USING INTERVAL ARITHMETIC FOR ELECTRONIC CIRCUITS SIMULATION

By

Amin Maher Abdallah Baraka

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2015

USING INTERVAL ARITHMETIC FOR ELECTRONIC CIRCUITS SIMULATION

By

Amin Maher Abdallah Baraka

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Under the Supervision of

Dr. Hossam A. H. Fahmy
Associate Professor
Electronics and Communications Engineering Department
Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2015

USING INTERVAL ARITHMETIC FOR ELECTRONIC CIRCUITS SIMULATION

By

Amin Maher Abdallah Baraka

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Approved by the
Examining Committee

_____

Associate Prof. Hossam Aly Hassan Fahmy, Thesis Advisor

_____

Prof. Dr. Mohamad Mahmoud Riad El-Ghonemy, Internal Examiner

_____

Prof. Dr. Mohamed Amin Ibrahim Dessouky, External Examiner
Professor at Faculty of Engineering, Ain-Shams University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2015

Engineer: Amin Maher Abdallah Baraka
Date of Birth: 16/9/1982
Nationality: Egyptian
E-mail: amin.maher@gmail.com
Phone: +20 1011550660
Address: 8 Mohamed AlMahdy St., Nasr City, Cairo

Registration Date: 1/10/2010
Awarding Date: / /
Degree: Master of Science
Department: Electronics and Communications Engineering

Supervisors: Associate Prof. Hossam Aly Hassan Fahmy

Examiners: Prof. Dr. Mohamad Mahmoud Riad El-Ghonemy
Prof. Dr. Mohamed Amin Ibrahim Dessouky
(Professor at Faculty of Engineering, Ain-Shams University)
Associate Prof. Hossam Aly Hassan Fahmy

Title of Thesis: **Using Interval Arithmetic for Electronic Circuits Simulation**

Key Words: interval arithmetic, compact device models, circuit simulation, Monte-Carlo, design variability

Summary:

Semiconductor devices scaling provides higher circuit density and faster devices, this provides better performance and more functionality for electronic chips. Variations affect circuit behavior as more as device scaled down. Verification of circuit behavior under the uncertainty arises from different variations is a challenge. Monte-Carlo statistical analysis and corner case analysis are used to estimate the circuit behavior regards the variations. Interval arithmetic presents a potential alternative to evaluate circuit designs under variations uncertainties.

In this work, we present simulation flow that utilize using of existing designs by replacing statistical parameters variations by interval parameters, so it may replace or enhance the current conventional Monte-Carlo simulation flow. An interval-value based circuit simulation engine is implemented, and library for interval models for sources, linear elements and non-linear elements.

Models library is tested for accuracy against Monte-Carlo simulations. Simulator is tested using linear elements circuits, showing acceptable results for small circuits.

Acknowledgments

First and last I thank god (Allah) for helping me to complete this thesis, and I pray to him to make it a useful work.

I give my appreciation to my mother and father for their encouragement and prayers.

I give a special thanks to my wife who stood to my side and suffered a lot with me through my master study. And a little thanks for my kids

I would like to thank my supervisor Dr. Hossam for his support, guidance and understanding that makes my able to finish this work.

I would like to thank my manger Mohamed Selim for endless help and support and encouragement, and my friends Ahmed Hareedy and Islam Shaboon for their encouragement and advice.

Gratings for my friend Sherif Mansour, the companion through this journey.

My thanks to Mentor Graphics for support and resources.

Dedication

To
My Mother
My Father
and
My Wife

Contents

List of Tables x

List of Figures xi

Symbols and Abbreviations xiii

Abstract xv

1 Introduction 1

1.1 Process Variations 1

1.2 Account For Process Variations 2

1.2.1 Corner Simulations 2

1.2.2 Monte-Carlo Simulation 2

1.3 Research Objective and Contribution 3

1.4 Thesis Organization 3

2 Interval Arithmetic Theories 5

2.1 Classical Interval Arithmetic 5

2.1.1 Historical Notes 5

2.1.2 Notation 5

2.1.3 Definitions and Basic Concepts 6

2.1.4 Order Relations on Intervals 7

2.1.5 Basic Operations of Interval Arithmetic 7

2.1.6 Algebraic Properties of Interval Arithmetic 8
2.1.7 Interval Functions 9
2.1.8 Interval Arithmetic Limitations 9
2.1.9 Interval Arithmetic Tools 10
2.2 Modal Interval Arithmetic 10
2.2.1 Basics 11
2.2.2 Basic Operations of Modal Arithmetic 11
2.2.3 Modal Arithmetic Limitations 12
2.2.4 Modal Arithmetic Tools 12
2.3 Affine Arithmetic 12
2.3.1 Basic Concepts 13
2.3.2 Affine Arithmetic Operations 13
2.3.3 Affine Arithmetic Limitations 16
2.3.4 Affine Arithmetic Tools 16

3 Interval Circuits Simulator Design 17

3.1 Introduction 17
3.2 Simulation flow 17
3.3 Simulator Parts 18
3.3.1 Simulator Front-End 18
3.3.2 Simulator Back-End 21
3.3.3 Simulator Kernel 21
3.4 Simulator Testing 23
3.5 Available Analysis 24

4 Models Library 25

4.1 Voltage and Current Sources 25
4.2 Linear Elements 25

4.3 Non-Linear Elements . . . 28
4.3.1 Techniques to Get Interval Models . . . 28
4.3.2 Simple Diode Model . . . 31
4.3.3 Simple MOSFET Model . . . 34
4.3.4 Advanced MOSFET Models . . . 36

5 Results **37**
5.1 Unit Testing . . . 37
5.1.1 Capacitor Model . . . 37
5.1.2 Diode Model . . . 39
5.1.3 Simple MOSFET Model . . . 40
5.1.4 Advanced MOSFET Models . . . 41
5.2 Passive circuits testing . . . 44
5.2.1 Potential Divider . . . 44
5.2.2 R-2R Resistors Ladder . . . 45
5.2.3 Transmission Line R-C Model . . . 47

6 Conclusion **51**
6.1 Conclusion . . . 51
6.2 Future work . . . 51

Appendix A Interval Algorithms **53**
A.1 Interval Gauss–Seidel Method . . . 53
A.2 Interval Newton Method . . . 54
A.2.1 Multivariate Interval Newton Method . . . 54

Appendix B MVS MOSFET Model Interval Code **57**

References **67**

List of Tables

2.1 Affine Expressions and their interval counterparts 15

3.1 Design with statistical parameters and its interval equivalence 20

3.2 Models library 23

3.3 Simulator tests 23

4.1 Changing $if - else$ conditions to states 29

4.2 Diode model parameters 31

4.3 Verilog-a code for the diode model 31

4.4 C++ code for the diode model interval representation 32

4.5 C++ code for simple MOSFET model interval representation 34

4.6 C++ code for simple MOSFET model affine representation 35

5.1 Diode model parameters values 39

5.2 Diode I_d current model results 39

5.3 MOSFET model parameters values 40

5.4 MVS Test 1 model parameters values 41

5.5 MVS Test 2 model parameters values 42

5.6 MVS Test 3 model parameters values 43

5.7 8-Bits digital to analog converter (R-2R resistors network) simulation results 46

5.8 Transmission line simulation time 47

B.1 C++ code for MVS MOSFET model 57

B.2 C++ definitions for MVS MOSFET model 59

List of Figures

3.1 Monte-Carlo (MC) and interval based circuit simulation flow 19

3.2 Traditional transient simulation flow 22

4.1 Sources functions examples 26

4.2 Voltage source representation with MNA 26

4.3 Resistor representation with MNA 26

4.4 Capacitor model transient equivalent circuit 27

4.5 Another representation for capacitor model equivalent circuit 27

4.6 Inductor transient equivalent circuit 28

4.7 Interval comparing 30

5.1 RC section schematic 37

5.8 Potential divider circuit diagram and equivalent system of equations . . . 44

5.9 Resistor potential divider simulations results 45

5.11 Transmission line R-C model example 47

5.12 Transmission line one R-C section simulations results 48

5.13 Transmission line two R-C section simulations results 48

5.14 Transmission line three R-C section simulations results 49

Symbols and Abbreviations

$\mathbb{R}$	The set of real numbers
$\mathbb{IR}$	The set of real interval numbers
X, Y, Z	Interval variable symbols
$\underline{X}$	The infimum (minimum) of an interval number X
$\overline{X}$	The supremum (maximum) of an interval number X
$m(X)$	The mid-point of an interval number X
$r(X)$	The radius of an interval number X
$w(X)$	The width of an interval number X
$\underline{\cup}$	The interval hull operator
$\|X\|$	Greatest absolute value of an interval number X
$\|x\|$	Absolute value of a real number x
$\hat{x}, \hat{y}, \hat{z}$	Affine quantities
ϵ	Noise symbole in affine quantity
A	Ampere, electrical current unit
V	Volt, potential difference unit
Ω	Ohm, electrical resistance unit
F	Farad, electrical capacitance unit
s	Second, base unit of time in the international system of units
AA	Affine Arithmetic
BSIM4	Berkeley Short-channel IGFET Model
C-XSC	A C++ Class Library for Extended Scientific Computing
IA	Interval Arithmetic
MA	Modal Interval Arithmetic
MC	Monte-Carlo
MNA	Modified Nodal Analysis
MNA	Modified Nodal Analysis
MOSFET	Metal-Oxide Semiconductor Field-Effect Transistor
MVS	MIT Virtual Source MOSFET model
PWL	Piece-Wise Linear
SPICE	Simulation Program with Integrated Circuit Emphasis

Abstract

Electronics circuits designers and manufactures work to get better performance and more functionality for their chips. One way to achieve that is to scale the semiconductor devices, which provides higher circuit density and faster devices. Variations affect circuit behavior as more as device scaled down. Verification of circuit behavior under the uncertainty arises from different variations is a challenge. Monte-Carlo statistical analysis and corner case analysis are used to estimate the circuit behavior regards the variations. Interval arithmetic presents a potential alternative to evaluate circuit designs under variations uncertainties.

In this work, we present simulation flow that utilize using of existing designs by replacing statistical parameters variations by interval parameters, so it may replace or enhance the current conventional Monte-Carlo simulation flow. An interval-value based circuit simulation engine is implemented, and library for interval models for sources, linear elements and non-linear elements.

Models library is tested for accuracy against Monte-Carlo simulations. Simulator is tested using linear elements circuits, showing acceptable results for small circuits.

Chapter 1

Introduction

In this introductory chapter we present the problem of process variation, which represents the motive for this work. In section 1.2 current methods used to handle this problem are presented. In the rest of the chapter we present our research scope and contributions, then thesis organization is presented in section 1.4.

1.1 Process Variations

Electronics circuits designers and manufactures work to get better performance and more functionality for their chips. One way to achieve that is to scale the semiconductor devices, which provides higher circuit density and faster devices. Variations affect circuit behavior as devices are scaled down.

For circuit design; *Variation* is the deviation from intended or designed values for a structure or circuit parameter of concern [1]. Sources of variations can be environmental or Physical. Environmental variations includes factors arising during the operation of the circuit, as variations in power supply, switching activity, and temperature of the chip or across the chip. Process variations are the physical factors during fabrication process. Many processing steps can cause non-uniformity in the manufactured device properties.

Process variation results in permanent change in devices and interconnects attributes. Process variations become particularly important as the devices are scaled down, as they become a larger percentage of the device feature. Process variation results in random deviations from designed characteristics which can be modeled in form of probability density functions of the statistical parameters.

1.1.0.1 Process Variations Types

Process random variations can be classified into global and local variations. Global variations, inter-die, may be between lot of wafers (lot-to-lot), between two wafers (wafer-to-wafer) or between two dies on the same wafer (die-to-die). An example for global

variations is the loading effects in etching or deposition that impact the geometry of all the devices on a wafer. One example of inter-die variations is the loading effects in etching or deposition that impact the geometry of all the devices on a wafer [2]. Local variations, intra-die, are variations affect each device individually; these variations started to appear when transistors channels became less than 90 nanometers long. At these sizes transistors electrical properties may be affected by the roughness of a transistor's edges or the granularity in the crystal of the metal electrode that turns a transistor on or off [3].

1.2 Account For Process Variations

Verification of circuit behavior under the uncertainty arising from different variations is a challenge. Statistical analysis and corner case analysis are used to estimate the circuit behavior regarding the variations. MC, corner simulations are commonly used techniques to account for variations. Accuracy for MC depends on number of simulations runs [4], while for corner case, the complexity of the analysis increase as the number of parameters increase [5].

1.2.1 Corner Simulations

In order to account for process variability in circuit performance, typically, corner models are used to set the lower and upper limits of process variation [6]. Corner analysis is a traditional approach that ensures good yield, at the expense of a pessimistic design [7]. The corner case analysis works on the parameters bounds to provide the limits of the circuit behavior. In conventional circuit design technique, process variability is modeled by four worst-case corners: two for analog applications and two for digital. A standard set of model parameters (e.g. Vth) is used to account for process variability and model the worst-case corner performance of devices and circuits of the target CMOS technology [6].

The major problem with the corner models are that to increase the efficiency of this analysis, the model need to keep the correlations between the device parameters and the models include pessimistic corner values. The complexity of this analysis would increase to account for the increase in the number of parameters by adding more corners [5, 6]. The corner models offer the designers the capability to simulate the pass/fail results of a typical design and are usually pessimistic [6].

1.2.2 Monte-Carlo Simulation

MC simulation is a statistical analysis which randomly samples different parameters according to their statistical distribution [4]. MC is a direct method for evaluating statistical circuit performance, where the sampling is done then the circuit is simulated. Statistics on simulation results after several runs, sampling and simulation, can be done to find the performance distributions.

MC accuracy doesn't depend on the number of parameters cause the variation, but on the number of samples taken [4]. The cost of the MC is proportional to the number of samples, for about 10 times reduction in estimation error it requires a 100 times increase in the number of samples when estimating a Gaussian distribution [2]. For each sample the runtime required for simulation, is on the order of $\mathcal{O}(n^3)$, where n is the size of the circuit [2]. So MC may need large time to obtain accurate results when the number of samples required is large and the circuit size is large too.

1.3 Research Objective and Contribution

As seen; MC simulations may be time consuming. Approaches which use range arithmetic in circuit simulation show good results [2, 8] . Through our work we study replacing the MC simulations with an interval based simulation flow, which may integrate with or replace traditional MC simulation flow. Design aspects for an interval simulator are discussed. We evaluate the usage of different range arithmetic models, Interval arithmetic, modal interval arithmetic and affine arithmetic, to evaluate semiconductor device models and to do circuit simulation. The results for linear circuits and non-linear models are obtained.

A talk about this work has been presented on the 16th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics - SCAN2014 [9]. A paper then has been submitted for the post-conference proceedings to be published on Springer, Lecture Notes of Computer Science [10].

1.4 Thesis Organization

The thesis is organized after this introductory chapter as follows; chapter 2 provides a brief background about the interval arithmetic. In chapter 3 we present the design aspects for the interval simulator, and the simulator models library. Then in chapters 5 and 6; results for interval simulations are presented, and we provide the conclusion of our work.

Chapter 2

Interval Arithmetic Theories

In this chapter we introduce basic concepts for interval arithmetic computations. This is covering the classical interval arithmetic, modal arithmetic and affine arithmetic.

2.1 Classical Interval Arithmetic

2.1.1 Historical Notes

Interval Arithmetic (IA) defines a set of operations on intervals. History of classical IA backs to twentieth century fifties and sixties. The motive behind this was to limit rounding errors in numeric computations. Today; there are many algorithms that benefit from IA, as well as many tools and software packages that utilize IA. More about classical IA can be found in [11, 12], which are the main references for this section. In our work when we refer to IA, we mean the classical IA.

2.1.2 Notation

Endpoint notation. Closed interval denoted by [a, b] is the set of real numbers given by (2.1). Other types of intervals as open and half-open may appear through operations, our work concentrates on closed intervals.

$$[a, b] = \{x \in \mathbb{R} : a \leq x \leq b\} \tag{2.1}$$

Through this work; capital letters are denoting intervals and their bounds. Upper and lower bounds of an interval X will be denoted by $\underline{X}$ and $\overline{X}$, respectively. So interval X can be represented as (2.2). The set of real interval numbers is donated by $\mathbb{IR}$.

$$X = [\underline{X}, \overline{X}] \tag{2.2}$$

2.1.3 Definitions and Basic Concepts

Definition 2.1. Equal intervals. Two intervals X and Y are said to be equal if they are the same sets. This holds when the corresponding intervals' endpoints are equal.

$$X = Y \Leftrightarrow \underline{X} = \underline{Y} \; and \; \overline{X} = \overline{Y} \tag{2.3}$$

Definition 2.2. Degenerate Interval. The interval X is degenerate if $\underline{X} = \overline{X}$.

A degenerate interval contains a single real number x, so degenerate interval $[x, x]$ is represented by the real number x. As example we write $0 = [0, 0]$.

Definition 2.3. Intervals intersection. Intersection of two intervals X and Y is defined by (2.4).

$$X \cap Y = \{z : z \in X \text{ and } z \in Y\} \tag{2.4}$$

The intersection of two intervals X and Y is empty, that is X and Y have no points in common, if either $\overline{Y} < \underline{X}$ or $\overline{X} < \underline{Y}$. Then $X \cap Y = \phi$. If intersection is not empty (2.4) can be written as in (2.5).

$$X \cap Y = [max\{\underline{X}, \underline{Y}\}, min\{\overline{X}, \overline{Y}\}] \tag{2.5}$$

Definition 2.4. Intervals union. Union of two intervals X and Y is defined by (2.6).

$$X \cup Y = \{z : z \in X \text{ or } z \in Y\} \tag{2.6}$$

In general union of intervals is not represented by one interval, unless $X \cap Y \neq \phi$, then union can be represented by (2.7).

$$X \cup Y = [min\{\underline{X}, \underline{Y}\}, max\{\overline{X}, \overline{Y}\}] \tag{2.7}$$

Definition 2.5. Interval hull. The interval hull of two intervals X and Y is defined by (2.8).

$$X \underline{\cup} Y = [min\{\underline{X}, \underline{Y}\}, max\{\overline{X}, \overline{Y}\}] \tag{2.8}$$

By this definition interval hull is always an interval. For any two intervals X and Y, we have:

$$X \cup Y \subset X \underline{\cup} Y \tag{2.9}$$

Definition 2.6. Interval mid-point. For interval X; interval mid-point $m(X)$ is defined as:

$$m(x) = \frac{\underline{X} + \overline{X}}{2} \tag{2.10}$$

Definition 2.7. Interval radius. For interval X; interval radius $r(X)$ is defined as:

$$r(x) = \frac{\overline{X} - \underline{X}}{2} \tag{2.11}$$

2.1.4 Order Relations on Intervals

For real numbers the order relation is defined by relation $<$, for intervals X and $Y \in \mathbb{IR}$, we may define the order relation as:

$$X < Y \quad \Leftrightarrow \quad \overline{X} < \underline{Y} \tag{2.12}$$

Interval X then can be considered positive if $0 < X$, and negative if $X < 0$. Relation (2.13) is valid for X, Y and $Z \in \mathbb{IR}$, this relation is called *transitive* relation.

$$X < Y \text{ and } Y < Z \quad \Leftrightarrow \quad X < Z \tag{2.13}$$

Another ordered relation is defined as:

$$X \subseteq Y \quad \Leftrightarrow \quad \underline{Y} \leq \underline{X} \text{ and } \overline{X} \leq \overline{Y} \tag{2.14}$$

These relations are *partial* ordering relations, as not every pair of intervals is comparable under set inclusion. That is, if intervals X and Y are overlapped then, X is not contained in Y , nor is Y contained in X.

2.1.5 Basic Operations of Interval Arithmetic

Basic arithmetic operations on intervals can be represented by the end point notations as in (2.15) to (2.18).

$$X + Y = [\underline{X} + \underline{Y}, \overline{X} + \overline{Y}] \tag{2.15}$$

$$X - Y = [\underline{X} - \overline{Y}, \overline{X} - \underline{Y}] \tag{2.16}$$

$$X.Y = [minS, maxS], \tag{2.17}$$

where $S = \{\underline{X}.\underline{Y}, \underline{X}.\overline{Y}, \overline{X}.\underline{Y}, \overline{X}.\overline{Y}\}$.

$$X/Y = X.(1/Y), \tag{2.18}$$

where $1/Y = [1/\overline{Y}, 1/\underline{Y}]$ and $0 \notin Y$.

2.1.6 Algebraic Properties of Interval Arithmetic

2.1.6.1 Commutativity and Associativity

For any three intervals X, Y and $Z \in \mathbb{IR}$, we have:

$$X + Y = Y + X \tag{2.19}$$

$$X + (Y + Z) = (X + Y) + Z \tag{2.20}$$

$$X.Y = Y.X \tag{2.21}$$

$$X(Y.Z) = (X.Y)Z \tag{2.22}$$

2.1.6.2 Additive and Multiplicative Identity Elements

For any interval X, we have:

$$X + 0 = X \tag{2.23}$$

$$1.X = X \tag{2.24}$$

$$0.X = 0 \tag{2.25}$$

2.1.6.3 Additive and Multiplicative Inverse

Interval arithmetic has neither additive nor multiplicative inverses. That is in general,

$$X + (-X) \neq 0,$$

$$\frac{X}{X} \neq 1.$$

2.1.6.4 Distributive Law

The distributive law for real numbers (2.26), is not valid for intervals sets in general. For intervals the *sub-distributivity* law in (2.27) is held. When intervals have the same sign, the distributive law is held (2.28).

$$x(y + z) = xy + xz \tag{2.26}$$

$$X(Y + Z) \subseteq XY + XZ \tag{2.27}$$

$$X(Y + Z) = XY + XZ \quad , \quad Y.Z > 0 \tag{2.28}$$

2.1.7 Interval Functions

We want to find the range of $f(x)$, function on real value, when x is in interval X. This mapping is defined by (2.29).

$$f(X) = \{f(x) : x \in X\} \tag{2.29}$$

Elementary functions. When $f(x)$ is a monotonic function on x over interval X, then the range of the function is simply calculated by getting the end point values of the function. By this; an interval version for elementary functions, like e^X, $ln(X)$ and $\sqrt{X}$, is available.

$$f(X) = [f(\underline{X}), f(\overline{X})] \tag{2.30}$$

For other elementary functions that are not monotonic everywhere they could be defined on regions like $sin(X)$, $cos(X)$ and $abs(X)$.

General functions. A general function consists of set of basic arithmetic operations and elementary functions. The range for these functions may be calculated directly by replacing real valued variables with intervals. However this may produce unsatisfactory results as we shows in section 2.1.8. A function F in this case is called interval extension.

Definition 2.8. Interval extension. A function F is called interval extension for f, if for degenerate interval argument, F agrees with f, that is $F([x, x]) = f(x)$.

2.1.8 Interval Arithmetic Limitations

Interval arithmetic has limitations in sense of providing wider intervals in the functions range than that of (2.29). This is due to lake of distributivity and additive and multiplicative inverses in IA. Another issue with IA is the *interval dependency*, where it assumes that the variables intervals are independent from each other while it may be correlated. The following examples show some issues.

Example 2.1. In this example; we show the problem of interval dependency. Consider the function $f(x) = x^2$, we need to find the interval extension $F(X)$ for $X = [\underline{X}, \overline{X}]$.

$$\begin{aligned} F(X) &= X^2 \\ &= [\underline{X}, \overline{X}].[\underline{X}, \overline{X}] \end{aligned}$$

Replacing X^2 with $X.X$ doesn't provide accurate results unless X is positive or negative, i.e. monotonic on the domain of X. For instance if $X = [-1, 1]$ the result is $[-1, 1]$

while the most accurate results is $[0, 1]$. Note that we don't consider $[-1, 1]$ is wrong results as it already contains the desired range, however we can get more accurate results. $F(X) = X^2$ can be expressed as the definition on (2.29) be the following equation:

$$F(X) = X^2 = \begin{cases} [\underline{X}^2, \overline{X}^2] & , 0 \leq \underline{X} \\ [\overline{X}^2, \underline{X}^2] & , 0 \geq \overline{X} \\ [0, max\{\underline{X}^2, \overline{X}^2\}] & , \underline{X} < 0 < \overline{X} \end{cases} \tag{2.31}$$

Example 2.2. In the following example, we show the lack of distributivity and the interval dependency issues. Consider the function $f(x) = x(1 - x)$. The interval extension of this function can be expressed by the following equations.

$$F(X) = X(1 - X)$$
$$G(X) = X - X^2$$
$$H(X) = \frac{1}{4} - (X - \frac{1}{2})^2$$

When input interval is degenerate , the three formulas produce the same results. But they may produce different output intervals otherwise. For example if $X = [0, 1]$, then $F([0, 1]) = [0, 1]$, $G([0, 1]) = [-1, 1]$ and $H([0, 1]) = [0, \frac{1}{4}]$.

Example 2.3. In this example we show that absence of additive inverse; may prevent from solving simple equations. Consider A, C are constant intervals, then we fail to solve simple equation as $A + X = C$, putting $X = C - A$ doesn't satisfy the equation. Let $A = [1, 4]$ and $C = [3, 5]$, then $X = [3, 5] - [1, 4] = [-1, 4]$. But $[1, 4] + [-1, 4] \neq [3, 5]$

We can show the same example for $A.X = C$, where lack of multiplicative inverse makes the problem.

2.1.9 Interval Arithmetic Tools

A list of tools and software packages that utilize IA can be found in [13]. In our work a C++ class library for eXtended Scientific Computing (C-XSC) is used for interval arithmetic computations [14, 15].

2.2 Modal Interval Arithmetic

Modal interval Arithmetic (MA) can be considered as extension for the classical IA. MA solves some IA issues, which were shown in section 2.1. For this work; the main concern about MA is its algebraic properties, so we briefly introduce MA in this section.

2.2.1 Basics

MA has better algebraic relations than IA, modal intervals are acompletion of the classical intervals [16, 17]. MA is constructed from the set of numbers and a quantifier. Quantifiers define opposite selection modalities for the interval. Then we don't consider the interval order set only, but the whole set. The quantifiers are $\exists, \forall$.

2.2.1.1 Modal Interval Operators

For interval $X = [a, b]$ where $a, b \in \mathbb{R}$, we can define the following operators:

$$Dual(X) = [b, a] \tag{2.32}$$
$$Opp(X) = [-a, -b] \ = \ Dual(-X) \tag{2.33}$$
$$Impr(X) = [max\{a, b\}, min\{a, b\}] \tag{2.34}$$
$$Pro(X) = [min\{a, b\}, max\{a, b\}] \tag{2.35}$$

2.2.2 Basic Operations of Modal Arithmetic

For the basic arithmetic operations on modal intervals we can construct it the same way as IA, and using the operators defined by (2.32) to (2.35) .

Note that the algebraic properties of modal arithmetic operations are similar to Kaucher's completed interval arithmetic[18]. We will benefit from this in solving system of linear interval equations as described in chapter 3.

2.2.2.1 Algebraic Properties of Modal Arithmetic

Additive and multiplicative inverses. By using the operator $Dual()$. The additive and multiplicative inverses are found. For interval $X = [a, b]$:

$$\begin{aligned} X - Dual(X) &= [a, b] - [b, a] \\ &= [a - a, b - b] \\ &= [0, 0], \end{aligned}$$

$$\begin{aligned} X.\frac{1}{Dual(X)} &= [a, b].\frac{1}{[b, a]} \\ &= [\frac{a}{a}, \frac{b}{b}] \\ &= [1, 1]. \end{aligned}$$

Distributive law. Distributive law is stronger in MA than that in classical IA. For modal intervals X, Y and Z, we have:

$$Impr(X).Y + X.Z \subseteq X.(Y + Z) \subseteq Pro(X).Y + X.Z \tag{2.36}$$

Example 2.4. Let $X = [1, 3]$, $Y = [1, 1]$ and $Z = [-1, -1]$, then for MA we have:

$$\begin{aligned} X.(Y + Z) &= [1, 3].([1, 1] + [-1, -1]) \\ Impr(X).Y + X.Z &= [3, 1].[1, 1] + [1, 3].[-1, -1] \\ &= [3, 1] + [-3, -1] \\ &= [0, 0] \end{aligned}$$

Using IA:

$$\begin{aligned} X.(Y + Z) &= [1, 3].([1, 1] + [-1, -1]) \\ X.Y + X.Z &= [1, 3].[1, 1] + [1, 3].[-1, -1] \\ &= [1, 3] + [-3, -1] \\ &= [-2, 2] \end{aligned}$$

2.2.3 Modal Arithmetic Limitations

MA suffers from the dependency problem as the classical IA. Dependency problem is not solved by using the modal intervals. However MA in some cases produce tighter intervals than the classical ones, as shown in example 2.4 for the case of the sub-distributive law.

2.2.4 Modal Arithmetic Tools

The library used for our work is the same C-XSC library used for IA. We have modified the library to support the modal operators $Dual()$,$Opp()$, $Pro()$ and $Impr()$ to be used for the modal calculations.

2.3 Affine Arithmetic

Affine Arithmetic (AA) is another kind of improvement over the classical IA. In AA correlations of first order, are kept between input quantities and the computations. These correlations are kept in the variable affine form itself, which result usually better results than IA, and allow somehow to overcome the dependency problem.

2.3.1 Basic Concepts

In AA a quantity x is represented by a central value x_0, and the symbolic sum of terms $x_i\epsilon_i$ that represent the sources of uncertainty in the variable. Noise symbols ϵ_i are symbolic arbitrary variables that don't take a certain value, but lies in the interval $[-1, 1]$. The partial deviations x_i scale the noise symbols [19]. Then affine form is a first degree polynomial on the noise symbols, affine form can be represented as in (2.37).

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i\epsilon_i \quad , \epsilon_i \in [-1, 1] \tag{2.37}$$

A key concept in the AA is that quantities can share noise symbols, that is the noise symbol has a non-zero coefficient for these quantities. Quantities with no common symbols are completely independent, while others sharing some symbols, have a partial dependency for each noise symbol shared by their affine forms [19].

The radius $r(\hat{x})$ of the affine form is defined by (2.38). This radius represent the total deviation of $\hat{x}$.

$$r(\hat{x}) = \sum_{i=1}^{n} |x_i| \tag{2.38}$$

If a quantity x is represented with the affine form $\hat{x}$, then $x \in [x_0 - r(\hat{x}), x_0 + r(\hat{x})]$. Also, if $x \in [a, b]$, then x can be represented with the affine form $\hat{x} = x_0 + x_i\epsilon_i$, where $x_0 = (b + a)/2$ and $x_1 = (b-a)/2$. We can see here that we can make AA algorithms to input and output intervals.

2.3.2 Affine Arithmetic Operations

Addition and scalar multiplication. Linear arithmetic operations don't produce more noise terms in the output. Addition , subtraction and scalar multiplication are defined in equations (2.39) and (2.40).

$$\hat{x} \pm \hat{y} = (x_0 \pm y_0) + \sum_{i=1}^{n} (x_i \pm y_i)\epsilon_i \tag{2.39}$$

$$c\hat{x} = cx_0 + \sum_{i=1}^{n} cx_i\epsilon_i \tag{2.40}$$

Non-Linear functions. Extending non-affine operations requires that we use good affine approximation of the exact result and append an extra term to bound the error of this approximation. For example to compute a non-affine operation of two variables, $z =$

$f(x, y)$. Given affine forms $\hat{x}$ and $\hat{y}$ representing x and y respectively, we want to compute an affine form $\hat{z}$. First, we see z as a function of the noise symbols in $\epsilon_1, \ldots, \epsilon_n$ as following.

$$z = f(x_0 + \sum_{i=1}^{n} x_i\epsilon_i \ , \ y_0 + \sum_{i=1}^{n} y_i\epsilon_i)$$
$$z = f^*(\epsilon_1, \ldots, \epsilon_n),$$

where f^* is a function $\mathbb{U}^n \to \mathbb{R}$. In general, f^* is not an affine function of $\epsilon_1, \ldots, \epsilon_n$. So, we approximate f^* over $\mathbb{U}^n$ by an affine function f^a with error bound δ:

$$|f^a - f^*| \leq \delta \quad \text{for all} \quad \epsilon_1, \ldots, \epsilon_n \in \mathbb{U}. \tag{2.41}$$

Writing

$$f^a(\epsilon_1, \ldots, \epsilon_n) = z_0 + \sum_{i=1}^{n} z_i\epsilon_i \tag{2.42}$$

we obtain that $z = f(x, y)$ is represented by the affine form

$$\hat{z} = z_0 + \sum_{i=1}^{n} z_i\epsilon_i + z_{n+1}\epsilon_{n+1}, \tag{2.43}$$

where $z_{n+1} = \delta$ and ϵ_{n+1} is a new noise symbol. The challenge is to find an affine approximation f^a that is easy to compute but which has a small approximation error δ. Note that the introduction of the term $z_{n+1}\epsilon_{n+1}$ to represent the non-affine part of $f^*(\epsilon_1, \ldots, \epsilon_n)$ implies a loss of information: from this point on, the noise symbol ϵ_{n+1} will be implicitly assumed to be independent from $\epsilon_1, \ldots, \epsilon_n$, when in fact it is a (non-affine) function of them. Any subsequent operation that takes $\hat{z}$ as input will not be aware of this constraint between ϵ_{n+1} and $\epsilon_1, \ldots, \epsilon_n$, and therefore may return an affine form that is less precise than necessary.

Multiplication. We can define the multiplication as following, adding a new noise symbol ϵ_{n+1} to account for nonlinear operation.

$$\hat{x}.\hat{y} := (x_0.y_0) + \sum_{i=1}^{n}(x_0.y_i + x_i.y_0)\epsilon_i + r(\hat{x}).r(\hat{y}).\epsilon_{n+1} \tag{2.44}$$

Inversion and division. Inverse of $\hat{x}$ can be represented using Taylor series expansion as following:

$$\frac{1}{\hat{x}} = \frac{1}{x_0 + \sum_{i=1}^{n} x_i \epsilon_i}$$
$$= \frac{1}{x_0} - \frac{1}{x_0^2} \sum_{i=1}^{n} x_i \epsilon_i + \frac{1}{x_0^3} \left(\sum_{i=1}^{n} x_i \epsilon_i \right)^2$$

Then the affine form for the inverse can be represented by (2.45). Division then can be calculated as a multiplication by the inverse of the denominator [2].

$$\frac{1}{\hat{x}} = \frac{1}{x_0} - \frac{1}{x_0^2} \sum_{i=1}^{n} x_i \epsilon_i + k_0 + k_1 \epsilon_{n+1}, \tag{2.45}$$

$$k_0 = \frac{1}{x_0} + \frac{1}{x_0^3} \sum_{i=1}^{n} x_i^2, \tag{2.46}$$

$$k_1 = \frac{1}{x_0^3} \sqrt{5 \sum_{i=1}^{n} x_i^2 - 3 \sum_{i=1}^{n} x_i^4}. \tag{2.47}$$

Arbitrary functions. we may combine the arithmetic operations and the elementary functions to get affine representations for arbitrary functions.

2.3.2.1 Affine Arithmetic versus Interval Arithmetic Examples

Example 2.5. This example shows how the AA overcomes the dependency problem. For three affine quantities $\hat{x}$, $\hat{y}$ and $\hat{z}$, and the corresponding intervals X, Y and Z. Results are listed in table 2.1

Table 2.1: Affine Expressions and their interval counterparts

Affine arithmetic		Interval arithmetic	
Form	Width	Form	Width
$\hat{x} = 7 + 2\epsilon_1$	4.0	$X = [5, 9]$	4.0
$\hat{y} = 5 + 2\epsilon_1$	4.0	$Y = [3, 7]$	4.0
$\hat{z} = 5 + 2\epsilon_2$	4.0	$Z = [3, 7]$	4.0
$\hat{x} - \hat{y} = 2$	0.0	$X - Y = [-2, 6]$	8.0
$\hat{x} - \hat{z} = 2 + 2\epsilon_1 + 2\epsilon_2$	8.0	$X - Z = [-2, 6]$	8.0

Example 2.6. An example to show the output range of a general function using classical IA and AA is introduced here. Consider $f(x) = (1+x)(1-x)$ for $x \in [-2, 2]$. The actual range of this function is $[-3, 1]$. From the results below we can see that AA produces

narrower range than IA. AA doesn't produce the exact range due to truncation of higher order noise symbol.

Using interval arithmetic.

$$\begin{aligned} f([-2,2]) &= (1+[-2,2])(1-[-2,2]) \\ &= ([-1,3])([-1,3]) \\ &= [-3,9] \end{aligned}$$

Using affine arithmetic. Let $\hat{x} = 0 + 2\epsilon$, and note that $\epsilon \in [-1,1]$

$$\begin{aligned} f(\hat{x}) &= (1+2\epsilon)(1-2\epsilon) \\ &= 1-4\epsilon \\ f(\hat{x}) &\in [-3,5] \end{aligned}$$

2.3.3 Affine Arithmetic Limitations

Although AA provide better results than IA, It doesn't solve dependency problem completely, due to ignorance of correlation between the symbols generated from non-linear operations. Also the symbolic representations for the variables, may present a capacity problem, due to the increasing number of terms while calculation.

2.3.4 Affine Arithmetic Tools

For our work; `aaflib` library is used for affine arithmetic calculations [20].

Chapter 3

Interval Circuits Simulator Design

Through this chapter, we present interval simulator design requirements. A proposal for modifications for the traditional simulation flow, with a flow using existing designs by replacing statistical parameters variations by interval parameters. Simulator design aspects are presented in section 3.3, testing requirements are descried in section 3.4.

3.1 Introduction

In our work we try to put the guidelines for an intervals based simulator. The aim of the interval simulator is to eliminate the need for multiple runs of MC simulations, or at least reduce the number of runs. To keep the backward compatibility with currently used flows, we introduce interval simulation flow that can coexist with the traditional MC simulation flow. Interval simulation is used in [2, 8], in these simulators, the parameters causing system variability are kept in mind along the simulation. In our work, the simulator work on general intervals. We work to implement the simulator core and a set of models that use interval calculations.

3.2 Simulation flow

The proposed simulation flow for our simulator is targeting benefit from current simulation flows and already existing designs. As is known, many designs may be re-used from technology node to another. The interval based simulation flow introduced here works to keep the conventional front end without change. This flow can be coexisting with the current traditional MC simulation flow; the flow is shown in figure 3.1.

Here instead of doing N times MC simulations, it generates N samples out of the input design. Statistical parameters involved in the design are changed every time according to the defined probability density functions in the design. Each change in the parameters is recorded and then after N times, a new design is generated with the statistical parameters

replaced by an interval parameters. A key point here is that the sampling N times is trivial compared to sample and simulate N times.

In available circuit simulators, variations is usually defined by either Gauss, or uniform distributions. In uniform distribution, variation is given as number where parameter value may change around the mean value by at most this number. Table 3.1 shows an example for how a design with variations may be represented in equivalent interval design. In MC design, resistor nominal value is 1000Ω, while variation follows an uniform distribution allowing a change by $\pm 3\%$ of the nominal value. The capacitor in the other hand has a nominal of 1n F, and variation follows an Gaussian distribution with stranded deviation (σ) equal to 5% of the nominal value. Resistor value is represented in the interval design by an interval of $[970, 1030]$, while capacitor value is represented by an interval of $[8.5 \times 10^{-10}, 1.15 \times 10^{-9}]$, which represent three times the stranded deviation around the nominal value.

3.3 Simulator Parts

We can divide circuit simulator into four parts:

1. The front end that captures and processes the input circuit design and options.
2. The models library containing mathematical equations defining the characteristic of each component in the design.
3. The solver formulates and solves the equations describing the design.
4. The back end, in which the results are processed and displayed in a proper way.

Designers mostly interact with front and back ends only, so it is important to keep them as much as possible similar to the conventional ones. And for interval simulations, we may add specified features and options.

3.3.1 Simulator Front-End

Simulator front-end has to have two main capabilities to use both floating point parameters and interval parameters, as well keeping backward compatibility for old designs, these capabilities are:

- Along with the traditional parameters types as floating, integers and strings; interval parameter is a new type to deal with.
- The ability of converting the traditional methods that represent and control variability to interval based method. For example; it converts probability density functions, describing parameter variation, to interval notation to be used in interval simulation.

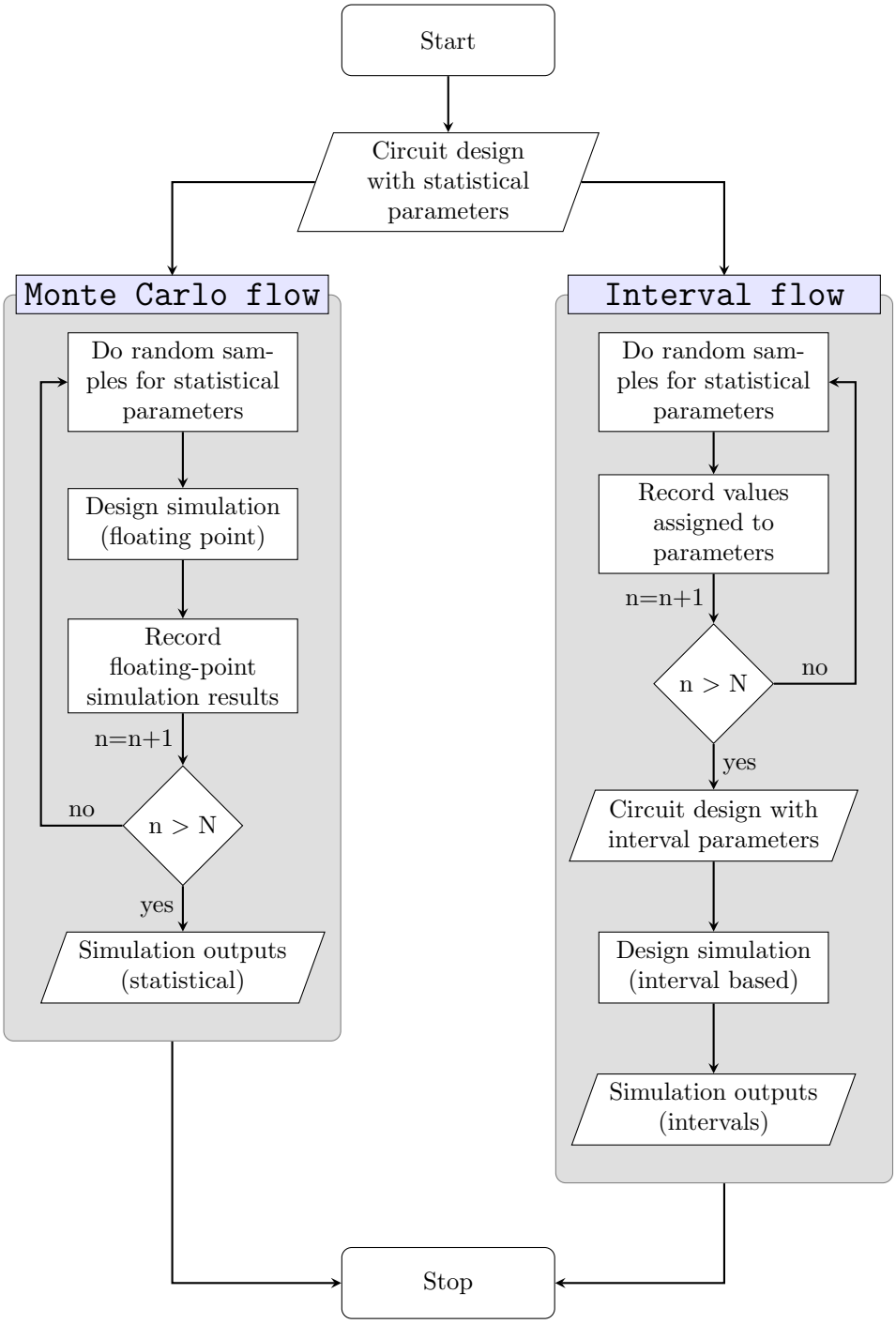


Figure 3.1: MC and interval based circuit simulation flow

Table 3.1: Design with statistical parameters and its interval equivalence

Statistical design

```
*
.param
+ tPeriod= 1e-6
+ tStep  = '1e-2*tPeriod'
+ tEnd   = '5.0*tPeriod'
+ tDelay = '0.2e-6'
+ tRaise = '1e-9'

.option dump_mcinfo
.param rvalnom  = 1000
.param cvalnom  = 1e-9

.subckt rc in out
.param rval = 'rvalnom'
+ lot=3%
.param cval = 'cvalnom'
+ lot/gauss=5%

rr in  out 'rval'
cc out 0   'cval'
.ends

x_s1 in out rc

vin in  0 0.0 pwl
+ 0.0              0.0
+ tDelay           0.0
+ 'tDelay+tRaise'  1.0
+ tEnd             1.0

.tranStep tEnd

.plot V(in) V(out)
.printV(in) V(out)
.mc 1000

.end
```

Interval design

```
*
.param
+ tPeriod= 1e-6
+ tStep  = '1e-2*tPeriod'
+ tEnd   = '5.0*tPeriod'
+ tDelay = '0.2e-6'
+ tRaise = '1e-9'

.option dump_mcinfo
.param rvalnom  = 1000
.param cvalnom  = 1e-9

.subckt rc in out
.param rval = [970,1030]
.param cval = [8.5e-10,1.15e-9]

rr in  out 'rval'
cc out 0   'cval'
.ends

x_s1 in out rc

vin in  0 0.0 pwl
+ 0.0              0.0
+ tDelay           0.0
+ 'tDelay+tRaise'  1.0
+ tEnd             1.0

.tranStep tEnd

.plot V(in) V(out)
.printV(in) V(out)

.end
```

The second feature is key feature to keep backward compatibility with the current flows, as shown in figure 3.1.

3.3.2 Simulator Back-End

The simulator back-end is responsible to process the simulation results, and present them in a proper way. This can be done directly from the simulator code, or by external tools and scripts. In our work we go for the second option for simplicity of implementation.

3.3.3 Simulator Kernel

Simulator kernel is the module where actual simulation is done. The system of equations that describes the input design is formulated using interconnects stated in the inputs, with the mathematical model for each device used.

Transient simulation for traditional and interval based simulation may be described by figure 3.2, the difference is replacing conventional operations and algorithms with these for interval arithmetic. To go through this flow we need to discuss two main parts of the simulator kernel:

- Solving algorithm: linearization, matrix solving and convergence
- Models library, which are involved in equations evaluation and linearization.

Solution algorithm. Analog circuits are defined by nonlinear system of differential algebraic equation. Nodes voltages and currents in branches are determined implicitly by solving the system equations. To deal with non-linear equations, we apply an interval version from Newton's method for solving system of non-linear equations. Basics of the method is introduced in appendix A and more details in [11]. For simplicity we use the Back Euler integration method for solving the differential equations.

Matrix solver. Conventional simulator usually uses LU decomposition to solving the system of linear equations comes out of Newton's iterations. For interval solver we use an interval version of Gauss-Seidel iterative method, algorithm is introduced in appendix A and more details in [11]. A modification for the Gauss-Seidel method for modal intervals is described in [21].

Models library. The basic set of elements in the models library consists of sources, passive (linear) elements and active (non-linear) elements. Table 3.2 shows the elements covered in this work.

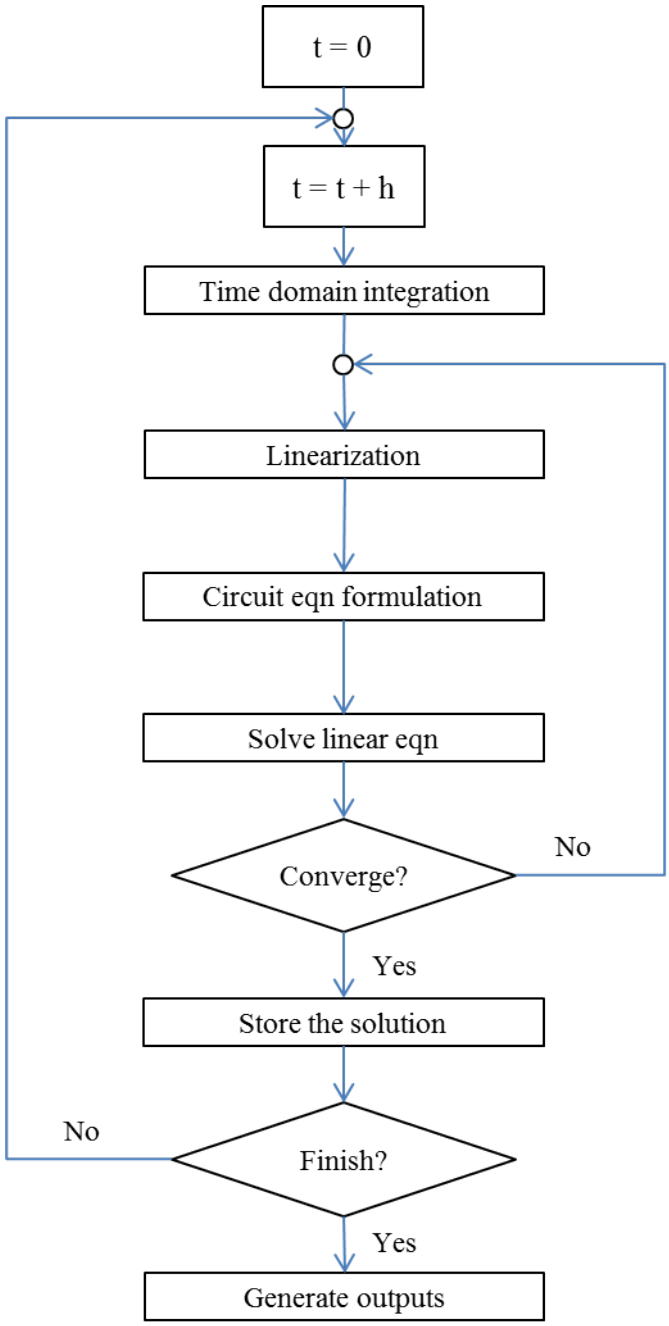


Figure 3.2: Traditional transient simulation flow

Table 3.2: Models library

Sources	Passive elements	Active elements
Current source	Resistor	Simple diode
Voltage source	Capacitor	Simple MOSFET model
	Inductor	Advanced MOSFET model

3.4 Simulator Testing

To ensure good results from the interval simulator, testing should cover various aspects. Testing should target individual parts and the integration between them. As time spent in the simulation is the main concern for the interval simulator, timing reports versus MC simulation should be recorded. In our work we compare against the time taken in 1000's MC runs. Testing is targeting mainly the kernel of the simulator, the models library and the solver, front-end and back-end are not a concern for time being.

For accuracy concerns each individual model has to be characterized using degenerate intervals test vectors on our simulator and a traditional circuit simulator, results in this case should be almost the same. Any difference should be only due to rounding errors. MC simulations results are then compared to normal intervals simulation, interval results should include MC results.

Unit testing described above is used to test the models library, to test the solver we use two types of circuits configurations. The first is the all passive elements circuits, this type of circuits is linear and so we can discard the linearization step out of the simulation to test only the linear matrix solver. The second configuration is done by adding non-linear models (active) to the circuit, using simple circuits of the second configuration one can test the algorithm used to solve the non-linear system of equations. For these testing types we will compare the timing versus MC simulations as well. The true test for the speed of the simulation would be a bigger circuit, this will measure the simulator efficiency and capacity. Table 3.3 summarizes the testing required for the simulator.

Table 3.3: Simulator tests

Test	Target
Unit testing	Element wise testing, to compare interval results versus floating point results.
Passive circuits testing [1]	Test accuracy of different algorithms of solving interval linear systems versus MC simulation.
Active elements circuits [1]	Test accuracy of different algorithms of solving interval non-linear systems versus MC simulation.
Big circuits [1]	Test the reliability of the simulator (capacity and speed).

3.5 Available Analysis

For time being only DC and transient analysis are available for our simulator.

[1]Time would be recorded for these types of testing

Chapter 4

Models Library

In this chapter we present details about the models library implemented for the interval simulator. We follow the Modified Nodal Analysis (MNA) way in constructing the system of equations that represent the system network. MNA is a widely used technique in circuit simulation [22]. Through the chapter we discuss issues and difficulties that arise from converting floating point models to interval or affine form models.

4.1 Voltage and Current Sources

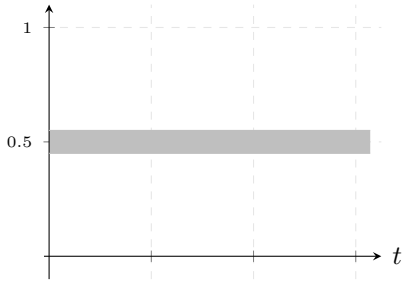
Our library contains independent current and voltage sources. Voltage and current sources accept interval functions, the output may be constant DC value, sine wave, step wave and Piecewise Linear (PWL) function. Examples of sources functions are in figure 4.1.

In MNA independent current source is straight forward implemented by contributing the current value in the current vector, at the nodes connecting the source. Figure 4.2 shows the voltage source representation with MNA, where these numbers represent the contribution to the system matrix and the right hand side.

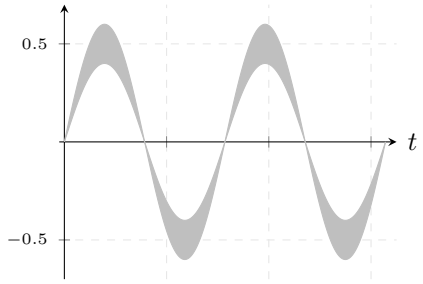
4.2 Linear Elements

The library contains the ideal linear elements, resistor, capacitor and inductors. The interval models for these ideal linear elements are just the same of these used for floating point models only we replace the floating point numbers with intervals. Using MNA, resistor contribution to the conductance matrix is illustrated by figure 4.3, where $G = 1/R$.

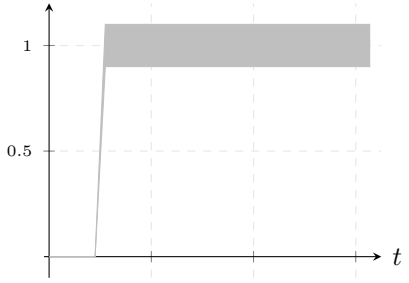
Capacitor implementation. for an ideal capacitor, equations (4.1) and (4.2) represent differential relation between capacitor current and voltage. For backward Euler integration, capacitor current can be represented by (4.3), and it can be rewritten as (4.4). This



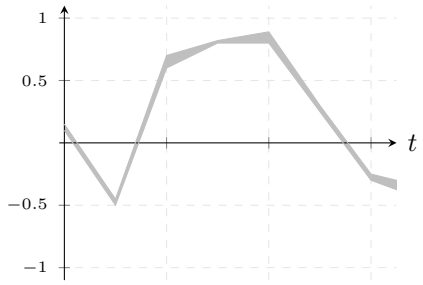
(a) DC interval source



(b) Sine wave interval source

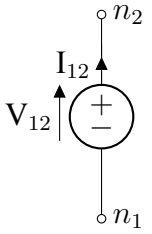


(c) Step wave interval source



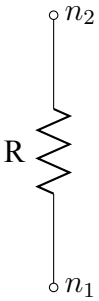
(d) PWL wave interval source

Figure 4.1: Sources functions examples



$$\begin{bmatrix} +0 & +0 & +1 \\ +0 & +0 & -1 \\ +1 & -1 & 0 \end{bmatrix} \begin{bmatrix} v_{n1} \\ v_{n2} \\ \mathrm{I}_{12} \end{bmatrix} = \begin{bmatrix} +0 \\ -0 \\ +\mathrm{V}_{12} \end{bmatrix}$$

Figure 4.2: Voltage source representation with MNA



$$\begin{bmatrix} +\mathrm{G} & -\mathrm{G} \\ -\mathrm{G} & +\mathrm{G} \end{bmatrix}$$

Figure 4.3: Resistor representation with MNA

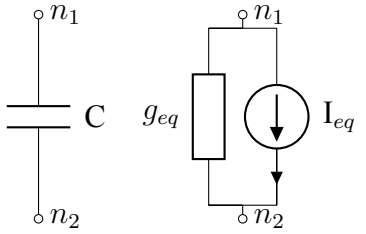
equation represent a current source and conductance, where h is the time step width, n is the time step number, $I_{eq} = -C/h^n.V_C^n$ and $g_{eq} = C/h^n$. Figure 4.4 shows the capacitor transient equivalent circuit, and the contribution in conductances matrix and currents vector.

$$I_C(t) = C.\frac{dV_c}{dt} \tag{4.1}$$

$$\frac{I_C(V,t)}{C} = \frac{dV_c}{dt} = f(x,t) \tag{4.2}$$

$$I_C^{n+1} = \frac{C}{h^n}V_C^{n+1} - \frac{C}{h^n}V_C^n \tag{4.3}$$

$$I_C^{n+1} = g_{eq}.V_C^{n+1} + I_{eq} \tag{4.4}$$



$$\begin{bmatrix} +g_{eq} & -g_{eq} \\ -g_{eq} & +g_{eq} \end{bmatrix} \begin{bmatrix} v_{n1}^{n+1} \\ v_{n2}^{n+1} \end{bmatrix} = \begin{bmatrix} -\mathrm{I}_{eq} \\ +\mathrm{I}_{eq} \end{bmatrix}$$

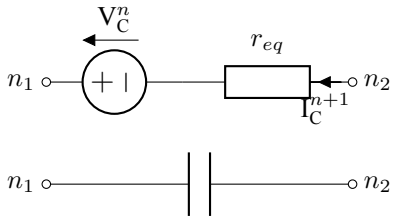
Figure 4.4: Capacitor model transient equivalent circuit

This representation for capacitor has an issue, that is the capacitor term appears in both equation sides. This makes the voltage on the capacitor to increase as the simulator iterations goes, which may produce bad results as we will see in chapter 5. To avoid this issue we add new equations to the system to solve for the capacitor current.

Equation (4.3) is rewritten as (4.5). The matrix representation and the equivalent circuit for this model are shown by the figure 4.5. Note that we reverse the direction of the capacitor current.

$$V_C^{n+1} = V_C^n - \frac{h^n}{C}I_C^{n+1} \tag{4.5}$$

$$= V_C^n - r_{eq}I_C^{n+1} \tag{4.6}$$



$$\begin{bmatrix} +0 & +0 & +1 \\ +0 & +0 & -1 \\ +1 & -1 & -r_{eq} \end{bmatrix} \begin{bmatrix} v_{n1}^{n+1} \\ v_{n2}^{n+1} \\ \mathrm{I_C^{n+1}} \end{bmatrix} = \begin{bmatrix} +0 \\ +0 \\ \mathrm{V_C^n} \end{bmatrix}$$

Figure 4.5: Another representation for capacitor model equivalent circuit

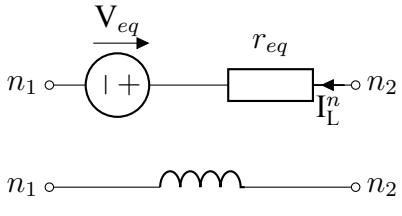
Inductor implementation. The same as for capacitor, equations (4.7) to (4.10) and figure 4.6 show the inductor transient equivalent circuit for back Euler integration.

$$V_L(t) = L.\frac{dI_L}{dt} \tag{4.7}$$

$$\frac{V_L(I,t)}{C} = \frac{dI_L}{dt} = f(x,t) \tag{4.8}$$

$$V_L^{n+1} = \frac{L}{h^n} I_L^{n+1} - \frac{L}{h^n} I_L^n \tag{4.9}$$

$$V_L^{n+1} = r_{eq}.I_L^{n+1} + V_{eq} \tag{4.10}$$



$$\begin{bmatrix} +0 & +0 & +1 \\ +0 & +0 & -1 \\ +1 & -1 & -r_{eq} \end{bmatrix} \begin{bmatrix} v_{n1}^{n+1} \\ v_{n2}^{n+1} \\ \mathrm{I}_\mathrm{L}^{n+1} \end{bmatrix} = \begin{bmatrix} +0 \\ +0 \\ V_{eq} \end{bmatrix}$$

Figure 4.6: Inductor transient equivalent circuit

4.3 Non-Linear Elements

With non-linear elements, we mean the circuit elements that have non-linear behavior between the voltage applied on element terminals and the current output from these terminals. Here we evaluate usage of affine arithmetic in implementing the interval version of model used. We expect that the dependency problem , discussed in chapter 2, would affect models accuracy as models contains more equations and more parameters to describe more effects. In the following we show first the technique used to implement the interval version of the models used, then we present more details and specific issues per model.

4.3.1 Techniques to Get Interval Models

As we discussed before, that getting the interval version of a function is not that simple by replacing the floating point calculations with relevant interval version. That is due to weak algebraic proprieties of IA, and due to the dependency problem.

4.3.1.1 Convert "if-else" conditions to states.

It is a common case to write some equations on regions, where at each region there is different formula, like the simple MOSFET equation (4.11). In general case we do not limit the inputs to the model such that branching conditions are confined to one region. For example in condition $(X > a)$, it happens that some points in the interval X may satisfy the condition and others may not. In this case we split the interval to sub-intervals, where each sub-interval satisfy part of the condition. $if - else$ conditions are then converted to separate if's. these if's represent states that may be exist at the same time. The output intervals of the different states are combined to get the final result. Table 4.1 shows an example code for getting the states.

Table 4.1: Changing $if - else$ conditions to states

Conditions example	States example

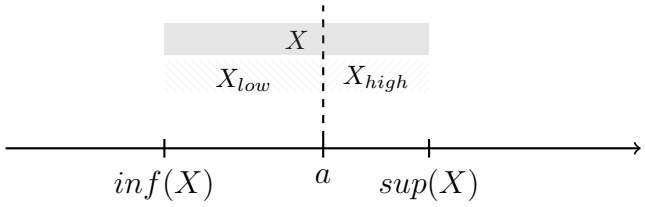
`double x,z,a;`
 	
`if (x > a)`
`  z = f(x);`
`else`
`  z = g(x);`
 	
`return(z);`
	`double a;`
	`interval X, X_low, X_high;`
	`interval Z, Z_low, Z_high;`
	
	`// Initialization`
	`X_low = X;`
	`X_high = X;`
	
	`if (sup(X_low) > a)`
	`  sup(X_low) = a;`
	
	`if (inf(X_high) < a)`
	`  inf(X_high) = a;`
	
	`// all points in the interval`
	`// should satisfy the condition`
	`if (X_low < a)`
	`  Z_low = G(X_low);`
	
	`if(X_high > a)`
	`  Z_high = F(x_high);`
	
	`// Z is the interval hull of`
	`  Z_low and Z_high`
	`Z = Z_low | Z_high;`
	
	`return(Z);`

Where $inf()$ is a function to get/set the lower limit of the interval, and $sup()$ is a function to get/set the upper limit of the interval.

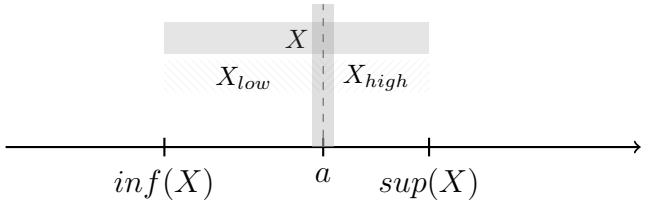
We have to note here that the output interval may contain values that are not in the output range. This happens when the function range is not continuous, and then the output is an interval hull, as described by definition 2.5.

4.3.1.2 Comparing two intervals.

We show in the introduction for the interval arithmetic; that operators like '*less than*' is not fully defined between intervals. This leaving ambiguity about how to split the interval. Here we use an assumption that the reference interval, the interval we compare with, is narrow relative to the input interval. Then consider it a single point, it may be any point in the reference interval. Then we can use the procedure described in section 4.3.1.1, figure 4.7 illustrate this comparison.



(a) Comparing interval to single point value



(b) Comparing interval to narrow interval

Figure 4.7: Interval comparing

4.3.1.3 Reduce dependency problem.

Whenever possible we re-write the equations to reduce the dependency problem. No specific rule here but reduce the appearance of the same variable in one function as possible. For example replacing repeated multiplication with a power function.

4.3.1.4 Using affine arithmetic.

Using of affine arithmetic provides better results concerning the dependency problem, but regarding the intervals comparisons we have the same issues described before. We cannot use the same techniques used for IA, because splitting interval here produce new variables, and so we lose the advantage of keeping correlation along the calculation. For AA, we use an assumption that the variation to the parameters does not change the domain of model equations. Then for conditions we compare the affine interval mean, interval mid-point, to each other.

4.3.2 Simple Diode Model

We begin our library for the non-linear elements, with a simple P-N junction diode model. The model is simple and have a monotonic behavior between input voltage and output current. The model parameters and model Verilog code, from [23], are in tables 4.2 and 4.3.

Code listed in table 4.3 defines the electrical characteristic diode current (I_d) versus the voltage difference across its terminals (V_d). As shown parameters involved in the equation are almost un-repeated, so results obtained from IA calculations and AA calculations are very close to each other as we would see in chapter 5.

Note that in C++ code function `IsDual()` is used to check if the interval is a proper interval, that is ordered interval. This is a way to check that after splitting the interval the corresponding condition applies to it. We may note that nested conditions are handled by splitting the interval again.

Table 4.2: Diode model parameters

Name	Description
area	device area
is	saturation current
n	emission coefficient
cjo	zero-bias junction capacitance
m	grading coefficient
phi	body potential
fc	forward bias capacitance factor
tt	transit time
bv	reverse breakdown voltage
rs	series resistance

Table 4.3: Verilog-a code for the diode model

```
module diode(anode,cathode);
 inout anode,cathode;
 electrical anode,cathode;

 parameter real area = 1 from (0:inf);
 parameter real is=1e-14 from (0:inf);
 parameter real n=1 from (0:inf);
 parameter real cjo=0 from [0:inf);
 parameter real m=0.5 from [0:inf);
 parameter real phi=0.7 exclude 0;
 parameter real fc=0.5 from (0:1];
 parameter real tt=1p from [0:inf);
 parameter real bv=1.0e+100 from [0:inf);
 parameter real rs=0 from [0:inf);

 real Vd, Id, Qd;
 real f1, f2, f3, fcp;
 real ibv;

```

```
 analog
   begin
     @(initial_step)
     begin
       f1 = (phi/(1 - m))*(1 - pow((1 - fc), m));
       f2 = pow((1 - fc), (1 + m));
       f3 = 1 - fc*(1 + m);
       fcp = fc*phi;
       ibv = is*bv/vt();
     end

     Vd = V(anode,cathode);
     Id = I(anode);

     // intrinsic diode.
     if (Vd < -5*n*vt())
     begin
       if (Vd == -bv)
         I(anode,cathode) <+ -area*ibv;
       else
         if (Vd > -bv)
           I(anode,cathode) <+ -area*is;
         else
           I(anode,cathode) <+ -area*is*(exp(-(bv + Vd)/vt())
                                - 1 + bv/vt());
     end
     else
       I(anode,cathode) <+ area*is*(exp((Vd - rs*Id)/(n*vt())) - 1);

     // capacitance (junction and diffusion).
     if (Vd <= fcp)
       Qd = tt*Id + area*cjo*phi
            * (1 - pow((1 - Vd/phi), (1 - m)))/(1 - m);
     else
       Qd = tt*Id + area*cjo*(f1 + (1/f2)*(f3*(Vd - fcp) +
                              (0.5*m/phi)*(Vd*Vd - fcp*fcp)));
     I(anode,cathode) <+ ddt(Qd);
   end
endmodule
```

Table 4.4: C++ code for the diode model interval representation

```
#include <iostream>
#include <cmath>
#include <interval.hpp>
#include "diode.h"

using namespace cxsc;
using namespace std;

#define k 1.3806488E-23
#define q 1.602176565E-19

// Interval util
#define EMPTY(x)                    x=cxsc::interval(cxsc::MaxReal, -
    cxsc::MaxReal)
#define INF(x)                      cxsc::_double(cxsc::Inf(x))
#define SET_INF(x,a)                cxsc::SetInf(x,a)
```

```
#define SUP(x)                          cxsc::_double(cxsc::Sup(x))
#define SET_SUP(x,a)                    cxsc::SetSup(x,a)

interval diode_interval(interval vAnode, interval vCathode)
{
  interval area  = interval(AREA);
  interval is    = interval(IS);
  double   n     =          (1);
  interval cjo   = interval(0);
  interval m     = interval(0.5);
  interval phi   = interval(PHI);
  interval fc    = interval(FC);
  interval tt    = interval(1.0e-12);
  double   bv    =          (1.0e+100);
  interval rs    = interval(0);
  double   temp  =          (25.0);

  interval Vd, Id, Qd;
  interval f1, f2, f3, fcp;
  interval ibv;
  double vt = (k * (temp + 273.15) / q);

  // init
  EMPTY(Id);
  f1 = (phi/(1 - m))*(1 - pow((1 - fc), m));
  f2 = pow((1 - fc), (1 + m));
  f3 = 1 - fc*(1 + m);
  fcp = fc*phi;
  ibv = is*bv/vt;

  Vd = vAnode - vCathode;

  // intrinsic diode.
  interval Vd_tmp = Vd;
  if (SUP(Vd) > (-5*n*vt))
    SET_SUP(Vd, (-5*n*vt));
  if (! IsDual(Vd))
  {
    {
      interval Vd_tmp = Vd;
      if (INF(Vd) < -bv)
        SET_INF(Vd, (-bv));
      if (! IsDual(Vd))
      {
        Id |= -area*is;
      }
      Vd = Vd_tmp;

      Vd_tmp = Vd;
      if (SUP(Vd) >  -bv)
        SET_SUP(Vd, (-bv));
      if (! IsDual(Vd))
      {
        Id |= -area*is*(exp(-(bv + Vd)/vt) - 1 + bv/vt);
      }
      Vd = Vd_tmp;
    }
  }
```

```
74    Vd = Vd_tmp;
75
76    Vd_tmp = Vd;
77    if (INF(Vd) < (-5*n*vt))
78      SET_INF(Vd, (-5*n*vt));
79    if (! IsDual(Vd))
80    {
81      Id |= area*is*(exp((Vd)/(n*vt)) - 1);
82    }
83    Vd = Vd_tmp;
84    return (Id);
85  }
```

4.3.3 Simple MOSFET Model

The simple MOSFET model is implemented based on the basic equation (4.11) [24]. Interval and affine model codes are listed in tables 4.5 and 4.6. We can note that the affine code is almost the same as the floating point code. We use `sqr()` function instead of multiplication, to reduce the dependency problem.

$$I_{ds} = \begin{cases} 0 & , V_{gs} < V_{th} \\ kp\,\frac{w}{l}\,(V_{gsEff}\,V_{ds} - 0.5\,V_{ds}^2)\,(1 + \lambda\,V_{ds}) & , V_{ds} \leq V_{gsEff} \\ 0.5\,kp\,\frac{w}{l}\,V_{gsEff}^2\,(1 + \lambda\,V_{ds}) & , V_{ds} \geq V_{gsEff} \end{cases} \tag{4.11}$$

Where $V_{gsEff} = V_{gs} - vto$.

Table 4.5: C++ code for simple MOSFET model interval representation

```
1   interval mosfet_interval(interval vd, interval vg, interval vs)
2   {
3     interval w     = interval(W);
4     interval l     = interval(L);
5     interval kp    = interval(KP);
6     interval vto   = interval(VTO);
7     interval lamda = interval(LAMDA);
8
9     interval vgs    = interval(vg - vs);
10    interval vds    = interval(vd - vs);
11    interval vgsEff = vgs - vto;
12
13    interval ids;
14    EMPTY(ids);
15
16    interval tmpVgs = vgs;
17    if (SUP(vgs) > SUP(vto))
18      SET_SUP(vgs, SUP(vto));
19    if (! IsDual(vgs))
20    {
21      ids |= 0.0;
22    }
23    vgs = tmpVgs;
24
25    tmpVgs = vgs;
26    if (INF(vgs) < INF(vto))
```

```
    SET_INF(vgs, INF(vto));
  if (! IsDual(vgs))
  {
    interval tmpVds = vds;
    if (SUP(vds) > SUP(vgsEff))
      SET_SUP(vds, SUP(vgsEff));
    if (! IsDual(vds))
    {
      ids |= kp * (w / l) * ((vgsEff)*vds - 0.5 * sqr(vds)) * (1 +
   lamda * vds);
    }
    vds = tmpVds;

    tmpVds = vds;
    if (INF(vds) < INF(vgsEff))
      SET_INF(vds, INF(vgsEff));
    if (! IsDual(vds))
    {
      ids |= 0.5 * kp * (w / l) * sqr(vgsEff) * (1 + lamda * vds);
    }
    vds = tmpVds;
  }
  vgs = tmpVgs;

  return (ids);
}
```

Table 4.6: C++ code for simple MOSFET model affine representation

```
interval mosfet_affine(interval vd, interval vg, interval vs)
{
  affine w     = affine(W);
  affine l     = affine(L);
  affine kp    = affine(KP);
  affine vto   = affine(VTO);
  affine lamda = affine(LAMDA);

  affine vgs    = affine(vg - vs);
  affine vds    = affine(vd - vs);
  affine vgsEff = vgs - vto;

  affine ids;

  if (vgs < vto)
  {
    ids = 0.0;
  }
  else
  {
    if (vds <= vgsEff)
    {
      ids = kp * (w / l) * ((vgsEff)*vds - 0.5 * sqr(vds)) * (1 +
   lamda * vds);
    }
    else
    {
      ids = 0.5 * kp * (w / l) * sqr(vgsEff) * (1 + lamda * vds);
    }
```

```
29    }
30
31    return (ids.convert()); // return interval value
32  }
```

4.3.4 Advanced MOSFET Models

4.3.4.1 MVS Model

The MIT Virtual Source (MVS) model is a semi-empirical compact model for nanoscale transistors that accurately describes the physics of quasi-ballistic transistors with only a few physical parameters [25]. The model code is about 300 lines only, which is considered small model. We implement an interval version form the model using affine form. Conversion to affine version is somehow straight froward, using the techniques described in this section. The affine code is listed in appendix B. Testing results are presented in chapter 5.

4.3.4.2 BSIM4 Model

BSIM4 is an accurate compact model for MOSFET transistors [26], which is widely used by electronic circuit industry. We implemented an interval version for BSIM4. However due to the model code complexity, 7500 lines of code, the results obtained were not correct compared to the floating point model.

Chapter 5

Results

Through this chapter we present the testing done to the simulator according to aspects stated in section 3.4. Testing is limited to the actuality achieved work in the interval simulator.

5.1 Unit Testing

Unit testing is aim to test the results of individual models, to ensure that interval results we get, are accurate and reasonable compared to floating point results. In the section we don't cover sources and linear elements, because they are simple in themselves, and the actual testing for them is hold in circuit testing. So next we show testing results for the non-linear (active) elements models.

5.1.1 Capacitor Model

We test the Capacitor model by check the step response for an RC section as in figure 5.1. Figure 5.2 shows results for the interval version of the capacitor traditional MNA model. Results in case of using degenerate intervals, that is no variation in R nor C values, are correct. While when the resistor value by %3, it produces wrong results as results do not include the nominal value. When varying both the resistor and capacitor values, simulation does not converge.
Modified Model shows better results, figure 5.3 present this results compared to results obtained from MC simulations

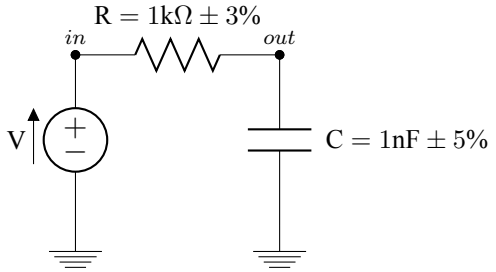


Figure 5.1: RC section schematic

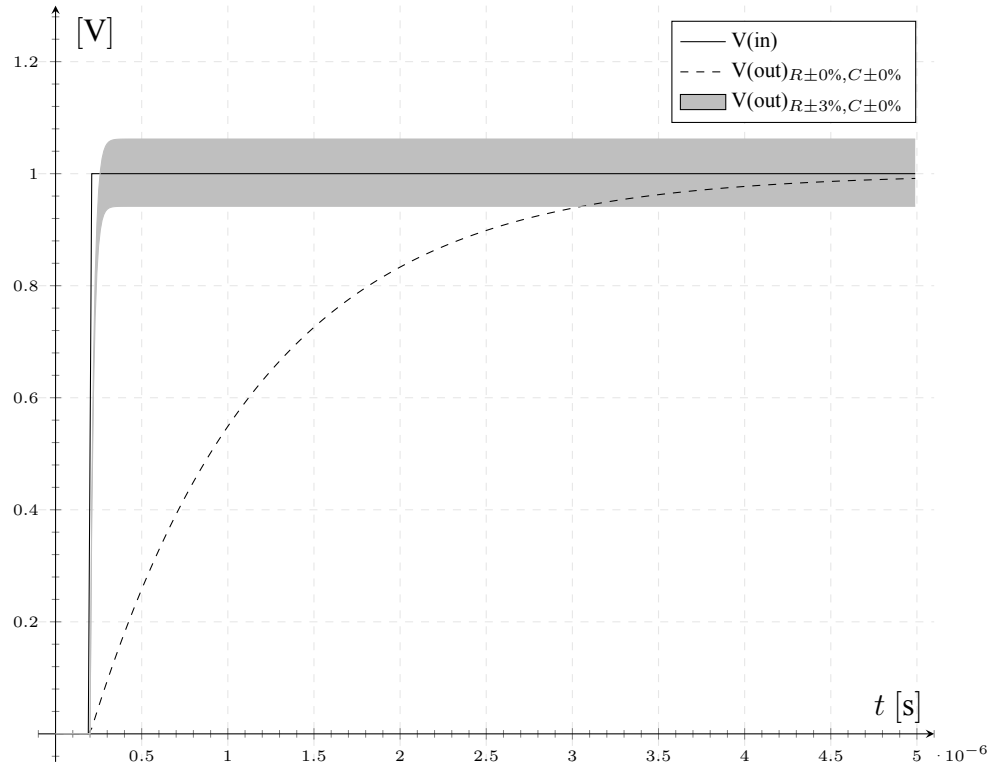


Figure 5.2: Capacitor step response using traditional representation

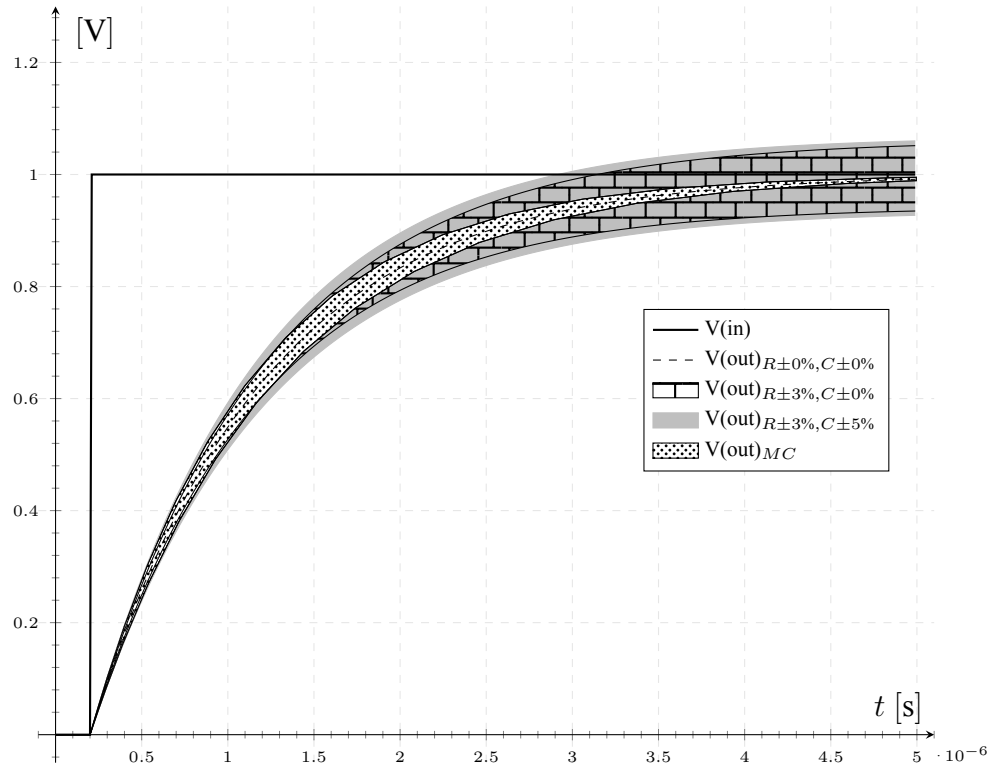


Figure 5.3: Capacitor step response using modified representation

5.1.2 Diode Model

Diode model presented in table 4.3 is tested using the parameters values listed in table 5.1. Table 5.2 shows simulation results. We can notice that results obtained from IA calculations and AA calculations are very near, that is because parameters involved in the equation is almost unrepeated.

Table 5.1: Diode model parameters values

Parameter	Value	Parameter	Value
area	$1.0 \pm 5\%$	fc	$0.5 \pm 2\%$
phi	$0.7 \pm 3\%$	is	$1.0 \times 10^{-14} \pm 2\%$

Table 5.2: Diode I_d current model results

V_d	Monte-Carlo		Interval	
	Low	High	Low	High
-1.0	-1.07×10^{-14}	-9.29×10^{-15}	-1.07×10^{-14}	-9.31×10^{-15}
-0.8	-1.07×10^{-14}	-9.29×10^{-15}	-1.07×10^{-14}	-9.31×10^{-15}
-0.6	-1.07×10^{-14}	-9.29×10^{-15}	-1.07×10^{-14}	-9.31×10^{-15}
-0.4	-1.07×10^{-14}	-9.29×10^{-15}	-1.07×10^{-14}	-9.31×10^{-15}
-0.2	-1.07×10^{-14}	-9.29×10^{-15}	-1.07×10^{-14}	-9.31×10^{-15}
0.0	0.00	0.00	0.00	0.00
0.2	2.24×10^{-11}	2.56×10^{-11}	2.24×10^{-11}	2.57×10^{-11}
0.4	5.38×10^{-08}	6.16×10^{-08}	5.37×10^{-08}	6.18×10^{-08}
0.6	1.29×10^{-04}	1.48×10^{-04}	1.29×10^{-04}	1.49×10^{-04}
0.8	3.11×10^{-01}	3.56×10^{-01}	3.10×10^{-01}	3.57×10^{-01}
1.0	$7.47 \times 10^{+02}$	$8.55 \times 10^{+02}$	$7.46 \times 10^{+02}$	$8.58 \times 10^{+02}$
1.2	$1.79 \times 10^{+06}$	$2.05 \times 10^{+06}$	$1.79 \times 10^{+06}$	$2.06 \times 10^{+06}$
V_d	Affine			
	Low	High		
-1.0	-1.07×10^{-14}	-9.29×10^{-15}		
-0.8	-1.07×10^{-14}	-9.29×10^{-15}		
-0.6	-1.07×10^{-14}	-9.29×10^{-15}		
-0.4	-1.07×10^{-14}	-9.29×10^{-15}		
-0.2	-1.07×10^{-14}	-9.29×10^{-15}		
0.0	0.00	0.00		
0.2	2.23×10^{-11}	2.57×10^{-11}		
0.4	5.36×10^{-08}	6.18×10^{-08}		
0.6	1.29×10^{-04}	1.49×10^{-04}		
0.8	3.10×10^{-01}	3.57×10^{-01}		
1.0	$7.44 \times 10^{+02}$	$8.58 \times 10^{+02}$		
1.2	$1.79 \times 10^{+06}$	$2.06 \times 10^{+06}$		

5.1.3 Simple MOSFET Model

Figure 5.4 shows results for MOSFET main current I_{ds} versus drain-source terminals voltage V_{ds} at different gate-source terminals voltage V_{gs}. The results obtained for Monte-Carlo, IA and AA simulations. Table 5.3 lists the parameters values used. The simplicity of equations makes the results are almost identical for the three types of simulations.

Table 5.3: MOSFET model parameters values

Parameter	Value
w	$1.0 \times 10^{-6} \pm 0.1\%$
l	$2.0 \times 10^{-6} \pm 0.1\%$
Kp	$100.0 \times 10^{-6} \pm 3\%$
vto	$0.70 \pm 2\%$
λ	$5.0 \times 10^{-3} \pm 0.5\%$

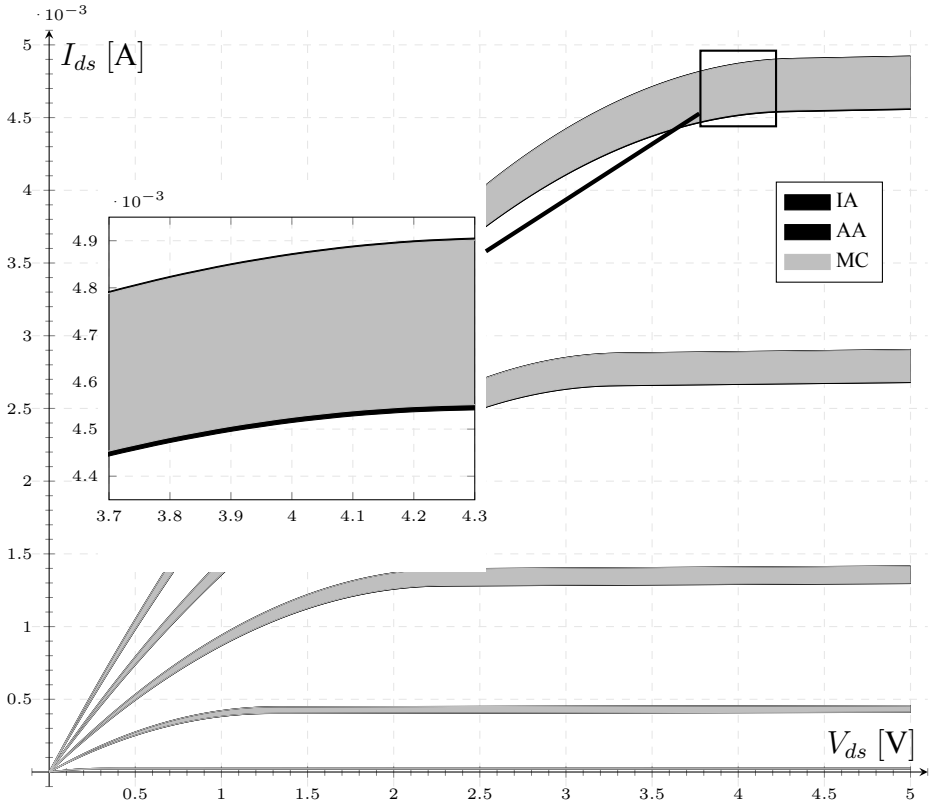


Figure 5.4: MOSFET simple model results

5.1.4 Advanced MOSFET Models

5.1.4.1 MVS MOSFET Model

The test for the model is for its terminal current I_{ds} versus the terminal voltage V_{ds} sweep. This is a basic test for MOSFET model. Tables 5.4, 5.5 and 5.6 list the parameters values used in testing, while figures 5.5, 5.6 and 5.7 show the results for each parameters set. Those figures show I_{ds}/V_{ds} curve for V_{gs} terminal voltage sweep from 0.5V to 3.0V with step of 0.5V.

Table 5.4: MVS Test 1 model parameters values

Parameter	Value	Parameter	Value
type	1	gamma	$0.1 \pm 1\%$
Tjun	300.0	mc	$0.2 \pm 3\%$
beta	1.8	Rs0	100.
W	$1 \times 10^{-4} \pm 2\%$	Rd0	100.
Lgdr	$32 \times 10^{-7} \pm 2\%$	n0	$1.68 \pm 1\%$
dLg	$9 \times 10^{-7} \pm 2.5\%$	nd	$0.1 \pm 2\%$
Cg	$2.57 \times 10^{-6} \pm 1\%$	vxo	$1.2 \times 10^{7} \pm 1\%$
alpha	$3.5 \pm 1\%$	mu	$200. \pm 3\%$
Cif	$1.38 \times 10^{-12} \pm 2\%$	Vt0	$0.4 \pm 1\%$
Cof	$1.47 \times 10^{-12} \pm 2.5\%$	delta	$0.15 \pm 2\%$
phib	$1.2 \pm 3\%$		

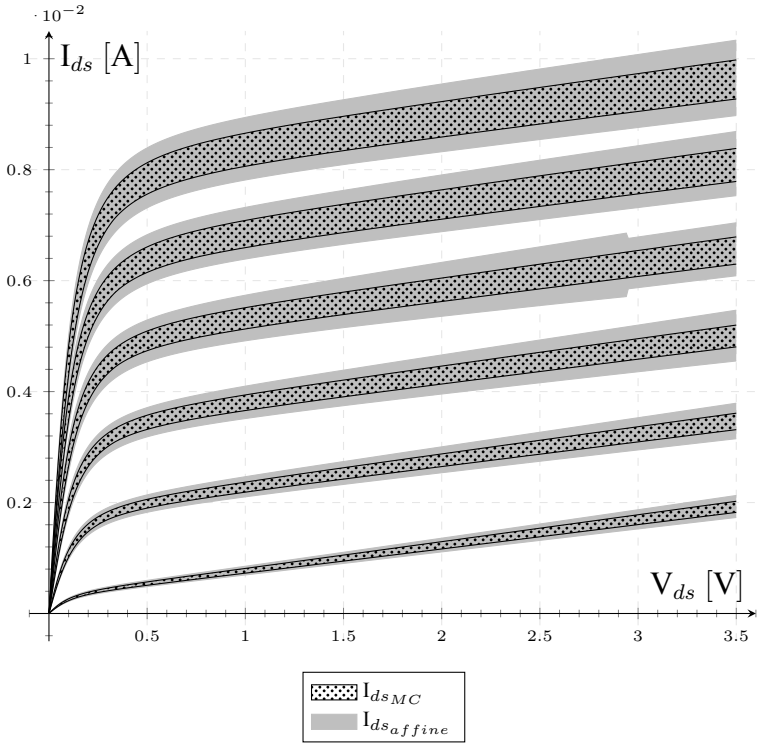


Figure 5.5: MVS Test 1 $I_{ds} results$

Table 5.5: MVS Test 2 model parameters values

Parameter	Value	Parameter	Value
type	1	gamma	0.
beta	1.8	mc	$0.2 \pm 2\%$
W	$1 \times 10^{-4} \pm 2\%$	Rs0	100.
Lgdr	$32 \times 10^{-7} \pm 3\%$	Rd0	100.
dLg	$9 \times 10^{-7} \pm 3\%$	n0	1.68
Cg	2.57×10^{-6}	nd	$0.1 \pm 3\%$
alpha	$3.5 \pm 1\%$	vxo	$1.2 \times 10^{7} \pm 1\%$
etov	$1.3 \times 10^{-3} \pm 2\%$	mu	$200. \pm 3\%$
Cif	0.	Vt0	$0.4 \pm 2\%$
Cof	0.	delta	$0.15 \pm 1\%$
phib	$1.2 \pm 1\%$		

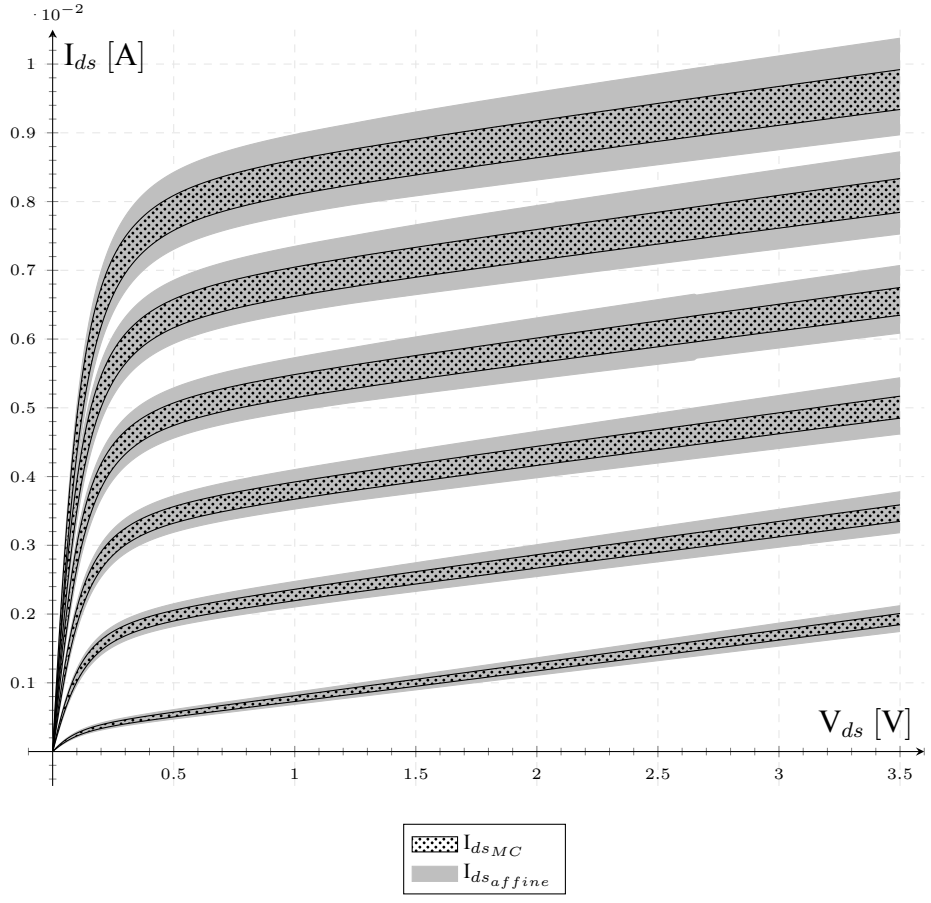


Figure 5.6: MVS Test 2 $I_{ds} results$

Table 5.6: MVS Test 3 model parameters values

Parameter	Value	Parameter	Value
type	-1	gamma	$0.1 \pm 2\%$
Tjun	300.0	mc	$0.2 \pm 3\%$
beta	1.8	Rs0	$100.$
W	$1.0 \times 10^{-4} \pm 1\%$	Rd0	$100.$
Lgdr	$32 \times 10^{-7} \pm 3\%$	n0	$1.68 \pm 1\%$
dLg	$8 \times 10^{-7} \pm 2\%$	nd	$0.1 \pm 1\%$
Cg	$2.57 \times 10^{-6} \pm 3\%$	vxo	$7542204. \pm 0.5\%$
alpha	$3.5 \pm 1\%$	mu	$165. \pm 2\%$
Cif	$1.38 \times 10^{-12} \pm 2\%$	Vt0	$0.5535 \pm 1\%$
Cof	$1.47 \times 10^{-12} \pm 2\%$	delta	$0.15 \pm 1\%$
phib	$1.2 \pm 3.5\%$		

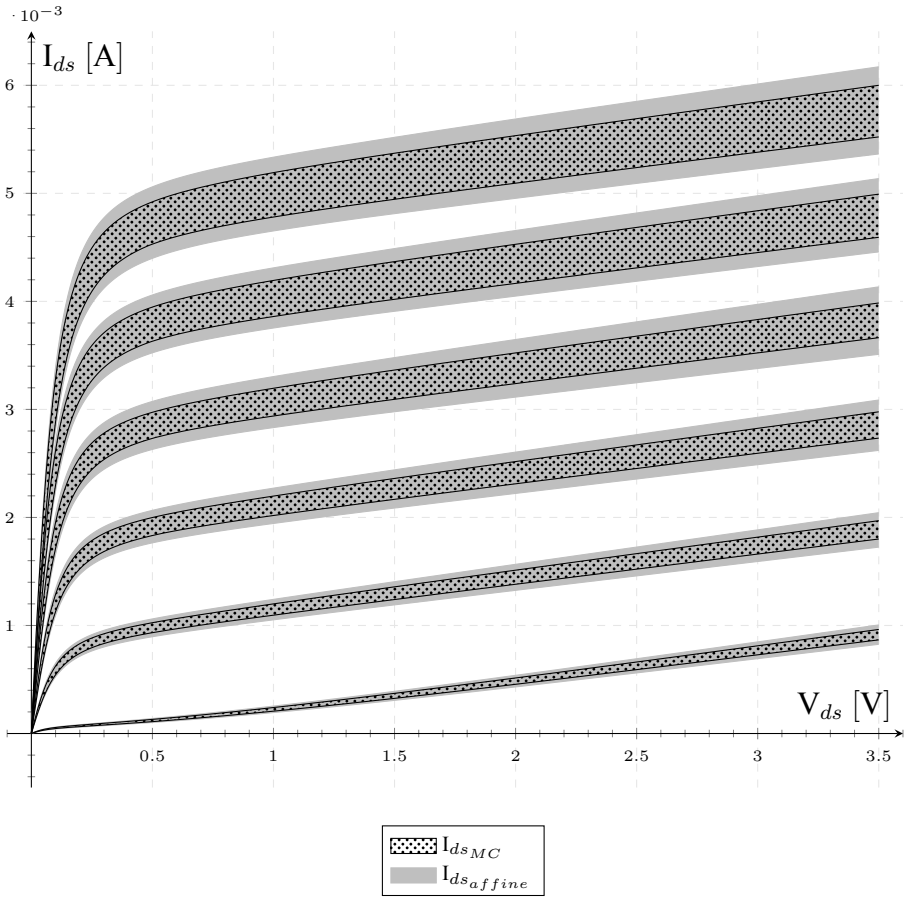


Figure 5.7: MVS Test 3 $I_{ds} results$

5.2 Passive circuits testing

Using passive circuits (combination of linear elements and sources), we can test the matrix solver. As well we can test the integration between the linear elements.

5.2.1 Potential Divider

In this test a simple 2 resistors network works as potential divider. Each resistor has percentage uncertainty in its value. Figures 5.8 and 5.9 show the circuit diagram, equivalent system of equations and the output simulations results corresponding to the expected output from hand calculations.

$$\begin{bmatrix} \mathrm{g_1} & -\mathrm{g_1} & 1 \\ -\mathrm{g_1} & (\mathrm{g_1}+\mathrm{g_2}) & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_{in} \\ v_{out} \\ i_{\mathrm{V_1}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \mathrm{V_1} \end{bmatrix}$$

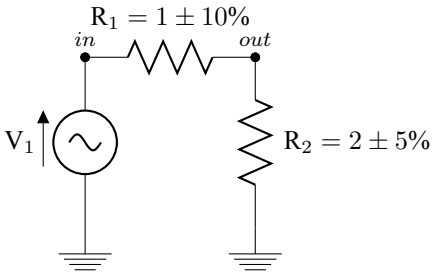


Figure 5.8: Potential divider circuit diagram and equivalent system of equations

The question here, what makes such simple network doesn't produce enough accurate results. The simulation results show that output voltage varies between 0.555 to 0.803 of the input voltage. While for hand calculations, output voltage varies between 6.333 to 0.7 of the input voltage. The reason behind this difference is the interval arithmetic limitations , dependency and the sub-distributivity, descried in section 2.1.8. Below we show how hand calculations for real numbers and interval numbers are different, and show that solving the matrix form representation of the system wide the output results even more.

Real numbers hand calculations.

$$\begin{aligned} \mathrm{V}(out)_{min} &= \frac{\mathrm{R}_{2min}}{\mathrm{R}_{1max}+\mathrm{R}_{2min}}\mathrm{V}(in)_{min} \\ &= \frac{1.9}{1.1+1.9}\mathrm{V}(in)_{min} \\ &= 0.6333\,\mathrm{V}(in)_{min} \\ \mathrm{V}(out)_{max} &= \frac{\mathrm{R}_{2max}}{\mathrm{R}_{1min}+\mathrm{R}_{2max}}\mathrm{V}(in)_{max} \\ &= \frac{2.1}{0.9+2.1}\mathrm{V}(in)_{max} \\ &= 0.7000\,\mathrm{V}(in)_{max} \end{aligned}$$

Interval hand calculations. Here we note the dependency problem where R_2 appears in nominator and the dominator.

$$\begin{aligned} V(out) &= \frac{R_2}{R_1 + R_2} V(in) \\ &= \frac{[1.9, 2.1]}{[0.9, 1.1] + [1.9, 2.1]} V(in) \\ &= \frac{[1.9, 2.1]}{[2.8, 3.2]} V(in) \\ &= [0.5937, 0.7501] V(in) \end{aligned}$$

Matrix form representation. The system matrix form coming from the MNA is shown in figure 5.8. We know that current flow through R_1 can be calculated as $(g_1(v_{in} - v_{out}))$. For the matrix form the current is actually calculated as $(g_1 v_{in} - g_1 v_{out})$, which in the interval form may produce wider interval results, as sub-distributivity relation (2.27) $g_1(v_{in} - v_{out}) \subseteq g_1 v_{in} - g_1 v_{out}$. This illustrates why we get wider results even compared to interval calculations by hand.

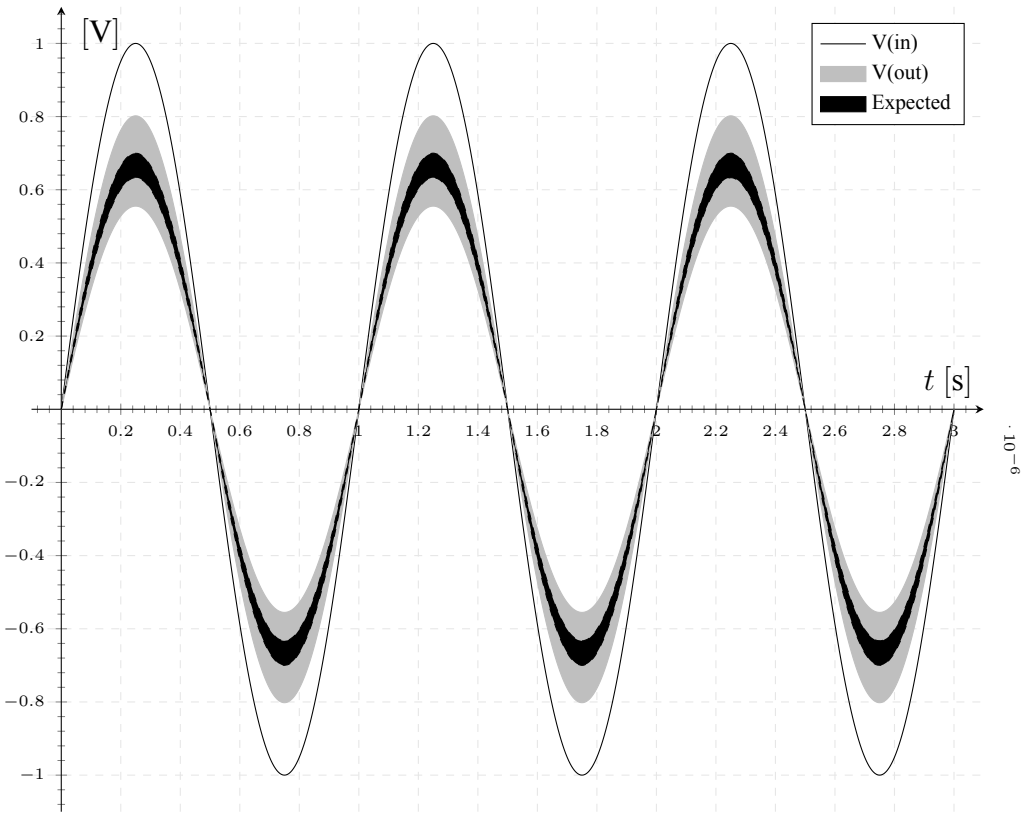


Figure 5.9: Resistor potential divider simulations results

5.2.2 R-2R Resistors Ladder

The R-2R resistors ladder network is an example for passive circuits, this network is used as passive digital to analog converter. The resistor network causes the digital bits to be

weighted in their contribution to the output voltage[**?**]. In this example we use an 8-bits converter as shown in figure 5.10. Digital word is input to the bits a_0 to a_7, bits are switched between logic 0 and logic 1 voltages. For this example Logic 0 is zero voltage and logic 1 is one volt.
A Monte-Carlo simulation of 1000 runs is hold for this example, where R takes the nominal value of 1Ω with a percentage uncertainty in the resistor value by 2%. So the R varies between $[0.98, 1.02]$ and $2R$ varies between $[1.96, 2.04]$. An interval simulation for this example is hold as well, one using the interval Gauss-Seidel and the other using interval Gauss-Seidel method for generalized solution sets. Results for some digital words and simulations average elapsed time are in table 5.7. In this example modal algorithm gives narrower results than interval algorithm, as well it converges faster. Interval simulation is about 2 times faster than Monte-Carlo simulation and modal is about 19 times faster although Monte-Carlo has more accurate results.

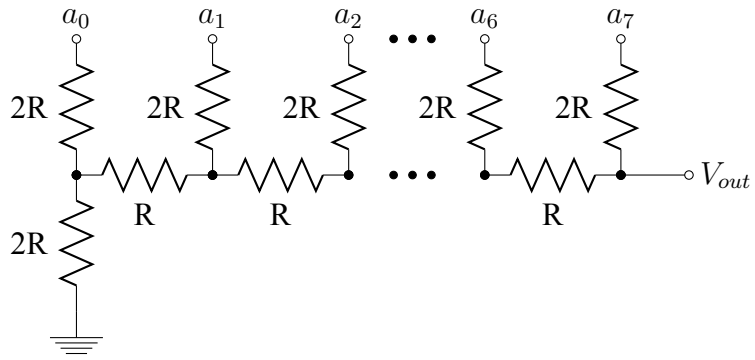


Figure 5.10: 8-Bits digital to analog converter (R-2R resistors network)

Table 5.7: 8-Bits digital to analog converter (R-2R resistors network) simulation results

Word	Monte-Carlo		Interval	
	Low	High	Low	High
00000000	0.00	0.00	-6.52×10^{-11}	6.52×10^{-11}
11111111	9.96×10^{-01}	9.96×10^{-01}	8.25×10^{-01}	1.18×10^{00}
00000001	3.86×10^{-03}	3.95×10^{-03}	1.30×10^{-03}	6.73×10^{-03}
01111111	4.91×10^{-01}	5.00×10^{-01}	3.73×10^{-01}	6.29×10^{-01}
10000000	4.97×10^{-01}	5.05×10^{-01}	4.52×10^{-01}	5.52×10^{-01}
Time (ms)	1530		840	
Word	Modal			
	Low	High		
00000000	-4.12×10^{-11}	3.98×10^{-11}		
11111111	9.09×10^{-01}	1.06×10^{00}		
00000001	1.81×10^{-03}	5.18×10^{-03}		
01111111	4.17×10^{-01}	5.57×10^{-01}		
10000000	4.92×10^{-01}	5.05×10^{-01}		
Time (ms)	80			

5.2.3 Transmission Line R-C Model

An example for passive circuits the transmission line (wire) R-C model, where wire may be represented by successive RC sections as in figure 5.11. In this test we do the simulations for n sections transmission line, $n = \{1, 2, 5\}$. With $R_{load} = 50\Omega$, $C = 5/n$pF $\pm$ 2.5% and $R = 20/n\Omega \pm 5\%$. Simulations time are recorded in table 5.8, where we use two settings for simulation time step, the first is 1^{-12}s and other is 1^{-13}s. The interval simulator fails to converge to solution using four sections or more. This is noticed from results in figures 5.12, 5.13 and 5.14.

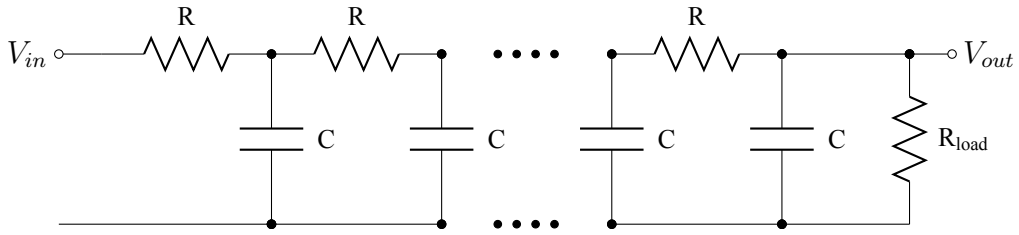


Figure 5.11: Transmission line R-C model example

Table 5.8: Transmission line simulation time

Simulation	1-Section		2-Sections		3-Sections	
	1^{-13}s	1^{-12}s	1^{-13}s	1^{-12}s	1^{-13}s	1^{-12}s
Monte-Carlo time (s)	983	101	834	85	840	85
Interval time (s)	82.3	9.3	173.5	21.19	314.16	29.75
Ratio	11.94	10.86	4.81	4.01	2.67	2.86

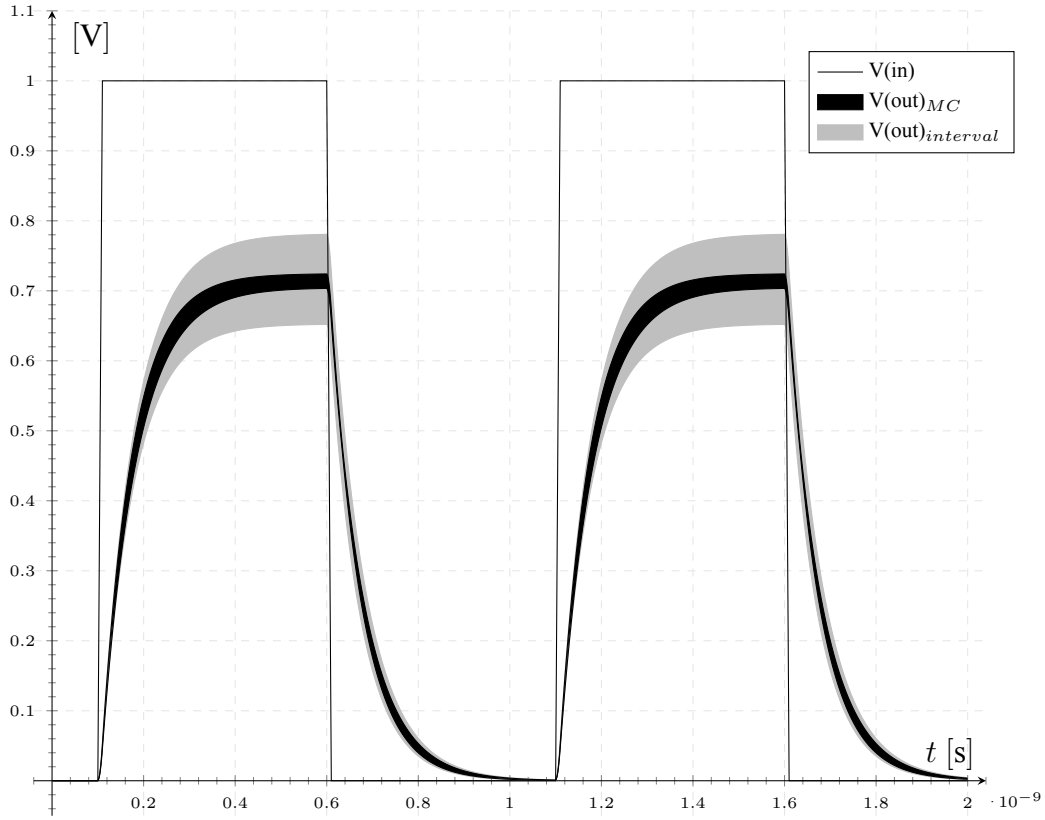


Figure 5.12: Transmission line one R-C section simulations results

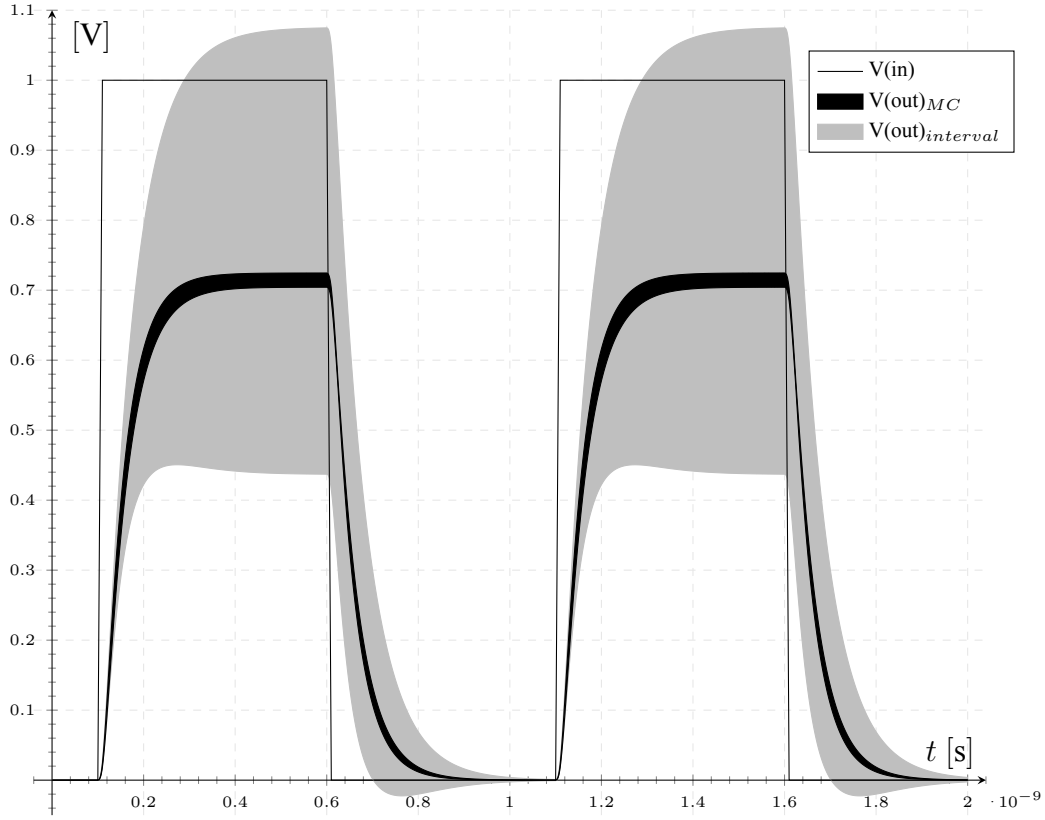


Figure 5.13: Transmission line two R-C section simulations results

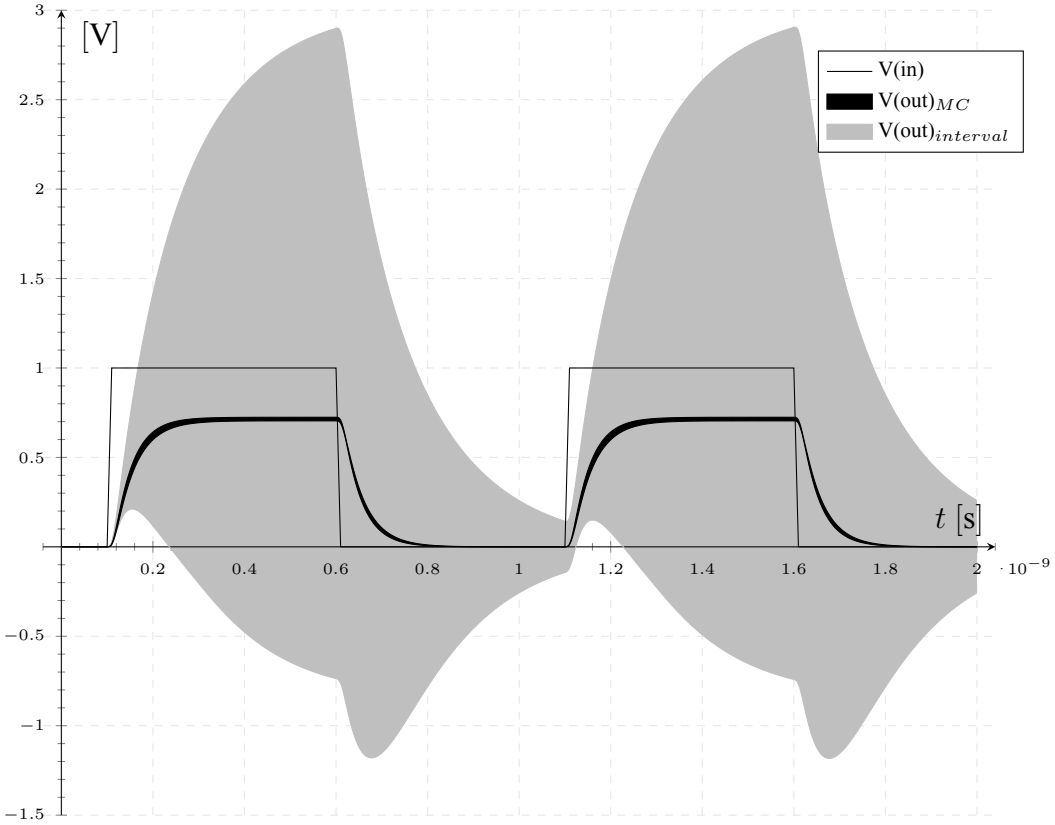


Figure 5.14: Transmission line three R-C section simulations results

Chapter 6

Conclusion

6.1 Conclusion

In this thesis we introduce a new interval based simulation flow to enhance the time required for simulation circuits variations. This flow may replace or coexist with the traditional Monte-Carlo simulations. To achieve this flow we build a simulator platform that uses interval arithmetic calculations. We provide models library for the simulator that contains sources, linear elements, diode, and MOSFET models. We do a hybrid usage of interval arithmetic and affine arithmetic to enhance the non-linear models accuracy.
Unit testing for the models show good accuracy for the models compared to the Monte-Carlo. Small passive circuits are tested showing better simulation time compared to Monte-Carlo simulation. Circuits with active elements fail to converge.
Some of this work results are presented in the 16th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics - SCAN2014 [9]. A paper then has been submitted for the post-conference proceedings to be published on Springer, Lecture Notes of Computer Science [10].

6.2 Future work

To enhance and complete this work, we suggest the following items:

- Enhance the algorithm used for solving the linear interval matrix.
- Fix implementation or use another algorithm for the non-linear interval equations.
- Complete the simulator front and back ends.
- Enhance BSIM4 results.
- Enable more analysis types other than the steady state ”DC” and time domain ”transiant” analysis.
- Test usage affine arithmetic in solving the system matrix.
- In this work we use definite time steps in simulation, a point to research about, is using interval time steps.

Appendix A

Interval Algorithms

In this appendix we provide the interval algorithms that we use for the interval simulator solver.

A.1 Interval Gauss–Seidel Method

Gauss–Seidel is one of the commonly used methods to solve system of interval linear equations. The interval Gauss–Seidel method is the interval version of the famous Gauss-Seidel method for solving system of algebraic linear equations. The following algorithm define the method as descried by Hansen [27].
We want to bound the solution set for (A.1); let **A** be an interval matrix, **B** is an interval vector.

$$\mathbf{AX} = \mathbf{B} \tag{A.1}$$

We may use a preconditioner matrix Y, typically $Y = (m(\mathbf{A}))^{-1}$, then we get:

$$\mathbf{GX} = \mathbf{C}, \quad \text{where} \quad \mathbf{G} = Y\mathbf{A} \quad \text{and} \quad \mathbf{C} = Y\mathbf{B} \tag{A.2}$$

Algorithm A.1: Interval Gauss–Seidel algorithm

Data: $\mathbf{AX} = \mathbf{B}$
$Y = (m(\mathbf{A}))^{-1}$;
$\mathbf{G} = Y.\mathbf{A}$;
$\mathbf{C} = Y.\mathbf{B}$;
$k = 0$;
$d = \infty$;
while *($d \geq \epsilon$) and ($k <$ Iterations Limit)* **do**
  for $i := 1$ *to* n **do**
    $X_i^{k+1} = (G_{ii}^{-1}) \left(C_i - \sum_{j=1}^{i-1} G_{ij} X_j^{k+1} - \sum_{j=i+1}^{n} G_{ij} X_j^k \right)$;
    $X_i^{k+1} = X_i^{k+1} \cap X_i^k$;
  end
  $k = k + 1$;
  $d =$ distance between $\mathbf{X}^{k+1}$ and $\mathbf{X}^{k+1}$;
end

A.2 Interval Newton Method

The interval Newton method for solving a nonlinear equation has many similarities with the traditional Newton method. Starting from an initial interval $X^{(0)}$, the univariate interval Newton method finds solution(s), if exist(s), of the equation,

$$f(x) = 0$$

where f is a continuously differentiable real-valued function of a real variable x. The univariate interval Newton operator can be deduced, using the mean value theorem, to be:

$$N(X) = y - \frac{f(y)}{F'(X)}, \tag{A.3}$$

where X is the interval in which the method searches for a solution, y is any real number satisfying $y \in X$, and $F'(X)$ is an inclusion monotonic interval extension of $f'(X)$. Usually, y is taken to be the mid of the interval. Hence, the univariate interval Newton operator becomes:

$$N(X) = m(X) - \frac{f(m(X))}{F'(X)} \tag{A.4}$$

finally, the interval Newton algorithm uses the following equation:

$$X^{(k+1)} = X^{(k)} \cap N\left(X^k\right) \quad , k = 0, 1, 2, \ldots \tag{A.5}$$

to update the interval X at each step. The following theorem discusses the existence and convergence of the algorithm.

Theorem A.1. If an interval $X^{(0)}$ contains zero x of $f(x)$, then so does $X^{(k)}$ for all $k = 0, 1, 2, \ldots$, defined by (A.5). Furthermore, the intervals $X^{(k)}$ form a nested sequence converging to x if $0 \notin F'(X)$.

Lemma A.1. Given a real rational function f of a single real variable x with rational extensions F , F' of f, f', respectively, such that f has a simple zero y in an interval $[x_1, x_2]$ for which $F([x_1, x_2])$ is defined and $F'([x_1, x_2])$ is defined and does not contain zero, there is an interval $x_0 \subseteq [x_1, x_2]$ containing y and a positive real number C such that

$$w\left(X^{k+1}\right) \leq C\left(w\left(X^{k+1}\right)\right)^2 \tag{A.6}$$

A.2.1 Multivariate Interval Newton Method

To solve the system of interval non-linear equations, described by (A.7), where $\mathbf{X}$ is a vector of intervals.

$$F(\mathbf{X}) = 0 \tag{A.7}$$

$$N(\mathbf{X}) = \mathbf{y} - \left(F'(\mathbf{X})\right)^{-1} f(\mathbf{y}) \tag{A.8}$$

Where $\mathbf{y}$ is a real contained in the interval vector $\mathbf{X}$ and $F'(\mathbf{X})$ is an element-wise interval extension of the Jacobian matrix over some box $\mathbf{X}$. To obtain the multivariate interval Newton operator, we may use the following equation:

$$N(\mathbf{X}) = m(\mathbf{m}) - \left(F'(\mathbf{X})\right)^{-1} f(m(\mathbf{m})) \tag{A.9}$$

which is analogue to (A.4). But in this case, we need to find inverse of an interval matrix which is not an easy operation. Instead of doing such complex operation, the multivariate Newton operator is redefined as:

$$N(\mathbf{X}) = \mathbf{y} + \mathbf{V} \tag{A.10}$$

where **V** bounds the solution set to:

$$\left(F'(\mathbf{X})\right)\mathbf{V} = -f(\mathbf{y}) \tag{A.11}$$

This equation can be solved using Gauss–Seidel Method descried in section A.1.

Appendix B

MVS MOSFET Model Interval Code

Table B.1: C++ code for MVS MOSFET model

```
////////////////////////////////////////////////////////////////////
//Copyright @ 2013 Massachusetts Institute of Technology (MIT)

//The terms under which the software and associated documentation (the
    Software) is provided are as the following:

//The Software is provided "as is", without warranty of any kind,
   express or implied, including but not limited to the warranties of
   merchantability, fitness for a particular purpose and
   noninfringement. In no event shall the authors or copyright holders
    be liable for any claim, damages or other liability, whether in an
    action of contract, tort or otherwise, arising from, out of or in
   connection with the Software or the use or other dealings in the
   Software.

//MIT grants, free of charge, to any users the right to modify, copy,
   and redistribute the Software, both within the user's organization
   and externally, subject to the following restrictions:

//1. The users agree not to charge for the MIT code itself but may
   charge for additions, extensions, or support.

//2. In any product based on the Software, the users agree to
   acknowledge the MIT VS Model Research Group that developed the
   software. This acknowledgment shall appear in the product
   documentation.

//3. The users agree to obey all U.S. Government restrictions
   governing redistribution or export of the software.

//4. The users agree to reproduce any copyright notice which appears
   on the software on any copy or modification of such made available
   to others.

//Agreed to by
//Dimitri A. Antoniadis, MIT
//May 27 2013
////////////////////////////////////////////////////////////////////

#ifndef MVS_H
#define MVS_H
```

```
#include "aa.h"
#include "aa_interval.h"
#include "aa_aaf.h"

#define  SMALL_VALUE  (1e-10)
#define  LARGE_VALUE  (40)
#define  KB           1.380648813E-23
#define  P_Q          1.60217656535E-19
#define  affine       AAF

class MVS
{
public:
    MVS();
    void init_model_parameters();
    void load_model();
    void eval(double V[]);
    double vt(double T) {return(KB * T / P_Q);}
    void print(void);
    affine param(double, double per=0.0);
    affine Id(void) {return(Idi_si);}

private:
    typedef enum {gnd=0, d, g, s, b, di, si} NODE_NAME;
    // Model paramters
    int       type      ;        //  type of transistor. nFET type=1; pFET type=-1
    int       CTM_select;        //  If CTM_select = 1, then classic DD-NVSAT model is used
    double    version   ;        //  MVS model version = 1.0.1
    double    Tjun      ;        //  Junction temperature [K]
    double    beta      ;        //  Saturation factor. Typ. nFET=1.8, pFET=1.6
    affine    W         ;        //  Transistor width [cm]
    affine    Lgdr      ;        //  Physical gate length [cm]. // This is the designed gate length for litho printing.
    affine    dLg       ;        //  Overlap length including both source and drain sides [cm]
    affine    Cg        ;        //  Gate-to-channel areal capacitance at the virtual source [F/cm^2]
    affine    etov      ;        //  Equivalent thickness of dielectric at S/D-G overlap [cm]
    affine    delta     ;        //  Drain-induced-barrier-lowering ( DIBL) [V/V]
    affine    n0        ;        //  Subthreshold swing factor [unit-less] {typically between 1.0 and 2.0}
    affine    Rs0       ;        //  Access resistance on s-terminal [ Ohms-micron]
    affine    Rd0       ;        //  Access resistance on d-terminal [ Ohms-micron]
    affine    Cif       ;        //  Inner fringing S or D capacitance [F/cm]
    affine    Cof       ;        //  Outer fringing S or D capacitance [F/cm]
    affine    vxo       ;        //  Virtual source injection velocity [cm/s]
    affine    mu        ;        //  Low-field mobility [cm^2/V.s]
    affine    phib      ;        //  ~abs(2*phif)>0 [V]
```

```
    affine    gamma     ;        //  Body factor  [sqrt(V)]
    affine    Vt0       ;        //  Strong inversion threshold
    voltage [V]
    affine    alpha     ;        //  Empirical parameter for threshold
     voltage shift between strong and weak inversion.
    affine    mc        ;        //  Choose an appropriate value
    between 0.01 to 10
    affine    CC        ;        //  Fitting parameter to adjust Vg-
    dependent inner fringe capacitances(Not used in this version)
    affine    nd        ;        //  Punch-through factor [1/V]
    affine    Idi_si,
              Id_di,
              Isi_s,
              Qsi_b,
              Qdi_b,
              Qg_b;
};


#endif
```

Table B.2: C++ definitions for MVS MOSFET model

```
#include <iostream>
#include "mvs.h"

#define abs(x) fabs(x)
#define pow(x,a) affine_pow(x,a)

inline affine    affine_pow(affine    x, double a)
{
  affine tmp;
  // For zero width intervals
  if (x.getcenter() == 0.0 && x.rad() == 0.0)
  {
    tmp=affine(0.0);
  }
  else
  {
    tmp = exp((a*log(x)));
  }
  return (tmp);
}

inline affine    affine_pow(affine    x, affine    a)
{
  return (aaf_pow(x,a));
}

affine MVS::param(double c, double per)
{
  affine tmp = affine(0.0);
  return(tmp);
}

affine MVS::param(double c, double per)
{
  affine temp(c);
```

```
36   if (per != 0.0)
37   {
38     temp = affine((c*(1-per/100.)),(c*(1+per/100.)));
39   }
40   return(temp);
41 }
42
43 MVS::MVS()
44 {
45   init_model_parameters();
46   load_model();
47 }
48
49 void MVS::init_model_parameters()
50 {
51   // Default values
52   type               = 1;              // from  [-1 : 1] exclude 0;
53   CTM_select         = 1;              // from  [1 : inf);   (For
       CTM_select other than 1,blended DD-NVSAT and ballistic charge
       transport model is used)
54   version            = 1.01;
55   Tjun               = 298.;           // from  [173:inf);
56   beta               = 1.7;            // from  (0:inf);
57   W                  = affine(1e-4);   // from  (0:inf);
58   Lgdr               = affine(80e-7);  // from  (0:inf);
59   dLg                = affine(10.5e-7); // from  (0:inf);
60   Cg                 = affine(2.2e-6); // from  (0:inf);
61   etov               = affine(1.3e-3); // from  (0:inf);
62   delta              = affine(0.10);   // from  [0:inf);
63   n0                 = affine(1.5);    // from  [0:inf);
64   Rs0                = affine(100);    // from  (0:inf);
65   Rd0                = affine(100);    // from  (0:inf);
66   Cif                = affine(1e-12);  // from  [0:inf);
67   Cof                = affine(2e-13);  // from  [0:inf);
68   vxo                = affine(0.765e7); // from  (0:inf);
69   mu                 = affine(200);    // from  (0:inf);
70   phib               = affine(1.2);    //  ~abs(2*phif)>0 [V]
71   gamma              = affine(0.0);    // from  [0:inf);
72   Vt0                = affine(0.486);
73   alpha              = affine(3.5);
74   mc                 = affine(0.2);    // from  [0.01 : 10]; (For,
       values outside of this range,convergence or accuracy of results is
       not guaranteed)
75   CC                 = affine(0.);     // from  [0:inf);
76   nd                 = affine(0.);     // from  [0:inf);
77 }
78
79 void MVS::eval(affine V[])
80 {
81   int dir;
82   double MvsDtmp01;
83   affine Vds, Vgs, Vgsraw, Vgd, Vgdraw, Vbs, Vdsi, Vgsi, Vgdi, Vbsi;
84   affine Rs, Rd;
85   affine Leff, me, S, phit;
86   affine n, nphit, aphit, Vtpcorr, eVgpre, FFpre, ab, Vcorr, Vgscorr,
       Vbscorr, Vt0bs, Vt0bs0, Vtp, Vtp0;
87   affine eVg, FF, eVg0, FF0, Qref, eta, eta0;
88   affine Qinv, Qinv_corr, vx0, Vdsats, Vdsat,Vdratio, Vdbeta,
```

```
  Vdbetabeta, Fsat, Id ;
affine Vgt, psis, Vgta, Vdsatq, Fsatq, x, den;
affine qsc, qdc, qi, kq, kq2, kq4, tol, qsb, qdb, qs, qd, Qs, Qd;
affine Qb, etai, Qinvi, dQinv, dibl_corr;
affine Qinvs, Qinvd, Qsov, Qdov, Vt0x, Vt0y, Fs_arg, Fs, Fd_arg, Fd,
   FFx, FFy, Qsif, Qdif, Qg, a, Cofs, Cofd;
affine MvsTtmp01;

// analog
{
  //Voltage definitions
  Vgsraw      = type * ( V[g] - V[si] );
  Vgdraw      = type * ( V[g] - V[di] );
  if (Vgsraw >= Vgdraw)
  {
    Vds   = type * ( V[d] - V[s] );
    Vgs   = type * ( V[g] - V[s] );
    Vbs   = type * ( V[b] - V[s] );
    Vdsi    = type * ( V[di] - V[si] );
    Vgsi    = Vgsraw;
    Vbsi    = type * ( V[b] - V[si] );
    dir   = 1;
  }
  else
  {
    Vds   = type * ( V[s] - V[d] );
    Vgs   = type * ( V[g] - V[d] );
    Vbs   = type * ( V[b] - V[d] );
    Vdsi    = type * ( V[si] - V[di] );
    Vgsi    = Vgdraw;
    Vbsi    = type * ( V[b] - V[di] );
    dir   = -1;
  }

//Parasitic element definition
Rs      = 1e-4/ W * Rs0;
  //  s-terminal resistance [ohms]
Rd      = Rs;
  //  d-terminal resistance [ohms] For symmetric source and drain Rd
  = Rs.
//Rd      = 1e-4/ W * Rd0;
    //  d-terminal resistance [ohms] {Uncomment for asymmetric source
   and drain resistance.}
Cofs      =   ( 0.345e-12/ etov ) * dLg/ 2.0 + Cof;         //  s-
  terminal outer fringing cap [F/cm]
Cofd      =   ( 0.345e-12/ etov ) * dLg/ 2.0 + Cof;         //  d-
  terminal outer fringing cap [F/cm]
Leff      =   Lgdr - dLg;
    //  Effective channel length [cm]. After subtracting overlap
  lengths on s and d side

phit            = vt(Tjun);
        //  Thermal voltage, kT/q [V]
me      = (9.1e-31) * mc;
  //  Carrier mass [Kg]
n     = n0 + nd * Vds;                                          //
    Total subthreshold swing factor taking punchthrough into account
  [unit-less]
```

```
132  nphit     =   n * phit;
       //  Product of n and phit [used as one variable]
133  aphit     = alpha * phit;
      //  Product of alpha and phit [used as one variable]
134
135
136  //Correct Vgsi and Vbsi
137  //Vcorr is computed using external Vbs and Vgs but internal Vdsi,
      Qinv and Qinv_corr are computed with uncorrected Vgs, Vbs and
      corrected Vgs, Vbs respectively.
138  Vtpcorr     = Vt0 + gamma * (sqrt(abs(phib - Vbs))- sqrt(phib))-
      Vdsi * delta;//  Calculated from extrinsic Vbs
139  eVgpre      =   exp(( Vgs - Vtpcorr )/ ( aphit * 1.5 ));      //
      Calculated from extrinsic Vgs
140  FFpre       =   1.0/ ( 1.0 + eVgpre );            //  Only used to
      compute the correction factor
141  ab      = 2 * ( 1 - 0.99 * FFpre ) * phit;
142  Vcorr     = ( 1.0 + 2.0 * delta ) * ( ab/ 2.0 ) * ( exp( -Vdsi/ ab )
      );  //  Correction to intrinsic Vgs
143  Vgscorr     = Vgsi + Vcorr;               //  Intrinsic Vgs
      corrected (to be used for charge and current computation)
144  Vbscorr     = Vbsi + Vcorr;               //  Intrinsic Vgs
      corrected (to be used for charge and current computation)
145  Vt0bs     = Vt0 + gamma * (sqrt( abs( phib - Vbscorr)) - sqrt( phib
      ));   //  Computed from corrected intrinsic Vbs
146  Vt0bs0      = Vt0 + gamma * (sqrt( abs( phib - Vbsi)) - sqrt( phib )
      );  //  Computed from uncorrected intrinsic Vbs
147  Vtp     = Vt0bs - Vdsi * delta - 0.5 * aphit;         //  Computed
      from corrected intrinsic Vbs and intrinsic Vds
148  Vtp0      = Vt0bs0 - Vdsi * delta - 0.5 * aphit;        //  Computed
       from uncorrected intrinsic Vbs and intrinsic Vds
149  eVg     = exp(( Vgscorr - Vtp )/ ( aphit ));        //  Compute eVg
      factor from corrected intrinsic Vgs
150  FF      = 1.0/ ( 1.0 + eVg );
151  eVg0      = exp(( Vgsi - Vtp0 )/ ( aphit ));        //  Compute eVg
      factor from uncorrected intrinsic Vgs
152  FF0     = 1.0/ ( 1.0 + eVg0 );
153  Qref      = Cg * nphit;
154  eta     = ( Vgscorr - ( Vt0bs - Vdsi * delta - FF * aphit ))/ (
      nphit );  //  Compute eta factor from corrected intrinsic Vgs and
      intrinsic Vds
155  eta0      = ( Vgsi - ( Vt0bs0 - Vdsi * delta - FFpre * aphit ))/ (
      nphit ); //  Compute eta0 factor from uncorrected intrinsic Vgs and
       internal Vds.
156  //  Using FF instead of FF0 in eta0 gives smoother capacitances.
157
158  //Charge at VS in saturation (Qinv)
159  if (eta  <= LARGE_VALUE)
160  {
161    Qinv_corr   =   Qref * log( 1.0 + exp(eta) );
162  }
163  else
164  {
165    Qinv_corr = Qref * eta;
166  }
167  if (eta0 <= LARGE_VALUE)
168  {
169    Qinv    = Qref * log( 1.0 + exp(eta0) );         //  Compute
```

```
    charge w/ uncorrected intrinsic Vgs for use later on in charge
    partitioning
170 }
171 else
172 {
173   Qinv    = Qref * eta0;
174 }
175
176
177 //Transport equations
178 vx0        = vxo;
179 Vdsats     = vx0 * Leff/ mu;
180 Vdsat      = Vdsats * ( 1.0 - FF ) + phit * FF;        //
    Saturation drain voltage for current
181 Vdratio    = abs( Vdsi/ Vdsat);
182 Vdbeta     = pow( Vdratio, beta);
183 MvsTtmp01   = (1.0 + Vdbeta);
184 MvsDtmp01   = (1.0/ beta);
185 Vdbetabeta =   pow(MvsTtmp01,MvsDtmp01);
186 Fsat       = Vdratio / Vdbetabeta;           //  Transition function
    from linear to saturation.
187 //  Fsat = 1 when Vds>>Vdsat; Fsat= Vds when Vds<<Vdsat
188
189 //Total drain current
190 Id      = Qinv_corr * vx0 * Fsat * W;
191
192 //Calculation of intrinsic charge partitioning factors (qs and qd)
193 Vgt     = Qinv/ Cg;               //  Use charge computed from
    uncorrected intrinsic Vgs
194
195 // Approximate solution for psis is weak inversion
196 if (gamma == 0)
197 {
198   a     =   1.0;
199   if (eta0 <= LARGE_VALUE)
200   {
201     psis    =   phib + phit * ( 1.0 + log( log( 1.0 + SMALL_VALUE +
    exp( eta0 ))));
202   }
203   else
204   {
205     psis    = phib + phit * ( 1.0 + log( eta0 ));
206   }
207 }
208 else
209 {
210   if (eta0 <= LARGE_VALUE)
211   {
212     psis  = phib + ( 1.0 - gamma )/ ( 1.0 + gamma ) * phit * ( 1.0 +
    log( log( 1.0 + SMALL_VALUE + exp( eta0 ))));
213   }
214   else
215   {
216     psis  = phib + ( 1.0 - gamma )/ ( 1.0 + gamma ) * phit * ( 1.0 +
    log( eta0 ));
217   }
218   a   = 1.0 + gamma/ ( 2.0 * sqrt( abs( psis - ( Vbsi ))));
219 }
```

```
220  Vgta      = Vgt/ a;                //  Vdsat in strong inversion
221  Vdsatq      = sqrt( FF0 * aphit * aphit + Vgta * Vgta);     //
      Vdsat approx. to extend to weak inversion;
222  //  The multiplier of phit has strong effect on Cgd discontinuity at
       Vd=0.
223
224  // Modified Fsat for calculation of charge partitioning
225  //DD—NVSAT charge
226  Fsatq     = abs( Vdsi/ Vdsatq )/ ( pow( 1.0 + pow( abs( Vdsi/ Vdsatq
       ), beta ), 1.0/ beta ));
227  x     = 1.0 — Fsatq;
228  MvsTtmp01 = x + 1;
229  den     = 15 * sqr(MvsTtmp01);
230  qsc       = Qinv *(6 + 12 * x + 8 * sqr(x) + 4 * power(x,3))/ den;
231  qdc     = Qinv *(4 + 8 * x + 12 * sqr(x) + 6 * power(x,3))/ den;
232  qi      = qsc + qdc;             //  Charge in the channel
233
234
235  //QB charge
236  kq      =   0.0;
237  tol     =   ( SMALL_VALUE * vxo/ 100.0 ) * ( SMALL_VALUE * vxo/
      100.0 ) * me/ ( 2 * P_Q );
238  if (tol > Vdsi)/*(Vdsi <= tol)*/
239  {
240    kq2   = ( 2.0 * P_Q/ me * Vdsi )/ ( sqr(vx0) ) * 10000.0;
241    kq4   = sqr(kq2);
242    qsb   = Qinv * ( 0.5 — kq2/ 24.0 + kq4/ 80.0 );
243    qdb   = Qinv * ( 0.5 — 0.125 * kq2 + kq4/ 16.0 );
244  }
245  else
246  {
247    kq    = sqrt( 2.0 * P_Q/ me * Vdsi )/ vx0 * 100.0;
248    kq2   = sqr(kq);
249    qsb   = Qinv * ( asinh( kq )/ kq — ( sqrt( kq2 + 1.0 ) — 1.0 )/
      kq2);
250    qdb   = Qinv * (( sqrt( kq2 + 1.0 )— 1.0 )/ kq2);
251  }
252
253
254  // Flag for classic or ballistic charge partitioning:
255  if (CTM_select == 1)                    //  Ballistic blended with
      classic DD—NVSAT
256  {
257    qs    = qsc;               //  Calculation of "ballistic" channel
       charge partitioning factors, qsb and qdb.
258    qd    = qdc;               //  Here it is assumed that the
      potential increases parabolically from the
259  }                        //  virtual source point, where Qinv_corr is
       known to Vds—dvd at the drain.
260  else                          //  Hence carrier velocity increases
      linearly by kq (below) depending on the
261  {
262    qs    = qsc * ( 1 — Fsatq * Fsatq ) + qsb * Fsatq * Fsatq;    //
      efecive ballistic mass of the carriers.
263    qd    = qdc * ( 1 — Fsatq * Fsatq ) + qdb * Fsatq * Fsatq;
264  }
265
266
```

```
267  //Body charge based on approximate surface potential (psis)
      calculation with delta=0 using psis=phib in Qb gives continuous Cgs
      , Cgd, Cdd in SI, while Cdd is smooth anyway.
268  Qb      = -type * W * Leff * ( Cg * gamma * sqrt( abs( psis - Vbsi )
      ) + ( a - 1.0 )/ ( 1.0 * a ) * Qinv * ( 1.0 - qi ));
269
270
271  //DIBL effect on drain charge calculation.
272  //Calculate dQinv at virtual source due to DIBL only. Then:Correct
      the qd factor to reflect this channel charge change due to Vd
273  //Vt0bs0 and FF=FF0 causes least discontinuity in Cgs and Cgd but
      produces a spike in Cdd at Vds=0 (in weak inversion.  But bad in
      strong inversion)
274  etai      = ( Vgsi - ( Vt0bs0 - FF * aphit ))/ ( nphit );
275  if (etai <= LARGE_VALUE)
276  {
277    Qinvi   = Qref * log( 1.0 + exp( etai ));
278  }
279  else
280  {
281    Qinvi   = Qref * etai;
282  }
283  dQinv     = Qinv - Qinvi;
284  dibl_corr   = ( 1.0 - FF0 ) * ( 1.0 - Fsatq ) * qi * dQinv;
285  qd      = qd - dibl_corr;
286
287
288  //Inversion charge partitioning to terminals s and d
289  Qinvs     = type * Leff * (( 1 + dir ) * qs + ( 1 - dir ) * qd)/
      2.0;
290  Qinvd     = type * Leff * (( 1 - dir ) * qs + ( 1 + dir ) * qd)/
      2.0;
291
292
293  //Outer fringing capacitance
294  Qsov      = Cofs * ( V[g] - V[si] );
295  Qdov      = Cofd * ( V[g] - V[di] );
296
297
298  //Inner fringing capacitance
299  Vt0x      = Vt0 + gamma * ( sqrt( abs( phib - type * ( V[b] - V[si]
      ))) - sqrt(phib));
300  Vt0y      = Vt0 + gamma * ( sqrt( abs( phib - type * ( V[b] - V[di]
      ))) - sqrt(phib));
301  Fs_arg      = ( Vgsraw - ( Vt0x - Vdsi * delta * Fsat ) + aphit *
      0.5 )/ ( 1.1 * nphit );
302  if (Fs_arg <= LARGE_VALUE)
303  {
304    Fs    = 1.0 + exp( Fs_arg );
305    FFx   = Vgsraw - nphit * log( Fs );
306  }
307  else
308  {
309    Fs    = 0.0;                //  Not used
310    FFx   = Vgsraw - nphit * Fs_arg;
311  }
312  Fd_arg      = ( Vgdraw - ( Vt0y - Vdsi * delta * Fsat ) + aphit *
      0.5 )/ ( 1.1 * nphit );
```

```
313   if (Fd_arg <= LARGE_VALUE)
314   {
315     Fd    = 1.0 + exp( Fd_arg );
316     FFy   = Vgdraw - nphit * log( Fd );
317   }
318   else
319   {
320     Fd    = 0.0;                //  Not used
321     FFy   = Vgdraw - nphit * Fd_arg;
322   }
323   Qsif      = type * ( Cif + CC * Vgsraw ) * FFx;
324   Qdif      = type * ( Cif + CC * Vgdraw ) * FFy;
325
326
327   //Partitioned charge
328   Qs      = -W * ( Qinvs + Qsov + Qsif );        //  s-terminal charge
329   Qd      = -W * ( Qinvd + Qdov + Qdif );        //  d-terminal charge
330   Qg      = -( Qs + Qd + Qb );                   //  g-terminal charge
331
332   //Sub-circuit initialization
333   Idi_si = type * dir * Id;
334   Id_di  = ( V[d] - V[di] )/ Rd;
335   Isi_s  = ( V[si] - V[s] )/ Rs;
336   Qsi_b  = ( Qs );              //  charge term: node si to node b
337   Qdi_b  = ( Qd );              //  charge term: node di to node b
338   Qg_b   = ( Qg );              //  charge term: node g to node b
339   }
340 }
341
342 void MVS::print(void )
343 {
344   std::cout << Idi_si << std::endl;
345 }
346
347 void MVS::load_model(void )
348 {
349   // Model card
350 #include "ptype_lib1.h"
351 }
```

References

[1] D. S. Boning and S. Nassif, "Models of process variations in device and interconnect," in *Design of High Performance Microprocessor Circuits, chapter 6*. IEEE Press, 1999.

[2] Q. Y. Tang, "Uncertainty propagation in transistor-level statistical circuit analysis," Ph.D. dissertation, UC Berkeley, 2011.

[3] M. Miranda, "When every atom counts," *Spectrum, IEEE*, vol. 49, no. 7, pp. 32–32, July 2012.

[4] C. Jacoboni and P. Lugli, *The Monte Carlo method for semiconductor device simulation*. Springer, 1989, vol. 3.

[5] H. E. Graeb, *Analog Design Centering and Sizing*, 1st ed. Springer Publishing Company, Incorporated, 2007.

[6] S. Saha, "Compact MOSFET modeling for process variability-aware VLSI circuit design," *Access, IEEE*, vol. 2, pp. 104–115, 2014.

[7] G. Neuberger, G. Wirth, and R. Reis, "Process variability," in *Protecting Chips Against Hold Time Violations Due to Variability*. Springer, 2014, pp. 5–14.

[8] D. Grabowski, M. Olbrich, and E. Barke, "Analog circuit simulation using range arithmetics," in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*. IEEE, 2008, pp. 762–767.

[9] A. M. Baraka and H. A. H. Fahmy, "Using range arithmetic in evaluation of compact models."

[10] LNCS - lecture notes of computer science. http://www.springer.com/computer/lncs?SGWID=0-164-0-0-0.

[11] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to interval analysis*. Philadelphia, PA, USA: Siam, 2009.

[12] G. Alefeld and G. Mayer, "Interval analysis: theory and applications," *Journal of Computational and Applied Mathematics*, vol. 121, no. 1–2, pp. 421 – 464, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042700003423

[13] V. Kreinovich. Interval computations. http://www.cs.utep.edu/interval-comp. Accessed: 2015-02-03.

[14] W. Hofschuster and W. Krämer, “C-XSC 2.0–A C++ library for extended scientific computing,” in *Numerical software with result verification*. Springer, 2004, pp. 15–35.

[15] C-XSC - a C++ class library. http://www2.math.uni-wuppertal.de/~xsc/xsc/cxsc.html. Universitaet Wuppertal.

[16] E. Gardeñes, M. Sainz, L. Jorba, R. Calm, R. Estela, H. Mielgo, and A. Trepat, “Model intervals,” *Reliable Computing*, vol. 7, no. 2, pp. 77–111, 2001. [Online]. Available: http://dx.doi.org/10.1023/A%3A1011465930178

[17] N. T. Hayes, “Introduction to modal intervals,” IEEE Interval Standard Working Group, Tech. Rep., 2009.

[18] E. Kaucher, “Interval analysis in the extended interval space ir,” in *Fundamentals of Numerical Computation (Computer-Oriented Numerical Analysis)*, ser. Computing Supplementum, G. Alefeld and R. Grigorieff, Eds. Springer Vienna, 1980, vol. 2, pp. 33–49.

[19] L. de Figueiredo and J. Stolfi, “Affine arithmetic: Concepts and applications,” *Numerical Algorithms*, vol. 37, no. 1-4, pp. 147–158, 2004.

[20] aaflib - an affine arithmetic C++ library. http://aaflib.sourceforge.net.

[21] S. P. Shary, “Interval gauss-seidel method for generalized solution sets to interval linear systems,” *Reliable computing*, vol. 7, no. 2, pp. 141–155, 2001.

[22] L. Wedepohl and L. Jackson, “Modified nodal analysis: an essential addition to electrical circuit theory and analysis,” *Engineering Science and Education Journal*, vol. 11, no. 3, pp. 84–92, Jun 2002.

[23] Accellera Verilog Analog Mixed-Signal Group. Verilog-AMS sample library - semiconductor device models. http://www.eda.org/verilog-ams/htmlpages/sample_lib.sdm.html. Accessed: 2015-02-03.

[24] R. F. Pierret *et al.*, “Semiconductor device fundamentals,” 1996.

[25] S. Rakheja and D. Antoniadis, “Mvs nanotransistor model (silicon),” Oct 2014. [Online]. Available: https://nanohub.org/publications/15

[26] Berkeley. Bsim group - bsim4. [Online]. Available: http://www-device.eecs.berkeley.edu/bsim/?page=BSIM4

[27] E. Hansen and S. Sengupta, “Bounding solutions of systems of equations using interval analysis,” *BIT Numerical Mathematics*, vol. 21, no. 2, pp. 203–211, 1981.

ملخص البحث

يعمل مصممي و مصنعي الدوائر الإلكترونية على الحصول أداء أفضل و وظائف أكثر لرقائقهم الإلكترونية. إحدى الطرق للحصول على ذلك هو تصغير نبائط أشباه الموصلات، الذي يمكن من وضع عدد أكبر من الدوائر على نفس المساحة، كما يزيد من سرعة النبائط. إلا أن عملية التصغير تلك تصاحبها زيادة في المتغيرات العشوائية ناتجة عن عملية التصنيع مما يؤثر على أداء الدوائر الإلكترونية. إن التحقق من الأداء المطلوب من الدوائر الإلكترونية أصبح يمثل تحديا مع زيادة تأثير متغيرات عملية التصنيع. في هذا السياق جرت العادة على استخدام وسائل التحليل الإحصائي لتوقع أثر هذه المتغيرات على أداء الدوائر الإلكترونية في مرحلة التصميم. يمثل حساب الفترة بديل متوقع للوسائل التقليدية في تحقيق أداء الدوائر الإلكترونية في وجود متغيرات عملية التصنيع.

نقدم في هذا العمل نظام محاكة يفعل استخدام التصميمات الحالية، مستبدلا المتغيرات الإحصائية بالفترات. و بذلك يمكن من استبدال أو تحسين نظم المحاكاة الإحصائية المعتادة. كما تم بناء محاكي دوائر إلكترونية يعتمد استخدام حساب الفترة و كذا مجموعة من النماذج للنبائط الإلكترونية المستخدمة.

تم اختبار دقة النتائج للنماذج المستخدمة قياسا على المحاكة الإحصائية. و تم استخدام المحاكي لاختبار دوائر خطية صغيرة، منتجاً نتائج طيبة.

مهنـــــــــدس: **أمين ماهر عبدالله بركة**

تاريـخ الميـلاد: **١٩٨٢\٩\١٦**

الجنسيـــــــة: **مصري**

تاريخ التسجيل: **٢٠١٠\١٠\١**

تـــاريخ المنـــح: \ \

القســـــــــم: **هندسة الإلكترونيات و الإتصالات الكهربية**

الدرجـــــــــة: **ماجستير العلوم**

المشرفون: **أ.م.د حسام علي حسن فهمي**

الممتحنـــون: **أ.د محمد محمود رياض الغنيمي**
أ.د محمد أمين إبراهيم دسوقي
(أستاذ بكلية الهندسة – جامعة عين شمس)
أ.م.د حسام علي حسن فهمي

عنـــوان الرسالـة:

استخدام حساب الفترات لمحاكاة الدوائر الإلكترونية

الكلمات الدالة:

حساب الفترة، نماذج النبائط المدمجة، محاكاة الدوائر، مونت-كارلو، تغير التصميم

ملخـص البحـث:

يستخدم تصغير نبائط أشباه الموصلات للحصول على أداء أفضل و وظائف أكثر لرقائق الإلكترونية. يصاحب عملية التصغير زيادة في المتغيرات العشوائية ناتجة عن عملية التصنيع مما يؤثر على أداء الدوائر الإلكترونية. إن التحقق من الأداء المطلوب من الدوائر الإلكترونية أصبح يمثل تحديا مع زيادة تأثير متغيرات عملية التصنيع. في هذا السياق جرت العادة على استخدام وسائل التحليل الإحصائي – مونت كارلو– لتوقع أثر هذه المتغيرات على أداء الدوائر الإلكترونية في مرحلة التصميم. يمثل حساب الفترة بديل متوقع للوسائل التقليدية في تحقيق أداء الدوائر الإلكترونية في وجود متغيرات عملية التصنيع.

نقدم في هذا العمل نظام محاكة يفعل استخدام التصميمات الحالية، مستبدلا المتغيرات الإحصائية بالفترات. و بذلك يمكن من استبدال أو تحسين نظم المحاكاة الإحصائية المعتادة. كما تم بناء محاكي دوائر إلكترونية يعتمد استخدام حساب الفترة و كذا مجموعة من النماذج للنبائط الإلكترونية المستخدمة.

تم اختبار دقة النماذج مقارنة بالمحاكاة الإحصائية، كما تم استخدام المحاكي لاختبار بعض الدوائر الخطية مظهرا نتائج طيبة.

استخدام حساب الفترات لمحاكاة الدوائر الإلكترونية

اعداد

أمين ماهر عبدالله بركة

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة

كجزء من متطلبات الحصول على درجة ماجستير العلوم

في

هندسة الإلكترونيات و الإتصالات الكهربية

يعتمد من لجنة الممتحنين:

الاستاذ الدكتور: حسام علي حسن فهمي — المشرف الرئيسي

الاستاذ الدكتور: محمد محمود رياض الغنيمي — الممتحن الداخلي

الاستاذ الدكتور: محمد أمين إبراهيم دسوقي — الممتحن الخارجي

أستاذ بكلية الهندسة – جامعة عين شمس

كلية الهندسة – جامعة القاهرة

الجيزة – جمهورية مصر العربية

٢٠١٥

استخدام حساب الفترات لمحاكاة الدوائر الإلكترونية

اعداد

أمين ماهر عبدالله بركة

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة

كجزء من متطلبات الحصول على درجة ماجستير العلوم

في

هندسة الإلكترونيات و الإتصالات الكهربية

تحت اشراف

د. حسام علي حسن فهمي

أستاذ مساعد

كلية الهندسة – جامعة القاهرة

الجيزة – جمهورية مصر العربية

٢٠١٥

استخدام حساب الفترات لمحاكاة الدوائر الإلكترونية

اعداد

أمين ماهر عبدالله بركة

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة

كجزء من متطلبات الحصول على درجة ماجستير العلوم

في

هندسة الإلكترونيات و الإتصالات الكهربية

كلية الهندسة – جامعة القاهرة

الجيزة – جمهورية مصر العربية

٢٠١٥