CAD Lab3 0811562 何祁恩

1. Parse the library & netlist file & pattern file.

I have experience using Perl for document parsing with regular expressions during a summer internship, and I am now implementing a similar approach in C++. The process involves pattern identification and extraction for desired information. Additionally, I efficiently remove comments in Verilog code. For managing lookup tables and the circuit with cells and nets, I utilize a hash table for constant-time access. Inspired by FM partitioning, I adopt a nested structure to store net information in cells and cell information in nets, facilitating efficient traversal of the directed acyclic graph. Lastly, to parse pattern data, I leverage a vector hash table, allowing me to feed patterns to the circuit in O(1) time complexity multiplied by the number of input pattern bits.

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. Calculate Gate Info & Gate Power

By considering the path sensitization rule and utilizing the HW2 lookup table, we can readily determine the output logic value for each gate. Furthermore, in calculating gate power, we adopt a conservative approach that calculates both internal power and switching power. While this calculation may incur some overhead, it ensures that the overall power will not exceed the values determined in this analysis.

4. Coverage Analysis

Now, with numerous patterns at hand, we aim to assess how a gate toggles within the first 20 toggles. This coverage analysis method, considering path sensitization, effectively simulates precise power and timing analysis without accounting for the fault path. Additionally, in exhaustive pattern simulations, it can curtail the patterns that could lead to hidden arcs (no toggle from pattern1 to pattern2), thereby reducing simulation time without sacrificing coverage.

Summary:

In this project, I apply my Perl parsing experience to C++, efficiently handling Verilog code and employing data structures like hash tables for circuit management. I calculate output loading by traversing cells, determine gate logic and power with path sensitization, and conduct coverage analysis for gate toggling within the first 20 patterns. This approach ensures accurate power and timing simulation while optimizing for efficiency and coverage.