THE AMERICAN UNIVERSITY IN CAIRO
الجامعة الأمريكية بالقاهرة

Verilog Netlist Enhancer
Project Report

By
Hadeel Mabrouk, 900-16-3213
Alaa Mahmoud, 900-16-23-16

To
Professor. Mohamed Shalaan

Due date: Nov, 12th, 2019

- **Project Description**

  The Verilog netlist enhancer is a utility that modifies a certain given Verilog netlist to enhance its maximum fanout and delay, by either:

  - Sizing up cells with large fan-out
  - Cloning high fan-out cells
  - Adding buffers to high fan-out cells

  It accepts the Verilog netlist (to be enhanced) as well as the library liberty file. Then, the user specifies the maximum acceptable delay as well as the maximum desired fan-out for any cell in the design. The utility searches the netlist for the cells that violate these two rules and applies the above techniques to resolve the violations.

- **Approach**

  Our approach was to enhance the fanout by cloning, and adding buffer to cells with high fanouts to decrease the fanout and distribute the load between them. Other than cloning and adding buffers, the verilog netlist enhancer decreases the maximum fanout by sizing up cells that have large fanout.

- **Dependencies**

  In our project we used the Liberty-Parser library to extract data from library liberty files. On the other hand, for Verilog files parsing, we used the regular expression operations library (Regex). Also, we used NetworkX library, to represent the netlist as a directed graph, to help visualize the circuit and calculate the delay of the longest path, and for the graph drawing, we used Matplotlib.

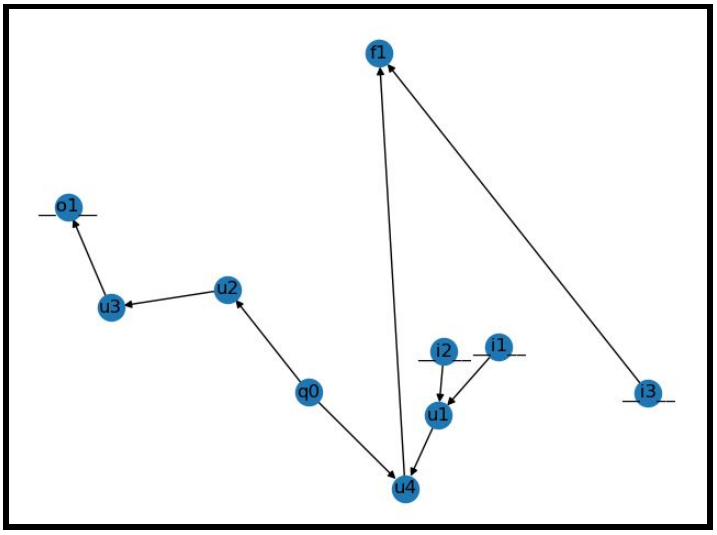
- **Design and Implementation**

The utility is composed of two main classes, the liberty data extraction class, and the Verilog netlist class. The liberty data extraction class is composed of three public functions:

  - The first function returns the input capacitance of a certain cell given its input pin name, to be used later in calculating the load capacitance on each output in the netlist.
  - The seconds function returns back the transition delay time, given the name of the cell, the pin name of the input, and the load capacitance. In this function, the utility uses interpolation to return back the transition delay assuming sublinearitly between each two discrete values.
  - The third function returns the value of the capacitance in the middle of the timing table, to be assumed as the load capacitance of the circuit outputs.

The Verilog netlist class is composed of three main data structures, the netlist dictionary, the wires dictionary, and the netlist directed graph.

  - The netlist dictionary is composed of the circuit instance as key, and its inputs, output, and load capacitance as values.
  - The wires dictionary is composed of the wire name as key, and its source and destinations as values.
  - In the netlist graph, the cells instances are represented as nodes, whereas the wires connecting them are represented as the edges connecting the nodes.



Also, the class is composed of eight main public functions.

- The first function reports the maximum delay, by using the longest path function of the DiGraph to return the delay of the netlist critical path.
- The second function reports the number of cells of each type in the circuit, by counting the netlist directory items.
- The third function returns the maximum existing fanout of the circuit.
- The fourth function applies adding buffers to cells instances whose fanout exceeds the maximum fanout given by the user. It calls a private function that adds buffers equal to the desired output to the cells with large fanout, and distribute the load on the added buffers, and then calls this function again until no cell exceeds the maximum fanout.
- The fifth function clones cells with large fanout and distribute the load on the cloned cells, to decrease the fanout. However, this may increase the fanout of other cells, so the function keeps iterating until the condition is satisfied, or returns an error message that the desired fanout cannot be satisfied.

- The sixth function applies sizing up to the cells that exist in the STL with larger size, to decrease the delay of the critical path. It applies sizing up to cells in the critical path using a greedy algorithm that checks whether sizing up would increase or decrease the total delay and takes a decision accordingly, and keep iterating of different cells, as well as updating the critical path, until the delay constraint is satisfied, or it prints an error message.
- The seventh function is called to draw the graph representing the netlist for visual representation.
- The eight function prints out the netlist after the enhancement to be applied by the user to the original Verilog netlist file.

- **Bonus Features:**
  - The function that reports the total delay. In reporting the delay, we took the more accurate approach of getting the critical path delay rather than summing up the delays of all cells, using the longest path function of the DiGraph to return the delay of the netlist critical path.
  - The other function that reports the number of cells of each type in the circuit, by counting the netlist directory items.
  - The visual representation of the netlist using NetworkX.

- **Snapshots of The Utility Interface:**

```
(base) D:\Uni\Fall 19\-Verilog-Netlist-Enhancer-master>python main.py
please enter the name of the v file: buffering_test.v
please enter the name of the liberty file: osu035.lib


================================================================================
Please choose an option of the following:
- delay: report the maximum delay of the circuit
- n-cells: report the total number of cells in the circuit
- fanout: report the max fanout of the circuit
- buffer: satisfy the max fanout constraint using buffering
- clone: try to satisfy the max fanout constraint using cloning
- size: do greedy sizing algorithm to decrease the delay of the critical path
- graph: visualize the circuit as a graph
- netlist: print the current verilog netlist
- quit: quit the program
================================================================================
> fanout
8


================================================================================
Please choose an option of the following:
- delay: report the maximum delay of the circuit
- n-cells: report the total number of cells in the circuit
- fanout: report the max fanout of the circuit
- buffer: satisfy the max fanout constraint using buffering
- clone: try to satisfy the max fanout constraint using cloning
- size: do greedy sizing algorithm to decrease the delay of the critical path
- graph: visualize the circuit as a graph
- netlist: print the current verilog netlist
- quit: quit the program
================================================================================
> buffer
please enter the desired max fanout: 4
Max Fanout:  4
```

```
================================================================================
Please choose an option of the following:
- delay: report the maximum delay of the circuit
- n-cells: report the total number of cells in the circuit
- fanout: report the max fanout of the circuit
- buffer: satisfy the max fanout constraint using buffering
- clone: try to satisfy the max fanout constraint using cloning
- size: do greedy sizing algorithm to decrease the delay of the critical path
- graph: visualize the circuit as a graph
- netlist: print the current verilog netlist
- quit: quit the program
================================================================================
> delay
0.5209206846000001
```

```
module simple (
inp1,
inp2,
iccad_clk,
out
);

// Start PIs
input inp1;
input inp2;
input iccad_clk;

// Start POs
output n2;
output n3;
output n4;
output n5;
output n6;
output n7;
output n8;
output n9;

// Start wires
wire n1;
wire n2;
wire n3;
wire n4;
wire n5;
wire inp1;
wire inp2;

// Start cells
NAND2X1 u1 ( .A(inp1), .B(inp2), .Y(n1) );
INVX1 u2 ( .A(__wire0__), .Y(n2) );
INVX1 u3 ( .A(__wire0__), .Y(n3) );
INVX1 u4 ( .A(__wire0__), .Y(n4) );
INVX1 u5 ( .A(__wire0__), .Y(n5) );
INVX1 u6 ( .A(__wire1__), .Y(n6) );
INVX1 u7 ( .A(__wire1__), .Y(n7) );
INVX1 u8 ( .A(__wire1__), .Y(n8) );
INVX1 u9 ( .A(__wire1__), .Y(n9) );
BUFX2 __buffer0__ ( .A(n1), .Y(__wire0__) );
BUFX2 __buffer1__ ( .A(n1), .Y(__wire1__) );
```

- **Limitations**

The current version of the utility does not support circuits having cells with multiple outputs. Also, the list of the supported cells is fixed, and the load capacitance of the circuit is assumed to be a fixed value, equal to the value of the middle capacitance in the delay timing table. Also, the utility assumes the transition time to be always a fixed value. Furthermore, the current version does not support calculating the delay for the rising and falling transitions of the inputs, as it always chooses the larger transition delay.

- **Further Suggestions**

Further suggestions include using optimization algorithms to enhance the delay and fanout in a more automated way, as well as enhancing the visualization of the circuit by

making the nodes size and shape representative of its size and type, highlighting violations, and/or making the thickness of the edge representative of its weight.