CSCE 3304

Verilog Beauty Salon

Project I Report

Ahmed Wael Abouseida

Mahmoud Wael Elkhodary

Aya Yasser Harb

Outline:

This application reads Verilog gate level net-list files, and outputs the levels the logical level of each gate used.

Technology Used:

- Python3
- Tkinter
- TC/TCK

Design:

The Components:

The Gate Class and the Wire Class are the two main classes used with this design and they represent the different gates present, the input, the output, and the wires respectively.

Parser:

This component does the following actions:

- Opening the file that should be parsed according to the file path given.
- It will read the file line by line, and uses regex expressions to captures the assign statements before getting or detecting the other components.
- It will read the file again, starting from the beginning, this time detecting the Gates, wires, inputs and outputs. The supported gates are only two-input gates; it is however no hard to implement more than two gates.

Sorting:

A lot of attempts were taken to tackle this requirement. Initially a ready DAG library (networkx) for python existed and tried by the team. This library included creating different

types of graphs, most importantly was the Directed graph. Also, the library supported adding attributes to nodes (gates), and supported topological sorting for the sake of levels for each gate. Several test cases failed to be sorted because the flipflops created a non-acyclic graph which was not suitable for topological sorting. Link to the library:

http://networkx.github.io/documentation/networkx-1.9.1/reference/algorithms.dag.html

Another attempt was made using a different approach. This time the data structures created by the parsing phase (list of gates and list of wires), were used immediately instead of generating a graph. This is done by two functions: getfirstlevel and getnextlevel. The first function will find all the gates connected directly to the inputs of the parsed module, and returns a list of these gates. The "getnextlevel" function will then receive this returned list, find the gates directly connected to the gates in the received list, and finally returns a list of these gates. The "getnextlevel" is repeated until all the gates are connected to each other. This approach worked on small gatelevel netlist files, however errors were found when running it on large files, where wrong gate levels were outputted.

The GUI:

There was a fourth component to the project, a tkinter GUI system that failed at the last minute, and this was excluded out of the project due to insufficient time to fix it. The Tkinter Canvas Widget was used to create the main elements of GUI, the drawing of the circuit. This is based on the concepts of creating canvas items to represent the gates and the wires. The wires are drawn with the already built item of "line", however the gates were all built using "image". Using the list sent from the python code, tkinter would start by picking all gates which are level one. The coordinates are spaced over the y access , all starting at the x=0. Next are level two items which undergo the same process, with only the x coordinate different, since it's a multiple of its level. In here, the fill attribute of the shapes is decided upon by the level and a look-up table containing different hex colours mapping to each level.

After all gates have been placed on the canvas, Next comes adding the inputs and outputs, which are added immediately to all inputs of level zero and all outputs of level=Max Level. The last item to be added is the connecting wires between the gates, starting with the first level, the output port is used as a search term among the input ports of all other gates, the coordinates are then used to draw this line. Once the output port is done, it is deleted so it is not repeated again. The program stops once all output ports are set to NULL.

Testing Strategy:

For testing, the 5 given sample Verilog gate level net-list files were used. These files covered a wide scope of applications, and they covered all samples. Follows is the test plan applied:

Test File	Test Application	Expected Result
6502_cpu.g.v	Whole Application	Failed, because the circuit contained cycles
Mul4x4.g.v	Whole Application	Success
6502_alu.g.v	Whole Application	Success
Wallace.g.v	Whole Application	Success
Booth.g.v	Whole Application	Success
None existant path	Opening the File	Error

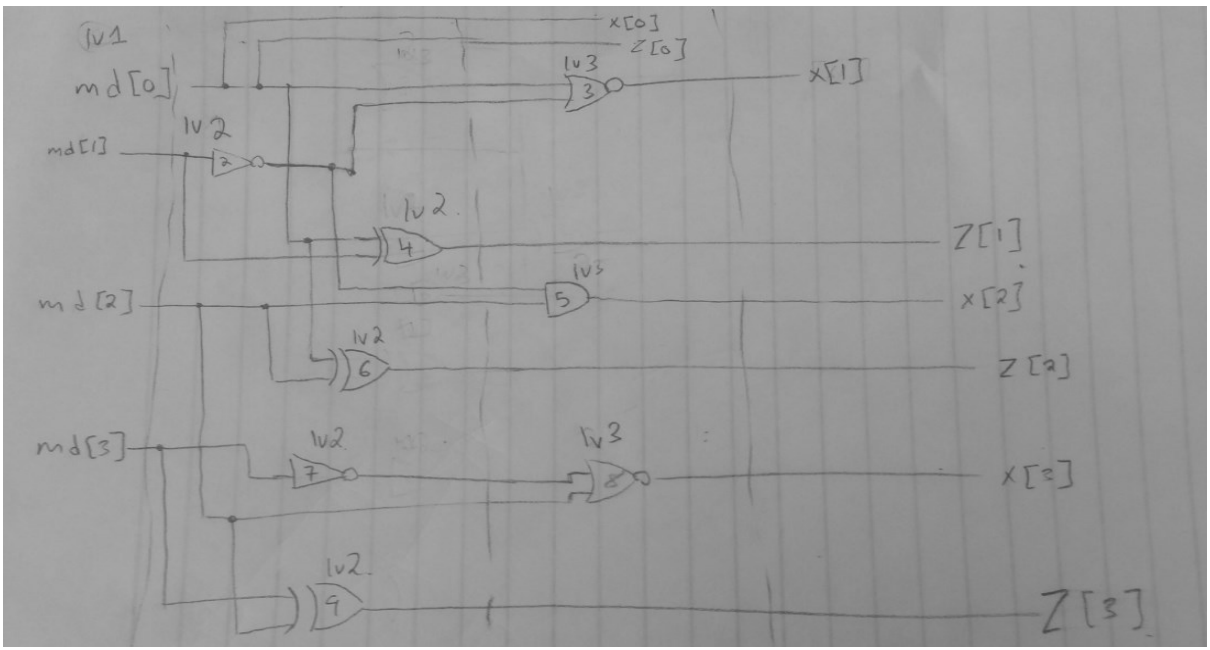
Testing:

Test File	Test Application	Expected Result	Output	Result as Expected?
6502_cpu.g.v	Whole	Fail	File Open	√

	Application		Correct Gates, wrong level	
Mul4x4.g.v	Whole Application	Success	File Open Correct Gates Wrong level	No
6502_alu.g.v	Whole Application	Success	File Open Correct Gates Wrong level	No
Wallace.g.v	Whole Application	Success	File Open Correct Gates Wrong level	No
Booth.g.v	Whole Application	Success	File Open Correct Gates and Levels	√
None existant path	Opening the File	Error	File failed to Open	√

Verification of Correctness:

To ensure the application worked, the booth.v was drawn out by hand to compare it to the output by the application. This was exactly, without any faults. Below is the diagram drawn to check the correctness.



Application Interface:

Start by changing the path of the file you want to process in the code, this is the first part in the main function, and is the 171st line of the code. Then proceed to run the code and you will get all the gates with their levels, their inputs and their outputs produced.