PhD Thesis

# Statistical characterization, analysis and modeling of 

 speed performance in digital standard cell designs
## subject to process variations

Mastrandrea Antonio

Cycle XXVI (2010-2013)


Sapienza University of Rome
(DIET) Electronic Engineering Faculty

## Supervisor

Prof. Dr. Olivieri Mauro

## Second supervisor

Prof. Dr. Irrera Fernanda

## Date of the graduation

13.12.2013 first discussion; 10.3.2014 last discussion

## Contents

Abstract ..... 2

1. Propagation Delay in nano-Cmos ICs ..... 5
1.1. Need for High Speed Design ..... 5
1.2. Propagation Delay: Introduction and Types ..... 8
1.3. Propagation Delay Models ..... 9
1.3.1. Model for Propagation Delay Evaluation ..... 12
1.3.2. RC Chain Propagation Delay Model ..... 13
1.3.3. Charge Propagation Delay Model ..... 15
1.3.4. Logical Effort ..... 16
1.4. State of the Art Models ..... 18
1.5. Objectives of the Thesis ..... 20
1.6. Contributions of the Thesis ..... 20
1.7. Organization of the thesis ..... 21
2. Statistical Variations in nano-scale CMOS ICs ..... 22
2.1. Process and Operating Variations ..... 22
2.1.1. Introduction, Sources and Solutions ..... 22
2.2. Global and Local (i.e. mismatch) Process Variations ..... 25
2.3. Process Corner Models ..... 25
2.4. Impact of Transistor Parameters ..... 29
2.4.1. Transistor Dimensions (W, L) ..... 30
2.4.2. Threshold Voltage $\left(V_{T}\right)$ ..... 31
2.4.3. Oxide Capacitance ..... 32
2.4.4. Mobility ..... 33
3. Propagation delay model developed ..... 35
3.1. Overview ..... 35
3.2. Deterministic Propagation Delay Estimation Model ..... 37
3.2.1. Single stage ..... 40
3.2.2. Multi stage ..... 46
3.2.3. Slew time ..... 46
3.2.4. Load capacitance ..... 48
3.3. Statistical Propagation Delay Estimation Model ..... 52
3.3.1. Global Variation Analysis Implementation ..... 54
3.3.2. Extension to Local Variation Analysis ..... 55
3.4. Model Implementation ..... 56
3.5. Summary ..... 59
4. Results on deterministic propagation delay prediction in nominal condi-
tions ..... 61
4.1. Overview ..... 61
4.2. Deterministic single stage ..... 62
4.2.1. inverter ..... 62
4.2.2. nand2 ..... 63
4.2.3. nor2 ..... 63
4.2.4. ao12_n ..... 64
4.2.5. ao22 n ..... 65
4.2.6. ao31 n ..... 66
4.2.7. ao32 n ..... 67
4.2.8. ao33 n ..... 68
4.2.9. ao112_n ..... 69
4.2.10. ao212_n ..... 70
4.2.11. ao222 n ..... 71
4.2.12. Discussion ..... 72
4.3. Deterministic multi stage ..... 73
4.3.1. inverter chain ..... 74
4.3.2. nand2 chain ..... 80
4.3.3. Full Adder ..... 89
4.3.4. Discussion ..... 90
4.4. Summary ..... 90
5. Results on statistical propagation delay prediction in variable process conditions ..... 93
5.1. Statistical single stage ..... 93
5.1.1. Inverter ..... 94
5.1.2. Nand2 ..... 95
5.2. Statistical multi stage ..... 96
5.2.1. 9 inverter ..... 97
5.2.2. 9 nand2 ..... 98
5.3. Statistical Multi Stage for Macrocell Design/Complex Circuits ..... 99
5.4. Summary ..... 101
6. Conclusions ..... 103
Bibliography ..... 107
A. VHDL code ..... 117
A.1. Example: NAND2 DUT at logic level ..... 117
A.2. Example: NAND2 behavioral at logic level ..... 121
A.3. Example: NAND2 testbench at logic level ..... 121
A.4. Modelsim ..... 128
A.4.1. Compile a library by command line ..... 128
A.4.2. TCL script file ..... 129
A.4.3. Run TCL script file ..... 130
B. C code ..... 131
B.1. Create new SPICE netlist ..... 131
B.2. SPICE output elaboration ..... 133
B.3. Table to VDHL matrix ..... 135
C. Script code ..... 139
C.1. Calculate $\tau$ parameter ..... 139
C.2. Calculate $C_{\text {in }}$ Capacitance ..... 143
C.3. Example: Deterministic circuit level simulation ..... 148
C.4. Example: Statistical circuit level simulation ..... 151
C.5. General use: deleting a type of file in all subdirectory ..... 153
C.6. General use: modify a file whit sed command ..... 154
D. Ngspice netlist ..... 157
D.1. Ngspice ..... 157
D.1.1. Show and showmod commands ..... 158
D.1.2. Alter and altermod commands ..... 159
D.1.3. Setcirc command ..... 160
D.1.4. Print command ..... 162
D.1.5. Write command ..... 162
D.1.6. .meas command ..... 163
D.1.7. Batch mode ..... 164
D.2. Example: inverter netlist ..... 164
D.3. Subcircuits netlist ..... 165
Publications and presentations ..... 184

## List of Figures

1.1. Propagation delay definitions ..... 9
1.2. An RC-transmission line model ..... 14
1.3. typical tool-chain ..... 19
2.1. Process and Environmental variations ..... 26
2.2. Corner models ..... 27
3.1. Four current drivers in a cell and associated logic drivers. ..... 38
3.2. Equivalent circuit for the propagation delay model. ..... 40
3.3. Simulation setup for model parameter calibration ..... 45
3.4. multistage path ..... 47
3.5. Behavior of output slew time vs the quantity $t_{I_{-} O}$ ..... 48
3.6. Input pin capacitance characterization setup. ..... 49
3.7. SPICE characterization of input pin capacitance (two-input AND cell) with respect to input slew of the driver cell and to different input logic patterns of the target cell. ..... 50
3.8. Behavior of to as affected by L (transistor drawn length) variation. Other technology variations have a similar effect. ..... 53
3.9. Database structure for the logic-driver-based timing simulation envi- ronment. Arrows indicate dependencies. ..... 57
3.10. Basic scheme of standard cell description. ..... 58
3.11. Implementation of the input pin capacitance simulation model. ..... 59
4.1. VHDL vs SPICE $t_{L H}$ NOT cell. Input slew time 10 ps and 50 ps . ..... 63
4.2. ao12_n input A. Differente slew time (left 10ps,rigth 50ps) ..... 67
5.1. Statistical analysis of single-stage: nand2 gate gaussian ..... 96
5.2. Critical path through Execute Stage of FIR filter ..... 101
5.3. Critical path through Execute Stage of MIPS processor ..... 102

## List of Tables

2.1. Process variation modules affecting the transistor parameters ..... 30
3.1. Active and passive driver pair for model calibration ..... 44
3.2. Sample of database record. AND cell (input IN1 with IN2='0') ..... 51
3.3. Sample of database record. AND cell (input IN1 with $\mathrm{IN} 2=$ ' 1 ') ..... 52
4.1. Absolute and Relative error of Inverter: SPICE vs VHDL comparison ..... 64
4.2. Absolute and Relative error of NAND2: SPICE vs VHDL comparison ..... 65
4.3. Absolute and Relative error of NOR2: SPICE vs VHDL comparison ..... 66
4.4. Absolute and Relative error of AO12_n: SPICE vs VHDL compari- son. Input A ..... 68
4.5. Absolute and Relative error of AO22_n: SPICE vs VHDL compari- son. Input A ..... 69
4.6. Absolute and Relative error of AO31_n: SPICE vs VHDL compari- son. Input A ..... 70
4.7. Absolute and Relative error of AO32_n: SPICE vs VHDL compari- son. Input A ..... 71
4.8. Absolute and Relative error of AO33 n: SPICE vs VHDL compari- son. Input A ..... 72
4.9. Absolute and Relative error of AO112_n: SPICE vs VHDL compar- ison. Input A ..... 73
4.10. Absolute and Relative error of AO212_n: SPICE vs VHDL compar- ison. Input A ..... 74
4.11. Absolute and Relative error of AO222_n: SPICE vs VHDL compar- ison. Input A ..... 75
4.12. Cell verification status (single-stage) ..... 76
4.13. Absolute value of propagation delay (3 NOT chain) ..... 76
4.14. Absolute and Relative error of 3NOT chain: SPICE vs VHDL com- parison. ..... 77
4.15. Absolute value of propagation delay (5 NOT chain) ..... 77
4.16. Absolute and Relative error of 5NOT chain: SPICE vs VHDL com- parison. ..... 78
4.17. Absolute value of propagation delay (7 NOT chain) ..... 78
4.18. Absolute and Relative error of 7NOT chain: SPICE vs VHDL com- parison. ..... 79
4.19. Absolute value of propagation delay (9 NOT chain) ..... 80
4.20. Absolute and Relative error of 9NOT chain: SPICE vs VHDL com- parison. ..... 80
4.21. Absolute value of propagation delay (3 nand2 chain). ..... 81
4.22. Absolute and Relative error of 3NAND2 chain (input A): SPICE vs VHDL comparison. ..... 82
4.23. Absolute and Relative error of 3NAND2 chain (input B): SPICE vs VHDL comparison. ..... 83
4.24. Absolute value of propagation delay (5 nand2 chain) ..... 84
4.25. Absolute and Relative error of 5NAND2 chain (input A): SPICE vs VHDL comparison. ..... 84
4.26. Absolute and Relative error of 5NAND2 chain (input B): SPICE vs VHDL comparison. ..... 86
4.27. Absolute value of propagation delay ( 7 nand2 chain) ..... 87
4.28. Absolute and Relative error of 7NAND2 chain (input A): SPICE vs VHDL comparison. ..... 88
4.29. Absolute and Relative error of 7NAND2 chain (input B): SPICE vs VHDL comparison. ..... 89
4.30. Absolute value of propagation delay (9 nand2 chain) ..... 90
4.31. Absolute and Relative error of 9NAND2 chain (input A): SPICE vs VHDL comparison. ..... 91
4.32. Absolute and Relative error of 9NAND2 chain (input B): SPICE vs VHDL comparison. ..... 92
4.33. Relative error of different Full Adder chain: SPICE vs VHDL com- parison. ..... 92
5.1. Statistical analysis of single-stage: inverter gate ..... 94
5.2. Statistical analysis of single-stage: nand2 gate ..... 95
5.3. statistical analysis of multi-stage: 9 inverter gate ..... 97
5.4. statistical analysis of multi-stage: 9 nand2 gate ..... 98
5.5. Statistical analysis of multi-stage for complex circuits: propagation delay comparation ..... 99
5.6. Statistical analysis of multi-stage for complex circuits: execution time comparation ..... 100
5.7. Simulation time FIR filter (SPICE vs HDL) ..... 100

## Abstract

The present dissertation was developed within a European project, which envisages the development of standard cell in 45 nm technology. A critical aspect of the design flow in standard cell in nanometer CMOS technologies is the performance impact of statistical variations of the technological process of manufacture. In particular, this work focuses on the effects of variations on the propagation delay of logic cells, which influence decisively the speed and performance of integrated circuits. In particular, the problem that has been resolved in this thesis is the ability to evaluate the effects of changes of technology parameters statistically, through a simulation at a logical level, which avoids the computational burden of a circuit-level simulation. At first, a propagation delay model for cells consisting of a single-stage CMOS developed in conjunction with a compatible model which is implemented in hardware descriptive language (VHDL). A particular attention has been given to the independence of the model from technology, so as to make it applicable to different technologies; only few technology parameters are selected for changes. Later, the propagation delay model has been extended to multi-stage cells and, potentially, to circuits composed of an arbitrary number of cells. Finally, we have tried to answer the following question: is it possible to evaluate the propagation delay variations of a cell statistically (in terms of mean value and variance) without using a circuit simulator such as Spice
when the cell is affected due to the random variation of technological parameters? Considering the random variations of channel length (L) of the channel width (W), oxide thickness ( $T_{o x}$ ) and the doping concentration ( $N_{D p}, N_{D n}$ ), the developed model shows some encouraging results, such as reporting an error of a few percentage points on the mean value and the variance for the propagation delays of the circuits which has been used for an example. The advantage, in terms of time of simulation of the model at the logical level compared with a model at SPICE-level, is at least 2 orders of magnitude.

# 1. Propagation Delay in nano-Cmos ICs 

### 1.1. Need for High Speed Design

Since the beginning of CMOS technology, the increasing number of transistors in each die and high performance in single IC has been the fundamental driving factors for the semiconductor industry and process technology. The capability to add more transistors in each die allowed chip manufacturers to put more parts of a system into one package and decrease not only just the sizes of the electronic devices we use today but also the cost and propagation delay. The strong competition in the semiconductor industry has motivate the integrated circuit manufacturers to achieve these goals sharply. These challenging objectives which are more transistors per die with high performance have been exponentially growing by following Moore's law. The power dissipation of the Integrated Circuit (IC) also is another factor which has been growing at an appalling value. In today's era, the overweening power consumption of coetaneous circuits has become a dominant design concern. However, the issue of propagation delay time is one of the main concerns that have hindered the future scaling of transistors. A Very Large Scale Integrated (VLSI) integrated circuits consists of number of energy storage elements, most of them are capacitors,
and few are needed for computation capacitances which result in interference to circuit operation. The capacitors are persistently charged and discharged by resistive elements during circuit operations which results in energy dissipation in terms of heat. The amount of heat dissipated puts a restriction on the computational performance of the circuit, or the number of times the transistors in the circuit can switch for a given power budget. One could argue that the shrinking of devices has reduced the amount of parasitic capacitance and this alleviates power dissipation problems. However, the increase in the number of devices due to the increase in device density has more than compensated for the decrease in the parasitic capacitance of a single device .

Contemporary digital logic circuits have millions of transistors on a single silicon integrated circuit chip. It is desirable to learn the circuit performance during the design stage. One of the most important performance measures of digital logic circuits is the propagation delays of switching signals propagating through the logic gates of the circuit.

Propagation delay models in nano-scale CMOS digital circuits provide an initial design solution for integrated circuits which believe to be one of the essential design specifications. Both pecuniary and workforce resources constrain the design process and that leading to the need for a more accurate entry point further along in the design cycle. The standardization for any given process technology can be attained by verifying an existing propagation delay method and its resulting propagation delay model.

Requiring specialized design solutions, the full custom design for very large-scale integrated circuits (VLSI) delivers many unique design issues. The basic elements of full custom design which are unresolvable associated together are physical-design area, circuit manufacture's cost, circuit's speed and the power of circuit. The area
and cost are often referenced symmetric since the cost per die is directly proportional to the amount of dies one wafer can yield. The cost to manufacture a silicon wafer is typically fixed and therefore the cost per die is directly linked to the area of the die. If a die increases in size, less will fit on a single wafer, and the cost of each die will rises respectively.

Process technology prescribes that there is a maximum die size that can be manufactured reliably and sets a scope to the size of the circuits that one die may contain. This is the reason that the whole motherboards within personal computers are not entirely on a single chip. Therefore, every technology comes nearer and implements more transistors per die than the previous generations. The eventual objective of developing an entire system on a single chip is yet to be made.

High speed is a process technology restricting constant. There are several ways to define speed and the most practical definition is based on describing the digital speed. The digital speed scope can be estimated by creating an odd number of inverter chain in a loop. This circuit will hover at the highest possible frequency for a given digital circuit. This speed value is not practical since most digital design is implemented with combinational logic. Hence, the target speed for a system is usually inferred from a conventional circuit topology and tested for highest speed. Technology scaling has always done for the sake of increasing transistor count i.e. density and operating frequency and to fulfill the fabrication market demand of increasing number of transistors with each new technology node, more and more number of functions in single IC. However, as a drawback, this scaling always promoted the unwanted leakage. Downsizing of the channel length gives rise to short channel effects, which also increases the sub-threshold leakage. Lowering the supply voltage in digital applications is one of the reliable ways for low power consideration, however lowering supply voltage increase propagation delays in digital circuits.

Threshold voltage ( $V_{t h}$ ) also scaled along with supply voltage to maintain switching activity, but Vth does not scale much as limited by sub-threshold leakage. Scaling of oxide thickness increases the gate tunneling currents, so there is limit on oxide thickness reduction as oxide thickness needed to maintain the current drive and keep threshold variation under control. Therefore, transistor density, functionality and speed have increased with technology scaling on one hand, but power density and variability have also increased on the other hand. Moreover maximum integration density is limited by the power while the circuit switching speed is limited by the variations.

A well-defined circuit architectural specification is required in order to develop an accurate propagation delay model with minimal calibration effort. Examination of existing propagation delay model calibration methodologies gives a platform for the development of enhancements in accuracy. The architectural design specification limits the balance between accuracy and solution acquisition time. Every full-custom integrated circuit design presents a unique accuracy and effort requirements and the best solutions are commonly contained of a hybrid model of theoretical equations fitted with simulation-based fitting coefficients.

### 1.2. Propagation Delay: Introduction and Types

Propagation delay has been the main figure of merit defining the performance of a digital design since the early ages of electronics, as it determines the clock period of synchronous systems and ultimately the speed of digital devices in any application. Here we give the basic definition of propagation delay measurement, which will be used as a reference in the following sections.

When a signal is applied at the input of a logic gate it experiences a lapse of time
while reaching the output. The propagation delay of digital gates is then a measure of time that defines how fast a gate responds to changes in signals applied at inputs.


Figure 1.1.: Propagation delay definitions

Propagation delay is defined as the time difference measured between the transitions at $50 \%$ of the full voltage swing, for both input and output signals as shown in Figure 1.1 for the case of an inverter cell. Since digital gates have different response time for rising and falling input signal, we distinguish two values, described as $t_{L H}$ and $t_{H L}$ in the Figure 1.1, respectively to identify the propagation delay of low to high and high to low output transitions. The propagation delay is defined as the average of $t_{L H}$ and $t_{H L}$.

### 1.3. Propagation Delay Models

The speed of a logic block's input and output load is the dependency of each logic block. Circuit design complexity comes from the interdependence of the individual logic blocks within a design. If speed of one block is raised, the block driving it experienced an increase in load which subsequently slows down the speed of the
respective block. Increasing the blocks can be inferred in such a way that any chosen stage's previous stage can propagate the issue of all the way to the first input of the entire circuit. Circuits can have thousands of initial timing issues that would lead to gross over-corrections if not addressed properly. This is where the use of a propagation delay models can provide significant contribution.

The ultimate device size can be precisely predicted by a suitable propagation delay model. A propagation delay model can help the designer to avoid numerous iterations of device sizing and testing required by an improperly chosen initial devicesizing scheme. The accuracy and complexity of a propagation delay model changes based on the individual requirements of the designer. The ordinary designs can use less complex propagation delay models while designs with greater complexity require huge complexity in the respective propagation delay models.

The previous contribution related to propagation delay modeling exist in large quantity in the existing literature. The process for building a propagation delay model is based on developing an understanding of common behaviors and effects for a given technology and translating those effects into a reproducible system for rapid analysis. The existing developed models are well-known analytical propagation delay model, and simulation results to calibrate the original model with corresponding coefficients. The resulting model accounts for second order effects omitted from the original analytical model. The calibrated model offers an alternative to broad circuit analysis, by trading accuracy for rapid design accomplishment.

The fundamental propagation delay models consists on small number of factors such as output load, circuit voltage and manufacturing technology that control a real circuit propagation delay. Experimental and theoretical work on the topics of input slope, fan-out, interconnect, and logical effort provide modeling strategies to consider the modeling effects are neglected in the fundamental models. Updating a
basic propagation delay model with elaborated modeling effects and fitting the model to a given process that give a rise in modeling accuracy with minimal increase to the modeling complexity.

Propagation delay models for CMOS digital logic normally exclude second-order effects due to their limited impact on modeling accuracy. The total propagation delay accuracy for most digital circuits is often $90 \%-95 \%$ for input slope, device sizing, and output-load. The effect of second order effects are described within the limit of the long channel CMOS propagation delay model. Those effects consists of substrate biasing, carrier saturation velocity, body-effect, and channel length modulation.

The above said exceptions simplify the derivation in great extent and resulting in the propagation delay model formulation. These effects can be catered in order to get the accuracy when precision is needed and when the exact application architecture is defined. Channel length modulation is only considered in short-channels, where the effective channel length of a MOS device is approximately equal to the source and drain junction depths.

Following all the definitions and terminologies discussed above, the resulting propagation delay models for rising and falling transitions of a standard CMOS inverter are [1]:

$$
\begin{equation*}
\tau_{P_{H L}}=\frac{C_{\text {load }}}{k_{n}\left(V_{D D}-V_{T, n}\right)}\left[\frac{2 V_{T, n}}{V_{D D}-V_{T, n}}+\ln \left(\frac{4\left(V_{D D}-V_{T, n}\right)}{V_{D D}}-1\right)\right] \tag{1.1}
\end{equation*}
$$

$$
\begin{equation*}
\tau_{P_{L H}}=\frac{C_{\text {load }}}{k_{p}\left(V_{D D}-\left|V_{T, p}\right|\right)}\left[\frac{2\left|V_{T, p}\right|}{V_{D D}-\left|V_{T, p}\right|}+\ln \left(\frac{4\left(V_{D D}-\left|V_{T, p}\right|\right)}{V_{D D}}-1\right)\right] \tag{1.2}
\end{equation*}
$$

$$
\text { Where } \quad k_{n}=\mu_{n} \cdot C_{o x}\left(\frac{W_{n}}{L_{n}}\right) \quad \& \quad k_{p}=\mu_{p} \cdot C_{o x}\left(\frac{W_{p}}{L_{p}}\right)
$$

$C_{\text {load }}$ is a Capacitive load applied to the output of the inverter;
$V_{D D}$ is a Drain voltage applied to PMOS Drain Terminal;
$V_{T}$ is a Threshold voltage for a transistor;
$C_{o x}$ is Gate-Oxide capacitance
$\mu_{n}, \mu_{p}$ are mobility of electrons and holes through transistor channel;
$k_{n}, k_{p}$ are transconductance of the NMOS and PMOS transistor.
The above equations Equation 1.1, Equation 1.2 represent propagation models which are expressed without the involvement of body biasing effects, saturation velocity, and channel length modulation. In order to enhance the accuracy of uncomplicated propagation delay models, iterative analysis and back-fitting has been represents to give a faster and reliable solution. The Logical Effort method also supports the iterative analysis. To allow accurate initial solutions, the magnitude of improvement fluctuates across different manufacturing technologies and reveals no simple trends.

### 1.3.1. Model for Propagation Delay Evaluation

CMOS inverter propagation delay requires consideration for input slope effects and modeling of the source-drain series resistances. The resulting methodology consists of semi-empirical fitting coefficients matched to a propagation delay model for CMOS inverters. The number of research works discusses the propagation delay for inverters and some particularly focus on the effects related to the source-drain resistance and the input slope.

Propagation delay is the amount of time from an input signal passing through $\frac{V d d}{2}$, until the output transition in the opposing direction through $\frac{V d d}{2}$. The propagation
delay can be categorized into two parts. The first part is the propagation delay resulting from an instantaneous input, or step input and the second part is the contribution from the input slope. The second part can be found experimentally computing the step input propagation delay in SPICE. The realistic propagation delay of a sloped input and subtracting the step propagation delay from the sloped input propagation delay can also be measured in the same way. The difference between the two propagation delays is the party done by input slope.

### 1.3.2. RC Chain Propagation Delay Model

The current behavior in an RC chain provides the method of accounting propagation delay models for RC chains. Three different RC models named as interconnect, transmission-gate, and downstream load are comprise on the existing structures for modeling current networks propagation delay. Propagation delays can also be simulated through equivalent RC transmission line models. A step input current generator closely matches results of a transfer function model. The ultimate circuit optimizations using the above discussed method, results in circuit driving paths with less signal-buffer stages and therefore less total power and silicon area used.

There are three transmission propagation delay models show circuit topologies for interconnect or line impedance, pass-gate or transmission-gate impedance, and CMOS logic buffers. The standard transmission line model and input step-response current generator are driving a resistor-capacitor network as shown in the Figure below.

The propagation delay for a transmission line is not modeled with current source but with an input voltage source. The demeanor of an RC ladder network was enough close to the first order circuit model while using Elmore's time constants. There is


Figure 1.2.: An RC-transmission line model
an assumption taken that the signal transition was complete at maximum Vdd or ground. Hence, it has effectively infinite period. Therefore, the CMOS buffers that drive the RC ladders do not match with voltage sources but with current sources. This demeanor is the accelerator for choosing current input sources for the models rather than the traditional voltage inputs found in most transmission signal analysis schemes.

The simplest model are a resistance and a capacitance. Assuming that the capacitance is discarged and the input is a rising step pulse initially. The equation for this simple RC circuit is:

$$
\begin{equation*}
V_{\text {out }}(t)=V_{D D}\left(1-e^{-\frac{t}{R C}}\right) \tag{1.3}
\end{equation*}
$$

The time at $50 \%$ of $V_{\text {out }}$ is propagation delay Low-High $\left(\tau_{p_{L H}}\right)$ :

$$
\begin{equation*}
V_{\text {out } / 50 \%}\left(t^{*}\right) \sim \frac{V_{D D}}{2}=V_{D D}\left(1-e^{-\frac{t^{*}}{R C}}\right) \tag{1.4}
\end{equation*}
$$

$$
\begin{aligned}
& \ln \left(1-\frac{1}{2}\right)=\ln \left(e^{-\frac{t^{*}}{R C}}\right) \\
& t^{*}=\tau_{p_{L H}}=\ln (2) \cdot R C \sim 0.69 \cdot R C
\end{aligned}
$$

This is a computation for lumped RC network. For propagation delay calculation for distributed RC network, usualy using Elmore propagation delay formula. As a special case of RC network is RC ladder network shown in Figure 1.2. For this case the Elmore propagation delay for a generic node N is calculated as:

$$
\begin{equation*}
\tau_{D_{N}}=\sum_{j=1}^{N} C_{j} \sum_{i=1}^{j} R_{i} \tag{1.5}
\end{equation*}
$$

The findings by considering an input current source to drive RC ladder networks leads to a simplified propagation delay model compared to state of the art circuit propagation delay models. The above discussed method of optimizing paths has produced lower propagation delays and finally need less signal repeaters than state of the art methods. The consumption of simplified logic to get the same signal-timing goal means an overall savings of power and silicon area in the ultimate product.

### 1.3.3. Charge Propagation Delay Model

The available charge and the resulting propagation delay can be expressed in the charge propagation delay model in terms of some common relationship between
both kinds of propagation delays. There is a way to estimate the propagation delay for complex CMOS gates by using the methodology of inverter propagation delay model. The inverter propagation delay model is based on an nth-power law MOSFET model. Transistor collapsing schemes which are developed for complex gates are taken into account for the impacts of the body effect, short-channel and internal coupling capacitance.

CMOS device stacks can be simplified in terms of slope propagation delay curves. The slop delay curves represent a conventional inverter with a varying output load. Constructing a complex stack compare to a simple inverter model can simplify the evaluation of gate-level complex circuits. Capacitive values for the parasitic and load capacitors are integrated together to show a single static load. The currents are derived from propagation delay, slope and integrated capacitances. The charge propagation delay definition may be extended by deriving a table between delay-in and delay-out values. The resultant table will be the simplified complex circuits into a simple propagation delay chart, which have curves for each previous complex device that will be reduced down to a resultant inverter.

### 1.3.4. Logical Effort

Logical Effort is a technique for analyzing propagation delays of digital-circuit. It uses the equivalent information to recognize the relative trade-offs between circuitdesign complexity and circuit-speed as well. The rapid circuits tend to have very much logical complexity and power consumption . This method proposed two methodology for realizing abilities of circuit and its scope. These methods are called as logical and electrical effort .

The fundamental assumption of this technique can be established by qualitative analysis of a simple circuit. There are design tradeoffs among speed, size, power,
and capacitive-load for an inverter of any given manufacturing technology. The output propagation delay for MOS device can be further derived with the following equation:

$$
\begin{equation*}
d_{\tau}=d \cdot \tau \tag{1.6}
\end{equation*}
$$

Where d represents the collection of all other effects lumped into a singular quantity and $\tau$ is the basic propagation delay unit for an inverter driving a fan-out of one, without considering any parasitic capacitances. The lumped-effects d is further simplified in to two major parts:

$$
\begin{equation*}
d=f+p \tag{1.7}
\end{equation*}
$$

Where d is the realized propagation delay for the inverter with all the parasitic and other effects combined. The fixed portion of the propagation delay, when no load is attached, is parasitic propagation delay which represents as $p$, and the variable portion $f$ is called the "effort delay" or "stage effort" that depends on the complexity of the logic and fanout of it:

$$
\begin{equation*}
f=g \cdot h \tag{1.8}
\end{equation*}
$$

$h$ represent output load and is named "electrical effort". It can be computed as

$$
\begin{equation*}
h=\frac{C_{\text {out }}}{C_{\text {in }}} \tag{1.9}
\end{equation*}
$$

Electrical effort shows the ratio of a circuit's output load capacitance relative to the input capacitance.
$g$ is a "logical effort" and is represented a complexity of logic. For an Inverter cell is 1 , for other cell is calculated as the ratio between the sum of the widths of the mos of each input of cell divided by the sum of the widths of the mos of an inverter. For example for a NAND2 cell, both input have a Nmos with $\mathrm{W}=2$ and Pmos with $\mathrm{W}=2$ then the logical effort is $\frac{4}{3}$ for both input.

An inverter and a NAND2 gate of equal number transistor sizes and driving equal capacitive loads will produce different magnitudes of current due to their relative logical complexity. This difference is accounted for in the term for logical effort. The same circuit driving different fixed capacitive loads will result in varying current delivery. This behavior is shown with the term for electrical effort.

### 1.4. State of the Art Models

Accurate timing analysis is a challenging task in modern very large scale integration design flows, due to the presence of nonlinear effects in nano-scale CMOS standard cells, such as velocity saturation, input-output coupling, voltage feed-through , short circuit effect from simultaneous pull-up and pull-down conduction, and others. Currently, the estimation of the propagation delays of logic cells in a complex semicustom design is accomplished by electronic design automation (EDA) tools called
propagation delay calculators and back annotated on the gate-level design netlist in a deterministic scheme, i.e., assuming the worst-case technology and operating conditions corner. The propagation delay models used in propagation delay calculators have evolved from simple lookup tables, to polynomial models, nonlinear models, and more recent current-based models.

Current nano-scale digital CMOS IC design, the growing statistical variability in process parameters makes traditional logic-level corner-based simulation approaches not adequate for a realistic estimation of the fabrication yield. Simultaneously SPICE transistor-level Monte Carlo analysis is impractical for complex designs due to the huge simulation time. Therefore, there is a need for logic level models featuring technology variation aware timing, thus suitable for implementing accurate Monte Carlo analysis through fast logic-level simulators [9].

Traditional logic level post-synthesis timing simulation relies on a typical tool-chain which is basically organized as follows:


Figure 1.3.: typical tool-chain

Propagation delay calculator tools read the characterization database and compute a fixed propagation delay for each node in the logic level netlist. The characterization database are written in standard formats (e.g. Liberty). Propagation delay calcu-
lators tools are based on propagation delay models which have evolved during the last 20 years through several generations. Among the most important propagation delay models used in the past up to today there are:

- Propagation delay Lookup Tables (indexed by input slew, load cap) - past
- Polynomial propagation delay models (such as SPDM) - past
- Non-linear propagation delay models (NLDM) - past
- Current based propagation delay models (CCS, ECSM) - used today

Current based propagation delay models guarantee the highest accuracy ever (at the expense of a huge characterization database size) and are still in evolution.

### 1.5. Objectives of the Thesis

The overall objective of this research work is to develop new and novel techniques and model which can estimate the propagation delay in single and multi-level cell paths. The sub-objectives are as follows.

1. Develop a methodology to compute propagation delay for small and large both kind of circuits deterministically and statistically.
2. To build up a general systematic mechanism to design Synchronous early-completion-prediction adders (ECPAs) units targeting nano-scale CMOS technologies.

### 1.6. Contributions of the Thesis

The contributions of this thesis are as follows.

1. A novel model has been introduced for nano-scale CMOS circuits for propagation delay modeling for both deterministic and statistical (single and multi-stages).
2. A novel synchronous early completion prediction adder's methodology with significant results is developed.

### 1.7. Organization of the thesis

This thesis is organized is as follows.

Chapter 1 provides the motivation for conducting research on propagation delay in nano-scale CMOS logic with basic definitions followed by the objectives and contributions of this research work.

Chapter 2 provides the literature review and importance of statistical variations in nano-scale in current era with impact of transistor parameters.

Chapter 3 elaborates the proposed models named as deterministic and statistical with model implementation methodology and algorithm in detail.

Chapter 4 represent the results and analysis in detail on our proposed deterministic model implementation which is being done only on few selected small and medium level circuits.

Chapter 5 shows the results and analysis in detail on our proposed statistical model implementation which is also being done on small, medium and as well as on largescale level circuits.

Chapter 6 provides the conclusion of this research work.

# 2. Statistical Variations in nano-scale CMOS ICs 

### 2.1. Process and Operating Variations

IC process variation (PV) was already a concern for high-performance circuits, but in recent technology nodes, with reduced power and threshold margins, its impact has become even more extensive. Process variations are always exist in spite of designer preferences and degrades the yield therefore there is need to model the timing violations or propagation delay modeling by applying novel variation tolerant design techniques.

### 2.1.1. Introduction, Sources and Solutions

Traditionally, measure of quality of performance and power estimation are adopted on the premise that the electrical characteristics and operating conditions of every device in the design matches the model specifications. However, with continued miniaturization of device dimensions to current nano-scale regime, it has become highly difficult to maintain the same level of manufacturing control and uniformity. This leads the devices to behave and interact in a complex manner differently from
model characteristics. Moreover, devices that were supposed to be identical are differing in their electrical characteristics which sometimes can lead to functional failures. Environmental factors i.e. operating variations such as supply voltage and temperature experienced by devices in different parts of the chip (or on different chips) also vary due to different levels of device densities, switching activities, and noise integrity of the various blocks. The raising mismatch between the gold standard models and the actual device parameters occurs when voltage and temperature stress go through by the devices with uninterrupted usage reduces their electrical characteristics. The above discussed elements can add together to make the current design substantially different from the state of the art design which results in degradation of overall performance of the design. Furthermore, the variation in chip power occurs from its absolute values which is due to the exponential dependency among process parameters, device parameters and transistor leakage. Ultimately the number of shippable products reduced due to the resultant degraded parametric yields which represent the number of manufactured chips that fulfil the expected performance, reliability and power specifications. The price for each effective chip rises, which has a direct impact on the bottom line dollar value of the design with the initial design price being the same. The effect of fluctuations on the parametric yield is generally degraded by guard-banding to give enough tolerance margins during design, which is requital to designing at a non-optimal power-performance point. Designs are assigned to function at smaller than absolute frequencies to assure functional conformity in the existence of fluctuations. The expected guard-banding also rises, particularly if worst-case conditions are turned over with raising fluctuation. A practical amount of design margining expected by understanding the several sources of fluctuation and their effect on circuits can assist in finding, and also give revelations to rationalize the effects of fluctuation. To determine the magnitude of
fluctuation, classifying various effects and their dependence and giving feedback to the manufacturing team can be achieved by knowing the on-chip characterization of circuits. These techniques can also be used in the field to detect possible failure conditions, analyze the chip over time and align global parameters which can help lower these effects. Variations usually called as fluctuations can be categorized into three main types such as process, environmental, and temporal variations. Process variations are the result of irregular control over the fabrication process. Environmental variations such as temperature and supply voltage increase throughout the operation of the circuit due to shift in the operating conditions. Temporal variations occur due to the shift in the device characteristics over time. The remaining chapter will give detail about the sources of variations and the affected circuits usually used to characterize these effects. The effect of process and environmental variations on performance is getting broad with each reached technology node. There are several reasons for the occurrence of process variation like non similarities brings in during process outcomes in the variation in lithography, doping, in gate oxide and etc or while transporting wafer, little shift in temperature may increase to process variation during manufacturing thereafter changing the performance of the transistor [2, 3].

Degrade sources of variations- By suitable fabrication of the devices i.e. the particular device is fabricated in a way that it decreases the effect of variation, by realizing the circuit using

- Multi-layer, multi-threshold insertion,
- Circuit style and logic decisions,
- Power delivery and thermal design

Lowering the effect of variation at circuit design level (pre-silicon)-

- Leakage reduction techniques,
- Variation tolerant circuits,
- Dynamic compensation circuits

Decreasing the effect of variation at post silicon level-

- Clock tuning
- Adaptive body bias
- Adaptive supply voltages


### 2.2. Global and Local (i.e. mismatch) Process Variations

Global process parameters (e.g. oxide thickness) are wafer-to-wafer, chip-to-chip variations, or batch-to-batch variations. Local process parameters (e.g. threshold voltages of transistors) impact each device of a chip individually i.e. variability between two devices might looking identical to each other.

Local process parameters can become the reason of mismatch and may disturb fundamental design principles of creating constant differences and rations of currents and voltages between such pairs, the influence of local process parameters on the act of the circuit can be much higher than those of the global process parameters .

### 2.3. Process Corner Models

It is very certain in industry to analyze the statistical variation using five types of worst case models, everyone is defined by a two letter acronym title describing the


Figure 2.1.: Process and Environmental variations
relative performance characteristics of the p-channel and n-channel devices. The letters elaborate the operation as being typical or nominal (T), fast (F) or slow (S). For instance, "FS" denotes the position where the n-device is fast and the p-device is slow. Usually, the design is complete in typical p, typical n (TT) condition, and checked for all four corners (SS, FF, FS and SF). There is an assumption being taken is that the provided a yield level, the four corners bound all worst possible scenarios of extreme performance. In case the circuit can fulfill the test on these corners, then certain yield is guaranteed. This methodology has been successfully followed in circuit design practice, yet it faces two significant challenges.

The initial challenge is related to the definition and extraction of the corners. Existing process corner models are defined and extracted using the measurement data on individual transistors [4]. For digital logic circuits, the worst-case performance commonly measured by critical propagation delay occurs when single transistors are


Figure 2.2.: Corner models
also at the worst case.

The raising variability in IC design, changes the premises to look for novel design methodologies. The digital process corners are becoming ineffective as the design can be operational at all corners but not at some combination of intermediate values. Plus and minus three sigma ( $\pm 3 \sigma$ ) points generally selected to represent fast and slow corners for such devices. These corners are provided to represent process variation that the designer must account for in their designs. This variation can cause significant changes in the performances of digital signals, and can sometimes result in catastrophic failure of the entire system. Digital corners account for global variation and in a digital design context referred as 'slow' and 'fast' which are irrelevant sometimes especially in analog design. Digital corners do not include local variations effects which are critical in present scenario. Digital corners are not design-specific which is necessary to determine the impact of variation.

The characteristics discussed above lower the accuracy of the digital corners which become the reason to limit the process corners to use as accurate indicators of variation limits especially for analog designs. Traditional corner model based analysis and design approaches provide guard-bands for parameter variations and are, there-
fore, prone to introducing pessimism in the design. Such pessimism can lead to increased design effort and a longer time to market, which ultimately may result in lost revenues. In some cases, a change in the original specifications might also be required while, unbeknownst to the designer performance is actually left on the table. Furthermore, traditional analysis is limited to verifying the functional correctness by simulating the design at a number of process corners. However, worst-case conditions in a circuit may not always occur with all parameters at their worst or best process conditions.

However, a single corner file cannot simultaneously model best-case and worst-case process parameters for different interconnects in a single simulation. Suppose that the worst case for a pipeline stage will occur when the wires within the logic are at their slowest process corner and the wires responsible for the clock delay or skew between the two stages is at the best-case corner. So a traditional analysis requires that two parts of the design are simulated separately, resulting in a less unified, more cumbersome and less reliable analysis approach. The strength of statistical analysis is that the impact of parameter variation on all portions of a design are simultaneously captured in a single comprehensive analysis, allowing correlations and impact on yield to be properly understood. The magnitude of process variations have grown, there has been an increasing realization that traditional design methodologies (for both analysis and optimization) are no longer acceptable. The extent of variations in gate length, as an example, are predicted to increase from $35 \%$ in a 130 nm technology to almost $60 \%$ in a 70 nm technology. These fluctuations are generally specified as the fraction $3<\sigma / \mu(3<\sigma$ is assumed to be the worst case shift in the parameter), where $\sigma$ and $\mu$ are the standard deviation and mean of the process parameter, respectively. Thus, a $60 \%$ variation in 70 nm technology implies that the standard deviation of the distribution of gate length across a large number
of samples is 14 nm . With variations as large as these, it becomes very significant that the designers handle these variations in a statistical manner rather than using guard-bands in deterministic analysis.

### 2.4. Impact of Transistor Parameters

All the variation sources affect the electrical properties of the transistors and interconnect in several ways. These affects can better understand in terms of transistor performance. In typical digital Integrated Circuits, a transistor either charges or discharges the capacitive load and the time for this operation determines the performance of the transistor. This time is a function of driven capacitance, the required voltage and the current used to drive the capacitance as shown in Equation 2.1

$$
\begin{equation*}
t_{d}=\frac{C_{\text {load }} \cdot V_{D D}}{I} \tag{2.1}
\end{equation*}
$$

To understand, we have used Equation 2.2 shows the ideal I-V equation for the MOSFET in saturation region, where $\mu$ is the mobility of the charge carrier through the device, Cox is the gate oxide capacitance, W and L are the width and length of the transistor respectively. $V_{T}$ is the threshold voltage of the device and $V_{G S}$ is the gate to source bias voltage.

$$
\begin{equation*}
I_{D}=\frac{1}{2} \cdot \mu \cdot C_{o x} \cdot \frac{W}{L} \cdot\left(V_{G S}-V_{D D}\right)^{2} \tag{2.2}
\end{equation*}
$$

$$
\begin{equation*}
t_{d}=\frac{C_{\text {load }} \cdot V_{D D}}{\frac{1}{2} \cdot \mu \cdot C_{o x} \cdot \frac{W}{L} \cdot\left(V_{G S}-V_{D D}\right)^{2}} \tag{2.3}
\end{equation*}
$$

Although these equation is ideal and neglect several details as far as modern nanoscale transistors are concerned, however it is sufficient to illustrate the impact of variation sources.

| MOSFET Parameters | Relative Process Variation Module |
| :---: | :---: |
| Width (W) | Lithography, Etching |
| Length (L) | Lithography, Etching |
| Threshold Voltage (VT) | Ion Implantation, Annealing, Gate Oxidation, (Lithography, Etching) |
| Oxide Capacitance (Cox) | Gate Oxidation |
| Mobility ( $\mu$ ) | Ion Implantation, Annealing, Diffusion, Nitride Deposition |

Table 2.1.: Process variation modules affecting the transistor parameters

Table 2.1 shows the MOSFET parameters which directly affected by the process variations. It is clear that single process variation can affect multiple transistor parameters. Separating the impact of one process variation from another variation is difficult.

### 2.4.1. Transistor Dimensions (W, L)

Ever increasing number of transistors, more and more number of functions in single IC has always been the demand for IC fabrication market; therefore, technology scaling has always done for the sake of increasing transistor count and frequency . It is obvious from Eq. 2 that the width (W) and length (L) are the critical parameters in determining the current in transistor. Either W must be increase or L decreases to increase the current and thereafter performance of the device.

Downsizing also reduces the load capacitance and increases the transistor density in ICs, making length ( L ) one of the most critical dimensions in the transistor.

The etching and lithographic pattering define both the length (L) and width (W). Normally in circuit designing, W is always larger than L , therefore making channel length more prone to the impact of process variations (unless we do not have smallest width devices). As shown in Equation 2.3, the propagation delay of transistor is directly proportional to the channel length; therefore, any variation in channel length will be directly reflected in the transistor delay. As per the International Technology Roadmap for Semiconductor (ITRS) projection for channel length in , as the transistor length has decreased below the wavelength of light patterning them, the relative variations in the channel length has increased.

In the year 2001 and 2003, projection were $10 \%$ into the foreseeable future, however in 2005 and 2007, the $3 \sigma / \mu$ projections increased to $12 \%$ which will increasing the performance variability and this performance variability will enhance with each scaled technology node, without adopting the mitigation techniques. The variations in channel length also results in threshold voltage variations due to the Drain-Induced Barrier Lowering Phenomena (DIBL) and has severe effect on channel length below 100nm. Modern processes show measured channel length variation of $3-5 \% \sigma / \mu$, consistent with the ITRS projections .

### 2.4.2. Threshold Voltage ( $V_{T}$ )

Threshold voltage of the MOSFET is the gate to source bias $\left(V_{G S}\right)$ that responsible for the channel formation below the gate, allows current conduction from source to the drain of the transistor. In ideal long-channel MOSFET, the doping concentration in the channel and oxide capacitances (oxide thickness) determines the threshold
voltage as shown in the Equation 2.4

$$
\begin{equation*}
V_{T_{0}}=V_{F B}+2 \cdot \varphi_{F_{p}}+\gamma \cdot \sqrt{2 \cdot \varphi_{F_{p}}+V_{S B}} \tag{2.4}
\end{equation*}
$$

Where VFB is the flat band voltage, $\phi \mathrm{F} \_\mathrm{P}$ is the Fermi potential of the substrate. $V_{F B}, \phi$ and $F_{P}$ are dependent on the doping concentration only while $\gamma$ depends on both the doping concentration and oxide capacitance. In short-channel devices, some effects such as DIBL result in $V_{T}$ being additionally depend on the channel length (L), source/drain junction depth $\left(X_{j}\right)$, and stresses. As a result, several process steps affected the $V_{T}$. Due to this susceptibility and the intrinsic random variability of Random Dopant Fluctuations (RDF), $V_{T}$ is the most prone parameter to the process variations, with $3 \sigma$ variations on the order of $30 \%$ or more of mean. However, it has studied a lot in the literature.

Since the mean value of $V_{T}$ reduces with technology generations, relative variations $\sigma / \mu$ increases even more rapidly.

### 2.4.3. Oxide Capacitance

Gate oxide capacitance is the capacitance between the gate stack (polysilicon and silicon dioxide) and the inverted channel of the MOSFET. Equation 2.5 shows that the oxide capacitance is function of the oxide thickness (Tox) and the dielectric constant of silicon dioxide or other gate insulator.

$$
\begin{equation*}
C_{o x}=\frac{\varepsilon_{o x}}{t_{o x}} \tag{2.5}
\end{equation*}
$$

As we know, gate oxidation (thermal growth of silicon dioxide or silicon nitride) is a relatively well-controlled process step. However, with the technology scaling, gate oxide thickness is also scaled down as needed to maintain the current drive and keep threshold variation under control, Scaling of oxide thickness increases the gate tunneling currents, so there is limit on oxide thickness reduction as oxide thickness. The order of oxide thickness is around 1 nm (few atomic layes), therefore even a small variation one atomic, have a severe impact on oxide capacitance as well as threshold voltage and mobility .

Scaling of oxide thickness increases the gate tunneling currents, which leads to increase in leakage currents. However, gate oxide leakage has limited by the introduction high-K dielectrics. New gate oxide material not only control the gate leakage currents, but also reduces the impact of variability on oxide capacitance due to much larger physical oxide thickness .

### 2.4.4. Mobility

Mobility is generally defines as how freely charge carriers either electrons or holes can travel through the channel of the MOSFET in response to the applied electric field as shown in equation below

$$
\begin{equation*}
\mu_{n, p}=\frac{q \tau_{c}}{2 m_{n, p}} \tag{2.6}
\end{equation*}
$$

Where q is the electronic charge, $\tau_{c}$ is the mean free time between carrier collisions, and $m_{n, p}$ is the effective mass of either of an electron or a hole. However, mobility generally considered as a function of doping concentration. Since the mean free time between two collision is determined by doping concentration and somehow on
effective mass. Therefore, any process steps, which, affects doping concentration or stress, will affect transistor mobility. Ion implantation and annealing affect mobility as these process steps determine doping concentration.

## 3. Propagation delay model <br> developed

### 3.1. Overview

The number of previous research work targeted the definition of a compact model for the (deterministic) propagation delay of a CMOS stage. The first representative example is [5], where the propagation delay of a CMOS inverter is estimated as a function of the alpha-power saturation current law in a CMOS transistor; yet, only a very approximate empirical model of the effect of the input slew time is developed. A more complex, charge-based analytical model was developed in [6], still limited to the single inverter propagation delay. An empirical extension to a more complex gate, based on transistor stacks, was introduced in [7]. Finally, a current-based statistical propagation delay model for single cells was illustrated in [8], showing an analytical approach to obtain a statistical behavior computation. All of the above studies assume a known input slew time and a known load capacitance value. When analyzing the multicell paths, in addition to the cell propagation delay, the cell output slew time estimation is essential as it affects the propagation delay of the subsequent cell. An analytical model dedicated to output slew time was reported in [9]. Even more importantly, in a multicell path the input capacitance of each cell
input pin must be properly estimated, as it represent the load capacitance for the preceding cell, and its physical value depends on the actual voltages of all the input pins of the cell. Finally, the logical effort model [10]is a widely adopted paradigm for reasoning on optimal multistage circuit sizing, but it is originally conceived for manual optimization and is inherently a simplified fully linear model, explicitly neglecting transistor-stacks diffusion capacitances, Miller and feed-through effects, output slew time, and assuming a simple load capacitance model. The proposed approach is intrinsically more general than [9], [6],[5], [7], and differently from [10] it addresses the highest possible accuracy in modeling nonlinear effects and parasitic effects. It also differs from [8] as we do not develop an ad hoc statistical model for every single cell, but for specific entities (logic drivers) that can be combined to model virtually any CMOS cell, by means of HDL specifications. Also, the approach diverges from statistical static timing analyzers, such as PrimeTime VX , as it is a statistical timing simulation model, allowing the designer to see statistical effects on the operation of the digital system on actual data. As such, the proposed approach captures the data-dependent dynamic behavior of actual propagation delays, which are normally not caught by a static analysis. In addition, the proposed approach includes a dynamic input capacitance model that is not featured in standard propagation delay format-based tools like PrimeTime VX. On the other hand, the proposed approach requires to run trace vector simulation to obtain path propagation delay calculation, hence activity can affect the quality of the results, and the availability of a set of data trace covering all possible cases of interest can be a limitation. The proposed approach is complementary with respect to recent previous works on process variability analysis. In [11], an analytical methodology was introduced for statically computing the probability density function of the total propagation delay in a multicell path, while the proposed approach addresses the
logic-level simulation of multicell paths as a composition of single-cell behaviors. In [12], an analysis of the SPICE parameters affected by process variations was carried out, which is interesting for the characterization phase of the proposed approach. In [13], the focus was on the impact of intracell mismatch on single cell propagation delay, while the proposed approach is best suited to cell-to-cell variation in multicell designs. An approach integrating Random SPICE within Monte Carlo static timing analysis was illustrated in [14]. The Monte Carlo SPICE simulations used for comparison of the results of the proposed approach are run by NGSPICE BSIM4 [15], [16].

### 3.2. Deterministic Propagation Delay Estimation Model

In our present implementation, we targeted the propagation delay of generic CMOS logic stages subject to single- input switching for critical path analysis; the method can be extended to multiple input switching. To exactly define the logic-driver-based propagation delay model suitable for any generic cell circuit topology, we identify the following basic terms:

- Definition 1: We refer as switching element to either an N-type transistor, or a P-type transistor, or a transmission-gate. Any switching element has a single input control terminal (gate terminal of the transistor).
- Definition 2: In a single-CMOS-stage digital cell, a current driver is any chain of switching elements connecting the output node to the supply node, or to the ground node, or to a primary input of the cell (the latter case occurring in pass-transistor/transmission-gate logic). In a CMOS logic cell, several current drivers can be identified (Figure 3.1). Depending on the voltage values on the


Figure 3.1.: Four current drivers in a cell and associated logic drivers.
input terminals, a current driver may conduct a current to/from the output node of the cell and therefore pull up/down the output voltage. When a current driver starts conducting as a consequence of the input switching, we call it active driver; when a current driver stops conducting as a consequence of the input switching, we call it passive driver. In any CMOS logic single stage, for each possible input switching there are an active driver and (usually) a passive driver. It is possible to abstract the operation of the current drivers within a cell as the operation of virtual logic units, each corresponding to a current driver as shown in Figure 3.1. The general formal definition of such logic drivers follows:

- Definition 3: Given a current driver composed of N switching elements, a logic driver is a logic unit associated with it, defined by:

1. An output logic signal, corresponding to the drain terminal of the first switching element of the driver, which is always connected to the output node of the cell by definition;
2. A set of N+ 1 logic input signals, one for each control terminal (gate terminals) of the switching elements plus one corresponding to the source terminal of the final switching element;
3. A set of $\mathrm{N}+1$ propagation delay values, one per each input-output signal pair;
4. A logic behavior expressing the relation between the set of $\mathrm{N}+1$ input signals and the output signal.

The logic value returned by a logic driver corresponding to a current driver made of only one nMOS switching element, has the following basic form:

```
if gate = '1' or gate = 'H'
            then return source;
elsif gate = '0' or gate = 'L' or gate = 'U'
    then return 'Z';
elsif gate = 'X' or gate = 'W'
    then return ' X';
elsif gate = 'Z'
    then return ' -';
```

Such untimed logic behavior can be recursively extended to the case of logic drivers corresponding to N switching elements. In general, a cell behavior can always be described as a composition of logic drivers, expressed using a HDL code. When an input signal transition occurs in a cell, the logic driver corresponding to the active driver which drives the output logic transition with a certain propagation delay.

### 3.2.1. Single stage

The proposed approach computes the propagation delays associated with a logic driver through the circuit model as shown in Figure 3.2. The term $i_{\text {drive }}(t)$ designates


Figure 3.2.: Equivalent circuit for the propagation delay model.
the total current resulting as the algebraic sum of the currents flowing through the active driver and the passive driver. Except for the simplest cases, in a CMOS logic single stage there are transistors which are neither on the active nor on the passive driver. Because of several effects (e.g., Miller, feed-through), the transistor parasitic capacitances located on the active and passive drivers have a different impact on
the propagation delay with respect to the transistor parasitic capacitances located outside the active and passive drivers. The three capacitors in Fig. 3 correspond to such different parasitic capacitances. $C_{F A N O U T}$ accounts for capacitive load connected to the output, i.e., basically fan-out capacitance. $C_{\text {INTRINSIC }}$ accounts for the capacitances associated with the drain terminals of the transistors inside the cell but outside the active and passive drivers, whose voltage switches as a consequence of the input switching (thus contributing to the cell propagation delay). We refer to such parasitic capacitances as intrinsic capacitances. Physically, they are diffusion capacitances and diffusion-metal contacts capacitances. Finally, $C_{D R I V E}$ accounts for the capacitances associated with the drain terminals on the active or passive drivers, whose voltage switches as a consequence of the input switching (thus contributing to the propagation delay). We refer to such parasitic capacitances as drive capacitances. Physically, they correspond to diffusion capacitances, diffusion-metal contacts capacitances and virtual additional Miller-effect capacitances. In addition, the discharge time of $C_{D R I V E}$ considers the voltage feed-through phenomenon and its effect on the total rising/falling propagation delay of the drain voltage, so that $C_{\text {DRIVE }}$ partially corresponds to a formal quantity rather than a measurable physical capacitance. According to Figure 3.2 we have the following:

$$
\begin{equation*}
i_{\text {drive }}(t)=\left(C_{\text {INTRINSIC }}+C_{\text {FANOUT }}+C_{\text {DRIVE }}\right) \cdot \frac{\partial V_{\text {out }}(t)}{\partial t} \tag{3.1}
\end{equation*}
$$

Hence by integrating and applying the mean value theorem, we obtain that the propagation delay $T_{p d}$ corresponding to an output voltage swing $V_{S}$ is

$$
\begin{equation*}
T_{p d}=\frac{V_{S} \cdot\left(C_{I N T R I N S I C}+C_{F A N O U T}+C_{D R I V E}\right)}{I_{A V G}} \tag{3.2}
\end{equation*}
$$

where $I_{A V G}$ is the average value of $i_{\text {drive }}(t)$ in the time interval $\left[0, T_{p d}\right] . V_{S}$ is usually $V_{D D} / 2$ for standard cell library propagation delay characterization.

We can expand the intrinsic capacitance value as follows:

$$
\begin{equation*}
C_{I N T R I N S I C}=C_{I_{m i n}} \cdot \sum_{j}\left(X \cdot W_{I}(j)-a\right) \tag{3.3}
\end{equation*}
$$

where $W_{I}(j)$ is a weight expressing the width of every transistor $j$ contributing to the intrinsic capacitance, normalized to the minimum width, $a$ is a constant expressing the difference between drawn width and effective width and can be derived from technology data, $C_{\text {Imin }}$ is the intrinsic capacitance contributed by a minimum size transistor, and $X$ is the scaling factor of the cell with respect to the minimum size template for that cell.

We can expand the drive capacitance value as follows:

$$
\begin{equation*}
C_{D R I V E}=C_{D_{\min }} \cdot \sum_{j}\left(X \cdot W_{D}(j)-a\right) \tag{3.4}
\end{equation*}
$$

where $W_{D}(j)$ is a weight expressing the width of every transistor $j$ contributing to the drive capacitance and $C_{D \text { min }}$ is the intrinsic capacitance contributed by a minimum size transistor.

Furthermore we recall that it is a common practice in standard cell characterization to express the fan-out capacitance as a multiple of a reference standard load, i.e., as the ratio between $C_{F A N O U T}$ and a reference minimal gate capacitance $C_{g m i n}$ in the given technology. By defining the quantities:

$$
\begin{aligned}
& \tau_{D}=\frac{V_{S} \cdot C_{D_{\min }}}{I_{\text {avg }}} \\
& \tau_{I}=\frac{V_{S} \cdot C_{I_{\min }}}{I_{\text {avg }}}
\end{aligned}
$$

$$
\tau_{O}=\frac{V_{S} \cdot C_{O_{\min }}}{I_{\text {avg }}}
$$

we obtain the following expression of the propagation delay associated with a specific active/passive driver pair and a specific input switching:

$$
\begin{equation*}
T_{p d}=\tau_{D} \cdot \sum_{j}\left(X \cdot W_{D}(j)-a\right)+\tau_{I} \cdot \sum_{j}\left(X \cdot W_{I}(j)-a\right)+\tau_{O} \cdot \frac{C_{F A N O U T}}{C_{g_{\min }}} \tag{3.5}
\end{equation*}
$$

The technology-dependent timing parameters $\tau_{D}, \tau_{I}$ and $\tau_{O}$ can be determined by characterizing all the possible pairs of active and passive drivers of interest for the cell library to be characterized, for each possible switching input of the active driver. Inside a cell, we can say that certain active and passive drivers occur for each input pattern. A certain $\tau_{D}, \tau_{I}$ and $\tau_{O}$ value, obtained from characterizing an active and a passive driver, is applied to all the cells in which those active and passive drivers can occur. In our present project we focused on the active/passive driver pairs listed in Table 3.1, which allow modeling any CMOS cell having not more than four stacked transistors, referring to worst-case single input switching conditions. The procedure to characterize all the parameters that appear in Equation 3.5 relies on SPICE BSIM4 simulations of ad hoc circuit structures based on the selected active/passive driver pairs.

### 3.2.1.1. characterizing parameter of model

The simulation setup for characterizing $\tau_{D}$ and $\tau_{O}$ is shown in Figure 3.3 (top). According to Equation 3.5, the propagation delay of the active/passive driver pair is modeled as follows:

$$
\tau_{D}\left(C_{F A N O U T} ; X ; t_{\text {sleew }_{i n}}\right) \cdot \sum_{j}\left(X \cdot W_{D}(j)-a\right)+\tau_{O}\left(C_{F A N O U T} ; X ; t_{\text {sleew }}^{i n} \text { }\right) \cdot \frac{C_{F A N O U T}}{C_{g_{m i n}}}
$$

Table 3.1.: Active and passive driver pair for model calibration

| Active driver | Passive driver | No. of input switching cases to be characterized |
| :---: | :---: | :---: |
| 1 NMOS | none | 1 |
| 1 TG | None | 1 |
| 2 TG | None | 1 |
| 1 NMOS | 1 PMOS | 1 |
| 1 NMOS | 2 PMOS | 2 |
| 1 NMOS | 3 PMOS | 3 |
| 1 NMOS | 4 PMOS | 4 |
| 2 NMOS | 1 PMOS | 2 |
| 2 NMOS | 2 PMOS | 4 |
| 2 NMOS | 3 PMOS | 6 |
| 3 NMOS | 1 PMOS | 3 |
| 3 NMOS | 2 PMOS | 6 |
| 3 NMOS | 3 PMOS | 9 |
| 4 NMOS | 1 PMOS | 4 |
| 1 PMOS | 1 NMOS | 1 |
| 2 PMOS | 1 NMOS | 2 |
| 3 PMOS | 1 NMOS | 3 |
| 4 PMOS | 1 NMOS | 4 |
| 1 PMOS | 2 NMOS | 2 |
| 2 PMOS | 2 NMOS | 4 |
| 3 PMOS | 2 NMOS | 6 |
| 1 PMOS | 3 NMOS | 3 |
| 2 PMOS | 3 NMOS | 6 |
| 3 PMOS | 3 NMOS | 9 |
| 1 PMOS | 4 NMOS | 4 |
|  |  |  |

where the sum is constant for the given driver pair and for the chosen input pin. The parameter $C_{g \text { min }}$ can be chosen as the input capacitance of a minimal inverter under nominal conditions. By increasing $C_{\text {FANOUT }}$ of a quantity $\delta C$ small enough to consider $\tau_{D}$ and $\tau_{O}$ as unaffected, we measure a propagation delay increase given by the following:

$$
\begin{equation*}
\delta T_{p d}=\tau_{O}\left(C_{F A N O U T} ; X ; t_{\text {sleewin }_{i n}}\right) \cdot \frac{\delta C}{C_{m i n}} \tag{3.7}
\end{equation*}
$$



Figure 3.3.: Simulation setup for model parameter calibration

Therefore, we derive the value of $\tau_{O}$ and consequently the value of $\tau_{D}$ by substitution. The simulation setup for characterizing $\tau_{I}$ is shown in Figure 3.3 (bottom). The measured propagation delay of the active/passive driver pair with an additional Ntype (P-type) transistor connected between the output node and ground $\left(V_{d d}\right)$ is modeled by the following:

$$
\begin{aligned}
& T_{p d}=\tau_{D}\left(C_{F A N O U T} ; X ; t_{\text {sleew }_{i n}}\right) \cdot \sum_{j}\left(X \cdot W_{D}(j)-a\right) \\
& +\tau_{I\left(C_{\text {FANOUT } ; X ; t_{\text {sleew }}^{i n}}\right)} \cdot \sum_{j}\left(X \cdot W_{I}(j)-a\right)
\end{aligned}
$$

$$
\begin{equation*}
+\tau_{O}\left(C_{F A N O U T} ; X ; t_{\text {sleew }_{i n}}\right) \cdot \frac{C_{F A N O U T}}{C_{g_{m i n}}} \tag{3.8}
\end{equation*}
$$

### 3.2.2. Multi stage

To be able to calculate the propagation delay of a chain of cells, the premier task is to compute the slew time of input signal and the load capacitance for all stages. The output slew time of one stage will become the input signal of the upcoming cell and so on. To estimate the slew time of its input signal and its load capacitance for the n-th stage, it is possible to extrapolate both input signal and load capacitance from the multi-stage circuit. Considering a sub-circuit from multi-stage circuit to limit it to a single stage and calculate its propagation delay. Follow the same methodology for all sub-circuitry and add all propagation delays of sub-circuits to find the resultant propagation delay of a required multi-stage as shown in the following figure (Figure 3.4).

### 3.2.3. Slew time

We measured the slew time of voltage transitions as the time for passing from $20 \%$ to $80 \%$ of full voltage swing. The impact of the input slew time on the propagation delay occurs as a modification in the average current driven by the active and passive driver, thus ultimately affecting the timing functions $\tau_{D}, \tau_{I}$ and $\tau_{O}$. Our analyses showed that such dependence on the slew time is not univocal for different values of the load capacitance, and it is therefore very difficult to capture it in a compact


Figure 3.4.: multistage path
yet accurate model. To accurately consider the effect, we chose to characterize the timing functions $\tau_{D}, \tau_{I}$ and $\tau_{O}$ associated with each active/passive driver pair for different values of input signal slew time, ranging from 1 to 100 ps , which are compatible with practical cases in the given $45-\mathrm{nm}$ technology. (In SPICE BSIM4 simulation the output slew time of a single-stage cell results below 90 ps in case of load/input capacitance ratio $=30$, which is a very conservative case with respect to practical usage of standard cells in real designs [10]). The computation of the output slew time is important for multistage paths, as the output slew time of a cell affects the propagation delay of the subsequent cell in the path. The slew time is mainly related to the slope of the voltage transition in its central partmostly dependent on $\tau_{I}$ and $\tau_{O}$ values and scarcely to the initial shape of it-mostly affected by the parasitics modeled by $\tau_{D}$. Therefore, we conjectured and extensively verified that the output signal slew time tr be a linear function of the input slew time tslew in
and of the sole contribution of $\tau_{I}$ and $\tau_{O}$ to the propagation delay value

$$
\begin{equation*}
t_{\text {slew }_{\text {out }}}=\alpha+\beta \cdot t_{I, O}+\gamma \cdot t_{\text {slew }_{I N}} \tag{3.9}
\end{equation*}
$$

where the auxiliary variable $t_{I, O}$ is

$$
t_{I, O}=\left(\tau_{I} \cdot \sum_{j}\left(X \cdot W_{I}(j)-a\right)+\tau_{O} \cdot \frac{C_{F A N O U T}}{C_{g_{m i n}}}\right)
$$

$\beta$ and $\gamma$ have practically the same two values for all the cells in the given technology, while $\alpha$ is characteristic of each active/passive driver pair, independently from X . The typical fitting of the above model and the actual slew time measured from SPICE circuit simulations is shown in Figure 3.5.


Figure 3.5.: Behavior of output slew time vs the quantity $t_{I_{-} O}$.

### 3.2.4. Load capacitance

A particularly challenging issue in accurately modeling the propagation delay of multistage paths composed of an arbitrary number of CMOS cells is the accurate model of the input pin capacitance in each cell, which acts as the load capacitance
of the preceding cell in the path. The MOSFET parasitic capacitances contributing to the total input capacitance of a cell vary their value during voltage transitions [1], hence logic level propagation delay models always rely on an average value that we refer to as equivalent input capacitance, to be properly characterized. A commonly adopted setup to characterize the equivalent input capacitance of a target cell is shown in Figure 3.6: by comparing the propagation delay of a reference cell


Figure 3.6.: Input pin capacitance characterization setup.
(inverter) driving the target cell input pin and driving a known capacitor, it is possible to determine the equivalent capacitance value for the input pin of the target cell. As a result of our analysis, the equivalent capacitance $C_{I N}$ associated with the input pin of a generic CMOS cell depends on the following factors:

1) voltage transition (high-low and low-high);
2) logic state of the other input pin of the target cell (e.g., node B logic value, in Figure 3.6);
3) slew time of the input signal of the driver cell (e.g., node Y slew time, in Figure 3.6).

A sample of the SPICE results for the equivalent input capacitance characterization for different target cells, evidencing in particular the dependence on the second and third factors (the first factor is widely recognized in commercial cell library


Figure 3.7.: SPICE characterization of input pin capacitance (two-input AND cell) with respect to input slew of the driver cell and to different input logic patterns of the target cell.
characterization files) are shown in Figure 3.7. Similar outcomes are obtained for all types of cells. The third factor, in particular, might seem surprising if we think of the input capacitance as a property of the sole target cell, but in reality the equivalent input capacitance results from the coupling of the target cell and its driver. Interestingly, we found that the slew time of the input signal of the driver has a nonnegligible impact on the equivalent input capacitance of the target cell, while the type of driver cell does not have a significant impact in this respect, except for very rare cases.

An impact of the driver slew time up to $20 \%$ on the equivalent input capacitance is found. Importantly for the propagation delay computation algorithm, implementing the slew time effect referring to the input of the driver cell (e.g., node Y slew time in Figure 3.6) is more efficient than referring to the input of the target cell (e.g., node A slew time in Figure 3.6): the latter case would imply an iterative computation at simulation time, because the target cell input slew time affects the equivalent

Table 3.2.: Sample of database record. AND cell (input IN1 with IN2='0')

| cell size <br> factor | driver <br> slew (ps) | pin cap. <br> HL (fF) | pin cap. <br> LH (fF) | pin IN1 logic value | pin IN2 logic <br> value |
| :--- | :---: | :---: | :---: | :---: | :---: |
| X1 | 10 | 0,16 | 0,17 | pin under test | LOW |
| X1 | 50 | 0,17 | 0,18 | pin under test | LOW |
| X2 | 10 | 0,35 | 0,36 | pin under test | LOW |
| X2 | 50 | 0,36 | 0,38 | pin under test | LOW |
| X3 | 10 | 0,53 | 0,56 | pin under test | LOW |
| X3 | 50 | 0,55 | 0,58 | pin under test | LOW |
| X4 | 10 | 0,72 | 0,75 | pin under test | LOW |
| X4 | 50 | 0,73 | 0,77 | pin under test | LOW |
| X5 | 10 | 0,91 | 0,95 | pin under test | LOW |
| X5 | 50 | 0,92 | 0,97 | pin under test | LOW |
| X10 | 10 | 1,83 | 1,92 | pin under test | LOW |
| X10 | 50 | 1,84 | 1,94 | pin under test | LOW |
| X20 | 10 | 3,68 | 3,86 | pin under test | LOW |
| X20 | 50 | 3,69 | 3,87 | pin under test | LOW |

load capacitance seen by the driver cell, which in turns affects the target cell input slew time. While the distinction of voltage transition is commonly supported by conventional propagation delay calculators integrated in EDA tools, to the best of our knowledge the impact of the other input pin logic values of the target cell and the impact of the input slew time of the driver cell are not supported yet. We implemented the characterization of the equivalent input capacitance by storing a pair of capacitance values, corresponding to 10 and 50 ps driver slew time, for each input pin, transition direction, and logic pattern on the other input pins. For different slew time values we perform a linear interpolation given the reference pair of values. By this simple approach, we halve the propagation delay error due to imprecise input capacitance assessment. The same approach is feasible for accurately characterizing and modeling the behavior of the equivalent capacitance associated with complex interconnect loads; in the present version of the project they are modeled as capacitive loads based on conventional parasitic extraction tables.

A characterization database has been developed for the whole cell library and inte-

Table 3.3.: Sample of database record. AND cell (input IN1 with IN2='1')

| cell size <br> factor | driver <br> slew (ps) | pin cap. <br> HL (fF) | pin cap. <br> LH (fF) | pin IN1 logic value | pin IN2 logic <br> value |
| :--- | :---: | :---: | :---: | :---: | :---: |
| X1 | 10 | 0,17 | 0,18 | pin under test | HIGH |
| X1 | 50 | 0,20 | 0,19 | pin under test | HIGH |
| X2 | 10 | 0,39 | 0,38 | pin under test | HIGH |
| X2 | 50 | 0,42 | 0,41 | pin under test | HIGH |
| X3 | 10 | 0,60 | 0,59 | pin under test | HIGH |
| X3 | 50 | 0,63 | 0,61 | pin under test | HIGH |
| X4 | 10 | 0,82 | 0,79 | pin under test | HIGH |
| X4 | 50 | 0,85 | 0,82 | pin under test | HIGH |
| X5 | 10 | 1,04 | 1,00 | pin under test | HIGH |
| X5 | 50 | 1,07 | 1,03 | pin under test | HIGH |
| X10 | 10 | 2,14 | 2,04 | pin under test | HIGH |
| X10 | 50 | 2,16 | 2,07 | pin under test | HIGH |
| X20 | 10 | 4,43 | 4,12 | pin under test | HIGH |
| X20 | 50 | 4,45 | 4,13 | pin under test | HIGH |

grated in the VHDL model of the library. The Table 3.2 and Table 3.3 are a sample of the database record referring to the AND2 cell for an input. The other input ...

The other importan think is why calculate capacitance for all combination

### 3.3. Statistical Propagation Delay Estimation Model

The significance of the logic driver paradigm, based on the behavior of the $\tau_{D}, \tau_{I}$ and $\tau_{O}$ functions and on the novel characterization of the input capacitances, is notable when we take into account technology variability. The proposed model allows the timing simulation of a digital design (connection of cells) as a logic-level event-driven simulation. As such, the statistical simulation of an entire digital design can be accomplished by logic-level Monte Carlo simulation in which variations are introduced in the $\tau$ function values and capacitance function values associated with the cells. If the same variations are applied to $\tau$ functions and capacitances in all the cells
at each Monte Carlo iteration, we perform a die-to-die, global variation statistical analysis; if an individual variation is applied to $\tau$ functions and capacitances in each cell at each Monte-Carlo iteration, we perform intradie, local variation analysis. In this research work, the implementation of global process variation analysis is the target. We explain the present implementation and the concept for implementing local mismatch analysis within the same paradigm. To explore the behavior of the $\tau_{D}, \tau_{I}$ and $\tau_{O}$ timing functions and of the input pin equivalent capacitances in presence of technology variations, we built an automated script that iteratively runs the characterization procedure, through SPICE BSIM4 simulations, with random variations in technology parameters at each iteration. The variations considered are L, W, oxide thickness Tox, and channel doping Ndep. The behavior we observed is that only a vertical shift of the $\tau_{D}, \tau_{I}$ and $\tau_{O}$ functions is significantly affected by a technology variation (Figure 3.8). The vertical shift results to be relatively


Figure 3.8.: Behavior of to as affected by L (transistor drawn length) variation. Other technology variations have a similar effect.
dependent on the input signal slew time, and very modestly dependent on the type of driver pair (Table 3.1) and the size factor of the circuits. For input pin capacitances, we observed that the effect of a variation in technology parameters turns into a multiplying factor common to all the cells in the library. Accordingly, the
timing function values and the pin capacitance value of a cell subject to statistical process variations can be expressed as follows:

$$
\begin{aligned}
& \tau_{D}=\tau_{D_{\text {nominal }}}+\Delta \tau_{D} \\
& \tau_{I}=\tau_{I_{\text {nominal }}}+\Delta \tau_{I} \\
& \tau_{O}=\tau_{O_{\text {nominal }}}+\Delta \tau_{O} \\
& C_{I N}=C_{I N_{\text {nominal }}}\left(1+\Delta C_{I N}\right)
\end{aligned}
$$

where the shift values $\tau_{D}, \tau_{I}, \tau_{O}$ and $C_{I N}$ are random variables to be statistically characterized.

### 3.3.1. Global Variation Analysis Implementation

For the implementation of global variation analysis, we assumed a Gaussian distribution of the parameters L, W, $T_{o x}$, and $N_{\text {dep }}$, with $3 \sigma$ variation of $15 \%$, widely used in statistical CMOS simulations. By running the characterization scripts for 10000 times with pseudo-random generated parameters, we collected and stored a vector ( $\tau$ variation vector) of 10000 shift values $\tau_{D}, \tau_{I}$ and $\tau_{O}$, referring to input slew times of 10 and 50 ps , for a total of 60000 sample shift value arrays, which are valid for any cell in the given technology. We furthermore collected a vector (pincap variation vector) of 10000 values of $C_{I N}$, usable for any cell of the library in the given technology. The full statistical characterization is carried out at 1 V supply. As the variations in the timing functions with respect to process variations are directly sampled from SPICE simulation and stored with no fitting function, it is expectable that any nonlinear behavior of timing function variations will be captured at any voltage, provided that a sufficient high number of Monte Carlo
samples are used. We implemented a Monte Carlo analytical propagation delay calculator that repeatedly computes the propagation delay of given circuit, according to Equation 3.5, applying the 10000 -element $\tau$ variation vector and pincap variation vector and obtaining the propagation delay statistical behavior. We furthermore integrated the Monte Carlo iterations in a very-high-speed-integrated-circuit hardware description language (VHDL) environment, allowing the statistical simulation of any design based on our VHDL cell library models, thus enabling Monte Carlo analysis on a fast logic level event- driven simulator.

### 3.3.2. Extension to Local Variation Analysis

When we have to consider local mismatch variations, the shift of the tau function values and multiplying factor of the pin capacitance value of a given cell can be expressed as follows:

$$
\begin{aligned}
& \tau_{D}=\tau_{D_{\text {nominal }}}+\Delta \tau_{D_{\text {global }}}+\Delta \tau_{D_{\text {local }}} \\
& \tau_{I}=\tau_{I_{\text {nominal }}}+\Delta \tau_{I_{\text {global }}}+\Delta \tau_{I_{\text {local }}} \\
& \tau_{O}=\tau_{O_{\text {nominal }}}+\Delta \tau_{O_{\text {global }}}+\Delta \tau_{O_{\text {local }}} \\
& C_{I N}=C_{I N_{\text {nominal }}}\left(1+\Delta C_{I N_{\text {global }}}+\Delta C_{I N_{\text {local }}}\right)
\end{aligned}
$$

The terms $\Delta \tau_{\mathrm{D}_{\text {global }}}, \Delta \tau_{\mathrm{I}_{\text {global }}}, \Delta \tau_{\mathrm{O}_{\text {global }}}$ and $C_{\mathrm{IN}_{\text {global }}}$ are global, design-independent process variations and can be characterized as shown above. The terms $\Delta \tau_{\mathrm{D}_{\text {local }}}$, $\Delta \tau_{\mathrm{I}_{\text {local }}}, \Delta \tau_{\mathrm{O}_{\text {local }}}$ and $C_{\mathrm{IN}_{\text {local }}}$ are specific for each cell instance, and must be generated for a placed and routed design. We can include them in the logic-level Monte Carlo simulation as random variables generated at run-time with spatialdependent variance. The spatial-dependent variance of $\Delta \tau_{\mathrm{D}_{\text {local }}}, \Delta \tau_{\mathrm{I}_{\text {local }}}, \Delta \tau_{\mathrm{O}_{\text {local }}}$ and $C_{\mathrm{IN}_{\text {local }}}$
can be a priori characterized as a function of geometric distance between generic cells, by a design-independent Monte-Carlo SPICE characterization with random variations in the SPICE device parameters according to the established distancedependent mismatch models like Pelgrom's rule [17] and its extensions. In the present implementation of the cell library model, the distance dependent characterization of the variance of $\tau_{D}, \tau_{I}, \tau_{O}$ and $C_{I N}$ is not implemented but it does not imply any modification to the propagation delay model and pin capacitance model (it only affects the values passed as $\tau$ functions and $C_{I N}$ at each iteration). A limitation of the proposed approach, due to the inherent structure of the proposed propagation delay model, is that local transistor mismatch inside a current driver within a cell cannot be modeled; thus spatial variation analysis is feasible for cell-to-cell variations or at most for intracell driver-to-driver variations.

### 3.4. Model Implementation

The present implementation of the logic driver paradigm is developed in VHDL for a library based on 21 different logic elements, each modeled with eight sizes, totaling 168 standard cells modeled and verified. The database structure of the simulation environment, with storage usage details, is shown in Figure 3.9.

The executable model of a standard cell is composed of three code sections devoted to propagation delay computation, logic operation and input capacitance computation, respectively, according to Figure 3.10.

The model relies on a resolved type logic_drive_logic for representing logic signals, which is a record carrying the logic value (std_logic levels), the slew time, and a pair of minimum/maximum capacitance values associated with the input pin of


Figure 3.9.: Database structure for the logic-driver-based timing simulation environment. Arrows indicate dependencies.
a cell. The type is resolved with respect to conflicts in logic levels in a similar way to std_logic type, and it is also resolved with respect to capacitance values so that several signals connected to the same net will sum up their capacitance values. The propagation delay computation section relies on propagation delay parameter functions tau_d, tau_i, tau_o (corresponding to $\tau_{D}, \tau_{I}$ and $\tau_{O}$ ) defined for each of the active/passive driver pairs to be considered in the cell library. The values returned by these functions depend on the transistor size factor, the input slew time, and the fan-out capacitance. The values are stored in constant arrays of real numbers during the technology characterization phase. The logic operation section relies on signal drive functions Nmos_kdrive(), Pmos_kdrive(), implementing the logic behavior of logic drivers and applying the propagation delays calculated in the preceding section. Logic drivers ranging from a single input signal $(k=1)$ up to four input signals $(\mathrm{k}=4)$ are modeled, for both pMOS active drivers and nMOS active drivers. The input capacitance section relies on a c_in() function associated with each input of a cell, returning a pair of reference capacitance values. The two
$\begin{array}{ll}\text { entity nand2_dut is } & 1 \\ 2\end{array}$
generic (X:real:=1.0); —-resize factor 3
port (A, B: inout logic_drive_logic; 4
Z: inout logic_drive_logic); 5
end nand2_dut; 6
architecture abstract of nand2_dut is 10
- here: internal signal declarations, 11
-- transistor size definitions, etc. 12
begin 13
- delay computation section
14
- delay values computed for each 15
- possible logic driver activation 16
-- example: 17
time_ld1_A_on $<=\ldots$... call of tau functions 18
- logic operation section — 19
- all logic drivers in cell 20
-- example: 21
ld1: Nmos_2drive(Z, A, B, time_ld1_A_on, ..); 22
- input capacitance computation section — 23
A.pincap $<=\ldots-$ call of c_in functions 24
B.pincap $<=\ldots-$ call of c_in functions 25
end abstract; 26
27

Figure 3.10.: Basic scheme of standard cell description.
values correspond to the input capacitance measured in the cell when the input slew time of another cell driving the pin is 10 and 50 ps respectively. The values returned by C_i n() also depend from the logic states of the other input pins of the same cell and from the cell size factor, and are stored in constant arrays of real numbers during the technology characterization phase. Example of the operation of the model in a multicell path (Figure 3.11): the VHDL code section devoted to propagation delay computation in cell C 1 reads the total capacitance on signal Z as a pair of reference values $\{3$ and 6 fF$\}$ resulting as the sum of the capacitances


Figure 3.11.: Implementation of the input pin capacitance simulation model.
connected to Z. According to the actual input slew time 30 ps on the switching node A, cell C 1 computes its actual load capacitance to be used with the tau functions as a linear interpolation ( 4.5 fF ). An interesting feature is that-because of the event-driven operation of HDL languages - all the quantities in the cell model (e.g., capacitances and timing functions) are recalculated on demand, i.e., only when some of the involved signals changes.

### 3.5. Summary

We have discussed the overall methodologies of deterministic and statistical propagation delay estimation techniques. The deterministic propagation delay estimation is divided into the conceptual definitions and detail description of parameters used for single and multi stages. The estimation methodology for slew time and load capacitate is discussed later in detail by basic formulations and SPICE calculations. Furthermore, the statistical propagation delay estimation methodology has been introduce with both global and local variation injection. Global variations considers broadly in many parameters in which we have considered oxide thickness and de-
pletion only. For local process variations we have considered the threshold voltage variations which affects the propagation delay of nano-meter technologies in a big extent. Lastly, the model framework has been introduced in detail which shows the methodology and algorithm in detail for estimating the propagation delay for all possible discussed techniques.

# 4. Results on deterministic propagation delay prediction in nominal conditions 

### 4.1. Overview

This chapter will elaborate the implementation in detail in the terms of results and its analysis. The results of deterministic and statistical propagation delay behavior will be given in the form of tables and figures; whereas the methodology has already been explained in the previous chapter. The deterministic propagation delay will be categorized in two types such as deterministic single stage and deterministic multi stage. Furthermore, the statistical propagation delay will be divided in to two forms as well which is in the same manner such as statistical single and multistage propagation delay. These results have been implemented on both transistor-level and logic layers. There are number of circuits ; for example; small scale circuits (such as inverter) to medium level circuits (such as full adders) and to large scale circuits (such as 32-bit FIR filter) have been taken into account for propagation delay estimation simulations. The analysis will be done following by every result sub section.

### 4.2. Deterministic single stage

The detailed definition and methodology of deterministic single stage has been explained in chapter 3. The methodology for this work is implemented in VHDL and verified with SPICE simulations. The implementation is being done on various small and medium scale circuits. These circuits are mainly Inverter, NOT, NAND and Full Adders with different functionalities. Different capacitance loads, sizes, input slew times will be taken as input in the following results as well. The percentage of error and comparison between proposed techniques in terms of relative errors will be accounted as the output results. At the end of deterministic single stage an analysis on all results will be discussed.

### 4.2.1. inverter

The first cell which taken into account is inverter. The analysis has been divided into two cases high-to-low and low-to-high transactions for number of different cases. For both transitions, 340 combinations of cell size factor, input slew time and load capacitance has been tested.

Figure 4.1 shows how model support non linear effect of nano-scale CMOS for different input slew time.

Table 4.1 shows a part of the result for different case of cell size factor, input slew time and load capacitance. It reports both absolute and relative errors. The relative error is very less than $1 \%$.


Figure 4.1.: VHDL vs SPICE $t_{L H}$ NOT cell. Input slew time 10ps and 50ps.

### 4.2.2. nand2

The nand2 gate is tested for both input and for all transitions of output for a total of cases equal to 680 (combinations of cell size factor, input slew time and load capacitance).

Table 4.2 shows a part of the results for different case of cell size factor, input slew time and load capacitance for input A (the input A in this case is connected to gate of Nmos with source to ground. For all cases the relative error is less than $1 \%$.

### 4.2.3. nor2

The nor2 gate is tested for both input and for all transitions of output for a total of cases equal to 680 (combinations of cell size factor, input slew time and load capacitance).

Table 4.3 shows a part of the results for different case of cell size factor, input slew time and load capacitance for input A (the input A in this case is connected to gate

Table 4.1.: Absolute and Relative error of Inverter: SPICE vs VHDL comparison

| not | x 1 tr10ps |  | x 1 tr 50 ps |  | x 10 tr10ps |  | x 10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error $\%$ | abs err (ps) | error $\%$ | abs err (ps) | error $\%$ | abs err (ps) | error \% | abs err (ps) |
| 0 | 0.02 | 0.00 | 0.00 | 0.00 | -0.05 | 0.00 | -0.03 | 0.00 |
| 0.15 | -0.02 | 0.00 | 0.00 | 0.00 | 0.01 | 0.00 | -0.02 | 0.00 |
| 0.33 | 0.02 | 0.00 | -0.05 | 0.00 | 0.01 | 0.00 | -0.01 | 0.00 |
| 0.66 | 0.01 | 0.00 | -0.01 | 0.00 | 0.01 | 0.00 | 0.00 | 0.00 |
| 1 | 0.02 | 0.00 | 0.00 | 0.00 | -0.02 | 0.00 | -0.01 | 0.00 |
| 1.25 | 0.01 | 0.00 | 0.02 | 0.00 | 0.02 | 0.00 | 0.02 | 0.00 |
| 1.5 | 0.03 | -0.01 | 0.02 | 0.00 | 0.03 | -0.01 | -0.01 | 0.00 |
| 1.75 | 0.05 | -0.01 | -0.45 | 0.01 | 0.46 | -0.10 | -0.25 | 0.01 |
| 2 | 0.03 | -0.01 | 0.01 | 0.00 | -0.01 | 0.00 | -0.01 | 0.00 |
| 3 | -0.01 | 0.00 | -0.03 | 0.00 | -0.02 | 0.01 | -0.02 | 0.00 |
| 4 | -0.02 | 0.01 | 0.00 | 0.00 | 0.02 | -0.01 | 0.02 | 0.00 |
| 5 | -0.03 | 0.01 | 0.00 | 0.00 | -0.04 | 0.02 | -0.01 | 0.00 |
| 6 | -0.03 | 0.02 | -0.01 | 0.00 | 0.01 | -0.01 | 0.01 | 0.00 |
| 7 | -0.03 | 0.02 | 0.00 | 0.00 | 0.01 | -0.01 | 0.02 | 0.00 |
| 8 | -0.04 | 0.03 | -0.02 | 0.00 | 0.01 | -0.01 | 0.02 | 0.00 |
| 9 | -0.04 | 0.03 | 0.03 | 0.00 | 0.02 | -0.02 | -0.01 | 0.00 |
| 10 | -0.03 | 0.03 | 0.04 | 0.00 | -0.03 | 0.03 | 0.01 | 0.00 |

of Nmos with source to ground. For all cases the relative error is less than $1 \%$.

### 4.2.4. ao12_n

The ao12_n gate is another cell of standar library. It have 3 inputs. The analyses have been performed for all input and for all transitions of output for a total of cases equal to 1360 (combinations of cell size factor, input slew time and load capacitance and logic value of input).

Figure 4.2 shows two cases of input slew time for a different load capacitance.
Table 4.4 shows a part of the results for different case of cell size factor, input slew time and load capacitance for input A (the input A in this case is connected to gate

Table 4.2.: Absolute and Relative error of NAND2: SPICE vs VHDL comparison

| nand2a | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 1 \operatorname{tr} 50 \mathrm{ps}$ |  | x 10 tr10ps |  | x 10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | 0.98 | -0.04 | 0.95 | -0.04 | 0.65 | -0.05 | 0.78 | -0.06 |
| 0.15 | 0.54 | -0.03 | 0.93 | -0.04 | 0.41 | -0.04 | 0.71 | -0.05 |
| 0.33 | 0.38 | -0.03 | 0.84 | -0.03 | 0.29 | -0.03 | 0.57 | -0.04 |
| 0.66 | 0.29 | -0.03 | 0.73 | -0.03 | 0.24 | -0.04 | 0.55 | -0.04 |
| 1 | 0.23 | -0.03 | 0.71 | -0.03 | 0.17 | -0.03 | 0.52 | -0.04 |
| 1.25 | 0.23 | -0.03 | 0.62 | -0.03 | 0.17 | -0.03 | 0.48 | -0.04 |
| 1.5 | 0.17 | -0.03 | 0.71 | -0.04 | 0.16 | -0.04 | 0.42 | -0.04 |
| 1.75 | 0.21 | -0.04 | 0.42 | -0.02 | 0.17 | -0.04 | 0.27 | -0.02 |
| 2 | 0.20 | -0.04 | 0.46 | -0.03 | 0.10 | -0.03 | 0.35 | -0.03 |
| 3 | 0.13 | -0.04 | 0.40 | -0.02 | 0.12 | -0.04 | 0.33 | -0.03 |
| 4 | 0.11 | -0.04 | 0.37 | -0.03 | 0.11 | -0.05 | 0.30 | -0.03 |
| 5 | 0.09 | -0.04 | 0.31 | -0.02 | 0.10 | -0.05 | 0.23 | -0.03 |
| 6 | 0.08 | -0.05 | 0.34 | -0.03 | 0.06 | -0.04 | 0.22 | -0.03 |
| 7 | 0.08 | -0.05 | 0.32 | -0.03 | 0.06 | -0.04 | 0.25 | -0.04 |
| 8 | 0.07 | -0.05 | 0.25 | -0.03 | 0.05 | -0.04 | 0.21 | -0.03 |
| 9 | 0.06 | -0.05 | 0.28 | -0.03 | 0.11 | -0.10 | 0.22 | -0.04 |
| 10 | 0.06 | -0.06 | 0.24 | -0.03 | 0.11 | -0.10 | 0.19 | -0.03 |

of Nmos with source to ground in two Nmos stack. For all cases the relative error is less to $1 \%$.

### 4.2.5. ao22_n

The ao22_n gate is one of the renouned cell of standar library. It have 4 inputs. The simulations have been done and tested for all the inputs and for all transitions of output for a total of cases equal to 1360 (combinations of cell size factor, input slew time and load capacitance and logic value of input). The relative error is less to $1 \%$ for most of the cases, but for only few cases, in particular such as small load capacitance, the relative error is observed less than $3 \%$.

Table 4.5 shows a part of the results for different case of cell size factor, input slew

Table 4.3.: Absolute and Relative error of NOR2: SPICE vs VHDL comparison

| nor2 | x1 tr10ps |  | x1 tr50ps |  | x10 tr10ps |  | x10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error $\%$ | abs err (ps) | error $\%$ | abs err (ps) | error \% | abs err (ps) |
| 0 | -0.06 | 0.00 | 0.67 | -0.01 | 0.02 | 0.00 | 1.20 | -0.02 |
| 0.15 | 0.02 | 0.00 | 0.57 | -0.01 | -0.01 | 0.00 | 0.98 | -0.02 |
| 0.33 | 0.02 | 0.00 | 0.47 | -0.01 | 0.00 | 0.00 | 0.78 | -0.02 |
| 0.66 | 0.01 | 0.00 | 0.32 | -0.01 | 0.01 | 0.00 | 0.49 | -0.02 |
| 1 | -0.01 | 0.00 | 0.31 | -0.01 | -0.01 | 0.00 | 0.41 | -0.01 |
| 1.25 | 0.02 | 0.00 | 0.25 | -0.01 | 0.01 | 0.00 | 0.32 | -0.01 |
| 1.5 | 0.02 | 0.00 | 0.18 | -0.01 | 0.00 | 0.00 | 0.26 | -0.01 |
| 1.75 | 0.01 | 0.00 | -0.45 | 0.02 | 0.31 | -0.06 | -0.01 | 0.00 |
| 2 | 0.01 | 0.00 | 0.17 | -0.01 | -0.01 | 0.00 | 0.22 | -0.01 |
| 3 | 0.00 | 0.00 | 0.14 | -0.01 | 0.00 | 0.00 | 0.15 | -0.01 |
| 4 | 0.00 | 0.00 | 0.12 | -0.01 | -0.01 | 0.00 | 0.12 | -0.01 |
| 5 | 0.00 | 0.00 | 0.08 | 0.00 | -0.01 | 0.01 | 0.10 | -0.01 |
| 6 | 0.00 | 0.00 | 0.07 | 0.00 | 0.01 | 0.00 | 0.06 | -0.01 |
| 7 | 0.00 | 0.00 | 0.07 | 0.00 | 0.00 | 0.00 | 0.12 | -0.01 |
| 8 | 0.00 | 0.00 | 0.04 | 0.00 | 0.01 | -0.01 | 0.07 | -0.01 |
| 9 | 0.00 | 0.00 | 0.05 | 0.00 | 0.01 | -0.01 | 0.04 | 0.00 |
| 10 | 0.00 | 0.00 | 0.03 | 0.00 | -0.02 | 0.01 | 0.04 | -0.01 |

time and load capacitance for input A (the input A in this case is connected to gate of Nmos with source to ground and the Pmos with a source to $V_{D D}$ ).

### 4.2.6. ao31_n

The ao31 n gate is taken from standard cell library. It have 4 inputs as well. The pull-down have two stack in parallel, one with three Nmos and other with one. The results have been achived by testing the analysis for all input and for all transitions of output for a total of cases equal to 2040 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For number of cases the relative error is less than $1 \%$, but for some case, in particular for a small load capacitance, the relative error is less to $2 \%$.

Table 4.6 shows a part of the results for different case of cell size factor, input slew


Figure 4.2.: ao12_n input A. Differente slew time (left 10ps,rigth 50ps)
time and load capacitance for input A (the input A in this case is connected to gate of Nmos with source to ground and the Pmos with a source to $V_{D D}$ ).

### 4.2.7. ao32_n

The ao32_n gate is another cell of standar library. It have also 5 inputs. The pulldown part have two stack in parallel, one with three Nmos transistors and other with two nmos transistors. The results are done by testing the analysis for all input and for all transitions of output for a total of cases equal to 4080 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For number of cases the relative error is less to $2 \%$, but for some cases, in particular for a small load capacitance, the relative error is less than $3 \%$.

Table 4.7 shows a part of the results for different case of cell size factor, input slew time and load capacitance for input A (the input A in this case is connected to gate of Nmos with source to ground and the Pmos with a source to $V_{D D}$ ).

Table 4.4.: Absolute and Relative error of AO12_n: SPICE vs VHDL comparison. Input A

| ao12__n | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 1 \operatorname{tr} 50 \mathrm{ps}$ |  | x 10 tr 10 ps |  | x 10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | -0.6 | -0.02 | -14.5 | -1.06 | 1.0 | 0.03 | -14.7 | -0.96 |
| 0.15 | -1.0 | -0.05 | -7.7 | -0.72 | 0.7 | 0.02 | -13.7 | -0.92 |
| 0.33 | -1.4 | -0.09 | -4.8 | -0.56 | 0.3 | 0.01 | -12.6 | -0.87 |
| 0.66 | -1.9 | -0.18 | -2.9 | -0.43 | 0.0 | 0.00 | -10.9 | -0.80 |
| 1 | -2.1 | -0.24 | -2.2 | -0.39 | -0.3 | -0.01 | -9.6 | -0.75 |
| 1.25 | -1.8 | -0.25 | -1.8 | -0.35 | -0.3 | -0.01 | -8.8 | -0.71 |
| 1.5 | -1.7 | -0.26 | -1.6 | -0.35 | -0.5 | -0.02 | -8.1 | -0.68 |
| 1.75 | -1.6 | -0.27 | -1.0 | -0.25 | -0.7 | -0.03 | -7.7 | -0.67 |
| 2 | -1.4 | -0.27 | -1.1 | -0.29 | -0.6 | -0.03 | -7.0 | -0.62 |
| 3 | -1.0 | -0.26 | -0.9 | -0.29 | -0.8 | -0.05 | -5.4 | -0.54 |
| 4 | -0.9 | -0.30 | -0.7 | -0.29 | -1.1 | -0.07 | -4.5 | -0.49 |
| 5 | -0.6 | -0.27 | -0.6 | -0.31 | -1.3 | -0.09 | -3.8 | -0.46 |
| 6 | -0.5 | -0.24 | -0.5 | -0.27 | -1.5 | -0.11 | -3.3 | -0.42 |
| 7 | -0.4 | -0.21 | -0.6 | -0.35 | -1.6 | -0.14 | -2.9 | -0.39 |
| 8 | -0.6 | -0.37 | -0.3 | -0.23 | -1.8 | -0.16 | -2.6 | -0.38 |
| 9 | -0.5 | -0.36 | -0.4 | -0.28 | -1.9 | -0.18 | -2.4 | -0.36 |
| 10 | -0.5 | -0.36 | -0.4 | -0.31 | -1.7 | -0.18 | -2.1 | -0.34 |

### 4.2.8. ao33_n

The ao33_n gate has also been taken from standard cell library. It have 6 inputs. The pull-down have two stacks in parallel, both with three Nmos transistors. The simulations are performed for all input and for all transitions of output for a total of cases equal to 4080 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For a lot of case the relative error is less to $3 \%$, but for some cases, in particular for a small load capacitance, the relative error is less to $5 \%$.

Table 4.8 shows a part of the results for different cases of cell size factor, input slew time and load capacitance for input A (the input A in this case is connected to gate of Nmos with source to ground and the Pmos with a source to $V_{D D}$ ).

Table 4.5.: Absolute and Relative error of AO22_n: SPICE vs VHDL comparison. Input A

| ao22_n | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 1 \operatorname{tr} 50 \mathrm{ps}$ |  | x 10 tr10ps |  | x 10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | 3.1 | 0.11 | -8.0 | -0.36 | 3.5 | 0.12 | -7.6 | -0.33 |
| 0.15 | 0.3 | 0.02 | -5.3 | -0.32 | 3.7 | 0.13 | -7.3 | -0.33 |
| 0.33 | -0.1 | 0.00 | -3.7 | -0.29 | 2.4 | 0.09 | -7.1 | -0.33 |
| 0.66 | -1.1 | -0.09 | -2.4 | -0.25 | 1.9 | 0.08 | -6.6 | -0.33 |
| 1 | -1.4 | -0.14 | -1.9 | -0.24 | 0.8 | 0.03 | -6.1 | -0.33 |
| 1.25 | -1.4 | -0.16 | -1.6 | -0.23 | 0.7 | 0.03 | -5.9 | -0.33 |
| 1.5 | -1.3 | -0.17 | -1.4 | -0.23 | 0.5 | 0.02 | -5.5 | -0.32 |
| 1.75 | -1.3 | -0.19 | -1.4 | -0.25 | -0.1 | -0.01 | -5.5 | -0.33 |
| 2 | -1.2 | -0.19 | -1.2 | -0.23 | -0.1 | 0.00 | -5.0 | -0.31 |
| 3 | -0.9 | -0.20 | -0.9 | -0.23 | 0.2 | 0.01 | -4.2 | -0.30 |
| 4 | -0.8 | -0.21 | -0.7 | -0.23 | -0.6 | -0.04 | -3.6 | -0.28 |
| 5 | -0.6 | -0.19 | -0.6 | -0.21 | -0.9 | -0.06 | -3.2 | -0.27 |
| 6 | -0.4 | -0.17 | -0.5 | -0.20 | -1.1 | -0.08 | -2.8 | -0.27 |
| 7 | -0.5 | -0.22 | -0.4 | -0.18 | -1.3 | -0.10 | -2.6 | -0.26 |
| 8 | -0.4 | -0.21 | -0.4 | -0.22 | -1.3 | -0.11 | -2.3 | -0.25 |
| 9 | -0.4 | -0.20 | -0.3 | -0.16 | -1.4 | -0.12 | -2.2 | -0.26 |
| 10 | -0.3 | -0.19 | -0.3 | -0.19 | -1.4 | -0.13 | -2.0 | -0.25 |

### 4.2.9. ao112_n

The ao112_n gate is another cell of standar library. It have 4 input. The pull-down network have three stack in parallel, one with two series Nmos and two with one. I tested for all input and for all transactions of output for a total of cases equal to 4080 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For a lot of case the relative error is less to $1 \%$.

Table 4.9 shows a part of the results for different cases of cell size factor, input slew time and load capacitance for input A (the input A in this case is connected to gate of Nmos transistor with source to ground and the Pmos transistor with a source to $\left.V_{D D}\right)$.

Table 4.6.: Absolute and Relative error of AO31_n: SPICE vs VHDL comparison. Input A

| ao31__n | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 1 \operatorname{tr} 50 \mathrm{ps}$ |  | x 10 tr10ps |  | $\mathrm{x} 10 \operatorname{tr} 50 \mathrm{ps}$ |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | 2.1 | 0.10 | 1.4 | 0.07 | 2.9 | 0.14 | 2.2 | 0.10 |
| 0.15 | 0.1 | 0.00 | 0.5 | 0.03 | 2.5 | 0.12 | 2.2 | 0.10 |
| 0.33 | -1.2 | -0.08 | -0.2 | -0.02 | 2.1 | 0.10 | 2.0 | 0.10 |
| 0.66 | -2.0 | -0.17 | -0.9 | -0.09 | 1.4 | 0.07 | 1.6 | 0.09 |
| 1 | -2.2 | -0.24 | -1.2 | -0.14 | 1.2 | 0.06 | 1.4 | 0.07 |
| 1.25 | -2.0 | -0.25 | -1.3 | -0.17 | 1.1 | 0.06 | 1.2 | 0.07 |
| 1.5 | -2.1 | -0.28 | -1.3 | -0.20 | 0.8 | 0.04 | 1.1 | 0.06 |
| 1.75 | -2.0 | -0.30 | -1.3 | -0.21 | 0.3 | 0.01 | 0.9 | 0.06 |
| 2 | -1.9 | -0.31 | -1.3 | -0.23 | 0.0 | 0.00 | 0.8 | 0.05 |
| 3 | -1.5 | -0.33 | -1.2 | -0.27 | -0.6 | -0.04 | 0.3 | 0.02 |
| 4 | -1.3 | -0.35 | -1.1 | -0.32 | -1.1 | -0.08 | 0.0 | 0.00 |
| 5 | -1.1 | -0.37 | -0.9 | -0.30 | -1.4 | -0.11 | -0.3 | -0.03 |
| 6 | -1.0 | -0.37 | -0.9 | -0.34 | -1.6 | -0.14 | -0.5 | -0.04 |
| 7 | -0.9 | -0.40 | -0.7 | -0.33 | -1.7 | -0.16 | -0.6 | -0.06 |
| 8 | -0.8 | -0.38 | -0.7 | -0.34 | -1.9 | -0.18 | -0.7 | -0.08 |
| 9 | -0.7 | -0.38 | -0.6 | -0.35 | -1.9 | -0.20 | -0.8 | -0.09 |
| 10 | -0.7 | -0.40 | -0.6 | -0.36 | -2.0 | -0.21 | -1.0 | -0.11 |

### 4.2.10. ao212_n

The ao212_n gate is taken from standard cell library. It have 5 inputs. The pulldown have three stacks in parallel, two with two series Nmos transistors and another with only one Nmos transistor. I tested for all inputs and for all transitions of output for a total of cases equal to 4080 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For number of cases the relative error is less to $1 \%$, but for some cases, in particular for a small load capacitance, the relative error is less than $2 \%$.

Table 4.10 shows a part of the results for different case of cell size factor, input slew time and load capacitance for input A (the input A in this case is connected to gate of Nmos transistor with source to ground and the Pmos transistor with a source to $\left.V_{D D}\right)$.

Table 4.7.: Absolute and Relative error of AO32_n: SPICE vs VHDL comparison. Input A

| ao32_n | x 1 tr 10 ps |  | x 1 tr 50 ps |  | $\mathrm{x} 10 \operatorname{tr} 10 \mathrm{ps}$ |  | x10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error $\%$ | abs err (ps) | error $\%$ | abs err $(\mathrm{ps})$ | error $\%$ | abs err $(\mathrm{ps})$ | error $\%$ | abs err (ps) |
| 0 | 1.9 | 0.11 | -0.1 | 0.00 | 3.1 | 0.18 | 1.0 | 0.06 |
| 0.15 | 0.5 | 0.03 | -0.3 | -0.02 | 2.8 | 0.16 | 1.1 | 0.06 |
| 0.33 | -0.9 | -0.07 | -0.3 | -0.03 | 2.5 | 0.14 | 1.0 | 0.06 |
| 0.66 | -1.9 | -0.18 | -0.5 | -0.06 | 2.1 | 0.13 | 0.8 | 0.05 |
| 1 | -2.2 | -0.26 | -0.6 | -0.07 | 2.0 | 0.12 | 0.6 | 0.04 |
| 1.25 | -2.1 | -0.28 | -0.6 | -0.08 | 1.9 | 0.12 | 0.5 | 0.03 |
| 1.5 | -2.2 | -0.32 | -0.6 | -0.09 | 1.5 | 0.10 | 0.5 | 0.03 |
| 1.75 | -2.2 | -0.35 | -0.5 | -0.09 | 0.9 | 0.06 | 0.5 | 0.03 |
| 2 | -2.0 | -0.36 | -0.6 | -0.11 | 0.6 | 0.04 | 0.4 | 0.03 |
| 3 | -1.7 | -0.39 | -0.6 | -0.15 | -0.2 | -0.01 | 0.2 | 0.01 |
| 4 | -1.5 | -0.42 | -0.7 | -0.19 | -0.7 | -0.05 | 0.0 | 0.00 |
| 5 | -1.3 | -0.45 | -0.6 | -0.21 | -1.1 | -0.10 | -0.1 | -0.01 |
| 6 | -1.1 | -0.44 | -0.6 | -0.26 | -1.3 | -0.12 | -0.1 | -0.01 |
| 7 | -1.1 | -0.48 | -0.6 | -0.27 | -1.5 | -0.15 | -0.2 | -0.02 |
| 8 | -0.9 | -0.46 | -0.6 | -0.28 | -1.7 | -0.17 | -0.2 | -0.02 |
| 10 | -0.8 | -0.45 | -0.5 | -0.30 | -1.8 | -0.20 | -0.2 | -0.03 |
| -0.7 | -0.45 | -0.5 | -0.31 | -1.8 | -0.21 | -0.3 | -0.04 |  |

### 4.2.11. ao222_n

The ao222_n gate is another cell of the standar library. It have 6 inputs. The pulldown have three stacks in parallel, all with two series Nmos. I tested for all inputs and for all transitions of output for a total of cases equal to 8160 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For several number of cases the relative error is less to $1 \%$, but for few cases, in particular for a small load capacitance, the relative error is less than $2 \%$.

Table 4.11 shows a part of the results for different case of cell size factor, input slew time and load capacitance for input A (the input A in this case is connected to gate of Nmos with source to ground and the Pmos with a source to $V_{D D}$ ).

Table 4.8.: Absolute and Relative error of AO33_n: SPICE vs VHDL comparison. Input A

| ao33_n | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 1 \operatorname{tr} 50 \mathrm{ps}$ |  | x 10 tr10ps |  | x 10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | 1.2 | 0.09 | -5.6 | -0.43 | 3.0 | 0.22 | -4.2 | -0.33 |
| 0.15 | -0.5 | -0.04 | -4.5 | -0.40 | 2.6 | 0.19 | -4.0 | -0.32 |
| 0.33 | -2.0 | -0.18 | -3.4 | -0.34 | 2.2 | 0.16 | -3.9 | -0.31 |
| 0.66 | -2.9 | -0.33 | -2.4 | -0.30 | 1.8 | 0.13 | -3.8 | -0.32 |
| 1 | -3.3 | -0.44 | -1.9 | -0.27 | 1.7 | 0.13 | -3.7 | -0.32 |
| 1.25 | -3.4 | -0.50 | -1.7 | -0.27 | 1.7 | 0.13 | -3.6 | -0.32 |
| 1.5 | -3.5 | -0.57 | -1.5 | -0.26 | 1.1 | 0.08 | -3.5 | -0.31 |
| 1.75 | -3.6 | -0.63 | -1.4 | -0.25 | 0.2 | 0.02 | -3.4 | -0.30 |
| 2 | -3.5 | -0.67 | -1.4 | -0.28 | -0.1 | 0.00 | -3.2 | -0.29 |
| 3 | -3.3 | -0.81 | -1.5 | -0.38 | -0.9 | -0.08 | -2.9 | -0.29 |
| 4 | -3.0 | -0.90 | -1.5 | -0.47 | -1.4 | -0.13 | -2.6 | -0.27 |
| 5 | -2.7 | -0.96 | -1.5 | -0.54 | -1.8 | -0.18 | -2.4 | -0.26 |
| 6 | -2.4 | -0.98 | -1.5 | -0.63 | -2.0 | -0.22 | -2.1 | -0.24 |
| 7 | -2.2 | -1.03 | -1.4 | -0.66 | -2.2 | -0.25 | -1.9 | -0.23 |
| 8 | -1.9 | -1.01 | -1.3 | -0.69 | -2.4 | -0.29 | -1.7 | -0.22 |
| 9 | -1.7 | -1.00 | -1.2 | -0.72 | -2.6 | -0.33 | -1.6 | -0.21 |
| 10 | -1.6 | -1.00 | -1.1 | -0.73 | -2.7 | -0.35 | -1.5 | -0.21 |

### 4.2.12. Discussion

The results obtained by calculating the deterministic single stage or called as nominal propagation delay of single CMOS stages by means of Equation 3.5 show a very good agreement with SPICE BSIM4 simulations. A detailed sample of the obtained results, showing that the nonlinearity of the timing functions $\tau_{D}, \tau_{I}$ and $\tau_{O}$ models the non-linear behavior of the propagation delay for small loads are shown in Figure 4.1.

Table 4.12 lists all cell single-stage standard cells tested, elements each implemented with different drive strength factors X (cell size factor), with varying input slew time and load capacitance $C_{\text {load }}$.

Table 4.9.: Absolute and Relative error of AO112_n: SPICE vs VHDL comparison. Input A

| ao112_n | x 1 tr 10 ps |  | x 1 tr 50 ps |  | x10 tr10ps | x10 tr50ps |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error $\%$ | abs err (ps) | error $\%$ | abs err $(\mathrm{ps})$ | error $\%$ | abs err $(\mathrm{ps})$ | error $\%$ | abs err (ps) |
| 0 | -0.1 | -0.01 | -0.2 | -0.02 | 0.5 | 0.03 | 0.3 | 0.04 |
| 0.15 | -0.1 | 0.00 | -0.1 | -0.01 | 0.4 | 0.03 | 0.3 | 0.03 |
| 0.33 | 0.0 | 0.00 | 0.0 | -0.01 | 0.6 | 0.04 | 0.3 | 0.04 |
| 0.66 | 0.1 | 0.01 | -0.1 | -0.01 | 0.4 | 0.03 | 0.3 | 0.03 |
| 1 | 0.1 | 0.01 | -0.1 | -0.02 | 0.4 | 0.03 | 0.3 | 0.04 |
| 1.25 | 0.1 | 0.02 | -0.1 | -0.02 | 0.5 | 0.04 | 0.2 | 0.03 |
| 1.5 | 0.0 | 0.00 | 0.0 | 0.00 | 0.5 | 0.04 | 0.3 | 0.04 |
| 1.75 | 0.0 | 0.00 | 0.0 | -0.01 | 0.4 | 0.03 | 0.2 | 0.02 |
| 2 | 0.1 | 0.03 | 0.0 | 0.01 | 0.4 | 0.03 | 0.2 | 0.02 |
| 3 | 0.2 | 0.06 | 0.0 | 0.00 | 0.4 | 0.03 | 0.2 | 0.03 |
| 4 | 0.1 | 0.04 | 0.1 | 0.02 | 0.3 | 0.03 | 0.2 | 0.03 |
| 5 | 0.2 | 0.08 | 0.1 | 0.03 | 0.3 | 0.03 | 0.2 | 0.03 |
| 6 | 0.0 | -0.01 | 0.1 | 0.09 | 0.4 | 0.04 | 0.1 | 0.02 |
| 7 | 0.0 | 0.01 | 0.0 | 0.03 | 0.3 | 0.04 | 0.2 | 0.03 |
| 8 | 0.0 | 0.02 | 0.0 | -0.02 | 0.4 | 0.05 | 0.1 | 0.02 |
| 9 | 0.1 | 0.04 | 0.2 | 0.12 | 0.3 | 0.04 | 0.1 | 0.03 |
| 10 | 0.1 | 0.05 | 0.1 | 0.11 | 0.4 | 0.05 | 0.1 | 0.03 |

### 4.3. Deterministic multi stage

The basic definition and methodology of deterministic multi stage has been discussed in previous chapter. The methodology for deterministic multi stage is implemented in VHDL and verified with SPICE simulations. The implementation is being done on several small and medium scale circuits. There are seventeen different capacitance loads, four types of size value, six input slew times has been considered in the following results. The percentage of error and absolute relative errors are taken into account. The analysis will be discussed at the end of the section about deterministic multi stage results.

Table 4.10.: Absolute and Relative error of AO212_n: SPICE vs VHDL comparison. Input A

| ao212_n | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 1 \operatorname{tr} 50 \mathrm{ps}$ |  | x 10 tr10ps |  | x 10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | 1.2 | 0.08 | -1.2 | -0.11 | 0.7 | 0.05 | -1.7 | -0.15 |
| 0.15 | 0.3 | 0.03 | -1.0 | -0.11 | 0.6 | 0.04 | -1.6 | -0.15 |
| 0.33 | -0.2 | -0.02 | -0.9 | -0.11 | 0.6 | 0.04 | -1.6 | -0.16 |
| 0.66 | -0.7 | -0.08 | -0.7 | -0.11 | 0.4 | 0.03 | -1.6 | -0.16 |
| 1 | -0.9 | -0.12 | -0.7 | -0.12 | 0.2 | 0.02 | -1.6 | -0.16 |
| 1.25 | -0.9 | -0.13 | -0.6 | -0.11 | 0.1 | 0.01 | -1.5 | -0.16 |
| 1.5 | -0.9 | -0.14 | -0.7 | -0.13 | 0.0 | 0.00 | -1.5 | -0.15 |
| 1.75 | -1.0 | -0.17 | -0.6 | -0.13 | -0.2 | -0.02 | -1.6 | -0.17 |
| 2 | -0.8 | -0.16 | -0.6 | -0.14 | -0.3 | -0.02 | -1.5 | -0.16 |
| 3 | -0.6 | -0.16 | -0.5 | -0.13 | -0.5 | -0.04 | -1.4 | -0.16 |
| 4 | -0.6 | -0.20 | -0.5 | -0.16 | -0.8 | -0.07 | -1.3 | -0.16 |
| 5 | -0.4 | -0.16 | -0.4 | -0.14 | -1.0 | -0.10 | -1.3 | -0.17 |
| 6 | -0.5 | -0.20 | -0.3 | -0.15 | -1.0 | -0.11 | -1.3 | -0.17 |
| 7 | -0.4 | -0.18 | -0.2 | -0.13 | -1.1 | -0.12 | -1.2 | -0.17 |
| 8 | -0.4 | -0.23 | -0.3 | -0.16 | -1.2 | -0.13 | -1.2 | -0.17 |
| 9 | -0.3 | -0.16 | -0.2 | -0.14 | -1.2 | -0.15 | -1.2 | -0.18 |
| 10 | -0.4 | -0.25 | -0.2 | -0.17 | -1.2 | -0.15 | -1.1 | -0.18 |

### 4.3.1. inverter chain

The first test performed for chain cell is a chain of inverters. Different chain is implemented to check if the error is constant or increasing with stage of chain. As can be seen from the following results, the error rate remains almost constant for most of the tests.

The chain of three, five, seven and nine inverters are designed in VHDL as structural module of standard library developed and in SPICE with subcircuits that you can find in section D.3.

The error is not increasing between the inverter chains. I suppose that there are no of problems for any inverter chain in any condition. The results' analysis show that there will be not an issue for any inverter chain with any possible condition in terms of applying this methodolgy.

Table 4.11.: Absolute and Relative error of AO222_n: SPICE vs VHDL comparison. Input A

| ao222_n | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 1 \operatorname{tr} 50 \mathrm{ps}$ |  | x 10 tr 10 ps |  | x 10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | -1.1 | -0.12 | 1.4 | 0.20 | 0.0 | 0.00 | 2.1 | 0.30 |
| 0.15 | -1.9 | -0.23 | 0.9 | 0.13 | -0.3 | -0.03 | 2.2 | 0.31 |
| 0.33 | -2.4 | -0.33 | 0.6 | 0.09 | -0.4 | -0.04 | 2.0 | 0.28 |
| 0.66 | -2.9 | -0.45 | 0.1 | 0.03 | -0.6 | -0.07 | 1.9 | 0.27 |
| 1 | -2.9 | -0.51 | -0.1 | -0.02 | -0.8 | -0.09 | 1.7 | 0.25 |
| 1.25 | -3.0 | -0.59 | -0.3 | -0.06 | -1.0 | -0.11 | 1.6 | 0.23 |
| 1.5 | -2.9 | -0.61 | -0.4 | -0.10 | -1.1 | -0.13 | 1.5 | 0.22 |
| 1.75 | -2.9 | -0.64 | -0.3 | -0.09 | -1.4 | -0.16 | 1.4 | 0.21 |
| 2 | -2.8 | -0.68 | -0.4 | -0.12 | -1.4 | -0.16 | 1.4 | 0.21 |
| 3 | -2.6 | -0.78 | -0.7 | -0.25 | -1.7 | -0.22 | 1.1 | 0.17 |
| 4 | -2.3 | -0.85 | -0.9 | -0.33 | -2.0 | -0.26 | 0.9 | 0.15 |
| 5 | -2.1 | -0.90 | -1.0 | -0.44 | -2.1 | -0.29 | 0.7 | 0.13 |
| 6 | -1.9 | -0.93 | -0.9 | -0.48 | -2.3 | -0.33 | 0.6 | 0.11 |
| 7 | -1.8 | -0.98 | -0.9 | -0.54 | -2.4 | -0.36 | 0.6 | 0.10 |
| 8 | -1.7 | -1.03 | -0.9 | -0.59 | -2.4 | -0.38 | 0.4 | 0.08 |
| 9 | -1.4 | -0.96 | -0.9 | -0.60 | -2.5 | -0.42 | 0.3 | 0.05 |
| 10 | -1.4 | -1.03 | -0.8 | -0.63 | -2.6 | -0.43 | 0.2 | 0.04 |

Following are the results for chains of inverters mentioned above.

### 4.3.1.1. 3 inverter

The 3 inverter chain is tested for all transitions of output for a total of cases equal to 480 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For several number of cases, the relative error is less than $1 \%$, but for some cases, in particular for a small load capacitance, the relative error is less to $2 \%$ and for high size factor the relative error is less $3 \%$.

To understand the magnitude of the propagation delays for that cell, Table 4.13 shows the value of propagation delay for High-Low and Low-High transition.

Table 4.14 shows a part of the results for different case of cell size factor, input slew time and load capacitance

Table 4.12.: Cell verification status (single-stage)

| Cell | \# of input-output <br> transition cases tested | \# of different <br> sizes tested | \# of different <br> loads tested | \# of different input <br> slew times tested |
| :---: | :--- | :---: | :---: | :---: |
| Not | 2 (complete) | 8 | 17 | 6 |
| Nand2 | 4 (complete) | 8 | 17 | 6 |
| Nand3 | 6 (complete) | 8 | 17 | 6 |
| Nand4 | 8 (complete) | 8 | 17 | 6 |
| nor2 | 4 (complete) | 8 | 17 | 6 |
| nor3 | 6 (complete) | 8 | 17 | 6 |
| nor4 | 8 (complete) | 8 | 17 | 6 |
| ao12n | 8 (complete) | 8 | 17 | 6 |
| ao112n | 12 (complete) | 8 | 17 | 6 |
| ao212n | 24 (complete) | 8 | 17 | 6 |
| ao222n | 24 (to be completed) | 8 | 17 | 6 |
| ao22n | 16 (complete) | 8 | 17 | 6 |
| ao31n | 12 (complete) | 8 | 17 | 6 |
| ao32n | 24 (complete) | 8 | 17 | 6 |
| ao33n | 36 (complete) | 17 | 6 |  |

Table 4.13.: Absolute value of propagation delay (3 NOT chain)

| 3 not | $\mathrm{X}=1, t_{\text {slew }}=10 \mathrm{ps}$ |  |
| :---: | :---: | :---: |
| $C_{\text {load }}(\mathrm{fF})$ | $t_{L H}(\mathrm{ps})$ | $t_{H L}(\mathrm{ps})$ |
| 0.15 | 8.68 | 8.57 |
| 1.00 | 15.34 | 10.23 |
| 10.00 | 82.47 | 13.14 |

### 4.3.1.2. 5 inverter

The 5 inverter chain is tested for all transitions of output for a total number of cases equal to 480 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For several number of cases, the relative error is less to $1 \%$, but for some cases, in particular for a small load capacitance, the relative error is less than $2 \%$ and for high size factor the relative error is less $3 \%$.

To understand the magnitude of the propagation delays for that cell, Table 4.15

Table 4.14.: Absolute and Relative error of 3NOT chain: SPICE vs VHDL comparison.

| 3not | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 1 \operatorname{tr} 50 \mathrm{ps}$ |  | x 10 tr10ps |  | $\mathrm{x} 10 \operatorname{tr} 50 \mathrm{ps}$ |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | -0.4 | -0.03 | 2.2 | 0.21 | -3.2 | -0.22 | 0.8 | 0.17 |
| 0.15 | 0.0 | 0.00 | 1.9 | 0.20 | -3.2 | -0.23 | 0.8 | 0.16 |
| 0.33 | 1.1 | 0.11 | 2.3 | 0.29 | -3.2 | -0.24 | 0.7 | 0.15 |
| 0.66 | 1.3 | 0.17 | 2.3 | 0.35 | -3.2 | -0.25 | 0.7 | 0.13 |
| 1 | 1.2 | 0.18 | 2.0 | 0.37 | -3.2 | -0.26 | 0.7 | 0.12 |
| 1.25 | 1.1 | 0.19 | 1.8 | 0.37 | -3.1 | -0.26 | 0.7 | 0.12 |
| 1.5 | 1.0 | 0.19 | 1.7 | 0.38 | -2.7 | -0.23 | 0.7 | 0.13 |
| 1.75 | 0.9 | 0.19 | 1.6 | 0.38 | -2.4 | -0.21 | 0.8 | 0.15 |
| 2 | 0.8 | 0.19 | 1.5 | 0.38 | -2.1 | -0.19 | 0.8 | 0.17 |
| 3 | 0.6 | 0.20 | 1.1 | 0.39 | -1.4 | -0.13 | 1.0 | 0.21 |
| 4 | 0.5 | 0.20 | 0.9 | 0.39 | -0.9 | -0.10 | 1.0 | 0.23 |
| 5 | 0.4 | 0.20 | 0.8 | 0.39 | -0.6 | -0.07 | 1.1 | 0.26 |
| 6 | 0.4 | 0.20 | 0.7 | 0.40 | -0.5 | -0.05 | 1.2 | 0.27 |
| 7 | 0.3 | 0.21 | 0.6 | 0.40 | -0.3 | -0.04 | 1.2 | 0.29 |
| 8 | 0.3 | 0.21 | 0.5 | 0.40 | -0.2 | -0.02 | 1.2 | 0.30 |
| 9 | 0.3 | 0.21 | 0.5 | 0.41 | 0.0 | -0.01 | 1.2 | 0.32 |
| 10 | 0.3 | 0.22 | 0.5 | 0.41 | 0.1 | 0.01 | 1.3 | 0.33 |

shows the value of propagation delay for High-Low and Low-High transition.
Table 4.15.: Absolute value of propagation delay (5 NOT chain)

| 5 not | $\mathrm{X}=1, t_{\text {slew }}=10 \mathrm{ps}$ |  |
| :---: | :---: | :---: |
| $C_{\text {load }}(\mathrm{fF})$ | $t_{L H}(\mathrm{ps})$ | $t_{H L}(\mathrm{ps})$ |
| 0.15 | 13.87 | 13.98 |
| 1.00 | 21.40 | 20.64 |
| 10.00 | 99.26 | 87.77 |

Table 4.16 shows a part of the results for different case of cell size factor, input slew time and load capacitance.

Table 4.16.: Absolute and Relative error of 5NOT chain: SPICE vs VHDL comparison.

| 5not | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 1 \operatorname{tr} 50 \mathrm{ps}$ |  | x 10 tr10ps |  | x 10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | -0.1 | -0.02 | 1.2 | 0.18 | -3.3 | -0.41 | -0.2 | -0.06 |
| 0.15 | 0.1 | 0.02 | 1.1 | 0.18 | -3.3 | -0.42 | -0.3 | -0.07 |
| 0.33 | 0.8 | 0.13 | 1.6 | 0.29 | -3.3 | -0.42 | -0.3 | -0.08 |
| 0.66 | 1.0 | 0.19 | 1.7 | 0.35 | -3.3 | -0.43 | -0.3 | -0.10 |
| 1 | 1.0 | 0.21 | 1.6 | 0.37 | -3.3 | -0.44 | -0.3 | -0.11 |
| 1.25 | 0.9 | 0.21 | 1.5 | 0.37 | -3.2 | -0.44 | -0.3 | -0.12 |
| 1.5 | 0.9 | 0.21 | 1.4 | 0.38 | -3.0 | -0.41 | -0.2 | -0.10 |
| 1.75 | 0.8 | 0.21 | 1.3 | 0.38 | -2.8 | -0.39 | -0.2 | -0.08 |
| 2 | 0.7 | 0.22 | 1.2 | 0.38 | -2.6 | -0.37 | -0.1 | -0.06 |
| 3 | 0.6 | 0.22 | 1.0 | 0.38 | -2.1 | -0.31 | 0.0 | -0.01 |
| 4 | 0.5 | 0.22 | 0.8 | 0.39 | -1.7 | -0.27 | 0.1 | 0.03 |
| 5 | 0.4 | 0.22 | 0.7 | 0.39 | -1.5 | -0.25 | 0.2 | 0.05 |
| 6 | 0.4 | 0.23 | 0.6 | 0.39 | -1.3 | -0.23 | 0.3 | 0.07 |
| 7 | 0.3 | 0.23 | 0.5 | 0.39 | -1.2 | -0.22 | 0.3 | 0.09 |
| 8 | 0.3 | 0.23 | 0.5 | 0.40 | -1.1 | -0.20 | 0.4 | 0.10 |
| 9 | 0.3 | 0.24 | 0.5 | 0.40 | -0.9 | -0.18 | 0.4 | 0.12 |
| 10 | 0.3 | 0.24 | 0.4 | 0.40 | -0.8 | -0.17 | 0.5 | 0.13 |

### 4.3.1.3. 7 inverter

The 7 inverter chain is tested for all tranisitions of output for a total number of cases equal to 480 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For several number of cases, the relative error is less than $1 \%$ and for high size factor the relative error is less than $3 \%$.

To understand the magnitude of the propagation delays for that cell, Table 4.17 shows the value of propagation delay for High-Low and Low-High transition.

Table 4.17.: Absolute value of propagation delay ( 7 NOT chain)

| $\mathbf{7}$ not | $\mathrm{X}=1, t_{\text {slew }}=10 \mathrm{ps}$ |  |
| :---: | :---: | :---: |
| $C_{\text {load }}(\mathrm{fF})$ | $t_{L H}(\mathrm{ps})$ | $t_{H L}(\mathrm{ps})$ |
| 0.15 | 19.16 | 19.27 |
| 1.00 | 26.70 | 25.93 |
| 10.00 | 104.56 | 93.07 |

Table 4.18 shows a part of the results for different case of cell size factor, input slew time and load capacitance.

Table 4.18.: Absolute and Relative error of 7NOT chain: SPICE vs VHDL comparison.

| 7not | x 1 tr 10 ps |  | x 1 tr50ps |  | x10 tr10ps | x10 tr50ps |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error $\%$ | abs err (ps) | error $\%$ | abs err (ps) | error $\%$ | abs err (ps) |
| 0 | 0.0 | 0.01 | 1.0 | 0.20 | -3.3 | -0.58 | -0.6 | -0.23 |
| 0.15 | 0.2 | 0.04 | 1.0 | 0.21 | -3.3 | -0.59 | -0.6 | -0.24 |
| 0.33 | 0.7 | 0.16 | 1.3 | 0.31 | -3.3 | -0.59 | -0.6 | -0.25 |
| 0.66 | 0.9 | 0.21 | 1.4 | 0.38 | -3.3 | -0.60 | -0.6 | -0.27 |
| 1 | 0.9 | 0.23 | 1.4 | 0.39 | -3.3 | -0.61 | -0.6 | -0.29 |
| 1.25 | 0.8 | 0.23 | 1.3 | 0.40 | -3.2 | -0.61 | -0.6 | -0.29 |
| 1.5 | 0.8 | 0.24 | 1.2 | 0.40 | -3.1 | -0.58 | -0.6 | -0.27 |
| 1.75 | 0.7 | 0.24 | 1.2 | 0.40 | -2.9 | -0.56 | -0.5 | -0.25 |
| 2 | 0.7 | 0.24 | 1.1 | 0.40 | -2.8 | -0.54 | -0.5 | -0.23 |
| 3 | 0.6 | 0.24 | 0.9 | 0.41 | -2.4 | -0.48 | -0.3 | -0.18 |
| 4 | 0.5 | 0.25 | 0.8 | 0.41 | -2.1 | -0.44 | -0.2 | -0.14 |
| 5 | 0.4 | 0.25 | 0.7 | 0.41 | -1.9 | -0.42 | -0.2 | -0.12 |
| 6 | 0.4 | 0.25 | 0.6 | 0.41 | -1.8 | -0.40 | -0.1 | -0.10 |
| 7 | 0.3 | 0.25 | 0.5 | 0.42 | -1.7 | -0.39 | -0.1 | -0.08 |
| 8 | 0.3 | 0.26 | 0.5 | 0.42 | -1.5 | -0.37 | 0.0 | -0.07 |
| 9 | 0.3 | 0.26 | 0.5 | 0.42 | -1.4 | -0.35 | 0.0 | -0.05 |
| 10 | 0.3 | 0.27 | 0.4 | 0.43 | -1.3 | -0.34 | 0.1 | -0.04 |

### 4.3.1.4. 9 inverter

The 9 inverter chain is tested for all transitions of output for a total of cases equal to 480 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For various number of cases, the relative error is less than $1 \%$, but for few cases, specifically for high size factor the relative error is less $3 \%$.

To understand the magnitude of the propagation delays for that cell, Table 4.19 shows the value of propagation delay for High-Low and Low-High transition.

Table 4.19.: Absolute value of propagation delay (9 NOT chain)

| $\mathbf{9}$ not | $\mathrm{X}=1, t_{\text {slew }}=10 \mathrm{ps}$ |  |
| :---: | :---: | :---: |
| $C_{\text {load }}(\mathrm{fF})$ | $t_{L H}(\mathrm{ps})$ | $t_{H L}(\mathrm{ps})$ |
| 0.15 | 24.46 | 24.57 |
| 1.00 | 31.99 | 31.23 |
| 10.00 | 109.86 | 98.36 |

Table 4.20 shows a part of the results for different case of cell size factor, input slew time and load capacitance.

Table 4.20.: Absolute and Relative error of 9NOT chain: SPICE vs VHDL comparison.

| 9not | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | x1 tr50ps |  | x 10 tr10ps |  | x 10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | -0.1 | -0.03 | -0.9 | -0.23 | 3.2 | 0.75 | 0.8 | 0.40 |
| 0.15 | -0.3 | 0.06 | -0.9 | 0.23 | 3.2 | -0.76 | 0.8 | -0.41 |
| 0.33 | -0.7 | 0.18 | -1.2 | 0.34 | 3.2 | -0.76 | 0.8 | -0.43 |
| 0.66 | -0.8 | 0.24 | -1.3 | 0.40 | 3.2 | -0.77 | 0.8 | -0.44 |
| 1 | -0.8 | 0.25 | -1.2 | 0.42 | 3.2 | -0.78 | 0.8 | -0.46 |
| 1.25 | -0.8 | 0.26 | -1.2 | 0.42 | 3.1 | -0.78 | 0.8 | -0.46 |
| 1.5 | -0.7 | 0.26 | -1.1 | 0.42 | 3.0 | -0.75 | 0.8 | -0.44 |
| 1.75 | -0.7 | 0.26 | -1.1 | 0.43 | 2.9 | -0.73 | 0.7 | -0.42 |
| 2 | -0.7 | 0.26 | -1.0 | 0.43 | 2.8 | -0.71 | 0.7 | -0.40 |
| 3 | -0.6 | 0.27 | -0.9 | 0.43 | 2.5 | -0.65 | 0.6 | -0.35 |
| 4 | -0.5 | 0.27 | -0.8 | 0.43 | 2.3 | -0.61 | 0.5 | -0.31 |
| 5 | -0.4 | 0.27 | -0.7 | 0.44 | 2.1 | -0.59 | 0.4 | -0.29 |
| 6 | -0.4 | 0.27 | -0.6 | 0.44 | 2.0 | -0.57 | 0.4 | -0.27 |
| 7 | -0.4 | 0.28 | -0.5 | 0.44 | 1.9 | -0.56 | 0.3 | -0.25 |
| 8 | -0.3 | 0.28 | -0.5 | 0.44 | 1.8 | -0.54 | 0.3 | -0.24 |
| 9 | -0.3 | 0.28 | -0.5 | 0.45 | 1.7 | -0.52 | 0.2 | -0.22 |
| 10 | -0.3 | 0.29 | -0.4 | 0.45 | 1.6 | -0.51 | 0.2 | -0.21 |

### 4.3.2. nand2 chain

The other chain cell tested is nand2 chain. Also for this cell chain, the error rate remains almost constant for most of the tests.

The chain of three, five, seven and nine nand2 are designed in VHDL as structural module of standard library developed and in SPICE with subcircuits that you can find in section D.3.

In this case, there will be no problems for any type of nand2 chain with any condition and one can apply the same methodology.

Following are the results for chains of nand2 mentioned above.

### 4.3.2.1. 3 nand 2 input $A$

The 3nand2 input A (Nmos transistor with source connected with ground) chain is tested for all transitions of output for a total of cases equal to 480 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For several number of cases, the relative error is less than $1 \%$, but for some case, in particular for a small load capacitance and for high size factor the relative error is less $3 \%$.

To understand the magnitude of the propagation delays for that cell, Table 4.21 shows the value of propagation delay for High-Low and Low-High transition (A input on the left of table).

Table 4.21.: Absolute value of propagation delay (3 nand2 chain)

| 3 nand2 | $\mathrm{X}=1, t_{\text {slew }}=10 \mathrm{ps}$ |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| input | A |  | B |  |
| $C_{\text {load }}(\mathrm{fF})$ | $t_{L H}(\mathrm{ps})$ | $t_{H L}(\mathrm{ps})$ | $t_{L H}(\mathrm{ps})$ | $t_{H L}(\mathrm{ps})$ |
| 0.15 | 12.36 | 10.13 | 11.10 | 10.24 |
| 1.00 | 20.11 | 15.60 | 18.85 | 15.70 |
| 10.00 | 98.06 | 69.02 | 96.81 | 69.06 |

Table 4.22 shows a part of the results for different case of cell size factor, input slew time and load capacitance.

Table 4.22.: Absolute and Relative error of 3NAND2 chain (input A): SPICE vs VHDL comparison.

| IN A | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | x1tr50ps |  | x 10 tr10ps |  | x10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | -0.3 | -0.03 | -0.2 | 0.02 | 3.1 | 0.31 | 1.2 | 0.16 |
| 0.15 | -1.0 | -0.12 | -0.4 | -0.02 | 3.0 | 0.31 | 1.2 | 0.16 |
| 0.33 | -1.7 | -0.22 | -0.8 | -0.09 | 3.0 | 0.31 | 1.2 | 0.17 |
| 0.66 | -2.2 | -0.33 | -1.4 | -0.20 | 3.0 | 0.32 | 1.3 | 0.18 |
| 1 | -2.2 | -0.37 | -1.5 | -0.25 | 2.5 | 0.27 | 1.0 | 0.14 |
| 1.25 | -2.1 | -0.39 | -1.5 | -0.27 | 2.3 | 0.26 | 1.0 | 0.14 |
| 1.5 | -1.9 | -0.40 | -1.4 | -0.28 | 2.2 | 0.24 | 0.9 | 0.13 |
| 1.75 | -1.8 | -0.41 | -1.3 | -0.29 | 2.0 | 0.23 | 0.8 | 0.12 |
| 2 | -1.7 | -0.41 | -1.3 | -0.30 | 1.9 | 0.21 | 0.8 | 0.11 |
| 3 | -1.3 | -0.42 | -1.0 | -0.31 | 1.5 | 0.17 | 0.6 | 0.09 |
| 4 | -1.1 | -0.42 | -0.9 | -0.32 | 1.1 | 0.13 | 0.3 | 0.05 |
| 5 | -0.9 | -0.43 | -0.8 | -0.32 | 0.7 | 0.09 | 0.1 | 0.01 |
| 6 | -0.8 | -0.43 | -0.7 | -0.32 | 0.5 | 0.06 | -0.2 | -0.02 |
| 7 | -0.7 | -0.43 | -0.6 | -0.33 | 0.2 | 0.03 | -0.4 | -0.05 |
| 8 | -0.7 | -0.43 | -0.5 | -0.33 | 0.0 | -0.01 | -0.5 | -0.08 |
| 9 | -0.6 | -0.44 | -0.5 | -0.34 | -0.2 | -0.04 | -0.7 | -0.12 |
| 10 | -0.6 | -0.45 | -0.5 | -0.34 | -0.4 | -0.07 | -0.9 | -0.15 |

### 4.3.2.2. 3 nand 2 input $B$

The 3nand2 input B (Nmos transistor with drain connected with output) chain is tested for all transitions of output for a total of cases equal to 480 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For several number of cases, the relative error is less than $1 \%$, but for some case, in particular for a small load capacitance and for high size factor the relative error is less than $5 \%$.

To understand the magnitude of the propagation delays for that cell, Table 4.21 shows the value of propagation delay for High-Low and Low-High transition (B input on the right of table).

Table 4.23 shows a part of the results for different case of cell size factor, input slew time and load capacitance.

Table 4.23.: Absolute and Relative error of 3NAND2 chain (input B): SPICE vs VHDL comparison.

| In B | x 1 tr10ps |  | x 1 tr 50 ps |  | x 10 tr10ps |  | x10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error $\%$ | abs err $(\mathrm{ps})$ | error \% | abs err (ps) |
| 0 | 1.4 | 0.13 | 1.1 | 0.15 | 5.1 | 0.50 | 2.3 | 0.29 |
| 0.15 | 0.5 | 0.05 | 0.8 | 0.13 | 5.0 | 0.51 | 2.3 | 0.30 |
| 0.33 | -0.4 | -0.05 | 0.3 | 0.06 | 5.0 | 0.51 | 2.3 | 0.30 |
| 0.66 | -1.2 | -0.16 | -0.4 | -0.05 | 4.9 | 0.51 | 2.4 | 0.32 |
| 1 | -1.3 | -0.21 | -0.7 | -0.10 | 4.3 | 0.47 | 2.1 | 0.28 |
| 1.25 | -1.3 | -0.23 | -0.7 | -0.12 | 4.2 | 0.45 | 2.0 | 0.28 |
| 1.5 | -1.2 | -0.24 | -0.7 | -0.14 | 4.0 | 0.44 | 2.0 | 0.28 |
| 1.75 | -1.1 | -0.24 | -0.7 | -0.14 | 3.8 | 0.43 | 1.9 | 0.27 |
| 2 | -1.1 | -0.25 | -0.7 | -0.15 | 3.6 | 0.41 | 1.8 | 0.26 |
| 3 | -0.9 | -0.26 | -0.6 | -0.16 | 3.1 | 0.37 | 1.6 | 0.24 |
| 4 | -0.7 | -0.26 | -0.5 | -0.17 | 2.6 | 0.33 | 1.3 | 0.20 |
| 5 | -0.6 | -0.26 | -0.4 | -0.17 | 2.1 | 0.29 | 1.0 | 0.16 |
| 6 | -0.5 | -0.26 | -0.4 | -0.17 | 1.8 | 0.25 | 0.7 | 0.13 |
| 7 | -0.5 | -0.26 | -0.4 | -0.18 | 1.5 | 0.22 | 0.5 | 0.10 |
| 8 | -0.4 | -0.27 | -0.3 | -0.18 | 1.2 | 0.19 | 0.3 | 0.07 |
| 9 | -0.4 | -0.27 | -0.3 | -0.18 | 0.9 | 0.16 | 0.1 | 0.03 |
| 10 | -0.3 | -0.25 | -0.2 | -0.17 | 0.7 | 0.12 | -0.1 | 0.00 |

### 4.3.2.3. 5 nand 2 input $A$

The 5nand2 input A (Nmos transistor with source connected with ground) chain is tested for all transitions of output for a total of cases equal to 480 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For several number of cases, the relative error is less than $1 \%$, but for some cases, in particular for a small load capacitance and for high size factor the relative error is less than $3 \%$.

To understand the magnitude of the propagation delays for that cell, Table 4.24 shows the value of propagation delay for High-Low and Low-High transition (A input on the right of table).

Table 4.24.: Absolute value of propagation delay (5 nand2 chain)

| 5 nand2 | $\mathrm{X}=1, t_{\text {slew }}=10 \mathrm{ps}$ |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| input | A |  | B |  |
| $C_{\text {load }}(\mathrm{fF})$ | $t_{L H}(\mathrm{ps})$ | $t_{H L}(\mathrm{ps})$ | $t_{L H}(\mathrm{ps})$ | $t_{H L}(\mathrm{ps})$ |
| 0.15 | 19.39 | 17.16 | 18.13 | 17.27 |
| 1.00 | 27.15 | 22.63 | 25.89 | 22.74 |
| 10.00 | 105.10 | 76.05 | 103.84 | 76.10 |

Table 4.25 shows a part of the results for different cases of cell size factor, input slew time and load capacitance.

Table 4.25.: Absolute and Relative error of 5NAND2 chain (input A): SPICE vs VHDL comparison.

| IN A | x1 tr10ps |  | x1 tr50ps |  | x10 tr10ps |  | x10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | -0.7 | -0.12 | -0.2 | -0.02 | 3.5 | 0.61 | 2.6 | 0.52 |
| 0.15 | -1.2 | -0.21 | -0.6 | -0.10 | 3.5 | 0.62 | 2.6 | 0.52 |
| 0.33 | -1.6 | -0.32 | -1.0 | -0.21 | 3.5 | 0.62 | 2.5 | 0.52 |
| 0.66 | -2.0 | -0.43 | -1.4 | -0.32 | 3.4 | 0.62 | 2.5 | 0.53 |
| 1 | -2.0 | -0.48 | -1.5 | -0.37 | 3.1 | 0.57 | 2.3 | 0.48 |
| 1.25 | -1.9 | -0.50 | -1.4 | -0.39 | 3.0 | 0.56 | 2.2 | 0.47 |
| 1.5 | -1.8 | -0.51 | -1.4 | -0.40 | 2.9 | 0.55 | 2.1 | 0.45 |
| 1.75 | -1.7 | -0.51 | -1.3 | -0.40 | 2.8 | 0.53 | 2.0 | 0.43 |
| 2 | -1.6 | -0.52 | -1.3 | -0.41 | 2.7 | 0.52 | 2.0 | 0.42 |
| 3 | -1.4 | -0.52 | -1.1 | -0.41 | 2.4 | 0.48 | 1.7 | 0.38 |
| 4 | -1.2 | -0.53 | -0.9 | -0.42 | 2.1 | 0.43 | 1.5 | 0.34 |
| 5 | -1.0 | -0.53 | -0.8 | -0.42 | 1.9 | 0.39 | 1.3 | 0.30 |
| 6 | -0.9 | -0.53 | -0.7 | -0.42 | 1.6 | 0.35 | 1.1 | 0.26 |
| 7 | -0.8 | -0.53 | -0.7 | -0.42 | 1.5 | 0.32 | 0.9 | 0.23 |
| 8 | -0.7 | -0.54 | -0.6 | -0.43 | 1.3 | 0.29 | 0.8 | 0.20 |
| 9 | -0.7 | -0.54 | -0.6 | -0.43 | 1.1 | 0.26 | 0.6 | 0.17 |
| 10 | -0.6 | -0.55 | -0.5 | -0.44 | 0.9 | 0.22 | 0.5 | 0.13 |

### 4.3.2.4. 5 nand 2 input $B$

The 5nand2 input B (Nmos with drain connected with output) chain is tested for all transitions of output for a total of cases equal to 480 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For various number of cases, the relative error is less than $1 \%$, but for some cases, in particular for a small load capacitance and for high size factor the relative error is less than $5 \%$.

In order to observe the magnitude of the propagation delays for that cell, Table 4.24 shows the value of propagation delay for High-Low and Low-High transition (B input on the right of table).

Table 4.26 shows a part of the results with several different cases of cell size factor, input slew time and load capacitance.

### 4.3.2.5. 7 nand 2 input $A$

The 7nand2 input A (Nmos transistor with source connected with ground) chain is tested for all transitions of output for a total of cases equal to 480 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For several number of cases, the relative error is less than $1 \%$, but for some case, in particular for a small load capacitance and for high size factor the relative error is less $4 \%$.

To understand the magnitude of the propagation delays for that cell, Table 4.27 shows the value of propagation delay for High-Low and Low-High transition (A input on the right of table).

Table 4.26.: Absolute and Relative error of 5NAND2 chain (input B): SPICE vs VHDL comparison.

| IN B | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 1 \operatorname{tr} 50 \mathrm{ps}$ |  | $\mathrm{x} 10 \operatorname{tr} 10 \mathrm{ps}$ |  | x10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | 0.3 | 0.05 | 0.7 | 0.14 | 4.7 | 0.82 | 3.4 | 0.67 |
| 0.15 | -0.2 | -0.04 | 0.2 | 0.05 | 4.7 | 0.82 | 3.3 | 0.68 |
| 0.33 | -0.8 | -0.14 | -0.3 | -0.05 | 4.7 | 0.82 | 3.3 | 0.68 |
| 0.66 | -1.2 | -0.26 | -0.7 | -0.16 | 4.6 | 0.83 | 3.3 | 0.68 |
| 1 | -1.3 | -0.31 | -0.9 | -0.21 | 4.3 | 0.78 | 3.0 | 0.63 |
| 1.25 | -1.3 | -0.32 | -0.9 | -0.23 | 4.2 | 0.76 | 2.9 | 0.62 |
| 1.5 | -1.2 | -0.33 | -0.9 | -0.24 | 4.1 | 0.75 | 2.9 | 0.61 |
| 1.75 | -1.2 | -0.34 | -0.8 | -0.25 | 3.9 | 0.73 | 2.7 | 0.59 |
| 2 | -1.1 | -0.34 | -0.8 | -0.25 | 3.8 | 0.72 | 2.7 | 0.58 |
| 3 | -0.9 | -0.35 | -0.7 | -0.26 | 3.5 | 0.68 | 2.4 | 0.54 |
| 4 | -0.8 | -0.35 | -0.6 | -0.26 | 3.1 | 0.64 | 2.1 | 0.49 |
| 5 | -0.7 | -0.36 | -0.5 | -0.26 | 2.8 | 0.59 | 1.9 | 0.45 |
| 6 | -0.6 | -0.36 | -0.5 | -0.26 | 2.6 | 0.56 | 1.7 | 0.42 |
| 7 | -0.6 | -0.36 | -0.4 | -0.27 | 2.4 | 0.53 | 1.5 | 0.38 |
| 8 | -0.5 | -0.36 | -0.4 | -0.27 | 2.2 | 0.50 | 1.4 | 0.35 |
| 9 | -0.5 | -0.36 | -0.4 | -0.27 | 1.9 | 0.46 | 1.2 | 0.32 |
| 10 | -0.4 | -0.35 | -0.3 | -0.25 | 1.7 | 0.43 | 1.0 | 0.29 |

Table 4.28 shows a part of the results for different cases of cell size factor, input slew time and load capacitance.

### 4.3.2.6. 7 nand 2 input $B$

The 7nand2 input B (Nmos with drain connected with output) chain is tested for all transitions of output for a total of cases equal to 480 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For several number of cases, the relative error is less than $1 \%$, but for some case, in particular for a small load capacitance and for high size factor the relative error is less than $5 \%$.

To understand the magnitude of the propagation delays for that cell, Table 4.27

Table 4.27.: Absolute value of propagation delay (7 nand2 chain)

| 7 nand2 | $\mathrm{X}=1, t_{\text {slew }}=10 \mathrm{ps}$ |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| input | A |  | B |  |
| $C_{\text {load }}(\mathrm{fF})$ | $t_{L H}(\mathrm{ps})$ | $t_{H L}(\mathrm{ps})$ | $t_{L H}(\mathrm{ps})$ | $t_{H L}(\mathrm{ps})$ |
| 0.15 | 26.43 | 24.20 | 25.16 | 24.30 |
| 1.00 | 34.18 | 29.67 | 32.92 | 29.77 |
| 10.00 | 112.14 | 83.09 | 110.87 | 83.13 |

shows the value of propagation delay for High-Low and Low-High transition (B input on the right of table).

Table 4.29 shows a part of the results for different cases of cell size factor, input slew time and load capacitance.

### 4.3.2.7. 9 nand 2 input $A$

The 9nand2 input A (Nmos with source connected with ground) chain is tested for all transitions of output for a total of cases equal to 480 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For several number of cases, the relative error is less than $1 \%$, but for some case, in particular for a small load capacitance and for high size factor the relative error is less $4 \%$.

To understand the magnitude of the propagation delays for that cell, Table 4.30 shows the value of propagation delay for High-Low and Low-High transition (A input on the right of table).

Table 4.31 shows a part of the results for different case of cell size factor, input slew time and load capacitance.

Table 4.28.: Absolute and Relative error of 7NAND2 chain (input A): SPICE vs VHDL comparison.

| IN A | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 1 \operatorname{tr} 50 \mathrm{ps}$ |  | $\mathrm{x} 10 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 10 \operatorname{tr} 50 \mathrm{ps}$ |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | -0.9 | -0.22 | -0.5 | -0.11 | 3.7 | 0.92 | 3.0 | 0.82 |
| 0.15 | -1.2 | -0.31 | -0.8 | -0.20 | 3.7 | 0.92 | 3.0 | 0.83 |
| 0.33 | -1.6 | -0.41 | -1.1 | -0.30 | 3.6 | 0.92 | 3.0 | 0.83 |
| 0.66 | -1.8 | -0.53 | -1.4 | -0.42 | 3.6 | 0.93 | 3.0 | 0.84 |
| 1 | -1.8 | -0.58 | -1.4 | -0.46 | 3.4 | 0.88 | 2.8 | 0.78 |
| 1.25 | -1.8 | -0.59 | -1.4 | -0.48 | 3.3 | 0.86 | 2.7 | 0.77 |
| 1.5 | -1.7 | -0.60 | -1.4 | -0.49 | 3.3 | 0.85 | 2.7 | 0.76 |
| 1.75 | -1.7 | -0.61 | -1.3 | -0.50 | 3.2 | 0.83 | 2.6 | 0.74 |
| 2 | -1.6 | -0.61 | -1.3 | -0.50 | 3.1 | 0.82 | 2.5 | 0.73 |
| 3 | -1.4 | -0.62 | -1.1 | -0.51 | 2.9 | 0.78 | 2.3 | 0.69 |
| 4 | -1.2 | -0.62 | -1.0 | -0.51 | 2.6 | 0.74 | 2.1 | 0.64 |
| 5 | -1.0 | -0.63 | -0.9 | -0.51 | 2.4 | 0.69 | 1.9 | 0.60 |
| 6 | -0.9 | -0.63 | -0.8 | -0.52 | 2.3 | 0.66 | 1.8 | 0.57 |
| 7 | -0.9 | -0.63 | -0.7 | -0.52 | 2.1 | 0.63 | 1.6 | 0.53 |
| 8 | -0.8 | -0.63 | -0.7 | -0.52 | 2.0 | 0.60 | 1.5 | 0.50 |
| 9 | -0.7 | -0.64 | -0.6 | -0.53 | 1.8 | 0.56 | 1.4 | 0.47 |
| 10 | -0.7 | -0.65 | -0.6 | -0.54 | 1.7 | 0.53 | 1.3 | 0.44 |

### 4.3.2.8. 9 nand2 input $B$

The 9nand2 input B (Nmos with drain connected with output) chain is tested for all transitions of output for a total number of cases equal to 480 (combinations of cell size factor, input slew time, load capacitance and logic value of other input). For several number of cases, the relative error is less than $1 \%$, but for some case, in particular for a small load capacitance and for high size factor the relative error is less 4\%.

To understand the magnitude of the propagation delays for that cell, Table 4.30 shows the value of propagation delay for High-Low and Low-High transition (B input on the right of table).

Table 4.32 shows a part of the results for dissimilar cases of cell size factor, input slew time and load capacitance.

Table 4.29.: Absolute and Relative error of 7NAND2 chain (input B): SPICE vs VHDL comparison.

| IN B | $\mathrm{x} 1 \operatorname{tr} 10 \mathrm{ps}$ |  | $\mathrm{x} 1 \operatorname{tr} 50 \mathrm{ps}$ |  | $\mathrm{x} 10 \operatorname{tr} 10 \mathrm{ps}$ |  | x10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | -0.2 | -0.05 | 0.1 | 0.05 | 4.5 | 1.12 | 3.6 | 0.98 |
| 0.15 | -0.6 | -0.14 | -0.2 | -0.04 | 4.5 | 1.13 | 3.6 | 0.98 |
| 0.33 | -0.9 | -0.24 | -0.5 | -0.15 | 4.5 | 1.13 | 3.5 | 0.98 |
| 0.66 | -1.3 | -0.35 | -0.9 | -0.26 | 4.5 | 1.13 | 3.5 | 0.99 |
| 1 | -1.3 | -0.40 | -1.0 | -0.31 | 4.2 | 1.08 | 3.3 | 0.94 |
| 1.25 | -1.3 | -0.42 | -1.0 | -0.33 | 4.1 | 1.07 | 3.3 | 0.93 |
| 1.5 | -1.3 | -0.43 | -0.9 | -0.34 | 4.1 | 1.06 | 3.2 | 0.91 |
| 1.75 | -1.2 | -0.44 | -0.9 | -0.34 | 4.0 | 1.04 | 3.1 | 0.90 |
| 2 | -1.2 | -0.44 | -0.9 | -0.35 | 3.9 | 1.03 | 3.0 | 0.88 |
| 3 | -1.0 | -0.45 | -0.8 | -0.35 | 3.6 | 0.99 | 2.8 | 0.84 |
| 4 | -0.9 | -0.45 | -0.7 | -0.36 | 3.4 | 0.94 | 2.6 | 0.80 |
| 5 | -0.8 | -0.45 | -0.6 | -0.36 | 3.2 | 0.90 | 2.4 | 0.76 |
| 6 | -0.7 | -0.45 | -0.6 | -0.36 | 3.0 | 0.86 | 2.3 | 0.72 |
| 7 | -0.6 | -0.46 | -0.5 | -0.36 | 2.8 | 0.83 | 2.1 | 0.69 |
| 8 | -0.6 | -0.46 | -0.5 | -0.36 | 2.6 | 0.80 | 2.0 | 0.66 |
| 9 | -0.5 | -0.46 | -0.4 | -0.36 | 2.5 | 0.77 | 1.8 | 0.63 |
| 10 | -0.5 | -0.45 | -0.4 | -0.35 | 2.3 | 0.73 | 1.7 | 0.59 |

### 4.3.3. Full Adder

The Full Adder 1 bit gate is another cell of standar library. For this cell I test several cases of FA chain because it seems that the error would grow with the number of cascaded stages, but one can see below that error is constant with the increasing number of stages.

Table 4.33 summarizes the results for all Full Adder chain tested.

Table 4.30.: Absolute value of propagation delay (9 nand2 chain)

| 9 nand2 | $\mathrm{X}=1, t_{\text {slew }}=10 \mathrm{ps}$ |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| input | A |  | B |  |
| $C_{\text {load }}(\mathrm{fF})$ | $t_{L H}(\mathrm{ps})$ | $t_{H L}(\mathrm{ps})$ | $t_{L H}(\mathrm{ps})$ | $t_{H L}(\mathrm{ps})$ |
| 0.15 | 33.46 | 31.23 | 32.19 | 31.33 |
| 1.00 | 41.22 | 36.70 | 39.95 | 36.80 |
| 10.00 | 119.17 | 90.12 | 117.90 | 90.16 |

### 4.3.4. Discussion

In the proposed approach, a multicell path propagation delay is always obtained by the (event-driven) timing simulation of the connection of cells constituting the path. The propagation delay model reproduces the timing behavior of each logic cell in the path and does not rely on any global analytical calculation of path propagation delay. Tables list the results obtained for a reference set of 24 multistage cells and circuits, including two-stage standard logic elements, standard cell chains, and ripple carry adders (with inverted carry output).

### 4.4. Summary

We have discussed the results of deterministic propagation delay estimation techniques. The results obtained by calculating the nominal propagation delay of single CMOS stages are very good in comparison with SPICE BSIM4 simulations. A multicell path propagation delay is always obtained by the (event-driven) timing simulation of the connection of cells constituting the path. The propagation delay model reproduces the timing behavior of each logic cell in the path and does not rely on any global analytical calculation of path propagation delay.

Table 4.31.: Absolute and Relative error of 9NAND2 chain (input A): SPICE vs VHDL comparison.

| IN A | x1 tr10ps |  | x1tr50ps |  | $\mathrm{x} 10 \operatorname{tr} 10 \mathrm{ps}$ |  | x10 tr50ps |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) | error \% | abs err (ps) |
| 0 | -1.0 | -0.32 | -0.7 | -0.21 | 3.8 | 1.22 | 3.2 | 1.13 |
| 0.15 | -1.3 | -0.41 | -0.9 | -0.30 | 3.8 | 1.23 | 3.2 | 1.13 |
| 0.33 | -1.5 | -0.51 | -1.1 | -0.40 | 3.7 | 1.23 | 3.2 | 1.13 |
| 0.66 | -1.7 | -0.62 | -1.4 | -0.51 | 3.7 | 1.23 | 3.2 | 1.14 |
| 1 | -1.8 | -0.67 | -1.4 | -0.56 | 3.6 | 1.18 | 3.0 | 1.09 |
| 1.25 | -1.7 | -0.69 | -1.4 | -0.58 | 3.5 | 1.17 | 3.0 | 1.08 |
| 1.5 | -1.7 | -0.70 | -1.4 | -0.59 | 3.4 | 1.16 | 2.9 | 1.06 |
| 1.75 | -1.6 | -0.71 | -1.3 | -0.59 | 3.4 | 1.14 | 2.9 | 1.05 |
| 2 | -1.6 | -0.71 | -1.3 | -0.60 | 3.3 | 1.13 | 2.8 | 1.03 |
| 3 | -1.4 | -0.72 | -1.1 | -0.61 | 3.1 | 1.09 | 2.7 | 0.99 |
| 4 | -1.2 | -0.72 | -1.0 | -0.61 | 2.9 | 1.04 | 2.5 | 0.95 |
| 5 | -1.1 | -0.72 | -0.9 | -0.61 | 2.8 | 1.00 | 2.4 | 0.91 |
| 6 | -1.0 | -0.72 | -0.8 | -0.61 | 2.6 | 0.96 | 2.2 | 0.87 |
| 7 | -0.9 | -0.73 | -0.8 | -0.61 | 2.5 | 0.93 | 2.1 | 0.84 |
| 8 | -0.8 | -0.73 | -0.7 | -0.62 | 2.4 | 0.90 | 2.0 | 0.81 |
| 9 | -0.8 | -0.74 | -0.7 | -0.62 | 2.2 | 0.87 | 1.9 | 0.78 |
| 10 | -0.7 | -0.74 | -0.6 | -0.63 | 2.1 | 0.83 | 1.8 | 0.74 |

Table 4.32.: Absolute and Relative error of 9NAND2 chain (input B): SPICE vs VHDL comparison.

| IN B | x 1 tr 10 ps |  | x 1 tr 50 ps |  | x 10 tr 10 ps | x10 tr50ps |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Cload (fF) | error \% | abs err (ps) | error $\%$ | abs err (ps) | error $\%$ | abs err (ps) | error \% | abs err (ps) |
| 0 | -0.5 | -0.14 | -0.2 | -0.05 | 4.4 | 1.43 | 3.7 | 1.29 |
| 0.15 | -0.7 | -0.23 | -0.4 | -0.14 | 4.4 | 1.43 | 3.7 | 1.29 |
| 0.33 | -1.0 | -0.34 | -0.7 | -0.24 | 4.4 | 1.43 | 3.7 | 1.29 |
| 0.66 | -1.3 | -0.45 | -1.0 | -0.36 | 4.4 | 1.44 | 3.7 | 1.30 |
| 1 | -1.3 | -0.50 | -1.0 | -0.40 | 4.2 | 1.39 | 3.5 | 1.24 |
| 1.25 | -1.3 | -0.52 | -1.0 | -0.42 | 4.1 | 1.38 | 3.4 | 1.23 |
| 1.5 | -1.3 | -0.53 | -1.0 | -0.43 | 4.1 | 1.36 | 3.4 | 1.22 |
| 1.75 | -1.2 | -0.53 | -1.0 | -0.44 | 4.0 | 1.34 | 3.3 | 1.20 |
| 2 | -1.2 | -0.54 | -1.0 | -0.44 | 3.9 | 1.33 | 3.3 | 1.19 |
| 3 | -1.1 | -0.54 | -0.9 | -0.45 | 3.7 | 1.29 | 3.1 | 1.15 |
| 4 | -0.9 | -0.55 | -0.8 | -0.45 | 3.5 | 1.25 | 2.9 | 1.10 |
| 5 | -0.8 | -0.55 | -0.7 | -0.45 | 3.4 | 1.20 | 2.8 | 1.06 |
| 6 | -0.8 | -0.55 | -0.6 | -0.46 | 3.2 | 1.17 | 2.6 | 1.03 |
| 7 | -0.7 | -0.55 | -0.6 | -0.46 | 3.1 | 1.14 | 2.5 | 0.99 |
| 8 | -0.6 | -0.55 | -0.5 | -0.46 | 2.9 | 1.11 | 2.4 | 0.96 |
| 9 | -0.6 | -0.55 | -0.5 | -0.46 | 2.8 | 1.07 | 2.3 | 0.93 |
| 10 | -0.5 | -0.54 | -0.5 | -0.45 | 2.6 | 1.04 | 2.1 | 0.90 |

Table 4.33.: Relative error of different Full Adder chain: SPICE vs VHDL comparison.

| cell | 10 ps |  |  |  |  |  | 50 ps |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 0.33 fF |  | 1 fF |  | 5 fF |  | 0.33 fF |  | 1 fF |  | 5 fF |  |
|  | x 1 | x 10 | x1 | x 10 | x1 | x 10 | x1 | x10 | x1 | x10 | x 1 | x10 |
| fa | 0.0 | 0.7 | -0.8 | -0.6 | -0.4 | -2.0 | -1.0 | -2.7 | -0.9 | -2.7 | -0.4 | -0.9 |
| 2_fa | 0.7 | 2.4 | -0.5 | 2.2 | -3.4 | 0.5 | 4.3 | -2.5 | 3.9 | -2.5 | 0.1 | -2.9 |
| 4_fa | 2.7 | -2.6 | 2.2 | -2.6 | -0.1 | -3.1 | 4.4 | -5.0 | 3.9 | -5.0 | 1.2 | -5.4 |
| 8_fa | 4.3 | -4.4 | 4.0 | -4.4 | 2.4 | -4.6 | 5.1 | -5.5 | 4.8 | -5.5 | 3.1 | -5.6 |
| 16_fa | 5.4 | -4.8 | 5.2 | -4.8 | 4.2 | -4.9 | 5.4 | -5.7 | 5.3 | -5.7 | 4.3 | -5.8 |
| 32_fa | 5.6 | -5.4 | 5.5 | -5.4 | 5.0 | -5.4 | 5.6 | -5.8 | 5.5 | -5.8 | 5.0 | -5.8 |
| 64_fa | 5.6 | -5.7 | 5.6 | -5.7 | 5.3 | -5.7 | 5.7 | -5.8 | 5.6 | -5.8 | 5.4 | -5.8 |

## 5. Results on statistical propagation delay prediction in variable process conditions

### 5.1. Statistical single stage

The definitions and methodology of statistical single stage can be seen in chapter 3 for details. The methodology for this part of research is implemented in VHDL and verified with complex SPICE Monte Carlo simulations. The implementation is done on various scales of circuits. The circuits are inverter-chains, NOT, NAND and Full Adders with different functionalities. Three types of capacitance loads, two sizes X, two slew times will be taken as input in the following results. The deviation and mean error in percentage and comparison between proposed techniques as the output results. The summary of results for statistical single stage will be discussed in detail in terms of analysis.

### 5.1.1. Inverter

The first test was realized on a Inverter gate. Unlike the deterministic simulations only few cases were carried out here because the simulation time at circuit level was significantly greater. In fact, 10000 simulations were carried out for each case in statistical simulations.

The following table (Table 5.1) summarizes same of the results, with the change of driver strenght, slew time and load capacitance.

The first two rows in the table represent the percentage mean and deviation error between the developed model at logical-level versus circuit-level simulations.

Table 5.1.: Statistical analysis of single-stage: inverter gate

| Inverter |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| X | 1 |  |  |  |  |  | 10 |  |  |  |  |  |
| tr (ps) | 10 |  |  | 50 |  |  | 10 |  |  | 50 |  |  |
| cload (fF) | 0.33 | 1 | 5 | 0.33 | 1 | 5 | 0.33 | 1 | 5 | 0.33 | 1 | 5 |
| mean err \% | 0.3 | 0.0 | 0.0 | 0.3 | 0.0 | -0.4 | -0.1 | 0.0 | 0.1 | 1.4 | 1.9 | 0.2 |
| deviation err \% | -2.6 | 0.0 | 1.1 | 8.9 | 0.0 | -0.4 | 1.9 | -5.6 | 3.5 | 33.2 | -34.1 | 8.8 |
| mean vhdl (ps) | 4.97 | 10.72 | 45.30 | 7.06 | 14.67 | 49.22 | 1.76 | 2.53 | 6.03 | 2.29 | 3.56 | 8.82 |
| mean spice (ps) | 4.96 | 10.72 | 45.31 | 7.04 | 14.67 | 49.42 | 1.76 | 2.54 | 6.02 | 2.26 | 3.50 | 8.80 |
| deviation vhdl (ps) | 0.61 | 1.43 | 6.37 | 1.17 | 1.93 | 6.56 | 0.27 | 0.33 | 0.76 | 0.83 | 0.55 | 1.32 |
| deviation spice (ps) | 0.63 | 1.43 | 6.30 | 1.06 | 1.93 | 6.59 | 0.26 | 0.35 | 0.74 | 0.55 | 0.74 | 1.21 |

In the rest of the following rows of the Table 5.1, the results of mean VHDL, mean SPICE, deviation VHDL and deviation SPICE values whit respect to driver strenght, slew time and load capacitance are represented.

The comparison of these models represent very close values of mean and standard deviation. The worst case between these model is with mean $1.4 \%$ and standard deviation $33.3 \%$ but the absolute error value is only 0.28 ps.

### 5.1.2. Nand2

Another case was nand2 gate. Unlike the deterministic simulations only few cases were carried out here because the simulation time at circuit level was significantly greater. In fact, 10000 simulations were carried out for each case in statistical simulations.

The following table (Table 5.2) summarizes same of the results, with the change of driver strenght, slew time and load capacitance.

The first two rows in the table represent the percentage mean and deviation error between the developed model at logical-level versus circuit-level simulations.

Table 5.2.: Statistical analysis of single-stage: nand2 gate

| nand2 |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| X | 1 |  |  |  |  |  | 10 |  |  |  |  |  |
| $\operatorname{tr}$ (ps) | 10 |  |  | 50 |  |  | 10 |  |  | 50 |  |  |
| cload (fF) | 0.33 | 1 | 5 | 0.33 | 1 | 5 | 0.33 | 1 | 5 | 0.33 | 1 | 5 |
| mean err \% | 0.4 | 0.3 | 0.1 | 0.5 | 0.2 | -0.3 | 0.9 | 1.0 | 0.4 | 1.2 | 0.9 | 0.4 |
| deviation err \% | 2.8 | 0.1 | 1.4 | 7.1 | 0.0 | -2.0 | -3.0 | -0.6 | 2.7 | 13.3 | 11.6 | 4.3 |
| mean vhdl (ps) | 6.80 | 12.58 | 47.17 | 11.29 | 17.81 | 51.41 | 4.05 | 4.66 | 7.77 | 7.58 | 8.37 | 12.40 |
| mean spice (ps) | 6.78 | 12.54 | 47.12 | 11.23 | 17.78 | 51.57 | 4.02 | 4.61 | 7.74 | 7.49 | 8.30 | 12.35 |
| deviation vhdl (ps) | 0.87 | 1.68 | 6.64 | 1.22 | 1.98 | 6.65 | 0.52 | 0.59 | 1.01 | 0.88 | 0.94 | 1.34 |
| deviation spice (ps) | 0.85 | 1.68 | 6.55 | 1.13 | 1.98 | 6.78 | 0.54 | 0.59 | 0.98 | 0.76 | 0.83 | 1.29 |

In the rest of the following rows of the Table 5.2, the results of mean VHDL, mean SPICE, deviation VHDL and deviation SPICE values whit respect to driver strenght, slew time and load capacitance are represented.

The following Figure 5.1 represents the comparison between statistical VHDL model with statistical SPICE simulations of extracted density functions. The comparison of these models represent very close values of mean and standard deviation. The
worst case between these model is with mean $1.2 \%$ and standard deviation $13.33 \%$ but the absolute error value is only 0.08 ps .


Figure 5.1.: Statistical analysis of single-stage: nand2 gate gaussian

### 5.2. Statistical multi stage

The definitions and methodology of statistical single stage can be seen in chapter 3 for details. The methodology for this part of research is implemented in VHDL and verified with complex SPICE Monte Carlo simulations. The implementation is done on various scales of circuits. The circuits are inverter-chains, NOT, NAND and Full Adders with different functionalities. Three types of capacitance loads, two sizes X , two slew times will be taken as input in the following results. The deviation and mean error in percentage and comparison between proposed techniques as the
output results. The summary of results for statistical single stage will be discussed in detail in terms of analysis.

### 5.2.1. 9 inverter

The first test was realized on a 9 Inverter gate chain. Unlike the deterministic simulations only few cases were carried out here because the simulation time at circuit level was significantly greater. In fact, 10000 simulations were carried out for each case in statistical simulations.

The following table (Table 5.3) summarizes same of the results, with the change of driver strenght, slew time and load capacitance.

The first two rows in the table represent the percentage mean and deviation error between the developed model at logical-level versus circuit-level simulations.

Table 5.3.: statistical analysis of multi-stage: 9 inverter gate

| 9 inverter |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| X | 1 |  |  |  |  |  | 10 |  |  |  |  |  |
| tr (ps) | 10 |  |  | 50 |  |  | 10 |  |  | 50 |  |  |
| cload (fF) | 0.33 | 1 | 5 | 0.33 | 1 | 5 | 0.33 | 1 | 5 | 0.33 | 1 | 5 |
| mean err \% | -0.6 | -0.6 | -0.3 | -1.3 | -1.2 | -0.6 | 3.2 | 3.3 | 2.4 | 1.7 | 1.8 | 1.1 |
| deviation err \% | -14.7 | -11.3 | -2.7 | -4.8 | -3.3 | 0.9 | -9.3 | -9.3 | -8.4 | -0.3 | -0.4 | -0.3 |
| mean vhdl (ps) | 25.97 | 31.80 | 66.46 | 27.66 | 33.49 | 68.15 | 24.13 | 24.82 | 27.99 | 25.93 | 26.62 | 29.80 |
| mean spice (ps) | 26.11 | 31.99 | 66.67 | 28.02 | 33.90 | 68.57 | 23.35 | 24.00 | 27.33 | 25.49 | 26.14 | 29.47 |
| deviation vhdl (ps) | 3.37 | 4.14 | 8.96 | 4.00 | 4.77 | 9.59 | 3.19 | 3.27 | 3.67 | 3.80 | 3.88 | 4.29 |
| deviation spice (ps) | 3.87 | 4.60 | 9.20 | 4.19 | 4.92 | 9.50 | 3.49 | 3.57 | 3.98 | 3.81 | 3.89 | 4.30 |

In the rest of the following rows of the Table 5.3, the results of mean VHDL, mean SPICE, deviation VHDL and deviation SPICE values whit respect to driver strenght, slew time and load capacitance are represented.

The comparison of these models represent very close values of mean and standard deviation. The worst case between these model is with standard deviation $-14.7 \%$
but the absolute error value is only 0.5 ps .

### 5.2.2. 9 nand2

Another test was realized on a 9 nand2 gate chain. Unlike the deterministic simulations only few cases were carried out here because the simulation time at circuit level was significantly greater. In fact, 10000 simulations were carried out for each case in statistical simulations.

The following table (Table 5.4) summarizes same of the results, with the change of driver strenght, slew time and load capacitance.

The first two rows in the table represent the percentage mean and deviation error between the developed model at logical-level versus circuit-level simulations.

Table 5.4.: statistical analysis of multi-stage: 9 nand2 gate

| 9nand2 |  |  |  |  |  |  |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| X | 1 |  |  |  |  |  | 10 |  |  |  |  |  |
| $\operatorname{tr}(\mathrm{ps})$ | 10 |  |  | 50 |  |  | 10 |  |  | 50 |  |  |
| cload (fF) | 0.33 | 1 | 5 | 0.33 | 1 | 5 | 0.33 | 1 | 5 | 0.33 | 1 | 5 |
| mean err \% | -1.4 | -1.5 | -0.8 | -0.5 | -0.7 | -0.4 | 3.8 | 3.5 | 2.6 | 3.4 | 3.1 | 2.4 |
| deviation err \% | -6.5 | -5.6 | -0.5 | -2.5 | -2.1 | 1.3 | -0.1 | 0.0 | -0.4 | 2.0 | 2.0 | 1.6 |
| mean vhdl (ps) | 34.68 | 40.59 | 75.33 | 39.44 | 45.35 | 80.08 | 33.78 | 34.34 | 37.60 | 38.18 | 38.74 | 42.00 |
| mean spice (ps) | 35.18 | 41.21 | 75.97 | 39.65 | 45.68 | 80.44 | 32.49 | 33.14 | 36.61 | 36.89 | 37.53 | 41.00 |
| deviation vhdl (ps) | 4.72 | 5.48 | 10.30 | 5.14 | 5.90 | 10.72 | 4.65 | 4.74 | 5.16 | 5.00 | 5.09 | 5.52 |
| deviation spice (ps) | 5.03 | 5.79 | 10.35 | 5.27 | 6.03 | 10.58 | 4.66 | 4.74 | 5.18 | 4.90 | 4.99 | 5.43 |

In the rest of the following rows of the Table 5.4, the results of mean VHDL, mean SPICE, deviation VHDL and deviation SPICE values whit respect to driver strenght, slew time and load capacitance are represented.

The comparison of these models represent very close values of mean and standard
deviation. The worst case between these model is with standard deviation -6.5\% but the absolute error value is only 0.33 ps .

### 5.3. Statistical Multi Stage for Macrocell Design/Complex Circuits

Another test are performed for speed and accuracy of the proposed approach on full macro-cell designs, namely 16 -bit and 32 -bit ALU, an order-2 32 -bit FIR fully pipelined FIR filter (Figure 5.2), and a 2-stage-pipelined MIPS processor design without hardware multiplier (Figure 5.3). In order for the analysis to have wider cell coverage, the adders in the ALUs were synthesized with XOR, AND and OR cells, while the adders and multipliers in the FIR filter use Full-Adder cells. In SPICE analysis, the netlist was limited to the circuit critical path. The results on accuracy are shown in Table 5.5 and the result on speed performance are shown in Table 5.6.

Table 5.5.: Statistical analysis of multi-stage for complex circuits: propagation delay comparation

| Circuit | Cell | Transistor | Crit. Path. Delay |  | Crit. Path. Delay |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | count | count | mean value <br> $(S P I C E)$ | variance <br> (SPICE) | mean value <br> (proposed model) | variance <br> (proposed model) |
| 16 bit ALU | 336 | 2112 | 283.6 ps | 38.1 ps | 294.8 ps | 41.9 ps |
| 32 bit ALU | 672 | 4224 | 534.2 ps | 72.3 ps | 559.5 ps | 80.2 ps |
| 32 bit FIR filter | 6176 | 104768 | 1633.9 ps | 222.4 ps | 1584.4 ps | 221.0 ps |
| 2-stage MIPS core | 1635 | 26876 | 537.9 ps | 82.6 ps | 561.0 ps | 81.3 ps |

The set of design cases allows giving a rough assessment of the run time behavior with respect to circuit size, although event-driven simulation makes run time and quality of results strongly dependent on circuit activity.

In order to better understand the execution time behavior with circuit size we made

Table 5.6.: Statistical analysis of multi-stage for complex circuits: execution time comparation

| Circuit | Execution Time for <br> $10^{3} \mathrm{MC}$ <br> iterations (SPICE) | Execution Time for <br> $10^{3} \mathrm{MC}$ iterations <br> (proposed model) |
| :---: | :---: | :---: |
| 16 bit ALU | 61.1 hrs | 2.9 hrs |
| 32 bit ALU | 149.4 hrs | 3.1 hrs |
| 32 bit FIR filter | 2900 hrs | 10.8 hrs |
| 2-stage MIPS core | 325.3 hrs | 4.5 hrs |

a dedicated test on a non-pipelined 16 -bit FIR filter with increasing filter order, thus maintaining basically the same circuit structure and activity with increasing complexity. The test was made on the execution of only one simulation iteration, in order to be able to concentrate on very large circuit size. In SPICE analysis, the netlist was limited to the circuit critical path.

Table 5.7.: Simulation time FIR filter (SPICE vs HDL)

| taps fir | stage critical | total mos | total | simulation time (s) |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
|  | path |  | hibrary cell | hdl |  |
| 2 | 128 | 22780 | 1554 | 241.7 | 1.9 |
| 4 | 256 | 38488 | 2612 | 995.5 | 2.9 |
| 8 | 512 | 69904 | 4728 | 4147.3 | 5.5 |
| 16 | 1024 | 132736 | 8960 | 16971.9 | 13.0 |
| 32 | 2048 | 258400 | 17424 | 67887.5 (estimated) | 28.7 |
| 64 | 4096 | 509728 | 34352 | 271550.2 (estimated) | 130.1 |
| 96 | 6144 | 761056 | 51280 | 543100.3 (estimated) | 393.2 |
| 128 | 8192 | 1012384 | 68208 | 1086200.6 (estimated) | 646.1 |

The results are shown in Table 5.7. According to our results, the computation run time of the proposed model for such high-activity circuits is approximately linear with the cell count, with an increase in the linear slope occurring at about 20000 cells. In fact, the actual speedup with respect to SPICE in large circuits is affected by the time for loading the design simulation database, the cache memory usage and
memory management issues related to circuit size, which result in large speedup variability (up to more than 1000X in isolated cases) and may be the subject for further code optimization.


Figure 5.2.: Critical path through Execute Stage of FIR filter

### 5.4. Summary

We have discussed the results of statistical propagation delay estimation techniques. The comparison of the accuracy of statistical propagation delay analysis has been performed by logic-driver-based VHDL Monte Carlo simulation with respect to SPICE BSIM4 Monte Carlo simulation. Results for single-stage cells (where there is no impact of input pin capacitance variability) and for multistage cells and circuits (where input pin capacitance variability has significant impact) are very encouraging.

The proposed propagation delay computation routines completed the analysis with more than $100 \times$ speedup over SPICE, running on the same machine, even if a speed-optimized implementation is definitely not addressed yet.

Execute Stage


Figure 5.3.: Critical path through Execute Stage of MIPS processor

## 6. Conclusions

Since the beginning of the technologies to the nano-scale regime, it was clear that the technology scaling would affect the technological parameters in terms of propagation delay. This become the reason and motivations for the circuit designers to introduce the novel methods of simulation in the estimation field of propagation delay. Synopsys, a world leader in software and IP for semiconductor design and manufacturing, also introduces Statistical Static Timing Analysis (SSTA) into existing production design flows. This fact helps us to understand how the issue of statistical variations of the technological parameters is topical in todays era. Through the introduction of a propagation delay model, which supports the variations of technological parameters, this work attempted to introduce a methodology to logic level that could be used in the simulation of digital circuits.

Extensive literature review has been done on the state of the art techniques regarding the models of propagation delay at the logic level, this research work has focused in introducing the variations in technological parameters in the propagation delay model. The first contribution of this work is to develop a propagation delay model at the digital logic level. The necessity of having a simulator that could test the performance of any digital circuit and would go beyond the test of worst case or average case has been developed. The model has been applied on on a number of small-scale circuits that are in a classical standard cell library following by more
complex circuits. The results represent the average error is about $1 \%$ in comparison of the developed model at logic level with a SPICE circuit simulator. With reference to the state of the art approaches, in order to obtain a real value of the contribution to the critical path propagation delay, it iterates the calculation of the propagation delay because the output signal to a cell influences the load carrying capacity which influences the contribution of the propagation delay cell which should repeat the calculation until the convergence of the results. In this work we have solved this problem with our proposed approach which show that the load capacity doesn't depend on the output signal of the cell but on the input. By following the proposed technique there is no need to repeat the calculations which saves considerable computation time.

The deterministic and statistical propagation models has been introduced by observing variations in the technological parameters. The analysis of the variations of technological parameters has helped us to understand the propagation delay behavior on test circuits such as small-scale and as well as big circuits (e.g. FIR filters and microprocesors). This allowed us to find a way to integrate the model variations with the technology parameters. The achieved results show relative error which is less than $10 \%$ in comparison with transistor level SPICE simulation. This also provides less simulation computation time of more than two orders of magnitude with our proposed model with comparison to SPICE simulation.

Furthermore, a general systematic methodology introduce to design Synchronous early-completion-prediction adders (ECPAs) ECPA units, directing nano-scale CMOS technologies. The novel methodology is fully compatible with standard VLSI macrocell design tools and standard adder structures which includes automatic definition of critical test patterns for post layout verification. An example design circuit has been developed and results have been reported in terms of speed and power which
are better than previous works reported in literature. The proposed method use the well-known high-speed carry-select and hybrid carry-select/carry-lookahead as reference addition schemes, and the prediction logic does not affect the adder logic design in any way. The design method is implemented through a standard VLSI custom macrocell design tool chain. The methodology includes an automatic way to generate critical test patterns for the ECPA postlayout validation. The resulting ECPA circuit complexity is competitive with conventional high-speed adders, as the hardware overhead is only $10 \%$ of the adder logic. A design case in 32 nm CMOS technology, simulated at post layout SPICE BSIM4 level which results in sustaining a 6 GHz clock frequency with correct cycle time predictions. Results on statistical speed performance advantage, power consumption reduction, and NBTI mitigation have been obtained with respect to a fixed latency implementation of the same adder architecture.

## Bibliography

[1] Sung-Mo Kang and Yusuf Leblebici. Cmos Digital Integrated Circuits, 3/E. Tata McGraw-Hill Education, 2003.
[2] Shekhar Borkar, Tanay Karnik, Siva Narendra, Jim Tschanz, Ali Keshavarzi, and Vivek De. Parameter variations and impact on circuits and microarchitecture. In Proceedings of the 40 th annual Design Automation Conference, pages 338-342. ACM, 2003.
[3] R Sokel. Transistor scaling with constant subthreshold leakage. Electron Device Letters, IEEE, 4(4):85-87, 1983.
[4] Narain Arora. Mosfet Modeling for Vlsi Simulation: Theory And Practice (International Series on Advances in Solid State Electronics). World Scientific Publishing Co., Inc., 2006.
[5] Takayasu Sakurai and A Richard Newton. Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas. Solid-State Circuits, IEEE Journal of, 25(2):584-594, 1990.
[6] José Luis Rosselló and Jaume Segura. An analytical charge-based compact delay model for submicrometer cmos inverters. Circuits and Systems I: Regular Papers, IEEE Transactions on, 51(7):1301-1311, 2004.
[7] Takayasu Sakurai and A Richard Newton. Delay analysis of series-connected mosfet circuits. Solid-State Circuits, IEEE Journal of, 26(2):122-131, 1991.
[8] Hanif Fatemi, Shahin Nazarian, and Massoud Pedram. Statistical logic cell delay analysis using a current-based model. In Proceedings of the 43rd annual Design Automation Conference, pages 253-256. ACM, 2006.
[9] Massimo Alioto, Massimo Poli, and Gaetano Palumbo. Efficient and accurate models of output transition time in cmos logic. In Electronics, Circuits and Systems, 200\%. ICECS 2007. 14th IEEE International Conference on, pages 1264-1267. IEEE, 2007.
[10] Ivan Edward Sutherland, Robert Fletcher Sproull, and David F Harris. Logical effort: designing fast CMOS circuits. Morgan Kaufmann, 1999.
[11] Rahul Rithe, Sharon Chou, Jie Gu, Alice Wang, Satyendra Datla, Gordon Gammie, Dennis Buss, and Anantha Chandrakasan. The effect of random dopant fluctuations on logic timing at low voltage. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 20(5):911-924, 2012.
[12] Binjie Cheng, Daryoosh Dideban, Negin Moezi, Campbell Millar, Gareth Roy, Xingsheng Wang, Scott Roy, and Asen Asenov. Statistical-variability compactmodeling strategies for bsim4 and psp. Design $\&$ Test of Computers, IEEE, 27(2):26-35, 2010.
[13] Savithri Sundareswaran, Jacob A Abraham, Rajendran Panda, and Alexandre Ardelea. Characterization of standard cells for intra-cell mismatch variations. Semiconductor Manufacturing, IEEE Transactions on, 22(1):40-49, 2009.
[14] Michael Merrett, Plamen Asenov, Yangang Wang, Mark Zwolinski, Dave Reid, Campbell Millar, Scott Roy, Zhenyu Liu, Steve Furber, and Asen Asenov. Modelling circuit performance variations due to statistical variability: Monte carlo
static timing analysis. In Design, Automation $\mathfrak{G}$ Test in Europe Conference $\mathfrak{6}$ Exhibition (DATE), 2011, pages 1-4. IEEE, 2011.
[15] Francesco Lannutti, Paolo Nenzi, and Mauro Olivieri. Klu sparse direct linear solver implementation into ngspice. In Mixed Design of Integrated Circuits and Systems (MIXDES), 2012 Proceedings of the 19th International Conference, pages 69-73. IEEE, 2012.
[16] Fabrizio Ramundo, Paolo Nenzi, and Mauro Olivieri. First integration of mosfet band-to-band-tunneling current in bsim4. Microelectronics Journal, 2011.
[17] Marcel JM Pelgrom, Aad CJ Duinmaijer, and Anton PG Welbers. Matching properties of mos transistors. Solid-State Circuits, IEEE Journal of, 24(5):1433-1439, 1989.
[18] John F Croix and DF Wong. Blade and razor: cell and interconnect delay analysis using current-based models. In Design Automation Conference, 2003. Proceedings, pages 386-389. IEEE, 2003.
[19] Antonio Mastrandrea, Francesco Menichelli, and Mauro Olivieri. A delay model allowing nano-cmos standard cells statistical simulation at the logic level. In Ph. D. Research in Microelectronics and Electronics (PRIME), 2011 7th Conference on, pages 217-220. IEEE, 2011.
[20] Harry JM Veendrick. Short-circuit dissipation of static cmos circuitry and its impact on the design of buffer circuits. Solid-State Circuits, IEEE Journal of, 19(4):468-473, 1984.
[21] OJ Bedrij. Carry-select adder. Electronic Computers, IRE Transactions on, (3):340-346, 1962.
[22] Bruce E Briley. Some new results on average worst case carry. Computers, IEEE Transactions on, 100(5):459-463, 1973.
[23] Yiran Chen, Hai Li, Cheng-Kok Koh, Guangyu Sun, Jing Li, Yuan Xie, and Kaushik Roy. Variable-latency adder (vl-adder) designs for low power and nbti tolerance. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 18(11):1621-1624, 2010.
[24] Yiran Chen, Hai Li, Jing Li, and Cheng-Kok Koh. Variable-latency adder (vladder): new arithmetic circuit design practice to overcome nbti. In Proceedings of the 2007 international symposium on Low power electronics and design, pages 195-200. ACM, 2007.
[25] Alessandro De Gloria and Mauro Olivieri. Completion-detecting carry select addition. IEE Proceedings-Computers and Digital Techniques, 147(2):93-100, 2000.
[26] Alessandro De Gloria and Mauro Olivieri. Statistical carry lookahead adders. Computers, IEEE Transactions on, 45(3):340-347, 1996.
[27] LEE Jeehan and Kunihiro Asada. A synchronous completion prediction adder (scpa). IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 80(3):606-609, 1997.
[28] DJ Kinniment. An evaluation of asynchronous addition. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 4(1):137-140, 1996.
[29] David Koes, Tiberiu Chelcea, Charles Onyeama, and Seth C Goldstein. Adding faster with application specific early termination. Technical report, DTIC Document, 2005.
[30] Y Kondo, N Ikumi, K Ueno, J Mori, and M Hirano. An early-completiondetecting alu for a 1 ghz 64 b datapath. In Solid-State Circuits Conference, 1997. Digest of Technical Papers. 43 rd ISSCC., 1997 IEEE International, pages 418-419. IEEE, 1997.
[31] Steven M Nowick, Kenneth Y Yun, Peter A Beerel, and Ayoob E Dooply. Speculative completion for the design of high-performance asynchronous dynamic adders. In Advanced Research in Asynchronous Circuits and Systems, 1997. Proceedings., Third International Symposium on, pages 210-223. IEEE, 1997.
[32] Mauro Olivieri. Design of synchronous and asynchronous variable-latency pipelined multipliers. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 9(2):365-376, 2001.
[33] Jan M Rabaey, Anantha P Chandrakasan, and Borivoje Nikolic. Digital integrated circuits, volume 2. Prentice hall Englewood Cliffs, 2002.
[34] George W Reitwiesner. The determination of carry propagation length for binary addition. Electronic Computers, IRE Transactions on, (1):35-38, 1960.
[35] Rakesh Vattikonda, Wenping Wang, and Yu Cao. Modeling and minimization of pmos nbti effect for robust nanometer design. In Proceedings of the 43 rd annual Design Automation Conference, pages 1047-1052. ACM, 2006.
[36] Neil HE Weste and Kamran Eshraghian. Principles of CMOS VLSI design: a systems perspective, volume 1. Addison-Wesley, 1994.
[37] R Jacob Baker. CMOS: circuit design, layout, and simulation, volume 18. Wiley-IEEE Press, 2011.
[38] L Bisdounis, O Koufopavlou, and S Nikolaidis. Modelling output waveform and propagation delay of a cmos inverter in the submicron range. IEE ProceedingsCircuits, Devices and Systems, 145(6):402-408, 1998.
[39] L Bisdounis, S Nikolaidis, and O Koufopavlou. Analytical transient response and propagation delay evaluation of the cmos inverter for short-channel devices. Solid-State Circuits, IEEE Journal of, 33(2):302-306, 1998.
[40] Labros Bisdounis, S Nikolaidis, O Koufopavlou, and CE Goutis. Switching
response modeling of the cmos inverter for sub-micron devices. In Proceedings of the conference on Design, automation and test in Europe, pages 729-737. IEEE Computer Society, 1998.
[41] Jeremy M Buan. Calibration method of an analytical propagation delay model. 2007.
[42] D Burdia, G Grigore, and C Ionascu. Delay and short-circuit power expressions characterizing a cmos inverter driving resistive interconnect. In Signals, Circuits and Systems, 2003. SCS 2003. International Symposium on, volume 2, pages 597-600. IEEE, 2003.
[43] Kai Chen, Chenming Hu, Peng Fang, and Ashawant Gupta. Experimental confirmation of an accurate cmos gate delay model for gate oxide and voltage scaling. Electron Device Letters, IEEE, 18(6):275-277, 1997.
[44] HC Chow and W-S Feng. Model for propagation delay evaluation of cmos inverter including input slope effects for timing verification. Electronics Letters, 28(12):1159-1160, 1992.
[45] J Costa Andre, JP Teixeira, IC Teixeira, J Buxo, and M Bafleur. Propagation delay modelling of mos digital networks. In Electrotechnical Conference, 1989. Proceedings.'Integrating Research, Industry and Education in Energy and Communication Engineering', MELECON'89., Mediterranean, pages 311-314. IEEE, 1989.
[46] Daniel Etiemble, V Adeline, Nguyen H Duyet, and JC Ballegeer. Microcomputer oriented algorithms for delay evaluation of mos gates. In Design Automation, 1984. 21st Conference on, pages 358-364. IEEE, 1984.
[47] Vassilios Gerousis, Nghiem Pan, and Dave Weaver. New delay model for $0.5 \mu$
cmos asic. In ASIC Conference and Exhibit, 1993. Proceedings., Sixth Annual IEEE International, pages 511-514. IEEE.
[48] Nils Hedenstierna and Kjell O Jeppson. Cmos circuit speed and buffer optimization. IEEE Trans. Computer-Aided Design, 6(2):270-281, 1987.
[49] Akio Hirata, Hidetoshi Onodera, and K Tamura. Estimation of propagation delay considering short-circuit current for static cmos gates. Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 45(11):11941198, 1998.
[50] Kjell O Jeppson. Modeling the influence of the transistor gain ratio and the input-to-output coupling capacitance on the cmos inverter delay. Solid-State Circuits, IEEE Journal of, 29(6):646-654, 1994.
[51] B Labouygues, J Schindler, P Maurine, N Azemard, D Auvergne, et al. Continuous representation of the performance of a cmos library. In Solid-State Circuits Conference, 2003. ESSCIRC'03. Proceedings of the 29th European, pages 595598. IEEE, 2003.
[52] Philippe Maurine, Nadine Azemard, and Daniel Auvergne. General representation of cmos structure transition time for timing library representation. Electronics Letters, 38(4):175-177, 2002.
[53] Philippe Maurine, Mustapha Rezzoug, and Daniel Auvergne. Output transition time modeling of cmos structures. In Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on, volume 5, pages 363-366. IEEE, 2001.
[54] Spiridon Nikolaidis and Alexander Chatzigeorgiou. Modeling the transistor chain operation in cmos gates for short channel devices. Circuits and Systems

I: Fundamental Theory and Applications, IEEE Transactions on, 46(10):11911202, 1999.
[55] S Nikolaidis, A Chatzigeorgiou, and ED Kyriakis-Bitzaros. Delay and power estimation for a cmos inverter driving rc interconnect loads. In Circuits and Systems, 1998. ISCAS'98. Proceedings of the 1998 IEEE International Symposium on, volume 6, pages 368-371. IEEE, 1998.
[56] Gaetano Palumbo and Massimo Poli. Propagation delay model of a current driven rc chain for an optimized design. Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 50(4):572-575, 2003.
[57] Venkatapathi N Rayapati and Bozena Kaminska. Interconnect propagation delay modeling and validation for the $16-\mathrm{mb}$ cmos sram chip. Components, Packaging, and Manufacturing Technology, Part B: Advanced Packaging, IEEE Transactions on, 19(3):605-614, 1996.
[58] JL Rossello and J Segura. Simple and accurate propagation delay model for submicron cmos gates based on charge analysis. Electronics Letters, 38(15):772774, 2002.
[59] José L Rosselló, Carol de Benito, and Jaume Segura. A compact gate-level energy and delay model of dynamic cmos gates. Circuits and Systems II: Express Briefs, IEEE Transactions on, 52(10):685-689, 2005.
[60] José Luis Rosselló and Jaume Segura. Power-delay modeling of dynamic cmos gates for circuit optimization. In Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on, pages 494-499. IEEE, 2001.
[61] Maitham Shams and Mohamed I Elmasry. Delay optimization of cmos logic circuits using closed-form expressions. In Computer Design, 1999.(ICCD'99) International Conference on, pages 563-568. IEEE, 1999.
[62] Kevin T Tang and Eby G Friedman. Transient analysis of a cmos inverter driving resistive interconnect. In Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on, volume 4, pages 269-272. IEEE, 2000.
[63] Srinivasa R Vemuru and AR Thorbjornsen. Delay-modeling of nand gates. In Circuits and Systems, 1990., Proceedings of the 33rd Midwest Symposium on, pages 922-925. IEEE, 1990
[64] Neil HE Weste and Kamran Eshraghian. Principles of cmos vlsi design: a systems perspective. NASA STI/Recon Technical Report A, 85:47028, 1985.
[65] Chung-Yu Wu, Jen-Sheng Hwang, Chih Chang, and Ching-Chu Chang. An efficient timing model for cmos combinational logic gates. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 4(4):636650, 1985.
[66] Y-H Yang and C-Y Wu. Analysis and modelling of initial delay time and its impact on propagation delay of cmos logic gates. In Circuits, Devices and Systems, IEE Proceedings G, volume 136, pages 245-254. IET, 1989.
[67] JA del Alamo. Integrated Microelectronic Devices: Physics and Modeling. Prentice Hall, 2007.
[68] Swarup Bhunia and Saibal Mukhopadhyay. Low-power variation-tolerant design in nanometer silicon. Springer, 2011.
[69] M Cho, K Maitra, and S Mukhopadhyay. Analysis of the impact of interfacial oxide thickness variation on metal-gate high-k circuits. In Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE, pages 285-288. IEEE, 2008.
[70] S Datta, G Dewey, M Doczy, BS Doyle, S Hareland, B Jin, J Kavalieros, R Kotlyar, M Metz, and N Zelick. High mobility si/sige strained channel mos
transistors with hfo~ 2 /tin gate stack. In INTERNATIONAL ELECTRON DEVICES MEETING, pages 653-656. IEEE; 1998, 2003.
[71] Kelin J Kuhn. Reducing variation in advanced logic technologies: Approaches to process and design for manufacturability of nanoscale cmos. In Electron Devices Meeting, 2007. IEDM 2007. IEEE International, pages 471-474. IEEE, 2007.
[72] Kaizad Mistry, C Allen, C Auth, B Beattie, D Bergstrom, M Bost, M Brazier, M Buehler, A Cappellani, R Chau, et al. A 45nm logic technology with high$\mathrm{k}+$ metal gate transistors, strained silicon, 9 cu interconnect layers, 193nm dry patterning, and $100 \%$ pb-free packaging. In Electron Devices Meeting, 2007. IEDM 2007. IEEE International, pages 247-250. IEEE, 2007.
[73] Saraju Mohanty. Low-power high-level synthesis for nanoscale CMOS circuits. Springer, 2008.
[74] Sani Nassif, Kerry Bernstein, David J Frank, Anne Gattiker, Wilfried Haensch, Brian L Ji, E Nowak, D Pearson, and NJ Rohrer. High performance cmos variability in the 65 nm regime and beyond. In Electron Devices Meeting, 2007. IEDM 2007. IEEE International, pages 569-571. IEEE, 2007.
[75] Wei Zhao, Frank Liu, Kanak Agarwal, Dhruva Acharyya, Sani R Nassif, Kevin J Nowka, and Yu Cao. Rigorous extraction of process variations for $65-\mathrm{nm}$ cmos design. Semiconductor Manufacturing, IEEE Transactions on, 22(1):196-203, 2009.

## A. VHDL code

The delay model propagation developed has been tested at logic level to many circuits. This are writed with a standard cells library. A example of this library are written below for a NAND2 cell. You must find a cell under test, a behavioral description for test a logic function and finally, you find a testbench for check a cell. In the last section you find an example to use Modelsim by command line and a script in TCL language to automatize a simulation.

## A.1. Example: NAND2 DUT at logic level

```
-- ************** MODERN abstract model
-- cell: NAND2
-- *************************************************
library ieee;
use ieee.std_logic_1164.all;
library modern2010;
use modern2010.common_type.all;
use modern2010.logical_drive_primitives.all;
use modern2010.logical_drive_delay_basic.all;
-- generic parameter list
-- tech = technology record, set of technology dependent parameters.
-- c_load = total fan-out load cap inout fF;
-- X = cell drive strength (resize factor with respect to minimal \
    size X1)
entity nand2_dut is
    generic ( X:real:=1.0;
    --Z_load: load_net_record;
    tech: technology_record);
    port (
```

```
        nodeA, nodeB: inout logical_drive_logic \
        := ('Z', 66 fs, (0.0,0.0));
        nodeZ: inout logical_drive_logic \
        := ('Z', 66 fs, (0.0,0.0)) );
end nand2_dut;
architecture abstract of nand2_dut is
-- driver size factor (naming: w_drivenumber)
constant w_ld1_nodeA: real := 2.0;
constant w_ld1_nodeB: real := 2.0;
constant w_ld2_nodeA: real := 2.0;
constant w_ld3_nodeB: real := 2.0;
-- timing virtual signals
signal t_ld1_nodeA_on: timing_record:=(0.0 ps, 0.0 ps);
signal t_ld1_nodeB_on: timing_record:=(0.0 ps, 0.0 ps);
signal t_ld2_nodeA_on: timing_record:=(0.0 ps, 0.0 ps);
signal t_ld3_nodeB_on: timing_record:=(0.0 ps, 0.0 ps);
signal t_ld1_off: time := 1 ps;
signal t_ld2_off: time := 1 ps;
signal t_ld3_off: time := 1 ps;
-- vdd, gnd
signal node1 : logical_drive_logic := ('Z', 66 fs, (0.0,0.0)) ;
signal node0 : logical_drive_logic := ('Z', 66 fs, (0.0,0.0)) ;
-- input and output auxiliary nodes
signal xnodeZ : logical_drive_logic := ('Z', 66 fs, (0.0,0.0)) ;
signal xnodeA : logical_drive_logic := ('Z', 66 fs, (0.0,0.0))
signal xnodeB : logical_drive_logic := ('Z', 66 fs, (0.0,0.0)) ;
-- regular internal nodes
signal node4 : logical_drive_logic := ('Z', 66 fs, (0.0,0.0)) ;
begin
-- signal setup ----------------------------------------
    node1 <= ('1', 0 ps, (0.0,0.0));
    node0 <= ('0', 0 ps, (0.0,0.0));
    xnodeA <= nodeA; -- this is not useless
    xnodeB <= nodeB; -- this is not useless
-- delay computation section ---------------------------
    t_ld1_off <= T_n_zeta(1, tech);
    t_ld2_off <= T_p_zeta(1, tech);
    t_ld3_off <= T_p_zeta(1, tech);
    t_ld1_nodeA_on <= nmos_delay(
        W_on => W_ld1_nodeA , -- active device width
        W_dg => W_ld1_nodeA + W_ld2_nodeA, --total miller width
        n_dg => 1.0, -- N device count related to W_dg
        p_dg => 1.0, -- P device count related to W_dg
```

```
    W_d => w_ld3_nodeB, --total switching device width
    n_d => 0.0, -- N device count related to W_d
    p_d => 1.0, -- P device count related to W_d
    --W_g => Z_load_n, -- FO capacitance
    W_g => nodeZ.pincap, -- FO capacitance
    stack_size_on => 2, --size of stacked structure
    stack_size_off => 1,
    pos_on => 2, -- position in stacked structure
    pos_off => 1,
    t_r => nodeA.slew, -- active device input slew time
    X => X, -- cell resize factor vs minimum size
    tech => tech); -- technology record
t_ld1_nodeB_on <= nmos_delay(
    W_on => w_ld1_nodeB,
    W_dg => w_ld3_nodeB + w_ld1_nodeB+ w_ld1_nodeB,
    n_dg => 2.0,
    p_dg => 1.0,
    W_d => w_ld2_nodeA,
    n_d => 0.0,
    p_d => 1.0,
    --W_g => Z_load_n,
    W_g => nodeZ.pincap,
    stack_size_on => 2,
    stack_size_off => 1,
    pos_on => 1,
    pos_off => 1,
    t_r => nodeB.slew,
    X => X,
    tech => tech);
t_ld2_nodeA_on <= pmos_delay(
    W_on => W_ld2_nodeA,
    W_dg => w_ld2_nodeA + w_ld1_nodeA,
    n_dg => 1.0,
    p_dg => 1.0,
    W_d => w_ld3_nodeB,
    n_d => 0.0,
    p_d => 1.0,
    --W_g => Z_load_p,
    W_g => nodeZ.pincap,
    stack_size_on => 1,
    stack_size_off => 2,
    pos_on => 1,
    pos_off => 2,
    t_r => nodeA.slew,
    X => X,
    tech => tech);
t_ld3_nodeB_on <= pmos_delay(
    W_on => w_ld3_nodeB,
    W_dg => w_ld3_nodeB + w_ld1_nodeB+ w_ld1_nodeB,
    n_dg => 2.0,
    p_dg => 1.0,
    W_d => w_ld2_nodeA,
```

```
    n_d => 0.0,
    p_d => 1.0,
    --W_g => Z_load_p,
    W_g => nodeZ.pincap,
    stack_size_on => 1,
    stack_size_off => 2,
    pos_on => 1,
    pos_off => 1,
    t_r => nodeB.slew,
    X => X,
    tech => tech);
-- pincap value
    nodeA.pincap <=
    c_in_2(
        X => X,
        t_r => nodeA.slew,
        stack_size_on => 2,
            stack_size_off => 1,
            pos_on => 1,
            pos_off => 1,
        type_name => NAND2_cell,
        IN1 => nodeA,
        IN2 => nodeB
            );
    nodeB.pincap <=
    c_in_2(
        X => X ,
        t_r => nodeB.slew,
        stack_size_on => 2,
            stack_size_off => 1,
            pos_on => 2,
            pos_off => 1,
        type_name => NAND2_cell,
        IN1 => nodeB,
        IN2 => nodeA
            );
-- logical drivers section ------------------------
    --Mn1 Mn2
    ld1: nmos_2drive(xnodez, xnodeA, xnodeB, node0, t_ld1_nodeA_on, \
        t_ld1_nodeB_on, t_ld1_off);
    --Mp1
    ld2: pmos_drive(xnodeZ, xnodeA, node1, t_ld2_nodeA_on, t_ld2_off);
    --Mp2
    ld3: pmos_drive(xnodeZ, xnodeB, node1, t_ld3_nodeB_on, t_ld3_off);
    nodeZ <= xnodeZ;
end abstract;
```


## A.2. Example: NAND2 behavioral at logic level

```
-- **************** MODERN abstract model
-- cell: NAND2 logic reference
-- ******************************************************
library ieee;
use ieee.std_logic_1164.all;
-- generic parameter list
-- tpd_min = intrinsic delay of minimal inverter
-- tpd_FO4 = total fan-out-4 delay of minimal inverter;
-- W_FO_value = actual cell fan-out expressed as
-- external load cap / minimal inverter input cap
-- X_factor = cell drive strength (resize factor
-- with respect to minimal size X1)
entity nand2 is
    generic (
        tpd_min: time := 10 ps; -- not used in logic model
        tpd_FO4: time := 50 ps; -- not used in logic model
        W_FO_value: integer := 0; -- not used in logic model
        X_factor: integer := 1); -- not used in logic model
        port (
        nodeA, nodeB: inout std_logic := 'Z';
        nodeZ: inout std_logic := 'Z');
end nand2;
------------------------------------------------
architecture abstract of nand2 is
-- definition of timing constants
constant tau_i: time := (tpd_min/3.0);
constant tau_o: time := (tpd_FO4 - tpd_min)/(3*4);
begin
    nodeZ <= nodeA nand nodeB after tau_i;
end abstract;
--------------------------------------------------
```


## A.3. Example: NAND2 testbench at logic level

1

```
-- cell tested: NAND2
-- ******************************************************
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_textio.ALL;
library modern2010
use modern2010.logical_drive_primitives.all;
use modern2010.logical_drive_delay_basic.all;
library std;
use std.textio.all;
entity nand2_testbench is
generic (
    X: real;
    pr: std_logic;
    random_sim: boolean;
    Z_load: load_net_record;
    tr: time;
    period: time := 2 ns);
port (
    nodeA,nodeB:inout logical_drive_logic:=('Z', 66 fs, (0.0,0.0));
    nodeZ, nodeY:inout logical_drive_logic:=('Z', 66 fs, (0.0,0.0))
    );
end nand2_testbench;
--------------------------------------------------
-------------------------------------
architecture testbench_arch of nand2_testbench is
signal i: integer := 0; -- time count
signal reference_clock: std_logic := 'O'; -- time count
-- SEGNALI CALCOLO RITARDI
signal tempo_1, tpA_hl, tpA_lh, tpB_hl, tpB_lh: time:=0 ns;
signal nodeA_int, nodeB_int: std_logic;
-- END SEGNALI CALCOLO RITARDI
--funzione scrivi intestazione
impure function scrivi_intestazione return time is
    file to_file_int: text open APPEND_MODE is "ritardi.txt";
    file to_file_int_m: text open WRITE_MODE is \
            "ritardi_montecarlo_X"&real'image(X)&"_tr"&\
            time'image(tr)&"_cl"&real'image(Z_load.Z_load_PDN)&".txt";
    variable buf_in: line;
    variable nul: time;
begin
```

```
    if random_sim then
            WRITE(buf_in,string'("RitardoA_LH"));
            WRITE(buf_in,HT);
        WRITE(buf_in,string'("RitardoB_LH"));
        WRITE(buf_in,HT);
        WRITE(buf_in,string'("RitardoA_HL"));
        WRITE(buf_in,HT);
        WRITE(buf_in,string'("RitardoB_HL"));
        WRITELINE(to_file_int_m,buf_in);
        file_close(to_file_int_m);
    else
        WRITE(buf_in,string'("X"));
        WRITE(buf_in,integer(X));
        WRITE(buf_in,string'(".0"));
        WRITE(buf_in,HT);
        WRITE(buf_in,string'("tr="));
        WRITE(buf_in,integer(tr/1 fs)/1000);
        WRITE(buf_in,string'(" ps"));
        WRITE(buf_in,LF);
        WRITE(buf_in,string'("Z_load_PUP"));
        WRITE(buf_in,HT);
        WRITE(buf_in,string'("RitardoA_LH"));
        WRITE(buf_in,HT);
        WRITE(buf_in,string'("RitardoB_LH"));
        WRITE(buf_in,HT);
        WRITE(buf_in,string'("Z_load_PDN"));
        WRITE(buf_in,HT);
        WRITE(buf_in,string'("RitardoA_HL"));
        WRITE(buf_in,HT);
        WRITE(buf_in,string'("RitardoB_HL"));
        WRITELINE(to_file_int,buf_in);
        file_close(to_file_int);
    end if;
        return nul;
end function scrivi_intestazione;
--end funzione scrivi intestazione
--funzione scrivi ritardo
impure function \
    scrivi_ritardo(tpA_lh, tpA_hl, tpB_lh, tpB_hl: in time)
    return time is
        file to_file:text open APPEND_MODE is "ritardi.txt";
        file to_file_m: text open WRITE_MODE is \
            "ritardi_montecarlo_X"&real'image(X)&\
            "_tr"&time'image(tr)&"_cl"&\
            real'image(Z_load.Z_load_PDN)&".txt";
        variable buf_in:line;
        variable tempo_ritardo:time;
begin
    if random_sim then
```

```
    WRITE(buf_in,real(tpA_lh/1.0 fs)/1000.0);
    WRITE(buf_in,HT);
    WRITE(buf_in,real(tpB_lh/1.0 fs)/1000.0);
    WRITE(buf_in,HT);
    WRITE(buf_in,real(tpA_hl/1.0 fs)/1000.0);
    WRITE(buf_in,HT);
    WRITE(buf_in,real(tpB_hl/1.0 fs)/1000.0);
    WRITELINE(to_file_m,buf_in);
        file_close(to_file_m);
    else
    WRITE(buf_in,Z_load.Z_load_PUP);
        WRITE(buf_in,HT);
    WRITE(buf_in,real(tpA_lh/1.0 fs)/1000.0);
    WRITE(buf_in,HT);
    WRITE(buf_in,real(tpB_lh/1.0 fs)/1000.0);
    WRITE(buf_in,HT);
        WRITE(buf_in,Z_load.Z_load_PDN);
        WRITE(buf_in,HT);
    WRITE(buf_in,real(tpA_hl/1.0 fs)/1000.0);
    WRITE(buf_in,HT);
    WRITE(buf_in,real(tpB_hl/1.0 fs)/1000.0);
    WRITELINE(to_file,buf_in);
        file_close(to_file);
    end if;
return tempo_ritardo;
end function scrivi_ritardo;
--end funzione scrivi ritardo
begin
-- reference clock generator
reference_clock <= not reference_clock after period;
-- input_generator
process (reference_clock)
type test_vector_type is array (0 to 12) \
    of std_logic_vector (1 downto 0);
variable test_vector: test_vector_type :=
(
"00",
"01",
"00",
" 10",
"00",
"01",
"11",
"01",
"00",
" 10",
" 11",
" 10",
```

```
"00"
);
begin
            if i < test_vector'length then
            nodeA.level <= (test_vector(i)(0)) ;
    nodeA.slew <= tr;
            nodeB.level <= (test_vector(i)(1));
        nodeB.slew <= tr;
            --CALCOLO DEL RITARDO
            nodeA_int <= (test_vector(i)(0));
            nodeB_int <= (test_vector(i)(1));
            --CALCOLO DEL RITARDO
            i <= i+1;
            end if;
end process;
--check process
check: process (reference_clock)
begin
    ASSERT nodeZ.level = nodeY.level
    REPORT "Uscite diverse"
    SEVERITY WARNING;
    end process check;
-- end check process
--CALCOLO DEL RITARDO
calcolo_ritardi: process (nodeY.level)
begin
    if (nodeA_int'last_event < nodeB_int'last_event) then
                case nodeY.level is
                        when 'O' => tpA_hl <= nodeA_int'last_event;
                    when '1' => tpA_lh <= nodeA_int'last_event;
                    when others => null;
                end case;
                else
                case nodeY.level is
                    when 'O' => tpB_hl <= nodeB_int'last_event;
                    when '1' => tpB_lh <= nodeB_int'last_event;
                    when others => null;
        end case;
    end if;
end process calcolo_ritardi;
--END CALCOLO DEL RITARDO
process
    variable ret: time;
    begin
        wait for 90 ns;
        ret:= scrivi_ritardo(tpA_lh, tpA_hl, tpB_lh, tpB_hl);
```

```
    end process;
process
    variable ret: time;
    begin
        if (pr='0') then ret:= scrivi_intestazione; end if;
        wait for 101 ns;
    end process;
end testbench_arch;
----------------------------------------------------------------------
-----------------------------------------------------------------------
----------------------- TESTBENCH PLATFORM--------------------------
-------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
library modern2010;
use modern2010.logical_drive_primitives.all;
use modern2010.logical_drive_delay_basic.all;
------------------------------------------------------
entity nand2_testbench_platform is
    generic (
        pr: std_logic := '0';
        tr: time := 1 ps;
    X: real := 1.0;
    Z_load_PUP: real := 10.0;
    Z_load_PDN: real := 10.0;
        period: time := 2 ns;
        i: integer:= 0;
        random_sim: boolean:= false);
end nand2_testbench_platform;
----------------------------------------------------
architecture testbench_platform of nand2_testbench_platform is
component nand2 is
    port (
        nodeA, nodeB: inout std_logic := 'Z';
        nodeZ: inout std_logic := 'Z' );
end component nand2;
component nand2_dut is
    generic (
        X: real := 1.0;
        --Z_load: load_net_record;
        tech: technology_record);
    port (
        nodeA, nodeB:inout logical_drive_logic:=('Z', 66 fs, (0.0,0.0));
```

```
    nodeZ:inout logical_drive_logic:=('Z', 66 fs, (0.0,0.0)) );
end component nand2_dut;
component nand2_testbench is
    generic (
        X: real;
        pr: std_logic;
        random_sim: boolean;
        Z_load: load_net_record;
        tr: time;
        period: time := 2 ns);
    port (
        nodeA, nodeB:inout logical_drive_logic:=('Z', 66 fs, (0.0,0.0));
        nodeZ, nodeY:inout logical_drive_logic:=('Z', 66 fs, (0.0,0.0))
    );
end component nand2_testbench;
signal nodeA, nodeB, nodeZ, nodeY: \
    logical_drive_logic := ('Z', 66 fs, (0.0,0.0));
signal nodeZ_stdlogic: std_logic := 'Z';
begin
nodeZ <= (nodeZ_stdlogic, 1 ps, (0.0,0.0));
nodeY.pincap <= (Z_load_PDN,Z_load_PDN) when (nodeY.level='1') else
    (Z_load_PUP,Z_load_PUP);
cell: nand2
    port map (nodeA.level, nodeB.level, nodeZ_stdlogic);
cell_dut: nand2_dut
    generic map (X => X,
        -- Z_load =>
            -- (Z_load_PUP => Z_load_PUP,
            -- Z_load_PDN => Z_load_PDN),
        tech =>
            (a_n_var =>0.0,
            a_p_var => 0.0,
            t_min_n => 0.000127 ns,
                    t_min_p => 0.0002555 ns,
                    random_sim => random_sim,
                    i=>i))
        port map (nodeA, nodeB, nodeY);
tb: nand2_testbench
    generic map (period=>period, tr=>tr,
    Z_load =>(Z_load_PUP => Z_load_PUP, Z_load_PDN => Z_load_PDN),
    pr => pr, random_sim => random_sim, X => X)
    port map (nodeA, nodeB, nodeZ, nodeY);
end testbench_platform;
```

```
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
```

```
-----------------------MONTECARLO---------------------------------------
```

-----------------------MONTECARLO---------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
library ieee;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_1164.all;
library modern2010;
library modern2010;
use modern2010.logical_drive_primitives.all;
use modern2010.logical_drive_primitives.all;
use modern2010.logical_drive_delay_basic.all;
use modern2010.logical_drive_delay_basic.all;
entity montecarlo is
entity montecarlo is
end montecarlo;
end montecarlo;
----------------------------------------------------
----------------------------------------------------
architecture super_testbench of montecarlo is
architecture super_testbench of montecarlo is
component nand2_testbench_platform is
component nand2_testbench_platform is
generic (
generic (
i: integer);
i: integer);
end component nand2_testbench_platform;
end component nand2_testbench_platform;
begin
begin
nand2_montecarlo: for i in 0 to 9999 generate
nand2_montecarlo: for i in 0 to 9999 generate
begin
begin
montecarlo_instance: nand2_testbench_platform
montecarlo_instance: nand2_testbench_platform
generic map (i=>i);
generic map (i=>i);
end generate;
end generate;
end architecture super_testbench;

```
end architecture super_testbench;
```


## A.4. Modelsim

Modelsim is a simulation tool for VHDL code (and not only).
Praticaly you can use it at command line and you can write script in TCL language.

## A.4.1. Compile a library by command line

If you have a library named modern2010 whit a series of VHDL file, the next step by command line COMPILE the library:

```
cd UNRM_modern2010_packages
vlib modern2010
vcom -work modern2010 common_type.vhd
vcom -work modern2010 logical_drive_primitives.vhd
vcom -work modern2010 matrix.vhd
vcom -work modern2010 cin_matrix.vhd
vcom -work modern2010 logical_drive_delay_basic.vhd
```


## A.4.2. TCL script file

You can create a script file in TCL language for Modelsim and you can run it without use graphic interface of Modelsim. A example of script file is write below.

```
#file script: example_modelsim_script.do
set path_ [pwd]
set PATHcella {/UNRM_modern2010_ref_cells_v5/}
set PATHlib {/UNRM_modern2010_packages/modern2010/}
set cella nand2
cd "$path_$PATHcella$cella"
#create work folder
vlib work
#map library
vmap modern2010 $path_$PATHlib
#compile
vcom -quiet -work work "$path_$PATHcella$cella/*.vhd"
#simulation
vsim -c -quiet -t fs -novopt -Grandom_sim="false" -Gtr="10 ps" \
    -GX="$1" -GZ_load_PUP="1" -GZ_load_PDN="1" \
    "work.${cella}_testbench_platform"
run 100 ns
quit -sim
quit -f
```

In this example you can map a library, compile and simulate your project. In simulation istruction (vsim) you can see how change "generic" value of your entity.

## A.4.3. Run TCL script file

You can run a script by command line with the command below.

1
vsim -quiet -c -do example_modelsim_script.do

## B. C code

In this appendix there are the programs mainly used in the automation of simulations and processing of the results. They range from programs to generate a new netlist from a basic netlist to programs which elaborate SPICE output file, up to the elaboration of all the results to create the matrices used in the VHDL code.

## B.1. Create new SPICE netlist

This program generate a new file. New file is a copy of input file with different technology parameters that you can change by command line.

```
#include <string>
#include <iostream>
#include <fstream>
#include <stdlib.h>
using namespace std;
//use:
//.genNETvar "name" L W ndepvarn ndepvarp toxpvar tr XX cload
int main(int argc,char *argv[]) {
    string MP2,MN2,s4ff,s8ff,
        nomeBASE,L,W,ndep_n, ndep_p,toxp,tr, XX,cload,
        riga;
```

```
nomeBASE= argv[1];
L=argv[2];
W=argv [3];
ndep_n=argv[4];
ndep_p=argv [5];
toxp=argv[6];
tr=argv[7];
XX=argv [8];
cload=argv[9];
if (L=="0") L="45n";
if (W=="0") W="45n";
if (ndep_n=="0") ndep_n="6.5e+018";
if (ndep_p=="0") ndep_p="2.8e+018";
if (toxp=="0") toxp="6.5e-010";
string nomeIN=nomeBASE+".net";
ifstream in(&nomeIN[0]);
string nomeOUT=nomeBASE+" _.net";
ofstream out(&nomeOUT[0]);
while(getline(in, riga)){
    if (riga.substr (0,11)==".PARAM Lmin")
        out << ".PARAM Lmin=" << L << "n\n";
    else if (riga.substr (0,11)==".PARAM Wmin")
        out << ".PARAM Wmin=" << W << "n\n";
    else if (riga.substr (0,15)==".PARAM ndepVARn")
        out << ".PARAM ndepVARn=" << ndep_n << "\n";
    else if (riga.substr (0,15)==".PARAM ndepVARp")
        out << ".PARAM ndepVARp=" << ndep_p << "\n";
    else if (riga.substr (0,14)==".PARAM toxpVAR")
        out << ".PARAM toxpVAR=" << toxp << "\n";
    else if (riga.substr (0,10)==".PARAM tr=")
        out << ".PARAM tr=" << tr << "p\n";
    else if (riga.substr (0,10)==".PARAM XX=")
        out << ".PARAM XX=" << XX << "\n";
    else if (riga.substr (0,11)==".PARAM XXX=")
        out << ".PARAM XXX=" << XX << "\n";
    else if (riga.substr (0,10)=="Cout nodeZ")
        out << "Cout nodeZ 0 " << cload << "f\n";
    else if (riga.substr (0,13)=="Cout2 nodeCo4")
                //Cout2 nodeCo4 0 5f
        out << "Cout2 nodeCo4 0 " << cload << "f\n";
    else if (riga.substr (0,12)=="Cout2 nodeCo")
                //Cout2 nodeCo4 0 5f
        out << "Cout2 nodeCo 0 " << cload << "f\n";
    else if (riga.substr (0,28)==".INCLUDE ../../subckt/subckt")
        out << ".INCLUDE ../../../subckt/subckt\n";
    else{
```

```
            out << riga << "\n";
        }
    }
    out.close();
    in.close();
return 0;
}
```


## B.2. SPICE output elaboration

This program finds in file output of Spice the strings "delaylh" and "delayhl". If these strings are present, the program finds the times value above and writes in a file a table with this values.

```
#include <string>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <limits>
using namespace std;
int main(int argc,char * argv[]) {
    ifstream inNOM1(argv[2]);
    string nomeFile(argv[1]);
    ofstream outritardi(argv[1],ios::app);
    string s;
    string delay_lh_4ff,delay_hl_4ff;
    long double f_delay_lh_4ff,f_delay_hl_4ff;
    int i=1,j=0;
    int ini=0,fin=0;
    int cont4f=0;
    char d_lh4ff[20],d_hl4ff[20];
```

```
while(getline(inNOM1, s) && (!(cont4f==2)) ){
    if (s.substr (0, 8)=="delay_lh") {
        i=0;
        while(s[i]!='.'){
            i++;
        }
        delay_lh_4ff =s.substr(i-1,16);
        cont4f++;
    }
    if (s.substr (0, 8)=="delay_hl") {
        i=0;
        while(s[i]!='.'){
            i++;
        }
        delay_hl_4ff =s.substr(i-1,16);
        cont4f++;
    }
}
inNOM1.close();
for(i=0;i<delay_lh_4ff.length();i++)
{
    d_lh4ff[i]=delay_lh_4ff[i];
}
f_delay_lh_4ff=atof(d_lh4ff);
for(i=0;i<delay_hl_4ff.length();i++)
{
    d_hl4ff[i]=delay_hl_4ff[i];
}
f_delay_hl_4ff=atof(d_hl4ff);
float XX,tr,cload;
tr=atof(argv[7]);
XX=atof(argv[8]);
cload=atof(argv[9]);
float L_var=atof(argv[10]),
    W_var=atof(argv[11]),
        Ndepn_var=atof(argv[12]),
        Ndepp_var=atof(argv [13]),
        tox_var=atof(argv[14]);
outritardi.precision(10);
outritardi << argv[6] << "\t" << f_delay_lh_4ff << "\t" \
    << f_delay_hl_4ff << "\t" << XX << "\t" << tr << \
    "\t" << cload << "\t" << L_var << "\t" << W_var << \
    "\t" << Ndepn_var << "\t" << Ndepp_var << "\t" << \
    tox_var << "\n";
outritardi.close();
```


## B.3. Table to VDHL matrix

This program is an example to create VHDL matrix by a file with tables of value.

```
/*
authors: Antonio Mastrandrea
date: 2010-11-26
    rev1 2012-02-13: change dimension matrix
    Create matrix:
        read a file with tau in column:
            X= 1 tr=1
            Cload tauO_n X1_1ps taui_n X1_1ps taurap_n X1_1ps
            0.00 0.000378556 0.000146375 0.000320935
            0.15 0.000342995 0.000124285 0.000359595
            0.33 0.000334367 0.000119916 0.000380228
            0.66 0.00033184 0.000118354 0.000392316
        create 3 file with matrix for tau_i tau_O and tau_m:
            tau_i:=(
                    (0.000378556,0.000342995,0.000334367,0.00033184,\ldots.),
                    (...
                    ...
                );
            tau_0:= (
                (0.000146375,0.000124285,0.000119916,0.000118354,...),
                (... ),
                ...
                );
            tau_m:= (
                (0.000320935,0.000359595,0.000380228,0.000392316,\ldots..),
                (... ),
                ...
                );
*/
#include <string>
#include <iostream>
#include <fstream>
#include <stdlib.h>
```

```
#include <limits>
#include <sstream>
#define NUM_CAP 37
#define NUM_CAP_P NUM_CAP+1
using namespace std;
//function ***************************
void find_value(string s,float* f);
void write_v_file(float *f);
void init_files_out();
void init_row();
void end_row();
void finalize_files();
/*
type of cell----------------------+
file open--------------------+ |
name program-----------+ / |
            l 1 1
            l lll
                            $ prog tau_n 1_1
*/
ifstream infile;
ofstream outtauo,outtaui,outtaum;
string typecell,tauo,taui,taum;
string s;
int main(int argc,char * argv[]) {
    float f1[3];
    //open 1 file
    cout << "open file: "<< argv[1]<<"\n";
        infile.open(argv[1]);
    typecell=argv[2];
    tauo="tau_o_"+typecell+".mat";
    taui="tau_i_"+typecell+".mat";
    taum="tau_m_"+typecell+".mat";
    outtauo.open(tauo.c_str(),ios::app);
    outtaui.open(taui.c_str(),ios::app);
    outtaum.open(taum.c_str(),ios::app);
    init_files_out();
    init_row();
    int conta_column=0;
    while(getline(infile, s) ){
        if ((s.substr (0,2)=="X=")||(s.substr (0,2)=="x="))
            cout << "x= '" << s <<"'\n";
```

```
    else if ((s=="")) cout << "line empty \"\"\n";
    else if ( (s.substr (0,3)=="Clo")||(s.substr (0,3)=="clo") )
        cout << "line intestation \"\"\n";
    else
    {
    ++conta_column;
    if (conta_column==NUM_CAP_P)
    {
            end_row();
            conta_column=1;
            init_row();
    }
    find_value(s,f1);
    //write_v_file(f1);
    outtauo << f1[0] << " ns" ;
    outtaui << f1[1] << " ns";
    outtaum << f1[2] << " ns";
    if(conta_column!=NUM_CAP)
    {
        outtauo <<"\t,\t";
        outtaui <<"\t,\t";
        outtaum <<"\t,\t";
        }
        //cout << s << "\n";
        }
    }
    finalize_files();
return 0;
}
void find_value(string s,float* f)
{
    //string s1;
    float a,b,c,d;
    std::istringstream iss(s);
    iss >> a ;
    iss >> *f;
    iss >> *(f+1);
    iss >> *(f+2);
    cout << a << "\t\t";
    cout << *f << "\t\t";
    cout << *(f+1) << "\t\t";
    cout << *(f+2) << "\n";
    //cout << b << "\n";
}
void write_v_file(float *f)
{
```

```
144 }
1 4 5
146
147
148
1 4 9
1 5 0
1 5 1
152
1 5 3
154
1 5 5
156
157
158
1 5 9
160
1 6 1
162
1 6 3
164
165
166
167
168
1 6 9
1 7 0
1 7 1
172
1 7 3
1 7 4
1 7 5
176
1 7 7
178
1 7 9
1 8 0
void init_files_out()
{
    outtauo << "constant "
        << tauo.substr(0,tauo.length()-4) << ": matrix_tau := (";
    outtaui << "constant "
        << taui.substr(0,taui.length()-4) << ": matrix_tau := (";
    outtaum << "constant "
        << taum.substr(0,taum.length()-4) << ": matrix_tau := (";
}
void init_row()
{
    outtauo << "\n\t(\t";
    outtaui << "\n\t(\t";
    outtaum << "\n\t(\t";
}
void end_row()
{
    outtauo << "\t),";
    outtaui << "\t),";
    outtaum << "\t),";
}
void finalize_files()
{
    outtauo << "\t)\n);\n";
    outtaui << "\t)\n);\n";
    outtaum << "\t)\n);\n";
    outtauo.close();
    outtaui.close();
    outtaum.close();
    infile.close();
}
```


## C. Script code

The delay model propagation developed needs a series of simulations at circuit level in order to extract a set of parameters as you can see in Figure 3.9. Afterwards there are a series of the scripts for the automatic generation of these parameters.

## C.1. Calculate $\tau$ parameter

```
#!/bin/bash
export DATE_INIZIO=$(date)
#####################################################
######## valori da cambiare in base alla cella########
################## ##################################
fileBASE="NOT" ingresso="unico"
rimuovi=NO #rimuove i file di testo creati nella precedente
    simulazione (i file di testo sono stati aperti
    in modalità append)
crea=SI #crea la struttura delle cartelle
cfile=SI #copia i file base nella cartella al variare di
    X e tr
simula=SI #simula i 4 file di base per ogni coppia di
    Cload e produce un file delle tau per ogni
    cartella
############################################################
tauN="tau_n_C${fileBASE}${ingresso}.txt"
tauP="tau_p_Cl${fileBASE}${ingresso}.txt"
ritardi="Ritardi_${fileBASE}${ingresso}.txt"
############################################################
####### verifica dell'esistenza dei file di base ########
############################################################
if [ -e ${fileBASE}conMN2.net ] &&
    [ -e ${fileBASE}conMP2.net ] &&
    [ -e ${fileBASE}senza_4fF.net ] &&
    [ -e ${fileBASE}senza_8fF.net ]
then
    echo "i file .net di base sono presenti"
else
    echo "mancano i file di base"
```

```
    exit
fi
#########################################################
#########################################################
if [ $rimuovi == "SI" ] then
    echo "rimuovo i file di testo della simulazione
        precedente"
    for tr in 1 10 20 30 40 50
    do
        for XX in 1 5 10 20
        do
            cartella="X${XX}_${tr}ps"
            cd $cartella
            rm *.txt
            rm *fF.net
            cd ..
        done
    done
fi
if [ $crea == "SI" ]
then
    echo "creo le cartelle"
    for tr in 1 10 20 30 40 50
    do
        for XX in 1 5 10 20
        do
            cartella="X${XX}_${tr}ps"
            mkdir $cartella
            echo creata $cartella
        done
    done
fi
if [ $cfile == "SI" ]
then
    echo "copio i file di base"
    for tr in 1 10 20 30 40 50
    do
        for XX in 1 5 10 20
        do
            cartella="X${XX}_${tr}ps"
            for i in ${fileBASE}*.net
            do
                    ./gNetVarTR_X $i $tr $XX
            done
            mv *ps.net $cartella
        done
    done
fi
if [ $simula == "SI" ]
then
    #inizia le simulazioni
    echo "inizio simulazioni"
```

```
CL[0]=0.00;
CL[1]=0.15;
CL[2]=0.33;
CL [3]=0.66;
CL [4]=1.00;
CL [5]=1.25;
CL[6]=1.5;
CL[7]=1.75;
CL[8]=2;
CL[9]=3;
CL[10]=4;
CL[11]=5;
CL[12]=6;
CL[13]=7;
CL[14]=8;
CL[15]=9;
CL[16]=10;
CL[17]=11;
#simulazioni
for tr in 10 20 30 40 50
do
    for XX in 1 5 10 20
    do
        cartella="X${XX}_${tr}ps"
        cd $cartella
        j=1
        #nel file dove raccolgo tutti i tau scrivo
            l'intestazione per ogni serie di misura
        echo "X= ${XX} tr= ${tr}" >> ../tau_n_.txt
        echo "Cload tau0_n X${XX}_${tr}ps
                    taui_n X${XX}_${tr}ps taurap_n
                    X${XX}_${tr}ps" >> ../tau_n_.txt
        echo "X= ${XX} tr= ${tr}" >>
                    "tau_n_Cl_X${XX}_tr${tr}ps.txt"
        echo "Cload tau0_n X${XX}_${tr}ps
                    taui_n X${XX}_${tr}ps
                    taurap_n X${XX}_${tr}ps">>
                    "tau_n_Cl_X${XX}_tr${tr}ps.txt"
        echo "X= ${XX} tr= ${tr}" >> ../tau_p_.txt
        echo "Cload tau0_p X${XX}_${tr}ps taui_p
                    X${XX}_${tr}ps taurap_p X${XX}_${tr}ps"
                                >> ../tau_p_.txt
        echo "X= ${XX} tr= ${tr}" >>
            "tau_p_Cl_X${XX}_tr${tr}ps.txt"
        echo "Cload tau0_p X${XX}_${tr}ps
                    taui_p X${XX}_${tr}ps taurap_p
                    X${XX}_${tr}ps" >>
                    "tau_p_Cl_X${XX}_tr${tr}ps.txt"
        echo "X= ${XX} tr= ${tr}" >>
                                    ../ritardi_.txt
        echo "Cload 4ff_lh X${XX}_${tr}ps 8ff_lh
            X${XX}_${tr}ps conMN2_lh
```

```
            X${XX}_${tr}ps 4ff_hl X${XX}_${tr}ps
            8ff_hl X${XX}_${tr}ps
            conMP2_hl X${XX}_${tr}ps"
                >> ../ritardi_.txt
            echo "X= ${XX} tr= ${tr}" >>
                "ritardi_Cl_X${XX}_tr${tr}ps.txt"
            echo "Cload 4ff_lh X${XX}_${tr}ps
                8ff_lh X${XX}_${tr}ps conMN2_lh
                X${XX}_${tr}ps 4ff_hl X${XX}_${tr}ps
                    8ff_hl X${XX}_${tr}ps conMP2_hl
            X${XX}_${tr}ps" >>
            "ritardi_Cl_X${XX}_tr${tr}ps.txt"
            for cload in 0..16
            do
            j=1
            .././gNetVarC "${fileBASE}conMN2X${XX}_tr
                    ${tr}ps.net" "${CL[$cload]}"
            .././gNetVarC "${fileBASE}conMP2X${XX}_tr
                ${tr}ps.net" "${CL[$cload]}"
            .././gNetVarC "${fileBASE}senza_4fFX${XX}
                _tr${tr}ps.net" "${CL[$cload]}"
            a=$[$cload+1]
            .././gNetVarC "${fileBASE}senza_8fFX${XX}
                _tr${tr}ps.net" "${CL[$a]}"
            for i in *fF.net
            do
                ngspice -b $i > "${j}.txt"
                j=$[$j+1]
            done
            .././calcolaTAU "tauCl_X${XX}_tr${tr}ps.txt"
                    "1.txt" "2.txt" "3.txt" "4.txt"
                    "${CL[$cload]} ${CL[$cload+1]}" "${XX}"
                    "${CL[$a]}" "${CL[$cload]}"
            rm *fF.net #temp
            rm "1.txt" "2.txt" "3.txt" "4.txt"
            done
            echo "" >> ../tau_n_.txt
            echo "" >> ../tau_p_.txt
            echo "" >> ../ritardi_.txt
            cd ..
            #echo creata $cartella
        done
        echo "" >> tau_n_.txt
        echo "" >> tau_p_.txt
        echo "" >> ritardi_.txt
    done
fi
echo "inizio ${fileBASE} ${ingresso} : $DATE_INIZIO"
    > time_${fileBASE}.txt echo "fine ${fileBASE}
    ${ingresso} : $(date)" >> time_${fileBASE}.txt
echo "inizio ${fileBASE} ${ingresso}: $DATE_INIZIO"
    echo "fine ${fileBASE} ${ingresso}: $(date)"
```


## C.2. Calculate $C_{\text {in }}$ Capacitance

```
#!/bin/bash
cella="not"
ingresso="a"
netHL="not_not_HL.net"
netHL_cl="not cload HL.net"
netLH="not_not_LH.net"
netLH_cl="not_cload_LH.net"
g++ gNetVarC.cpp -o gNetVarC
g++ gNetVarTR_X.cpp -o gNetVarTR_X
g++ calcolaRitardiHL.cpp -o calcolaRitardiHL
g++ calcolaRitardiLH.cpp -o calcolaRitardiLH
g++ somma.cpp -o somma
g++ confronta.cpp -o confronta
path_=$(pwd)
creaNET_C=${path_}/./gNetVarC
creaNET_TR_X=${path_}/./gNetVarTR_X
trovaRITARDI_HL=${path_}/./calcolaRitardiHL
trovaRITARDI_LH=${path_}/./calcolaRitardiLH
SOMMA=${path_}/./somma
CONFRONTA=${path_}/./confronta
TEST_=${path_}/./test
###################################################################
# Cin HL
###################################################################
simulazioni=0
for XX in 1 2 3 4 5 10 20
do
    for tr in 10 50
    do
            ${creaNET_TR_X} "${cella}/${ingresso}/${netHL}" $tr $XX
            cd "${cella}/${ingresso}"
            netlist_=$(ls *ps.net)
            ngspice -b ${netlist_} > "1.txt"
            delayHL=$(${trovaRITARDI_HL} "1.txt")
            rm ${netlist_} "1.txt"
```

```
#echo "il ritardo è ${delayHL}"
cd ${path_}
Cl=0
HLfind=1
while [ $HLfind != 0 ]
do
    simulazioni=$(${SOMMA} ${simulazioni} 1)
    Cl=$(${SOMMA} $Cl 1)
    ${creaNET_C} ${netHL_cl} ${Cl} $XX $tr
    netlist_=$(ls *fF.net)
    ngspice -b ${netlist_} > "1.txt"
    delayHL_1=$(${trovaRITARDI_HL} "1.txt")
    rm ${netlist_} "1.txt"
    HLfind=$(${CONFRONTA} ${delayHL_1} ${delayHL})
done
HLfind=1
Cl=$(${SOMMA} $Cl -1)
while [ $HLfind != 0 ]
do
    simulazioni=$(${SOMMA} ${simulazioni} 1)
    Cl=$(${SOMMA} $Cl 0.1)
    ${creaNET_C} ${netHL_cl} ${Cl} $XX $tr
    netlist_=$(ls *fF.net)
    ngspice -b ${netlist_} > "1.txt"
    delayHL_1=$(${trovaRITARDI_HL} "1.txt")
    rm ${netlist_} "1.txt"
    HLfind=$(${CONFRONTA} ${delayHL_1} ${delayHL})
    done
    HLfind=1
    Cl=$(${SOMMA} $Cl -0.1)
    while [ $HLfind != 0 ]
    do
        simulazioni=$(${SOMMA} ${simulazioni} 1)
    Cl=$(${SOMMA} $Cl 0.01)
    ${creaNET_C} ${netHL_cl} ${Cl} $XX $tr
    netlist_=$(ls *fF.net)
    ngspice -b ${netlist_} > "1.txt"
    delayHL_1=$(${trovaRITARDI_HL} "1.txt")
    rm ${netlist_} "1.txt"
    HLfind=$(${CONFRONTA} ${delayHL_1} ${delayHL})
```

```
done
HLfind=1
Cl=$(${SOMMA} $Cl -0.01)
while [ $HLfind != 0 ]
do
    simulazioni=$(${SOMMA} ${simulazioni} 1)
    Cl=$(${SOMMA} $Cl 0.001)
    ${creaNET_C} ${netHL_cl} ${Cl} $XX $tr
    netlist_=$(ls *fF.net)
    ngspic1 2 3 4 5 10 20e -b ${netlist_} > "1.txt"
    delayHL_1=$(${trovaRITARDI_HL} "1.txt")
    rm ${netlist_} "1.txt"
        HLfind=$(${CONFRONTA} ${delayHL_1} ${delayHL})
done
Cl=$(${SOMMA} $Cl -0.001)
HLfind=1
while [ $HLfind != 0 ]
do
    simulazioni=$(${SOMMA} ${simulazioni} 1)
    Cl=$(${SOMMA} $Cl 0.0001)
    ${creaNET_C} ${netHL_cl} ${Cl} $XX $tr
    netlist_=$(ls *fF.net)
    ngspice -b ${netlist_} > "1.txt"
    delayHL_1=$(${trovaRITARDI_HL} "1.txt")
    rm ${netlist_} "1.txt"
    HLfind=$(${CONFRONTA} ${delayHL_1} ${delayHL})
done
Cl=$(${SOMMA} $Cl -0.0001)
simulazioni=$(${SOMMA} ${simulazioni} 1)
${creaNET_C} ${netHL_cl} ${Cl} $XX $tr
netlist_=$(ls *fF.net)
ngspice -b ${netlist_} > "1.txt"
delayHL_1=$(${trovaRITARDI_HL} "1.txt")
rm ${netlist_} "1.txt"
echo "X=$XX tr=$tr ps"
echo "delayHLbase= ${delayHL}"
echo "delayHLtrovato= ${delayHL_1}"
echo "CloadHL= ${Cl}"
```

```
    echo "X=$XX tr=$tr ps delayHLbase= ${delayHL} \
        delayHLtrovato= ${delayHL_1} CloadHL= ${Cl}"\
        >> test${cella}_${ingresso}.txt
    echo "X= $XX tr= $tr ps \
        delayHLbase= ${delayHL} delayHLtrovato= \
        ${delayHL_1} CloadHL= ${Cl} " >> \
        test${cella}_${ingresso}GRAF.txt
    done
done
    echo "simulazioni totali per trovare 14 CHL: ${simulazioni}"
    echo "simulazioni totali per trovare 14 CHL: ${simulazioni}" \
        >> test${cella}_${ingresso}.txt
        echo " " >> test${cella}_${ingresso}.txt
###################################################################
# Cin LH
###################################################################
simulazioni=0
for XX in 1 2 3 4 5 10 20
do
    for tr in 10 50
    do
        ${creaNET_TR_X} "${cella}/${ingresso}/${netLH}" $tr $XX
        cd "${cella}/${ingresso}"
        netlist_=$(ls *ps.net)
        ngspice -b ${netlist_} > "1.txt"
        delayLH=$(${trovaRITARDI_LH} "1.txt")
        rm ${netlist_} "1.txt"
        #echo "il ritardo è ${delayLH}"
        cd ${path_}
        Cl=0
        LHfind=1
        while [ $LHfind != 0 ]
        do
            simulazioni=$(${SOMMA} ${simulazioni} 1)
                Cl=$(${SOMMA} $Cl 1)
                ${creaNET_C} ${netLH_cl} ${Cl} $XX $tr
            netlist_=$(ls *fF.net)
            ngspice -b ${netlist_} > "1.txt"
                delayLH_1=$(${trovaRITARDI_LH} "1.txt")
                rm ${netlist_} "1.txt"
                LHfind=$(${CONFRONTA} ${delayLH_1} ${delayLH})
            done
            LHfind=1
```

201 202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252

```
    Cl=$(${SOMMA} $Cl -1)
    while [ $LHfind != 0 ]
    do
        simulazioni=$(${SOMMA} ${simulazioni} 1)
        Cl=$(${SOMMA} $Cl 0.1)
        ${creaNET_C} ${netLH_cl} ${Cl} $XX $tr
    netlist_=$(ls *fF.net)
    ngspice -b ${netlist_} > "1.txt"
    delayLH_1=$(${trovaRITARDI_LH} "1.txt")
    rm ${netlist_} "1.txt"
    LHfind=$(${CONFRONTA} ${delayLH_1} ${delayLH})
    done
    LHfind=1
    Cl=$(${SOMMA} $Cl -0.1)
    while [ $LHfind != 0 ]
    do
        simulazioni=$(${SOMMA} ${simulazioni} 1)
        Cl=$(${SOMMA} $Cl 0.01)
        ${creaNET_C} ${netLH_cl} ${Cl} $XX $tr
        netlist_=$(ls *fF.net)
        ngspice -b ${netlist_} > "1.txt"
        delayLH_1=$(${trovaRITARDI_LH} "1.txt")
        rm ${netlist_} "1.txt"
        LHfind=$(${CONFRONTA} ${delayLH_1} ${delayLH})
    done
    LHfind=1
    Cl=$(${SOMMA} $Cl -0.01)
    while [ $LHfind != 0 ]
    do
        simulazioni=$(${SOMMA} ${simulazioni} 1)
        Cl=$(${SOMMA} $Cl 0.001)
        ${creaNET_C} ${netLH_cl} ${Cl} $XX $tr
        netlist_=$(ls *fF.net)
        ngspice -b ${netlist_} > "1.txt"
        delayLH_1=$(${trovaRITARDI_LH} "1.txt")
        rm ${netlist_} "1.txt"
        LHfind=$(${CONFRONTA} ${delayLH_1} ${delayLH})
    done
Cl=$(${SOMMA} $Cl -0.001)
```


## LHfind=1

while [ \$LHfind ! = 0 ]
do
simulazioni=\$(\$\{SOMMA\} \$\{simulazioni\} 1)
$\mathrm{Cl}=\$(\$\{\mathrm{SOMMA}\} \$ \mathrm{Cl} 0.0001)$
\$\{creaNET_C\} \$\{netLH_cl\} \$\{Cl\} \$XX \$tr
netlist_=\$(ls *fF.net)
ngspice -b \$\{netlist_\} > "1.txt"
delayLH_1=\$(\$\{trovaRITARDI_LH\} "1.txt")
rm \$\{netlist_\} "1.txt"
LHfind=\$(\$\{CONFRONTA\} \$\{delayLH_1\} \$\{delayLH\})
done
$\mathrm{Cl}=\$(\$\{\mathrm{SOMMA}\} \$ \mathrm{Cl}-0.0001)$
simulazioni=\$(\$\{SOMMA\} \$\{simulazioni\} 1)
\$\{creaNET_C\} \$\{netLH_cl\} \$\{Cl\} \$XX \$tr
netlist_=\$(ls *fF.net)
ngspice -b \$\{netlist_\} > "1.txt"
delayLH_1=\$(\$\{trovaRITARDI_LH\} "1.txt")
rm \$\{netlist_\} "1.txt"
echo "X=\$XX tr=\$tr ps"
echo "delayLHbase= \$\{delayLH\}"
echo "delayLHtrovato= \$\{delayLH_1\}"
echo "CloadLH= \$\{Cl\}"
echo ${ }^{X}=\$ X X$ tr=\$tr ps delayLHbase= \$\{delayLH\} \}
delayLHtrovato= \$\{delayLH_1\} CloadLH= \}
\$\{Cl\}" >> test\$\{cella\}_\$\{ingresso\}.txt
echo $" X=\$ X X$ tr= \$tr ps \}
delayHLbase = \$\{delayLH\} delayHLtrovato= \}
\$\{delayLH_1\} CloadHL= \$\{Cl\} \}
" >> test\$\{cella\}_\$\{ingresso\}GRAF.txt
done
done
echo "simulazioni totali per trovare 14 CLH: \$\{simulazioni\}"
echo "simulazioni totali per trovare 14 CLH: \$\{simulazioni\}" \}
>> test\$\{cella\}_\$\{ingresso\}.txt
exit

## C.3. Example: Deterministic circuit level simulation

```
#!/bin/bash
```

```
g++ calcolaRitardi.cpp -o calcolaRitardi
g++ gNetVarC.cpp -o gNetVarC
g++ gNetVarTR_X.cpp -o gNetVarTR_X
sleep 1
export DATE_INIZIO=$(date)
##########################################################
fileBASE="64_fa1"
ingresso="mezzo"
rimuovi=NO #remove last result
crea=SI #create folder structure
cfile=SI #copy netlist in folder structure
simula=SI #simulate
##########################################################
if [ $rimuovi == "SI" ]
then
    echo "remove last result in progress..."
    for tr in 10 20 30 40 50
    do
        for XX in 1 5 10 20
        do
            cartella="X${XX}_${tr}ps"
            cd $cartella
            rm *.txt
            rm *fF.net
            cd ..
                        #echo create $cartella
                    done
    done
fi
if [ $crea == "SI" ]
then
    echo "creo le cartelle"
    for tr in 10 50 #20 30 40 50
    do
        for XX in 1 10 #1 5 10 20
        do
            cartella="X${XX}_${tr}ps"
            cartella="X${XX}_${tr}ps"
                        mkdir $cartella
                                echo creata $cartella
                                done
    done
fi
if [ $cfile == "SI" ]
then
    echo "file copy..."
```

```
    for tr in 10 20 30 40 50
    do
    for XX in 1 5 10 20
    do
        cartella="X${XX}_${tr}ps"
        for i in ${fileBASE}*.net
        do
            ./gNetVarTR_X $i $tr $XX
                done
                mv *ps.net $cartella
            done
    done
fi
if [ $simula == "SI" ]
then
    #init simulations
    CL[0]=0.00;
    CL[1]=0.15;
    CL[2]=0.33;
    CL [3]=0.66;
    CL[4]=1.00;
    CL[5]=1.25;
    CL[6]=1.5;
    CL[7]=1.75;
    CL[8]=2;
    CL[9]=3
    CL[10]=4;
    CL[11]=5;
    CL[12]=6;
    CL[13]=7;
    CL[14]=8;simulation at circuit level simulation
    CL[15]=9;
    CL[16]=10;
    CL[17]=11;
for tr in 10 20 30 40 50
do
    for XX in 1 5 10 20
    do
        cartella="X${XX}_${tr}ps"
        cd $cartella
        j=1
        echo "X= ${XX} tr= ${tr}" >> ../ritardi_.txt
        echo "Cload lh_cin X${XX}_${tr}ps\
                            hl_cin X${XX}_${tr}ps" >> ../ritardi_.txt
        echo "X= ${XX} tr= ${tr}" >> \
                        "ritardi_Cl_X${XX}_tr${tr}ps.txt"
        echo "Cload lh_cin X${XX}_${tr}ps hl_cin \
            X${XX}_${tr}ps" >> "ritardi_Cl_X${XX}_tr${tr}ps.txt"
```

107


```
        do
            j=1
            .././gNetVarC "${fileBASE}X${XX}_tr${tr}ps.net" \
                                    "${CL[$cload]}"
            for i in *fF.net
            do
                ngspice -b $i > "${j}.txt"
                j=$[$j+1]
            done
            .././calcolaRitardi "tauCl_X${XX}_tr${tr}ps.txt" \
                    "1.txt" "2.txt" "3.txt" "4.txt" "${CL[$cload]}_\
                    ${CL[$cload+1]}" "${XX}" "${CL[$a]}" "${CL[$cload]}"
            rm "1.txt" "2.txt" "3.txt" "4.txt"
        done
        echo "" >> ../ritardi_.txt
        cd ..
        done
        echo "" >> ritardi_.txt
        done
fi
mv ritardi_.txt ritardi_${fileBASE}.txt
echo "inizio ${fileBASE} ${ingresso} : $DATE_INIZIO" > \
        time_${fileBASE}.txt
echo "fine ${fileBASE} ${ingresso} : $(date)" >> \
        time_${fileBASE}.txt
echo "inizio ${fileBASE} ${ingresso}: $DATE_INIZIO"
echo "fine ${fileBASE} ${ingresso}: $(date)"
exit
```


## C.4. Example: Statistical circuit level simulation

```
#!/bin/bash
pathATTUALE="variazioniCASUALI"
cprogramm="cprogramm"
cella="a1b0c_cout_x1_tr10_cl1"
fileBASE=$1
PARAMSPATH=../../TechParams
iterazioni=10000
j=45
```

```
declare -i COUNT=0
declare -i CCOUNT=0
if [ -e $pathATTUALE ]
then
    echo "work folder find"
else
    mkdir $pathatTUALE
fi
#**********************************************
# technology parameter
#*************************************************
        COUNT=0
        for i in 'cat $PARAMSPATH/Toxp_$j.par'; do
        Toxp[$COUNT]=$i
        COUNT=COUNT +1
        done;
            COUNT=0
            for i in 'cat $PARAMSPATH/L_$j.par'; do
        L[$COUNT]=$i
        COUNT=COUNT+1
            done;
            COUNT=0
            for i in 'cat $PARAMSPATH/W_$j.par'; do
        W[$COUNT]=$i
            COUNT= COUNT +1
            done;
            COUNT=0
            for i in 'cat $PARAMSPATH/Nn_$j.par'; do
                Nn[$COUNT]=$i
                COUNT=COUNT+1
            done;
            COUNT=0
            for i in 'cat $PARAMSPATH/Np_$j.par'; do
        Np[$COUNT]=$i
        COUNT=COUNT+1
            done;
#********************************************************
# End technology parameter
#**************************************************
#********************************************************
# begin simulations
#****************************************************
XX=1
tr=10
cload=1
echo "begin simulations"
```

```
echo
echo "iterazione delay_lh_${perc}X${XX}_${tr}ps \
    delay_hl_${perc}${cload}_X${XX}_${tr}ps X tr\
    cload L W Ndepn Ndepp tox" >> \
    ${pathATTUALE}/delay_random.txt
COUNT=0
# delay random value
#**************************************************************
while [ "$COUNT" -lt "$iterazioni" ]
do
    ../../${cprogramm}/./genNETvar ${fileBASE} ${L[$COUNT]} \
        ${W[$COUNT]} ${Nn[$COUNT]} ${Np[$COUNT]} ${Toxp[$COUNT]}\
        ${tr} ${XX} ${cload}
    mv *_.net ${pathATTUALE}
    cd ${pathATTUALE}
    i=0
    for netS in *_.net
    do
        ngspice -b $netS > "${COUNT}_${i}.txt"
        i=$[$i+1]
    done
    ../../../${cprogramm}/./calcolaDELAYrandom delay_random.txt\
    "${COUNT}_0.txt" "${COUNT}_1.txt" "${COUNT}_2.txt" \
    "${COUNT}_3.txt" "${COUNT}" ${tr} ${XX} ${cload} \
        ${L[$COUNT]} ${W[$COUNT]} ${Nn[$COUNT]} ${Np[$COUNT]} \
        ${Toxp[$COUNT]}
    rm ${COUNT}*.txt
    COUNT=COUNT+1
    cd ..
done
exit
```


## C.5. General use: deleting a type of file in all subdirectory

```
#!/bin/bash
path_=$(pwd)
for i in $(ls)
```

```
do
    if [ -d $i ] && [ $i != "model" ]
    then
        cd ${path_}/${i}
        for folder in $(ls)
        do
            if [ -d $folder ]
            then
                cd ${path_}/${i}/${folder}
                rm *.mat
                cd ${path_}/${i}
            fi
        done
        cd ${path_}
    fi
done
exit
```


## C.6. General use: modify a file whit sed command

```
#!/bin/bash
new_tr="1 5 10 20 30 40 50"
new_X="1 2 3 4 5 10 20"
new_CL=(0.0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 \
    0.65 0.7 0.75 0.8 0.85 0.9 0.95 1.0 1.25 1.5 1.75 2.0
    2.75 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0)
new_CL_="0.0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 \
    0.65 0.7 0.75 0.8 0.85 0.9 0.95 1.0
    2.75 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0"
```



```
    0.65 0.7 0.75 0.8 0.85 0.9 0.95 1 1.25 1.5 1.75 2.0 2.25 2.50 \
    2.75 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0)
new_CL1_="0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55 0.6 \
    0.65 0.7 0.75 0.8 0.85 0.9 0.95 1 1.25 1.5 1.75 2.0
    2.75 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0"
len=${#new_CL[*]}
len=${#new_CL1[*]}
let len--
if [ 2 = 2 ]
then
```

```
path_= "$(pwd)"
for i in $(ls)
do
    if [ -d $i ]
    then
        cd ${path_}/${i}
        for folder in $(ls)
        do
                if [ -d $folder ]
                then
                cd ${path_}/${i}/${folder}
                sed s/'for tr in'/"for tr in $new_tr #"/ \
                        creaCARTtr1_10_50ps_X1_10_20_Cvar > run_tau_1.sh
                sed s/'for XX in'/"for XX in $new_X #"/ run_tau_1.sh \
                        > run_tau_2.sh
                sed s/'CL\[0\]'/"CL=(${new_CL_}) # CL[0]"/ run_tau_2.sh \
                        > run_tau_3.sh
                sed s/'CL\[1\]'/"CL1=(${new_CL1_}) # CL[1]"/ run_tau_3.sh\
                    > run_tau_4.sh
                sed s/'CL\[2\]'/"# CL[2]"/ run_tau_4.sh > run_tau_5.sh
                sed s/'CL\[3\]'/"# CL[3]"/ run_tau_5.sh > run_tau_6.sh
                sed s/'CL\[4\]'/"# CL[4]"/ run_tau_6.sh > run_tau_7.sh
                sed s/'CL\[5\]'/"# CL[5]"/ run_tau_7.sh > run_tau_8.sh
                sed s/'CL\[6\]'/"# CL[6]"/ run_tau_8.sh > run_tau_9.sh
                sed s/'CL\[7\]'/"# CL[7]"/ run_tau_9.sh > run_tau_10.sh
                sed s/'CL\[8\]'/"# CL[8]"/ run_tau_10.sh > run_tau_11.sh
                sed s/'CL\[9\]'/"# CL[9]"/ run_tau_11.sh > run_tau_12.sh
                sed s/'CL\[10\]'/"# CL[10]"/ run_tau_12.sh > run_tau_13.sh
                sed s/'CL\[11\]'/"# CL[11]"/ run_tau_13.sh > run_tau_14.sh
                sed s/'CL\[12\]'/"# CL[12]"/ run_tau_14.sh > run_tau_15.sh
                sed s/'CL\[13\]'/"# CL[13]"/ run_tau_15.sh > run_tau_16.sh
                sed s/'CL\[14\]'/"# CL[14]"/ run_tau_16.sh > run_tau_17.sh
                sed s/'CL\[15\]'/"# CL[15]"/ run_tau_17.sh > run_tau_18.sh
                sed s/'CL\[16\]'/"# CL[16]"/ run_tau_18.sh > run_tau_19.sh
                sed s/'CL\[17\]'/"# CL[17]"/ run_tau_19.sh > run_tau_20.sh
                sed s/"..\/..\/..\/.\/gNetVarC"/"..\/.\/gNetVarC"/ \
                        run_tau_20.sh > run_tau_21.sh
                sed s/'for cload in'/"for cload in {0..${len}} # for \
                        cload in"/ run_tau_21.sh > run_tau_22.sh
                sed s/'a=$\[$cload+1\]'/"#a=$\[\$cload+1\]"/ \
                    run_tau_22.sh > run_tau_23.sh
                    sed s/'\${CL\[\$a\]}'/"\${CL1\[\$cload\]}"/ \
                        run_tau_23.sh > run_tau_24.sh
```

```
                sed s/'\/bin\/bash'/"\/bin\/bash\n\ng++ \
                        calcolaTAU*.cpp -o calcolaTAU\ng++ gNetVarC.cpp \
                        -o gNetVarC\ng++ gNetVarTR_X.cpp -o gNetVarTR_X"/ \
                        run_tau_24.sh > run_tau.sh
                rm run_tau_*.sh
                    #rm run_tau2.sh
                    chmod +x run_tau.sh
                    cd ${path_}/${i}
            fi
        done
        cd ${path_}
        fi
    done
else
    echo "..."
    exit
fi
exit
```


## D. Ngspice netlist

## D.1. Ngspice

NGspice is an open source software used for general purpose circuit simulation for linear and non-linear circuit analysis. NGspice supports analog, digital and mixedmode simulations and provides various modes of analysis like DC Analysis (Operating Point, Transfer Function, DC Sweep), AC Small-Signal Analysis, Transient Analysis, Pole-Zero Analysis, Small-Signal Distortion Analysis, Sensitivity Analysis, Noise Analysis etc.

The general structure of an NGspice netlist consists of a first line as a title line, which is generally the name of netlist, and must end up with .END command. In between of these two lines, the netlist of the circuit is composed of several set of lines, which define the circuit topology and element values, analysis description, output description, model parameters, and a set of control lines. All lines are put together and taken as input file by NGspice. NGspice is format free i.e. the order of lines is arbitrary and case insensitive.

NGspice can be used either in interactive or in batch mode. Interactive mode allows the user to load the netlist and proceed by providing the required commands through terminal, while batch mode collects all tasks in a single script and execute them altogether, without further interaction with user. In the following we show the main aspects of both simulation modes.

In interactive mode, to load a circuit , either write in the terminal "ngspice" followed
by the name of netlist, or simply invoke "ngspice" and use source command to load the netlist as shown below. In both the cases, after fetching the netlist, the NGspice prompt writes the first line of netlist, which generally consists of the name of the netlist, e.g. "basic_inverter" and then wait for other commands.

```
$ngspice name.net
ngspice 1 ->
```

or

```
$ngspice
ngspice 1 -> source name.net
ngspice 2 ->
```

Now, you can run the simulation directly. To run the simulation write run command in the shell. If netlist doesn't have errors, NGspice writes a set of information like temperature, initial values of nodes, number of time steps it takes to calculate the result of measure command and after completing the execution, the simulator will return with NGspice prompt. If any error had been encountered, the corresponding message would be displayed on the terminal. Before or after the run command you can write other commands to verify the netlist, alter parameters, save data or load new netlists. In the following, a selection of the most important commands will be presented.

## D.1.1. Show and showmod commands

The show command is used to print the actual value of parameters, especially to check them after changes have been issued. We remind that, due to NGspice parser code policy, if you don't provide space chars in particular parts of command line, it will fail to work. For the show command, a space char is needed before and after
the ":" character. The show command can be used to retrieve information on one or several devices.

The output of the above command is a list of all parameters of device(s). To view one or more parameters of one or several devices, the show command can be used in an extended way.

We can use showmod command if we are interested in listing the values of model parameters (e.g. threshold voltage, oxide thickness, etc.). This command has similar syntax of show command.

An example of the show and showmod command are the following:

```
ngspice 2 ->show mp1 : w
    BSIM4v6: Berkeley Short Channel IGFET Model-4
        device mp1
        model pmos
            w 9e-08
ngspice 3 ->show mn1,mp1 : w,l
    BSIM4v6: Berkeley Short Channel IGFET Model-4
        device mp1 mn1
        model pmos nmos
            w 9e-08 4.5e-08
            1 4.5e-08 4.5e-08
ngspice 4 ->showmod mp1,mn1 : vth0,toxe
    BSIM4v6 models (Berkeley Short Channel IGFET Model-4)
        model pmos nmos
            vth0 -0.23122 0.3423
            toxe 9.2e-10 9e-10
```


## D.1.2. Alter and altermod commands

The alter command is used to change any parameter of a device, e.g.the width (W) or length (L) of MOS transistors, without changing the whole netlist file. For example, alter command is used on a PMOS device (mp1) whose width parameter (W) is set to 100 nm :

```
ngspice 5 ->alter @mp1[w] 100n
ngspice 6 ->show mp1 : w
    BSIM4v6: Berkeley Short Channel IGFET Model-4
    device mp1
    model pmos
        W 1e-07
```

In a similar way, altermod command operates on models and is used to change model parameters. The example shown below changes the threshold voltage parameter (vth0) on a BSIM4 model:

```
ngspice 7 ->showmod mn1 : vth0
    BSIM4v6 models (Berkeley Short Channel IGFET Model-4)
    model nmos
    vth0 0.3423
ngspice 8 -> altermod @mn1[vth0] 0.4
ngspice 9 -> showmod mn1 : vth0
    BSIM4v4 models (Berkeley Short Channel IGFET Model-4)
        model nmos
        vth0 0.4
```


## D.1.3. Setcirc command

We already mentioned that the source command is used to load a netlist. Multiple netlists can be loaded repeating the source command. When a netlist is loaded, it becomes the current circuit. The setcirc command is used to select the current netlist.

A practical example is shown below. At first, we source a netlist, named basic_inverter.net, then we source a second one (copy_basic_inverter.net).Using the command setcirc without parameters, a list of loaded netlists will be printed, followed by a request to set as active the desired circuit number.

Similarly to setcirc, the setplot command can be used to select a specific plot as the current plot.

```
ngspice 1 ->source basic_inverter.net
Circuit: Basic_Inverter
ngspice 2 ->source Copy_basic_inverter.net
Circuit: Copy_Basic_Inverter
ngspice 3 ->setcirc
    Type the number of the desired circuit:
Current 1 Copy_Basic_Inverter
            2 Basic_Inverter
? 2
ngspice 4 ->run
ngspice 5 ->setcirc
    Type the number of the desired circuit:
        1 Copy_Basic_Inverter
Current 2 Basic_Inverter
? 1
ngspice 6 ->run
```

```
ngspice 7 ->setplot
    Type the name of the desired plot:
        new New plot
Current tran2 Copy_Basic_Inverter (Transient Analysis)
        tran1 Basic_Inverter (Transient Analysis)
        const Constant values (constants)
? tran1
ngspice 8 ->setplot
    Type the name of the desired plot:
        new New plot
        tran2 Copy_Basic_Inverter (Transient Analysis)
Current tran1 Basic_Inverter (Transient Analysis)
        const Constant values (constants)
? tran2
ngspice 9 ->setcirc
    Type the number of the desired circuit:
Current 1 Copy_Basic_Inverter
            2 Basic_Inverter
? 2
ngspice 10 ->alter @mn1[w] 90n
ngspice 11 ->run
ngspice 12 ->setplot
    Type the name of the desired plot:
        new New plot
Current tran3 Basic_Inverter (Transient Analysis)
    tran2 Copy_Basic_Inverter (Transient Analysis)
    tran1 Basic_Inverter (Transient Analysis)
    const Constant values (constants)
```


## D.1.4. Print command

The print command is used to print the vectors values on screen. The type of analysis can be TRAN, DC, AC, NOISE, etc. while the number of output variables in a single print command line can be up to eight. There is no limit on the number of .print lines for each type of analysis. Algebraic expressions are also allowed in .print command lines and expression may contain numerical values, constants, predefined functions, simulator output, parameters defined by a .param statement etc.

## D.1.5. Write command

The write command is used to write data or expressions to a file. The default format is a compact binary file, but it can be changed to ASCII with the "set filetype $=$ ascii" command. An example for basic_inverter.net netlist is shown below, where the write command is used to produce a file containing voltage values for nodes marked "nodea" and "nodez".

```
Title: basic_inverter
Date: Wed Jun 27 10:50:00 2012
Plotname: Transient Analysis
Flags: real
No. Variables: 3
No. Points: 10020
Variables:
    O time time
    1 nodea voltage
    2 nodez voltage
Values:
V 0.000000000000000e+00
    0.000000000000000e+00
    9.998529155265871e-01
1 1.000000000000000e-15
    0.000000000000000e+00
    9.998529155265875e-01
..
10019 1.000000000000000e-09
```

```
0.000000000000000e+00
9.909450186479611e-01
```


## D.1.6. .meas command

Another important statement in NGspice netlists is .meas or .measure. In general it can be used to analyze the output data of a TRAN, AC or DC simulation. The command is executed immediately after the simulation has finished.

The .meas command prints the results of a user-defined data analysis to the standard output. The user defined analysis may include the measurement of propagation delays, rise time, fall time, peak-to-peak voltage, minimum or maximum voltage, the integral or derivative over a specified period, depending on the type of analysis. It is a measure of time lapse between events. Care is needed to understand the right evolution of signals, otherwise the measurement will be wrong without showing errors.

If we consider the .meas analysis in a netlist, we can see that trigger-target (trigtarg) options has been used to measure the time lapse between two points that, by definition are the low-to-high and high-to-low propagation delays. We report the lines for better clarity:

```
.meas tran delay_LH trig v(nodea) val=0.5 fall=1 \
    targ v(nodeZ) val=0.5 rise=1
.meas tran delay_HL trig v(nodea) val=0.5 rise=1 \
    targ v(nodeZ) val=0.5 fall=1
```

Going inside each statement, we can see that the first line measures the time between V (nodea) reaching $50 \%$ of Vdd (i.e. 0.5 V , since Vdd is 1 V ) in the first falling edge and V (nodeZ) reaching $50 \%$ of Vdd for the first rising edge (that is, by definition,
$t_{L H}$ delay).Similarly, the second line measures the time between V(nodea) reaching $50 \%$ of $V_{d d}$ in the first rising edge and $\mathrm{V}\left(\right.$ nodeZ) reaching $50 \%$ of $V_{d d}$ for the first falling edge (i.e. $t_{H L}$ delay).

## D.1.7. Batch mode

To work in batch mode, NGspice uses -b command line option. The sintax to launch NGspice in batch mode is:

1

```
ngspice -b <netlist name>
```

The above command allows NGspice to write outputs in the terminal (e.g. .meas values, .print, etc).

To write the same outputs in a file.

```
ngspice -b <netlist name> -o <outputfilename>
```

A netlist can contain a command script section (delimited by.control and.endc statemets) that allows users to automate complex simulations. Within this section, simulations can be run multiple times while altering parameters, and output vectors can be analyzed and plotted for each iteration. Expressions, functions, constants, commands, variables, vectors and control structures may be also assembled into such scripts. The section can contain loops and conditional code.

## D.2. Example: inverter netlist

```
Inverter netlist
.option filetype=ascii
.PARAM Lmin \(=45\) n
.PARAM Wmin \(=45\) n
.PARAM XX=1
.PARAM \(\mathrm{tr}=10 \mathrm{p}\)
.INCLUDE ../../model/45nm_MGK.pm
.TRAN 0.1p 400p
.PRINT tran V(nodeIN) V(nodeZ)
. meas tran delay_LH \}
    trig \(v(\) nodeIN \()\) val \(=0.5\) fall \(=1 \backslash\)
    targ \(\mathrm{v}(\) nodeZ \()\) val \(=0.5\) rise \(=1\)
. meas tran delay_HL \}
    trig v(nodeIN) val=0.5 rise=1 \}
    \(\operatorname{targ} \mathrm{v}(\) nodeZ \()\) val \(=0.5\) fall \(=1\)
Mn1 nodeZ nodeIN 00 nmos \(\mathrm{W}=\{\mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}\)
Mp1 nodeZ nodeIN node1 node1 pmos \(\mathrm{W}=\{\mathrm{XX} * 1 * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}\)
Cout nodeZ 0 3f
Vin nodeIN 0 PWL(0 0 10p \(0\{10 \mathrm{p}+\mathrm{tr}\} 1200 \mathrm{p} 1\{200 \mathrm{p}+\mathrm{tr}\} 0)\)
Vdd node1 0 1
end
```


## D.3. Subcircuits netlist

```
*Designed by Antonio Mastrandrea
**************************************************************************
*subcircuit
*examples
*xnand1 0 node1 nodeZ nodeA nodeB NAND2_SUB <--drive strength =1
*xnand2 0 node1 nodeZ nodeA nodeB NAND2_SUB XX=2 <--drive strength =2
*xxor1 0 node1 nodeZ nodeB nodeA XOR2_SUB
```

```
************************************************************************
* not
************************************************************************
* Wmin, Lmin and MOS model must be declarate before
*
*drive strength - _+
*inA - + /
*uscita - _ / /
*vdd - _ / / /
*massa - _ _ / / /
*
* v v v v v
.subckt NOT_SUB 1 2 3 4 XX=1
Mn1 3 4 4 1 1 1 nmos W={1*XX*Wmin } L={Lmin}
Mp1 3 4 2 2 pmos W={2*XX*Wmin} L={Lmin }
. ends
*************************************************************************
********************************************************************
* nand2
**************************************************************************
* Wmin, Lmin and MOS model must be declarate before
*
*drive strength - - +
*inB --mos uscita ——_+ /
*inA -_mos massa___+ / /
*uscita - _ / / /
*vdd - _+ / / / /
*massa - +l l l l l l l l
* v v v v v v
.subckt NAND2 SUB 1 2 3 4 5 XX=1
Mn1 6
Mn2 3 5 6 1 nmos W={2*XX*Wmin} L={Lmin}
Mp1 3 4 4 2 2 pmos W={2*XX*Wmin} L={Lmin }
Mp2 3 5 2 2 pmos W={2*XX*Wmin} L={Lmin}
.ends
***************************************************************************
*************************************************************************
* nand3
***********************************************************************
*drive strength--+
*inC--+ /
*inB -_+_ / /
*inA _ _ / / /
*output _ _ / / /
*vdd - _ _ / / / |
*ground-_+_/ / / / / /
*
* v v v v v v v
.subckt NAND3 SUB 1 2 3 4 5 6 XX=1
```

```
Mn1 7 4 1 1 nmos W={3*XX*Wmin} L={Lmin}
Mn2 8 5 5 7 1 nmos W={3*XX*Wmin} L={Lmin}
Mn3 3 3 6 8 8 1 nmos W={3*XX*Wmin} L={Lmin}
Mp1 3 4 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mp2 3 5 2 2 pmos W={2*XX*Wmin } L={Lmin }
Mp3 3 6 2 2 2 pmos W={2*XX*Wmin } L={Lmin }
.ends
************************************************************************
**********************************************************************
* nand4
*************************************************************************
*drive strengh--+
*inD-_+1
*inC--+ / /
*inB - _+ / / /
*inA - _ / / / /
*output - + + 1 1 1/ 1 1
*vdd - _+ 1 | | | | 1 
*ground-<+1 1 1 1 1 1 1
*
* v v v v v v v v
.subckt NAND4_SUB 1 2 2 3 4 5 6 7 XX=1
Mn1 8 4 1 1 nmos W={4*XX*Wmin} L={Lmin}
Mn2 9 5 5 8 1 nmos W={4*XX*Wmin } L={Lmin }
Mn3 10 6 6 9 1 1 nmos W={4*XX *Wmin} L={Lmin}
Mn4 3 7 7 10 1 nmos W={4*XX*Wmin} L={Lmin }
Mp1 3 4 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mp2 3 5 5 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mp3 3 6 6 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mp4 3 7 2 2 pmos W={2*XX*Wmin } L={Lmin}
. ends
****************************************************************************
**********************************************************************
* nor2
**************************************************************************
* Wmin, Lmin and MOS model must be declarate before
*drive strength - _+
*inB --mos uscita-_+ /
*inA - mos vdd—__+ / /
*uscita _ _ / / /
*vdd - + / / 1 / 
*massa - _ + / / / /
*
* v v v v v v
.subckt NOR2_SUB 1 2 3 4 5 XX=1
Mn1 3 4 1 1 1 nmos W={1*XX*Wmin} L={Lmin}
```



```
170 Mp1 10 4 2 2 pmos W={8*XX*Wmin} L={Lmin}
171 Mp2 9 5 10 2 pmos W={8*XX*Wmin} L={Lmin}
172 Mp3 8 6 9 9 2 pmos W={8*XX*Wmin} L={Lmin}
173 Mp4 3 7 8 2 pmos W={8*XX*Wmin}} L={Lmin
174
175 . ends
176
1 7 7
178
179
180
1 8 1
182
183
1 8 4
185
186
187
188
189
1 9 0
**************************************************************************
***********************************************************************
* xor2
*************************************************************************
    * Wmin, Lmin and MOS model must be declarate before
*
*drive strength -
*inB - + / 
*inA - + / /
*uscita - + / / /
*vdd - _ + / / 1
*massa __+ ll l l l ll
* v v v v v v
.subckt XOR2_SUB 1
*NOT
Mp5 node5n 5 2 2 pmos L={Lmin} W ={2*XX*Wmin}
Mn5 node5n 5
Mp6 node4n 4 2 2 pmos L={Lmin} W={2*XX*Wmin}
Mn6 node4n 4 1 1 nmos L={Lmin} W={XX*Wmin}
*PULL UP
Mp1 3 4 int4 2 pmos L={Lmin } W={4*XX*Wmin }
Mp2 3 node4n int1 2 pmos L={Lmin } W={4*XX*Wmin}
Mp3 int4 node5n 2 2 pmos L={Lmin } W={4*XX*Wmin }
Mp4 int1 5 2 2 pmos L={Lmin } W = {4*XX*Wmin}
*PULL DOWN
Mn4 6
Mn3 3 5 5 6 1 nmos L={Lmin} W ={2*XX *Wmin}
Mn2 int3 node4n 1 1 nmos L={Lmin } W={2*XX *Wmin}
Mn1 3 node5n int3 1 nmos L={Lmin } W={2*XX*Wmin }
. ends
*********************************************************************
**************************************************************************
* and2
* Wmin, Lmin and MOS model must be declarate before
*
*drive strength--+
*inB --mos uscita-_ l
*inA - mos massa-_ + / /
*uscita - + / / /
*vdd - _ / / / 
```

226 Mn1 6 4 1 1 1 nmos W={2*XX *Wmin } L={Lmin }
227 Mn2 7 5 6 1 nmos W={2*XX*Wmin}}\quad\textrm{L}={\textrm{Lmin}
228 Mp1 7 4 2 2 pmos W={2*XX*Wmin} L={Lmin }
229 Mp2 7 5 2 2 pmos W={2*XX*Wmin} L={Lmin }
230
231 Mp3 3 7 2 2 pmos W={2*XX *Wmin} L={Lmin }
232 Mn3 3 7 1 1 nmos W={1*XX*Wmin}}\quad\textrm{L}={\textrm{Lmin}
233
234
235
236
237
238
239
240 *
241 *
242
243
244
245
246
247
248
249
250 *
251
252
253
254
255
256
257
270 *
271
272
273

```


```

224 * v v v v v v

```
224 * v v v v v v
225 .subckt AND2 SUB \(12 \begin{array}{lllll} & 3 & 4\end{array}\) XX=1
225 .subckt AND2 SUB \(12 \begin{array}{lllll} & 3 & 4\end{array}\) XX=1
```

Mn1 6
.ends
********************************************************************
********************************************************************

* and3
****************************************************************************
* Wmin, Lmin and MOS model must be declarate before
* 

*drive strength——+
*inC-_+_ /
*inB - -mos output—_+ / /
*inA - -mos ground-_+ / / /
*output _ _ / / / /
*vdd - + / / / / 1
*groung-_+

* v v v v v v v
.subckt AND3_SUB 1 2 3 4 5 6 XX=1
Mn1 7 4 4 1 1 n nmos W={3*XX*Wmin} L={Lmin}
Mn2 8 5 5 7 1 nmos W={3*XX*Wmin} L={Lmin}
Mn3 9 6 6 8 1 nmos W={3*XX*Wmin} L={Lmin}
Mp1 9 4 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mp2 9 5 5 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mp3 9 6 6 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mp4 3 9 9 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mn4 3 9 9 1 1 nmos W={1*XX *Wmin} L={Lmin}
. ends
***************************************************************************
* and4
****************************************************************************
* Wmin, Lmin and MOS model must be declarate before
* 

*drive strength——+
*inD-_+_/
*inC--+ / /

```
```

274
275
276
277
278
279
280 *
281
282
283
284
285
286
287
288
289
290

| 326 | Mp1 nodecn nodec node1 node1 pmos $\mathrm{W}=\{2 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| :---: | :---: |
| 327 | Mn1 nodecn nodec $00 \mathrm{nmos} \mathrm{W}=\{\mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 328 | . ends |
| 329 |  |
| 330 | ********************************************************************* |
| 331 |  |
| 332 | ********************************************************************* |
| 333 | * mux31 |
| 334 | **************************************************************************) |
| 335 | *drive strength |
| 336 | *in5-(nodeS1)-_+ |
| 337 | *in $4-($ nodeS 0$)-\ldots+1$ |
| 338 | *in3-(nodeD2)- / / |
| 339 | *in2 - (nodeD1)- / / / |
| 340 | *in1-(nodeD0 -+1 / / / |
| 341 | *output- +1 / / / |
| 342 |  |
| 343 | *ground- / / / / / / / |
| 344 | * |
| 345 |  |
| 346 | .subckt mux31_SUB 0 node1 nodeZ nodeA nodeB nodeC nodeD nodeE $\mathrm{XX}=1$ |
| 347 |  |
| 348 | Mp8 nodeU1 nodee nodeU2 node1 pmos $\mathrm{W}=\{3 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 349 | Mp3 nodea noded nodeU1 node1 pmos $\mathrm{W}=\{3 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 350 | Mp2 nodeeN nodee node1 node1 pmos $\mathrm{W}=\{2 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 351 | Mp1 nodedN noded node1 node1 pmos $\mathrm{W}=\{2 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 352 | Mp4 nodeb nodedN nodeU1 node1 pmos $\mathrm{W}=\{3 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 353 | Mn3 nodea nodedN nodeU1 0 nmos $\mathrm{W}=\{3 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 354 | Mn8 nodeU1 nodeeN nodeU2 0 nmos $\mathrm{W}=\{3 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 355 | Mn4 nodeb noded nodeU1 0 nmos $\mathrm{W}=\{3 * \mathrm{XX} * W \min \} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 356 |  |
| 357 | Mn7 nodez $1000 \mathrm{nmos} \mathrm{W}=\{\mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 358 | Mp7 nodez 1 node1 node1 pmos $\mathrm{W}=\{2 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 359 | Mn6 1 nodeU2 00 nmos $\mathrm{W}=\{\mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 360 |  |
| 361 | Mp6 1 nodeU2 node1 node1 pmos $\mathrm{W}=\{2 * \mathrm{XX} * W \min \} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 362 | Mp5 nodec nodeeN nodeU2 node1 pmos $\mathrm{W}=\{2 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 363 | Mn 2 nodeeN nodee $00 \mathrm{nmos} \mathrm{W}=\{\mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 364 |  |
| 365 | Mn1 nodedN noded $00 \mathrm{nmos} \mathrm{W}=\{\mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 366 | Mn5 nodec nodee nodeU2 0 nmos $\mathrm{W}=\{2 * \mathrm{XX} * W \min \} \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 367 |  |
| 368 | . ends |
| 369 | ********************************************************************** |
| 370 |  |
| 371 | ********************************************************************** |
| 372 | * mux41 |
| 373 |  |
| 374 | *drive strength |
| 375 | *in6-(nodeS1)-_+ |
| 376 | *in5 - nodeS 0$)-\ldots+1$ |
| 377 | *in4 - (nodeD3) - + / |



|  |  |  |
| :---: | :---: | :---: |
| 430 * |  |  |
| 432 | Mn1 node21 100 | $\mathrm{nmos} \mathrm{W}=\{\mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{2 * \mathrm{Lmin}\}$ |
| 433 | Mp1 node21 1 node1 node1 | pmos $\mathrm{W}=\{\mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{2 *$ Lmin $\}$ |
| 434 | Mn2 1 node21 00 | $\mathrm{nmos} \mathrm{W}=\{1 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 435 | Mp2 1 node21 node1 node1 | pmos $\mathrm{W}=\{2 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 436 |  |  |
| 437 | * notG |  |
| 438 | Mp5 nodeaN nodea node1 node1 | pmos $\mathrm{W}=\{2 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 439 | Mn5 nodeaN nodea 00 | $\mathrm{nmos} \mathrm{W}=\{\mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{\mathrm{Lmin}\}$ |
| 440 | . ends |  |
| 441 | ********************************************************************** |  |
| 442 |  |  |
| 443 | ********************************************************************* |  |
| 444 | * AO12 |  |
| 445 | ********************************************************************* |  |
| 446 | * Wmin, Lmin and MOS model must be declarate before |  |
| 447 |  |  |
| 448 | *drive strength——+ |  |
| 449 | *inC --mos out-gnd-_+ / |  |
|  | *inB - mos Vdd-gnd- + / / |  |
|  | *inA - mos Vdd-out- + / / |  |
| 452 | * uscita $-+1 / 1 /$ |  |
| 453 | *vdd $-+1 / 1 / 1$ |  |
| 454 | * massa $-+1 / 1111$ |  |
| 455 | * \| | | | | | | |  |
| 456 | * $\quad \mathrm{v}$ v v v v v v |  |
| 457 | .subckt AO12_SUB 1223456 XX=1 |  |
| 458 | Mn1 $94-481 n m o s h=\{2 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{\mathrm{Lmin}\}$ |  |
| 459 |  |  |
| 460 |  |  |
| 461 |  |  |
| 462 | Mp1 7 4 $222 \mathrm{pmos} \mathrm{W}=\{4 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{\mathrm{Lmin}\}$ |  |
| 463 | Mp2 $7 \mathrm{~T}_{5}^{5}$ |  |
| 464 |  |  |
| 465 |  |  |
| 466 | Mn4 $30 \begin{array}{llllll}9 & 1 & 1 \\ \mathrm{nmos} \\ \mathrm{W}=\{\mathrm{XX} * \mathrm{Wmin}\} \\ \mathrm{L}=\{\mathrm{Lmin}\}\end{array}$ |  |
| 467 |  |  |
| 468 | . ends |  |
| 469 |  |  |
| 470 |  |  |
| 471 | ********************************************************************* |  |
| 472 | * AO22 |  |
| 473 |  |  |
| 474 | * Wmin, Lmin and MOS model must be declarate before |  |
| 475 |  |  |
| 476 | * drive strength--+ |  |
| 477 | *inD --mos Vdd-gnd- + 1 |  |
| 478 | *inC - mos Vdd-out $-+1 /$ |  |
| 479 | *inB - mos out-gnd——+ / / / |  |
| 480 | *inA - mos out-out-_+ / / / |  |
| 481 | *uscita $-+1 / 1 / 1$ |  |

```
4 8 2
4 8 3
4 8 4
4 8 5
486 .subckt AO22_SUB 1 2 % 3 4 5 6 6 7 XX=1
487 * D G S B Name W L
4 8 8 * P U L L ~ D O W N ~
489 Mn1 11 4 9 9 1 nmos W={2*XX*Wmin} L={Lmin}
490 Mn2 9 5 1 1 nmos W={2*XX*Wmin} L={Lmin}
4 9 1
4 9 2
4 9 3
4 9 4
495
4 9 6
4 9 7
4 9 8
4 9 9
500
501
502
503
504
505
506
507
508
509
510
5 1 1
512*
513 *
514 *
515 *
5 1 6 * i n C ~ - ~ < m o s ~ V d d - g n d — — + ~ / ~ / ~
5 1 7 ~ * i n B ~ - — — m o s ~ V d d - m i d , m i d - ~ < ~ + ~ / ~ / ~
518 *
519 *
520 *
521
522
523
524
525
526
527
528
529
530
531
5
532
533
```



```
534 Mp3 8 6 2 2 pmos W={4*XX*Wmin} L={Lmin }
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557 *
558
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580 * AO33
5 8 1
582
583 *
584 *drive strength-_+
585 *inF -_mos out-gnd-_ + /
```

```
586 *inE - -mos out-mid,mid__lllllllllll
597 * D G S B Name W L
598 Mn1 11 4 12 1 1 nmos W={3*XX*Wmin} L={Lmin}
599 Mn2 12 5 13 1 nmos W={3*XX*Wmin} L={Lmin}
600 Mn3 13 6 1 1 1 nmos W={3*XX*Wmin} L={Lmin}
601 Mn4 11 7 7 14 1 1 nmos W={3*XX*Wmin} L={Lmin}
602 Mn5 14 8 15 15 1 nmos W={3*XX*Wmin} L={Lmin}
603 Mn6 15 9 1 1 nmos W={3*XX*Wmin} L={Lmin}
6 0 4
605 Mp1 10 4 4 2 2 2 pmos W={4*XX*Wmin} L={Lmin}
606 Mp2 10 5 2 2 pmos W={4*XX*Wmin} L={Lmin }
607 Mp3 10 6 2 2 2 pmos W={4*XX*Wmin} L={Lmin}
608 Mp4 11 7 10 2 pmos W={4*XX*Wmin} L={Lmin}
609 Mp5 11 8 10 2 pmos W={4*XX*Wmin} L={Lmin}
610 Mp6 11 9 10 2 pmos W={4*XX*Wmin} L={Lmin}
6 1 1
612 Mn7 3 111 1 1 nmos W={XX*Wmin} L={Lmin}
613 Mp7 3 11 2 2 pmos W={2*XX*Wmin } L={Lmin }
6 1 4
6 1 5
6 1 6
6 1 7
618 * AO112
6 1 9
6 2 0 ~ * ~ W m i n , ~ L m i n ~ a n d ~ M O S ~ m o d e l ~ m u s t ~ b e ~ d e c l a r a t e ~ b e f o r e
621 *
622 *drive strength-- +
6 2 3
6 2 4
625
6 2 6
6 2 7
6 2 8
6 2 9
6 3 0
631 * v v v v v v v v
632 .subckt AO112_SUB 1 2 3 4 5 6 7 XX=1
6 3 3
6 3 4
635 Mn1 10 4 111 1 nmos W={2*XX*Wmin} L={Lmin}
636 Mn2 11 5 1 1 1 nmos W={2*XX*Wmin} L={Lmin }
637 Mn3 10 6 1 1 1 nmos W={1*XX*Wmin} L={Lmin}
```

```
638 Mn4 10 7 1 1 nmos W={1*XX*Wmin} L={Lmin }
6 3 9
640 Mp1 8 4 2 2 pmos W={6*XX*Wmin} L={Lmin }
641 Mp2 8 5 2 2 pmos W={6*XX*Wmin} L={Lmin}
642 Mp3 9 6 8 8 2 pmos W={6*XX*Wmin} L={Lmin}
643 Mp4 10 7 7 9 2 pmos W={6*XX*Wmin} L={Lmin}
644
645 Mn5 3 3 10 1 1 1 nmos W={XX *Wmin} L={Lmin}
646 Mp5 3 10 2 2 pmos W={2*XX*Wmin} L={Lmin }
6 4 7 ~ . ~ e n d s
648 *
6 4 9
650 ***************************************************************************
651 * AO212
6 5 2
653 * Wmin, Lmin and MOS model must be declarate before
654 *
655 *drive strength--+
6 5 6
6 5 7
6 5 8
659
660 *
661 *
662
663
665
666 .subckt AO212 SUB 1 2 1 3 4 5 5 6 7 8 XX=1
667 * D G S B Name W L
668 Mn1 12 4 13 1 nmos W={2*XX*Wmin} L={Lmin}
669 Mn2 13 5 1 1 1 nmos W={2*XX*Wmin} L={Lmin }
670 Mn3 12 6 6 14 1 1 nmos W={2*XX*Wmin} L={Lmin}
671 Mn4 14 7 7 1 1 1 nmos W={2*XX*Wmin} L={Lmin}
672
6 7 3
6 7 4 ~ M
6 7 5 ~ N
676 Mp3 9 6 6 2 2 
6 7 7
678
6 7 9
680
6 8 1
6 8 2
6 8 3
6 8 4
6 8 5
687 * AO222
688***************************************************************************
6 8 9 ~ * ~ W m i n , ~ L m i n ~ a n d ~ M O S ~ m o d e l ~ m u s t ~ b e ~ d e c l a r a t e ~ b e f o r e
```

```
690 *
lllll
lllll
lllll
lllll
lllll
lllll
lllll
lllll
lllll
lllll
lllol
lllll
lllll
lllll
lllll
lllll
lllll
lllll
lllll
7 1 2
713 Mp1 12 4 11 2 pmos W={6*XX *Wmin} L={Lmin}
7 1 4
715
717 Mp5 10 8 2 2 2 pmos W={6*XX*Wmin} L={Lmin}
718
7 1 9
720 Mn7 3 12 1 1 nmos W={XX*Wmin} L={Lmin}
7 2 1
72
7 2 3
724
725
726
727
728
7 2 9
7 3 0
7 3 1
7 3 2
733
734
735
736
737
738
739
740
7 4 1
Mp2 12 5 111 2 pmos W={6*XX*Wmin} L={Lmin}
Mp6 10 9 2 2 pmos W={6*XX*Wmin } L={Lmin }
Mp7 3 12 2 2 pmos W={2*XX*Wmin } L={Lmin }
. ends
*************************************************************************
* DFPQ
*drive strength--+
*inB ---(like i/p)-nodeD-_+ /
*inA ---(like clk)-nodeA—_ l |
*output-M / / l
*vdd -_+
*ground-- +
* * | | v l 
.subckt DFPQ SUB 0 node1 nodeZ nodeCP noded XX=1
Mp41 noded nodeCP node21 node1 pmos W={2*XX*Wmin} L={Lmin}
Mn4 noded nodeCPn node21 0 nmos W={2*XX*Wmin} L={Lmin}
```

742

Mn1 node21 noded5 00
Mp2 noded5 node21 node1 node1
Mn2 noded5 node21 00
Mp6 nodez node51 node1 node1
Mn6 nodez node51 00
. ends

* DFPRQN


## *REACTION +

## *TRANSMISSION GATE

 *RESETMn5 noded5 nodeCP node51 $0 \quad$ nmos $\mathrm{W}=\{2 * \mathrm{XX} * W \min \} \quad \mathrm{L}=\{\mathrm{Lmin}\}$
Mp51 noded5 nodeCPn node51 node1 pmos $\mathrm{W}=\{2 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{L} \min \}$
Mn11 nodeCPn nodeCP $0 \quad 0 \quad \mathrm{nmos} \mathrm{W}=\{1 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\mathrm{Lmin}\}$
Mp11 nodeCPn nodeCP node1 node1 pmos $\mathrm{W}=\{2 * \mathrm{XX} * W \min \} \mathrm{L}=\{\operatorname{Lmin}\}$
Mp1 node21 noded5 node1 node1 pmos $\mathrm{W}=\{1 * \mathrm{XX} * W \min \} \quad \mathrm{L}=\{2 * \operatorname{Lmin}\}$

Mp7 node51 nodez node1 node1 pmos $\mathrm{W}=\{1 * \mathrm{XX} *$ Wmin $\} \mathrm{L}=\{\operatorname{Lmin}\}$
Mn7 node51 nodez $00 n \operatorname{nmos} \mathrm{~W}=\{1 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\operatorname{Lmin}\}$
$* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *$
$* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *$

. subckt DFPRQN_SUB 0 node1 nodez nodeCP noded nodeRN XX=1

Mn8 nodez node22 0 $0 \quad \mathrm{nmos} \mathrm{W}=\{2 * X X * W m i n\} \quad \mathrm{L}=\{\operatorname{Lmin}\}$
Mp8 nodez node22 node1 node1 pmos $\mathrm{W}=\{4 * \mathrm{XX} *$ Wmin $\} \quad \mathrm{L}=\{\operatorname{Lmin}\}$
Mn7 node22 nodez $0 \quad 0 \quad n \operatorname{mos} \mathrm{~W}=\{1 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{\mathrm{Lmin}\}$
Mp7 node22 nodez node1 node1 pmos $\mathrm{W}=\{1 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{\operatorname{Lmin}\}$
Mn2 1 node21 $0 \quad 0 \quad n \operatorname{mos} W=\{2 * X X * W m i n\} \quad L=\{L m i n\}$
Mp2 1 node21 node1 node1 pmos $\mathrm{W}=\{4 * X X * W m i n\} \quad \mathrm{L}=\{\mathrm{Lmin}\}$
Mn1 node21 $10 \quad 0 \quad$ nmos $\mathrm{W}=\{1 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{2 *$ Lmin $\}$ Mp1 node21 1 node1 node1 pmos $\mathrm{W}=\{1 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{2 * \operatorname{Lmin}\}$
pmos $\mathrm{W}=\{2 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{\mathrm{L} \min \}$ nmos $W=\{2 * X X * W \min \} \quad L=\{L m i n\}$ Mn5 2 nodeCP node22 $0 \quad \operatorname{mos} \mathrm{~W}=\{2 * \mathrm{XX} * \mathrm{Wmin}\} \mathrm{L}=\{\operatorname{Lmin}\}$ Mp51 2 nodeCPN node22 node1 pmos $\mathrm{W}=\{2 * \mathrm{XX} * W \min \} \mathrm{L}=\{\mathrm{Lmin}\}$
$\operatorname{nmos} \mathrm{W}=\{4 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{\mathrm{L} \min \}$
nmos $\mathrm{W}=\{4 * \mathrm{XX} * \mathrm{Wmin}\} \quad \mathrm{L}=\{\mathrm{L}$ min $\}$



## Publications and Presentations

[1] Antonio Mastrandrea, Francesco Menichelli, Mauro Olivieri, "A delay model allowing nano-CMOS standard cells statistical simulation at the logic level", PRIME-2011, 7th Conference on PhD Research in Microelectronics \& Electronics, 3-7 July, Madonna di Campiglio, Trento, Italy (BRONZE LEAF CERTIFICATE from the Scientific Committee).
[2] Olivieri, M., Menichelli, F., Mastrandrea A., Ramundo, F., Nenzi, P., "Contributions in evaluating the statistical impact of technology variations on delay and power dissipation of logic cells", ECMI 2010, 16-th European Conference on Mathematics for Industry, Wuppertal, Germany, July 26-30, 2010.
[3] Paolo Nenzi, Vittorio Delitala, Marco Garzuoli, Robert Larice, Antonio Mastrandrea, Mauro Olivieri, Stefano Perticaroli, Fabrizio Ramundo, Lionel SainteCluque, Ljiljana Trajkovic, Holger Vogt, Dietmar Warning,"Ngspice: an Open Platform for Modeling and Simulation from Device to Board Level" 8 Dec. 2010, California MOS-AK.
[4] Mauro Olivieri and Antonio Mastrandrea, "A new logic level delay modeling paradigm for nano-CMOS standard cells variation-aware simulation", Workshop on Variability modelling and mitigation techniques in current and future technologies, DATE 2012, Dresden, Germany - March 16, 2012.
[5] Mauro Olivieri and Antonio Mastrandrea, "Logic drivers: a logic level delay modeling paradigm for nano-CMOS standard cells statistical simulation", IEEE transactions on Very Large Scale Integration Systems.
[6] Mauro Olivieri and Antonio Mastrandrea, "A General Design Methodology for Synchronous Early-Completion-Prediction Adders in Nano-CMOS DSP Architectures", Hindawi's Independent Journals.
[7] Mauro Olivieri, Francesco Menichelli, Antonio Mastrandrea, Zia Abbas, "Chapter 7 - SPICE Simulations of Digital IC Blocks", Springer Book [publishing].
[8] Mauro Olivieri and Antonio Mastrandrea, "A new logic-level delay modeling paradigm for nano-CMOS standard cells variation-aware simulation", 44a Riunione annuale del Gruppo Italiano di Elettronica, Marina di Carrara, 20-22 Giugno 2012.
[9] Zia Abbas, Antonio Mastrandrea, Mauro Olivieri, "A voltage-based leakage current calculation scheme and its application to nano-scale MOSFET and FinFET standard-cell designs", IEEE Transactions on Very Large Scale Integration Systems.
[10] Usman Khalid, Antonio Mastrandrea, Mauro Olivieri, "Novel Approaches to Quantify Failure Probability due to Process Variations in Nano-scale CMOS logic ", Belgrade, Serbia 12-15 May 2014 [accepted for proceeding].

