

**Digital Design II**

**Project 2: Static Time Analysis**

**Date: 5/20/2018**

**Names:**

**Ahmed Refaay – 900141806**

**Lotfy Abdel Khaliq – 900143801**

**Noor Abdel Hamid - 900143227**

**Yahia Sherif - 900141148**

**1- Objectives:**

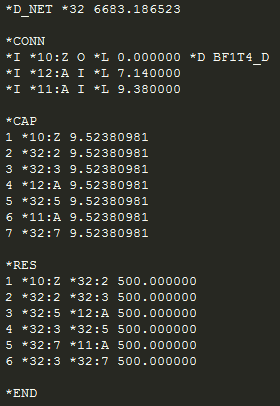
* Create a tool to calculate the delays in all timing paths and report slacks.
* Generate timing reports.

**2- Steps and Work division:**

* Parasitic delays and Slews from SPEF (Lotfy)
* Liberty File and Calculating the delays and output transitions (Refaay and Yahia)
* DAG creation (Noor)
* Creating and Analysing a small Example to test our work using Lab2 and Lab3
* Creating another Example that contains Sequential Elements
* Forward and Backward propagation to calculate AAT & RAT

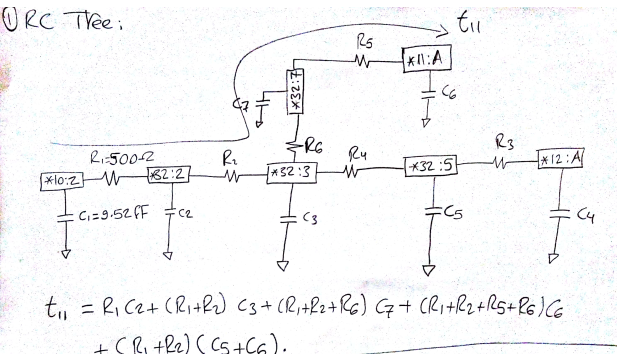
**SPEF file:**  
  
 Standard Parasitic Exchange Format (SPEF) is an [IEEE](https://en.wikipedia.org/wiki/IEEE) standard for representing parasitic data of wires in a chip in [ASCII](https://en.wikipedia.org/wiki/ASCII) format, and is fed to STA tool to do post layout Static Timing Analysis.

**Extracting the delays of the NETs from SPEF using Elmore delay:**

1- Sample NET from SPEF:

* (\*CONN) section contains all the branches and the start of the NET.
* (\*CAP) section contains all the Nodes in the NET and the capacitance of each Node
* (\*RES) section contains all the edges between the nodes and the resistance of that edge

2- We create a Graph, with Nodes from the (\*CAP section) and Edges from the (\*RES section).



3- Then we call the function get\_day() that recursively loops over all the branches of the NET and calculate the delay of each branch using Elmore delay.

**get\_delay() function:**

**Inputs:**

* Struct NET that contains all the data obtained from paring the SPEF file
* String new edge (the edge that we want to calculate the delay on.
* Double res\_so\_far (the accumulated resistance)
* Double delay\_so\_far (the accumulated delay)
* String f (the end of branch node)
* Map <string, double> Nodes\_delays that maps each Node to its delay (to be used ated to calculate the branch slew

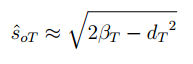
**Algorithm:** The function first gets the starting node which is either an input port of the output of a gate. Then it extracts the capacitance of the Node at the end of the edge, calculates the dlay (R\*C) then call the function again with the end node of the edge and look for the edges that have the same node as a source, and we also give it the updated delay and resistance to be used in the next node.

Base case: the function returns the final delay when it reaches the last edge where there are no nodes that has the end node of the previous edge as the starting node.

**Output:**

The function puts all the delays of each branch in the vector out\_branches. This output will be fed to our DAG to.

**Extracting the Slew of the NET from SPEF:**  
 The value of the output slew (so) on any given tap node T can be approximated

by a two-step process. First, we compute the output slew of the impulse response on T, which is approximated by

Where

**get\_slew() function:**



**Inputs:**

* Struct NET that contains all the data obtained from paring the SPEF file
* String new edge (the edge that we want to calculate the delay on.
* Double res\_so\_far (the accumulated resistance)
* Double slew\_so\_far (the accumulated delay)
* String f (the end of branch node)

**Algorithm:** The function first gets the starting node which is either an input port of the output of a gate. Then it extracts the capacitance and the delay of the Node at the end of the edge, calculates the Beta and calculate the slew, then call the function again with the end node of the edge and look for the edges that have the same node as a source, and we also give it the updated slew and resistance to be used in the next node.

Base case: the function returns the final slew when it reaches the last edge where there are no nodes that has the end node of the previous edge as the starting node.

**Output:**

The function puts all the delays of each branch in the vector out\_branches\_slew. This output will be fed to our DAG to.

**Liberty file:**

The .lib file is an ASCII representation of the timing and power parameters associated with any cell in a particular semiconductor technology. The timing and power parameters are obtained by characterizing the cell.

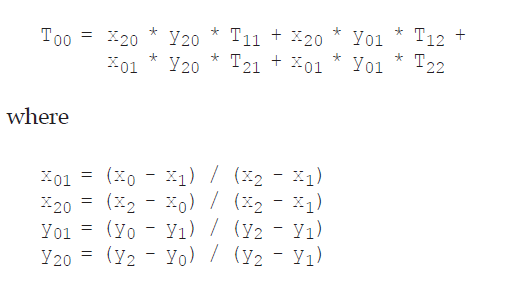
Cell-based delay calculation is modeled by characterizing cell delay and output transition time (output slew) as a function of input transition time (input slew) and the capacitive load on the output of the cell.

So basically, the first objective is to parse the liberty file and to fill 2 vectors one for the early and the other for the late. To achieve this part, we used the supplied parser of the liberty file with the contest education files, then we started to modified it to satisfy the sequential blocks compatibility. The main objective from parsing the liberty file is to have the the output delay and the output transition.

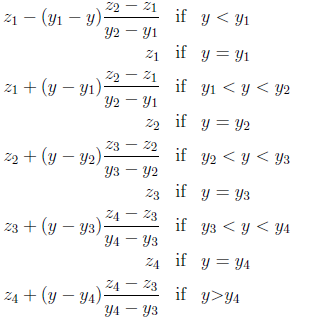
**Extracting the Output delay and output transition from the Liberty:**

A function calc is used to calculate the value of the output delay and the output transition one at a time. The function take 8 variables. The first one is the gate name that we want to look in the liberty file. We take this and search for it in the map of gates, which has all the gates inside the liberty file. The second variable is the In\_name, which is the input pin name. We have to check the name of the input pin so that we use the correct tables. The third variable is the Out\_name, which is the name of the output pin of the gate itself. The fourth variable is the input transition time, which is called also input slew. This value corresponds to the values in the index\_2 array. The fifth variable, is the capacitive load and it is index\_1.The next three variables are boolean variables, which represents we want a early or late model, rise or Fall,delay or slew. These values are inserted to the function to calculate the output delay and skew.

There are four scenarios for the input slew and cload. Which are both numbers are found in the Liberty file, one of them is found in the table and the last possibility is that we don't have any of the input numbers in the tables. In case of there is no numbers found in the tables, for the variable one and variable two cload and input slew, we use the technique called interpolation. Interpolation calculates the average value, which is in the middle between all four numbers.



The above formula shows clearly how to calculate the delay/ slew in the rise or the fall with the given variable T00. X0 and Y0 are the input variables to the functions and the rest X1, X2, Y1and Y2 are the values before and after of X0 and Y0. By the given formula the delay will be calculated. There is also a test case, where the values could be larger than the largest value in the array or it can be lower than the lowest possible number in the array. In this sense, we should take the largest two values or the smallest two values depending on the case we have. The second case is when we have one of the variables in the array and the other value is not found. So we have to use the extrapolation for the 1D, which is very similar to the interpolation, but with different formula.



In the extrapolation, there are 3 different cases. There is one variable found in the array and the other not found, so the variable not found in the array we use extrapolation on it to get the average value of the delay/slew. The first case that the input variable is smaller than the smallest value in the array, we then take the first and the second values in the array and apply to them calculations. We use the first equation to calculate the delay/slew. The second case that the variable is in between the values in the array, we the get the smallest difference between the variable and the array data to get the closest one to the input variable and then we take both the larger and the smaller values and use them to calculate the delay/slew. The last case, is that we have the input variable larger than the largest value in the array. We then use the largest two values in the array to calculate the delay/ slew.

The last possibility is that both numbers are found in the arrays in index\_1 and in index\_2, in that case we map to the single delay/ slew that is given in the table.

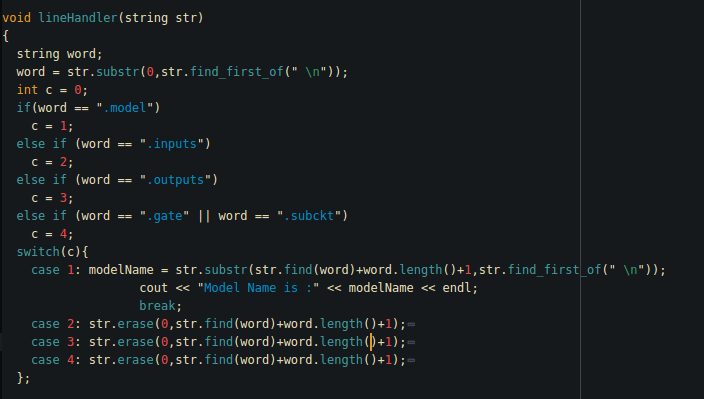
**.Blif file** (Berkeley Logic Interchange Format)

The goal of BLIF is to describe a logic-level hierarchical circuit in textual form. A circuit is an arbitrary combinational or sequential network of logic functions.

**The Objective from Blif Parser:**

To get the information about the gate level netlist and the connection among nodes which are representing inputs, outputs, wires. Therefore, we would be able to generate the Graph

**LineHandler Function (For BLIF):**

****

This function do two important jobs for the project at the **First Milestone.** First, it parses the .blif file extracting the gates with connections of each pin with the over all circuit. Then, it develops a graph connecting all those nodes with each others and keep track of its previous and next for traversal sake.

**However**

The .BLIF file has no benefit along with the .SPEF file to extract wire delays since delays in .SPEF is defined by an instance name of a gate and its pin and the .BLIF has no information about the instances so it would be hard to keep track of a path and match it with the .SPEF. Therefore, **we shifted to .V parser where all gates are defined by an instance name.**

**.V Parser & DAGcreator**

Same as the .BLIF we made a parser of the .v file where the syntax is different; however, the output is nearly same with difference that .V give us information of the existing inputs-outputs-wires before going into cells definitions. In addition, our main objective to extract instance names for the .SPEF delays identifications.

**The Objective**:-

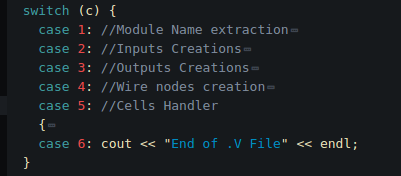
* Extract instance names for .SPEF to identify the delays.
* Pre-define wire names so this would make the Topological sort easier

**Functions:-**



**LineHandler** Function does the same functionality as in the .BLIF with different consideration for the syntax.

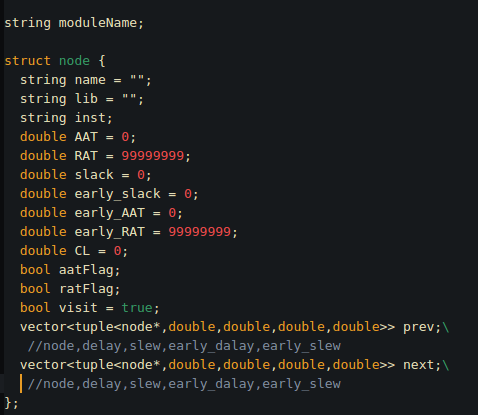
So as the LineHandler in the .BLIF it also initiated the creation of the graph among nodes given by the information we get from each cell line.



So this switch case works for the different Cases based on the line got from the file and ended with **;**

And starting from the second case it initiates nodes creation which is the critical part for Graph creator.

Before getting into the Details of **DAGcreator,** let’s study the structure unit of the Graph which is the **node**.



The node struct contains information about each pin in the DAG and is connected to other nodes through a vector of next and previous nodes. In addition, every next/previous node is saved along with four informations (late delay-late slew- early delay- early slew) in a tuple that represents the edge between two nodes.

**Algorithm of DAGcreator**

* Create all input/output/wire nodes
* Save each primary node into a map
* Loop over the cells
  + Each pin in a cell is connected to a previous primary node so create a new node for cell pin and associate an edge between the two nodes
  + Check For inputs/outputs pin of the gate to create edges with the internal gate nodes
  + Update the connections

- Call Function **SetupWireDelay** to set up delays between external nodes (wire delay)

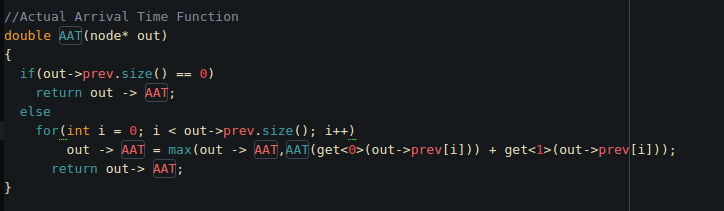
- Same with **SetupWireSlew** for the wires

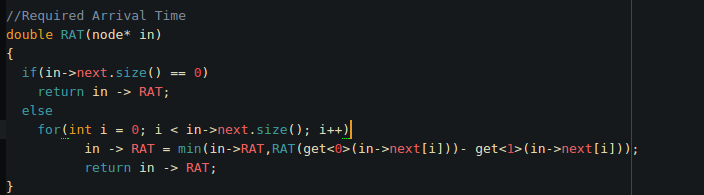
- In the function **ConnectInOut**, where we connect pins of the same gate instance based on the info from .LIB file, the delay and the slew of internal connected nodes are calculated as well from the .Lib file and update the edges on the nodes

- **AAT& RAT (Late):** they are recursive function to loop over the nodes and update it with the well-known formula - so as for the Early version)

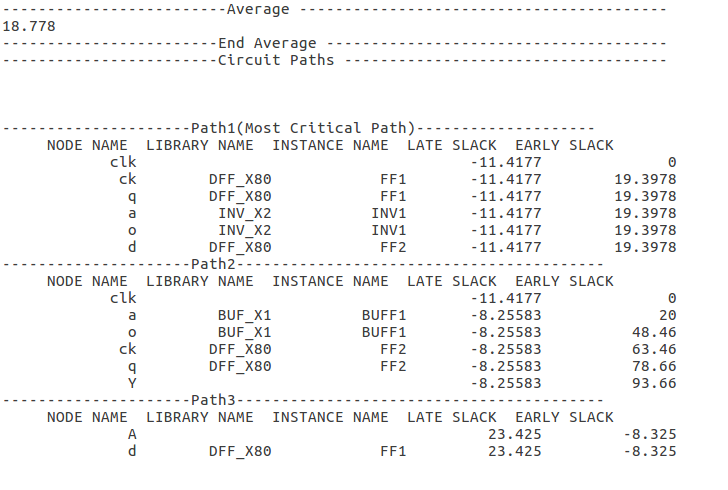
- **Traversal Function:** loop over the paths of the DAG and print all info for the timing report

AAT

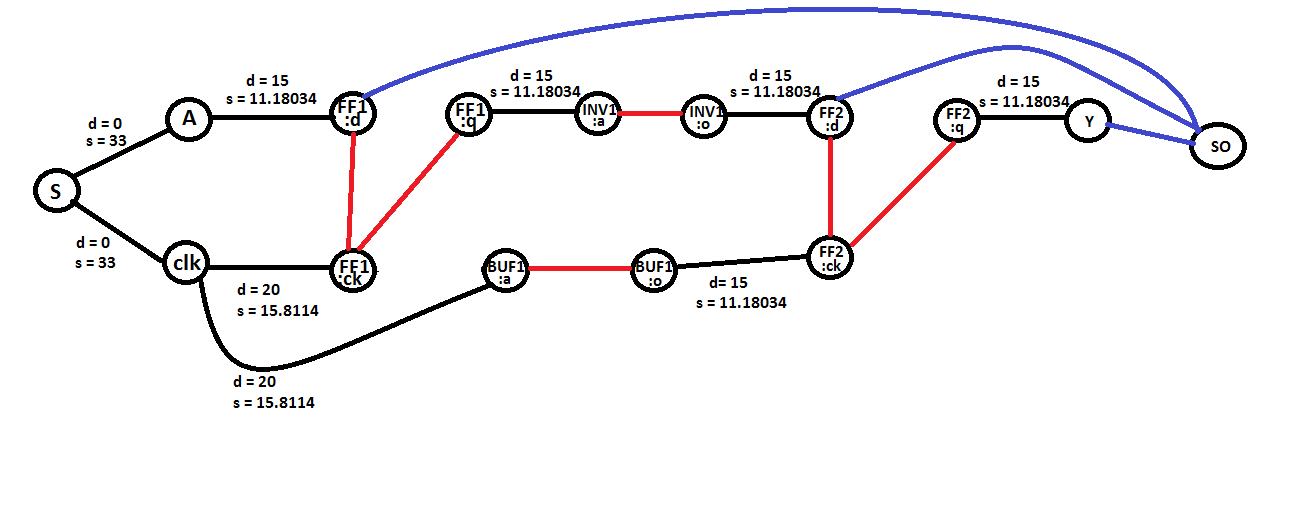


RAT

Timing report

****

**Our DAG**



**Limitations:  
  
1- SPEF must be an RC tree**

**2- SDC file has to follow our format:  
  
 c*reate\_clock -period 10 -waveform {0 6} -name clk clk***

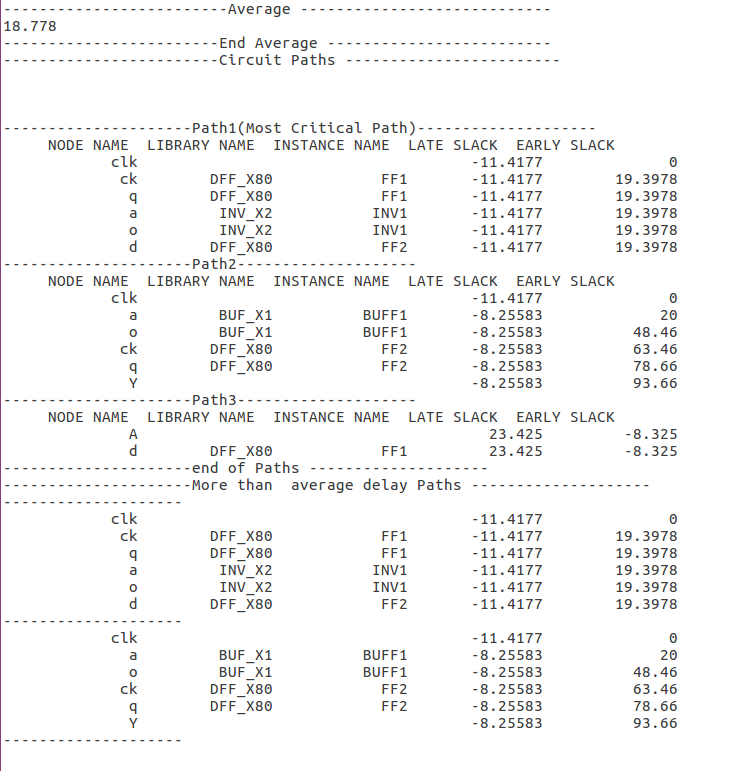
***set\_input\_delay -clock clk 0 [all\_inputs]***

***set\_output\_delay -clock clk 20 [all\_outputs]***

***set\_input\_transition -clock clk 33 [all\_inputs]***

***set\_load 35 [all\_outputs]***

Timing report

****