CSCE 337: Digital Design II
Projects 2

In this project you are required to develop (using any programming language of your choice) a static timing analysis tool that gets (1) the Verilog gate level net list of a digital circuit, (2) the library of the cells used in the circuit, (3) the net capacitances file, (4) the clocks skew file and (5) timing constraints (the clock period, input delays and output delays). The tool checks all the timing paths and reports the violating timing paths, if any. Also, your tool should perform basic optimizations (area recovery) as well as basic timing violations fixing.

The output from the tool is a report of 10 (or less in case of small circuits) paths with the least slacks sorted according to the slack (ascending). For each reported path, you should show the delay across each gate (stage) and the incremental delay as in the following example:

```
-----------------------------------------------------------
Pin         type        Incr        Path delay
-----------------------------------------------------------
A[0]        (in)        0.00        0.00             r
U10/A       (NAND2)     0.00        0.00             r
U10/Y       (NAND2)     1.00        1.00             r
U9/A        (NOT1)      0.00        1.00             f
U9/Y        (NOT1)      1.00        2.00             f
U2/B        (AND2)      0.00        2.00             f
U2/Y        (AND2)      1.00        3.00             f
U1/A        (AND2)      0.00        3.00             f
U1/Y        (AND2)      1.00        4.00             f
M[2]        (out)       0.00        4.00             f
-----------------------------------------------------------
Data Arrival Time                   4.00
Data Required Time                  5.00
-----------------------------------------------------------
Slack                               1.00
```

For testing purpose you need to generate gate level netlist for some designs using yosys synthesis tool.

Requirements

- Task 1: Reading the Verilog gate level net list file, and identify the timing paths and construct the DAG. [10%]
- Task 2: Read the library file and convert it into internal data structure(s) [10%]
- Task 3: Read the clock skews file [5%]
- Task 4: Read the net capacitances file [5%]
- Task 5: Perform Timing analysis by calculating the Arrival and Required times at each node of the DAG as well as the slacks. Also, generating the timing report [40%]
- Task 6: Area and Power Optimization by reducing the cell sizes over timing paths with positive slacks (try to achieve zero slack) [15%]
- Task 7: Fix any timing violation using cell sizing and cell cloning techniques. [15%]

Rules:

- The project should be demonstrated before the final exam. Each team is responsible for scheduling an appointment with Dr. Shalan for project demonstration (15-20 minutes per team).

Static Timing Analysis (STA)

STA is a method of computing the expected timing of a digital circuit without requiring simulation. The delays over the timing paths must be calculated in order to determine the maximum clock frequency that can be used in the system. The path with the longest delay, critical path, determines the clock frequency. To find the critical path the timing paths are usually represented using a Directed Acyclic Graph (DAG): G = <V, E>. as shown in the following figure.

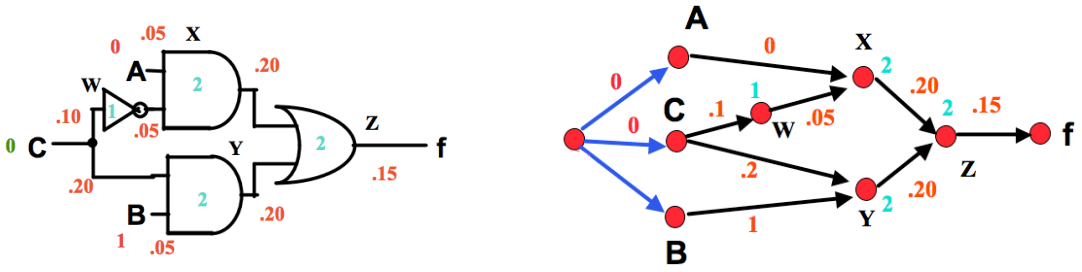
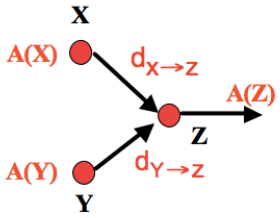


Figure 1. DAG representation of a circuit

Arrival time in green, gate delay in blue and interconnect (wire) delay in red. Vertices represent gates, primary inputs and primary outputs. Edges represent wires. Let's define the Actual arrival time A(v) for a node v as latest time that signal can arrive at node v.



$$A(v) = \max_{u \in FI(v)} (A(u) + d_{u \to v})$$

Figure 2. Arrival Time Calculation

Where $d_{u \to v}$is delay from u to v, and *FI(v) = {X,Y}* ; *v={z}.* The calculation begins with the start-point and progresses toward the end-point(s). Figure 3 shows an example of Arrival Time calculation for the circuit in Figure 1.

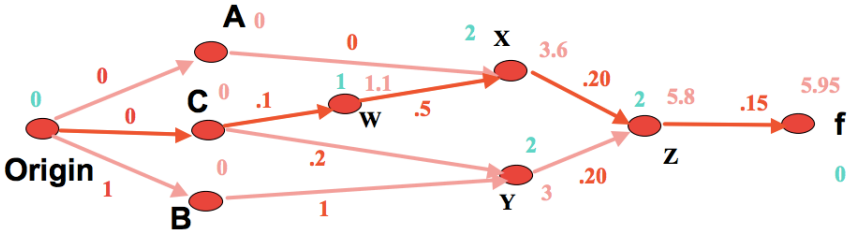


Figure 3. Arrival Time Calculation Example

To find the path with the largest delay you may use the following algorithm [proposed by Kirkpatrick and Clark in "PERT as an aid to logic design" paper published in IBM Journal of Research and Development 1966].

```
// Compute longest path in a graph G = <V,E,delay,Origin>
// delay is set of labels, Origin is the super-source of the DAG
Longest path(G){
  Forward_prop(Origin)
}

Forward-prop(W){
  for each vertex v in W
    for each edge <v,w> from v
      Final-delay(w) = max(Final-delay(w),delay(v)+delay(w)+delay(<v,w>))
      if all incoming edges of w have been traversed, add w to W }
```

An example of applying the algorithm is provided in the project archive.

Optimization

If we have constraints (clock frequency, input delay, output delay, etc.,) need to propagate slack (Required Time – Arrival Time) to each node. The measures how much timing margin exists at each node. The slack can be used to optimize particular timing paths (trade slack for power, area, etc.). To calculate the Required arrival time (R) at any node, we use:

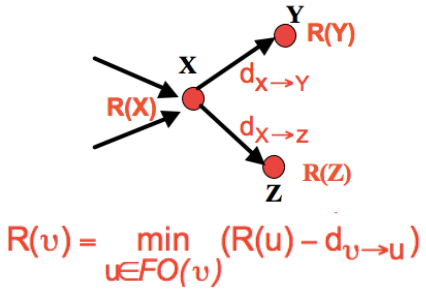


Figure 4.

Where *FO(v)={Y,Z}* and *v={X}.* The Required time calculation starts from the timing path end-point and progresses toward the start-point. Example of Required time calculation (Assume required time at output R(f) = 5.80) is in shown in Figure 5.

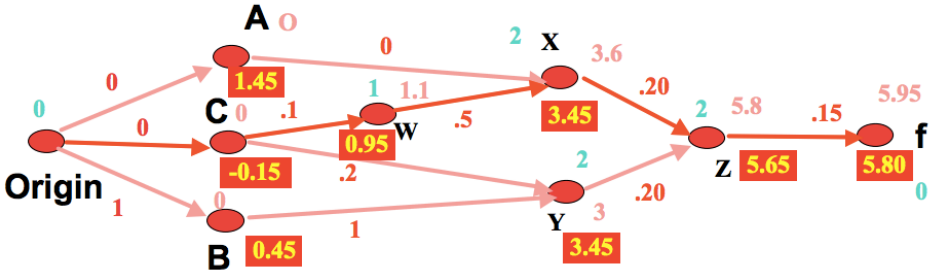


Figure 5.

From arrival and required time can compute slack. Slack reflects criticality of a node. Slack distribution is key for timing optimization. For each node *v*:

$$S(v) = R(v) - A(v)$$

- Positive slack: Node is not on critical path. Timing constraints met. We can use the positive slack to optimize the design, For example, reducing the size of some cells to reduce the area and the power while keeping Slack≥0.
- Zero slack: Node is on critical path. Timing constraints are barely met.
- Negative slack: There is a timing violation. We need to fix the timing problem.

Figure 6 shows the slack calculations for the circuit in Figure 1.

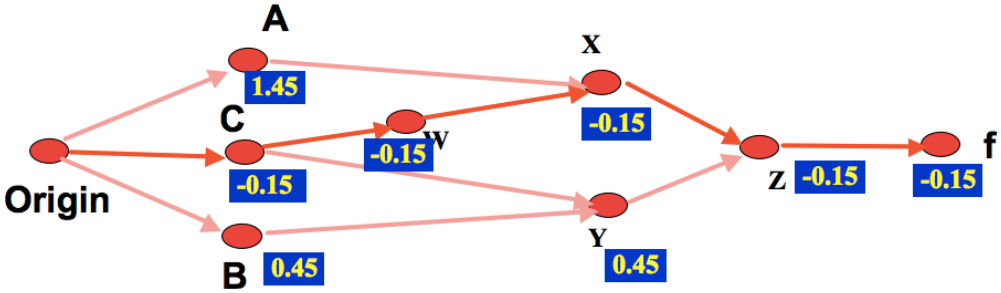
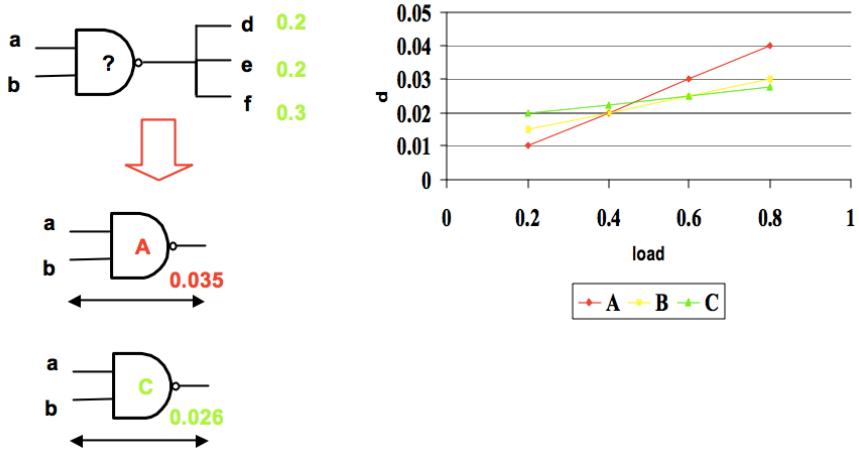


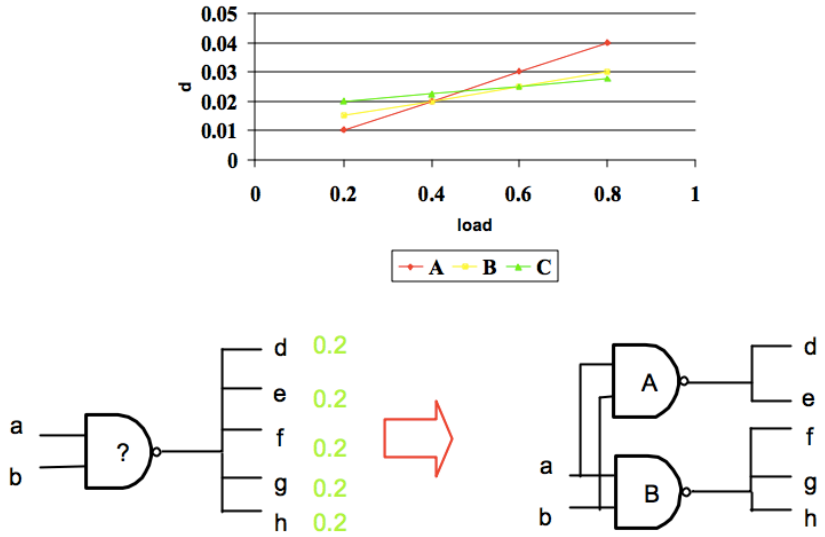
Figure 6.

Many transformations can be applied to fix the timing problems:

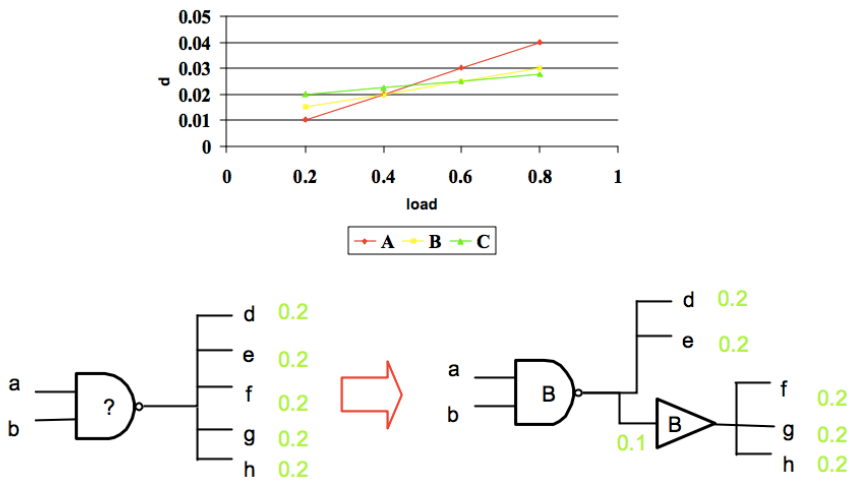
(a) Resizing



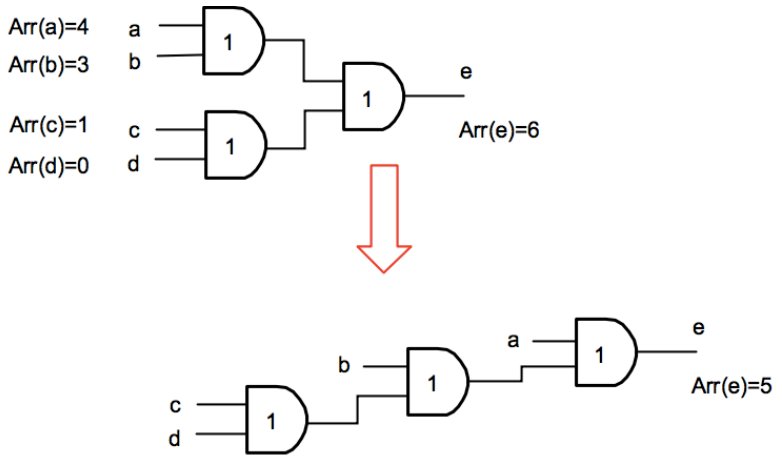
(b) Cloning



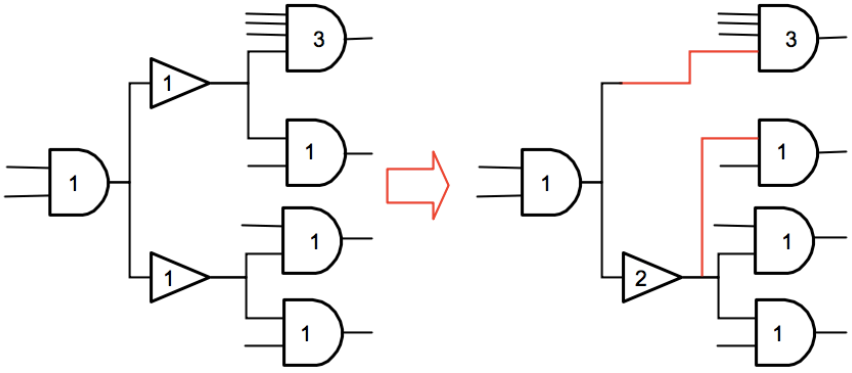
(c) Buffering



(d) Re-designing Fan-in Tree



(e) Re-designing Fan-out Tree



Resources

Yosys synthesis tool:

- http://www.clifford.at/yosys/

Liberty Standard Cell Library Format:

- http://link.springer.com.library.aucegypt.edu:2048/chapter/10.1007/978-0-387-93820-2_3

Liberty parsers:

- http://search.cpan.org/~yorkwu/Liberty-Parser-0.04/lib/Liberty/Parser.pm
- http://search.cpan.org/~ega/Parse-Liberty-0.1/lib/Parse/Liberty.pm, https://code.google.com/p/liberty-parser/downloads/list
- http://vlsicad.ucsd.edu/~sharma/Research/software/liberty_parser/

Verilog Perl library:

- http://search.cpan.org/~wsnyder/Verilog-Perl-3.316/Netlist.pm
- http://www.burbleland.com/v2html/rvp.html

Good Book (You may read Chapter 8):

- http://link.springer.com.library.aucegypt.edu:2048/book/10.1007/978-90-481-9591-6/page/1