EEL 4712C - Digital Design: Lab Report 6

Cole Rottenberg
11062528

April 29th, 2024

Lab Report

Problem Statement

The Goal of the Lab is to explore the use of OSVVM for creating testbenches for VHDL code. The lab will focus on creating a testbench for a simple ALU design. The ALU design will have two 8-bit inputs and an 8-bit output. The ALU will have the following operations: ADD, SUB, AND, OR. The ALU will have an output signal as well as three flags: zero, positive, and negative. The testbench will be used to verify the functionality of the ALU.

Design

The design of the ALU testbench will be based on the OSVVM library. The testbench will be used to verify the functionality of the ALU. The testbench will have a process that will check each bin for an operation and if the coverage is complete. The testbench will run until all the coverages of input opcodes and output flags are met. The testbench will also have a process that will check the output flags and the operations of the ALU. We will have a final process that will generate these random inputs and opcodes for the ALU. The testbench will be used to verify the functionality of the ALU.

Implementation

The implementation followed mostly what the ring_buffer_tb did. The testbench was created with the OSVVM library. There were some minor changes due to the lack of a clock within the ALU but overall the testbench matched the one within the example ring_buffer_tb. The testbench was able to verify the functionality of the ALU. The testbench was able to verify the functionality of the ALU. The implementation of the testbench can be seen in Listing 1.

Testing

The actual testing of a testbench is a little difficult as it is often difficult to decode whether the ALU is broken or the testbench is broken. In the end the only bit of testing that was overwhelmingly helpful was the coverage data which helped my find a bug in my code regarding the operations using signed datatypes and SLVs.

Conclusions

This was on of the most helpful labs and I am excited for the opportunity to use OSVVM in the future. The lab was a success and I was able to verify the functionality of the ALU. The only problem I encountered was a bug in my code regarding the operations using signed datatypes and SLVs. In the future I will be more careful with my datatypes and how I use them in my code.

As seen below in the images, the part 1 testbench was able to compile and simulate. This can be seen in Figures 1 and 2. The ALU testbench was also able to compile and simulate. This can be seen in Figure 3.

Appendix

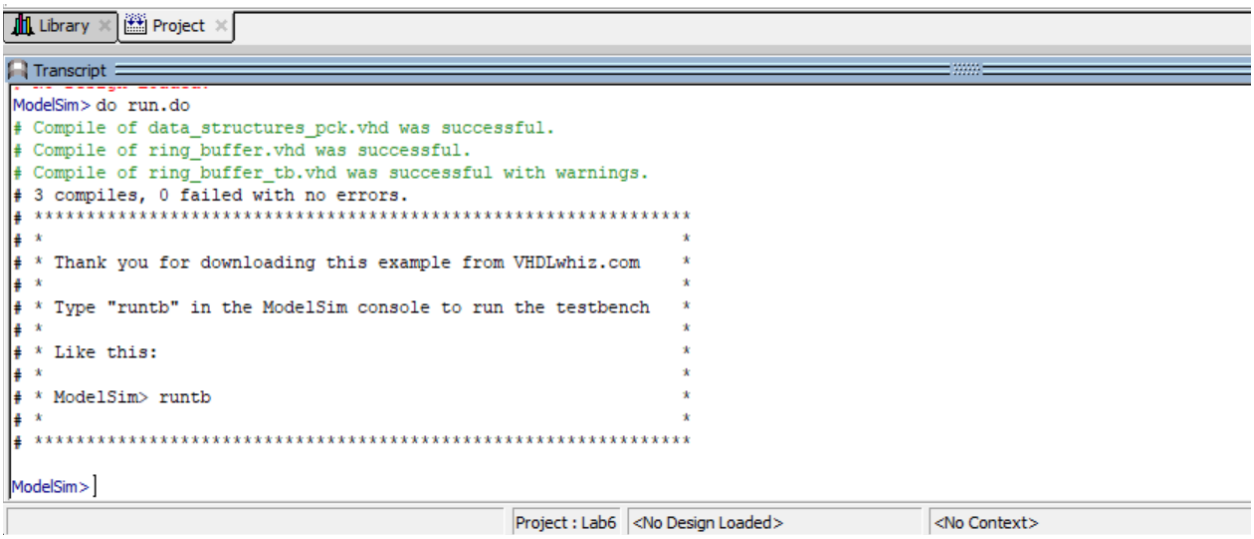

Figure 1: Testbench Compilation

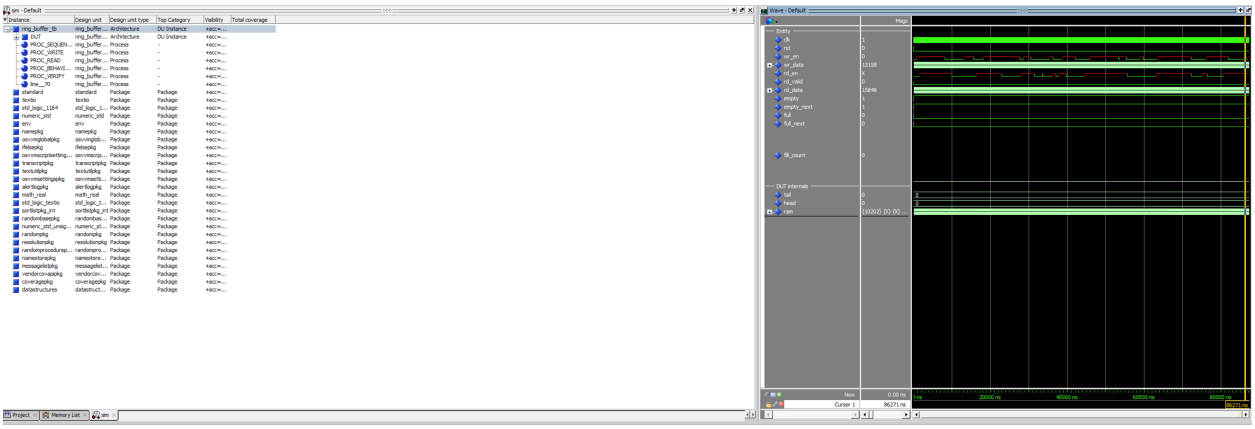

Figure 2: Testbench Simulation

```
Loading osvvm.CoveragePkg(body)
Loading work.alu_tb(tb)
Loading work.alu(arch)
** Warning: Design size of 27721 statements exceeds ModelSim-Intel FPGA Starter Edition recommended capacity.
Expect performance to be adversely affected.
** Warning: NUMERIC_STD.">": metavalue detected, returning FALSE
   Time: 0 ps  Iteration: 0  Instance: /alu_tb/dut
** Warning: NUMERIC_STD."=": metavalue detected, returning FALSE
   Time: 0 ps  Iteration: 0  Instance: /alu_tb/dut
** Note: Coverage goals met
   Time: 285 ns  Iteration: 0  Instance: /alu_tb
%% WriteBin:
%% Addition  Bin:(1)   Count = 8  AtLeast = 1
%% WriteBin:
%% Subtraction  Bin:(1)   Count = 7  AtLeast = 1
%% WriteBin:
%% And  Bin:(1)   Count = 6  AtLeast = 1
%% WriteBin:
%% Or  Bin:(1)   Count = 7  AtLeast = 1
%% WriteBin:
%% Positive  Bin:(1)   Count = 19  AtLeast = 1
%% WriteBin:
%% Negative  Bin:(1)   Count = 8  AtLeast = 1
Break in Process PROC_SEQUENCER at C:/Users/coler/Documents/uf/spring24/dd/lab6/sim/alu_tb.vhd line 99
```

Figure 3: ALU Testbench Simulation

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  use std.env.finish;
6
7  library osvvm;
8  use osvvm.RandomPkg.all;
9  use osvvm.CoveragePkg.all;
10
11
12 -- enter your code below
13 entity alu_tb is
14 end alu_tb;
15
16 architecture tb of alu_tb is
17   -- ALU signals
18   constant clock_period : time := 10 ns;
19   signal clk : std_logic := '0';
20   -- Inputs
21   constant WIDTH : integer := 8;
22   signal in0, in1 : std_logic_vector(WIDTH-1 downto 0);
23   signal sel : std_logic_vector(1 downto 0);
24
25   -- Outputs
26   signal output : std_logic_vector(WIDTH-1 downto 0);
27   signal neg : std_logic;
28   signal zero : std_logic;
29   signal posi : std_logic;
30
31   -- OSVVM Shared Variables
32   shared variable rv : RandomPType;
```

```
33    shared variable bin1, bin2, bin3, bin4, bin5, bin6, bin7 : CovPType; --
          7 coverage bins
34  begin
35    -- ALU instance
36    dut : entity work.alu
37      port map(
38        in0 => in0,
39        in1 => in1,
40        sel => sel,
41        output => output,
42        neg => neg,
43        zero => zero,
44        posi => posi
45      );
46
47    -- clock generation
48    clk <= not clk after clock_period/2;
49
50  -- Process sequencer
51
52    PROC_SEQUENCER : process
53    begin
54
55      -- Set up coverage bins for the ALU
56      bin1.AddBins("Addition", ONE_BIN);
57      bin2.AddBins("Subtraction", ONE_BIN);
58      bin3.AddBins("And", ONE_BIN);
59      bin4.AddBins("Or", ONE_BIN);
60      bin5.AddBins("Positive", ONE_BIN);
61      bin6.AddBins("Negative", ONE_BIN);
62      bin7.AddBins("Zero", ONE_BIN);
63
64      wait until rising_edge(clk);
65
66      loop
67        wait until rising_edge(clk);
68
69        -- Collect coverage data for the ALU
70        bin1.ICover(to_integer(sel = "00"));
71        bin2.ICover(to_integer(sel = "01"));
72        bin3.ICover(to_integer(sel = "10"));
73        bin4.ICover(to_integer(sel = "11"));
74        bin5.ICover(to_integer(posi = '1'));
75        bin6.ICover(to_integer(neg = '1'));
76        bin7.ICover(to_integer(zero = '1'));
77
78        -- Stop the test when all coverage goals have been met
79        exit when
80          bin1.IsCovered and
81          bin2.IsCovered and
82          bin3.IsCovered and
83          bin4.IsCovered and
84          bin5.IsCovered and
85          bin6.IsCovered and
```

```
86          bin7.IsCovered;
87      end loop;
88
89      report("Coverage goals met");
90
91      -- Print coverage data
92      bin1.WriteBin;
93      bin2.WriteBin;
94      bin3.WriteBin;
95      bin4.WriteBin;
96      bin5.WriteBin;
97      bin6.WriteBin;
98
99      finish;
100   end process;
101
102 -- Generate Random Values for the ALU
103   PROC_RANDOM : process
104   begin
105     in0 <= std_logic_vector(to_unsigned(rv.RandInt(0, 128), WIDTH));
106     in1 <= std_logic_vector(to_unsigned(rv.RandInt(0, 128), WIDTH));
107     -- All four possible ALU operations 00, 01, 10, 11
108     sel <= std_logic_vector(to_unsigned(rv.RandInt(0, 3), 2));
109
110     wait for 10 ns;
111   end process;
112
113   -- Emulate the ALU operation
114   PROC_BEHAVIORAL_MODEL : process
115   -- Variables for the ALU operation
116   -- We need to turn the SLVs into  signed
117   variable in0_int, in1_int : signed(WIDTH-1 downto 0);
118   variable output_int : signed(WIDTH-1 downto 0);
119
120   begin
121     in0_int := signed(in0);
122     in1_int := signed(in1);
123     wait until rising_edge(clk);
124     output_int := signed(output);
125     case sel is
126       when "00" =>
127         assert output = std_logic_vector(signed(in0) + signed(in1))
128           report "Addition failed"
129           severity failure;
130       when "01" =>
131         assert output = std_logic_vector(signed(in0) - signed(in1))
132           report "Subtraction failed"
133           severity failure;
134       when "10" =>
135         assert output = std_logic_vector(signed(in0 and in1))
136           report "And failed"
137           severity failure;
138       when "11" =>
139         assert output = std_logic_vector(signed(in0 or in1))
```

```
140             report "Or failed"
141             severity failure;
142         when others =>
143           report "Invalid operation"
144             severity failure;
145       end case;
146
147       -- Check the ALU flags
148       if output_int > 0 then
149         assert posi = '1'
150           report "Positive flag failed"
151           severity failure;
152       end if;
153       if output_int < 0 then
154         assert neg = '1'
155           report "Negative flag failed"
156           severity failure;
157       end if;
158       if output_int = 0 then
159         assert zero = '1'
160           report "Zero flag failed"
161           severity failure;
162       end if;
163
164     end process;
165
166 end tb;
```

Listing 1: ALU Testbench