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 — Ing. FILIP KOCINA
AUTOR PRÁCE

SUPERVISOR — doc. Ing. JIŘÍ KUNOVSKÝ, CSc.
ŠKOLITEL

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.

Index

[15] 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.

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

[17] 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.

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

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

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

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

Contents

Bibliography

[1] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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/`

Nomenclature

$\boldsymbol{y'}$	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
ULSI	Ultra Large-Scale Integration
VLSI	Very Large-Scale Integration

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/`

proximation can model the lengths of the transient responses. The Capacitor Substitution Method (CSM) is described in Section 3.2.

– *The proposed method should be efficient.*
This hypothesis is clearly answered in Chapter 4. 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.

5.2 Research contribution

During my doctoral studies, I developed programming equipment for solving systems of ordinary differential equations [16] 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 3.2) and XOR (Section 3.2.4), further latches and flip-flops and finally an adder (Section 4.1) and a multiplier (Section 4.2).

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 4.1.6.

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 3.2. 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 4.2 and the results of the experiments are shown in Section 4.2.4.

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.

5.3 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.

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

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.

Chapter 5

Conclusion

In this thesis, I discussed the simulation of 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 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, an n-bit D shift register, a complementing circuit, a T flip-flop 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 is unusable for this purpose. The results of the experiments are presented in Section 4.1.6 and Section 4.2.4.

5.1 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 3. The basic CMOS logic gates are analyzed and modeled. The generic algorithm is introduced in Section 4.3.
- *The transistors could be replaced by RC circuits.*
  Yes, the operation of the transistors can be approximated by RC circuits. This ap-

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

4.3.1 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[9] NOR: x, y, z are the inputs; the equations are generated consecutively and the current for three inputs is:

$$i \;=\; \frac{1}{R_i} \cdot \Big(U - u_{C_1} - u_{C_2} - u_{C_3} - u_{C_{456}}\Big), \tag{4.20}$$

the three ODEs for the three inputs:

$$\begin{aligned} u'_{C_1} &= \frac{1}{C_1} \cdot \left(i - \frac{1}{R_{\overline{x}}} \cdot u_{C_1}\right), \quad u_{C_1}(0) \;=\; 0 \\ u'_{C_2} &= \frac{1}{C_2} \cdot \left(i - \frac{1}{R_{\overline{y}}} \cdot u_{C_2}\right), \quad u_{C_2}(0) \;=\; 0 \\ u'_{C_3} &= \frac{1}{C_3} \cdot \left(i - \frac{1}{R_{\overline{z}}} \cdot u_{C_3}\right), \quad u_{C_3}(0) \;=\; 0 \end{aligned} \tag{4.21}$$

and the ODE for the output:

$$u'_{C_{456}} \;=\; \frac{1}{C_{456}} \cdot \left(i - \frac{R_x \cdot R_y + R_x \cdot R_z + R_y \cdot R_z}{R_x \cdot R_y \cdot R_z} \cdot u_{C_{456}}\right), \quad u_{C_{456}}(0) \;=\; 3.3. \tag{4.22}$$

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

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

Chapter 2

Differential–Algebraic Equations

A large number of technical problems can be described using a system of ordinary differential and algebraic equations [10, 11]. These systems can be formally written as

$$\begin{aligned} w'_1 &= f_1(w_1, \ldots, w_n, x_1, \ldots, x_m), \quad w_1(t_0) \;=\; w_1^0 \\ &\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ w'_n &= f_n(w_1, \ldots, w_n, x_1, \ldots, x_m), \quad w_n(t_0) \;=\; w_n^0 \\ x_1 &= g_1(w_1, \ldots, w_n, x_1, \ldots, x_m) \\ &\vdots \\ x_m &= g_m(w_1, \ldots, w_n, x_1, \ldots, x_m) \end{aligned} \tag{2.1}$$

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 [6, 9].

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

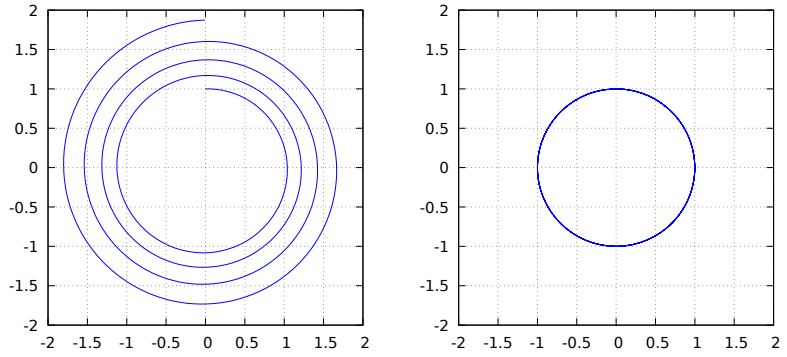
The simplest numerical method for solving ordinary differential equations is the explicit Euler method. It is a special form of the explicit Taylor method of the first order:

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

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]. Some drawbacks of the explicit Euler method can be removed by the implicit form of the Euler method [15].



(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 [2, 3]. Step size h_n can be variable, but this is rare. The following fourth-order method (2.7) appears to be the most frequent (with constant step size h),

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

Table 4.9 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 4.9: Parallel simulation

The comparison of CSM and SPICE is shown in Table 4.10. 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 4.10: Acceleration of CSM compared to SPICE

4.3 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.

$$\mathbf{A} = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&0&0&0&0&0&0&0 \\ 0&1&1&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&1&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&1&0&0&0 \\ 1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&1&0&1&1&0 \end{pmatrix} \quad \mathbf{u} = \begin{pmatrix} x \\ y \\ z \\ \uparrow \\ \downarrow \\ \uparrow \\ \downarrow \\ \uparrow \\ \downarrow \\ \downarrow \\ \downarrow \end{pmatrix} \tag{4.19}$$

The lines of matrix $\mathbf{A}$ correspond with the lines of vector $\mathbf{u}$. To obtain the inputs for an operation, matrix–vector multiplication is performed.

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 4.6, is the two's complement of

$$-0.10000110101011101101100011011101_b$$

that corresponds with the correct result of (4.18).

4.2.4 Experiments

Table 4.7 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 [14].

# 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 4.7: Booth's multiplier – parameters

The results of serial simulation are given in Table 4.8. The last three lines of the SPICE results are incomplete since SPICE runs longer than a day[8].

	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 4.8: Serial simulation

[8] 1 day = 86 400 s

see [4, 20].

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

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) [1, 17]. 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.

2.2 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).

2.2.1 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.

$$u = (x + 2y) \cdot (x - 2y) \tag{2.9}$$

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

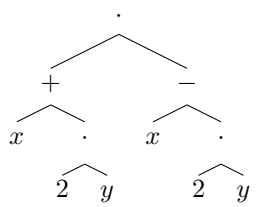


Figure 2.2: Syntax tree

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):

$$(x+2y)\cdot(x-2y) \quad \rightarrow \quad x\cdot x + x\cdot(-2y) + 2y\cdot x + 2y\cdot(-2y). \tag{2.10}$$

Now the expression can be rearranged (using the commutative law) and variables can be merged (using the associative law and expressing multiplications as exponentiations)[2]:

$$x\cdot x + x\cdot(-2y) + 2y\cdot x + 2y\cdot(-2y) \quad \rightarrow \quad x^2 - 4\cdot y^2. \tag{2.11}$$

2.2.2 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 (2.12).

$$y \;=\; \sum_{i=1}^{m} a_i \prod_{j=1}^{n_i} v_{ij}^{r_{ij}}, \quad m, n_i \;\in\; \mathbb{N}_0,\; r_{ij} \;\in\; \mathbb{N},\; a_i \;\in\; \mathbb{R}\setminus\{0\},\; v_{ij} \;\in\; Var \tag{2.12}$$

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

4.2.3 Verification

First, the correct operation of the multiplier was verified. The multiplication (4.18) was performed (the numbers were generated randomly).

$$-0.10011010000101111011101011001 00_b \cdot 0.11011111110000010000000100001011_b \tag{4.18}$$

$-0.10011010000101111011101011001 00_b$ is $10110010111101000010001010011100_b$ in two's complement. The partial results of the algorithm are shown in Table 4.6.

#	Extended multiplicand	Result
1	INIT	00000000000000000000000000000000
2	10110010111101000010001010011100**00**	00000000000000000000000000000000
3	11011001011110100001000101001110**00**	00000000000000000000000000000000
4	11101100101111010000100010100111**10**	11001000000011111011111101111010
5	11110110010111101000010001010011**11**	11100100000001111101111110111101
6	11111011001011110100001000101010**11**	11110010000000111110111111011110
7	11111101100101111010000100010101**01**	00110000111100100011100001110100
8	11111110110010111101000010001010**00**	00011000011110010001110000111010
9	11111111011001011110100001000101**10**	11010100010011000100110110010111
10	11111111101100101111010000100010**01**	00100010000101100110011101010001
11	11111111110110010111101000010001**10**	11011001000110101111001100100011
12	11111111111011001011110100001000**01**	00100100011111010111010000010111
13	11111111111101100101111010000100**00**	00010010001111101101110100001011
14	11111111111110110010111101000010**00**	00001001000111110110111010000101
15	11111111111111011001011110100001**10**	11001100100111110111011010111101
16	11111111111111101100101111010000**01**	00011110001111111111101111100100
17	11111111111111110110010111101000**00**	00001111000111111111110111110010
18	11111111111111111011001011110100**00**	00000111000111111111111011111001
19	11111111111111111101100101111010**00**	00000011110001111111111101111100
20	11111111111111111110110010111101**10**	11001001111100111011111100111000
21	11111111111111111111011001011101**01**	00011100111010100010000000100001
22	11111111111111111111101100101110**10**	11010110100001001100111110001011
23	11111111111111111111110110010111**01**	11101011010000100110011111000101
24	11111111111111111111111011001010**11**	11110101101000010011001111100010
25	11111111111111111111111101100101**11**	11111010110100001001100111110001
26	11111111111111111111111111011001**01**	00110101010110001000110101111110
27	11111111111111111111111111101100**10**	11100010101111000000011000111001
28	11111111111111111111111111110110**01**	00101001010011100100001110100010
29	11111111111111111111111111111011**00**	00010100101001110010000111010001
30	11111111111111111111111111111101**10**	11010010011000110101000001100011
31	11111111111111111111111111111110**11**	11101001001100011010100000110001
32	11111111111111111111111111111111**01**	00101100100010010001010010011110
33	11111111111111111111111111111111**10**	10111100101010001001001110010011

Table 4.6: 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

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 (4.16).

$$\begin{aligned} Q &= \overline{Q} \uparrow \left(\overline{RES} \uparrow T \uparrow \overline{q} \uparrow CLK \uparrow En\right) \\ \overline{Q} &= \overline{RES} \uparrow Q \uparrow (T \uparrow q \uparrow CLK \uparrow En) \end{aligned} \tag{4.16}$$

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

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 (4.17).

$$r_i = (\bar{c} \uparrow x_i) \uparrow (c \uparrow y_i) \tag{4.17}$$

4.2.2 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.

Chapter 3

Solving Electronic Circuits

This chapter focuses on electronic circuits. These circuits contain not only resistors, capacitors and coils, but also semiconductors.

3.1 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.

3.1.1 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 [7] for more information.

3.1.2 FOS

VLSI circuits were initially simulated in Fast ODE Solver (FOS) [16], 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 (3.1) 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.

3.2 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.

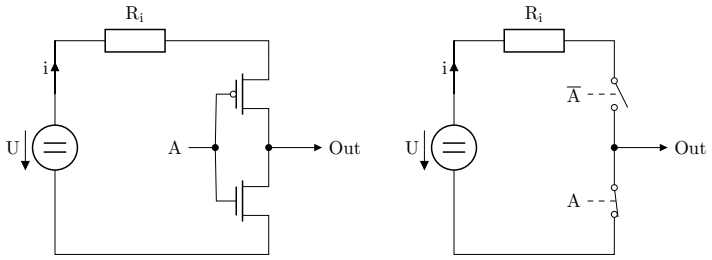
The general purpose transistors N3306M and P3306M were chosen for simulation. The corresponding SPICE models of these transistors follow[1].

```
.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
```

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

3.2.1 CMOS inverter

Figure 3.1a 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 3.1b. 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 3.1: Inverter [21]

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

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[6] 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 [19]. 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.[7]

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 (4.11).

$$\begin{aligned} c_1 &= c_0 \vee x_0 \\ c_2 &= c_1 \vee x_1 \\ c_3 &= c_2 \vee x_2 \\ c_4 &= c_3 \vee x_3 \end{aligned} \tag{4.11}$$

After substituting c_i, (4.12) is derived.

$$\begin{aligned} c_1 &= c_0 \vee x_0 \\ c_2 &= c_0 \vee x_0 \vee x_1 \\ c_3 &= c_0 \vee x_0 \vee x_1 \vee x_2 \\ c_4 &= c_0 \vee x_0 \vee x_1 \vee x_2 \vee x_3 \end{aligned} \tag{4.12}$$

By transforming (4.12) to use only basic CMOS gates, (4.13) is obtained.

$$\begin{aligned} c_1 &= \overline{c_0 \downarrow x_0} \\ c_2 &= \overline{c_0 \downarrow x_0 \downarrow x_1} \\ c_3 &= \overline{c_0 \downarrow x_0 \downarrow x_1 \downarrow x_2} \\ c_4 &= \overline{c_0 \downarrow x_0 \downarrow x_1 \downarrow x_2 \downarrow x_3} \end{aligned} \tag{4.13}$$

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

$$G = x_1 \vee x_2 \vee x_3 \vee x_4 \tag{4.14}$$

or by (4.15) using the basic logic gates.

$$G = \overline{x_1 \downarrow x_2 \downarrow x_3 \downarrow x_4} \tag{4.15}$$

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).

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

[7]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.

# 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 4.5: 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.

4.2 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.

4.2.1 Booth's algorithm

Booth's algorithm is fast and uses only the operations addition and bit shift. The principle is discussed in [8].

D shift register

The multiplier stores numbers in D shift registers [12]. 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 [18]). The function of an n-bit D shift register is described by (4.10).

$$\begin{aligned} S_i &= x_i \uparrow En \uparrow Wr \\ R_i &= S_i \uparrow En \uparrow Wr \end{aligned}$$

$$\begin{aligned} D_i &= \overline{RES} \uparrow q_{i+1} \uparrow Sh \uparrow CLK \\ Q_i &= S_i \uparrow D_i \uparrow \overline{Q}_i \\ \overline{Q}_i &= \overline{RES} \uparrow R_i \uparrow Q_i \uparrow (D_i \uparrow Sh \uparrow CLK) \end{aligned} \tag{4.10}$$

$$\begin{aligned} d_i &= \overline{RES} \uparrow Q_i \uparrow Sh \uparrow \overline{CLK} \\ q_i &= S_i \uparrow d_i \uparrow \overline{q}_i \\ \overline{q}_i &= \overline{RES} \uparrow R_i \uparrow q_i \uparrow \left(d_i \uparrow Sh \uparrow \overline{CLK}\right) \end{aligned}$$

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

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).

```
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 3.2 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.

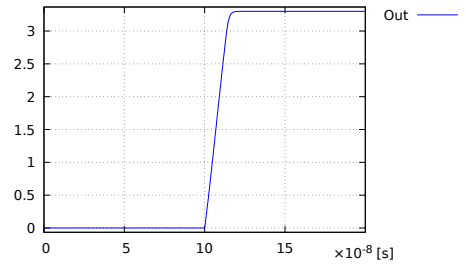


Figure 3.2: Inverter – SPICE [V]

The curve in Figure 3.2 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 3.3 is performed.

[2]The value depends on the logic used.

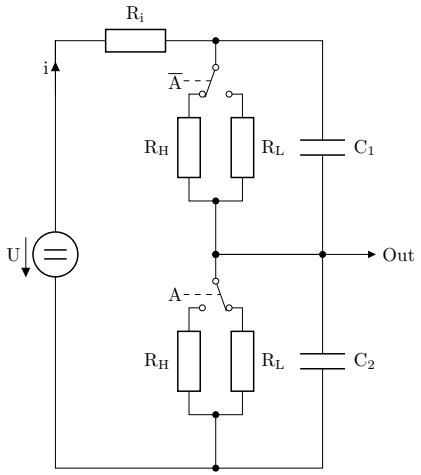


Figure 3.3: Inverter – substituted by CSM

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 (3.1) 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.

$$\begin{aligned} i &= \frac{1}{R_i} \cdot \left(U - u_{C_1} - u_{C_2}\right) \\ u'_{C_1} &= \frac{1}{C_1} \cdot \left(i - \frac{1}{R_{\overline{A}}} \cdot u_{C_1}\right), \quad u_{C_1}(0) = 3.3 \\ u'_{C_2} &= \frac{1}{C_2} \cdot \left(i - \frac{1}{R_A} \cdot u_{C_2}\right), \quad u_{C_2}(0) = 0 \end{aligned} \tag{3.1}$$

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

$$\begin{aligned} \boldsymbol{x}' &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \\ \boldsymbol{y} &= \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u} \end{aligned} \tag{3.2}$$

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 (3.3) describe the transient response of the inverter when toggling the output

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

The results of the serial simulation of CLA adders are shown in Table 4.3 (the SPICE simulations running longer than a day[5] 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 4.3: Serial simulation

The results of the parallel simulation are shown in Table 4.4. 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 4.4: Parallel simulation

Table 4.5 shows the acceleration of the serial and parallel computations achieved by CSM compared to SPICE.

[5]1 day = 86 400 s

4.1.5 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 [14], 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 [5] even define a fifth category: Ultra Large-Scale Integration (ULSI). However, ULSI is commonly included in VLSI by other authors.

4.1.6 Experiments

The experiments were performed on our research server[3]. All simulation times were chosen carefully to attain a final steady state. The experiments were performed using SPICE[4] and CSM; specialized software was developed for the VLSI simulation (see Section 3.1.2).

Table 4.2 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 4.2: CLA adder – parameters

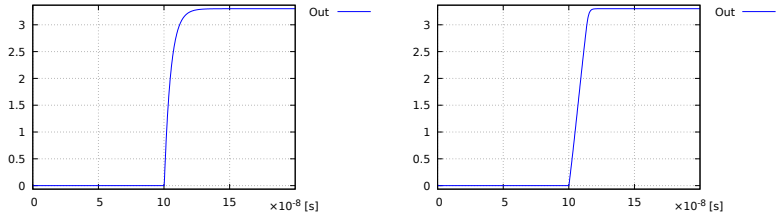
The interesting thing is that the number of ordinary differential equations is smaller than the number of transistors. This is caused by merging parallel capacitors.

[3] 2× Intel Xeon E5-2630v2 (2.6 GHz, 6/12-core, 15 MB cache), 32 GB RAM
[4] NGSpice v26.1 with default settings

from logical zero to logical one.

$$\boldsymbol{A} = \begin{pmatrix} -\frac{R_L+R_i}{C_1 R_L R_i} & -\frac{1}{C_1 R_i} \\ -\frac{1}{C_2 R_i} & -\frac{R_H+R_i}{C_2 R_H R_i} \end{pmatrix} \quad \boldsymbol{B} = \begin{pmatrix} \frac{U}{C_1 R_i} \\ \frac{U}{C_2 R_i} \end{pmatrix}$$
$$\boldsymbol{C} = \begin{pmatrix} 0 & 1 \end{pmatrix} \quad \boldsymbol{D} = 0 \qquad (3.3)$$
$$\boldsymbol{x} = \begin{pmatrix} u_{C_1} \\ u_{C_2} \end{pmatrix} \quad \boldsymbol{u} = 1$$

Figure 3.4a shows the solution of (3.1) 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 3.4b (solution by SPICE) is included for comparison.



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

Figure 3.4: Inverter – solution

The error of the approximation is shown in Figure 3.5. 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.

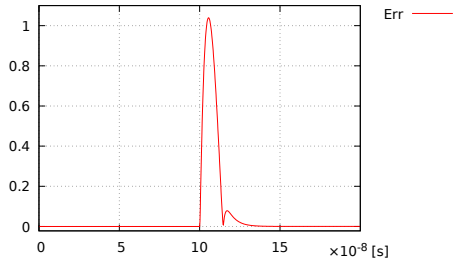
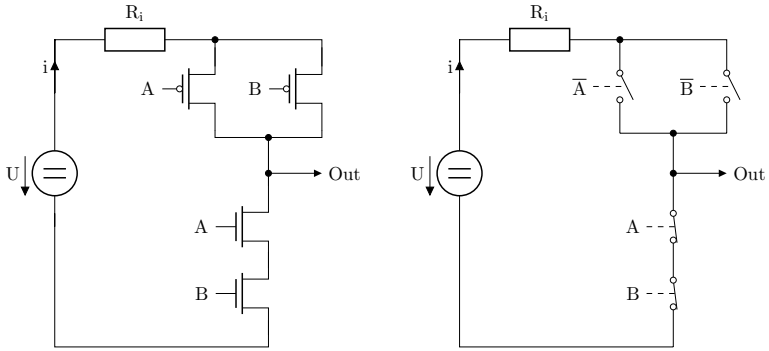


Figure 3.5: Inverter – approximation error [V]

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.

3.2.2 CMOS NAND

The scheme of CMOS NAND is shown in Figure 3.6a and the function of this electronic circuit is explained in Figure 3.6b. 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 3.6: NAND [21]

The NAND logical inputs are given in Table 3.1. 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 3.1: NAND – input

CMOS NAND can be solved using the following SPICE netlist (abbreviated form):

```
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 3.7.

using NAND and NOR gates and inverted; the propagation delay of c_i remains 2 logic gates as it is calculated by (4.7) – using De Morgan's laws.

$$\begin{aligned} c_1 &= \overline{g_0} \uparrow (p_0 \uparrow c_0) \\ c_2 &= \overline{g_1} \uparrow (p_1 \uparrow g_0) \uparrow (p_1 \uparrow p_0 \uparrow c_0) \\ c_3 &= \overline{g_2} \uparrow (p_2 \uparrow g_1) \uparrow (p_2 \uparrow p_1 \uparrow g_0) \uparrow (p_2 \uparrow p_1 \uparrow p_0 \uparrow c_0) \\ c_4 &= \overline{g_3} \uparrow (p_3 \uparrow g_2) \uparrow (p_3 \uparrow p_2 \uparrow g_1) \uparrow (p_3 \uparrow p_2 \uparrow p_1 \uparrow g_0) \\ &\quad \uparrow (p_3 \uparrow p_2 \uparrow p_1 \uparrow p_0 \uparrow c_0) \end{aligned} \tag{4.7}$$

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 (4.8).

$$\begin{aligned} G &= g_3 \vee (p_3 \wedge g_2) \vee (p_3 \wedge p_2 \wedge g_1) \vee (p_3 \wedge p_2 \wedge p_1 \wedge g_0) \\ P &= p_3 \wedge p_2 \wedge p_1 \wedge p_0 \end{aligned} \tag{4.8}$$

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

$$\begin{aligned} G &= \overline{g_3} \uparrow (p_3 \uparrow g_2) \uparrow (p_3 \uparrow p_2 \uparrow g_1) \uparrow (p_3 \uparrow p_2 \uparrow p_1 \uparrow g_0) \\ P &= \overline{p_3 \uparrow p_2 \uparrow p_1 \uparrow p_0} \end{aligned} \tag{4.9}$$

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

Table 4.1 shows that the length of the transient response of a 64-bit adder is considerably shorter. For more information, see [19].

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	1111111111111111111111111111111111111111111111101000100000000000
2.0	1111111111111101111111111111101111111111101100000000000000000000
2.2	1111111111101100111111101110100011101110110000000000000000000000
2.4	1110111011000000111011001000000011101000100000000000000000000000
2.6	1100100000000000110000000000000010000000000000000000000000000000
2.8	1000000000000000000000000000000000000000000000000000000000000000
3.0	0000000000000000000000000000000000000000000000000000000000000000

Table 4.1: CLA adder – transient response

propagates slowly through all 1-bit adders, resulting in the 16-bit adder carry. The result overflows – denoted by square brackets, see (4.3).

$$1111111111111111_b + 1 \;=\; [1]0000000000000000_b \tag{4.3}$$

The situation is shown in Figure 4.1. All bits of the 16-bit adder are set to zero after the corresponding transient response.

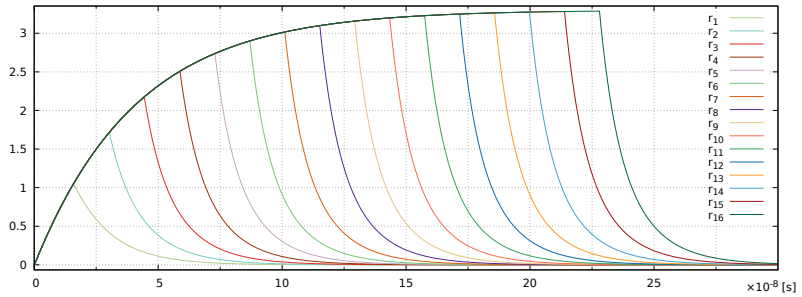


Figure 4.1: Carry propagation [V]

4.1.4 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 [19]. It uses values g_i (generate) and p_i (propagate), calculated by (4.4).

$$\begin{aligned} g_i &= a_i \wedge b_i \\ p_i &= a_i \vee b_i \end{aligned} \tag{4.4}$$

Particular carries can be calculated using (4.5).

$$\begin{aligned} c_1 &= g_0 \vee (p_0 \wedge c_0) \\ c_2 &= g_1 \vee (p_1 \wedge c_1) \\ c_3 &= g_2 \vee (p_2 \wedge c_2) \\ c_4 &= g_3 \vee (p_3 \wedge c_3) \end{aligned} \tag{4.5}$$

After substituting carries c_1, c_2 and c_3, (4.6) is obtained.

$$\begin{aligned} c_1 &= g_0 \vee (p_0 \wedge c_0) \\ c_2 &= g_1 \vee (p_1 \wedge g_0) \vee (p_1 \wedge p_0 \wedge c_0) \\ c_3 &= g_2 \vee (p_2 \wedge g_1) \vee (p_2 \wedge p_1 \wedge g_0) \vee (p_2 \wedge p_1 \wedge p_0 \wedge c_0) \\ c_4 &= g_3 \vee (p_3 \wedge g_2) \vee (p_3 \wedge p_2 \wedge g_1) \vee (p_3 \wedge p_2 \wedge p_1 \wedge g_0) \\ &\quad \vee (p_3 \wedge p_2 \wedge p_1 \wedge p_0 \wedge c_0) \end{aligned} \tag{4.6}$$

The propagation delay of the CLA is 3 logic gates if AND and OR are used. In case of NAND and NOR gates[2], the propagation delay is 4 logic gates – g_i and p_i are calculated

[2]NANDs and NORs are commonly used in electronic circuits (each gate consists of four transistors, see Figure 3.6a and 3.11a).

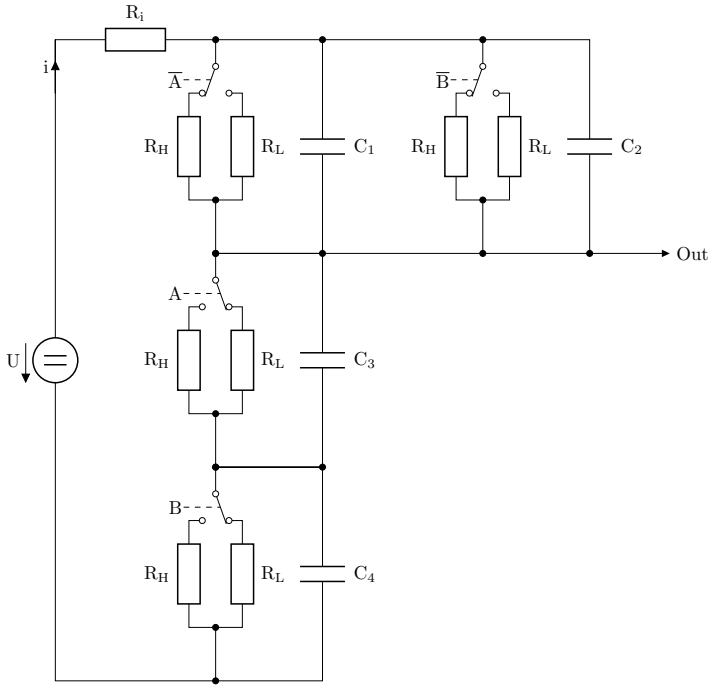


Figure 3.7: NAND – substituted by CSM

Capacitors C_1 and C_2 are parallel; therefore, they can be merged into one capacitor $C_{12} = C_1 + C_2$; see Figure 3.8.

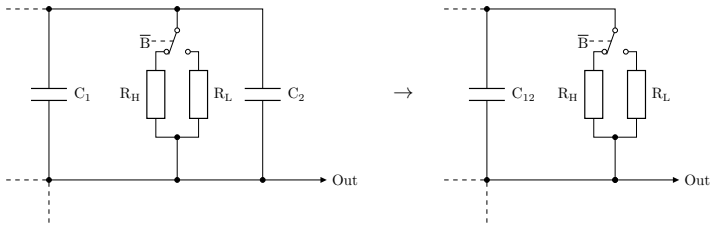
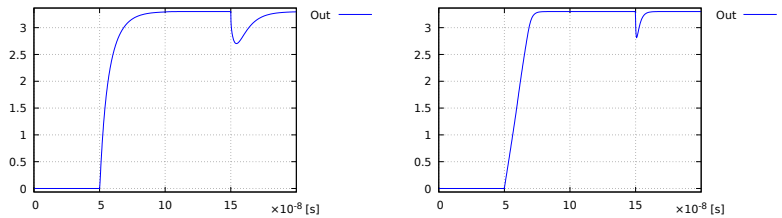


Figure 3.8: NAND – merging capacitors

The circuit in Figure 3.7 is described by (3.4)[4]; capacitor C_{12} is precharged to attain the initial value of zero.

$$\begin{aligned} i &= \frac{1}{R_i} \cdot \left(U - u_{C_{12}} - u_{C_3} - u_{C_4}\right) \\ u'_{C_{12}} &= \frac{1}{C_{12}} \cdot \left(i - \frac{R_{\overline{A}} + R_{\overline{B}}}{R_{\overline{A}} \cdot R_{\overline{B}}} \cdot u_{C_{12}}\right), \quad u_{C_{12}}(0) = 3.3 \\ u'_{C_3} &= \frac{1}{C_3} \cdot \left(i - \frac{1}{R_A} \cdot u_{C_3}\right), \qquad u_{C_3}(0) = 0 \\ u'_{C_4} &= \frac{1}{C_4} \cdot \left(i - \frac{1}{R_B} \cdot u_{C_4}\right), \qquad u_{C_4}(0) = 0 \end{aligned} \tag{3.4}$$

The solution for the values from Table 3.1 is shown in Figure 3.9. 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 3.9: NAND – solution

The error of the approximation is shown in Figure 3.10.

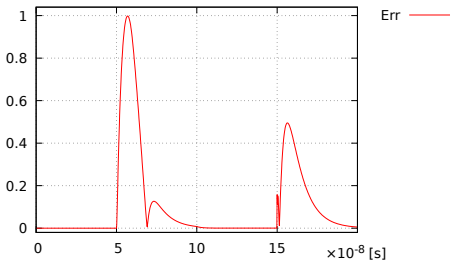


Figure 3.10: NAND – approximation error [V]

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

Chapter 4

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 [14].

4.1 Adder

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

4.1.1 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 (4.1).

$$\begin{aligned} output &= x \oplus y \\ carry &= x \wedge y = \overline{x \uparrow y} \end{aligned} \tag{4.1}$$

4.1.2 Full adder

The full adder has three inputs – two summands and an input carry. The expressions in (4.2) are used for calculating the output and the carry.

$$\begin{aligned} output &= x \oplus y \oplus c_0 \\ carry &= (x \wedge y) \vee (x \wedge c_0) \vee (y \wedge c_0) \\ &= (x \uparrow y) \uparrow (x \uparrow c_0) \uparrow (y \uparrow c_0) \end{aligned} \tag{4.2}$$

4.1.3 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

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

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 3.3: 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 (3.8) describes the three-input XOR – derived from the Conjunctive Normal Form (CNF).

$$x \oplus y \oplus z \;=\; \overline{((x \wedge y) \wedge \overline{z})} \wedge \overline{((y \wedge z) \wedge \overline{x})} \wedge \overline{((x \wedge z) \wedge \overline{y})} \wedge \overline{(\overline{x} \wedge \overline{y} \wedge \overline{z})} \tag{3.8}$$

Equation (3.9) is again obtained from (3.8) using De Morgan's laws. The NOR operator is known as Peirce's arrow (denoted as $\downarrow$) [13]. Note that both $\uparrow$ and $\downarrow$ are 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.

$$x \oplus y \oplus z \;=\; ((x \uparrow y) \downarrow z) \downarrow ((y \uparrow z) \downarrow x) \downarrow ((x \uparrow z) \downarrow y) \downarrow (x \downarrow y \downarrow z) \tag{3.9}$$

Table 3.4 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

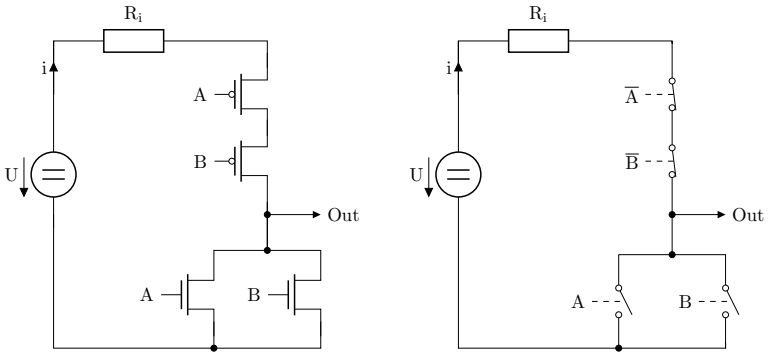
Table 3.4: XOR with three inputs

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

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

3.2.3 CMOS NOR

The scheme of CMOS NOR is shown in Figure 3.11a and the function of this electronic circuit is explained in Figure 3.11b. 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 3.11: NOR [21]

The NOR logical inputs are given in Table 3.2. The time domain is split into equally long segments.

A	0	1	1	0
B	0	0	1	1

Table 3.2: NOR – input

The abbreviated SPICE netlist of CMOS NOR follows.

```
Vdd 1 0 DC 3.3
Ri 2 1 0.1
* A = 0, 1, 1, 0
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)
* B = 0, 0, 1, 1
V4 4 0 PWL(0 0 1e-07 0 1e-07 3.3 2e-07 3.3)
* nor(A, B)
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 1p
```

The transformation of CMOS NOR is shown in Figure 3.12. Capacitors C_3 and C_4 are parallel; therefore, one capacitor $C_{34} = C_3 + C_4$ is used in equations.

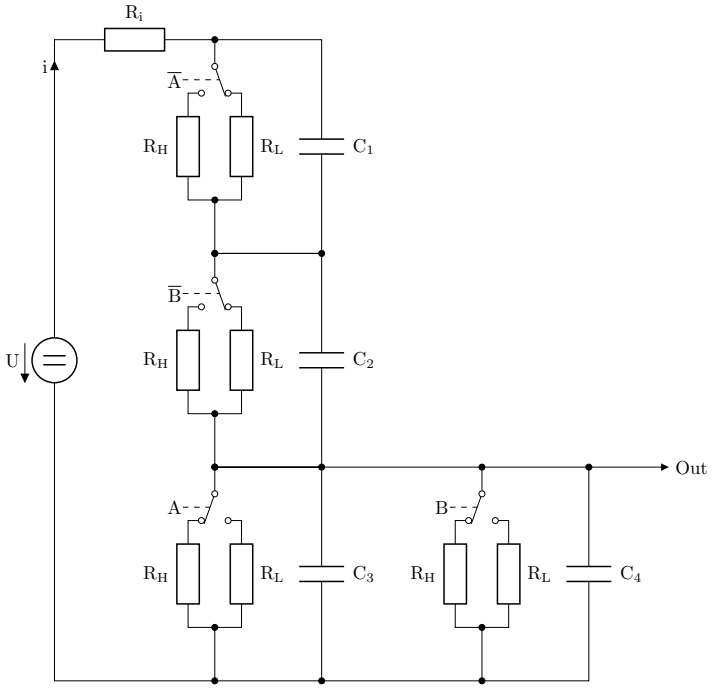


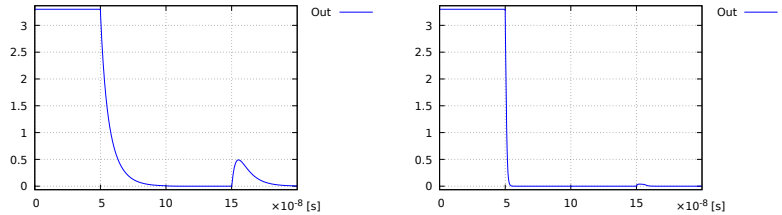
Figure 3.12: NOR – substituted by CSM

The circuit in Figure 3.12 is described by equations (3.5)[5]; capacitor C_{34} (merged from C_3 and C_4) is precharged.

$$\begin{aligned} i &= \frac{1}{R_i} \cdot \left(U - u_{C_1} - u_{C_2} - u_{C_{34}}\right) \\ u'_{C_1} &= \frac{1}{C_1} \cdot \left(i - \frac{1}{R_{\overline{A}}} \cdot u_{C_1}\right), \qquad u_{C_1}(0) = 0 \\ u'_{C_2} &= \frac{1}{C_2} \cdot \left(i - \frac{1}{R_{\overline{B}}} \cdot u_{C_2}\right), \qquad u_{C_2}(0) = 0 \\ u'_{C_{34}} &= \frac{1}{C_{34}} \cdot \left(i - \frac{R_A + R_B}{R_A \cdot R_B} \cdot u_{C_{34}}\right), \quad u_{C_{34}}(0) = 3.3 \end{aligned} \tag{3.5}$$

The solution for the values from Table 3.2 is shown in Figure 3.13. 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 (3.4), capacity $C_{34} = C_3 + C_4$.



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

Figure 3.13: NOR – solution

The error of the approximation is shown in Figure 3.14. The approximation error during the transient response now exceeds even 2 V; but as already stated, it is only a minor problem.

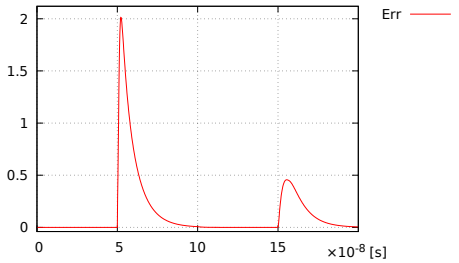


Figure 3.14: NOR – approximation error [V]

3.2.4 XOR

XOR can be constructed using the basic CMOS logic gates described above (inverters, NANDs and NORs). The XOR is used in adders. Equation (3.6) is in the Disjunctive Normal Form (DNF).

$$x \oplus y = (\overline{x} \wedge y) \vee (x \wedge \overline{y}) \tag{3.6}$$

Logic gates AND and OR are compound gates (additional inverters are required), but if De Morgan's laws are used, the expression in (3.7) is acquired that uses the simplest logic gates which can be constructed in electronics. The Sheffer stroke (denoted as $\uparrow$) [13] is a logical operation equivalent to the negated conjunction operation, NAND.

$$x \oplus y = (\overline{x} \uparrow y) \uparrow (x \uparrow \overline{y}) \tag{3.7}$$

Table 3.3 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.