Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-2012-06

M.Sc. Thesis

Implementing and evaluating a simplified transistor model for timing analysis of integrated circuits

Xinyue Zheng

Abstract

Static Timing Analysis (STA) is one approach to verify the timing of a digital circuit. The currently used Gate Level Model (GLM) has limitations on performing STA for circuits when taking process variations into consideration. The transistor level model is developed taking the statistical factors into account. This thesis presents an implementation of the simplified transistor model in Verilog-AMS such that the model can be installed as a compiled model in existing commercial circuit simulators, such as Spectre. A direct comparison between the proposed transistor model and the sophisticated Berkeley Short-channel IGFET Model (BSIM) is presented. Furthermore, the transistor model is extended with process variations awareness for statistical timing analysis. The polynomial curve fitting scheme is proposed in this thesis to improve the model accuracy. The evaluation results indicate that the proposed method has approximately 70% improvement in terms of estimating one of the components i.e., drain-source current I_{ds} for the statistical transistor model.

TU Delft

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Implementing and evaluating a simplified transistor model for timing analysis of integrated circuits

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Computer Engineering

by

Xinyue Zheng
born in Changchun, China

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Implementing and evaluating a simplified transistor model for timing analysis of integrated circuits"** by **Xinyue Zheng** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 28 August 2012

Chairman:

Prof. dr. ir. Edoardo Charbon

Advisor:

Dr. ir. Michel Berkelaar

Committee Members:

Prof. dr. ir. Edoardo Charbon

Dr. ir. Nick van der Meijs

Dr. ir. Michel Berkelaar

Dr. ir. Arjan van Genderen

Abstract

Static Timing Analysis (STA) is one approach to verify the timing of a digital circuit. The currently used Gate Level Model (GLM) has limitations on performing STA for circuits when taking process variations into consideration. The transistor level model is developed taking the statistical factors into account. This thesis presents an implementation of the simplified transistor model in Verilog-AMS such that the model can be installed as a compiled model in existing commercial circuit simulators, such as Spectre. A direct comparison between the proposed transistor model and the sophisticated Berkeley Short-channel IGFET Model (BSIM) is presented. Furthermore, the transistor model is extended with process variations awareness for statistical timing analysis. The polynomial curve fitting scheme is proposed in this thesis to improve the model accuracy. The evaluation results indicate that the proposed method has approximately 70% improvement in terms of estimating one of the components i.e., drain-source current I_{ds} for the statistical transistor model.

Acknowledgments

Firstly, I would like to give my utmost gratitude to my daily supervisor Dr. ir. Michel Berkelaar, who is the leader of MODERN project. He directed me on the right track of the project, and gave me many valuable advice and feedbacks. It still remains fresh in my memory that Michel helped me to contact the person from Cadence and for requesting XML scripts for Automatic Device Model Synthesizer (ADSM) tool. Additionally, with great patience, he reviewed this thesis many times and proposed large amount of suggestions. This thesis would not have been completed without those valuable advises.

Secondly, I would like to thank my other supervisor Dr. ir. Nick van der Meijs. Thanks to his feedbacks and for my thesis. In addition, I want to express my sincere appreciation to Qin Tang, a Ph.D student from MODERN project. She has supported me during the entire master project, and helped me overcome difficulties I faced during the study. The project would not have come easily without her assistance. Also, many thanks to the other group member Dr. ir. Amir Zjajo for his assistance throughout the project. What is more, I need to thank my colleague Javier Rodriguez. We spent lots of time discussing together, and he proposed many precious suggestions to improve the quality of my work.

I would like to thank Prof. dr. ir. Edoardo Charbon and Dr. ir. Arjan van Genderen too to be my thesis defence committee members, thanks for their precious time.

Moreover, I thank my dear friend Ijeoma Okeke, who pointed out the grammar problems in my thesis, which helped greatly with refining the thesis. Thank my friends Sachin and Nupur, they shared many precious opinions with me, which were quite helpful. Last but not least, I thank my parents, all my relatives and all my friends. It is with your firm support that I could get through all the difficulties of studying abroad alone. These two years mean a lot to me, and I am grateful to you all.

Xinyue Zheng
Delft, The Netherlands
28 August 2012

Contents

List of Figures

List of Tables

1 Introduction

Static Timing Analysis (STA) is an approach to evaluate the timing of a digital circuit mathematically. The timing variation of a circuit is critical since the delay of the circuit decides its maximum operating frequency. STA simplifies the design tasks by analysing the entire design once and the required timing check are performed for all the possible paths of the design [1]. Gate Level Models (GLM) have been broadly adopted in the industry as a traditional method for STA. Elaborate models for all the standard logic cells are required in GLM in order to perform STA. For instance, Non Linear Delay Model (NLDM) is a typical table-based GLM. The delay and slew of a standard gate are stored in Look Up Tables (LUTs) indexed by the input slew and the effective output load capacitance. Such a strategy abstracts the gate behaviour from a gate level and used to be very efficient and accurate for STA. However, the weakness of conventional GLM was gradually exposed as the technological trends of transistors go further to 45nm and below [2]. Firstly, the inputs of GLMs are simply regarded as saturated ramps, which can not represent the real input signals properly, especially in the presence of noise [3]. Secondly, GLMs fail to capture the multi-port coupled interconnect load due to the over-simplified output equivalent capacitance [2]. In addition, the process variations of transistors can not be ignored any more in nanometer technology. The standard gate is composed of transistors, such that the model complexity and the characterization time for gates grow explosively when the process variations for all the transistor dimensions and their correlations are taken into consideration. The research focus has recently shifted to Transistor Level Model (TLM) development due to the problems mentioned above.

1.1 The Main Problem

GLM has encountered difficulties in analysing the standard cell precisely. Over the past few decades, intensive research has been carried out with regard to improve the accuracy of models. Composite Current Source model (CCS) and Effective Current Source Model (ECSM) have been proposed to improve the accuracy of the signal waveform representation, which to some degree improves the model accuracy. Additionally, models such as the Weibull-based waveform model [4] and multi-port current source model [5] have been proposed to address the accuracy problem. However, abstracting a gate roughly according to the input waveform and output load capacitance ignores too many details inside a cell. The black-box feature of GLMs is the real obstacle for further improving the accuracy of the model [6]. In contrast, TLMs include the physical behaviour of a transistor, which fundamentally overcomes the flaw of GLMs in evaluating the timing of the circuits. The Berkeley Short-channel IGFET Model (BSIM) is a transistor model proposed by the University of California at Berkeley. BSIM4, as

the fourth version of the BSIM family, addresses the transistor's physical effects into the sub-100nm regime [7]. It is currently regarded as one of the most accurate and complex models for transistors. However, large scale circuits can be composed of millions of transistors. It is too slow to run BSIM4 for timing analysis at the transistor level. The transistor model should be simple enough to ensure the speed of the timing analysis. Efforts have been put into developing fast transistor models [8, 9]. One of the simplified TLMs is proposed by Tang from MODERN group [10]. The model is shown in Figure 1.1.

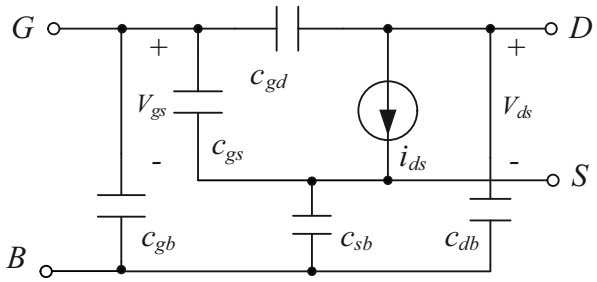


Figure 1.1: a simplified NMOS transistor model

Here we call the proposed Statistical Simplified Transistor Model SSTM. As indicated in Figure 1.1, SSTM is built up with five parasitic capacitances (C_{gd},C_{gs},C_{gb},C_{db} and C_{sb}) and the drain-source current source I_{ds}. The simple structure of the model enables statistical timing analysis for digital circuits at the transistor level. Furthermore, a new Random Differential Equations (RDE)-based simulation engine used for non-Monte Carlo (MC) statistical timing analysis is presented in paper [6]. The model combined with the new simulation-like engine can do both deterministic timing analysis and non-MC statistical timing analysis fast and accurately.

1.2 Thesis Goal

The goal of this thesis is to evaluate the accuracy of the proposed SSTM. However, SSTM and the simulation-like engine were implemented as an entire circuit simulation tool at the early development stage. The accuracy of the SSTM was not known yet since the circuit simulation errors were both from the model and the engine. In order to remove the difference caused by the simulation engine, the proposed SSTM is to be installed on a commercial circuit simulator Spectre because our golden model BSIM4 is running on Spectre. In such a way, the model can be compared with BSIM4 directly. Figure 1.2 shows the way to test the SSTM by comparing with BSIM4.

As shown in the left side of Figure 1.2, the reference sample is set up as the BSIM4 running on the simulation engine provided by Spectre. The SSTM combined with the same engine is to be tested. In such a way, the difference between the simulation results are the relative errors caused by the SSTM. In addition, SSTM is capable to carry out statistical timing analysis too. The accuracy of the statistical feature of the model considering the transistor process variations is also evaluated by comparing with BSIM4 running on the same simulation engine.

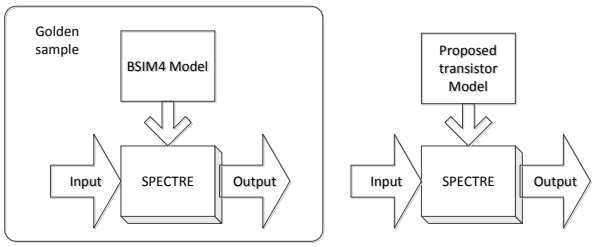


Figure 1.2: comparison on deterministic timing analysis simulation

1.3 Thesis Outline

The rest of the thesis is organized as follows

- Chapter 2 introduces the background of STA. The principle of GLM is explained with the context of Composite Current Source (CCS). The principle of the transistor level circuit simulation is also stated in this chapter.
- Chapter 3 explains the concept of the compact model and the details of the transistor model implementation. Moreover, the compiled model interface of Spectre and the possible ways of installing the new model on the simulation engine are discussed.
- Chapter 4 presents the model performance by doing deterministic timing analysis at both gate level (single logic gate) and the circuit level (ISCAS benchmark circuits).
- Chapter 5 discusses the way to extend the model with process variations awareness. The accuracy of the process variations aware model is checked through running Monte Carlo simulation for statistical timing analysis at both gate level and circuit level.
- Chapter 6 gives conclusions of the thesis and suggestions for future work

2 Literature Review

All electronic circuits experience signal delay from their inputs to outputs. The maximum frequency capability of a circuit is determined by the critical path of the circuit. The process of timing analysis is to figure out the critical path of the circuit and analyse the delay of the critical path, which is usually done by Static Timing Analysis (STA). STA is static because the timing analysis is independent of the waveform of the input signals [1]. In practice, the STA engine usually behaves as an integrated block in an Electronic Design Automation (EDA) tool. It provides the interface to other engines such as the FPGA synthesis and routing engines which require timing information.

In digital circuit design, standard timing measurements such as propagation delay and slew, represent the timing characteristics of the gate. The propagation delay represents the signal traversing time through the gate, and is calculated as the time period when the output and the input signals cross the delay threshold (commonly set as 50% of the power supply V_{dd}). The t_d in Figure 2.1 (b) shows the definition of propagation delay. Signal slew reflects the change rate of the signal, which measures the quality of the transitions [11]. The slew is defined as the period when signal crosses the slew thresholds, which were chosen as 30% and 70% of the V_{dd} in this thesis. The t_{slew} in the Figure 2.1 (c) shows the definition of the slew for a signal.

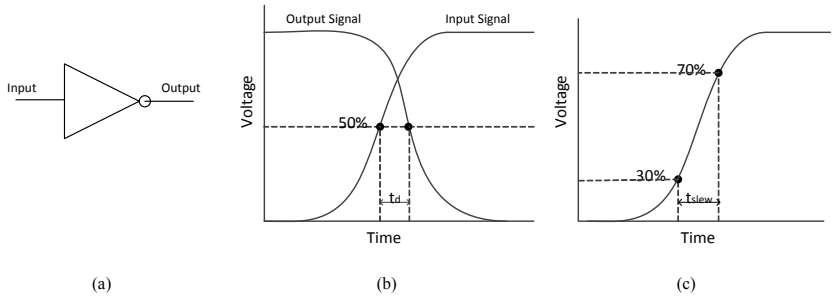


Figure 2.1: Definition of propagation delay and slew of a gate

According to the definition of the delay and the slew, equation (2.1) can be derived.

$$t_{delay} = t|_{V_{out}=50\%V_{dd}} - t|_{V_{in}=50\%V_{dd}} \tag{2.1a}$$

$$S = t|_{V_{out}=70\%V_{dd}} - t|_{V_{out}=30\%V_{dd}} \tag{2.1b}$$

Where t_{delay} and S are the delay and the output slew of a gate.

2.1 Composite Current Source (CCS) model for STA

A few decades ago, the linear transistor model was used in Gate Level Model (GLM) scheme for characterizing the standard logic gates. As a consequence of the transistor scaling, the conventional linear model was not adequate to account for more complex physical effects such as short channel effects, power supply noise etc. [12]. The table-based Non-Linear Delay Model (NLDM) was developed for modelling the new generation transistor. The propagation delay of a gate in NLDM is regarded as a function of the input slew and the output load capacitance. In order to speed up the timing analysis, the timing information of the standard cells is pre-characterized and stored in a library, which contains 2-D LUTs with input signal slew and effective output load capacitance as indexes. The input signal slew rate decides how fast the MOSFET is turned on or off, which affects the speed for conducting current through the transistor, thus influencing the arrival time of the output signal. The output load capacitance will impact the speed for charging or discharging capacitors. If we regard the gate as a RC network, the time constant τ is given as

$$\tau = R_{eq} \times C_{load} \tag{2.2}$$

where R_{eq} is the equivalent resistance of the RC network, and C_{load} is the output load capacitance. τ is the time taken for the voltage across a capacitor to fall to e^{-1} of its value. Therefore the effective output load capacitance will impact the output signal arrival time too.

The NLDM has seen wide adoption and was reasonably accurate. However, with the transistor scaling reached the deep sub-micron regime, issues such as the crosstalk effect and Multi Input Switching (MIS) restrict the accuracy of the NLDM. The crude delay approximation technique based on the input slew and the effective output loads is not sufficient for modern STA any longer. Therefore, the NLDM has been replaced gradually by the recent developed Current Source Model. Composite Current Source (CCS) is one typical current source model taking physical effects like Miller effect, high interconnect impedance and noise propagation into account [13]. The model is shown in Figure 2.2.

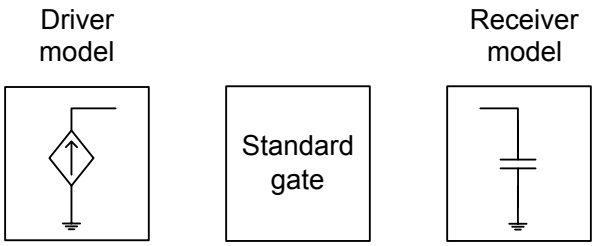


Figure 2.2: CCS model

The driver model is a non-linear current source dependent on both the input slew and the voltage, while the receiver model is the effective output load capacitor. As shown in the upper part of Figure 2.3, the driver model of the gate G2 is the receiver model of its previous stage gate G1, and the receiver model of the gate G2 is the driver

model of its next stage gate G3. An input stimulus and a load capacitor are required for the gate characterization. The procedure for the characterization is shown in the bottom of Figure 2.3. The current flow at the output pin along with the current and voltage at the input pin are measured with a combination of different input slews and output load capacitors [14].

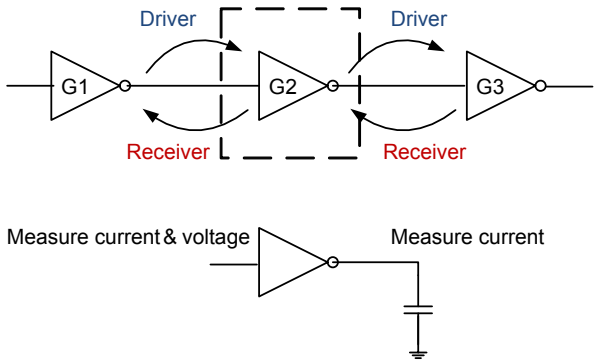


Figure 2.3: Characterization for CCS model

The LUTs for the driver model and the receiver model are shown in Figure 2.4. The two values $C1$ and $C2$ in the receiver model table are due to the fact that the effective input capacitance of the gate at the next stage is dependent on the input voltage. In order to increase the accuracy of the model, $C1$ is the equivalent capacitance before the input signal reaches the delay threshold, and $C2$ is the equivalent capacitance after the input signal reaches the threshold.

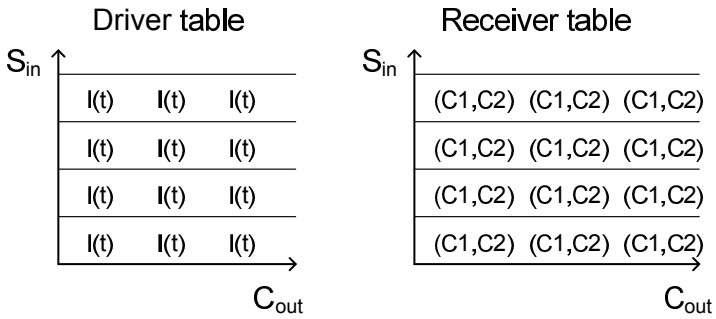


Figure 2.4: Driver and receiver model tables for CCS model

The LUT for the output current waveform of the CCS model is shown in Figure 2.5.

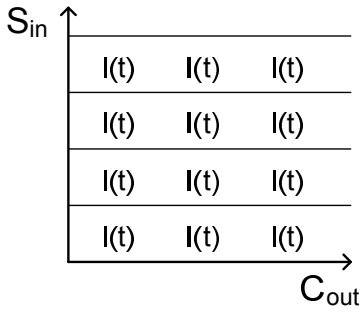


Figure 2.5: LUT for CCS model

The output current is accessed from the LUT of the CCS model. Based on the I-V relationship for the capacitor, the output voltage is expressed by Equation (2.3).

$$V(t) = \int_0^t \frac{I(t)}{C} dt \tag{2.3}$$

Where I, V, and C are the current, voltage, and equivalent capacitance for the output pin respectively.

2.2 Transistor level circuit simulation

In electronic engineering, nodal analysis is a systematic method to set up and solve the equations, such that the voltages of the nodes in a circuit can be calculated. Nodal analysis can be summarized to three main steps [15].

1. Choose ground as a reference node.

2. Name all the remaining unknown nodes.

3. Write equations based on Kirchhoff's Circuit Law (KCL) to each node not connected to a voltage source

The equations derived from the circuits are usually differential equations because charging and discharging the devices such as capacitors and inductors are time dependent. Furthermore, the circuit contains the non-linear elements such as transistors which will cause the equations to be non-linear. However, a discrete computer program does not have the ability to solve the differential equations directly. Instead, it attempts to discretise the differential equations into algebraic equations and to solve the numerical integration problem with the finite difference method. The Backward Euler, Forward Euler and Trapezoidal rule are numerical methods commonly used in computer aided analysis to solve the Ordinary Differential Equations (ODE).

An example ODE is shown in Equation (2.4)

$$\frac{d}{dt}x(t) = Ax(t) \tag{2.4}$$

The solution of Equation (2.4) is expressed by Equation (2.5)

$$x(t_{n+1}) = x(t_n) + \int_{t_n}^{t_{n+1}} Ax(\tau)d\tau \tag{2.5}$$

where

$$\int_{t_n}^{t_{n+1}} Ax(\tau)d\tau \approx \triangle t Ax(t_n) \tag{2.6}$$

$$\approx \triangle t Ax(t_{n+1}) \tag{2.7}$$

$$\approx \frac{\triangle t}{2}(Ax(t_n) + Ax(t_{n+1})) \tag{2.8}$$

The three expressions represent the ways to approximate the numerical integration with the Backward Euler, Forward Euler and Trapezoidal rule respectively. The Backward Euler and Trapezoidal rules are implicit methods since both the current state $x(t_n)$ and the future state $x(t_{n+1})$ are involved in calculating the current state. In contrast, the explicit method directly indicate how to calculate the next stage based on the current stage, which is more straightforward. However, the explicit method suffers from a stability problem caused by the equation which requires extremely small step size. The solution begins oscillating if the chosen step size is located out of the stable region. Implicit methods ensure numerical solutions to be always stable. A large step size means high efficiency for solving the equation. However, solving implicit equations sometimes can be very complicated.

The circuit simulation flow is shown in Figure 2.6 [16]. The outer loop in Figure 2.6 indicates the way to solve the differential equations. An initial solution is given at the first step, then the computer program recursively finds the solution for the ODE till the end of the simulation. The inner loop in Figure 2.6 linearises the circuit equations around the candidate solution. The Newton-Raphson method is usually applied to repeat the linearising procedure until convergence is reached.

The conventional nodal analysis has difficulty in expressing the voltage or voltage dependent source, extra equations for each voltage source are required. To overcome such problem and set up the matrices of the equations efficiently in the computer program, Modified Nodal Analysis (MNA), an enhancement of the nodal analysis approach is employed in the circuit simulator. Apart from determining the node voltages like the classical nodal analysis do, MNA has the ability to obtain the branch currents too. Additionally, all linear circuit elements are accommodated by MNA without virtually pre- and post-processing [17]. MNA has become the fundamental method for all the circuit simulators. The details of the MNA is not discussed in this thesis. Readers who are interested in MNA can read [18] for more details.

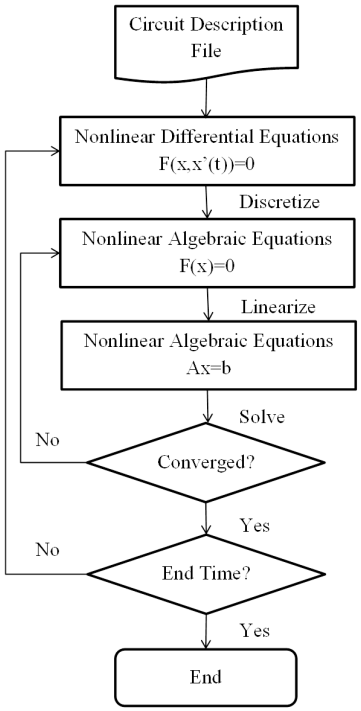


Figure 2.6: Principle of circuit simulation engine

Simplified Transistor Model Implementation

3

Computer-aided design tools such as Simulation Program with Integrated Circuit Emphasis (SPICE) have become crucial in Integrated Circuit (IC) design. SPICE is a general-purpose analogue electronic circuit simulator capable of running DC analysis, small signal analysis, nonlinear transient analysis etc. [19]. The SPICE-like simulators such as Spectre and HSPICE are becoming popular and have gradually achieved the dominance of the circuit simulators market. NGSPICE is another member of SPICE family, which is an open source simulator aimed at research field. The variety of simulators offers multiple choices for this project. Spectre was finally chosen because it is one of the most commonly used simulator in industry field. Figure 3.1 [20] shows the basic data flow approach using BSIM for circuit analysis in Spectre.

The Process File shown in Figure 3.1 is also called the model card, which defines the size-independent parameters for the transistor model such as saturation velocity, subthreshold swing factor etc. Different models have different parameters, thus different model cards. The number of parameters also depends on the complexity of the model.

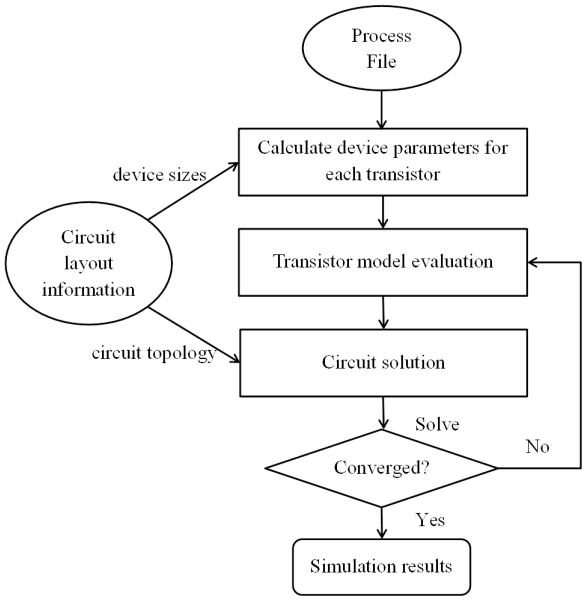


Figure 3.1: Circuit simulation flow

The particular meaning and value of the model is specified in the model user's guide (such as BSIM4 User's Guide). The next step is to determine the device parameters based on the sizes of the used transistors which need to be passed to the simulator. With the device parameters, the transistor models can be evaluated. In the end, the simulation engine (Spectre) will set up the equations based on the circuit topology with the help of Modified Nodal Analysis (MNA), and generate the simulation results when a converged solution is reached.

The complex BSIM4 model has hundreds of parameters in the process file, which requires large amount of resources (time and memory) to evaluate the model. The proposed transistor model addresses the problem by simplifying the model structure. Moreover, all elements of the model are characterized ahead and stored in LUTs. Such scheme considerably reduces the transistor model evaluation time, therefore making the circuit simulation at the transistor level practicable.

In the following sections, we will firstly introduce the concept of the compact model and its implementation, then discuss the interface of Spectre. Finally, the possible ways to install the model in Spectre will be explained.

3.1 Compact model

The model engineer would like to focus more on the model development rather than handle details of the simulation engine's interface. Actually it is not necessary for users to know how the simulator runs in the background. The modelling language targeted on model development benefits users by hiding the circuits simulator's implementation details. A circuit can be modelled at different levels. Figure 3.2 [21] shows an example of three modelling strategies and the relations between their speeds and accuracies.

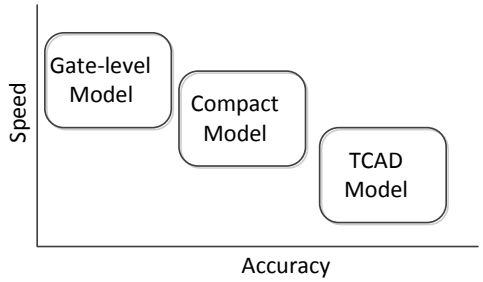


Figure 3.2: Circuit modelling strategy

As indicated by the name, Gate-level Model describes the circuit as a set of logic gates and their interconnections. A compact model like BSIM is a transistor level model used for circuit design. The transistor is described by the physically-motivated currents and voltages equations [21]. A Technology Computer Aided Design (TCAD) model dives deep into a transistor structure, taking the transistor fabrication process such as diffusion and ion implantation into consideration. As shown in Figure 3.2, the more details of the device are taken into account, the more accurate the model is. Correspondingly, the speed goes down when the complexity of the model increases.

The compact model is desirable in this thesis since the SSTM to be implemented is at transistor level. SSTM is seen as a black-box with input nodes. The voltages of all input nodes are given, and the outputs of the model are the branch currents.

The multiple ways of implementing models increase the difficulty for integrating the models to the Electronic Design Automation (EDA). A uniform compact model standardization ensures all the devices developed efficiently and fit for the simulators easily. GNU is an influential organization dedicating for free software development. According to the compact model standardization from GNU perspective, the compact model should be freedom to run and to adapt the model into users' demands [22].

3.2 Verilog-AMS

The standard language of compact modelling is C in order to achieve consistency with the language in which SPICE is currently implemented. However, Verilog-AMS (Analogue Mixed Signal) was released and promoted by the Compact Model Council aimed at coding the compact model for the next generation MOSFET models [21]. The Verilog-AMS language and its subset Verilog-A are both Hardware Description Languages (HDL) providing great support for compact device modelling. The extension of the IEEE-1364 Verilog digital HDL standard makes the modelling feasible in analogue and mixed-domain [23]. A growing number of semiconductor company and EDA vendors are involved in supporting the Verilog-AMS proposal.

The straightforward syntax makes the language of Verilog-AMS easy to master, which accelerates the model development procedure. Furthermore, Verilog-AMS provides the derivative function used for calculating time dependent components such as capacitors and inductors, thereby reducing the designers' burdens of hand coding derivatives during the model development. The uniform derivative function also guarantees the same computing precision which makes the model portable. With the merits explained above, Verilog-AMS is chosen in this thesis to implement the proposed transistor model SSTM.

3.3 Compiled Model Interface (CMI)

Spectre, as a member of SPICE family, is developed in the C programming language. The new devices after being compiled are integrated to the Spectre simulator through the C-language Compiled Model Interface (CMI). The CMI has become into a common model interface of EDA tools for customers to install their proprietary models [24]. Spectre currently supports ADMS CMI and Hardware Description Language (HDL) CMI, which will be discussed in the following sections.

In the C programming language, a library is used to archive the object codes of commonly used functions. The executable code is formed after the linker links the application object codes with the required libraries. The concept of the library provides the developers a way to release their new function as a single component to the interface. Thus the new model can be installed to the CMI as a library.

The C library is categorized as the static library and the shared object library. As

indicated by the name, the static library is linked statically at the compile time. A copy of the library is required for every program which calls the library. Thus the same piece of the codes are duplicated which turns the executable codes very cumbersome. In contrast, the shared object library is dynamically linked to the program. Instead of the contents (data and text) of the library, only the name of the library is recorded in the executable file. Although each process calling a share library will map it at a different virtual address, the same physical copy is shared by all the functions [25]. Figure 3.3 [26] explains the different concepts between the static library and the shared object library. Both program A and B in Figure 3.3 require to call library X. The library X in the static library scheme (left part of the figure) is loaded to the memory twice. While the library X in the dynamically shared object scheme (right part of the figure) is loaded to the memory once. Thus the efficiency of the memory usage is increased as well.

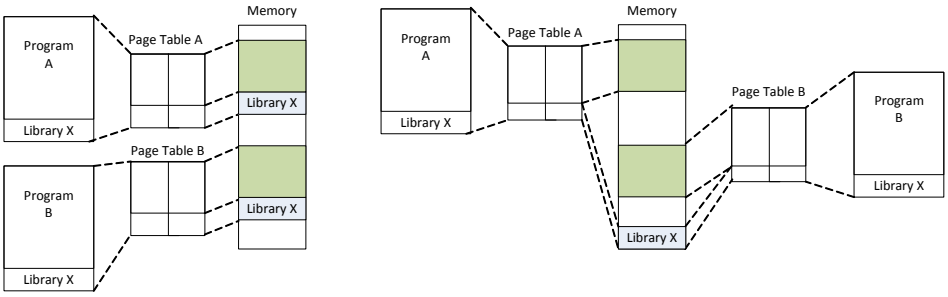


Figure 3.3: Statistical library and shared object library

Furthermore, the concept of the shared object library improves the flexibility of the interface. Since the shared object library can be resolved during the runtime, updating the library does not affect the existing programs. Particularly, updating the model does not impact the Spectre simulator. The new model is compiled as a shared object library by C compiler such as *gcc*, and to be installed through the dynamic linker.

The shared object library must start with letters *lib* and end with the suffix *.so*. According to the users guide [27], Spectre demands a set of configuration files specifying the search path of the new library. Optionally, the search path can also be indicated through the *-cmiconfig* arguments.

3.3.1 Translator from Verilog-AMS to C

As indicated in section 3.2 and section 3.3, The model is developed in the Verilog-AMS language, and being installed through the C-language CMI of Spectre. A converter is therefore necessary to translate the Verilog-AMS model source codes to the ready to compile C codes. The corresponding C codes must be compatible with the Spectre interface. Figure 3.4 demonstrates the whole procedure for installing the model on the Spectre interface.

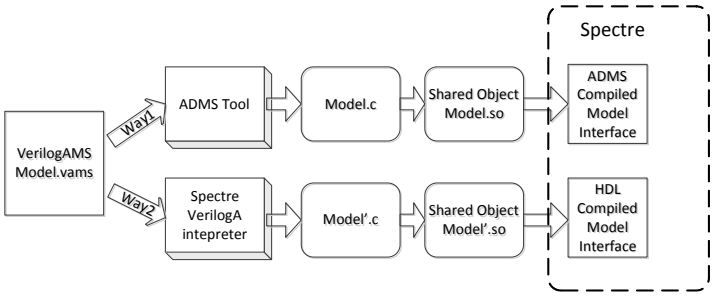


Figure 3.4: Two methods of generating C codes from Verilog-AMS

As shown in Figure 3.4, there are two translating schemes. Model.c and Model'.c are two different sets of C source codes generated by the ADMS tool and Spectre Verilog-A interpreter respectively. Both of the codes will be compiled to the shared object library Model.so and Model'.so at the next step. Finally, these shared library are installed to the corresponding Spectre CMI i.e., ADMS CMI and HDL CMI.

3.3.2 Automatic Device Model Synthesizer (ADMS) tool

The Automatic Device Model Synthesizer (ADMS) tool is a code translator which converts the compact model from Verilog-AMS into ready to compile C code based on the requirement of CMI [28]. The application for the ADMS tool is shown in Figure 3.4. The concept of how ADMS tool works is illustrated in Figure 3.5.

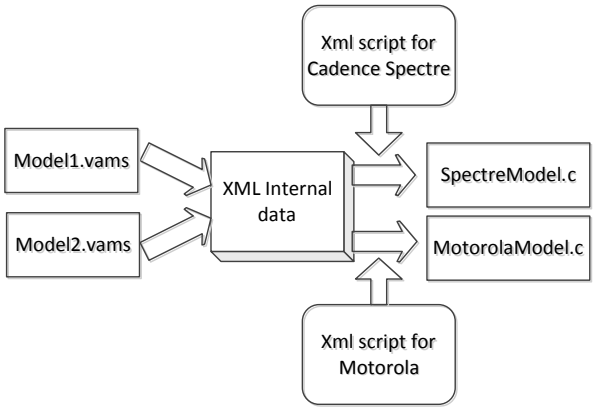


Figure 3.5: The principle of ADMS tool

Firstly, the parser turns the Verilog-AMS source codes into XML. XML stands for Extensible Markup Language, which was designed to transport and store data. Currently XML has been broadly used in the Web domain. Regardless of the information types, XML wraps the information with labels, and stores it in a tree structure referred to as an XML tree. As shown in Figure 3.5 [28], the Verilog-AMS source codes (Model1.vams or Model2.vams) are marked to satisfy the XML format, which are stored in the tree structure. Since each CMI follows a special format, different scripts such as Candence Spectre or Motorola are required to rebuild the information according the specific format requirement from the CMIs of the simulators. Thus the corresponding

C codes SpectreModel.c or MotorolaModel.c are created which match the particular interface.

Spectre provides a set of ADMS-XML scripts, indicating the details of requirements of its interface. Thus the C codes satisfying the Spectre CMI requirement can be created from ADMS tool with the help of scripts. The version MMSIM7.2 of Spectre is used in this project. The corresponding ADMS-XML scripts can be obtained through the following steps

1. A tool called cmiExtract is stored at directory ⟨your MMSIM installation directory⟩/tools/spectre/bin/cmiExtract

2. Run the tool cmiExtract to extract the CMI scripts and documentation. The default directory is ⟨your MMSIM installation directory⟩/tools/

3. The directory containing the XML scripts will be found in the sub directory ⟨your MMSIM installation directory⟩/tools/spectrecmi/adms_xml_scripts

A typical run of ADMS tool is shown as below [28]

admsXml ⟨modelfile⟩.va -e ⟨myinterface-file1⟩.xml -e ⟨myinterface-file2⟩.xml -e...

ADMS tool is capable of generating optimized codes. According to the survey in [28], the hand-coded built-in model is only 10% faster than the model automatically generated by the ADMS tool.

3.3.3 Spectre Verilog-AMS interpreter

Although ADMS tool is a very promising code translation solution, the versions between ADMS tool and ADMS-XML scripts provided by Spectre must match with each other. Due to the fact that the Spectre version used in the project is 2.2.7, while the version required from the available ADMS tool is 2.2.9 or higher, the incompatibility problems showed up while creating C codes. Therefore, the direct translation scheme i.e., a Verilog-AMS interpreter provided by Spectre is selected in the project. The Verilog-AMS interpreter checks the syntax of Verilog-AMS and generates the C codes for Spectre CMI. The model is finally installed to SpectreHDL (Hardware Description Language) CMI. Figure 3.4 shows the application for Verilog-AMS interpreter. Compared to ADMS which is suitable for any simulator, the Verilog-AMS interpreter is used explicitly for the Spectre, which is less flexible. Since the codes generated from the Verilog-AMS interpreter are highly compatible with the Spectre interface. Verilog-AMS interpreter was selected in this project.

3.4 Model Implementation

As discussed in Section 1.1, the SSTM to be implemented is shown in Figure 3.6.

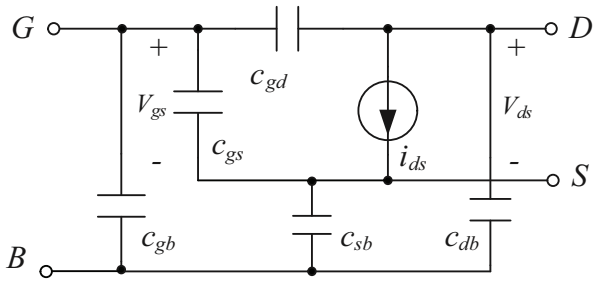


Figure 3.6: a simplified NMOS transistor model

Determining the branch contribution equations is the crucial step in terms of the model description. Based on the current-voltage (I-V) relations for the capacitor, the branch contributions of the simplified transistor model are described by Equation (3.1)

$$I_{br\_gd} = C_{gd} \times \frac{dV_{gd}}{dt} \tag{3.1a}$$

$$I_{br\_gs} = C_{gs} \times \frac{dV_{gs}}{dt} \tag{3.1b}$$

$$I_{br\_gb} = C_{gb} \times \frac{dV_{gb}}{dt} \tag{3.1c}$$

$$I_{br\_db} = C_{db} \times \frac{dV_{db}}{dt} \tag{3.1d}$$

$$I_{br\_sb} = C_{sb} \times \frac{dV_{sb}}{dt} \tag{3.1e}$$

$$I_{br\_ds} = i_{ds} \tag{3.1f}$$

Where g, d, s, and b are the four terminals of the transistor, whose voltages can be probed. The four nodes behave as the inputs to the model in the model description. The five intrinsic capacitances C_{gs}, C_{gd}, C_{gb}, C_{db} and C_{sb} are stored in five two dimensional LUTs indexed by V_{ds} and V_{gs}. I_{ds} is the source-drain current, which is more crucial for the accuracy of the model than the intrinsic capacitances. Thus I_{ds} is stored in a three dimensional LUT indexed by V_{ds}, V_{gs} and V_{sb} (V_{bs} for the PMOS transistor model).

The voltage of the power supply (V_{dd}) in the technology used in the project is $1.1V$. While the LUTs has $0.2V$ margin such that the signal ranges for V_{ds} and V_{gs} in NMOS model are from $-0.2V$ to $1.3V$. In this case, the precision is guaranteed even if the signals reach out of the boundary. The step size is $0.05V$. The index V_{sb} starts from $0V$ and ends with $1.1V$ with step size $0.1V$. In the PMOS transistor model, the index range for both V_{ds} and V_{gs} are from $-1.3V$ to $0.2V$. The index V_{bs} starts from $0V$ and ends with $1.1V$ with step size $0.1V$.

In summary, there are five 31×31 (31 grids for both V_{ds} and V_{gs} in LUT) tables for parasitic capacitors and one $31 \times 31 \times 12$ (12 grids for V_{sb} in LUT) table for I_{ds} for each type of the transistor model.

3.4.1 Loading Lookup Tables (LUTs)

The LUTs are stored as text files. However, reading data directly from files will lead to performance degradation since carrying out file operations is extremely expensive. Statically loading LUTs into codes is preferred in such case since the size and the number of LUTs are not huge. LUTs are loaded before the model is compiled, and they will behave as a 31×31 matrix to be accessed locally in the program. A set of bash scripts help with loading LUTs according to the transistor's length and width indicated by the user in a configuration file. The scheme on generating a model containing LUTs is demonstrated in Figure 3.7.

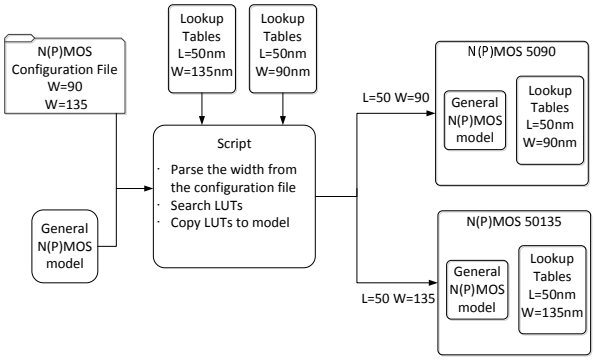


Figure 3.7: Load LUTs to model

Each complete model in Figure 3.7 has two ingredients. A general NMOS or PMOS model including all the calculation functions and LUTs for elements of the model (five intrinsic capacitances and I_{ds}) with the specified transistor width. The user should indicate the desiring width in the configuration file, such as $90nm$ and $135nm$ in Figure 3.7. All corresponding LUTs are searched and loaded to form a width determined model which is ready to compile. The implementation of the bash script to search and load LUTs can be found in Appendix B.

3.4.2 Interpolation of intrinsic capacitances and I_{ds}

Interpolation is an approach to estimate the data located in between the sampled grids of a table. The model's accuracy highly depends on the interpolation method, hence the interpolation scheme is of great importance for all the table-based models. Interpolation can be basically classified as linear interpolation, polynomial interpolation and spline interpolation. As indicated by the name, the data construction is based on a straight line or a polynomial curve. The choice of the interpolation method is based on the trade-off between the accuracy and complexity of the algorithm. Since the transistor model emphasizes its simplicity with the guarantee of the accuracy, linear interpolation is used in the design. In particular, bilinear interpolation is suitable for the model since the five intrinsic capacitance tables are two dimensional. The closest grids around C i.e., V_{gs1}, V_{gs2}, V_{ds1} and V_{ds2} shown in Figure 3.8 [29], have to be calculated at the first step. The values of C_{11}, C_{12}, C_{21} and C_{22} are obtained from the LUT. C_1 and C_2 are computed during the interpolation on dimension V_{gs}. C is the desired capacitance

obtained by applying the same interpolation rule on dimension V_{ds}. Equations (3.2) show the details of the calculation [29].

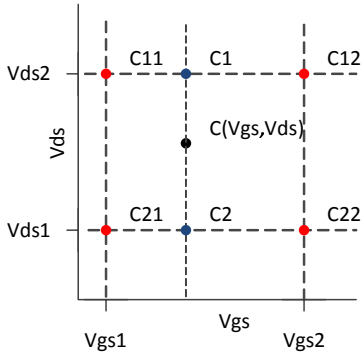


Figure 3.8: Interpolation over intrinsic capacitors

$$C_1 \approx \frac{V_{gs2} - V_{gs}}{V_{gs2} - V_{gs1}} C_{11} + \frac{V_{gs} - V_{gs1}}{V_{gs2} - V_{gs1}} C_{12} \tag{3.2a}$$

$$C_2 \approx \frac{V_{gs2} - V_{gs}}{V_{gs2} - V_{gs1}} C_{21} + \frac{V_{gs} - V_{gs1}}{V_{gs2} - V_{gs1}} C_{22} \tag{3.2b}$$

$$C \approx \frac{V_{ds2} - V_{ds}}{V_{ds2} - V_{ds1}} C_1 + \frac{V_{ds} - V_{ds1}}{V_{ds2} - V_{ds1}} C_2 \tag{3.2c}$$

Due to the fact that I_{ds} is stored in a three dimensional table, one more dimension extension on bilinear interpolation i.e., trilinear interpolation is performed in order to obtain the new I_{ds}. Equation (3.3) explains how to construct I_{ds} employing trilinear interpolation on dimension V_{ds}, V_{gs} and V_{sb} (V_{sb} for PMOS). In Figure 3.9 [29], I_1 and I_2 are results of performing bilinear interpolation twice on dimension formed by V_{ds} and V_{gs}. The final I_{ds} can be interpolated from Equation (3.3).

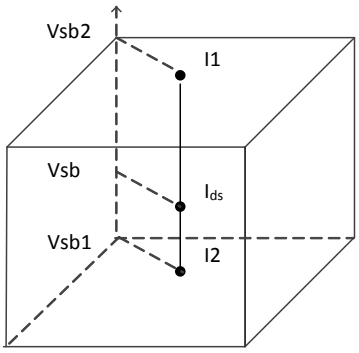


Figure 3.9: Interpolation over I_{ds}

$$I_{ds} \approx \frac{V_{bs2} - V_{bs}}{V_{bs2} - V_{bs1}} I_2 + \frac{V_{bs} - V_{bs1}}{V_{bs2} - V_{bs1}} I_1 \tag{3.3}$$

I_{ds} is the desired current obtained by applying trilinear interpolation algorithm.

3.4.3 Procedure for model development

The whole procedure for model development can be summarized in the following steps. The steps marked with ▲ are additional steps used for Monte Carlo simulation in statistical timing analysis, which will be discussed in Chapter 5. Codes of the NMOS model can be found in Appendix A

- Define the model with four input parameters *d*, *g*, *s*, *b* representing four transistor terminals *drain*, *gate*, *source* and *bulk* respectively.
- Define the branches of the circuits *br_gd*, *br_gs*, *br_gb*, *br_db*, *br_sb*, and *br_ds*.
- Load LUTs by running a bash script. The length and width of a transistor is specified in a configuration file. Thereafter, a transistor model with particular length and width is generated.
- Look up the current I_{ds} and the five intrinsic capacitors from LUTs.
- ▲ Pass the transistor length process variation dL to the model.
- ▲ Run Matlab to generate the length sensitivity coefficients tables.
- ▲ Load the length sensitivity coefficients tables into model by running bash scripts.
- ▲ Calculate sensitivities based on coefficients.
- ▲ Construct the new current I_{ds} and the gate capacitances according to length sensitivities.
- Interpolate the current I_{ds} and five intrinsic capacitors.
- Solve the equation for each branch of the circuit.

4 Deterministic Timing Analysis

The timing behaviour of the circuit is dependent on the factors such as the local temperature, the power supply and process variations during the transistor fabrication. In this chapter, we ignore all these variations by assuming the proposed model runs in a deterministic environment. The related uncertain factors will be discussed in the next chapter for the statistical timing analysis.

BSIM4 model is by far one of the most sophisticated transistor models. It has become one of the industry standard models used in circuit simulations. The BSIM4 model along with the Spectre simulation engine are regarded as golden samples to be compared in this project. The five intrinsic capacitances and the drain-source current I_{ds} in SSTM are characterized from the BSIM4 model. The accuracy of the SSTM is approved if the timing behaviour of the new model is close enough to the BSIM4 model. SSTM combined with the Spectre simulation engine is tested through running the deterministic timing analysis for digital circuits. Same circuits are also tested using BSIM4 with the same simulation engine. In this way two different models i.e., BSIM4 and SSTM are running on the same Spectre simulation engine. The relative error is the net error caused by SSTM.

4.1 Direct Current (DC) analysis on transistor model

The accuracy of the SSTM is affected by both the source-drain current I_{ds} and five intrinsic capacitances. DC analysis is the first step for model testing. All the five intrinsic capacitances can be ignored since the direct current is not able to flow across a capacitor. The proposed SSTM in DC analysis can be further simplified as a single current source I_{ds}, thus the accuracy of the current source I_{ds} in the model is checked.

The transistor level testing starts from a single NMOS transistor. The bulk and source terminals are connected to the ground. The gate has constant voltage of $1.1V$ to enable the NMOS transistor. The voltage of the drain (V_d) is swept from $-0.2V$ to $1.3V$, which covers the entire V_{ds} range for the LUT. The step size of the LUT is $0.05V$, while the step size for DC analysis is set to $0.01V$ such that the interpolation accuracy of the current I_{ds} is checked. The value of I_{ds} is obtained from the LUTs. Both V_{gs} and V_{sb} are constant in the DC analysis. The current flowing through d and s is directly influenced by the voltage across d and s. The current voltage characteristic (I-V curve) of NMOS transistor is shown in Figure 4.1. The two lines indicating I_{ds} are both captured from BSIM4 and estimated from the proposed SSTM.

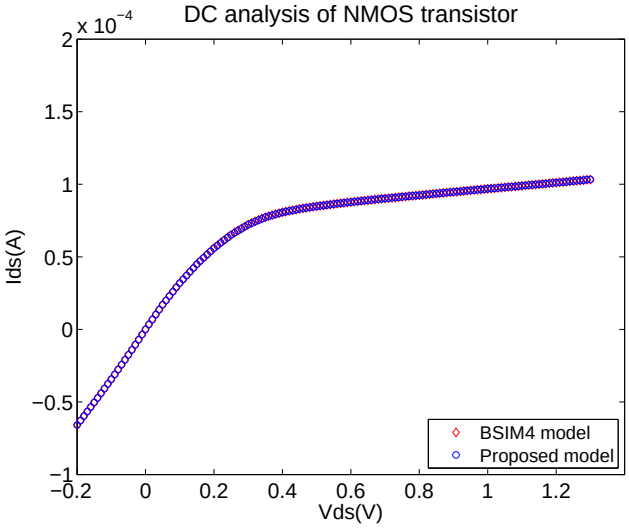


Figure 4.1: I-V curve of NMOS transistor

The p-type MOSFET(PMOS) transistor is tested in a similar way as the NMOS transistor. Figure 4.2 shows the current I_{ds} both captured from BSIM4 and estimated from the proposed SSTM.

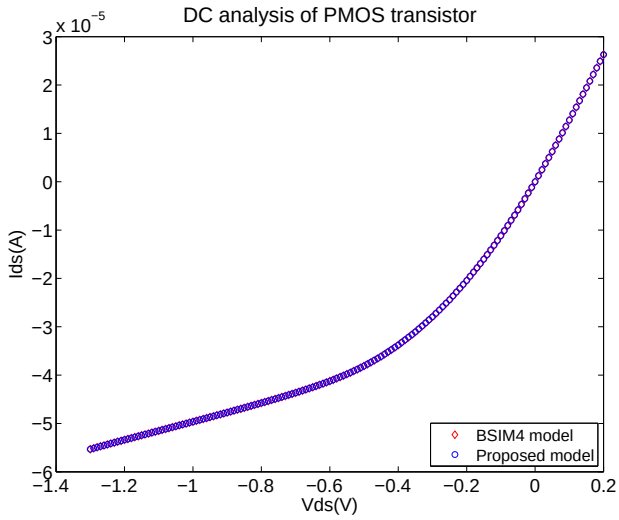


Figure 4.2: I-V curve of PMOS transistor

Both Figure 4.1 and Figure 4.2 reflect that the I_{ds} is accurately approximated. The current I_{ds} curve from the SSTM is very close to BSIM4 model. Taking NMOS transistor as an example, Figure 4.3 and 4.4 demonstrate the relative and the absolute interpolation errors compared with BSIM4 respectively.

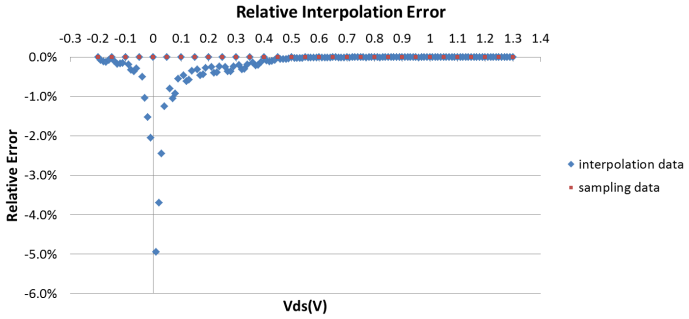


Figure 4.3: Relative interpolation error for I_{ds}

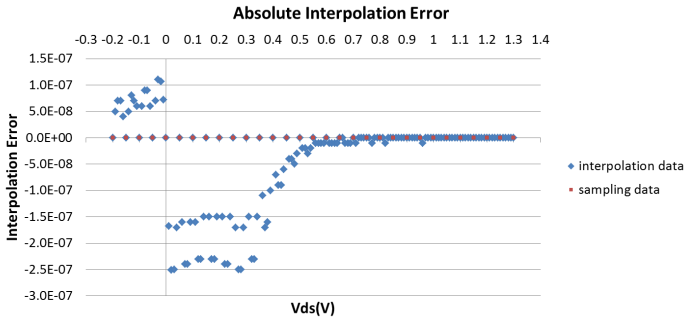


Figure 4.4: Absolute interpolation error for I_{ds}

The data shows that the maximum I_{ds} interpolation error for NMOS (PMOS) transistor is around -4.7% (3.3%), which further proves that the I_{ds} approximation is precise.

4.2 Transient analysis on transistor model

The next step, after DC analysis, is to apply the transient analysis for the transistor model. Rising ramp and falling ramp are two types of input signals commonly used for digital circuit simulation. During the transient analysis, an initial solution is given according to the DC analysis. The calculation based on input signals is time dependent, and the influence from the intrinsic capacitances to the SSTM is tested in transient analysis.

4.2.1 Transient analysis on single standard logic gate

Unlike in full-custom Integrated Circuit(IC) design that designers have to dig into the transistor level, the most commonly used gates such as INV, NAND, NOR. are packed up and stored in a library in the semi-custom design. The gates are accessed from the library and their detailed implementations are ignored, which reduces the labour intensity of the digital IC design.

NanGate was founded in October 2004, which is a provider of physical Intellectual Property (IP), providing tools, analysis and optimization of digital design [30].

NanGate Open Cell Library is an open source standard cell library dedicated to the predictive technology, which is mainly targeted to the research domain. VTH, VTG and VTL stand for High, General and Low Threshold Voltage technique respectively, those are included in the NanGate Open Cell library. In particular, $45nm$ VTL technique is chosen for this project due to the following reasons. Firstly, the low threshold has the low commutation point, which means the transistors turn on or off faster than the high threshold voltage technique. The voltage supply V_{dd} in VTL is as low as $1.1V$. Secondly, the SSTM emphasizes its statistical feature which will be introduced in Chapter 5. The transistor with $45nm$ technique is very sensitive to process variations, such characteristic better supports the accuracy verification for SSTM. Lastly, the $45nm$ VTL technique is adopted for the ISCAS85 benchmark circuits which will be used for SSTM testing.

Table 4.1 shows the typical input slew and output load capacitances given by the NanGate 45nm Open Cell Library.

Gate size: *X1*	input slew(ps)	7.5	18.75	37.5	75	150	300	600
	capacitance(fF)	0.4	0.8	1.6	3.2	6.4	12.8	25.6
Gate size: *X2*	input slew(ps)	7.5	18.75	37.5	75	150	300	600
	capacitance(fF)	0.4	1.6	3.2	6.4	12.8	25.6	51.2
Gate size: *X4*	input slew(ps)	7.5	18.75	37.5	75	150	300	600
	capacitance(fF)	0.4	3.2	6.4	12.8	25.6	51.2	102.4

Table 4.1: Typical inputs in NanGate 45nm Open Cell Library

All the cells are classified by their driving strengths, which are tagged with *X1*, *X2*, and *X4* as shown in Table 4.1. *X2* represents the cell which is composed of the largest unfolded transistors. The *X1* gate has transistors which are half of the size of the *X2* gate, thus the driving ability compared with *X2* reduces by half. *X4* implies a copy of the internal structure connected parallel to its original structure, the transistors sizes of *X4* are twice as big as the size of *X2*, which has the strongest driving strength.

Some of the gates like NAND, AND, NOR and OR are classified by their inputs number. The number following the gate name specifies how many inputs of this gate. For instance NAND3 indicates a NAND gate with three inputs. Transient analysis is carried out on standard gates with all possible combinations over transistor sizes and input numbers. The standard gates constituted by both the BSIM4 and the proposed SSTM are tested with same benchmark circuits. The propagation delay and the output slew of a gate from both models are computed, and the relative errors for SSTM against BSIM4 are obtained.

The testing environment is set up as below:

- Both rising and falling input ramps are fed into a standard gate.
- The rising input signal rises from ground $0V$ to V_{dd} $1.1V$, while the falling input ramp drops from V_{dd} $1.1V$ to ground $0V$.

- The input slews vary based on the range indicated from NanGate 45nm Open Cell Library (Table 4.1).
- The effective output load capacitances vary according to the range from NanGate 45nm Open Cell Library (Table 4.1).
- Cells with more than one input such as NAND2, NAND3 etc. have only one input signal varied, the rest of the pins are fixed to either the power supply V_{dd} or the ground to enable the gate.
- The temperature is set to 27.0°C.

The transient analysis on NAND gates are picked as examples to explain in this thesis, the rest of the gates can be analysed in the same way. The delay of the gate composed of SSTM is compared with the delay of the gate composed of BSIM4. Each gate was simulated with 7 input signals with different slew rates and 9 different output capacitances. All these combinations lead to 7×9 comparisons for each gate. The relative delay errors for NAND gate are shown in Figure 4.5a and Figure 4.5b.

The following conclusion is drawn from analysing the gate delay error of NAND gates.

1. When the type, driving strength, and input slew of a gate are fixed, a smaller load capacitance usually results in bigger errors. As seen from the Figure 4.5a, the general error for $0.4fF$ output load is generally larger than the error for $102.4fF$.

   This can be explained through the transistor level schematic of NAND gate. As shown in Figure 4.6, the intrinsic capacitance C_{db} of a NMOS transistor is parallel connected with the output loads.

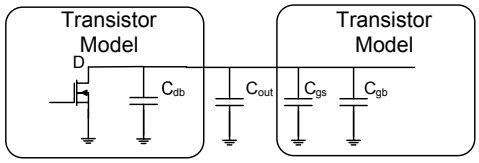


Figure 4.6: Intrinsic capacitor C_{db} and the effective output load

   Since modelling C_{db} introduces error, and the error grows when the ratio of C_{db}/C_{out} increases. The accuracy of C_{db} for the model is crucial when the load capacitances are small. Figure 4.7 shows the formation of junction capacitance C_{db} [31].

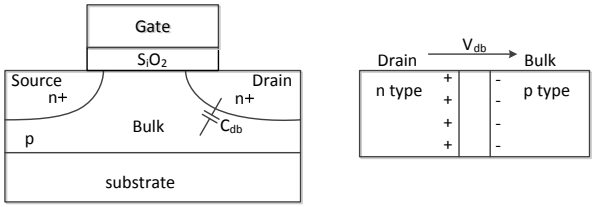
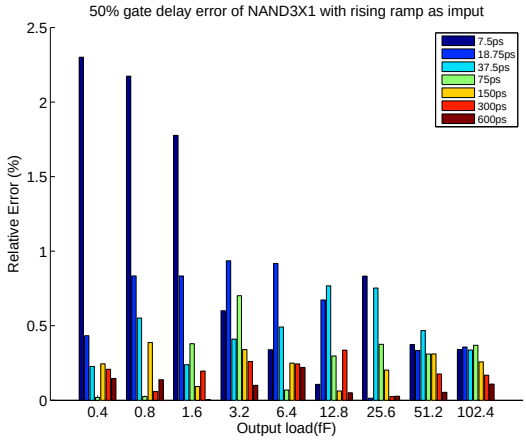
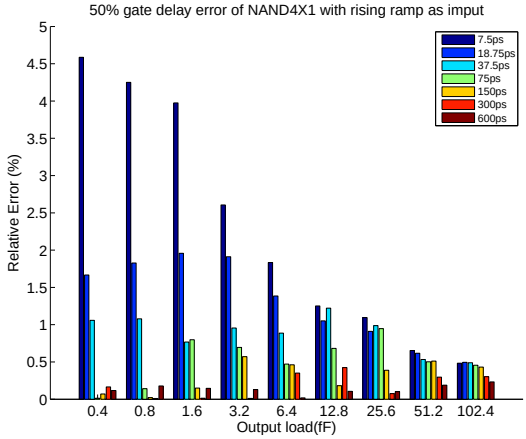


Figure 4.7: Junction capacitance C_{db}



(a) Relative delay error for NAND3_X1 gate



(b) Relative delay error for NAND4_X1 gate

Figure 4.5: Relative delay error for NAND gate

Taking NMOS transistor as an example, the drain is made from n-type materials and the bulk is made from p-type materials. As shown in Figure 4.7, C_{db} is the capacitance formed from the PN-junction. Equation (4.1) expresses the way of calculating PN-junction capacitances [31]:

$$C_j(V) = A\sqrt{\frac{\varepsilon_{si} q}{2}\left(\frac{N_A N_D}{N_A + N_D}\right)}\frac{1}{\sqrt{\phi_0 - V}} \tag{4.1a}$$

$$\phi_0 = \frac{kT}{q} ln\left(\frac{N_A N_D}{n_i^2}\right) \tag{4.1b}$$

where N_D and N_A are the doping densities of n-type and p-type areas respectively, ϕ_0 is the built-in junction potential, ε_{si} is the dielectric constant, k is the Boltzmann's constant, T is the temperature, q is the electron charge, and n_i is the intrinsic carrier concentration in silicon.

Replacing voltage in Equation (4.1) by V_{db}, the C_{db} is computed as

$$C_{db}(V_{db}) = A\sqrt{\frac{\varepsilon_{si}q}{2}(\frac{N_A N_D}{N_A + N_D})}\frac{1}{\sqrt{\phi_0 - V_{db}}} \tag{4.2}$$

where V_{db}=V_{ds}-V_{bs}. This means the value of junction capacitance C_{db} is dependent on the terminal voltage V_{bs}. However C_{db} in the SSTM model is accessed through the LUT indexed by V_{ds} and V_{gs}, the influence from V_{bs} is omitted in the model. The V_{bs} for the NMOS transistors in NAND gate are not constant. So the relative error with small load capacitance is higher than the error with large load capacitance.

2. When the type, driving strength and load capacitance of a gate are fixed, a sharp input signal induces large errors. Figure 4.5a shows that for each particular output load, the error for 7.5ps input slew is significantly larger than the error for 600ps input slew.

   A steep input ramp stimulates a steep output signal. The output voltage V_{ds} can not keep the pace of the sharp change voltage of input V_{gs}. The transistor crosses the saturation region and stays in the linear region for most of the transient time. However, the large proportion of data are characterized from the saturation region, and the resolution for linear region is not enough for SSTM compared to the saturation region. From the I_{ds} interpolation errors, which was illustrated in Figure 4.3, we can see the huge errors are concentrated at the linear region. Hence the relative error for an input signal with small slew is larger than the one with large slew.

3. The error grows with the input numbers of a gate. Comparing errors between NAND3 gate and NAND4 gate in Figure 4.5a and Figure 4.5b respectively, we found that the error trend for NAND4 gate starts dropping from about 4.5%, while the trend for NAND3 begins falling from around 2.5%.

   Firstly, the output junction capacitors C_{db} for all PMOS transistors in NAND gate are connected in parallel. The total capacitance of capacitors connected in parallel equals to the sum of their individual capacitance, which means the equivalent gate output capacitance C'_{db} is the sum of all the individual C_{db} for PMOS transistors. According to the previous discussion, the relative large C_{db} leads to large results deviation. The error reduces accordingly with the growth of load capacitance.

   Secondly, the high-stacked structure of NMOS transistors in NAND gate introduces errors. Taking NAND3 gate in Figure 4.8 [32] as an example, suppose pins $A1$ and $A2$ are tied with V_{dd}, $A3$ starts rising from $0V$ to V_{dd}. The NMOS transistor M3 is going to turn on. Meanwhile, it discharges node N2. We know a transistor turns on when

$$V_{gs} > V_{th} \tag{4.3}$$

   For transistor M2, $V_g = V_{dd}$, $V_s = V_{N2}$. Hence equation (4.3) can be derived as

$$V_{dd} - V_{N2} > V_{th} \tag{4.4a}$$
$$V_{N2} < V_{dd} - V_{th} \tag{4.4b}$$

It means that the current can conduct through M2 until node N2 is discharged below $V_{dd} - V_{th}$, and the same story applies to transistor M1. The internal delay to conduct current through the stack structure impacts the timing behaviour directly. The more input numbers of a gate, the longer stacks is, the longer it takes to conduct the current through the stack structure. The error introduced from the high stack structure is more obvious. Thus the gate with large number of inputs has larger error than the one with small number of inputs.

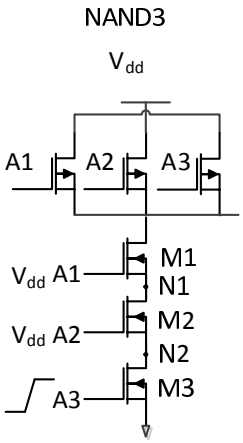


Figure 4.8: Internal charge effects

The distribution of the delay and the output slew error on NAND gate considering all combinations of input slews and load capacitances are shown in Figure 4.9. The simulation results for other standard cells can be found in Appendix D.

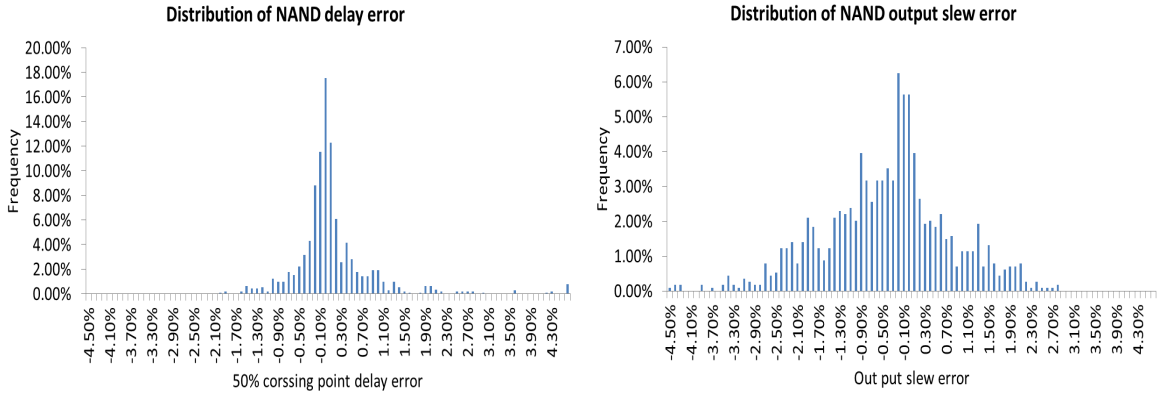


Figure 4.9: Error distribution for standard gates

From Figure 4.9 we can see that the propagation delay errors mainly concentrate on the range from -1% to 1%. Around 18% of the cases have only -0.1% propagation delay error. The most output slew errors are located between -2% and 2%. The small relative error indicates the proposed SSTM is in general accurate for digital circuit

design. However some gates with some corner cases (small output load and fast inputs) suffer from huge errors during the simulation.

The mean[1], standard deviation[2], maximum and minimum delay errors for all the standard gates are presented in Figure 4.10 and Figure 4.11. The meanings of index numbers are shown in Table 4.2

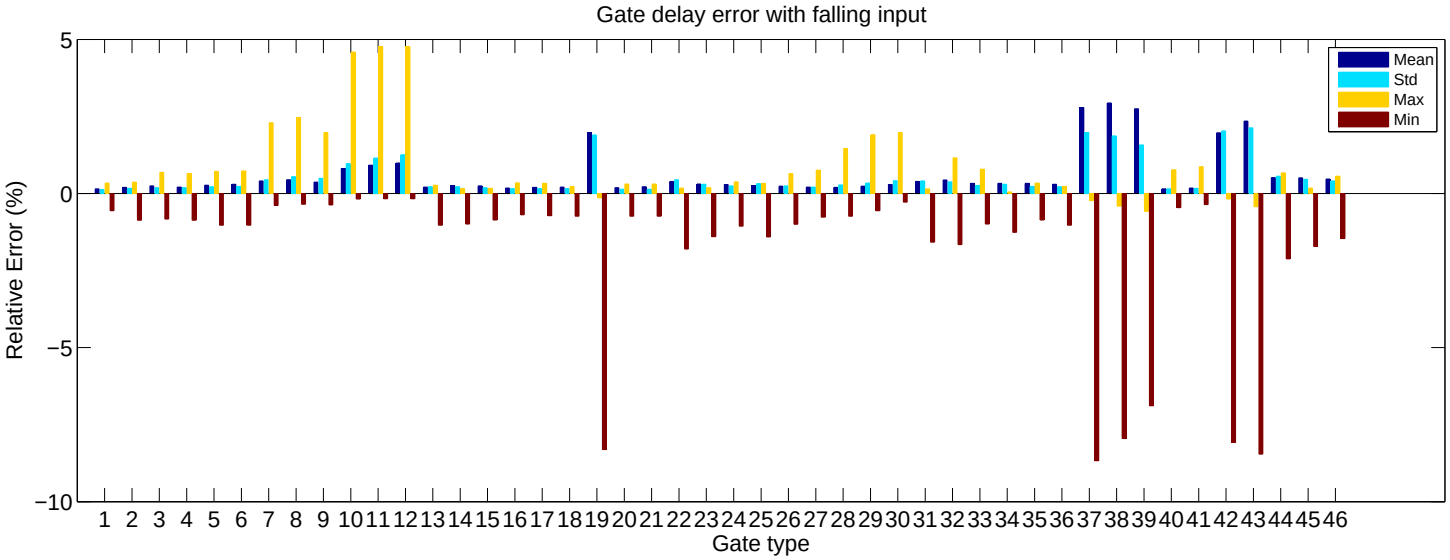


Figure 4.10: Gate delay errors with falling input signal

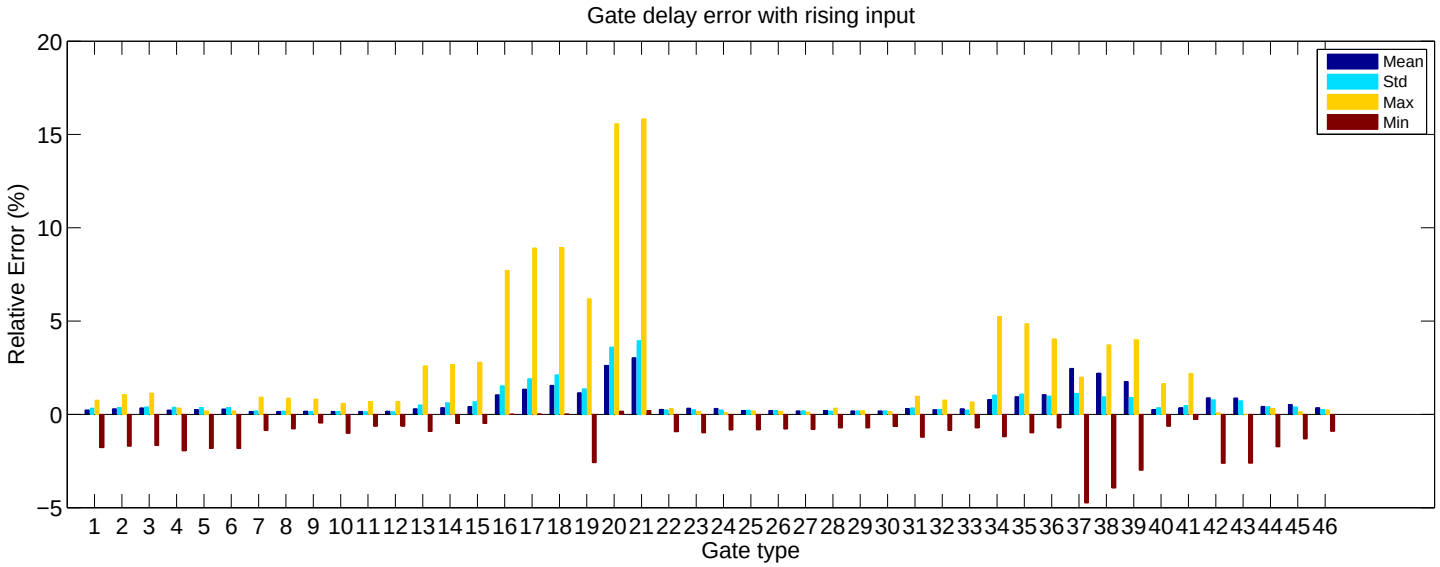


Figure 4.11: Gate delay errors with falling input signal

[1]Here we choose the mean of absolute errors, since the positive and negative errors will cancel each other
[2]The standard deviation of absolute errors

Table 4.2: Index of gate type for Figure 4.10 and Figure 4.11

1	INVX1	13	NOR2X1	25	AND3X1	37	OR4X1
2	INVX2	14	NOR2X2	26	AND3X2	38	OR4X2
3	INVX4	15	NOR2X4	27	AND3X4	39	OR4X4
4	NAND2X1	16	NOR3X1	28	AND4X1	40	XNORX1
5	NAND2X2	17	NOR3X2	29	AND4X1	41	XNORX2
6	NAND2X4	18	NOR3X4	30	AND4X1	42	XORX1
7	NAND3X1	19	NOR4X1	31	OR2X1	43	XORX2
8	NAND3X2	20	NOR4X2	32	OR2X2	44	BUFX1
9	NAND3X4	21	NOR4X4	33	OR2X4	45	BUFX2
10	NAND4X1	22	AND2X1	34	OR3X1	46	BUFX4
11	NAND4X2	23	AND2X2	35	OR3X2		
12	NAND4X4	24	AND2X4	36	OR3X4		

Through analysing the data in the table, we can say that the mean errors for most of the gates are less than 1%. Some gates with large inputs such as NAND4, NOR4, AND4, and OR4 under certain circumstances do not perform well.

Firstly, the errors for NOR4 and OR4 are caused mainly from the rising input, while for NAND4 and AND4 are mainly resulted from the falling input. Considering the high-stack structure of NMOS transistor in NAND gate again, when the gate is injected with a rising signal, the NMOS will switch on, and will dominate the output signal. The error caused from the NMOS stack structure is more obvious. In contrast, when injected with a falling signal, the PMOS transistors will switch on and PMOS will mainly effect the output signal. However, discharging the paralleled PMOS transistors for the NAND gate has a small internal delay, which guarantees the output accuracy. In contrast, the NOR gate has the opposite schematic as NAND gate, i.e., NMOS transistors are connected in parallel to ground and PMOS transistors are connected in series to V_{dd}. The stacked PMOS structure dominants the error, hence the falling input signal causes larger error than rising input signal.

Secondly, the NOR4 gate and the OR4 gate have huge maximum error (16% and 6% respectively). The errors are mainly from the small output loads and the fast input signals. One reason to explain such a phenomenon is that the proposed SSTM is a capacitance-based model. The capacitances are non linear and they are dependent on the terminal voltages. The capacitance of a voltage dependent capacitor is calculated by Equation (4.5).

$$dQ = C(V(t))dV \tag{4.5a}$$

$$Q = \int_0^V C(V(t))dV \tag{4.5b}$$

However, based on the discussion in Chapter 2, the computer program does not have the ability to solve the integration problem directly. The numerical method such as Backward Euler will introduce the local truncation error at every time step when calculate the integration function. Consequently, the non-conserved charge will impact

the current accuracy, and such error is accumulated. The situation become worse when the capacitor is heavily dependent on the voltages, and the voltages change fast with the time. The error grows when the inputs switch fast since the voltage changes more for each time step.

If we look at the transistors sizes for the logic gates, the NOR4_X4 gate has PMOSs with width of 650nm and NMOSs with width of 180nm. The large transistor size indicate large voltage dependent capacitor in the model. The large number of inputs indicate more voltage dependent capacitors in the model. The voltage dependent capacitors will dominate the gate outputs if the output loads are small. This may explain why huge errors happen on the NOR4_X4 and OR4_X4 gates.

4.2.2 Transient analysis on the critical path of International Symposium on Circuits and Systems (ISCAS) benchmark circuits

Transient analysis for all the standard gates which are built up by the proposed transistor model can achieve very accurate delay and output slew. Large scale circuits are the combinations of standard cells. A benchmark circuit combining the standard gates is meant to test the combinational logic behaviour. ISCAS-85 [33] benchmark circuits including c432, c499, c880, c1908, c2670, c3540, c5315, c6288 and c7552 have been accepted widely for combinational circuits testing. The varieties of gates allows full test on each type of the gate. ISCAS benchmark circuits are broadly employed in design verification, power consumption and timing analysis in digital circuit design field [33]. The critical path of the ISCAS-85 benchmark circuits are simulated. Cadence encounter tool helps with generating the circuit critical path, which also tells if the rising or the falling input ramp will cause the worst case of the delay. The simulation environment is set up as follows

- The input slews for all the benchmark circuits are set to $20ps$.
- The power supply V_{dd} is $1.1V$, and the temperature is 27.0°C .
- The standard gates are connected in series. Output loads capacitances are used to represent the wire capacitance in the real circuit.

The accuracy of the proposed transistor model is judged by the degree of similarity to BSIM4. Figure 4.12 and Figure 4.13 illustrate examples about transient analysis for circuit C432. Both the delay and the output slew are compared with BSIM4. The structure of C432 circuit is a set of standard gates connected in series. The name and the order of the standard gates are shown in the X-axis in Figure 4.12. The bars indicate the net error caused by standard gate at each stage, and the line shows the accumulated error for the entire circuit.

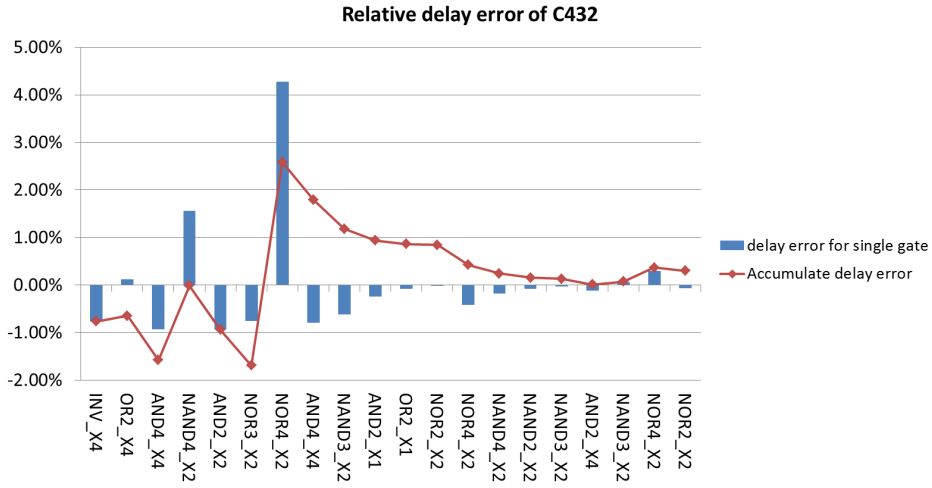


Figure 4.12: Relative delay error for C432

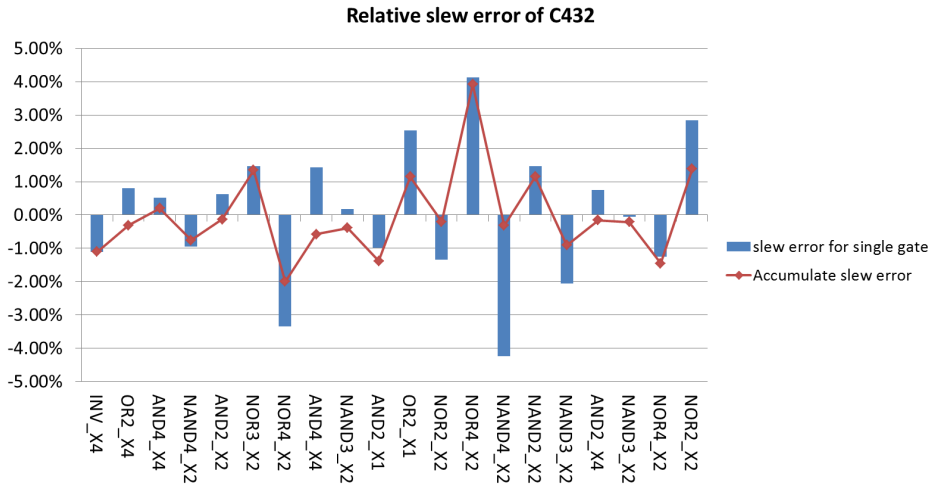


Figure 4.13: Relative delay error for C432

Figure 4.12 and Figure 4.13 show that NOR4X2 has the maximum standard gate delay error, which is around 4.3%, while NAND4X2 has the maximum output slew error about -4.2%. The gate error can be either positive or negative and they can cancel each other. The final relative circuit propagation delay error compared with BSIM4 is lower than 1%, and the final output slew error is less than 2%. It means that the proposed SSTM is capable of guaranteeing the simulation accuracy at the circuit level.

Table 4.3 reports the gate delay and the output slew error for all the benchmark circuits. All the tests are carried out with the worst delay cases for critical paths.

	Relative Error(%)	
circuit	50% gate delay	output slew
c432	0.30	1.38
c499	-0.95	0.44
c880	-0.21	0.57
c1355	-0.12	0.42
c1908	-0.73	-0.06
c2670	-0.53	-0.29
c3540	-0.59	-1.44
c5315	-0.61	-1.78
c6288	-0.99	-0.08
c7552	0.00	0.01

Table 4.3: Delay and slew error for critical path of ISCAS-85 benchmark circuits

Table 4.3 presents that the gate delay errors for all benchmark circuits are less than 1%. In particular, the maximum delay error is -0.99% for circuit C6288. In addition, The output slew errors for all benchmark circuits are less than 2%. The maximum output slew error is -1.78% for C5315. The small error shows that the SSTM is accurate enough in terms of building up standard cells.

5 Statistical Timing Analysis

Nothing in the world is exactly identical. Process variations and environmental variations are two sources of variations that mainly cause the parameters of a circuit to deviate from their nominal values [34].

Both of the variations impact significantly the timing behaviour of the circuits. As indicated by the name, environmental variation is the change in the true operating environment such as the temperature or the supply voltage [34]. Process variations such as channel length, width, threshold voltage are uncertain due to the chip manufacturing. With the transistor fabrication technology scaling down, the process variations in the nanometer regime cannot be ignored any more. Process variations are typically categorized as intra-die (or within-die) variations and inter-die (or die-to-die) variations [35].

5.1 The sensitivity of the transistor model

The traditional approach to handle uncertain factors is to introduce the corner case for a device. For instance, the best case assumes the transistor is fast, while the worst case assumes the transistor is slow. However, this assumption indicates that if one transistor is fast, all transistors are fast. It handles well for global deviations such as the temperature, power supply voltage etc., but not reasonable for local deviations. Modelling the gate according to the worst case is very likely to underestimate the performance [36]. Statistical Static Timing Analysis (SSTA) technology targets statistically analyse the delay variations due to intra-die process variations [37].

The table-based GLM has its limitation on being extended from STA to SSTA considering process variations. Extra characterizations and interpolations are required for each transistor dimension such as length, width etc. for performing SSTA. The number of tables for the standard cell library will grow explosively when considering all the cells with process variations. In contrast, characterizing only two types (NMOS and PMOS) of transistors are demanded in TLM for SSTA, which dramatically reduces the characterization time and the library size. In order to perform SSTA, the proposed simplified transistor model (introduced in chapter 3) is extended with process variations awareness. Figure 5.1 [6] shows the extension of the model concerning the variability in the circuits parameters. The parameter vector ξ represent all types of process variations such as the channel length, the thick of the isolation layer, the threshold voltage etc.

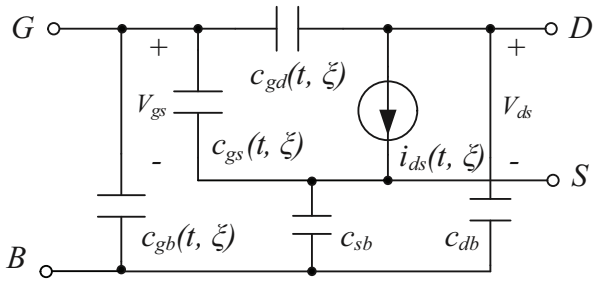


Figure 5.1: a simplified model

As shown in Figure 5.1, the simplified transistor model is composed of the drain-source current I_{ds} and the five parasitic capacitances C_{gb}, C_{gs}, C_{gd}, C_{sb} and C_{db}. Those components are dependent on the transistor dimensions such as length, width etc. Since the transistor dimensions vary during the transistor fabrications, the value of the model components change accordingly. It is necessary to assign the right value to the components of the model when running statistical timing analysis. One possible way to determine component values is to characterize the transistor for each dimension, and then apply interpolations. For example, the transistor length in the real process procedure is possible to differ from $35nm$ to $70nm$. Thus transistor characterizations for sampling lengths are done first. The components values for the rest of the lengths can be achieved through interpolating over lengths. The accuracy of this method depends on the sampling step size. A small step size will result in a cumbersome model, causing the burden for both loading and computing the model. Too few samples will degrade the accuracy of the model. However, for the many varying parameters (5-8) the table numbers as well as the interpolation time will grow explosively. With attempting to ensure the model accuracy, and to secure the model simplicity, this interpolation method is impractical and is not adopted in the implementation.

In order to estimate the components value of the model with process variations, we need to know at what degree the variation impacts the component of the model. Sensitivity is defined to measure such a degree. It can be computed by Equation (5.1).

$$S = \frac{\triangle A}{\triangle P} \tag{5.1}$$

where S is the sensitivity, $\triangle P$ represents the possible transistor process variations (length, width etc.), and $\triangle A$ denotes the difference of the corresponding model parameter caused by the process variation. Both S and A are dependent on the terminal voltages V_{ds} and V_{gs}.

5.1.1 The length sensitivity of the transistor model

In particular, the length sensitivity of the transistor is introduced in this thesis work. Other types of sensitivity can be studied in the same manner. The length sensitivity is achieved by replacing $\triangle P$ in Equation (5.1) with $\triangle L$, which is shown in Equation (5.2)

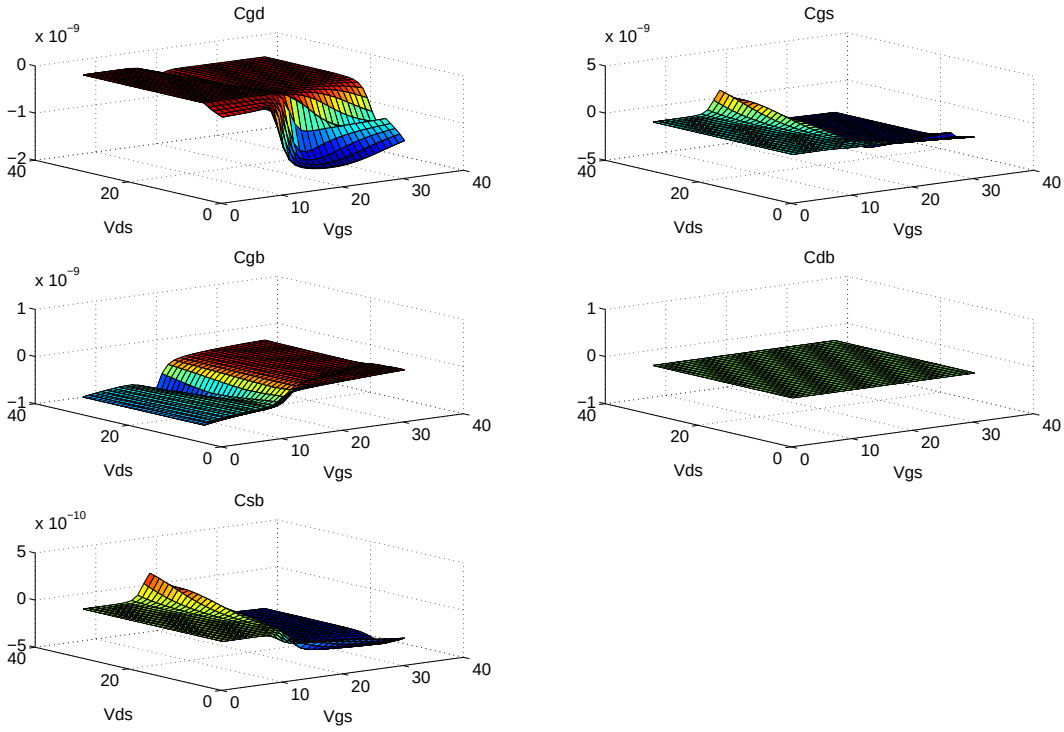


Figure 5.2: Length sensitivities of intrinsic capacitances

$$\mathbf{S_C} = \frac{\mathbf{C}}{\triangle L} \quad \mathbf{S_C}, \mathbf{C} \in \mathbf{R}^{31\times 31} \tag{5.2a}$$

$$\mathbf{S_I} = \frac{\mathbf{I}}{\triangle L} \quad \mathbf{S_I}, \mathbf{I} \in \mathbf{R}^{31\times 31\times 12} \tag{5.2b}$$

Where $\triangle L$ is the difference of the transistor length. $\mathbf{S_C}$ and $\mathbf{S_I}$ are length sensitivities for capacitance and current respectively. We notice that the sensitivity is a matrix, because its value is dependent on the terminal voltages.

In order to see how the parasitic capacitances are sensitive to the transistor length, we compute the length sensitivities for all capacitances and demonstrate them in Figure 5.2. All the data are from NMOS transistor with the width of $90nm$.

1. The length sensitivity for capacitor C_{sb} is one order of magnitude smaller than its other counterparts. Thus the influence caused from C_{sb} is negligible. In order to reduce the model complexity and speed up the computing, the value of C_{sb} in the model for statistical timing analysis is kept constant.

2. The length sensitivity for capacitor C_{db} is always zero. It means that C_{db} always keeps the same value no matter how the length differs. The reason can be explained through the PN-junction capacitance equation of C_{db},

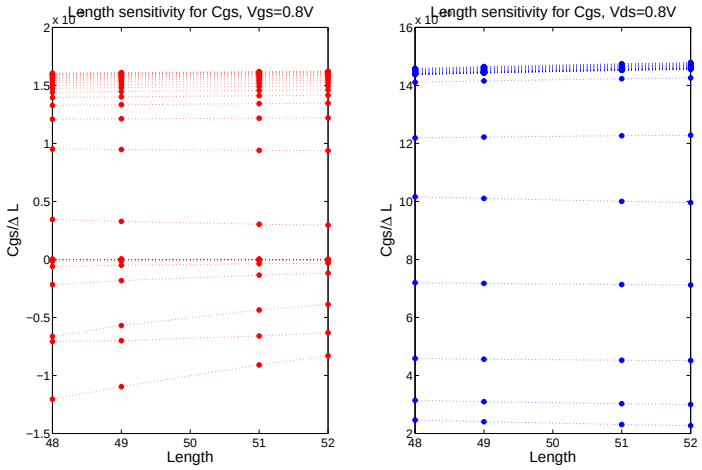


Figure 5.3: length sensitivities for C_{gs} with different transistor lengths

$$C_{db}(V_{db}) = A\sqrt{\frac{\varepsilon_{si}q}{2}(\frac{N_A N_D}{N_A + N_D})}\frac{1}{\sqrt{\phi_0 - V_{db}}}$$

where N_D and N_A are doping densities of n-type and p-type areas respectively, ϕ_0 is the built-in junction potential, ε_{si} is the dielectric constant, q is the electron charge.

All the parameters used to calculate C_{db} are independent of the transistor length. Thus the junction capacitor C_{db} is also seen as a constant in statistical timing analysis.

3. The gate capacitances C_{gs}, C_{gd}, C_{gb} are sensitive to the transistor length. We plot length sensitivities for all three gate capacitances at four sampling lengths i.e., $48nm$, $49nm$, $51nm$ and $52nm$ in Figure 5.3, Figure 5.4 and Figure 5.5. Taking C_{gs} as an example. The left part of Figure 5.3 assumes the terminal V_{gs} is fixed, each column indicates a specific transistor length, which has 31 points indicating the V_{ds} varies from $-0.2V$ to $1.3V$ with step size $0.05V$ (the entire range of the LUT). The value of each point is the length sensitivity for a particular pair of (V_{ds} and V_{ds}). The same story applies for the right part of Figure 5.3. Instead, the V_{ds} is assumed fixed, and the V_{gs} varies.

   We can see from two graphs that the length sensitivities are dependent on both V_{ds} and V_{gs}. Since the points in each column (length) change in their own way, the sensitivity is also dependent on the length. However, the irregular trends increase the difficulty in approximating the new capacitance.

Additionally, the growth trend of the current I_{ds} is shown in Figure 5.6. Similar as gate capacitances, the current I_{ds} does not change uniformly with the transistor length within the entire operating region. Specifically speaking, The value of the I_{ds} is more sensitive to the length in the saturation region than other two regions. Therefore the length sensitivity of the I_{ds} is also dependent on the transistor length.

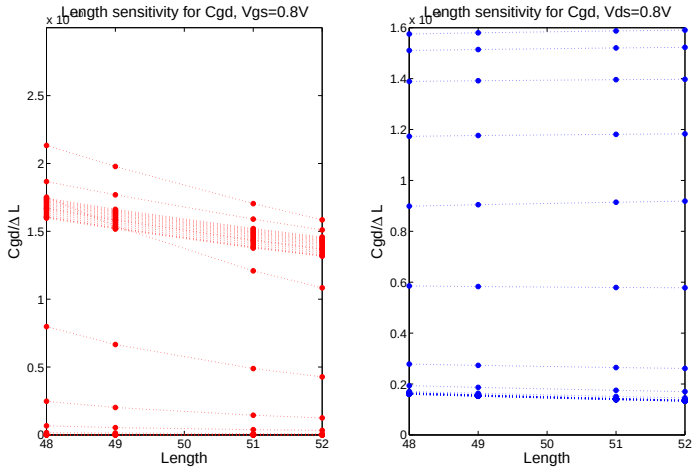


Figure 5.4: length sensitivities for C_{gd} with different transistor lengths

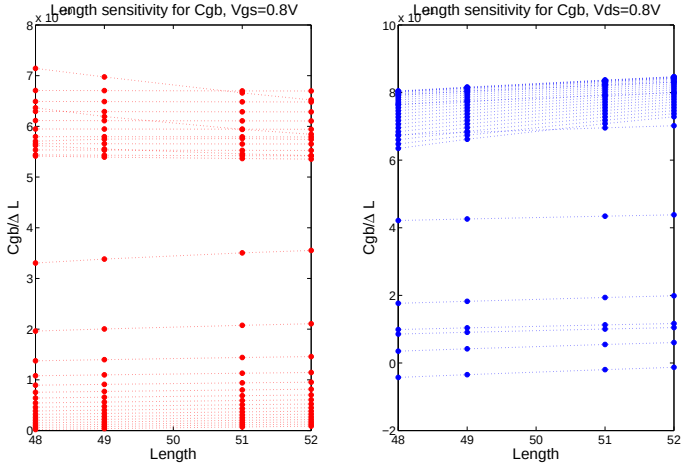


Figure 5.5: length sensitivities for C_{gb} with different transistor lengths

The challenges listed above lead to the difficulty in estimating the gate capacitances and I_{ds} for the model for any given transistor length due to process variations. Two possible methods of determining the length sensitivities are discussed in the following sub-sections, their merits and weaknesses will be discussed too.

5.1.2 Fixed length sensitivity

One intuitive way of deciding the length sensitivity is to roughly regard the gate capacitances and I_{ds} growing linearly with the transistor length. In other words, the $\triangle\mathbf{A}$ in Equation (5.2) is independent of the transistor length L.

This fixed sensitivity scheme requires to characterize only one set of the extra length of transistors. This strategy was used at the early development stage of the model. $50nm$ is set as the nominal length of the transistor because we take an extra $5nm$ as

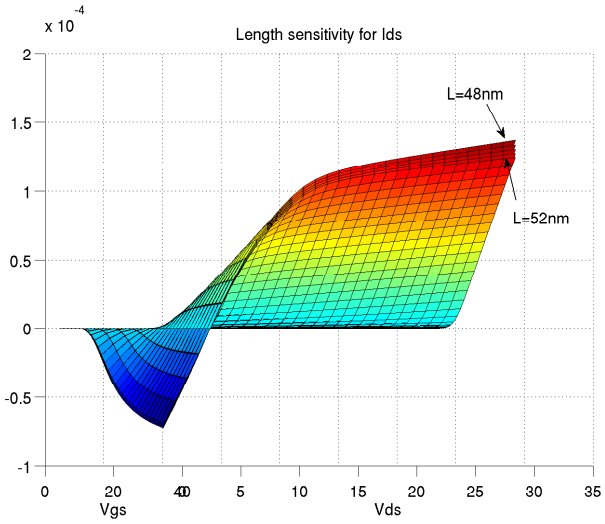


Figure 5.6: I_{ds} values with five transistor length

the error tolerance from the $45nmVTL$ technique. The LUTs for the nominal intrinsic capacitances and I_{ds} are characterized. Besides, the LUTs for the transistors with length $nominal + dev$ are also characterized. Here dev is the standard deviation of the length process variation. For instance, $2nm$ is used for the standard deviation of the process variation. Hence $52nm$ transistor is characterized. Equation (5.2) is rewritten as Equation (5.3).

$$\mathbf{S_C} = \frac{\mathbf{C_{52}} - \mathbf{C_{50}}}{\triangle L} \quad \mathbf{S_C}, \mathbf{C_{52}}, \mathbf{C_{52}} \in \mathbf{R}^{31\times31} \tag{5.3a}$$

$$\mathbf{S_I} = \frac{\mathbf{I_{52}} - \mathbf{I_{50}}}{\triangle L} \quad \mathbf{S_I}, \mathbf{I_{52}}, \mathbf{I_{52}} \in \mathbf{R}^{31\times31\times12} \tag{5.3b}$$

where $\mathbf{S_C}$ represents the length sensitivity for the gate capacitance, $\mathbf{C_{52}}$ and $\mathbf{C_{50}}$ are the capacitance LUTs characterized from transistors with lengths of $52nm$ and $50nm$ respectively; $\mathbf{S_I}$ represents the length sensitivity for the I_{ds}, $\mathbf{I_{52}}$ and $\mathbf{I_{50}}$ are the I_{ds} LUTs characterized from transistors with lengths of $52nm$ and $50nm$ respectively. $\triangle L$ is $2nm$.

Simplicity is the main advantage of this method since characterizing the intrinsic capacitances and the I_{ds} requires large amount of CPU time. The accuracy, however, suffers from the over-simplified approximation method, especially when the standard deviation of length σ_L is large. When the transistor length reaches far from the nominal value, the estimation is not accurate at all.

5.1.3 The proposed scheme i.e., polynomial curve fitting for the length sensitivity

A more elaborate way to construct the gate capacitances C_{gs}, C_{gb} and C_{gd} and source-drain current I_{ds} from a given channel length L is essential. The target is to pursue

a tolerable estimation error while maintaining the model's simplicity. Describing the trend of the length sensitivity with the polynomials according to the curve fitting theory is put forward in this thesis. Compared to the fixed length sensitivity scheme, the polynomial curve strategy improves the data approximation accuracy, with which a wider process variation range for statistical timing analysis is achievable. Meanwhile, the complexity of the model does not increase considerably, making the statistical timing analysis for the circuit with MC method practicable. The details of this scheme will be discussed in the following sub-section.

5.1.3.1 Constructing the polynomial coefficients based on the least squares approach

The concept of curve fitting is to create a general function to catch the trend among a set of data. The existing data is regarded as the sampling points, and the rest of the points are obtained through the function. Figure 5.7 shows an example for applying the first degree (linear) polynomial $y = ax + b$ to fit a second-order curve $y = x^2$. There are 10 given observations in Figure 5.7, which are $[(1,1),(2,4),(3,9),(4,16),(5,25),(6,36),(7,49),(8,64),(9,81),(10,100)]$. The straight line $y = ax + b$ is the trend of the sampling points estimated by the first-order polynomial.

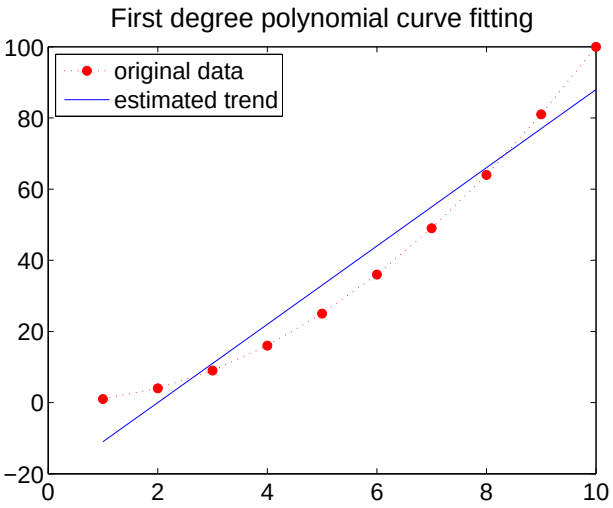


Figure 5.7: Using one degree polynomial $y = ax + b$ to fit on data from $y = x^2$

The way to construct the polynomial coefficients (a and b) is of utmost importance to ensure that $f(x)$ in the example is the line that most closely match the given observations. The least squares approach is one of the many strategies used to achieve the best polynomial curve. In particular, the linear least squares approach is introduced to create the polynomial coefficients due to the simple computational feature. Here we show an example the way that linear least square method is used to generate a first-order polynomial ($y = ax + b$). Polynomials with higher degree are computed in the same way. The higher order the polynomial curve is, the more accurate the estimation

is.

The error of the given observation points can be described in Equation (5.4) [38], which sums the distances (approximation error) between the estimating data and the original data at the sampling points.

$$E = \sum_{i=1}^{n} [y_i - (ax_i + b)]^2 \tag{5.4}$$

where E is the estimating error, (x_i, y_i) indicates an observation point, n is the number of the given observation points, a and b are polynomial coefficients.

The error is minimum when the derivative of the error with respect to a and b both reach to 0. Hence the polynomial coefficients a and b are obtained through solving the Equation (5.5).

$$\frac{\partial E}{a} = -2 \sum_{i=1}^{n} x_i (y_i - ax_i - b) = 0 \tag{5.5a}$$

$$\frac{\partial E}{b} = -2 \sum_{i=1}^{n} x_i (y_i - ax_i - b) = 0 \tag{5.5b}$$

5.1.3.2 Determining the polynomial term for current I_{ds}

As discussed in Section 5.1.3.1, (x_i, y_i) denotes an observation point. x_i is used as the polynomial term to estimate the point y_i'. A good coherence between x_i and y_i will dramatically increase the approximation accuracy without modifying the polynomial order. Thus it is crucial to choose the observation points x_i, such that y_i is strong relevant to x_i. For instance, the polynomial curve in the previous example can be expressed as

$$y = a_1 x + b_1$$

The polynomial can be also expressed as

$$y = a_2 \frac{1}{x} + b_2$$

where a_1, b_1 and a_2, b_2 are the corresponding polynomial coefficients to terms x and $\frac{1}{x}$. Now if we go back to the current (I_{ds}) estimation according to the transistor length (L_{eff}), the prior knowledge is required to determine which shape of of the transistor length dominates the values of the current I_{ds}.

The fundamental transistor model Shichman-Hodges model was published in the 1960s by Shichman and Hodges. It constitutes a well known representation of MOS transistors [39], and is chosen by Simulation Program with Integrated Circuit Emphasis(SPICE) simulator as level-1 model. The Level-1 model uses a linear approximation in both linear and saturation region to predict the channel length according to the drain voltage(V_{ds}), which is only accurate for long channel transistor with old technology. The Equations from (5.6) to (5.8) explain Level-1 transistors in different operating regions [40].

1. Cutoff region($V_{GS} < V_T$):

$$I_{ds} = 0 \tag{5.6}$$

2. Linear region($V_{DS} < V_{GS} - V_T$):

$$I_{ds} = \frac{\mu C_{is}}{2} \times \frac{W}{L_{eff}}[2(V_{GS} - V_T) - V_{DS}](1 + \lambda V_{DS}) \tag{5.7}$$

3. Saturation region($0 < V_{GS} - V_T < V_{DS}$):

$$I_{ds} = \frac{\mu C_{ox}}{2} \times \frac{W}{L_{eff}}(V_{GS} - V_T)^2(1 + \lambda V_{DS}) \tag{5.8}$$

In the preceding equations, μ is the charge-carrier mobility, C_{ox} is the capacitance per unit area of the gate dielectric, V_T is the threshold voltage, λ is the channel length modulation parameter, W and L_{eff} are the width and length of the transistor channel.

SPICE level-2 has multiple supplements compared with level-1, which includes the second-order effect such as the influence of sort/narrow channel, the saturation effects due to limited drift velocity, and the temperature dependence and so on. SPICE level-3 is more sophisticated. For instance, the extension is related to the threshold voltage sensitivity, length sensitivity, width, drain voltage and other issues. In short, the higher the level, the more model issues are considered, the more accurate the model is. The BSIM with version 4.6.4 used in this thesis is taken as SPICE level-54.

Since our main concern lies in determining how the general trends the channel length L_{eff} and the terminal voltages V_{gs} and V_{ds} impact the model, the second order effects from the length and terminal voltages are neglected. SPICE level-1 is selected for analysing. Parameters irrelevant with L_{eff}, V_{gs} and V_{ds} in SPICE level-1 equations can be seen as constants. Thus the relationship between the current I_{ds} and the terminal voltages V_{gs} and V_{ds} are simplified as

1. Cutoff region($V_{gs} < V_T$):

$$I_{ds} = 0 \tag{5.9}$$

2. Linear region($V_{ds} < V_{gs} - V_T$):

$$I_{ds} \propto \frac{K_1}{L_{eff}} f(V_{gs}, V_{ds}) \tag{5.10}$$

3. Saturation region($0 < V_{gs} - V_T < V_{ds}$):

$$I_{ds} \propto \frac{K_2}{L_{eff}} g(V_{gs}, V_{ds}) \tag{5.11}$$

where K_1 and K_2 are constants, f and g are two functions which depend on V_{gs} and V_{ds}. The equations show that I_{ds} grows inversely proportional to transistor channel length L_{eff} in both the linear and saturation regions. That is to say, the $\frac{1}{L_{eff}}$ in

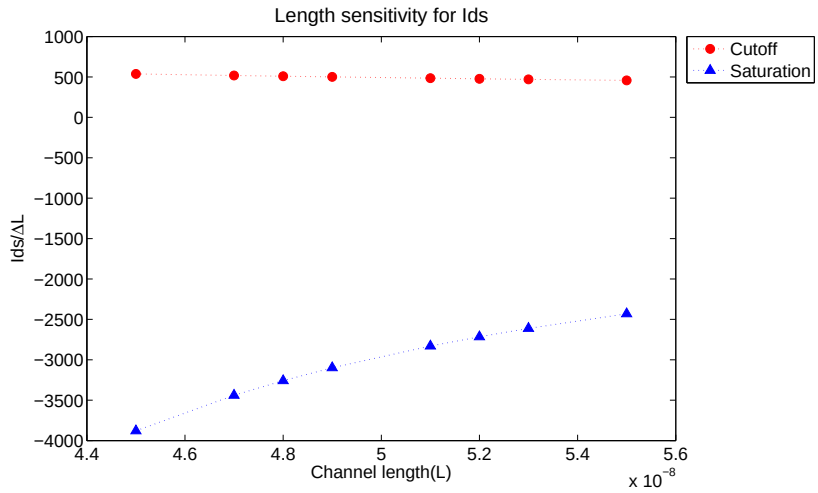


Figure 5.8: Sensitivity of length in different region

Equation (5.10) and Equation (5.11) is the dominant term, determining the general trend of the function.

Another challenge is that the length sensitivity for both the gate capacitances and I_{ds} are dependent on the transistor terminal voltages V_{ds} and V_{gs}. The trend of the length sensitivities alters with each pair of V_{ds} and V_{gs}. This can be seen more clearly through Figure 5.8

The two lines in Figure 5.8 represent the growth trends of length sensitivities of I_{ds} with respect to L_{eff} in cut-off and saturation region respectively. The trends of two lines in the graph are obviously different with each other, indicating that one single polynomial curve cannot precisely represent the length sensitivity for the entire transistor operating range. Instead, each pair of V_{gs} and V_{ds} requires an individual polynomial curve.

However, it is worth mentioning that I_{ds} is stored in a three dimension LUT indexed by V_{ds} ,V_{gs} and V_{sb} (V_{bs} for PMOS). The influence on length sensitivities from V_{sb} compared with V_{ds} and V_{gs} is not obvious. Figure 5.9 shows an example for the length sensitivity at a particular pair (V_{ds}, V_{gs}). The two curves are length sensitivities for I_{ds} when $V_{sb} = 0V$ (the start of the LUT) and $V_{sb} = 1.1V$ (the end of the LUT).

The trends of both curves in Figure 5.9 are similar to each other, which means the V_{sb} does not considerably affect the length sensitivity trend for I_{ds} compared with V_{ds} and V_{gs}. Considering the trade-off between the accuracy and complexity of the model, the length sensitivity of the model is assumed independent on V_{sb}. Hence only the length sensitivity at $V_{sb} = 0$ is calculated.

Regardless of the dimension V_{sb}, the LUT considering V_{ds} and V_{gs} is a 31×31 table, there are 31×31 polynomial curves in total to construct the new I_{ds}.

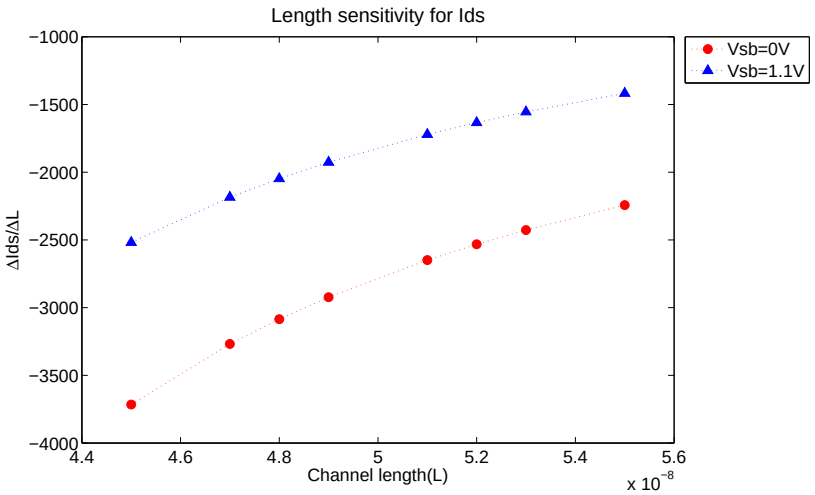


Figure 5.9: The influence on length sensitivity from V_{sb}

5.1.3.3 Determining the polynomial order for current I_{ds}

A well determined order of a polynomial curve can balance the accuracy and the cost of the data approximation. As mentioned in Section 5.1.3.2, $\frac{1}{L_{eff}}$ is set as the polynomial term. All the polynomials with degree from first to fifth have been tried in the thesis in order to achieve the optimum between accuracy and cost. A $90nm$ width NMOS transistor is chosen as an example to analyse.

In this thesis we assume the transistor length obeys a Gaussian distribution. $50nm$ is set as the mean of the length (μ_L), and the standard distribution σ_L is set to $1nm$. $3\sigma_L$ is equal to $3nm$. According to the 3σ principle, more than 99% of the transistor lengths will be located inbetween nominal length$\pm 3\sigma nm$, which is from $47nm$ to $53nm$. Thus $45nm$, $47nm$, $48nm$, $49nm$, $51nm$, $52nm$, $53nm$, and $55nm$ are used for the sampling transistor lengths. The I_{ds} values are estimated at the sampling lengths via the polynomial curve in order to compare to the existing samples characterized from BSIM4. Figure 5.10 and Figure 5.11 illustrate the maximum and average error of the I_{ds} values for the entire transistor operating range at each observation length.

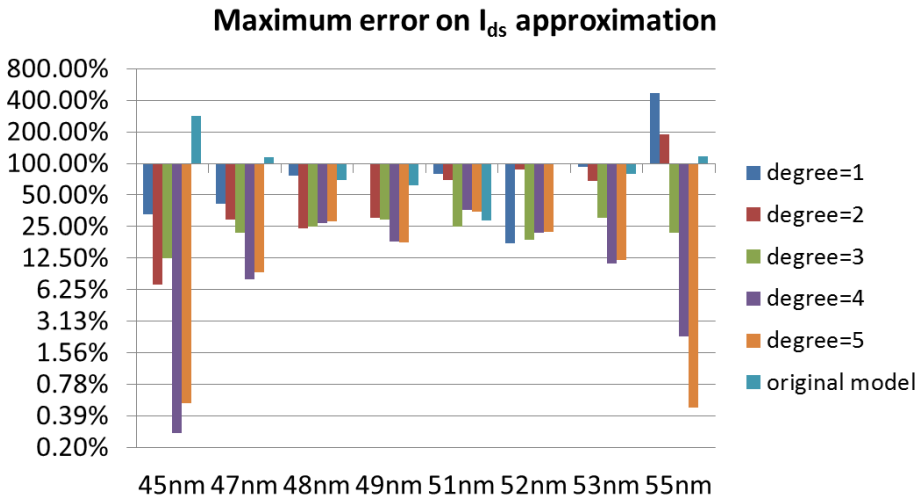


Figure 5.10: Maximum error on I_{ds} approximation

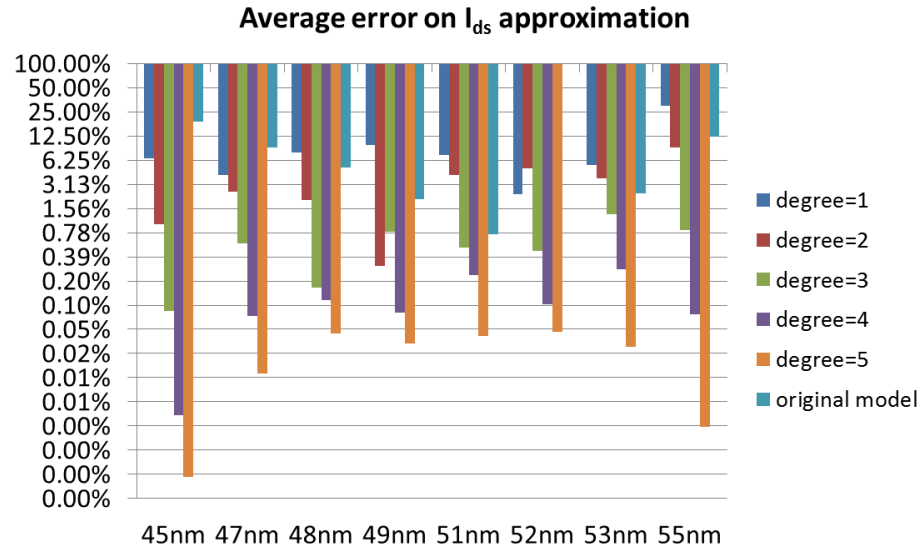


Figure 5.11: Average error on I_{ds} approximation

The figures above reflect the estimation error among polynomials with different degrees. The 1st order polynomial curve fitting strategy at $55nm$ has maximum error larger than 450%, and the corresponding average error is around 30%, which will severely impact the model accuracy. The 2nd order polynomial curve has obvious improvement in terms of accuracy compared with the 1st order polynomial curve. The average errors for all the lengths are less than 5%, but the maximum errors at $55nm$ is still as high as 200%, thus further improvement is required.

All the 3rd 4th and the 5th order polynomial curves have the maximum error about 30%. However, such error only happens once among 31×31 cases. Actually most of the huge errors are located at the cut-off region. The 'cut-off' points for different length

transistors are located at different positions in LUTs. So the I_{ds} with a particular pair of (V_{gs}, V_{ds}) can suddenly drop to three order magnitudes smaller than the previous point (for another length). Since the polynomial curve cannot follow such a fast change, increasing the polynomial order does not improve the accuracy too much. Specifically speaking, more than 99.7% of our samplings are located between $47nm$ and $53nm$. All the 3rd, 4th, 5th order polynomials have almost the same maximum error in this region. Besides, the average of the absolute errors for the 3rd order polynomial is less than 1%, which is accurate enough in the digital circuit design. The 3rd order polynomial curve is chosen for generating the new I_{ds}. However, if the standard deviation of the length is larger, the 4th order polynomial is the better option since it has large improvement for the maximum errors at length $45nm$ and $55nm$. With the same strategy, it is trivial to change the polynomial curve from the 3rd order to the 4th order. What is more, as indicated from Figure 5.11, the 5th order polynomial curve compared with the 4th order polynomial curve does not have the significant improvement any more. The maximum errors at length $48nm$ and $52nm$ are even worse than the 4th order curve. We can get the conclusion that the 4th order curve reaches the limitation for estimating the current I_{ds}.

The maximum and the average I_{ds} approximation errors with the fixed sensitivity scheme are also shown in Figure 5.10 and Figure 5.11. Since $52nm$ was chosen as sampling length to generate the sensitivity matrix, the relative error at $52nm$ is 0. For the rest of lengths, the 3rd order degree polynomial curve has obvious accuracy improvement especially for the case when the length is far from the nominal length. For instance, the maximum error at length $45nm$ drops from 250% to 10% and the average error reduces from 20% to less than 1%. Comparing with 1 LUT in the fixed sensitivity scheme, the 3rd polynomial scheme requires 4 LUTs. The complexity of the model grows very little.

In summary, If we average both the maximum and the average errors for all the sampling length from $45nm$ to $55nm$, the improvement can be concluded in Table 5.1. The improvement shows the proposed polynomial curve fitting strategy is more suitable for statistical timing analysis.

	3rd Poly. %	Original model %	Improvement %
Maximum error	22.94	93.83	70.89
Average error	0.60	6.29	5.69

Table 5.1: The improvement of estimating I_{ds} compared with the original model

5.1.3.4 Construct new current I_{ds}

As discussed in previous sections, $\frac{1}{L_{eff}}$ is the polynomial variable, and the 3rd order polynomial curves are chosen for the I_{ds} approximation. Equation (5.12) describes the

way to calculate the current $\mathbf{I_{ds}}$.

$$\mathbf{I^*_{ds}}|_{V_{sb}=0} = (\frac{1}{L_{eff}})^3 \cdot \mathbf{a} + (\frac{1}{L_{eff}})^2 \cdot \mathbf{b} + \frac{1}{L_{eff}} \cdot \mathbf{c} + \mathbf{d} \quad \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \in \mathbf{R}^{31\times31} \tag{5.12a}$$

$$\mathbf{S}|_{V_{sb}=0} = \frac{\mathbf{I^*_{ds}}|_{V_{sb}=0} - \mathbf{I_{ds\_norm}}|_{V_{sb}=0}}{L_{eff} - L_{norm}} \quad \mathbf{S}, \mathbf{I^*_{ds}}, \mathbf{I_{ds\_norm}} \in \mathbf{R}^{31\times31\times12} \tag{5.12b}$$

$$\mathbf{I^*_{ds}} = \mathbf{I_{ds\_norm}} + (L_{eff} - L_{norm}) \cdot \mathbf{S}|_{V_{sb}=0} \tag{5.12c}$$

where $\mathbf{I^*_{ds}}$ is the estimated drain-source current. $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$ are polynomial coefficients, L_{eff} is the effective channel length of the transistor, $\mathbf{S}$ is the length sensitivity, $\mathbf{I_{ds\_norm}}$ is the nominal current, and L_{norm} is the nominal length which is $50nm$.

Figure 5.12 shows an example of current I_{ds} approximation for NMOS $90nm$ width transistor with specific terminal voltages $V_{gs} = 0.8V$ and $V_{ds} = 0.8V$ (in saturation region).

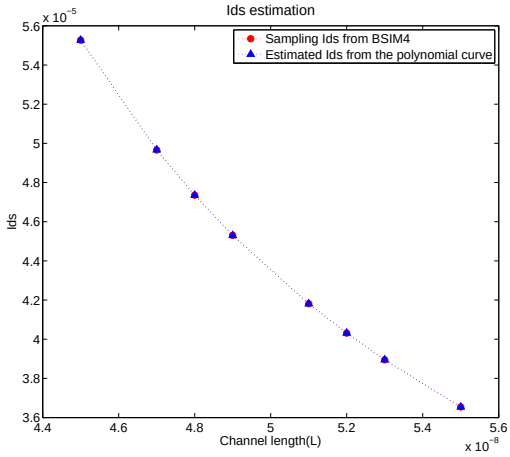


Figure 5.12: An example of I_{ds} estimation

We can see from the figure that the estimated I_{ds} is almost overlap to the original sampling values from BSIM4.

5.1.3.5 Estimating the gate capacitances

Similar to the procedure of estimating the current I_{ds}, the shape of the polynomial term and the order of the polynomial curve should be determined. There are five parasitic capacitors in the model. As discussed in Section 5.1.1 that the junction capacitances C_{db} and C_{sb} are not sensitive to the length variation, only the gate capacitances C_{gb}, C_{gd} and C_{gs} are to be estimated.

The gate capacitances are formed due to an insulator film such as SiO_2 exists between the substrate and the gate of a MOSFET. Figure 5.13 illustrates the MOSFET structure and the gate capacitances.

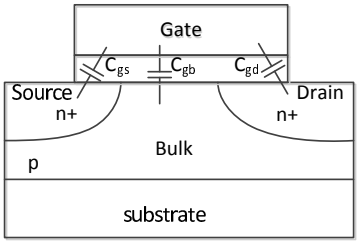


Figure 5.13: Gate capacitances of NMOS transistor

All the gate capacitors are voltage dependent, and their values alter at different operating regions of the device. Through studying paper [31], we get the following conclusions.

1. The source-drain channel in the linear region is considered uniform. The entire channel capacitance can be regarded as $C_{chanel} = WL_{eff}C_{ox}$. Therefore, the gate capacitors can be modelled as

   $C_{gs} = C_{gd} = \frac{1}{2}WL_{eff}C_{ox}$.

   where C_{ox} is the capacitance per unit gate area, W is the width of the transistor, and L_{eff} is the effective channel length of the transistor.

2. The channel at the saturation region is not uniform, it has a tapered shape and is pinched off near the drain end. The gate capacitors in the saturation region can be roughly modelled as

   $C_{gs} \approx \frac{2}{3}WL_{eff}C_{ox}$ $C_{gd} = C_{gb} = 0$.

3. In the cut-off region, the channel is not formed. The gate capacitors can be modelled as

   $C_{gs} = C_{gd} = 0$ $C_{gb} = WL_{eff}C_{ox}$.

The gate capacitive effects in three operating regions of MOSFET are presented as in Table 5.2

	Operating Region		
Capacitance	Cut-off	Linear	Saturation
C_{gb}	$C_{ox}WL_{eff}$	0	0
C_{gd}	0	$\frac{1}{2}C_{ox}WL_{eff}$	0
C_{gs}	0	$\frac{1}{2}C_{ox}WL_{eff}$	$\frac{2}{3}C_{ox}WL_{eff}$

Table 5.2: Gate capacitances in three operating regions

We can conclude from Table 5.2 that all gate capacitances grow linearly with the effective transistor length L_{eff}. However, the formulas in Table 5.2 are the rough estimations, some effects such as the overlapping capacitances are neglected. Figure 5.14

is plotted as an experimental exploration of the length sensitivities for the gate capacitances. This experiment chooses $90nm$ width NMOS transistor with terminal voltages $V_{gs} = 0.8V$ and $V_{ds} = 0.8V$, which make the transistor running in the saturation region.

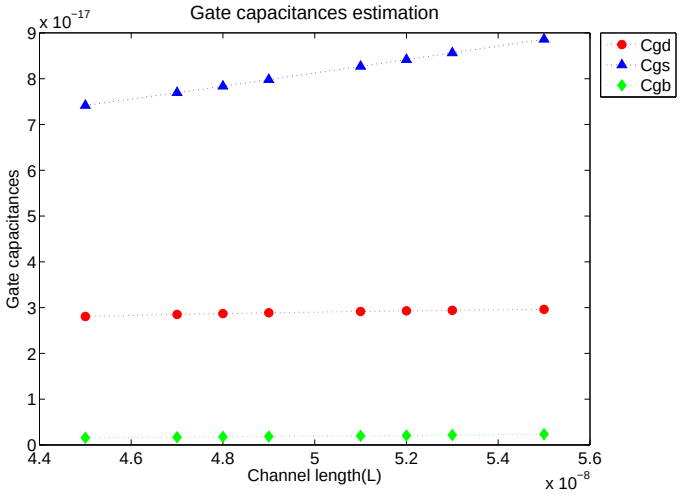


Figure 5.14: Gate capacitors length sensitivities

The three curves represent the length sensitivities for gate capacitances i.e., C_{gd}, C_{gs} and C_{gb}. Two conclusions can be drawn from observing Figure 5.14.

1. The C_{gb} is less sensitive to the transistor length. The average of the length sensitivities for all the sampling lengths of the three gate capacitances are calculated. The average length sensitivity for C_{gb} is 6.1125×10^{-18}, while for C_{gs} is 7.6092×10^{-10}, and for C_{gd} is 2.3158×10^{-10}. The length sensitivity for C_{gb} compared with the others can be neglected. Thus C_{gb} is regarded as a constant matrix during the statistical timing analysis.

2. The gate capacitances C_{gs} and C_{gd} change almost linearly with the effective channel length L_{eff}. Thus L_{eff} is selected as the polynomial term. Since the contributions to the model accuracy from the gate capacitances are not as critical as the drain-source current I_{ds}, a 1st order polynomial curve is employed to approximate gate capacitances.

Equation (5.13) shows the way to compute gate capacitances at a given length.

$$\mathbf{C^*_{gate}} = L_{eff} \cdot \mathbf{a} + \mathbf{b} \quad \mathbf{C^*_{gate}}, \mathbf{a}, \mathbf{b} \in \mathbf{R}^{31 \times 31} \tag{5.13}$$

Where $\mathbf{C^*_{gate}}$ is the new gate capacitances, $\mathbf{a}$, $\mathbf{b}$ are coefficients of the polynomial curves, L_{eff} is the effective channel length of the transistor. The polynomial coefficients are stored in LUTs and are loaded into the model before the model is compiled.

The relative average errors of the estimated gate capacitances compared with sampling values are summarized in Table 5.3.

Generally speaking, the maximum errors for both C_{gs} and C_{gd} estimation are less than 10%, and the average errors are less than 1%. In summary, the gate capacitances

	Relative error C_{gs}(%)		Relative Error C_{gd}(%)	
Channel length(nm)	Maximum	Average	Maximum	Average
45	6.77	0.31	2.79	0.32
47	1.82	0.068	0.69	0.049
48	3.86	0.14	1.26	0.13
49	4.74	0.18	1.37	0.17
51	3.26	0.14	1.08	0.13
52	1.88	0.071	0.56	0.071
53	0.81	0.028	0.26	0.013
55	6.19	0.24	1.79	0.23

Table 5.3: Relative error of gate capacitances estimation with one order polynomial curve fitting

approximated by the 1st order polynomial curves are precise for statistical timing analysis. The implementation of the Matlab script to generate the polynomial coefficients can be found at Appendix C

5.2 Monte Carlo simulation of the proposed transistor model

Monte Carlo (MC) method is an analytical technique to repeat the same computation with large amount of times, and vary the uncertain variable randomly at each iteration. The distribution of the outputs indicates how possible such a result can happen. MC simulation is straightforward and reliable, which is frequently used in circuit simulation to evaluate and estimate the timing behaviour of a circuit concerning transistor process variations. The process variation of a transistor is uncertain, the statistical feature of the transistor model is tested by running the same circuit simulation thousands of times, and randomly changes the process variation in each iteration of the simulation. Mean and standard deviation are two typical parameters to describe a distribution in statistics.

According to the discussion in Section 5.1.3.3 and Section 5.1.3.5, 1st order polynomial curves are used to approximate the intrinsic capacitances and 3rd order polynomial curves are used for estimating the source-drain current I_{ds}. Those polynomial coefficients are stored in LUTs, which are loaded statically into the model to avoid the expensive file operations. Figure 5.15 shows the concept for generating the statistical model.

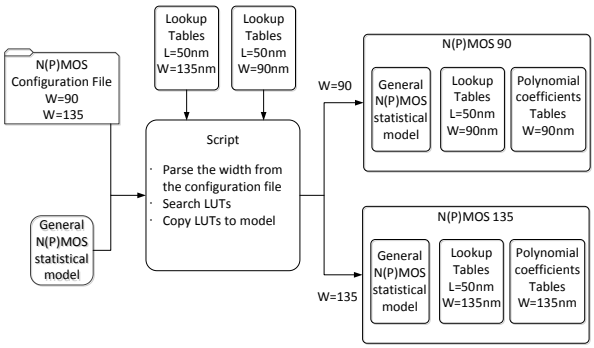


Figure 5.15: Load polynomial coefficients LUTs

As shown in Figure 5.15, each complete model has three ingredients. 1). A general NMOS or PMOS statistical model including all calculation functions 2). LUTs for the components of the nominal model (intrinsic capacitances and I_{ds}) with the specified transistor width. 3). The polynomial coefficients LUTs with the specified transistor width. The user should indicate the desiring width in the configuration file, such as $90nm$ and $135nm$ in Figure 5.15. The corresponding LUTs are searched and loaded to form a width determined model which is ready to compile.

5.3 Experimental Results

Similar to the deterministic timing analysis, evaluating a single gate composed of the proposed SSTM is the first step for statistical timing analysis. The transistor model is aware of process variations in length. In this project, we assume transistor lengths obey Gaussian distribution with mean μ_L of $50nm$ and standard deviation σ_L of $1nm$. The transistor length is regarded as a parameter passed to the model. 10000 iterations MC transient analysis are running.

5.3.1 Monte Carlo Simulation on Single Gate

The inverter is one of the elementary logic gates, the structure of which can be simply a single NMOS and a PMOS transistor connected in series. The statistical properties for an inverter with drive strengths X1, X2, and X4 are checked in this section. The testing environment is set up as below.

- Both rising and falling input signals are tested.
- According to the range ($7.5ps$ to $600ps$) given by the NanGate library (Table 4.1), the input slews were set as $15ps$ for all the inverter.
- According to the driving ability from Table 4.1, the output capacitance loads are chosen among $1.6fF$, $3.2fF$ and $6.4fF$.
- Other parameters such as the power supply, the temperature etc. stay constant as during the deterministic timing analysis.

Sine both 30% and 70% crossing points of the output signals are necessary for calculating the output slew, the two time points are tested individually. The testing results of MC simulation on inverters are shown in Figure 5.16, Figure 5.17, and Figure 5.18.

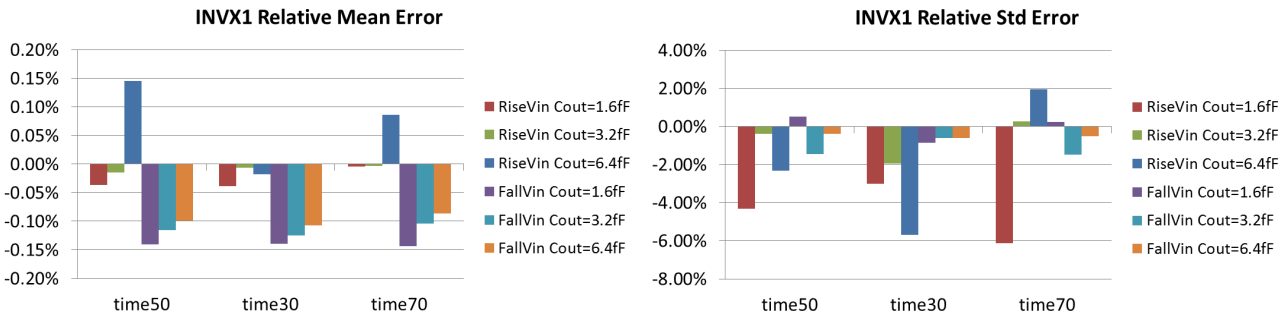


Figure 5.16: Relative time points errors for INVX1 compared with BSIM4

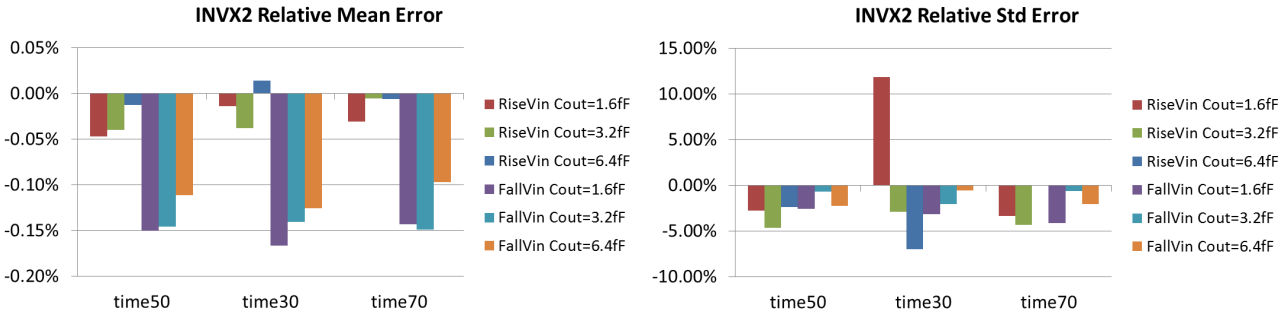


Figure 5.17: Relative time points errors for INVX2 compared with BSIM4

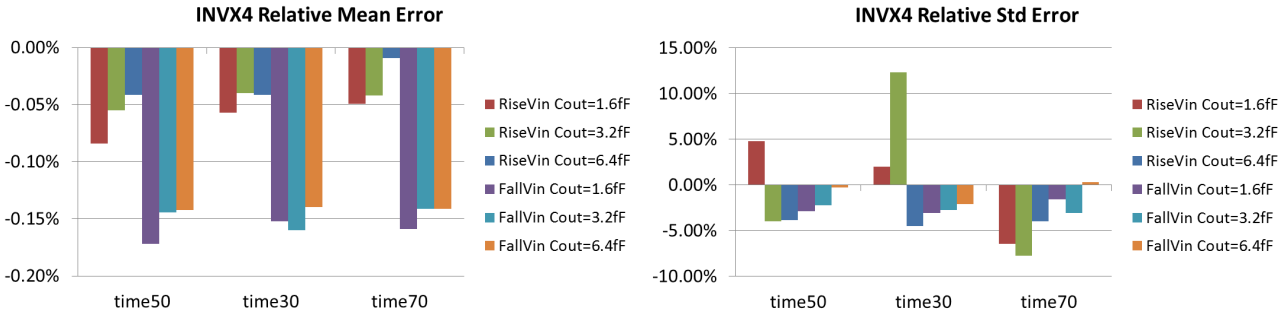


Figure 5.18: Relative time points errors for INVX4 compared with BSIM4

The mean error of all the time points are less than 1%, which means the output waveform of the proposed model at the concerning points ($t_{30\%}$, $t_{50\%}$ and $t_{70\%}$) on average are very close to BSIM4 model. Standard deviation reflects the spread of the arrival times to their averages. During the test, most of the gates have the relative standard deviation errors less than 5%. However large errors happen sometimes. For

instance, the error of 30% crossing point for inverter X2 with $1.6fF$ output loads is higher than 10%, the distribution of which is analysed in Figure 5.19.

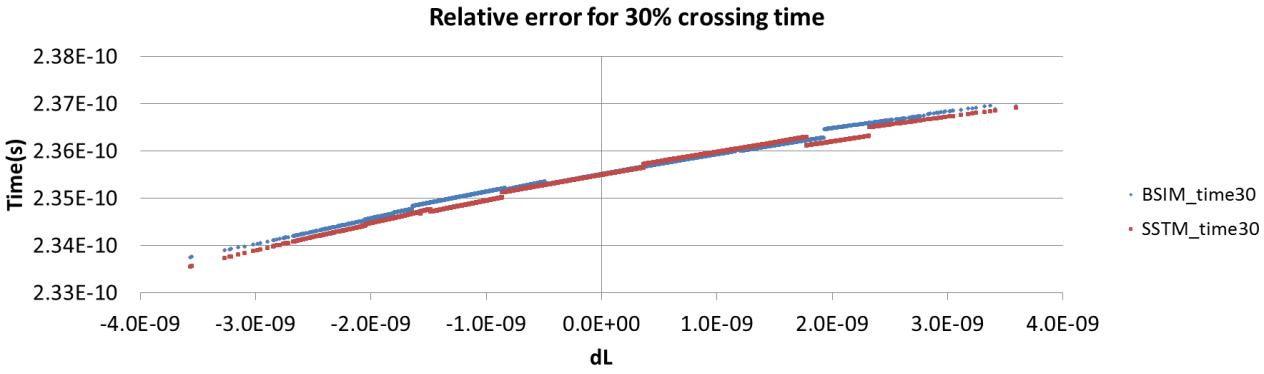


Figure 5.19: Arrival time of 70% V_{dd} with respect to the transistor length

Figure 5.19 illustrates the arrival time when the output voltages cross 30% of the power supply V_{dd}. The two lines are testing results from both SSTM and BSIM4. We can see that the line for SSTM grows twisted with the line for BSIM. Apart from that, it is interesting to notice that the arrival time of the 30% crossing point of BSIM does not grow continuously with the transistor length, so does the value of SSTM. The jumping points cannot be predicted and approximated appropriately in the proposed SSTM. The standard deviation error grows when jumps happen more frequently. However, the reason why jumps exist in BSIM is beyond the scope of this thesis, thus further study needs to be done.

In addition, the testing results reflect the fact that the standard deviation error for the output slew is much more higher than the individual time points ($t_{30\%}$ and $t_{70\%}$).

The output slew is defined as $t_{30\%} - t_{70\%}$. The way of computing the standard deviation for the difference of the two points is shown by Equation (5.14)

$$\sigma(t_{30\%} - t_{70\%}) = \sqrt{\sigma^2 t_{30\%} + \sigma^2 t_{70\%} - 2cov(t_{30\%}, t_{70\%})} \tag{5.14}$$

Where $cov(t_{30\%}, t_{70\%})$ is the covariance between points $t_{30\%}$ and $t_{70\%}$, which describes how much two variables tend to be similar to each other. The covariance can be calculated by Equation (5.15)

$$cov(t_{30\%}, t_{70\%}) = E[(t_{30\%} - E(t_{30\%})(t_{70\%} - E(t_{70\%})] = \rho_{t_{30\%}, t_{70\%}} \sigma t_{30\%} \sigma t_{70\%} \tag{5.15}$$

where $\rho_{t_{30\%}, t_{70\%}}$ is the correlation coefficient, which describes the dependence degree of two data sets ($t_{30\%}$, $t_{70\%}$). If we take inverter X1 as an example, the correlation coefficient for BSIM is 0.981, while the correlation coefficient for SSTM is 0.99. It means $t_{30\%}$ and $t_{70\%}$ have very strong correlation with each other for both model. Hence the Equation (5.14) can be approximated as

$$\sigma(t_{30\%} - t_{70\%}) \approx \sigma t_{30\%} - \sigma t_{70\%} \tag{5.16}$$

If the standard deviation $\sigma t_{30\%}$ and $\sigma t_{70\%}$ are very close to each other, their difference $\sigma t_{30\%} - \sigma t_{70\%}$ can be very small, and the relative error of the difference in such case will become huge easily. The covariance refers to the second moments computing of two jointly distributed variables. The standard deviation of the output slew is not close to BSIM4 model any more since the two points $t_{30\%}$ and $t_{70\%}$ are not two independent events. The implementation method of the SSTM guarantees the accuracy of single variable distribution till the second moments computing, because the standard deviation error of a single time point can be guaranteed. However, the accuracy for the jointly distributed variables is not enough. Improving the interpolation methods and increasing the LUT resolution are necessary for higher accuracy requirements.

The structures of NAND and NOR gates compared to an inverter are slightly more complicated. NAND2X2 and NOR2X2 are tested in this thesis. The testing environments such as input slew, the length distributions output loads are set same as for the inverter. Figure 5.20 and Figure 5.21 show the testing results.

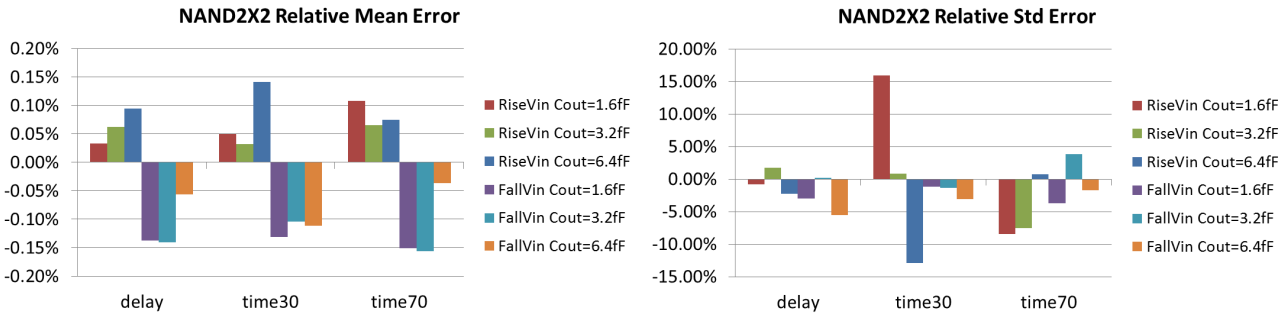


Figure 5.20: Relative time points errors of NAND2X2 compared with BSIM4

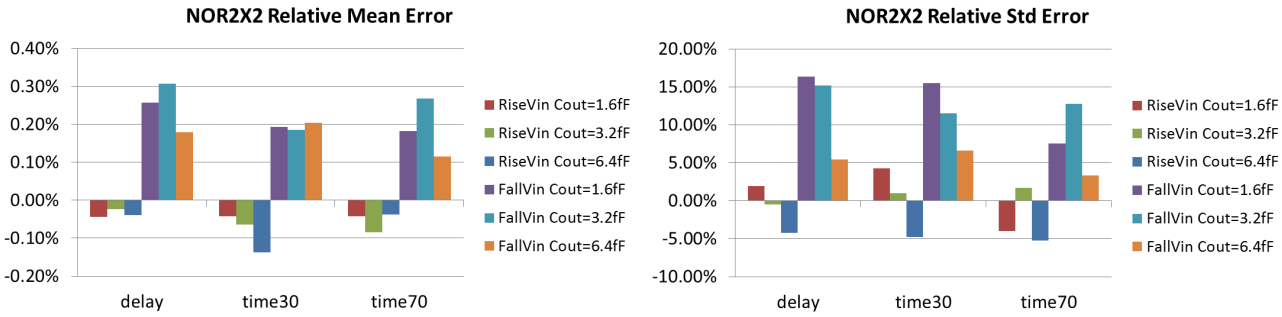


Figure 5.21: Relative time points error of NOR2X2 compared with BSIM4

The relative standard deviation errors for NOR2 gate with small output loads and falling inputs are around 15%. The error caused from the falling input is larger than the one from rising input, and the error reduces when output loads increase. The reasons can be explained in the same way as the deterministic timing analysis.

5.3.2 Monte Carlo Simulation on Inverter Chain

The same as the testing procedure introduced in the deterministic timing analysis, the benchmark circuits are used for circuit level testing after the accuracy of standard cells composed of the proposed SSTM is tested. Characterizing transistors for statistical timing analysis requires large amount of time because one transistor width is corresponding to 9 transistors lengths from $45nm$ to $55nm$ in the implementation, and all the required widths and lengths of the transistors need to be characterized. Here we choose a 7 stages inverter chain as the benchmark circuit since the characterization for the transistor used to build up the inverter with size from X1 to X4 are already available when doing single inverter gate analysis.

All the 7 gates share the same length deviation variable. In other words, all the lengths of transistors in the circuit increase or decrease by the same amount at same time in order to simplify the testing procedure. The testing environments are listed as below

- Both falling and rising inputs are tested.
- The input slew is set to $25ps$.
- The inverter chain is composed of 7 identical INV_X1s, each of the inverter is followed by a $5fF$ output load to represent the wire.
- Other statistical parameters such as mean and standard deviation of transistor length and environment parameters like temperatures and voltage supply have the same configuration as in the standard cell testing.

The relative mean and standard deviation error against BSIM4 are compared gate by gate. Figure 5.22 is the testing results for inverter chain with both rising and falling input. The bars indicate the mean error and the corresponding standard deviation errors for each individual gate respectively.

The mean errors for all stages with both rising and falling are less than 1%, and the maximum standard deviation error is around -4% at the second stage with rising input. The final output delay and slew are summarized in Table 5.4.

Input type	Delay error(%)		Slew error(%)	
	mean μ	standard deviation σ	mean μ	standard deviation σ
R	-0.19	-0.49	0.03	0.20
F	-0.23	-0.55	0.05	0.24

Table 5.4: statistical timing analysis for 7 stages inverter chain

The output delay error of the whole circuits is the addition of the individual errors. The effect of the positive and negative values are just cancelled. The eventual waveform of the inverter chain is very close to the circuit with BSIM4 model. In summary, the small error compared with BSIM4 for inverter chain indicates the proposed SSTM is accurate for statistical timing analysis at the circuit level.

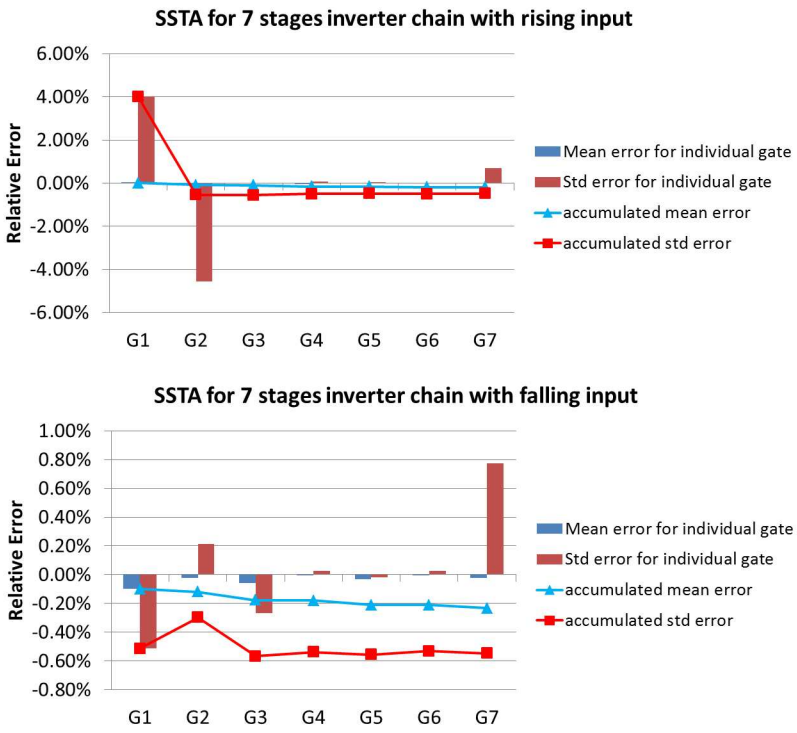


Figure 5.22: statistical timing analysis on 7 stages INV chain

6 Conclusion

6.1 Thesis contribution

This thesis implemented and evaluated a simplified transistor model, such that the model can be installed on commercial simulation engines, such as Spectre. It allows a direct comparison of the proposed model with the sophisticated model BSIM4. What is more, the model is extended into the statistical domain for SSTA due to the transistor process variations. Using the polynomial curve fitting scheme to estimate the components of the model was proposed in this thesis. With such a scheme, the estimation accuracy of the model components such as I_{ds} has been improved. The polynomial scheme has approximately 70.89% improvement of the maximum error and 5.69% improvement of the average error in terms of estimating I_{ds} compared to the original scheme.

6.2 Thesis work summary

The background of STA was firstly introduced in the thesis, which includes why the STA is essential for the digital circuit design and how GLM works for STA. The limitations of GLM were explained and it was raised as the main problem of the thesis. A new simplified transistor model was proposed by Qin Tang and the concept of the model was discussed. The goal of the thesis was to implement the new model such that the model can be plugged into the popular commercial circuit simulation software Spectre from Cadence. The possible ways to install the model were discussed and the interface of the Spectre was introduced as well. The proposed model and the golden model BSIM4 were run on the same simulation engine, and the net error caused by the proposed model was measured. Furthermore, the model was extended to process variations awareness which allows the SSTA performing on integrated circuits. The polynomial curve fitting method was brought to the transistor length interpolation, which significantly improved the accuracy and the available length range of the model without increasing the complexity of the model too much. The improvement is obvious especially when the length varies far from the nominal value since the model at the early design stage applied the fixed sensitivity approximation approach. Both deterministic STA and SSTA were performed to test the accuracy of the model. Among deterministic STA, followed by the DC analysis, we tested the standard cells from NanGate library, and the critical path of ISCAS-85 benchmark circuits. In terms of SSTA, the typical standard cells such as inverter, NAND2 and NOR2 were tested. A seven stage inverter chain was simulated in order to get the circuit level performance of the proposed model. The possible reasons for big errors in testing results were discussed, for some the solutions were given.

6.3 Future work

Of course, there are still spaces to improve the project work. Future work can be divided into two categories. On one hand, to refine the model implementation method. For instance, to come up with a better interpolation scheme to estimate components of model accurately and faster. On the other hand, as mentioned in Figure 5.19, the timing behaviour of BSIM4 model jumps at some certain lengths, and so does the SSTM. The reasons are not clear yet. One guess can be from the model itself, the internal function of BSIM4 is not continues with the length variation. Another guess can be from the simulation engine, the numerical method is discrete such that the step size is not appropriately set. Furthermore, the relative errors for some standard gates compared with BSIM4 are large. Although the possible reasons were given in the thesis, further investigations are still needed. Solutions can be given for further refining the model.

Bibliography

[1] J. Bhasker and R. Chadha. *Static Timing Analysis for Nanometer Designs: A Practical Approach*. Springer, 2009.

[2] Qin Tang, Amir Zjajo, Michel Berkelaar, and Nick van der Meijs. Transistor-level gate modeling for nano cmos circuit verification considering statistical process variations. In *Proceedings of the 20th international conference on Integrated circuit and system design: power and timing modeling, optimization and simulation*, PATMOS'10, 2011.

[3] A. Goel and S. Vrudhula. Statistical waveform and current source based standard cell models for accurate timing analysis. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 227 –230, june 2008.

[4] Chirayu S. Amin, Florentin Dartu, and Yehea I. Ismail. Weibull based analytical waveform model. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, ICCAD '03, Washington, DC, USA, 2003. IEEE Computer Society.

[5] Chirayu Amin, Chandramouli Kashyap, Noel Menezes, Kip Killpack, and Eli Chiprout. A multi-port current source model for multiple-input switching effects in cmos library cells. In *Proceedings of the 43rd annual Design Automation Conference*, DAC '06, pages 247–252, New York, NY, USA, 2006. ACM.

[6] Qin Tang, Amir Zjajo, Michel Berkelaar, and Nick van der Meijs. Rde-based transistor-level gate simulation for statistical static timing analysis. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 787–792, New York, NY, USA, 2010. ACM.

[7] UC Berkely Device Group. Bsim4. Available online: http://www-device.eecs.berkeley.edu/bsim/?page=BSIM4.

[8] Pawan Kulshreshtha, Robert Palermo, Mohammad Mortazavi, Cyrus Bamji, and Hakan Yalcin. Transistor-level timing analysis using embedded simulation. In *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, ICCAD '00, pages 344–349, Piscataway, NJ, USA, 2000. IEEE Press.

[9] S. Raja, F. Varadi, M. Becer, and J. Geada. Transistor level gate modeling for accurate and fast timing, noise, and power analysis. In *Proceedings of the 45th annual Design Automation Conference*, DAC '08, pages 456–461, New York, NY, USA, 2008. ACM.

[10] Modeling and design of reliable, process variation-aware nanoelectronic device (MODERN).

[11] Ashish Nigam. Standard cell behavior analysis and waveform set model for statistical static timing analysis. *Master Thesis TUDelft*, pages 10 –11, june 2010.

[12] A. Goel and S. Vrudhula. Current source based standard cell model for accurate signal integrity and timing analysis. In *Design, Automation and Test in Europe, 2008. DATE '08*, pages 574 –579, march 2008.

[13] T. El Motassadeq. CCS vs NLDM comparison based on a complete automated correlation flow between primetime and hspice. In *Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International*, pages 1–5, april 2011.

[14] Synopsys CCS timing technical white paper, 2005. Available online: http://www.opensourceliberty.org/ccspaper/ccs_timing_wp.pdf.

[15] Keith E. Holbert. Nodal analysis tutorial, 2009. Available online: http://holbert.faculty.asu.edu/ece201/nodalanalysis.html.

[16] Farid N. Najm. *Circuit Simulation*. Wiley-IEEE Press, Hoboken, NJ, USA, 2010.

[17] L.M. Wedepohl and L. Jackson. Modified nodal analysis: an essential addition to electrical circuit theory and analysis. *Engineering Science and Education Journal*, 11(3):84 –92, jun 2002.

[18] L. Pillage. *Electronic Circuit & System Simulation Methods (SRE)*. McGraw-Hill Companies,Incorporated, 1998.

[19] L.W. Nagel. *SPICE2: a computer program to simulate semiconductor circuits*. Memorandum. Electronics Research Laboratory, College of Engineering, University of California, 1975.

[20] B.J. Sheu, D.L. Scharfetter, P.-K. Ko, and M.-C. Jeng. BSIM: Berkeley short-channel IGFET model for mos transistors. *Solid-State Circuits, IEEE Journal of*, 22(4):558 – 566, aug 1987.

[21] G.J. Coram. How to (and how not to) write a compact model in verilog-a. In *Behavioral Modeling and Simulation Conference, 2004. BMAS 2004. Proceedings of the 2004 IEEE International*, pages 97–106, oct. 2004.

[22] Wladek Grabinski Paolo Nenzi. Compact model standardization a GNU perspective. 2008.

[23] Analog and mixed-signal standard for Verilog HDL moves forward, 2000. Available online: http://www.prnewswire.com.

[24] Diana Moncoqut. Hicum model in Spectre, 2004. Available online: http://www.iee.et.tu-dresden.de.

[25] P. Van Der Linden. *Expert C Programming: Deep C Secrets*. Safari Tech Books Online. SunSoft Press, 1994.

[26] Unniversity of Alberta. Understanding memory, 2010. Available online: http://www.ualberta.ca/CNS/RESEARCH/LinuxClusters/mem.html.

[27] Affirma spectre circuit simulator user guide, 1999. Available online: http://bwrc.eecs.berkeley.edu/designtools/module_design/.

[28] Sergey Sukharev. New enhancements in ADMS and Spectre CMI XML scripts. March 2006.

[29] E.W. Cheney and W.A. Light. *A Course in Approximation Theory*. Graduate Studies in Mathematics. American Mathematical Society, 2000.

[30] NanGate Website. Available online: http://www.nangate.com.

[31] Nebi Caka, Milaim Zabeli, Myzafere Limani, and Qamil Kabashi. Impact of mosfet parameters on its parasitic capacitances. In *Proceedings of the 6th WSEAS International Conference on Electronics, Hardware, Wireless and Optical Communications*, EHAC'07, pages 55–59, Stevens Point, Wisconsin, USA, 2007. World Scientific and Engineering Academy and Society (WSEAS).

[32] S. Raja, F. Varadi, M. Becer, and J. Geada. Transistor level gate modeling for accurate and fast timing, noise, and power analysis. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 456 –461, june 2008.

[33] M.C. Hansen, H. Yalcin, and J.P. Hayes. Unveiling the iscas-85 benchmarks: a case study in reverse engineering. *Design Test of Computers, IEEE*, 16(3):72 –80, 1999.

[34] S.S. Sapatnekar. *Timing*. Kluwer Academic Publishers, 2004.

[35] Brendan Hargreaves, Henrik Hult, and Sherief Reda. Within-die process variations: how accurately can they be statistically modeled? In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, ASP-DAC '08, pages 524–530, Los Alamitos, CA, USA, 2008. IEEE Computer Society Press.

[36] Michel Berkelaar. Statistical delay calculation, a linear time method. 1997.

[37] Toshiyuki Shibuya. Izumi Nitta and Katsumi Homma. Statistical static timing analysis technology. *FUJITSU Sci.Tech.J.*, 19(3):518 –523, april 2007.

[38] A. Bjõrck. *Numerical Methods for Least Squares Problems*. Miscellaneous Bks. Society for Industrial and Applied Mathematics, 1996.

[39] M. Tadeusiewicz. Modeling and stability in mos transistor circuits. In *Electronics, Circuits and Systems, 1998 IEEE International Conference on*, volume 1, pages 71 –74 vol.1, 1998.

[40] J. Krumm. *Circuit Analysis Methodology for Organic Transistors*. 2008.

Appendix A

The proposed simplified transistor model (SSTM)

```
//
    ********************************************************************************

//Nmos.vams
//Created 6 Feb, 2012
//Last changed 9 May, 2012
//Version 6.1 ; Author Xinyue
//a simple NMOS transistor
//Description: This is a simple transistor model based on the paper. It
    Takes 'Vd', 'Vg', 'Vs' and 'Vb' as inputs
//The model parameters 'Cgd', 'Cgs', 'Cgb', 'Cdb', 'Csb' are from the LUT
    . Each of them is stored in an one dimension array.
//NMOS LUT horizontal axis: vds[-0.2,1.3]; vertical axis: vgs[-0.2,1.3];
    step size 0.1v; 3rd dimension Isb[0,1.1]; step size 0.05v; 31 grids in
     total
//PMOS LUT horizontal axis: vds[-1.3,0.2]; vertical axis: vgs[-1.3,0.2];
    step size 0.1v; 3rd dimension Ibs[0,1.1]; step size 0.05v; 31 grids in
     total
//A sensitive length parameter is added for Montecarlo Simulation
//
    ********************************************************************************


//use the following construct in order to use this example in other
    simulators.

`include "constants.h"
`include "discipline.h"

`define LUTSTEP      0.05
`define HALFLUTSTEP      0.025
`define ISBLUTSTEP       0.1
`define ISBHALFLUTSTEP   0.05
`define STARTGRIDVOLT    -0.2
`define ENDGRIDVOLT      1.3
`define SBSTARTGRIDVOLT 0
`define SBENDGRIDVOLT    1.1
`define GRIDSPERLINE     31
`define ISBGRIDSPERLINE 12
`define SMALLNUMBER      1.38778e-25
`define NOMLEN       50e-9

```

```
//declaration of mymodel **The order of the nodes should be consistent
   with the order in netlist
module mynmos(d,g,s,b);
    inout d, g, s, b;
    electrical d,g,s,b;

    //Branch definitions
    branch (g,s) br_gs;
    branch (g,d) br_gd;
    branch (g,b) br_gb;
    branch (d,s) br_ds;
    branch (d,b) br_db;
    branch (s,b) br_sb;

    parameter real dL=0 from[-50e-9:50e-9];

    //insert LUT from script

    //local variables declaration

    //define indexes. ind_gs1, ind_gs2, ind_ds1, ind_ds2 are the nearest
       point on axis around Vgs and Vds, final_ind_gs1, final_ind_gs2,
       final_ind_ds1, final_ind_ds2, final_ind_sb1, final_ind_sb2 are the
        corresponding indexes after inter/extra polation
    integer ind_gs1, ind_gs2, ind_ds1, ind_ds2, ind_sb1, ind_sb2, x1, x2,
        x3, x4, final_ind_gs1, final_ind_gs2, final_ind_ds1,
       final_ind_ds2, final_ind_sb1, final_ind_sb2;
    //slp are used for interpolation
    real slpg, slpd;
    //------------C_x1-----C_1------C_x2-----------
    //----------------------C_----------------------
    //------------C_x3-----C_2------C_x4-----------
    //Cxx[x1],Cxx[x2],Cxx[x3],Cxx[x4] are grids in LUT, which are the
       nearest points around Cxx. Cxx is obtained with bilinear
       interpolation from Cxx[x1],Cxx[x2],Cxx[x3],Cxx[x4]
    real Cdbx1, Cdbx2, Cdbx3, Cdbx4, Cdb1, Cdb2, Cdb, Cgbx1, Cgbx2, Cgbx3
       , Cgbx4, Cgb1, Cgb2, Cgb, Cgdx1, Cgdx2, Cgdx3, Cgdx4, Cgd1, Cgd2,
       Cgd, Cgsx1, Cgsx2, Cgsx3, Cgsx4, Cgs1, Cgs2, Cgs, Csbx1, Csbx2,
       Csbx3, Csbx4, Csb1, Csb2, Csb;
    //------------Ids1x1---Ids11--------Ids1x2-----------
    //----------------------Ids1-------------------------
    //------------Ids1x3---Ids12--------Ids1x4-----------

    //------------Ids2x1----Ids21-------Ids2x2-----------
    //----------------------Ids2-------------------------
    //------------Ids2x3----Ids22-------Ids2x4-----------
    real Ids1x1, Ids1x2, Ids1x3, Ids1x4, Ids2x1, Ids2x2, Ids2x3, Ids2x4,
       Ids11, Ids12, Ids1, Ids21, Ids22, Ids2, Ids;
    //Vgs, Vds, Vsb are the voltages across the nodes of the transistor.
    real Vgs, Vds, Vsb;
//**parameters for MC simulation**
//MC has exactly the same variables as the statical case
```

```
        real Idscoef_a1, Idscoef_a2, Idscoef_a3, Idscoef_a4, Idscoef_b1,
            Idscoef_b2, Idscoef_b3, Idscoef_b4, Idscoef_c1, Idscoef_c2,
            Idscoef_c3, Idscoef_c4, Idscoef_d1, Idscoef_d2, Idscoef_d3,
            Idscoef_d4;
        real Cgdcoef_a1, Cgdcoef_a2, Cgdcoef_a3, Cgdcoef_a4, Cgdcoef_b1,
            Cgdcoef_b2, Cgdcoef_b3, Cgdcoef_b4;
        real Cgscoef_a1, Cgscoef_a2, Cgscoef_a3, Cgscoef_a4, Cgscoef_b1,
            Cgscoef_b2, Cgscoef_b3, Cgscoef_b4;
        real sensIds1,sensIds2,sensIds3,sensIds4;
76
        real  MCIds1x1, MCIds1x2, MCIds1x3, MCIds1x4, MCIds2x1, MCIds2x2,
            MCIds2x3, MCIds2x4, MCIds11, MCIds12, MCIds1, MCIds21, MCIds22,
            MCIds2, MCIds;
        real MCCgdx1,MCCgdx2,MCCgdx3,MCCgdx4, MCCgd1, MCCgd2, MCCgd;
        real MCCgsx1,MCCgsx2,MCCgsx3,MCCgsx4, MCCgs1, MCCgs2, MCCgs;
        real L_newCgate, L_newIds, L_newIds2, L_newIds3;
81      real Ids1x1_norm, Ids1x2_norm, Ids1x3_norm, Ids1x4_norm;

        analog begin

            //position location: Get the index value according to the input
                voltage.
86          //Bilinear interpolation on variables x and y.
            //fix the bug, the indexes are according to Vgs instead of Vg,
                Vds instead of Vd, respectively. they have same value in NMOS,
                 but different in PMOS
            begin
    //Find the corresponding range [ind_gs1, V(gs) ,ind_gs2) according to V(
        gs).
    //HALFLUTSTEP=step(LUTSTEP)/2. Here we minus a number (HALFLUTSTEP-
        SMALLNUMBER) to avoid overflow on grid point  //SMALLNUMBER is just a
        random small value
91              ind_gs1 = (V(br_gs)-(`HALFLUTSTEP-`SMALLNUMBER)-(
                    `STARTGRIDVOLT))/`LUTSTEP;
                ind_gs2 = ind_gs1 + 1;
                ind_ds1 = (V(br_ds)-(`HALFLUTSTEP-`SMALLNUMBER)-(
                    `STARTGRIDVOLT))/`LUTSTEP;
                ind_ds2 = ind_ds1 + 1;
                ind_sb1 = (V(br_sb)-(`ISBHALFLUTSTEP-`SMALLNUMBER)-
                    `SBSTARTGRIDVOLT)/`ISBLUTSTEP;
96              ind_sb2 = ind_sb1 + 1;


        //ind_gs normal case: ind_gs belongs to [0,30], which means vgs
            belongs to [-0.2,1.3]
                if(ind_gs2<=(`GRIDSPERLINE-1) && ind_gs1>=0)begin
101                 final_ind_gs1 = ind_gs1;
                    final_ind_gs2 = ind_gs2;
                    Vgs = V(br_gs);
                end
        //ind_ds normal case: ind_gs belongs to [0,30], which means vds
            belongs to [-0.2,1.3]
106             if(ind_ds2<=(`GRIDSPERLINE-1) && ind_ds1>=0)begin
```

```
                final_ind_ds1 = ind_ds1;
                final_ind_ds2 = ind_ds2;
                Vds = V(br_ds);
            end
//***extrapolation on Vgs and Vds;zero order is applied here***
            // Vgs>=ENDGRIDVOLT
            if(ind_gs2>=`GRIDSPERLINE)begin
                final_ind_gs1 = (`GRIDSPERLINE-2);
                final_ind_gs2 = (`GRIDSPERLINE-1);
                Vgs = `ENDGRIDVOLT;
            end
            // Vds>=ENDGRIDVOLT
            if(ind_ds2>=`GRIDSPERLINE)begin
                final_ind_ds1 = (`GRIDSPERLINE-2);
                final_ind_ds2 = (`GRIDSPERLINE-1);
                Vds = `ENDGRIDVOLT;
            end
            //Vgs<STARTGRIDVOLT
            if(ind_gs1<0)begin
                final_ind_gs1 = 0;
                final_ind_gs2 = 1;
                Vgs = `STARTGRIDVOLT;
            end
            // Vds<STARTGRIDVOLT
            if(ind_ds1<0)begin
                final_ind_ds1 = 0;
                final_ind_ds2 = 1;
                Vds = `STARTGRIDVOLT;
            end

                //*****Assign indexes and get capacitor values from LUT
                    *****
                //------------Cxx[x1]-----------Cxx[x2]-----------
                //----------------------Cxx----------------------
                //------------Cxx[x3]-----------Cxx[x4]-----------

                x1 = final_ind_ds1*`GRIDSPERLINE + final_ind_gs1;
                x2 = final_ind_ds1*`GRIDSPERLINE + final_ind_gs2;
                x3 = final_ind_ds2*`GRIDSPERLINE + final_ind_gs1;
                x4 = final_ind_ds2*`GRIDSPERLINE + final_ind_gs2;


                Cdbx1 = cdbData[x1];
                Cdbx2 = cdbData[x2];
                Cdbx3 = cdbData[x3];
                Cdbx4 = cdbData[x4];

                Cgbx1 = cgbData[x1];
                Cgbx2 = cgbData[x2];
                Cgbx3 = cgbData[x3];
                Cgbx4 = cgbData[x4];

                Cgdx1 = cgdData[x1];
```

```
                Cgdx2 = cgdData[x2];
                Cgdx3 = cgdData[x3];
Cgdx4 = cgdData[x4];

                Cgsx1 = cgsData[x1];
                Cgsx2 = cgsData[x2];
                Cgsx3 = cgsData[x3];
Cgsx4 = cgsData[x4];

                Csbx1 = csbData[x1];
                Csbx2 = csbData[x2];
                Csbx3 = csbData[x3];
Csbx4 = csbData[x4];

                //*****Assign Coefficients for MC simulation*****

                Idscoef_a1 = MCids3_a[x1];
Idscoef_a2 = MCids3_a[x2];
                Idscoef_a3 = MCids3_a[x3];
                Idscoef_a4 = MCids3_a[x4];

                Idscoef_b1 = MCids3_b[x1];
Idscoef_b2 = MCids3_b[x2];
                Idscoef_b3 = MCids3_b[x3];
                Idscoef_b4 = MCids3_b[x4];

                Idscoef_c1 = MCids3_c[x1];
Idscoef_c2 = MCids3_c[x2];
                Idscoef_c3 = MCids3_c[x3];
                Idscoef_c4 = MCids3_c[x4];


Idscoef_d1 = MCids3_d[x1];
                Idscoef_d2 = MCids3_d[x2];
                Idscoef_d3 = MCids3_d[x3];
                Idscoef_d4 = MCids3_d[x4];

Cgdcoef_a1 = MCCgd1_a[x1];
                Cgdcoef_a2 = MCCgd1_a[x2];
                Cgdcoef_a3 = MCCgd1_a[x3];
                Cgdcoef_a4 = MCCgd1_a[x4];

Cgdcoef_b1 = MCCgd1_b[x1];
                Cgdcoef_b2 = MCCgd1_b[x2];
                Cgdcoef_b3 = MCCgd1_b[x3];
                Cgdcoef_b4 = MCCgd1_b[x4];

Cgscoef_a1 = MCCgs1_a[x1];
                Cgscoef_a2 = MCCgs1_a[x2];
                Cgscoef_a3 = MCCgs1_a[x3];
                Cgscoef_a4 = MCCgs1_a[x4];

Cgscoef_b1 = MCCgs1_b[x1];
```

```
                    Cgscoef_b2 = MCCgs1_b[x2];
                    Cgscoef_b3 = MCCgs1_b[x3];
                    Cgscoef_b4 = MCCgs1_b[x4];

216 //Assign indexes and get Ids values; LUT is break into 12 pieces and
        stored in idsaa-idsal based on Vsb
                    if(ind_sb1<=0) begin
                        Ids1x1 = idsaa[x1];
                        Ids1x2 = idsaa[x2];
                        Ids1x3 = idsaa[x3];
221                     Ids1x4 = idsaa[x4];

                        Ids2x1 = idsab[x1];
                        Ids2x2 = idsab[x2];
                        Ids2x3 = idsab[x3];
226                     Ids2x4 = idsab[x4];
                    end
                    if(ind_sb1==1) begin
                        Ids1x1 = idsab[x1];
                        Ids1x2 = idsab[x2];
231                     Ids1x3 = idsab[x3];
                        Ids1x4 = idsab[x4];

                        Ids2x1 = idsac[x1];
                        Ids2x2 = idsac[x2];
236                     Ids2x3 = idsac[x3];
                        Ids2x4 = idsac[x4];
                    end
                    if(ind_sb1==2) begin
                        Ids1x1 = idsac[x1];
241                     Ids1x2 = idsac[x2];
                        Ids1x3 = idsac[x3];
                        Ids1x4 = idsac[x4];

                        Ids2x1 = idsad[x1];
246                     Ids2x2 = idsad[x2];
                        Ids2x3 = idsad[x3];
                        Ids2x4 = idsad[x4];
                    end
                    if(ind_sb1==3) begin
251                     Ids1x1 = idsad[x1];
                        Ids1x2 = idsad[x2];
                        Ids1x3 = idsad[x3];
                        Ids1x4 = idsad[x4];

256                     Ids2x1 = idsae[x1];
                        Ids2x2 = idsae[x2];
                        Ids2x3 = idsae[x3];
                        Ids2x4 = idsae[x4];
                    end
261                 if(ind_sb1==4) begin
                        Ids1x1 = idsae[x1];
                        Ids1x2 = idsae[x2];
```

```
                Ids1x3 = idsae[x3];
                Ids1x4 = idsae[x4];

                Ids2x1 = idsaf[x1];
                Ids2x2 = idsaf[x2];
                Ids2x3 = idsaf[x3];
                Ids2x4 = idsaf[x4];
            end
            if(ind_sb1==5) begin
                Ids1x1 = idsaf[x1];
                Ids1x2 = idsaf[x2];
                Ids1x3 = idsaf[x3];
                Ids1x4 = idsaf[x4];

                Ids2x1 = idsag[x1];
                Ids2x2 = idsag[x2];
                Ids2x3 = idsag[x3];
                Ids2x4 = idsag[x4];
            end
            if(ind_sb1==6) begin
                Ids1x1 = idsag[x1];
                Ids1x2 = idsag[x2];
                Ids1x3 = idsag[x3];
                Ids1x4 = idsag[x4];

                Ids2x1 = idsah[x1];
                Ids2x2 = idsah[x2];
                Ids2x3 = idsah[x3];
                Ids2x4 = idsah[x4];
            end
            if(ind_sb1==7) begin
                Ids1x1 = idsah[x1];
                Ids1x2 = idsah[x2];
                Ids1x3 = idsah[x3];
                Ids1x4 = idsah[x4];

                Ids2x1 = idsai[x1];
                Ids2x2 = idsai[x2];
                Ids2x3 = idsai[x3];
                Ids2x4 = idsai[x4];
            end
            if(ind_sb1==8) begin
                Ids1x1 = idsai[x1];
                Ids1x2 = idsai[x2];
                Ids1x3 = idsai[x3];
                Ids1x4 = idsai[x4];

                Ids2x1 = idsaj[x1];
                Ids2x2 = idsaj[x2];
                Ids2x3 = idsaj[x3];
                Ids2x4 = idsaj[x4];
            end
            if(ind_sb1==9) begin
```

```
                    Ids1x1 = idsaj[x1];
                    Ids1x2 = idsaj[x2];
                    Ids1x3 = idsaj[x3];
                    Ids1x4 = idsaj[x4];

                    Ids2x1 = idsak[x1];
                    Ids2x2 = idsak[x2];
                    Ids2x3 = idsak[x3];
                    Ids2x4 = idsak[x4];
                end
                if(ind_sb1>=10) begin
                    Ids1x1 = idsak[x1];
                    Ids1x2 = idsak[x2];
                    Ids1x3 = idsak[x3];
                    Ids1x4 = idsak[x4];

                    Ids2x1 = idsal[x1];
                    Ids2x2 = idsal[x2];
                    Ids2x3 = idsal[x3];
                    Ids2x4 = idsal[x4];
                end

                //normal case of Vsb: ind_sb belongs to [0,11], which
                    means Vsb belongs to [0,1.1]
                Vsb = V(br_sb);
                final_ind_sb1 = ind_sb1;
                final_ind_sb2 = ind_sb2;

            //*****extrapolation on Vsb; extrapolation is done by zero order.
                //Vsb<0v
                if(ind_sb1<0)begin
                    Vsb = `SBSTARTGRIDVOLT;
                    final_ind_sb1 = 0;
                    final_ind_sb2 = 1;
                end
                //Vsb>=1.1v
                if(ind_sb2>=(`ISBGRIDSPERLINE-1))begin
                    Vsb = `SBENDGRIDVOLT;
                    final_ind_sb1 = `ISBGRIDSPERLINE-2;
                    final_ind_sb2 = `ISBGRIDSPERLINE-1;
                end

            //MC simulation: Compute the sensitivity according to the
                coefficients
                //sens = a*dL^3 + b*dL^2 + c*dL + d; dL is the process
                    variation
                L_newIds=1/(`NOMLEN + dL);
                L_newIds2=L_newIds*L_newIds;
                L_newIds3=L_newIds2*L_newIds;
                L_newCgate=`NOMLEN + dL;

                //length sensitivity for Ids is based on Vsb=0
                Ids1x1_norm = idsaa[x1];
```

```
            Ids1x2_norm = idsaa[x2];
            Ids1x3_norm = idsaa[x3];
            Ids1x4_norm = idsaa[x4];

371         sensIds1=Idscoef_a1*L_newIds3 + Idscoef_b1*L_newIds2 +
               Idscoef_c1*L_newIds + Idscoef_d1 - Ids1x1_norm;
            sensIds2=Idscoef_a2*L_newIds3 + Idscoef_b2*L_newIds2 +
               Idscoef_c2*L_newIds + Idscoef_d2 - Ids1x2_norm;
            sensIds3=Idscoef_a3*L_newIds3 + Idscoef_b3*L_newIds2 +
               Idscoef_c3*L_newIds + Idscoef_d3 - Ids1x3_norm;
            sensIds4=Idscoef_a4*L_newIds3 + Idscoef_b4*L_newIds2 +
               Idscoef_c4*L_newIds + Idscoef_d4 - Ids1x4_norm;

376
            MCIds1x1=Ids1x1+sensIds1;
            MCIds1x2=Ids1x2+sensIds2;
            MCIds1x3=Ids1x3+sensIds3;
            MCIds1x4=Ids1x4+sensIds4;
381
            MCIds2x1=Ids2x1+sensIds1;
            MCIds2x2=Ids2x2+sensIds2;
            MCIds2x3=Ids2x3+sensIds3;
            MCIds2x4=Ids2x4+sensIds4;
386
            MCCgdx1=Cgdcoef_a1*L_newCgate + Cgdcoef_b1;
            MCCgdx2=Cgdcoef_a2*L_newCgate + Cgdcoef_b2;
            MCCgdx3=Cgdcoef_a3*L_newCgate + Cgdcoef_b3;
            MCCgdx4=Cgdcoef_a4*L_newCgate + Cgdcoef_b4;
391
            MCCgsx1=Cgscoef_a1*L_newCgate + Cgscoef_b1;
            MCCgsx2=Cgscoef_a2*L_newCgate + Cgscoef_b2;
            MCCgsx3=Cgscoef_a3*L_newCgate + Cgscoef_b3;
            MCCgsx4=Cgscoef_a4*L_newCgate + Cgscoef_b4;
396
            /****Bilinear interpolate 5 capacitors and Ids, according to
               Vgs and Vds***/
                slpg=(Vgs-(`STARTGRIDVOLT+final_ind_gs1*`LUTSTEP))/
                   `LUTSTEP;
                slpd=(Vds-(`STARTGRIDVOLT+final_ind_ds1*`LUTSTEP))/
                   `LUTSTEP;

401         //Cdb keeps same in MC
                Cdb1 = Cdbx1+slpg*(Cdbx2-Cdbx1);
                Cdb2 = Cdbx3+slpg*(Cdbx4-Cdbx3);
                Cdb = Cdb1+slpd*(Cdb2-Cdb1);

406         //Csb can be ignored in MC
                Csb1 = Csbx1+slpg*(Csbx2-Csbx1);
                Csb2 = Csbx3+slpg*(Csbx4-Csbx3);
                Csb = Csb1+slpd*(Csb2-Csb1);

411         //interpolation on Cgd & MCCgd
                Cgd1 = Cgdx1+slpg*(Cgdx2-Cgdx1);
```

```
                Cgd2 = Cgdx3+slpg*(Cgdx4-Cgdx3);
                Cgd = Cgd1+slpd*(Cgd2-Cgd1);
                MCCgd1 = MCCgdx1+slpg*(MCCgdx2-MCCgdx1);
                MCCgd2 = MCCgdx3+slpg*(MCCgdx4-MCCgdx3);
                MCCgd = MCCgd1+slpd*(MCCgd2-MCCgd1);

            //interpolation on Cgs & MCCgs
                Cgs1 = Cgsx1+slpg*(Cgsx2-Cgsx1);
                Cgs2 = Cgsx3+slpg*(Cgsx4-Cgsx3);
                Cgs = Cgs1+slpd*(Cgs2-Cgs1);
                MCCgs1 = MCCgsx1+slpg*(MCCgsx2-MCCgsx1);
                MCCgs2 = MCCgsx3+slpg*(MCCgsx4-MCCgsx3);
                MCCgs = MCCgs1+slpd*(MCCgs2-MCCgs1);

            //interpolation on Cgb & MCCgb
                Cgb1 = Cgbx1+slpg*(Cgbx2-Cgbx1);
                Cgb2 = Cgbx3+slpg*(Cgbx4-Cgbx3);
                Cgb = Cgb1+slpd*(Cgb2-Cgb1);

            //interpolation on Ids
                Ids11 = Ids1x1+slpg*(Ids1x2-Ids1x1);
                Ids12 = Ids1x3+slpg*(Ids1x4-Ids1x3);
                Ids1 = Ids11+slpd*(Ids12-Ids11);
                MCIds11 = MCIds1x1+slpg*(MCIds1x2-MCIds1x1);
                MCIds12 = MCIds1x3+slpg*(MCIds1x4-MCIds1x3);
                MCIds1 = MCIds11+slpd*(MCIds12-MCIds11);

                Ids21 = Ids2x1+slpg*(Ids2x2-Ids2x1);
                Ids22 = Ids2x3+slpg*(Ids2x4-Ids2x3);
                Ids2 = Ids21+slpd*(Ids22-Ids21);
                MCIds21 = MCIds2x1+slpg*(MCIds2x2-MCIds2x1);
                MCIds22 = MCIds2x3+slpg*(MCIds2x4-MCIds2x3);
                MCIds2 = MCIds21+slpd*(MCIds22-MCIds21);

            Ids = Ids1+(Vsb - final_ind_sb1*`ISBLUTSTEP)*((Ids2-Ids1)/
                `ISBLUTSTEP);
            MCIds = MCIds1+(Vsb - final_ind_sb1*`ISBLUTSTEP)*((MCIds2-
                MCIds1)/`ISBLUTSTEP);

    //for debugging
    //          file =$fopen("nmos.log","w");
    //          $fstrobe(file,"dL is %e",dL);
    //          $fstrobe(file,"Cgs1 is %e", Cgsx1);
    //          $fstrobe(file,"MCCgs1 is %e", MCCgsx1);
    //          ...
    //          $fclose(file);


        end //begin block
        //branch sourses contributions
        begin
            I(br_gd) <+ ddt(V(br_gd))*MCCgd;
            I(br_gs) <+ ddt(V(br_gs))*MCCgs;
```

```
                I(br_ds) <+ MCIds; //+ Ides
                I(br_gb) <+ ddt(V(br_gb))*Cgb;
466             I(br_sb) <+ ddt(V(br_sb))*Csb;
                I(br_db) <+ ddt(V(br_db))*Cdb;

            end
        end //analog
471 endmodule
```

Appendix B

Bash scripts of load LUTs into code

```
#!/bin/sh

#pass the length & width from the configuration file
while read curline; do
    echo $curline > tmpfilen
    #read N_len N_wid
    N_len=`awk '{print $1}' tmpfilen`
    N_wid=`awk '{print $2}' tmpfilen`
    #the path for SSTM  ../Model/Nmos.vams
    cp ../Model/Nmos.vams .
    sed -i "s/mynmos/nmos${N_len}${N_wid}/g" Nmos.vams
    cp ../VTL_nmos_char/L"$N_len"n_Wn"$N_wid"n/idsData.rpt .
    LUTSIZE=960
    #split IdsData into 12 files Idsaa-Idsal
    split -l 32 idsData.rpt ids
    #insert parameters Idsaa-Idsal to new model
    for i in `ls | grep idsa`
    do
        j="`cat $i`"
        sed -i "/\/\/insert LUT from script/i\  parameter real `echo $i
            `[0:$LUTSIZE]={`echo $j`};" Nmos.vams
    done
    rm ids*
    #insert parameters Cdb, Cgb, Cgd, Cgs, Csb to new model
    for i in cdbData.rpt cgbData.rpt cgdData.rpt cgsData.rpt csbData.rpt
    do
        cp ../VTL_nmos_char/L"$N_len"n_Wn"$N_wid"n/$i .
        #Some of the capacitors have negative values! turn them into
            positive values
        sed 's/-//g' $i > ./$i\_2
        sed 's/e/e-/g' $i\_2 > $i
        j=`cat $i`
    #set parameters without extension
        wo_ext=`basename $i .rpt`;
        sed -i "/\/\/insert LUT from script/i\  parameter real `echo
            $wo_ext`[0:$LUTSIZE]={`echo $j`};" Nmos.vams
        rm $i\_2
        rm $i
    done
    #***** statistical extension *****
    #insert MC coef into new model
    for i in `ls ../VTL_nmos_char/L"$N_len"n_Wn"$N_wid"n | grep MC`
    do
```

```
        j=`cat ../VTL_nmos_char/L"$N_len"n_Wn"$N_wid"n/$i`

    #set parameters without extension
44      wo_ext=`basename $i .rpt`;
        sed -i "/\/\/insert LUT from script/i\  parameter real `echo
            $wo_ext`[0:$LUTSIZE]={`echo $j`};" Nmos.vams
    done

    #turn every ',}' into '}'
49  sed -i 's/,};/};/g' Nmos.vams
    mv Nmos.vams ../MCfiles/nmosMC${N_len}${N_wid}.vams
done < ../conf_n
rm tmpfilen
```

Appendix C

Matlab scripts for generating polynomial coefficients LUT

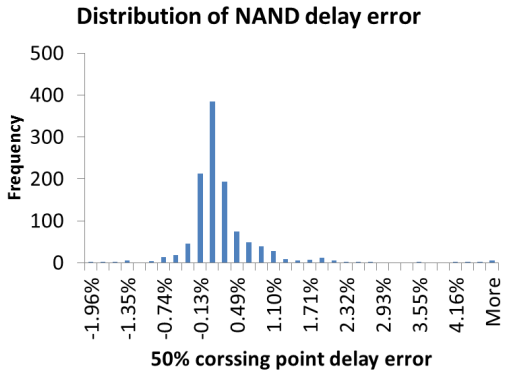
```
%This script is used for generate the coefficients for 3rd order
    polynomial curve.
%This file is used for Nmos width=90nm. The characterization LUTs are
    stored in path: L(length)n_Wn90n/*Data.rpt
mth=0;
ids45=csvread('L45n_Wn90n/idsData.rpt',31*mth,0,[31*mth 0 31*mth+30 30]);
ids47=csvread('L47n_Wn90n/idsData.rpt',31*mth,0,[31*mth 0 31*mth+30 30]);
ids48=csvread('L48n_Wn90n/idsData.rpt',31*mth,0,[31*mth 0 31*mth+30 30]);
ids49=csvread('L49n_Wn90n/idsData.rpt',31*mth,0,[31*mth 0 31*mth+30 30]);
ids50=csvread('L50n_Wn90n/idsData.rpt',31*mth,0,[31*mth 0 31*mth+30 30]);
ids51=csvread('L51n_Wn90n/idsData.rpt',31*mth,0,[31*mth 0 31*mth+30 30]);
ids52=csvread('L52n_Wn90n/idsData.rpt',31*mth,0,[31*mth 0 31*mth+30 30]);
ids53=csvread('L53n_Wn90n/idsData.rpt',31*mth,0,[31*mth 0 31*mth+30 30]);
ids55=csvread('L55n_Wn90n/idsData.rpt',31*mth,0,[31*mth 0 31*mth+30 30]);

Cgd45=csvread('L45n_Wn90n/cgdData.rpt',0,0, [0 0 30 30]);
Cgd47=csvread('L47n_Wn90n/cgdData.rpt',0,0, [0 0 30 30]);
Cgd48=csvread('L48n_Wn90n/cgdData.rpt',0,0, [0 0 30 30]);
Cgd49=csvread('L49n_Wn90n/cgdData.rpt',0,0, [0 0 30 30]);
Cgd50=csvread('L50n_Wn90n/cgdData.rpt',0,0, [0 0 30 30]);
Cgd51=csvread('L51n_Wn90n/cgdData.rpt',0,0, [0 0 30 30]);
Cgd52=csvread('L52n_Wn90n/cgdData.rpt',0,0, [0 0 30 30]);
Cgd53=csvread('L53n_Wn90n/cgdData.rpt',0,0, [0 0 30 30]);
Cgd55=csvread('L55n_Wn90n/cgdData.rpt',0,0, [0 0 30 30]);

%Capacitance should be positive!!!!
abs_Cgd45=abs(Cgd45);
abs_Cgd47=abs(Cgd47);
abs_Cgd48=abs(Cgd48);
abs_Cgd49=abs(Cgd49);
abs_Cgd50=abs(Cgd50);
abs_Cgd51=abs(Cgd51);
abs_Cgd52=abs(Cgd52);
abs_Cgd53=abs(Cgd53);
abs_Cgd55=abs(Cgd55);

Cgs45=csvread('L45n_Wn90n/cgsData.rpt',0,0, [0 0 30 30]);
Cgs47=csvread('L47n_Wn90n/cgsData.rpt',0,0, [0 0 30 30]);
Cgs48=csvread('L48n_Wn90n/cgsData.rpt',0,0, [0 0 30 30]);
Cgs49=csvread('L49n_Wn90n/cgsData.rpt',0,0, [0 0 30 30]);
Cgs50=csvread('L50n_Wn90n/cgsData.rpt',0,0, [0 0 30 30]);
Cgs51=csvread('L51n_Wn90n/cgsData.rpt',0,0, [0 0 30 30]);
Cgs52=csvread('L52n_Wn90n/cgsData.rpt',0,0, [0 0 30 30]);
```

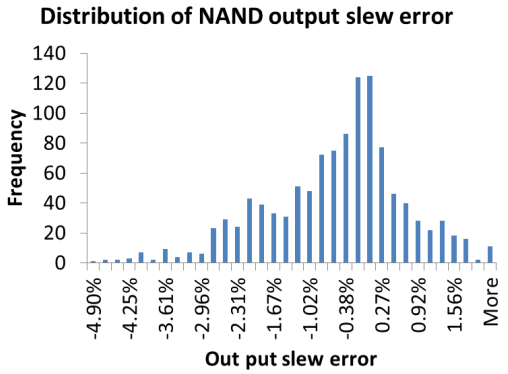
```
Cgs53=csvread('L53n_Wn90n/cgsData.rpt',0,0, [0 0 30 30]);
Cgs55=csvread('L55n_Wn90n/cgsData.rpt',0,0, [0 0 30 30]);

%Capacitance should be positive!!!!
abs_Cgs45=abs(Cgs45);
abs_Cgs47=abs(Cgs47);
abs_Cgs48=abs(Cgs48);
abs_Cgs49=abs(Cgs49);
abs_Cgs50=abs(Cgs50);
abs_Cgs51=abs(Cgs51);
abs_Cgs52=abs(Cgs52);
abs_Cgs53=abs(Cgs53);
abs_Cgs55=abs(Cgs55);

Cgb45=csvread('L45n_Wn90n/cgbData.rpt',0,0, [0 0 30 30]);
Cgb47=csvread('L47n_Wn90n/cgbData.rpt',0,0, [0 0 30 30]);
Cgb48=csvread('L48n_Wn90n/cgbData.rpt',0,0, [0 0 30 30]);
Cgb49=csvread('L49n_Wn90n/cgbData.rpt',0,0, [0 0 30 30]);
Cgb50=csvread('L50n_Wn90n/cgbData.rpt',0,0, [0 0 30 30]);
Cgb51=csvread('L51n_Wn90n/cgbData.rpt',0,0, [0 0 30 30]);
Cgb52=csvread('L52n_Wn90n/cgbData.rpt',0,0, [0 0 30 30]);
Cgb53=csvread('L53n_Wn90n/cgbData.rpt',0,0, [0 0 30 30]);
Cgb55=csvread('L55n_Wn90n/cgbData.rpt',0,0, [0 0 30 30]);

%Capacitance should be positive!!!!
abs_Cgb45=abs(Cgb45);
abs_Cgb47=abs(Cgb47);
abs_Cgb48=abs(Cgb48);
abs_Cgb49=abs(Cgb49);
abs_Cgb50=abs(Cgb50);
abs_Cgb51=abs(Cgb51);
abs_Cgb52=abs(Cgb52);
abs_Cgb53=abs(Cgb53);
abs_Cgb55=abs(Cgb55);

L=[45e-9,47e-9,48e-9,49e-9,51e-9,52e-9,53e-9,55e-9];
L_new=1./L;

%cat(DIM, A1, A2, ...) i.e. 'sens' has three dimensions
sens=cat(3, ids45, ids47, ids48, ids49, ids51, ids52, ids53, ids55);

Cgssens=cat(3, abs_Cgs45, abs_Cgs47, abs_Cgs48, abs_Cgs49, abs_Cgs51,
    abs_Cgs52, abs_Cgs53, abs_Cgs55);

Cgdsens=cat(3, abs_Cgd45, abs_Cgd47, abs_Cgd48, abs_Cgd49, abs_Cgd51,
    abs_Cgd52, abs_Cgd53, abs_Cgd55);

% sens = a*L_new^3 + b*L_new^2 + c*L_new + d
for i=1:31
    for j=1:31
    imsens=[sens(i,j,1),sens(i,j,2),sens(i,j,3),sens(i,j,4),sens(i,j,5),
        sens(i,j,6), sens(i,j,7),sens(i,j,8)];
```

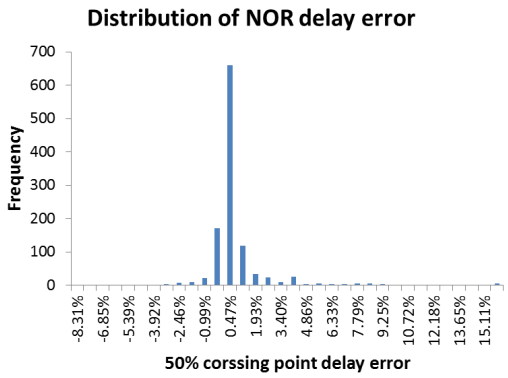
```
        coef=polyfit(L_new,imsens,3);
        %used for CSVwrite
        Idscoef(i,j,1)=coef(1);
        Idscoef(i,j,2)=coef(2);
96      Idscoef(i,j,3)=coef(3);
        Idscoef(i,j,4)=coef(4);

        Cgdimsens=[Cgdsens(i,j,1),Cgdsens(i,j,2),Cgdsens(i,j,3),Cgdsens(i,j
            ,4), Cgdsens(i,j,5),Cgdsens(i,j,6),Cgdsens(i,j,7),Cgdsens(i,j,8)];
        Cgsimsens=[Cgssens(i,j,1),Cgssens(i,j,2),Cgssens(i,j,3),Cgssens(i,j
            ,4), Cgssens(i,j,5),Cgssens(i,j,6),Cgssens(i,j,7),Cgssens(i,j,8)];
101
        Cgdcoef=polyfit(L,Cgdimsens,1);
        Cgdcoef2(i,j,1)=Cgdcoef(1);
        Cgdcoef2(i,j,2)=Cgdcoef(2);

106     Cgscoef=polyfit(L,Cgsimsens,1);
        Cgscoef2(i,j,1)=Cgscoef(1);
        Cgscoef2(i,j,2)=Cgscoef(2);

        %the estimated current or capacitances
111     Idsnew=polyval(coef,L_new);
        Cgdnew=polyval(Cgdcoef,L);
        Cgsnew=polyval(Cgscoef,L);
        end
    end
116
    csvwrite('MCids3_a.rpt',Idscoef(:,:,1));
    csvwrite('MCids3_b.rpt',Idscoef(:,:,2));
    csvwrite('MCids3_c.rpt',Idscoef(:,:,3));
    csvwrite('MCids3_d.rpt',Idscoef(:,:,4));
121


    csvwrite('MCCgd1_a.rpt',Cgdcoef2(:,:,1));
    csvwrite('MCCgd1_b.rpt',Cgdcoef2(:,:,2));

126 csvwrite('MCCgs1_a.rpt',Cgscoef2(:,:,1));
    csvwrite('MCCgs1_b.rpt',Cgscoef2(:,:,2));
```

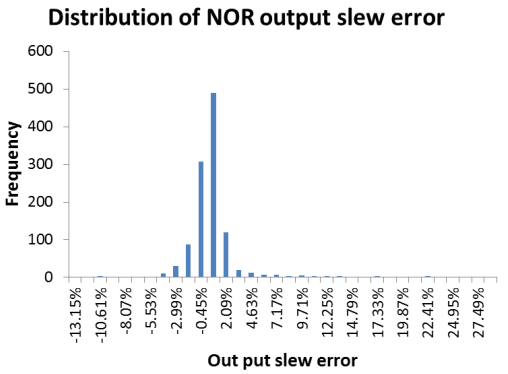
Appendix D

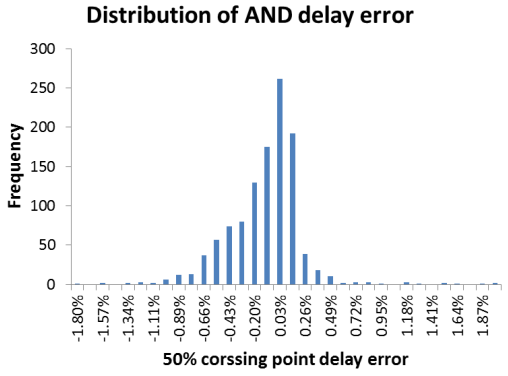
Distributions of relative delay and output slew errors for standard gate compared with BSIM4 considering all combinations of input slew and output capacitances.

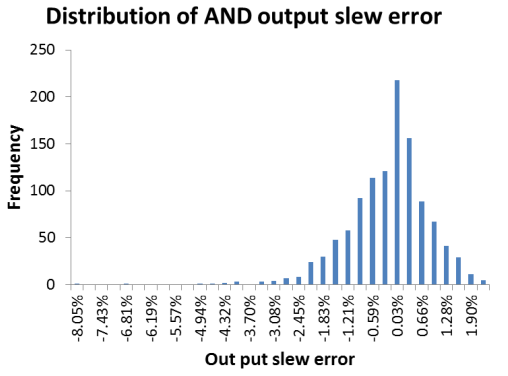


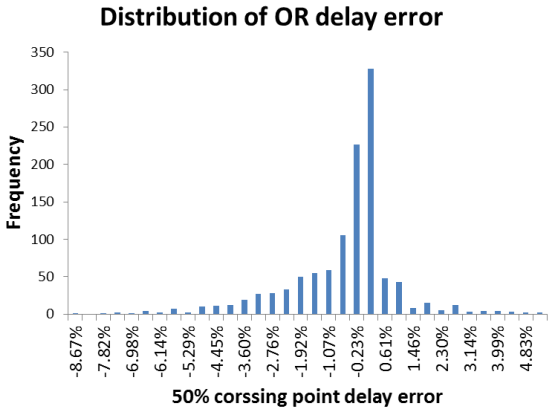


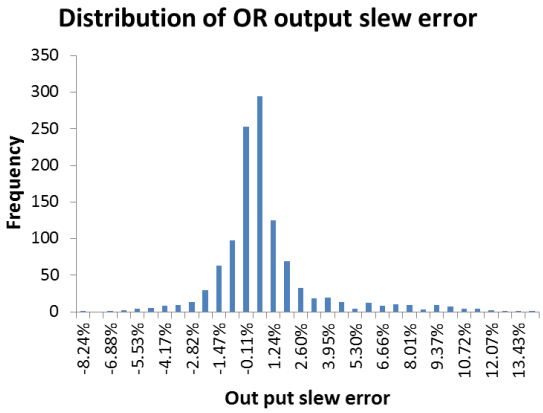


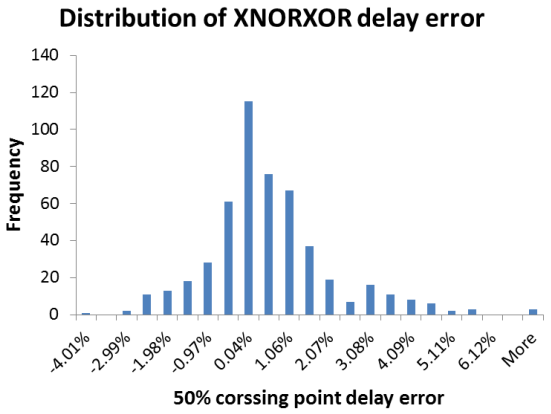


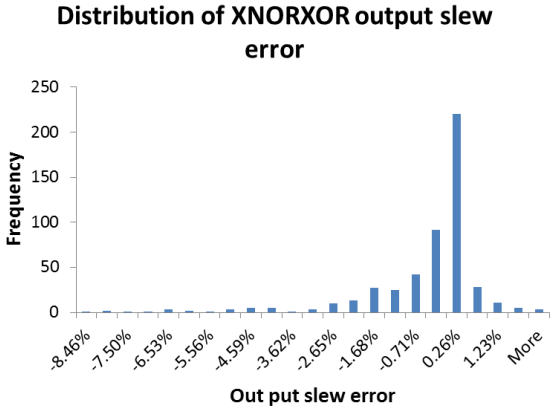


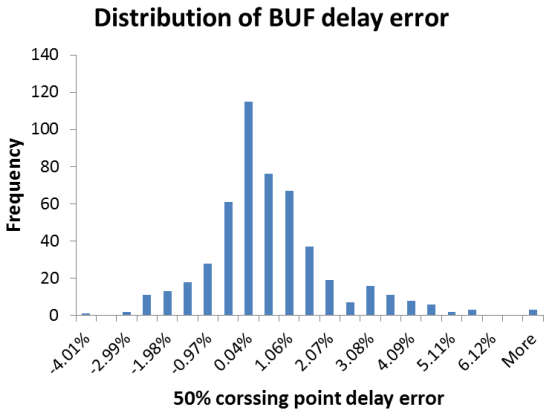


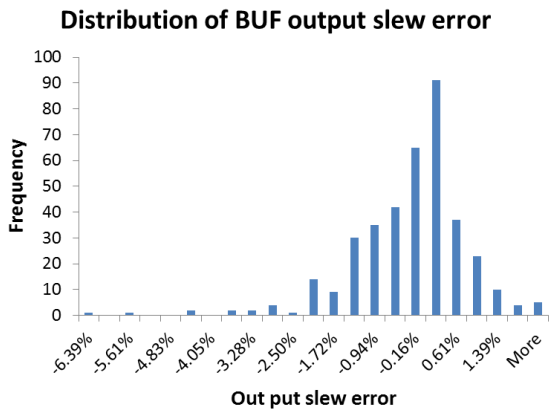

Distribution of OR delay error
Frequency
350
300
250
200
150
100
50
0
-8.67%
-7.82%
-6.98%
-6.14%
-5.29%
-4.45%
-3.60%
-2.76%
-1.92%
-1.07%
-0.23%
0.61%
1.46%
2.30%
3.14%
3.99%
4.83%
50% corssing point delay error

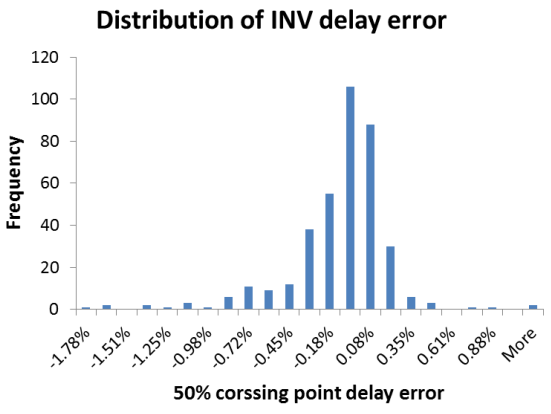

Distribution of OR output slew error
Frequency
350
300
250
200
150
100
50
0
-8.24%
-6.88%
-5.53%
-4.17%
-2.82%
-1.47%
-0.11%
1.24%
2.60%
3.95%
5.30%
6.66%
8.01%
9.37%
10.72%
12.07%
13.43%
Out put slew error

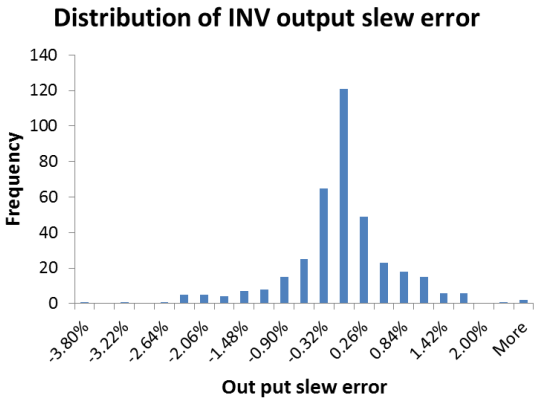

Distribution of XNORXOR delay error
Frequency
140
120
100
80
60
40
20
0
-4.01%
-2.99%
-1.98%
-0.97%
0.04%
1.06%
2.07%
3.08%
4.09%
5.11%
6.12%
More
50% corssing point delay error


Distribution of XNORXOR output slew error
Frequency
250
200
150
100
50
0
-8.46%
-7.50%
-6.53%
-5.56%
-4.59%
-3.62%
-2.65%
-1.68%
-0.71%
0.26%
1.23%
More
Out put slew error


Distribution of BUF delay error
Frequency
140
120
100
80
60
40
20
0
-4.01%
-2.99%
-1.98%
-0.97%
0.04%
1.06%
2.07%
3.08%
4.09%
5.11%
6.12%
More
50% corssing point delay error


Distribution of BUF output slew error
Frequency
100
90
80
70
60
50
40
30
20
10
0
-6.39%
-5.61%
-4.83%
-4.05%
-3.28%
-2.50%
-1.72%
-0.94%
-0.16%
0.61%
1.39%
More
Out put slew error


Distribution of INV delay error
Frequency
120
100
80
60
40
20
0
-1.78%
-1.51%
-1.25%
-0.98%
-0.72%
-0.45%
-0.18%
0.08%
0.35%
0.61%
0.88%
More
50% corssing point delay error


Distribution of INV output slew error
Frequency
140
120
100
80
60
40
20
0
-3.80%
-3.22%
-2.64%
-2.06%
-1.48%
-0.90%
-0.32%
0.26%
0.84%
1.42%
2.00%
More
Out put slew error