# EECS 478 Assignment 3 Report

Yongjoo Park (UMID:4562 2850)
pyongjoo@umich.edu

April 23, 2013

## 1 Topological Sort

### 1.1 Implementation

I only used the nodes connections automatically constructed as the *blif* file is read by the function *readBlif* declared in *circuit.h*. The method nicely parse the file and set the fan-in nodes for each node. This corresponds to parent nodes if we represent the circuit in graph in which directed edge is drawn from the node that provides output values to the node that needs the value to compute its output value.

In order to implement the topological sort, I used the algorithm introduced in a Wikipedia page[1]. There are two algorithms introduced, and I used the second one which uses depth-first search internally to find the parent nodes in the graph representation of the given circuit, or descendent nodes in the graph in which the direction of edges are inverted so that the parent nodes depend on their child nodes. This conversion, reversing the order of the direction of the edges, is to make use of the depth-first search algorithm that is traditionally designed to search from parent nodes to their descendent nodes. Below is the pseudo code of the algorithm which is the same one introduced in [1].

My own implementation of the algorithm is the method *topologicalSort* and *topologicalVisit*. The latter method is used internally by *topologicalSort* although I did not the make the method private just for the ease of testing during the development cycle.

[1]http://en.wikipedia.org/wiki/Topological_sorting

**Algorithm 1** Topological sort algorithm based on depth-first search.

```
L is empty list that topologically sorted nodes will be stored.
if there are unmarked ndoes then
    select an unmarked node n
    visit(n)
end if
function VISIT(node n)
    if n has a temporary mark then
        stop
    end if
    if n is not marked then
        mark n temporarily
        for each node m with an edge from n to m do
            visit(m)
        end for
        mark n permanently
        add n to L
    end if
end function
```

### 1.2 Verification

I used a fairly simple circuit to verify the correct behavior of the function. You can find the test file *top_test.blif* in the archived package. Since my simulation module (next section) relies on the topological sorting too, the behavior of this module can also be verified indirectly as checking the simulation module.

## 2 Simulation

### 2.1 Implementation

I have added an instance variable *sim_value* for Node class so a simulated output value of the node can be referred later. The simulation value of each node depends on parents nodes in a graphical representation of a given circuit in which the output values of the parent nodes should be known first to correctly simulate the values of its child nodes.

First I found the order of the simulation using the topological sorting method implemented in the above section, and used the order found to simulate each node. The simulation for each node is performed by checking the *truthTable* entries of each node. (The output value is set to one, if appropriate truth table entry has been inserted.)

### 2.2 Verification

I have tested the correct behavior of my own simulator implementation with AND, OR, and 16-bit adder implementation provided in the earlier assignment. In order to generate input files for 16-bit adder simulation, I have coded a python script for which you can find *gen_tests.py* in the archived package.

## 3 References

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms.* MIT press, 2001.