Programming Assignment 2:

Timing Analyzer with False Path Detection (Due: 11/24)

Problem Descriptions:

Given a gate-level netlist and a cell library, implement the three steps below to show the longest delay time and their corresponding paths under a given input pattern.

Delay Calculation:

Step1: Build the delay graph according to the given netlist

Step2: Calculate the delay and value of each node in the graph following topological order

Step3: Output the longest and delay and their corresponding paths

Input Format:

1. Flattened gate-level Verilog netlist

This file follows the standard Verilog gate-level syntax. In order to simplify the problem, the given netlist is flattened, i.e., only one module exists in the file. Only three kinds of cells (NAND2, NOR2, Inverter) will appear in this file. Sequential circuits are not necessary to be considered. However, your program should be able to deal with extra space (or TAB), extra new lines, block comments (/**/), and the comment lines starting with double slash (//) in this netlist file. Below is a simple example. Please be noted 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
```

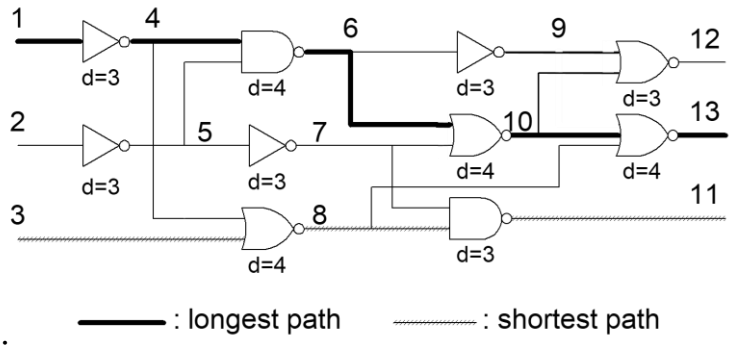

Figure 1 : An example of the gate-level netlist and its corresponding descriptions.

2. Input patterns

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 separate line.

Below is a simple example for the netlist in Figure 1.

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

3. Simplified .LIB file

This file is simplified from the standard .LIB format [1], which collects the timing and power information for each cell. In order to simplify the problem, we assume that the timing and power information of the same output is the same for all input path, i.e., A1->ZN and A2->ZN have exactly the same delay and power. Therefore, only one table is recorded for each output. The table index is defined in **`lu_table_template`**. The two indexes are total output loading and input transition time. The default units are also given in the file, which are ns (`timing`), pF (`capacitance`), V (`voltage`), and mA (`current`). Please be noted 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. Below is a simple example for the library information of an inverter. For simplicity, the numbers in this example may not be exactly the same as in the given library file. Please note that we would always use the same .lib file given to you for any verifications of your program.

```
lu_table_template(table10){
   variable_1 : total_output_net_capacitance;
   variable_2 : input_transition_time;
   index_1 ("0.0014,0.0030,0.0062,0.0125,0.0251,0.0504,0.1010");
   index_2 ("0.0208,0.0336,0.0600,0.1112,0.2136,0.4192,0.8304");
}
cell (INVX1) {
  pin(I) {
       direction: input;
      capacitance: 0.0109;
         }
  pin(ZN) {
           direction: output;
     capacitance: 0.0;
     internal_power() {
         rise_power(table10) {   // internal power when output is rising
           value(“0.0066,0.0090,0.0091,0.0094,0.0112,0.0147,0.0221”,\
    “0.0023,0.0012,0.0084,0.0098,0.0107,0.0146,0.0224”,\
    “0.0059,0.0089,0.0055,0.0094,0.0112,0.0138,0.0215”,\
    “0.0098,0.0133,0.0067,0.0074,0.0118,0.0139,0.0199”,\
    “0.0093,0.0038,0.0016,0.0059,0.0100,0.0131,0.0187”,\
    “0.0076,0.0065,0.0062,0.0059,0.0015,0.0130,0.0168”,\
    “0.0075,0.0059,0.0048,0.0019,0.0000,0.0000,0.0192”);
    }
         fall_power(table10)    // internal power when output is falling
           value(…);   // values are omitted in this example
      }   // end internal_power
```

```
        timing() {
            cell_rise(table10) {   // cell delay when output is rising
              value("0.0134,0.0160,0.0199,0.0225,0.0317,0.0402,0.0504",\
          "0.0153,0.0184,0.0230,0.0261,0.0371,0.0474,0.0599",\
          "0.0189,0.0224,0.0281,0.0326,0.0458,0.0599,0.0766",\
          "0.0260,0.0293,0.0330,0.0426,0.0545,0.0800,0.1039",\
          "0.0400,0.0405,0.0458,0.0581,0.0764,0.0993,0.1468",\
          "0.0663,0.0686,0.0733,0.0837,0.1082,0.1440,0.1890",\
          "0.1222,0.1244,0.1292,0.1386,0.1595,0.2086,0.2792");
          }
              cell_fall(table10)    // cell delay when output is falling
                value(…);   // values are omitted in this example
              rise_transition(table10)   // output rising time
                value(…);   // values are omitted in this example
              fall_transition(table10)   // output falling time
                value(…);   // values are omitted in this example
            }   // end timing
        }   // end pin
    }   // end cell
```

Output Format:

After executing your program, three output files should be generated for each step listed in the very beginning of this programming assignment. The output data for each file would be described in this section latter, and you should follow the same file names as below. (If the file name of the Verilog netlist file is "`c17.v`", the `<case_name>` would be "`c17`".)

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 1.

First, you have to construct the circuit graph according to the given netlist. In this step, we proceed to 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 **0.03 pF**. Finally, sort your results by the instance name with default comparators for string in c++ and set the file name as `<student_id>_<case_name>_load.txt`.

```
g1 0.0173
g2 0.0189
…
g8 0.03  // primary output loading
g9 0.03  // primary output loading
g10 0.03 // primary output loading
```

Step 2.

Before finding the worst-case delay path for the whole circuit, you have to determine the delay and transition time of each instance based on the circuit connection and given input patterns. In order to verify the results of this step, please also report the output value and

timing information of each instance in the following format. Each row shows an instance name, its output value, its corresponding worst-case delay, and its output transition time in order. Since you have the logic value of each input line, do remember to determine the delay of each instance based on path sensitization. Please sort your results by the instance name with default comparators for string in c++ and set the file name as `<student_id>_<case_name>_delay.txt`.

```
g1 1 0.0160 0.0074  // output=1, propagation delay=0.0160, output rising time = 0.0074
g2 0 0.0107 0.0048  // output=0, propagation delay=0.0107, output falling time = 0.0048
g3 0 0.0215 0.0093
…
```

If there are more than one input patterns, please report the delay information of each input pattern separated by a blank line.

```
g1 logic_output delay transition_time // the beginning of the first pattern
g2 logic_output delay transition_time
…
g10 logic_output delay transition_time
// blank line here !!
g1 logic_output delay transition_time  // the beginning of the second pattern
g2 logic_output delay transition_time
…
g10 logic_output delay transition_time
```

Step 3.

According to this delay information, the **longest sensitizable path** of each input pattern is reported in different lines. Take Figure 1 as an example, 3 paths are reported as below for 3 different input patterns. Please be noted that it is only an example to explain the format. The paths and delay values may not be correct. In order to simplify the results, only net names are required to be reported. Instance name is not required. Please set the output file name as `<student_id>_<case_name>_path.txt`. You must **output the longest delays and their corresponding sensitizable paths** based on the given library file *testlib.lib*. If there are more than one input patterns, please report the delay and path of each input pattern separately in different lines.

```
Longest delay = 15, the path is: n1 -> n4 -> n6 -> n10 -> n13
Longest delay = 14, the path is: n1 -> n4 -> n6 -> n9 -> n12
Longest delay = 12, the path is: n2 -> n5 -> n6 -> n10 -> n13
```

Requirement:

The program must be able to receive commands in following format.

```
$ <student_id>.o  netlist_file –p   input.pat    –l   testlib.lib
```

For example:

```
$ 309510165.o     c17.v         –p   c17.pat      –l   testlib.lib
```

For the boundary conditions, please set the input transition time of each primary input as **zero**, and set the output loading of each primary output as **0.03 pF**. Then, 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. In order to simplify the problem, there are several assumptions while calculating the worst-case delay as listed below.

1. While calculating the output value of each instance, assume it is in zero-delay mode. In other words, please **ignore the glitch** during logic transition and use the final value directly. You can get the input values from the outputs of your driving gates.
2. If there are multiple paths from cell inputs to cell output, please use the sensitization rules to find the sensitizable path and use it to determine the delay value.
3. For each sensitizable path, there are two tables (`cell_rise`, `cell_fall`) for the delay time when output is rising or falling. If the output value of this cell is 1, we use the table (`cell_rise`); otherwise, we use the table (`cell_fall`).
4. As to the output transition time, there are also two tables (`rise_transition`, `fall_transition`) for the transition time when output is rising or falling. If the output value of this cell is 1, we use the table (`rise_transition`); otherwise, we use the table (`fall_transition`). This value is used the input transition time of the successor cells.

Your grade depends on the correctness and runtime of your program. Please compress all the source code, the output files for the benchmarks, and a simple report explaining the implementation details, how to compile your program, and your comments to a compressed file. Then submit it to New E3 system before the deadline.

Grading Policy:

You are encouraged to complete this assignment step by step, and you would get your grade for each step with corresponding percentage of the total grade as shown below. The remaining 10 percent not included for the correctness of each step would be rated by the ranking of runtime among all students who pass all three steps. Please note that you would not get any grade for the runtime performance unless you pass all three steps.

1. Correctness of step 1 result (50%)
2. Correctness of step 2 result (30%)
3. Correctness of step 3 result (10%)
4. Runtime performance (10%)

Reference:

1. Liberty User Guides and Reference Manual Suite, https://media.c3d2.de/mgoblin_media/media_entries/659/Liberty_User_Guides_and_Reference_Manual_Suite_Version_2017.06.pdf