CAD Programming Assignment 2
Static Timing Analysis
(Due: 8/28)

1. Problem Description

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.

2. Input Format

(a) Flattened gate-level Verilog netlist (.*v*)

This file follows the standard Verilog gate-level syntax.

To simplify the problem, the given netlist is flattened, i.e., only one module exists in the file. Also, only three kinds of cells (NAND2, NOR2, INV) will appear in this file. Sequential circuits are not necessary to be considered.

Your program should be able to deal with extra space (or TAB), extra new lines, block comments (/**/), and the comment lines starting with a double slash (//) in this netlist file. Below is a simple example. Please note that the delay values in this example are just assumptions to simplify the calculation. Actual delay should be obtained from the given library information.

```
module prob2(n11, n12, n13, n1, n2, n3);
      output n11, n12, n13;
      input n1, n2, n3;
      //internal wires
      wire n4, n5, n6, n7, n8, n9, n10;
      INVX1  g1(.ZN(n4), .I(n1));
      INVX1  g2(.ZN(n5), .I(n2));
      NANDX1 g3(.ZN(n6), .A1(n4), .A2(n5));
      INVX1  g4(.ZN(n7), .I(n5));
      NOR2X1 g5(.ZN(n8), .A1(n4), .A2(n3));
      INVX1  g6(.ZN(n9), .I(n6));
      NOR2X1 g7(.ZN(n10), .A1(n6), .A2(n7));
      NANDX1 g8(.ZN(n11), .A1(n7), .A2(n8));
      NOR2X1 g9(.ZN(n12), .A1(n9), .A2(n10));
      NOR2X1 g10(.ZN(n13), .A1(n10), .A2(n8));
endmodule
```

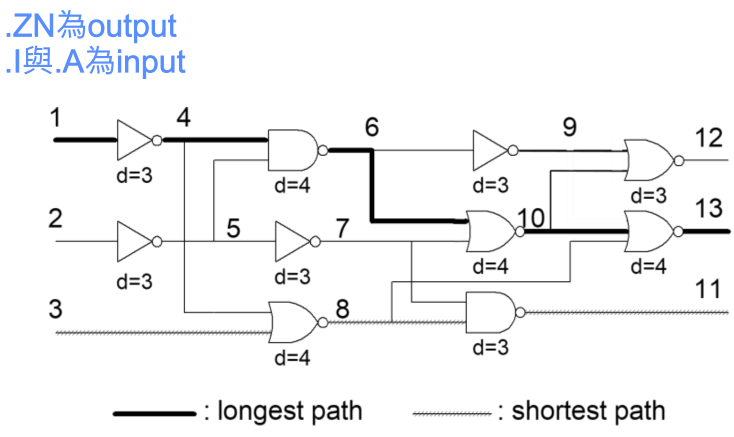

Figure 1 : An example of the gate-level netlist and its corresponding description

(b) Simplified liberty file (*test_lib.lib*)

This file is simplified from the standard liberty format, which collects the timing and power information for each cell. The default units are also given in the file, which 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 is the same for all input paths, i.e., A1->ZN and A2->ZN have the same delay and power. Therefore, only one table is recorded for each output.

The table index is defined in lu_table_template, which includes total output loading and input transition time. Please note that the total output loading and input transition time should be calculated based on the real circuit connection. The input capacitance of a cell is the output loading of the preceding cell. The output rising/falling time of a cell will be the input transition time of the succeeding cells.

Check "LibertyFile.pdf" for examples and more details. For simplicity, the numbers in the example may not be the same as in the given library file. TA will always use the same .lib file given to you to verify your program.

(c) Input patterns (.pat)

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 Figure 2. In the below example, the first pattern is n1 = 0, n2 = 1, n3 = 0.

```
input n1, n2, n3
0 1 0
1 1 0
1 1 1
.end
```

Figure 2: An example of the input pattern

3. Output Format

After executing your program, three 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 <case_name> would be "***example***".

Step 1: ***<student_id>_<case_name>_load.txt***
Step 2: ***<student_id>_<case_name>_delay.txt***
Step 3: ***<student_id>_<case_name>_path.txt***
Step 4: ***<student_id>_<case_name>_gate_info.txt***

(a) Step 1: Calculate the output loading of each instance

Construct the circuit graph according to the given netlist. In this step, we verify the correctness of the graph construction by checking the output loading of each cell. The output loading of each cell would be the summation of the input capacitance of all the fanout cells. For the primary output in the given netlist, please set the output loading as 0.03pF. 最後一級的output設為0.03pF

Besides, you must print each value to the 6th decimal place.
浮點數精度要到第六位

```
g8 0.030000
g9 0.030000
g10 0.030000
……
g4 0.017647
g1 0.017337
g6 0.010501
```

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

Before finding the worst-case delay path for the whole circuit, you must determine the propagation delay and output transition time of each instance based on the circuit connection. Input transition time is defined by the input that arrives latest if the instance has multiple inputs. Propagation delay and output transition time of the instance are then defined by the worst-case output, i.e., you need to calculate both output situations (output = 0 & 1); the one that has a larger cell propagation delay will be the worst-case output.

If the output is 0, the output transition time is defined by its output falling time.
If the output is 1, the output transition time is defined by its output rising time.
In this assignment, you don't need to consider the correctness of gate logic; just find the output that produces a larger propagation delay. output transition time 要看最後抵達該gate的訊號的rise/fall time

To verify the results of this step, please report the worst-case output and timing information of each instance in the following format. Each row shows an instance name, worst-case output value, corresponding worst-case delay, and output transition time in order. Also, we assume that each wire between any two instances has a 0.005ns delay.
You must print the propagation delay value and output transition time to the 6th decimal place.

```
g9 1 0.089355 0.145344 // output=1, propagation delay=0.089355, output rise time=0. 145344
g10 1 0.089355 0.145344
g8 1 0.079914 0.128288
g7 1 0.066956 0.112308
……
g6 0 0.041380 0.049434 // output=0, propagation delay=0.041380, output fall time=0.049434
g4 0 0.034014 0.047876
g2 0 0.022635 0.033406
g1 0 0.022268 0.031807
```

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

According to the delay information, compare the worst-case delay of all outputs. The largest one is the longest delay, and the smallest one is the shortest delay. Report them in different lines. To simplify the results, only the nets' names must be reported; the instances' name is not required. You must output the longest and the shortest delays and their corresponding delay path based on the library file *"**test_lib.lib**"*. If the instance has multiple input signals that have the same arrival time, you can output either one of them.

Name the output file as "***<student_id>_<case_name>_path.txt***". Also, you must print the value of the path delay to the 6th decimal place.

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

(d) 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 input pattern.

2. **Cell delay, transition time**

   After you get the logic output of each gate, look up the table according to input transition time and output capacitance. For the input gate, the input transition time is 0ns. For the input transition time, you must choose which input to use as the input transition time, which is related to the controlling value and the total delay. For the sensitization rule, please refer to note (g) for more details. Sort your results according to the instance number in ascending order and set the file name as "***<student_id>_<case_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 more than one input pattern, please report the delay information of each input pattern separated by 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)
```

Important Notice

(a) 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.

Boundary Conditions:

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

(b) To simplify the problem, there are several assumptions while calculating the worst-case delay, as below.

- If there are multiple paths from cell inputs to cell output, input transition time is defined by the input that arrives later.
- For the propagation delay, there are two tables (`cell_rise`, `cell_fall`).

  If the output value of this cell is 1, use the table `cell_rise`.

  If the output value of this cell is 0, use the table `cell_fall`.
- For the output transition time, there are also two tables (`rise_transition`, `fall_transition`).

  If the output value of this cell is 1, use the table `rise_transition`.

  If the output value of this cell is 0, use the table `fall_transition`.

  The successor cells will use this value as the input transition time.
- For wire delay, assume every wire between any two instances has 0.005ns delay.

(c) Print all values to the 6th decimal place except the worst-case output value of each instance.

(d)

4. Compile and Execution

- You must include "***Makefile***" for TA to compile your code.
- The name of the execution file after compiling must be "***<student_id>***"
- Your program must be able to receive commands in the following format.

```
$./<student_id>  netlist_file  -l  test_lib.lib
```

  For example:

```
$./311510369  example.v  -l  test_lib.lib
```

- Be sure that it can compile and execute successfully on our workstation.
- Do not print any words on the terminal when executing.

5. Grading Policy

(a) Correctness (70%)

You are encouraged to complete this assignment step by step. For each test case, you would get your grade for each step with a corresponding percentage of the total grade, as shown below.

- Step 1 result (10%)
- Step 2 result (20%)
- Step 3 result (20%)
- Step 4 result (20%)

(b) Runtime performance (22%)

The remaining 20% not included for the correctness of each step would be rated by the ranking of runtime among all students who pass all three steps. The ranking of runtime is based on the sum of execution time for each case. Please note that you will not get any grade for the runtime performance unless you pass all three steps.

For each case, the runtime limit is 300 seconds. It will be regarded as failed if you exceed the limitation. (Hint: Make good use of the container of C++ STL to avoid exceeding the runtime limit)

(c) Report (8%)

- Data Structure (2%)
- Algorithm flow chart (2%)
- Explanation of each flow (4%)
- Do not paste the code into the report!!
  - Pseudo-code is allowed
  - Otherwise -4 points

* Hidden cases will be used to evaluate your program.

* Violates file naming rules: -10 points

* Doesn't follow the requirements of output format: -10 points

* Print words on the terminal while executing: -10 points

* Plagiarism: -100 points

6. Submission Steps

(a) Put these files in a folder, the name of the folder is your student_id.

1. Source code(.cpp, .h), no naming restrictions for source code
2. Makefile
3. A simple report that explains the implementation details, name the report as "***<student_id>_report.pdf***"

(b) Use the command below to compress the folder in the Linux environment; the compressed file name should also be your student_id.

```
$tar cvf <student_id>.tar <student_id>
```

(c) Submit <student_id>.tar to E3 before the deadline.

* If you have any questions, please ask on the HW2 discussion forum of E3.