BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

ADVANCED ELECTRONIC CIRCUITS SIMULATION METHODS

MODERNÍ METODY MODELOVÁNÍ A SIMULACE ELEKTRONICKÝCH OBVODŮ

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR
AUTOR PRÁCE

Ing. FILIP KOCINA

SUPERVISOR
ŠKOLITEL

doc. Ing. JIŘÍ KUNOVSKÝ, CSc.

BRNO 2017

Abstract

The thesis deals with the simulation of electronic circuits. It describes the Capacitor Substitution Method (CSM) to transform electronic circuits into electric circuits which can then be solved using numerical methods, namely the Modern Taylor Series Method (MTSM). This method is distinguished by automatic order selection, halving the step size as required and the wide area of stability according to the order. Within the thesis, specialized programming equipment to solve ordinary differential equations using MTSM was created by the author of the thesis, with many improvements to the algorithms (compared to TKSL/386). These algorithms involve the simplification of generic expressions into polynomials, parallelization independent of the integration method etc. This software runs on a Linux server which communicates using the TCP/IP stack. The equipment was successfully used to simulate VLSI circuits whose solution by CSM was much faster and more memory-efficient than the state-of-the-art SPICE.

Abstrakt

Disertační práce se zabývá simulací elektronických obvodů. Popisuje metodu kapacitorové substituce (CSM) pro převod elektronických obvodů na elektrické obvody, jež mohou být následně řešeny pomocí numerických metod, zejména Moderní metodou Taylorovy řady (MTSM). Tato metoda se odlišuje automatickým výběrem řádu, půlením kroku v případě potřeby a rozsáhlou oblastí stability podle zvoleného řádu. V rámci disertační práce bylo autorem disertace vytvořeno specializované programové vybavení pro řešení obyčejných diferenciálních rovnic pomocí MTSM, s mnoha vylepšeními v algoritmech (v porovnání s TKSL/386). Tyto algoritmy zahrnují zjednodušování obecných výrazů na polynomy, paralelizaci nezávislou na integrační metodě atp. Tento software běží na linuxovém serveru, který komunikuje pomocí protokolu TCP/IP. Toto vybavení bylo úspěšně použito pro simulaci VLSI obvodů, jejichž řešení pomocí CSM bylo značně rychlejší a spotřebovávalo méně paměti než state-of-the-art SPICE.

Keywords

Modern Taylor Series Method, Capacitor Substitution Method, ordinary differential equations, electronic circuits, logic gates, inverter, NAND, NOR, XOR, RS latch, D latch, JK flip-flop, T flip-flop, binary adder, Booth's algorithm, VLSI.

Klíčová slova

Moderní metoda Taylorovy řady, metoda kapacitorové substituce, obyčejné diferenciální rovnice, elektronické obvody, logická hradla, invertor, NAND, NOR, XOR, RS klopný obvod, D klopný obvod, JK klopný obvod, T klopný obvod, binární sčítačka, Boothův algoritmus, VLSI.

Reference

KOCINA, Filip. *Advanced Electronic Circuits Simulation Methods.* Brno, 2017. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Kunovský Jiří.

Advanced Electronic Circuits Simulation Methods

Declaration

I hereby declare that this PhD thesis was prepared as an original author's work under the supervision of Assoc. Prof. Jiří Kunovský. All the relevant information sources which were used during the preparation of this thesis are properly cited and included in the list of references.

.....
Filip Kocina
June 28, 2017

Acknowledgements

I would like to thank Assoc. Prof. Jiří Kunovský for his very kind, valuable and insightful advice which helped me to create this thesis. Many thanks belong to my family and friends who supported me throughout my studies, although they might not understand all the details of my work; it would have been harder to complete the thesis without their encouragement.

Contents

List of Figures

List of Tables

Nomenclature

y'	Time Derivative ($\dot{y}$)
CLA	Carry Look-ahead
CLU	Carry Look-ahead Unit
CMOS	Complementary Metal–Oxide–Semiconductor
CNF	Conjunctive Normal Form
CPU	Central Processing Unit
CSM	Capacitor Substitution Method
DNF	Disjunctive Normal Form
EPS	Error Per Step
FOS	Fast ODE Solver
ILA	Invert Look-ahead
ILU	Invert Look-ahead Unit
LSB	Least Significant Bit
LSI	Large-Scale Integration
MSB	Most Significant Bit
MSI	Medium-Scale Integration
MTSM	Modern Taylor Series Method
NMOS	Negative Metal–Oxide–Semiconductor
ODE	Ordinary Differential Equation
ORD	Order of Method
PMOS	Positive Metal–Oxide–Semiconductor
SSI	Small-Scale Integration
TMAX	Maximum Simulation Time
ULSI	Ultra Large-Scale Integration
VLSI	Very Large-Scale Integration

Chapter 1

Introduction

Many real-world problems lead to large systems of ordinary differential equations (ODEs). These systems cannot be solved analytically; therefore, numerical methods are involved. Many numerical methods exist, differing in complexity, accuracy, speed and flexibility. Some methods can be substituted using superior variants, while others are used together with more sophisticated optimizations for a specific purpose.

In this thesis, some methods are mentioned, but the main subject of interest is a very precise, fast and flexible method that uses the Taylor series. The method can solve many technical initial-value problems. This method is used in the software I developed to compute large systems of differential equations. The software runs on a Linux server, accepting the tasks using the TCP/IP stack.

The main part of the thesis is devoted to electric/electronic circuits simulation. The electric circuits discussed contain only resistors, capacitors and coils, while electronic circuits also include semiconductors like diodes and transistors. Both the diode and the transistor are represented using their exponential characteristics. The Capacitor Substitution Method (CSM) developed is used for the simulation of transistors. Further, the simulation of various electronic components is proposed.

CSM is much faster and more memory-efficient than the state-of-the-art SPICE. Relatively large Very Large-Scale Integration (VLSI) circuits (over a million transistors) have been successfully simulated in less than four minutes. For example, multiple-bit adders and multipliers are used. These circuits are simulated using both CSM and SPICE and the results are compared. The simulation of the multipliers is relatively slow when compared to the adders, since more algorithmic cycles have to be simulated.

1.1 Motivation

The simulation of electronic circuits is still a challenging problem. The simulation of VLSI circuits is complicated and time-consuming using the existing software, which implies inconvenience for everyday usage. This is due to the precise simulation of the individual transistors that is performed. This simulation uses a large amount of resources.

Another approach to the simulation of electronic circuits is to consider primarily the steady state of the transistor and the length of the transient response. The rest of the behavior (and possible errors) can be ignored. The approach benefits from the fact that most of the time only the length of the transient response is required and it is irrelevant

that the error during the transient response is relatively high. This approach is the main subject of the thesis.

1.2 Aims

This thesis deals with three research hypotheses:

- The equations describing an electronic circuit can be systematically created.
- The transistors could be replaced by RC circuits.
- The proposed method should be efficient.

1.3 Overview of the work

The thesis is divided into the following chapters. The first chapter introduces the thesis.

The second chapter focuses on systems of differential–algebraic equations and mentions some numerical methods which can be used for solving ordinary differential equations (ODEs), especially the Modern Taylor Series Method (MTSM). After a brief overview of the methods, the automatic transformation used for MTSM is explained and the transformation into the minimal form is introduced. This transformation can be performed on all elementary functions (trigonometric, inverse trigonometric, hyperbolic, inverse hyperbolic, exponential function, natural logarithm and square root) and the basic mathematical operations (addition, subtraction, multiplication and division); the minimal form is then a polynomial of the variables.

The third chapter discusses the characteristics of MTSM – accuracy (leading to a need for arbitrary precision arithmetic), the speed of calculation (based on accuracy and the step size), stiff systems (that require implicit methods) and the stopping rule (some drawbacks of the default chosen one). The principle of calculating MTSM terms and the practical usage of MTSM are mentioned at the end of the chapter.

Electric circuits and their solution are discussed in the fourth chapter. These circuits and methods are used throughout the rest of the thesis. Symbolic and numerical solutions are compared and improvements in the acceleration of the transient response are shown. After that, a few parallel methods are mentioned and the acceleration of various approaches to solving the telegraph line is presented.

The fifth chapter builds upon the fourth chapter and focuses on solving electronic circuits. A few methods to solve the diode and the transistor are analyzed. Complementary Metal–Oxide–Semiconductor (CMOS) technology is chosen since it is the most widely used technology in electronic circuits. The main contribution of this thesis is the Capacitor Substitution Method (CSM) which is proposed later in the chapter. This method was implemented by the author of the thesis in a general-purpose programming language. Various electronic circuits are analyzed using this method: basic CMOS gates (inverter, NAND, NOR) and XOR. The results were compared with the state-of-the-art SPICE. The method can be used to simulate arbitrary circuits consisting of the gates.

The sixth chapter focuses on VLSI circuits simulation and shows the experiments and comparison between CSM and SPICE. The CMOS latches and flip-flops are analyzed; the most important are experiments with multiple-bit adders and multipliers. The experiments confirm that CSM is very suitable for VLSI circuits simulations. CLA adders up to the

16kb adder (over a million transistors) and 256-bit multiplier (using Booth's algorithm) were successfully simulated. The effectiveness of CSM is compared to SPICE and the results show that CSM is much faster and uses fewer resources. The chapter also analyzes the potential of an easy machine representation of any electronic circuit – by an adjacency matrix and a vector of operations (nodes of the graph). This representation can be automatically generated by some tool for electronic circuits design.

The last chapter concludes the thesis. It summarizes the aims achieved and outlines the possibilities for future research. At the end of the chapter, there is an overview of my doctoral work. A list of publications is presented after this chapter.

Chapter 2

Differential–Algebraic Equations

A large number of technical problems can be described using a system of ordinary differential and algebraic equations [17, 18]. These systems can be formally written as

$$\begin{aligned} w_1' &= f_1(w_1, \ldots, w_n, x_1, \ldots, x_m), \quad w_1(t_0) = w_1^0 \\ &\vdots \qquad\qquad\qquad\qquad\qquad\qquad\quad \vdots \\ w_n' &= f_n(w_1, \ldots, w_n, x_1, \ldots, x_m), \quad w_n(t_0) = w_n^0 \\ x_1 &= g_1(w_1, \ldots, w_n, x_1, \ldots, x_m) \\ &\vdots \\ x_m &= g_m(w_1, \ldots, w_n, x_1, \ldots, x_m) \end{aligned} \tag{2.1}$$

consisting of n ordinary differential equations and m algebraic equations. Few systems can be solved analytically; therefore, numerical methods are most commonly used to find the solution [11, 16].

2.1 Numerical methods

Various numerical methods can be used to solve ordinary differential equations (ODEs). The methods for solving algebraic equations are not mentioned since they are not required for the proposed method of solving electronic circuits.

Initial-value problems described by ODEs can be solved using many different methods. Let (2.2) specify the initial-value problem.

$$y' = f(t, y), \quad y(t_0) = y_0 \tag{2.2}$$

Then the problem can be solved by a numerical method – several methods are mentioned in the following subsections.

2.1.1 Euler method

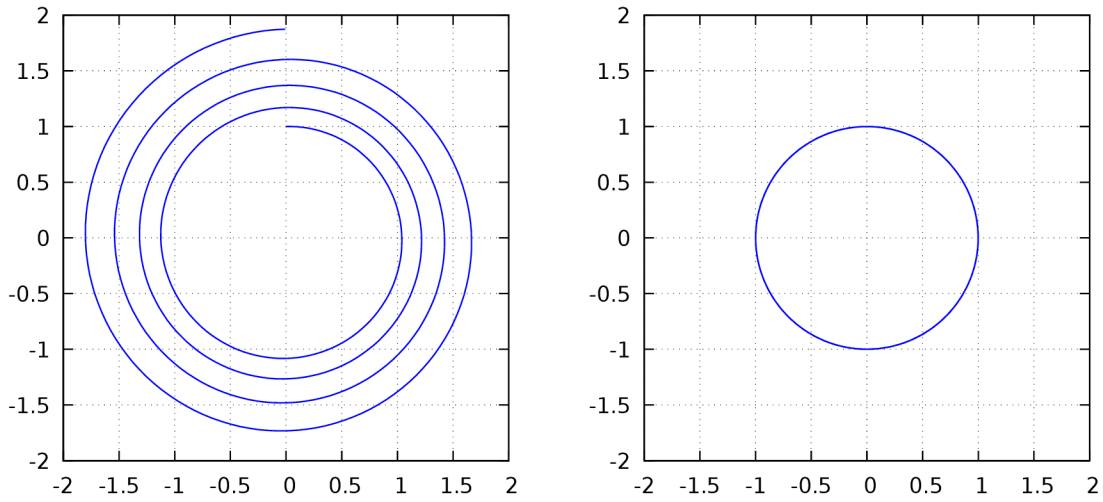
The simplest numerical method for solving ordinary differential equations is the explicit Euler method. It is a special form of the explicit Taylor method of the first order:

$$y_{n+1} = y_n + h \cdot f(t_n, y_n). \tag{2.3}$$

The simplicity of the Euler method unfortunately implies very low accuracy. Step size h has to be small for more precise results and the method is completely unusable for stiff systems. The circle test can be used as an easy demonstration of the poor precision of the Euler method:

$$y'' = -y, \quad y(0) = 0, \quad y'(0) = 1. \tag{2.4}$$

Figure 2.1 shows the problem clearly[1] (the source code can be found in Appendix A.1). Some drawbacks of the explicit Euler method can be removed by the implicit form of the Euler method [23].



(a) Euler method

(b) Exact solution

Figure 2.1: Circle test

2.1.2 Runge–Kutta methods

Runge–Kutta methods are commonly used for solving initial-value problems. These methods are often chosen in many technical branches, mainly for solving non-stiff problems. The next value is calculated by

$$y_{n+1} = y_n + \sum_{i=1}^{ORD} w_i k_i \tag{2.5}$$

where weights w_i are constant and coefficients k_i are calculated by (2.6); function $f(t, y)$ is the right side of the solved ordinary differential equation.

$$k_i = h_n \cdot f\left(t_n + \alpha_i h_n,\ y_n + \sum_{j=1}^{i-1} \beta_{ij} k_j\right) \tag{2.6}$$

Weights w_i, vector α and matrix β determine a specific method – they are often derived from the Taylor series [6, 7]. Step size h_n can be variable, but this is rare. The following

[1] Dependency of y' on y with step size 0.05 is shown.

fourth-order method (2.7) appears to be the most frequent (with constant step size h), see [8, 38].

$$\begin{aligned} k_1 &= h \cdot f(t_n, y_n) \\ k_2 &= h \cdot f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \\ k_3 &= h \cdot f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right) \\ k_4 &= h \cdot f(t_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \frac{1}{6}\Big(k_1 + 2k_2 + 2k_3 + k_4\Big) \end{aligned} \tag{2.7}$$

2.1.3 Modern Taylor Series Method

The Modern Taylor Series Method (MTSM) uses not only the first derivative for calculating the next value, but also higher derivatives. These derivatives are obtained by consequent differentiating the previous derivatives (the right side of the equation is the first derivative) [2, 33]. The value in every point is obtained by their combination (2.8).

$$y_{n+1} = y_n + \sum_{i=1}^{ORD_n} \frac{y_n^{(i)} h_n^i}{i!} \tag{2.8}$$

In practice, it is impossible to use an infinite sum of MTSM terms. The number of terms is determined by the order of the method (ORD_n). Contrary to the previous methods, it is possible to choose any order: the higher the order chosen, the more accurate the solution calculated. The MTSM order changes automatically during the calculation; the calculation in the current time step ends when the stopping rule is met: the absolute values of three successive MTSM terms are less than the required accuracy (`EPS`). Although higher orders allow the use of a bigger step size, multiple-precision arithmetic has to be often used in that case; otherwise, the results would not be accurate.

As an example, the calculation of Euler's number can be shown. By solving the differential equation

$$y' = y, \quad y(0) = 1, \tag{2.9}$$

Euler's number can be obtained in one time step with precision up to 10 000 decimal positions in less than one second (in one step with order 3 278). See [26] for more details.

2.2 Automatic transformation

The most important part of MTSM is the automatic transformation. It is the cornerstone of solving ordinary differential equations using this method. This transformation is performed automatically in the software. Arbitrary precision arithmetic is important for the automatic transformation; it uses more bits for numbers (typically hundreds and more), unlike the built-in number types (`double`, `float`, etc.)

Function $x(t)$ is assumed to be smooth, with derivative $x'(t) = u(t)$; both $x(t)$ and $u(t)$ should already be in polynomial form (transformed recursively using the rules that follow). Then we can derive transformed expressions for every elementary function. First, a derivative for the simple argument t is shown; further, an arbitrarily complex argument is used.

2.2.1 Trigonometric functions

Function `sin`

$$\sin'(t) \;=\; \cos(t)$$

After the differentiation of

$$y(t) \;=\; \sin(x(t)), \tag{2.10}$$

we obtain

$$y'(t) \;=\; x'(t) \cdot \cos(x(t)), \quad y(t_0) \;=\; \sin(x(t_0))$$

where $x'(t)$ is substituted by $u(t)$ and $\cos(x(t))$ by $v(t)$:

$$y'(t) \;=\; u(t) \cdot v(t), \quad y(t_0) \;=\; \sin(x(t_0));$$

expression $v(t)$ is differentiated, $\sin(t)$ substituted by $y(t)$ and we obtain

$$v'(t) \;=\; -u(t) \cdot y(t), \quad v(t_0) \;=\; \cos(x(t_0)).$$

Hence, the resulting transformed system of ODEs is (2.11).

$$\begin{aligned} y'(t) \;&=\; u(t) \cdot v(t), & y(t_0) \;&=\; \sin(x(t_0)) \\ v'(t) \;&=\; -u(t) \cdot y(t), & v(t_0) \;&=\; \cos(x(t_0)) \end{aligned} \tag{2.11}$$

Function `cos`

$$\cos'(t) \;=\; -\sin(t)$$

The transformation of $\cos(x(t))$ is performed in a similar manner:

$$\begin{aligned} y(t) \;&=\; \cos(x(t)) \\ y'(t) \;&=\; -x'(t) \cdot \sin(x(t)), & y(t_0) \;&=\; \cos(x(t_0)) \end{aligned} \tag{2.12}$$

and after substituting $\sin(x(t))$ and $x'(t)$, we obtain the system of ODEs (2.13).

$$\begin{aligned} y'(t) \;&=\; -u(t) \cdot v(t), & y(t_0) \;&=\; \cos(x(t_0)) \\ v'(t) \;&=\; u(t) \cdot y(t), & v(t_0) \;&=\; \sin(x(t_0)) \end{aligned} \tag{2.13}$$

Function `tan`

$$\tan'(t) \;=\; 1 + \tan^2(t)$$

The transformation of $\tan(x(t))$ is analogical:

$$\begin{aligned} y(t) \;&=\; \tan(x(t)) \\ y'(t) \;&=\; x'(t) \cdot (1 + y^2(t)), & y(t_0) \;&=\; \tan(x(t_0)) \end{aligned} \tag{2.14}$$

and the result is (2.15).

$$y'(t) \;=\; u(t) \cdot (1 + y^2(t)), \quad y(t_0) \;=\; \tan(x(t_0)) \tag{2.15}$$

Function `cot`

$$\cot'(t) \;=\; -(1+\cot^2(t))$$

The transformation of $\cot(x(t))$ is almost the same as the transformation of $\tan(x(t))$:

$$\begin{aligned} y(t) &= \cot(x(t)) \\ y'(t) &= -x'(t)\cdot(1+y^2(t)), \quad y(t_0) \;=\; \cot(x(t_0)) \end{aligned} \tag{2.16}$$

and the final form is

$$y'(t) \;=\; -u(t)\cdot(1+y^2(t)), \quad y(t_0) \;=\; \cot(x(t_0)). \tag{2.17}$$

2.2.2 Inverse trigonometric functions

Function `arcsin`

$$\arcsin'(t) \;=\; \frac{1}{\sqrt{1-t^2}}$$

The transformation of $\arcsin(x(t))$ is more complicated than the previous transformations:

$$\begin{aligned} y(t) &= \arcsin(x(t)) \\ y'(t) &= \frac{x'(t)}{\sqrt{1-x^2(t)}}, \quad y(t_0) \;=\; \arcsin(x(t_0)). \end{aligned} \tag{2.18}$$

The result is:

$$\begin{aligned} y'(t) &= \frac{u(t)}{v(t)}, & y(t_0) &= \arcsin(x(t_0)) \\ v'(t) &= -\frac{x(t)\cdot u(t)}{v(t)}, & v(t_0) &= \sqrt{1-x^2(t_0)}. \end{aligned} \tag{2.19}$$

Function `arccos`

$$\arccos'(t) \;=\; -\frac{1}{\sqrt{1-t^2}}$$

The transformation of $\arccos(x(t))$ is analogical:

$$\begin{aligned} y(t) &= \arccos(x(t)) \\ y'(t) &= -\frac{x'(t)}{\sqrt{1-x^2(t)}}, \quad y(t_0) \;=\; \arccos(x(t_0)) \end{aligned} \tag{2.20}$$

and the result is:

$$\begin{aligned} y'(t) &= -\frac{u(t)}{v(t)}, & y(t_0) &= \arccos(x(t_0)) \\ v'(t) &= -\frac{x(t)\cdot u(t)}{v(t)}, & v(t_0) &= \sqrt{1-x^2(t_0)}. \end{aligned} \tag{2.21}$$

Function `arctan`

$$\arctan'(t) \;=\; \frac{1}{1+t^2}$$

The transformation of $\arctan(x(t))$ is a little simpler:

$$\begin{aligned} y(t) &= \arctan(x(t)) \\ y'(t) &= \frac{x'(t)}{1+x^2(t)}, \qquad y(t_0) = \arctan(x(t_0)) \end{aligned} \tag{2.22}$$

and the final form:

$$y'(t) \;=\; \frac{u(t)}{1+x^2(t)}, \quad y(t_0) \;=\; \arctan(x(t_0)). \tag{2.23}$$

Function `arccot`

$$\operatorname{arccot}'(t) \;=\; -\frac{1}{1+t^2}$$

The transformation of $\operatorname{arccot}(x(t))$ differs from the transformation of $\arctan(x(t))$ only by the minus sign and the initial condition:

$$\begin{aligned} y(t) &= \operatorname{arccot}(x(t)) \\ y'(t) &= -\frac{x'(t)}{1+x^2(t)}, \quad y(t_0) = \operatorname{arccot}(x(t_0)) \end{aligned} \tag{2.24}$$

and the result:

$$y'(t) \;=\; -\frac{u(t)}{1+x^2(t)}, \quad y(t_0) \;=\; \operatorname{arccot}(x(t_0)). \tag{2.25}$$

2.2.3 Hyperbolic functions

Function `sinh`

$$\sinh'(t) \;=\; \cosh(t)$$

The transformation of hyperbolic sine is similar to trigonometric sine, except for the minus sign and the initial conditions:

$$\begin{aligned} y(t) &= \sinh(x(t)) \\ y'(t) &= x'(t)\cdot\cosh(x(t)), & y(t_0) &= \sinh(x(t_0)) \\ y'(t) &= u(t)\cdot v(t), & y(t_0) &= \sinh(x(t_0)) \\ v'(t) &= u(t)\cdot y(t), & v(t_0) &= \cosh(x(t_0)). \end{aligned} \tag{2.26}$$

Function `cosh`

$$\cosh'(t) \;=\; \sinh(t)$$

Analogically:

$$\begin{aligned} y(t) &= \cosh(x(t)) \\ y'(t) &= x'(t)\cdot\sinh(x(t)), & y(t_0) &= \cosh(x(t_0)) \\ y'(t) &= u(t)\cdot v(t), & y(t_0) &= \cosh(x(t_0)) \\ v'(t) &= u(t)\cdot y(t), & v(t_0) &= \sinh(x(t_0)). \end{aligned} \tag{2.27}$$

Function `tanh`

$$\tanh'(t) \;=\; 1 - \tanh^2(t)$$

Similarly:

$$\begin{aligned} y(t) &= \tanh(x(t)) \\ y'(t) &= x'(t) \cdot (1 - y^2(t)), \quad y(t_0) = \tanh(x(t_0)) \\ y'(t) &= u(t) \cdot (1 - y^2(t)), \quad y(t_0) = \tanh(x(t_0)). \end{aligned} \tag{2.28}$$

Function `coth`

$$\coth'(t) \;=\; 1 - \coth^2(t)$$

Almost the same as $\tanh(t)$:

$$\begin{aligned} y(t) &= \coth(x(t)) \\ y'(t) &= x'(t) \cdot (1 - y^2(t)), \quad y(t_0) = \coth(x(t_0)) \\ y'(t) &= u(t) \cdot (1 - y^2(t)), \quad y(t_0) = \coth(x(t_0)). \end{aligned} \tag{2.29}$$

2.2.4 Inverse hyperbolic functions

Function `argsinh`

$$\operatorname{argsinh}'(t) \;=\; \frac{1}{\sqrt{1+t^2}}$$

The transformation of inverse hyperbolic sine is similar to inverse trigonometric sine, except for the minus sign and the initial conditions:

$$\begin{aligned} y(t) &= \operatorname{argsinh}(x(t)) \\ y'(t) &= \frac{x'(t)}{\sqrt{1+x^2(t)}}, \quad y(t_0) = \operatorname{argsinh}(x(t_0)) \\ y'(t) &= \frac{u(t)}{v(t)}, \quad y(t_0) = \operatorname{argsinh}(x(t_0)) \\ v'(t) &= \frac{x(t) \cdot u(t)}{v(t)}, \quad v(t_0) = \sqrt{1+x^2(t_0)}. \end{aligned} \tag{2.30}$$

Function `argcosh`

$$\operatorname{argcosh}'(t) \;=\; \frac{1}{\sqrt{t^2-1}}$$

Analogically:

$$\begin{aligned} y(t) &= \operatorname{argcosh}(x(t)) \\ y'(t) &= \frac{x'(t)}{\sqrt{x^2(t)-1}}, \quad y(t_0) = \operatorname{argcosh}(x(t_0)) \\ y'(t) &= \frac{u(t)}{v(t)}, \quad y(t_0) = \operatorname{argcosh}(x(t_0)) \\ v'(t) &= \frac{x(t) \cdot u(t)}{v(t)}, \quad v(t_0) = \sqrt{x^2(t_0)-1}. \end{aligned} \tag{2.31}$$

Function `argtanh`

$$\operatorname{argtanh}'(t) \;=\; \frac{1}{1-t^2}$$

Similarly:

$$\begin{aligned} y(t) &= \operatorname{argtanh}(x(t)) \\ y'(t) &= \frac{x'(t)}{1-x^2(t)}, \qquad y(t_0) = \operatorname{argtanh}(x(t_0)) \\ y'(t) &= \frac{u(t)}{1-x^2(t)}, \qquad y(t_0) = \operatorname{argtanh}(x(t_0)). \end{aligned} \tag{2.32}$$

Function `argcoth`

$$\operatorname{argcoth}'(t) \;=\; \frac{1}{1-t^2}$$

The transformation of inverse hyperbolic cotangent is almost the same, except for the initial condition:

$$\begin{aligned} y(t) &= \operatorname{argcoth}(x(t)) \\ y'(t) &= \frac{x'(t)}{1-x^2(t)}, \qquad y(t_0) = \operatorname{argcoth}(x(t_0)) \\ y'(t) &= \frac{u(t)}{1-x^2(t)}, \qquad y(t_0) = \operatorname{argcoth}(x(t_0)). \end{aligned} \tag{2.33}$$

2.2.5 Exponential function

$$\left(e^t\right)' \;=\; e^t$$

Simply:

$$\begin{aligned} y(t) &= e^{x(t)} \\ y'(t) &= x'(t)\cdot e^{x(t)}, \quad y(t_0) = e^{x(t_0)} \\ y'(t) &= u(t)\cdot y(t), \quad y(t_0) = e^{x(t_0)}. \end{aligned} \tag{2.34}$$

2.2.6 Natural logarithm

$$\ln'(t) \;=\; \frac{1}{t}$$

Analogically:

$$\begin{aligned} y(t) &= \ln(x(t)) \\ y'(t) &= \frac{x'(t)}{x(t)}, \qquad y(t_0) = \ln(x(t_0)) \\ y'(t) &= \frac{u(t)}{x(t)}, \qquad y(t_0) = \ln(x(t_0)). \end{aligned} \tag{2.35}$$

2.2.7 Square root

$$\left(\sqrt{t}\right)' \;=\; \frac{1}{2\sqrt{t}}$$

This transformation is also made in the results of the previous transformations (where the square root is used):

$$\begin{aligned} y(t) &= \sqrt{x(t)} \\ y'(t) &= \frac{x'(t)}{2y(t)}, \quad y(t_0) = \sqrt{x(t_0)} \\ y'(t) &= \frac{u(t)}{2y(t)}, \quad y(t_0) = \sqrt{x(t_0)}. \end{aligned} \tag{2.36}$$

2.2.8 Division

Divisions can be eliminated after the previous transformations have been performed – this can be performed simultaneously with the other transformations, but some of them can produce other divisions; therefore, it is better to replace divisions afterwards.

Suppose we have

$$y(t) = \frac{1}{x(t)}. \tag{2.37}$$

If the first derivative of (2.37) is formed, the division can be avoided by using the common substitution (2.38).

$$y'(t) = -\frac{x'(t)}{x^2(t)} = -u(t) \cdot y^2(t), \quad y(t_0) = \frac{1}{x(t_0)} \tag{2.38}$$

Then every division can be substituted by the multiplication of the first argument and the transformation of the second argument inversed.

2.2.9 Example

To understand the automatic transformation more clearly, consider the following differential equation[2].

$$y'(t) = \frac{\sin\left(e^{\sinh(t)}\right)}{\cosh(t)}, \quad y(0) = 0 \tag{2.39}$$

We can transform (2.39) into an equivalent system of differential equations (2.44) using the following steps:

First, $\sinh(t)$ is transformed:

$$\begin{aligned} u(t) &= \sinh(t) \\ u'(t) &= \sinh'(t) = \cosh(t) = v(t), \quad u(0) = \sinh(0) = 0 \\ v'(t) &= \cosh'(t) = \sinh(t) = u(t), \quad v(0) = \cosh(0) = 1 \end{aligned} \tag{2.40}$$

then, $e^{u(t)}$ is substituted by:

$$\begin{aligned} w(t) &= e^{u(t)} \\ w'(t) &= \left(e^{u(t)}\right)' = w(t) \cdot u'(t) = w(t) \cdot v(t), \quad w(0) = e^{u(0)} = 1 \end{aligned} \tag{2.41}$$

further, $\sin(w(t))$ becomes

$$\begin{aligned} r(t) &= \sin(w(t)) \\ r'(t) &= \cos(w(t)) \cdot w'(t) = s(t) \cdot w(t) \cdot v(t), \qquad r(0) = \sin(w(0)) = \sin(1) \\ s'(t) &= -\sin(w(t)) \cdot w'(t) = -r(t) \cdot w(t) \cdot v(t), \quad s(0) = \cos(w(0)) = \cos(1) \end{aligned} \tag{2.42}$$

[2]This equation contains trigonometric, hyperbolic and exponential functions.

and $\cosh(t)$ has already appeared as $v(t)$. At the end, we can also transform the inverse value of $v(t)$:

$$\begin{aligned} q(t) &= \frac{1}{v(t)} = v^{-1}(t) \\ q'(t) &= -v^{-2}(t) \cdot v'(t) = -q^2(t) \cdot u(t), \quad q(0) = \frac{1}{v(0)} = 1 \end{aligned} \tag{2.43}$$

The final system of ODEs is (2.44).

$$\begin{aligned} q'(t) &= -q^2(t) \cdot u(t), & q(0) &= 1 \\ r'(t) &= s(t) \cdot w(t) \cdot v(t), & r(0) &= \sin(1) \\ s'(t) &= -r(t) \cdot w(t) \cdot v(t), & s(0) &= \cos(1) \\ u'(t) &= v(t), & u(0) &= 0 \\ v'(t) &= u(t), & v(0) &= 1 \\ w'(t) &= w(t) \cdot v(t), & w(0) &= 1 \\ y'(t) &= r(t) \cdot q(t), & y(0) &= 0 \end{aligned} \tag{2.44}$$

2.3 Transformation into basic operations

Using the automatic transformation, each elementary function can be transformed into basic operations – addition, subtraction, multiplication and division. The division can be replaced by the multiplication; moreover, the subtraction can be replaced by the addition with the opposite sign of the second argument (it can be performed either via multiplication by -1 or using an unary minus).

2.3.1 Transformation into the minimal form

Now we have only additions and multiplications, so it is possible to use the commutative, the distributive and the associative laws to rearrange an expression into its minimal form. The following expression is taken as an example.

$$u = (x + 2y) \cdot (x - 2y) \tag{2.45}$$

The expression is parsed into the syntax tree in Figure 2.2.

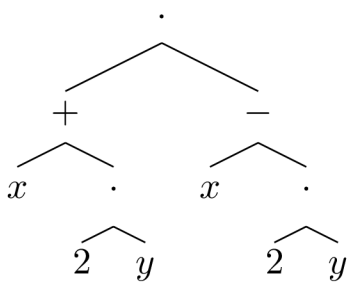


Figure 2.2: Syntax tree

The transformation begins by determining the operation in the root node – multiplication, so the transformations are performed on the left and right children nodes and the

result is a multiplication by terms (using the distributive law):

$$(x + 2y) \cdot (x - 2y) \quad \rightarrow \quad x \cdot x + x \cdot (-2y) + 2y \cdot x + 2y \cdot (-2y). \tag{2.46}$$

Now the expression can be rearranged (using the commutative law) and variables can be merged (using the associative law and expressing multiplications as exponentiations)[3]:

$$x \cdot x + x \cdot (-2y) + 2y \cdot x + 2y \cdot (-2y) \quad \rightarrow \quad x^2 - 4 \cdot y^2. \tag{2.47}$$

2.3.2 Minimal form

The proposed algorithm leads to the unique minimal form if the result is sorted (assigning indices to variables), coefficients a_i are non-zero, at most one product is empty ($n_i = 0$), no variable is repeated within any term and no product is duplicated. The final minimal form is described by a polynomial (2.48).

$$y \; = \; \sum_{i=1}^{m} a_i \prod_{j=1}^{n_i} v_{ij}^{r_{ij}}, \quad m, n_i \; \in \; \mathbb{N}_0, \; r_{ij} \; \in \; \mathbb{N}, \; a_i \; \in \; \mathbb{R} \setminus \{0\}, \; v_{ij} \; \in \; Var \tag{2.48}$$

[3] It is a common formula $(a + b) \cdot (a - b) = a^2 - b^2$.

Chapter 3

Characteristics of MTSM

The thesis deals mainly with explicit MTSM. This method is characterized by high accuracy, speed and flexibility. The method uses as many MTSM terms as needed to attain the required accuracy (arbitrary order) and the accuracy does not depend on the step size.

3.1 Accuracy of calculation

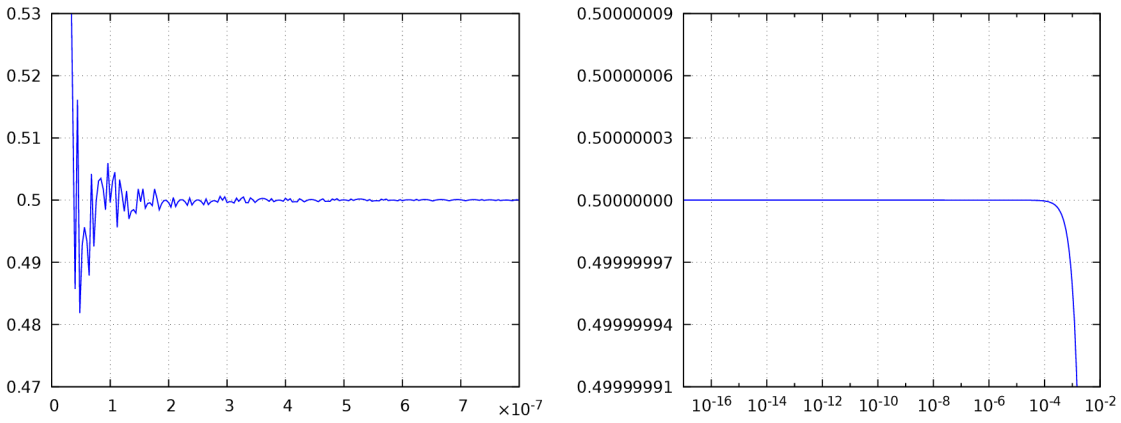
Higher accuracy can be achieved using arbitrary precision arithmetic. For example, the numerical solution of (3.1) is not accurate when using common arithmetic (`double`).

$$\lim_{x \to 0} \frac{1 - \cos(x)}{x^2} \tag{3.1}$$

The limit can be solved analytically, using L'Hospital's rule twice, the correct value is

$$\lim_{x \to 0} \frac{1 - \cos(x)}{x^2} = \lim_{x \to 0} \frac{\sin(x)}{2x} = \lim_{x \to 0} \frac{\cos(x)}{2} = \frac{1}{2}. \tag{3.2}$$

When solving the limit (3.1) numerically using double arithmetic, the result significantly oscillates around the value 0.5 when x approaches zero – Figure 3.1a. However, when the arbitrary precision arithmetic is used, the oscillation does not occur: the result approaches zero smoothly – Figure 3.1b.



(a) Using number type `double`

(b) Using library `MPFR`

Figure 3.1: Numerical solution (approaching limit from right)

3.2 Speed of calculation

The speed of calculation using MTSM depends on the required precision and the step size. The order of the method changes in accordance with the chosen parameters. Compare the system of ODEs (3.3), which generates functions $\sin(t)$ and $\cos(t)$,

$$\begin{aligned} y' &= z, \quad & y(0) &= 0 \\ z' &= -y, \quad & z(0) &= 1 \end{aligned} \tag{3.3}$$

and (3.4), which generates e^t.

$$w' = w, \quad w(0) = 1 \tag{3.4}$$

Tables 3.1 and 3.2 summarize the average calculation times[1] and the maximal order. The length of the time interval `TMAX` $= 2\pi$ for (3.3) and `TMAX` $= 100$ for (3.4): the last rows of the tables show the durations of one step.

dt\eps	$\mathbf{10^{-6}}$	$\mathbf{10^{-10}}$	$\mathbf{10^{-20}}$
0.01	$2.30 \cdot 10^{-3}$	$3.55 \cdot 10^{-3}$	$5.30 \cdot 10^{-3}$
0.1	$3.65 \cdot 10^{-4}$	$4.85 \cdot 10^{-4}$	$8.00 \cdot 10^{-4}$
1	$6.90 \cdot 10^{-5}$	$9.60 \cdot 10^{-5}$	$1.50 \cdot 10^{-4}$
$\frac{\pi}{2}$	$5.06 \cdot 10^{-5}$	$6.65 \cdot 10^{-5}$	$1.05 \cdot 10^{-4}$
π	$3.55 \cdot 10^{-5}$	$4.58 \cdot 10^{-5}$	$6.65 \cdot 10^{-5}$
2π	$2.33 \cdot 10^{-5}$	$2.88 \cdot 10^{-5}$	$3.89 \cdot 10^{-5}$

(a) Calculation time [s]

dt\eps	$\mathbf{10^{-6}}$	$\mathbf{10^{-10}}$	$\mathbf{10^{-20}}$
0.01	3	5	8
0.1	5	7	12
1	10	14	22
$\frac{\pi}{2}$	12	16	25
π	17	22	33
2π	27	33	45

(b) Maximal order

Table 3.1: Solution of $y'' = -y$

dt\eps	$\mathbf{10^{-6}}$	$\mathbf{10^{-10}}$	$\mathbf{10^{-20}}$
0.01	$7.20 \cdot 10^{-2}$	$8.30 \cdot 10^{-2}$	$1.01 \cdot 10^{-1}$
0.1	$1.01 \cdot 10^{-2}$	$1.16 \cdot 10^{-2}$	$1.40 \cdot 10^{-2}$
1	$1.76 \cdot 10^{-3}$	$1.90 \cdot 10^{-3}$	$2.38 \cdot 10^{-3}$
10	$4.21 \cdot 10^{-4}$	$4.52 \cdot 10^{-4}$	$5.69 \cdot 10^{-4}$
50	$2.20 \cdot 10^{-4}$	$2.29 \cdot 10^{-4}$	$2.52 \cdot 10^{-4}$
100	$1.86 \cdot 10^{-4}$	$1.90 \cdot 10^{-4}$	$1.98 \cdot 10^{-4}$

(a) Calculation time [s]

dt\eps	$\mathbf{10^{-6}}$	$\mathbf{10^{-10}}$	$\mathbf{10^{-20}}$
0.01	18	19	22
0.1	25	27	31
1	41	44	50
10	87	91	102
50	188	195	211
100	282	291	312

(b) Maximal order

Table 3.2: Solution of $w' = w$

It is obvious from the tables that a bigger step size leads to a shorter calculation time, regardless of the required accuracy. Tables 3.1b and 3.2b demonstrate that it is essential to use a higher order when increasing the step size to maintain the required precision.

[1] The simulation was reiterated many times since it is impossible to measure such small calculation times; each calculation time is the time measured divided by the number of repetitions.

3.3 Stiff systems

MTSM is also capable of solving systems which cannot be solved by general methods (Euler, Runge–Kutta, ...). One problematic group of problems is represented by stiff systems. As an example, consider the following system of ODEs [40].

$$\begin{aligned} y' &= z, & y(0) &= 1 \\ z' &= -ay - (a+1)z, & z(0) &= -1 \end{aligned} \tag{3.5}$$

This system is stiff when the value of parameter a is large. The analytical solution is always $y = e^{-t}$ and $z = -e^{-t}$ regardless of this parameter. Table 3.3 contains thirty significant digits of the solution for $a = 10^{1234}$.

t	y	z
0	1	-1
5	0.00673794699908546709663604842315	-0.00673794699908546709663604842315

Table 3.3: Solution of stiff system

The table confirms that $y = -z$ for given digits. Of course, multiple-precision arithmetic has to be used (4kB numbers) – therefore, the accuracy of the result is high and only one step has to be calculated. For more about stiffness, see [40].

3.3.1 Implicit form of MTSM

In general, stiff systems have to be solved using the implicit form of MTSM (or another method designed specifically for stiff problems) which has a significantly larger area of stability than the explicit form of MTSM – see [40]. Values y_{n+1} are calculated solving (3.6) by some iterative method, for example Newton–Raphson [5, 38].

$$y_{n+1} = y_n - \sum_{k=1}^{ORD} \frac{(-h)^k}{k!} y_{n+1}^{(k)} \tag{3.6}$$

See [3, 12] for more information about solving stiff systems using implicit MTSM and [27, 28] for more about stiffness in initial-value problems.

3.4 Stopping rule

The stopping rule is used to determine the necessary order of MTSM during the calculation. If an incorrect stopping rule is chosen, the solution of some problems can be incorrect. For example, consider (3.7).

$$\begin{aligned} x'(t) &= 1, & x(0) &= 0 \\ y'(t) &= x^3(t), & y(0) &= 0 \end{aligned} \tag{3.7}$$

The analytical solution of the system is $x(t) = t$ and $y(t) = 0.25t^4$. The solution of the system is incorrect for the default stopping rule used by the MTSM implementations – absolute values of three consecutive terms less than `EPS`. Table 3.4 shows that the method did not calculate higher-order terms in $t = 0.1$ since the first three terms were zero – the value should be $2.5 \cdot 10^{-5}$.

t	y	exact
0	0	0
0.1	0	0.000025
0.2	0.000375	0.0004
0.3	0.002	0.002025
0.4	0.006375	0.0064
0.5	0.0156	0.015625
0.6	0.032375	0.0324
0.7	0.06	0.060025
0.8	0.102375	0.1024
0.9	0.164	0.164025
1	0.249975	0.25

Table 3.4: Problem with stopping rule

Table 3.4 also demonstrates the principle drawback of the Euler method, because the Euler method is the same as the Taylor method of the first order. Smaller step size only increases accuracy, but the step size would have to be smaller than $\sqrt[4]{4 \cdot \texttt{EPS}}$ to attain the maximal error of the Euler method smaller than precision `EPS` – the error is $0.25dt^4$ from the first step.

The problem appears only for polynomials; there is no problem in exponential solutions. Therefore, the stopping rule has to be chosen very carefully to solve any problem correctly.

3.5 Principle of calculating MTSM terms

The following text is based on [26]. The method proposed in this thesis transforms electronic circuits into electric circuits which are described by a system of m linear ODEs that can be expressed in the form of (3.8).

$$\begin{aligned} y' &= a_{11} \cdot y + a_{12} \cdot z + \cdots + a_{1m} \cdot w + b_1, & y(0) &= y_0 \\ z' &= a_{21} \cdot y + a_{22} \cdot z + \cdots + a_{2m} \cdot w + b_2, & z(0) &= z_0 \\ &\vdots \\ w' &= a_{m1} \cdot y + a_{m2} \cdot z + \cdots + a_{mm} \cdot w + b_m, & w(0) &= w_0 \end{aligned} \tag{3.8}$$

Each equation of (3.8) consists of m terms with coefficients a_{rs}, where r denotes the index of the equation and s denotes the index of the term. Further, each equation contains one constant b_r and an initial condition – $y(0)$, $z(0)$, …, $w(0)$.

The first MTSM terms are expressed by (3.9) and they are denoted as $DY_{n,1}$, $DZ_{n,1}$, …, $DW_{n,1}$.

$$\begin{aligned} DY_{n,1} &= h \cdot y'_n \\ &= h \cdot (a_{11} \cdot y_n + a_{12} \cdot z_n + \cdots + a_{1m} \cdot w_n + b_1) \\ DZ_{n,1} &= h \cdot z'_n \\ &= h \cdot (a_{21} \cdot y_n + a_{22} \cdot z_n + \cdots + a_{2m} \cdot w_n + b_2) \\ &\vdots \\ DW_{n,1} &= h \cdot w'_n \\ &= h \cdot (a_{m1} \cdot y_n + a_{m2} \cdot z_n + \cdots + a_{mm} \cdot w_n + b_m) \end{aligned} \tag{3.9}$$

The second MTSM terms ($DY_{n,2}$, $DZ_{n,2}$, ..., $DW_{n,2}$) can be calculated as follows.

$$\begin{aligned} DY_{n,2} &= \frac{h}{2} \cdot DY'_{n,1} \\ &= \frac{h}{2} \cdot (a_{11} \cdot DY_{n,1} + a_{12} \cdot DZ_{n,1} + \cdots + a_{1m} \cdot DW_{n,1}) \\ DZ_{n,2} &= \frac{h}{2} \cdot DZ'_{n,1} \\ &= \frac{h}{2} \cdot (a_{21} \cdot DY_{n,1} + a_{22} \cdot DZ_{n,1} + \cdots + a_{2m} \cdot DW_{n,1}) \\ &\vdots \\ DW_{n,2} &= \frac{h}{2} \cdot DW'_{n,1} \\ &= \frac{h}{2} \cdot (a_{m1} \cdot DY_{n,1} + a_{m2} \cdot DZ_{n,1} + \cdots + a_{mm} \cdot DW_{n,1}) \end{aligned} \tag{3.10}$$

The higher order MTSM terms are calculated in a similar way.

If the stopping rule is met, the final result in the current time step is obtained as the sum of individual MTSM terms.

$$\begin{aligned} y_{n+1} &= y_n + DY_{n,1} + DY_{n,2} + \cdots + DY_{n,ORD_n} \\ z_{n+1} &= z_n + DZ_{n,1} + DZ_{n,2} + \cdots + DZ_{n,ORD_n} \\ &\vdots \\ w_{n+1} &= w_n + DW_{n,1} + DW_{n,2} + \cdots + DW_{n,ORD_n} \end{aligned} \tag{3.11}$$

The calculation in the other time steps is performed in a similar way and the results obtained by (3.11) then serve as the initial conditions for the next step. See [26] for more information.

3.6 Practical usage

MTSM can be used for solving various problems. This section introduces some of those which can be effectively solved by this method.

3.6.1 Mechanical oscillator

The first example is the solution of a mechanical oscillator. The movement of the oscillator is described by

$$y'' + ky' + a\sin(y) = 0 \tag{3.12}$$

where parameter k is the attenuation of the oscillation and parameter a is the toughness of the oscillator. The initial values are determined by the initial position

$$y(0) = \varphi_0 = 0,$$

and initial speed

$$y'(0) = v_0 = 15.$$

The system of two ordinary differential equations (3.13) is obtained by modifying the differential equation (3.12).

$$\begin{aligned} y'(t) &= z(t), & y(0) &= 0 \\ z'(t) &= -k \cdot z(t) - a \cdot \sin(y(t)), & z(0) &= 15 \end{aligned} \tag{3.13}$$

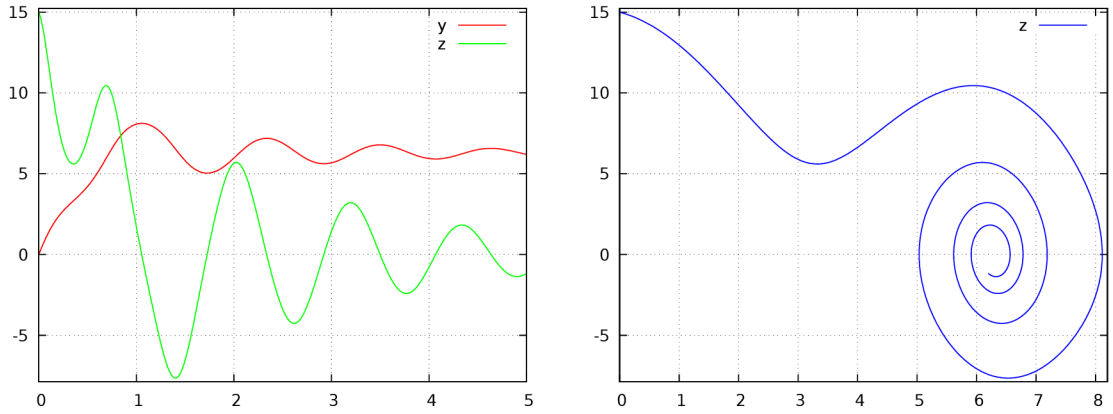
The automatic transformation substitutes $\sin(y(t))$ with two differential equations (3.14).

$$\begin{aligned} u'(t) &= v(t) \cdot z(t), \quad & u(0) &= \sin(y(0)) = 0 \\ v'(t) &= -u(t) \cdot z(t), \quad & v(0) &= \cos(y(0)) = 1 \end{aligned} \tag{3.14}$$

Finally, the autonomous form (3.15) is created, for which the generation of MTSM terms is quite simple.

$$\begin{aligned} y'(t) &= z(t), & y(0) &= 0 \\ z'(t) &= -k \cdot z(t) - a \cdot u(t), & z(0) &= 15 \\ u'(t) &= v(t) \cdot z(t), & u(0) &= 0 \\ v'(t) &= -u(t) \cdot z(t), & v(0) &= 1 \end{aligned} \tag{3.15}$$

By solving the system, the graph of the position and the speed is obtained – Figure 3.2a. The dependency of the speed on the position is shown in Figure 3.2b. The source code can be found in Appendix A.3.



(a) Position and speed

(b) Dependency of speed on position

Figure 3.2: Solution of mechanical oscillator

3.6.2 Calculation of a definite integral

MTSM is suitable not only for solving systems of ODEs but also for calculating a definite integral. The function (3.16) is used as an example.

$$f(x) = \sin\Big(\ln(x+1)\Big) \cdot \ln(x+1) \cdot \frac{(x-1)^2}{x+1} \tag{3.16}$$

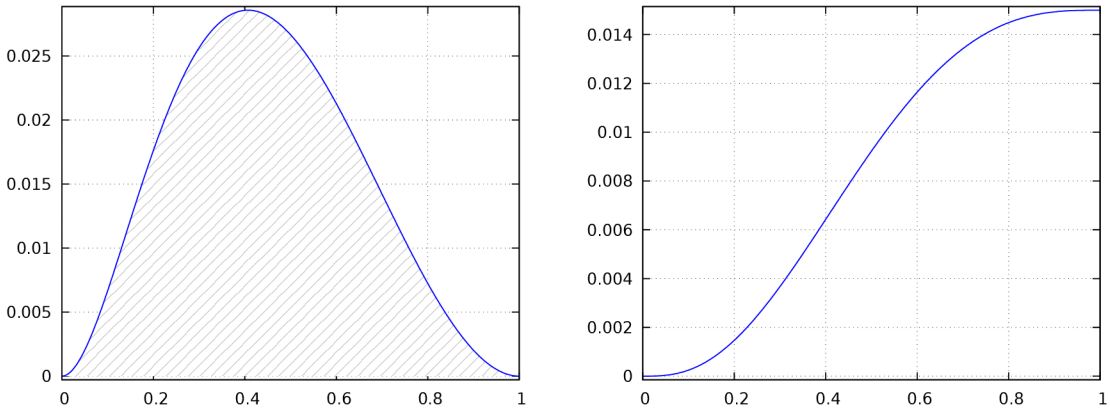
The graph of the function with a filled area under the curve is shown in Figure 3.3a. The calculation of the area is performed in the interval $<0, 1>$ – that means evaluating the definite integral (3.17).

$$\int_0^1 \sin\Big(\ln(x+1)\Big) \cdot \ln(x+1) \cdot \frac{(x-1)^2}{x+1}\, \mathrm{d}x \tag{3.17}$$

The integral can be transformed into the ordinary differential equation (3.18)

$$y' = \sin\Big(\ln(x+1)\Big) \cdot \ln(x+1) \cdot \frac{(x-1)^2}{x+1}, \quad y(0) = 0, \tag{3.18}$$

and the solution in time $t = 1$ gives the result of the definite integral – see Figure 3.3b.



(a) Function (3.16)

(b) Solution of ODE (3.18)

Figure 3.3: Calculation of definite integral

3.6.3 Fourier coefficients

Definite integrals are also used for calculating Fourier coefficients. As an example, consider the function (3.19).

$$\begin{aligned} f(t) \;=\; & 16\sin^5(t) + 4\cos^3(t) - \sin^2(t) + \cos^2(t) + 2\sin(t)\cos(t) - \sin(5t) \\ & + 5\sin(3t) - \cos(3t) - \cos(2t) - \sin(2t) - 10\sin(t) \end{aligned} \tag{3.19}$$

The calculation is performed by expressing the Fourier coefficients for the function (3.19) as integrals (3.20):

$$\begin{aligned} a_k \;&=\; \frac{1}{\pi} \cdot \int_0^{2\pi} f(t) \cdot \cos(k\omega t), \\ b_k \;&=\; \frac{1}{\pi} \cdot \int_0^{2\pi} f(t) \cdot \sin(k\omega t), \end{aligned} \tag{3.20}$$

transforming it into the system of ordinary differential equations (3.21) and solving this system ($\omega = 1$ rad/s).

$$\begin{aligned} a'_k \;&=\; \frac{1}{\pi} \cdot f(t) \cdot \cos(k\omega t), \quad a_k(0) \;=\; 0 \\ b'_k \;&=\; \frac{1}{\pi} \cdot f(t) \cdot \sin(k\omega t), \quad b_k(0) \;=\; 0 \end{aligned} \tag{3.21}$$

Tables 3.5 and 3.6 summarize the results calculated in time $t = 2\pi$, which is the period of the Fourier coefficients calculation.

a_0	a_1	a_2	a_3	a_4	a_5
$1.428 \cdot 10^{-18}$	3	$4.390 \cdot 10^{-18}$	$6.806 \cdot 10^{-18}$	$1.270 \cdot 10^{-18}$	$1.509 \cdot 10^{-18}$

Table 3.5: Fourier coefficients a_k

b_1	b_2	b_3	b_4	b_5
$4.372 \cdot 10^{-18}$	$-2.342 \cdot 10^{-18}$	$-5.998 \cdot 10^{-18}$	$2.682 \cdot 10^{-18}$	$1.319 \cdot 10^{-18}$

Table 3.6: Fourier coefficients b_k

The results in the tables show that the only non-zero coefficient is $a_1 = 3$ (other coefficients are almost zero). Therefore, the function (3.19) can be simplified into the form $f(t) = 3\cos(t)$. Figure 3.4 illustrates the solution of the system of ODEs (3.21).

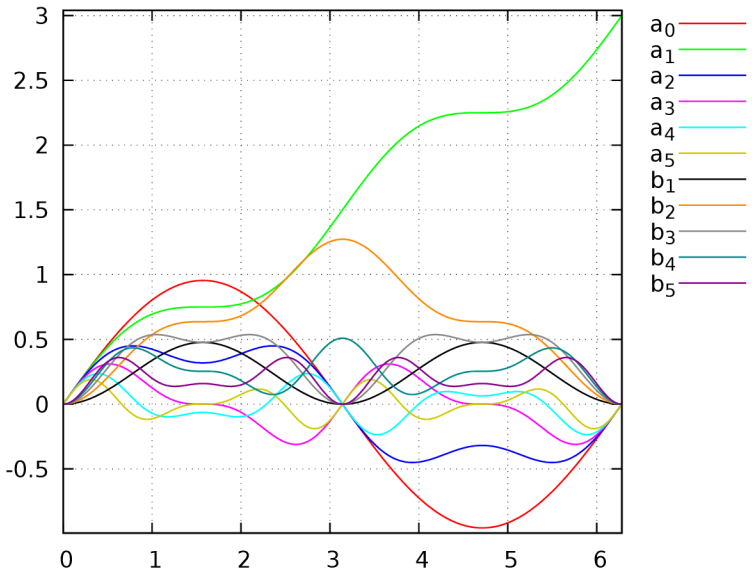


Figure 3.4: Calculation of Fourier coefficients

Analytical proof

The function (3.19) can be simplified using trigonometric formulas [4].

$$\begin{aligned} f(t) &= 16\left(\frac{1}{16}\Big(10\sin(t) - 5\sin(3t) + \sin(5t)\Big)\right) + 4\left(\frac{1}{4}\Big(3\cos(t) + \cos(3t)\Big)\right) + \cos(2t) \\ &\quad + \sin(2t) - \sin(5t) + 5\sin(3t) - \cos(3t) - \cos(2t) - \sin(2t) - 10\sin(t) \\ f(t) &= 10\sin(t) - 5\sin(3t) + \sin(5t) + 3\cos(t) + \cos(3t) - \sin(5t) + 5\sin(3t) - \cos(3t) \\ &\quad - 10\sin(t) \\ f(t) &= 3\cos(t) \end{aligned}$$

□

Chapter 4

Solving Electric Circuits

Electric circuits with a harmonic (sine-wave) voltage source can be solved either symbolically or numerically. If the voltage source is not a harmonic sine signal or if a simulation of the transient response is required, then the circuit has to be solved numerically.

4.1 Phasor diagrams

This method is symbolic and solves a problem graphically. It represents voltages and currents by vectors (phasors) in a phasor diagram. It is based on the knowledge that the voltage on a coil is shifted by $\frac{\pi}{2}$ before the current flowing through the coil and the voltage on a capacitor is delayed by the same phase after the current flowing through the capacitor. The relationship between phasors and the value of the voltage dependent on time is illustrated in Figure 4.1. See [34] for more information.

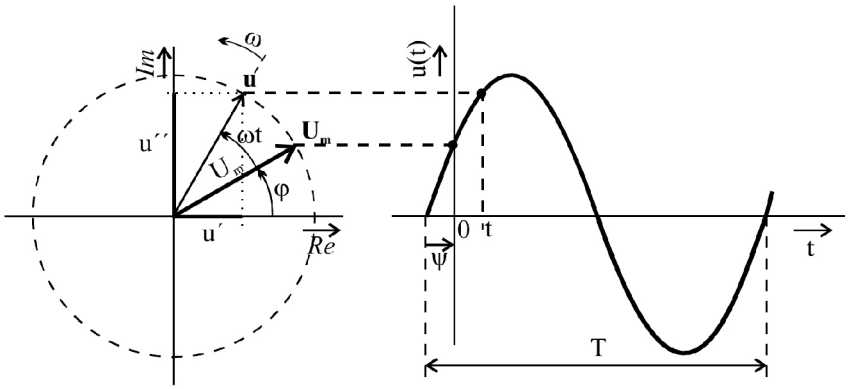


Figure 4.1: Relationship between phasors and voltage [30]

4.1.1 Serial RLC circuit

Consider the circuit in Figure 4.2. Phasor I, which represents current i, is charted in a phasor diagram; U_R, representing the voltage of the resistor, is in a phase with it; U_L and U_C (phasors for the voltages on the coil and the capacitor) are orthogonal to I. The sum of all voltages gives the total voltage (phasor U).

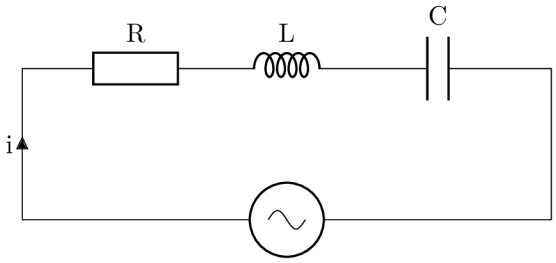
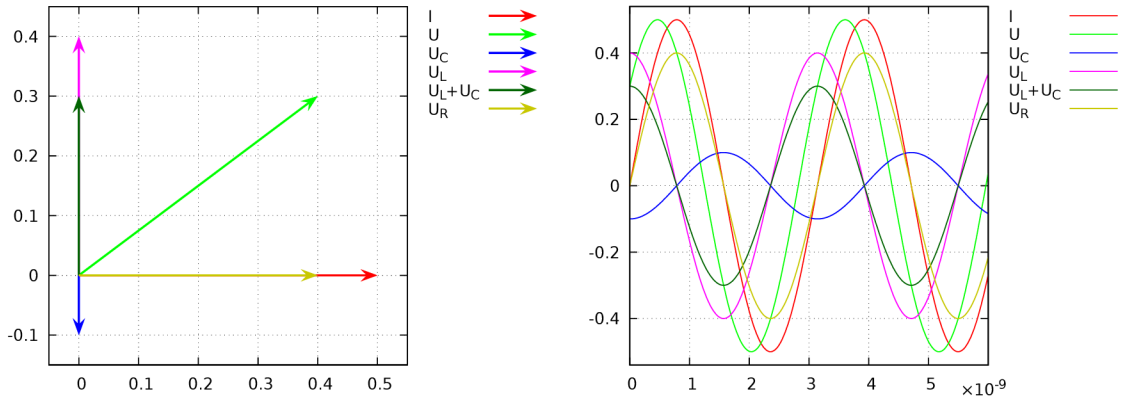


Figure 4.2: RLC circuit

The result for parameters $\omega = 2 \cdot 10^9$ rad/s, $R = 0.8\ \Omega$, $L = 4 \cdot 10^{-10}$ H, $C = 2.5 \cdot 10^{-9}$ F is illustrated in Figure 4.3. Figure 4.3b shows the solution in time.



(a) Phasor diagram

(b) Solution

Figure 4.3: Solution of RLC circuit

4.1.2 Serial-parallel circuits

With a more complex circuit, e. g. in Figure 4.4, the phasor diagram is not as clear as in the previous case.

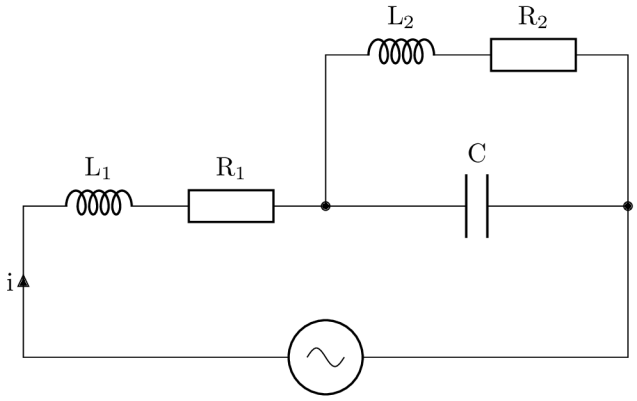
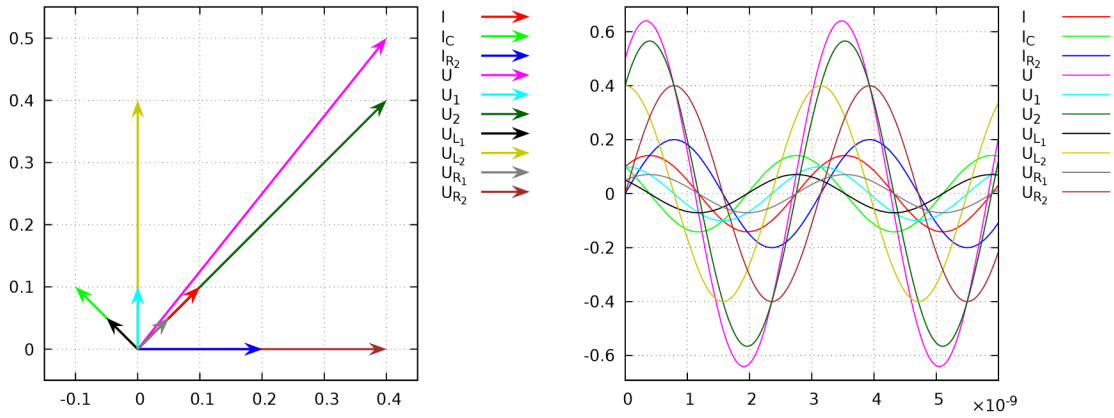


Figure 4.4: More complex circuit

The result is shown in Figure 4.5 ($\omega = 2{\cdot}10^9$ rad/s, $R_1 = 0.5\ \Omega$, $R_2 = 2\ \Omega$, $L_1 = 2.5{\cdot}10^{-10}$ H, $L_2 = 10^{-9}$ H, $C = 1.25 \cdot 10^{-10}$ F).



(a) Phasor diagram

(b) Solution

Figure 4.5: Solution of more complex circuit

4.2 Symbolic-complex method

Complex numbers can be mapped into vectors in the Gauss plane. This property can be used to solve electric circuits using common algebraic operations in a complex domain. Term capacitor reactance (capacitance) is introduced for a capacitor:

$$X_C = \frac{1}{j\omega C} = -\frac{j}{\omega C} \tag{4.1}$$

and coil reactance (inductance) is established for a coil:

$$X_L = j\omega L. \tag{4.2}$$

Then the whole impedance of the circuit in Figure 4.4 can be calculated by complex numbers.

$$\begin{aligned} Z &= X_{L_1} + R_1 + \frac{(X_{L_2} + R_2) \cdot X_C}{X_{L_2} + R_2 + X_C} = j\omega L_1 + R_1 + \frac{(j\omega L_2 + R_2) \cdot (-\frac{j}{\omega C})}{j\omega L_2 + R_2 - \frac{j}{\omega C}} \\ Z &= \frac{R_1 D + R_2}{D} + \frac{\omega L_1 D - \omega^3 C L_2^2 + \omega L_2 - \omega C R_2^2}{D} j \\ D &= \omega^2 C \left(\omega^2 C L_2^2 + C R_2^2 - 2L_2\right) + 1 \end{aligned} \tag{4.3}$$

4.3 Numerical solution

Now consider the numerical solution of electric circuits. Electric circuits can be solved numerically using the well-known Kirchhoff's laws [39]. As an example, the electric circuit in Figure 4.6 is analyzed.

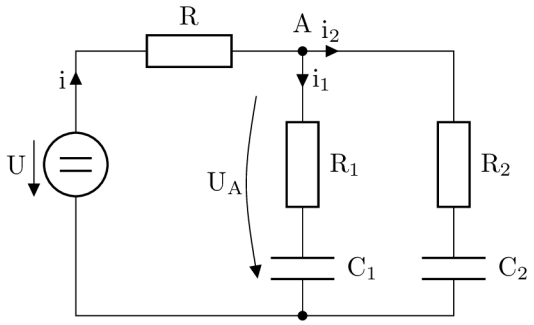


Figure 4.6: Electric circuit

The circuit consists of three resistors and two capacitors – a system of two ODEs (4.4) is created:

$$\begin{aligned} u'_{C_1} &= \frac{1}{C_1} i_1, & u_{C_1}(0) &= 0 \\ u'_{C_2} &= \frac{1}{C_2} i_2, & u_{C_2}(0) &= 0 \end{aligned}$$

$$\begin{aligned} i &= \frac{U - U_A}{R} \\ i_1 &= \frac{U_A - U_{C_1}}{R_1} \\ i_2 &= \frac{U_A - U_{C_2}}{R_2} \end{aligned} \tag{4.4}$$

Voltage U_A is determined by

$$i = i_1 + i_2,$$

where i, i_1 and i_2 are substituted:

$$\frac{U - U_A}{R} = \frac{U_A - U_{C_1}}{R_1} + \frac{U_A - U_{C_2}}{R_2}$$

and after a modification:

$$U_A = \frac{U \cdot R_1 \cdot R_2 + U_{C_1} \cdot R \cdot R_2 + U_{C_2} \cdot R \cdot R_1}{R \cdot R_1 + R \cdot R_2 + R_1 \cdot R_2}.$$

For parameters $U = 10$ V, $R = 80\ \Omega$, $R_1 = 10\ \Omega$, $R_2 = 40\ \Omega$, $C_1 = 1$ F, $C_2 = 2$ F, the solution in Figure 4.7 is obtained.

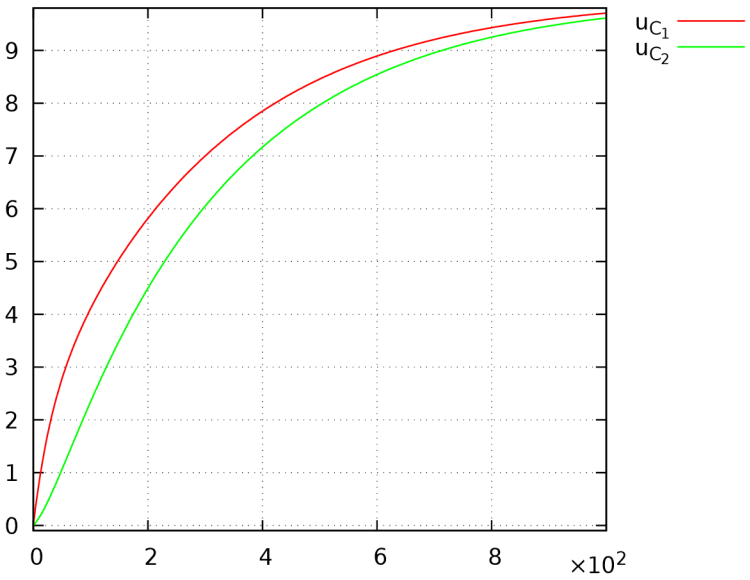


Figure 4.7: Solution of electric circuit

4.3.1 Elimination of algebraic operations

The number of algebraic operations for the calculation of U_A can be decreased by adding parasitic capacitor C_p, as demonstrated by Figure 4.8.

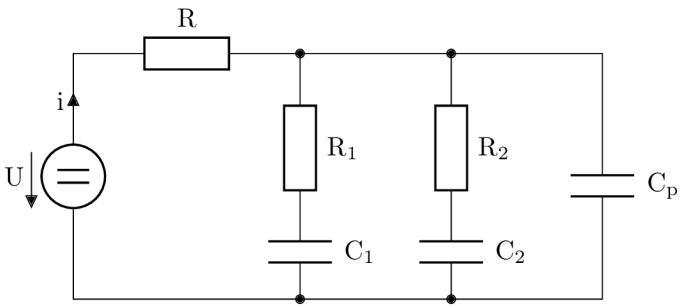
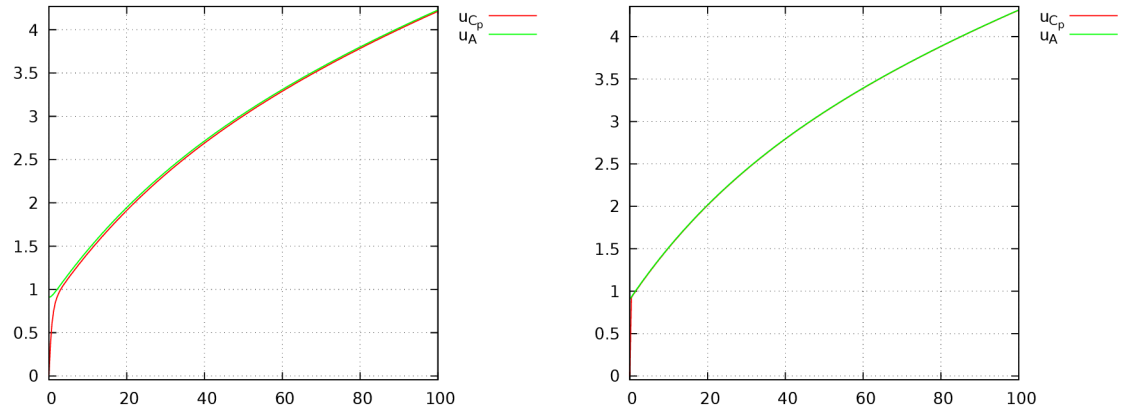


Figure 4.8: Electric circuit with parasitic capacitor

The accuracy of the approximation of voltage U_A is given by the value of parasitic capacity C_p. In practice[1], the value of parasitic capacity C_p can be up to 10 % of capacities C_1 and C_2. The lower the value of capacity C_p chosen, the more precise the approximation of voltage U_A obtained. On the other hand, by lowering the value of C_p, the stiffness of the resulting system increases, but it is a minor problem for MTSM.

Figure 4.9 shows that a suitable value for capacity C_p is 1 % of capacity C_1. In this case, the transient response, during which the capacitor is being charged and the value of the voltage of the parasitic capacity does not correspond to voltage U_A, takes approximately one second.

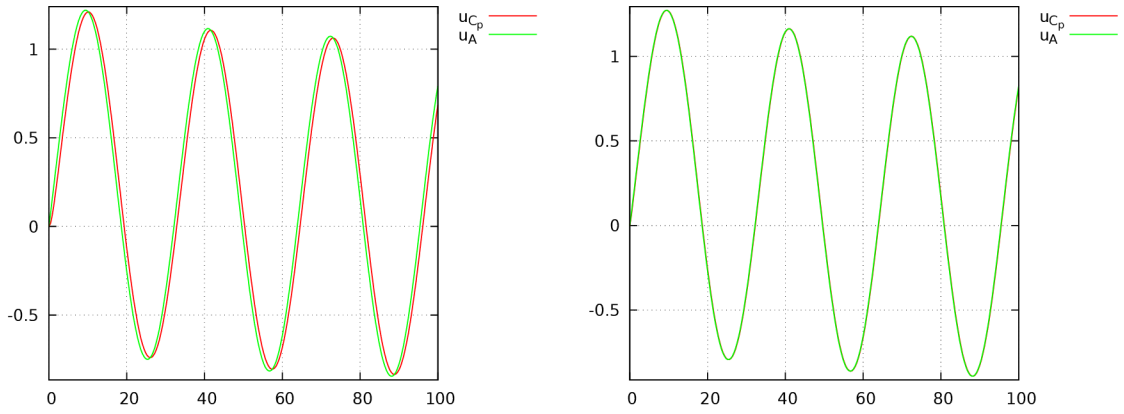
[1] Regarding the tolerance of components.



(a) $C_p = 0.1$ F (b) $C_p = 0.01$ F

Figure 4.9: Comparison of U_{C_p} and U_A

The direct-voltage source can be replaced by the alternating-voltage source[2]. If the value of capacity C_p is 1 % of capacity C_1 (Figure 4.10b), the solution converges better than in the case of 10 % of capacity C_1 (Figure 4.10a).



(a) $C_p = 0.1$ F (b) $C_p = 0.01$ F

Figure 4.10: Comparison of U_{C_p} and U_A for alternating-voltage source

4.3.2 Shortening the transient response

As already stated, a transient response occurs when approximating voltage U_A with a parasitic capacity. This response can be shortened not only by lowering the parasitic capacity but also by using compensating capacity C_k – see Figure 4.11.

[2]With $\omega = 0.2$ rad/s.

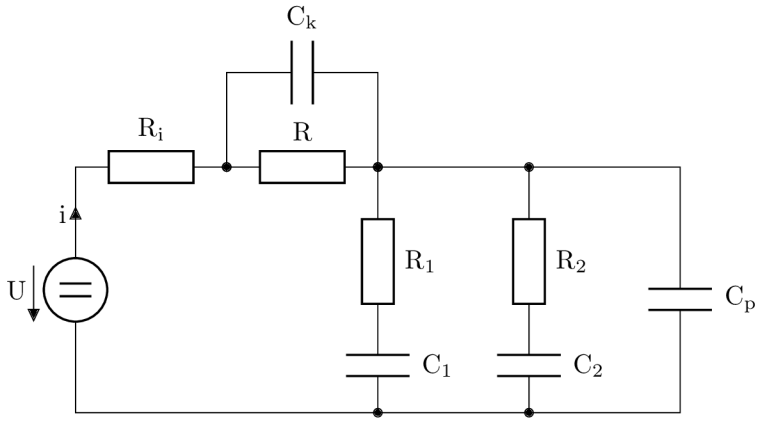
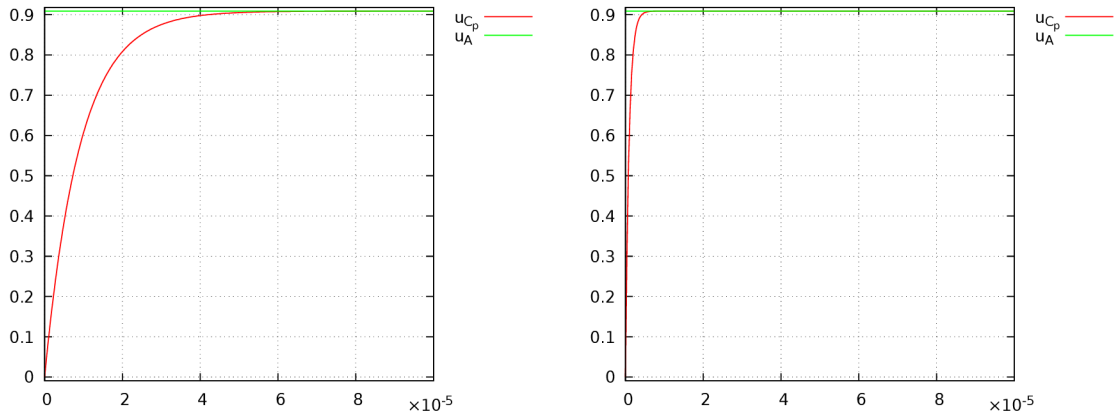


Figure 4.11: Electric circuit with compensating capacity

By solving the electric circuit, a much shorter transient response is obtained – Figure 4.12: in this case, the transient response fades out by 10^{-4} s.



(a) $C_k = 0.001$ F, $R_i = 0.01\ \Omega$ (b) $C_k = 0.001$ F, $R_i = 0.001\ \Omega$

Figure 4.12: Transient response of circuit with compensating capacity

4.4 Telegraph line

The telegraph line is a simple electric circuit which can be used for comparing various approaches to calculation since parameters can be changed as well as the complexity of the circuit (number of segments). The telegraph-line model can be described by the partial differential equations (4.5) [43].

$$\begin{aligned} L \cdot C \frac{\partial^2 u(x,t)}{\partial t^2} + (L \cdot G + C \cdot R) \frac{\partial u(x,t)}{\partial t} + R \cdot G \cdot u(x,t) - \frac{\partial^2 u(x,t)}{\partial x^2} &= 0 \\ L \cdot C \frac{\partial^2 i(x,t)}{\partial t^2} + (L \cdot G + C \cdot R) \frac{\partial i(x,t)}{\partial t} + R \cdot G \cdot i(x,t) - \frac{\partial^2 i(x,t)}{\partial x^2} &= 0 \end{aligned} \tag{4.5}$$

The circuit in its simplest form consists of a source of voltage, two resistors R_1 and R_2, N capacitors and N coils. Parameter N determines the number of segments, which can be relatively large. Various types of voltage sources can be used: harmonic sine, an impulse

etc. The behavior of the model can be observed by the analysis of input voltage u_1 and output voltage u_2 – see Figure 4.13.

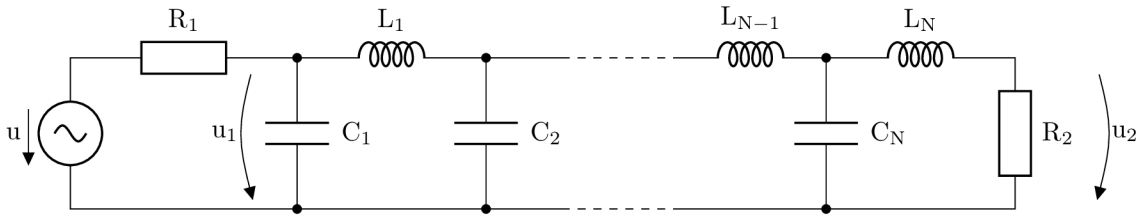


Figure 4.13: Telegraph equation model

The value of the output voltage can be influenced by the sizes of the input (R_1) and output (R_2) loads. If the line is adjusted, (4.6) has to hold.

$$R_1 \;=\; R_2 \;=\; \sqrt{\frac{L}{C}} \tag{4.6}$$

The same value for all inductances is used:

$$L \;=\; 10^{-8}\ \mathrm{H},$$

and the same value for all capacitors:

$$C \;=\; 10^{-12}\ \mathrm{F}.$$

Then, to adjust the line, both resistances have to be set to

$$R_1 \;=\; R_2 \;=\; 100\ \Omega.$$

4.4.1 Symbolic solution

If a harmonic voltage source is used, the problem can be solved using complex numbers that represent impedances. Resistors have real impedances while capacitances and inductances are represented by imaginary impedances. All impedances are complex numbers, so they can be combined using the basic mathematical operations:

$$X_L \;=\; j\omega L, \quad X_C \;=\; -\frac{j}{\omega C}, \quad X_{L_N R_2} \;=\; X_{L_N} + R_2, \quad X_{C_N L_N R_2} \;=\; \frac{X_{C_N} X_{L_N R_2}}{X_{C_N} + X_{L_N R_2}}, \quad \ldots$$

Figure 4.14 shows the calculated voltages for 50 segments. A harmonic sine-wave signal was chosen as a source of voltage with frequency $\omega = 10^9$ rad/s and amplitude $A = 2$ V.

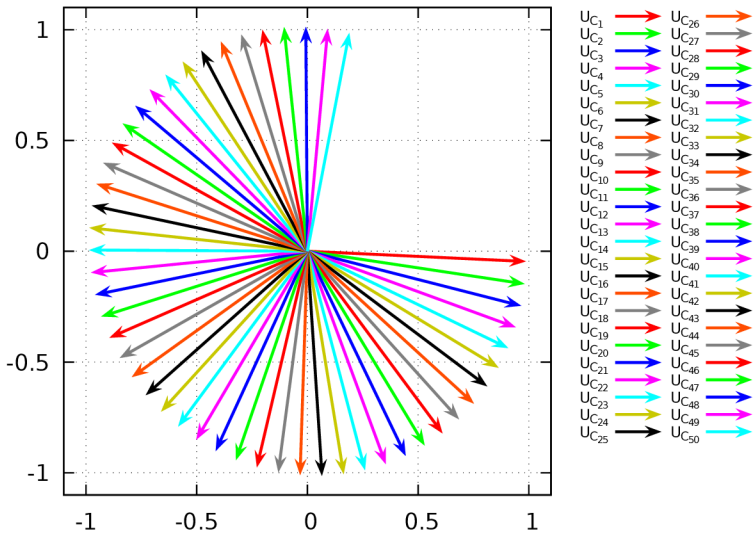


Figure 4.14: Voltage phasors

If the sine waves are calculated using phasors from Figure 4.14, the symbolic solution of the telegraph line is obtained (Figure 4.15).

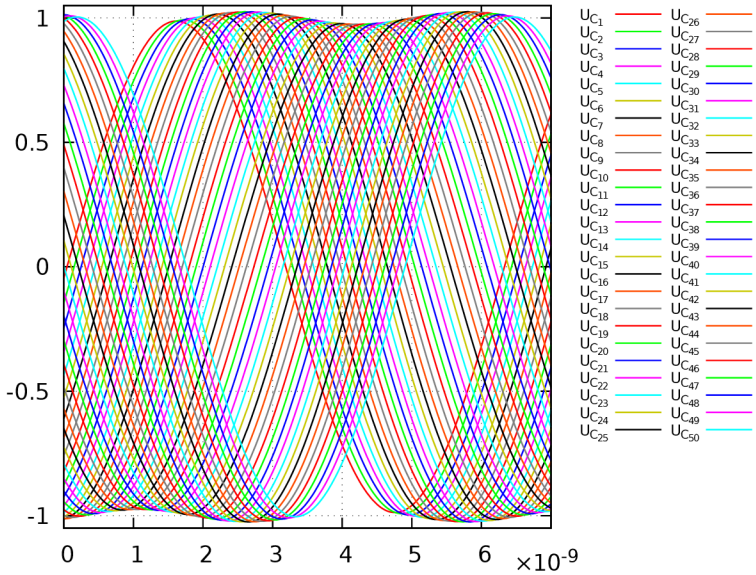


Figure 4.15: Symbolic solution

4.4.2 Numerical solution

The symbolic solution can be used only for harmonic input signals. If the input is not a harmonic signal or the analysis of the transient response is required, the numerical solution has to be used. Of course, the numerical solution can also be used for harmonic input signals. The system of equations (4.7) is equivalent to the previous symbolic calculation.

The appropriate telegraph-line model can be seen in Figure 4.13.

$$\begin{aligned}
u' &= \omega v, & u(0) &= 0 \\
v' &= -\omega u, & v(0) &= U \\
i &= \frac{1}{R_1}\left(u - u_{C_1}\right) & & \\
u'_{C_1} &= \frac{1}{C}\left(i - i_{L_1}\right), & u_{C_1}(0) &= 0 \\
i'_{L_1} &= \frac{1}{L}\left(u_{C_1} - u_{C_2}\right), & i_{L_1}(0) &= 0 \\
u'_{C_2} &= \frac{1}{C}\left(i_{L_1} - i_{L_2}\right), & u_{C_2}(0) &= 0 \\
&\vdots & &\vdots \\
i'_{L_N} &= \frac{1}{L}\left(u_{C_N} - R_2 \cdot i_{L_N}\right), & i_{L_N}(0) &= 0
\end{aligned} \tag{4.7}$$

Figure 4.16 shows the numerical solution of the same system that was solved symbolically. The numerical solution is similar, but the transient response can be seen which cannot be described by the symbolic solution. The transient response is also responsible for the apparent roughness – confirmed by MATLAB solvers (`ode45`, `ode23s` and `ode23t`).

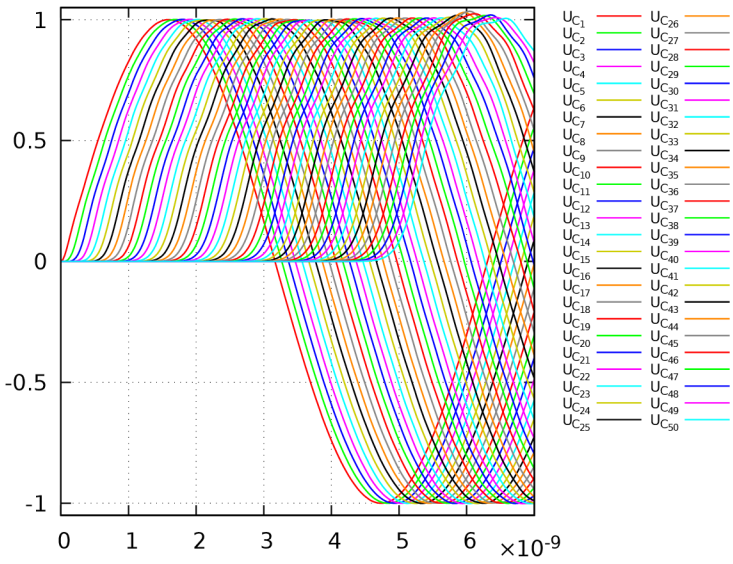


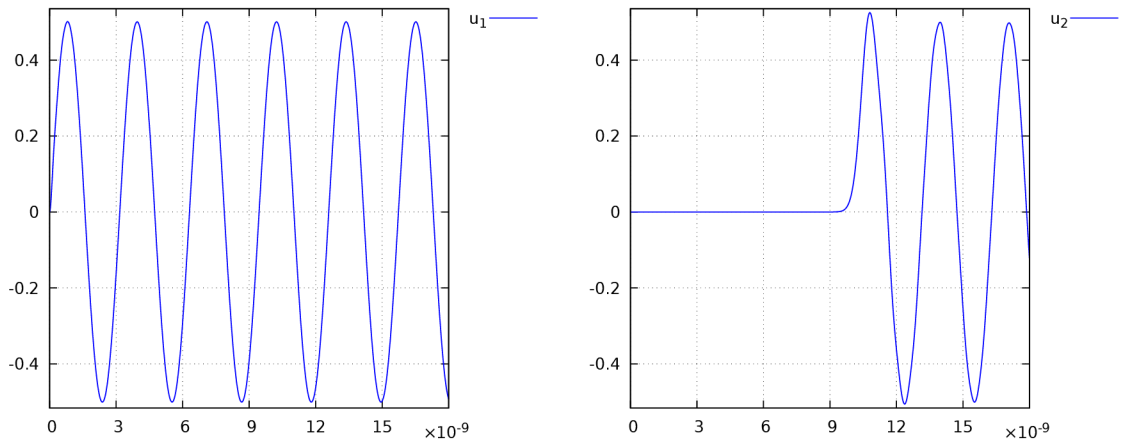
Figure 4.16: Numerical solution

4.4.3 Dependency of output voltage on input voltage

In this subsection, the dependency of output voltage on input voltage is shown; the comparison is performed for 100 segments and parameters L and C have the same values as in the previous sections.

The plots on the left represent voltage u_1 at the beginning of the telegraph line (also designated as the input voltage). The corresponding plots on the right represent voltage u_2 at the end of the telegraph line (also designated as the output voltage). Both plots in each pair have the same scale.

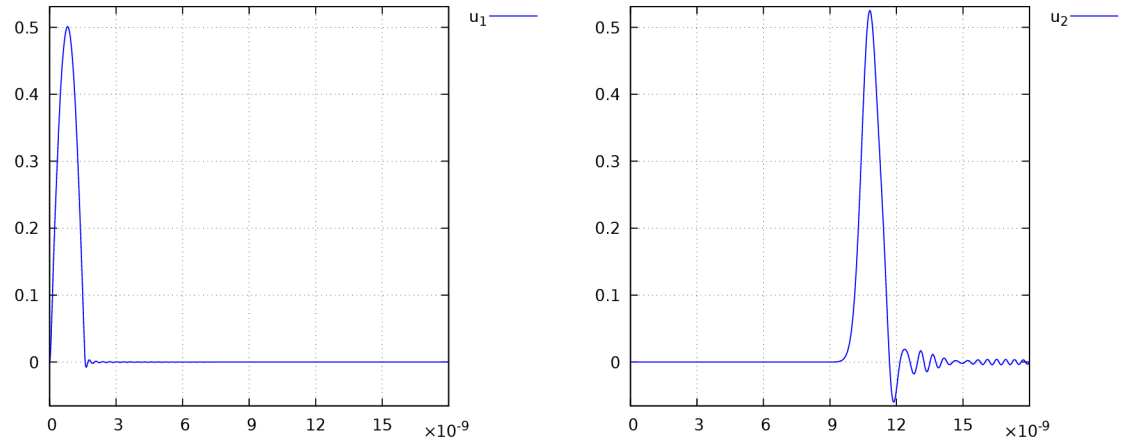
If the line is adjusted, then the results obtained are as expected – no bounces occur, the amplitude of the output voltage is almost the same as the amplitude of the input voltage, and the output signal is only delayed. Figure 4.17 shows the solution for a source of harmonic input signal with amplitude $A = 1$ V and angular frequency $\omega = 2 \cdot 10^9$ rad/s.



(a) Input voltage (b) Output voltage

Figure 4.17: Adjusted telegraph line – response for harmonic signal

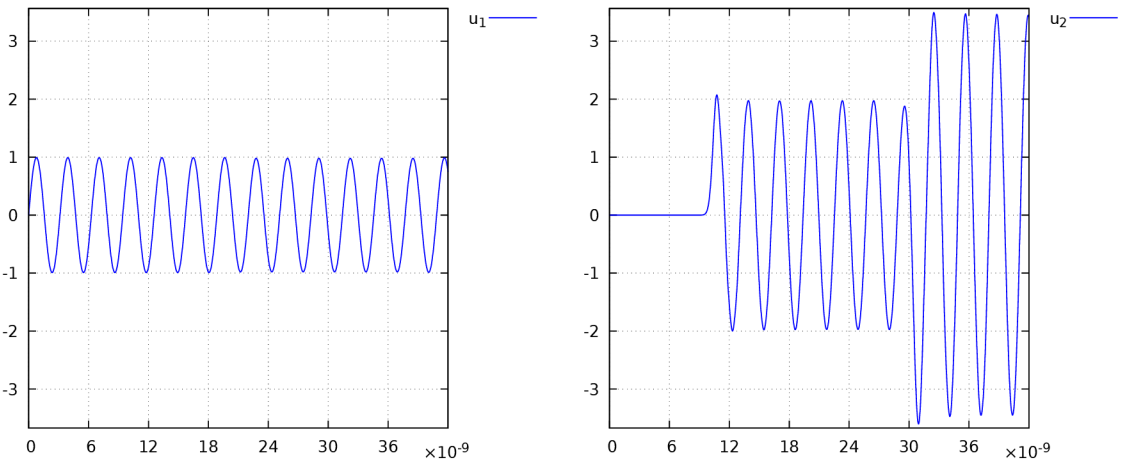
If the input voltage is an impulse modeled by one half of the sine wave (the values of the parameters are the same as in the previous example), the solution in Figure 4.18 is obtained.



(a) Input voltage (b) Output voltage

Figure 4.18: Adjusted telegraph line – response for impulse

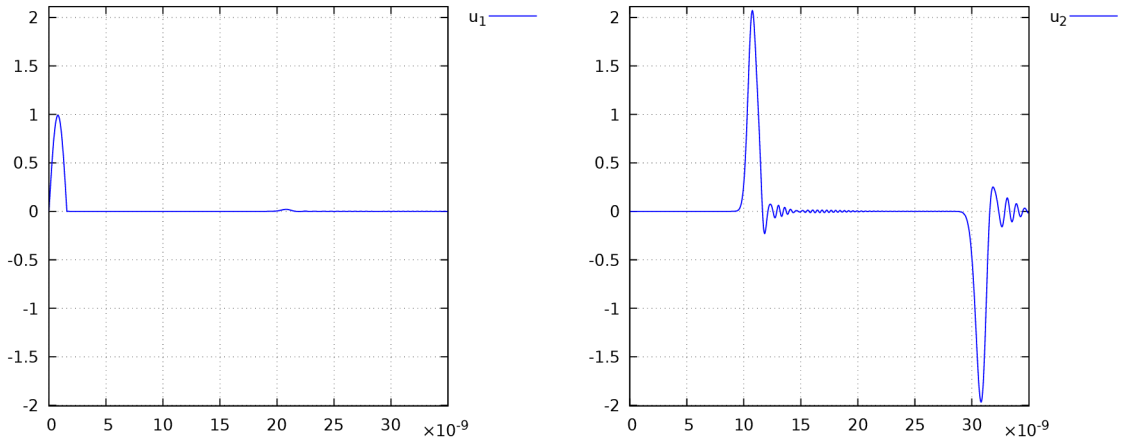
If we change $R_1 = 1\ \Omega$ and $R_2 = \infty\ \Omega$ (the circuit is opened from the right, ∞ is modeled as $10^{12}\ \Omega$), the solution in Figure 4.19 is obtained. The output signal is delayed and amplified (this can cause damage to the connected equipment).



(a) Input voltage (b) Output voltage

Figure 4.19: Open telegraph line – response for harmonic signal

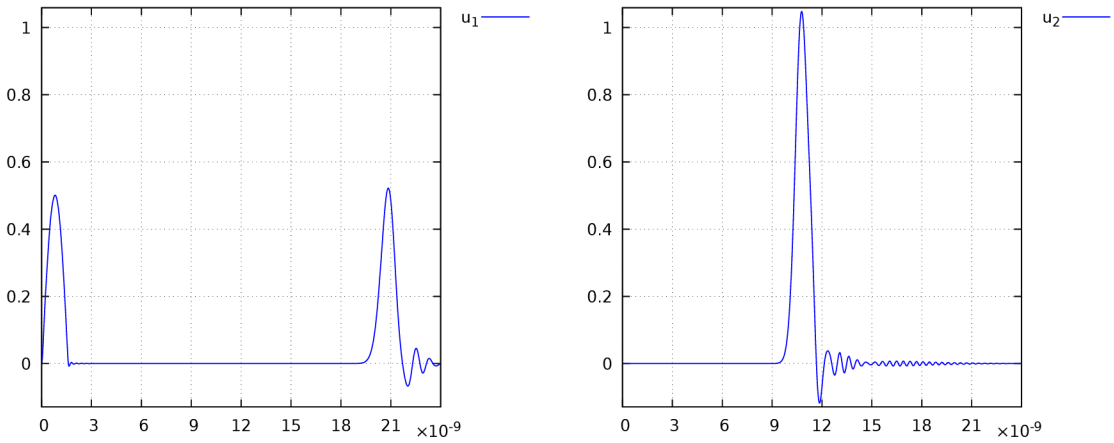
Again, if the source of voltage is changed to an impulse input, the result is shown in Figure 4.20.



(a) Input voltage (b) Output voltage

Figure 4.20: Open telegraph line – response for impulse

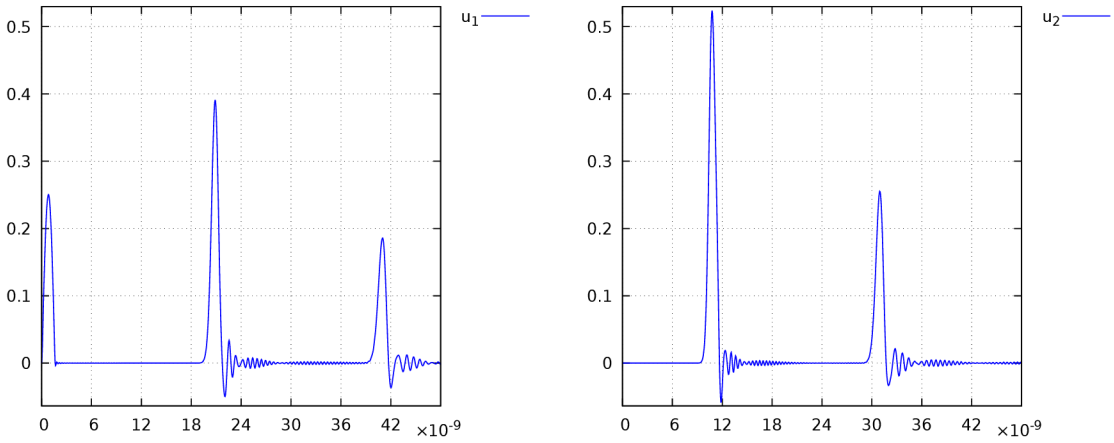
For $R_1 = 100\ \Omega$ and $R_2 = \infty\ \Omega$, the result is rather unexpected. The input impulse bounces back, see Figure 4.21. This behavior can be used to detect where the line is cut.



(a) Input voltage (b) Output voltage

Figure 4.21: Open telegraph line with $R_1 = 100\ \Omega$

If we raise $R_1 = 300\ \Omega$, the impulse bounced back even has a higher amplitude than the input impulse, see Figure 4.22.



(a) Input voltage (b) Output voltage

Figure 4.22: Open telegraph line with higher R_1

4.5 Parallel methods

A few parallel methods are mentioned in this section. A comparison of the time consumption of parallel and serial approaches is summarized in tables. A system of differential–algebraic equations can be represented by the block scheme in Figure 4.23.

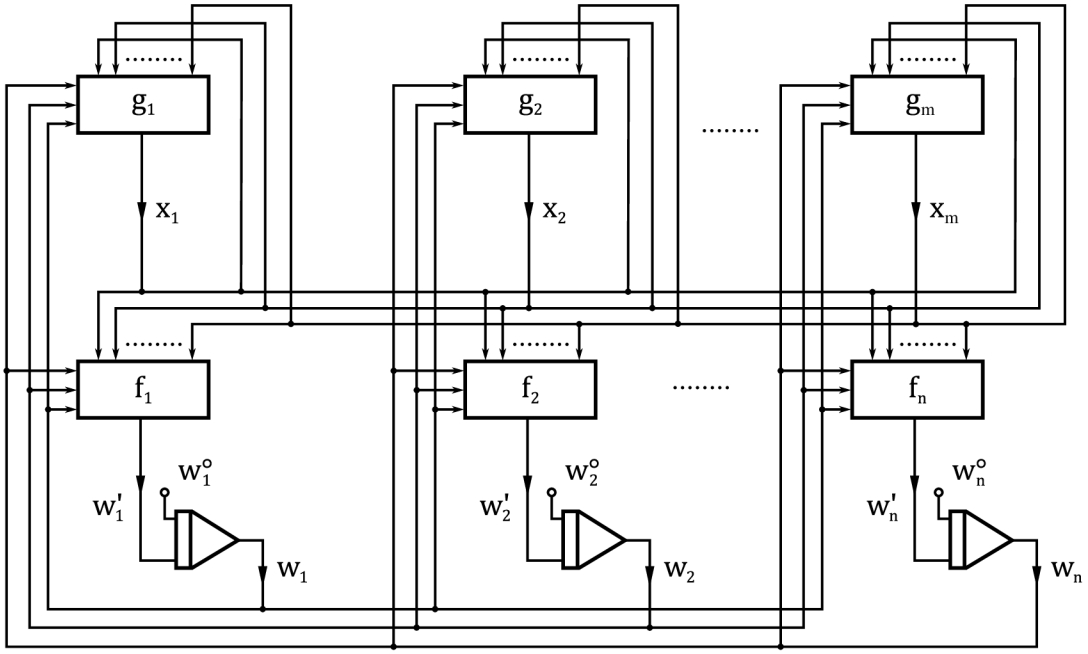


Figure 4.23: Parallel cooperation [36]

All integrators work concurrently; therefore, the calculation can be performed in separate threads of a processor. The communication among the threads consists only of state values.

4.5.1 Generic parallelization

It can be observed from Figure 4.23 that the independent parts could be calculated in parallel. The first approach to the parallel solution is based on solving the individual equations in parallel. Typically, a server contains several processors (M) that have several cores (N). Processors can support hyper-threading (multiple threads per core) – the number of threads (O) is typically two or one (the latter for processors without support for hyper-threading). Therefore, it is possible to launch $M \cdot N \cdot O$ software threads which wait for the work provided by the master thread.

For this approach to parallelization, it is essential that no thread uses any resources which can be changed by other threads. This behavior can be attained by enabling the possibility of modifying only the part of the memory used by the equation assigned to the thread which contains higher derivatives (all except the first one). The memory containing the first derivative and the state value and the memory of other equations are marked as unchangeable. Therefore, the first derivative expressions of all equations are strictly constant and therefore usable by all threads without locking. Higher derivatives are not

required by other threads – their calculation is hidden since every derivative contains only variables and all derivatives are expanded.

This generic approach separates the parallelization from the method used for calculation; therefore, it can be used for any method (Runge–Kutta, Adams–Bashforth, MTSM etc.) but the implementation of the method has to respect these restrictions.

This approach and its advantages can be demonstrated using the model of the telegraph line. First, the calculation times of a serial solution were analyzed. The experiments were undertaken 33 times; the medians of the calculation times measured are shown in Table 4.1. The first column contains numbers of segments. The next two columns (53 and 64 bits) show the durations when internal number types (`double` and `long double`) are used. The remaining columns were obtained using arbitrary precision arithmetic (library MPFR [19]). It is obvious from the table that the number of segments S and arithmetic width significantly influence the calculation times.

S\bits	**53**	**64**	**128**	**256**
100	1.212	1.347	2.895	3.589
200	5.333	5.966	9.835	11.194
300	12.189	13.346	20.752	22.720
400	21.513	23.316	35.591	38.160
500	33.409	35.991	54.260	57.449
600	47.830	51.161	76.607	80.580
700	64.667	68.992	102.437	107.671
800	83.757	89.799	131.749	138.469
900	105.252	113.223	165.394	173.104
1000	128.834	138.929	202.695	211.560

Table 4.1: Serial solution [s]

Further, the calculation times of the parallel solution were measured. The experiments were again undertaken 33 times, this time using 24 threads, on the same server as before; the advantage of six processors with two cores each with two threads was now fully utilized. The calculation times are shown in Table 4.2.

S\bits	**53**	**64**	**128**	**256**
100	0.443	0.381	0.656	0.830
200	1.749	1.469	1.958	2.113
300	3.923	3.220	4.022	3.907
400	6.933	5.708	6.883	6.321
500	10.808	8.814	10.327	9.182
600	15.463	12.723	14.677	12.768
700	20.971	17.316	19.596	16.822
800	27.376	22.327	25.397	21.512
900	34.606	28.319	31.710	26.300
1000	42.697	35.220	38.787	32.354

Table 4.2: Parallel solution [s]

Table 4.2 shows that the parallel solution is significantly faster than the serial one. The acceleration of parallel to serial solution is shown in Table 4.3.

S\bits	53	64	128	256
100	2.736	3.535	4.413	4.324
200	3.049	4.061	5.023	5.298
300	3.107	4.145	5.160	5.815
400	3.103	4.085	5.171	6.037
500	3.091	4.083	5.254	6.257
600	3.093	4.021	5.220	6.311
700	3.084	3.984	5.227	6.401
800	3.060	4.022	5.188	6.437
900	3.041	3.998	5.216	6.582
1000	3.017	3.945	5.226	6.539

Table 4.3: Acceleration

As Table 4.3 shows, the acceleration does not correspond to the number of the threads launched. The problem can be found in shared resources, like the operational memory, limited processor/motherboard caches, narrow buses etc. However, a considerable acceleration is achieved.

4.5.2 Acceleration for linear ODEs

Generic parallelization achieves a significant acceleration – but for linear ODEs, the calculation can be accelerated even more. The main problem with parallel calculation is the inter-thread communication that slows the calculation down significantly. If the communication is avoided, the solution can be obtained faster. Very interesting results were published in [41].

Adaptation of the generic formula

The generic formula of explicit MTSM for the calculation of the next values

$$y(t_n + h_n) \doteq \sum_{k=0}^{ORD} \frac{y^{(k)}(t_n)}{k!} h_n^k, \tag{4.8}$$

where ORD is maximal order, can be easily adapted for parallel computations, especially for systems of linear differential equations. These systems can be described by (4.9):

$$\boldsymbol{y}' = \boldsymbol{A} \cdot \boldsymbol{y} + \boldsymbol{b} \tag{4.9}$$

where $\boldsymbol{A}$ is a Jacobian matrix and $\boldsymbol{y}, \boldsymbol{b}$ are column vectors. This form can be used for a clear description of very large systems.

Higher derivatives can be expressed:

$$\begin{aligned} \boldsymbol{y}'' &= \boldsymbol{A} \cdot \boldsymbol{y}' = \boldsymbol{A}^2 \cdot \boldsymbol{y} + \boldsymbol{A} \cdot \boldsymbol{b} \\ \boldsymbol{y}^{(k)} &= \boldsymbol{A}^k \cdot \boldsymbol{y} + \boldsymbol{A}^{k-1} \cdot \boldsymbol{b} \end{aligned} \tag{4.10}$$

and after substituting the expression (4.10) into the formula (4.8), (4.11) is obtained.

$$\boldsymbol{y}(t_n + h_n) \doteq \boldsymbol{y}(t_n) + \sum_{k=1}^{ORD} \frac{\boldsymbol{A}^k \cdot \boldsymbol{y}(t_n) + \boldsymbol{A}^{k-1} \cdot \boldsymbol{b}}{k!} h_n^k \tag{4.11}$$

If the sum is split, (4.12) is obtained

$$\boldsymbol{y}(t_n + h_n) \doteq \left(\boldsymbol{I} + \sum_{k=1}^{ORD} \frac{\boldsymbol{A}^k}{k!} h_n^k\right) \cdot \boldsymbol{y}(t_n) + \sum_{k=1}^{ORD} \frac{\boldsymbol{A}^{k-1} \cdot \boldsymbol{b}}{k!} h_n^k \tag{4.12}$$

and for fixed step size h:

$$\boldsymbol{y_{n+1}} = \left(\sum_{k=0}^{ORD} \frac{\boldsymbol{A}^k}{k!} h^k\right) \cdot \boldsymbol{y_n} + \sum_{k=1}^{ORD} \frac{\boldsymbol{A}^{k-1} \cdot \boldsymbol{b}}{k!} h^k \tag{4.13}$$

which can be transformed into

$$\boldsymbol{y_{n+1}} = \hat{\boldsymbol{A}} \cdot \boldsymbol{y_n} + \hat{\boldsymbol{b}} \tag{4.14}$$

where $\hat{\boldsymbol{A}}$ is a transformed matrix and $\hat{\boldsymbol{b}}$ is a transformed vector.

$$\begin{aligned} \hat{\boldsymbol{A}} &= \sum_{k=0}^{ORD} \frac{\boldsymbol{A}^k}{k!} h^k \\ \hat{\boldsymbol{b}} &= \sum_{k=1}^{ORD} \frac{\boldsymbol{A}^{k-1} \cdot \boldsymbol{b}}{k!} h^k \end{aligned} \tag{4.15}$$

The conversion into the parallel version is now evident. Partial sums

$$\boldsymbol{A_l} = \sum_{k=0}^{\frac{ORD}{N}-1} \frac{\boldsymbol{A}^{Nk+l-1}}{(Nk+l)!} h^{Nk+l} \tag{4.16}$$

are computed in N threads[3] and there is no need for any communication or synchronization between threads except for loading the matrix at the beginning and handing back computed partial sums at the end. The reason for decreasing the exponent by one is single computation of partial sums for both the transformed matrix and the transformed vector (otherwise, double calculation would have to be performed).

The final transformed matrix $\hat{\boldsymbol{A}}$ and transformed vector $\hat{\boldsymbol{b}}$ are calculated afterwards (the sum is calculated only once):

$$\begin{aligned} \hat{\boldsymbol{A}} &= \left(\sum_{l=1}^{N} \boldsymbol{A_l}\right) \cdot \boldsymbol{A} + \boldsymbol{I} \\ \hat{\boldsymbol{b}} &= \left(\sum_{l=1}^{N} \boldsymbol{A_l}\right) \cdot \boldsymbol{b} \end{aligned} \tag{4.17}$$

Comparison of the approaches

As the problem of the telegraph line is linear, the problem can be also described using only matrix $\boldsymbol{A}$ and vectors $\boldsymbol{b}$ and $\boldsymbol{y_0}$, where $\boldsymbol{y_0}^T = (u_0, v_0, u_{C_1}^0, i_{L_1}^0, \ldots, i_{L_N}^0)$ is a vector of the

[3] 1-based indexing is used.

initial conditions.

$$\boldsymbol{A} = \begin{pmatrix} 0 & \omega & 0 & 0 & 0 & 0 & \dots & 0 \\ -\omega & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ \frac{1}{R_1 C} & 0 & -\frac{1}{R_1 C} & -\frac{1}{C} & 0 & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{L} & 0 & -\frac{1}{L} & 0 & \dots & 0 \\ 0 & 0 & 0 & \frac{1}{C} & 0 & -\frac{1}{C} & \dots & 0 \\ \vdots & & \ddots & & \ddots & & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \frac{1}{C} & 0 & -\frac{1}{C} \\ 0 & 0 & 0 & 0 & 0 & \dots & \frac{1}{L} & -\frac{R_2}{L} \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \quad \boldsymbol{y_0} = \begin{pmatrix} 0 \\ U \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

The calculation times for linear systems which were computed in parallel are shown in Table 4.4.

S\bits	**53**	**64**	**128**	**256**
100	0.473	0.410	0.538	0.693
200	1.749	1.447	1.457	1.501
300	3.862	3.165	2.895	2.637
400	6.735	5.572	4.794	4.045
500	10.369	8.602	7.197	5.597
600	14.938	12.385	10.108	7.470
700	20.290	16.786	13.474	9.561
800	26.360	21.861	17.163	11.867
900	33.077	27.682	21.410	14.443
1000	41.212	34.216	26.025	17.169

Table 4.4: Parallel solution of linear ODEs [s]

Table 4.5 compares both approaches to parallelization – the generic approach (see Section 4.5.1) and acceleration for linear ODEs[4] (Section 4.5.2). The acceleration is already not as high; moreover there is a limitation for linear systems of ODEs.

S\bits	**53**	**64**	**128**	**256**
100	0.937	0.929	1.219	1.198
200	1.000	1.015	1.344	1.408
300	1.016	1.017	1.389	1.482
400	1.029	1.024	1.436	1.563
500	1.042	1.025	1.435	1.641
600	1.035	1.027	1.452	1.709
700	1.034	1.032	1.454	1.759
800	1.039	1.021	1.480	1.813
900	1.046	1.023	1.481	1.821
1000	1.036	1.029	1.490	1.884

Table 4.5: Acceleration of linear heuristic

[4]Performed together with generic parallelization.

The calculation times of the parallel computation with heuristic for linear systems compared to the calculation with no acceleration are shown in Table 4.6.

S\bits	**53**	**64**	**128**	**256**
100	2.562	3.285	5.381	5.179
200	3.049	4.123	6.750	7.458
300	3.156	4.217	7.168	8.616
400	3.194	4.184	7.424	9.434
500	3.222	4.184	7.539	10.264
600	3.202	4.131	7.579	10.787
700	3.187	4.110	7.603	11.261
800	3.177	4.108	7.676	11.668
900	3.182	4.090	7.725	11.985
1000	3.126	4.060	7.788	12.322

Table 4.6: Total acceleration

The acceleration is relatively high; the more bits for numbers used, the greater the acceleration attained – a larger step size can be chosen.

Chapter 5

Solving Electronic Circuits

The solution of electric circuits was discussed in the previous chapter. This chapter focuses on electronic circuits. These circuits contain not only resistors, capacitors and coils, but also semiconductors.

5.1 Semiconductors

Semiconductor components are described in this section. The analysis of a diode is performed first, then a transistor is also analyzed.

5.1.1 Diode

A diode is defined via exponential Volt-Ampere characteristic that can be expressed by

$$i_d \;=\; a \cdot \left(e^{b \cdot u_d} - 1 \right) \tag{5.1}$$

where a and b are material parameters (e. g. $a = 10^{-18}$, $b = 50$). Consider the simple electronic circuit in Figure 5.1.

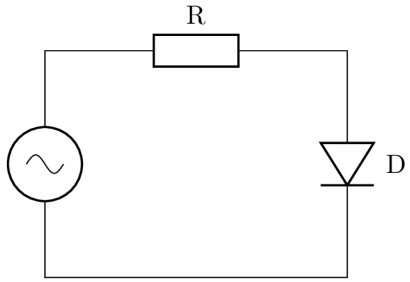


Figure 5.1: Electronic circuit with diode

This circuit can be solved iteratively, for example using the Newton–Raphson method – see Figure 5.2; the numbers represent the order in which the tangents/normals are constructed – for more see [38].

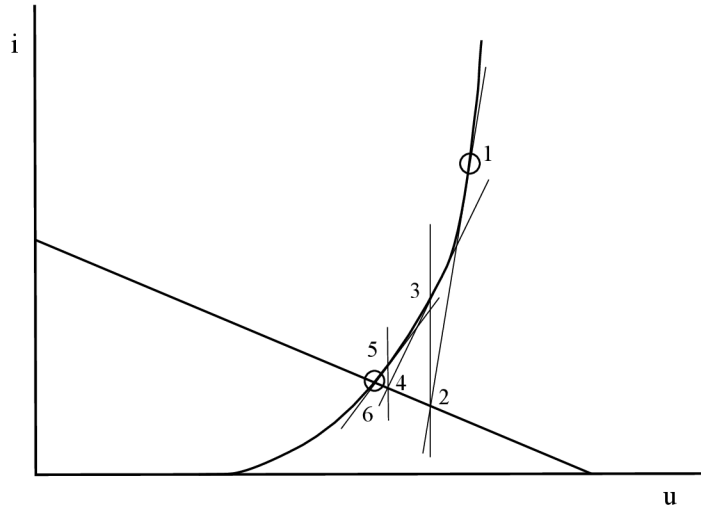


Figure 5.2: Solution by Newton–Raphson method

This example can also be solved numerically. First, the expression (5.1) is differentiated:

$$i'_d \;=\; a \cdot e^{b \cdot u_d} \cdot b \cdot u'_d, \tag{5.2}$$

then the exponential is substituted with the expression $i_d + a$:

$$i'_d \;=\; (i_d + a) \cdot b \cdot u'_d, \tag{5.3}$$

further, voltage u_d is expressed and substituted into (5.3):

$$i'_d \;=\; (i_d + a) \cdot b \cdot u' - (i_d + a) \cdot b \cdot R \cdot i'_d \tag{5.4}$$

and at the end, the expression (5.4) is adjusted to the explicit form:

$$i'_d \;=\; \frac{b\,(i_d + a)}{1 + bR\,(i_d + a)} u', \quad i_d(0) \;=\; 0. \tag{5.5}$$

The problem is, of course, with the derivative of input voltage, but it can be solved by the method of generating differential equations. If the input voltage is a harmonic sine signal $u = A \cdot \sin(\omega t)$, where amplitude $A = 1$ V and frequency $\omega = 1$ rad/s, the system of three ordinary differential equations (5.6) is obtained (in which the automatic transformation has already been performed). For another input voltage, e. g. $u = 1 - e^{-\frac{t}{\tau}}$, the procedure would be analogical.

$$\begin{aligned} i'_d &= \frac{b\,(i_d + a)}{1 + bR\,(i_d + a)} v, & i_d(0) &= 0 \\ u' &= v, & u(0) &= 0 \\ v' &= -u, & v(0) &= 1 \end{aligned} \tag{5.6}$$

By solving the system (5.6), the result in Figure 5.3 is obtained.

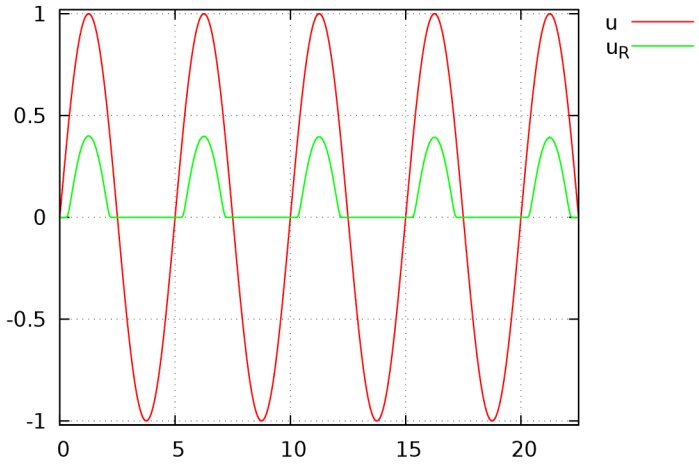


Figure 5.3: Solution of system

5.1.2 Transistor

The transistor can be solved analogically to the diode, using the Newton–Raphson method. The solution is illustrated in Figure 5.4 – the numbers represent the order in which the tangents/normals are constructed.

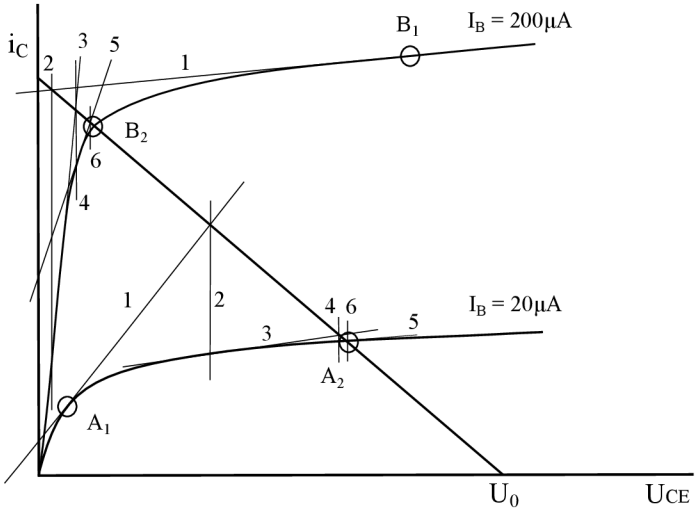


Figure 5.4: Solution of electronic circuit with transistor

5.2 CMOS

This section of the thesis is based on [1]. Complementary Metal–Oxide–Semiconductor (CMOS) technology is assumed since it is the most widely used technology in electronics. The idea and the basic concepts of the CMOS circuits design were invented by Frank Wanlass. The fundamental idea of using complementary MOS devices (positive PMOS and

negative NMOS transistors) was quite novel at the time due to the rising popularity of the Bipolar Junction Transistor as a replacement for the vacuum tube.

Today, the CMOS technology is still dominant for manufacturing integrated circuits. It is likely that it will be dominant for the foreseeable future since CMOS transistors are manufacturable, have low power requirements, are low-cost and scalable. This scalability was first observed and described by Gordon Moore (founder of Intel, 1965) in Moore's law. This law states that the number of devices on a chip will double every 18 to 24 months [32]. The gate lengths of the initial CMOS transistors were considerably longer than they are today and an increasing number of smaller transistors can be fitted onto the chip.

CMOS devices have high noise immunity and low static power consumption. One transistor of the pair is always off and the significant power draw occurs while switching between states (on and off). It is primarily this behavior that makes the CMOS technology useful for the implementation of VLSI circuits.

See [9] for more information about CMOS, [25] for more general information about the construction and design of analog and hybrid computers, and [29] about the system on a chip and integrated design. The fabrication process is explained in [45].

5.3 Approaches to VLSI simulation

In this section, the approaches to Very Large-Scale Integration (VLSI) simulation are briefly discussed. Two different approaches are mentioned – SPICE and FOS.

5.3.1 SPICE

SPICE is widely used for analog circuits simulation since it can compute the full large-signal behavior of arbitrary circuits. SPICE uses a few numerical methods for numerical integration. The Newton integration method is suitable for finding the solution of circuits with non-linear elements. The sparse matrix method is used to save memory by storing only non-zero elements. The implicit integration method is used to integrate the differential equations that describe the circuit reactances.

Numerical integration is necessary for analog circuits simulation. SPICE uses second order integration methods. Most SPICE implementations follow Berkeley SPICE and provide two forms of second order implicit integration: Gear and trapezoidal. Trapezoidal integration is both faster and more accurate than Gear; however, trapezoidal integration can cause numerical artifacts. These artifacts manifest themselves as an oscillation around the precise solution in each time step. See [13] for more information.

5.3.2 FOS

VLSI circuits were initially simulated in Fast ODE Solver (FOS) [24], which was primarily designed for the solution of general ODEs with the integrated support of arbitrary precision arithmetic. FOS supports several numerical methods including MTSM, which is used in the thesis.

General ODEs do not need to be reassembled very often. In contrast, the ODEs describing VLSI circuits have to be reassembled frequently. For example, the ODEs in (5.7) have to be reassembled whenever the input changes from true to false and vice versa. Using selective reassembly, the computation was accelerated 20–50 times. Due to this acceleration, it was possible to simulate the 512-bit adder (almost VLSI) in approximately 90 minutes.

As this acceleration was not sufficient, a specialized system was developed for VLSI simulation. Thanks to this system, a circuit with over 1 million transistors was simulated in approximately 180 minutes using 7.5 GB of RAM.

The three-address instructions that accelerate the computation in FOS were further omitted because of high memory usage. CSM produces a system of linear ODEs; each MTSM term is calculated from the previous term. Thanks to this approach, the calculation of the same system now uses less memory – less than 320 MB (in contrast to the previous 7.5 GB) – and moreover, it is faster. When calculating the voltage MTSM terms, only one addition and two multiplications are used. When calculating the MTSM terms of the current, the number of additions is the same as the number of inputs.

5.4 Capacitor Substitution Method

In this section, the Capacitor Substitution Method (CSM) is introduced. It is a sophisticated approximation of electronic circuits consisting of CMOS transistors by electric circuits that consist only of capacitors and resistors. These circuits are suitable for further simulation.

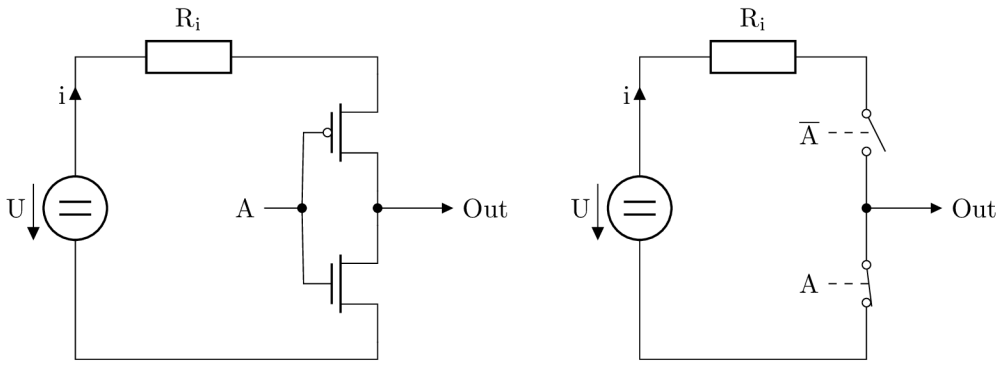
The general purpose transistors N3306M and P3306M were chosen for simulation. The corresponding SPICE models of these transistors follow[1].

```
1 .MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
2 + CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
3 .MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
4 + CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
```

The behavior of SPICE models was taken as the reference output. The basic logic gates are modeled using CSM as described below.

5.4.1 CMOS inverter

Figure 5.5a presents the scheme of a CMOS inverter. The inverter consists of PMOS and NMOS transistors. The function of this scheme can be demonstrated by the electric circuit in Figure 5.5b. This logic gate is necessary for AND and OR gates construction when De Morgan's laws cannot be used – for example, in the case of CLA (see next chapter).



(a) Electronic circuit (b) Electric analogy

Figure 5.5: Inverter [44]

[1] Retrieved from `http://www.datasheetarchive.jp/`.

Logical one is represented by high voltage (3.3 V for recent CMOS transistors[2]); logical zero is represented by low voltage (0 V). The behavior of the inverter is as follows:

– if $A = 1$, the PMOS transistor (upper one) is closed and the NMOS is open;

– if $A = 0$, the PMOS transistor is open and the NMOS is closed.

The corresponding SPICE model of the inverter is shown as the abbreviated SPICE netlist below (the capacitor smooths the output). The full version of the SPICE source code can be found in Appendix D.1.

```
1 Vdd 1 0 DC 3.3
2 Ri 2 1 0.1
3 * A = 1, 0
4 V3 3 0 PWL(0 3.3 1e-07 3.3 1e-07 0 2e-07 0)
5 * not(A)
6 M4a 4 3 2 2 P3306M
7 M4b 4 3 0 0 N3306M
8 C4c 4 0 1p
```

Figure 5.6 shows the output of the inverter using SPICE. The input is logical one in the interval $t \in \langle 0, 10^{-7}) $ [s] and logical zero otherwise. The expected result is: if input A is logical one, then Out corresponds to logical zero; if input A is logical zero, then Out corresponds to logical one.

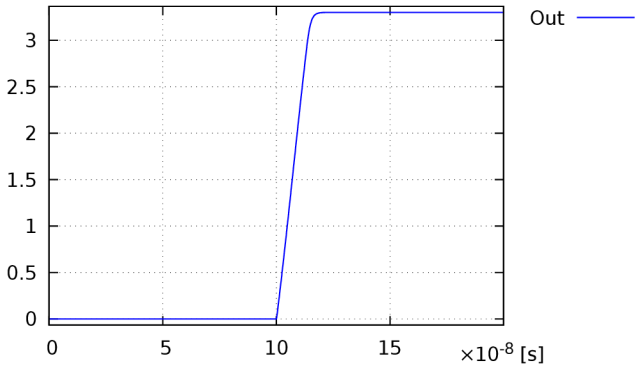


Figure 5.6: Inverter – SPICE [V]

The curve in Figure 5.6 is similar to the curve for charging the capacitor. To develop the substituting circuit, the basic concept from electric circuits simulation is used. Each transistor should be substituted by a capacitor which has the input voltage appropriate to the state of the transistor.

When the transistor is closed, the capacitor has to charge; when the transistor is open, the capacitor has to discharge. The input voltage can be controlled by switching the values of resistors, R_L for an open transistor and R_H for a closed transistor. The values of the resistors are denoted as R_A, $R_{\overline{A}}$, R_B or $R_{\overline{B}}$ depending on the value controlling the switch. Therefore, the substitution in Figure 5.7 is performed.

[2]The value depends on the logic used.

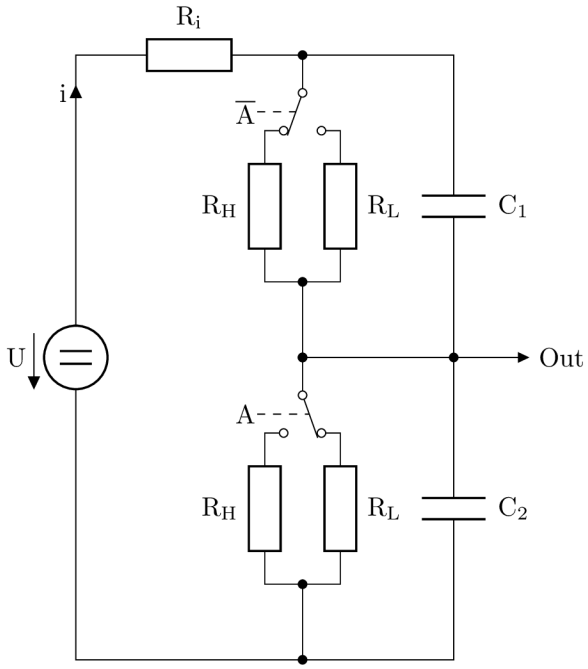


Figure 5.7: Inverter – substituted by CSM

If input A is logical one[3], the PMOS transistor is closed – the upper part of the circuit is switched into the high-resistor branch (with R_H) – and the NMOS transistor is open – the lower part is switched into the low-resistor branch (with R_L). The system of ODEs (5.7) for this regular electric circuit can be constructed (the first algebraic equation is in explicit form; therefore, value i can be used directly in other equations); capacitor C_1 is precharged to avoid a considerable initial transient response.

$$\begin{aligned} i &= \frac{1}{R_i} \cdot \left(U - u_{C_1} - u_{C_2} \right) \\ u'_{C_1} &= \frac{1}{C_1} \cdot \left(i - \frac{1}{R_{\overline{A}}} \cdot u_{C_1} \right), \quad u_{C_1}(0) = 3.3 \\ u'_{C_2} &= \frac{1}{C_2} \cdot \left(i - \frac{1}{R_A} \cdot u_{C_2} \right), \quad u_{C_2}(0) = 0 \end{aligned} \tag{5.7}$$

Parameters R_i, C_1, C_2, R_L and R_H can be determined using the MATLAB function `greyest`. This function can estimate the parameters of linear models to correspond with the SPICE model. The system is described by

$$\begin{aligned} \boldsymbol{x}' &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \\ \boldsymbol{y} &= \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u} \end{aligned} \tag{5.8}$$

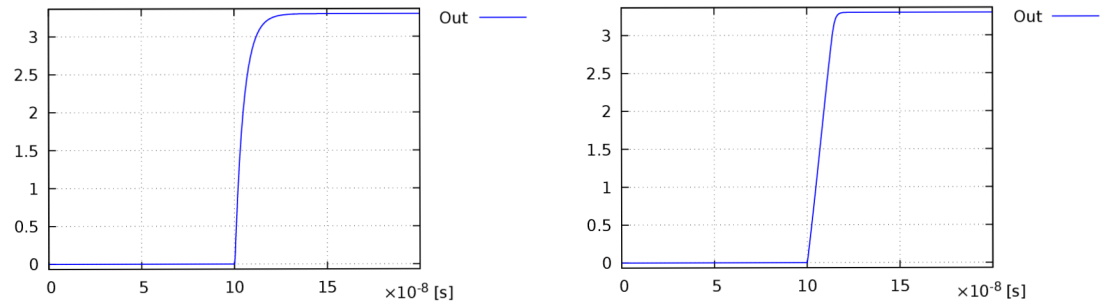
where $\boldsymbol{x}$ is a vector of variables, $\boldsymbol{A}$ is a Jacobian matrix, $\boldsymbol{B}$ is a matrix/vector of constants, $\boldsymbol{u}$ is a vector/scalar of inputs and $\boldsymbol{C}$ and $\boldsymbol{D}$ define how to evaluate output value $\boldsymbol{y}$. The expressions in (5.9) describe the transient response of the inverter when toggling the output

[3]That is more than half the nominal voltage value.

from logical zero to logical one.

$$\begin{aligned} \boldsymbol{A} &= \begin{pmatrix} -\frac{R_L+R_i}{C_1 R_L R_i} & -\frac{1}{C_1 R_i} \\ \frac{1}{C_2 R_i} & -\frac{R_H+R_i}{C_2 R_H R_i} \end{pmatrix} & \boldsymbol{B} &= \begin{pmatrix} \frac{U}{C_1 R_i} \\ \frac{U}{C_2 R_i} \end{pmatrix} \\ \boldsymbol{C} &= \begin{pmatrix} 0 & 1 \end{pmatrix} & \boldsymbol{D} &= 0 \\ \boldsymbol{x} &= \begin{pmatrix} u_{C_1} \\ u_{C_2} \end{pmatrix} & \boldsymbol{u} &= 1 \end{aligned} \tag{5.9}$$

Figure 5.8a shows the solution of (5.7) for parameters $U = 3.3$ V, $R_i = 0.120792\ \Omega$, $C_1 = C_2 = 3.851953 \cdot 10^{-9}$ F, $R_L = 0.601435\ \Omega$, $R_H = 10^{10}\ \Omega$. If input A is logical one (i. e. 0 ns $\leq$ *time* $<$ 100 ns), then *Out* corresponds to logical zero; if input A is logical zero (100 ns $\leq$ *time* $\leq$ 200 ns), then *Out* corresponds to logical one. Figure 5.8b (solution by SPICE) is included for comparison.



(a) CSM [V] (b) SPICE [V]

Figure 5.8: Inverter – solution

The error of the approximation is shown in Figure 5.9. The approximation error is relatively high (over 1 V) during the transient response, but the most important fact is that it is low in the stable state.

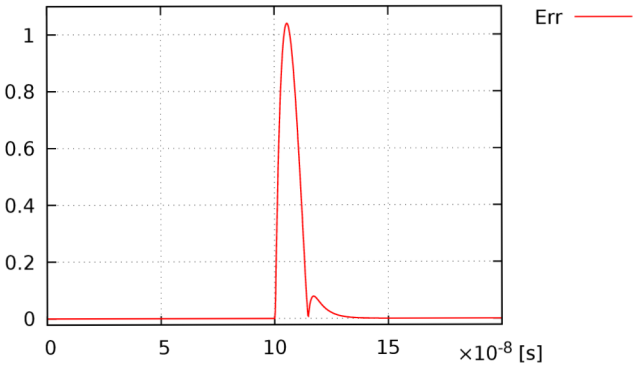
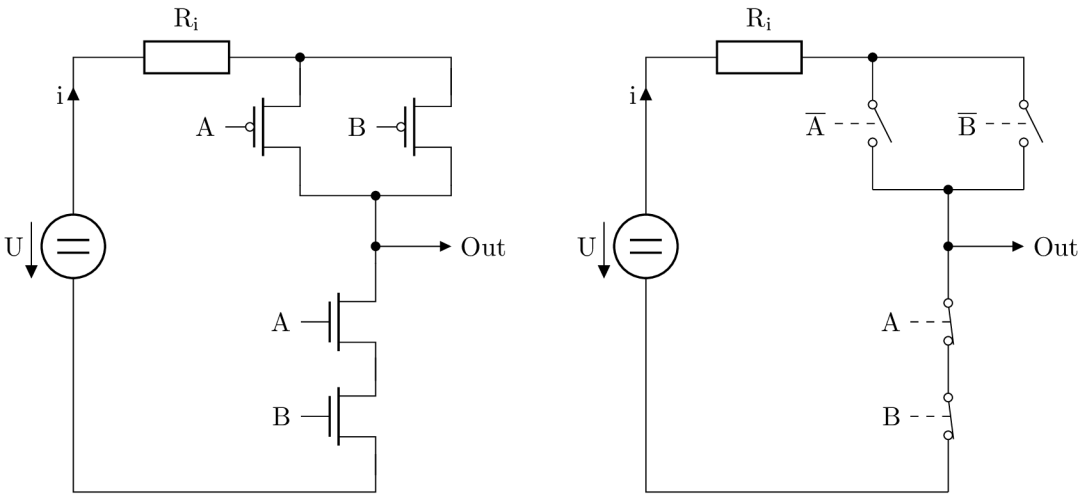


Figure 5.9: Inverter – approximation error [V]

The transient responses for CSM and SPICE correspond quite well. The difference between responses is caused by the approximation. The important aspect is that the resulting values after the transient response and the lengths of the transient responses are similar.

5.4.2 CMOS NAND

The scheme of CMOS NAND is shown in Figure 5.10a and the function of this electronic circuit is explained in Figure 5.10b. This logic gate forms the cornerstone of computer logic; any logic function can be constructed using only NANDs (with De Morgan's laws). The NAND consists of two parallel PMOS transistors and two serial NMOS transistors.



(a) Electronic circuit (b) Electric analogy

Figure 5.10: NAND [44]

The NAND logical inputs are given in Table 5.1. As mentioned earlier, they control the values of resistors (depending on opening/closing the transistors in the electronic circuit). The time domain is divided into equally long segments.

A	1	1	0	0
B	1	0	0	1

Table 5.1: NAND – input

CMOS NAND can be solved using the following SPICE netlist (abbreviated form; the full source code can be found in Appendix D.2):

```
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* A = 1, 1, 0, 0
V3 3 0 PWL(0 3.3 1e-07 3.3 1e-07 0 2e-07 0)
* B = 1, 0, 0, 1
V4 4 0 PWL(0 3.3 5e-08 3.3 5e-08 0 1.5e-07 0 1.5e-07 3.3 2e-07 3.3)
* nand(A, B)
M5a 5 3 2 2 P3306M
M5b 5 4 2 2 P3306M
M5c 5 3 6 6 N3306M
M5d 6 4 0 0 N3306M
C5e 5 0 1p
```

The transformation of NAND is analogical to the transformation of the inverter. The only difference is that it consists of two pairs of transistors. It is shown in Figure 5.11.

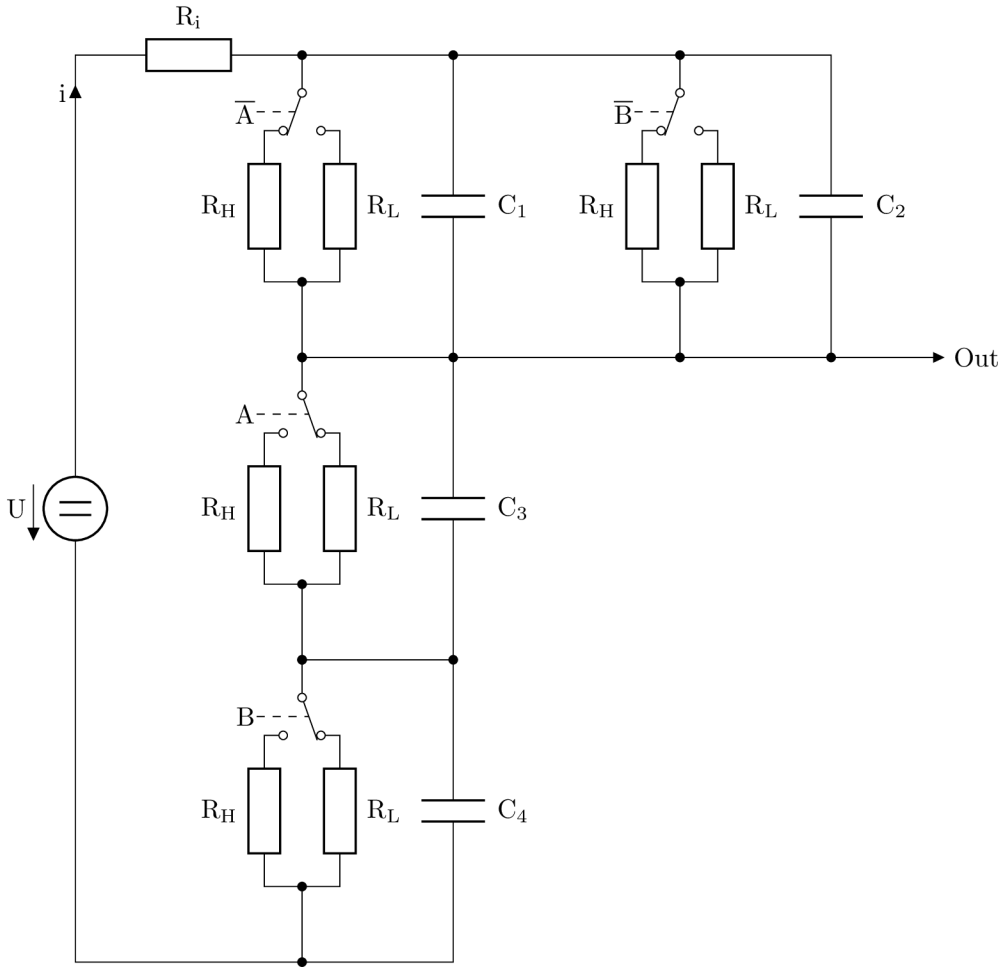


Figure 5.11: NAND – substituted by CSM

Capacitors C_1 and C_2 are parallel; therefore, they can be merged into one capacitor $C_{12} = C_1 + C_2$; see Figure 5.12.

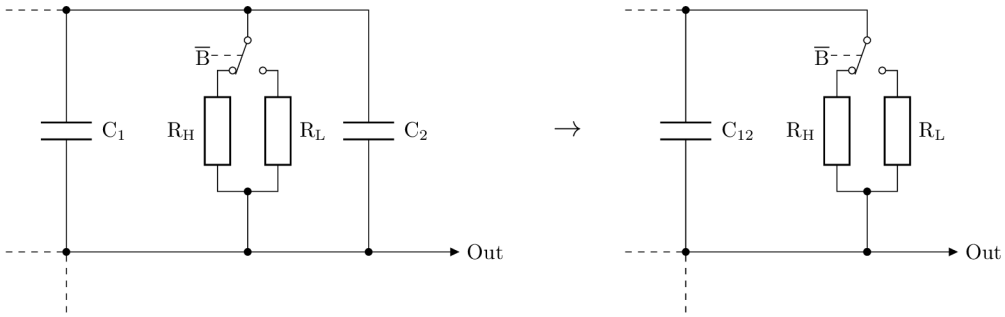
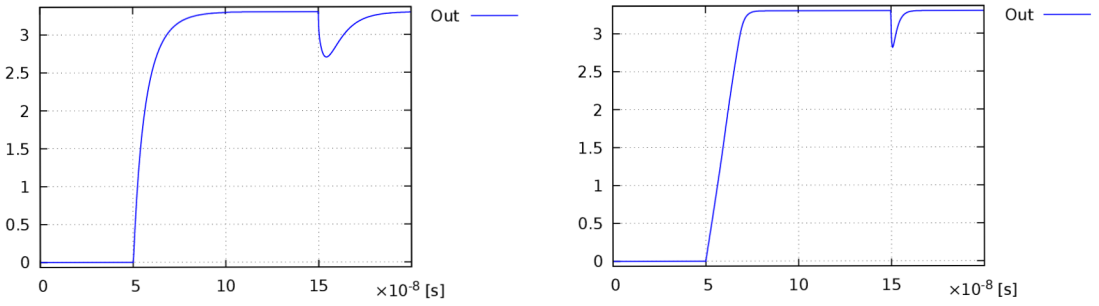


Figure 5.12: NAND – merging capacitors

The circuit in Figure 5.11 is described by (5.10)[4]; capacitor C_{12} is precharged to attain the initial value of zero.

$$\begin{aligned} i &= \frac{1}{R_i} \cdot \left(U - u_{C_{12}} - u_{C_3} - u_{C_4} \right) \\ u'_{C_{12}} &= \frac{1}{C_{12}} \cdot \left(i - \frac{R_{\overline{A}} + R_{\overline{B}}}{R_{\overline{A}} \cdot R_{\overline{B}}} \cdot u_{C_{12}} \right), \quad u_{C_{12}}(0) = 3.3 \\ u'_{C_3} &= \frac{1}{C_3} \cdot \left(i - \frac{1}{R_A} \cdot u_{C_3} \right), \quad u_{C_3}(0) = 0 \\ u'_{C_4} &= \frac{1}{C_4} \cdot \left(i - \frac{1}{R_B} \cdot u_{C_4} \right), \quad u_{C_4}(0) = 0 \end{aligned} \tag{5.10}$$

The solution for the values from Table 5.1 is shown in Figure 5.13. The transient responses in both figures begin at 50 ns and 150 ns. The circuit truly behaves like the NAND gate and the result using CSM is again virtually identical to the result obtained using SPICE; all transient responses reach a steady state by 50 ns.



(a) CSM [V] (b) SPICE [V]

Figure 5.13: NAND – solution

The error of the approximation is shown in Figure 5.14.

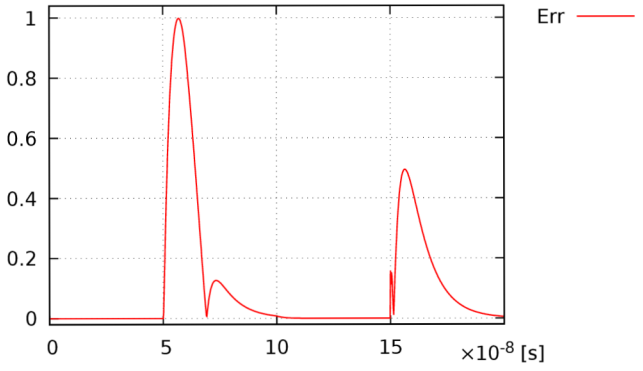
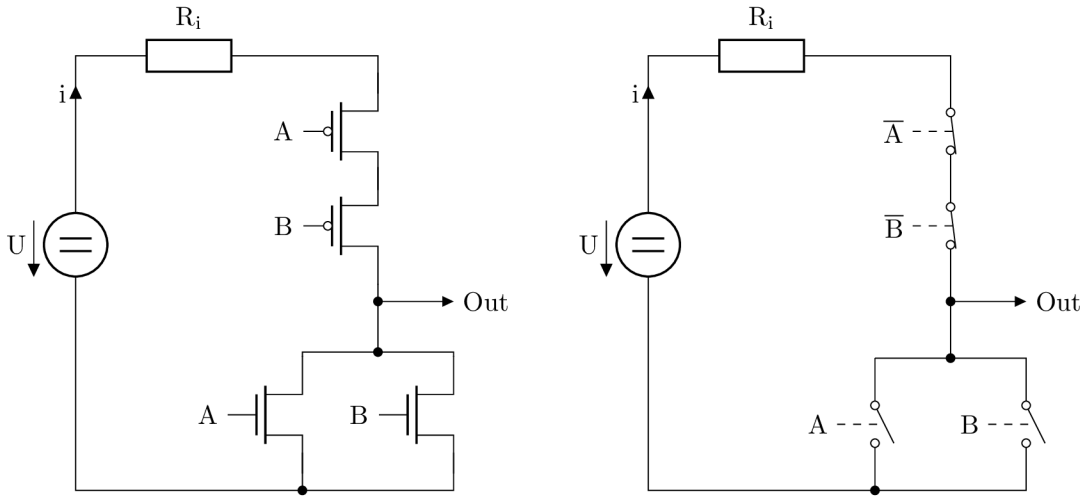


Figure 5.14: NAND – approximation error [V]

[4] $U = 3.3$ V, $R_i = 0.120792\ \Omega$, $C_1 = C_2 = C_3 = C_4 = 3.851953 \cdot 10^{-9}$ F, $C_{12} = C_1 + C_2$, $R_L = 0.601435\ \Omega$, $R_H = 10^{10}\ \Omega$

5.4.3 CMOS NOR

The scheme of CMOS NOR is shown in Figure 5.15a and the function of this electronic circuit is explained in Figure 5.15b. Similarly to the NAND gate, all other logic gates can be constructed using only NOR gates. The NOR consists of two serial PMOS transistors and two parallel NMOS transistors.



(a) Electronic circuit (b) Electric analogy

Figure 5.15: NOR [44]

The NOR logical inputs are given in Table 5.2. The time domain is split into equally long segments.

A	0	1	1	0
B	0	0	1	1

Table 5.2: NOR – input

The abbreviated SPICE netlist of CMOS NOR follows (the full version of the source code can be found in Appendix D.4).

```
1  Vdd 1 0 DC 3.3
2  Ri 2 1 0.1
3  * A = 0, 1, 1, 0
4  V3 3 0 PWL(0 0 5e-08 0 5e-08 3.3 1.5e-07 3.3 1.5e-07 0 2e-07 0)
5  * B = 0, 0, 1, 1
6  V4 4 0 PWL(0 0 1e-07 0 1e-07 3.3 2e-07 3.3)
7  * nor(A, B)
8  M5a 6 3 2 2 P3306M
9  M5b 5 4 6 6 P3306M
10 M5c 5 3 0 0 N3306M
11 M5d 5 4 0 0 N3306M
12 C5e 5 0 1p
```

The transformation of CMOS NOR is shown in Figure 5.16. Capacitors C_3 and C_4 are parallel; therefore, one capacitor $C_{34} = C_3 + C_4$ is used in equations.

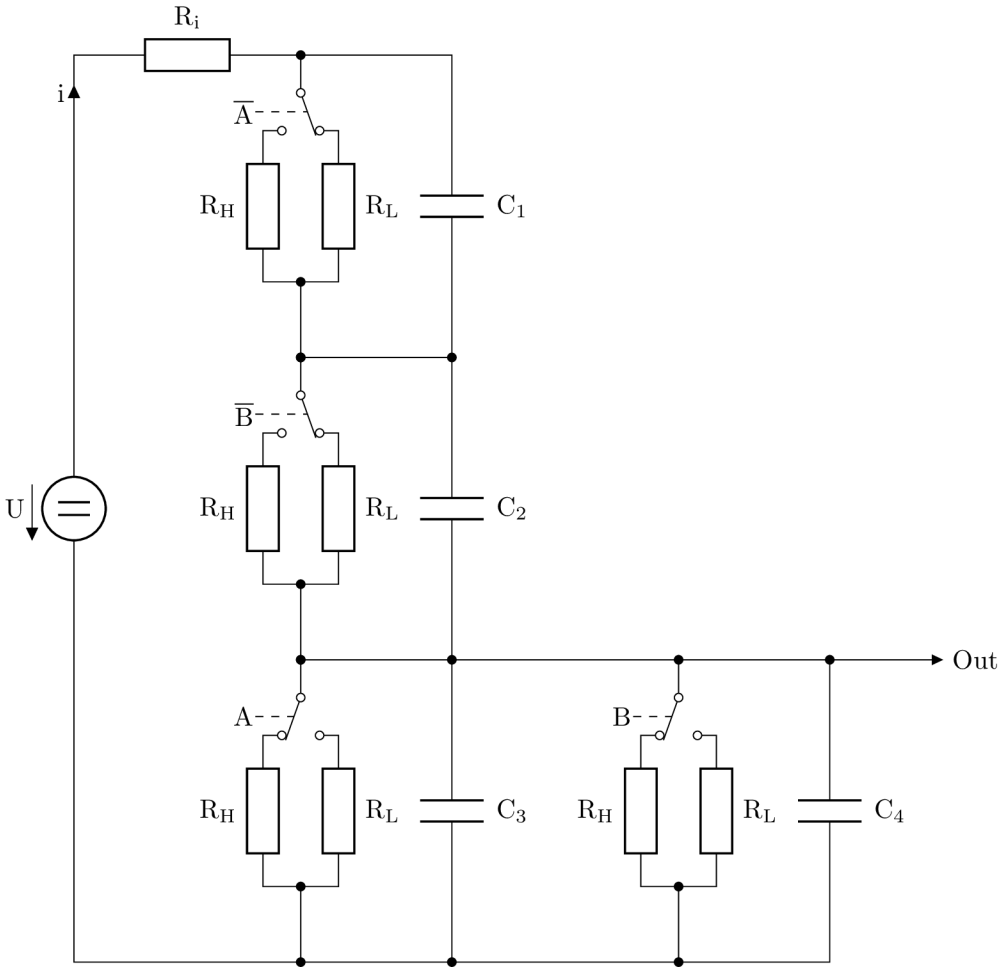


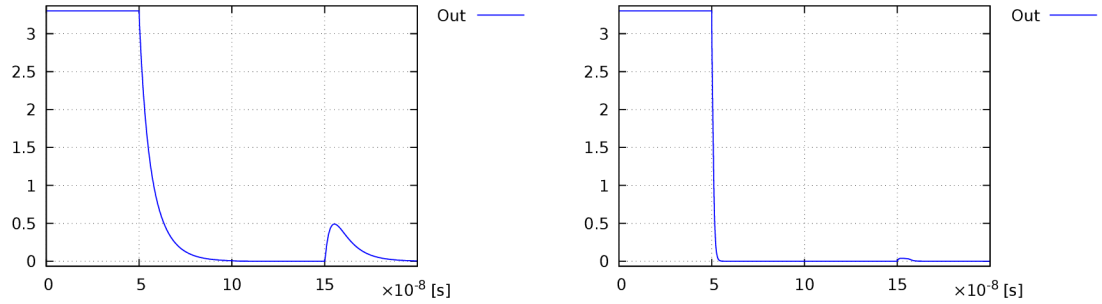
Figure 5.16: NOR – substituted by CSM

The circuit in Figure 5.16 is described by equations (5.11)[5]; capacitor C_{34} (merged from C_3 and C_4) is precharged.

$$\begin{aligned} i &= \frac{1}{R_i} \cdot \left(U - u_{C_1} - u_{C_2} - u_{C_{34}}\right) \\ u'_{C_1} &= \frac{1}{C_1} \cdot \left(i - \frac{1}{R_{\overline{A}}} \cdot u_{C_1}\right), \qquad u_{C_1}(0) = 0 \\ u'_{C_2} &= \frac{1}{C_2} \cdot \left(i - \frac{1}{R_{\overline{B}}} \cdot u_{C_2}\right), \qquad u_{C_2}(0) = 0 \\ u'_{C_{34}} &= \frac{1}{C_{34}} \cdot \left(i - \frac{R_A + R_B}{R_A \cdot R_B} \cdot u_{C_{34}}\right), \quad u_{C_{34}}(0) = 3.3 \end{aligned} \tag{5.11}$$

The solution for the values from Table 5.2 is shown in Figure 5.17. The transient responses begin at 50 ns and 150 ns. The circuit truly behaves like the NOR gate.

[5]Parameters are the same as for (5.10), capacity $C_{34} = C_3 + C_4$.



(a) CSM [V] (b) SPICE [V]

Figure 5.17: NOR – solution

The error of the approximation is shown in Figure 5.18. The approximation error during the transient response now exceeds even 2 V; but as already stated, it is only a minor problem.

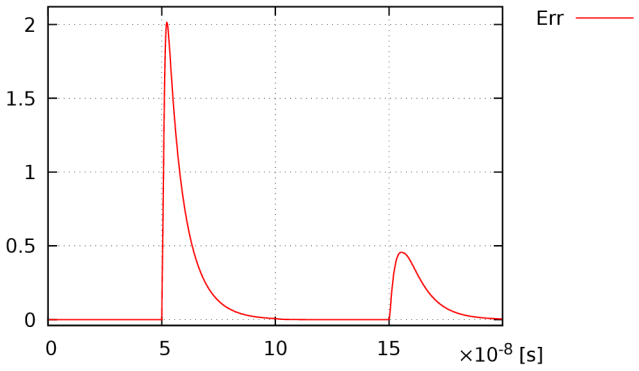


Figure 5.18: NOR – approximation error [V]

5.4.4 XOR

XOR can be constructed using the basic CMOS logic gates described above (inverters, NANDs and NORs). The XOR is used in adders. Equation (5.12) is in the Disjunctive Normal Form (DNF).

$$x \oplus y \;=\; (\overline{x} \wedge y) \vee (x \wedge \overline{y}) \tag{5.12}$$

Logic gates AND and OR are compound gates (additional inverters are required), but if De Morgan's laws are used, the expression in (5.13) is acquired that uses the simplest logic gates which can be constructed in electronics. The Sheffer stroke (denoted as $\uparrow$) [21] is a logical operation equivalent to the negated conjunction operation, NAND.

$$x \oplus y \;=\; (\overline{x} \uparrow y) \uparrow (x \uparrow \overline{y}) \tag{5.13}$$

The abbreviated SPICE netlist of XOR follows (the full version can be found in Appendix D.6).

```
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* x = 0, 0, 1, 1
V3 3 0 PWL(0 0 2e-07 0 2e-07 3.3 4e-07 3.3)
* y = 0, 1, 0, 1
V4 4 0 PWL(0 0 1e-07 0 1e-07 3.3 2e-07 3.3 2e-07 0 3e-07 0 3e-07 3.3 4e-07
+ 3.3)
* not(x)
M5a 6 3 2 2 P3306M
M5b 6 3 0 0 N3306M
R5c 5 6 0.1
* nand(not(x), y)
M7a 8 5 2 2 P3306M
M7b 8 4 2 2 P3306M
M7c 8 5 9 9 N3306M
M7d 9 4 0 0 N3306M
R7e 7 8 0.1
* not(y)
M10a 11 4 2 2 P3306M
M10b 11 4 0 0 N3306M
R10c 10 11 0.1
* nand(x, not(y))
M12a 13 3 2 2 P3306M
M12b 13 10 2 2 P3306M
M12c 13 3 14 14 N3306M
M12d 14 10 0 0 N3306M
R12e 12 13 0.1
* nand(nand(not(x), y), nand(x, not(y)))
M15a 15 7 2 2 P3306M
M15b 15 12 2 2 P3306M
M15c 15 7 17 17 N3306M
M15d 17 12 0 0 N3306M
C15e 15 0 1e-12
```

Table 5.3 summarizes the results near the end of each time segment. The last two columns contain the output voltages of the circuit representing XOR solved by CSM and SPICE, respectively.

t [10^{-7} s]	x	y	Res	CSM [V]	SPICE [V]
0.99	0	0	0	0.000000	0.000000
1.99	0	1	1	3.299943	3.299999
2.99	1	0	1	3.299959	3.299999
3.99	1	1	0	0.001769	0.000000

Table 5.3: XOR

XOR with three inputs

To construct a full adder, three-input XORs are required. Although two XORs could be used, it is rather useful to have XOR with three inputs[6]. Equation (5.14) describes the three-input XOR – derived from the Conjunctive Normal Form (CNF).

$$x \oplus y \oplus z \;=\; \overline{((x \wedge y) \wedge \overline{z})} \wedge \overline{((y \wedge z) \wedge \overline{x})} \wedge \overline{((x \wedge z) \wedge \overline{y})} \wedge \overline{(\overline{x} \wedge \overline{y} \wedge \overline{z})} \tag{5.14}$$

Equation (5.15) is again obtained from (5.14) using De Morgan's laws. The NOR operator is known as Peirce's arrow (denoted as $\downarrow$) [21]. Note that both $\uparrow$ and $\downarrow$ are

[6]The three-input XOR delay is shorter than the delay of two XORs and fewer basic logic gates are used.

assumed to be non-associative in our formal system – the parentheses delimit separate logic gates; that is, a 3-input NOR is used to evaluate the last parenthesis and a 4-input NOR is used to summarize the partial results.

$$x \oplus y \oplus z \;=\; ((x \uparrow y) \downarrow z) \downarrow ((y \uparrow z) \downarrow x) \downarrow ((x \uparrow z) \downarrow y) \downarrow (x \downarrow y \downarrow z) \tag{5.15}$$

Table 5.4 summarizes the results near the end of each time segment (eight segments in total).

t [10^{-7} s]	**x**	**y**	**z**	**Res**	**CSM [V]**	**SPICE [V]**
1.49	0	0	0	0	0.000030	0.000001
2.99	0	0	1	1	3.199193	3.198240
4.49	0	1	0	1	3.297346	3.299999
5.99	0	1	1	0	0.000276	0.000001
7.49	1	0	0	1	3.203977	3.203666
8.99	1	0	1	0	0.000276	0.000002
10.49	1	1	0	0	0.000133	0.000019
11.99	1	1	1	1	3.201614	3.203512

Table 5.4: XOR with three inputs

The output values of CSM and SPICE correspond quite well. The behavior of a three-input XOR is correctly analyzed.

Chapter 6

VLSI

Very Large-Scale Integration (VLSI) circuits typically comprise hundreds of thousands of transistors on a chip. They can be assembled only from CMOS NANDs or NORs[1] as both gates can form any logical operation. Therefore it is possible to simulate VLSI circuits using the CSM described above. More about VLSI design can be found in [22].

6.1 CMOS latches

CMOS latches are electronic circuits constructed using the basic CMOS logic gates (typically inverted, which are constructed from fewer transistors than non-inverted). Latches can store one bit of information (0 or 1) as long as they are power-supplied. Their values can be changed by inputs (specific to every latch).

They are also called asynchronous flip-flops, since their function is not dependent on the clock signal. Thanks to the asynchronous behavior, changes to the value of the latches are immediate; however, it is not always an advantage – the problem occurs in the simulation of the JK latch. For more, see [37].

6.1.1 RS latch

The RS latch is a fundamental latch, since all other latches and flip-flops are built on it. It consists of two NORs, two inputs R and S and two outputs Q and $\overline{Q}$. Input S sets the value to logical one and input R resets the value to zero. Output Q holds the value and output $\overline{Q}$ is its complement. Figure 6.1 shows the scheme.

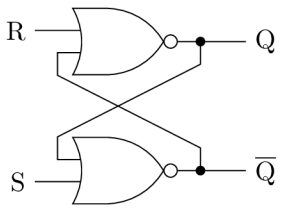


Figure 6.1: RS latch

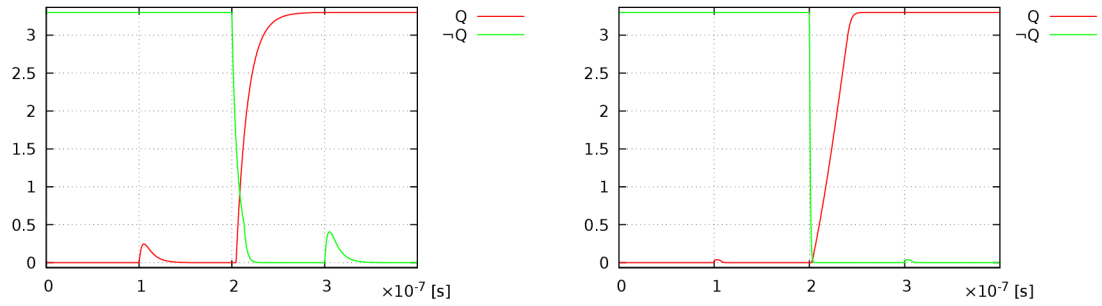
[1] Although it is better to use both types of gates and an inverter.

Table 6.1 contains input values. First, the value is reset to zero and then the value is set to logical one. All inputs acquire logical zero or one in the form of the corresponding voltage values.

R	1	0	0	0
S	0	0	1	0

Table 6.1: RS latch – inputs

Figure 6.2 illustrates the behavior of the circuit. The length of the basic time segment is determined by two logic-gate delays, i. e. 100 ns, since the latch requires changes to the value of two gates. The value of the latch is preset to zero to avoid the initial transient response; therefore, the initial reset only confirms the value. Transient responses can be observed after each change of input values – there are two minor transient responses beginning at 100 ns and 300 ns, but they have no impact on the resulting behavior, since they are in tolerance. The main transient response can be observed at 200 ns where switching of the values occurs.



(a) CSM [V] (b) SPICE [V]

Figure 6.2: RS latch – solution

The approximation errors for both Q and $\overline{Q}$ are shown in Figure 6.3.

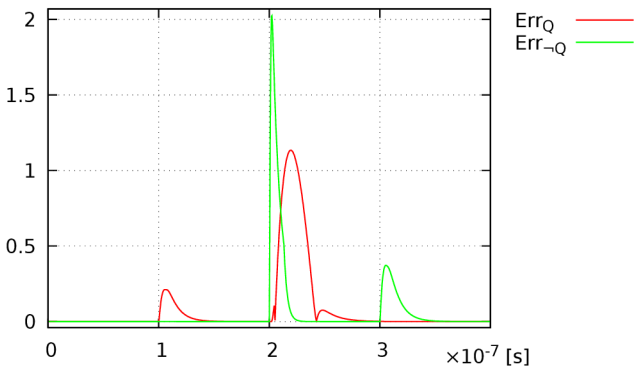


Figure 6.3: RS latch – approximation error [V]

The transient response is responsible for the problem of inconsistent values: there is a small time segment when values Q and $\overline{Q}$ are the same. This state is invalid and can

cause some side effects. The problem does not appear in flip-flops, since the value is considered valid after the change (it is ensured by the clock).

The disadvantage of the RS latch is that both inputs cannot be logical one at the same time; this is considered an invalid input, which would lead to an invalid state. Therefore, it is solved in other latches and flip-flops using various approaches.

6.1.2 D latch

D latches are the cornerstones of D registers which are used in Booth's algorithm analyzed in Section 6.4. The problem of invalid inputs is solved by setting input R to the complement of input S. However, another input, Wr, is necessary[2]; otherwise, the latch would not hold the value (the input would propagate). The scheme is illustrated in Figure 6.4. CMOS NORs were substituted by CMOS NANDs since the input value is inverted by the leftmost NAND (the toggling value is now logical zero).

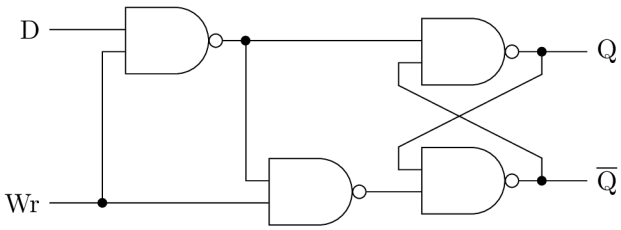


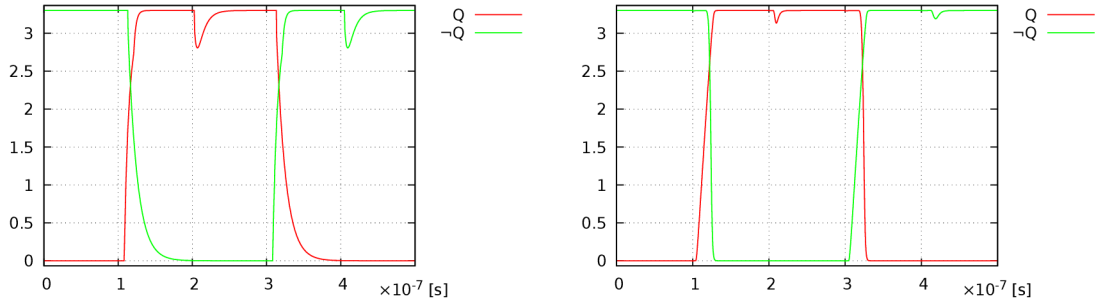
Figure 6.4: D latch

Input values are given in Table 6.2. Value D changes in time, but the value propagates only when the latch is enabled by input Wr (even time segments).

D	0	1	0	0	1
Wr	0	1	0	1	0

Table 6.2: D latch – inputs

Figure 6.5 shows the solution for the initial value equal to zero. Again, the transient responses occur after input changes (note the transient responses after Wr changes to logical zero).



(a) CSM [V]

(b) SPICE [V]

Figure 6.5: D latch – solution

[2]This version is referred to in the literature as gated.

The approximation error is shown in Figure 6.6.

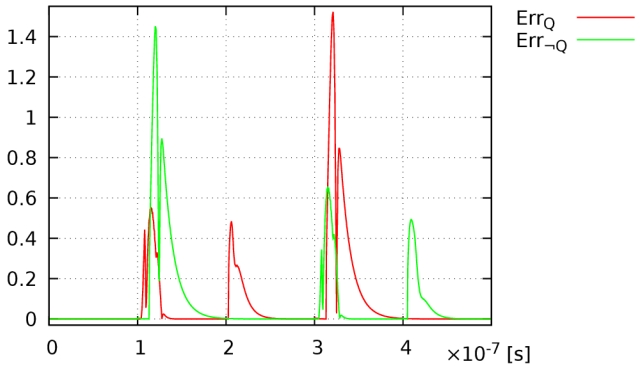


Figure 6.6: D latch – approximation error [V]

6.1.3 JK latch

The JK latch is the basis of the master–slave JK flip-flop (as explained in Section 6.2.2) that is used for the T flip-flop required by the implementation of Booth's algorithm. The JK latch eliminates the problem of invalid input combination by adjusting the function to toggle the values when both inputs (of the RS latch) are logical ones. Otherwise, input J corresponds to S and input K corresponds to R. The scheme of the latch is illustrated in Figure 6.7.

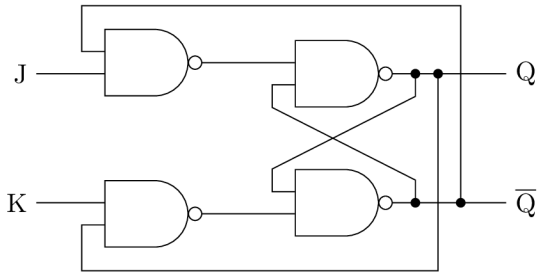


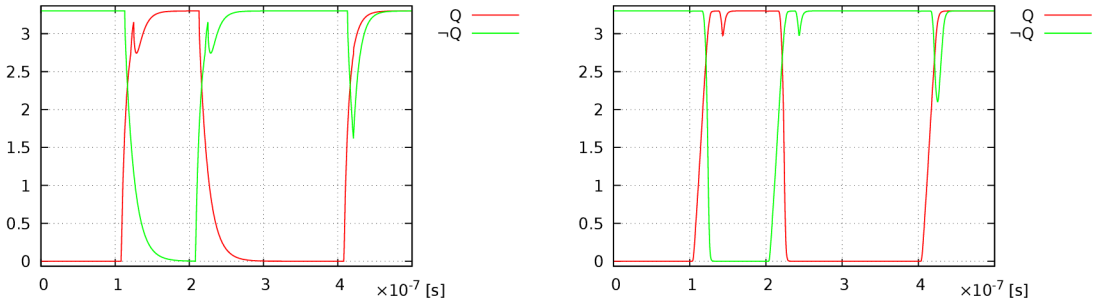
Figure 6.7: JK latch

Table 6.3 contains the inputs. The first time segment is left empty. The latch is set in the second time segment and reset in the third. The fourth time segment preserves the value and toggling appears in the last segment.

J	0	1	0	0	1
K	0	0	1	0	1

Table 6.3: JK latch – inputs

The behavior of the latch is illustrated in Figure 6.8. All time segments except for the last one show correct behavior. In the last time segment, a problem appears: when value Q toggles to logical one, the second input NAND acquires zero value after a short transient response while the value of the first input NAND remains zero; therefore, both values Q and $\overline{Q}$ converge to logical one.



(a) CSM [V] (b) SPICE [V]

Figure 6.8: JK latch – solution

The approximation error is shown in Figure 6.9.

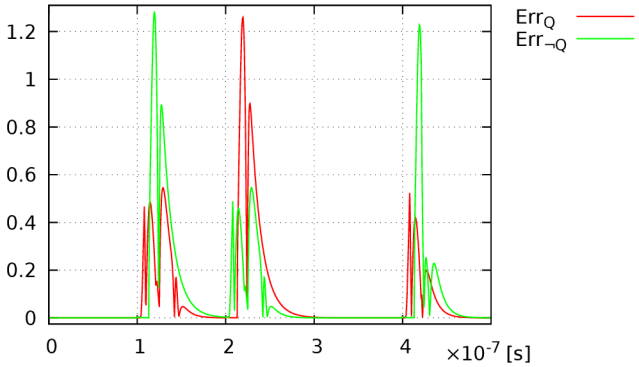


Figure 6.9: JK latch – approximation error [V]

To avoid the problem, precise timing is necessary; however, a much better solution, as demonstrated below, is a synchronous master–slave JK flip-flop.

6.2 CMOS flip-flops

CMOS flip-flops are synchronized by the clock. The time period of the clock is four logic-gate delays, $T = 200$ ns, since changing the values of two logic gates is typically required in the active time segment. For more, see [37].

6.2.1 D flip-flop

The D flip-flop is very similar to the D latch; the only difference is the clock input. The circuit can change its state only if the clock is logical one. It is employed in the D shift registers used by Booth's algorithm for the multiplicand and the result. The scheme is illustrated in Figure 6.10.

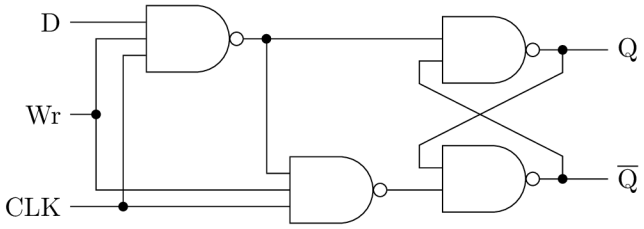


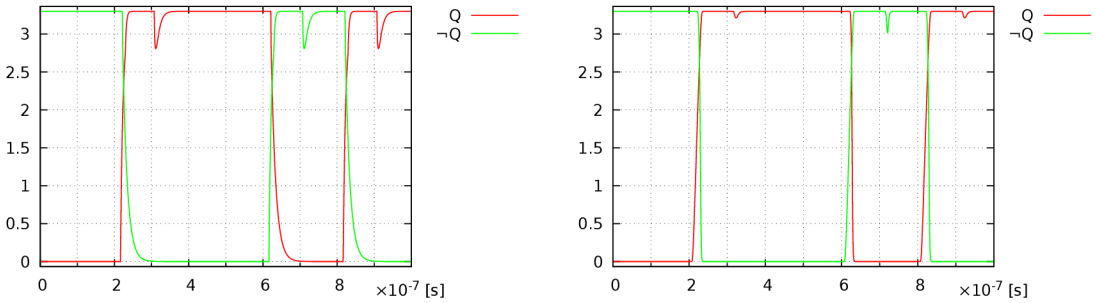
Figure 6.10: D flip-flop

Input values are given in Table 6.4. Input D is logical one in two time segments and the circuit is enabled by input Wr in three time segments.

D	0	1	0	0	1
Wr	0	1	0	1	1

Table 6.4: D flip-flop – inputs

The output changes synchronously; the behavior, which is illustrated in Figure 6.11, is correct.



(a) CSM [V] (b) SPICE [V]

Figure 6.11: D flip-flop – solution

The approximation errors of both Q and $\overline{Q}$ are shown in Figure 6.12.

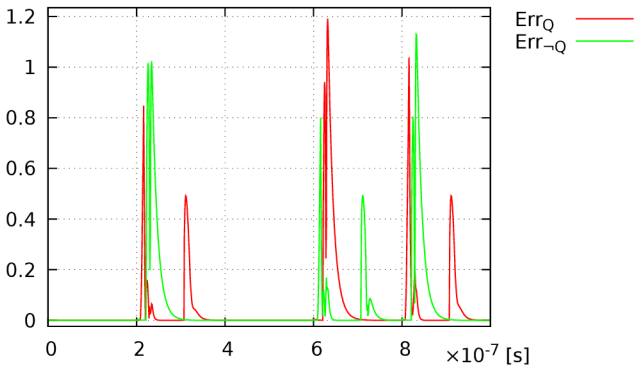


Figure 6.12: D flip-flop – approximation error [V]

6.2.2 JK flip-flop

Figure 6.13 shows a master–slave JK flip-flop, which eliminates the problem of the JK latch. This approach ensures that the output of the master flip-flop changes first (on the positive edge of the clock) while the slave flip-flop changes its state afterwards (on the negative edge). The JK flip-flop is used in the form of the T flip-flop in the implementation of Booth's algorithm (analyzed in Section 6.4).

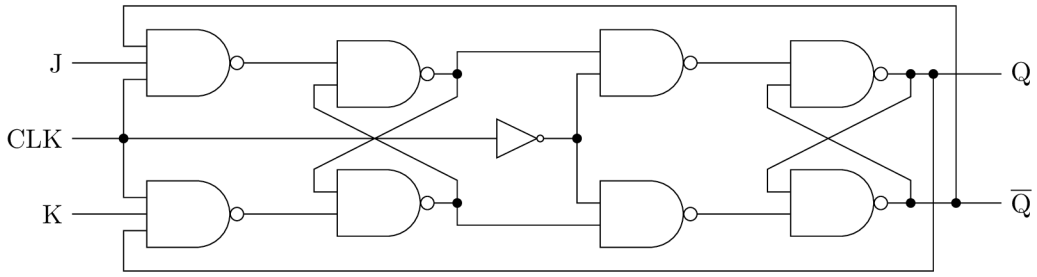


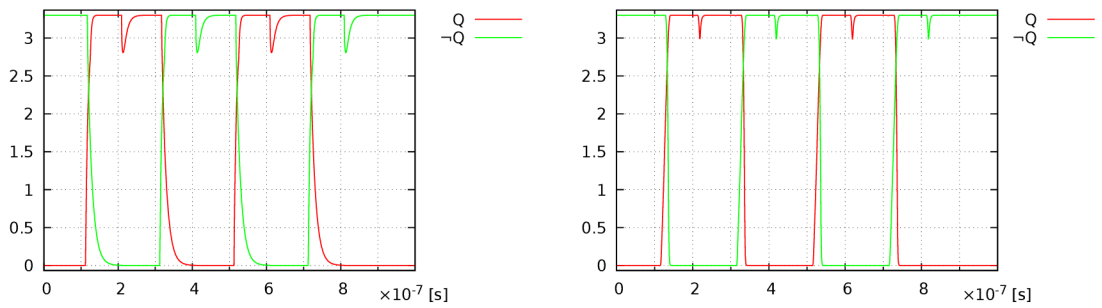
Figure 6.13: JK flip-flop

The input values are given in Table 6.5. The flip-flop is set in the first time segment, reset in the second segment and toggled in the third and the fourth time segment. It holds the value in the last segment.

J	1	0	1	1	0
K	0	1	1	1	0

Table 6.5: JK flip-flop – inputs

The behavior of the master–slave JK flip-flop is illustrated in Figure 6.14. It is obvious from the graph that the timing problem has now been solved.



(a) CSM [V]

(b) SPICE [V]

Figure 6.14: JK flip-flop – solution

The approximation error is shown in Figure 6.15.

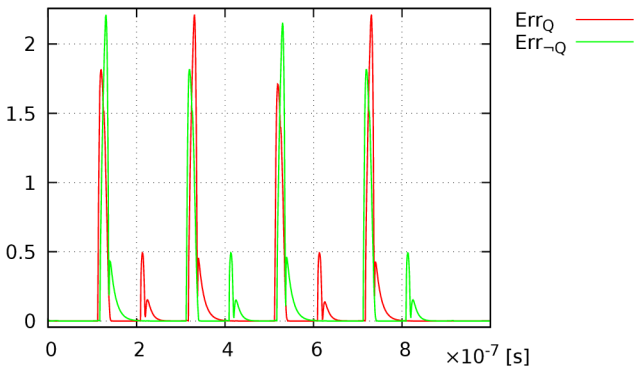


Figure 6.15: JK flip-flop – approximation error [V]

6.3 Adder

Addition is a basic arithmetic operation. It is a very suitable operation for the simulation of VLSI circuits, as it is easily scalable.

6.3.1 Half adder

The half adder has only two inputs (summands). It can be used for the calculation of the least significant bit (LSB) of multiple-bit adders without an input carry. The output and the carry are calculated by (6.1).

$$\begin{aligned} output &= x \oplus y \\ carry &= x \wedge y = \overline{x \uparrow y} \end{aligned} \tag{6.1}$$

6.3.2 Full adder

The full adder has three inputs – two summands and an input carry. The expressions in (6.2) are used for calculating the output and the carry.

$$\begin{aligned} output &= x \oplus y \oplus c_0 \\ carry &= (x \wedge y) \vee (x \wedge c_0) \vee (y \wedge c_0) \\ &= (x \uparrow y) \uparrow (x \uparrow c_0) \uparrow (y \uparrow c_0) \end{aligned} \tag{6.2}$$

6.3.3 Transient response

The traditional ripple-carry adder has a disadvantage – it takes a long time to propagate the carry to 1-bit adders representing more significant bits when calculating the sum. Assuming a 16-bit adder and the inputs 1111111111111111_b and 1, the carry of the least significant bit propagates slowly through all 1-bit adders, resulting in the 16-bit adder carry. The result overflows – denoted by square brackets, see (6.3).

$$1111111111111111_b + 1 = [1]0000000000000000_b \tag{6.3}$$

The situation is shown in Figure 6.16. All bits of the 16-bit adder are set to zero after the corresponding transient response.

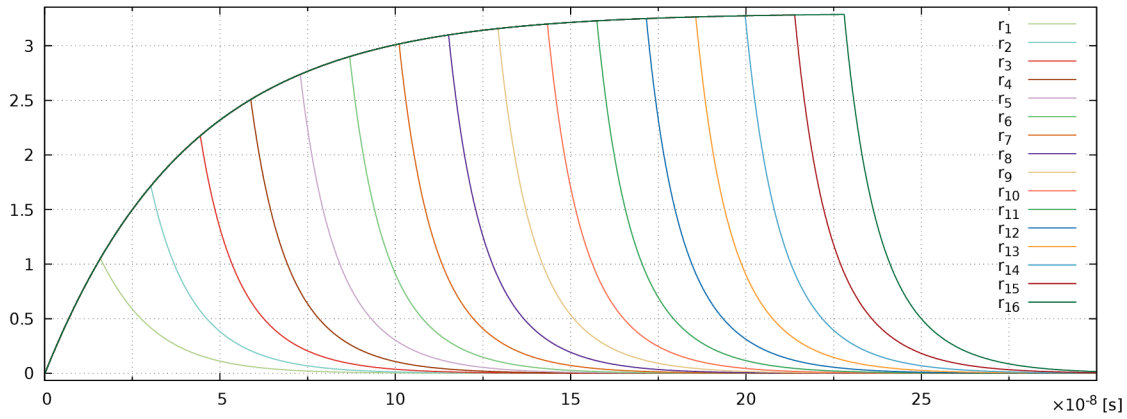


Figure 6.16: Carry propagation [V]

6.3.4 CLA adder

To avoid a significant delay of the adder (especially for multiple-bit numbers), a specific circuit – Carry Look-ahead (CLA) – can be constructed [37]. It uses values g_i (generate) and p_i (propagate), calculated by (6.4).

$$\begin{aligned} g_i &= a_i \wedge b_i \\ p_i &= a_i \vee b_i \end{aligned} \tag{6.4}$$

Particular carries can be calculated using (6.5).

$$\begin{aligned} c_1 &= g_0 \vee (p_0 \wedge c_0) \\ c_2 &= g_1 \vee (p_1 \wedge c_1) \\ c_3 &= g_2 \vee (p_2 \wedge c_2) \\ c_4 &= g_3 \vee (p_3 \wedge c_3) \end{aligned} \tag{6.5}$$

After substituting carries c_1, c_2 and c_3, (6.6) is obtained.

$$\begin{aligned} c_1 &= g_0 \vee (p_0 \wedge c_0) \\ c_2 &= g_1 \vee (p_1 \wedge g_0) \vee (p_1 \wedge p_0 \wedge c_0) \\ c_3 &= g_2 \vee (p_2 \wedge g_1) \vee (p_2 \wedge p_1 \wedge g_0) \vee (p_2 \wedge p_1 \wedge p_0 \wedge c_0) \\ c_4 &= g_3 \vee (p_3 \wedge g_2) \vee (p_3 \wedge p_2 \wedge g_1) \vee (p_3 \wedge p_2 \wedge p_1 \wedge g_0) \\ &\quad \vee (p_3 \wedge p_2 \wedge p_1 \wedge p_0 \wedge c_0) \end{aligned} \tag{6.6}$$

The propagation delay of the CLA is 3 logic gates if AND and OR are used. In case of NAND and NOR gates[3], the propagation delay is 4 logic gates – g_i and p_i are calculated using NAND and NOR gates and inverted; the propagation delay of c_i remains 2 logic gates as it is calculated by (6.7) – using De Morgan's laws.

[3]NANDs and NORs are commonly used in electronic circuits (each gate consists of four transistors, see Figure 5.10a and 5.15a).

$$
\begin{aligned}
c_1 &= \overline{g_0} \uparrow (p_0 \uparrow c_0) \\
c_2 &= \overline{g_1} \uparrow (p_1 \uparrow g_0) \uparrow (p_1 \uparrow p_0 \uparrow c_0) \\
c_3 &= \overline{g_2} \uparrow (p_2 \uparrow g_1) \uparrow (p_2 \uparrow p_1 \uparrow g_0) \uparrow (p_2 \uparrow p_1 \uparrow p_0 \uparrow c_0) \\
c_4 &= \overline{g_3} \uparrow (p_3 \uparrow g_2) \uparrow (p_3 \uparrow p_2 \uparrow g_1) \uparrow (p_3 \uparrow p_2 \uparrow p_1 \uparrow g_0) \\
&\quad \uparrow (p_3 \uparrow p_2 \uparrow p_1 \uparrow p_0 \uparrow c_0)
\end{aligned} \tag{6.7}
$$

To evaluate the carries more quickly, another type of electronic circuit is required – Carry Look-ahead Unit (CLU). It combines the values in the same manner as CLA, but its input values are calculated by (6.8).

$$
\begin{aligned}
G &= g_3 \vee (p_3 \wedge g_2) \vee (p_3 \wedge p_2 \wedge g_1) \vee (p_3 \wedge p_2 \wedge p_1 \wedge g_0) \\
P &= p_3 \wedge p_2 \wedge p_1 \wedge p_0
\end{aligned} \tag{6.8}
$$

Again, it can be transformed into the equivalent form with basic CMOS logic gates using De Morgan's laws.

$$
\begin{aligned}
G &= \overline{g_3} \uparrow (p_3 \uparrow g_2) \uparrow (p_3 \uparrow p_2 \uparrow g_1) \uparrow (p_3 \uparrow p_2 \uparrow p_1 \uparrow g_0) \\
P &= \overline{p_3 \uparrow p_2 \uparrow p_1 \uparrow p_0}
\end{aligned} \tag{6.9}
$$

The accelerating circuit takes the form of a tree with CLAs on the lowest level and CLUs on the other levels.

Table 6.6 shows that the length of the transient response of a 64-bit adder is considerably shorter. For more information, see [37].

t [10^{-7} s]	**Result**
0.0	1111111111111111111111111111111111111111111111111111111111111111
0.2	1111111111111111111111111111111111111111111111111111111111111111
0.4	1111111111111111111111111111111111111111111111111111111111111111
0.6	1111111111111111111111111111111111111111111111111111111111111110
0.8	1111111111111111111111111111111111111111111111111111111111111110
1.0	1111111111111111111111111111111111111111111111111111111111110000
1.2	1111111111111111111111111111111111111111111111111111111111110000
1.4	1111111111111111111111111111111111111111111111111111111011100000
1.6	1111111111111111111111111111111111111111111111111110110011000000
1.8	1111111111111111111111111111111111111111111111010001000000000000
2.0	1111111111111101111111111111101111111111101100000000000000000000
2.2	1111111111011001111111011101000111011101100000000000000000000000
2.4	1110111011000000111011001000000011101000100000000000000000000000
2.6	1100100000000000110000000000000010000000000000000000000000000000
2.8	1000000000000000000000000000000000000000000000000000000000000000
3.0	0000000000000000000000000000000000000000000000000000000000000000

Table 6.6: CLA adder – transient response

6.3.5 Scale of integration

The size of an electronic circuit is determined by the scale of integration. It is classified differently by various authors – according to [22], the main categories are:

- Small-Scale Integration (SSI) – less than 10 logic gates;
- Medium-Scale Integration (MSI) – 10 to 1 000 logic gates;
- Large-Scale Integration (LSI) – up to 10 000 logic gates;
- Very Large-Scale Integration (VLSI) – more than 10 000 logic gates.

Some authors [10] even define a fifth category: Ultra Large-Scale Integration (ULSI). However, ULSI is commonly included in VLSI by other authors.

6.3.6 Experiments

The experiments were performed on our research server[4]. All simulation times were chosen carefully to attain a final steady state. The experiments were performed using SPICE[5] and CSM; specialized software was developed for the VLSI simulation (see Section 5.3.2).

Table 6.7 summarizes the parameters for individual test cases. The multiple-bit adders with CLU+CLA trees were used for simulation. The first column (denoted as h_T) shows the tree heights, the second one the number of bits, the next columns contain the number of transistors, logic gates and ordinary differential equations respectively and the last columns contain the delays in multiples of basic logic-gate delays (these determine the simulation times) and the scale of the integration. The number of bits used is proportional to the tree height. The even rows contain the parameters of adders with half CLAs, as not all carries are required.

h_T	# bits	# transistors	# gates	# ODE	Delay	SI
2	16	1272	286	922	11	MSI
3	32	2568	581	1865	15	MSI
3	64	5176	1166	3754	15	LSI
4	128	10376	2341	7529	19	LSI
4	256	20792	4686	15082	19	LSI
5	512	41608	9381	30185	23	LSI
5	1024	83256	18766	60394	23	VLSI
6	2048	166536	37541	120809	27	VLSI
6	4096	333112	75086	241642	27	VLSI
7	8192	666248	150181	483305	31	VLSI
7	16384	1332536	300366	966634	31	VLSI

Table 6.7: CLA adder – parameters

The interesting thing is that the number of ordinary differential equations is smaller than the number of transistors. This is caused by merging parallel capacitors.

[4] 2× Intel Xeon E5-2630v2 (2.6 GHz, 6/12-core, 15 MB cache), 32 GB RAM
[5] NGSpice v26.1 with default settings

The results of the serial simulation of CLA adders are shown in Table 6.8 (the SPICE simulations running longer than a day[6] were terminated before the end – it is irrelevant how long they run). The results show the memory usage (denoted as MEM) and the time consumption (Time) depending on the number of bits (# bits).

	CSM		**SPICE**	
# bits	**MEM [MB]**	**Time [s]**	**MEM [MB]**	**Time [s]**
16	0.30	0.39	5.51	0.90
32	0.55	0.97	10.73	3.62
64	1.33	1.95	20.52	11.65
128	2.37	4.77	39.85	45.55
256	4.95	9.61	79.27	292.32
512	9.84	22.26	159.46	1427.53
1024	19.45	49.97	–	> 86400
2048	38.96	127.99	–	> 86400
4096	77.99	271.23	–	> 86400
8192	155.99	630.67	–	> 86400
16384	312.03	1316.48	–	> 86400

Table 6.8: Serial simulation

The results of the parallel simulation are shown in Table 6.9. SPICE is omitted as the chosen implementation does not support parallel computation. The results show that the memory consumption remains almost the same as in the serial simulation. The time consumption is considerably lower. The decrease in the acceleration ratio for 4 096 bits and more is probably caused by small caches.

# bits	**MEM [MB]**	**Time [s]**	**Acceleration**
16	0.30	0.19	2.0526
32	0.82	0.35	2.7714
64	1.34	0.49	3.9796
128	2.63	1.05	4.5429
256	5.20	1.58	6.0823
512	10.09	3.16	7.0443
1024	19.89	5.84	8.5565
2048	39.23	14.50	8.8269
4096	78.40	33.36	8.1304
8192	157.30	85.95	7.3376
16384	313.01	217.61	6.0497

Table 6.9: Parallel simulation

Table 6.10 shows the acceleration of the serial and parallel computations achieved by CSM compared to SPICE.

[6] 1 day = 86 400 s

# bits	Serial	Parallel
16	2.3077	4.7368
32	3.7320	10.3429
64	5.9744	23.7755
128	9.5493	43.3810
256	30.4183	185.0127
512	64.1298	451.7500
1024	> 1729.0374	> 14794.5205

Table 6.10: Acceleration of CSM compared to SPICE

The results achieved show that the simulation by CSM is much faster than SPICE for larger circuits. It takes more than a day to simulate them using SPICE.

6.4 Multiplier

Besides addition, multiplication is another important arithmetic operation. To calculate any MTSM term of any ODE in the autonomous form (transformed by the automatic transformation), only addition and multiplication are necessary.

6.4.1 Booth's algorithm

Booth's algorithm is fast and uses only the operations addition and bit shift. The principle is discussed in [15].

D shift register

The multiplier stores numbers in D shift registers [20]. These registers consist of D flip-flops; a master–slave D flip-flop is used for our purposes since it performs shifting in a safe manner (single D flip-flop requires very careful timing [35]). The function of an n-bit D shift register is described by (6.10).

$$\begin{aligned}
S_i &= x_i \uparrow En \uparrow Wr \\
R_i &= S_i \uparrow En \uparrow Wr \\
\\
D_i &= \overline{RES} \uparrow q_{i+1} \uparrow Sh \uparrow CLK \\
Q_i &= S_i \uparrow D_i \uparrow \overline{Q}_i \\
\overline{Q}_i &= \overline{RES} \uparrow R_i \uparrow Q_i \uparrow (D_i \uparrow Sh \uparrow CLK) \\
\\
d_i &= \overline{RES} \uparrow Q_i \uparrow Sh \uparrow \overline{CLK} \\
q_i &= S_i \uparrow d_i \uparrow \overline{q}_i \\
\overline{q}_i &= \overline{RES} \uparrow R_i \uparrow q_i \uparrow \left(d_i \uparrow Sh \uparrow \overline{CLK}\right)
\end{aligned} \tag{6.10}$$

The register can be asynchronously filled with number x if enabled by signals En and Wr (parallel input). The register can be asynchronously reset by signal RES. D_i is a negated

input of the master RS latch and Q_i and $\overline{Q}_i$ are the master state values. The variables of the slave RS latch are denoted by the lower case.

The register can be shifted by signal Sh, with the shift being performed in two steps: first, the master RS latch loads the value from a higher bit[7] and then the slave RS latch loads the value of the master RS latch in the negative half of the clock – this avoids double shift.

Two's complement

Two's complement is used for the subtraction of the multiplier. Commonly, it is performed by an inversion and an addition of one [37]. A better approach is to start from the least significant bit (LSB), leaving all zeros intact up to the first one – other bits are inverted.[8]

The latter algorithm suffers from the same problem as the adder – slow carry propagation. As in the case of the adder, a propagation circuit can be constructed – the Invert Look-ahead (ILA). In this case, the carry remains one from the first non-zero carry – the carries are calculated by (6.11).

$$\begin{aligned} c_1 &= c_0 \vee x_0 \\ c_2 &= c_1 \vee x_1 \\ c_3 &= c_2 \vee x_2 \\ c_4 &= c_3 \vee x_3 \end{aligned} \tag{6.11}$$

After substituting c_i, (6.12) is derived.

$$\begin{aligned} c_1 &= c_0 \vee x_0 \\ c_2 &= c_0 \vee x_0 \vee x_1 \\ c_3 &= c_0 \vee x_0 \vee x_1 \vee x_2 \\ c_4 &= c_0 \vee x_0 \vee x_1 \vee x_2 \vee x_3 \end{aligned} \tag{6.12}$$

By transforming (6.12) to use only basic CMOS gates, (6.13) is obtained.

$$\begin{aligned} c_1 &= \overline{c_0 \downarrow x_0} \\ c_2 &= \overline{c_0 \downarrow x_0 \downarrow x_1} \\ c_3 &= \overline{c_0 \downarrow x_0 \downarrow x_1 \downarrow x_2} \\ c_4 &= \overline{c_0 \downarrow x_0 \downarrow x_1 \downarrow x_2 \downarrow x_3} \end{aligned} \tag{6.13}$$

As in the case of the adder, another type of accelerating circuit is required – the Invert Look-ahead Unit (ILU). It combines the input values in the same way as ILA, but the input values of this circuit are calculated by

$$G = x_1 \vee x_2 \vee x_3 \vee x_4 \tag{6.14}$$

or by (6.15) using the basic logic gates.

$$G = \overline{x_1 \downarrow x_2 \downarrow x_3 \downarrow x_4} \tag{6.15}$$

Then an accelerating tree is established: the lowest level consists of ILAs and the other levels of ILUs (grouping four units from the nearest lower level).

[7] That is q_{i+1} for all bits save the most significant bit (MSB). MSB uses its own value q_i.

[8] The idea uses the fact that all zeros are inverted into ones and when one is added they become zeros again and the zero which arose from the first one is changed into the overflowed one.

T flip-flop

The result and an operand of the adder used to add or subtract the multiplier are kept in a pair of n-bit registers and after every required addition (some segments do not require an addition) the functions of the registers are toggled to avoid copying the result. The master–slave T flip-flop serves this purpose. It is derived from the master–slave JK flip-flop where both inputs are either zero or one. The function is described by (6.16).

$$\begin{aligned} Q &= \overline{Q} \uparrow \left(\overline{RES} \uparrow T \uparrow \overline{q} \uparrow CLK \uparrow En\right) \\ \overline{Q} &= \overline{RES} \uparrow Q \uparrow (T \uparrow q \uparrow CLK \uparrow En) \\ q &= \overline{q} \uparrow \left(\overline{RES} \uparrow Q \uparrow \overline{CLK}\right) \\ \overline{q} &= \overline{RES} \uparrow q \uparrow \left(\overline{Q} \uparrow \overline{CLK}\right) \end{aligned} \tag{6.16}$$

Multiplexer

A two-input n-bit multiplexer (i. e. the multiplexer with control bit c selecting from two n-bit inputs) is used for selecting the appropriate register containing the result of the multiplication; another multiplexer is used for the selection of the second adder input. The function of the multiplexer is described by (6.17).

$$r_i = (\overline{c} \uparrow x_i) \uparrow (c \uparrow y_i) \tag{6.17}$$

6.4.2 Multiplier components

The components used for the multiplier construction are:

- an $(n+1)$-bit D shift register for the multiplicand;
- an n-bit D register containing the multiplier;
- an n-bit accelerated complementing (two's complement) circuit;
- an n-bit D register containing the complemented multiplier;
- two n-bit D shift registers for the result (toggled to avoid copying);
- a one-bit T flip-flop for the decision which result register is used;
- a one-bit delaying D register to store the decision which result register is used;
- a two-input n-bit multiplexer selecting the right result register (also the first adder input);
- an XOR deciding whether to switch between the result registers (based on two LSBs of the multiplicand);
- a two-input n-bit multiplexer for the selection of the second adder input (the multiplier or the complemented multiplier);
- an n-bit CLA adder.

6.4.3 Verification

First, the correct operation of the multiplier was verified. The multiplication (6.18) was performed (the numbers were generated randomly).

$$-0.10011010000101111011101011001000_b \cdot 0.11011111110000010000001000010111_b \quad (6.18)$$

$-0.10011010000101111011101011001000_b$ is $10110010111101000010001010011100_b$ in two's complement. The partial results of the algorithm are shown in Table 6.11.

#	Extended multiplicand	Result
1	INIT	00000000000000000000000000000000
2	10110010111101000010001010011100**00**	00000000000000000000000000000000
3	11011001011110100001000101001110**00**	00000000000000000000000000000000
4	11101100101111010000100010100111**10**	11001000000011111011111101111010
5	11110110010111101000010001010011**11**	11100100000001111101111110111101
6	11111011001011110100001000101001**11**	11110010000000111110111111011110
7	11111101100101111010000100010100**01**	00110000111100100011100001110100
8	11111110110010111101000010001010**00**	00011000011110010001110000111010
9	11111111011001011110100001000101**10**	11010100010011000100110110010111
10	11111111101100101111010000100010**01**	00100010000101100110011101010001
11	11111111110110010111101000010001**10**	11011001000110101111001100100011
12	11111111111011001011110100001000**01**	00100100011111011011101000010111
13	11111111111101100101111010000100**00**	00010010001111101101110100001011
14	11111111111110110010111101000010**00**	00001001000111110110111010000101
15	11111111111111011001011110100001**10**	11001100100111110111011010111101
16	11111111111111101100101111010000**01**	00011110001111111111101111100100
17	11111111111111110110010111101000**00**	00001111000111111111110111110010
18	11111111111111111011001011110100**00**	00000111100011111111111011111001
19	11111111111111111101100101111010**00**	00000011110001111111111101111100
20	11111111111111111110110010111101**10**	11001001111100111011111100111000
21	11111111111111111111011001011110**01**	00011100111010100010000000100001
22	11111111111111111111011001011110**10**	11010110100001001100111110001011
23	11111111111111111111101100101011**11**	11101011010000100110011111000101
24	11111111111111111111111011001011**11**	11110101101000010011001111100010
25	11111111111111111111111011001011**11**	11111010110100001001100111110001
26	11111111111111111111111101100101**01**	00110101010110001000110101111110
27	11111111111111111111111111011001**10**	11100010101111000000011000111001
28	11111111111111111111111111011001**01**	00101001010011100100001110100010
29	11111111111111111111111111111011**00**	00010100101001110010000111010001
30	11111111111111111111111111111101**10**	11010010011000110101000001100011
31	11111111111111111111111111111110**11**	11101001001100011010100000110001
32	11111111111111111111111111111111**01**	00101100100010010001010010011110
33	11111111111111111111111111111111**10**	10111100101010001001001110010011

Table 6.11: Booth's multiplier – partial results

The initialization is the first phase of the algorithm – the result registers, the T flip-flop and the delaying one-bit register are zeroed; then, the input values are stored in appropriate

registers and the two's complement of the multiplier is calculated and stored in the register. The other phases (which are performed n-times) always consist of three subphases:

1. From two LSBs of the multiplicand, the type of operation is determined (00 and 11 mean no operation, 01 addition and 10 subtraction).
2. The multiplier (or its complement in the case of subtraction) is added to the result and stored in the alternate result register. This phase takes longer than the others, depending on the adder delay.
3. If the operation was to add or subtract, the alternate result register is chosen as the multiplier result (and the adder input) by storing the value of the T flip-flop into the delaying register. If this is not the last phase, the multiplicand and the result are shifted one bit right.

The result of the calculation performed, which is given in the last line of Table 6.11, is the two's complement of

$$-0.10000110101011101101100011011 01_b$$

that corresponds with the correct result of (6.18).

6.4.4 Experiments

Table 6.12 summarizes the parameters for individual test cases. Fast adder and complement circuits were used and the simulation time was chosen appropriately to solve the whole multiplication process. The penultimate column (containing the multiplier delays in the number of time segments) determines the simulation times. The last column contains the scale of the integration [22].

# bits	# transistors	# gates	# ODE	Delay	SI
16	6190	1169	4264	51	LSI
32	12274	2324	8461	132	LSI
64	24456	4625	16853	260	LSI
128	48812	9236	33642	645	LSI
256	97538	18449	67218	1285	VLSI

Table 6.12: Booth's multiplier – parameters

The results of serial simulation are given in Table 6.13. The last three lines of the SPICE results are incomplete since SPICE runs longer than a day[9].

	CSM		SPICE	
# bits	MEM [MB]	Time [s]	MEM [MB]	Time [s]
16	1.34	59.02	24.54	179.74
32	2.88	284.97	55.27	1310.88
64	5.46	1141.98	–	> 86400
128	11.12	5424.41	–	> 86400
256	22.17	26051.81	–	> 86400

Table 6.13: Serial simulation

[9] 1 day = 86 400 s

Table 6.14 summarizes the results of parallel simulation. Parallel simulation by SPICE is not included since the chosen implementation does not support it. The last column of the table contains the acceleration of parallel to serial simulation.

# bits	MEM [MB]	Time [s]	Acceleration
16	1.59	15.48	3.8127
32	3.14	61.00	4.6716
64	5.71	178.37	6.4023
128	11.38	712.17	7.6167
256	22.47	2534.44	10.2791

Table 6.14: Parallel simulation

The comparison of CSM and SPICE is shown in Table 6.15. It is evident that CSM runs much faster than SPICE and is capable of solving larger circuits with low memory overhead.

# bits	Serial	Parallel
16	3.0454	11.6111
32	4.6001	21.4898
64	> 75.6581	> 484.3864

Table 6.15: Acceleration of CSM compared to SPICE

6.5 Generic CMOS circuits

Generic CMOS circuits can be described using an adjacency matrix. The matrix is sparse and can be automatically transformed into a system of ODEs describing the electronic circuit using CSM. For example, the adjacency matrix and the vector of operations for three-input XOR follow.

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \quad \mathbf{u} = \begin{pmatrix} x \\ y \\ z \\ \uparrow \\ \downarrow \\ \uparrow \\ \downarrow \\ \uparrow \\ \downarrow \\ \downarrow \\ \downarrow \end{pmatrix} \tag{6.19}$$

The lines of matrix **A** correspond with the lines of vector **u**. To obtain the inputs for an operation, matrix–vector multiplication is performed.

6.5.1 Generating ODEs

The algorithm for the construction of ODEs is straightforward: each non-zero line of the adjacency matrix defines the inputs for the operation. For example, the penultimate line describes a three-input[10] NOR: x, y, z are the inputs; the equations are generated consecutively and the current for three inputs is:

$$i \;=\; \frac{1}{R_i} \cdot \Big(U - u_{C_1} - u_{C_2} - u_{C_3} - u_{C_{456}}\Big), \tag{6.20}$$

the three ODEs for the three inputs:

$$\begin{aligned} u'_{C_1} &= \frac{1}{C_1} \cdot \left(i - \frac{1}{R_{\overline{x}}} \cdot u_{C_1}\right), \quad u_{C_1}(0) = 0 \\ u'_{C_2} &= \frac{1}{C_2} \cdot \left(i - \frac{1}{R_{\overline{y}}} \cdot u_{C_2}\right), \quad u_{C_2}(0) = 0 \\ u'_{C_3} &= \frac{1}{C_3} \cdot \left(i - \frac{1}{R_{\overline{z}}} \cdot u_{C_3}\right), \quad u_{C_3}(0) = 0 \end{aligned} \tag{6.21}$$

and the ODE for the output:

$$u'_{C_{456}} \;=\; \frac{1}{C_{456}} \cdot \left(i - \frac{R_x \cdot R_y + R_x \cdot R_z + R_y \cdot R_z}{R_x \cdot R_y \cdot R_z} \cdot u_{C_{456}}\right), \quad u_{C_{456}}(0) \;=\; 3.3. \tag{6.22}$$

The construction of the equations corresponding to the other lines is analogical.

[10] The number of non-zero elements on the line defines the operation arity.

Chapter 7

Conclusion

In this thesis, I discussed the simulation of electric and electronic circuits. First, I described several numerical methods used for solving systems of ordinary differential equations (ODEs), but I mainly focused on the Modern Taylor Series Method (MTSM), which is very accurate, fast and flexible. The automatic transformation used by this method was explained thoroughly (demonstrated on elementary functions) and the simplification to the minimal form was introduced at the end of the second chapter.

Next, I showed the main characteristics of MTSM, pointing out the need for arbitrary precision arithmetic, automatic order selection, problems connected with the appropriate stopping rule choice and quicker computations than when using other methods. A comparison of calculation speed by MTSM and MATLAB [31] was undertaken in [42] using sets of problems proposed in [14].

Various methods of solving electric circuits were discussed, comparing the symbolic and numerical approach and showing possibilities for computation acceleration, including a few methods of parallelization. These methods were demonstrated on the telegraph-line problem as it is easily scalable.

The main part of the thesis dealt with solving electronic circuits. The Capacitor Substitution Method (CSM) was introduced and CMOS gates (inverter, NAND, NOR) were simulated. Then more complex circuits were modeled using the CMOS gates: XOR, latches (RS, D, JK), flip-flops (D, JK, T), an n-bit D shift register, a complementing circuit and a two-input n-bit multiplexer. From the circuits previously created, large circuits were constructed: a multiple-bit adder and a multiplier.

The proposed method can be easily parallelized since all logic gates are independent while no switching occurs; therefore, I separated the equations representing the electronic circuits, merging together approximately a thousand ODEs[1]. This approach appeared to be the most efficient one. Moreover, the simulation can be distributed among more computers on condition that the communication bus is reasonably fast. The acceleration of the parallel approach compared to the sequential approach is quite considerable.

CSM was successfully used for the simulation of VLSI circuits: multiple-bit adders (a combinational logic circuit) and multipliers (a sequential logic circuit) were simulated. The simulation using CSM was compared to the state-of-the-art system (SPICE) and the acceleration is quite impressive: CSM is capable of solving more than a million transistors in less than 4 minutes, while SPICE is unable to solve it within 24 hours; therefore, SPICE

[1] It would be inefficient to simulate each logic gate separately.

is unusable for this purpose. The results of the experiments are presented in Section 6.3.6 and Section 6.4.4.

7.1 Aims achieved

This thesis dealt with three research hypotheses:

- *The equations describing an electronic circuit can be systematically created.*
  The detailed assembling of ODEs is presented in Chapter 5. The basic CMOS logic gates are analyzed and modeled. The generic algorithm is introduced in Section 6.5.

- *The transistors could be replaced by RC circuits.*
  Yes, the operation of the transistors can be approximated by RC circuits. This approximation can model the lengths of the transient responses. The Capacitor Substitution Method (CSM) is described in Section 5.4.

- *The proposed method should be efficient.*
  This hypothesis is clearly answered in Chapter 6. It can be seen from the results, that the proposed CSM is much faster (much more than 1000×) and more memory-efficient than the state-of-the-art SPICE.

7.2 Research contribution

During my doctoral studies, I developed programming equipment for solving systems of ordinary differential equations [24] which is further being improved and also adapted for educational purposes (it is used in the course High Performance Computations[2] at the Faculty of Information Technology, Brno University of Technology).

I dealt with the simulation of electric and electronic circuits, which led to the proposal of the Capacitor Substitution Method (CSM). I simulated various electronic circuits using this method, starting with the basic CMOS gates (inverter, NAND, NOR, see Section 5.4) and XOR (Section 5.4.4), further latches (Section 6.1) and flip-flops (Section 6.2) and finally an adder (Section 6.3) and a multiplier (Section 6.4).

Further, I created specialized software for simulating VLSI, which was used for the simulation of relatively large VLSI circuits by CSM. The comparison of my approach to the state of the art is convincing: up to a 16kb adder with 1 332 536 transistors (300 366 logic gates) was successfully simulated in less than four minutes; SPICE was unable to solve even small VLSI circuits – a 1kb adder with 83 256 transistors (18 766 logic gates) – in a reasonable time. The experiments were performed in Section 6.3.6.

I also successfully used CSM for the simulation of sequential logic circuits represented by a multiplier. First, I constructed the basic elements required for the construction of the multiplier using Booth's algorithm with the CMOS logic gates proposed in Section 5.4. Then, I connected them together to form the whole multiplier. The created circuit is quite complex and the process of its modeling can serve as an example for modeling larger and more complex VLSI circuits such as microprocessors. The multiplier is thoroughly analyzed in Section 6.4 and the results of the experiments are shown in Section 6.4.4.

[2] `http://www.fit.vutbr.cz/study/courses/VNV/index.php.en`

The main ideas of the thesis were published at the prestigious[3] IEEE International Conference on High Performance Computing & Simulation (HPCS 2017)[4] and accepted by the scientific community.

7.3 Future research

The thesis outlines the possibilities of VLSI circuits simulation. Relatively large circuits can be simulated and future research can focus on modeling a basic Central Processing Unit (CPU) with basic operations simulated. After modeling a basic CPU, a more complex CPU model can be proposed. Such an extensive model consists of billions of transistors; therefore, a supercomputer will have to be involved.

[3] `http://portal.core.edu.au/conf-ranks/?search=hpcs&by=all&source=CORE2017`

[4] Kocina, F.; Kunovský, J.: Advanced VLSI Circuits Simulation. In *Proceedings of the 15th International Conference on High Performance Computing & Simulation*. Institute of Electrical and Electronics Engineers. 2017.

List of Publications

Prestigious conferences[1]

2017 Kocina, F.; Kunovský, J.: Advanced VLSI Circuits Simulation. In *Proceedings of the 15th International Conference on High Performance Computing & Simulation*. Institute of Electrical and Electronics Engineers. 2017.

2016 Kocina, F.; Nečasová, G.; Veigend, P.; et al.: Parallel Solution of Higher Order Differential Equations. In *Proceedings of the 14th International Conference on High Performance Computing & Simulation*. Institute of Electrical and Electronics Engineers. 2016. ISBN 978-1-5090-2088-1.

2015 Valenta, V.; Nečasová, G.; Kocina, F.; et al.: Adaptive Solution of the Wave Equation. In *Proceedings of the 5th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. Science and Technology Press. 2015. ISBN 978-989-758-120-5.

Proceedings with relatively high ranking[2]

2016 Kocina, F.; Nečasová, G.; Veigend, P.; et al.: Modelling VLSI Circuits Using Taylor Series. In *Proceedings of the 14th International Conference of Numerical Analysis and Applied Mathematics*. American Institute of Physics. 2016. ISSN 0094-243X.

Nečasová, G.; Kocina, F.; Veigend, P.; et al.: Solving Wave Equation Using Finite Differences and Taylor Series. In *Proceedings of the 14th International Conference of Numerical Analysis and Applied Mathematics*. American Institute of Physics. 2016. ISSN 0094-243X.

Veigend, P.; Nečasová, G.; Kocina, F.; et al.: Real Time Simulation of Transport Delay. In *Proceedings of the 14th International Conference of Numerical Analysis and Applied Mathematics*. American Institute of Physics. 2016. ISSN 0094-243X.

Chaloupka, J.; Kocina, F.; Veigend, P.; et al.: Multiple Integral Computations. In *Proceedings of the 14th International Conference of Numerical Analysis and Applied Mathematics*. American Institute of Physics. 2016. ISSN 0094-243X.

[1] `http://portal.core.edu.au/conf-ranks/`, `http://dblp.uni-trier.de/`
[2] `http://www.scimagojr.com/`

2015 Kocina, F.; Šátek, V.; Veigend, P.; et al.: New Trends in Taylor Series Based Applications. In *Proceedings of the 13th International Conference of Numerical Analysis and Applied Mathematics*. American Institute of Physics. 2015. ISBN 978-0-7354-1392-4.

Veigend, P.; Kunovský, J.; Kocina, F.; et al.: Electronic Representation of Wave Equation. In *Proceedings of the 13th International Conference of Numerical Analysis and Applied Mathematics*. American Institute of Physics. 2015. ISBN 978-0-7354-1392-4.

2014 Kocina, F.; Kunovský, J.; Marek, M.; et al.: New Trends in Taylor Series Based Computations. In *Proceedings of the 12th International Conference of Numerical Analysis and Applied Mathematics*. American Institute of Physics. 2014. ISBN 978-0-7354-1287-3.

2013 Šátek, V.; Kunovský, J.; Kocina, F.; et al.: Taylor Series Based Computations and MATLAB ODE Solvers Comparisons. In *Proceedings of the 11th International Conference of Numerical Analysis and Applied Mathematics*. American Institute of Physics. 2013. ISBN 978-0-7354-1184-5.

Publications in Web of Science[3] or Scopus[4]

2015 Šátek, V.; Kocina, F.; Kunovský, J.; et al.: Taylor Series Based Solution of Linear ODE Systems and MATLAB Solvers Comparison. In *Proceedings of the 8th Vienna International Conference on Mathematical Modelling*. Elsevier Science Direct. 2015. ISBN 978-3-901608-46-9.

Kunovský, J.; Šátek, V.; Kocina, F.; et al.: The Positive Properties of Modern Taylor Series Method. In *Proceedings of the 13th International Scientific Conference on Informatics*. Institute of Electrical and Electronics Engineers. 2015. ISBN 978-1-4673-9867-1.

2013 Kopřiva, J.; Kunovský, J.; Kocina, F.; et al.: Numerical integration in the RNS. In *Proceedings of the 12th International Scientific Conference on Informatics*. Faculty of Electrical Engineering and Informatics, Technical University of Košice. 2013. ISBN 978-80-8143-127-2.

Other publications

2015 Kocina, F.; Veigend, P.; Nečasová, G.; et al.: Parallel Computations of Differential Equations. In *Proceedings of the 10th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*. Litera. 2015. ISBN 978-80-214-5254-1.

[3] `http://apps.webofknowledge.com/`
[4] `http://www.scopus.com/`

Bibliography

[1] Baker, R. J.: *CMOS: Circuit Design, Layout, and Simulation*. John Wiley & Sons. 3rd edition. 2010. ISBN 978-0-470-88132-3.

[2] Barrio, R.; Blesa, F.; Lara, M.: VSVO Formulation of the Taylor Method for the Numerical Solution of ODEs. *Computers and Mathematics with Applications*. 2005. ISSN 0898-1221.

[3] Barton, D.: On Taylor Series and Stiff Equations. *ACM Transactions on Mathematical Software*. 1980. ISSN 0098-3500.

[4] Bartsch, H. J.: *Handbook of Mathematical Formulas*. Academic Press. 1974. ISBN 978-0-12-080050-6.

[5] Burden, R. L.; Faires, J. D.: *Numerical Analysis*. Cengage Learning. 9th edition. 2010. ISBN 978-0-538-73351-9.

[6] Butcher, J. C.: Coefficients for the Study of Runge–Kutta Integration Processes. *Journal of the Australian Mathematical Society*. 1963. ISSN 1446-7887.

[7] Butcher, J. C.: Implicit Runge–Kutta Processes. *Mathematics of Computation*. 1964. ISSN 0025-5718.

[8] Butcher, J. C.: *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons. 3rd edition. 2016. ISBN 978-1-119-12150-3.

[9] Calhoun, B. H.: *Design Principles for Digital CMOS Integrated Circuits*. National Technology & Science Press. 2012. ISBN 978-1-934891-14-8.

[10] Chang, C. Y.; Sze, S. M.: *ULSI Technology*. McGraw–Hill. 1996. ISBN 978-0-07-114105-5.

[11] Collins, P. J.: *Differential and Integral Equations*. Oxford University Press. 2006. ISBN 978-0-19-853382-5.

[12] Corliss, G.; Kirlinger, G.: On Implicit Taylor Series Methods for Stiff ODEs. In *Computer Arithmetic and Enclosure Methods*. 1991. ISBN 978-0-444-89834-0.

[13] Engelhardt, M.: SPICE Differentiation. *LT Journal of Analog Innovation*. 2015.

[14] Enright, W. H.; Pryce, J. D.: Two FORTRAN Packages For Assessing Initial Value Methods. *ACM Transactions on Mathematical Software*. 1987. ISSN 0098-3500.

[15] Flynn, M. J.; Oberman, S. F.: *Advanced Computer Arithmetic Design*. John Wiley & Sons. 2001. ISBN 978-0-471-41209-0.

[16] Griffiths, D. F.; Higham, D. J.: *Numerical Methods for Ordinary Differential Equations*. Springer. 2010. ISBN 978-0-85729-147-9.

[17] Hairer, E.; Nørsett, S. P.; Wanner, G.: *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer. 2nd edition. 1993. ISBN 978-3-540-56670-0.

[18] Hairer, E.; Wanner, G.: *Solving Ordinary Differential Equations II: Stiff and Differential–Algebraic Problems*. Springer. 2nd edition. 1996. ISBN 978-3-540-60452-5.

[19] Hanrot, G.; Lefèvre, V.; Pélissier, P.; et al.: The GNU MPFR Library. Retrieved from: `http://www.mpfr.org/`

[20] Hwang, E. O.: *Digital Logic and Microprocessor Design with VHDL*. Thomson. 2006. ISBN 978-0-534-46593-3.

[21] Kaeslin, H.: *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication*. Cambridge University Press. 2008. ISBN 978-0-521-88267-5.

[22] Kishore, K. L.; Prabhakar, V. S. V.: *VLSI Design*. I. K. International Publishing House. 2010. ISBN 978-93-80026-67-1.

[23] Koch, O.; Kofler, P.; Weinmüller, E. B.: The Implicit Euler Method for the Numerical Solution of Singular Initial Value Problems. *Applied Numerical Mathematics*. 2000. ISSN 0168-9274.

[24] Kocina, F.: FOS: Fast ODE Solver. Software. Retrieved from: `http://www.fit.vutbr.cz/~ikocina/prods.php`

[25] Korn, G. A.; Korn, T. M.: *Electronic Analog and Hybrid Computers*. McGraw–Hill. 2nd edition. 1972. ISBN 978-0-07-035363-3.

[26] Kunovský, J.: *Modern Taylor Series Method*. Habilitation work. Faculty of Electrical Engineering and Computer Science, Brno University of Technology. 1994.

[27] Kunovský, J.; Pindryč, M.; Šátek, V.; et al.: Stiff Systems in Theory and Practice. In *Proceedings of the 6th EUROSIM Congress on Modelling and Simulation*. 2007. ISBN 978-3-901608-32-2.

[28] Kunovský, J.; Šátek, V.; Vopěnka, V.; et al.: Stiffness in Technical Initial Problems. In *Proceedings of the 10th International Conference of Numerical Analysis and Applied Mathematics*. 2012. ISBN 978-0-7354-1091-6.

[29] Larsson, E.: *Introduction to Advanced System-on-Chip Test Design and Optimization*. Springer. 2005. ISBN 978-1-4020-3207-3.

[30] Lazar, J.: Electronics in Physical Experiment. February 2008. Retrieved from: `http://www.isibrno.cz/~joe/elektronika/elektronika_3.pdf`

[31] MathWorks: The Language of Technical Computing. Retrieved from: `http://www.mathworks.com/products/matlab`

[32] Moore, G. E.: Cramming More Components onto Integrated Circuits. In *Proceedings of the IEEE*. 1998. ISSN 0018-9219.

[33] Nedialkov, S. N.; Pryce, J. D.: Solving Differential Algebraic Equations by Taylor Series III. *Journal of Numerical Analysis, Industrial and Applied Mathematics*. 2008. ISSN 1790-8140.

[34] Paul, C. R.: *Introduction to Electromagnetic Compatibility*. John Wiley & Sons. 2nd edition. 2006. ISBN 978-0-471-75500-5.

[35] Pedroni, V. A.: *Digital Electronics and Design with VHDL*. Elsevier. 2008. ISBN 978-0-12-374270-4.

[36] Rábová, Z.; Zendulka, J.; Češka, M.; et al.: *Modeling and Simulation*. Brno University of Technology. 1992. ISBN 80-214-0480-9.

[37] Rafiquzzaman, M.: *Fundamentals of Digital Logic and Microcomputer Design*. John Wiley & Sons. 5th edition. 2005. ISBN 978-0-471-73349-2.

[38] Ralston, A.; Rabinowitz, P.: *A First Course in Numerical Analysis*. McGraw–Hill. 1978. ISBN 978-0-07-051158-3.

[39] Sabah, N. H.: *Electric Circuits and Signals*. CRC Press. 2007. ISBN 978-1-4200-4592-5.

[40] Šátek, V.: *Stiff Systems Analysis*. PhD thesis. Faculty of Information Technology, Brno University of Technology. 2012.

[41] Šátek, V.; Kocina, F.; Kunovský, J.; et al.: Taylor Series Based Solution of Linear ODE Systems and MATLAB Solvers Comparison. In *Proceedings of the 8th Vienna International Conference on Mathematical Modelling*. 2015. ISBN 978-3-901608-46-9.

[42] Šátek, V.; Kunovský, J.; Kocina, F.; et al.: Taylor Series Based Computations and MATLAB ODE Solvers Comparisons. In *Proceedings of the 11th International Conference of Numerical Analysis and Applied Mathematics*. 2013. ISBN 978-0-7354-1184-5.

[43] Veigend, P.; Kunovský, J.; Kocina, F.; et al.: Electronic Representation of Wave Equation. In *Proceedings of the 13th International Conference of Numerical Analysis and Applied Mathematics*. 2015. ISBN 978-0-7354-1392-4.

[44] Wakerly, J. F.: *Digital Design: Principles and Practices*. Pearson Education. 4th edition. 2006. ISBN 978-0-13-186389-7.

[45] Xiu, L.: *VLSI Circuit Design Methodology Demystified: A Conceptual Taxonomy*. John Wiley & Sons. 2007. ISBN 978-0-470-19910-7.

Appendices

List of Appendices

Appendix A

Practical usage

A.1 Circle test

```
setup {
  dt = 0.05;
  method = Euler;
  tmax = 8*pi+dt/3;
  lim = 2;
}

graph {
  colors = blue;
  domain = y;
  format = pdf;
  height = 480;
  legend = off;
  square = on;
  width = 480;
  xmax = lim;
  xmin = -lim;
  ymax = lim;
  ymin = -lim;
}

y' = z &0;
z' = -y &1;
```

A.2 Stiff system

```
setup {
  digits = 30;
  dt = tmax;
  eps = 1e-32;
  graphs = off;
  method = MTSM;
  prec = 32768;
  tmax = 5;
  a = 1e1234;
}

y' = z &1;
z' = -a*y-(a+1)*z &-1;
```

A.3 Mechanical oscillator

```
setup {
  dt = tmax/1000;
  tmax = 5;
  a = 31.4;
  k = 1;
  pos = 0;
  v0 = 15;
}

graph {
  format = pdf;
  lpos = inside;
  name = yz;
  show = y, z;
}

graph {
  colors = blue;
  domain = y;
  format = pdf;
  lpos = inside;
  name = pol;
  show = z;
  xmax = 8.2;
  xmin = 0;
}

y' = z &pos;
z' = -k*z-a*u &v0;
u' = v*z &sin(pos);
v' = -u*z &cos(pos);
```

A.4 Definite integral

```
setup {
  dt = tmax/250;
}

graph {
  colors = blue;
  format = pdf;
  legend = off;
  show = y;
}

y' = sin(ln(t+1))*ln(t+1)*(t-1)*(t-1)/(t+1) &0;
```

A.5 Fourier coefficients

```
setup {
  dt = tmax/1000;
  tmax = 2*pi;
}

graph {
  colors = #F00, #0F0, #00F, #F0F, #0FF, #CC0, #000, #F80, #888, #088,
           #808;
  format = pdf;
  labels = "%c_%d";
```

```
  show = a?, b?;
}

f = 16*sin(t)^5+4*cos(t)^3-sin(t)^2+cos(t)^2+2*sin(t)*cos(t)-sin(5*t)
  + 5*sin(3*t)-cos(3*t)-cos(2*t)-sin(2*t)-10*sin(t);

a0' = 2/tmax*f &0;
a1' = 2/tmax*f*cos(t) &0;
a2' = 2/tmax*f*cos(2*t) &0;
a3' = 2/tmax*f*cos(3*t) &0;
a4' = 2/tmax*f*cos(4*t) &0;
a5' = 2/tmax*f*cos(5*t) &0;

b1' = 2/tmax*f*sin(t) &0;
b2' = 2/tmax*f*sin(2*t) &0;
b3' = 2/tmax*f*sin(3*t) &0;
b4' = 2/tmax*f*sin(4*t) &0;
b5' = 2/tmax*f*sin(5*t) &0;
```

Appendix B

Electric circuits

B.1 Algebraic operations

```
setup {
  dt = tmax/250;
  tmax = 1000;
  U = 10;
  R = 80;
  R1 = 10;
  R2 = 40;
  C1 = 1;
  C2 = 2;
}

graph {
  format = pdf;
  labels = "u_{C_%d}";
  show = uC1, uC2;
  xfmt = "%2.0t";
  xmult = "10^2";
  xmultx = 1.035;
  xmulty = -0.065;
  xtics = 0, tmax/5, 4*tmax/5;
}

uC1' = 1/C1*i1 &0;
uC2' = 1/C2*i2 &0;
i1 = 1/R1*(uA-uC1);
i2 = 1/R2*(uA-uC2);
uA = (U*R1*R2+uC1*R*R2+uC2*R*R1)/(R*R1+R*R2+R1*R2);
```

B.2 Parasitic capacity

```
setup {
  dt = tmax/250;
  tmax = 100;
  U = 10;
//  om = 0.2;
  R = 80;
  R1 = 10;
  R2 = 40;
  C1 = 1;
  C2 = 2;
  Cp = 0.1;
//  Cp = 0.01;
}
```

```
graph {
  format = pdf;
  labels = "u_{C_p}", "u_A";
  show = uCp, uA;
}

//u = U*sin(om*t);
uC1' = 1/C1*i1 &0;
uC2' = 1/C2*i2 &0;
uCp' = 1/Cp*(i-i1-i2) &0;
i = 1/R*(/*u*/U-uCp);
i1 = 1/R1*(uCp-uC1);
i2 = 1/R2*(uCp-uC2);
uA = (U*R1*R2+uC1*R*R2+uC2*R*R1)/(R*R1+R*R2+R1*R2);
```

B.3 Compensating capacity

```
setup {
  dt = tmax/250;
  tmax = 1e-4;
  U = 10;
  R = 80;
  Ri = 1e-2;
  R1 = 10;
  R2 = 40;
  C1 = 1;
  C2 = 2;
  Cp = 0.01;
  Ck = 1e-3;
}

graph {
  format = pdf;
  labels = "u_{C_p}", "u_A";
  show = uCp, uA;
  xfmt = "%2.0t";
  xmult = "10^{-5}";
  xmulty = -0.065;
  xtics = 0, 2e-5, 9e-5;
}

uC1' = 1/C1*i1 &0;
uC2' = 1/C2*i2 &0;
uCp' = 1/Cp*(i-i1-i2) &0;
uCk' = 1/Ck*(i-1/R*uCk) &0;
i = 1/Ri*(U-uCp-uCk);
i1 = 1/R1*(uCp-uC1);
i2 = 1/R2*(uCp-uC2);
uA = (U*R1*R2+uC1*R*R2+uC2*R*R1)/(R*R1+R*R2+R1*R2);
```

B.4 Telegraph line

```
setup {
  dt = tmax/1000;
  tmax = 1.8e-8;
  om = 2e9;
  L = 1e-8;
  C = 1e-12;
//  R1 = 1;
  R1 = 100;
//  R1 = 300;
```

```
  R2 = 100;
//  R2 = 1e12;
}

graph {
  colors = blue;
  format = pdf;
  height = 640;
  labels = "u_1";
  show = uc1;
  square = true;
  width = 640;
  xbase = 1000;
  xfmt = "%2.0l";
  xmult = "10^{-9}";
  xtics = 0, tmax/6, 5*tmax/6;
}

graph {
  colors = blue;
  format = pdf;
  height = 640;
  labels = "u_2";
  show = u2;
  square = true;
  width = 640;
  xbase = 1000;
  xfmt = "%2.0l";
  xmult = "10^{-9}";
  xtics = 0, tmax/6, 5*tmax/6;
}

u = sin(om*t) /*if t<pi/om*/;
i = 1/R1*(u-uc1);
uc1' = 1/C*(i-i1) &0;
i1' = 1/L*(uc1-uc2) &0;
uc2' = 1/C*(i1-i2) &0;
i2' = 1/L*(uc2-uc3) &0;
uc3' = 1/C*(i2-i3) &0;
i3' = 1/L*(uc3-uc4) &0;
uc4' = 1/C*(i3-i4) &0;
i4' = 1/L*(uc4-uc5) &0;
uc5' = 1/C*(i4-i5) &0;
i5' = 1/L*(uc5-uc6) &0;
uc6' = 1/C*(i5-i6) &0;
i6' = 1/L*(uc6-uc7) &0;
uc7' = 1/C*(i6-i7) &0;
i7' = 1/L*(uc7-uc8) &0;
uc8' = 1/C*(i7-i8) &0;
i8' = 1/L*(uc8-uc9) &0;
uc9' = 1/C*(i8-i9) &0;
i9' = 1/L*(uc9-uc10) &0;
uc10' = 1/C*(i9-i10) &0;
i10' = 1/L*(uc10-uc11) &0;
uc11' = 1/C*(i10-i11) &0;
i11' = 1/L*(uc11-uc12) &0;
uc12' = 1/C*(i11-i12) &0;
i12' = 1/L*(uc12-uc13) &0;
uc13' = 1/C*(i12-i13) &0;
i13' = 1/L*(uc13-uc14) &0;
uc14' = 1/C*(i13-i14) &0;
i14' = 1/L*(uc14-uc15) &0;
uc15' = 1/C*(i14-i15) &0;
i15' = 1/L*(uc15-uc16) &0;
uc16' = 1/C*(i15-i16) &0;
i16' = 1/L*(uc16-uc17) &0;
uc17' = 1/C*(i16-i17) &0;
i17' = 1/L*(uc17-uc18) &0;
```

```
uc18' = 1/C*(i17-i18) &0;
i18' = 1/L*(uc18-uc19) &0;
uc19' = 1/C*(i18-i19) &0;
i19' = 1/L*(uc19-uc20) &0;
uc20' = 1/C*(i19-i20) &0;
i20' = 1/L*(uc20-uc21) &0;
uc21' = 1/C*(i20-i21) &0;
i21' = 1/L*(uc21-uc22) &0;
uc22' = 1/C*(i21-i22) &0;
i22' = 1/L*(uc22-uc23) &0;
uc23' = 1/C*(i22-i23) &0;
i23' = 1/L*(uc23-uc24) &0;
uc24' = 1/C*(i23-i24) &0;
i24' = 1/L*(uc24-uc25) &0;
uc25' = 1/C*(i24-i25) &0;
i25' = 1/L*(uc25-uc26) &0;
uc26' = 1/C*(i25-i26) &0;
i26' = 1/L*(uc26-uc27) &0;
uc27' = 1/C*(i26-i27) &0;
i27' = 1/L*(uc27-uc28) &0;
uc28' = 1/C*(i27-i28) &0;
i28' = 1/L*(uc28-uc29) &0;
uc29' = 1/C*(i28-i29) &0;
i29' = 1/L*(uc29-uc30) &0;
uc30' = 1/C*(i29-i30) &0;
i30' = 1/L*(uc30-uc31) &0;
uc31' = 1/C*(i30-i31) &0;
i31' = 1/L*(uc31-uc32) &0;
uc32' = 1/C*(i31-i32) &0;
i32' = 1/L*(uc32-uc33) &0;
uc33' = 1/C*(i32-i33) &0;
i33' = 1/L*(uc33-uc34) &0;
uc34' = 1/C*(i33-i34) &0;
i34' = 1/L*(uc34-uc35) &0;
uc35' = 1/C*(i34-i35) &0;
i35' = 1/L*(uc35-uc36) &0;
uc36' = 1/C*(i35-i36) &0;
i36' = 1/L*(uc36-uc37) &0;
uc37' = 1/C*(i36-i37) &0;
i37' = 1/L*(uc37-uc38) &0;
uc38' = 1/C*(i37-i38) &0;
i38' = 1/L*(uc38-uc39) &0;
uc39' = 1/C*(i38-i39) &0;
i39' = 1/L*(uc39-uc40) &0;
uc40' = 1/C*(i39-i40) &0;
i40' = 1/L*(uc40-uc41) &0;
uc41' = 1/C*(i40-i41) &0;
i41' = 1/L*(uc41-uc42) &0;
uc42' = 1/C*(i41-i42) &0;
i42' = 1/L*(uc42-uc43) &0;
uc43' = 1/C*(i42-i43) &0;
i43' = 1/L*(uc43-uc44) &0;
uc44' = 1/C*(i43-i44) &0;
i44' = 1/L*(uc44-uc45) &0;
uc45' = 1/C*(i44-i45) &0;
i45' = 1/L*(uc45-uc46) &0;
uc46' = 1/C*(i45-i46) &0;
i46' = 1/L*(uc46-uc47) &0;
uc47' = 1/C*(i46-i47) &0;
i47' = 1/L*(uc47-uc48) &0;
uc48' = 1/C*(i47-i48) &0;
i48' = 1/L*(uc48-uc49) &0;
uc49' = 1/C*(i48-i49) &0;
i49' = 1/L*(uc49-uc50) &0;
uc50' = 1/C*(i49-i50) &0;
i50' = 1/L*(uc50-u2) &0;
u2 = R2*i50;
```

Appendix C

Electronic circuits

C.1 Diode

```
setup {
  dt = tmax/5000;
  tmax = 4.5/f;
  a = 1e-18;
  b = 50;
  R = 1e9;
  om = 2*pi*f;
  f = 0.2;
}

graph {
  format = pdf;
  height = 420;
  labels = u, u_R;
  show = u, uR;
  width = 640;
}

u' = om*v &0;
v' = -om*u &1;
iD' = b*(iD+a)/(1+b*R*(iD+a))*v &0;
uR = iD*R;
```

C.2 Inverter

```
setup {
  dt = tmax/1000;
  tmax = 2e-7;
  U = 3.3;
  Ri = 0.120792;
  Ropen = 0.601435;
  Rclosed = 1e10;
  C = 3.851953e-9;
}

graph {
  colors = blue;
  format = pdf;
  height = 360;
  labels = Out;
  show = uc2;
  width = 640;
  xbase = 1e8;
```

```
  xfmt = "%2.0l";
  xmult = "10^{-8} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/4, 3*tmax/4;
  yspace = 0.02;
}

x = 1, 0;

//not(x)
Rp1 = x*Rclosed+(1-x)*Ropen;
Rn1 = x*Ropen+(1-x)*Rclosed;
i = (U-uc1-uc2)/Ri;
uc1' = (i-uc1/Rp1)/C &U;
uc2' = (i-uc2*(1/Rn1))/C &0;
```

C.3 NAND

```
setup {
  dt = tmax/1000;
  tmax = 2e-7;
  U = 3.3;
  Ri = 0.120792;
  Ropen = 0.601435;
  Rclosed = 1e10;
  C = 3.851953e-9;
}

graph {
  colors = blue;
  format = pdf;
  height = 360;
  labels = Out;
  show = out;
  width = 640;
  xbase = 1e8;
  xfmt = "%2.0l";
  xmult = "10^{-8} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/4, 3*tmax/4;
  yspace = 0.02;
}

x = 1, 0, 0, 1;
y = 1, 1, 0, 0;

//nand(x, y)
Rp1 = x*Rclosed+(1-x)*Ropen;
Rn1 = x*Ropen+(1-x)*Rclosed;
Rp2 = y*Rclosed+(1-y)*Ropen;
Rn2 = y*Ropen+(1-y)*Rclosed;
i = (U-uc12-uc3-uc4)/Ri;
uc12' = (i-uc12*(1/Rp1+1/Rp2))/(2*C) &U;
uc3' = (i-uc3/Rn1)/C &0;
uc4' = (i-uc4/Rn2)/C &0;
out = uc3+uc4;
```

C.4 NAND with three inputs

```
setup {
  dt = tmax/1000;
  tmax = 4e-7;
  U = 3.3;
  Ri = 0.120792;
  Ropen = 0.601435;
  Rclosed = 1e10;
  C = 3.851953e-9;
}

graph {
  colors = blue;
  format = pdf;
  height = 360;
  labels = Out;
  show = out;
  width = 640;
  xbase = 1e8;
  xfmt = "%2.0l";
  xmult = "10^{-8} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/8, 7*tmax/8;
  yspace = 0.02;
}

x = 0, 0, 0, 0, 1, 1, 1, 1;
y = 0, 0, 1, 1, 0, 0, 1, 1;
z = 0, 1, 0, 1, 0, 1, 0, 1;

//nand(x, y, z)
Rp1 = x*Rclosed+(1-x)*Ropen;
Rn1 = x*Ropen+(1-x)*Rclosed;
Rp2 = y*Rclosed+(1-y)*Ropen;
Rn2 = y*Ropen+(1-y)*Rclosed;
Rp3 = z*Rclosed+(1-z)*Ropen;
Rn3 = z*Ropen+(1-z)*Rclosed;
i = (U-uc123-uc4-uc5-uc6)/Ri;
uc123' = (i-uc123*(1/Rp1+1/Rp2+1/Rp3))/(3*C) &0;
uc4' = (i-uc4/Rn1)/C &U/3;
uc5' = (i-uc5/Rn2)/C &U/3;
uc6' = (i-uc6/Rn3)/C &U/3;
out = uc4+uc5+uc6;
```

C.5 NOR

```
setup {
  dt = tmax/1000;
  tmax = 2e-7;
  U = 3.3;
  Ri = 0.120792;
  Ropen = 0.601435;
  Rclosed = 1e10;
  C = 3.851953e-9;
}

graph {
  colors = blue;
  format = pdf;
  height = 360;
  labels = Out;
  show = uc34;
```

```
  width = 640;
  xbase = 1e8;
  xfmt = "%2.0l";
  xmult = "10^{-8} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/4, 3*tmax/4;
  yspace = 0.02;
}

x = 0, 1, 1, 0;
y = 0, 0, 1, 1;

//nor(x, y)
Rp1 = x*Rclosed+(1-x)*Ropen;
Rn1 = x*Ropen+(1-x)*Rclosed;
Rp2 = y*Rclosed+(1-y)*Ropen;
Rn2 = y*Ropen+(1-y)*Rclosed;
i = (U-uc1-uc2-uc34)/Ri;
uc1' = (i-uc1/Rp1)/C &0;
uc2' = (i-uc2/Rp2)/C &0;
uc34' = (i-uc34*(1/Rn1+1/Rn2))/(2*C) &U;
```

C.6 NOR with three inputs

```
setup {
  dt = tmax/1000;
  tmax = 4e-7;
  U = 3.3;
  Ri = 0.120792;
  Ropen = 0.601435;
  Rclosed = 1e10;
  C = 3.851953e-9;
}

graph {
  colors = blue;
  format = pdf;
  height = 360;
  labels = Out;
  show = uc456;
  width = 640;
  xbase = 1e8;
  xfmt = "%2.0l";
  xmult = "10^{-8} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/8, 7*tmax/8;
  yspace = 0.02;
}

x = 0, 0, 0, 0, 1, 1, 1, 1;
y = 0, 0, 1, 1, 0, 0, 1, 1;
z = 0, 1, 0, 1, 0, 1, 0, 1;

//nor(x, y, z)
Rp1 = x*Rclosed+(1-x)*Ropen;
Rn1 = x*Ropen+(1-x)*Rclosed;
Rp2 = y*Rclosed+(1-y)*Ropen;
Rn2 = y*Ropen+(1-y)*Rclosed;
Rp3 = z*Rclosed+(1-z)*Ropen;
Rn3 = z*Ropen+(1-z)*Rclosed;
i = (U-uc1-uc2-uc3-uc456)/Ri;
uc1' = (i-uc1/Rp1)/C &0;
uc2' = (i-uc2/Rp2)/C &0;
```

```
uc3' = (i-uc3/Rp3)/C &0;
uc456' = (i-uc456*(1/Rn1+1/Rn2+1/Rn3))/(3*C) &U;
```

C.7 XOR

```
setup {
  dt = tmax/1000;
  tmax = 4e-7;
  U = 3.3;
  Ri = 0.1;
  Ropen = 0.5;
  Rclosed = 1e10;
  C = 4e-9;
}

graph {
  format = pdf;
  height = 360;
  show = x, y, out, ucA;
  width = 640;
  xbase = 1e8;
  xfmt = "%2.0l";
  xmult = "10^{-8} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/4, 3*tmax/4;
  yspace = 0.02;
}

x = 0, 0, 1, 1;
y = 0, 1, 0, 1;

//not(x)
Rp1B = x*Rclosed+(1-x)*Ropen;
Rn1B = x*Ropen+(1-x)*Rclosed;
iB = (U-uc1B-ucB)/Ri;
uc1B' = (iB-uc1B/Rp1B)/C &0;
ucB' = (iB-ucB*(1/Rn1B))/C &U;

//nand(not(x), y)
in1D = 1 if ucB>=U/2;
Rp1D = in1D*Rclosed+(1-in1D)*Ropen;
Rn1D = in1D*Ropen+(1-in1D)*Rclosed;
Rp2D = y*Rclosed+(1-y)*Ropen;
Rn2D = y*Ropen+(1-y)*Rclosed;
iD = (U-ucD2-uc1D-uc2D)/Ri;
ucD2' = (iD-ucD2*(1/Rp1D+1/Rp2D))/(2*C) &0;
uc1D' = (iD-uc1D/Rn1D)/C &U/2;
uc2D' = (iD-uc2D/Rn2D)/C &U/2;
ucD = uc1D+uc2D;

//not(y)
Rp1C = y*Rclosed+(1-y)*Ropen;
Rn1C = y*Ropen+(1-y)*Rclosed;
iC = (U-uc1C-ucC)/Ri;
uc1C' = (iC-uc1C/Rp1C)/C &0;
ucC' = (iC-ucC*(1/Rn1C))/C &U;

//nand(x, not(y))
Rp1E = x*Rclosed+(1-x)*Ropen;
Rn1E = x*Ropen+(1-x)*Rclosed;
in2E = 1 if ucC>=U/2;
Rp2E = in2E*Rclosed+(1-in2E)*Ropen;
Rn2E = in2E*Ropen+(1-in2E)*Rclosed;
iE = (U-ucE2-uc1E-uc2E)/Ri;
```

```
ucE2' = (iE-ucE2*(1/Rp1E+1/Rp2E))/(2*C) &0;
uc1E' = (iE-uc1E/Rn1E)/C &U/2;
uc2E' = (iE-uc2E/Rn2E)/C &U/2;
ucE = uc1E+uc2E;

//nand(nand(not(x), y), nand(x, not(y)))
in1A = 1 if ucD>=U/2;
Rp1A = in1A*Rclosed+(1-in1A)*Ropen;
Rn1A = in1A*Ropen+(1-in1A)*Rclosed;
in2A = 1 if ucE>=U/2;
Rp2A = in2A*Rclosed+(1-in2A)*Ropen;
Rn2A = in2A*Ropen+(1-in2A)*Rclosed;
iA = (U-ucA2-uc1A-uc2A)/Ri;
ucA2' = (iA-ucA2*(1/Rp1A+1/Rp2A))/(2*C) &U;
uc1A' = (iA-uc1A/Rn1A)/C &0;
uc2A' = (iA-uc2A/Rn2A)/C &0;
ucA = uc1A+uc2A;
out = 1 if ucA>=U/2;
```

C.8 XOR with three inputs

```
setup {
  dt = tmax/1000;
  tmax = 1.2e-6;
  U = 3.3;
  Ri = 0.1;
  Ropen = 0.5;
  Rclosed = 1e10;
  C = 4e-9;
}

graph {
  format = pdf;
  height = 360;
  show = x, y, z, out, ucA;
  width = 640;
  xbase = 1e8;
  xfmt = "%2.0l";
  xmult = "10^{-8} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/8, 7*tmax/8;
  yspace = 0.02;
}

x = 0, 0, 0, 0, 1, 1, 1, 1;
y = 0, 0, 1, 1, 0, 0, 1, 1;
z = 0, 1, 0, 1, 0, 1, 0, 1;

//nand(x, y)
Rp1B = x*Rclosed+(1-x)*Ropen;
Rn1B = x*Ropen+(1-x)*Rclosed;
Rp2B = y*Rclosed+(1-y)*Ropen;
Rn2B = y*Ropen+(1-y)*Rclosed;
iB = (U-ucB2-uc1B-uc2B)/Ri;
ucB2' = (iB-ucB2*(1/Rp1B+1/Rp2B))/(2*C) &0;
uc1B' = (iB-uc1B/Rn1B)/C &U/2;
uc2B' = (iB-uc2B/Rn2B)/C &U/2;
ucB = uc1B+uc2B;

//nor(nand(x, y), z)
in1C = 1 if ucB>=U/2;
Rp1C = in1C*Rclosed+(1-in1C)*Ropen;
Rn1C = in1C*Ropen+(1-in1C)*Rclosed;
Rp2C = z*Rclosed+(1-z)*Ropen;
```

```
Rn2C = z*Ropen+(1-z)*Rclosed;
iC = (U-uc1C-uc2C-ucC)/Ri;
uc1C' = (iC-uc1C/Rp1C)/C &U/2;
uc2C' = (iC-uc2C/Rp2C)/C &U/2;
ucC' = (iC-ucC*(1/Rn1C+1/Rn2C))/(2*C) &0;

//nand(y, z)
Rp1D = y*Rclosed+(1-y)*Ropen;
Rn1D = y*Ropen+(1-y)*Rclosed;
Rp2D = z*Rclosed+(1-z)*Ropen;
Rn2D = z*Ropen+(1-z)*Rclosed;
iD = (U-ucD2-uc1D-uc2D)/Ri;
ucD2' = (iD-ucD2*(1/Rp1D+1/Rp2D))/(2*C) &0;
uc1D' = (iD-uc1D/Rn1D)/C &U/2;
uc2D' = (iD-uc2D/Rn2D)/C &U/2;
ucD = uc1D+uc2D;

//nor(nand(y, z), x)
in1E = 1 if ucD>=U/2;
Rp1E = in1E*Rclosed+(1-in1E)*Ropen;
Rn1E = in1E*Ropen+(1-in1E)*Rclosed;
Rp2E = x*Rclosed+(1-x)*Ropen;
Rn2E = x*Ropen+(1-x)*Rclosed;
iE = (U-uc1E-uc2E-ucE)/Ri;
uc1E' = (iE-uc1E/Rp1E)/C &U/2;
uc2E' = (iE-uc2E/Rp2E)/C &U/2;
ucE' = (iE-ucE*(1/Rn1E+1/Rn2E))/(2*C) &0;

//nand(x, z)
Rp1F = x*Rclosed+(1-x)*Ropen;
Rn1F = x*Ropen+(1-x)*Rclosed;
Rp2F = z*Rclosed+(1-z)*Ropen;
Rn2F = z*Ropen+(1-z)*Rclosed;
iF = (U-ucF2-uc1F-uc2F)/Ri;
ucF2' = (iF-ucF2*(1/Rp1F+1/Rp2F))/(2*C) &0;
uc1F' = (iF-uc1F/Rn1F)/C &U/2;
uc2F' = (iF-uc2F/Rn2F)/C &U/2;
ucF = uc1F+uc2F;

//nor(nand(x, z), y)
in1G = 1 if ucF>=U/2;
Rp1G = in1G*Rclosed+(1-in1G)*Ropen;
Rn1G = in1G*Ropen+(1-in1G)*Rclosed;
Rp2G = y*Rclosed+(1-y)*Ropen;
Rn2G = y*Ropen+(1-y)*Rclosed;
iG = (U-uc1G-uc2G-ucG)/Ri;
uc1G' = (iG-uc1G/Rp1G)/C &U/2;
uc2G' = (iG-uc2G/Rp2G)/C &U/2;
ucG' = (iG-ucG*(1/Rn1G+1/Rn2G))/(2*C) &0;

//nor(x, y, z)
Rp1H = x*Rclosed+(1-x)*Ropen;
Rn1H = x*Ropen+(1-x)*Rclosed;
Rp2H = y*Rclosed+(1-y)*Ropen;
Rn2H = y*Ropen+(1-y)*Rclosed;
Rp3H = z*Rclosed+(1-z)*Ropen;
Rn3H = z*Ropen+(1-z)*Rclosed;
iH = (U-uc1H-uc2H-uc3H-ucH)/Ri;
uc1H' = (iH-uc1H/Rp1H)/C &0;
uc2H' = (iH-uc2H/Rp2H)/C &0;
uc3H' = (iH-uc3H/Rp3H)/C &0;
ucH' = (iH-ucH*(1/Rn1H+1/Rn2H+1/Rn3H))/(3*C) &U;

//nor(nor(nand(x, y), z), nor(nand(y, z), x), nor(nand(x, z), y), nor(x,
//y, z))
in1A = 1 if ucC>=U/2;
Rp1A = in1A*Rclosed+(1-in1A)*Ropen;
Rn1A = in1A*Ropen+(1-in1A)*Rclosed;
```

```
in2A = 1 if ucE>=U/2;
Rp2A = in2A*Rclosed+(1-in2A)*Ropen;
Rn2A = in2A*Ropen+(1-in2A)*Rclosed;
in3A = 1 if ucG>=U/2;
Rp3A = in3A*Rclosed+(1-in3A)*Ropen;
Rn3A = in3A*Ropen+(1-in3A)*Rclosed;
in4A = 1 if ucH>=U/2;
Rp4A = in4A*Rclosed+(1-in4A)*Ropen;
Rn4A = in4A*Ropen+(1-in4A)*Rclosed;
iA = (U-uc1A-uc2A-uc3A-uc4A-ucA)/Ri;
uc1A' = (iA-uc1A/Rp1A)/C &U/4;
uc2A' = (iA-uc2A/Rp2A)/C &U/4;
uc3A' = (iA-uc3A/Rp3A)/C &U/4;
uc4A' = (iA-uc4A/Rp4A)/C &U/4;
ucA' = (iA-ucA*(1/Rn1A+1/Rn2A+1/Rn3A+1/Rn4A))/(4*C) &0;
out = 1 if ucA>=U/2;
```

Appendix D

Electronic circuits (SPICE)

D.1 Inverter

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* x = 1, 0
V3 3 0 PWL(0 3.3 1e-07 3.3 1e-07 0 2e-07 0)
* not(x)
M4a 4 3 2 2 P3306M
M4b 4 3 0 0 N3306M
C4c 4 0 1e-12
.TRAN 2e-10 2e-07 0 2e-10
.PRINT TRAN V(4)
.PROBE
.END
```

D.2 NAND

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* x = 1, 0
V3 3 0 PWL(0 3.3 1e-07 3.3 1e-07 0 2e-07 0)
* y = 1, 0, 0, 1
V4 4 0 PWL(0 3.3 5e-08 3.3 5e-08 0 1.5e-07 0 1.5e-07 3.3 2e-07 3.3)
* nand(x, y)
M5a 5 3 2 2 P3306M
M5b 5 4 2 2 P3306M
M5c 5 3 6 6 N3306M
M5d 6 4 0 0 N3306M
C5e 5 0 1e-12
.TRAN 2e-10 2e-07 0 2e-10
.PRINT TRAN V(5)
.PROBE
.END
```

D.3 NAND with three inputs

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* x = 0, 1
V3 3 0 PWL(0 0 2e-07 0 2e-07 3.3 4e-07 3.3)
* y = 0, 1, 0, 1
V4 4 0 PWL(0 0 1e-07 0 1e-07 3.3 2e-07 3.3 2e-07 0 3e-07 0 3e-07 3.3 4e-07
+ 3.3)
* z = 0, 1, 0, 1, 0, 1, 0, 1
V5 5 0 PWL(0 0 5e-08 0 5e-08 3.3 1e-07 3.3 1e-07 0 1.5e-07 0 1.5e-07 3.3
+ 2e-07 3.3 2e-07 0 2.5e-07 0 2.5e-07 3.3 3e-07 3.3 3e-07 0 3.5e-07 0
+ 3.5e-07 3.3 4e-07 3.3)
* nand(x, y, z)
M6a 6 3 2 2 P3306M
M6b 6 4 2 2 P3306M
M6c 6 5 2 2 P3306M
M6d 6 3 7 7 N3306M
M6e 7 4 8 8 N3306M
M6f 8 5 0 0 N3306M
C6g 6 0 1e-12
.TRAN 4e-10 4e-07 0 4e-10
.PRINT TRAN V(6)
.PROBE
.END
```

D.4 NOR

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* x = 1, 0, 0, 1
V3 3 0 PWL(0 3.3 5e-08 3.3 5e-08 0 1.5e-07 0 1.5e-07 3.3 2e-07 3.3)
* y = 1, 0
V4 4 0 PWL(0 3.3 1e-07 3.3 1e-07 0 2e-07 0)
* nor(x, y)
M5a 6 3 2 2 P3306M
M5b 5 4 6 6 P3306M
M5c 5 3 0 0 N3306M
M5d 5 4 0 0 N3306M
C5e 5 0 1e-12
.TRAN 2e-10 2e-07 0 2e-10
.PRINT TRAN V(5)
.PROBE
.END
```

D.5 NOR with three inputs

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
```

```
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* x = 0, 1
V3 3 0 PWL(0 0 2e-07 0 2e-07 3.3 4e-07 3.3)
* y = 0, 1, 0, 1
V4 4 0 PWL(0 0 1e-07 0 1e-07 3.3 2e-07 3.3 2e-07 0 3e-07 0 3e-07 3.3 4e-07
+ 3.3)
* z = 0, 1, 0, 1, 0, 1, 0, 1
V5 5 0 PWL(0 0 5e-08 0 5e-08 3.3 1e-07 3.3 1e-07 0 1.5e-07 0 1.5e-07 3.3
+ 2e-07 3.3 2e-07 0 2.5e-07 0 2.5e-07 3.3 3e-07 3.3 3e-07 0 3.5e-07 0
+ 3.5e-07 3.3 4e-07 3.3)
* nor(x, y, z)
M6a 7 3 2 2 P3306M
M6b 8 4 7 7 P3306M
M6c 6 5 8 8 P3306M
M6d 6 3 0 0 N3306M
M6e 6 4 0 0 N3306M
M6f 6 5 0 0 N3306M
C6g 6 0 1e-12
.TRAN 4e-10 4e-07 0 4e-10
.PRINT TRAN V(6)
.PROBE
.END
```

D.6 XOR

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* x = 0, 0, 1, 1
V3 3 0 PWL(0 0 2e-07 0 2e-07 3.3 4e-07 3.3)
* y = 0, 1, 0, 1
V4 4 0 PWL(0 0 1e-07 0 1e-07 3.3 2e-07 3.3 2e-07 0 3e-07 0 3e-07 3.3 4e-07
+ 3.3)
* not(x)
M5a 6 3 2 2 P3306M
M5b 6 3 0 0 N3306M
R5c 5 6 0.1
* nand(not(x), y)
M7a 8 5 2 2 P3306M
M7b 8 4 2 2 P3306M
M7c 8 5 9 9 N3306M
M7d 9 4 0 0 N3306M
R7e 7 8 0.1
* not(y)
M10a 11 4 2 2 P3306M
M10b 11 4 0 0 N3306M
R10c 10 11 0.1
* nand(x, not(y))
M12a 13 3 2 2 P3306M
M12b 13 10 2 2 P3306M
M12c 13 3 14 14 N3306M
M12d 14 10 0 0 N3306M
R12e 12 13 0.1
* nand(nand(not(x), y), nand(x, not(y)))
M15a 15 7 2 2 P3306M
M15b 15 12 2 2 P3306M
M15c 15 7 17 17 N3306M
M15d 17 12 0 0 N3306M
C15e 15 0 1e-12
```

```
.TRAN 4e-10 4e-07 0 4e-10
.PRINT TRAN V(15)
.PROBE
.END
```

D.7 XOR with three inputs

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* x = 0, 0, 0, 0, 1, 1, 1, 1
V3 3 0 PWL(0 0 6e-07 0 6e-07 3.3 1.2e-06 3.3)
* y = 0, 0, 1, 1, 0, 0, 1, 1
V4 4 0 PWL(0 0 3e-07 0 3e-07 3.3 6e-07 3.3 6e-07 0 9e-07 0 9e-07 3.3
+ 1.2e-06 3.3)
* z = 0, 1, 0, 1, 0, 1, 0, 1
V5 5 0 PWL(0 0 1.5e-07 0 1.5e-07 3.3 3e-07 3.3 3e-07 0 4.5e-07 0 4.5e-07
+ 3.3 6e-07 3.3 6e-07 0 7.5e-07 0 7.5e-07 3.3 9e-07 3.3 9e-07 0 1.05e-06 0
+ 1.05e-06 3.3 1.2e-06 3.3)
* nand(x, y)
M6a 7 3 2 2 P3306M
M6b 7 4 2 2 P3306M
M6c 7 3 8 8 N3306M
M6d 8 4 0 0 N3306M
R6e 6 7 0.1
* nor(nand(x, y), z)
M9a 10 6 2 2 P3306M
M9b 11 5 10 10 P3306M
M9c 11 6 0 0 N3306M
M9d 11 5 0 0 N3306M
R9e 9 11 0.1
* nand(y, z)
M12a 13 4 2 2 P3306M
M12b 13 5 2 2 P3306M
M12c 13 4 14 14 N3306M
M12d 14 5 0 0 N3306M
R12e 12 13 0.1
* nor(nand(y, z), x)
M15a 16 12 2 2 P3306M
M15b 17 3 16 16 P3306M
M15c 17 12 0 0 N3306M
M15d 17 3 0 0 N3306M
R15e 15 17 0.1
* nand(x, z)
M18a 19 3 2 2 P3306M
M18b 19 5 2 2 P3306M
M18c 19 3 20 20 N3306M
M18d 20 5 0 0 N3306M
R18e 18 19 0.1
* nor(nand(x, z), y)
M21a 22 18 2 2 P3306M
M21b 23 4 22 22 P3306M
M21c 23 18 0 0 N3306M
M21d 23 4 0 0 N3306M
R21e 21 23 0.1
* nor(x, y, z)
M24a 25 3 2 2 P3306M
M24b 26 4 25 25 P3306M
M24c 27 5 26 26 P3306M
M24d 27 3 0 0 N3306M
M24e 27 4 0 0 N3306M
```

```
M24f 27 5 0 0 N3306M
R24g 24 27 0.1
* nor(nor(nand(x, y), z), nor(nand(y, z), x), nor(nand(x, z), y), nor(x,
* y, z))
M28a 29 9 2 2 P3306M
M28b 30 15 29 29 P3306M
M28c 31 21 30 30 P3306M
M28d 28 24 31 31 P3306M
M28e 28 9 0 0 N3306M
M28f 28 15 0 0 N3306M
M28g 28 21 0 0 N3306M
M28h 28 24 0 0 N3306M
C28i 28 0 1e-12
.TRAN 1.2e-09 1.2e-06 0 1.2e-09
.PRINT TRAN V(28)
.PROBE
.END
```

Appendix E

Latches and flip-flops

E.1 RS latch

```
setup {
  dt = tmax/1000;
  tmax = 4e-7;
  U = 3.3;
  Ri = 0.120792;
  Ropen = 0.601435;
  Rclosed = 1e10;
  C = 3.851953e-9;
}

graph {
  format = pdf;
  height = 360;
  labels = Q, "{/Symbol \330}Q";
  lalign = right;
  show = ucA, ucB;
  width = 640;
  xfmt = "%2.0l";
  xmult = "10^{-7} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/4, 3*tmax/4;
  yspace = 0.02;
}

R = 1, 0, 0, 0;
S = 0, 0, 1, 0;

//nor(R, Qn)
Rp1A = R*Rclosed+(1-R)*Ropen;
Rn1A = R*Ropen+(1-R)*Rclosed;
Rp2A = Qn*Rclosed+(1-Qn)*Ropen;
Rn2A = Qn*Ropen+(1-Qn)*Rclosed;
iA = (U-uc1A-uc2A-ucA)/Ri;
uc1A' = (iA-uc1A/Rp1A)/C &U/2;
uc2A' = (iA-uc2A/Rp2A)/C &U/2;
ucA' = (iA-ucA*(1/Rn1A+1/Rn2A))/(2*C) &0;
Q = 1 if ucA>=U/2;

//nor(Q, S)
Rp1B = Q*Rclosed+(1-Q)*Ropen;
Rn1B = Q*Ropen+(1-Q)*Rclosed;
Rp2B = S*Rclosed+(1-S)*Ropen;
Rn2B = S*Ropen+(1-S)*Rclosed;
iB = (U-uc1B-uc2B-ucB)/Ri;
uc1B' = (iB-uc1B/Rp1B)/C &0;
```

```
uc2B' = (iB-uc2B/Rp2B)/C &0;
ucB' = (iB-ucB*(1/Rn1B+1/Rn2B))/(2*C) &U;
Qn = 1 if ucB>=U/2;
```

E.2 D latch

```
setup {
  reach = 5;
  dt = tmax/1000;
  tmax = 5e-7;
  U = 3.3;
  Ri = 0.120792;
  Ropen = 0.601435;
  Rclosed = 1e10;
  C = 3.851953e-9;
}

graph {
  format = pdf;
  height = 360;
  labels = Q, "{/Symbol \330}Q";
  lalign = right;
  show = ucB, ucD;
  width = 640;
  xfmt = "%2.0l";
  xmult = "10^{-7} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/5, 4*tmax/5;
  yspace = 0.02;
}

D = 0, 1, 0, 0, 1;
Wr = 0, 1, 0, 1, 0;

//nand(D, Wr)
Rp1A = D*Rclosed+(1-D)*Ropen;
Rn1A = D*Ropen+(1-D)*Rclosed;
Rp2A = Wr*Rclosed+(1-Wr)*Ropen;
Rn2A = Wr*Ropen+(1-Wr)*Rclosed;
iA = (U-ucA2-uc1A-uc2A)/Ri;
ucA2' = (iA-ucA2*(1/Rp1A+1/Rp2A))/(2*C) &0;
uc1A' = (iA-uc1A/Rn1A)/C &U/2;
uc2A' = (iA-uc2A/Rn2A)/C &U/2;
ucA = uc1A+uc2A;
Dn = 1 if ucA>=U/2;

//nand(Dn, Qn)
Rp1B = Dn*Rclosed+(1-Dn)*Ropen;
Rn1B = Dn*Ropen+(1-Dn)*Rclosed;
Rp2B = Qn*Rclosed+(1-Qn)*Ropen;
Rn2B = Qn*Ropen+(1-Qn)*Rclosed;
iB = (U-ucB2-uc1B-uc2B)/Ri;
ucB2' = (iB-ucB2*(1/Rp1B+1/Rp2B))/(2*C) &U;
uc1B' = (iB-uc1B/Rn1B)/C &0;
uc2B' = (iB-uc2B/Rn2B)/C &0;
ucB = uc1B+uc2B;
Q = 1 if ucB>=U/2;

//nand(Dn, Wr)
Rp1C = Dn*Rclosed+(1-Dn)*Ropen;
Rn1C = Dn*Ropen+(1-Dn)*Rclosed;
Rp2C = Wr*Rclosed+(1-Wr)*Ropen;
Rn2C = Wr*Ropen+(1-Wr)*Rclosed;
iC = (U-ucC2-uc1C-uc2C)/Ri;
```

```
ucC2' = (iC-ucC2*(1/Rp1C+1/Rp2C))/(2*C) &0;
uc1C' = (iC-uc1C/Rn1C)/C &U/2;
uc2C' = (iC-uc2C/Rn2C)/C &U/2;
ucC = uc1C+uc2C;

//nand(nand(Dn, Wr), Q)
in1D = 1 if ucC>=U/2;
Rp1D = in1D*Rclosed+(1-in1D)*Ropen;
Rn1D = in1D*Ropen+(1-in1D)*Rclosed;
Rp2D = Q*Rclosed+(1-Q)*Ropen;
Rn2D = Q*Ropen+(1-Q)*Rclosed;
iD = (U-ucD2-uc1D-uc2D)/Ri;
ucD2' = (iD-ucD2*(1/Rp1D+1/Rp2D))/(2*C) &0;
uc1D' = (iD-uc1D/Rn1D)/C &U/2;
uc2D' = (iD-uc2D/Rn2D)/C &U/2;
ucD = uc1D+uc2D;
Qn = 1 if ucD>=U/2;
```

E.3 JK latch

```
setup {
  reach = 5;
  dt = tmax/1000;
  tmax = 5e-7;
  U = 3.3;
  Ri = 0.120792;
  Ropen = 0.601435;
  Rclosed = 1e10;
  C = 3.851953e-9;
}

graph {
  format = pdf;
  height = 360;
  labels = Q, "{/Symbol \330}Q";
  lalign = right;
  show = ucB, ucD;
  width = 640;
  xfmt = "%2.0l";
  xmult = "10^{-7} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/5, 4*tmax/5;
  yspace = 0.02;
}

J = 0, 1, 0, 0, 1;
K = 0, 0, 1, 0, 1;

//nand(J, Qn)
Rp1A = J*Rclosed+(1-J)*Ropen;
Rn1A = J*Ropen+(1-J)*Rclosed;
Rp2A = Qn*Rclosed+(1-Qn)*Ropen;
Rn2A = Qn*Ropen+(1-Qn)*Rclosed;
iA = (U-ucA2-uc1A-uc2A)/Ri;
ucA2' = (iA-ucA2*(1/Rp1A+1/Rp2A))/(2*C) &0;
uc1A' = (iA-uc1A/Rn1A)/C &U/2;
uc2A' = (iA-uc2A/Rn2A)/C &U/2;
ucA = uc1A+uc2A;

//nand(Qn, nand(J, Qn))
Rp1B = Qn*Rclosed+(1-Qn)*Ropen;
Rn1B = Qn*Ropen+(1-Qn)*Rclosed;
in2B = 1 if ucA>=U/2;
Rp2B = in2B*Rclosed+(1-in2B)*Ropen;
```

```
Rn2B = in2B*Ropen+(1-in2B)*Rclosed;
iB = (U-ucB2-uc1B-uc2B)/Ri;
ucB2' = (iB-ucB2*(1/Rp1B+1/Rp2B))/(2*C) &U;
uc1B' = (iB-uc1B/Rn1B)/C &0;
uc2B' = (iB-uc2B/Rn2B)/C &0;
ucB = uc1B+uc2B;
Q = 1 if ucB>=U/2;

//nand(K, Q)
Rp1C = K*Rclosed+(1-K)*Ropen;
Rn1C = K*Ropen+(1-K)*Rclosed;
Rp2C = Q*Rclosed+(1-Q)*Ropen;
Rn2C = Q*Ropen+(1-Q)*Rclosed;
iC = (U-ucC2-uc1C-uc2C)/Ri;
ucC2' = (iC-ucC2*(1/Rp1C+1/Rp2C))/(2*C) &U;
uc1C' = (iC-uc1C/Rn1C)/C &0;
uc2C' = (iC-uc2C/Rn2C)/C &0;
ucC = uc1C+uc2C;

//nand(Q, nand(K, Q))
Rp1D = Q*Rclosed+(1-Q)*Ropen;
Rn1D = Q*Ropen+(1-Q)*Rclosed;
in2D = 1 if ucC>=U/2;
Rp2D = in2D*Rclosed+(1-in2D)*Ropen;
Rn2D = in2D*Ropen+(1-in2D)*Rclosed;
iD = (U-ucD2-uc1D-uc2D)/Ri;
ucD2' = (iD-ucD2*(1/Rp1D+1/Rp2D))/(2*C) &0;
uc1D' = (iD-uc1D/Rn1D)/C &U/2;
uc2D' = (iD-uc2D/Rn2D)/C &U/2;
ucD = uc1D+uc2D;
Qn = 1 if ucD>=U/2;
```

E.4 D flip-flop

```
setup {
  dt = tmax/1000;
  tmax = 1e-6;
  U = 3.3;
  Ri = 0.120792;
  Ropen = 0.601435;
  Rclosed = 1e10;
  C = 3.851953e-9;
}

graph {
  format = pdf;
  height = 360;
  labels = Q, "{/Symbol \330}Q";
  lalign = right;
  mxtics = 2;
  show = ucB, ucD;
  width = 640;
  xfmt = "%2.0l";
  xmult = "10^{-7} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/5, 4.5*tmax/5;
  yspace = 0.02;
}

D = 0, 1, 0, 0, 1;
CLK = 1, 0, 1, 0, 1, 0, 1, 0, 1, 0;
Wr = 0, 1, 0, 1, 1;

//nand(D, Wr, CLK)
```

```
Rp1A = D*Rclosed+(1-D)*Ropen;
Rn1A = D*Ropen+(1-D)*Rclosed;
Rp2A = Wr*Rclosed+(1-Wr)*Ropen;
Rn2A = Wr*Ropen+(1-Wr)*Rclosed;
Rp3A = CLK*Rclosed+(1-CLK)*Ropen;
Rn3A = CLK*Ropen+(1-CLK)*Rclosed;
iA = (U-ucA2-uc1A-uc2A-uc3A)/Ri;
ucA2' = (iA-ucA2*(1/Rp1A+1/Rp2A+1/Rp3A))/(3*C) &0;
uc1A' = (iA-uc1A/Rn1A)/C &U/3;
uc2A' = (iA-uc2A/Rn2A)/C &U/3;
uc3A' = (iA-uc3A/Rn3A)/C &U/3;
ucA = uc1A+uc2A+uc3A;
Dn = 1 if ucA>=U/2;

//nand(Dn, Qn)
Rp1B = Dn*Rclosed+(1-Dn)*Ropen;
Rn1B = Dn*Ropen+(1-Dn)*Rclosed;
Rp2B = Qn*Rclosed+(1-Qn)*Ropen;
Rn2B = Qn*Ropen+(1-Qn)*Rclosed;
iB = (U-ucB2-uc1B-uc2B)/Ri;
ucB2' = (iB-ucB2*(1/Rp1B+1/Rp2B))/(2*C) &U;
uc1B' = (iB-uc1B/Rn1B)/C &0;
uc2B' = (iB-uc2B/Rn2B)/C &0;
ucB = uc1B+uc2B;
Q = 1 if ucB>=U/2;

//nand(Dn, Wr, CLK)
Rp1C = Dn*Rclosed+(1-Dn)*Ropen;
Rn1C = Dn*Ropen+(1-Dn)*Rclosed;
Rp2C = Wr*Rclosed+(1-Wr)*Ropen;
Rn2C = Wr*Ropen+(1-Wr)*Rclosed;
Rp3C = CLK*Rclosed+(1-CLK)*Ropen;
Rn3C = CLK*Ropen+(1-CLK)*Rclosed;
iC = (U-ucC2-uc1C-uc2C-uc3C)/Ri;
ucC2' = (iC-ucC2*(1/Rp1C+1/Rp2C+1/Rp3C))/(3*C) &U;
uc1C' = (iC-uc1C/Rn1C)/C &0;
uc2C' = (iC-uc2C/Rn2C)/C &0;
uc3C' = (iC-uc3C/Rn3C)/C &0;
ucC = uc1C+uc2C+uc3C;

//nand(Q, nand(Dn, Wr, CLK))
Rp1D = Q*Rclosed+(1-Q)*Ropen;
Rn1D = Q*Ropen+(1-Q)*Rclosed;
in2D = 1 if ucC>=U/2;
Rp2D = in2D*Rclosed+(1-in2D)*Ropen;
Rn2D = in2D*Ropen+(1-in2D)*Rclosed;
iD = (U-ucD2-uc1D-uc2D)/Ri;
ucD2' = (iD-ucD2*(1/Rp1D+1/Rp2D))/(2*C) &0;
uc1D' = (iD-uc1D/Rn1D)/C &U/2;
uc2D' = (iD-uc2D/Rn2D)/C &U/2;
ucD = uc1D+uc2D;
Qn = 1 if ucD>=U/2;
```

E.5 JK flip-flop

```
setup {
  dt = tmax/1000;
  tmax = 1e-6;
  U = 3.3;
  Ri = 0.120792;
  Ropen = 0.601435;
  Rclosed = 1e10;
  C = 3.851953e-9;
}

```

```
graph {
  format = pdf;
  height = 360;
  labels = Q, "{/Symbol \330}Q";
  lalign = right;
  mxtics = 2;
  show = ucG, ucI;
  width = 640;
  xfmt = "%2.0l";
  xmult = "10^{-7} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/5, 4.5*tmax/5;
  yspace = 0.02;
}

J = 1, 0, 1, 1, 0;
K = 0, 1, 1, 1, 0;
CLK = 1, 0, 1, 0, 1, 0, 1, 0, 1, 0;

//nand(J, Qn2, CLK)
Rp1A = J*Rclosed+(1-J)*Ropen;
Rn1A = J*Ropen+(1-J)*Rclosed;
Rp2A = Qn2*Rclosed+(1-Qn2)*Ropen;
Rn2A = Qn2*Ropen+(1-Qn2)*Rclosed;
Rp3A = CLK*Rclosed+(1-CLK)*Ropen;
Rn3A = CLK*Ropen+(1-CLK)*Rclosed;
iA = (U-ucA2-uc1A-uc2A-uc3A)/Ri;
ucA2' = (iA-ucA2*(1/Rp1A+1/Rp2A+1/Rp3A))/(3*C) &0;
uc1A' = (iA-uc1A/Rn1A)/C &U/3;
uc2A' = (iA-uc2A/Rn2A)/C &U/3;
uc3A' = (iA-uc3A/Rn3A)/C &U/3;
ucA = uc1A+uc2A+uc3A;

//nand(Qn1, nand(J, Qn2, CLK))
Rp1B = Qn1*Rclosed+(1-Qn1)*Ropen;
Rn1B = Qn1*Ropen+(1-Qn1)*Rclosed;
in2B = 1 if ucA>=U/2;
Rp2B = in2B*Rclosed+(1-in2B)*Ropen;
Rn2B = in2B*Ropen+(1-in2B)*Rclosed;
iB = (U-ucB2-uc1B-uc2B)/Ri;
ucB2' = (iB-ucB2*(1/Rp1B+1/Rp2B))/(2*C) &U;
uc1B' = (iB-uc1B/Rn1B)/C &0;
uc2B' = (iB-uc2B/Rn2B)/C &0;
ucB = uc1B+uc2B;
Q1 = 1 if ucB>=U/2;

//nand(K, Q2, CLK)
Rp1C = K*Rclosed+(1-K)*Ropen;
Rn1C = K*Ropen+(1-K)*Rclosed;
Rp2C = Q2*Rclosed+(1-Q2)*Ropen;
Rn2C = Q2*Ropen+(1-Q2)*Rclosed;
Rp3C = CLK*Rclosed+(1-CLK)*Ropen;
Rn3C = CLK*Ropen+(1-CLK)*Rclosed;
iC = (U-ucC2-uc1C-uc2C-uc3C)/Ri;
ucC2' = (iC-ucC2*(1/Rp1C+1/Rp2C+1/Rp3C))/(3*C) &0;
uc1C' = (iC-uc1C/Rn1C)/C &U/3;
uc2C' = (iC-uc2C/Rn2C)/C &U/3;
uc3C' = (iC-uc3C/Rn3C)/C &U/3;
ucC = uc1C+uc2C+uc3C;

//nand(Q1, nand(K, Q2, CLK))
Rp1D = Q1*Rclosed+(1-Q1)*Ropen;
Rn1D = Q1*Ropen+(1-Q1)*Rclosed;
in2D = 1 if ucC>=U/2;
Rp2D = in2D*Rclosed+(1-in2D)*Ropen;
Rn2D = in2D*Ropen+(1-in2D)*Rclosed;
iD = (U-ucD2-uc1D-uc2D)/Ri;
```

```
ucD2' = (iD-ucD2*(1/Rp1D+1/Rp2D))/(2*C) &0;
uc1D' = (iD-uc1D/Rn1D)/C &U/2;
uc2D' = (iD-uc2D/Rn2D)/C &U/2;
ucD = uc1D+uc2D;
Qn1 = 1 if ucD>=U/2;

//not(CLK)
Rp1E = CLK*Rclosed+(1-CLK)*Ropen;
Rn1E = CLK*Ropen+(1-CLK)*Rclosed;
iE = (U-uc1E-ucE)/Ri;
uc1E' = (iE-uc1E/Rp1E)/C &0;
ucE' = (iE-ucE*(1/Rn1E))/C &U;
CLKn = 1 if ucE>=U/2;

//nand(Q1, CLKn)
Rp1F = Q1*Rclosed+(1-Q1)*Ropen;
Rn1F = Q1*Ropen+(1-Q1)*Rclosed;
Rp2F = CLKn*Rclosed+(1-CLKn)*Ropen;
Rn2F = CLKn*Ropen+(1-CLKn)*Rclosed;
iF = (U-ucF2-uc1F-uc2F)/Ri;
ucF2' = (iF-ucF2*(1/Rp1F+1/Rp2F))/(2*C) &0;
uc1F' = (iF-uc1F/Rn1F)/C &U/2;
uc2F' = (iF-uc2F/Rn2F)/C &U/2;
ucF = uc1F+uc2F;

//nand(Qn2, nand(Q1, CLKn))
Rp1G = Qn2*Rclosed+(1-Qn2)*Ropen;
Rn1G = Qn2*Ropen+(1-Qn2)*Rclosed;
in2G = 1 if ucF>=U/2;
Rp2G = in2G*Rclosed+(1-in2G)*Ropen;
Rn2G = in2G*Ropen+(1-in2G)*Rclosed;
iG = (U-ucG2-uc1G-uc2G)/Ri;
ucG2' = (iG-ucG2*(1/Rp1G+1/Rp2G))/(2*C) &U;
uc1G' = (iG-uc1G/Rn1G)/C &0;
uc2G' = (iG-uc2G/Rn2G)/C &0;
ucG = uc1G+uc2G;
Q2 = 1 if ucG>=U/2;

//nand(Qn1, CLKn)
Rp1H = Qn1*Rclosed+(1-Qn1)*Ropen;
Rn1H = Qn1*Ropen+(1-Qn1)*Rclosed;
Rp2H = CLKn*Rclosed+(1-CLKn)*Ropen;
Rn2H = CLKn*Ropen+(1-CLKn)*Rclosed;
iH = (U-ucH2-uc1H-uc2H)/Ri;
ucH2' = (iH-ucH2*(1/Rp1H+1/Rp2H))/(2*C) &0;
uc1H' = (iH-uc1H/Rn1H)/C &U/2;
uc2H' = (iH-uc2H/Rn2H)/C &U/2;
ucH = uc1H+uc2H;

//nand(Q2, nand(Qn1, CLKn))
Rp1I = Q2*Rclosed+(1-Q2)*Ropen;
Rn1I = Q2*Ropen+(1-Q2)*Rclosed;
in2I = 1 if ucH>=U/2;
Rp2I = in2I*Rclosed+(1-in2I)*Ropen;
Rn2I = in2I*Ropen+(1-in2I)*Rclosed;
iI = (U-ucI2-uc1I-uc2I)/Ri;
ucI2' = (iI-ucI2*(1/Rp1I+1/Rp2I))/(2*C) &0;
uc1I' = (iI-uc1I/Rn1I)/C &U/2;
uc2I' = (iI-uc2I/Rn2I)/C &U/2;
ucI = uc1I+uc2I;
Qn2 = 1 if ucI>=U/2;
```

E.6 T flip-flop

```
setup {
```

```
  tmax = 1e-6;
  dt = tmax/1000;
  U = 3.3;
  Ri = 0.120792;
  Ropen = 0.601435;
  Rclosed = 1e10;
  C = 3.851953e-9;
}

graph {
  format = pdf;
  height = 360;
  labels = Q, "{/Symbol \330}Q";
  lalign = right;
  mxtics = 2;
  show = ucG, ucI;
  width = 640;
  xfmt = "%2.0l";
  xmult = "10^{-7} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/5, 4.5*tmax/5;
  yspace = 0.02;
}

CLK = 1, 0, 1, 0, 1, 0, 1, 0, 1, 0;
En = 0, 1, 1, 1, 1;
RESn = En;
T = 0, 1, 0, 0, 1;

//not(CLK)
Rp1A = CLK*Rclosed+(1-CLK)*Ropen;
Rn1A = CLK*Ropen+(1-CLK)*Rclosed;
iA = (U-uc1A-ucA)/Ri;
uc1A' = (iA-uc1A/Rp1A)/C &0;
ucA' = (iA-ucA*(1/Rn1A))/C &U;
CLKn = 1 if ucA>=U/2;

//nand(RESn, T, Qn, CLK, En)
Rp1B = RESn*Rclosed+(1-RESn)*Ropen;
Rn1B = RESn*Ropen+(1-RESn)*Rclosed;
Rp2B = T*Rclosed+(1-T)*Ropen;
Rn2B = T*Ropen+(1-T)*Rclosed;
Rp3B = Qn*Rclosed+(1-Qn)*Ropen;
Rn3B = Qn*Ropen+(1-Qn)*Rclosed;
Rp4B = CLK*Rclosed+(1-CLK)*Ropen;
Rn4B = CLK*Ropen+(1-CLK)*Rclosed;
Rp5B = En*Rclosed+(1-En)*Ropen;
Rn5B = En*Ropen+(1-En)*Rclosed;
iB = (U-ucB2-uc1B-uc2B-uc3B-uc4B-uc5B)/Ri;
ucB2' = (iB-ucB2*(1/Rp1B+1/Rp2B+1/Rp3B+1/Rp4B+1/Rp5B))/(5*C) &0;
uc1B' = (iB-uc1B/Rn1B)/C &U/5;
uc2B' = (iB-uc2B/Rn2B)/C &U/5;
uc3B' = (iB-uc3B/Rn3B)/C &U/5;
uc4B' = (iB-uc4B/Rn4B)/C &U/5;
uc5B' = (iB-uc5B/Rn5B)/C &U/5;
ucB = uc1B+uc2B+uc3B+uc4B+uc5B;

//nand(QMn, nand(RESn, T, Qn, CLK, En))
Rp1C = QMn*Rclosed+(1-QMn)*Ropen;
Rn1C = QMn*Ropen+(1-QMn)*Rclosed;
in2C = 1 if ucB>=U/2;
Rp2C = in2C*Rclosed+(1-in2C)*Ropen;
Rn2C = in2C*Ropen+(1-in2C)*Rclosed;
iC = (U-ucC2-uc1C-uc2C)/Ri;
ucC2' = (iC-ucC2*(1/Rp1C+1/Rp2C))/(2*C) &U;
uc1C' = (iC-uc1C/Rn1C)/C &0;
uc2C' = (iC-uc2C/Rn2C)/C &0;
```

```
ucC = uc1C+uc2C;
QM = 1 if ucC>=U/2;

//nand(T, Q, CLK, En)
Rp1D = T*Rclosed+(1-T)*Ropen;
Rn1D = T*Ropen+(1-T)*Rclosed;
Rp2D = Q*Rclosed+(1-Q)*Ropen;
Rn2D = Q*Ropen+(1-Q)*Rclosed;
Rp3D = CLK*Rclosed+(1-CLK)*Ropen;
Rn3D = CLK*Ropen+(1-CLK)*Rclosed;
Rp4D = En*Rclosed+(1-En)*Ropen;
Rn4D = En*Ropen+(1-En)*Rclosed;
iD = (U-ucD2-uc1D-uc2D-uc3D-uc4D)/Ri;
ucD2' = (iD-ucD2*(1/Rp1D+1/Rp2D+1/Rp3D+1/Rp4D))/(4*C) &0;
uc1D' = (iD-uc1D/Rn1D)/C &U/4;
uc2D' = (iD-uc2D/Rn2D)/C &U/4;
uc3D' = (iD-uc3D/Rn3D)/C &U/4;
uc4D' = (iD-uc4D/Rn4D)/C &U/4;
ucD = uc1D+uc2D+uc3D+uc4D;

//nand(RESn, QM, nand(T, Q, CLK, En))
Rp1E = RESn*Rclosed+(1-RESn)*Ropen;
Rn1E = RESn*Ropen+(1-RESn)*Rclosed;
Rp2E = QM*Rclosed+(1-QM)*Ropen;
Rn2E = QM*Ropen+(1-QM)*Rclosed;
in3E = 1 if ucD>=U/2;
Rp3E = in3E*Rclosed+(1-in3E)*Ropen;
Rn3E = in3E*Ropen+(1-in3E)*Rclosed;
iE = (U-ucE2-uc1E-uc2E-uc3E)/Ri;
ucE2' = (iE-ucE2*(1/Rp1E+1/Rp2E+1/Rp3E))/(3*C) &U;
uc1E' = (iE-uc1E/Rn1E)/C &0;
uc2E' = (iE-uc2E/Rn2E)/C &0;
uc3E' = (iE-uc3E/Rn3E)/C &0;
ucE = uc1E+uc2E+uc3E;
QMn = 1 if ucE>=U/2;

//nand(RESn, QM, CLKn)
Rp1F = RESn*Rclosed+(1-RESn)*Ropen;
Rn1F = RESn*Ropen+(1-RESn)*Rclosed;
Rp2F = QM*Rclosed+(1-QM)*Ropen;
Rn2F = QM*Ropen+(1-QM)*Rclosed;
Rp3F = CLKn*Rclosed+(1-CLKn)*Ropen;
Rn3F = CLKn*Ropen+(1-CLKn)*Rclosed;
iF = (U-ucF2-uc1F-uc2F-uc3F)/Ri;
ucF2' = (iF-ucF2*(1/Rp1F+1/Rp2F+1/Rp3F))/(3*C) &0;
uc1F' = (iF-uc1F/Rn1F)/C &U/3;
uc2F' = (iF-uc2F/Rn2F)/C &U/3;
uc3F' = (iF-uc3F/Rn3F)/C &U/3;
ucF = uc1F+uc2F+uc3F;

//nand(Qn, nand(RESn, QM, CLKn))
Rp1G = Qn*Rclosed+(1-Qn)*Ropen;
Rn1G = Qn*Ropen+(1-Qn)*Rclosed;
in2G = 1 if ucF>=U/2;
Rp2G = in2G*Rclosed+(1-in2G)*Ropen;
Rn2G = in2G*Ropen+(1-in2G)*Rclosed;
iG = (U-ucG2-uc1G-uc2G)/Ri;
ucG2' = (iG-ucG2*(1/Rp1G+1/Rp2G))/(2*C) &U;
uc1G' = (iG-uc1G/Rn1G)/C &0;
uc2G' = (iG-uc2G/Rn2G)/C &0;
ucG = uc1G+uc2G;
Q = 1 if ucG>=U/2;

//nand(QMn, CLKn)
Rp1H = QMn*Rclosed+(1-QMn)*Ropen;
Rn1H = QMn*Ropen+(1-QMn)*Rclosed;
Rp2H = CLKn*Rclosed+(1-CLKn)*Ropen;
Rn2H = CLKn*Ropen+(1-CLKn)*Rclosed;
```

```
iH = (U-ucH2-uc1H-uc2H)/Ri;
ucH2' = (iH-ucH2*(1/Rp1H+1/Rp2H))/(2*C) &0;
uc1H' = (iH-uc1H/Rn1H)/C &U/2;
uc2H' = (iH-uc2H/Rn2H)/C &U/2;
ucH = uc1H+uc2H;

//nand(RESn, Q, nand(QMn, CLKn))
Rp1I = RESn*Rclosed+(1-RESn)*Ropen;
Rn1I = RESn*Ropen+(1-RESn)*Rclosed;
Rp2I = Q*Rclosed+(1-Q)*Ropen;
Rn2I = Q*Ropen+(1-Q)*Rclosed;
in3I = 1 if ucH>=U/2;
Rp3I = in3I*Rclosed+(1-in3I)*Ropen;
Rn3I = in3I*Ropen+(1-in3I)*Rclosed;
iI = (U-ucI2-uc1I-uc2I-uc3I)/Ri;
ucI2' = (iI-ucI2*(1/Rp1I+1/Rp2I+1/Rp3I))/(3*C) &0;
uc1I' = (iI-uc1I/Rn1I)/C &U/3;
uc2I' = (iI-uc2I/Rn2I)/C &U/3;
uc3I' = (iI-uc3I/Rn3I)/C &U/3;
ucI = uc1I+uc2I+uc3I;
Qn = 1 if ucI>=U/2;
```

Appendix F

Latches and flip-flops (SPICE)

F.1 RS latch

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* R = 1, 0, 0, 0
V3 3 0 PWL(0 3.3 1e-07 3.3 1e-07 0 4e-07 0)
* S = 0, 0, 1, 0
V4 4 0 PWL(0 0 2e-07 0 2e-07 3.3 3e-07 3.3 3e-07 0 4e-07 0)
* Q = nor(R, Qn)
M5a 6 3 2 2 P3306M
M5b 8 7 6 6 P3306M
M5c 8 3 0 0 N3306M
M5d 8 7 0 0 N3306M
R5e 5 8 0.1
C5f 8 0 1e-12
* Qn = nor(S, Q)
M7a 9 4 2 2 P3306M
M7b 10 5 9 9 P3306M
M7c 10 4 0 0 N3306M
M7d 10 5 0 0 N3306M
R7e 7 10 0.1
C7f 10 0 1e-12
.TRAN 4e-10 4e-07 0 4e-10
.PRINT TRAN V(5) V(7)
.PROBE
.END
```

F.2 D latch

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* D = 0, 1, 0, 0, 1
V3 3 0 PWL(0 0 1e-07 0 1e-07 3.3 2e-07 3.3 2e-07 0 4e-07 0 4e-07 3.3
+ 5e-07 3.3)
* Wr = 0, 1, 0, 1, 0
```

```
V4 4 0 PWL(0 0 1e-07 0 1e-07 3.3 2e-07 3.3 2e-07 0 3e-07 0 3e-07 3.3
+ 4e-07 3.3 4e-07 0 5e-07 0)
* Dn = nand(D, Wr)
M5a 6 3 2 2 P3306M
M5b 6 4 2 2 P3306M
M5c 6 3 7 7 N3306M
M5d 7 4 0 0 N3306M
R5e 5 6 0.1
C5f 6 0 1e-12
* Q = nand(Dn, Qn)
M8a 9 5 2 2 P3306M
M8b 9 10 2 2 P3306M
M8c 9 5 11 11 N3306M
M8d 11 10 0 0 N3306M
R8e 8 9 0.1
C8f 9 0 1e-12
* nand(Dn, Wr)
M12a 13 5 2 2 P3306M
M12b 13 4 2 2 P3306M
M12c 13 5 14 14 N3306M
M12d 14 4 0 0 N3306M
R12e 12 13 0.1
* Qn = nand(nand(Dn, Wr), Q)
M10a 15 12 2 2 P3306M
M10b 15 8 2 2 P3306M
M10c 15 12 16 16 N3306M
M10d 16 8 0 0 N3306M
R10e 10 15 0.1
C10f 15 0 1e-12
.TRAN 5e-10 5e-07 0 5e-10
.PRINT TRAN V(8) V(10)
.PROBE
.END
```

F.3 JK latch

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* J = 0, 1, 0, 0, 1
V3 3 0 PWL(0 0 1e-07 0 1e-07 3.3 2e-07 3.3 2e-07 0 4e-07 0 4e-07 3.3 5e-07
+ 3.3)
* K = 0, 0, 1, 0, 1
V4 4 0 PWL(0 0 2e-07 0 2e-07 3.3 3e-07 3.3 3e-07 0 4e-07 0 4e-07 3.3 5e-07
+ 3.3)
* nand(J, Qn)
M5a 6 3 2 2 P3306M
M5b 6 7 2 2 P3306M
M5c 6 3 8 8 N3306M
M5d 8 7 0 0 N3306M
R5e 5 6 0.1
* Q = nand(Qn, nand(J, Qn))
M9a 10 7 2 2 P3306M
M9b 10 5 2 2 P3306M
M9c 10 7 11 11 N3306M
M9d 11 5 0 0 N3306M
R9e 9 10 0.1
C9f 10 0 1e-12
* nand(K, Q)
M12a 13 4 2 2 P3306M
M12b 13 9 2 2 P3306M
```

```
M12c 13 4 14 14 N3306M
M12d 14 9 0 0 N3306M
R12e 12 13 0.1
* Qn = nand(Q, nand(K, Q))
M7a 15 9 2 2 P3306M
M7b 15 12 2 2 P3306M
M7c 15 9 16 16 N3306M
M7d 16 12 0 0 N3306M
R7e 7 15 0.1
C7f 15 0 1e-12
.TRAN 5e-10 5e-07 0 5e-10
.PRINT TRAN V(9) V(7)
.PROBE
.END
```

F.4 D flip-flop

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* D = 0, 1, 0, 0, 1
V3 3 0 PWL(0 0 2e-07 0 2e-07 3.3 4e-07 3.3 4e-07 0 8e-07 0 8e-07 3.3 1e-06
+ 3.3)
* CLK = 1, 0, 1, 0, 1, 0, 1, 0, 1, 0
V4 4 0 PWL(0 3.3 1e-07 3.3 1e-07 0 2e-07 0 2e-07 3.3 3e-07 3.3 3e-07 0
+ 4e-07 0 4e-07 3.3 5e-07 3.3 5e-07 0 6e-07 0 6e-07 3.3 7e-07 3.3 7e-07 0
+ 8e-07 0 8e-07 3.3 9e-07 3.3 9e-07 0 1e-06 0)
* Wr = 0, 1, 0, 1, 1
V5 5 0 PWL(0 0 2e-07 0 2e-07 3.3 4e-07 3.3 4e-07 0 6e-07 0 6e-07 3.3 1e-06
+ 3.3)
* Dn = nand(D, Wr, CLK)
M6a 7 3 2 2 P3306M
M6b 7 5 2 2 P3306M
M6c 7 4 2 2 P3306M
M6d 7 3 8 8 N3306M
M6e 8 5 9 9 N3306M
M6f 9 4 0 0 N3306M
R6g 6 7 0.1
C6h 7 0 1e-12
* Q = nand(Dn, Qn)
M10a 11 6 2 2 P3306M
M10b 11 12 2 2 P3306M
M10c 11 6 13 13 N3306M
M10d 13 12 0 0 N3306M
R10e 10 11 0.1
C10f 11 0 1e-12
* nand(Dn, Wr, CLK)
M14a 15 6 2 2 P3306M
M14b 15 5 2 2 P3306M
M14c 15 4 2 2 P3306M
M14d 15 6 16 16 N3306M
M14e 16 5 17 17 N3306M
M14f 17 4 0 0 N3306M
R14g 14 15 0.1
* Qn = nand(Q, nand(Dn, Wr, CLK))
M12a 18 10 2 2 P3306M
M12b 18 14 2 2 P3306M
M12c 18 10 19 19 N3306M
M12d 19 14 0 0 N3306M
R12e 12 18 0.1
C12f 18 0 1e-12
```

```
.TRAN 1e-09 1e-06 0 1e-09
.PRINT TRAN V(10) V(12)
.PROBE
.END
```

F.5 JK flip-flop

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* J = 1, 0, 1, 1, 0
V3 3 0 PWL(0 3.3 2e-07 3.3 2e-07 0 4e-07 0 4e-07 3.3 8e-07 3.3 8e-07 0
+ 1e-06 0)
* K = 0, 1, 1, 1, 0
V4 4 0 PWL(0 0 2e-07 0 2e-07 3.3 8e-07 3.3 8e-07 0 1e-06 0)
* CLK = 1, 0, 1, 0, 1, 0, 1, 0, 1, 0
V5 5 0 PWL(0 3.3 1e-07 3.3 1e-07 0 2e-07 0 2e-07 3.3 3e-07 3.3 3e-07 0
+ 4e-07 0 4e-07 3.3 5e-07 3.3 5e-07 0 6e-07 0 6e-07 3.3 7e-07 3.3
+ 7e-07 0 8e-07 0 8e-07 3.3 9e-07 3.3 9e-07 0 1e-06 0)
* nand(J, Qn2, CLK)
M6a 7 3 2 2 P3306M
M6b 7 8 2 2 P3306M
M6c 7 5 2 2 P3306M
M6d 7 3 9 9 N3306M
M6e 9 8 10 10 N3306M
M6f 10 5 0 0 N3306M
R6g 6 7 0.1
* Q1 = nand(Qn1, nand(J, Qn2, CLK))
M11a 12 13 2 2 P3306M
M11b 12 6 2 2 P3306M
M11c 12 13 14 14 N3306M
M11d 14 6 0 0 N3306M
R11e 11 12 0.1
C11f 12 0 1e-12
* nand(K, Q2, CLK)
M15a 16 4 2 2 P3306M
M15b 16 17 2 2 P3306M
M15c 16 5 2 2 P3306M
M15d 16 4 18 18 N3306M
M15e 18 17 19 19 N3306M
M15f 19 5 0 0 N3306M
R15g 15 16 0.1
* Qn1 = nand(Q1, nand(K, Q2, CLK))
M13a 20 11 2 2 P3306M
M13b 20 15 2 2 P3306M
M13c 20 11 21 21 N3306M
M13d 21 15 0 0 N3306M
R13e 13 20 0.1
C13f 20 0 1e-12
* CLKn = not(CLK)
M22a 23 5 2 2 P3306M
M22b 23 5 0 0 N3306M
R22c 22 23 0.1
C22d 23 0 1e-12
* nand(Q1, CLKn)
M24a 25 11 2 2 P3306M
M24b 25 22 2 2 P3306M
M24c 25 11 26 26 N3306M
M24d 26 22 0 0 N3306M
R24e 24 25 0.1
* Q2 = nand(Qn2, nand(Q1, CLKn))
```

```
M17a 27 8 2 2 P3306M
M17b 27 24 2 2 P3306M
M17c 27 8 28 28 N3306M
M17d 28 24 0 0 N3306M
R17e 17 27 0.1
C17f 27 0 1e-12
* nand(Qn1, CLKn)
M29a 30 13 2 2 P3306M
M29b 30 22 2 2 P3306M
M29c 30 13 31 31 N3306M
M29d 31 22 0 0 N3306M
R29e 29 30 0.1
* Qn2 = nand(Q2, nand(Qn1, CLKn))
M8a 32 17 2 2 P3306M
M8b 32 29 2 2 P3306M
M8c 32 17 33 33 N3306M
M8d 33 29 0 0 N3306M
R8e 8 32 0.1
C8f 32 0 1e-12
.TRAN 1e-09 1e-06 0 1e-09
.PRINT TRAN V(11) V(13) V(22) V(17) V(8)
.PROBE
.END
```

F.6 T flip-flop

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* CLK = 1, 0, 1, 0, 1, 0, 1, 0, 1, 0
V3 3 0 PWL(0 3.3 1e-07 3.3 1e-07 0 2e-07 0 2e-07 3.3 3e-07 3.3 3e-07 0
+ 4e-07 0 4e-07 3.3 5e-07 3.3 5e-07 0 6e-07 0 6e-07 3.3 7e-07 3.3
+ 7e-07 0 8e-07 0 8e-07 3.3 9e-07 3.3 9e-07 0 1e-06 0)
* CLKn = not(CLK)
M4a 5 3 2 2 P3306M
M4b 5 3 0 0 N3306M
R4c 4 5 0.1
C4d 5 0 1e-12
* En = 0, 1, 1, 1, 1
V6 6 0 PWL(0 0 2e-07 0 2e-07 3.3 1e-06 3.3)
* RESn = En
* T = 0, 1, 0, 0, 1
V7 7 0 PWL(0 0 2e-07 0 2e-07 3.3 4e-07 3.3 4e-07 0 8e-07 0 8e-07 3.3 1e-06
+ 3.3)
* nand(RESn, T, Qn, CLK, En)
M8a 9 6 2 2 P3306M
M8b 9 7 2 2 P3306M
M8c 9 10 2 2 P3306M
M8d 9 3 2 2 P3306M
M8e 9 6 2 2 P3306M
M8f 9 6 11 11 N3306M
M8g 11 7 12 12 N3306M
M8h 12 10 13 13 N3306M
M8i 13 3 14 14 N3306M
M8j 14 6 0 0 N3306M
R8k 8 9 0.1
* QM = nand(QMn, nand(RESn, T, Qn, CLK, En))
M15a 16 17 2 2 P3306M
M15b 16 8 2 2 P3306M
M15c 16 17 18 18 N3306M
M15d 18 8 0 0 N3306M
```

```
R15e 15 16 0.1
C15f 16 0 1e-12
* nand(T, Q, CLK, En)
M19a 20 7 2 2 P3306M
M19b 20 21 2 2 P3306M
M19c 20 3 2 2 P3306M
M19d 20 6 2 2 P3306M
M19e 20 7 22 22 N3306M
M19f 22 21 23 23 N3306M
M19g 23 3 24 24 N3306M
M19h 24 6 0 0 N3306M
R19i 19 20 0.1
* QMn = nand(RESn, QM, nand(T, Q, CLK, En))
M17a 25 6 2 2 P3306M
M17b 25 15 2 2 P3306M
M17c 25 19 2 2 P3306M
M17d 25 6 26 26 N3306M
M17e 26 15 27 27 N3306M
M17f 27 19 0 0 N3306M
R17g 17 25 0.1
C17h 25 0 1e-12
* nand(RESn, QM, CLKn)
M28a 29 6 2 2 P3306M
M28b 29 15 2 2 P3306M
M28c 29 4 2 2 P3306M
M28d 29 6 30 30 N3306M
M28e 30 15 31 31 N3306M
M28f 31 4 0 0 N3306M
R28g 28 29 0.1
* Q = nand(Qn, nand(RESn, QM, CLKn))
M21a 32 10 2 2 P3306M
M21b 32 28 2 2 P3306M
M21c 32 10 33 33 N3306M
M21d 33 28 0 0 N3306M
R21e 21 32 0.1
C21f 32 0 1e-12
* nand(QMn, CLKn)
M34a 35 17 2 2 P3306M
M34b 35 4 2 2 P3306M
M34c 35 17 36 36 N3306M
M34d 36 4 0 0 N3306M
R34e 34 35 0.1
* Qn = nand(RESn, Q, nand(QMn, CLKn))
M10a 37 6 2 2 P3306M
M10b 37 21 2 2 P3306M
M10c 37 34 2 2 P3306M
M10d 37 6 38 38 N3306M
M10e 38 21 39 39 N3306M
M10f 39 34 0 0 N3306M
R10g 10 37 0.1
C10h 37 0 1e-12
.TRAN 1e-09 1e-06 0 1e-09
.PRINT TRAN V(4) V(6) V(15) V(17) V(21) V(10)
.PROBE
.END
```

Appendix G

VLSI

G.1 Half adder

```
setup {
  dt = tmax/1000;
  tmax = 4e-7;
  U = 3.3;
  Ri = 0.120792;
  Ropen = 0.601435;
  Rclosed = 1e10;
  C = 3.851953e-9;
}

graph {
  format = pdf;
  height = 360;
  labels = ucA, ucF;
  show = ucA, ucF;
  width = 640;
  xbase = 1e8;
  xfmt = "%2.0l";
  xmult = "10^{-8} [s]";
  xmultx = 1.09;
  xmulty = -0.09;
  xtics = 0, tmax/8, 7*tmax/8;
  yspace = 0.02;
}

x = 1;
y = 1;

//not(x)
Rp1B = x*Rclosed+(1-x)*Ropen;
Rn1B = x*Ropen+(1-x)*Rclosed;
iB = (U-uc1B-ucB)/Ri;
uc1B' = (iB-uc1B/Rp1B)/C &0;
ucB' = (iB-ucB*(1/Rn1B))/C &U;

//nand(not(x), y)
in1D = 1 if ucB>=U/2;
Rp1D = in1D*Rclosed+(1-in1D)*Ropen;
Rn1D = in1D*Ropen+(1-in1D)*Rclosed;
Rp2D = y*Rclosed+(1-y)*Ropen;
Rn2D = y*Ropen+(1-y)*Rclosed;
iD = (U-ucD2-uc1D-uc2D)/Ri;
ucD2' = (iD-ucD2*(1/Rp1D+1/Rp2D))/(2*C) &0;
uc1D' = (iD-uc1D/Rn1D)/C &U/2;
uc2D' = (iD-uc2D/Rn2D)/C &U/2;
ucD = uc1D+uc2D;
```

```
47
48 //not(y)
49 Rp1C = y*Rclosed+(1-y)*Ropen;
50 Rn1C = y*Ropen+(1-y)*Rclosed;
51 iC = (U-uc1C-ucC)/Ri;
52 uc1C' = (iC-uc1C/Rp1C)/C &0;
53 ucC' = (iC-ucC*(1/Rn1C))/C &U;
54
55 //nand(x, not(y))
56 Rp1E = x*Rclosed+(1-x)*Ropen;
57 Rn1E = x*Ropen+(1-x)*Rclosed;
58 in2E = 1 if ucC>=U/2;
59 Rp2E = in2E*Rclosed+(1-in2E)*Ropen;
60 Rn2E = in2E*Ropen+(1-in2E)*Rclosed;
61 iE = (U-ucE2-uc1E-uc2E)/Ri;
62 ucE2' = (iE-ucE2*(1/Rp1E+1/Rp2E))/(2*C) &0;
63 uc1E' = (iE-uc1E/Rn1E)/C &U/2;
64 uc2E' = (iE-uc2E/Rn2E)/C &U/2;
65 ucE = uc1E+uc2E;
66
67 //nand(nand(not(x), y), nand(x, not(y)))
68 in1A = 1 if ucD>=U/2;
69 Rp1A = in1A*Rclosed+(1-in1A)*Ropen;
70 Rn1A = in1A*Ropen+(1-in1A)*Rclosed;
71 in2A = 1 if ucE>=U/2;
72 Rp2A = in2A*Rclosed+(1-in2A)*Ropen;
73 Rn2A = in2A*Ropen+(1-in2A)*Rclosed;
74 iA = (U-ucA2-uc1A-uc2A)/Ri;
75 ucA2' = (iA-ucA2*(1/Rp1A+1/Rp2A))/(2*C) &U;
76 uc1A' = (iA-uc1A/Rn1A)/C &0;
77 uc2A' = (iA-uc2A/Rn2A)/C &0;
78 ucA = uc1A+uc2A;
79 result = 1 if ucA>=U/2;
80
81 //nand(x, y)
82 Rp1G = x*Rclosed+(1-x)*Ropen;
83 Rn1G = x*Ropen+(1-x)*Rclosed;
84 Rp2G = y*Rclosed+(1-y)*Ropen;
85 Rn2G = y*Ropen+(1-y)*Rclosed;
86 iG = (U-ucG2-uc1G-uc2G)/Ri;
87 ucG2' = (iG-ucG2*(1/Rp1G+1/Rp2G))/(2*C) &0;
88 uc1G' = (iG-uc1G/Rn1G)/C &U/2;
89 uc2G' = (iG-uc2G/Rn2G)/C &U/2;
90 ucG = uc1G+uc2G;
91
92 //not(nand(x, y))
93 in1F = 1 if ucG>=U/2;
94 Rp1F = in1F*Rclosed+(1-in1F)*Ropen;
95 Rn1F = in1F*Ropen+(1-in1F)*Rclosed;
96 iF = (U-uc1F-ucF)/Ri;
97 uc1F' = (iF-uc1F/Rp1F)/C &U;
98 ucF' = (iF-ucF*(1/Rn1F))/C &0;
99 carry = 1 if ucF>=U/2;
```

G.2 Full adder

```
1 setup {
2   dt = tmax/1000;
3   tmax = 4e-7;
4   U = 3.3;
5   Ri = 0.120792;
6   Ropen = 0.601435;
7   Rclosed = 1e10;
8   C = 3.851953e-9;
9 }
```

```
10
11  graph {
12    format = pdf;
13    height = 360;
14    labels = ucA, ucL;
15    show = ucA, ucL;
16    width = 640;
17    xbase = 1e8;
18    xfmt = "%2.0l";
19    xmult = "10^{-8} [s]";
20    xmultx = 1.09;
21    xmulty = -0.09;
22    xtics = 0, tmax/8, 7*tmax/8;
23    yspace = 0.02;
24  }
25
26  x = 1;
27  y = 0;
28  c0 = 1;
29
30  //nand(x, y)
31  Rp1B = x*Rclosed+(1-x)*Ropen;
32  Rn1B = x*Ropen+(1-x)*Rclosed;
33  Rp2B = y*Rclosed+(1-y)*Ropen;
34  Rn2B = y*Ropen+(1-y)*Rclosed;
35  iB = (U-ucB2-uc1B-uc2B)/Ri;
36  ucB2' = (iB-ucB2*(1/Rp1B+1/Rp2B))/(2*C) &0;
37  uc1B' = (iB-uc1B/Rn1B)/C &U/2;
38  uc2B' = (iB-uc2B/Rn2B)/C &U/2;
39  ucB = uc1B+uc2B;
40
41  //nor(nand(x, y), c0)
42  in1C = 1 if ucB>=U/2;
43  Rp1C = in1C*Rclosed+(1-in1C)*Ropen;
44  Rn1C = in1C*Ropen+(1-in1C)*Rclosed;
45  Rp2C = c0*Rclosed+(1-c0)*Ropen;
46  Rn2C = c0*Ropen+(1-c0)*Rclosed;
47  iC = (U-uc1C-uc2C-ucC)/Ri;
48  uc1C' = (iC-uc1C/Rp1C)/C &U/2;
49  uc2C' = (iC-uc2C/Rp2C)/C &U/2;
50  ucC' = (iC-ucC*(1/Rn1C+1/Rn2C))/(2*C) &0;
51
52  //nand(y, c0)
53  Rp1D = y*Rclosed+(1-y)*Ropen;
54  Rn1D = y*Ropen+(1-y)*Rclosed;
55  Rp2D = c0*Rclosed+(1-c0)*Ropen;
56  Rn2D = c0*Ropen+(1-c0)*Rclosed;
57  iD = (U-ucD2-uc1D-uc2D)/Ri;
58  ucD2' = (iD-ucD2*(1/Rp1D+1/Rp2D))/(2*C) &0;
59  uc1D' = (iD-uc1D/Rn1D)/C &U/2;
60  uc2D' = (iD-uc2D/Rn2D)/C &U/2;
61  ucD = uc1D+uc2D;
62
63  //nor(nand(y, c0), x)
64  in1E = 1 if ucD>=U/2;
65  Rp1E = in1E*Rclosed+(1-in1E)*Ropen;
66  Rn1E = in1E*Ropen+(1-in1E)*Rclosed;
67  Rp2E = x*Rclosed+(1-x)*Ropen;
68  Rn2E = x*Ropen+(1-x)*Rclosed;
69  iE = (U-uc1E-uc2E-ucE)/Ri;
70  uc1E' = (iE-uc1E/Rp1E)/C &U/2;
71  uc2E' = (iE-uc2E/Rp2E)/C &U/2;
72  ucE' = (iE-ucE*(1/Rn1E+1/Rn2E))/(2*C) &0;
73
74  //nand(x, c0)
75  Rp1F = x*Rclosed+(1-x)*Ropen;
76  Rn1F = x*Ropen+(1-x)*Rclosed;
77  Rp2F = c0*Rclosed+(1-c0)*Ropen;
```

```
Rn2F = c0*Ropen+(1-c0)*Rclosed;
iF = (U-ucF2-uc1F-uc2F)/Ri;
ucF2' = (iF-ucF2*(1/Rp1F+1/Rp2F))/(2*C) &0;
uc1F' = (iF-uc1F/Rn1F)/C &U/2;
uc2F' = (iF-uc2F/Rn2F)/C &U/2;
ucF = uc1F+uc2F;

//nor(nand(x, c0), y)
in1G = 1 if ucF>=U/2;
Rp1G = in1G*Rclosed+(1-in1G)*Ropen;
Rn1G = in1G*Ropen+(1-in1G)*Rclosed;
Rp2G = y*Rclosed+(1-y)*Ropen;
Rn2G = y*Ropen+(1-y)*Rclosed;
iG = (U-uc1G-uc2G-ucG)/Ri;
uc1G' = (iG-uc1G/Rp1G)/C &U/2;
uc2G' = (iG-uc2G/Rp2G)/C &U/2;
ucG' = (iG-ucG*(1/Rn1G+1/Rn2G))/(2*C) &0;

//nor(x, y, c0)
Rp1H = x*Rclosed+(1-x)*Ropen;
Rn1H = x*Ropen+(1-x)*Rclosed;
Rp2H = y*Rclosed+(1-y)*Ropen;
Rn2H = y*Ropen+(1-y)*Rclosed;
Rp3H = c0*Rclosed+(1-c0)*Ropen;
Rn3H = c0*Ropen+(1-c0)*Rclosed;
iH = (U-uc1H-uc2H-uc3H-ucH)/Ri;
uc1H' = (iH-uc1H/Rp1H)/C &0;
uc2H' = (iH-uc2H/Rp2H)/C &0;
uc3H' = (iH-uc3H/Rp3H)/C &0;
ucH' = (iH-ucH*(1/Rn1H+1/Rn2H+1/Rn3H))/(3*C) &U;

//nor(nor(nand(x, y), c0), nor(nand(y, c0), x), nor(nand(x, c0), y),
//nor(x, y, c0))
in1A = 1 if ucC>=U/2;
Rp1A = in1A*Rclosed+(1-in1A)*Ropen;
Rn1A = in1A*Ropen+(1-in1A)*Rclosed;
in2A = 1 if ucE>=U/2;
Rp2A = in2A*Rclosed+(1-in2A)*Ropen;
Rn2A = in2A*Ropen+(1-in2A)*Rclosed;
in3A = 1 if ucG>=U/2;
Rp3A = in3A*Rclosed+(1-in3A)*Ropen;
Rn3A = in3A*Ropen+(1-in3A)*Rclosed;
in4A = 1 if ucH>=U/2;
Rp4A = in4A*Rclosed+(1-in4A)*Ropen;
Rn4A = in4A*Ropen+(1-in4A)*Rclosed;
iA = (U-uc1A-uc2A-uc3A-uc4A-ucA)/Ri;
uc1A' = (iA-uc1A/Rp1A)/C &U/4;
uc2A' = (iA-uc2A/Rp2A)/C &U/4;
uc3A' = (iA-uc3A/Rp3A)/C &U/4;
uc4A' = (iA-uc4A/Rp4A)/C &U/4;
ucA' = (iA-ucA*(1/Rn1A+1/Rn2A+1/Rn3A+1/Rn4A))/(4*C) &0;
result = 1 if ucA>=U/2;

//nand(x, y)
Rp1I = x*Rclosed+(1-x)*Ropen;
Rn1I = x*Ropen+(1-x)*Rclosed;
Rp2I = y*Rclosed+(1-y)*Ropen;
Rn2I = y*Ropen+(1-y)*Rclosed;
iI = (U-ucI2-uc1I-uc2I)/Ri;
ucI2' = (iI-ucI2*(1/Rp1I+1/Rp2I))/(2*C) &0;
uc1I' = (iI-uc1I/Rn1I)/C &U/2;
uc2I' = (iI-uc2I/Rn2I)/C &U/2;
ucI = uc1I+uc2I;

//nand(x, c0)
Rp1J = x*Rclosed+(1-x)*Ropen;
Rn1J = x*Ropen+(1-x)*Rclosed;
Rp2J = c0*Rclosed+(1-c0)*Ropen;
```

```
Rn2J = c0*Ropen+(1-c0)*Rclosed;
iJ = (U-ucJ2-uc1J-uc2J)/Ri;
ucJ2' = (iJ-ucJ2*(1/Rp1J+1/Rp2J))/(2*C) &0;
uc1J' = (iJ-uc1J/Rn1J)/C &U/2;
uc2J' = (iJ-uc2J/Rn2J)/C &U/2;
ucJ = uc1J+uc2J;

//nand(y, c0)
Rp1K = y*Rclosed+(1-y)*Ropen;
Rn1K = y*Ropen+(1-y)*Rclosed;
Rp2K = c0*Rclosed+(1-c0)*Ropen;
Rn2K = c0*Ropen+(1-c0)*Rclosed;
iK = (U-ucK2-uc1K-uc2K)/Ri;
ucK2' = (iK-ucK2*(1/Rp1K+1/Rp2K))/(2*C) &0;
uc1K' = (iK-uc1K/Rn1K)/C &U/2;
uc2K' = (iK-uc2K/Rn2K)/C &U/2;
ucK = uc1K+uc2K;

//nand(nand(x, y), nand(x, c0), nand(y, c0))
in1L = 1 if ucI>=U/2;
Rp1L = in1L*Rclosed+(1-in1L)*Ropen;
Rn1L = in1L*Ropen+(1-in1L)*Rclosed;
in2L = 1 if ucJ>=U/2;
Rp2L = in2L*Rclosed+(1-in2L)*Ropen;
Rn2L = in2L*Ropen+(1-in2L)*Rclosed;
in3L = 1 if ucK>=U/2;
Rp3L = in3L*Rclosed+(1-in3L)*Ropen;
Rn3L = in3L*Ropen+(1-in3L)*Rclosed;
iL = (U-ucL2-uc1L-uc2L-uc3L)/Ri;
ucL2' = (iL-ucL2*(1/Rp1L+1/Rp2L+1/Rp3L))/(3*C) &U;
uc1L' = (iL-uc1L/Rn1L)/C &0;
uc2L' = (iL-uc2L/Rn2L)/C &0;
uc3L' = (iL-uc3L/Rn3L)/C &0;
ucL = uc1L+uc2L+uc3L;
carry = 1 if ucL>=U/2;
```

Appendix H

VLSI (SPICE)

H.1 Half adder

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* x = 1
V3 3 0 PWL(0 3.3 4e-07 3.3)
* y = 1
V4 4 0 PWL(0 3.3 4e-07 3.3)
* not(x)
M5a 6 3 2 2 P3306M
M5b 6 3 0 0 N3306M
R5c 5 6 0.1
* nand(not(x), y)
M7a 8 5 2 2 P3306M
M7b 8 4 2 2 P3306M
M7c 8 5 9 9 N3306M
M7d 9 4 0 0 N3306M
R7e 7 8 0.1
* not(y)
M10a 11 4 2 2 P3306M
M10b 11 4 0 0 N3306M
R10c 10 11 0.1
* nand(x, not(y))
M12a 13 3 2 2 P3306M
M12b 13 10 2 2 P3306M
M12c 13 3 14 14 N3306M
M12d 14 10 0 0 N3306M
R12e 12 13 0.1
* result = nand(nand(not(x), y), nand(x, not(y)))
M15a 16 7 2 2 P3306M
M15b 16 12 2 2 P3306M
M15c 16 7 17 17 N3306M
M15d 17 12 0 0 N3306M
R15e 15 16 0.1
C15f 16 0 1e-12
* nand(x, y)
M18a 19 3 2 2 P3306M
M18b 19 4 2 2 P3306M
M18c 19 3 20 20 N3306M
M18d 20 4 0 0 N3306M
R18e 18 19 0.1
* carry = not(nand(x, y))
M21a 22 18 2 2 P3306M
```

```
M21b 22 18 0 0 N3306M
R21c 21 22 0.1
C21d 22 0 1e-12
.TRAN 4e-10 4e-07 0 4e-10
.PRINT TRAN V(15) V(21)
.PROBE
.END
```

H.2 Full adder

```
Circuit
.MODEL N3306M NMOS VTO=1.824 RS=1.572 RD=1.436 IS=1E-15 KP=.1233
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1
.MODEL P3306M PMOS VTO=-2.875 RS=5.227 RD=7.524 IS=1E-15 KP=.145
+ CGSO=28E-12 CGDO=3E-12 CBD=35E-12 PB=1 LAMBDA=6.67E-3
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* x = 1
V3 3 0 PWL(0 3.3 4e-07 3.3)
* y = 0
V4 4 0 PWL(0 0 4e-07 0)
* c0 = 1
V5 5 0 PWL(0 3.3 4e-07 3.3)
* nand(x, y)
M6a 7 3 2 2 P3306M
M6b 7 4 2 2 P3306M
M6c 7 3 8 8 N3306M
M6d 8 4 0 0 N3306M
R6e 6 7 0.1
* nor(nand(x, y), c0)
M9a 10 6 2 2 P3306M
M9b 11 5 10 10 P3306M
M9c 11 6 0 0 N3306M
M9d 11 5 0 0 N3306M
R9e 9 11 0.1
* nand(y, c0)
M12a 13 4 2 2 P3306M
M12b 13 5 2 2 P3306M
M12c 13 4 14 14 N3306M
M12d 14 5 0 0 N3306M
R12e 12 13 0.1
* nor(nand(y, c0), x)
M15a 16 12 2 2 P3306M
M15b 17 3 16 16 P3306M
M15c 17 12 0 0 N3306M
M15d 17 3 0 0 N3306M
R15e 15 17 0.1
* nand(x, c0)
M18a 19 3 2 2 P3306M
M18b 19 5 2 2 P3306M
M18c 19 3 20 20 N3306M
M18d 20 5 0 0 N3306M
R18e 18 19 0.1
* nor(nand(x, c0), y)
M21a 22 18 2 2 P3306M
M21b 23 4 22 22 P3306M
M21c 23 18 0 0 N3306M
M21d 23 4 0 0 N3306M
R21e 21 23 0.1
* nor(x, y, c0)
M24a 25 3 2 2 P3306M
M24b 26 4 25 25 P3306M
M24c 27 5 26 26 P3306M
M24d 27 3 0 0 N3306M
M24e 27 4 0 0 N3306M
```

```
M24f 27 5 0 0 N3306M
R24g 24 27 0.1
* result = nor(nor(nand(x, y), c0), nor(nand(y, c0), x), nor(nand(x, c0),
* y), nor(x, y, c0))
M28a 29 9 2 2 P3306M
M28b 30 15 29 29 P3306M
M28c 31 21 30 30 P3306M
M28d 32 24 31 31 P3306M
M28e 32 9 0 0 N3306M
M28f 32 15 0 0 N3306M
M28g 32 21 0 0 N3306M
M28h 32 24 0 0 N3306M
R28i 28 32 0.1
C28j 32 0 1e-12
* nand(x, y)
M33a 34 3 2 2 P3306M
M33b 34 4 2 2 P3306M
M33c 34 3 35 35 N3306M
M33d 35 4 0 0 N3306M
R33e 33 34 0.1
* nand(x, c0)
M36a 37 3 2 2 P3306M
M36b 37 5 2 2 P3306M
M36c 37 3 38 38 N3306M
M36d 38 5 0 0 N3306M
R36e 36 37 0.1
* nand(y, c0)
M39a 40 4 2 2 P3306M
M39b 40 5 2 2 P3306M
M39c 40 4 41 41 N3306M
M39d 41 5 0 0 N3306M
R39e 39 40 0.1
* carry = nand(nand(x, y), nand(x, c0), nand(y, c0))
M42a 43 33 2 2 P3306M
M42b 43 36 2 2 P3306M
M42c 43 39 2 2 P3306M
M42d 43 33 44 44 N3306M
M42e 44 36 45 45 N3306M
M42f 45 39 0 0 N3306M
R42g 42 43 0.1
C42h 43 0 1e-12
.TRAN 4e-10 4e-07 0 4e-10
.PRINT TRAN V(28) V(42)
.PROBE
.END
```

Index

V