Richard Ojo

Lab 2 Report

LAB2 (a) Broken Adder Testing

```
module myAdder(A, B, CI, SUM, CO);
input CI;
input [7:0] A;
input [7:0] B;
output [7:0] SUM;
output CO;
wire [8:0] tmp;
  assign tmp = A + B + CI;
  assign SUM = tmp [7:0];
  assign CO = tmp [8];
endmodule
```

1) Broken Adder Exhaustive Testing

Testbench Code

```
module lab2atestbench();
reg [7:0] A, B;// Two Adder 8-bit inputs
wire [7:0] S,S_expected; //Adder Sum
wire Co, Co_expected;

reg [31:0] errors;

// instantiate device under test
ripple_adder dut (.X(A), .Y(B), .S(S),.Co(Co) );
myAdder expected (.A(A), .B(B), .CI(1'b0), .SUM(S_expected), .CO(Co_expected) );

integer i;//iterate for input A
integer j;//iterate for input B

initial begin
	errors = 0; // Initialize
	A = 8'h00; B = 8'h00; #10;

	for (i = 0; i <= 255; i = i + 1) begin
		for (j = 0; j <= 255; j = j + 1) begin
//		$display("Expected =  %d CarryOut = %d", S_expected, Co_expected);
//		$display("RippleAdder Sum =  %d CarryOut = %d", S, Co);

			if(S !== S_expected) begin
				$display("Error at time %t: A = %h B = %h got %h, expected %h", $time, A, B,
S, S_expected);
				errors = errors + 4'b0001; #1;
			end
```

```
                B = B + 4'b0001; #10; //Push next input to adder
            end

            A = A + 4'b0001; #10; //Push next input to adder

    end

    $display("Tests completed with %d errors", errors);
    $finish; // End simulation
end

endmodule
```

Display Output

```
# Error at time                669493: A = fe B = 21 got df, expected 1f
# Error at time                670124: A = fe B = 60 got 1e, expected 5e
# Error at time                670135: A = fe B = 61 got 1f, expected 5f
# Error at time                670766: A = fe B = a0 got 5e, expected 9e
# Error at time                670777: A = fe B = a1 got 5f, expected 9f
# Error at time                671408: A = fe B = e0 got 9e, expected de
# Error at time                671419: A = fe B = e1 got 9f, expected df
# Error at time                672060: A = ff B = 20 got df, expected 1f
# Error at time                672701: A = ff B = 60 got 1f, expected 5f
# Error at time                673342: A = ff B = a0 got 5f, expected 9f
# Error at time                673983: A = ff B = e0 got 9f, expected df
# Tests completed with      16384 errors
# ** Note: $finish    : C:/Modeltech_pe_edu_10.4a/examples/lab2atestbench(42)
#    Time: 674314 ns  Iteration: 0  Instance: /lab2atestbench
# 1
# Break in Module lab2atestbench at C:/Modeltech_pe_edu_10.4a/examples/lab2atestbench line 42

VSIM 37>
```

2) Broken Adder Randomized Testing

Test Bench Code

```
module lab2arandomtestbench();
reg [7:0] A, B;// Two Adder 8-bit inputs
wire [7:0] S,S_expected; //Adder Sum
wire Co, Co_expected;
reg [31:0] errors;

// instantiate device under test
ripple_adder dut (.X(A), .Y(B), .S(S),.Co(Co) );
myAdder expected (.A(A), .B(B), .CI(1'b0), .SUM(S_expected), .CO(Co_expected) );

integer seed;//8bit seed

initial begin
    // Initialize
    A = 8'h00; B = 8'h00; #10;
    seed = 8;
```

```
        errors = 0;

        forever begin
                A = $random(seed); B = $random(seed); #10;

                $display("Expected =  %d CarryOut = %d", S_expected, Co_expected);
                $display("RippleAdder Sum =  %d CarryOut = %d", S, Co);

                if(S !== S_expected) begin
                        $display("Error at time %t: A = %h B = %h got %h, expected %h", $time, A, B, S, S_expected);
    #1;
                        $stop;
                        $finish; // End simulation
                end
        end
    end
    endmodule
```

Display Output

```
VSIM 22> run -all
# Expected =  133 CarryOut = 1
# RippleAdder Sum =  133 CarryOut = 1
# Expected =  238 CarryOut = 0
# RippleAdder Sum =  238 CarryOut = 0
# Expected =   18 CarryOut = 1
# RippleAdder Sum =   18 CarryOut = 1
# Expected =  251 CarryOut = 0
# RippleAdder Sum =   59 CarryOut = 1
# Error at time                   71: A = 9e B = 5d got 3b, expected fb
# ** Note: $finish    : C:/Modeltech_pe_edu_10.4a/examples/lab2arandomtestbench.v(25)
#    Time: 72 ns  Iteration: 0  Instance: /lab2arandomtestbench
# 1
# Break in Module lab2arandomtestbench at C:/Modeltech_pe_edu_10.4a/examples/lab2arandomtestbench.v line 25
```

LAB2 (b) - Broken FSM Testing

Test Bench Code

```
module lab2btestbench();
reg Clock, Reset, In;
wire Out;
wire [2:0] State;

reg [7:0] TestValues [0:23];//Test values from file

fsmtemp dut(.Clock(Clock), .Reset(Reset), .In(In), .Out(Out), .State(State) );

integer i, j;

//generate clock
initial Clock = 0;
always begin
	Clock = ~Clock; #5; // 10ns period
end

initial begin
	$readmemb("TestValues.input", TestValues); // Read vectors into TestValues, Note: $readmemh reads
testvector files written in hexadecimal
	Reset = 1; #10; Reset = 0; #10; // Apply reset
	Reset = 1; #10; Reset = 0; #10; // Apply reset
	Reset = 1;
	$display("Initial FSM state: %h", State);//Observe initial State

	//Outer for loop iterates through each 8-bit chunk
	for (i = 0; i < 23; i = i + 1) begin
		$display("TestValues[%d] = %h or %b", i, TestValues[i], TestValues[i] );

		//Inner for loop iterates through each bit
		for (j = 0; j < 8; j = j + 1) begin
			In = TestValues[i][j]; #10;//Push input bit to fsm
	end

		$display("FSM state: %h Output: %h", State, Out);//Observe State
		Reset = 0; #10; Reset = 1; #10; // Apply reset wait
```

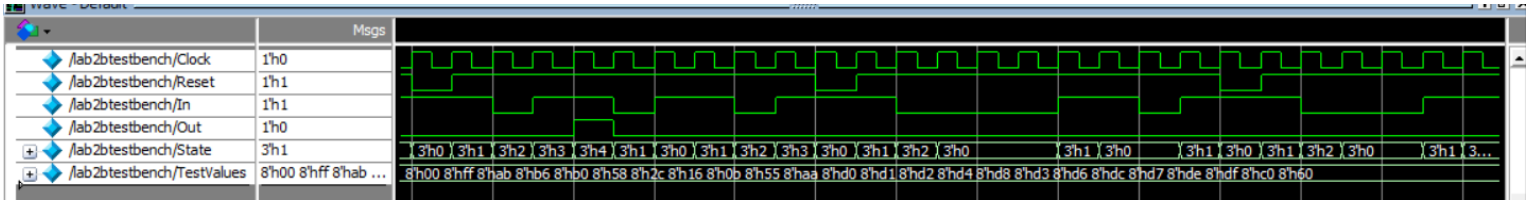
```
        end

        $finish;//The end of test
end
endmodule
```

Display

Passing input patterns will have an FSM state 4 and Output: 1*
Failing input patterns will reset to State 0*

```
VSIM 4> run -all
# ** Warning: (vsim-PLI-3407) Too many data words read c
#    Time: 0 ns  Iteration: 0  Instance: /lab2btestbench
# Initial FSM state: 0
# TestValues[          0] = 00 or 00000000
# FSM state: 0 Output: 0
# TestValues[          1] = ff or 11111111
# FSM state: 0 Output: 0
# TestValues[          2] = ab or 10101011
# FSM state: 3 Output: 0
# TestValues[          3] = b6 or 10110110
# FSM state: 3 Output: 0
# TestValues[          4] = b0 or 10110000
# FSM state: 1 Output: 0
# TestValues[          5] = 58 or 01011000
# FSM state: 2 Output: 0
# TestValues[          6] = 2c or 00101100
# FSM state: 0 Output: 0
# TestValues[          7] = 16 or 00010110
# FSM state: 0 Output: 0
# TestValues[          8] = 0b or 00001011
# FSM state: 0 Output: 0
# TestValues[          9] = 55 or 01010101
# FSM state: 2 Output: 0
# TestValues[         10] = aa or 10101010
# FSM state: 3 Output: 0
# TestValues[         11] = d0 or 11010000
# FSM state: 4 Output: 1
# TestValues[         12] = d1 or 11010001
# FSM state: 4 Output: 1
# TestValues[         13] = d2 or 11010010
# FSM state: 4 Output: 1
# TestValues[         14] = d4 or 11010100
# FSM state: 4 Output: 1
# TestValues[         15] = d8 or 11011000
# FSM state: 0 Output: 0
# TestValues[         16] = d3 or 11010011
# FSM state: 4 Output: 1
# TestValues[         17] = d6 or 11010110
# FSM state: 0 Output: 0
# TestValues[         18] = dc or 11011100
# FSM state: 4 Output: 1
# TestValues[         19] = d7 or 11010111
# FSM state: 4 Output: 1
# TestValues[         20] = de or 11011110
# FSM state: 0 Output: 0
# TestValues[         21] = df or 11011111
# FSM state: 0 Output: 0
# TestValues[         22] = c0 or 11000000
# FSM state: 0 Output: 0
# ** Note: $finish    : C:/Modeltech_pe_edu_10.4a/examples/lab2btestbench.v(38)
#    Time: 2340 ns  Iteration: 0  Instance: /lab2btestbench
# 1
# Break in Module lab2btestbench at C:/Modeltech_pe_edu_10.4a/examples/lab2btestbench.v line 38
VSIM 5> quit -sim
```

Waveform
Passing input patterns will have a State: 3'h4 and Out: 1*
Failing input patterns reset to State: 3'h0 and Out: 0*



It appears that passing input patterns begin with 1101.