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

List of Figures

List of Tables

Abstract

The present dissertation was developed within a European project, which envisages the development of standard cell in 45nm 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_{ox}) and the doping concentration (N_{Dp}, N_{Dn}), 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_{th}) 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.

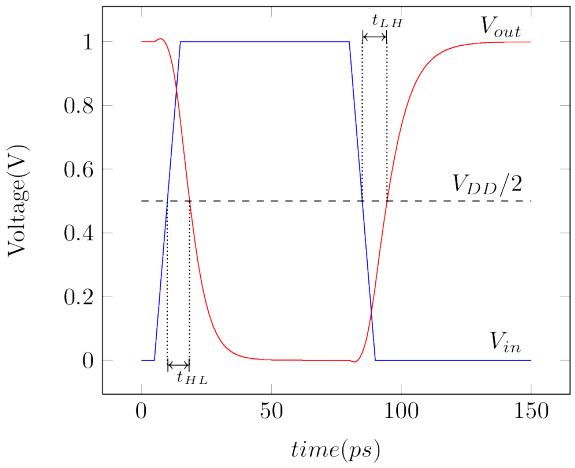

Figure 1.1.: Propagation delay definitions

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_{LH} and t_{HL} 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_{LH} and t_{HL}.

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 device-sizing 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]:

$$\tau_{P_{HL}} = \frac{C_{load}}{k_n \left(V_{DD} - V_{T,n}\right)} \left[\frac{2V_{T,n}}{V_{DD} - V_{T,n}} + \ln\left(\frac{4\left(V_{DD} - V_{T,n}\right)}{V_{DD}} - 1\right)\right] \tag{1.1}$$

$$\tau_{P_{LH}} = \frac{C_{load}}{k_p \left(V_{DD} - |V_{T,p}|\right)} \left[\frac{2\,|V_{T,p}|}{V_{DD} - |V_{T,p}|} + \ln\left(\frac{4\left(V_{DD} - |V_{T,p}|\right)}{V_{DD}} - 1\right)\right] \tag{1.2}$$

$$Where \qquad k_n = \mu_n \cdot C_{ox}\left(\frac{W_n}{L_n}\right) \qquad \& \qquad k_p = \mu_p \cdot C_{ox}\left(\frac{W_p}{L_p}\right)$$

C_{load} is a Capacitive load applied to the output of the inverter;

V_{DD} is a Drain voltage applied to PMOS Drain Terminal;

V_T is a Threshold voltage for a transistor;

C_{ox} is Gate-Oxide capacitance

μ_n,μ_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 $\frac{Vdd}{2}$, until the output transition in the opposing direction through $\frac{Vdd}{2}$. 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

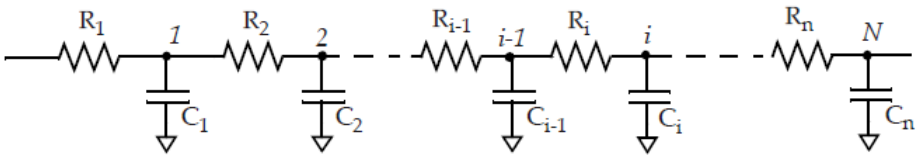

Figure 1.2.: An RC-transmission line model

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:

$$V_{out}(t) = V_{DD}\left(1 - e^{-\frac{t}{RC}}\right) \tag{1.3}$$

The time at 50% of V_{out} is propagation delay Low-High ($\tau_{p_{LH}}$):

$$V_{out/50\%}(t^*) \sim \frac{V_{DD}}{2} = V_{DD}\left(1 - e^{-\frac{t^*}{RC}}\right) \tag{1.4}$$

$$\ln(1 - \frac{1}{2}) = \ln\left(e^{-\frac{t^*}{RC}}\right)$$

$$t^* = \tau_{p_{LH}} = \ln(2) \cdot RC \sim 0.69 \cdot RC$$

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:

$$\tau_{D_N} = \sum_{j=1}^{N} C_j \sum_{i=1}^{j} R_i \tag{1.5}$$

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 circuit-design 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:

$$d_\tau = d \cdot \tau \tag{1.6}$$

Where d represents the collection of all other effects lumped into a singular quantity and τ 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:

$$d = f + p \tag{1.7}$$

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:

$$f = g \cdot h \tag{1.8}$$

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

$$h = \frac{C_{out}}{C_{in}} \tag{1.9}$$

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 W=2 and Pmos with 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 semi-custom 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:

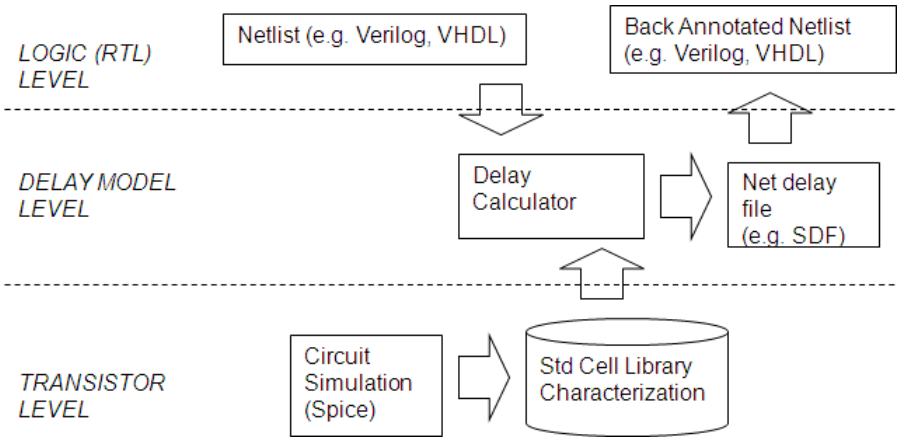

Figure 1.3.: typical tool-chain

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 large-scale 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

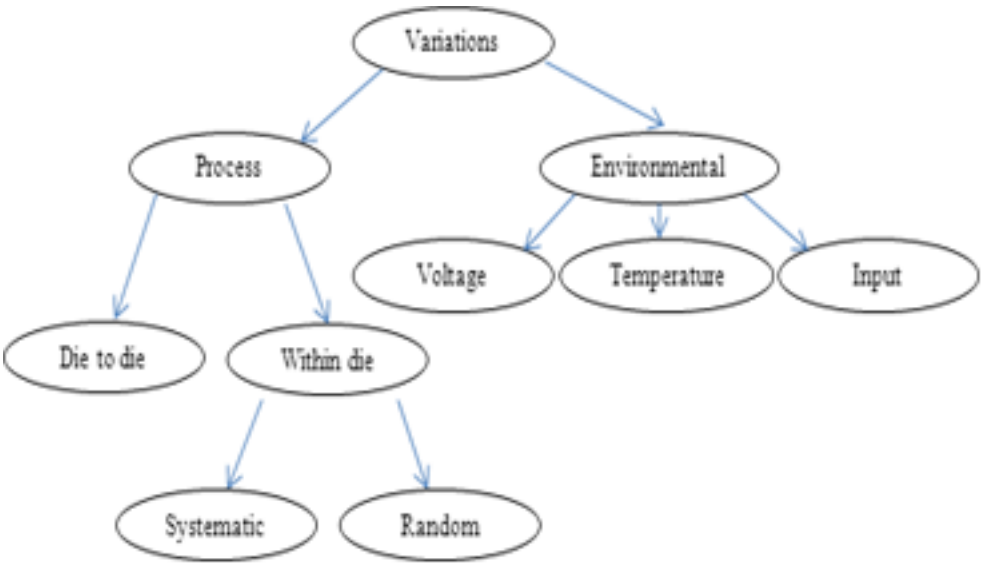

Figure 2.1.: Process and Environmental variations

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

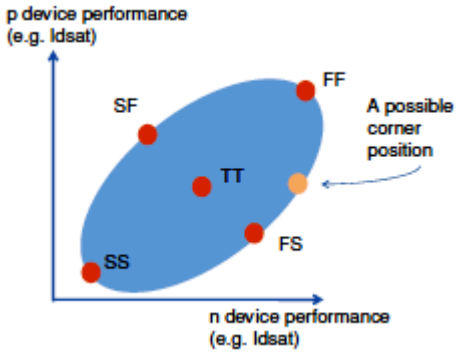

Figure 2.2.: Corner models

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 σ and μ 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

$$t_d = \frac{C_{load} \cdot V_{DD}}{I} \tag{2.1}$$

To understand, we have used Equation 2.2 shows the ideal I-V equation for the MOSFET in saturation region, where µ 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_{GS} is the gate to source bias voltage.

$$I_D = \frac{1}{2} \cdot \mu \cdot C_{ox} \cdot \frac{W}{L} \cdot (V_{GS} - V_{DD})^2 \tag{2.2}$$

$$t_d = \frac{C_{load} \cdot V_{DD}}{\frac{1}{2} \cdot \mu \cdot C_{ox} \cdot \frac{W}{L} \cdot (V_{GS} - V_{DD})^2} \tag{2.3}$$

Although these equation is ideal and neglect several details as far as modern nano-scale 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 (μ)	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% σ/μ, consistent with the ITRS projections .

2.4.2. Threshold Voltage (V_T)

Threshold voltage of the MOSFET is the gate to source bias (V_{GS}) 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

$$V_{T_0} = V_{FB} + 2 \cdot \varphi_{F_p} + \gamma \cdot \sqrt{2 \cdot \varphi_{F_p} + V_{SB}} \tag{2.4}$$

Where VFB is the flat band voltage, ϕF_P is the Fermi potential of the substrate. V_{FB}, ϕ and F_P are dependent on the doping concentration only while γ 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 (X_j), 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 3σ 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 σ/μ 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.

$$C_{ox} = \frac{\varepsilon_{ox}}{t_{ox}} \tag{2.5}$$

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 1nm (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

$$\mu_{n,p} = \frac{q\tau_c}{2m_{n,p}} \tag{2.6}$$

Where q is the electronic charge,τ_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 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

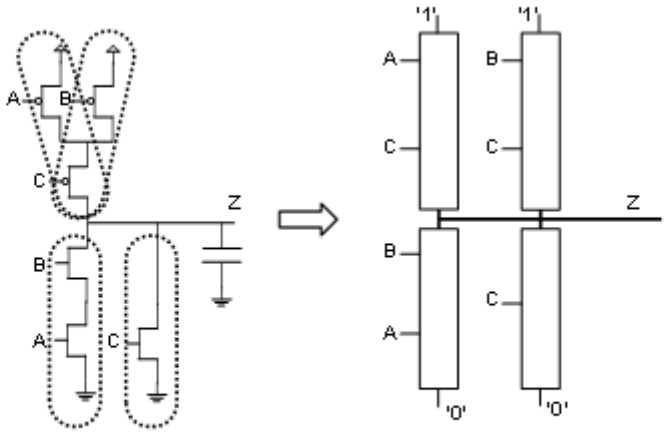

Figure 3.1.: Four current drivers in a cell and associated logic drivers.

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 N+1 propagation delay values, one per each input-output signal pair;
4. A logic behavior expressing the relation between the set of 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'                                    1
        then return source;                                    2
elsif gate = '0' or gate = 'L' or gate = 'U'                   3
        then return 'Z';                                       4
elsif gate = 'X' or gate = 'W'                                 5
        then return 'X';                                       6
elsif gate = 'Z'                                               7
        then return '-';                                       8
```

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 $i_{drive}(t)$ designates

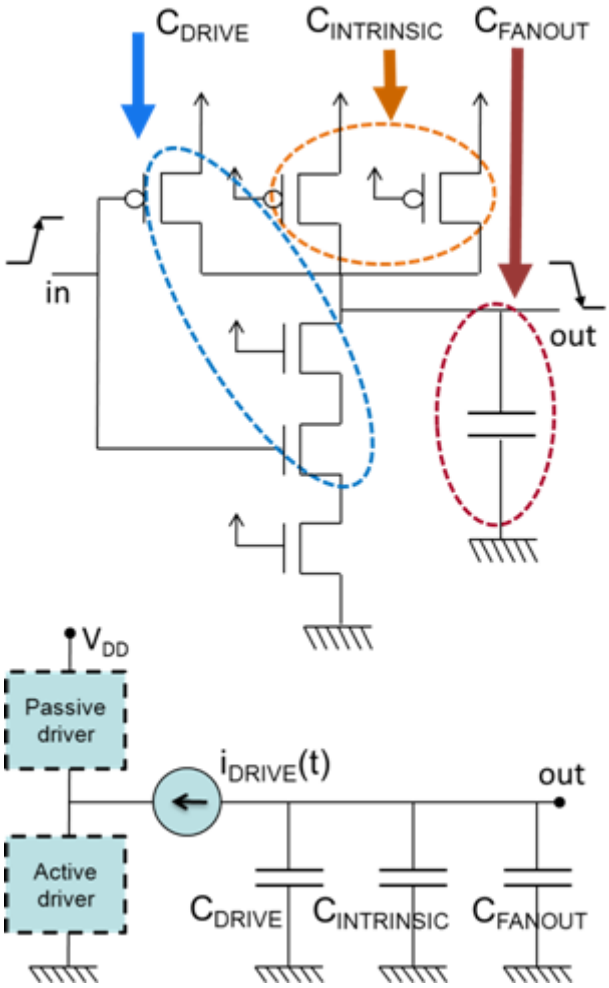

Figure 3.2.: Equivalent circuit for the propagation delay model.

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_{FANOUT} accounts for capacitive load connected to the output, i.e., basically fan-out capacitance. $C_{INTRINSIC}$ 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_{DRIVE} 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_{DRIVE} considers the voltage feed-through phenomenon and its effect on the total rising/falling propagation delay of the drain voltage, so that C_{DRIVE} partially corresponds to a formal quantity rather than a measurable physical capacitance. According to Figure 3.2 we have the following:

$$i_{drive}(t) = (C_{INTRINSIC} + C_{FANOUT} + C_{DRIVE}) \cdot \frac{\partial V_{out}(t)}{\partial t} \tag{3.1}$$

Hence by integrating and applying the mean value theorem, we obtain that the propagation delay T_{pd} corresponding to an output voltage swing V_S is

$$T_{pd} = \frac{V_S \cdot (C_{INTRINSIC} + C_{FANOUT} + C_{DRIVE})}{I_{AVG}} \tag{3.2}$$

where I_{AVG} is the average value of $i_{drive}(t)$ in the time interval [0, T_{pd}]. V_S is usually $V_{DD}/2$ for standard cell library propagation delay characterization.

We can expand the intrinsic capacitance value as follows:

$$C_{INTRINSIC} = C_{I_{min}} \cdot \sum_j (X \cdot W_I(j) - a) \tag{3.3}$$

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_{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:

$$C_{DRIVE} = C_{D_{min}} \cdot \sum_j (X \cdot W_D(j) - a) \tag{3.4}$$

where $W_D(j)$ is a weight expressing the width of every transistor j contributing to the drive capacitance and C_{Dmin} 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_{FANOUT} and a reference minimal gate capacitance C_{gmin} in the given technology. By defining the quantities:

$$\tau_D = \frac{V_S \cdot C_{D_{min}}}{I_{avg}},$$

$$\tau_I = \frac{V_S \cdot C_{I_{min}}}{I_{avg}},$$

$$\tau_O = \frac{V_S \cdot C_{O_{min}}}{I_{avg}}$$

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

$$T_{pd} = \tau_D \cdot \sum_j (X \cdot W_D(j) - a) + \tau_I \cdot \sum_j (X \cdot W_I(j) - a) + \tau_O \cdot \frac{C_{FANOUT}}{C_{g_{min}}} \quad (3.5)$$

The technology-dependent timing parameters τ_D, τ_I and τ_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 τ_D, τ_I and τ_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 τ_D and τ_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:

$$\tau_D\,(C_{FANOUT}; X; t_{sleew_{in}}) \cdot \sum_j (X \cdot W_D(j) - a) + \tau_O\,(C_{FANOUT}; X; t_{sleew_{in}}) \cdot \frac{C_{FANOUT}}{C_{g_{min}}}$$

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

(3.6)

where the sum is constant for the given driver pair and for the chosen input pin. The parameter C_{gmin} can be chosen as the input capacitance of a minimal inverter under nominal conditions. By increasing C_{FANOUT} of a quantity δC small enough to consider τ_D and τ_O as unaffected, we measure a propagation delay increase given by the following:

$$\delta T_{pd} = \tau_O \left(C_{FANOUT}; X; t_{sleew_{in}} \right) \cdot \frac{\delta C}{C_{min}} \tag{3.7}$$

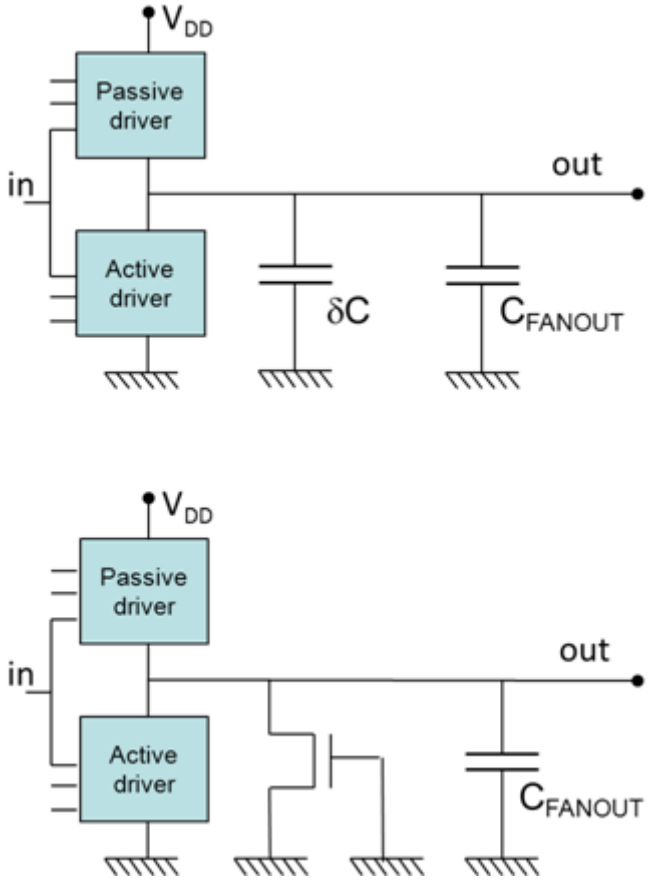

Figure 3.3.: Simulation setup for model parameter calibration

Therefore, we derive the value of τ_O and consequently the value of τ_D by substitution.

The simulation setup for characterizing τ_I is shown in Figure 3.3 (bottom). The measured propagation delay of the active/passive driver pair with an additional N-type (P-type) transistor connected between the output node and ground (V_{dd}) is modeled by the following:

$$T_{pd} = \tau_D \left(C_{FANOUT}; X; t_{sleew_{in}}\right) \cdot \sum_j \left(X \cdot W_D(j) - a\right)$$

$$+\tau_{I\left(C_{FANOUT}; X; t_{sleew_{in}}\right)} \cdot \sum_j \left(X \cdot W_I(j) - a\right)$$

$$+\tau_O\left(C_{FANOUT}; X; t_{sleew_{in}}\right) \cdot \frac{C_{FANOUT}}{C_{g_{min}}} \tag{3.8}$$

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 τ_D, τ_I and τ_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

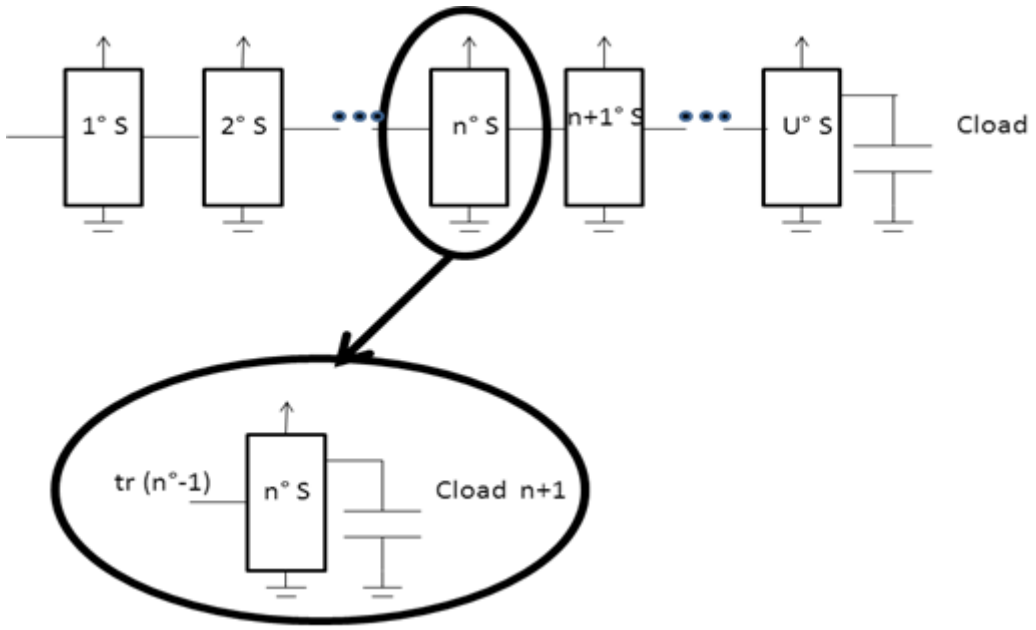

Figure 3.4.: multistage path

yet accurate model. To accurately consider the effect, we chose to characterize the timing functions τ_D, τ_I and τ_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-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 τ_I and τ_O values and scarcely to the initial shape of it–mostly affected by the parasitics modeled by τ_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 τ_I and τ_O to the propagation delay value

$$t_{slew_{out}} = \alpha + \beta \cdot t_{I,O} + \gamma \cdot t_{slew_{IN}} \tag{3.9}$$

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

$$t_{I,O} = \left(\tau_I \cdot \sum_j (X \cdot W_I(j) - a) + \tau_O \cdot \frac{C_{FANOUT}}{C_{g_{min}}} \right)$$

β and γ have practically the same two values for all the cells in the given technology, while α 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.

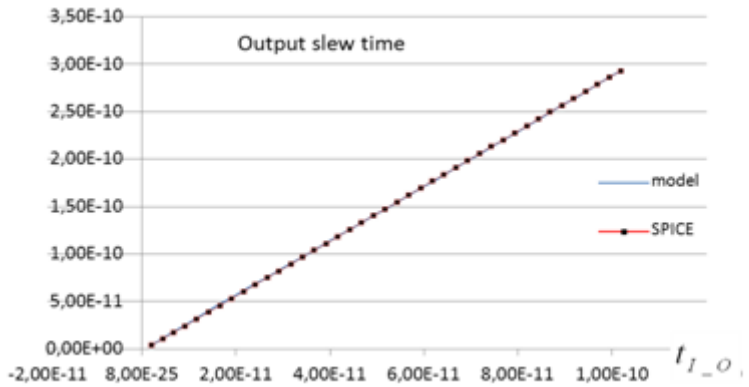

Figure 3.5.: Behavior of output slew time vs the quantity t_{I_O} .

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

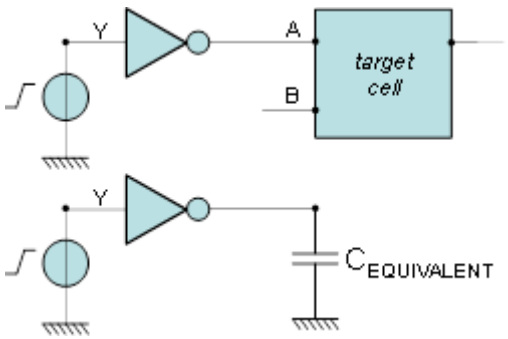

Figure 3.6.: Input pin capacitance characterization setup.

(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_{IN} 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

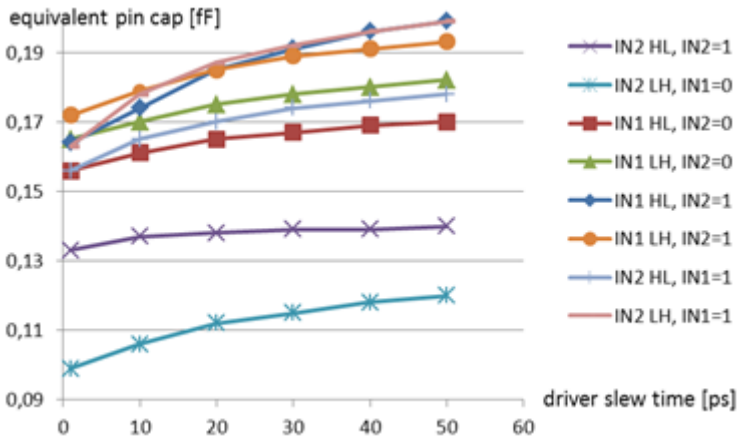

Figure 3.7.: SPICE characterization of input pin capacitance (two-input AND cell) with respect to input slew of the driver cell and to different input logic patterns of the target cell.

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 factor	driver slew (ps)	pin cap. HL (fF)	pin cap. LH (fF)	pin IN1 logic value	pin IN2 logic 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 factor	driver slew (ps)	pin cap. HL (fF)	pin cap. LH (fF)	pin IN1 logic value	pin IN2 logic 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 τ_D, τ_I and τ_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 τ function values and capacitance function values associated with the cells. If the same variations are applied to τ 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 τ 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 τ_D, τ_I and τ_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 τ_D, τ_I and τ_O functions is significantly affected by a technology variation (Figure 3.8). The vertical shift results to be relatively

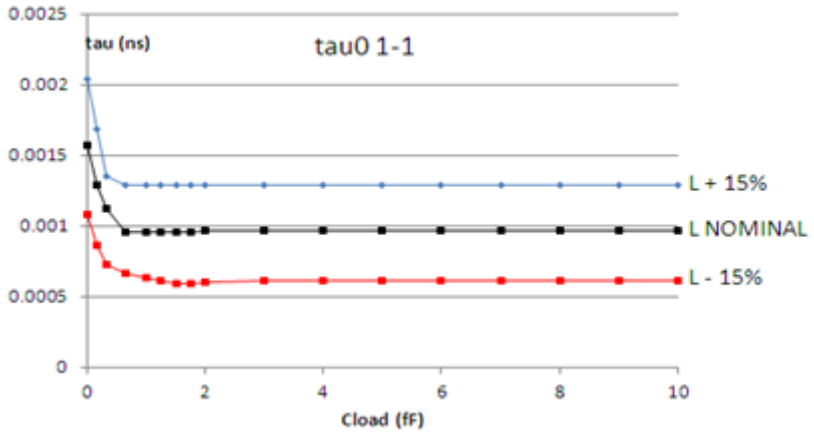

Figure 3.8.: Behavior of τo as affected by L (transistor drawn length) variation. Other technology variations have a similar effect.

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:

$$\tau_D = \tau_{D_{nominal}} + \Delta\tau_D$$

$$\tau_I = \tau_{I_{nominal}} + \Delta\tau_I$$

$$\tau_O = \tau_{O_{nominal}} + \Delta\tau_O$$

$$C_{IN} = C_{IN_{nominal}} (1 + \Delta C_{IN})$$

where the shift values τ_D, τ_I, τ_O and C_{IN} 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_{ox}, and N_{dep}, with 3σ 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 (τ variation vector) of 10000 shift values τ_D, τ_I and τ_O , referring to input slew times of 10 and 50 ps, for a total of 60 000 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_{IN} , 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 τ 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:

$$\tau_D = \tau_{D_{nominal}} + \Delta\tau_{D_{global}} + \Delta\tau_{D_{local}}$$

$$\tau_I = \tau_{I_{nominal}} + \Delta\tau_{I_{global}} + \Delta\tau_{I_{local}}$$

$$\tau_O = \tau_{O_{nominal}} + \Delta\tau_{O_{global}} + \Delta\tau_{O_{local}}$$

$$C_{IN} = C_{IN_{nominal}} \left(1 + \Delta C_{IN_{global}} + \Delta C_{IN_{local}}\right)$$

The terms $\Delta\tau_{\mathrm{D_{global}}}$, $\Delta\tau_{\mathrm{I_{global}}}$, $\Delta\tau_{\mathrm{O_{global}}}$ and $C_{\mathrm{IN_{global}}}$ are global, design-independent process variations and can be characterized as shown above. The terms $\Delta\tau_{\mathrm{D_{local}}}$, $\Delta\tau_{\mathrm{I_{local}}}$, $\Delta\tau_{\mathrm{O_{local}}}$ and $C_{\mathrm{IN_{local}}}$ 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 $\Delta\tau_{\mathrm{D_{local}}}$, $\Delta\tau_{\mathrm{I_{local}}}$, $\Delta\tau_{\mathrm{O_{local}}}$ and $C_{\mathrm{IN_{local}}}$

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 distance-dependent 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 τ_D, τ_I, τ_O and C_{IN} 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 τ functions and C_{IN} 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

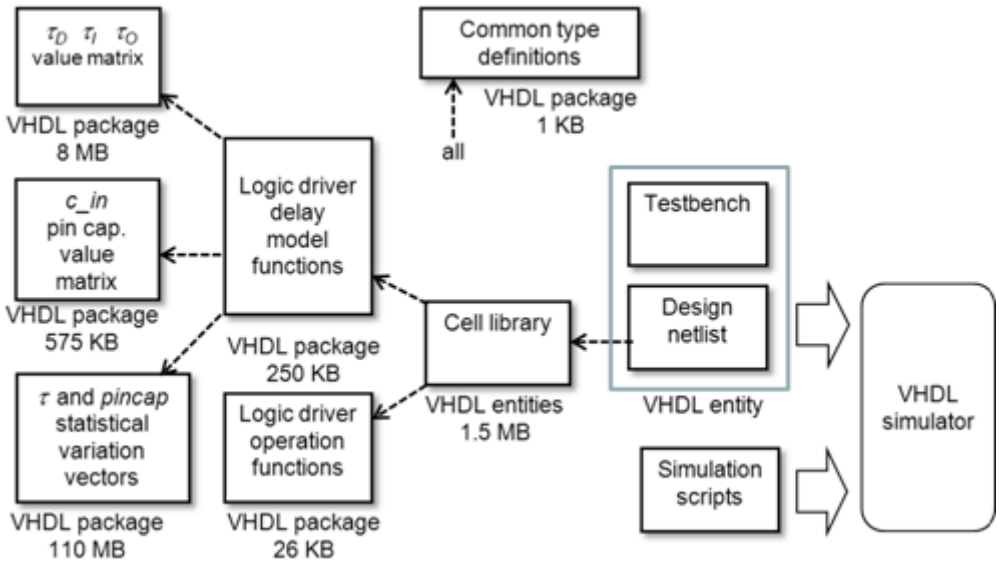

Figure 3.9.: Database structure for the logic-driver-based timing simulation environment. Arrows indicate dependencies.

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 τ_D, τ_I and τ_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 (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

```
------------------------------------------------------------------------
entity nand2_dut is
        generic (X:real:=1.0); --resize factor
        port (A, B: inout logic_drive_logic;
        Z: inout logic_drive_logic);
end nand2_dut;
------------------------------------------------------------------------


------------------------------------------------------------------------
architecture abstract of nand2_dut is
  -- here: internal signal declarations,
  -- transistor size definitions, etc.
begin
  -- delay computation section ------------------------
    -- delay values computed for each
        -- possible logic driver activation
        -- example:
        time_ld1_A_on <= ... -- call of tau functions
  -- logic operation section --------------------------
        -- all logic drivers in cell
        -- example:
        ld1: Nmos_2drive(Z, A, B, time_ld1_A_on, ..);
  -- input capacitance computation section ------------
        A.pincap <= ... -- call of c_in functions
        B.pincap <= ... -- call of c_in functions
end abstract;
------------------------------------------------------------------------
```

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 C1 reads the total capacitance on signal Z as a pair of reference values {3 and 6 fF} resulting as the sum of the capacitances

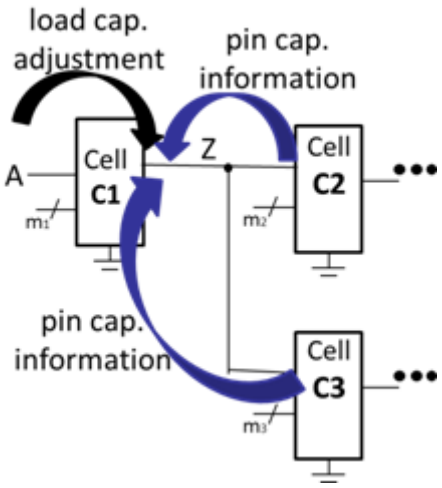

Figure 3.11.: Implementation of the input pin capacitance simulation model.

connected to Z. According to the actual input slew time 30 ps on the switching node A, cell C1 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%.

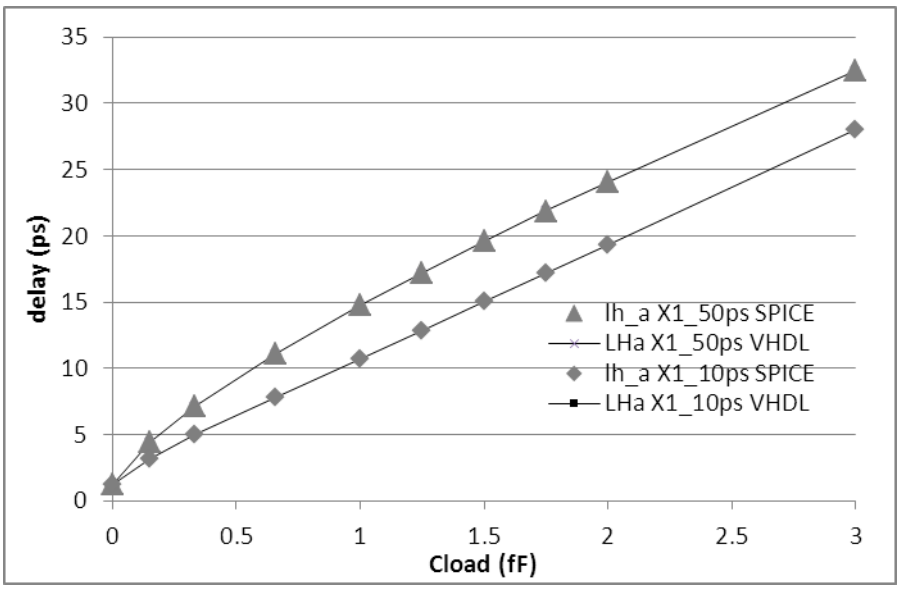

Figure 4.1.: VHDL vs SPICE t_{LH} NOT cell. Input slew time 10ps and 50ps.

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	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	0.02	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	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	0.98	-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_{DD}).

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

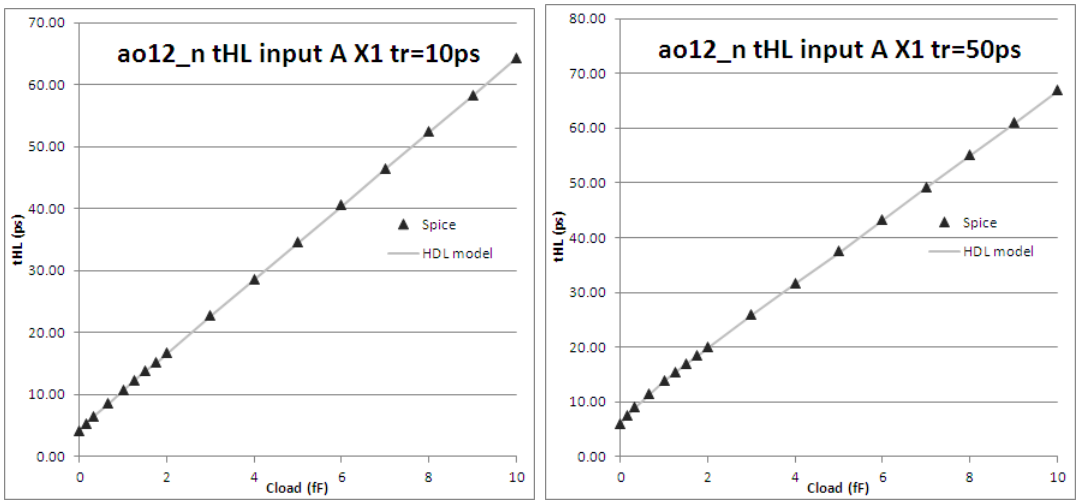

Figure 4.2.: ao12_n input A. Differente slew time (left 10ps,rigth 50ps)

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_{DD}).

4.2.7. ao32_n

The ao32_n gate is another cell of standar library. It have also 5 inputs. The pull-down 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_{DD}).

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

ao12_n	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	-0.6	-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_{DD}).

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

ao22_n	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	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 V_{DD}).

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

ao31_n	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	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 pull-down 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 V_{DD}).

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

ao32_n	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	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
9	-0.8	-0.45	-0.5	-0.30	-1.8	-0.20	-0.2	-0.03
10	-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 pull-down 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_{DD}).

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

ao33_n	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	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 τ_D, τ_I and τ_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_{load}.

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

ao112_n	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	-0.1	-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	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	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	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	-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 transition cases tested	# of different sizes tested	# of different loads tested	# of different input 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)	8	17	6

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

3 not	X=1, t_{slew}=10ps	
C_{load}(fF)	t_{LH}(ps)	t_{HL}(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	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	-0.4	-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	X=1, t_{slew}=10ps	
C_{load}(fF)	t_{LH}(ps)	t_{HL}(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	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	-0.1	-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)

7 not	X=1, t_{slew}=10ps	
C_{load}(fF)	t_{LH}(ps)	t_{HL}(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	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	0.0	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)

9 not	X=1, t_{slew}=10ps	
C_{load}(fF)	t_{LH}(ps)	t_{HL}(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	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	-0.1	-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 nand2 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	X=1, t_{slew}=10ps			
input	A		B	
C_{load}(fF)	t_{LH}(ps)	t_{HL}(ps)	t_{LH}(ps)	t_{HL}(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	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	-0.3	-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 nand2 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	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	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 nand2 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	X=1, t_{slew}=10ps			
input	A		B	
C_{load}(fF)	t_{LH}(ps)	t_{HL}(ps)	t_{LH}(ps)	t_{HL}(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 nand2 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 nand2 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	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	0.3	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 nand2 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	X=1, t_{slew}=10ps			
input	A		B	
C_{load}(fF)	t_{LH}(ps)	t_{HL}(ps)	t_{LH}(ps)	t_{HL}(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 nand2 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	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	-0.9	-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	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	-0.2	-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	X=1, t_{slew}=10ps			
input	A		B	
C_{load}(fF)	t_{LH}(ps)	t_{HL}(ps)	t_{LH}(ps)	t_{HL}(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		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	-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	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	-0.5	-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	
	x1	x10	x1	x10	x1	x10	x1	x10	x1	x10	x1	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					
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.

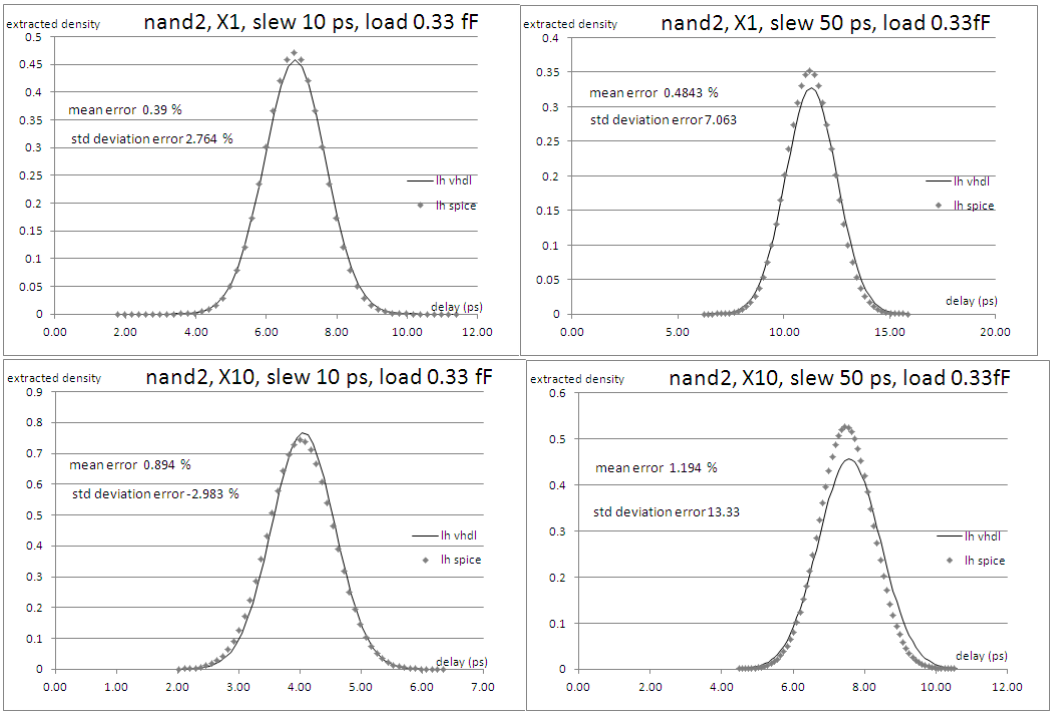

Figure 5.1.: Statistical analysis of single-stage: nand2 gate gaussian

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					
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 %	-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 count	Transistor count	Crit. Path. Delay		Crit. Path. Delay	
			mean value (SPICE)	variance (SPICE)	mean value (proposed model)	variance (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 10^3 MC iterations (SPICE)	Execution Time for 10^3 MC iterations (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 path	total mos	total library cell	simulation time (s)	
				hspice	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.

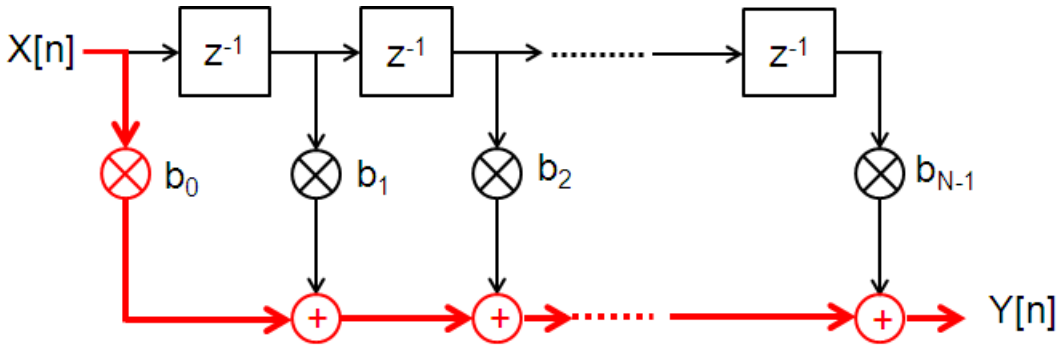

Figure 5.2.: Critical path through Execute Stage of FIR filter

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× speedup over SPICE, running on the same machine, even if a speed-optimized implementation is definitely not addressed yet.

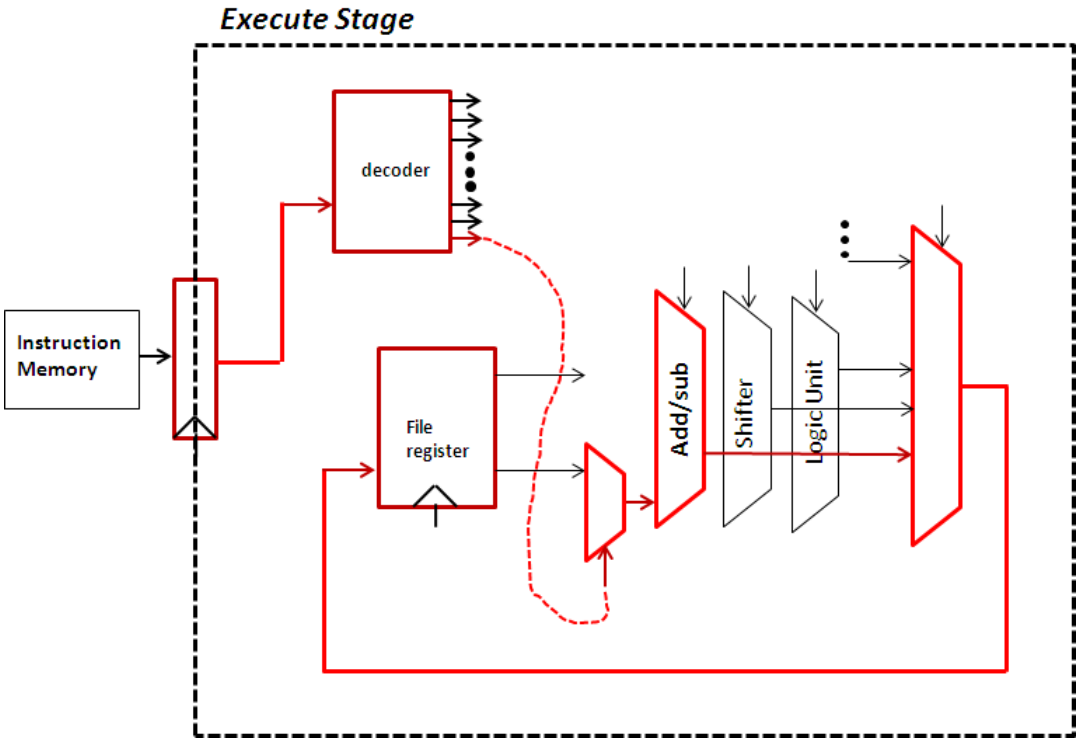

Figure 5.3.: Critical path through Execute Stage of MIPS processor

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 40th 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, 2007. 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 compact-modeling 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 & Test in Europe Conference & 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 (vl-adder): 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-completion-detecting alu for a 1 ghz 64 b datapath. In *Solid-State Circuits Conference, 1997. Digest of Technical Papers. 43rd 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 43rd 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 Proceedings-Circuits, 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 0.5μ

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):1194–1198, 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 595–598. 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):1191–1202, 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-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):772–774, 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):636–650, 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-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 65nm 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-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

```
1  -- ************** MODERN abstract model ************
2  -- cell: NAND2
3  -- ************************************************
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7
8  library modern2010;
9  use modern2010.common_type.all;
10 use modern2010.logical_drive_primitives.all;
11 use modern2010.logical_drive_delay_basic.all;
12
13 ------------------------------------------------
14 -- generic parameter list
15 -- tech = technology record, set of technology dependent parameters.
16 -- c_load = total fan-out load cap inout fF;
17 -- X = cell drive strength (resize factor with respect to minimal \
18     size X1)
19 ------------------------------------------------
20 entity nand2_dut is
21     generic ( X:real:=1.0;
22     --Z_load: load_net_record;
23     tech: technology_record);
24     port (
```

```
25        nodeA, nodeB: inout logical_drive_logic \
26         := ('Z', 66 fs, (0.0,0.0));
27        nodeZ: inout logical_drive_logic \
28         := ('Z', 66 fs, (0.0,0.0)) );
29 end nand2_dut;
30 ------------------------------------------------
31
32 ------------------------------------------------
33 architecture abstract of nand2_dut is
34 -- driver size factor (naming: w_drivenumber)
35 constant w_ld1_nodeA: real := 2.0;
36 constant w_ld1_nodeB: real := 2.0;
37 constant w_ld2_nodeA: real := 2.0;
38 constant w_ld3_nodeB: real := 2.0;
39
40 -- timing virtual signals
41 signal t_ld1_nodeA_on: timing_record:=(0.0 ps, 0.0 ps);
42 signal t_ld1_nodeB_on: timing_record:=(0.0 ps, 0.0 ps);
43 signal t_ld2_nodeA_on: timing_record:=(0.0 ps, 0.0 ps);
44 signal t_ld3_nodeB_on: timing_record:=(0.0 ps, 0.0 ps);
45 signal t_ld1_off: time := 1 ps;
46 signal t_ld2_off: time := 1 ps;
47 signal t_ld3_off: time := 1 ps;
48
49 -- vdd, gnd
50 signal node1 : logical_drive_logic := ('Z', 66 fs, (0.0,0.0)) ;
51 signal node0 : logical_drive_logic := ('Z', 66 fs, (0.0,0.0)) ;
52
53 -- input and output auxiliary nodes
54 signal xnodeZ : logical_drive_logic := ('Z', 66 fs, (0.0,0.0)) ;
55 signal xnodeA : logical_drive_logic := ('Z', 66 fs, (0.0,0.0)) ;
56 signal xnodeB : logical_drive_logic := ('Z', 66 fs, (0.0,0.0)) ;
57
58 -- regular internal nodes
59 signal node4 : logical_drive_logic := ('Z', 66 fs, (0.0,0.0)) ;
60
61 begin
62 -- signal setup ----------------------------------
63     node1 <= ('1', 0 ps, (0.0,0.0));
64     node0 <= ('0', 0 ps, (0.0,0.0));
65     xnodeA <= nodeA; -- this is not useless
66     xnodeB <= nodeB; -- this is not useless
67
68 -- delay computation section ---------------------
69   t_ld1_off <= T_n_zeta(1, tech);
70   t_ld2_off <= T_p_zeta(1, tech);
71   t_ld3_off <= T_p_zeta(1, tech);
72   t_ld1_nodeA_on <= nmos_delay(
73         W_on => w_ld1_nodeA ,  -- active device width
74         W_dg => w_ld1_nodeA + w_ld2_nodeA, --total miller width
75         n_dg => 1.0,  -- N device count related to W_dg
76         p_dg => 1.0,  -- P device count related to W_dg
```

```
77          W_d => w_ld3_nodeB, --total switching device width
78          n_d => 0.0,    -- N device count related to W_d
79          p_d => 1.0,    -- P device count related to W_d
80          --W_g => Z_load_n,  -- FO capacitance
81        W_g => nodeZ.pincap,  -- FO capacitance
82          stack_size_on => 2,  --size of stacked structure
83          stack_size_off => 1,
84          pos_on => 2,  -- position in stacked structure
85          pos_off => 1,
86          t_r => nodeA.slew,  -- active device input slew time
87          X => X,        -- cell resize factor vs minimum size
88          tech => tech);            -- technology record
89      t_ld1_nodeB_on <= nmos_delay(
90          W_on => w_ld1_nodeB,
91          W_dg => w_ld3_nodeB + w_ld1_nodeB+ w_ld1_nodeB,
92          n_dg => 2.0,
93          p_dg => 1.0,
94          W_d => w_ld2_nodeA,
95          n_d => 0.0,
96          p_d => 1.0,
97          --W_g => Z_load_n,
98        W_g => nodeZ.pincap,
99          stack_size_on => 2,
100         stack_size_off => 1,
101         pos_on => 1,
102         pos_off => 1,
103         t_r => nodeB.slew,
104         X => X,
105         tech => tech);
106     t_ld2_nodeA_on <= pmos_delay(
107         W_on => w_ld2_nodeA,
108         W_dg => w_ld2_nodeA + w_ld1_nodeA,
109         n_dg => 1.0,
110         p_dg => 1.0,
111         W_d => w_ld3_nodeB,
112         n_d => 0.0,
113         p_d => 1.0,
114         --W_g => Z_load_p,
115       W_g => nodeZ.pincap,
116         stack_size_on => 1,
117         stack_size_off => 2,
118         pos_on => 1,
119         pos_off => 2,
120         t_r => nodeA.slew,
121         X => X,
122         tech => tech);
123     t_ld3_nodeB_on <= pmos_delay(
124         W_on => w_ld3_nodeB,
125         W_dg => w_ld3_nodeB + w_ld1_nodeB+ w_ld1_nodeB,
126         n_dg => 2.0,
127         p_dg => 1.0,
128         W_d => w_ld2_nodeA,
```

```
129         n_d => 0.0,
130         p_d => 1.0,
131         --W_g => Z_load_p,
132     W_g => nodeZ.pincap,
133         stack_size_on => 1,
134         stack_size_off => 2,
135         pos_on => 1,
136         pos_off => 1,
137         t_r => nodeB.slew,
138         X => X,
139         tech => tech);
140 -- pincap value
141   nodeA.pincap <=
142   c_in_2(
143     X => X,
144     t_r => nodeA.slew,
145     stack_size_on => 2,
146         stack_size_off => 1,
147         pos_on => 1,
148         pos_off => 1,
149     type_name => NAND2_cell,
150     IN1 => nodeA,
151     IN2 => nodeB
152      );
153   nodeB.pincap <=
154   c_in_2(
155     X => X,
156     t_r => nodeB.slew,
157     stack_size_on => 2,
158         stack_size_off => 1,
159         pos_on => 2,
160         pos_off => 1,
161     type_name => NAND2_cell,
162     IN1 => nodeB,
163     IN2 => nodeA
164      );
165
166 -- logical drivers section -------------------
167   --Mn1 Mn2
168   ld1: nmos_2drive(xnodez, xnodeA, xnodeB, node0, t_ld1_nodeA_on, \
169     t_ld1_nodeB_on, t_ld1_off);
170   --Mp1
171   ld2: pmos_drive(xnodeZ, xnodeA, node1, t_ld2_nodeA_on, t_ld2_off);
172   --Mp2
173   ld3: pmos_drive(xnodeZ, xnodeB, node1, t_ld3_nodeB_on, t_ld3_off);
174   nodeZ <= xnodeZ;
175 end abstract;
176 ------------------------------------------------
```

A.2. Example: NAND2 behavioral at logic level

```
1  -- ************** MODERN abstract model ************
2  -- cell: NAND2 logic reference
3  -- ************************************************
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7
8  ------------------------------------------------
9  -- generic parameter list
10 -- tpd_min = intrinsic delay of minimal inverter
11 -- tpd_FO4 = total fan-out-4 delay of minimal inverter;
12 -- W_FO_value = actual cell fan-out expressed as
13 --       external load cap / minimal inverter input cap
14 -- X_factor = cell drive strength (resize factor
15 --           with respect to minimal size X1)
16
17 entity nand2 is
18     generic (
19        tpd_min: time := 10 ps; -- not used in logic model
20        tpd_FO4: time := 50 ps; -- not used in logic model
21        W_FO_value: integer := 0; -- not used in logic model
22        X_factor: integer := 1); -- not used in logic model
23     port (
24        nodeA, nodeB: inout std_logic := 'Z';
25        nodeZ: inout std_logic := 'Z');
26 end nand2;
27 ------------------------------------------------
28
29 ------------------------------------------------
30 architecture abstract of nand2 is
31
32 -- definition of timing constants
33 constant tau_i: time := (tpd_min/3.0);
34 constant tau_o: time := (tpd_FO4 - tpd_min)/(3*4);
35
36 begin
37     nodeZ <= nodeA nand nodeB after tau_i;
38 end abstract;
39 ------------------------------------------------
```

A.3. Example: NAND2 testbench at logic level

```
1  -- ************** MODERN testbench ************
```

```
2  -- cell tested: NAND2
3  -- ***********************************************
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7
8  use IEEE.std_logic_textio.ALL;
9
10 library modern2010;
11 use modern2010.logical_drive_primitives.all;
12 use modern2010.logical_drive_delay_basic.all;
13
14 library std;
15 use std.textio.all;
16
17 ------------------------------------------------
18 entity nand2_testbench is
19 generic (
20    X: real;
21    pr: std_logic;
22    random_sim: boolean;
23    Z_load: load_net_record;
24    tr: time;
25    period: time := 2 ns);
26 port (
27    nodeA,nodeB:inout logical_drive_logic:=('Z', 66 fs, (0.0,0.0));
28    nodeZ,nodeY:inout logical_drive_logic:=('Z', 66 fs, (0.0,0.0))
29   );
30 end nand2_testbench;
31 ------------------------------------------------
32
33 ------------------------------------------------
34 architecture testbench_arch of nand2_testbench is
35
36 signal i: integer := 0; -- time count
37 signal reference_clock: std_logic := '0'; -- time count
38
39 -- SEGNALI CALCOLO RITARDI
40 signal tempo_1, tpA_hl, tpA_lh, tpB_hl, tpB_lh: time:=0 ns;
41 signal nodeA_int, nodeB_int: std_logic;
42 -- END SEGNALI CALCOLO RITARDI
43
44 --funzione scrivi intestazione
45 impure function scrivi_intestazione return time is
46    file to_file_int: text open APPEND_MODE is "ritardi.txt";
47    file to_file_int_m: text open WRITE_MODE is \
48     "ritardi_montecarlo_X"&real'image(X)&"_tr"&\
49     time'image(tr)&"_cl"&real'image(Z_load.Z_load_PDN)&".txt";
50    variable buf_in: line;
51    variable nul: time;
52 begin
53
```

```
54    if random_sim then
55        WRITE(buf_in,string'("RitardoA_LH"));
56        WRITE(buf_in,HT);
57        WRITE(buf_in,string'("RitardoB_LH"));
58        WRITE(buf_in,HT);
59        WRITE(buf_in,string'("RitardoA_HL"));
60        WRITE(buf_in,HT);
61        WRITE(buf_in,string'("RitardoB_HL"));
62        WRITELINE(to_file_int_m,buf_in);
63        file_close(to_file_int_m);
64    else
65      WRITE(buf_in,string'("X"));
66      WRITE(buf_in,integer(X));
67      WRITE(buf_in,string'(".0"));
68      WRITE(buf_in,HT);
69      WRITE(buf_in,string'("tr="));
70      WRITE(buf_in,integer(tr/1 fs)/1000);
71      WRITE(buf_in,string'(" ps"));
72      WRITE(buf_in,LF);
73      WRITE(buf_in,string'("Z_load_PUP"));
74        WRITE(buf_in,HT);
75        WRITE(buf_in,string'("RitardoA_LH"));
76        WRITE(buf_in,HT);
77        WRITE(buf_in,string'("RitardoB_LH"));
78        WRITE(buf_in,HT);
79        WRITE(buf_in,string'("Z_load_PDN"));
80        WRITE(buf_in,HT);
81        WRITE(buf_in,string'("RitardoA_HL"));
82        WRITE(buf_in,HT);
83        WRITE(buf_in,string'("RitardoB_HL"));
84        WRITELINE(to_file_int,buf_in);
85        file_close(to_file_int);
86    end if;
87
88        return nul;
89  end function scrivi_intestazione;
90  --end funzione scrivi intestazione
91
92  --funzione scrivi ritardo
93  impure function \
94    scrivi_ritardo(tpA_lh, tpA_hl, tpB_lh, tpB_hl: in time)
95    return time is
96      file to_file:text open APPEND_MODE is "ritardi.txt";
97      file to_file_m: text open WRITE_MODE is \
98        "ritardi_montecarlo_X"&real'image(X)&\
99        "_tr"&time'image(tr)&"_cl"&\
100       real'image(Z_load.Z_load_PDN)&".txt";
101     variable buf_in:line;
102     variable tempo_ritardo:time;
103 begin
104
105   if random_sim then
```

```
106     WRITE(buf_in,real(tpA_lh/1.0 fs)/1000.0);
107     WRITE(buf_in,HT);
108     WRITE(buf_in,real(tpB_lh/1.0 fs)/1000.0);
109     WRITE(buf_in,HT);
110     WRITE(buf_in,real(tpA_hl/1.0 fs)/1000.0);
111     WRITE(buf_in,HT);
112     WRITE(buf_in,real(tpB_hl/1.0 fs)/1000.0);
113     WRITELINE(to_file_m,buf_in);
114       file_close(to_file_m);
115   else
116     WRITE(buf_in,Z_load.Z_load_PUP);
117       WRITE(buf_in,HT);
118     WRITE(buf_in,real(tpA_lh/1.0 fs)/1000.0);
119     WRITE(buf_in,HT);
120     WRITE(buf_in,real(tpB_lh/1.0 fs)/1000.0);
121     WRITE(buf_in,HT);
122       WRITE(buf_in,Z_load.Z_load_PDN);
123       WRITE(buf_in,HT);
124     WRITE(buf_in,real(tpA_hl/1.0 fs)/1000.0);
125     WRITE(buf_in,HT);
126     WRITE(buf_in,real(tpB_hl/1.0 fs)/1000.0);
127     WRITELINE(to_file,buf_in);
128       file_close(to_file);
129   end if;
130
131 return tempo_ritardo;
132 end function scrivi_ritardo;
133 --end funzione scrivi ritardo
134
135 begin
136
137 -- reference clock generator
138 reference_clock <= not reference_clock after period;
139
140 -- input_generator
141 process (reference_clock)
142 type test_vector_type is array (0 to 12) \
143     of std_logic_vector (1 downto 0);
144 variable test_vector: test_vector_type :=
145 (
146 "00",
147 "01",
148 "00",
149 "10",
150 "00",
151 "01",
152 "11",
153 "01",
154 "00",
155 "10",
156 "11",
157 "10",
```

```
158 "00"
159 );
160 begin
161     if i < test_vector'length then
162      nodeA.level <= (test_vector(i)(0)) ;
163    nodeA.slew <= tr;
164      nodeB.level <= (test_vector(i)(1));
165    nodeB.slew <= tr;
166      --CALCOLO DEL RITARDO
167      nodeA_int <= (test_vector(i)(0));
168      nodeB_int <= (test_vector(i)(1));
169      --CALCOLO DEL RITARDO
170      i <= i+1;
171     end if;
172 end process;
173
174 --check process
175 check: process (reference_clock)
176 begin
177
178    ASSERT nodeZ.level = nodeY.level
179    REPORT "Uscite diverse"
180    SEVERITY WARNING;
181
182   end process check;
183 -- end check process
184
185 --CALCOLO DEL RITARDO
186 calcolo_ritardi: process (nodeY.level)
187 begin
188     if (nodeA_int'last_event < nodeB_int'last_event) then
189         case nodeY.level is
190             when '0' => tpA_hl <= nodeA_int'last_event;
191
192             when '1' => tpA_lh <= nodeA_int'last_event;
193             when others => null;
194         end case;
195         else
196         case nodeY.level is
197             when '0' => tpB_hl <= nodeB_int'last_event;
198
199             when '1' => tpB_lh <= nodeB_int'last_event;
200             when others => null;
201         end case;
202     end if;
203 end process calcolo_ritardi;
204 --END CALCOLO DEL RITARDO
205 process
206     variable ret: time;
207     begin
208         wait for 90 ns;
209         ret:= scrivi_ritardo(tpA_lh, tpA_hl, tpB_lh, tpB_hl);
```

```
210     end process;
211
212 process
213     variable ret: time;
214     begin
215         if (pr='0') then ret:= scrivi_intestazione; end if;
216         wait  for 101 ns;
217     end process;
218
219 end testbench_arch;
220
221 ------------------------------------------------------------
222 ------------------------------------------------------------
223 ----------------------TESTBENCH PLATFORM--------------------
224 ------------------------------------------------------------
225 ------------------------------------------------------------
226 library ieee;
227 use ieee.std_logic_1164.all;
228
229 library modern2010;
230 use modern2010.logical_drive_primitives.all;
231 use modern2010.logical_drive_delay_basic.all;
232 ------------------------------------------------
233 entity nand2_testbench_platform is
234     generic (
235      pr: std_logic := '0';
236      tr: time := 1 ps;
237      X: real := 1.0;
238      Z_load_PUP: real := 10.0;
239      Z_load_PDN: real := 10.0;
240        period: time := 2 ns;
241        i: integer:= 0;
242        random_sim: boolean:= false);
243 end nand2_testbench_platform;
244 ------------------------------------------------
245
246 ------------------------------------------------
247 architecture testbench_platform of nand2_testbench_platform is
248
249 component nand2 is
250  port (
251    nodeA, nodeB: inout std_logic := 'Z';
252    nodeZ: inout std_logic := 'Z' );
253 end component nand2;
254
255 component nand2_dut is
256  generic (
257    X: real := 1.0;
258    --Z_load: load_net_record;
259    tech: technology_record);
260  port (
261    nodeA,nodeB:inout logical_drive_logic:=('Z', 66 fs, (0.0,0.0));
```

```
262     nodeZ:inout logical_drive_logic:=('Z', 66 fs, (0.0,0.0)) );
263 end component nand2_dut;
264 
265 component nand2_testbench is
266  generic (
267    X: real;
268    pr: std_logic;
269    random_sim: boolean;
270    Z_load: load_net_record;
271    tr: time;
272    period: time := 2 ns);
273  port (
274    nodeA,nodeB:inout logical_drive_logic:=('Z', 66 fs, (0.0,0.0));
275    nodeZ,nodeY:inout logical_drive_logic:=('Z', 66 fs, (0.0,0.0))
276   );
277 end component nand2_testbench;
278 
279 signal nodeA, nodeB, nodeZ, nodeY: \
280     logical_drive_logic := ('Z', 66 fs, (0.0,0.0));
281 signal nodeZ_stdlogic: std_logic := 'Z';
282 
283 begin
284 
285 nodeZ <= (nodeZ_stdlogic, 1 ps, (0.0,0.0));
286 nodeY.pincap <= (Z_load_PDN,Z_load_PDN) when (nodeY.level='1') else
287         (Z_load_PUP,Z_load_PUP);
288 
289 cell: nand2
290     port map (nodeA.level, nodeB.level, nodeZ_stdlogic);
291 
292 cell_dut: nand2_dut
293  generic map (X => X,
294         -- Z_load =>
295            -- (Z_load_PUP => Z_load_PUP,
296             -- Z_load_PDN => Z_load_PDN),
297         tech =>
298             (a_n_var =>0.0,
299             a_p_var => 0.0,
300             t_min_n => 0.000127 ns,
301             t_min_p => 0.0002555 ns,
302             random_sim => random_sim,
303             i=>i))
304     port map (nodeA, nodeB, nodeY);
305 
306 tb: nand2_testbench
307     generic map (period=>period, tr=>tr,
308     Z_load =>(Z_load_PUP => Z_load_PUP, Z_load_PDN => Z_load_PDN),
309     pr => pr, random_sim => random_sim, X => X)
310     port map (nodeA, nodeB, nodeZ, nodeY);
311 
312 end testbench_platform;
313 
```

```
314 ----------------------------------------------------------------
315 ----------------------MONTECARLO----------------------------
316 ----------------------------------------------------------------
317 library ieee;
318 use ieee.std_logic_1164.all;
319
320 library modern2010;
321 use modern2010.logical_drive_primitives.all;
322 use modern2010.logical_drive_delay_basic.all;
323
324 ------------------------------------------------
325 entity montecarlo is
326 end montecarlo;
327 ------------------------------------------------
328
329 ------------------------------------------------
330 architecture super_testbench of montecarlo is
331
332 component nand2_testbench_platform is
333   generic (
334     i: integer);
335 end component nand2_testbench_platform;
336
337 begin
338
339 nand2_montecarlo: for i in 0 to 9999 generate
340 begin
341     montecarlo_instance: nand2_testbench_platform
342     generic map (i=>i);
343 end generate;
344
345 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.

```
1  #include <string>
2  #include <iostream>
3  #include <fstream>
4  #include <stdlib.h>
5
6  using namespace std;
7
8  //use:
9  //.genNETvar "name" L W ndepvarn ndepvarp toxpvar tr XX cload
10
11 int main(int argc,char *argv[]) {
12
13   string MP2,MN2,s4ff,s8ff,
14     nomeBASE,L,W,ndep_n,ndep_p,toxp,tr,XX,cload,
15     riga;
```

```
16
17    nomeBASE=argv[1];
18    L=argv[2];
19    W=argv[3];
20    ndep_n=argv[4];
21    ndep_p=argv[5];
22    toxp=argv[6];
23    tr=argv[7];
24    XX=argv[8];
25    cload=argv[9];
26
27
28    if (L=="0") L="45n";
29    if (W=="0") W="45n";
30    if (ndep_n=="0") ndep_n="6.5e+018";
31    if (ndep_p=="0") ndep_p="2.8e+018";
32    if (toxp=="0") toxp="6.5e-010";
33
34
35    string nomeIN=nomeBASE+".net";
36    ifstream in(&nomeIN[0]);
37
38    string nomeOUT=nomeBASE+"_.net";
39    ofstream out(&nomeOUT[0]);
40    while(getline(in, riga)){
41      if (riga.substr(0,11)==".PARAM Lmin")
42         out << ".PARAM Lmin=" << L << "n\n";
43      else if (riga.substr(0,11)==".PARAM Wmin")
44         out << ".PARAM Wmin=" << W << "n\n";
45      else if (riga.substr(0,15)==".PARAM ndepVARn")
46         out << ".PARAM ndepVARn=" << ndep_n << "\n";
47      else if (riga.substr(0,15)==".PARAM ndepVARp")
48         out << ".PARAM ndepVARp=" << ndep_p << "\n";
49      else if (riga.substr(0,14)==".PARAM toxpVAR")
50         out << ".PARAM toxpVAR=" << toxp << "\n";
51      else if (riga.substr(0,10)==".PARAM tr=")
52        out << ".PARAM tr=" << tr << "p\n";
53      else if (riga.substr(0,10)==".PARAM XX=")
54        out << ".PARAM XX=" << XX << "\n";
55      else if (riga.substr(0,11)==".PARAM XXX=")
56        out << ".PARAM XXX=" << XX << "\n";
57      else if (riga.substr(0,10)=="Cout nodeZ")
58        out << "Cout nodeZ 0 " << cload << "f\n";
59      else if (riga.substr(0,13)=="Cout2 nodeCo4")
60              //Cout2 nodeCo4 0 5f
61        out << "Cout2 nodeCo4 0 " << cload << "f\n";
62      else if (riga.substr(0,12)=="Cout2 nodeCo")
63              //Cout2 nodeCo4 0 5f
64        out << "Cout2 nodeCo 0 " << cload << "f\n";
65      else if (riga.substr(0,28)==".INCLUDE ../../subckt/subckt")
66        out << ".INCLUDE ../../../subckt/subckt\n";
67      else{
```

```
68        out << riga << "\n";
69      }
70
71    }
72    out.close();
73    in.close();
74
75  return 0;
76  }
```

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.

```
1   #include <string>
2   #include <iostream>
3   #include <fstream>
4   #include <stdlib.h>
5   #include <limits>
6
7   using namespace std;
8
9   int main(int argc,char * argv[]) {
10
11    ifstream inNOM1(argv[2]);
12    string nomeFile(argv[1]);
13
14    ofstream outritardi(argv[1],ios::app);
15
16
17    string s;
18    string delay_lh_4ff,delay_hl_4ff;
19    long double f_delay_lh_4ff,f_delay_hl_4ff;
20
21    int i=1,j=0;
22    int ini=0,fin=0;
23
24    int cont4f=0;
25
26    char d_lh4ff[20],d_hl4ff[20];
27
```

```
28  while(getline(inNOM1, s) && (!(cont4f==2)) ){
29      if (s.substr(0,8)=="delay_lh") {
30        i=0;
31        while(s[i]!='.'){
32          i++;
33        }
34        delay_lh_4ff =s.substr(i-1,16);
35        cont4f++;
36      }
37      if (s.substr(0,8)=="delay_hl") {
38        i=0;
39        while(s[i]!='.'){
40          i++;
41        }
42        delay_hl_4ff =s.substr(i-1,16);
43        cont4f++;
44      }
45  }
46
47  inNOM1.close();
48
49  for(i=0;i<delay_lh_4ff.length();i++)
50  {
51    d_lh4ff[i]=delay_lh_4ff[i];
52  }
53  f_delay_lh_4ff=atof(d_lh4ff);
54
55  for(i=0;i<delay_hl_4ff.length();i++)
56  {
57    d_hl4ff[i]=delay_hl_4ff[i];
58  }
59  f_delay_hl_4ff=atof(d_hl4ff);
60
61  float XX,tr,cload;
62  tr=atof(argv[7]);
63  XX=atof(argv[8]);
64  cload=atof(argv[9]);
65
66  float L_var=atof(argv[10]),
67        W_var=atof(argv[11]),
68        Ndepn_var=atof(argv[12]),
69        Ndepp_var=atof(argv[13]),
70        tox_var=atof(argv[14]);
71
72  outritardi.precision(10);
73  outritardi << argv[6] << "\t" << f_delay_lh_4ff << "\t" \
74      << f_delay_hl_4ff << "\t" << XX << "\t" << tr << \
75      "\t" << cload << "\t" << L_var << "\t" << W_var << \
76      "\t" << Ndepn_var << "\t" << Ndepp_var << "\t" << \
77      tox_var << "\n";
78
79  outritardi.close();
```

```
80 }
```

B.3. Table to VDHL matrix

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

```
1  /*
2  authors:  Antonio Mastrandrea
3  date:   2010-11-26
4    rev1  2012-02-13: change dimension matrix
5
6   Create matrix:
7      read a file with tau in column:
8           X=  1 tr= 1
9           Cload tau0_n X1_1ps taui_n X1_1ps taurap_n X1_1ps
10          0.00  0.000378556 0.000146375 0.000320935
11          0.15  0.000342995 0.000124285 0.000359595
12          0.33  0.000334367 0.000119916 0.000380228
13          0.66  0.00033184  0.000118354 0.000392316
14          ..........
15
16     create 3 file with matrix for tau_i tau_0 and tau_m:
17
18          tau_i:=(
19            (0.000378556,0.000342995,0.000334367,0.00033184,...),
20            (...                                              ),
21            ...
22            );
23
24          tau_0:=(
25            (0.000146375,0.000124285,0.000119916,0.000118354,...),
26            (...                                              ),
27            ...
28            );
29
30          tau_m:=(
31            (0.000320935,0.000359595,0.000380228,0.000392316,...),
32            (...                                              ),
33            ...
34            );
35 */
36 #include <string>
37 #include <iostream>
38 #include <fstream>
39 #include <stdlib.h>
```

```
40 #include <limits>
41 #include <sstream>
42
43 #define NUM_CAP 37
44 #define NUM_CAP_P NUM_CAP+1
45
46 using namespace std;
47
48 //function ************************
49 void find_value(string s,float* f);
50 void write_v_file(float *f);
51 void init_files_out();
52 void init_row();
53 void end_row();
54 void finalize_files();
55 /*
56 type of cell----------------------+
57 file open-------------------+     |
58 name program-----------+    |     |
59                        |    |     |
60                        |    |     |
61                        V    V     V
62                   $ prog tau_n 1_1
63 */
64 ifstream infile;
65 ofstream outtauo,outtaui,outtaum;
66 string typecell,tauo,taui,taum;
67 string s;
68
69 int main(int argc,char * argv[]) {
70
71
72   float f1[3];
73   //open 1 file
74   cout << "open file: "<< argv[1]<<"\n";
75       infile.open(argv[1]);
76
77   typecell=argv[2];
78   tauo="tau_o_"+typecell+".mat";
79   taui="tau_i_"+typecell+".mat";
80   taum="tau_m_"+typecell+".mat";
81   outtauo.open(tauo.c_str(),ios::app);
82   outtaui.open(taui.c_str(),ios::app);
83   outtaum.open(taum.c_str(),ios::app);
84
85   init_files_out();
86
87   init_row();
88   int conta_column=0;
89   while(getline(infile, s) ){
90     if ((s.substr(0,2)=="X=")||(s.substr(0,2)=="x="))
91       cout << "x= '" << s <<"'\n";
```

```
92      else if ((s=="")) cout << "line empty \"\"\n";
93      else if ( (s.substr(0,3)=="Clo")||(s.substr(0,3)=="clo") )
94        cout << "line intestation \"\"\n";
95      else
96      {
97      ++conta_column;
98      if (conta_column==NUM_CAP_P)
99      {
100       end_row();
101       conta_column=1;
102       init_row();
103     }
104     find_value(s,f1);
105     //write_v_file(f1);
106
107     outtauo << f1[0] << " ns" ;
108     outtaui << f1[1] << " ns";
109     outtaum << f1[2] << " ns";
110     if(conta_column!=NUM_CAP)
111     {
112       outtauo <<"\t,\t";
113       outtaui <<"\t,\t";
114       outtaum <<"\t,\t";
115     }
116
117     //cout << s << "\n";
118     }
119   }
120
121   finalize_files();
122 return 0;
123 }
124
125 void find_value(string s,float* f)
126 {
127   //string s1;
128   float a,b,c,d;
129   std::istringstream iss(s);
130   iss >> a ;
131   iss >> *f;
132   iss >> *(f+1);
133   iss >> *(f+2);
134   cout << a << "\t\t";
135   cout << *f << "\t\t";
136   cout << *(f+1) << "\t\t";
137   cout << *(f+2) << "\n";
138   //cout << b << "\n";
139 }
140
141 void write_v_file(float *f)
142 {
143
```

```
144 }
145
146 void init_files_out()
147 {
148   outtauo << "constant "
149     << tauo.substr(0,tauo.length()-4) << ": matrix_tau := (";
150   outtaui << "constant "
151     << taui.substr(0,taui.length()-4) << ": matrix_tau := (";
152   outtaum << "constant "
153     << taum.substr(0,taum.length()-4) << ": matrix_tau := (";
154  }
155
156 void init_row()
157 {
158   outtauo << "\n\t(\t";
159   outtaui << "\n\t(\t";
160   outtaum << "\n\t(\t";
161 }
162
163 void end_row()
164 {
165   outtauo << "\t),";
166   outtaui << "\t),";
167   outtaum << "\t),";
168 }
169
170 void finalize_files()
171 {
172   outtauo  << "\t)\n);\n";
173   outtaui  << "\t)\n);\n";
174   outtaum  << "\t)\n);\n";
175
176   outtauo.close();
177   outtaui.close();
178   outtaum.close();
179   infile.close();
180 }
```

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 τ 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"
```

```
31    exit
32  fi
33  ############################################################
34  ############################################################
35  if [ $rimuovi == "SI" ] then
36    echo "rimuovo i file di testo della simulazione
37       precedente"
38    for tr in  1 10 20 30 40 50
39    do
40      for XX in 1 5 10 20
41      do
42        cartella="X${XX}_${tr}ps"
43        cd $cartella
44        rm *.txt
45        rm *fF.net
46        cd ..
47      done
48    done
49  fi
50  if [ $crea == "SI" ]
51  then
52    echo "creo le cartelle"
53    for tr in  1 10 20 30 40 50
54    do
55      for XX in 1 5 10 20
56      do
57        cartella="X${XX}_${tr}ps"
58        mkdir $cartella
59        echo creata $cartella
60      done
61    done
62  fi
63  if [ $cfile == "SI" ]
64  then
65    echo "copio i file di base"
66    for tr in  1 10 20 30 40 50
67    do
68      for XX in 1 5 10 20
69      do
70        cartella="X${XX}_${tr}ps"
71        for i in ${fileBASE}*.net
72        do
73          ./gNetVarTR_X $i $tr $XX
74        done
75        mv *ps.net $cartella
76      done
77    done
78  fi
79  if [ $simula == "SI" ]
80  then
81    #inizia le simulazioni
82    echo "inizio simulazioni"
```

```
83  CL[0]=0.00;
84  CL[1]=0.15;
85  CL[2]=0.33;
86  CL[3]=0.66;
87  CL[4]=1.00;
88  CL[5]=1.25;
89  CL[6]=1.5;
90  CL[7]=1.75;
91  CL[8]=2;
92  CL[9]=3;
93  CL[10]=4;
94  CL[11]=5;
95  CL[12]=6;
96  CL[13]=7;
97  CL[14]=8;
98  CL[15]=9;
99  CL[16]=10;
100 CL[17]=11;
101 #simulazioni
102 for tr in 10 20 30 40 50
103 do
104   for XX in 1 5 10 20
105   do
106     cartella="X${XX}_${tr}ps"
107     cd $cartella
108     j=1
109     #nel file dove raccolgo tutti i tau scrivo
110      l'intestazione per ogni serie di misura
111     echo "X= ${XX} tr= ${tr}" >> ../tau_n_.txt
112     echo "Cload tau0_n X${XX}_${tr}ps
113         taui_n X${XX}_${tr}ps taurap_n
114         X${XX}_${tr}ps" >> ../tau_n_.txt
115     echo "X=  ${XX} tr= ${tr}" >>
116               "tau_n_Cl_X${XX}_tr${tr}ps.txt"
117     echo "Cload tau0_n X${XX}_${tr}ps
118         taui_n X${XX}_${tr}ps
119         taurap_n X${XX}_${tr}ps">>
120         "tau_n_Cl_X${XX}_tr${tr}ps.txt"
121     echo "X= ${XX} tr= ${tr}" >> ../tau_p_.txt
122     echo "Cload tau0_p X${XX}_${tr}ps taui_p
123         X${XX}_${tr}ps taurap_p X${XX}_${tr}ps"
124                 >> ../tau_p_.txt
125     echo "X= ${XX} tr= ${tr}" >>
126        "tau_p_Cl_X${XX}_tr${tr}ps.txt"
127     echo "Cload tau0_p X${XX}_${tr}ps
128         taui_p X${XX}_${tr}ps taurap_p
129         X${XX}_${tr}ps" >>
130         "tau_p_Cl_X${XX}_tr${tr}ps.txt"
131     echo "X= ${XX} tr= ${tr}" >>
132                 ../ritardi_.txt
133     echo "Cload 4ff_lh X${XX}_${tr}ps 8ff_lh
134        X${XX}_${tr}ps conMN2_lh
```

```
135           X${XX}_${tr}ps 4ff_hl X${XX}_${tr}ps
136           8ff_hl X${XX}_${tr}ps
137           conMP2_hl X${XX}_${tr}ps"
138           >> ../ritardi_.txt
139         echo "X= ${XX} tr= ${tr}" >>
140           "ritardi_Cl_X${XX}_tr${tr}ps.txt"
141         echo "Cload 4ff_lh X${XX}_${tr}ps
142           8ff_lh X${XX}_${tr}ps  conMN2_lh
143           X${XX}_${tr}ps 4ff_hl X${XX}_${tr}ps
144           8ff_hl X${XX}_${tr}ps  conMP2_hl
145           X${XX}_${tr}ps" >>
146           "ritardi_Cl_X${XX}_tr${tr}ps.txt"
147         for cload in 0..16
148         do
149           j=1
150           .././gNetVarC "${fileBASE}conMN2X${XX}_tr
151             ${tr}ps.net" "${CL[$cload]}"
152           .././gNetVarC "${fileBASE}conMP2X${XX}_tr
153             ${tr}ps.net" "${CL[$cload]}"
154           .././gNetVarC "${fileBASE}senza_4fFX${XX}
155             _tr${tr}ps.net" "${CL[$cload]}"
156           a=$[$cload+1]
157           .././gNetVarC "${fileBASE}senza_8fFX${XX}
158             _tr${tr}ps.net" "${CL[$a]}"
159           for i in *fF.net
160           do
161             ngspice -b $i > "${j}.txt"
162             j=$[$j+1]
163           done
164           .././calcolaTAU "tauCl_X${XX}_tr${tr}ps.txt"
165             "1.txt" "2.txt" "3.txt" "4.txt"
166             "${CL[$cload]}_${CL[$cload+1]}" "${XX}"
167             "${CL[$a]}" "${CL[$cload]}"
168           rm *fF.net #temp
169           rm "1.txt" "2.txt" "3.txt" "4.txt"
170         done
171         echo "" >> ../tau_n_.txt
172         echo "" >> ../tau_p_.txt
173         echo "" >> ../ritardi_.txt
174         cd ..
175         #echo creata $cartella
176       done
177       echo "" >> tau_n_.txt
178       echo "" >> tau_p_.txt
179       echo "" >> ritardi_.txt
180     done
181   fi
182   echo "inizio ${fileBASE} ${ingresso} : $DATE_INIZIO"
183     > time_${fileBASE}.txt echo "fine   ${fileBASE}
184     ${ingresso} : $(date)" >> time_${fileBASE}.txt
185   echo "inizio ${fileBASE} ${ingresso}: $DATE_INIZIO"
186     echo "fine   ${fileBASE} ${ingresso}: $(date)"
```

```
187 exit
```

C.2. Calculate C_{in} Capacitance

```
1  #!/bin/bash
2
3  cella="not"
4  ingresso="a"
5
6  netHL="not_not_HL.net"
7  netHL_cl="not_cload_HL.net"
8
9  netLH="not_not_LH.net"
10 netLH_cl="not_cload_LH.net"
11
12 g++ gNetVarC.cpp -o gNetVarC
13 g++ gNetVarTR_X.cpp -o gNetVarTR_X
14 g++ calcolaRitardiHL.cpp -o calcolaRitardiHL
15 g++ calcolaRitardiLH.cpp -o calcolaRitardiLH
16 g++ somma.cpp -o somma
17 g++ confronta.cpp -o confronta
18
19 path_=$(pwd)
20 creaNET_C=${path_}/./gNetVarC
21 creaNET_TR_X=${path_}/./gNetVarTR_X
22 trovaRITARDI_HL=${path_}/./calcolaRitardiHL
23 trovaRITARDI_LH=${path_}/./calcolaRitardiLH
24 SOMMA=${path_}/./somma
25 CONFRONTA=${path_}/./confronta
26
27 TEST_=${path_}/./test
28
29 ########################################################################
30 #   Cin HL
31 ########################################################################
32 simulazioni=0
33 for XX in 1 2 3 4 5 10 20
34 do
35   for tr in 10 50
36   do
37     ${creaNET_TR_X} "${cella}/${ingresso}/${netHL}" $tr $XX
38     cd "${cella}/${ingresso}"
39     netlist_=$(ls *ps.net)
40     ngspice -b ${netlist_} > "1.txt"
41
42     delayHL=$(${trovaRITARDI_HL} "1.txt")
43     rm ${netlist_} "1.txt"
44
```

```
45      #echo "il ritardo è ${delayHL}"
46      cd ${path_}
47
48      Cl=0
49      HLfind=1
50      while [ $HLfind != 0 ]
51      do
52        simulazioni=$(${SOMMA} ${simulazioni} 1)
53        Cl=$(${SOMMA} $Cl 1)
54        ${creaNET_C} ${netHL_cl} ${Cl} $XX $tr
55
56        netlist_=$(ls *fF.net)
57        ngspice -b ${netlist_} > "1.txt"
58
59        delayHL_1=$(${trovaRITARDI_HL} "1.txt")
60        rm ${netlist_} "1.txt"
61
62        HLfind=$(${CONFRONTA} ${delayHL_1} ${delayHL})
63      done
64
65      HLfind=1
66      Cl=$(${SOMMA} $Cl -1)
67      while [ $HLfind != 0 ]
68      do
69        simulazioni=$(${SOMMA} ${simulazioni} 1)
70        Cl=$(${SOMMA} $Cl 0.1)
71        ${creaNET_C} ${netHL_cl} ${Cl} $XX $tr
72
73        netlist_=$(ls *fF.net)
74        ngspice -b ${netlist_} > "1.txt"
75
76        delayHL_1=$(${trovaRITARDI_HL} "1.txt")
77        rm ${netlist_} "1.txt"
78
79        HLfind=$(${CONFRONTA} ${delayHL_1} ${delayHL})
80      done
81
82      HLfind=1
83      Cl=$(${SOMMA} $Cl -0.1)
84      while [ $HLfind != 0 ]
85      do
86        simulazioni=$(${SOMMA} ${simulazioni} 1)
87        Cl=$(${SOMMA} $Cl 0.01)
88        ${creaNET_C} ${netHL_cl} ${Cl} $XX $tr
89
90        netlist_=$(ls *fF.net)
91        ngspice -b ${netlist_} > "1.txt"
92
93        delayHL_1=$(${trovaRITARDI_HL} "1.txt")
94        rm ${netlist_} "1.txt"
95
96        HLfind=$(${CONFRONTA} ${delayHL_1} ${delayHL})
```

```
97      done
98
99      HLfind=1
100     Cl=$(${SOMMA} $Cl -0.01)
101     while [ $HLfind != 0 ]
102     do
103       simulazioni=$(${SOMMA} ${simulazioni} 1)
104       Cl=$(${SOMMA} $Cl 0.001)
105       ${creaNET_C} ${netHL_cl} ${Cl} $XX $tr
106
107       netlist_=$(ls *fF.net)
108       ngspic1 2 3 4 5 10 20e -b ${netlist_} > "1.txt"
109
110       delayHL_1=$(${trovaRITARDI_HL} "1.txt")
111       rm ${netlist_} "1.txt"
112
113       HLfind=$(${CONFRONTA} ${delayHL_1} ${delayHL})
114     done
115
116     Cl=$(${SOMMA} $Cl -0.001)
117
118     HLfind=1
119     while [ $HLfind != 0 ]
120     do
121       simulazioni=$(${SOMMA} ${simulazioni} 1)
122       Cl=$(${SOMMA} $Cl 0.0001)
123       ${creaNET_C} ${netHL_cl} ${Cl} $XX $tr
124
125       netlist_=$(ls *fF.net)
126       ngspice -b ${netlist_} > "1.txt"
127
128       delayHL_1=$(${trovaRITARDI_HL} "1.txt")
129       rm ${netlist_} "1.txt"
130
131       HLfind=$(${CONFRONTA} ${delayHL_1} ${delayHL})
132     done
133
134     Cl=$(${SOMMA} $Cl -0.0001)
135
136     simulazioni=$(${SOMMA} ${simulazioni} 1)
137     ${creaNET_C} ${netHL_cl} ${Cl} $XX $tr
138
139     netlist_=$(ls *fF.net)
140     ngspice -b ${netlist_} > "1.txt"
141
142     delayHL_1=$(${trovaRITARDI_HL} "1.txt")
143     rm ${netlist_} "1.txt"
144
145     echo "X=$XX tr=$tr ps"
146     echo "delayHLbase= ${delayHL}"
147     echo "delayHLtrovato= ${delayHL_1}"
148     echo "CloadHL= ${Cl}"
```

```
149     echo "X=$XX tr=$tr ps  delayHLbase= ${delayHL}  \
150         delayHLtrovato= ${delayHL_1}  CloadHL= ${Cl}"\
151         >> test${cella}_${ingresso}.txt
152     echo "X=  $XX tr= $tr ps  \
153         delayHLbase= ${delayHL}  delayHLtrovato= \
154         ${delayHL_1}  CloadHL=  ${Cl} " >> \
155         test${cella}_${ingresso}GRAF.txt
156   done
157 done
158 
159     echo "simulazioni totali per trovare 14 CHL: ${simulazioni}"
160     echo "simulazioni totali per trovare 14 CHL: ${simulazioni}" \
161         >> test${cella}_${ingresso}.txt
162     echo " " >> test${cella}_${ingresso}.txt
163 
164 ########################################################################
165 #   Cin LH
166 ########################################################################
167 simulazioni=0
168 for XX in 1 2 3 4 5 10 20
169 do
170   for tr in 10 50
171   do
172     ${creaNET_TR_X} "${cella}/${ingresso}/${netLH}" $tr $XX
173     cd "${cella}/${ingresso}"
174     netlist_=$(ls *ps.net)
175     ngspice -b ${netlist_} > "1.txt"
176 
177     delayLH=$(${trovaRITARDI_LH} "1.txt")
178     rm ${netlist_} "1.txt"
179 
180     #echo "il ritardo è ${delayLH}"
181     cd ${path_}
182 
183     Cl=0
184     LHfind=1
185     while [ $LHfind != 0 ]
186     do
187       simulazioni=$(${SOMMA} ${simulazioni} 1)
188       Cl=$(${SOMMA} $Cl 1)
189       ${creaNET_C} ${netLH_cl} ${Cl} $XX $tr
190 
191       netlist_=$(ls *fF.net)
192       ngspice -b ${netlist_} > "1.txt"
193 
194       delayLH_1=$(${trovaRITARDI_LH} "1.txt")
195       rm ${netlist_} "1.txt"
196 
197       LHfind=$(${CONFRONTA} ${delayLH_1} ${delayLH})
198     done
199 
200     LHfind=1
```

```
201     Cl=$(${SOMMA} $Cl -1)
202     while [ $LHfind != 0 ]
203     do
204       simulazioni=$(${SOMMA} ${simulazioni} 1)
205       Cl=$(${SOMMA} $Cl 0.1)
206       ${creaNET_C} ${netLH_cl} ${Cl} $XX $tr
207
208       netlist_=$(ls *fF.net)
209       ngspice -b ${netlist_} > "1.txt"
210
211       delayLH_1=$(${trovaRITARDI_LH} "1.txt")
212       rm ${netlist_} "1.txt"
213
214       LHfind=$(${CONFRONTA} ${delayLH_1} ${delayLH})
215     done
216
217     LHfind=1
218     Cl=$(${SOMMA} $Cl -0.1)
219     while [ $LHfind != 0 ]
220     do
221       simulazioni=$(${SOMMA} ${simulazioni} 1)
222       Cl=$(${SOMMA} $Cl 0.01)
223       ${creaNET_C} ${netLH_cl} ${Cl} $XX $tr
224
225       netlist_=$(ls *fF.net)
226       ngspice -b ${netlist_} > "1.txt"
227
228       delayLH_1=$(${trovaRITARDI_LH} "1.txt")
229       rm ${netlist_} "1.txt"
230
231       LHfind=$(${CONFRONTA} ${delayLH_1} ${delayLH})
232     done
233
234     LHfind=1
235     Cl=$(${SOMMA} $Cl -0.01)
236     while [ $LHfind != 0 ]
237     do
238       simulazioni=$(${SOMMA} ${simulazioni} 1)
239       Cl=$(${SOMMA} $Cl 0.001)
240       ${creaNET_C} ${netLH_cl} ${Cl} $XX $tr
241
242       netlist_=$(ls *fF.net)
243       ngspice -b ${netlist_} > "1.txt"
244
245       delayLH_1=$(${trovaRITARDI_LH} "1.txt")
246       rm ${netlist_} "1.txt"
247
248       LHfind=$(${CONFRONTA} ${delayLH_1} ${delayLH})
249     done
250
251     Cl=$(${SOMMA} $Cl -0.001)
252
```

```
253     LHfind=1
254     while [ $LHfind != 0 ]
255     do
256       simulazioni=$(${SOMMA} ${simulazioni} 1)
257       Cl=$(${SOMMA} $Cl 0.0001)
258       ${creaNET_C} ${netLH_cl} ${Cl} $XX $tr
259
260       netlist_=$(ls *fF.net)
261       ngspice -b ${netlist_} > "1.txt"
262
263       delayLH_1=$(${trovaRITARDI_LH} "1.txt")
264       rm ${netlist_} "1.txt"
265
266       LHfind=$(${CONFRONTA} ${delayLH_1} ${delayLH})
267     done
268     Cl=$(${SOMMA} $Cl -0.0001)
269
270     simulazioni=$(${SOMMA} ${simulazioni} 1)
271     ${creaNET_C} ${netLH_cl} ${Cl} $XX $tr
272
273     netlist_=$(ls *fF.net)
274     ngspice -b ${netlist_} > "1.txt"
275
276     delayLH_1=$(${trovaRITARDI_LH} "1.txt")
277     rm ${netlist_} "1.txt"
278
279     echo "X=$XX tr=$tr ps"
280     echo "delayLHbase= ${delayLH}"
281     echo "delayLHtrovato= ${delayLH_1}"
282     echo "CloadLH= ${Cl}"
283     echo "X=$XX tr=$tr ps  delayLHbase= ${delayLH}  \
284         delayLHtrovato= ${delayLH_1}  CloadLH= \
285         ${Cl}" >> test${cella}_${ingresso}.txt
286     echo "X=  $XX tr= $tr ps  \
287         delayHLbase= ${delayLH}  delayHLtrovato= \
288         ${delayLH_1}  CloadHL=  ${Cl} \
289         " >> test${cella}_${ingresso}GRAF.txt
290   done
291 done
292     echo "simulazioni totali per trovare 14 CLH: ${simulazioni}"
293     echo "simulazioni totali per trovare 14 CLH: ${simulazioni}" \
294         >> test${cella}_${ingresso}.txt
295 exit
```

C.3. Example: Deterministic circuit level simulation

```
1 #!/bin/bash
2
```

```
3  g++ calcolaRitardi.cpp -o calcolaRitardi
4  g++ gNetVarC.cpp -o gNetVarC
5  g++ gNetVarTR_X.cpp -o gNetVarTR_X
6  sleep 1
7
8  export DATE_INIZIO=$(date)
9  ############################################################
10 fileBASE="64_fa1"
11 ingresso="mezzo"
12
13 rimuovi=NO  #remove last result
14 crea=SI     #create folder structure
15 cfile=SI    #copy netlist in folder structure
16 simula=SI   #simulate
17 ############################################################
18 if [ $rimuovi == "SI" ]
19 then
20   echo "remove last result in progress..."
21   for tr in  10 20 30 40 50
22   do
23     for XX in 1 5 10 20
24     do
25       cartella="X${XX}_${tr}ps"
26       cd $cartella
27       rm *.txt
28       rm *fF.net
29       cd ..
30             #echo create $cartella
31           done
32   done
33 fi
34
35 if [ $crea == "SI" ]
36 then
37   echo "creo le cartelle"
38   for tr in  10 50 #20 30 40 50
39   do
40     for XX in 1 10 #1 5 10 20
41     do
42       cartella="X${XX}_${tr}ps"
43
44       cartella="X${XX}_${tr}ps"
45             mkdir $cartella
46             echo creata $cartella
47           done
48   done
49 fi
50
51 if [ $cfile == "SI" ]
52 then
53   echo "file copy..."
54
```

```
55    for tr in  10 20 30 40 50
56    do
57      for XX in 1 5 10 20
58      do
59        cartella="X${XX}_${tr}ps"
60        for i in ${fileBASE}*.net
61        do
62              ./gNetVarTR_X $i $tr $XX
63            done
64            mv *ps.net $cartella
65          done
66    done
67  fi
68
69  if [ $simula == "SI" ]
70  then
71    #init simulations
72
73    CL[0]=0.00;
74    CL[1]=0.15;
75    CL[2]=0.33;
76    CL[3]=0.66;
77    CL[4]=1.00;
78    CL[5]=1.25;
79    CL[6]=1.5;
80    CL[7]=1.75;
81    CL[8]=2;
82    CL[9]=3;
83    CL[10]=4;
84    CL[11]=5;
85    CL[12]=6;
86    CL[13]=7;
87    CL[14]=8;simulation at circuit level simulation
88    CL[15]=9;
89    CL[16]=10;
90    CL[17]=11;
91
92    for tr in  10 20 30 40 50
93    do
94      for XX in 1 5 10 20
95      do
96        cartella="X${XX}_${tr}ps"
97        cd $cartella
98        j=1
99        echo "X=  ${XX} tr= ${tr}" >> ../ritardi_.txt
100       echo "Cload lh_cin X${XX}_${tr}ps\
101             hl_cin X${XX}_${tr}ps" >> ../ritardi_.txt
102
103       echo "X=  ${XX} tr= ${tr}" >> \
104             "ritardi_Cl_X${XX}_tr${tr}ps.txt"
105       echo "Cload lh_cin X${XX}_${tr}ps hl_cin \
106          X${XX}_${tr}ps" >> "ritardi_Cl_X${XX}_tr${tr}ps.txt"
```

```
107
108       for cload in 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
109       do
110         j=1
111         .././gNetVarC "${fileBASE}X${XX}_tr${tr}ps.net" \
112                       "${CL[$cload]}"
113
114         for i in *fF.net
115         do
116             ngspice -b $i > "${j}.txt"
117             j=$[$j+1]
118         done
119         .././calcolaRitardi "tauCl_X${XX}_tr${tr}ps.txt" \
120            "1.txt" "2.txt" "3.txt" "4.txt" "${CL[$cload]}_\
121            ${CL[$cload+1]}" "${XX}" "${CL[$a]}" "${CL[$cload]}"
122         rm "1.txt" "2.txt" "3.txt" "4.txt"
123       done
124
125       echo "" >> ../ritardi_.txt
126       cd ..
127     done
128     echo "" >> ritardi_.txt
129   done
130 fi
131
132 mv ritardi_.txt ritardi_${fileBASE}.txt
133 echo "inizio ${fileBASE} ${ingresso} : $DATE_INIZIO" > \
134        time_${fileBASE}.txt
135 echo "fine   ${fileBASE} ${ingresso} : $(date)" >> \
136        time_${fileBASE}.txt
137 echo "inizio ${fileBASE} ${ingresso}: $DATE_INIZIO"
138 echo "fine   ${fileBASE} ${ingresso}: $(date)"
139 exit
```

C.4. Example: Statistical circuit level simulation

```
1  #!/bin/bash
2
3  pathATTUALE="variazioniCASUALI"
4  cprogramm="cprogramm"
5  cella="a1b0c_cout_x1_tr10_cl1"
6
7  fileBASE=$1
8
9  PARAMSPATH=../../TechParams
10
11 iterazioni=10000
12 j=45
```

```
13
14 declare -i COUNT=0
15 declare -i CCOUNT=0
16
17 if [ -e $pathATTUALE ]
18 then
19   echo "work folder find"
20 else
21   mkdir $pathATTUALE
22 fi
23
24 ##############################################
25 # technology parameter
26 ##############################################
27     COUNT=0
28     for i in `cat $PARAMSPATH/Toxp_$j.par`; do
29   Toxp[$COUNT]=$i
30   COUNT=COUNT+1
31     done;
32     COUNT=0
33     for i in `cat $PARAMSPATH/L_$j.par`; do
34   L[$COUNT]=$i
35   COUNT=COUNT+1
36     done;
37     COUNT=0
38     for i in `cat $PARAMSPATH/W_$j.par`; do
39   W[$COUNT]=$i
40       COUNT=COUNT+1
41     done;
42     COUNT=0
43     for i in `cat $PARAMSPATH/Nn_$j.par`; do
44       Nn[$COUNT]=$i
45       COUNT=COUNT+1
46     done;
47     COUNT=0
48     for i in `cat $PARAMSPATH/Np_$j.par`; do
49   Np[$COUNT]=$i
50   COUNT=COUNT+1
51     done;
52 ################################################
53 # End technology parameter
54 ################################################
55
56 ################################################
57 # begin simulations
58 ################################################
59
60 XX=1
61 tr=10
62 cload=1
63
64 echo "begin simulations"
```

```
65 echo
66 echo "iterazione  delay_lh_${perc}X${XX}_${tr}ps  \
67       delay_hl_${perc}${cload}_X${XX}_${tr}ps X tr\
68       cload L W Ndepn Ndepp tox" >> \
69       ${pathATTUALE}/delay_random.txt
70
71 COUNT=0
72
73 #*************************************************************
74 # delay random value
75 #*************************************************************
76
77 while [ "$COUNT" -lt "$iterazioni" ]
78 do
79   ../../${cprogramm}/./genNETvar ${fileBASE} ${L[$COUNT]} \
80     ${W[$COUNT]} ${Nn[$COUNT]} ${Np[$COUNT]} ${Toxp[$COUNT]}\
81     ${tr} ${XX} ${cload}
82
83   mv *_.net ${pathATTUALE}
84   cd ${pathATTUALE}
85   i=0
86   for netS in *_.net
87   do
88     ngspice -b $netS > "${COUNT}_${i}.txt"
89     i=$[$i+1]
90   done
91   ../../../${cprogramm}/./calcolaDELAYrandom delay_random.txt\
92   "${COUNT}_0.txt" "${COUNT}_1.txt" "${COUNT}_2.txt" \
93   "${COUNT}_3.txt" "${COUNT}" ${tr} ${XX} ${cload} \
94    ${L[$COUNT]} ${W[$COUNT]} ${Nn[$COUNT]} ${Np[$COUNT]} \
95    ${Toxp[$COUNT]}
96
97   rm ${COUNT}*.txt
98
99   COUNT=COUNT+1
100  cd ..
101 done
102
103 exit
```

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

```
1 #!/bin/bash
2 path_=$(pwd)
3
4 for i in $(ls)
```

```
5  do
6    if [ -d $i ] && [ $i != "model" ]
7    then
8      cd ${path_}/${i}
9
10     for folder in $(ls)
11     do
12       if [ -d $folder ]
13       then
14         cd ${path_}/${i}/${folder}
15
16         rm *.mat
17
18         cd ${path_}/${i}
19       fi
20     done
21     cd ${path_}
22   fi
23 done
24 exit
```

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

```
1  #!/bin/bash
2
3  new_tr="1 5 10 20 30 40 50"
4  new_X="1 2 3 4 5 10 20"
5
6  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 \
7    0.65 0.7 0.75 0.8 0.85 0.9 0.95 1.0 1.25 1.5 1.75 2.0 2.25 2.50 \
8    2.75 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0)
9  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 \
10   0.65 0.7 0.75 0.8 0.85 0.9 0.95 1.0 1.25 1.5 1.75 2.0 2.25 2.50 \
11   2.75 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0"
12
13 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 \
14   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 \
15   2.75 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0)
16 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 \
17   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 \
18   2.75 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0"
19 len=${#new_CL[*]}
20
21 len=${#new_CL1[*]}
22 let len--
23
24 if [ 2 = 2 ]
25 then
```

```
26  path_="$(pwd)"
27
28  for i in $(ls)
29  do
30    if [ -d $i ]
31    then
32      cd ${path_}/${i}
33
34      for folder in $(ls)
35      do
36        if [ -d $folder ]
37        then
38          cd ${path_}/${i}/${folder}
39
40
41          sed s/'for tr in'/"for tr in $new_tr #"/ \
42              creaCARTtr1_10_50ps_X1_10_20_Cvar > run_tau_1.sh
43          sed s/'for XX in'/"for XX in $new_X #"/ run_tau_1.sh \
44              > run_tau_2.sh
45
46          sed s/'CL\[0\]'/"CL=(${new_CL_}) # CL[0]"/ run_tau_2.sh \
47              > run_tau_3.sh
48          sed s/'CL\[1\]'/"CL1=(${new_CL1_}) # CL[1]"/ run_tau_3.sh\
49              > run_tau_4.sh
50          sed s/'CL\[2\]'/"# CL[2]"/ run_tau_4.sh > run_tau_5.sh
51          sed s/'CL\[3\]'/"# CL[3]"/ run_tau_5.sh > run_tau_6.sh
52          sed s/'CL\[4\]'/"# CL[4]"/ run_tau_6.sh > run_tau_7.sh
53          sed s/'CL\[5\]'/"# CL[5]"/ run_tau_7.sh > run_tau_8.sh
54          sed s/'CL\[6\]'/"# CL[6]"/ run_tau_8.sh > run_tau_9.sh
55          sed s/'CL\[7\]'/"# CL[7]"/ run_tau_9.sh > run_tau_10.sh
56          sed s/'CL\[8\]'/"# CL[8]"/ run_tau_10.sh > run_tau_11.sh
57          sed s/'CL\[9\]'/"# CL[9]"/ run_tau_11.sh > run_tau_12.sh
58          sed s/'CL\[10\]'/"# CL[10]"/ run_tau_12.sh > run_tau_13.sh
59          sed s/'CL\[11\]'/"# CL[11]"/ run_tau_13.sh > run_tau_14.sh
60          sed s/'CL\[12\]'/"# CL[12]"/ run_tau_14.sh > run_tau_15.sh
61          sed s/'CL\[13\]'/"# CL[13]"/ run_tau_15.sh > run_tau_16.sh
62          sed s/'CL\[14\]'/"# CL[14]"/ run_tau_16.sh > run_tau_17.sh
63          sed s/'CL\[15\]'/"# CL[15]"/ run_tau_17.sh > run_tau_18.sh
64          sed s/'CL\[16\]'/"# CL[16]"/ run_tau_18.sh > run_tau_19.sh
65          sed s/'CL\[17\]'/"# CL[17]"/ run_tau_19.sh > run_tau_20.sh
66
67          sed s/"..\/..\/..\/.\/gNetVarC"/"..\/.\/gNetVarC"/ \
68              run_tau_20.sh > run_tau_21.sh
69
70          sed s/'for cload in'/"for cload in {0..${len}} # for \
71              cload in"/ run_tau_21.sh > run_tau_22.sh
72
73          sed s/'a=$\[$cload+1\]'/"#a=$\[\$cload+1\]"/ \
74              run_tau_22.sh > run_tau_23.sh
75          sed s/'\${CL\[\$a\]}'/"\${CL1\[\$cload\]}"/ \
76              run_tau_23.sh > run_tau_24.sh
77
```

```
78            sed s/'\/bin\/bash'/"\/bin\/bash\n\ng++ \
79                calcolaTAU*.cpp -o calcolaTAU\ng++ gNetVarC.cpp \
80                -o gNetVarC\ng++ gNetVarTR_X.cpp -o gNetVarTR_X"/ \
81                run_tau_24.sh > run_tau.sh
82
83            rm run_tau_*.sh
84
85            #rm run_tau2.sh
86            chmod +x run_tau.sh
87
88            cd ${path_}/${i}
89          fi
90        done
91        cd ${path_}
92      fi
93    done
94
95  else
96    echo "..."
97    exit
98  fi
99
100 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 mixed-mode 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
       l                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 100nm:

```
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".

```
1  Title: basic_inverter
2  Date: Wed Jun 27 10:50:00  2012
3  Plotname: Transient Analysis
4  Flags: real
5  No. Variables: 3
6  No. Points: 10020
7  Variables:
8    0 time  time
9    1 nodea voltage
10   2 nodez voltage
11 Values:
12 0         0.000000000000000e+00
13           0.000000000000000e+00
14           9.998529155265871e-01
15
16 1         1.000000000000000e-15
17           0.000000000000000e+00
18           9.998529155265875e-01
19 ...
20 10019     1.000000000000000e-09
```

```
21          0.000000000000000e+00
22          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 (trig-targ) 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:

```
1 .meas tran delay_LH trig v(nodea) val=0.5 fall=1 \
2                     targ v(nodeZ) val=0.5 rise=1
3 .meas tran delay_HL trig v(nodea) val=0.5 rise=1 \
4                     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. 0.5V, since Vdd is 1V) in the first falling edge and V(nodeZ) reaching 50% of Vdd for the first rising edge (that is, by definition,

t_{LH} delay).Similarly, the second line measures the time between V(nodea) reaching 50% of V_{dd} in the first rising edge and V(nodeZ) reaching 50% of V_{dd} for the first falling edge (i.e. t_{HL} 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.

```
1 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=45n
.PARAM Wmin=45n
.PARAM XX=1
.PARAM tr=10p

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

Mn1 nodeZ nodeIN 0 0 nmos W={XX*Wmin} L={Lmin}
Mp1 nodeZ nodeIN node1 node1 pmos W={XX*1*Wmin} L={Lmin}

Cout nodeZ 0 3f

Vin nodeIN 0 PWL(0 0 10p 0 {10p+tr} 1 200p 1 {200p+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


```

```
14 ********************************************************************
15 * not
16 ********************************************************************
17 * Wmin, Lmin and MOS model must be declarate before
18 *
19 *drive strength ----------+
20 *inA ------------------+  /
21 *uscita ---------------+ /  /
22 *vdd ----------------+ / /  /
23 *massa ----------+ / / /  /
24 *               | | | |  |
25 *              v v v v  v
26 .subckt NOT_SUB 1 2 3 4 XX=1
27 Mn1 3 4 1 1 nmos W={1*XX*Wmin} L={Lmin}
28 Mp1 3 4 2 2 pmos W={2*XX*Wmin} L={Lmin}
29 .ends
30 ********************************************************************
31
32 ********************************************************************
33 * nand2
34 ********************************************************************
35 * Wmin, Lmin and MOS model must be declarate before
36 *
37 *drive strength ---------------+
38 *inB ----mos uscita--------+ /
39 *inA ----mos massa--------+ / /
40 *uscita ------------------+ / / /
41 *vdd --------------------+ / / / /
42 *massa ----------------+ / / / / /
43 *                     | | | | | |
44 *                    v v v v v v
45 .subckt NAND2_SUB 1 2 3 4 5 XX=1
46 Mn1 6 4 1 1 nmos W={2*XX*Wmin} L={Lmin}
47 Mn2 3 5 6 1 nmos W={2*XX*Wmin} L={Lmin}
48 Mp1 3 4 2 2 pmos W={2*XX*Wmin} L={Lmin}
49 Mp2 3 5 2 2 pmos W={2*XX*Wmin} L={Lmin}
50 .ends
51 ********************************************************************
52
53 ********************************************************************
54 * nand3
55 ********************************************************************
56 *drive strength-----------------+
57 *inC--------------------------+ /
58 *inB ------------------------+ / /
59 *inA --------------------+ / / /
60 *output ----------------+ / / / /
61 *vdd ------------------+ / / / / /
62 *ground------------+ / / / / / /
63 *                 | | | | | | |
64 *                v v v v v v v
65 .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 7 1 nmos W={3*XX*Wmin} L={Lmin}
Mn3 3 6 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 pmos W={2*XX*Wmin} L={Lmin}
.ends
*********************************************************************

*********************************************************************
* nand4
*********************************************************************
*drive strengh-------------------+
*inD-----------------------------+ /
*inC-----------------------------+ / /
*inB ----------------------------+ / / /
*inA ----------------------------+ / / / /
*output -------------------------+ / / / / /
*vdd ----------------------------+ / / / / / /
*ground-------------+ / / / / / / /
*                   | | | | | | | |
*                   v v v v v v v v
.subckt NAND4_SUB 1 2 3 4 5 6 7 XX=1
Mn1 8 4 1 1   nmos W={4*XX*Wmin} L={Lmin}
Mn2 9 5 8 1   nmos W={4*XX*Wmin} L={Lmin}
Mn3 10 6 9 1 nmos W={4*XX*Wmin} L={Lmin}
Mn4 3 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 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mp3 3 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 ----------------------+ / / / /
*massa ------------------+ / / / / /
*                        | | | | | |
*                        v v v v v v
.subckt NOR2_SUB 1 2 3 4 5 XX=1
Mn1 3 4 1 1 nmos W={1*XX*Wmin} L={Lmin}
```

```
118 Mn2 3 5 1 1 nmos W={1*XX*Wmin} L={Lmin}
119 Mp1 6 4 2 2 pmos W={4*XX*Wmin} L={Lmin}
120 Mp2 3 5 6 2 pmos W={4*XX*Wmin} L={Lmin}
121 .ends
122 ********************************************************************************
123
124 ********************************************************************************
125 * nor3
126 ********************************************************************************
127 * Wmin, Lmin and MOS model must be declarate before
128 *
129 *drive strenght-------------+
130 *inC--- ------------------+ /
131 *inB ------------------+ / /
132 *inA ------------------+ / / /
133 *output -------------+ / / / /
134 *vdd ----------------+ / / / / /
135 *ground-------------+ / / / / / /
136 *                   | | | | | | |
137 *                   v v v v v v v
138 .subckt NOR3_SUB 1 2 3 4 5 6 XX=1
139
140 Mn1 3 4 1 1 nmos W={1*XX*Wmin} L={Lmin}
141 Mn2 3 5 1 1 nmos W={1*XX*Wmin} L={Lmin}
142 Mn3 3 6 1 1 nmos W={1*XX*Wmin} L={Lmin}
143 Mp1 7 4 2 2 pmos W={6*XX*Wmin} L={Lmin}
144 Mp2 8 5 7 2 pmos W={6*XX*Wmin} L={Lmin}
145 Mp3 3 6 8 2 pmos W={6*XX*Wmin} L={Lmin}
146
147 .ends
148 ********************************************************************************
149
150 ********************************************************************************
151 * nor4
152 ********************************************************************************
153 * Wmin, Lmin and MOS model must be declarate before
154 *
155 *drive strenght-------------------+
156 *in4-----------------------------+    /
157 *in3--- ------------------------+ /    /
158 *in2 ------------------------+ / /    /
159 *in1 ------------------------+ / / /    /
160 *output -------------------+ / / / /    /
161 *vdd ----------------------+ / / / / /    /
162 *ground-------------------+ / / / / / /    /
163 *                         | | | | | | |    |
164 *                         v v v v v v v    v
165 .subckt NOR4_SUB 1 2 3 4 5 6 7  XX=1
166 Mn1 3 4 1 1 nmos W={1*XX*Wmin} L={Lmin}
167 Mn2 3 5 1 1 nmos W={1*XX*Wmin} L={Lmin}
168 Mn3 3 6 1 1 nmos W={1*XX*Wmin} L={Lmin}
169 Mn4 3 7 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 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 ********************************************************************************
177
178 ********************************************************************************
179 * xor2
180 ********************************************************************************
181 * Wmin, Lmin and MOS model must be declarate before
182 *
183 *drive strength ---------------+
184 *inB ------------------------+    |
185 *inA ---------------------+ |    |
186 *uscita ----------------+ | |    |
187 *vdd -----------------+ | | |    |
188 *massa ------------+ | | | |    |
189 *                  | | | | |    |
190 *                  v v v v v    v
191 .subckt XOR2_SUB 1 2 3 4 5  XX=1
192 *NOT
193 Mp5 node5n 5 2 2 pmos L={Lmin} W={2*XX*Wmin}
194 Mn5 node5n 5 1 1 nmos L={Lmin} W={XX*Wmin}
195 Mp6 node4n 4 2 2 pmos L={Lmin} W={2*XX*Wmin}
196 Mn6 node4n 4 1 1 nmos L={Lmin} W={XX*Wmin}
197
198 *PULL UP
199 Mp1 3       4      int4   2 pmos L={Lmin}  W={4*XX*Wmin}
200 Mp2 3     node4n   int1   2 pmos L={Lmin}  W={4*XX*Wmin}
201 Mp3 int4 node5n     2     2 pmos L={Lmin}  W={4*XX*Wmin}
202 Mp4 int1    5       2     2 pmos L={Lmin}  W={4*XX*Wmin}
203
204 *PULL DOWN
205 Mn4 6 4 1 1 nmos L={Lmin} W={2*XX*Wmin}
206 Mn3 3 5 6 1 nmos L={Lmin} W={2*XX*Wmin}
207 Mn2 int3 node4n 1 1 nmos L={Lmin} W={2*XX*Wmin}
208 Mn1 3 node5n int3 1 nmos L={Lmin} W={2*XX*Wmin}
209 .ends
210 ********************************************************************************
211
212 ********************************************************************************
213 * and2
214 ********************************************************************************
215 * Wmin, Lmin and MOS model must be declarate before
216 *
217 *drive strength---------------+
218 *inB ----mos uscita--------+   |
219 *inA ----mos massa--------+ |   |
220 *uscita -----------------+ | |   |
221 *vdd ------------------+ | | |   |
```

```
222 *massa ----------+ / / / / /
223 *                | | | | | |
224 *                v v v v v v
225 .subckt AND2_SUB  1 2 3 4 5  XX=1
226 Mn1 6 4 1 1 nmos W={2*XX*Wmin} L={Lmin}
227 Mn2 7 5 6 1 nmos W={2*XX*Wmin} L={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} L={Lmin}
233
234 .ends
235 ********************************************************************************
236
237 ********************************************************************************
238 * and3
239 ********************************************************************************
240 * Wmin, Lmin and MOS model must be declarate before
241 *
242 *drive strength----------------+
243 *inC---------------------------+ /
244 *inB ----mos output------------+ | /
245 *inA ----mos ground------------+ | | /
246 *output ----------------------+ | | | /
247 *vdd -------------------------+ | | | | /
248 *groung-----------------------+ | | | | | /
249 *                | | | | | | |
250 *                v v v v v v v
251 .subckt AND3_SUB  1 2 3 4 5 6  XX=1
252
253 Mn1 7 4 1 1 nmos W={3*XX*Wmin} L={Lmin}
254 Mn2 8 5 7 1 nmos W={3*XX*Wmin} L={Lmin}
255 Mn3 9 6 8 1 nmos W={3*XX*Wmin} L={Lmin}
256 Mp1 9 4 2 2 pmos W={2*XX*Wmin} L={Lmin}
257 Mp2 9 5 2 2 pmos W={2*XX*Wmin} L={Lmin}
258 Mp3 9 6 2 2 pmos W={2*XX*Wmin} L={Lmin}
259
260 Mp4 3 9 2 2 pmos W={2*XX*Wmin} L={Lmin}
261 Mn4 3 9 1 1 nmos W={1*XX*Wmin} L={Lmin}
262
263 .ends
264 ********************************************************************************
265
266 ********************************************************************************
267 * and4
268 ********************************************************************************
269 * Wmin, Lmin and MOS model must be declarate before
270 *
271 *drive strength----------------+
272 *inD---------------------------+ /
273 *inC---------------------------+ | /
```

```
*inB ----mos output--------+ / /   /
*inA ----mos ground--------+ / / /   /
*output ------------------+ / / / /   /
*vdd ---------------------+ / / / / /   /
*groung-------------------+ / / / / / /   /
*                         | | | | | | |   |
*                        v v v v v v v v
.subckt AND4_SUB  1 2 3 4 5 6 7 XX=1

Mn1 10 4 1 1 nmos W={4*XX*Wmin} L={Lmin}
Mn2 9 5 10 1 nmos W={4*XX*Wmin} L={Lmin}
Mn3 8 6 9 1 nmos W={4*XX*Wmin} L={Lmin}
Mn4 11 7 8 1 nmos W={4*XX*Wmin} L={Lmin}

Mp1 11 4 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mp2 11 5 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mp3 11 6 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mp4 11 7 2 2 pmos W={2*XX*Wmin} L={Lmin}

Mp5 3 11 2 2 pmos W={2*XX*Wmin} L={Lmin}
Mn5 3 11 1 1 nmos W={1*XX*Wmin} L={Lmin}

.ends
*********************************************************************

*********************************************************************
* mux21
*********************************************************************
*drive strength-----------------------------------------+
*inC-(nodeS0)-------------------------------------+      /
*inB-(nodeD1)---------------------------+         /      /
*inA-(nodeD0)-----------------------+    /      /      /
*output-----------------------+     /    /      /      /
*vdd ------------------+      /     /    /      /      /
*ground----------+     /      /     /    /      /      /
*                |     |      |     |    |      |      |
*               v     v      v     v    v      v      v
.subckt mux21_SUB 0 node1 nodez nodea nodeb nodec   XX=1

*PASS TRANSISTOR
Mn4 nodea nodecn nodeu 0 nmos W={2*XX*Wmin} L={Lmin}
Mn5 nodeb nodec nodeu 0 nmos W={2*XX*Wmin} L={Lmin}
Mp4 nodea nodec nodeu node1 pmos W={2*XX*Wmin} L={Lmin}
Mp5 nodeb nodecn nodeu node1 pmos W={2*XX*Wmin} L={Lmin}

*FINAL INVERTER
Mn3 nodez nodeun 0 0 nmos W={XX*Wmin} L={Lmin}
Mp3 nodez nodeun node1 node1 pmos W={2*XX*Wmin} L={Lmin}
Mn2 nodeun nodeu 0 0 nmos W={XX*Wmin} L={Lmin}
Mp2 nodeun nodeu node1 node1 pmos W={2*XX*Wmin} L={Lmin}

*NOT
```

```
Mp1 nodecn nodec node1 node1 pmos W={2*XX*Wmin} L={Lmin}
Mn1 nodecn nodec 0 0 nmos W={XX*Wmin} L={Lmin}
.ends

********************************************************************************

********************************************************************************
* mux31
********************************************************************************
*drive strength-------------------------------------------------------------------+
*in5-(nodeS1)---------------------------------------------------------+          |
*in4-(nodeS0)------------------------------------------------+        |          |
*in3-(nodeD2)-------------------------------------+          |        |          |
*in2-(nodeD1)--------------------------+          |        |        |          |
*in1-(nodeD0)-----------------+         |        |        |        |          |
*output-------------+         |        |        |        |        |          |
*vdd ---------+      |        |        |        |        |        |          |
*ground------+       |        |        |        |        |        |          |
*           |       |        |        |        |        |        |          |
*           v       v        v        v        v        v        v          v
.subckt mux31_SUB 0 node1 nodeZ nodeA nodeB nodeC nodeD nodeE   XX=1

Mp8 nodeU1 nodee nodeU2 node1 pmos W={3*XX*Wmin} L={Lmin}
Mp3 nodea noded nodeU1 node1 pmos W={3*XX*Wmin} L={Lmin}
Mp2 nodeeN nodee node1 node1 pmos W={2*XX*Wmin} L={Lmin}
Mp1 nodedN noded node1 node1 pmos W={2*XX*Wmin} L={Lmin}
Mp4 nodeb nodedN nodeU1 node1 pmos W={3*XX*Wmin} L={Lmin}
Mn3 nodea nodedN nodeU1 0 nmos W={3*XX*Wmin} L={Lmin}
Mn8 nodeU1 nodeeN nodeU2 0 nmos W={3*XX*Wmin} L={Lmin}
Mn4 nodeb noded nodeU1 0 nmos W={3*XX*Wmin} L={Lmin}

Mn7 nodez 1 0 0 nmos W={XX*Wmin} L={Lmin}
Mp7 nodez 1 node1 node1 pmos W={2*XX*Wmin} L={Lmin}
Mn6 1 nodeU2 0 0 nmos W={XX*Wmin} L={Lmin}

Mp6 1 nodeU2 node1 node1 pmos W={2*XX*Wmin} L={Lmin}
Mp5 nodec nodeeN nodeU2 node1 pmos W={2*XX*Wmin} L={Lmin}
Mn2 nodeeN nodee 0 0 nmos W={XX*Wmin} L={Lmin}

Mn1 nodedN noded 0 0 nmos W={XX*Wmin} L={Lmin}
Mn5 nodec nodee nodeU2 0 nmos W={2*XX*Wmin} L={Lmin}

.ends
********************************************************************************

********************************************************************************
* mux41
********************************************************************************
*drive strength-------------------------------------------------------------------+
*in6-(nodeS1)---------------------------------------------------------+          |
*in5-(nodeS0)------------------------------------------------+        |          |
*in4-(nodeD3)-------------------------------------+          |        |          |
```

```
*in3-(nodeD2)-----------------------------------+       /       /       /       /
*in2-(nodeD1)-------------------------------+       /       /       /       /       /
*in1-(nodeD0)---------------------------+       /       /       /       /       /       /
*output-------------------------+       /       /       /       /       /       /       /
*vdd ---------------------+       /       /       /       /       /       /       /       /
*ground-------------+    /       /       /       /       /       /       /       /       /
*                   |    |       |       |       |       |       |       |       |       |
*                   v    v       v       v       v       v       v       v       v       v
.subckt mux41_SUB 0 node1 nodeZ nodeA nodeB nodeC nodeD nodeE nodeF   XX=1

Mn1 nodeeN nodee 0 0 nmos W={XX*Wmin} L={Lmin}
Mp1 nodeeN nodee node1 node1 pmos W={2*XX*Wmin} L={Lmin}
Mn2 nodefN nodef 0 0 nmos W={XX*Wmin} L={Lmin}
Mp2 nodefN nodef node1 node1 pmos W={2*XX*Wmin} L={Lmin}
Mp3 nodea  nodee nodeU1 node1 pmos W={3*XX*Wmin} L={Lmin}
Mn3 nodea  nodeeN nodeU1 0 nmos W={3*XX*Wmin} L={Lmin}
Mn4 nodeb  nodee nodeU1 0 nmos W={3*XX*Wmin} L={Lmin}
Mp4 nodeb  nodeeN nodeU1 node1 pmos W={3*XX*Wmin} L={Lmin}
Mp5 nodec  nodee nodeU2 node1 pmos W={3*XX*Wmin} L={Lmin}
Mn5 nodec  nodeeN nodeU2 0 nmos W={3*XX*Wmin} L={Lmin}
Mp6 noded  nodeeN nodeU2 node1 pmos W={3*XX*Wmin} L={Lmin}
Mn6 noded  nodee nodeU2 0 nmos W={3*XX*Wmin} L={Lmin}
Mp7 nodeU1  nodef nodez node1 pmos W={3*XX*Wmin} L={Lmin}
Mn7 nodeU1  nodefN nodeU3 0 nmos W={3*XX*Wmin} L={Lmin}
Mp8 nodeU2  nodefN nodez node1 pmos W={3*XX*Wmin} L={Lmin}
Mn8 nodeU2  nodef nodeU3 0 nmos W={3*XX*Wmin} L={Lmin}

Mp9 1 nodeU3 node1 node1 pmos W={2*XX*Wmin} L={Lmin}
Mn9 1 nodeU3 0 0 nmos W={XX*Wmin} L={Lmin}

Mn10 nodez 1 0 0 nmos W={XX*Wmin} L={Lmin}
Mp10 nodez 1 node1 node1 pmos W={2*XX*Wmin} L={Lmin}
.ends
*********************************************************************

*********************************************************************
* DLatch
*********************************************************************
*drive strength-----------------------------+
*inB ---(like i/p)-nodeD-------------------+       /
*inA ---(like clk)-nodeG-----------+       /       /
*output-------------------------+       /       /       /
*vdd ---------------------+       /       /       /       /
*ground-------------+    /       /       /       /       /
*                   |    |       |       |       |       |
*                   v    v       v       v       v       v
.subckt DLatch_SUB 0 node1 nodeZ nodea nodeb XX=1

Mp41 nodeb nodeaN node21 node1          pmos W={2*XX*Wmin} L={Lmin}
Mn4 nodeb nodea node21 0                nmos W={2*XX*Wmin} L={Lmin}
Mn3 nodez 1 0 0                         nmos W={XX*Wmin} L={Lmin}
Mp3 nodez 1 node1 node1                 pmos W={2*XX*Wmin} L={Lmin}
```

```
430
431 *
432 Mn1 node21 1 0 0                                nmos W={XX*Wmin} L={2*Lmin}
433 Mp1 node21 1 node1 node1                        pmos W={XX*Wmin} L={2*Lmin}
434 Mn2 1 node21 0 0                                nmos W={1*XX*Wmin} L={Lmin}
435 Mp2 1 node21 node1 node1                        pmos W={2*XX*Wmin} L={Lmin}
436
437 *notG
438 Mp5 nodeaN nodea node1 node1                    pmos W={2*XX*Wmin} L={Lmin}
439 Mn5 nodeaN nodea 0 0                            nmos W={XX*Wmin} L={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----------+ |
450 *inB ----mos Vdd-gnd----------+ | |
451 *inA ----mos Vdd-out----------+ | | |
452 *uscita ----------------------+ | | | |
453 *vdd -------------------------+ | | | | |
454 *massa ----------------------+ | | | | | |
455 *                             | | | | | | |
456 *                             v v v v v v v
457 .subckt AO12_SUB  1 2 3 4 5 6 XX=1
458 Mn1 9 4 8 1 nmos W={2*XX*Wmin} L={Lmin}
459 Mn2 8 5 1 1 nmos W={2*XX*Wmin} L={Lmin}
460 Mn3 9 6 1 1 nmos W={1*XX*Wmin} L={Lmin}
461
462 Mp1 7 4 2 2 pmos W={4*XX*Wmin} L={Lmin}
463 Mp2 7 5 2 2 pmos W={4*XX*Wmin} L={Lmin}
464 Mp3 9 6 7 2 pmos W={4*XX*Wmin} L={Lmin}
465
466 Mn4 3 9 1 1 nmos W={XX*Wmin} L={Lmin}
467 Mp4 3 9 2 2 pmos W={2*XX*Wmin} L={Lmin}
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----------+ |
478 *inC ----mos Vdd-out----------+ | |
479 *inB ----mos out-gnd----------+ | | |
480 *inA ----mos out-out----------+ | | | |
481 *uscita ----------------------+ | | | | |
```

```
*vdd ----------------+ / / / / / /
*massa --------------+ / / / / / / /
*                    | | | | | | | |
*                    v v v v v v v v
.subckt AO22_SUB  1 2 3 4 5 6 7 XX=1
*    D   G   S   B   Name W L
*PULL DOWN
Mn1 11  4   9   1   nmos W={2*XX*Wmin} L={Lmin}
Mn2 9   5   1   1   nmos W={2*XX*Wmin} L={Lmin}

Mn3 11  6   10  1   nmos W={2*XX*Wmin} L={Lmin}
Mn4 10  7   1   1   nmos W={2*XX*Wmin} L={Lmin}

*PULL UP
Mp1 11  4   8   2   pmos w={4*XX*Wmin} L={Lmin}
Mp2 11  5   8   2   pmos W={4*XX*Wmin} L={Lmin}

Mp3 8   6   2   2   pmos W={4*XX*Wmin} L={Lmin}
Mp4 8   7   2   2   pmos W={4*XX*Wmin} L={Lmin}

*USCITA
Mn5 3   11  1   1   nmos W={Wmin} L={Lmin}
Mp5 3   11  2   2   pmos W={2*Wmin} L={Lmin}

.ends
********************************************************************************

********************************************************************************
* AO31
********************************************************************************
* Wmin, Lmin and MOS model must be declarate before
*
*drive strength----------------------+
*inD ----mos out-out-----------------+ /
*inC ----mos Vdd-gnd-----------------+ / /
*inB ----mos Vdd-mid,mid-------------+ / / /
*inA ----mos Vdd-out-----------------+ / / / /
*uscita ----------------------------+ / / / / /
*vdd -------------------------------+ / / / / / /
*massa -----------------------------+ / / / / / / /
*                                    | | | | | | | |
*                                    v v v v v v v v
.subckt AO31_SUB          1 2 3 4 5 6 7 XX=1
*    D   G   S   B   Name W L

Mn1 9   4   10  1   nmos W={3*XX*Wmin} L={Lmin}
Mn2 10  5   11  1   nmos W={3*XX*Wmin} L={Lmin}
Mn3 11  6   1   1   nmos W={3*XX*Wmin} L={Lmin}
Mn4 9   7   1   1   nmos W={1*XX*Wmin} L={Lmin}

Mp1 8   4   2   2   pmos W={4*XX*Wmin} L={Lmin}
Mp2 8   5   2   2   pmos W={4*XX*Wmin} L={Lmin}
```

```
534 Mp3 8   6   2   2   pmos W={4*XX*Wmin} L={Lmin}
535 Mp4 9   7   8   2   pmos W={4*XX*Wmin} L={Lmin}
536
537 Mn5 3   9   1   1   nmos W={1*XX*Wmin} L={Lmin}
538 Mp5 3   9   2   2   pmos W={2*XX*Wmin} L={Lmin}
539 .ends
540 ********************************************************************************
541
542 ********************************************************************************
543 * AO32
544 ********************************************************************************
545 * Wmin, Lmin and MOS model must be declarate before
546 *
547 *drive strength-------------------------+
548 *inE ----mos out-gnd--------------------+  |
549 *inD ----mos out-out-------------------+ |  |
550 *inC ----mos Vdd-gnd------------------+ | |  |
551 *inB ----mos Vdd-mid,mid-------------+ | | |  |
552 *inA ----mos Vdd-out-----------------+ | | | |  |
553 *uscita ----------------------------+ | | | | | |
554 *vdd -------------------------------+ | | | | | | |
555 *massa ----------------------------+ | | | | | | | |
556 *                                  | | | | | | | | |
557 *                                  v v v v v v v v  V
558 .subckt AO32_SUB                   1 2 3 4 5 6 7 8 XX=1
559 *     D   G   S   B   Name W L
560
561 Mn6   3   10  1   1   nmos W={XX*Wmin} L={Lmin}
562 Mp6   3   10  2   2   pmos W={2*XX*Wmin} L={Lmin}
563
564 Mn1   10  4   11  1   nmos W={3*XX*Wmin} L={Lmin}
565 Mn2   11  5   12  1   nmos W={3*XX*Wmin} L={Lmin}
566 Mn3   12  6   1   1   nmos W={3*XX*Wmin} L={Lmin}
567 Mn4   10  7   13  1   nmos W={2*XX*Wmin} L={Lmin}
568 Mn5   13  8   1   1   nmos W={2*XX*Wmin} L={Lmin}
569
570 Mp1   10  7   9   2   pmos W={4*XX*Wmin} L={Lmin}
571 Mp2   10  8   9   2   pmos W={4*XX*Wmin} L={Lmin}
572 Mp3   9   4   2   2   pmos W={4*XX*Wmin} L={Lmin}
573 Mp4   9   5   2   2   pmos W={4*XX*Wmin} L={Lmin}
574 Mp5   9   6   2   2   pmos W={4*XX*Wmin} L={Lmin}
575
576 .ends
577 ********************************************************************************
578
579 ********************************************************************************
580 * AO33
581 ********************************************************************************
582 * Wmin, Lmin and MOS model must be declarate before
583 *
584 *drive strength-------------------------+
585 *inF ----mos out-gnd--------------------+  |
```

```
*inE ----mos out-mid,mid----------+ /   /
*inD ----mos out-out--------------+ /  /   /
*inC ----mos Vdd-gnd--------------+ /  /  /   /
*inB ----mos Vdd-mid,mid---------+ /  /  /  /   /
*inA ----mos Vdd-out-------------+ /  /  /  /  /   /
*uscita ---------------------+ /  /  /  /  /  /   /
*vdd ------------------------+ /  /  /  /  /  /  /   /
*massa ----------------------+ /  /  /  /  /  /  /  /   /
*                            |  |  |  |  |  |  |  |  |   |
*                            v  v  v  v  v  v  v  v  v   v
.subckt AO33_SUB             1  2  3  4  5  6  7  8  9   XX=1
*    D   G   S   B   Name W L
Mn1 11  4   12  1   nmos W={3*XX*Wmin} L={Lmin}
Mn2 12  5   13  1   nmos W={3*XX*Wmin} L={Lmin}
Mn3 13  6   1   1   nmos W={3*XX*Wmin} L={Lmin}
Mn4 11  7   14  1   nmos W={3*XX*Wmin} L={Lmin}
Mn5 14  8   15  1   nmos W={3*XX*Wmin} L={Lmin}
Mn6 15  9   1   1   nmos W={3*XX*Wmin} L={Lmin}

Mp1 10  4   2   2   pmos W={4*XX*Wmin} L={Lmin}
Mp2 10  5   2   2   pmos W={4*XX*Wmin} L={Lmin}
Mp3 10  6   2   2   pmos W={4*XX*Wmin} L={Lmin}
Mp4 11  7   10  2   pmos W={4*XX*Wmin} L={Lmin}
Mp5 11  8   10  2   pmos W={4*XX*Wmin} L={Lmin}
Mp6 11  9   10  2   pmos W={4*XX*Wmin} L={Lmin}

Mn7 3   11  1   1   nmos W={XX*Wmin} L={Lmin}
Mp7 3   11  2   2   pmos W={2*XX*Wmin} L={Lmin}
.ends
********************************************************************************

********************************************************************************
* AO112
********************************************************************************
* Wmin, Lmin and MOS model must be declarate before
*
*drive strength-------------------+
*inD ----mos out-out--------------+ /
*inC ----mos mid,mid-gnd----------+ /  /
*inB ----mos Vdd-gnd------------+ /  /  /
*inA ----mos Vdd-out------------+ /  /  /  /
*uscita ---------------------+ /  /  /  /  /
*vdd ------------------------+ /  /  /  /  /  /
*massa ----------------------+ /  /  /  /  /  /  /
*                            |  |  |  |  |  |  |  |
*                            v  v  v  v  v  v  v  v
.subckt AO112_SUB            1  2  3  4  5  6  7  XX=1
*    D   G   S   B   Name W L

Mn1 10  4   11  1   nmos W={2*XX*Wmin} L={Lmin}
Mn2 11  5   1   1   nmos W={2*XX*Wmin} L={Lmin}
Mn3 10  6   1   1   nmos W={1*XX*Wmin} L={Lmin}
```

```
638 Mn4 10 7  1  1  nmos W={1*XX*Wmin} L={Lmin}
639
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  2  pmos W={6*XX*Wmin} L={Lmin}
643 Mp4 10 7  9  2  pmos W={6*XX*Wmin} L={Lmin}
644
645 Mn5 3  10 1  1  nmos W={XX*Wmin} L={Lmin}
646 Mp5 3  10 2  2  pmos W={2*XX*Wmin} L={Lmin}
647 .ends
648 ********************************************************************************
649
650 ********************************************************************************
651 * AO212
652 ********************************************************************************
653 * Wmin, Lmin and MOS model must be declarate before
654 *
655 *drive strength-------------------------+
656 *inE ----mos out-gnd------------------+ |
657 *inD ----mos Vdd-gnd------------------+ | |
658 *inC ----mos Vdd-out------------------+ | | |
659 *inB ----mos mid,mid-gnd--------+ | | | |
660 *inA ----mos Vdd-out------------+ | | | | |
661 *uscita ----------------------+ | | | | | | |
662 *vdd -------------------------+ | | | | | | | |
663 *massa -----------------------+ | | | | | | | | |
664 *                              | | | | | | | | |
665 *                              v v v v v v v v v
666 .subckt AO212_SUB              1 2 3 4 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  nmos W={2*XX*Wmin} L={Lmin}
670 Mn3 12 6  14 1  nmos W={2*XX*Wmin} L={Lmin}
671 Mn4 14 7  1  1  nmos W={2*XX*Wmin} L={Lmin}
672 Mn5 12 8  1  1  nmos W={XX*Wmin} L={Lmin}
673
674 Mp1 11 4  9  2  pmos W={6*XX*Wmin} L={Lmin}
675 Mp2 11 5  9  2  pmos W={6*XX*Wmin} L={Lmin}
676 Mp3 9  6  2  2  pmos W={6*XX*Wmin} L={Lmin}
677 Mp4 9  7  2  2  pmos W={6*XX*Wmin} L={Lmin}
678 Mp5 12 8  11 2  pmos W={6*XX*Wmin} L={Lmin}
679
680 Mn7 3  12 1  1  nmos W={XX*Wmin} L={Lmin}
681 Mp7 3  12 2  2  pmos W={2*XX*Wmin} L={Lmin}
682
683 .ends
684 ********************************************************************************
685
686 ********************************************************************************
687 * AO222
688 ********************************************************************************
689 * Wmin, Lmin and MOS model must be declarate before
```

```
690 *
691 *drive strength-------------------------------+
692 *inF ----mos Vdd-gnd-----------------------+ |
693 *inE ----mos Vdd-out-------------------+ | |
694 *inD ----mos mid,mid-gnd-------------+ | | |
695 *inC ----mos mid,mid-out-----------+ | | | |
696 *inB ----mos out-gnd-------------+ | | | | |
697 *inA ----mos out-out-----------+ | | | | | |
698 *uscita ----------------------+ | | | | | | |
699 *vdd -------------------------+ | | | | | | | |
700 *massa ---------------------+ | | | | | | | | |
701 *                           | | | | | | | | | |
702 *                           v v v v v v v v v v
703 .subckt AO222_SUB          1 2 3 4 5 6 7 8 9 XX=1
704 *    D  G  S  B  Name W L
705
706 Mn1 12 4  13 1  nmos W={2*XX*Wmin} L={Lmin}
707 Mn2 13 5  1  1  nmos W={2*XX*Wmin} L={Lmin}
708 Mn3 12 6  14 1  nmos W={2*XX*Wmin} L={Lmin}
709 Mn4 14 7  1  1  nmos W={2*XX*Wmin} L={Lmin}
710 Mn5 12 8  15 1  nmos W={2*XX*Wmin} L={Lmin}
711 Mn6 15 9  1  1  nmos W={2*XX*Wmin} L={Lmin}
712
713 Mp1 12 4  11 2  pmos W={6*XX*Wmin} L={Lmin}
714 Mp2 12 5  11 2  pmos W={6*XX*Wmin} L={Lmin}
715 Mp3 11 6  10 2  pmos W={6*XX*Wmin} L={Lmin}
716 Mp4 11 7  10 2  pmos W={6*XX*Wmin} L={Lmin}
717 Mp5 10 8  2  2  pmos W={6*XX*Wmin} L={Lmin}
718 Mp6 10 9  2  2  pmos W={6*XX*Wmin} L={Lmin}
719
720 Mn7 3  12 1  1  nmos W={XX*Wmin} L={Lmin}
721 Mp7 3  12 2  2  pmos W={2*XX*Wmin} L={Lmin}
722
723 .ends
724 ********************************************************************************
725
726 ********************************************************************************
727 * DFPQ
728 ********************************************************************************
729 *drive strength-----------------------------------------+
730 *inB ---(like i/p)-nodeD-------------------+       |
731 *inA ---(like clk)-nodeA-----------+       |       |
732 *output-------------------------+       |       |       |
733 *vdd ---------------------+       |       |       |       |
734 *ground--------------+    |       |       |       |       |
735 *                    |    |       |       |       |       |
736 *                    v    v       v       v       v       v
737 .subckt DFPQ_SUB    0 node1 nodeZ nodeCP noded XX=1
738
739 Mp41 noded nodeCP node21 node1  pmos W={2*XX*Wmin} L={Lmin}
740 Mn4  noded nodeCPn node21 0     nmos W={2*XX*Wmin} L={Lmin}
741
```

```
742 Mn5  noded5 nodeCP node51 0      nmos W={2*XX*Wmin} L={Lmin}
743 Mp51 noded5 nodeCPn node51 node1 pmos W={2*XX*Wmin} L={Lmin}
744
745 Mn11 nodeCPn nodeCP 0    0      nmos W={1*XX*Wmin} L={Lmin}
746 Mp11 nodeCPn nodeCP node1 node1 pmos W={2*XX*Wmin} L={Lmin}
747
748 Mp1 node21 noded5 node1 node1   pmos W={1*XX*Wmin} L={2*Lmin}
749 Mn1 node21 noded5 0 0           nmos W={1*XX*Wmin} L={2*Lmin}
750
751 Mp2 noded5 node21 node1 node1   pmos W={2*Wmin} L={Lmin}
752 Mn2 noded5 node21 0 0           nmos W={1*Wmin} L={Lmin}
753
754 Mp6 nodez  node51 node1 node1   pmos W={2*XX*Wmin} L={Lmin}
755 Mn6 nodez  node51 0 0           nmos W={1*XX*Wmin} L={Lmin}
756
757 Mp7 node51 nodez node1 node1    pmos W={1*XX*Wmin} L={Lmin}
758 Mn7 node51 nodez 0 0            nmos W={1*XX*Wmin} L={Lmin}
759 .ends
760 ********************************************************************************
761
762 ********************************************************************************
763 * DFPRQN
764 ********************************************************************************
765 *drive strength-----------------------------------------------+
766 *in3----(like resest)-nodeRN-------------------------+        |
767 *in2 ---(like i/p)-nodeD-----------------+           |        |
768 *in1 ---(like clk)-nodeCP-------+       |           |        |
769 *output-------------------------+       |           |        |
770 *vdd ----------------+          |       |           |        |
771 *ground-------+      |          |       |           |        |
772 *            |      |          |       |           |        |
773 *            v      v          v       v           v        v
774 .subckt DFPRQN_SUB   0 node1  nodez nodeCP noded nodeRN XX=1
775
776 *REACTION +
777 Mn8 nodez  node22 0     0     nmos W={2*XX*Wmin}   L={Lmin}
778 Mp8 nodez  node22 node1 node1 pmos W={4*XX*Wmin}   L={Lmin}
779 Mn7 node22 nodez  0     0     nmos W={1*XX*Wmin}   L={Lmin}
780 Mp7 node22 nodez  node1 node1 pmos W={1*XX*Wmin}   L={Lmin}
781 Mn2 1      node21 0     0     nmos W={2*XX*Wmin}   L={Lmin}
782 Mp2 1      node21 node1 node1 pmos W={4*XX*Wmin}   L={Lmin}
783 Mn1 node21 1    0     0     nmos W={1*XX*Wmin}   L={2*Lmin}
784 Mp1 node21 1    node1 node1 pmos W={1*XX*Wmin}   L={2*Lmin}
785
786 *TRANSMISSION GATE
787 Mp41 noded nodeCP node21 node1 pmos W={2*XX*Wmin} L={Lmin}
788 Mn4 noded nodeCPN node21 0     nmos W={2*XX*Wmin} L={Lmin}
789 Mn5 2 nodeCP node22 0          nmos W={2*XX*Wmin} L={Lmin}
790 Mp51 2 nodeCPN node22 node1    pmos W={2*XX*Wmin} L={Lmin}
791 *RESET
792 Mn21 3 nodeRN 0 0              nmos W={4*XX*Wmin} L={Lmin}
793 Mn20 2 1 3 0                   nmos W={4*XX*Wmin} L={Lmin}
```

```
794 Mp12 node22 nodeRN node1 node1 pmos W={2*XX*Wmin} L={Lmin}
795 Mp11 2 nodeRN node1 node1 pmos W={4*XX*Wmin} L={Lmin}
796 Mp10 2 1 node1 node1 pmos W={4*XX*Wmin} L={Lmin}
797 *
798 Mn3 nodeCPN nodeCP 0 0 nmos W={1*XX*Wmin} L={Lmin}
799 Mp3 nodeCPN nodeCP node1 node1 pmos W={2*XX*Wmin} L={Lmin}
800
801 .ends
802 ********************************************************************************
803
804 ********************************************************************************
805 * DFPHQ
806 ********************************************************************************
807 *drive strength-----------------------------------------------+
808 *in3----(like i/p)-nodeD-------------------------------+ |
809 *in2 ---(like clk)-nodeCP-----------------------+ | |
810 *in1 ---(like enable)-nodeE-------------+ | | |
811 *output-------------------------+ | | | |
812 *vdd ----------------------+ | | | | |
813 *ground---------------+ | | | | | |
814 * | | | | | | |
815 * v v v v v v v
816 .subckt DFPHQ_SUB 0 node1 nodeZ nodeE nodeCP nodeD XX=1
817
818 Mn6 nodeD nodeEN 1 0 nmos W={3*XX*Wmin} L={Lmin}
819 Mp9 nodeEN nodeE node1 node1 pmos W={2*XX*Wmin} L={Lmin}
820 Mn5 noded2 nodeCP node22 0 nmos W={2*XX*Wmin} L={Lmin}
821 Mp41 1 nodeCP node21 node1 pmos W={3*XX*Wmin} L={Lmin}
822 Mp8 nodeZ node22 node1 node1 pmos W={2*XX*Wmin} L={Lmin}
823 Mn4 1 nodeCPN node21 0 nmos W={3*XX*Wmin} L={Lmin}
824 Mp7 node22 nodeZ node1 node1 pmos W={1*XX*Wmin} L={Lmin}
825 Mn3 nodeCPN nodeCP 0 0 nmos W={1*XX*Wmin} L={Lmin}
826 Mp51 noded2 nodeCPN node22 node1 pmos W={2*XX*Wmin} L={Lmin}
827 Mn2 noded2 node21 0 0 nmos W={2*XX*Wmin} L={Lmin}
828 Mn1 node21 noded2 0 0 nmos W={1*XX*Wmin} L={2*Lmin}
829 Mp61 nodeD nodeE 1 node1 pmos W={3*XX*Wmin} L={Lmin}
830 Mp3 nodeCPN nodeCP node1 node1 pmos W={2*XX*Wmin} L={Lmin}
831 Mn9 nodeEN nodeE 0 0 nmos W={1*XX*Wmin} L={Lmin}
832 Mp2 noded2 node21 node1 node1 pmos W={2*XX*Wmin} L={Lmin}
833 Mn8 nodeZ node22 0 0 nmos W={1*XX*Wmin} L={Lmin}
834 Mp1 node21 noded2 node1 node1 pmos W={1*XX*Wmin} L={2*Lmin}
835 Mn7 node22 nodeZ 0 0 nmos W={1*XX*Wmin} L={Lmin}
836
837 .ends
838 ********************************************************************************
839
840 ********************************************************************************
841 * Full Adder
842 ********************************************************************************
843 *drive strength-----------------------------------------------------------+
844 *in3-----------------------nodeA-----------------------+ |
845 *in2 ----------------nodeB----------------+ | |
```

```
*in1 ----------nodeC----------+     /     /     /
*output2-(Carry)----------+     /     /     /     /
*output1-(sum)----------+     /     /     /     /
*vdd ----------+     /     /     /     /     /
*ground----------+   /     /     /     /     /     /
*                |   |     |     |     |     |     |
*                v   v     v     v     v     v     v
.subckt FA_SUB     0 node1 nodeSon nodeCon nodeC nodeB nodeA   XX=1

Mn1 nodecon     nodeb   node4   0       nmos W={2*XX*Wmin} L={Lmin}
Mp1 1           nodea   node1   node1   pmos W={4*XX*Wmin} L={Lmin}

Mn2 node4       nodea   0       0       nmos W={2*XX*Wmin} L={Lmin}
Mp2 1           nodeb   node1   node1   pmos W={4*XX*Wmin} L={Lmin}

Mn3 5           nodea   0       0       nmos W={2*XX*Wmin} L={Lmin}

Mp3 4           nodea   node1   node1   pmos W={4*XX*Wmin} L={Lmin}
Mp4  nodecon    nodeb   4       node1   pmos W={4*XX*Wmin} L={Lmin}

Mn4 5           nodeb   0       0       nmos W={2*XX*Wmin} L={Lmin}

Mp5 nodecon     nodec   1       node1   pmos W={4*XX*Wmin} L={Lmin}
Mn5 nodecon     nodec   5       0       nmos W={2*XX*Wmin} L={Lmin}

Mp6 2           nodea   node1   node1   pmos W={4*XX*Wmin} L={Lmin}
Mn6 3           nodea   0       0       nmos W={2*XX*Wmin} L={Lmin}
Mp7 2           nodeb   node1   node1   pmos W={4*XX*Wmin} L={Lmin}
Mn7 3           nodeb   0       0       nmos W={2*XX*Wmin} L={Lmin}

Mp8 2           nodec   node1   node1   pmos W={4*XX*Wmin} L={Lmin}
Mn8 3           nodec   0       0       nmos W={2*XX*Wmin} L={Lmin}

Mp9 nodeson     nodecon 2       node1   pmos W={4*XX*Wmin} L={Lmin}
Mn9 nodeson     nodecon 3       0       nmos W={2*XX*Wmin} L={Lmin}

Mp10 9          nodea   node1   node1   pmos W={6*XX*Wmin} L={Lmin}
Mn10 7          nodea   0       0       nmos W={3*XX*Wmin} L={Lmin}

Mp11 8          nodeb   9       node1   pmos W={6*XX*Wmin} L={Lmin}
Mn11 6          nodeb   7       0       nmos W={3*XX*Wmin} L={Lmin}

Mp12 nodeSon    nodec   8       node1   pmos W={6*XX*Wmin} L={Lmin}
Mn12 nodeSon    nodec   6       0       nmos W={3*XX*Wmin} L={Lmin}

*Mp13 nodeCo    nodecon node1   node1   pmos W={4*XX*Wmin} L={Lmin}
*Mn13 nodeCo    nodecon 0       0       nmos W={2*XX*Wmin} L={Lmin}

*Mp14 nodeSo    nodeSon node1   node1   pmos W={4*XX*Wmin} L={Lmin}

*Mn14 nodeSo    nodeSon 0       0       nmos W={2*XX*Wmin} L={Lmin}
.ends
```

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 Sainte-Cluque, 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].