BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

ELECTRONIC CIRCUITS SIMULATION

SIMULACE ELEKTRONICKÝCH OBVODŮ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MICHAL ŽABKA**
AUTOR PRÁCE

SUPERVISOR **Ing. VÁCLAV ŠÁTEK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2017

Brno University of Technology - Faculty of Information Technology

Department of Intelligent Systems | Academic year 2016/2017

Bachelor's Thesis Specification

For: **Žabka Michal**
Branch of study: Information Technology
Title: **Electronic Circuits Simulation**
Category: Modelling and Simulation

Instructions for project work:

1. Study the methods for numerical solution of ordinary differential equations (Cauchy problems) and focus on the numerical integration method using the Taylor series-based computation.
2. Analyse parallel solution of the Taylor series-based method and specify the mathematical operations which could be performed simultaneously and independently in separate processors.
3. Study the methods for solution of electronic circuits using the worldwide used software products (Dymola, MATLAB, Maple, Spice).
4. Design a model of CMOS inverter, CMOS NAND and CMOS NOR (use parasitic capacitances) and verify the correct functionality of your models.
5. Test your models on suitable examples and specify the potential speed-up of the computation in comparison with other classical methods.

Basic references:

- Bartsch, H.J.: Matematické vzorce, ACADEMIA 2009, ISBN 80-200-1448-9.
- Kunovský, J.: Modern Taylor series methods. Habilitation, 1995, VUT Brno.
- Kocina F., Nečasová G., Veigend P., Chaloupka J., Šátek V. and Kunovský J.: Modelling VLSI Circuits Using Taylor Series. In: 14th International Conference of Numerical Analysis and Applied Mathematics. Rhodes, 2016.
- Online circuit simulator with spice, University of Bercley.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Bachelor's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor: **Šátek Václav, Ing., Ph.D.**, DITS FIT BUT
Beginning of work: November 1, 2016
Date of delivery: May 17, 2017

Petr Hanáček
Associate Professor and Head of Department

Abstract

The aim of this bachelor thesis is to introduce the most popular numerical methods for solving differential equation. Following part describe a electronic circuits and simulation programs. First part of this thesis is focused on the Taylor series method computation and its parallel solution. In another chapters, it will be describe the methods for solution of electronic circuits, the process of designing a model of CMOS inverter, CMOS NAND and CMOS NOR. The final part of this thesis is focused on simulation in various simulation programs and evaluation of the effectiveness of individual methods.

Abstrakt

Cílem této bakalářské práce je seznámit se s nejpopularnějšími numerickými metodami pro výpočet diferenciálních rovnic, elektronických obvodů a simulačních programů. První část této práce je zaměřena na výpočet s využitím metody Taylorovy řady a jejích paralelních vlastností. V další kapitole budou popsány metody pro výpočet elektronických obvodů, proces návrhu modelu CMOS invertoru, CMOS NANDu a CMOS NORu. Závěrečná část této prace je zaměřena na simulaci těchto obvodů v různých simulačních programech a shodnocení efektivity jednotlivých metod.

Keywords

Differential equation, Taylor series method, CMOS, inverter, NAND, NOR, Jacobi matrix, simulation, TKSL, Matlab, SPICE.

Klíčová slova

Diferenciální rovnice, metoda Taylorovy řady, CMOS, invertor, NAND, NOR, Jacobiho matice, simulace, TKSL, Matlab, SPICE.

Reference

ŽABKA, Michal. *Electronic Circuits Simulation.* Brno, 2017. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Václav Šátek, Ph.D.

Electronic Circuits Simulation

Declaration

Hereby I declare that I have developed this bachelor thesis independently under supervision of Ing. Václav Šátek Ph.D. and all relevant information sources, which were used are cited and included in the list of references.

.....
Michal Žabka
May 18, 2017

Acknowledgements

I would like to thank Ing. Václav Šátek, Ph.D. for his professional advice, helpfulness and great patience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Simulations on digital systems are very popular nowadays. The advantage is price and security. With the development of computer systems, calculations of numerical methods are accelerated. The well-known numerical methods for solving initial value problem are Euler's method, Runge-Kutt method and Taylor series method. The efficiency of the Taylor series computation method and their parallel properties is enhanced. The aim of this thesis is simulation of CMOS circuit, specifically CMOS inverter, CMOS NAND and CMOS NOR. Simulation of these circuits is complicated because they are mainly composed of a transistor. The transistor is an active semiconductor component, which consists of pairs of PN transitions. Differential equations for describing a transistor are complex and have many variables. Therefore, these circuits will be rewritten into circuits comprised of DC source, resistor and capacitor. In this paper I will focus on continuous simulation of these circuits using the Taylor series method in various simulation environments.

The text is divided into eight chapters. In the first three chapters I will describe numerical methods and their properties. The other two chapters will cover the circuits and the design of the CMOS circuit. Finally, I will introduce simulation programs and simulate the designed circuits.

Chapter 2

Analytical and numerical solutions of differential equations

Equations, which except one or more independent variables contains unknown functions and their derivation, are called differential equations. Generally, the differential equation can be written as

$$F(x_1, x_2, \ldots, x_n, y, \frac{\delta y}{\delta x_1}, \frac{\delta y}{\delta x_2}, \ldots, \frac{\delta y}{\delta x_n}, \frac{\delta^2 y}{\delta x_1^2}, \frac{\delta^2 y}{\delta x_1 \delta x_2}, \ldots, \frac{\delta^2 y}{\delta x_1^2}, \frac{\delta^2 y}{\delta x_1 \delta x_n}, \frac{\delta^2 y}{\delta x_1^2}, \ldots, \frac{\delta^k y}{\delta x_n^k}) = 0, \tag{2.1}$$

where $y(x_1, x_2, \ldots, x_n)$ is unknown function n variables. Solution of differential equation are functions, which with their derivations identically meets this differential equation.

$$y = \varphi(x_1, x_2, \ldots, x_n). \tag{2.2}$$

Differential equations containing only unknown functions with one independent variable, their derivations and independent variable itself are called ordinary differential equations.

$$f(x, y, y', y'', \ldots, y^{(k)}) = 0. \tag{2.3}$$

Differential equations containing unknown functions with more independent variables, their derivations and independent variables itself are called partial differential equations.

$$f(x, y, z, z_x, z_y, z_{xx}, z_{yy}, z_{xy}, \ldots) = 0. \tag{2.4}$$

Order of differential equation is given by the highest derivation in the differential equation. Degree of differential equation is given by the highest power function and their derivations in the differential equation. First degree differential equations are called linear differential equations [5].

2.1 Numerical methods

Finding the analytical solution of large sets of differential equations is complex sometimes impossible. Currently it is increasingly being used numerical solution. The numerical solution of the ordinary differential equation with the initial condition

$$y' = f(x, y), y(x_0) = y_0 \tag{2.5}$$

is consider as sequence of values

$$[y(x_0) = y_0, [y(x_1) = y_1], \dots [y(x_N) = y_N] \tag{2.6}$$

These numbers approximate the values of the exact analytical solution $y(\, x_0) \,, y(\, x_1) \,, \dots, y(\, x_N)$ in points $x_0, x_1, \dots, x_N$. We usually choose an equidistant network, ie. $x_i - x_{i-1} = h; i = 1, 2, \dots, N$. Number h is called an integration step. Function values between the selected points can be determined either by interpolation from surrounding calculated points or by re-applying the numerical method using smaller integration step. The calculation is iterative. The numerical solution y_i in point x_i is calculated from the previous value of numerical solution y_{i-1}. This applies to one-step methods. There are also multi-step methods, which are using k previous solutions $y_{i-1}, y_{i-2}, \dots, y_{i-k}$ to calculate actual solution. Using multi-step methods it is problematic in the first steps of the calculation, when we still do not have a sufficient number of previous values. To start the calculation it is necessary to use one of the one-step methods. Differential equations of higher orders must be converted to a system of ordinary first-order differential equations. The following methods may be used to realize this conversion:

- the method of decreasing the order of derivation - simpler but requires the right side of the equation with no input derivatives
- the method of gradual integration - it also manages the equations with the derivatives on the right side

Both methods use the modification of the equation and the implementation of auxiliary variables. They are usable even on systems of higher order equations - just repeat the procedure for each given equation[10].

2.1.1 Approximation by Taylor method

If we approximate the function f differentiable at x_0 in a small neighborhood $U(x_0)$ by linear polynomial functions (first degree) $T_1(x)$, we use the function whose graph is tangent to graph of the function f at the point $[x_0, f(x_0)]$, in other words, we want to make the point x_0 match the function and value of the first derivative of the function f and polynomial T_1. If the function f is n times differentiable, we can improve the precision of approximation in a small neighborhood of a point x_0, by using a polynomial of n-th degree T_n, after which we will require to make the point x_0 coincided with the function f to the n-th derivative [11] [12].

Basic formula is:

$$y_{i+1} \;=\; y_i + h y_i' + \frac{h^2}{2!} y_i'' + ... + \frac{h^n}{n!} y_i^{(n)} \tag{2.7}$$

Effective calculation for linear ODE is achieved by calculation first derivation and other derivations are calculated as derivation of previous step.

$$y_i' \;=\; a\, y_i \tag{2.8}$$

$$y_i'' \;=\; a\, y_i' \;=\; a^2\, y_i \tag{2.9}$$

$$y_i''' \;=\; a\, y_i'' \;=\; a^3\, y_i \tag{2.10}$$

$$y_i^{(n)} = a^n\, y_i \tag{2.11}$$

Calculation of whole Taylor series is achieved by formula below:

$$\begin{aligned} DY0 &= y_i \\ DY1 &= h\, y_i' = h\, a\, yi = h\, a\, DY0 \\ DY2 &= \frac{h^2}{2}\, y_i' = \frac{h^2}{2}\, a^2\, yi = \frac{h}{2}\, a\, DY1 \\ DY3 &= \frac{h}{3}\, a\, DY2 \\ &\vdots \\ DYn &= \frac{h}{n}\, a\, DY_{n-1} \end{aligned}$$

2.1.2 Approximation by Euler method

The most popular numerical methods for solving initial value problem

$$y' = f(x, y), y(x_0) = y_0 \tag{2.12}$$

are called finite difference methods. Approximate values are obtained for the solution at a set of grid points.

$$x_0 < x_1 < x_2 < \cdots < x_n \tag{2.13}$$

and the approximate value at each x_n is obtained by using some of the values obtained in previous steps. We begin with a simple but computationally inefficient method attributed to Leonhard Euler [2]. Euler method is defined by:

$$y_{n+1} = y_n + hf(x_n, y_n) \quad n = 0, 1, 2, \cdots \tag{2.14}$$

where h is equal to step size. The result of the method may be more accurate by shortening the calculation step h. From a certain value, however, a rounding error will begin to occur, and further shortening will only increase the error.

2.1.3 Approximation by Runge-Kutta methods

Euler's method gave us one possible approach for solving differential equations numerically. The problem with Euler method is that you have to use a small interval size to obtain stable result. The Runge-Kutta method produces a better result in fewer steps.

Second order Runge-Kutta method

The Second Order Runge-Kutta algorithm is using an estimate for the derivative at the midpoint of the interval between y_n and y_{n+1} would result in a better approximation for the function at y_{n+1}, than would using the derivative at y_n (i.e., Euler method).

$$k_1 = hf(x_n, y_n) \tag{2.15}$$

$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \tag{2.16}$$

$$y_{n+1} = y_n + k_2 \tag{2.17}$$

Fourth order Runge-Kutta method

The most commonly used method is Runge-Kutta fourth order method. Based on Taylor series and takes into account the members of the higher orders. The necessary derivative of the function $f(\ x_0, y_0)$ calculate a more complex differential method using other auxiliary points between adjacent nodes in the network.

$$k_1 = hf(x_n, y_n) \tag{2.18}$$

$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \tag{2.19}$$

$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \tag{2.20}$$

$$k_4 = hf(x_n + h, y_n + k_3) \tag{2.21}$$

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} \tag{2.22}$$

Chapter 3

Parallel system using Taylor method

Numerical calculation of differential equations by Taylor series uses two basic operations: multiplication in the term (DYp_i) and addition(iterative calculation y_{i+1}). These two basic operations can be performed in series or in parallel. Similarly, communication between processors can be solved serially or in parallel [10].

3.1 Parallel-parallel integrator

Multiplication is realized in a parallel multiplier and addition is realized in a parallel adder. Block diagram is shown on the Fig. 3.1. The symbol of the integrator is marked with a dashed lines.

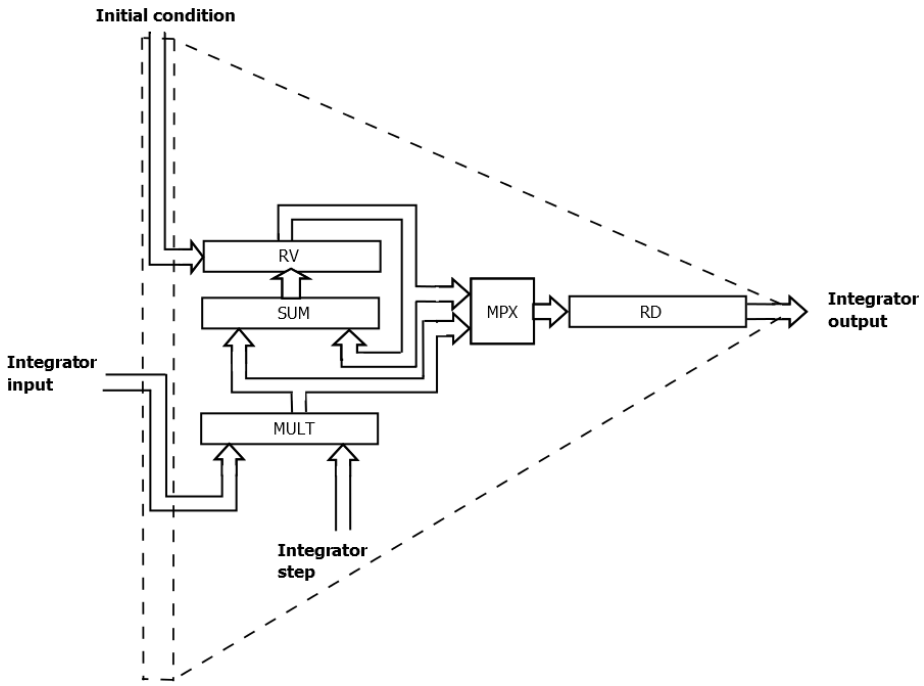

Figure 3.1: Scheme of the parallel-parallel integrator

The importance of individual blocks:

RV	Result register
RD	Multiplication register
MPX	Multiplexor
SUM	Parallel adder
MULT	Parallel multiplier

Integrator function: the cycle is initiated by storing the value y_i in the RD and RV register. The value $f(\ y_i)$ is displayed at the input, which is the output value of the element that is connected at the beginning of integrator. The integration step size is set to h. Multiplication (from the multiplier MULT) of these two inputs is copied to register RD and at the same time adding to register RD. Thus, in RD is a value of $DY1$ and in RV is a subtotal of $y_i + DY1$. In the next step, $f(\ DY1)$ and integration step $h/2$ appear at the input. Their multiplying is calculated by $DY2$, which is stored again in the RD sum with RV. The entire cycle is repeated until the desired accuracy or the maximum number of iterations is reached, yielding y_{i+1}. This type of integrator is the fastest type, the calculation time of one member of the Taylor series is:

$$t_{PP} = \tau_{nas} + \tau_{sec} + \tau_{sit} \tag{3.1}$$

Therefore, it is given by the sum of the time of the multiplication of τ_{nas} and the addition of τ_{sec}, or even the delay of the signals in the τ_{sit} interconnection network. However, at the expense of the greatest complexity of engagement, which has the biggest share combination multiplier. Another criterion is the number of inputs and outputs that is directly proportional to the parallel data bus width.

3.2 Serial-parallel integrator

In this variation, the multiplication is performed by a sequencing method based on the both multiplication algorithm. However addition is performed in parallel in one step as shown on Fig. 3.2.

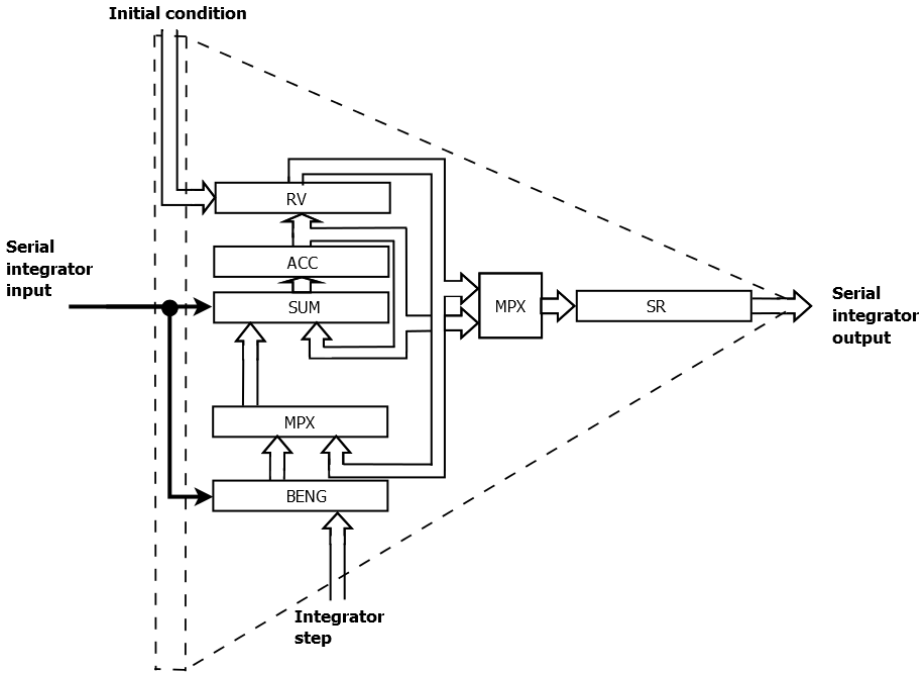

Figure 3.2: Scheme of the serial-parallel integrator

The importance of individual blocks:

RV	Result register
MPX	Multiplexor
SUM	Parallel adder
ACC	Accumulator
SR	Shift register
BNEG	Controlled-negation circuit (for sequence multiplication)

The principle of calculating the serial-parallel integrator is as follows: Y_i is stored in the RV and SR register. The integration step input has a value h, MPX is switched to a path from the block control negation BNEG. The calculation y_{i+1} then proceeds as follows: first, the accumulator ACC is reset, the least significant bit $f(\ y_i)$ is prepared at the input. This bit determines whether the multiplier from the RN register will be added to the ACC accumulator, subtracted (a negative form of the multiplier is created - the second complement) or whether it will be ignored (set to zero). The result of the adder is written into the ACC and the entire accumulator and shift register SR is then shifted by one position on the right side. This procedure is repeated until the last bit from $f(\ y_i)$ is received and then processed. Using this the multiplication of h and $f(\ y_i)$ is reached. The result of the multiplication is stored in the shift register SR. The multiplexer switches from BNEG to RV and the multiplier result DYp (stored in ACC) is added to the value stored in the result register RV and stored in the ACC and then into the RV. The entire cycle is repeated until the required accuracy or the maximum number of iterations is reached y_{i+1}.

The advantage of this approach is the small number of required wiring interface outlets, as data are inputting and outputting in series. If we want to refine the calculation by increasing the number of bits on which the numbers are displayed, then, unlike the previous variant, nothing changes in the total connection, „only“ changes the width of the registers, adder and extends the sequence of the control signals but the integrator interface, as well

as the interconnection network, will be retained. A disadvantage compared to the previous variant is a slowdown because multiplication is carried out in n steps (n = number of number bits). Calculation time of one member of the Taylor series is:

$$\begin{aligned} t_{SP} &= \tau_{nas} + \tau_{sec} + \tau_{sit} \\ t_{SP} &= n{\cdot}\tau_{sec} + \tau_{sec} + \tau_{sit} \end{aligned} \tag{3.2}$$

Therefore, is given by the sum of the multiplication time (which is actually in the addition steps $n{\cdot}\tau_{sec}$) and the addition of the result τ_{sec} of the multiplication result to the previous overall result or the delay of the signals in the τ_{sit} interconnection network. There is a possibility to optimize this integrator variant by using multiple bits overwriting at once (Booth's transcription with radix 4 or 8). To intermediate result of multiplication is added a multiple of the integration step, which is determined by the transcription of the input bit group. This allows you to reduce the number of steps required to multiply. In this variant, it is not enough just to create a positive and negative value for the multiplier. There must be a circuit that will create a 2 or 4-bit trans-coding at the same time.

3.3 Serial-serial integrator

This variant is based on the principle of a serial-parallel integrator. Unlike the previous variant, it performs sequentially not only the multiplication but also the addition operation. See Fig. 3.3.

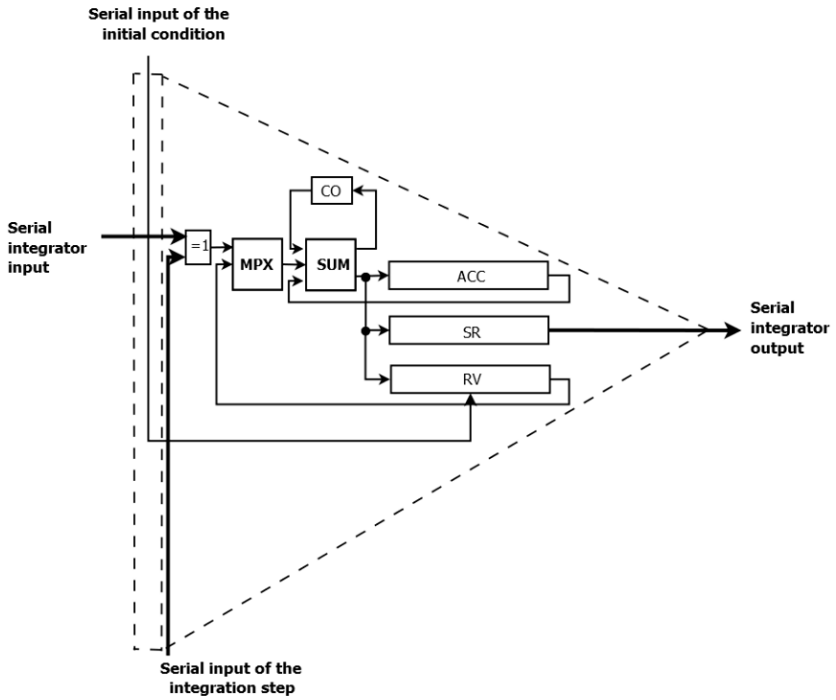

Figure 3.3: Scheme of the serial-serial integrator

The importance of individual blocks:

MPX	Multiplexor
SUM	Parallel adder
ACC	Accumulator
RV	Result shift register
SR	Output shift register
CO	flip-flop for transfer preservation

The function is as follows: First, ACC is reset, y_i is inserted into the RV. The CO transfer retention circuit is reset. The multiplexer MPX is set to the ACC path. At the integrator input, the least significant bit $f(\ y_i)$ appears, and the individual bits of the integration step, starting with the least significant bit, appear gradually on the serial input of the integration step. This sequence is added to the accumulator in series, depending on the integrator input value. When the intersection calculation is completed, a more significant bit$f(\ y_i)$ is set at the input and the output register SR is shifted at the same time. The whole procedure is repeated until the last (most significant) bit $f(\ y_i)$ appears at the integrator input. After the multiplication operation is completed, the value stored in the ACC (DYp) is stored in the SR output register and added in series to the value stored in the result register. This variant again has less connection requirements, but the price, which in this case is the number of cycles that are needed for the whole calculation (and therefore time), is given by an exponential relation.

$$\begin{aligned} t_{SS} &= \tau_{nas} + \tau_{sec} + \tau_{sit} \\ t_{SS} &= n{\cdot}n{\cdot}\tau_{sec} + n{\cdot}\tau_{sec} + \tau_{sit} \\ t_{SS} &= n^2{\cdot}\tau_{sec} + n{\cdot}\tau_{sec} + \tau_{sit} \end{aligned} \tag{3.3}$$

τ_{sec} in this version is the addition time of a full one-bit adder, n is the number of bits on which both multiplier and multiplication values are stored. More information can be found in [10].

Chapter 4

Electric circuits

The main point of this thesis is modeling and solving electric circuits. Electronic circuits are integral parts of nearly all of the technological advancement in our lives today. Television, radio, phones and computers. An electric circuits are often schematically represented as 4.1.

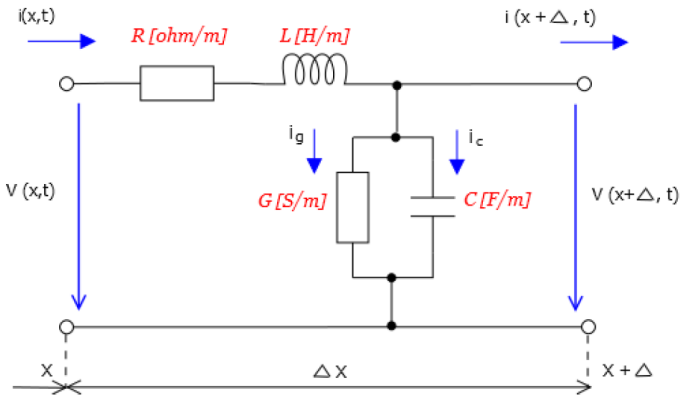

Figure 4.1: Electric circuit schema

Where R, L, G, C are per unit length quantities defined as follows:

- R : series resistance per unit length, for both conductors, in Ω/m.
- L : series inductance per unit length, for both conductors, in H/m.
- G : shunt conductance per unit length, in S/m.
- C : shunt capacitance per unit length, in F/m.

The parameters need to be calculated or estimated before any model can be built. Conductance G is ignored in short circuit studies because the inductance of the line is the dominant value.

4.1 Electric circuits Equations

In complex circuits, we can not simply use Ohm's Law alone to find the voltages or currents circulating within the circuit. For these types of calculations we need certain rules which allow us to obtain the circuit equations and for this we can use Kirchof's Circuit Law.

Kirchof's Circuit Law are a set of rules or laws which deal with the conservation of current and voltage within Electrical Circuits. These two rules are commonly known as: Kirchof's Circuit Laws with one of Kirchof's laws dealing with the current owing around a closed circuit, *Kirchof's Current Law(KCL)* while the other law deals with the voltage sources present in a closed circuit, *Kirchof's Voltage Law(KVL)*.

4.1.1 Kirchof's First Law(KCL)

Kirchof's Current Law or KCL, states that the *„total current or charge entering a junction or node is exactly equal to the charge leaving the node as it has no other place to go except to leave, as no charge is lost within the node"*. In other words the algebraic sum of all the currents entering and leaving a node must be equal to zero, $I(exiting) + I(entering) = 0$. This idea by Kirchof's is commonly known as the **Conservation of Charge**.

4.1.2 Kirchof's Second Law(KVL)

Kirchof's Voltage Law or KVL, states that *„in any closed loop network, the total voltage around the loop is equal to the sum of all the voltage drops within the same loop"* which is also equal to zero. In other words the algebraic sum of all voltages within the loop must be equal to zero. This idea by Kirchof's is known as the **Conservation of Energy.**

4.2 Example

Consider a simple parallel RL circuit with one voltage source, one resistor and one inductor as shown in Figure 4.2.
Parameters will be :
$U = 1V$
$R = 1\Omega$
$L = 1H$

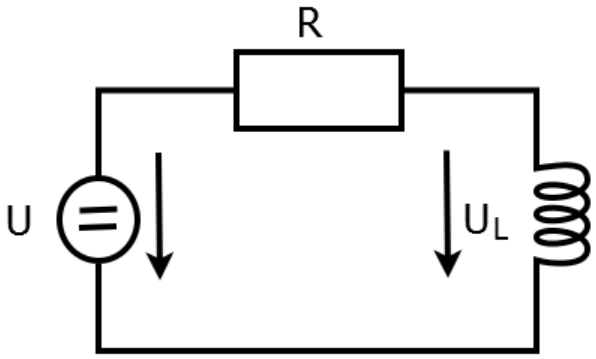

Figure 4.2: Electric circuit schema

Basic formula for describing RL circuit by differential equation is:

$$i' = \frac{1}{L} U_L \quad i(0) = 0 \tag{4.1}$$

Using Kirchof's second law we found U_L

$$i * R + U_L - U = 0 \tag{4.2}$$

$$U_L = U - i * R \tag{4.3}$$

then we put into formula

$$i' = \frac{1}{L} U - i * R \tag{4.4}$$

$$\frac{di}{dt} = \frac{1}{L} U - i * R \tag{4.5}$$

$$\frac{di}{U - i * R} = \frac{1}{L} dt \tag{4.6}$$

$$\frac{\frac{1}{L}(-R)\, di}{U - i * R} = \frac{1}{L} dt \tag{4.7}$$

integrate equation

$$\frac{-1}{L} ln(U - i * R) = \frac{1}{L} dt \tag{4.8}$$

$$ln(U - i * R) = \frac{-R}{L} t \tag{4.9}$$

$$ln(U - i * R) = lne^{\frac{-R}{L} t} + lnK \tag{4.10}$$

we logarithm both side

$$U - i * R = Ke^{\frac{-R}{L} t} \tag{4.11}$$

$$i * R = U - Ke^{\frac{-R}{L} t} \tag{4.12}$$

general solution

$$i = 1 - Ke^{t} \tag{4.13}$$

$$0 = 1 - Ke^{0} \tag{4.14}$$

$$K = 1 \tag{4.15}$$

particular solution

$$i = 1 - e^{-t} \tag{4.16}$$

Taylor series

$$e^{t} = 1 + \frac{t}{1!} + \frac{t^2}{2!} + \frac{t^3}{3!} \cdots + \frac{t^n}{n!} \tag{4.17}$$

For $t = 1$

$$i = e^{-t} \tag{4.18}$$

$$i = 1 - (1 - 1 + \frac{1^2}{2!} + \frac{1^3}{3!} + \frac{1^4}{4!} + \frac{1^5}{5!} + \frac{1^6}{6!} + \frac{1^7}{7!} + \frac{1^8}{8!} + \frac{1^9}{9!} + \frac{1^{10}}{10!} \tag{4.19}$$

$$i = 0.632120536 \tag{4.20}$$

Chapter 5

Simulation programs

There are numbers of software packages in the market such as Multisim, SPICE etc. which are excellent for the students to learn about simulation of electrical circuits. However, deeper understanding of the circuits can be promoted if students were writing their own code in environment like MATLAB or MAPLE and then simulate the circuit in Simulink to analyze the circuit behavior. This chapter will discuss several softwares, which may be used for simulation and solving diferential equations.

5.1 MATLAB

MATLAB is a numeric computation software for engineering and scientific calculations. The name MATLAB stands for MATRIX LABORATORY. MATLAB is primarily a tool for matrix computations. MATLAB software is receiving phenomenal attention from engineering and scientific communities. MATLAB avoids the awkward compilation often encountered in traditional languages such as C++, JAVA etc. Furthermore it provides an opportunity for easy debugging, easy to execute workspace, and uses the built in library functions which avoids hundreds of programming statements. Also, provides the convenience of Graphical User Interface (GUI) design tools as well as scientific modelling platform without laborious programming.

Matlab is very useful for doing numerical computations with matrices and vectors. It can also display information graphically. The best way to learn what Matlab can do is to work through some examples at the computer.

The main reason for using MATLAB in this thesis is that we can use it to analyze motion of an engineering system. To do this, we always need to solve a differential equation. MATLAB has powerful numerical methods to solve differential equations called ODE(ordinary differential equation)[3] [17] [14] [1].

5.2 MAPLE

MapleSimTM is a modeling environment for creating and simulating complex multidomain physical systems. It allows us to build component diagrams that represent physical systems in a graphical form. Using both symbolic and numeric approaches, MapleSim automatically generates model equations from a component diagram and runs high-fidelity simulations. We can use MapleSim to build models that integrate components from various engineering fields into a complete system. MapleSim features a library of over 300 modeling compo-

nents, including electrical and thermal devices, sensors and sources, signal blocks. We can also create custom components to suit our modeling and simulation needs. MapleSim uses the advanced symbolic and numeric capabilities of MapleTM to generate the mathematical models that simulate the behavior of a physical system. We can, therefore, apply simplification techniques to equations to create concise and numerically efficient models. MapleSim provides various pre-built templates in the form of Maple work sheets for viewing model equations and performing advanced analysis tasks, such as parameter optimization. To analyze our model and present our simulation results in an interactive format, we can use Maple features such as embedded components, plotting tools and document creation tools. We can also translate our models into C code and work with them in other applications and tools, including applications that allow us to perform real-time simulation. In MapleSim, all the calculations are performed automatically. We only need to draw the circuit and provide the component parameters. These principles can be applied equally to all engineering domains in MapleSim and allow us to connect components in one domain with components in others easily [15].

5.3 TKSL

System TKSL is simulation language for computing differential equations (initial value problems). All calculations are based on differential equations and are solved using Taylor series method. The system allows numerical solution. Entry is a system of differential equations. The program is designed for environments MSDOS.
TKSL / 386 is programmed in Pascal, and it is already on powerful computers being inadequate. Program TKSL / C was created one Faculty of information Technology in Brno. It is written in C ++ thus be the final solution could be used on Microsoft Windows operating systems. Also solves the problems and some of the original version [13].

5.4 SPICE

SPICE program analyze a circuit based on a text-file description of the circuit's components and connections. By itself, SPICE does not require a graphic interface and demands little in system resources. It is also very reliable. SPICE is fairly easy to use for simple circuits, and its non-graphic interface actually lends itself toward the analysis of circuits that can be difficult to draw. Graphics may look more attractive, but abstracted interfaces (text) are actually more efficient.

SPICE is a general-purpose circuit simulation program for nonlinear dc, nonlinear transient and linear ac analyses. Circuits may contain resistors, capacitors, inductors, mutual inductors, independent voltage and current sources, four types of dependent sources, transmission lines and the four most common semiconductor devices: diodes, BJT's, JFET's and MOSFET's.

SPICE has built-in models for the semiconductor devices and the user need to specify only the pertinent model parameter values. The model for the BJT is based on the integral charge model of Gummel and Poon. If the Gummel-Poon parameters are not specified, the model reduces to the simpler Ebers-Moll model. In either case, charge storage effects, ohmic resistances and a current-dependent output conductance may be included. The diode model can be used for either junction diodes or Schottky barrier diodes. The JFET model is based on the FET model of Shichman and Hodges. Three MOSFET models are implemented.

MOS1 is described by a square-law I-V characteristic MOS2 is an analytical model while MOS3 is a semi-empirical model. Both MOS2 and MOS3 include second-order effects such as channel length modulation, subthreshold conduction, scattering limited velocity saturation, small size effects and charge-controlled capacitances[16].

5.5 DYMOLA

Dymola is modeling and simulation tool suitable for modeling of various kinds of physical systems. It supports hierarchical model composition, libraries of truly reusable components, connectors and composite connections. Model libraries are available in many engineering domains. Dymola uses a new modeling methodology based on object orientation and equations. [6] [7].

Modelica is a high-level declarative language for describing mathematical behavior. It is typically applied to engineering systems and can be used to easily describe the behavior of different types of engineering components (e.g.,resistors, clutches, etc.). These components can be combined into subsystems, systems or even architectures. Modelica is compelling for several reasons. First and foremost, it is technically very capable. By using complex algorithms behind the scenes, Modelica compilers allow engineers to focus on high-level mathematical descriptions of component behavior and get high performance simulation capability in return without having to be deeply knowledgeable about complex topics like differential-algebraic equations, symbolic manipulation, numeric solvers, code generation, post-processing, etc.. The key to Modelica's technical success is the support for a wide range of modeling formalism that allow the description of both continuous and discrete behavior framed in the context of hybrid differential-algebraic equations. The language supports both causal (often used for control system design) and acausal (often used in creating schematic oriented physical designs) approaches within the same model. Finally, another compelling aspect of Modelica is the fact that it was designed from the start as an open language. The specification is freely available and tool vendors are encouraged to support the import and export of Modelica (without being compelled to pay). Modelica is really an ideal language for modeling the behavior of engineering systems in nearly any engineering domain. It seamlessly supports both physical design and control design in a single language. It is also multi-domain so it does not impose any artificial boundaries that restrict its use to select engineering domains or systems. The result is that it provides a complete set of capabilities for building lumped system models of nearly any engineering system [6] [7].

Chapter 6

Modeling of CMOS technology

This chapter is based on [9] [4]. Complementary metal–oxide–semiconductor is a technology for constructing integrated circuits. CMOS is used in most very large scale integrated (VLSI) or ultra-large scale integrated (ULSI) circuit chips. The term „VLSI“ is generally associated with chips containing thousands or millions of metal oxide semiconductor field effect transistors (MOSFETs). The term „ULSI“ is generally associated with chips containing billions, or more, MOSFETs. We will describe how to solve CMOS logic circuit using the capacitor substitution which leads to the system of differential equations that can be solved numerically.

6.1 CMOS inverter

The scheme of CMOS inverter (Fig. 6.1 left) presents an electronic circuit. The function of this scheme is feasible to be demonstrated by means of an electric circuit (Fig. 6.1 middle). Further it is possible to substitute each transistor with a pair of resistor-capacitor as shown in Fig. 6.1 (right).

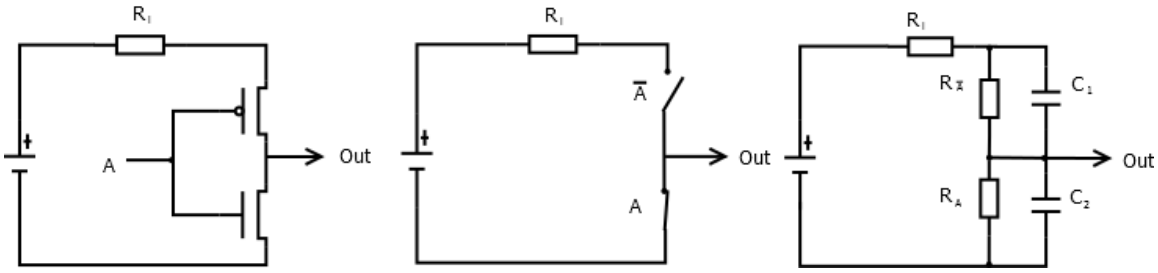

Figure 6.1: CMOS Inverter

If input A is logical one (e.g. 3.3 V for recent CMOS transistors but the value is dependent on an application), then the p-type transistor (upper one) is closed while the n-type transistor is open - it is simulated by high $R_{\overline{A}} = 10^{10}\Omega$ and low $R_A = 1\Omega$. Following differential equations for this regular electric circuit can be constructed (parameters $U = 3.3V; R_i = 1\Omega; C_1 = C_2 = 5\mu F; R_A; R_{\overline{A}} \in \{1, 10^{10}\}\Omega$; capacitor C_1 is precharged).

$$i = \frac{1}{R_i} \cdot (U - u_{C1} - u_{C2}) \tag{6.1}$$

$$u'_{C_1} = \frac{1}{C_1} \cdot (i - \frac{1}{R_{\overline{A}}} \cdot u_{C_1}), u_{C_1}(0) = 3.3 \tag{6.2}$$

$$u'_{C_2} = \frac{1}{C_2} \cdot (i - \frac{1}{R_A} \cdot u_{C_1}), u_{C_2}(0) = 0 \tag{6.3}$$

6.2 CMOS NAND and NOR

Transformation of CMOS NAND and NOR can be done similarly. NAND is shown in Fig. 2 (left), NOR is presented in Fig. 2 (right). Capacitors C_{12}, resp. C_{34} already consist of capacitors C_1 and C_2 (parallel capacitors), resp. C_3 and C_4, therefore $C_{12} = C_1 + C_2$ and $C_{34} = C_3 + C_4$.

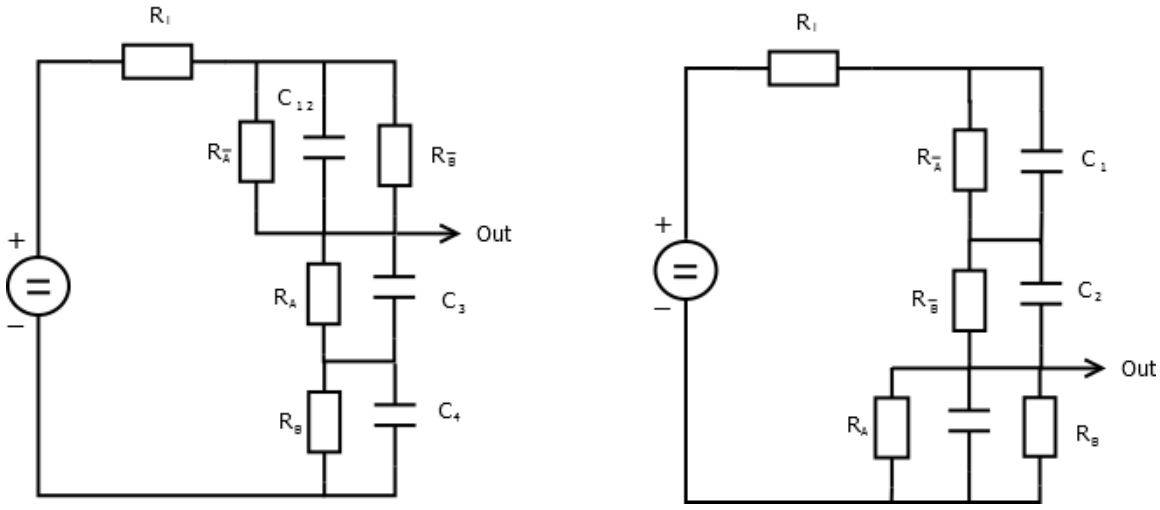

Figure 6.2: CMOS NAND/NOR

The left circuit (Fig. 6.2) is described by the following equations (parameters $U = 3.3V; R_i = 1\Omega; C_1 = C_2 = C_3 = C_4 = 5\mu F; C_{12} = C_1 + C_2; R_A, R_{\overline{A}}, R_B, R_{\overline{B}} \in \{1, 10^{10}\}\Omega$; capacitor C_{12} is precharged)

$$i = 1\frac{1}{R_i} \cdot (U - u_{C_{12}} - u_{C_1} - u_{C_2}) \tag{6.4}$$

$$u'_{C_{12}} = \frac{1}{C_{12}} \cdot (i - \frac{R_{\overline{A}} + R_{\overline{B}}}{R_{\overline{A}} \cdot R_{\overline{B}}} \cdot u_{C_{12}}), u_{C_{12}}(0) = 3.3 \tag{6.5}$$

$$u'_{C_3} = \frac{1}{C_3} \cdot (i - \frac{1}{R_A} \cdot u_{C_3}), u_{C_3}(0) = 0 \tag{6.6}$$

$$u'_{C_4} = \frac{1}{C_4} \cdot (i - \frac{1}{R_B} \cdot u_{C_4}), u_{C_4}(0) = 0 \tag{6.7}$$

The right circuit (Fig. 6.2) is described by the following equations (parameters are the same as for the left one, capacity $C_{34} = C_3 + C_4$, capacitors C_1 and C_2 are half-precharged).

$$i = 1\frac{1}{R_i} \cdot (\, U - u_{C1} - u_{C2} - u_{C_{34}}) \tag{6.8}$$

$$u'_{C_1} = \frac{1}{C_1} \cdot (\, i - \frac{1}{R_{\overline{A}}} \cdot u_{C_1}) \,, u_{C_1}(\,0) \, = 1.65 \tag{6.9}$$

$$u'_{C_2} = \frac{1}{C_2} \cdot (\, i - \frac{1}{R_{\overline{B}}} \cdot u_{C_1}) \,, u_{C_2}(\,0) \, = 1.65 \tag{6.10}$$

$$u'_{C_{34}} = \frac{1}{C_{34}} \cdot (\, i - \frac{R_A + R_B}{R_A \cdot R_B} \cdot u_{C_{34}}) \,, u_{C_{34}}(\,0) \, = 0 \tag{6.11}$$

Chapter 7

Experiments

7.1 Simulation of inverter in TKSL

$Tmax$ for simulation is set to $1ms$. Input A is set to logical 1 for interval $t =< 0; 0.5)ms$ and to logical 0 for interval $t =< 0.5, 1 > ms$. Therefore output OUT has for first interval very small value which indicates logical 0 and value close to 3.3 which indicates logical 1 for second interval. ORD is 10. See Fig. 7.1.

Figure 7.1: TKSL Inverter

7.2 Simulation of NAND in TKSL

This section will be about simulation of NAND circuit in TKSL. $Tmax$ is equal to 1ms. Values close to 0 equals to logical 0, values close to 3.3 equals to logical 1. Result of figure 7.2 is shown in table below:

t	A	B	OUT
0 - 0.24	0	0	1
0.25 - 0.49	0	1	1
0.5 - 0.74	1	0	1
0.75 - 1	1	1	0

Table 7.1: TKSL NAND

Figure 7.2: TKSL NAND

7.3 Simulation of NOR in TKSL

Now we change equations to simulate NOR circuit. $Tmax$ is equal to 1ms. Values close to 0 equals to logical 0, values close to 3.3 equals to logical 1. Result of Fig. 7.3 is shown in table below:

t	A	B	OUT
0 - 0.24	0	0	1
0.25 - 0.49	0	1	0
0.5 - 0.74	1	0	0
0.75 - 1	1	1	0

Table 7.2: TKSL NOR

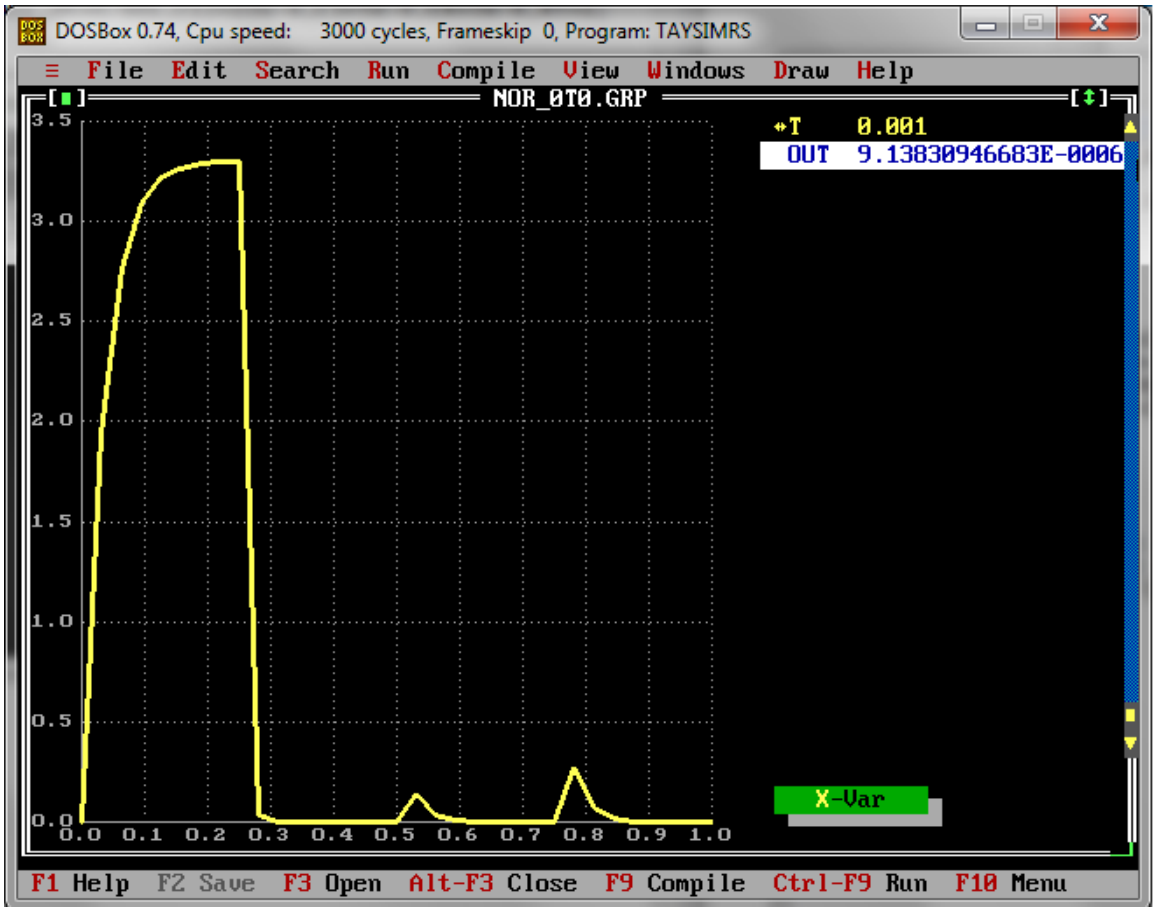

Figure 7.3: TKSL NOR

7.4 Simulation of inverter in Matlab

This section will be focused on different methods used for solving differential equation in Matlab. System of differential equations can be presented as:

$$y_1' = a_{11}\, y_1 + a_{12}\, y_2 + b_1 \qquad y_1(0) = y_{10} \tag{7.1}$$

$$y_2' = a_{21}\, y_1 + a_{22}\, y_2 + b_2 \qquad y_2(0) = y_{20} \tag{7.2}$$

These two differential equations can be put into matrix and vectors as shown below:

$$\vec{y} = \begin{pmatrix} \vec{y_1} \\ \vec{y_2} \end{pmatrix}, \qquad \vec{y(0)} = \begin{pmatrix} \vec{y_1(0)} \\ \vec{y_2(0)} \end{pmatrix}, \qquad A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \qquad \vec{b} = \begin{pmatrix} \vec{b_1} \\ \vec{b_2} \end{pmatrix} \tag{7.3}$$

where A is Jacobian matrix of constants and $\vec{b}$ is vector of constants.
Second derivative y'' using first derivative y' calculated in previous step is shown below:

$$y_1'' = a_{11}\, y_1' + a_{12}\, y_2' = a_{11}\,(a_{11}\, y_2 + a_{12}\, y_2 + b_1) + a_{12}\,(a_{21}\, y_1 + a_{22}\, y_2 + b_2) \tag{7.4}$$

$$y_2'' = a_{21}\, y_1' + a_{22}\, y_2' = a_{21}\,(a_{11}\, y_2 + a_{12}\, y_2 + b_1) + a_{22}\,(a_{21}\, y_1 + a_{22}\, y_2 + b_2) \tag{7.5}$$

$$\vec{y}^{(n)} = A \,.\, \vec{y}^{\,n-1} \qquad \overrightarrow{DY}_{(n)} = \frac{h}{n} \,.\, A \,.\, \overrightarrow{DY}_{n-1} \tag{7.6}$$

$$u_{c1}' = \frac{1}{C_1}\,(\frac{1}{R_1}\,(U - u_{c1} - u_{c2}) - \frac{1}{R_i}\,u_{c1}) \tag{7.7}$$

$$u_{c2}' = \frac{1}{C_2}\,(\frac{1}{R_2}(U - u_{c1} - u_{c2}) - \frac{1}{R_i}\,u_{c2}) \tag{7.8}$$

$$A = \begin{pmatrix} -\frac{1}{C_1\,R_1} - \frac{1}{C_1\,R_i} & -\frac{1}{C_1\,R_1} \\ -\frac{1}{C_2\,R_1} & -\frac{1}{C_2\,R_1} - \frac{1}{C_2\,R_i} \end{pmatrix}, \qquad \vec{b} = \begin{pmatrix} \frac{U}{C_1\,R_1} \\ \frac{U}{C_2\,R_2} \end{pmatrix} \tag{7.9}$$

$$U_c' = A\,u_c + \vec{b} \tag{7.10}$$

Matrix 7.9 was used as input into solving methods. We have chosen five solving methods: ORD23 solver, ORD23 solver with absolute tolerance equal to 1^{-10}, ORD45 solver, ORD45 solver with absolute tolerance equal to 1^{-10} and Taylor series method implemented in explicitTaylorLinear.m [8]. R_A is set as 10^{10}, which is equal to logical 0. Therefore output must be around 3.3, which is equal to logical 1. Taylor series method computation is shown in Fig. 7.4.

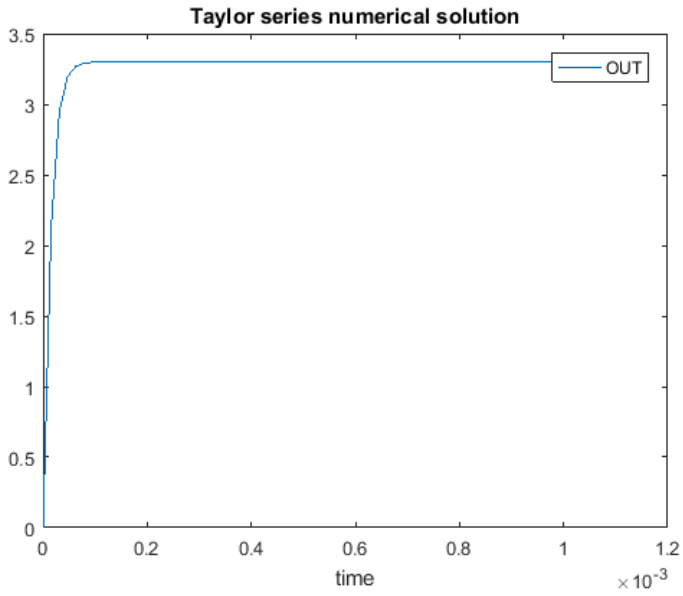

Figure 7.4: Matlab invertor Taylor method

When *OUT* is equal to logical 1, ODE solvers are producing very similar graphs as we can see in Fig. B.1 and Fig. B.2.

Comparison between efficiency of each method is shown in table 7.3.

Method	Elapsed time [s]	Number of steps	Absolute error in last step towards Taylor method
Taylor	0.010863	65	X
ODE23	0.013446	225	3.73883e-07
ODE23 Tolerance 1e-10	2.217596	4056	3.18265e-11
ODE45	0.018998	657	3.57374e-07
ODE45 Tolerance 1e-10	0.33462	1521	9.03322e-12

Table 7.3: Matlab invertor methods efficiency for input logical 0

We can see that if we choose ODE method with specified absolute tolerance elapse time of method is increased as well as number of steps, but accuracy of calculation is better. This apply to all simulations.

When we set R_A to 1, which equal to logical 1. Therefore output must be very small number around 0, which is equal to logical 0. Computation is shown in Fig. B.3,B.4,B.5,B.6,B.7.

Efficiency of methods is shown in table 7.4.

Method	Elapsed time [s]	Number of steps	Absolute error in last step towards Taylor method
Taylor	0.011316	33	X
ODE23	0.011989	189	5.70239e-07
ODE23 Tolerance 1e-10	0.33499	209	2.67532e-11
ODE45	0.020541	621	5.10708e-07
ODE45 Tolerance 1e-10	0.182838	633	6.81423e-11

Table 7.4: Matlab invertor methods efficiency for input logical 1

We can see for Taylor method $ORDER = 33$, which is higher than $ORDER = 10$ from TKSL simulation. Differences between number of steps from ODE solvers with absolute toleration and without is smaller than in table 7.3.

7.5 Simulation of NAND in Matlab

In this section we will simulate CMOS NAND. For this purpose we will need three differential equation. We can use following formula:

$$y_1' = a_{11}\, y_1 + a_{12}\, y_2 + a_{13}\, y_3 + b_1 \qquad y_1(0) = y_{10} \tag{7.11}$$

$$y_2' = a_{21}\, y_1 + a_{22}\, y_2 + a_{23}\, y_3 + b_2 \qquad y_2(0) = y_{20} \tag{7.12}$$

$$y_3' = a_{31}\, y_1 + a_{32}\, y_2 + a_{33}\, y_3 + b_3 \qquad y_3(0) = y_{30} \tag{7.13}$$

$$\vec{y} = \begin{pmatrix} \vec{y_1} \\ \vec{y_2} \\ \vec{y_3} \end{pmatrix}, \qquad y(\vec{0}) = \begin{pmatrix} \vec{y_1(0)} \\ \vec{y_2(0)} \\ \vec{y_3(0)} \end{pmatrix}, \qquad A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \qquad \vec{b} = \begin{pmatrix} \vec{b_1} \\ \vec{b_2} \\ \vec{b_3} \end{pmatrix} \tag{7.14}$$

We use equations to describe NAND circuit from previous chapter and put it into formula.

$$u_{c12}' = \frac{1}{C_{12}} \left(\frac{1}{R_i} \left(U - u_{c12} - u_{c3} - u_{c4} \right) - \frac{R_{\bar{A}} + R_{\bar{B}}}{R_{\bar{A}}.R_{\bar{B}}} \,.\, U_{c12} \right) \tag{7.15}$$

$$u_{c3}' = \frac{1}{C_3} \left(\frac{1}{R_i} (U - u_{c12} - u_{c3} - u_{c4}) - \frac{1}{R_A} U_{c3} \right) \tag{7.16}$$

$$u_{c4}' = \frac{1}{C_4} \left(\frac{1}{R_i} (U - u_{c12} - u_{c3} - u_{c4}) - \frac{1}{R_B} U_{c4} \right) \tag{7.17}$$

then we write Jacobian matrix A.

$$A = \begin{pmatrix} -\frac{1}{C_{12}\, R_i} - \frac{R_{\bar{A}}+R_{\bar{B}}}{C_{12}.R_{\bar{A}}.R_{\bar{B}}} & -\frac{1}{C_{12}\, R_i} & -\frac{1}{C_{12}\, R_i} \\ -\frac{1}{C_3\, R_i} & -\frac{1}{C_3\, R_i} - \frac{1}{C_3\, R_A} & -\frac{1}{C_3\, R_i} \\ -\frac{1}{C_4\, R_i} & -\frac{1}{C_4\, R_i} & -\frac{1}{C_4 R_i} - \frac{1}{C_4\, R_B} \end{pmatrix}, \qquad \vec{b} = \begin{pmatrix} \frac{u}{C_{12}\, R_i} \\ \frac{u}{C_3\, R_i} \\ \frac{u}{C_4\, R_i} \end{pmatrix} \tag{7.18}$$

NAND circuit has two logical inputs R_A and R_B. Setting resistors to 10^{10} is equal to logical 0 and to 1 is equal to logical 1. This give us four combinations. Efficiency of our solvers is described in tables below:

Method	Elapsed time [s]	Number of steps	Absolute error in last step towards Taylor method
Taylor	0.007699	65	X
ODE23	0.015746	236	6.10705e-07
ODE23 Tolerance 1e-10	2.898517	3789	7.37961e-11
ODE45	0.021694	697	1.86033e-07
ODE45 Tolerance 1e-10	0.464563	1521	8.08953e-11

Table 7.5: Matlab NAND methods efficiency for input logical 0 0

Method	Elapsed time [s]	Number of steps	Absolute error in last step towards Taylor method
Taylor	0.012599	65	X
ODE23	0.025384	260	1.14415e-06
ODE23 Tolerance 1e-10	3.175594	4164	6.39933e-12
ODE45	0.027306	769	1.20005e-07
ODE45 Tolerance 1e-10	0.495461	1609	1.03735e-11

Table 7.6: Matlab NAND methods efficiency for input logical 0 1

Method	Elapsed time [s]	Number of steps	Absolute error in last step towards Taylor method
Taylor	0.015629	65	X
ODE23	0.023419	260	1.14415e-06
ODE23 Tolerance 1e-10	3.312756	4164	2.97056e-11
ODE45	0.032606	769	1.20005e-07
ODE45 Tolerance 1e-10	0.473937	1609	2.61093e-11

Table 7.7: Matlab NAND methods efficiency for input logical 1 0

Method	Elapsed time [s]	Number of steps	Absolute error in last step towards Taylor method
Taylor	0.016928	65	X
ODE23	0.017443	249	1.46182e-06
ODE23 Tolerance 1e-10	0.445404	267	3.1655e-11
ODE45	0.026271	797	6.11117e-08
ODE45 Tolerance 1e-10	0.282186	809	5.1099e-12

Table 7.8: Matlab NAND methods efficiency for input logical 1 1

In Tab. 7.8 we can see that setting up tolerance for solver does not have big impact on number of steps, but still better accuracy of approximation. Taylor series approximation for input $R_A = 1, R_B = 1$ we can see in Fig. 7.5

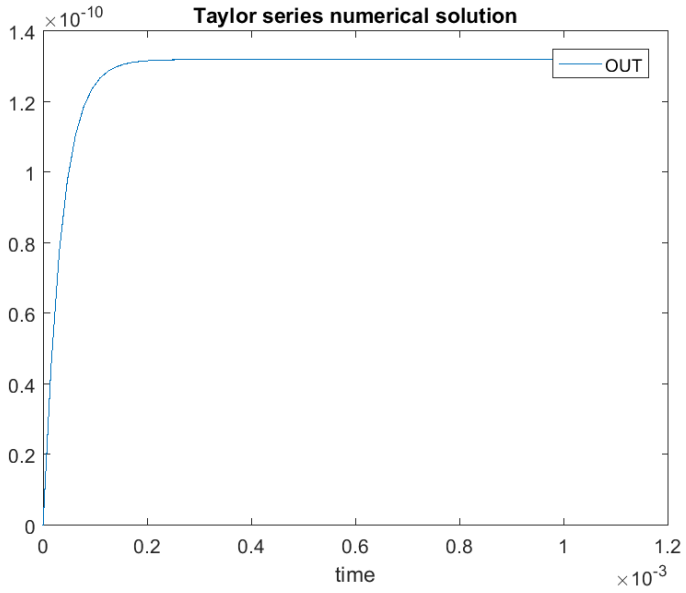

Figure 7.5: Matlab NAND Taylor method

ODE45 solution for same input values, is shown in Fig.7.6.

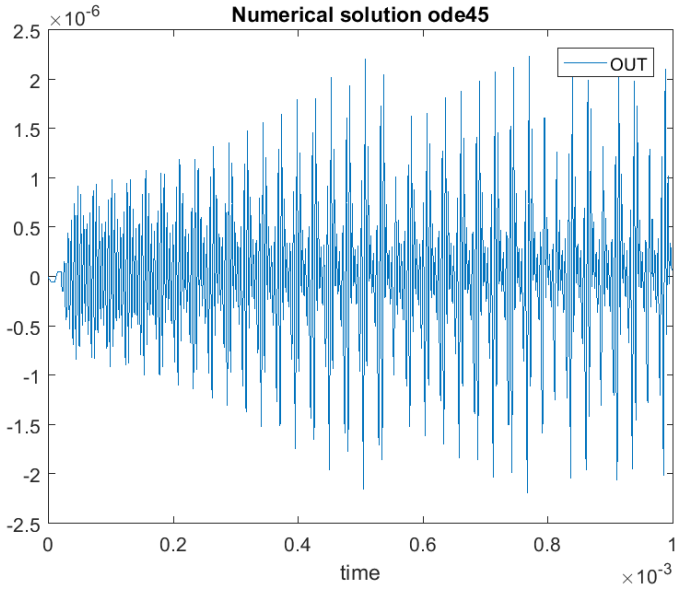

Figure 7.6: Matlab NAND ODE45 solver

7.6 Simulation of NOR in Matlab

In this section we will use again formula with three differential equations as in previous section. To describe NOR circuit we need following equations.

$$u'_{c1} = \frac{1}{C_1}\,(\frac{1}{R_i}(U - u_{c1} - u_{c2} - u_{c34}) - \frac{1}{R_{\bar{A}}}\,U_{c1}) \tag{7.19}$$

$$u'_{c2} = \frac{1}{C_2}\,(\frac{1}{R_i}(U - u_{c1} - u_{c2} - u_{c34}) - \frac{1}{R_{\bar{B}}}\,U_{c2}) \tag{7.20}$$

$$u'_{c34} = \frac{1}{C_{34}}\,(\frac{1}{R_i}\,(U - u_{c1} - u_{c2} - u_{c34}) - \frac{R_A + R_B}{R_A.R_B}\,.\,U_{c34}) \tag{7.21}$$

then we write Jacobian matrix A.

$$A = \begin{pmatrix} -\frac{1}{C_1\,R_i} - \frac{1}{C_1\,.\,R_{\bar{A}}} & -\frac{1}{C_1\,R_i} & -\frac{1}{C_1\,R_i} \\ -\frac{1}{C_2\,R_i} & -\frac{1}{C_2\,R_i} - \frac{1}{C_2\,R_{\bar{B}}} & -\frac{1}{C_2\,R_i} \\ -\frac{1}{C_{34}\,R_i} & -\frac{1}{C_{34}\,R_i} & -\frac{1}{C_{34}R_i} - \frac{R_A+R_B}{C_{34}\,R_A\,R_B} \end{pmatrix}, \qquad \vec{b} = \begin{pmatrix} \frac{u}{C_1\,R_i} \\ \frac{u}{C_2\,R_i} \\ \frac{u}{C_{34}\,R_i} \end{pmatrix} \tag{7.22}$$

We use same simulation logic as in NAND. Efficiency is shown in tables below:

Method	Elapsed time [s]	Number of steps	Absolute error in last step towards Taylor method
Taylor	0.013517	65	X
ODE23	0.017232	281	3.85254e-07
ODE23 Tolerance 1e-10	3.088210	3837	1.77844e-11
ODE45	0.034101	833	2.16721e-08
ODE45 Tolerance 1e-10	0.454832	1521	1.73772e-12

Table 7.9: Matlab NOR methods efficiency for input logical 0 0

Method	Elapsed time [s]	Number of steps	Absolute error in last step towards Taylor method
Taylor	0.012141	65	X
ODE23	0.016074	260	2.29293e-07
ODE23 Tolerance 1e-10	2.818410	3642	5.40116e-14
ODE45	0.026815	769	1.99429e-08
ODE45 Tolerance 1e-10	0.427478	1537	1.73707e-12

Table 7.10: Matlab NOR methods efficiency for input logical 0 1

Method	Elapsed time [s]	Number of steps	Absolute error in last step towards Taylor method
Taylor	0.014111	65	X
ODE23	0.016369	260	2.29293e-07
ODE23 Tolerance 1e-10	2.850956	3642	5.40575e-14
ODE45	0.030344	769	1.99429e-08
ODE45 Tolerance 1e-10	0.436918	1537	1.73703e-12

Table 7.11: Matlab NOR methods efficiency for input logical 1 0

Method	Elapsed time [s]	Number of steps	Absolute error in last step towards Taylor method
Taylor	0.012575	33	X
ODE23	0.014002	201	5.61782e-07
ODE23 Tolerance 1e-10	0.399256	222	7.0787e-11
ODE45	0.022746	657	5.05691e-08
ODE45 Tolerance 1e-10	0.215437	673	9.18822e-12

Table 7.12: Matlab NOR methods efficiency for input logical 1 1

7.7 Simulation of inverter in SPICE

Simulation in SPICE by operating point. When input A is equal to logical 0($RA = 100G\Omega$) then output OUT is equal to logical 1($V(\ out) \ = 3.3V$) as shown in figure 7.7. When input A is equal to logical 1($RA = 1\Omega$) then output OUT is equal to logical 0($V(\ out) \ = 3.3^{-11}V$) as shown in figure A.1.

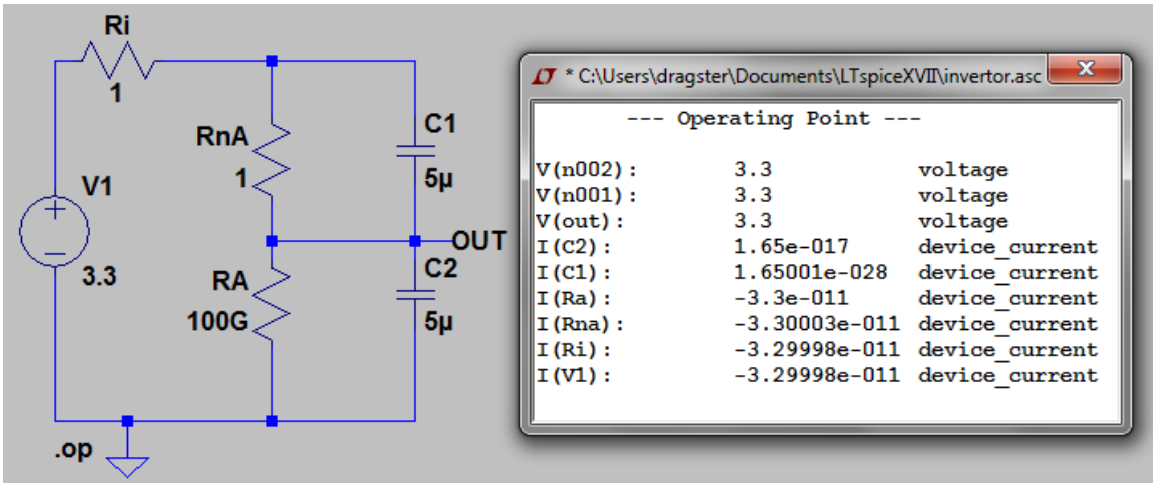

Figure 7.7: Spice Inverter

7.8 Simulation of NAND in SPICE

Simulation in SPICE by operating point. When input A is equal to logical 0($RA = 100G\Omega$) and input B is equal to logical 0($RB = 100G\Omega$) then output OUT is equal

to logical 1($V(\ out)\ = 3.3V$) as shown in figure 7.8. When input A is equal to logical 0($RA = 100G\Omega$) and input B is equal to logical 1($RB = 1\Omega$) then output OUT is equal to logical 1($V(\ out)\ = 3.3V$) as shown in figure A.2. When input A is equal to logical 1($RA = 1\Omega$) and input B is equal to logical 0($RB = 100G\Omega$) then output OUT is equal to logical 1($V(\ out)\ = 3.3V$) as shown in figure A.3. When input A is equal to logical 1($RA = 1\Omega$) and input B is equal to logical 1($RB = 1\Omega$) then output OUT is equal to logical 0($V(\ out)\ = 1.32^{-10}V$) as shown in figure A.4.

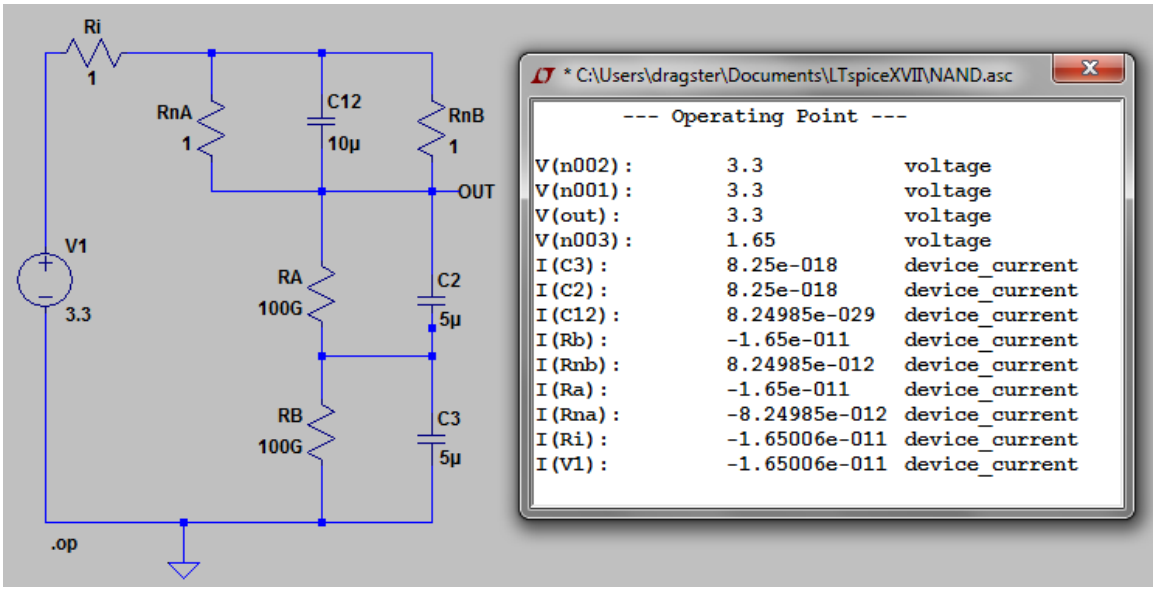

Figure 7.8: Spice NAND

7.9 Simulation of NOR in SPICE

Simulation in SPICE by operating point. When input A is equal to logical 0($RA = 100G\Omega$) and input B is equal to logical 0($RB = 100G\Omega$) then output OUT is equal to logical 1($V(\ out)\ = 3.3V$) as shown in figure 7.9. When input A is equal to logical 0($RA = 100G\Omega$) and input B is equal to logical 1($RB = 1\Omega$) then output OUT is equal to logical 0($V(\ out)\ = 3.3^{-11}V$) as shown in figure A.5. When input A is equal to logical 1($RA = 1\Omega$) and input B is equal to logical 0($RB = 100G\Omega$) then output OUT is equal to logical 0($V(\ out)\ = 3.3^{-11}V$) as shown in figure A.6. When input A is equal to logical 1($RA = 1\Omega$) and input B is equal to logical 1($RB = 1\Omega$) then output OUT is equal to logical 0($V(\ out)\ = 8.25^{-12}V$) as shown in figure A.4.

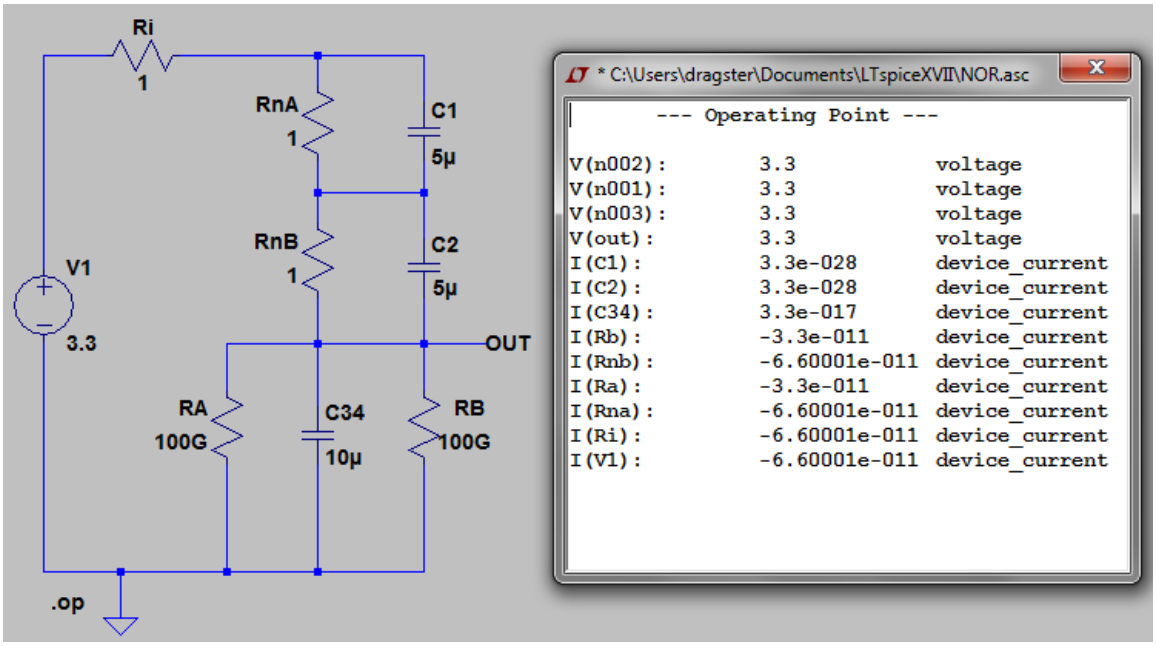

Figure 7.9: Spice NOR

Chapter 8

Conclusion

This thesis is consist of two parts, theoretical and practical. In the first part of this thesis, the numerical solutions of differential equations was described, with special effort of Taylor method, Euler method and Runge-Kutta method. The Taylor method was described more detailed. It includes the parallel-parallel integrator, serial-parallel integrator and serial-serial integrator. In the end of the theoretical part, it was introduced the basic theory around the simulation programs, together with several applications which may be used for simulation and solving differential equations.

The chapter 6 describe the CMOS logic circuits and its solution, which is based on the capacitor substitution. This solution leads to the system of differential equations, which can be solved numerically.

In the final chapter, the experiments was described. TKSL, Matlab and Spice software were used for the simulation. The circuits, which were simulated, includes NAND, NOR and inverter. The efficiency of the simulation was presented using tables which compare elapsed time, number of step and absolute error for each method.

Bibliography

[1] mathworks - Choose an ODE Solver. http://www.mathworks.com/help/matlab/math/choose-an-ode-solver.html. 2010-06-10.

[2] Atkinson, K. E.: *An introduction to numerical analysis*. University of Iowa. second edition. 1989. ISBN 0-471-62489-6. 595 pp.
Retrieved from:
http://www.asd-co.com/files/educational/Kendall%20Atkinson-An%20Introduction%20to%20Numerical%20Analysis-Wiley%20(1989).pdf

[3] Attia, J.: *Electronics and circuit analysis usiong Matlab*. Department of Electrical Engineering - Prairie View A&M University. 1999. ISBN 0-8493-1176-4. 386 pp.

[4] Baker, J.: *CMOS Circuit Design, Layout, and Simulation*. Canada. third edition. 2010. ISBN 978-0-470-88132-3. 966 pp.
Retrieved from:
https://www.u-cursos.cl/usuario/9553d43f5ccbf1cca06cc02562b4005e/mi_blog/r/CMOS_Circuit_Design__Layout__and_Simulation__3rd_Edition.pdf

[5] Bartsch, H.: *Matematické vzorce*. Academia. 2009. ISBN 80-200-1448-9.

[6] Dymola: *Dynamic Modeling Laboratory - Getting started with Dymola*. vol. 1. Scheelevägen 27,Sweden. April 2016. 93 pp.
Retrieved from: http://www.Dymola.com

[7] FREDRIKSSON, E.; FÜHRER, C.: *Modelica and Dymola Introduction*. Lund University. 2016-11-08. 52 pp.

[8] Kocina, F.; Satek, V.; Kunovsky, J.: Taylor Series Based Solution of Linear ODE Systems and MATLAB Solvers Comparison. November 17, 2015: page 2.

[9] Kocina, F.; Satek, V.; Kunovsky, J.; et al.: Modeling VLSI Circuit Using Taylor Series. November 17, 2015: page 4.

[10] Kraus, M.: *Paralelni vypocetni architektury zalozene na numericke integraci*. VUTBR - Department of inteligent systems, Ph.D. Thesis. 2013. 146 pp.

[11] Krupková, V.: *Matematická analýza pro předmět IMA na FIT, FEKT VUT v Brně*. 2008.

[12] KUNOVSKY, J.: *MODERN TAYLOR SERIES METHOD*. Bozetechova 2, 612 66 Brno, The Czech Republic. first edition. 1994. 115 pp.. habilitation work.

[13] Kunovsky, J.: TKSL High Performance Computing. 2002.
Retrieved from: http://www.fit.vutbr.cz/kunovsky/TKSL/tkslc.html

[14] MathWoks: MATLAB - The Language Of Technical Computing. [online]. 2016-MAR-03.
Retrieved from: http://www.mathworks.com/index.html?s_tid=gn_logo

[15] Monagan, M. B.; Geddes, K. O.; Heal, K. M.; et al.: *Maple 10 Programming Guide*. Waterloo ON, Canada: Maplesoft. 2005.

[16] Murthy, A.: *A Simpli ed Introduction to Circuit Simulation using SPICE OPUS*. College of Engineering, Mysore. 17th September 2004. ISBN 0- 07-021152-3. 61 pp.

[17] Young, T.; Mohlenkamp, M. J.: *Introduction to Numerical Methods and Matlab Programming for Engineers*. Department of Mathematics Ohio University. forth edition. May 5, 2015. 180 pp.
Retrieved from: https://www.math.ohiou.edu/courses/math3600/book.pdf

Appendices

List of Appendices

Appendix A

SPICE figures

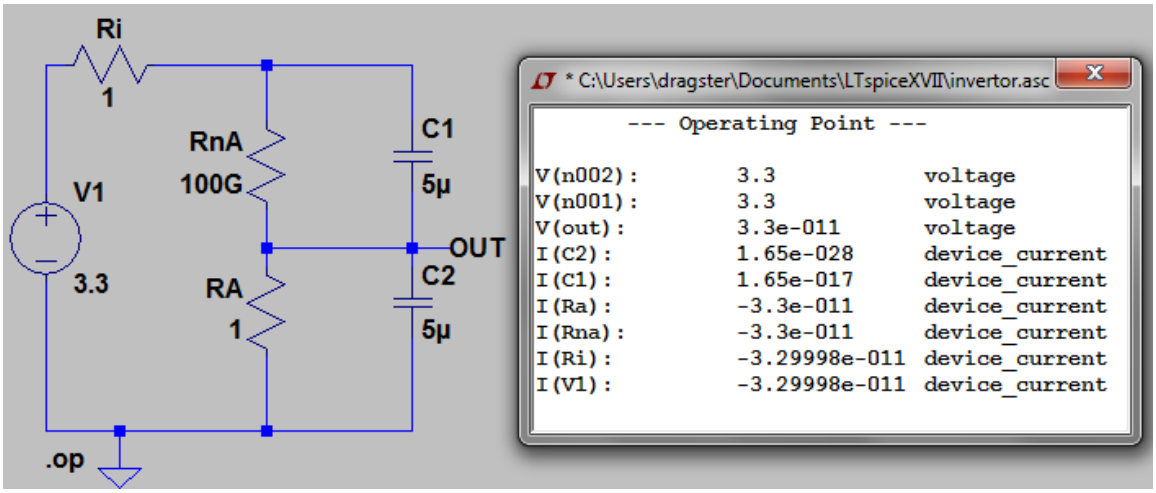

Figure A.1: Spice Inverter

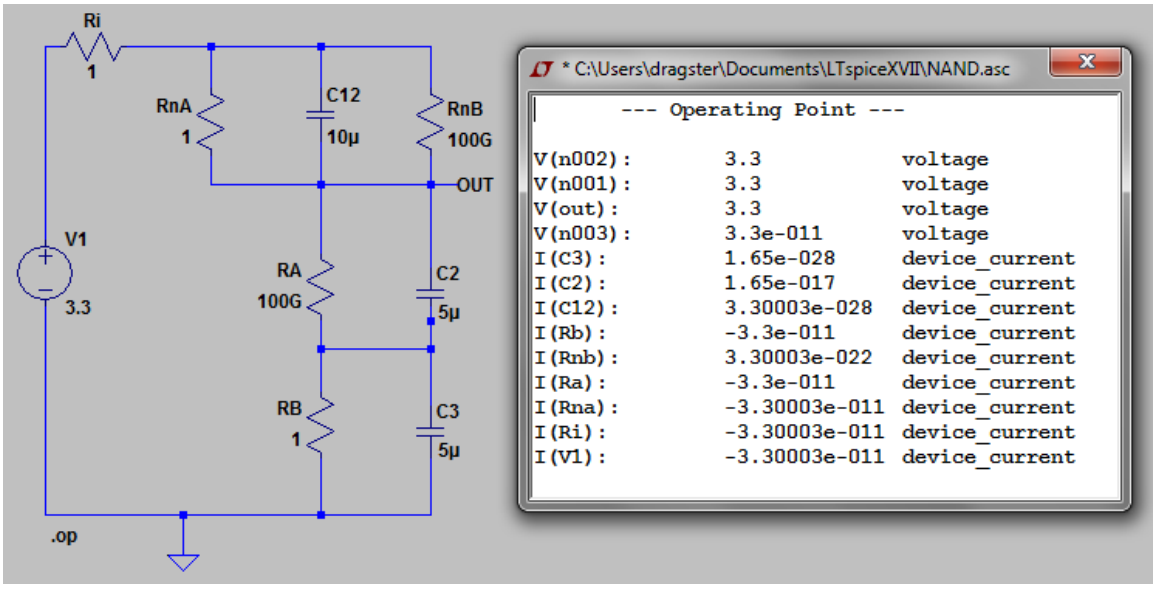

Figure A.2: Spice NAND

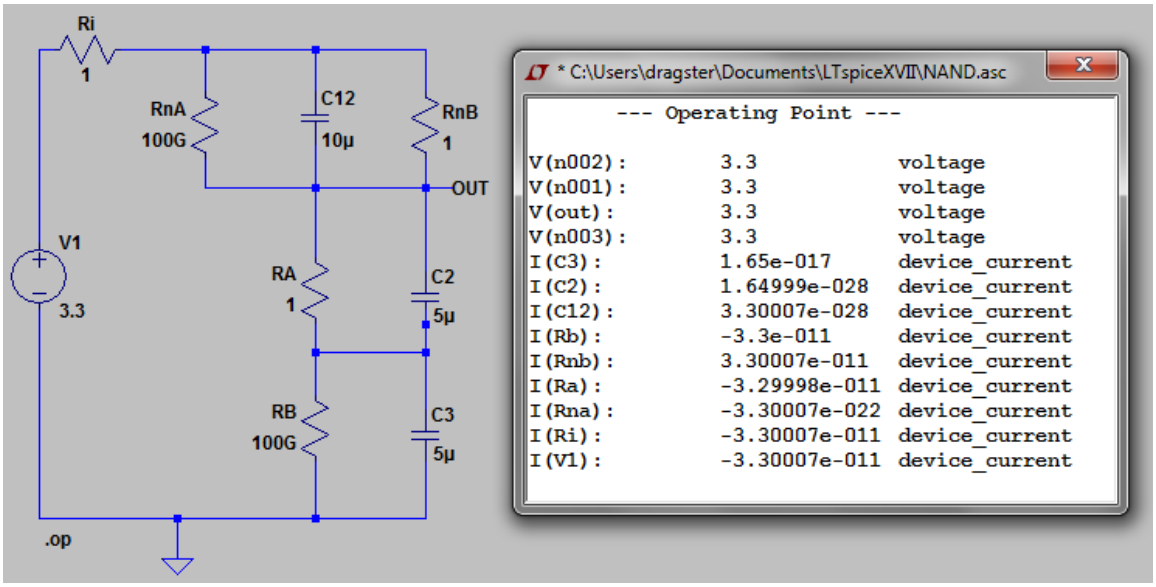

Figure A.3: Spice NAND

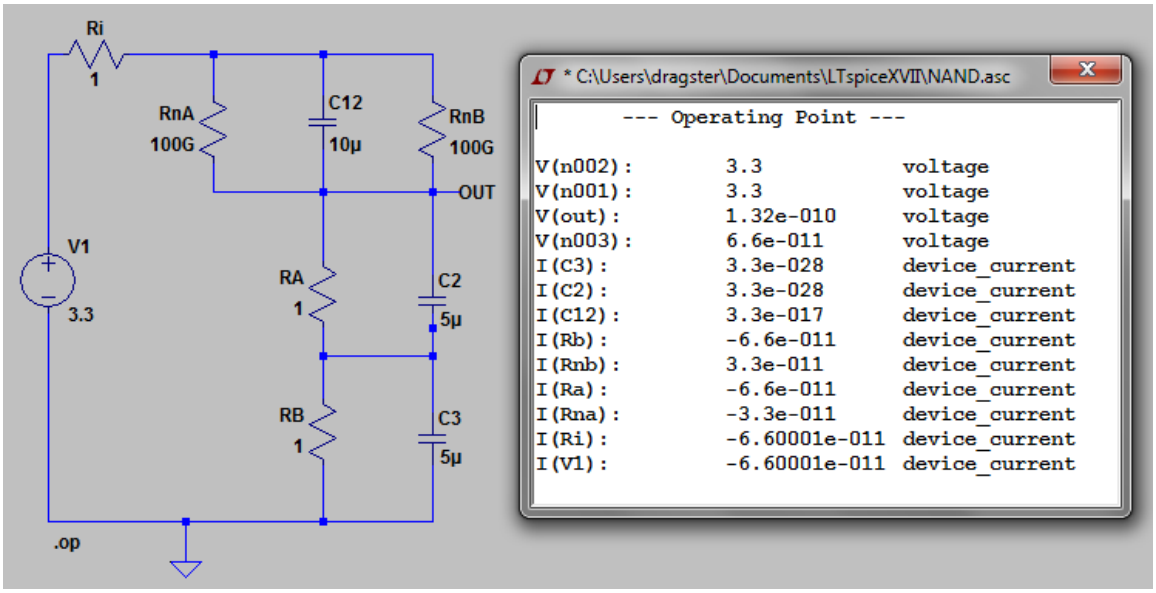

Figure A.4: Spice NAND

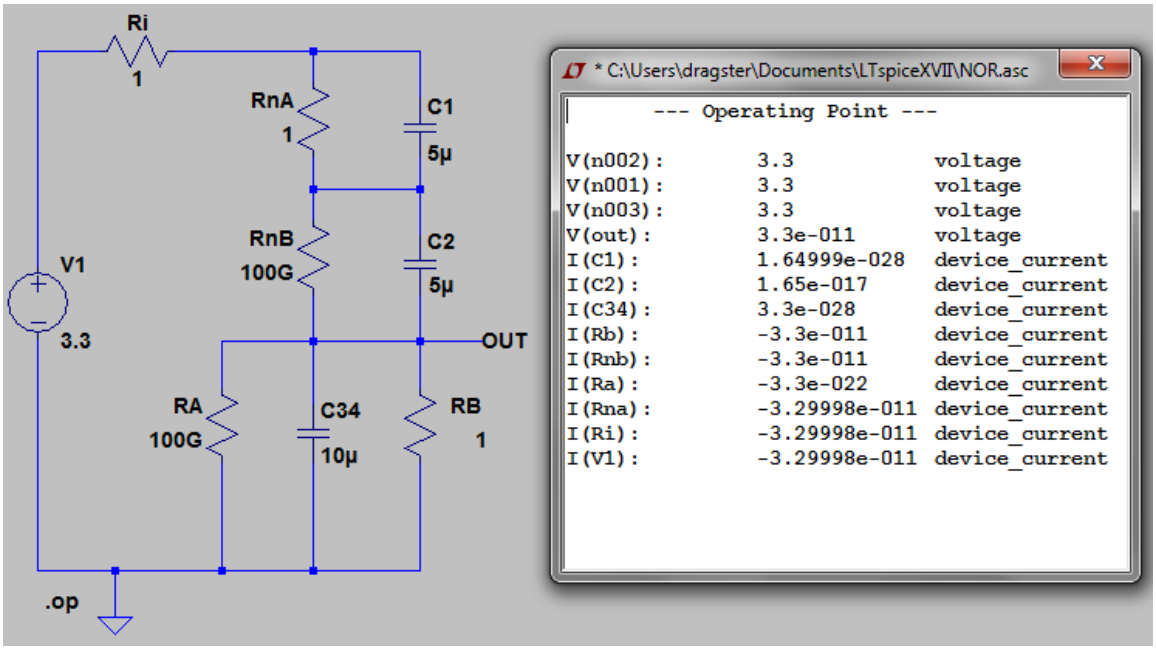

Figure A.5: Spice NOR

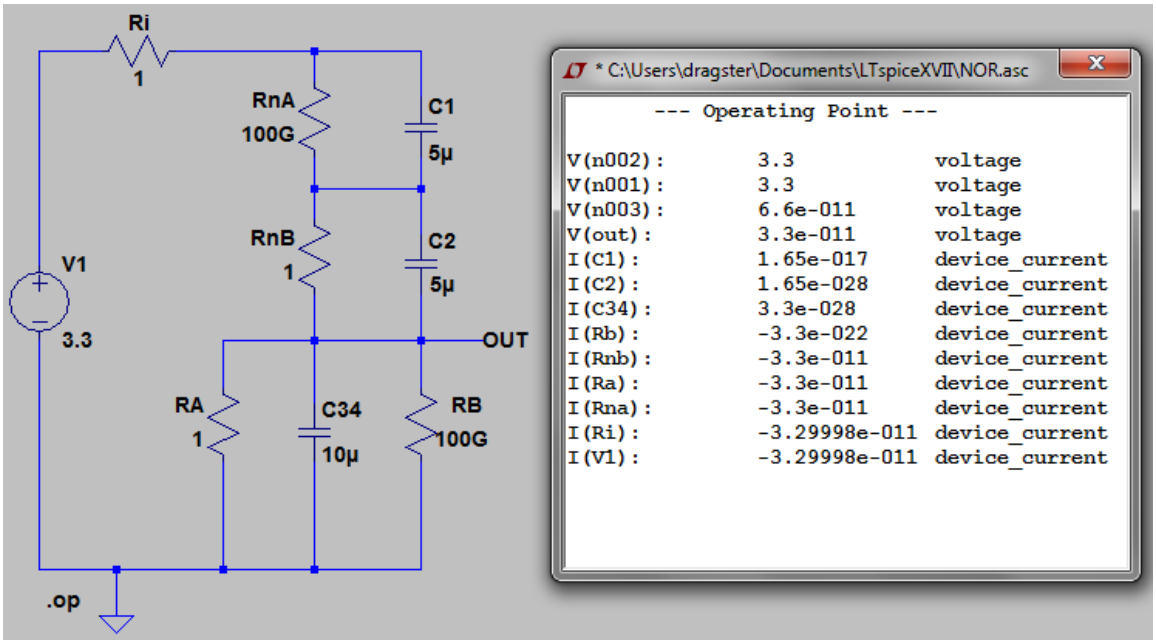

Figure A.6: Spice NOR

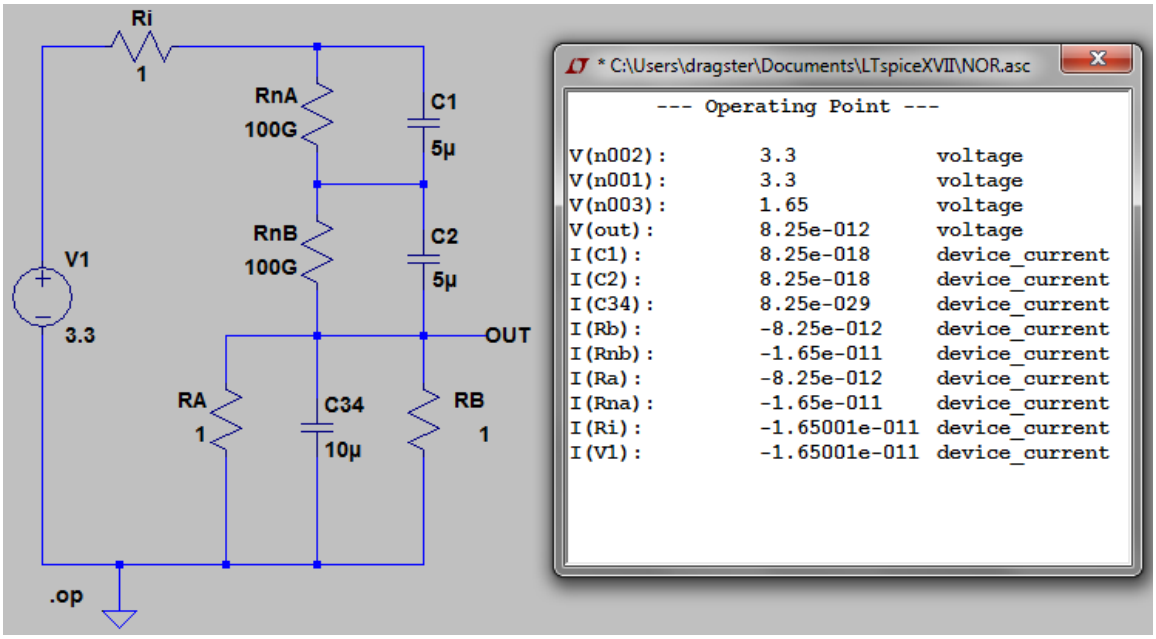

Figure A.7: Spice NOR

Appendix B

MATLAB figures

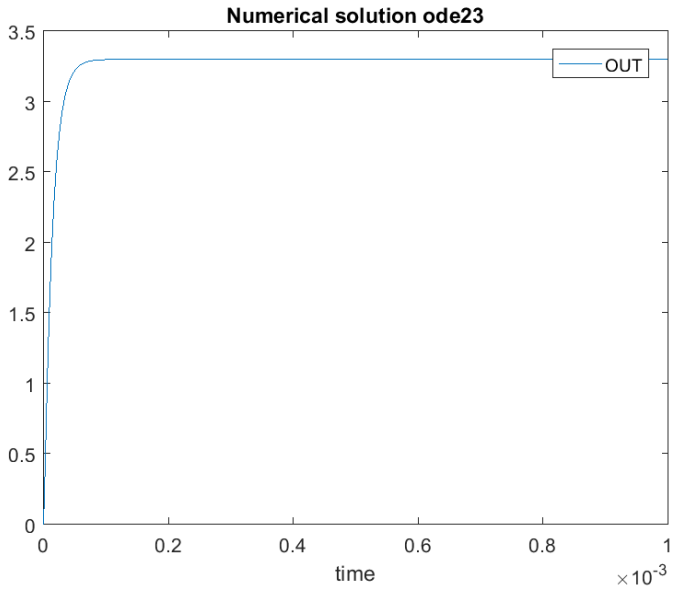

Figure B.1: Matlab invertor ODE23 for input equal to logical 0

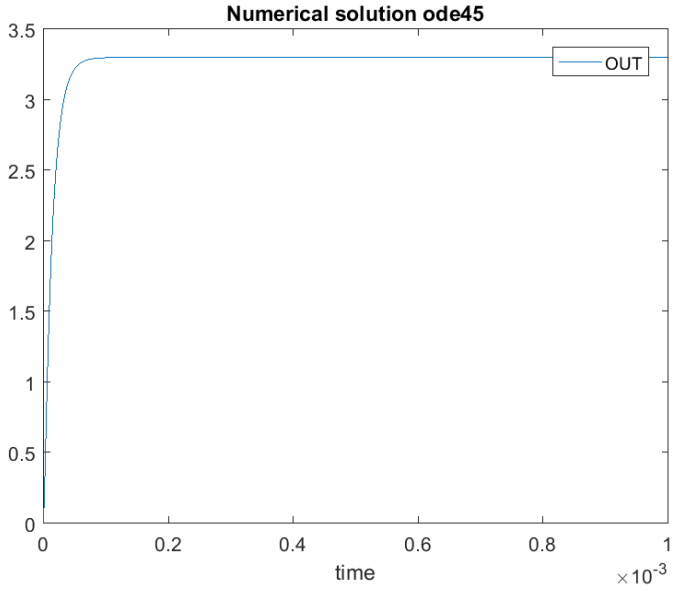

Figure B.2: Matlab invertor ODE45 for input equal to logical 0

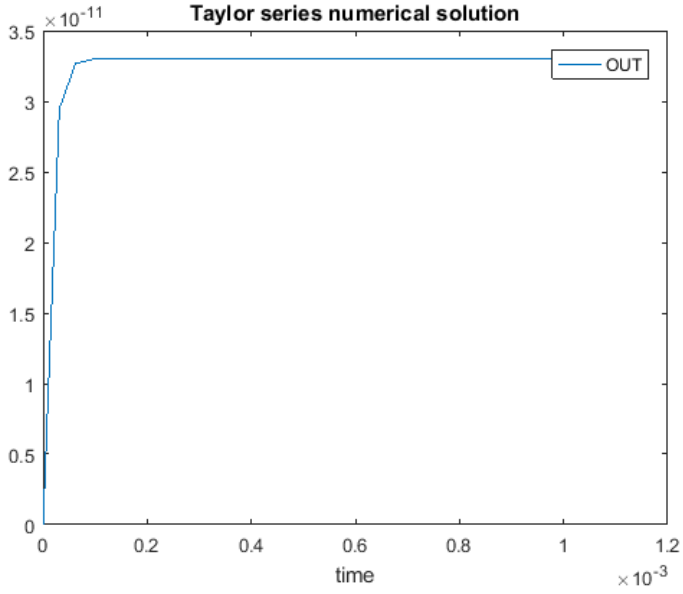

Figure B.3: Matlab invertor Taylor method for input equal to logical 1

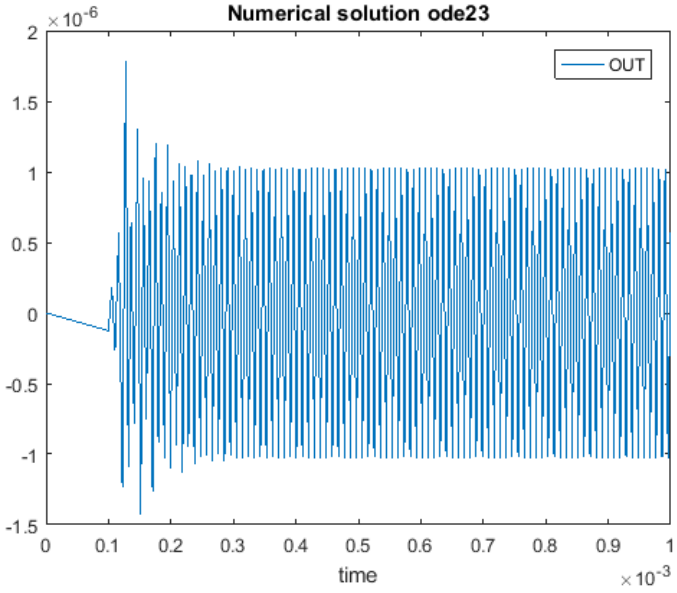

Figure B.4: Matlab invertor ODE23 for input equal to logical 1

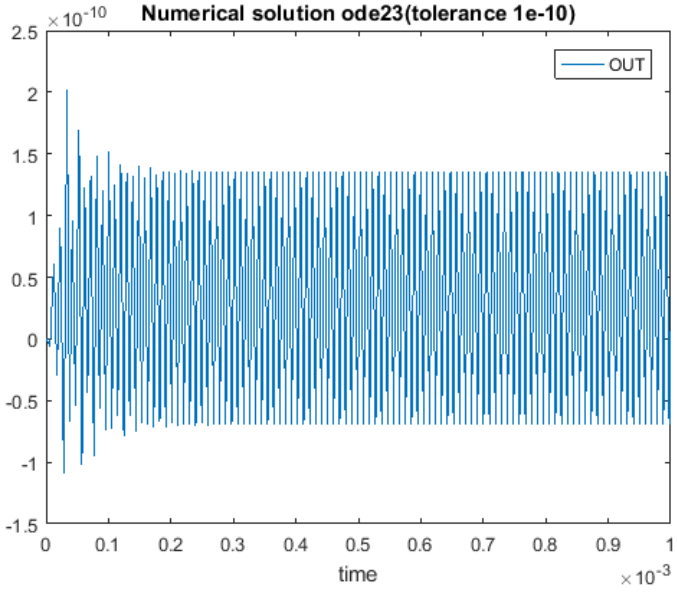

Figure B.5: Matlab invertor ODE23 with tolerance for input equal to logical 1

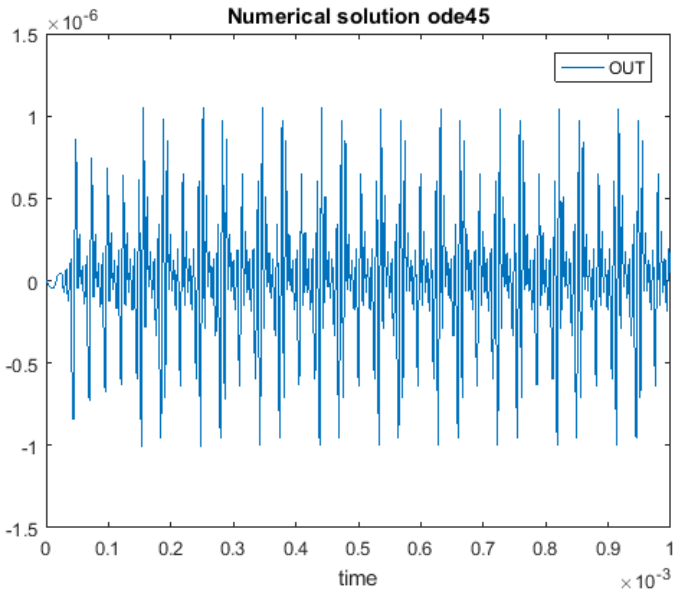

Figure B.6: Matlab invertor ODE45 for input equal to logical 1

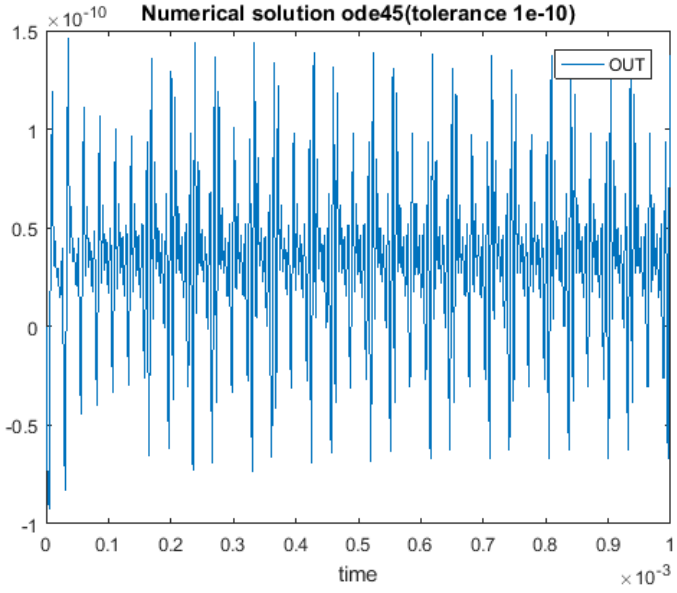

Figure B.7: Matlab invertor ODE45 with tolerance for input equal to logical 1