CAD Lab2 0811562 何祁恩

1. Parse the library & netlist file

I have used Perl for document parsing with regular expressions during a summer internship. Therefore, I am employing the same technique but in C++. I simply identify the pattern and extract the desired information. Additionally, I efficiently remove comments in Verilog code. For the data structure to store lookup tables and the circuit with cells and nets, I use a hash table for constant-time access. Furthermore, I store net information in cells and cell information in nets. This nested structure idea is inspired by FM partitioning. By doing so, we can traverse the directed acyclic graph efficiently.

2. Calculate output loading

As mentioned earlier, with net information stored in cells and cell information in nets, I can easily traverse the output of a cell to identify the next-stage cell. This allows me to sum the input capacitance together, determining the output loading of the current cell.

3. Topological Sort with node degree

I implement a topological sort using node degrees. Initially, I calculate the number of directed edges pointing to specific cells. I start by selecting cells with an in-degree of 0 and perform a breadth-first search (BFS). During this process, I update the in-degrees of neighboring cells. Once a cell's in-degree becomes 0, it is considered available for use.

[reference: https://www.geeksforgeeks.org/topological-sorting-indegree-based-solution/]

4. Perform STA

After obtaining the topological order of the cells, we can easily use the lookup tables built from the library to find the propagation delay value based on the input slew and output loading with different output state. In this lab, we are asked to find the most pessimistic timing arc. Additionally, I've implemented a linked-list structure to facilitate easy backtracking of the path.

5. Longest Path & Shortest Path

After calculating the propagation delay and arrival time for each cell, we can identify the longest and shortest delays among all cells connected to the primary output net. Leveraging the linked-list data structure mentioned in step 4, we can efficiently backtrack the path.

Summary:

This lab is really fun to me, to implement the simple STA engine is really cool and I really learn a lot of timing analysis by implementing this STA engine.