EE 5301 Fall 2018 Mini Project #1

This mini-project consists of two parts, with an intermediate and a final deadline. You will be required to submit an intermediate submission for Phase-1 by October 17, but this will not be graded at that time. We will grade both parts together, and you are welcome to change your submission to Phase-2 at the time of the final submission (deadline October 24).

Ideally, you should write these programs using C++ (but C will be acceptable too). Your task may be eased by using the C++ Standard Template Library (STL): using STL is not required, but is highly encouraged. A simple outline of the STL is provided on the Moodle page, but you will find many online resources on this topic (e.g., SGI's Standard Template Library – click on the "Index" link. "Table of contents" is very useful too).

You may develop your code on the platform of your choice BUT your program should compile on a Unix/Linux platform (with gcc or g++). Test your code on the CSE Lab machine. This is to help you avoid any problems since all submissions will be run on the CSE Lab machine and graded.

Your solution should be capable of handling a netlist with 100,000 gates. 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., b15_C). Your code will be tested with ISCA'S 85 and ISCA's 89 Benchmark programs.

Submission Process Please submit your code via Moodle as one .zip/.rar/.gz/.tgz 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 files within the archive and a brief description of how you have approached the problem. If you haven't been able to implement in whole, state what was your approach.
- A Makefile to compile your code into an executable: typing "make" should produce an executable called "parser". If you search, you will find plenty of resources online for building a makefile. Please do not submit the compiled executable file.

Description of the mini-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 is described in *.bench format, an example of which is given below:

** Circuit c17.bench from ISCAS85 benchmark suite**

INPUT(n1)
INPUT(n2)
INPUT(n3)
INPUT(n6)
INPUT(n7)
OUTPUT(n22)
OUTPUT(n23)

n10 = NAND(n1, n3)
n11 = NAND(n3, n6)
n16 = NAND(n2, n11)
n19 = NAND(n11, n7)
n22 = NAND(n10, n16)
n23 = NAND(n16, n19)

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.

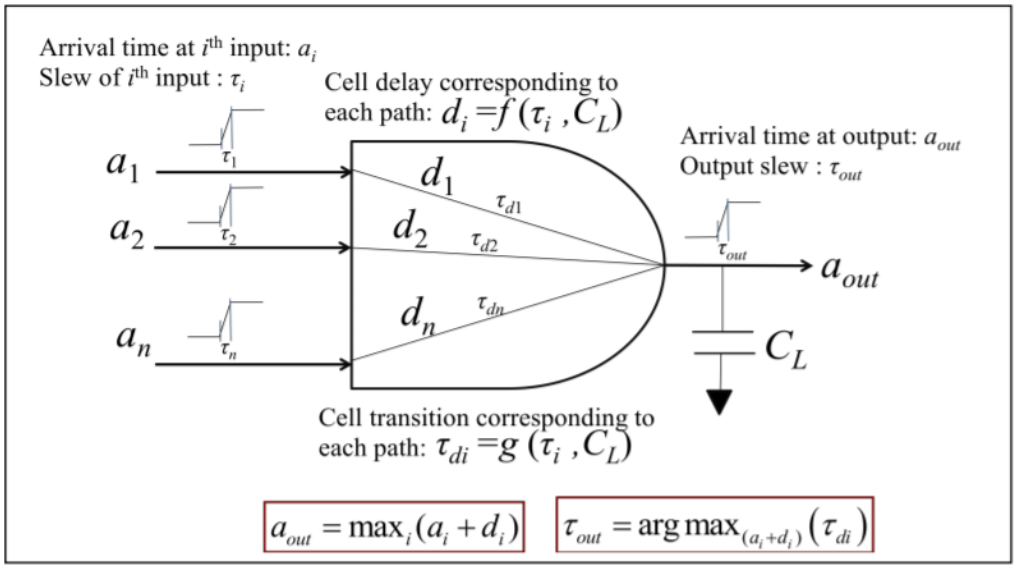

Figure 1. Delay (and output slew) of an *n*-input gate with a load capacitance of C_L

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, "sample_NLDM.lib". 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:

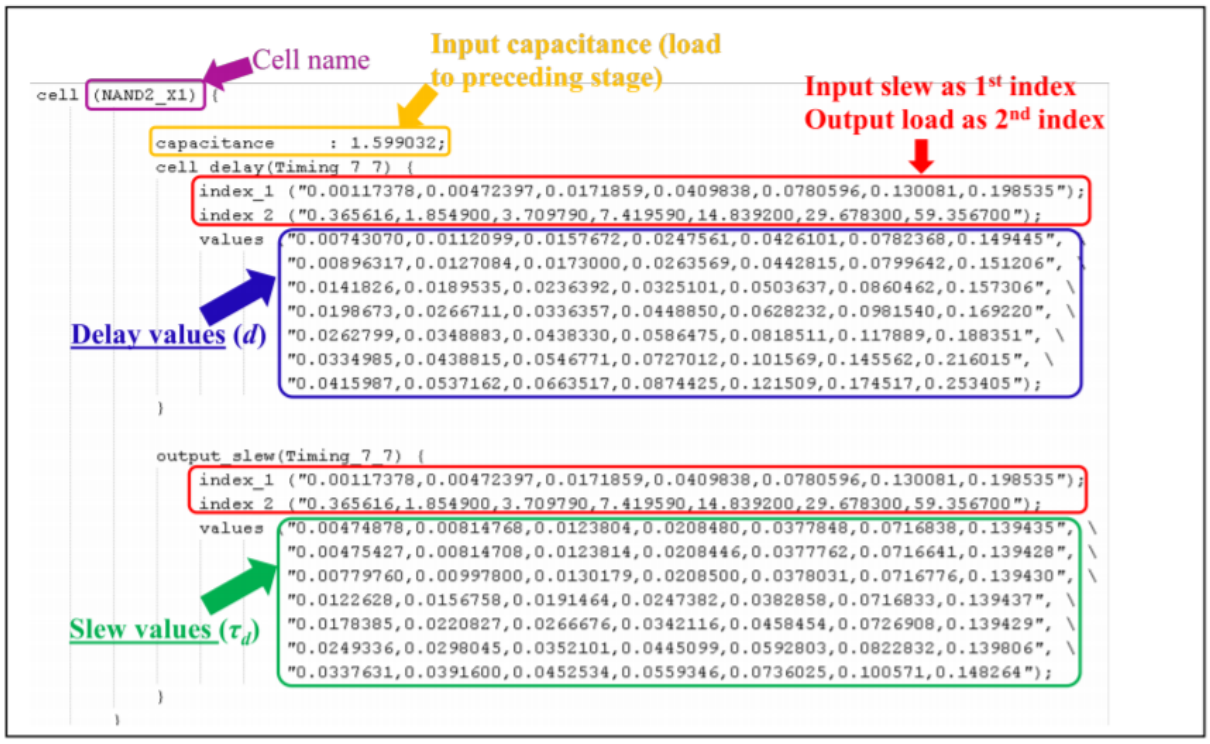

Figure 2. A snippet from the sample liberty file for a 2-input NAND gate.

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].
- 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].

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.

This mini-project is split into two phases:

Phase-1 (due October 17)

You have to parse the circuit as well as the liberty file, populate the necessary data structures, and produce the details of both the circuit and the NLDMs as necessary, using the keywords "read_ckt" and

“read_nldm”, respectively. For example, if the binary is called ./parser, then,

1) The command “`./parser read_ckt c17.bench`” should produce the details of the circuit as:

- Number of primary inputs and outputs
- Number and type of each gate,
- For each gate, print the number and type of its fanin and fanout gates

2) The commands for reading liberty file, sample_NLDM.lib, have two parts differentiated by two arguments in the command:

“`./parser read_nldm argument sample_NLDM.lib`”, `argument={“delays”, “slews”}`

2a) The command “`./parser read_nldm delays sample_NLDM.lib`” should produce the cell delays of the NLDM, i.e., print the cells and their corresponding LUTs of delay in a tabular form in a separate file.

2b) The command “`./parser read_nldm slews sample_NLDM.lib`” should produce the output slews of the NLDM, i.e., print the cells and their corresponding LUTs of output slews in a tabular form in a separate file.

Hint: Use `argc[]` and `argv[]` with inside main() to enable reading the arguments, `{read_ckt, read_nldm, delays, slews}`

The specific details of the expected output of this phase are provided in Appendix-1.

Your code will be tested on numerous other benchmark circuits some or all of which can be found in http://www.pld.ttu.ee/~maksim/benchmarks/iscas85/bench/.

Phase-2 (due October 24)

You have to perform STA on the circuit under test, find the slack at each gate output, and eventually, print the critical path. 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.

For this phase, your expected output is outlined in Appendix-2.

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-3. 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.

Finally, 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.

Example data structures which you can use for parsing the circuit netlist, the liberty file, and for performing STA, are provided in Appendix-4.

Appendices

Appendix-1: Sample expected output of Phase-1

The expected outputs are provided with respect to the following circuit (c17_dummy.bench), which has 5 primary inputs, 2 primary outputs, 4 two-input NAND gates, and 2 two-input NOR gates.

Note that you should index (or name) each gate by the name of its output wire.

** Circuit c17_dummy.bench adapted from c17.bench in ISCAS85 benchmark suite**

INPUT(n1)
INPUT(n2)
INPUT(n3)
INPUT(n6)
INPUT(n7)
OUTPUT(n22)
OUTPUT(n23)
n10 = NAND(n1, n3)
n11 = NAND(n3, n6)
n16 = NAND(n2, n11)
n19 = NOR(n11, n7)
n22 = NAND(n10, n16)
n23 = NOR(n16, n19)

The command "`./parser read_ckt c17_dummy.bench`" should produce a file called "ckt_details.txt", with each line as follows (contents within the dashed lines):

---

5 primary inputs
2 primary outputs
6 NAND gates
2 NOR gates

Fanout...
NAND-n10: NOR-n22
NAND-n11: NAND-n16, NOR-n19
NAND-n16: NAND-n22, NOR-n23
NOR-n19: NOR-n23
NAND-n22: OUTP
NOR-n23: OUTP

Fanin...

NAND-n10: INP-n1, INP-n3

NAND-n11: INP-n3, INP-n6

NAND-n16: INP-n2, NAND-n11

NOR-n19: NAND-n11, INP-n7

NAND-n22: NAND-n10, NAND-n16

NOR-n23: NAND-n16, NOR-n19

---

2) The command "`./parser read_nldm delays sample_NLDM.lib`" should produce a file, "delay_LUT.txt" with each line as follows (contents within the dashed lines):

---

cell: NAND2_X1

input slews: 0.00117378,0.00472397,0.0171859,0.0409838,0.0780596,0.130081,0.198535

load cap: 0.365616,1.854900,3.709790,7.419590,14.839200,29.678300,59.356700

delays:

0.00743070,0.0112099,0.0157672,0.0247561,0.0426101,0.0782368,0.149445;

0.00896317,0.0127084,0.0173000,0.0263569,0.0442815,0.0799642,0.151206;

0.0141826,0.0189535,0.0236392,0.0325101,0.0503637,0.0860462,0.157306;

0.0198673,0.0266711,0.0336357,0.0448850,0.0628232,0.0981540,0.169220;

0.0262799,0.0348883,0.0438330,0.0586475,0.0818511,0.117889,0.188351;

0.0334985,0.0438815,0.0546771,0.0727012,0.101569,0.145562,0.216015;

0.0415987,0.0537162,0.0663517,0.0874425,0.121509,0.174517,0.253405;

cell: NOR2_X1

input slews: 0.00117378,0.00472397,0.0171859,0.0409838,0.0780596,0.130081,0.198535

load cap: 0.365616,1.854900,3.709790,7.419590,14.839200,29.678300,59.356700

delays:

0.0136008,0.0161505,0.0205760,0.0292340,0.0462805,0.0801242,0.147594;

0.0142633,0.0167540,0.0211519,0.0298650,0.0470595,0.0810783,0.148696;

0.0202027,0.0226399,0.0267281,0.0350583,0.0519006,0.0857406,0.153365;

0.0282210,0.0316593,0.0372260,0.0469299,0.0634650,0.0965651,0.163573;

0.0381015,0.0422278,0.0490020,0.0610840,0.0815179,0.115113,0.181020;

0.0503320,0.0550648,0.0628653,0.0768972,0.101146,0.141009,0.207020;

0.0651817,0.0704989,0.0792584,0.0950330,0.122530,0.168656,0.242490;

... and so on for the other cells ...

---

2) The command "`./parser read_nldm slews sample_NLDM.lib`" should produce a file, "slew_LUT.txt" with each line as follows (contents within the dashed lines):

---

cell: NAND2_X1

input slews: 0.00117378,0.00472397,0.0171859,0.0409838,0.0780596,0.130081,0.198535

load cap: 0.365616,1.854900,3.709790,7.419590,14.839200,29.678300,59.356700

slews:

0.00474878,0.00814768,0.0123804,0.0208480,0.0377848,0.0716838,0.139435;

0.00475427,0.00814708,0.0123814,0.0208446,0.0377762,0.0716641,0.139428;

0.00779760,0.00997800,0.0130179,0.0208500,0.0378031,0.0716776,0.139430;

0.0122628,0.0156758,0.0191464,0.0247382,0.0382858,0.0716833,0.139437;

0.0178385,0.0220827,0.0266676,0.0342116,0.0458454,0.0726908,0.139429;

0.0249336,0.0298045,0.0352101,0.0445099,0.0592803,0.0822832,0.139806;

0.0337631,0.0391600,0.0452534,0.0559346,0.0736025,0.100571,0.148264;

... and so on for the other cells ...

Appendix-2: Sample expected output of Phase-2

The command "`./parser c17_dummy.bench`" should produce a file called "ckt_traversal.txt", with each line as follows (contents within the dashed lines):

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.

Appendix-3: 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:

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

For cases where you encounter a path in the gate for which the input slew (τ) and the load cap (C) are at the boundaries in the LUT use the adjacent values of C and τ for a 2D interpolation.

Appendix-4: Example data structures for the project

If you are using C++ for writing your code, the following are some suggested data structures, facilitated by the standard template library (STL). Look up at the resources and tutorials listed in the course webpage to understand the various template classes provided by STL.

Possible data structure to parse the netlist:

```
class node {
string name, outname; //name indicates cell type (nand, nor etc.),
outname denotes the output wire name
double Cload; //load cap of this node
```

vector<node*> inputs; //fanin nodes of this node

vector<node*> outputs; //fanout nodes of this node

vector<double> Tau_in; //vector of input slews (for all inputs to the gate), to be used for STA

vector<double> inp_arrival; //vector of input arrival times for input transitions (ignore rise or fall)

vector<double> outp_arrival; //vector of output arrival times, outp_arrival= inp_arrival + cell_delay; cell_delay will be calculated from NLDM

double max_out_arrival; //arrival time at the output of this gate using max on (inp_arrival + cell_delay)

double Tau_out; //Resulting output slew

...More functions/variables as required...

};

Possible data structure to parse the liberty file:

class LUT {

vector <string> Allgate_name; //all cells defined in the LUT

double*** All_delays; //array of tables (2D array) corresponding to each cell- may use a better representation like map<string, double**> which will map the cell name in string to the corresponding delay LUT

double*** All_slews; //same as All_delays, except it is now for the output slew values

double* Cload_vals; //corresponds to the 2nd index in the LUT

double* Tau_in_vals; //corresponds to the 1st index in the LUT- check definition in the LUT template provided at the beginning of the LUT dfinition

```
void assignarrays(string); //function to pass the NLDM file name from
which the above arrays can be populated

... More functions/variables as required...

};

LUT::assignarrays(string NLDM_file)

{

//define the arrays to be used during STA call later.

//also helps to simply assign the arrays so that a call to this
function will fetch the arrays, and you can easily print out the
details of this NLDM

...

}
```