CSCE 337 – Digital Design II
Project 1 – Logic Circuit Schematic Beautifier
Project Report

Mennatallah Yehia	900100570
Sara Kandil	900114861
Shaden Raouf	900111674
Sohaida Reyad	900120398

Introduction

Our Logic Circuit Schematic Beautifier is an application that accepts a gate-level netlist, parses it into a directed acyclic graph (DAG), applies Topological Sorting on the resulting DAG to get the sorting of the gates, and displays the results per level on a Qt GUI application. The netlist will be parsed using Perl resulting in a .txt file that contains the inputs, outputs, wires, and gates. This parsed file is the picked up by a Qt application so that an adjacency list of the DAG which will then be used by the Topological Sorting Kahn's algorithm so that it gets sorted.

Design & Implementation

The project is divided into three main modules:

1- Parsing the gate level netlist
2- Constructing DAG
3- Applying Topological Sort
4- Run GUI to display the results

Parsing the Gate-Level Netlist:

The parsing was done with Perl library "Perl Verilog". The library includes classes for to model modules, cells, pints, ports, etc….

The library was installed on linux and the sample code was used as a starting point. Then, using the documentation of the library available, the code was tailored to meet our specific needs.

The code parses the gate level netlist file then writes the data in a format, that we have agreed upon, in a text file.

The format we agreed on as follows:

- [number of input ports n] followed by n lines. Each line contains the name of one input port and its size (bus or wire).
- [number of output ports m] followed by m lines. Each line contains the name of one output port and its size.
- [number of inout ports k] followed by k lines. Each line contains the name of one inout port and its size.
- [number of wires w] followed by w lines, each line contains the name of one wire.
- [number of assign statements a] followed by a lines. Each line contains the left hand side and the right hand side of one assignment statement, separated by a tab.
- Then a list of all gates, each gate in one line with the following data:
  [type of gate] [name of gate] [number of inputs] then list of inputs of the gate then the output wire of the gate.

All fields are separated by tabs.

Constructing DAG:

The graphs are constructed using C++. A class called "**DAG**" has been constructed in order to translate the parsed text, which resulted from the Perl Parser, into two main elements:

1) An adjacency list which displays the connections between gates; this adjacency list is a 2D array (vector of vector). If one gate is the input of another, a '1' is saved in the cell that corresponds those two gates. For example, if the output of an AND gate is the input of an OR gate, then the cell at row OR and column AND will have a '1'.

2) Vector of gates, such that a class "**gate**" is one that stores the gate's name, number of inputs, list of inputs, list of outputs, and level (which initially is not set).

One of the main DAG class functions is readFile, which opens and parses a .txt file that contains the output of the Perl Parser developed as outlined above. This function reads the file fields and fills the vector of gates explained above.

The adjacency matrix is then filled by iterating over each gate and checking if its inputs are the outputs of other gates, which indicates a connection between them.

Apply Topological Sorting:

Kahn's Algorithm has been implemented in this project using C++. A class called "**TopoSort**" has been designed in a way such that an instance from this class accepts a pointer to a DAG object. Then the TopoSort constructor will get a list of all the gates that have no incoming edges (i.e. their inputs are not outputs of other gates). Those gates will then get level 1 (since level 0 is for inputs).

When KahnSort function is called, the algorithm is implemented to sort the gates in a vector called 'sortedGates', which stores the indexes of the gates that are sorted according to Kahn's Topological Sorting algorithm. Moreover, the level of each gate will be one more than the level of the gate it is connected to (i.e. the gate whose output is the input of the gate under inspection); this level is saved in DAG's 'gates' vector.

After sorting is done, the gates are then sorted by name and index in the DAG's 'gates' vector which will then be used to display the results in the GUI.

Run on GUI to Display Results:

GUI has been developed using Qt. The above mentioned DAG and TopoSort classes are included in the Qt project.

This GUI has been designed in a way such that the user will be able to browse for the parsed .txt file (the output of the Perl Parser) and then the program will execute the DAG and TopoSort functions. The result will be composed of 3 things:

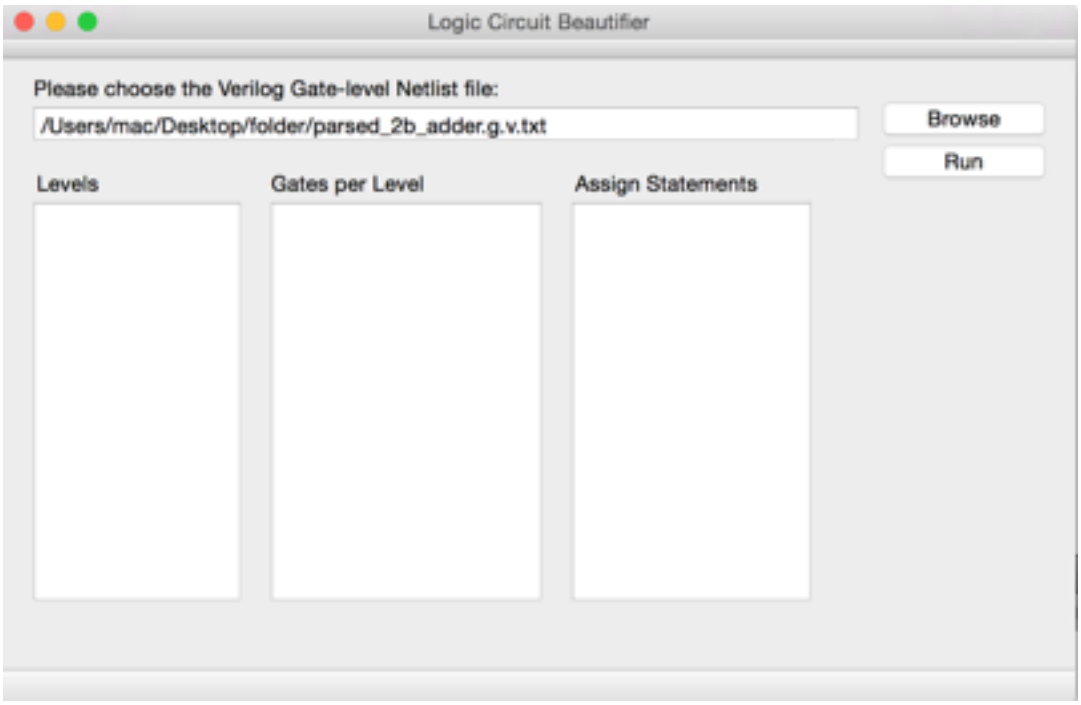
1) Levels, where 0 is the inputs level and the last level contains the outputs.
2) Gates per Level: By clicking on each level the user will get the gates in each level in a text browser to the right of the list of levels
3) Assign Statements: the list of assign pairs as indicated in the gate-level netlist.

Screenshots

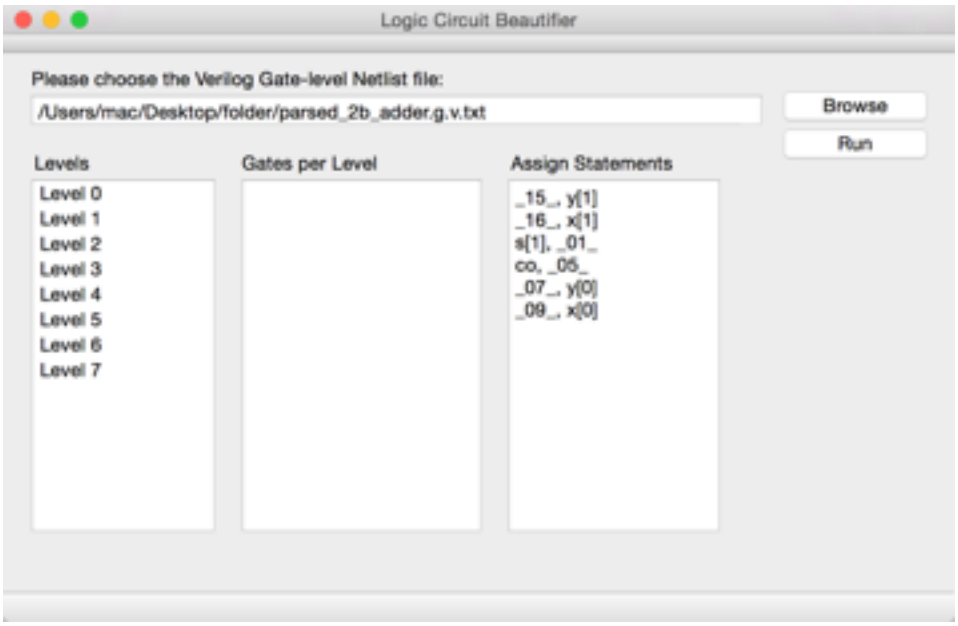
The window the opens upon running the program looks like this:

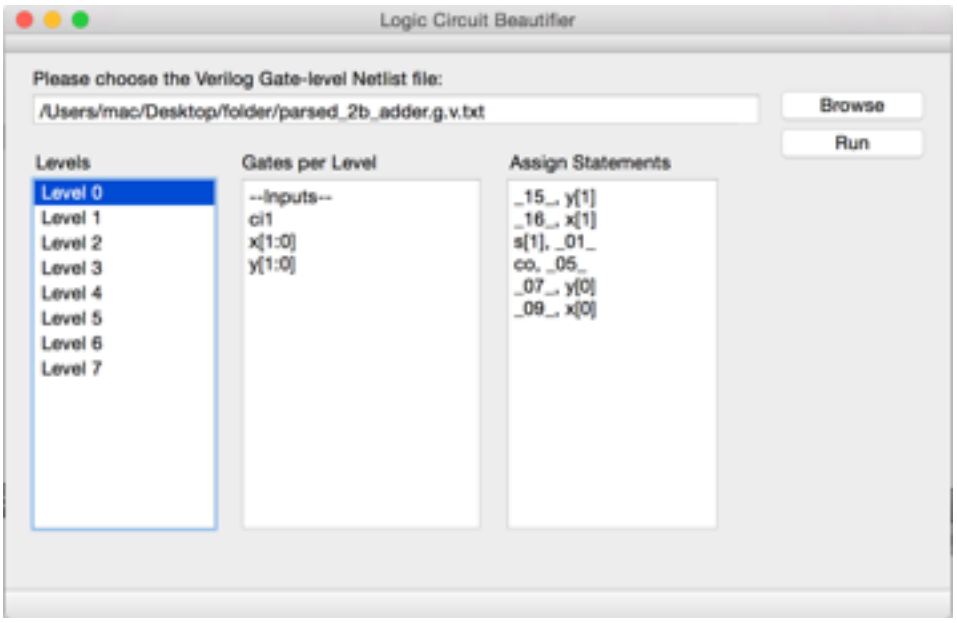
When the ‘Browse’ button is clicked, a file explorer will open to allow the user to choose the parsed .txt that the user wishes to analyze.

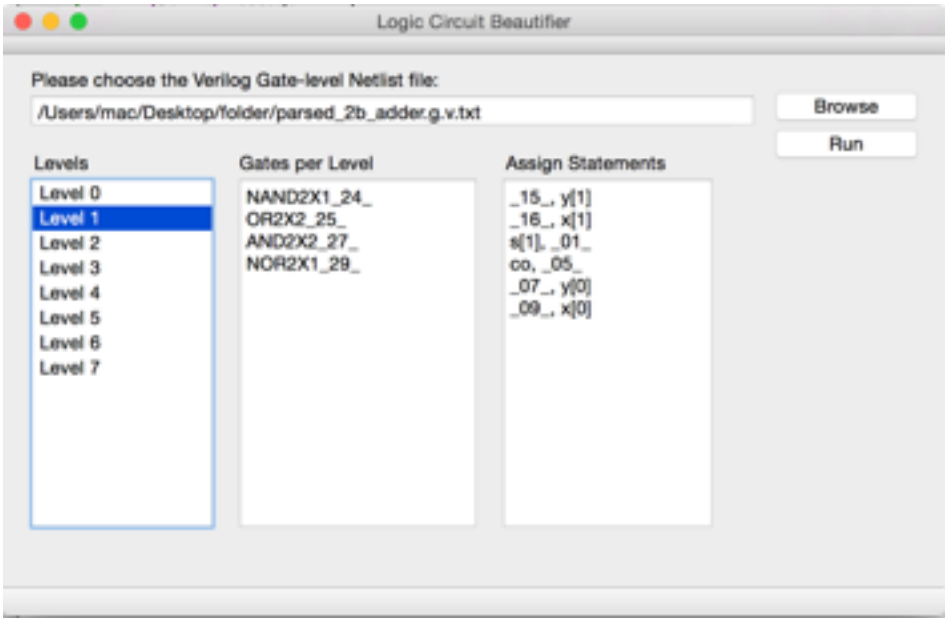
After choosing the file, the file path will appear in the text box for validation:

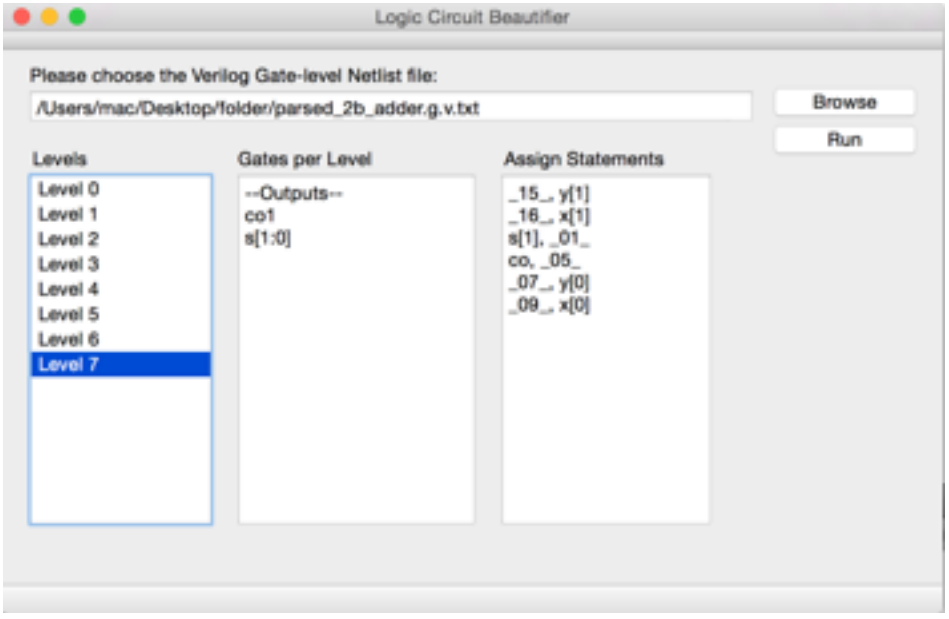


When the ‘Run’ button is pressed, the DAG and TopoSort functionalities execute. The levels will show on the left panel, and when a level is pressed, the corresponding level gates are displayed in the middle panel. The assign statements show in the right panel:




Logic Circuit Beautifier
Please choose the Verilog Gate-level Netlist file:
/Users/mac/Desktop/folder/parsed_2b_adder.g.v.txt
Browse
Run
Levels
Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
Gates per Level
--Inputs--
ci1
x[1:0]
y[1:0]
Assign Statements
_15_, y[1]
_16_, x[1]
s[1], _01_
co, _05_
_07_, y[0]
_09_, x[0]

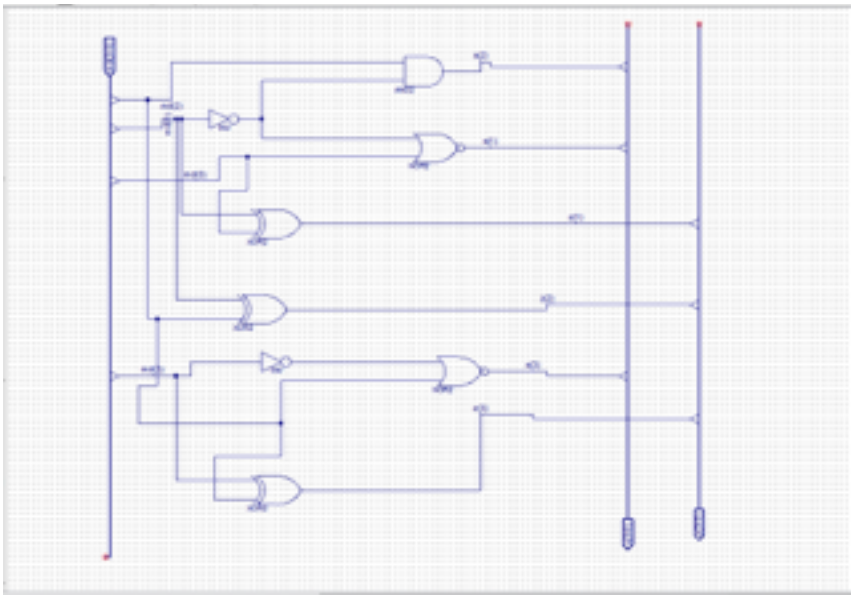

Logic Circuit Beautifier
Please choose the Verilog Gate-level Netlist file:
/Users/mac/Desktop/folder/parsed_2b_adder.g.v.txt
Browse
Run
Levels
Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
Gates per Level
NAND2X1_24_
OR2X2_25_
AND2X2_27_
NOR2X1_29_
Assign Statements
_15_, y[1]
_16_, x[1]
s[1], _01_
co, _05_
_07_, y[0]
_09_, x[0]


Logic Circuit Beautifier
Please choose the Verilog Gate-level Netlist file:
/Users/mac/Desktop/folder/parsed_2b_adder.g.v.txt
Browse
Run
Levels
Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
Gates per Level
--Outputs--
co1
s[1:0]
Assign Statements
_15_, y[1]
_16_, x[1]
s[1], _01_
co, _05_
_07_, y[0]
_09_, x[0]

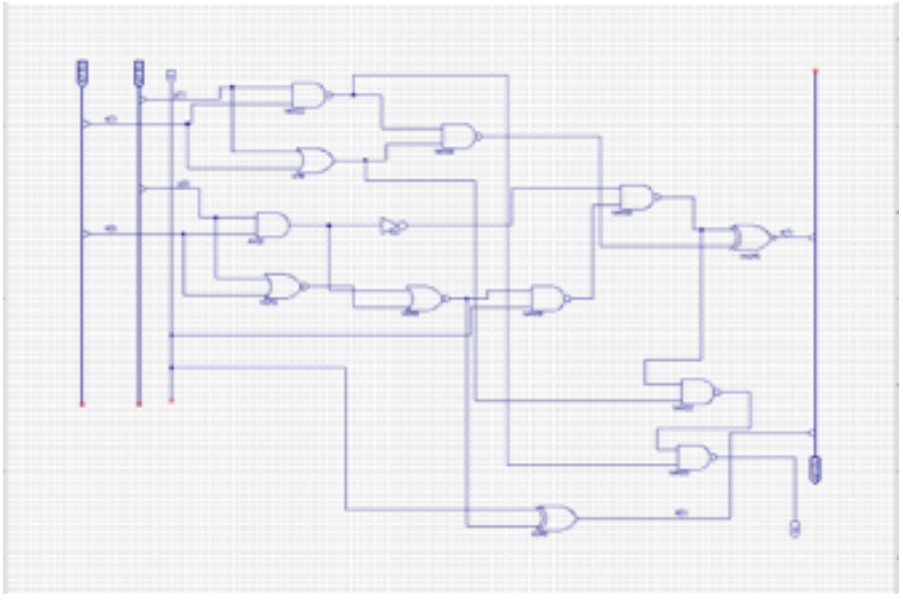
Testing

We were able to verify the output of the application through the following strategy:

1) Schematics were drawn for each circuit that was tested; the circuits drawn were that for files with names: ‘booth.v.g’ and ‘2b_adder.v.g’. The schematics are as follows, respectively:

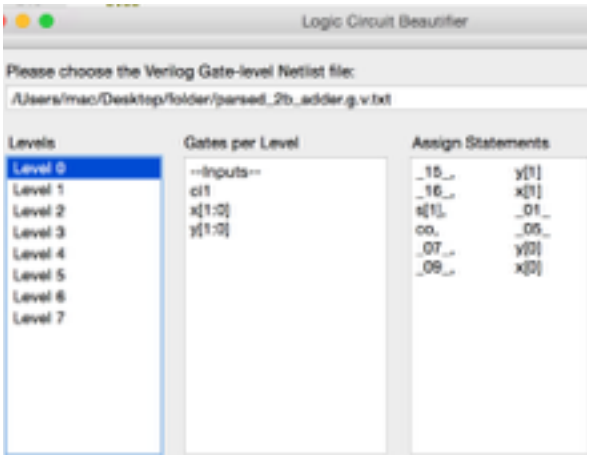


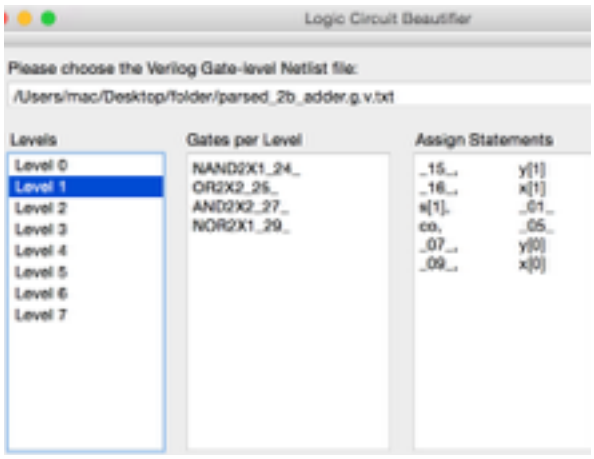
booth.v.g schematic

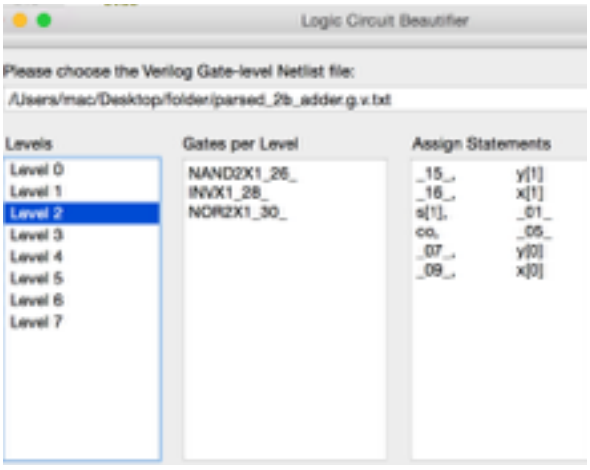


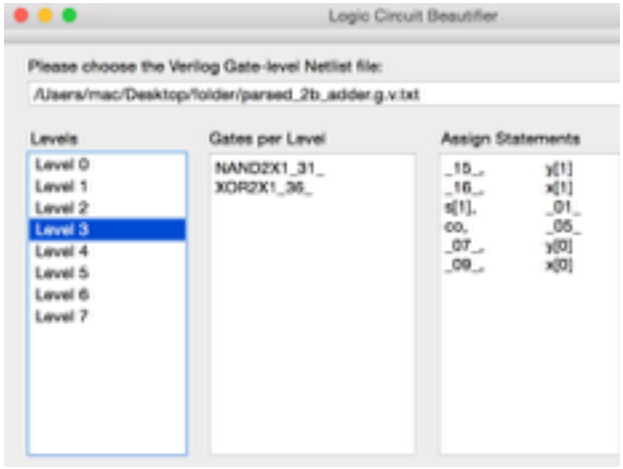
2b_adder.v.g schematic

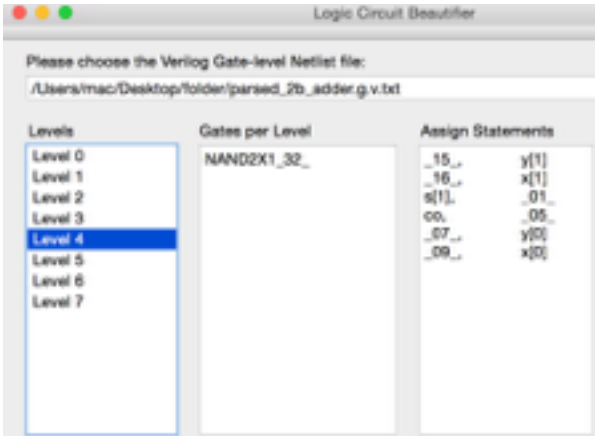
2) By looking at the following schematics, we were able to identify the levels of all gates. For example, in the 2b_adder, the output was as follows for each level (please compare with schematic above):

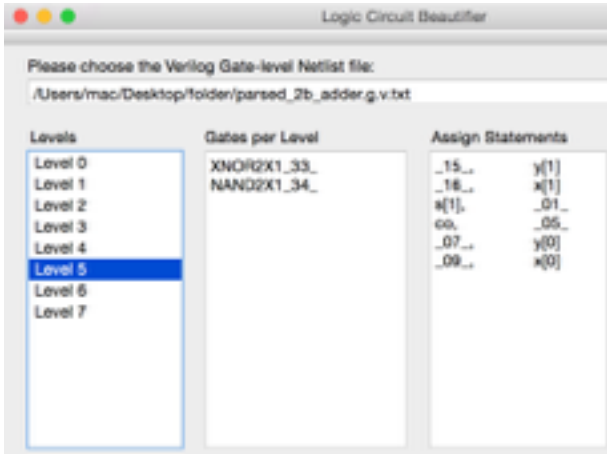


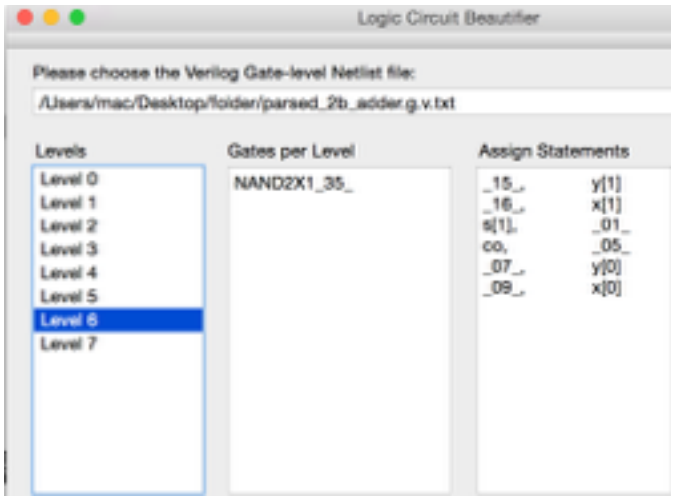


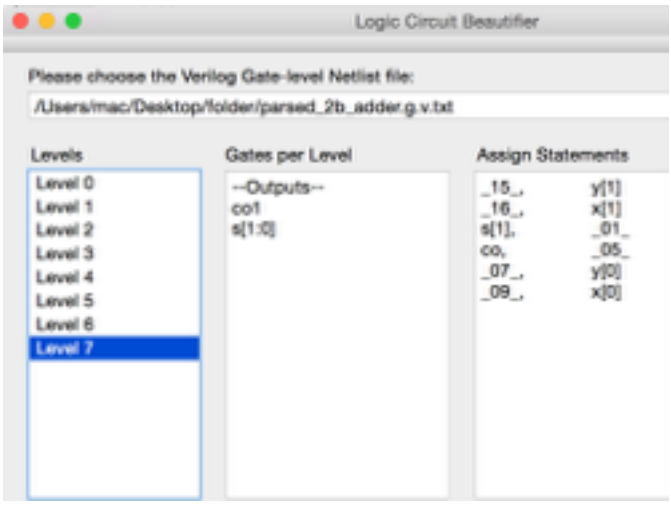









Logic Circuit Beautifier
Please choose the Verilog Gate-level Netlist file:
/Users/mac/Desktop/folder/parsed_2b_adder.g.v.txt
Levels
Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
Gates per Level
NAND2X1_35_
Assign Statements
_15_ y[1]
_16_ x[1]
s[1], _01_
co, _05_
_07_ y[0]
_09_ x[0]


Logic Circuit Beautifier
Please choose the Verilog Gate-level Netlist file:
/Users/mac/Desktop/folder/parsed_2b_adder.g.v.txt
Levels
Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
Gates per Level
--Outputs--
co1
s[1:0]
Assign Statements
_15_ y[1]
_16_ x[1]
s[1], _01_
co, _05_
_07_ y[0]
_09_ x[0]

Resources

"Topological Sorting." Wikipedia. Wikimedia Foundation, n.d. Web. 22 Mar. 2015. http://en.wikipedia.org/wiki/Topological_sorting

"Tutorials Point - Simply Easy Learning." Perl Environment Setup. N.p., n.d. Web. 22 Mar. 2015.http://www.tutorialspoint.com/perl/perl_environment.htm

"Verilog-Perl." Installing. N.p., n.d. Web. 22 Mar. 2015. http://www.veripool.org/projects/verilog-perl/wiki/Installing

"NAME." Verilog::Netlist. N.p., n.d. Web. 22 Mar. 2015.http://search.cpan.org/dist/Verilog-Perl/Netlist.pm