Lab 6 (All Sections) Prelab: Introduction to Verilog

Name:

Yerania Hernandez

Sign the following statement:

On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work

Yerania Hernandez

1 Objective

The main objective of this lab is to give hands on experience with Verilog HDL and Synopsys VCS. For this lab you are expected to know some basic Verilog programming and understand the basics of Synopsys VCS.

2 Introduction

The Verilog HDL is one of the two most common Hardware Description Languages (HDL) used by integrated circuit (IC) designers (the other is VHDL). HDLs allow the design to be simulated earlier in the design cycle in order to correct errors or experiment with different architectures. Designs described in HDL are technology-independent, easy to debug, and are usually more readable than schematics, particularly for large circuits.

Verilog can be used to describe designs at four levels of abstraction:

1. Algorithmic level (much like C code with if-else, while, case and loop statements).
2. Register transfer level (RTL uses registers connected by Boolean equations).
3. Gate level (interconnected AND, NOR etc.).
4. Switch level (the switches are MOS transistors inside gates).

The language also defines constructs that can be used to control the input and output of simulation. Verilog is also used as an input for logic synthesis programs which will generate a gate-level description (a netlist) for the circuit. The netlist can then be loaded on to a chip called FPGA (Field Programmable Gate Arrays), which is a generic chip on which any functionality can be programmed and re-programmed when required.

3 Questions

1. What is the difference between Structural and Behavioral programming style? Give an example for each style.

   Structural programming describes the structure of hardware components, including how ports of modules are connected together. Behavioral programming describes what should be done in a module, similar to C-like assignment statements.
   An example of structural programming is the following since it uses wires and built-in gates to describe the hardware piece of an inverter:

```
module gate(Y,A);
  input A;
```

2. **Half Adder**
   Design a half adder comprised of just $NAND$ gates (points will be deducted for using an XOR. Give the Truth Table for the circuit. Write a structural Verilog program for the half adder.

   Truth Table for Half Adder:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
module HalfAdd (Cout, Sum, A, B);
input A, B;
output Cout, Sum;
wire W1, W2, W3;

nand (W1, A, B);
nand (W2, A, W1);
nand (W3, W1, B);
nand (Sum, W2, W3);
nand (Cout, W1, W1);

endmodule
```

3. Develop a testbench for the Half Adder that verifies the structural model. The testbench will have no ports. Exhaustively simulate the circuit and print the output demonstrating that the model is correct. Text output can be generated using the `$monitor` and `$display` tasks.

```
module HalfAddTest_v;

  task passTest;
    input [1:0] actualOut, expectedOut;
    input [`STRLEN*8:0] testType;
    inout [3:0]         passed;

    if(actualOut == expectedOut) begin $display ("%s passed", testType); passed = passed + 1;
end
    else $display ("%s failed: %d should be %d", testType, actualOut, expectedOut);
  endtask

  task allPassed;
    input [3:0] passed;
    input [3:0] numTests;

    if(passed == numTests) $display ("All tests passed");
    else $display("Some tests failed");
  endtask

  // Inputs
  reg           A;
  reg           B;
  reg [3:0]      passed;

  // Outputs
  wire          Cout;
  wire          Sum;

  // Instantiate the Device Under Test (DUT)
  HalfAdd dut (
 .Cout(Cout),
 .Sum(Sum),
 .A(A),
```

4. **Multiplexor**
   Design a 2-to-1 MUX Circuit, that multiplexes two input signals to a single output based on the select signal.

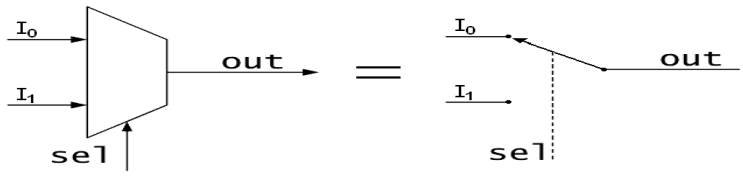


Fig. 1: Multiplexor

(a) Give the truth table for the circuit:

I0	I1	Sel	Out
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

(b) Design the Optimal Circuit design using the basic gates (not to be submitted).

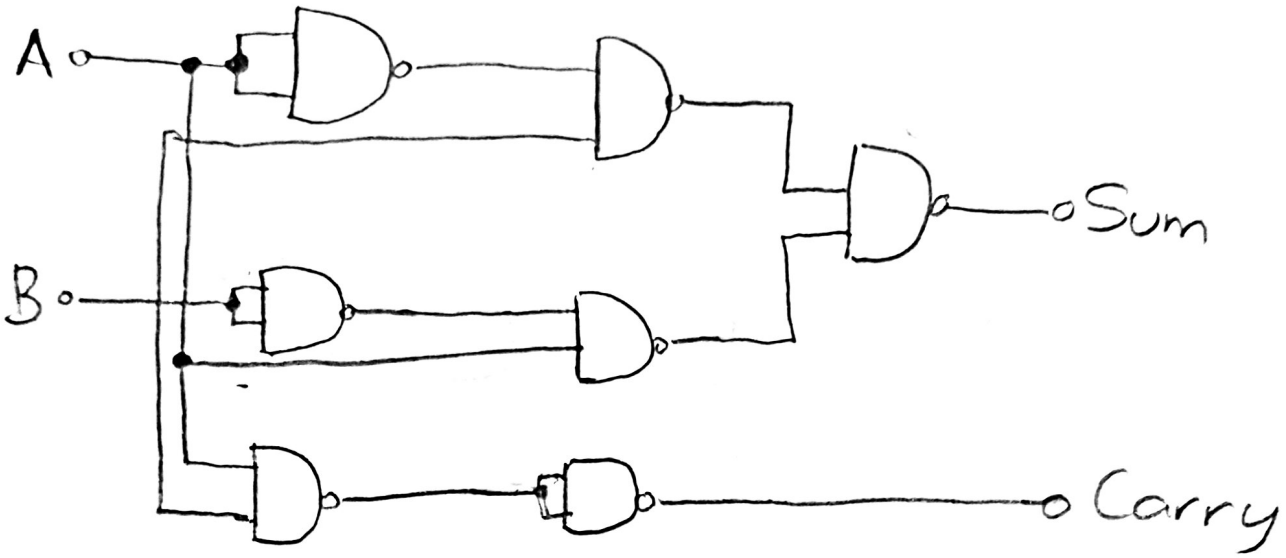
(c) Write a structural Verilog model for the 2-to-1 MUX:

```
module Mux21(out,, in, sel);
input [1:0] in;
input sel;
output out;
wire andA, andB;

and (andA, in[0], ~sel);
and (andB, in[1], sel);
or (out, andA, andB);

endmodule
```

(d) Develop a testbench for 2-to-1 MUX that verifies the structural model.

```
module Mux21Test_v;

task passTest;
 input actualOut, expectedOut;
 input [`STRLEN*8:0] testType;
 inout [7:0] passed;

 if(actualOut == expectedOut) begin $display ("%s passed", testType); passed = passed + 1; end
 else $display ("%s failed: %d should be %d", testType, actualOut, expectedOut);
endtask

task allPassed;
 input [7:0] passed;
 input [7:0] numTests;

 if(passed == numTests) $display ("All tests passed");
 else $display("Some tests failed");
endtask

task stim;
 input [1:0] newIn;
 input newSel;
 output [1:0] setIn;
 output setSel;

 begin
  #90;
  setIn = newIn;
  setSel = newSel;
 end
endtask

// Inputs
reg [1:0] in;
```

Half Adder



2 to 1 MUX (Optimal)