CPE 300L
DIGITAL SYSTEM ARCHITECTURE AND DESIGN LABORATORY

LABORATORY 3
DESIGN AND TESTING OF COMBINATIONAL CIRCUITS. TESTBENCH

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
UNIVERSITY OF NEVADA, LAS VEGAS

OBJECTIVE

Get familiar with Modelsim 10.1d software for testbenches and review the design of combinational circuits.

INTRODUCTION

ModelSim eases the process of finding design defects with an intelligently engineered debug environment.

The ModelSim debug environment efficiently displays design data for analysis and debug of all languages. ModelSim allows many debug and analysis capabilities to be employed post-simulation on saved results, as well as during live simulation runs.

Features of ModelSim:

- Unified mixed language simulation engine for ease of use and performance
- Native support of Verilog, SystemVerilog for design, and VHDL, for effective verification of sophisticated design environments
- Fast time-to-debug, easy to use, multi-language debug environment
- Advanced code coverage and analysis tools for fast time to coverage closure
- Interactive and Post-Sim Debug available so same debug environment used for both
- Powerful Waveform Compare for easy analysis of differences and bugs
- Advanced code coverage and analysis tools for fast time to coverage closure
- Unified Coverage Database with complete interactive and HTML reporting and processing for understanding and debugging coverage throughout your project
- Assertions
- Gain insight and visibility into your design by letting assertions notify you of an error so it can be fixed
- ModelSim provides a powerful library of checkers (OVL) letting you debug with assertions right away, without writing your own
- Assertions can also serve as documentation for your design, as comments are embedded into the code as you go

THEORY OF OPERATION

1. Testbench

Once a Verilog model for a system has been made, the next step is to test it. A model has to be tested and validated before it can be successfully used. A *testbench* is a piece of Verilog code that can provide input combinations to test a Verilog model for the system under test. It provides stimuli to the system or circuit under test. Testbenches are frequently used during simulation to provide sequences of inputs to the circuit or Verilog model. Below – an example for a 2-input AND gate.

Verilog code for AND gate:

```
01 module and_gate (
02     input a,
03     input b,
04     output out
05   );
06   assign out = a & b;
07 endmodule
```

Having a code, the associated testbench can be created. Testbench 1 contains just the signal declarations:

Step 1: signal declarations

```
01 module and_tb;
02   reg a, b;
03   wire out;
04
05  and_gate U0 (
06  .a      (a),
07  .b      (b),
08  .out    (out)
09  );
10 endmodule
```

Step 2: Adding the simulation data

```
01  module and_tb;
02   reg a, b;
03   wire out;
04
05   and_gate U0 (
06   .a   (a),
07   .b      (b),
08   .out    (out)
09   );
10
11   initial begin
```

```
12      $dumpfile ("and_gate.vcd");
13      $dumpvars;
14      a = 0;
15      b = 0;
16      #10
17      a = 0;
18      b = 1;
19      #10
20      a = 1;
21      b = 0;
22      #10
23      a = 1;
24      b = 1;
25      #10
26    a = 0;
27    b = 0;
28      $finish;
29    end
30  endmodule
```

This form of a testbench can be executed. Testbench result:

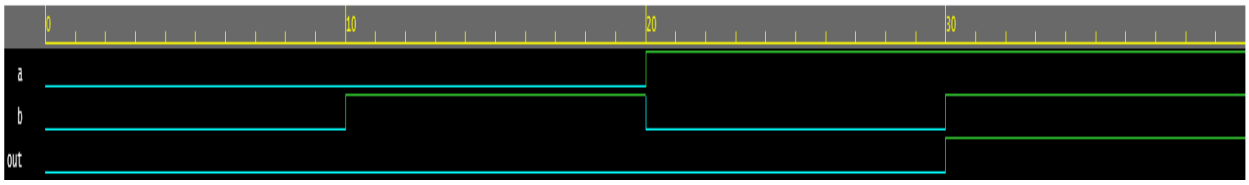

Fig. 2. Testbench result waveform

The output of the testbench can be also done in a form of value series. Consider the following code (lines 11-14 added, comparing to the previous code):

```
01  module and_tb;
02   reg a, b;
03   wire out;
04
05   and_gate U0 (
06   .a   (a),
07   .b       (b),
08   .out     (out)
09   );
10
11   initial  begin
12     $display("\ttime\ta,\tb,\tout");
13     $monitor("%d,\t%b,\t%b,\t%b",$time, a,b,out);
14   end
15
16   initial begin
17     $dumpfile ("and_gate.vcd");
18     $dumpvars;
```

```
19      a = 0;
20      b = 0;
21      #10
22      a = 0;
23      b = 1;
24      #10
25      a = 1;
26      b = 0;
27      #10
28      a = 1;
29      b = 1;
30      #10
31    a = 0;
32    b = 0;
33      $finish;
34    end
35 endmodule
```

This code will yield both *and_gate.vcd* file (used for waveforms) and the series of simulation values:

```
Compiler version J-2014.12-SP1-1; Runtime version J-2014.12-SP1-1;
time, a,  b,  out
0,    0,  0,  0
10,   0,  1,  0
20,   1,  0,  0
30,   1,  1,  1
40,   0,  0,  0
$finish called from file "testbench.sv", line 33.
$finish at simulation time 50
```

An initial construction is used for a one-pass activity flow. The activity flow for an initial construction is a sequence of procedural statements which begin activity at the start of the simulation and do not repeat. An always construction is used for cyclic behavior. The activity flow starts at the beginning of the simulation and repeats until the simulation ends. Data elements which are updated in procedural blocks must be defined as variable type data.

An example of an initial statement used to initialize three variables

```
reg a, b, c;

initial
begin
   a = 1'b0;
   b = 1'b1;
   c = 1'b0;
end
```

This initializes **a**, **b**, and **c** at the beginning of the simulation.

An example of an always statement used to create a clock signal,

```
reg clock;
always
  #5 clock = ~clock;
```

This always statement will execute the single statement used to update the variable clock repeatedly. The 5 unit time delay will cause it to wait 5 time units each time it updates clock with its complement. The effect is to toggle the value of clock every 5 time units which creates a 10 time unit period for clock.

Verilog Case Statement:

The case statement compares an expression to a series of cases and executes the statement or statement group associated with the first matching case:

- Case statement supports single or multiple statements.
- Group multiple statements using begin and end keywords.

Syntax of a case statement look as shown below.

case ()

< case1 > : < statement >

< case2 > : < statement >

.....

default : < statement >

endcase

Example: 2x1 Mux

```
module mux21(y,sel,a,b);
input sel,a,b;
output y;
reg y;
```

```
always @(*)
case (sel)
1'b0: y = a;
1'b1: y = b;
endcase
endmodule
```

Note: Other control statements like if, if..else are left for the students to explore.

LAB DELIVERIES

PRELAB

1. 3-bit ripple carry adder

Design a 3-bit ripple carry adder.

2. Decoder in Verilog

Write the decoder in Verilog using the truth table below. Simulate in Quartus. Include the simulation results in your prelab.

a	b	y_1	y_2	y_3	y_4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

3. Magnitude Comparator

This comparator compares two 1-bit numbers for magnitude. Three outputs: a>b, a=b, a<b.

- Design the comparator and implement in Verilog.
- Test your design in Quartus for correctness.

4. Complete the Altera Modelsim tutorial:

https://eelabs.faculty.unlv.edu/docs/guides/Altera_ModelSim_tutorial.pdf

PRELAB DELIVERIES:

1. Block diagram of a RCA.
2. Verilog code and waveform from Quartus.
3. Verilog code and waveform from Quartus.
4. Screenshots from the tutorials.

LAB EXPERIMENTS

1. **7-segment in Verilog**
   Write a Verilog code to use a 7-segment display on the DE2 board. Input should be given from the switch and respective result should be displayed on the 7-segment display.

2. **Magnitude Comparator from prelab 3.**

This comparator compares two 1-bit numbers for magnitude. Three outputs: a>b, a=b, a<b.

- Write the testbench.
- Simulate in Modelsim and obtain waveforms.
- Upload to DE2 and test the design onboard.

3. **Ripple adder**

Design the 3-bit ripple adder.

- Implement your design from prelab in Verilog.
- Write the testbench.
- Simulate in Modelsim and obtain waveforms.
- Upload to DE2 and test the design onboard. (result of the addition should be displayed on the 7 segment display)

4. **Simple ALU**

This 4 bit alu has following properties:

```
Input A:      4-bit
Input B:      4-bit
Input Cin:    1-bit
Output O:     4-bit
Output Cout:  1-bit
Input Control: 3-bit
```

Control	Opcode	Operation
000	AND	A and B
001	OR	A or B
010	ROR	Rotate Right, unary operation
011	ADD	A+B+Cin ; Cout = Carry
100	SUB	A-B-Cin ; Cout = Borrow
101	ROL	Rotate Left, unary operation
110	SHR	Shift Right, unary operation
111	SHL	Shift Left, unary operation

- Design and implement this ALU in Verilog
- Write a testbench
- Simulate in Modelsim and obtain waveforms
- Upload to DE2 and test the design onboard

POSTLAB REPORT:

Include the following elements in your postlab report:

<table>
<tr><th>Section</th><th>Element</th></tr>
<tr><td>1</td><td>Theory of operation
Include a brief description of every element and phenomenon that appears during the experiments.
a. What is a megafuntion in Quartus? List the applications of megafunctions.
b. What is continuous assignment and procedural assignment in Verilog?</td></tr>
<tr><td>2</td><td>Prelab report</td></tr>
<tr><td rowspan="5">3</td><td>Results of the experiments</td></tr>
<tr><td><table>
<tr><th>Experiment</th><th>Experiment Results</th></tr>
<tr><td>1</td><td>c. Verilog code
d. Pictures of DE2</td></tr>
<tr><td>2</td><td>a. Testbench code
b. Waveforms from ModelSim
c. Picture of DE2</td></tr>
<tr><td>3</td><td>a. Verilog Code
b. Testbench code
c. Waveforms from ModelSim
d. Picture of DE2</td></tr>
<tr><td>4</td><td>a. Verilog code
b. Testbench code
c. Waveforms from ModelSim
d. Pictures of DE2</td></tr>
</table></td></tr>
<tr><td rowspan="4">4</td><td>Answer the questions</td></tr>
<tr><td><table>
<tr><th>Question no.</th><th>Question</th></tr>
<tr><td>1</td><td>Why ModelSim is better than Quartus internal simulator? (elaborate)</td></tr>
<tr><td>2</td><td>How does initial block differ from always block?</td></tr>
<tr><td>3</td><td>What is the difference between $<i>finish</i> and <b>$stop in Verilog testbench?</td></tr>
</table></td></tr>
<tr><td>5</td><td>Conclusions
Write down your conclusions, things learned, problems encountered during the lab and how they were solved, etc.</td></tr>
</table>