CAD Programming Assignment 2
Static Timing Analysis

Lena Chen

October 2025

0 Objective

Given a gate-level netlist and a cell library, find the longest and the shortest delay and their corresponding path under a given input pattern. There are 4 steps to follow:

- Step 1: Build the delay graph according to the given netlist.
- Step 2: Calculate the delay of each instance in the graph following topological order.
- Step 3: Find the longest and the shortest delay and their corresponding path.
- Step 4: Calculate the logic value, delay, and transition time.

1 Deliverables

- Source code in C or C++ (`.c/.h` or `.cpp/.hpp`).
- A **Makefile** that builds an executable named `sta`. (A sample `Makefile` will be provided for reference; you may modify it as needed; it does not need to be identical.)
- The executable `sta` must generate an analyzer file.

Submission Instructions

Please put all required files into a folder named `student_ID_PA2` (where `student_ID` should be replaced with your actual student ID). Then compress it into a `.tar` archive before submission:

```
tar cvf student_ID_PA2.tar student_ID_PA2
```

2 Program Interface

```
./sta <netlist_file> -l <lib_file> -i <input_patterns_file>
```

- Input: The program accepts two primary input files:
  - Circuit netlist file (`<netlist_file>`).
  - Standard cell library file (`<lib_file>`).
  - Input patterns file (`<input_patterns_file>`)
- **Output:** Upon successful execution, the program will generate the following files:
  - Output loading of each instance file (`<lib_name>_<module_name>_load.txt`)
  - Propagation delay and output transition time of each instance file (`<lib_name>_<module_name>_delay.txt`)
  - Longest and shortest delay among all output paths file (`<lib_name>_<module_name>_path.txt`)
  - Logic output, cell delay, and transition time of each gate file (`<lib_name>_<module_name>_gate_info.txt`)
- **Exit code:** 0 indicates success; non-zero indicates an error.

3 File Formats

3.1 Netlist Input File

Your program will take a **gate-level Verilog netlist** as its primary input. This file describes the circuit's structure and adheres to the following specifications:

- Circuit Structure: The netlist is flattened, meaning it contains only a single top-level `module`. The circuit is guaranteed to be **purely combinational**; you do not need to handle sequential elements like flip-flops or latches.
- Cell Library: For simplicity, the netlist will exclusively use a simplified library of **3 standard cells**:
  - 2-Input Gates: `NAND2`, `NOR2`
  - 1-Input Gates: `INV` (inverter)
- Parsing Requirements: Your parser must be robust and correctly interpret the netlist's logic regardless of its formatting. It should properly handle:
  - Arbitrary whitespace (extra spaces, tabs, newlines).
  - Both single-line (`// ...`) and multi-line (`/* ... */`) comments.

Listing 1: The gate-level netlist named example.v

```
1  module prob2(n11, n12, n13, n1, n2, n3);
2  output n11, n12, n13;
3  input n1, n2, n3;
4  //internal wires
5  wire n4, n5, n6, n7, n8, n9, n10;
6      INVX1 g1(.ZN(n4), .I(n1));
7      INVX1 g2(.ZN(n5), .I(n2));
8      NANDX1 g3(.ZN(n6), .A1(n4), .A2(n5));
9      INVX1 g4(.ZN(n7), .I(n5));
10     NOR2X1 g5(.ZN(n8), .A1(n4), .A2(n3));
11     INVX1 g6(.ZN(n9), .I(n6));
12     NOR2X1 g7(.ZN(n10), .A1(n6), .A2(n7));
13     NANDX1 g8(.ZN(n11), .A1(n7), .A2(n8));
14     NOR2X1 g9(.ZN(n12), .A1(n9), .A2(n10));
15     NOR2X1 g10(.ZN(n13), .A1(n10), .A2(n8));
16 endmodule
```

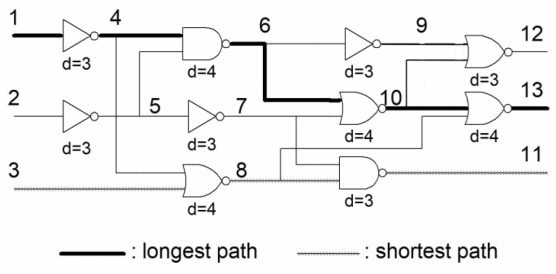

Figure 1: The corresponding circuit diagram of Listing 1

3.2 LIB Input File

This file is simplified from the standard `liberty` format, which records the timing and power information of each cell. The default units used in this file are: ns (timing), pF (capacitance), V (voltage), and mA (current).

To simplify the problem, we assume that the timing and power information of the same output are identical for all input paths. For example, the paths `A1→ZN` and `A2→ZN` share the same delay and power characteristics. Therefore, only one lookup table is recorded for each output.

The table index is defined in `lu_table_template`, which includes:

1. Total output loading
2. Input transition time

Please note that both quantities should be calculated based on the actual circuit connections:

- The input capacitance of a cell equals the output loading of its preceding cell.
- The output rising/falling time of a cell becomes the input transition time of its succeeding cells.

- Liberty File Parsing Requirements: Your parser must robustly interpret the `.lib` file and correctly extract timing and power information. Specifically, it should handle:
  - Arbitrary whitespace (extra spaces, tabs, newlines).
  - Comments (`/* ... */`) are supported in Liberty (single-line or multi-line).
  - Cell name order may differ. The gate type can be determined by the name prefix (`NAND`, `NOR`, `INV`).
  - Cells may appear in any order; your parser must correctly handle rearranged cell blocks, including all their internal sections (`pin`, `internal_power`, `timing`), without assuming a fixed order.
- Note: Rearranging cell blocks does not change the functionality of the library.

For additional examples and detailed explanations, refer to the document `LibertyFile.pdf`. For simplicity, the numerical values shown in the examples may differ from those in the provided library file. During evaluation, the TA will test your program using a different `.lib` file that follows the provided template.

3.3 Input Patterns File

This file gives the values of each input. The first line gives the name and order of each input. Input patterns are given in the following lines. Each pattern is given at a separate line. Note that the index of the input node is not necessarily consecutive. Also, they are not necessary in ascending order. Below is a simple example of the netlist in Listing 2. In the below example, the first pattern is n1 = 0, n2 = 1, n3 = 0.

Listing 2: An example of the input pattern

```
1 input n1, n2, n3
2 0 1 0
3 1 1 0
4 1 1 1
5 .end
```

3.4 Output File

After executing your program, four output files should be generated for each step listed at the very beginning of this assignment. You should follow the naming rules below. For example, if the file name of the netlist file is `example.v`, the `<module_name>` would be "example".

Step 1: `<lib_name>_<module_name>_load.txt`
Step 2: `<lib_name>_<module_name>_delay.txt`
Step 3: `<lib_name>_<module_name>_path.txt`
Step 4: `<lib_name>_<module_name>_gate_info.txt`

Step 1: Calculate the output loading of each instance

In this step, we construct the circuit graph according to the given netlist. To verify the correctness of the graph construction, we calculate the **output loading** of each cell.

1. Construct the circuit graph from the given netlist, where nodes represent logic cells and edges represent signal connections.
2. For each cell, find all fanout cells connected to its output.
3. Calculate the output loading by summing the input capacitances of the fanout cells.
4. For primary outputs, directly assign a loading of 0.03 pF.
5. Sort all instance numbers in ascending order.

6. Print each output loading value with precision up to the **6th decimal place**.

Listing 3: An example of `<lib_name>_<module_name>_load.txt`

```
g1  0.017647
g4  0.017337
g6  0.010501
...
g18  0.030000
g19  0.030000
g20  0.030000
```

Step 2: Calculate the propagation delay and output transition time of each instance

Before finding the worst-case delay path for the whole circuit, the propagation delay and output transition time of each instance must be determined based on the circuit connection. Input transition time is defined by the input that arrives latest if the instance has multiple inputs. The **input transition time** of an instance is defined by the input signal that arrives the latest if the instance has multiple inputs.

The **propagation delay** and **output transition time** of the instance are then determined by the worst-case output. Specifically, both output conditions (i.e., output = 0 and output = 1) should be evaluated, and the one with a larger propagation delay is considered the worst-case output.

- If the output is 0, the output transition time is defined by its *falling time*.
- If the output is 1, the output transition time is defined by its *rising time*.

In this assignment, the correctness of gate logic does not need to be considered. Instead, the goal is to find which output produces the larger propagation delay.

To verify the results of this step, report the worst-case output and timing information of each instance in the following format:

`<Instance Name>` `<Worst-case Output>` `<Propagation Delay>` `<Output Transition Time>`

Each row represents one instance in the circuit. Additionally, assume that each wire between any two instances has a fixed delay of **0.005 ns**. The propagation delay and output transition time values must be printed to the **6th decimal place**, and all instance numbers must be sorted in ascending order.

Listing 4: An example of `<lib_name>_<module_name>_delay.txt`

```
g1 0 0.022268 0.031807 // output=0, propagation delay=0.022268, output fall time=0.031807
g2 0 0.022635 0.033406
g4 0 0.034014 0.047876
...
g10 1 0.089355 0.145344 // output=1, propagation delay=0.089355, output rise time=0.145344
```

Step 3: Find the longest and the shortest delay from all output paths

According to the delay information, we need to compare the worst-case delays of all primary outputs.

- The output with the largest worst-case delay is considered the **longest delay**.
- The output with the smallest worst-case delay is considered the **shortest delay**.

The results should be reported as follows:

1. Only the **nets' names** are required in the delay path; the instances' names are not necessary.
2. Output both the longest and shortest delays, along with their corresponding delay paths.
3. The calculation of delays must be based on the library file `test_lib.lib`.

4. If an instance has multiple input signals that arrive at the same time, any one of them can be reported.

5. The value of the path delay must be printed to the **6th decimal place**.

Listing 5: An example of `<lib_name>_<module_name>_path.txt`

```
Longest delay = 0.246039, the path is: n2 -> n5 -> n6 -> n10 -> n12
Shortest delay = 0.160504, the path is: n1 -> n4 -> n8 -> n11
```

Step 4: Calculate logic output, cell delay, and transition time of each gate

1. Logic Output

Calculate the **output logic value** of each gate according to the given input pattern. This step determines the logical behavior of the circuit before evaluating the timing information.

2. Cell Delay and Transition Time

After obtaining the logic output of each gate, the corresponding **cell propagation delay** and **output transition time** are determined by looking up the provided timing table. The procedure is as follows:

- For the input gates, the input transition time is assumed to be **0 ns**.
- For gates with multiple inputs, the input transition time is selected based on the **controlling value** and the **total delay** path.

The results should be sorted according to the **instance number in ascending order**.If there are more than one input pattern, please report the delay information of each input pattern separated by a blank line.

Listing 6: An example of `<lib_name>_<module_name>_gate_info.txt`

```
g1 1 0.052093 0.079336
//output=1, output rise delay = 0.052093, output rise transition time = 0.079336
g2 1 0.070989 0.118823
g3 1 0.066306 0.108485
g4 1 0.043873 0.063469
g5 0 0.068176 0.111416
//output=0, output fall delay = 0.068176, output fall transition time = 0.111416
```

If there are multiple input patterns, separate each block with a blank line:

```
g1 (logic value) (cell delay) (transition time)
// the beginning of the first pattern
...
g10 (logic value) (cell delay) (transition time)
 // blank line here !!
g1 (logic value) (cell delay) (transition time)
// the beginning of the second pattern
...
g10 (logic value) (cell delay) (transition time)
```

4 Important Notice

The delay time of each instance can be determined by the input transition time (the output transition time of its driving cell) and the output loading (the total input capacitance for all fanout gates) according to the given delay table.

4.1 Boundary Conditions

- Set the input transition time of each primary input as 0ns.
- Set the output loading of each primary output as 0.03pF.

4.2 Assumptions for Worst-Case Delay Calculation

To simplify the problem, please follow the assumptions below while calculating the worst-case delay:

1. If there are multiple paths from cell inputs to the cell output, the **input transition time** is defined by the input that arrives the latest.
2. For the **propagation delay**, use the table corresponding to the output value:
   - If the output value of the cell is 1, use the `cell_rise` table.
   - If the output value of the cell is 0, use the `cell_fall` table.
3. For the **output transition time**, also use the table corresponding to the output value:
   - If the output value of the cell is 1, use the `rise_transition` table.
   - If the output value of the cell is 0, use the `fall_transition` table.

   The successor cells will use this calculated value as their input transition time.
4. For **wire delay**, assume every wire between any two instances has a fixed delay of 0.005ns.

Finally, you must print all numerical values to the **6th decimal place**, except for the worst-case output value of each instance.

5 Building and Running

5.1 Compilation

```
make
```

6 Makefile Requirements

- Target executable: `sta`.
- Default target: `all`.
- Must support `make clean`.
- Must be able to build on the course server.

7 Grading Policy

The total grade for this assignment is composed of two parts: **Correctness** and **Runtime Performance**, which together determine both the functional accuracy and the efficiency of your program.

- **Total: 100 points**
  - Correctness: 75%
  - Runtime Performance: 25%

Evaluation Details

- There are **5 test cases** in total: 2 public and 3 hidden.
- Each test case is worth **20 points**, divided as follows:
  - Step 1 result: 2 points
  - Step 2 result: 4 points
  - Step 3 result: 4 points
  - Step 4 result: 5 points
  - Performance: 5 points

Grading Criteria

1. Correctness (75%) Your implementation will first be evaluated based on the correctness of its results for each test case. All steps must produce correct outputs to receive full correctness credit.

Error Tolerance

Two values are considered equal if their absolute difference is within the tolerance:

$$|\,\text{userLoad} - \text{goldenLoad}\,| \leq \varepsilon$$

where

$$\varepsilon = 10^{-6}.$$

That is, differences smaller than 0.000001 are regarded as acceptable.

2. Runtime Performance (25%) Among all submissions that pass all steps, the remaining 25% of the grade will be based on runtime efficiency.

- The ranking is determined by the **execution time** of each test case.
- No runtime score is awarded unless all steps are correct.
- **Hint:** Efficient use of C++ STL containers is recommended.
- Suppose there are N valid submissions for a test case. Let $\text{rank}(i)$ be the ranking of submission i (1 = fastest, N = slowest). The performance score S_i is computed as:

$$S_i = \frac{N - \text{rank}(i) + 1}{N} \times 5$$

  Thus, the best submission (rank = 1) receives 5 points, while the slowest submission (rank = N) still receives $\frac{5}{N}$ points.
- **Time limit:** Each program has a maximum runtime of 1 minutes per case.

Additional Requirements

- Hidden test cases will be used for final grading.
- Violating file naming rules: **-10 points**.
- Failing to follow output format requirements: **-10 points**.
- Printing any text on the terminal during execution: **-10 points**.
- For submitting the executable file: **-10 points**.
- If **plagiarism** or **LLM-generated code** is detected, the score will be **0**.
- The use of **third-party solvers** is strictly prohibited.
- The program must compile and execute successfully on the official workstation.