EE5302 VLSI Design Automation II

Final Project Report

Static Timing Analysis with Signal Integrity Effects

Team Members

1. Akhil kumar Modadugu x500: modad004

2. Kartheshwar Shanmuga Sundaram x500: shanm107

Introduction:

In semiconductors, metals are used as interconnects to connect various cells and blocks in an integrated circuit. As the technology node is shrinking these metal interconnects seems to affect the performance of the designs. The effects are mainly caused by crosstalk and noise in the deep submicron process technologies. So these effects have to be considered when verifying a chip for timing.

Static timing analysis (STA) is used to verify integrated circuits for timing closure, other methods like timing simulation can also be used and it can also check the circuit for functionality where STA cannot because it does not depend on input vectors. But the runtime of STA is very less compared to timing simulation. As STA cannot check for functionality it may be prone to false path and multipath cycle path errors. But we choose STA as it helps in analysing all the timing paths in a circuit with a simple approach and STA can handle crosstalk and noise whereas the timing simulation method cannot.

In this project we are developing a STA tool which can check the given gate level netlist for timing considering crosstalk effects. In general crosstalk effect can be simply modeled by considering coupling capacitance between nets, but this simple approach seems to be overly pessimistic. So in our project we have added a timing window concept to reduce pessimism by considering only coupling capacitance between nets which are actually switching at the same time. The STA tool was tested on different synthesized netlists (generated after PnR) and compared with the IC compiler's timing reports, results were almost similar with 12% error.

Throughout the project work was shared equally in different stages like reading research papers, formulating the final concepts to implementation, writing code and the final report.

Background:

Before we start talking about STA, we define the terminologies used:

Net: A net is a metal interconnect which connects a driver cell to one or many fanout nodes.

Cell delay: Time difference between the 50% point of the signal at the input of the cell, and the 50% point of the signal at the fanout node.

Best case delay (BCD): The least cell delay that can happen for an operating condition.

Worst case delay (WCD): The maximum cell delay that can happen for an operating condition.

Early arrival time(EAT)/Late arrival time(LAT): The EATs and LATs for fanout node j of a gate with n inputs are defined as follows:

$$EAT_j = \min_{i=1,2...n}(EAT_{input\ i} + BCD_{input\ i \rightarrow fanout\ j})$$

$$LAT_j = \max_{i=1,2...n}(LAT_{input\ i} + WCD_{input\ i \rightarrow fanout\ j})$$

Slew: The time for which a signal transitions from10% to 90% of its final value.

Arrival Time: The calculated path delay at a node.

Required Arrival Time: Expected arrival time at a node.

Slack: Slack is the difference between Required and actual arrival time.

STA Working Mechanism:

STA calculates path delays in an integrated circuit. A path may contain many nets and cells. The path delay is the sum of all those cell and interconnect delays. Cell delay is a function of input slew and output capacitance.

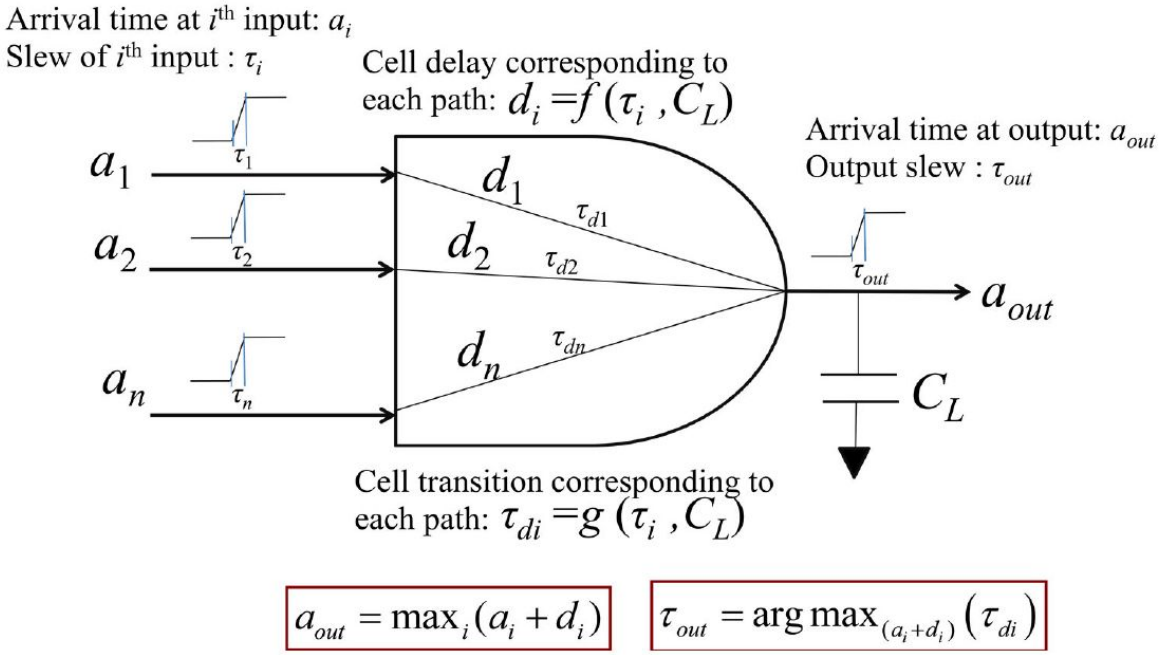

Figure 1. Delay (and output slew) of an n-input gate with a load capacitance of CL

Figure1 shows the calculation of output delay and slew for a 3input AND gate. Interconnect delay is the function of resistance and capacitance of interconnect metal. For simplicity our project considers only capacitance of wire for calculation of interconnect delay.

Since the transistor gates are nonlinear we will be using Nonlinear delay models(NLDM) liberty files for calculating output delay and slew.

```
timing () {
    related_pin : "A2";
    timing_sense : "positive_unate";
    cell_rise ("del_1_7_7") {
        index_1("0.016, 0.032, 0.064, 0.128, 0.256, 0.512, 1.024");
        index_2("0.1, 0.5, 1, 2, 4, 8, 16");
        values("0.0823482, 0.0840918, 0.0861906, 0.0901548, 0.0974833, 0.1102061, 0.1323354", \
        "0.0847795, 0.0865314, 0.0886387, 0.0926054, 0.0998136, 0.1126288, 0.1347947", \
        "0.0897225, 0.0914448, 0.0935254, 0.0974777, 0.1046771, 0.1173817, 0.1394445", \
        "0.0982357, 0.0999579, 0.1020366, 0.1059776, 0.1132079, 0.1260104, 0.1480864", \
        "0.1068003, 0.1085425, 0.1106509, 0.1148697, 0.1224585, 0.1353250, 0.1578208", \
        "0.1100673, 0.1118411, 0.1140116, 0.1181778, 0.1258911, 0.1403494, 0.1634643", \
        "0.0947546, 0.0970092, 0.0993052, 0.1037681, 0.1127003, 0.1273594, 0.1539119");
    }
```

Figure2. Snippet of saed32_lvt_ss0p95v125c NLDM file of a 2 input AND gate

For a gate, the NLDM file contains timing tables for rise and fall delay and also slew for rise and fall transition for every pin of the gate. Figure2 shows the timing table for pin A2 of the AND gate. Index_1 is the input transition time and Index_2 is output capacitance. So for a particular gate for its input transition time and output capacitance we can look up those values in index_2 and index_1 respectively and match the coordinates with that of ‘values’ tables to get the output delay. Generally our gate may have different values than one shown in index 1 and 2, so we use intrapolation and extrapolation to get delay values.

Crosstalk:

Crosstalk is the capacitive coupling between nets that can lead to a logical failure.

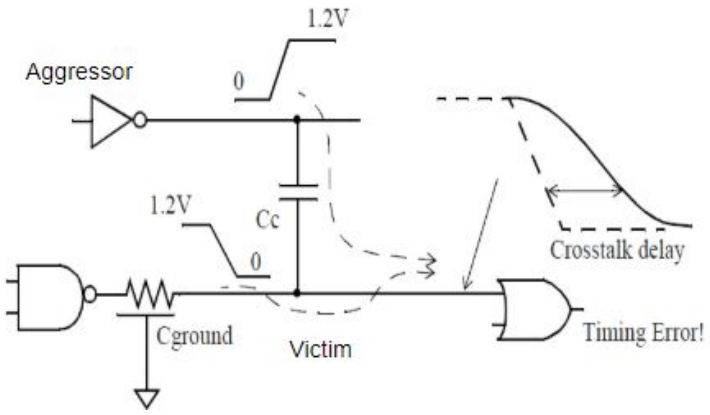

Figure3. Crosstalk effect by aggressor net

Worst case coupling capacitance case is shown in figure3, here both the aggressor and victim nets are switching at the same time and in opposite directions resulting in a delayed signal at output than expected. So we have to incorporate coupling capacitance into STA for accurate results. The impact of coupling capacitance on delay is usually estimated by scaling the coupling capacitances (often by a factor of 2) and modeling them as grounded. This coupling capacitance value is extracted from SPEF file which is generated after signoff by the IC-compiler or Innovus tool.

```
*CAP
1 *72:101 0.9662583
2 *72:102 0.920227
3 *72:103 0.464347
4 *72:104 0.2307457
5 *72:105 0.06768111
6 *72:106 0.003635078
7 *72:103 *166:11 0.08614472
8 *72:102 *166:11 0.6322755
9 *72:101 *166:15 0.6262416
0 *72:100 *166:15 0.692453
11 *72:99 *166:14 0.7372847
12 *72:98 *166:14 0.1249425
13 *72:97 *166:14 0.2114699
14 *72:96 *166:14 0.2114699
15 *72:95 *166:14 0.1751064
16 *72:94 *166:14 0.2654639
17 *72:93 *166:14 0.6754645
18 *72:92 *166:9 0.7159733
```

Figure4. Snippet from SPEF file

In Figure4 *72 and *166 are the alias for some net names. In the first line *72:101 0.9662583 tells that at node 101 on net *72 there is a capacitance of 0.9662583fF to ground. And from line 7 it shows coupling capacitance between net *72 and *166 at different nodes.

This simple approach has been shown to be overly pessimistic because we are assuming that all the coupled nets in a SPEF file are always coupled meaning always switching at the same time.

So in our project we include the Timing window concept to eliminate pessimism. Timing window of a net is the time period at which the net can switch. We use EAT's and LAT's to calculate timing windows. But for the initial run when calculating timing window for aggressor, the window may depend on the victim's timing window and vice-versa leading to a classical chicken egg problem. We overcome this problem by starting with a worst case assumption for the switching windows. Under this assumption, we assume all the coupled nets mentioned in the SPEF file as coupled. This can be done by initializing all EAT to 0 and all LAT to infinity. In the next iteration we will know which nets are actually switching at the same time using their timing windows and now we will only consider coupling capacitance for those nets which are switching at same time for delay calculation. This process is repeated until there is no shrink in timing windows.

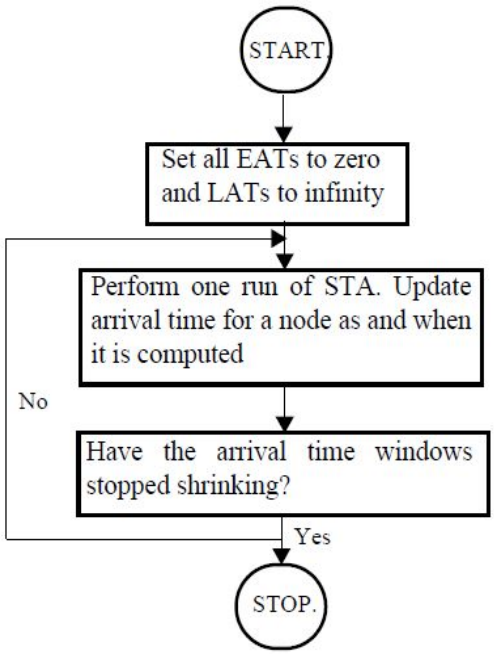

Figure5. Flowchart of Timing window algorithm

Implementation details:

In our project we used ISCAS85 benchmark verilog files and used Synopsys DC and IC compiler to synthesis and perform APR(Automatic place and route) respectively. By doing this we get a synthesized gate level netlist and SPEF max and min files generated for different operating conditions. We use these files as input to our STA tool which returns the critical path(least slack path) of the circuit.

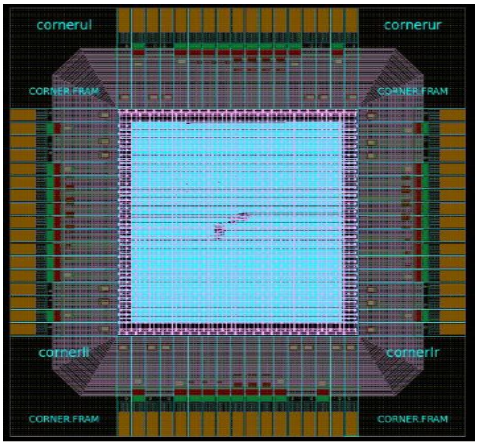

Figure6. Sample cell after APR using IC and DC Compiler

Tool Implementation:

The STA tool has been written in C++ language. The different functions which are present in our code are as listed below:

1) parseVerilogCktFile
2) parseLibFile
3) parseCktFile
4) parseSpefMin
5) parseSpefMax
6) timingWindow
7) topoSortUtil
8) topoSort
9) calculateOutputArrivalTime
10) delay_i
11) sumCapacitance

12) findmin

13) maxArrivalTime

14) printRiseCriticalPath

15) printFallCriticalPath

16) main

All the functions are explained in detail as follows.

parseVerilogCktFile

Arguments: char* argv[]

Return value: void

Purpose:

This function takes the placed verilog circuit file given in argv[5] and converts into a format which is easy to parse through. It reads the circuit file and creates the file "ParsedVerilogCktFile.txt" which has following information:

- If a gate is input then an entry in the "ParsedVerilogCktFile" is made as : INPUT "gatename"
- If a gate is output then an entry in the "ParsedVerilogCktFile" is made as : OUTPUT"gatename"
- If the gate is a standard cell or IOPAD then an entry in the "ParsedVerilogCktFile" as : "output node" = "standard cell name" "inputs to standard cell followed by space"

parseLibFile

Arguments: int argc, char* argv[]

Return value: void

Purpose:

The function takes Liberty files as input and parses all those files to store in a data structure. Liberty files used are saed32_lvt_ss0p95v125c, saed32_lvt_ss0p95v125c, saed32_lvt_ss0p95v125c and saed32io_wb_ss0p95v125c_2p25v. From these files for a particular gate we extract input capacitance, timing tables for each pin and index values for input transition and output capacitance. It updates the inverting flag which describes if a gate is inverting or not.

parseCktFile

Arguments: char* argv[]

Return value: void

Purpose:

This function is used to store the fan-in and fan-out information for the circuit file. It reads the "ParsedVerilogCktFile" and updates the map data structure "ckt_vector" based on

- If it's an INPUTthen sets the arrival time at input as 0 and input slew as argv[9]
- If its OUTPUT then it stores node name and makes fanout size as zero
- If it's any other standard cell then stores the node name and stores all the inputs in the fan-in vector. It also pushes back the node name in the input's fanout vector

parseSpefMin

Arguments: char* argv[], map<string, circuit > & ckt_vector

Return value: void

Purpose:

It is used to parse the SPEF min file (argv[6]) and update the capacitance in the ckt_vector.

The keyword “NAME_MAP” contains the alias names for nets which will be used in the file. For each net name the function creates an element in map<int, spef> SpefMap which contains the key as the alias name has entries: node name, coupling capacitance, load capacitance. String streams are used for parsing through the file. If the “first-word” while parsing is “D_NET” then that line has the following format --

D_NET “node name” "total capacitance”.

The lines following that will contain information about the node name. The “first-word” having the keyword “CAP” will contain the capacitance information associated with the node name. The format in which the capacitance information is present is as below

“Line_number” “Node1” “Node2” “Value”

It denotes that there is a coupling capacitance of value “Value” between “Node1” and “Node2”

If “Node2” is not there in a line it means its a ground capacitance of value “Value”

These information are stored in SpefMap.

After parsing through the entire file the ckt_vector is updated with the capacitance value with load capacitance without coupling capacitance.

parseSpefMax

Arguments: char* argv[], map<string, circuit > & ckt_vector

Return value: void

Purpose:

It is used to parse the SPEF max file (argv[7]) and update the capacitance in the ckt_vector.

The file format of SPEF max is similar to SPEF min and is parsed in a similar fashion.

After parsing through the entire file the ckt_vector is updated with the capacitance value with load capacitance with coupling capacitance multiplied by a factor of 2.

timingWindow

Arguments: char* argv[], map<string, circuit > & ckt_vector, map<string, circuit >&

ckt_vectorEAT, map<string, circuit >& ckt_vectorWorstLAT

Return value: void

Purpose:

This function takes in the EAT, LAT which is computed for all nodes earlier and updates the coupling capacitance value for each node to reduce pessimism. If for a node the timing window of the aggressor and the victim does not overlap then coupling capacitance is not considered, if its overlapping then capacitance is updated as coupling capacitance multiplied by a factor of 2.

topoSortUtil

Arguments: string v, queue<string>& Queue, map<string, circuit > & ckt_vector

Return value: void

Purpose:

This function is used to generate the order of processing of nodes in the ckt_vector. The algorithm is similar to Depth First Search where we start from a vertex, we first process it and then recursively call DFS for its adjacent vertices. Here we use a temporary stack. We don't process the vertex immediately, we first recursively call topological sorting for all its adjacent vertices, then push it to a stack. Finally, the contents of the stack gives the order of processing. A visited map is used to keep track if a node is already visited or not.

topoSort

Arguments: string v, queue<string>& Queue, map<string, circuit > & ckt_vector

Return value: void

Purpose:

This function is used to compute req_arr_time_rise, req_arr_time_fall, slack_rise, slack_fall, output_arrival_rise, output_arrival_fall for every node in the ckt_vector. The topoSortUtil gives order of processing nodes. For each node the output arrival time is computed using the function calculateOutputArrivalTime. If the fan-out vector of a node is zero then req_arr_time_rise and req_arr_time_fall for that node is made as argv[7] ie clock. If the fan-out size is non-zero then the req_arr_time_rise is computed by computing the minimum of (req_arr_time_rise - gate_delay_rise) at each fanout point using the findmin function. Similarly the req_arr_time_fall is computed by computing the minimum of (req_arr_time_fall - gate_delay_fall) at each fanout point using the findmin function. The slack_rise and slack_fall is updated accordingly. The whole process is repeated till the stack is empty.

calculateOutputArrivalTime

Arguments:string NodeName, map<string, circuit > & ckt_vector

Return value: void

Purpose:

This function is used to compute the output arrival time for a node. If the node is input then the output arrival time for the input node is zero. For any other node the delay through the gate at a particular pin is computed using the “delay_i” function which takes input slew, output capacitance, and pin under consideration as parameters and outputs delay for rise and fall

transitions. The output arrival time for rise transition is set as maximum of (fanin's output arrival time for rise transition + delay for rise transition) across all pins, if the gate is non-inverting. If the gate is inverting, then the output arrival time for rise transition is set as maximum of (fanin's output arrival time for fall transition + delay for rise transition) across all the pins of the gate. A similar strategy is followed for fall transition.

delay_i

Arguments: string NodeName, double slew_in_rise,double slew_in_fall, double cl, int Num, string str

Return value: void

Purpose:

This function is used to calculate the delay through a gate. The input slew for rise transition and fall transition, load capacitance is given as input to the function. The NldmMap gives the indexes for the NodeName, 7x7 tables for slew and delay for each pin in the gate. The pin which is to be considered is given by "Num". 2D intrapolation is used to compute the rise delay(Delay_rise), fall delay (Delay_fall) through the gate. The slew for rise transaction and fall transaction is stored in slew_vec_rise and slew_vec_fall.

sumCapacitance

Arguments: vector<string> fanout, map<string, circuit > & ckt_vector

Return value: double

Purpose:

This function returns the sum of all the pin capacitance of the fanout of a node. The fanout vector contains the gates which the current node is connected to. The NldmMap contains the pin

capacitance for each standard cell. An iterator searches for the fanout gate name in the vector and extracts the pin capacitance from the map. This process is repeated for all the fanout points and the values are added. The sum of all capacitance is returned.

findmin

Arguments: vector<double>comp

Return value: double

Purpose:

This function is used to return the minimum element in the comp vector.

maxArrivalTime

Arguments: vector<double> arrival_time

Return value: double

Purpose: This function is used to return the maximum element in the arrival_time vector.

printRiseCriticalPath

Arguments: string value, map<string, circuit > & ckt_vector

Return value: void

Purpose:

This function is used to print the critical path for rise transition. “value” indicates the output node which has the minimum slack for a rise transaction and is added to a vector. The inputs of this node which has the minimum slack for rise transition is chosen and added to the vector. The whole process is repeated until we reach any input node. The vector indicates the critical path for rise transaction, and it is printed into the output file.

printFallCriticalPath

Arguments: string value, map<string, circuit > & ckt_vector

Return value: void

Purpose:

This function is used to print the critical path for fall transition. “Value” indicates the output node which has the minimum slack for fall transaction and is added to a vector. The inputs of this node which has the minimum slack for fall transition is chosen and added to the vector. The whole process is repeated until we reach any input node. The vector indicates the critical path for fall transaction, and it is printed into the output file.

Main

Arguments: int argc, char* argv[]

Return value: int

Purpose: Create the flow for execution of functions. Call “parseVerilogCktFile” to create a circuit file that is easier to read. Compute EAT by calling functions: parseCktFile, parseSpefMin, topoSort. Compute LAT by calling functions parseCktFile, parseSpefMax, topoSort. Compute actual critical path using EAT and LAT values computed by calling parseCktFile, timingWindow, and topoSort. Print values into file by using printRiseCriticalPath, printFallCriticalPath functions.

Results:

The C++ code contains 1500 lines, the gnu compiler is used for compiling. The code has been tested on the CSE lab machine and is bug free.

The designs used for testing are taken from ISCAS'85 benchmark and the table shown below describes the results

File Name	Data Arrival Time at critical path (in ns)		Percentage of difference between tool and C++program (%)
	Computed using tool	Computed using our C++ program	
c432	3.78	4.09089	8.22
c499	4.79	4.47018	6.67
c880	4.81	5.38566	11.96
c1908	4.80	4.84994	1.04

Table 1: Results for design

After running the code, the file "Output.txt" is generated which contains the following details:

- Data Arrival time for rise transaction
- Data Arrival time for Fall transaction
- Gate slack for every node in design
- Critical path for Rise transition
- Critical path for Fall transition

```
The NLDM Liberty files which are used are as below:
D:\SEM1\VDA-Kia\Assignment\PA1\files\files\NLDM\saed32_hvt_ss0p95v125c.lib
D:\SEM1\VDA-Kia\Assignment\PA1\files\files\NLDM\saed32_lvt_ss0p95v125c.lib
D:\SEM1\VDA-Kia\Assignment\PA1\files\files\NLDM\saed32_rvt_ss0p95v125c.lib
D:\SEM1\VDA-Kia\Assignment\PA1\files\files\NLDM\saed32io_wb_ss0p95v125c_2p25v.lib
The verilog Circuit File under test is ----D:\SEM1\VDA-Kia\Assignment\PA1\files\files\c1908\c1908.vg

The SPEF files for RC calculation are as below
D:\SEM1\VDA-Kia\Assignment\PA1\files\files\c1908\c1908.spef.min
D:\SEM1\VDA-Kia\Assignment\PA1\files\files\c1908\c1908.spef.max

The clock which is used for considerstion is -- 5 ns

The clock which is used for considerstion is -- 5 ns

Circuit delay
Rise delay: 4.84994ns
Fall delay: 4.81567ns

Gate slacks :
N2899   D4I1025_EW  Rise :  0.292192 ns
N2899  D4I1025_EW  Fall :  0.27772 ns
n599   NBUFFX32_HVT  Rise :  0.292192 ns
n599  NBUFFX32_HVT  Fall :  0.27772 ns
n99   AND2X4_RVT  Rise :  0.292192 ns
n99  AND2X4_RVT  Fall :  0.27772 ns
n401   XOR2X2_LVT  Rise :  0.292192 ns
n401  XOR2X2_LVT  Fall :  0.27772 ns
n399   INVX1_RVT  Rise :  1.2644 ns
n399  INVX1_RVT  Fall :  1.15525 ns
n400   NAND2X4_RVT  Rise :  0.292192 ns
n400  NAND2X4_RVT  Fall :  0.27772 ns
N2892   D4I1025_NS  Rise :  0.397356 ns
N2892  D4I1025_NS  Fall :  0.410913 ns
n597   NBUFFX32_HVT  Rise :  0.397356 ns
n597  NBUFFX32_HVT  Fall :  0.410913 ns

Critical path RISE:
INPUT N49,  I1025_EW n510,  IBUFFX8_RVT n511,  INVX16_HVT n60,  NAND2X4_RVT n389,  XOR3X2_RVT n550,
INVX8_LVT n393,  AND2X4_LVT n502,  NBUFFX32_HVT n601,  D4I1025_NS N2886

Critical path FALL:
INPUT N49,  I1025_EW n510,  IBUFFX8_RVT n511,  INVX16_HVT n60,  NAND2X4_RVT n389,  XOR3X2_RVT n550,
INVX8_LVT n393,  AND2X4_LVT n502,  NBUFFX32_HVT n601,  D4I1025_NS N2886
```

Figure7. Output.txt file for c1908 design

Conclusion and future work:

To summarize we have included cross talk and timing windows concepts to the basic STA process to increase the accuracy. We have implemented the following:

- Created RTL to GDS flow for ISCAS'85 benchmark design
- Parsed NLDM liberty files
- Parsed post PnR netlist files
- Extracted parasitics from SPEF file
- Calculated delay through gate based on 2D intrapolation of NLDM file
- Calculated rise and fall delay independently based on inverting/non-inverting gates
- Computed Timing windows for reduced pessimism
- Extracted critical path for rise and fall transition

The results obtained have been compared with the tool's timing report files which seems satisfactory.

For calculating interconnect delay we have only considered wire and coupling capacitance, in future we would like to extend this project to consider all the RC's. Model order reduction can be used to reduce the RC tree to a pi model and then use the elmore delay concept to calculate interconnect delays. We can extend this also to include sequential elements where we have to break the design into different structures- inputs, outputs, reg2reg and perform timing analysis.

References:

[1] R. Arunachalam, K. Rajagopall and L. T. Pileggi, "TACO: timing analysis with COupling," Proceedings 37th Design Automation Conference, Los Angeles, CA, USA, 2000, pp. 266-269.

[2] P. D. Gross, R. Arunachalam, K. Rajagopal and L. T. Pileggi, "Determination of worst-case aggressor alignment for delay calculation," 1998 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (IEEE Cat. No.98CB36287), San Jose, CA, USA, 1998, pp. 212-219.

[3] Static Timing Analysis for Nanometer Designs: A Practical Approach.
Book by Jayaram Bhasker and Rakesh Chadha

[4] EE5301, EE5302 Lecture slides