EE 5301 Fall 2024 Programming Assignment #1

Deadline: Nov 7th, before midnight.

As before, you may develop your code on the platform of your choice BUT your program should compile on a Linux platform using a Makefile.

Your solution should be capable of handling a netlist as large as 150,000 gates (but do not statically allocate that much memory by default: only get as much memory as needed by a circuit). I would strongly encourage you to make sure it works for a range of circuits: small (e.g., c17), medium (e.g., c7552), and large (e.g., b19_1). The benchmark files are the same as the ones provided in PA0 (file name: cleaned_iscas89_99_circuits.zip)

Reading Material:

Read the following chapters from [Bhasker]:

J. Bhasker and R. Chadha, "Static Timing Analysis for Nanometer Designs: A Practical Approach", DOI: 10.1007/978-0-387-93820-2_6, Springer Science + Business Media, LLC, 2009

- Chapter 2
- Pages 43-53 (skip Sec. 3.2.3), 56-59 (Sec. 3.3 & 3.3.1).
- I also recommend 3.4 up to (and not including) 3.4.2., but this is optional.

Optional reading for STL: (thanks to Arvind Karandikar @ U of M for suggesting the books) :

- Nicolai M. Josuttis, "The C++ Standard Library: A Tutorial and Reference", Addison-Wesley, 1999, ISBN: 0-201-37926-0.
- David Vanevoorde and Nicolai M. Josuttis, "C++ Templates, the Complete Guide", Addison-Wesley, 2003, ISBN: 0-201-73484-2.

Submission Process

Please submit your code via Canvas as one .zip file containing all source and header files. Your archive should include

- A file called README, listing a brief (one-sentence), logically-organized description of each file within the archive.
- A Makefile to compile your code into an executable: typing "make" should produce an executable called "sta". If you search, you will find plenty of resources online for building a makefile. Please do not submit the compiled executable file or the .o files or other debugging files.

Description of the project

In this project you will calculate the delay of a circuit by performing static timing analysis (STA) on it, similar to the topological traversal of a graph as discussed in class. Each node of the graph corresponds to each gate of the circuit, while each edge denotes a wire connection. The circuit format is the same as PA0.

The delay of each gate is a nonlinear function of its load capacitance and the input slew, and its calculation is summarized in the following figure.

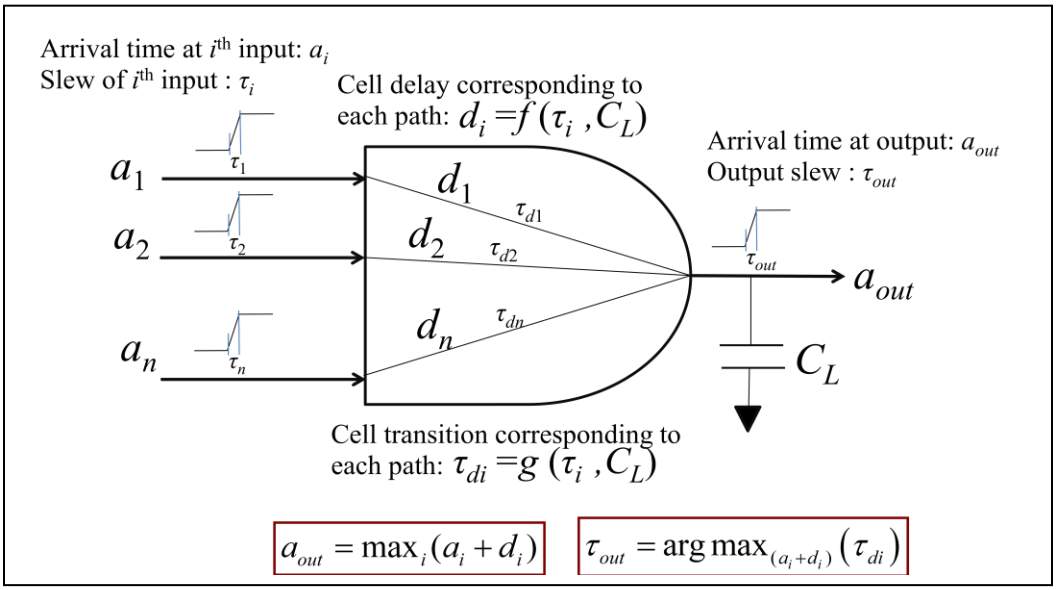

Figure 1. Delay (and output slew) of an *n*-input gate with a load capacitance of C_L

The functions, *f* and *g*, in Fig. 1, called the non-linear delay models (NLDMs), are typically provided by the foundry as look-up tables (LUTs). Each index of the LUT corresponds to the load capacitance (C_L) and input slew (τ_i) of a gate. The entries of the LUT represent either the delay (d) or the output slew values (τ_d) of the cell corresponding to (τ_i , C_L).

The NLDMs are stored in a *.lib file (called the "liberty" file). You are provided with one such file, "NLDM_lib_max2Inp". This file is an overly simplified version of a real liberty file, and currently only includes delay/slew values corresponding to 2-input gates.

A snippet from sample_NLDM.lib with explanation of each part is provided below:

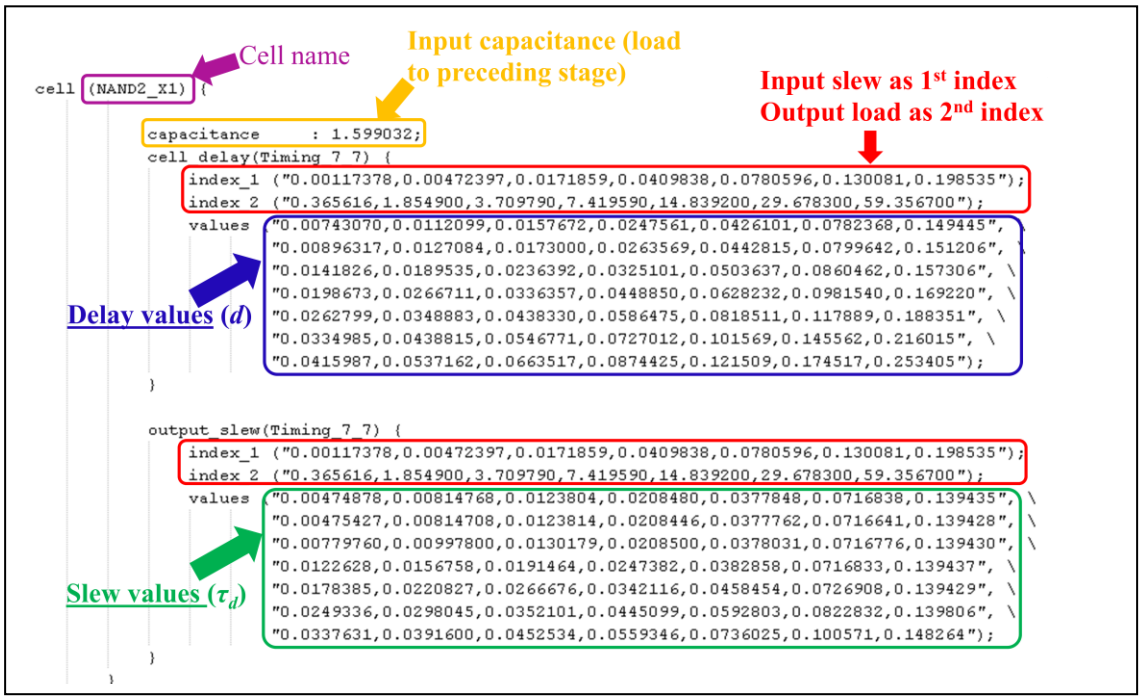

Figure 2. A snippet from the sample liberty file for a 2-input NAND gate.

To make your life easier, we are asking you to make the following simplifying assumptions:

- Since the liberty file only contains 2-input gates, for an *n*-input gate (with the same logic function) in a circuit under test, multiply the entry from the LUT with (*n*/2) to obtain the corresponding delay/slew values[1]. Obviously, we do not multiply the entry with (*n*/2) for 1-input (such as INV) and 2-input gates.
- The input capacitance of the *n*-input gate may be assumed to remain unchanged from that of the corresponding 2-input gate.
- No need to differentiate between rise and fall transitions and you can assume the same delay/slew LUTs for both. You can also assume the same delay/slew LUTs for all the input-output paths within the gate[2].

Deliverables

You have to perform STA on the circuit under test, find the slack at each gate output, and eventually, print the critical path. Your program should expect exactly two arguments: the first argument is the circuit file name, and the second argument would be the liberty file name. You can make the following assumptions:

- The arrival times at each primary input is 0, and input slew is 2 picoseconds (ps).
- The load capacitance of each gate is equal to the sum of capacitance of each of its fanout gates (i.e., ignore any wire capacitance).
- The load capacitance of the final stage of gates (which are simply connected to the primary outputs) is equal to four times the capacitance of an inverter from the liberty file.
- The required arrival time is 1.1 times the total circuit delay, and is the same at each primary output of the circuit.

The expected output of your program is outlined in **Appendix 1**.

For finding circuit delay, first perform the forward traversal of the circuit. At each gate, using the LUT, find the appropriate delays (and slews) of each path from its inputs to its output. The details of obtaining values corresponding to particular input slew and load cap for a path is provided in **Appendix 2**.
Next find the arrival time at the output of this gate by the "max" function as shown in Fig. 1. Repeat this until you reach the primary outputs. The maximum among the arrival times at all the primary outputs is the circuit delay.

For finding the slack at each gate, you need to perform a backward traversal of the circuit. Find the required arrival time at the output of each gate. The difference of the required arrival time and the actual arrival time (obtained from the forward traversal) is the slack at this gate.

To find the critical path of the circuit based on the slack values - start with the primary output with the minimum slack, and traverse backwards selecting the gates connected to this output with minimum slack, till you reach the primary input. If more than one primary output have the same value of the slack that is minimum, select any one randomly.

[1] In reality, the *n*-input gate would have its own LUT, but to make the assignment easier, we are asking you to make the simplifying assumption.
[2] In reality separate LUTs are provided for rise and fall transitions for each input-output path within the gate.

Handle flip-flops, labelled as DFF, as described in lecture by splitting them into a dummy input and output nodes. These split input and output nodes are treated identically to other input and output nodes. For example, b15_1.isc has the following node:

```
37 = DFF ( 107 )

// after splitting it becomes

INPUT ( 37 )

OUTPUT ( 107 )
```

After this splitting, there will be cases where the dummy input node (37 in this example) goes straight to an output. This should not cause any issues since the slack will be maximum in that case.

Finally, 10% of the grade is based on the runtime performance of the code, where each student is graded from 0-10 based on their percentile ranking of the code runtime. Since a lower runtime is preferred, the lower percentile rank gives a higher score. For example, if your submission runtime falls at or below the 10^{th} percentile, the score would be 10, but if it is above the 90^{th} percentile the score would be 0. Here are some tips to help improve performance:

1. Compile with optimization flags (https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html).
2. Use appropriate data structures like maps. The time to index them is nearly O(1) and is significantly faster compared to O(n) data structures like lists to find and retrieve information.
3. Ensure that objects like vectors and maps are being *passed-by-reference*. They should only be *passed-by-value* if required since they copy the entire data structure and can be slow.

As always, beware of premature optimization. So only worry about runtime performance once your code is working correctly.

The full grading rubric is written below:

- 20% - Program compiled.
- 10% - Input taken as command line argument.
- 10% - Output format is correct.
- 50% - Accuracy of output values.
- 10% - Program runtime performance based on percentile ranking.

Appendix 1: Sample expected output

The command "`./sta NLDM_lib_max2Inp c17_dummy.bench`" should produce a file called "ckt_traversal.txt", with each line as follows (contents within the dashed lines, **don't print the dashed lines in your code**):

```
Circuit delay: <val> ps

Gate slacks:
NAND-n10: <val> ps
NAND-n11: <val> ps
NAND-n16: <val> ps
NOR-n19: <val> ps
NAND-n22: <val> ps
NOR-n23: <val> ps

Critical path:
INP-n1, NAND-n10, NAND-n22 <or whatever the path that you find>
```

The `<val>` above is a placeholder for that value you will actually calculate in ps. Same with the critical path.

Note that the <> brackets are also a placeholder, so don't include them in your output. Also, the circuit file path MUST be input to the program as a command line argument and not be input from stdin/cin from terminal.

Appendix 2: Reading from an LUT

The LUTs in the sample liberty file are all 7x7 tables with 1st index corresponding to the input slew and the 2nd index to the load capacitance. The units are provided within this liberty file and they are nanosecond for time and femtoFarad for capacitance.

The input slews for which each LUT is provided are: ("0.001,0.004,0.017,0.041,0.078,0.130,0.198") ns
The load caps for which each LUT is provided are: ("0.365,1.855,3.709,7.419,14.839,29.678,59.357") fF

However, suppose you encounter a path in the gate for which input slew (τ) and load cap (C) do not exactly match any of the values for which the LUT is characterized. In that case, you will use a 2D interpolation. First you have to find the values of C_1, C_2, τ_1, τ_2 for which the values (delay/slew) are actually defined in the LUT as shown in Fig. 3, such that $C_1 \leq C < C_2$ and $\tau_1 \leq \tau < \tau_2$

	C_1	C_2
τ_1	v_{11}	v_{12}
τ_2	v_{21}	v_{22}

Figure 3. A part of the LUT of delay/slew, generalized by v_{ij} used to find the value at (τ , C).

Then the value, v, corresponding to (τ , C) for $C_1 \leq C < C_2$ and $\tau_1 \leq \tau < \tau_2$ is given by:

$$v = \frac{v_{11}(C_2 - C)(\tau_2 - \tau) + v_{12}(C - C_1)(\tau_2 - \tau) + v_{21}(C_2 - C)(\tau - \tau_1) + v_{22}(C - C_1)(\tau - \tau_1)}{(C_2 - C_1)(\tau_2 - \tau_1)}$$