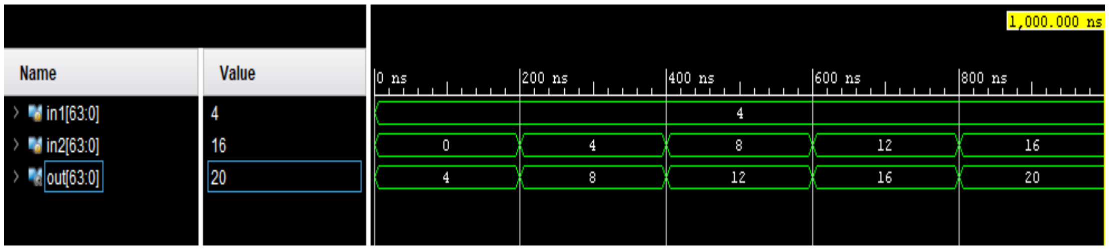
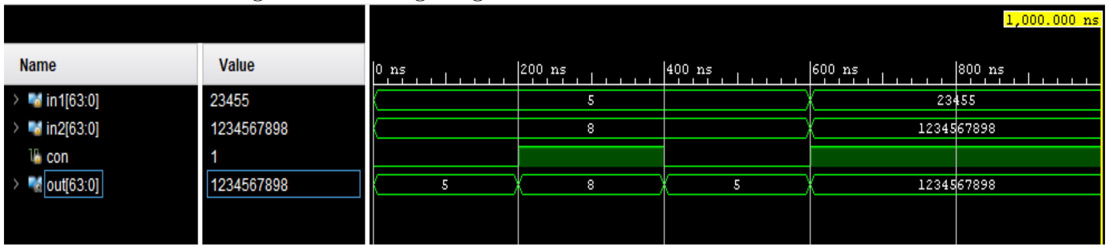


Figure 2: Timing diagram for Mux Test.



5 Conclusions

The coded Verilog files serve as the ARM's fetch section to fulfill its later goal of datapath. Lab 2 introduced an adder and mux to accomplish the program counter's incrementer and branching abilities. The adder added inputs which assigned the summation to its output. The mux had one output selected from multiple inputs based a controlling input. Ultimately, both programs contribute to the ARM's progress in memory addressing.