CS 355 PA5 – Verilog Simulation Part 2

1 Background Information and Notes

Simulation File Format: Once you have read in and parsed a Verilog netlist, we can simulate the logic design by providing input values and calculating output values based on the gate network. The user must specify the desired input combinations. The user will do so by providing an input simulation file (2nd command line argument). For this purpose let us define the following format:

Format	Example
PI1 PI2 ... PIn PI1_val PI2_val ... PIn_val ... PI1_val PI2_val ... PIn_val	a b c d 0 0 0 0 1 1 0 1 1 1 1 1

The first line of the simulation file should be a list of the PI's whose values will be provided by the user. This list of PI's can be in any order (doesn't have to match the order of the Verilog module port list) but will define the order that the input values will be specified in the lines below. The PI's and input values on a line should be separated by whitespace.

Following the first line should be any number of lines containing 1's and 0's (as many as PI's specified). The i-th value specified on the line should be interpreted as the value of the i-th PI specified on the first line. Essentially, each line represents a different input value to simulate and compute the PO values.

Your simulation should produce an output file (3rd command line argument). The format of the output file should be similar to that of the input file, but with the characters '=>' following the list of PI's (in the same order as they were read) and indicating the name of the PO's that were simulated. Then each following line should list the corresponding input vector followed by the '=>' characters and then the calculated output values followed by a '@' and the time delay of that output. For this assignment, you can assume ***each*** simulation vector starts at time 0, so the delay of each output will be the same for each row in the file. In a real simulator, time would increase at a certain stepsize for each input vector and if inputs didn't change from one vector to the next, certain paths may not incur any switching delay yielding differing delay numbers for the same PO. But again, we will simplify it to have each vector start at time 0 and re-evaluate the entire delay paths for each vector.

test.out

```
PI1 PI2 ... Pin => PO1 PO2 ... POm
PI1_val PI2_val ... PIn_val => PO1_val@t1 PO2_val@t2 ... POm_val@tm
...
PI1_val PI2_val ... PIn_val => PO1_val@t1 PO2_val@t2 ... POm_val@tm
```

Topological Sort: To efficiently compute the PO values, we need to find an ordering of nets/gates such that all the input values of driver gate(s) are computed and available to compute the load net's value. Obviously, all PI nets will have values provided by the input simulation file and are first in this ordering. Next, gates that depend only on PI's should be computed, then gates that depend on PI's and previously computed gate values, etc. This leads us to a topological ordering of the gates based on their net connections. For more information about topological sorting/ordering and related algorithms refer to the class lecture notes or other online resources.

At first glance, it would appear that we want to perform a topological sort of the gates in the network and then compute values in that order. However, we realize even from the simulation inputs and outputs that it is NOT **gates** that have values, but **nets** that have values. PI nets have input values provided from the simulation file and then we compute the value of PO nets. **Thus we want to perform a topological ordering of the net's in the design based on their dependence on other nets**. Also, because it is possible (though not desirable) to have a gate network where one net has multiple drivers, it will be easier to determine a net's value by ensuring that all driver gates can be evaluated at once (i.e. if we compute the value from a gate's perspective rather than the net's perspective the code would likely be more complicated because a gate's output may not be the 'final' value for that net. If the current gate outputs one value and another driver gate produces a different value, we would have to switch the net's value to an 'X'.

The example below shows a gate network and a graph from the perspective of net's and their dependence. Note that you do not need to create this net dependency graph structure. You already have it from the drivers/loads lists of each net and the inputs/outputs of each gate (i.e. **the dependencies of a net are all the net inputs of all the gate drivers of that net.**)

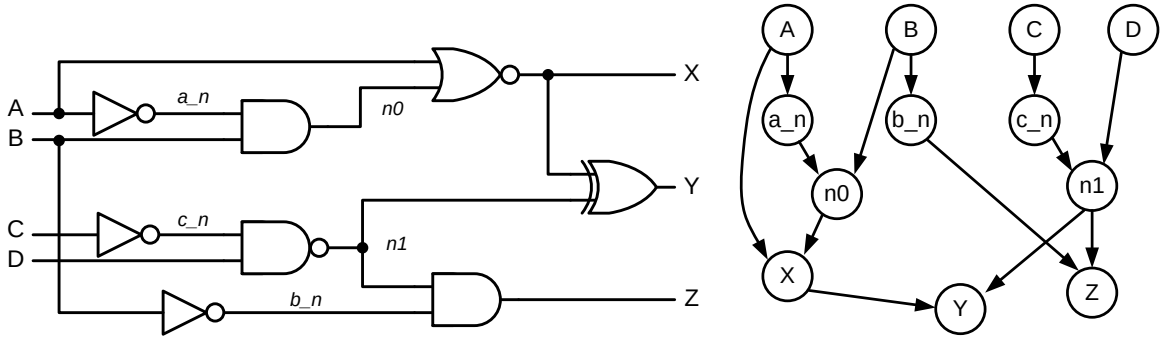

Figure 1 - Logic Gate Network and Corresponding Net Dependency Graph

Delay computation. The topological ordering also provides an opportunity to calculate delay. As a net, *n*, is entered into the topological sort order, we can compute its delay as:

$$delay(N) = MAX_{G\ =GATE\ DRIVERS\ of\ N}(MAX_{I=INPUT\ NETS\ of\ G}(delay(I)) \quad \textbf{[EQN 1]}$$

You can initialize the delay of each PI Net to 0 before the topological ordering.

Computing Gate Values: While the definition of the common logic gates And, Or, etc. are well known, dealing with X's adds some complexity. Based on their CMOS (transistor-level) implementation we can define their functionality using the descriptions below. Also, note that we do allow for the degenerative case of an AND, NAND, OR, NOR, or XOR gate having only a **single input**. In this case, the gate should pass or invert (if it is a NAND or NOR) whatever the single input value is (provided it is not 'X'…if it is an 'X' input, an 'X' output should be produced.) **Again remember if a net does have multiple drivers with differing output values, the net's value should be 'X'.**

AND	X	0	1
X	X	0	X
0	0	0	0
1	X	0	1

NAND	X	0	1
X	X	1	X
0	1	1	1
1	X	1	0

XOR	X	0	1
X	X	X	X
0	X	0	1
1	X	1	0

OR	X	0	1
X	X	X	1
0	X	0	1
1	1	1	1

NOR	X	0	1
X	X	X	0
0	X	1	0
1	0	0	0

NOT	X	0	1
X	X	X	X
0	X	1	X
1	X	X	0

Figure 2 - Gate Function Tables. As we iterate through input values, the top row indicates 'computed' value from inputs examined already and vertical left column indicates next input value. Corresponding table entries indicate the new, updated 'computed' value.

2 Requirements

Your program shall meet the following requirements for features and approach. You are free to add new classes, modify current ones with new methods or data members, etc.

Part 2

1) Your part 2 program should run with the following command line arguments
   ```
   > ./gatesim verilog_file input_sim_file output_sim_file
   ```
2) Implement the eval() virtual functions of the derived Gate classes according to the gate functions specified earlier in this document.
3) Implement the computeVal() function of the Net class. Remember if a net has multiple drivers whose values differ, the net should have a value of 'X'.
4) Implement a computeDelay() function of the Net class as described in EQN 1.
5) Implement a topological sort algorithm in the Design class to create a topological ordering of nets within a design.
6) Create a class called LogicSim that encapsulates the following functionality.
7) Read and parse the input simulation file. Check for the following errors and if found, display an informative message and line number, then exit.
   a) A specified PI on the first line does not exist in the design
   b) The number of input values for a vector does not match the number of PI's specified on the first line
   c) An input value is not in the set {'0', '1', 'X'}

8) For each input test vector from the simulation file, simulate the gate network to find the corresponding output values by calling computeVal() and computeDelay() for each net in topological order. Ensure that a net value from a previous input vector simulation doesn't affect the current vector's computations.
9) Create the output simulation file according to the format specified earlier in this document (see output example below).

Sample mux21.out

```
s a b => y
0 0 0 => 0@4
0 0 1 => 0@4
0 1 0 => 1@4
0 1 1 => 1@4
1 0 0 => 0@4
1 0 1 => 1@4
1 1 0 => 0@4
1 1 1 => 1@4
```

3 Procedure

Perform the following.

1. Complete part 2 of this program to meeting the requirements outline in the previous section.
2. **Indicate both team members name as a comment at the top of main.cpp**
3. Submit your completed code by checking in a tagged copy of your working code to your SVN archive using the command below…BE SURE TO REPLACE "g01" with your group number.

```
svn cp –m "PA6 submission" svn://parallel05.usc.edu/cs355g01/trunk
                           svn://parallel05.usc.edu/cs355g01/tags/pa6submit
```

Teammate Names: _____

Item	Outcome	Score	Max
Input Simulation File			
• A specified PI on the first line does not exist in the design	Yes / No		2
• The number of input values for a vector does not match the number of PI's specified on the first line	Yes / No		2
• An input value is not in the set {'0', '1', 'X'}	Yes / No		2
Output Simulation File Format			
• Lists PIs, '=>', then PO's	Yes / No		2
• Lists all test input vectors in form: inputs => outputs	Yes / No		1
Topological Sort and Delay computation			
• Correctly implements recursive topological sort	Yes / No		3
• Correctly implements EQN1 for delay computation	Yes / No		3
Correct Simulations			
• mux21	____ / 3		3
• adder4	____ / 6		8
• Instructor case 1	____ / 6		6
• Instructor case 2	____ / 6		6
• Instructor case 3	____ / 6		6
Gate Eval function			
• AND	Yes / No		1
• OR	Yes / No		1
• NAND	Yes / No		1
• NOR	Yes / No		1
• XOR	Yes / No		1
• NOT	Yes / No		1
Subtotal			50
Late Deductions (-10% per day)			
Total			