Institut für Industriemathematik
Universität Paderborn

Signal-Flow Based Circuit Simulation

Von der Fakultät für Elektrotechnik,
Informatik und Mathematik
der Universität Paderborn
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
– Dr. rer. nat. –
genehmigte Dissertation

von
Stefan Klus

Gutachter: Prof. Dr. Michael Dellnitz
Prof. Dr. Andrea Walther
Prof. Dr. Matthew West

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Dr. Michael Dellnitz for the guidance, support, and encouragement as well as for the freedom to choose my own approach to this interesting field of research. I also thank Prof. Dr. Andrea Walther and Prof. Dr. Matthew West for serving as a reviewer.

Moreover, I am indebted to the circuit simulator development team of our industrial partner for the smooth collaboration and the pleasant research visit.

I would like to express my thanks to Dr. Robert Preis, Stefan Sertl, Jun.-Prof. Dr. Sina Ober-Blöbaum, and Anna-Lena Meyer for many interesting and fruitful discussions and helpful comments. Thanks also to all other members and former members of the Institute for Industrial Mathematics and the Chair of Applied Mathematics, in particular Dr. Roberto Castelli, Kathrin Flaßkamp, Sebastian Hage-Packhäuser, Dr. Mirko Hessel-von Molo, Christian Horenkamp, Dr. Giorgio Mingotti, Dr. Marcus Post, Mariusz Slonina, Bianca Thiere, Robert Timmermann, Katrin Witting, and Anna Zanzottera.

Finally, I would like to thank my family and my friends.

Abstract

The simulation of integrated circuits enables the verification of the correct functioning and provides important performance values prior to their fabrication. In particular the accurate but time-consuming circuit-level simulation plays a central role in the design process. Due to the ever increasing complexity of integrated circuits and the associated rise in computing time, there is a continuing need in efficient numerical methods for the solution of the resulting high-dimensional systems of differential and algebraic equations.

The standard approach to solve these systems can be split into two main steps: the computation of consistent initial conditions and the numerical integration with implicit one-step or multi-step methods. In this thesis, we develop different models of the signal flow of integrated circuits and propose graph-based methods to speed up the simulation exploiting information on the underlying network structure.

The determination of consistent initial values necessitates the solution of a system of nonlinear equations. In order to improve the convergence of the Newton–Raphson method, which is usually used to solve the system of equations, and thus to reduce the simulation time, we compute an appropriate starting point using an event-driven switch-level algorithm in combination with a model of the logic signal flow that is based on the partitioning into channel-connected and strongly connected components.

Another possibility to reduce the runtime is to exploit subsystems that are temporarily inactive during the transient simulation. We introduce a dependency graph which enables the prediction of the influence of signal changes and a splitting of the system into active and inactive subsystems. Based on this decomposition, we define signal-flow based Runge–Kutta methods which automatically identify inactive subsystems and skip the recomputation of these regions. This leads to a significantly reduced number of time-consuming function evaluations.

Zusammenfassung

Durch die Simulation integrierter Schaltkreise lassen sich schon vor der Fertigung detaillierte Aussagen über die Funktionalität und das Leistungsverhalten treffen. Insbesondere die präzise aber rechenintensive Simulation auf Schaltkreisebene spielt dabei eine zentrale Rolle. Aufgrund der stetig steigenden Komplexität integrierter Schaltkreise und der damit verbundenen zunehmenden Simulationsdauer besteht weiterhin ein Bedarf an effizienten numerischen Verfahren zur Lösung der resultierenden hochdimensionalen differential-algebraischen Gleichungssysteme.

Die Standardvorgehensweise, diese Systeme zu lösen, kann in zwei wesentliche Schritte unterteilt werden: die Berechnung konsistenter Anfangsbedingungen und die anschließende numerische Integration mithilfe impliziter Einschritt- oder Mehrschrittverfahren. In der vorliegenden Arbeit werden unterschiedliche Modelle zur Beschreibung des Signalflusses integrierter Schaltkreise vorgestellt. Darauf aufbauend werden graphbasierte Verfahren entwickelt, die Simulation unter Ausnutzung der zugrundeliegenden Netzwerkstruktur zu beschleunigen.

Die Bestimmung konsistenter Anfangswerte erfordert die Lösung eines nichtlinearen Gleichungssystems. Dazu wird in der Regel das Newton-Raphson-Verfahren verwendet. Um die Konvergenz dieses Verfahrens zu beschleunigen und somit die Simulationsdauer zu reduzieren, wird ein Algorithmus präsentiert, der es ermöglicht, einen geeigneten Startwert für die Iteration zu berechnen. Zu diesem Zweck wird ein eventgesteuertes Verfahren zur Simulation auf Schalterebene mit einem Modell des logischen Signalflusses kombiniert, das auf der Zerlegung des Schaltkreises in kanalverbundene und stark zusammenhängende Komponenten basiert.

Eine weitere Möglichkeit, die Simulationsdauer zu reduzieren, besteht darin, die während der Transientenanalyse temporär inaktiven Bereiche auszunutzen. Dazu wird ein Abhängigkeitsgraph generiert, der Aussagen über den Verlauf von Signaländerungen und eine Partitionierung in aktive und inaktive Teilbereiche ermöglicht. Es werden dann signalfluss-basierte Runge-Kutta-Verfahren definiert, die inaktive Teilsysteme automatisch erkennen und die Neuberechnung dieser Bereiche vermeiden. Somit lässt sich die Anzahl der benötigten Funktionsauswertungen signifikant verringern.

Contents

List of Figures

'The idea behind digital computers may be explained by saying that these machines are intended to carry out any operations which could be done by a human computer.'

Alan Turing, Computing Machinery and Intelligence (1950)

1 Introduction

At the time of Alan Turing, who was one of the most influential persons in modern computer science, the word *computer* denoted a person who performed calculations by hand. Today electronic devices carry out the vast majority of all calculations. These devices find applications in many different areas such as data processing, telecommunication, automotive electronics, and also consumer electronics. This rapid development would not have been possible without the ongoing progress in microelectronics and the extensive use of computer-aided circuit simulation. Since the invention of the transistor in the year 1947 and the production of the first integrated circuit in the year 1958, the complexity and performance rose remarkably. Moore's law predicts a doubling of the number of transistors per chip for approximately every two years and it still seems to be valid. The ever increasing functionality and the decreasing product cycles complicate the design of new integrated circuits. Due to the rising complexity, there is a continuing need in efficient and reliable simulation methods.

The simulation of integrated circuits plays an important role in the design process since it enables the verification of the correct functioning and provides important performance values prior to their fabrication. This leads to a reduction of development time and costs. The first circuit simulators appeared in the late 1960s and set the stage for the widespread

simulators developed in the 1970s such as, for instance, SPICE[1] [Kun95].

With increasing circuit sizes, different simulation techniques emerged to cope with the complexity without sacrificing accuracy and reliability. These techniques can be divided into two categories: The first approach is to view the circuit as a continuous dynamical system with unknowns such as voltages, currents, charges, and fluxes. The combination of the characteristic equations of the basic circuit elements and Kirchhoff's laws results in general in a system of differential and algebraic equations. This level of abstraction is called the circuit level. The objective of an analog circuit-level simulator is to solve the system of equations numerically and to provide detailed information on the behavior and performance of the circuit [ROTH89].

The second approach treats the circuit as a digital network. Instead of continuous voltages and currents, only discrete signal states are used to describe the behavior. Basically, there are three different states, namely *low*, *high*, and *unknown*. In some cases also signal strength are considered to model, for example, different transistor sizes. Selective trace methods can be applied to recompute only the regions which are active during the simulation. Hence, also large-scale circuits can be simulated efficiently. However, since only digital states are considered, the dynamic behavior of the circuits cannot be examined accurately.

The logic level can be further divided into switch-level and gate-level methods. Since gate-level simulators could not adequately handle the bidirectional signal-flow of pass transistors or transmission gates, switch-level simulation methods which operate not on the gates of the circuit but directly on the transistors were developed. A transistor is regarded as a voltage-controlled switch which is – depending on the state of the gate and the type of the transistor – either on or off. There are several other levels of abstraction such as the register-transfer level or the algorithmic level. In this thesis, we will focus on circuit-level and switch-level simulation.

The circuit-level simulation of large-scale integrated circuits is very time-consuming. The aim of this thesis is to speed up the simulation exploiting information on the network structure and the signal flow of integrated circuits. The topology of a circuit can be viewed as a graph in which the edges or hyperedges represent the different circuit elements and the vertices the corresponding nodes. This graph representation is usually used to generate the mathematical model of the circuit, but not to solve the resulting system of differential and algebraic equations numerically. The equations are in general solved with standard

[1] SPICE: Simulation Program with Integrated Circuit Emphasis

algorithms which do not take into account the underlying circuit structure. We will present two different approaches to exploit the network structure in order to reduce the runtime of the simulation.

Prior to the numerical integration of the differential-algebraic equation, consistent initial values have to be computed. This requires the solution of a system of nonlinear equations. Without an appropriate initial guess, the standard Newton–Raphson method often fails to compute a solution so that usually more robust but slower continuation methods are used instead. We will propose a signal-flow based algorithm which computes an approximate operating point at the switch level. The approximate solution is then used as a starting point for the Newton–Raphson iteration. Provided that the approximate operating point is sufficiently close to a solution of the system of nonlinear equations, the number of iterations and thus the runtime of the operating point analysis can be reduced significantly.

Subsequent to the determination of consistent initial values, the system is usually solved with standard integration schemes such as BDF[2] methods or the trapezoidal rule. During the simulation, the major part of the circuit is in general inactive. Conventional integration schemes discretize the entire system with the same step size. As a result, inactive subsystems are recomputed with an unnecessarily high accuracy. With the aid of multirate integration schemes, it is possible to exploit these inactive regions and to reduce the number of time-consuming function evaluations. We will introduce signal-flow based Runge–Kutta methods which automatically identify inactive subsystems and recompute only the active parts of the system. To elaborate on the concepts, we will confine ourselves to specific CMOS[3] circuits which can be written as a system of ordinary differential equations.

The outline of this thesis is as follows: Chapter 2 contains an introduction to CMOS circuits and a description of the principles and basic equations that are required to assemble and solve the systems of equations at the circuit level. Furthermore, we briefly introduce the graph-theoretic tools and techniques needed for the generation of the circuit equations and in particular for the analysis of the signal flow.

In Chapter 3, we develop a model of the logic signal flow of integrated circuits which is based on the partitioning into channel-connected and strongly connected components. Subsequently, we present a switch-level algorithm and an extension of the capabilities of switch-level simulation in the direction of circuit-level simulation. The output of the new

[2]BDF: Backward Differentiation Formulae
[3]CMOS: Complementary Metal Oxide Semiconductor

algorithm is used to speed up the convergence of the DC analysis. An implementation of these methods has been integrated into an industrial circuit simulator. We conclude the chapter with a comparison of the standard and the signal-flow based approach using a set of benchmark circuits and cross-sections of large integrated circuits.

In Chapter 4, we propose integration schemes tailored to complex dynamical systems with inherent latency. The aim is to reduce the number of function evaluations and thus to speed up the simulation exploiting inactive subsystems. To this end, we introduce a graph which models the signal flow of a given complex system. We then analyze the system's behavior using tools from algebraic graph theory and develop signal-flow based Runge–Kutta methods which take into account the different rates of activity. Furthermore, we describe an extension of these methods to identify and exploit periodic subsystems. The impact of the signal-flow based integration schemes is illustrated by means of CMOS circuits and further examples.

All proposed algorithms described in Chapter 3 and Chapter 4, the approximate operating point analysis and the signal-flow based Runge–Kutta methods, have been implemented in C++ in order to analyze and improve the different approaches. Chapter 5 presents details on the implementation of these methods and provides information on the newly developed software tool *signalflow*.

A summary of the achieved results is given in Chapter 6. Furthermore, open problems and possible future directions to extend and enhance the proposed signal-flow based methods are discussed.

In Appendix A, we introduce some basic concepts in the theory of differential-algebraic equations and illustrate the differences between ordinary differential equations and differential-algebraic equations. Moreover, we present methods for the numerical solution of differential-algebraic equations.

2 Integrated circuits

In this chapter, we will outline the basic principles to model the behavior of integrated circuits. The chapter starts with a brief introduction to CMOS technology and a description of graph-theoretic tools and techniques which are required for the modeling and analysis of integrated circuits. Subsequently, we will present methods to set up and solve the circuit equations.

2.1 CMOS technology

CMOS technology is currently used in more than 95 % of all integrated circuits and will also in the foreseeable future remain dominant [Bak08]. This manufacturing technique enables the production of transistors in the nanoscale regime. Processors, memory chips, and microcontrollers, to name but a few, are mainly produced in CMOS technology. Another advantage of CMOS circuits is the low power consumption, the transistors draw power only when they are switching polarity [Voi06].

In CMOS circuits, both n-channel and p-channel MOSFETs[1] are used to implement logic functions. A MOSFET is a four-terminal device, the terminals are labeled drain (d),

[1] MOSFET: Metal Oxide Semiconductor Field Effect Transistor

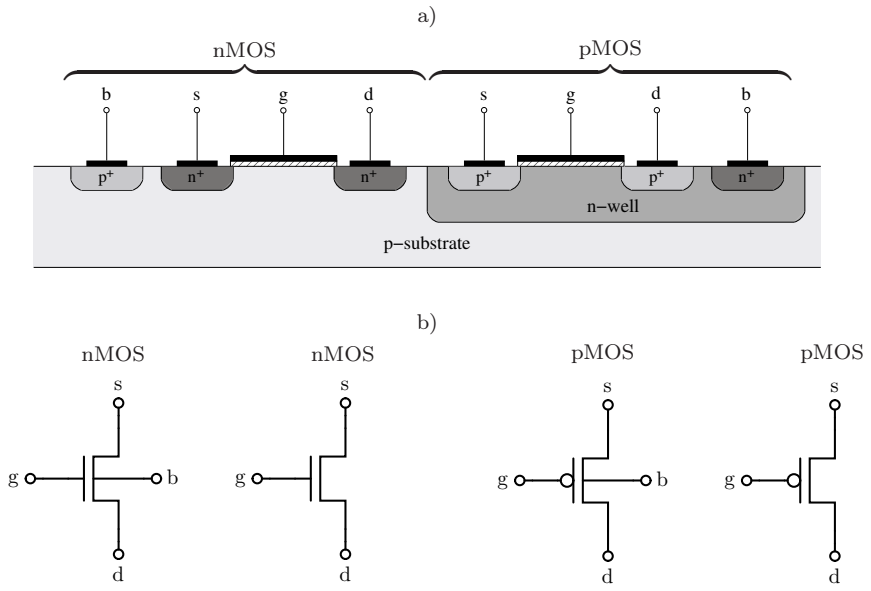


Figure 2.1: Layout and representation of n-channel and p-channel MOSFETs. a) Cross-sectional view. b) Corresponding circuit symbols.

gate (g), source (s), and bulk (b). Figure 2.1a shows the physical layout of the two different MOSFET types, Figure 2.1b the corresponding circuit symbols. If the bulk terminal of the n-channel or p-channel transistor is not drawn, then it is assumed to be connected to ground or the positive supply voltage, respectively. For a detailed description of the fabrication process and the functioning of CMOS circuits, we refer to [Hof04, Bak08].

In order to derive a simple analytical model of the current-voltage behavior, the operation of the MOSFET can be split into three different regions, namely the sub-threshold region, the triode region, and the saturation region [Hof04]. We describe the characteristics of these regions using the example of the nMOS transistor, the behavior of pMOS transistors can be explained analogously.

In the sub-threshold region, the nMOS transistor is turned off and the current between source and drain is assumed to be zero. If a positive voltage v_{gs} is applied, free holes with a positive charge are repelled from the region under the gate and pushed downward. At the same time, the electrons from the heavily doped n^+ regions are attracted. The voltage at which the electrons form a conducting channel between source and drain is called the threshold voltage $v_{th,n}$. When the gate-source voltage exceeds the threshold voltage, then the transistor is turned on and a current flows between source and drain. If $v_{ds} < v_{gs} - v_{th,n}$, the transistor is said to be in the triode region and can be regarded

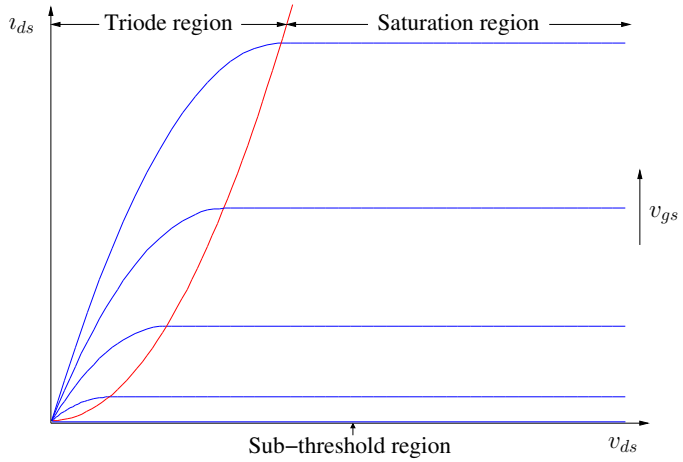


Figure 2.2: Output characteristics of the nMOS transistor. The blue curves show the dependence of the drain-source current $\imath_{ds}$ on the applied drain-source voltage v_{ds} for different gate-source voltages v_{gs}. The red line separates the triode region and the saturation region.

as a resistor between source and drain. If $v_{ds} \geq v_{gs} - v_{th,n}$, the transistor is said to be saturated and the current no longer rises with increasing v_{ds}.

The so-called Shichman–Hodges model [SH68] can be used as a first-order approximation of the MOSFET behavior. In most circuit simulators, the Shichman–Hodges model is called the *Level 1* model. The equations for the different regions of the nMOS and pMOS transistor can be written as

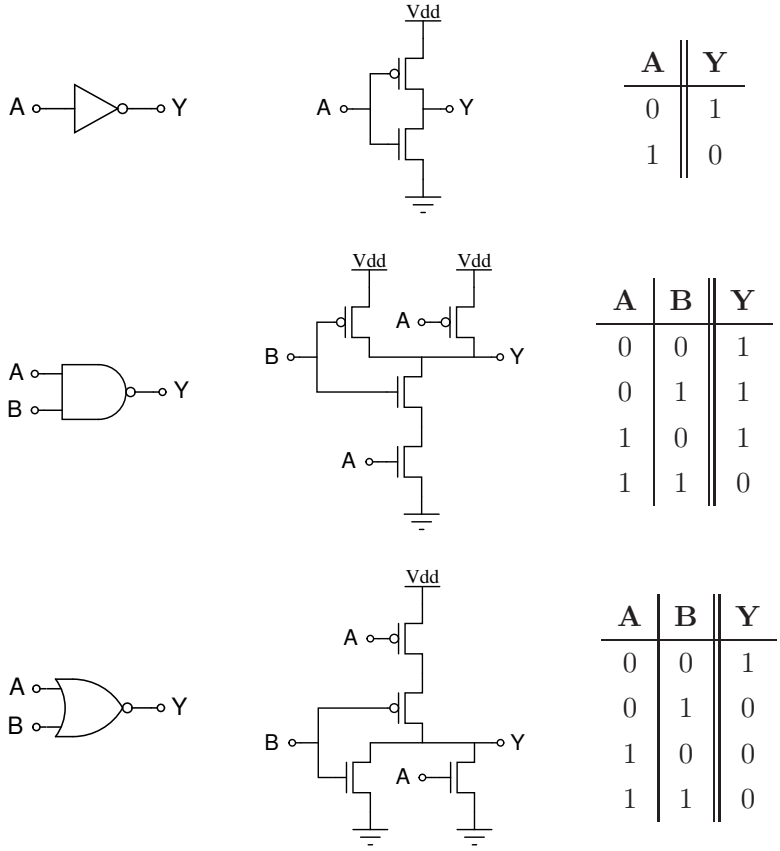
$$\imath_{ds,n} = \begin{cases} 0, & v_{gs} \leq v_{th,n}, \\ \beta_n \left[(v_{gs} - v_{th,n}) v_{ds} - \frac{v_{ds}^2}{2} \right], & v_{gs} > v_{th,n} \wedge v_{ds} < v_{gs} - v_{th,n}, \\ \frac{\beta_n}{2} (v_{gs} - v_{th,n})^2, & v_{gs} > v_{th,n} \wedge v_{ds} \geq v_{gs} - v_{th,n}, \end{cases} \tag{2.1a}$$

$$\imath_{ds,p} = \begin{cases} 0, & v_{gs} \geq v_{th,p}, \\ -\beta_p \left[(v_{gs} - v_{th,p}) v_{ds} - \frac{v_{ds}^2}{2} \right], & v_{gs} < v_{th,p} \wedge v_{ds} > v_{gs} - v_{th,p}, \\ -\frac{\beta_p}{2} (v_{gs} - v_{th,p})^2, & v_{gs} < v_{th,p} \wedge v_{ds} \leq v_{gs} - v_{th,p}, \end{cases} \tag{2.1b}$$

where β_n and β_p are parameters which depend on the width and length of the channel. That is, the drain-source current is a function of the voltages v_d, v_g, and v_s. The characteristics of the nMOS transistor are shown in Figure 2.2.

Since due to the shrinking transistor sizes also higher-order effects become increasingly important, usually more complex transistor models such as BSIM[2] are used [Hof04]. We will use the Shichman–Hodges model in Chapter 3 to replace the pure digital switch-level transistor model and in Chapter 4 for the analysis of the newly developed signal-flow based integration schemes.

From the digital point of view, a MOSFET can be regarded as a voltage-controlled switch. A voltage close to V_{dd} is assigned the digital value *high* or 1 and a voltage close to ground the digital value *low* or 0. The switch is closed if the gate of the nMOS transistor is high or the gate of the pMOS transistor is low, and the switch is open if the gate of the nMOS transistor is low or the gate of the pMOS transistor is high.



A	Y
0	1
1	0

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Figure 2.3: Inverter, NAND gate, and NOR gate. The left column shows the gate-level representation, the middle column the CMOS implementation, and the right column the corresponding truth table.

[2]BSIM: Berkeley Short channel IGFET Model

Figure 2.3 shows the CMOS implementation of the basic logic functions NOT, NAND, and NOR. Any other logic function can be realized in the same way using CMOS technology. This digital interpretation of the MOSFET behavior will be utilized in Chapter 3 to speed up the computation of operating points.

2.2 Directed graphs, multigraphs, and hypergraphs

For the simulation of integrated circuits and in particular for the analysis of the signal flow, tools and techniques from graph theory are essential. In this section, we briefly introduce the required definitions and algorithms. A more detailed description can be found in [CO93, DA93, CLRS01], for example.

Definition 2.1 (Directed graph) A *directed graph* $\mathfrak{G}$ is defined to be a pair $(\mathfrak{V}, \mathfrak{E})$, with $\mathfrak{V} = \{\mathfrak{v}_1, \ldots, \mathfrak{v}_n\}$ being the set of vertices and $\mathfrak{E} \subseteq \mathfrak{V} \times \mathfrak{V}$ being the set of edges.

If $(\mathfrak{u}, \mathfrak{v}) \in \mathfrak{E}$ is an edge of $\mathfrak{G}$, then the vertices $\mathfrak{u}$ and $\mathfrak{v}$ are said to be *adjacent*, $\mathfrak{u}$ is called the *tail* and $\mathfrak{v}$ is called the *head* of the edge. The *in-degree* of a vertex $\mathfrak{u}$ is defined to be the number of edges with $\mathfrak{u}$ as its head, the *out-degree* the number of edges with $\mathfrak{u}$ as its tail. An edge which joins a vertex to itself is called a *self-loop*. A graph can be represented by a matrix and vice versa.

Definition 2.2 (Adjacency matrix) For a graph $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$, define the *adjacency matrix* $A = (a_{ij}) \in \mathbb{R}^{n \times n}$ by

$$a_{ij} = \begin{cases} 1, & \text{if } (\mathfrak{v}_i, \mathfrak{v}_j) \in \mathfrak{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.2}$$

If the edges of a graph $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$ are weighted, that is, each edge $(\mathfrak{v}_i, \mathfrak{v}_j) \in \mathfrak{E}$ has a weight w_{ij}, then the (i, j)-entry of the *weighted adjacency matrix* is defined to be w_{ij}. Analogously, we can assign a matrix a directed graph as follows.

Definition 2.3 (Directed graph of a matrix) Given a matrix $A = (a_{ij}) \in \mathbb{R}^{n \times n}$, define $\mathfrak{V} = \{\mathfrak{v}_1, \ldots, \mathfrak{v}_n\}$ and $\mathfrak{E} = \{(\mathfrak{v}_i, \mathfrak{v}_j) \mid a_{ij} \neq 0\}$. With the matrix A, we associate the directed graph $\mathfrak{G}(A) = (\mathfrak{V}, \mathfrak{E})$.

Other graph-related matrices are the incidence matrix and the Laplacian of a graph. Kirchhoff's laws, for instance, can be easily expressed using the incidence matrix of the circuit graph, as we will describe in Section 2.3.

Definition 2.4 (Incidence matrix) Let the edges of a directed graph $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$ be numbered arbitrarily, i.e. $\mathfrak{E} = \{\mathfrak{e}_1, \ldots, \mathfrak{e}_m\}$. Then the *incidence matrix* $B = (b_{ij}) \in \mathbb{R}^{n \times m}$ is defined by

$$b_{ij} = \begin{cases} 0, & \text{if } \mathfrak{e}_j \text{ is not connected to } \mathfrak{v}_i, \\ 1, & \text{if } \mathfrak{e}_j \text{ leaves } \mathfrak{v}_i, \\ -1, & \text{if } \mathfrak{e}_j \text{ enters } \mathfrak{v}_i, \\ 2, & \text{if } \mathfrak{e}_j \text{ is a self-loop at } \mathfrak{v}_i. \end{cases} \tag{2.3}$$

Definition 2.5 (Laplacian) Let d_i be the degree of vertex $\mathfrak{v}_i$. The *Laplacian* of a graph $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$ is defined to be the matrix $L = D - A$, where $D = \operatorname{diag}(d_1, \ldots, d_n)$ is the degree matrix and A the adjacency matrix[3]. Depending on the application, either the in-degree or the out-degree can be used.

There is a close relation between a graph and its matrix representation. In Chapter 4, we will exploit this relationship in order to analyze the signal flow of complex dynamical networks.

Definition 2.6 (Path) The vertices $\mathfrak{u}$ and $\mathfrak{v}$ of a graph $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$ are defined to be connected by a *path* of length l, written $\mathfrak{u} \xrightarrow[\mathfrak{G}]{l} \mathfrak{v}$, if a sequence of – not necessarily distinct – vertices $(\mathfrak{u}_0, \mathfrak{u}_1, \ldots, \mathfrak{u}_l)$ exists such that $\mathfrak{u} = \mathfrak{u}_0$, $\mathfrak{v} = \mathfrak{u}_l$, and $(\mathfrak{u}_{i-1}, \mathfrak{u}_i) \in \mathfrak{E}$ for $i = 1, \ldots, l$.

If all vertices of a path are distinct, then it is called a *simple path*[4]. If a path from $\mathfrak{u}$ to $\mathfrak{v}$ exists, $\mathfrak{v}$ is defined to be *reachable* from $\mathfrak{u}$. The vertex $\mathfrak{v}$ is called a *successor* of $\mathfrak{u}$ and $\mathfrak{u}$ a *predecessor* of $\mathfrak{v}$. A path $\mathfrak{u} \xrightarrow[\mathfrak{G}]{l} \mathfrak{u}$ is said to form a *cycle*. A directed graph without cycles is also referred to as a *directed acyclic graph* or *DAG*.

Definition 2.7 (Reachability) Given a graph $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$, define $\operatorname{reach}^l_{\mathfrak{G}}(\mathfrak{u})$ to be the set of all vertices that are reachable from $\mathfrak{u}$ by a path of maximal length l, i.e.

$$\operatorname{reach}^l_{\mathfrak{G}}(\mathfrak{u}) = \{\mathfrak{v} \mid \exists\, \mathfrak{u} \xrightarrow[\mathfrak{G}]{k} \mathfrak{v},\ k \leq l\}. \tag{2.4}$$

Analogously, define $\operatorname{reach}^l_{\mathfrak{G}}(\mathfrak{U}) = \bigcup_{\mathfrak{u} \in \mathfrak{U}} \operatorname{reach}^l_{\mathfrak{G}}(\mathfrak{u})$ for subsets $\mathfrak{U} \subseteq \mathfrak{V}$.

If we want to take into account paths of arbitrary length, then we write $\operatorname{reach}^\infty_{\mathfrak{G}}(\mathfrak{u})$ and $\operatorname{reach}^\infty_{\mathfrak{G}}(\mathfrak{U})$, respectively.

[3] In some references the Laplacian is defined as $L = I - D^{-1}A$.

[4] In the literature sometimes the terms *walk* and *path* are used instead of *path* and *simple path*.

Definition 2.8 (Transitive closure) The *transitive closure* of a graph $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$ is defined to be the graph $\mathfrak{G}^* = (\mathfrak{V}, \mathfrak{E}^*)$, with $\mathfrak{E}^* = \{(\mathfrak{u}, \mathfrak{v}) \mid \mathfrak{v} \in \text{reach}_{\mathfrak{G}}^{\infty}(\mathfrak{u})\}$.

For the analysis of graphs, it is often required to visit the vertices and edges in a systematic way. The *depth-first search* or *DFS* is a simple and efficient method to process a graph: Starting at a given vertex, the depth-first search follows unexplored outgoing edges whenever possible. If all edges have been explored, then the search returns to the vertex from which it was discovered, until all reachable vertices have been processed. If still unvisited vertices exist, the search is continued from one of the unvisited vertices. This procedure is repeated until all vertices are discovered [CLRS01].

The depth-first search can also be used to compute the strongly connected components of a graph exploiting the fact that if $\mathfrak{v}$ is reachable from $\mathfrak{u}$ in $\mathfrak{G}$, then $\mathfrak{u}$ is reachable from $\mathfrak{v}$ in $\mathfrak{G}^T$, with $\mathfrak{G}^T = (\mathfrak{V}, \mathfrak{E}^T)$ and $\mathfrak{E}^T = \{(\mathfrak{v}, \mathfrak{u}) \mid (\mathfrak{u}, \mathfrak{v}) \in \mathfrak{E}\}$.

Definition 2.9 (Strongly connected component) A *strongly connected component* or *SCC* of a directed graph $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$ is a maximal set of vertices $\mathfrak{U} \subseteq \mathfrak{V}$ such that for each $\mathfrak{u}, \mathfrak{v} \in \mathfrak{U}$ holds that $\mathfrak{v}$ is reachable from $\mathfrak{u}$ and $\mathfrak{u}$ is reachable from $\mathfrak{v}$.

If each strongly connected component is condensed to a single vertex, then the resulting graph is a directed acyclic graph. The condensation of the strongly connected components allows for a topological ordering of the graph, that is, a sorting of the vertices such that $\mathfrak{u}$ comes before $\mathfrak{v}$ if there is an edge $(\mathfrak{u}, \mathfrak{v}) \in \mathfrak{E}$. This graph decomposition and ordering is extremely useful and will be applied several times in the following chapters.

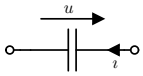
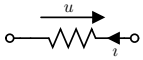
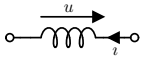
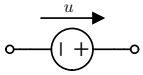
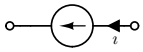
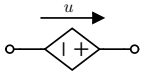
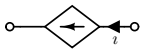
Sometimes also graphs with multiple edges connecting two vertices or edges connecting an arbitrary number of vertices are necessary to describe the network structure. To this end, the graph concept can be generalized to multigraphs and hypergraphs.

Definition 2.10 (Directed multigraph) A *directed multigraph* $\mathfrak{G}$ is a pair $(\mathfrak{V}, \mathfrak{E})$, with $\mathfrak{V} = \{\mathfrak{v}_1, \ldots, \mathfrak{v}_n\}$ being the set of vertices and $\mathfrak{E}$ being the multiset of ordered pairs of vertices.

Remark 2.11 A multiset is a collection of not necessarily distinct objects. Formally, a multiset over a set A can be defined as a pair (A, m), where m is a function $m : A \mapsto \mathbb{N}$ with $m(x)$ being the multiplicity of x.

Definition 2.12 (Hypergraph) Let $\mathcal{P}(\mathfrak{V})$ denote the power set of $\mathfrak{V}$. A *hypergraph* $\mathfrak{G}$ is a pair $(\mathfrak{V}, \mathfrak{E})$ consisting of a set of vertices $\mathfrak{V} = \{\mathfrak{v}_1, \ldots, \mathfrak{v}_n\}$ and a set of edges $\mathfrak{E} \subseteq \mathcal{P}(\mathfrak{V})$.

Table 2.1: Characteristic equations of the basic circuit elements.

Device	Symbol	Characteristics
Capacitor		$\imath = \dot{q}_C(t, u)$
Resistor		$\imath = g(t, u)$
Inductor		$u = \dot{\phi}_L(t, \imath)$
Voltage source		$u = v_s(t)$
Current source		$\imath = \imath_s(t)$
Controlled voltage source		$u = v_s(t, u_{ctrl}, \imath_{ctrl})$
Controlled current source		$\imath = \imath_s(t, u_{ctrl}, \imath_{ctrl})$

It is possible to define directed and undirected hypergraphs, but typically only undirected hypergraphs are considered. Clearly, the two definitions above can be combined to form multi-hypergraphs. The topology of an arbitrary integrated circuit, for example, can be viewed as a multi-hypergraph.

2.3 Circuit-level simulation

Due to the increasing integration density and the resulting parasitic effects, the accurate circuit-level simulation is still of great importance [Frö02]. At the circuit level, the behavior of the circuit is described in terms of branch voltages u, branch currents $\imath$, and node voltages v. Additionally, electrical charges q and magnetic fluxes ϕ are taken into account.

A circuit can be regarded as a network of capacitors, resistors, inductors, voltage sources, and current sources. The characteristic equations of these elements are shown in Table 2.1. More complex circuit elements such as, for instance, transistors can be replaced by equivalent circuits which are composed of the basic elements.

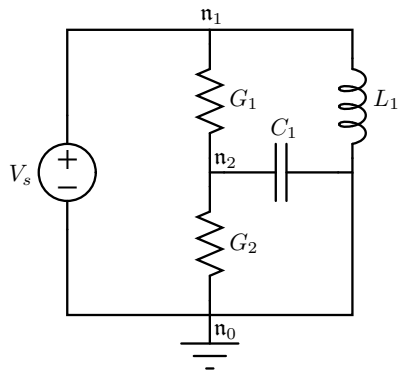


Figure 2.4: RLC circuit.

In addition to the characteristic equations of the elements, the topology of the circuit has to be taken into account. This can be accomplished using Kirchhoff's laws:

i) *Kirchhoff's current law (KCL):* The sum of all currents that enter a node is equal to zero.

ii) *Kirchhoff's voltage law (KVL):* The sum of voltages along each loop of the network is equal to zero.

The combination of the characteristic equations and Kirchhoff's laws leads in general to a mixed system of differential and algebraic equations of the form

$$F\big(t, x(t), \dot{x}(t)\big) = 0. \tag{2.5}$$

Example 2.13 Consider the circuit shown in Figure 2.4. Combining Kirchhoff's current law and the characteristic equations of the circuit elements, we obtain the index-1 system

$$\begin{aligned} 0 &= G_1(V_s(t) - v_2) + \imath_L - \imath_V, \\ 0 &= C_1 \frac{dv_2}{dt} - G_1(V_s(t) - v_2) + G_2 v_2, \\ 0 &= V_s(t) - L_1 \frac{d\imath_L}{dt}. \end{aligned}$$ ♢

Details on general differential-algebraic equations and their numerical treatment can be found in Appendix A. Below, we will focus on the differential-algebraic equations coming from the modified nodal analysis.

2.3.1 Modified nodal analysis

Due to the high complexity of modern integrated circuits, it is essential to generate the circuit equations in a systematic way. In most circuit simulators, the modified nodal analysis (MNA) is used to assemble the system of equations. In the following, we present the charge and flux oriented and the standard modified nodal analysis. A more detailed description can be found, for example, in [Est00, GFtM05, Voi06, Bäc07].

The circuit topology can be interpreted as a directed graph in which each edge represents a basic element and each vertex the corresponding node. This structure is completely determined by the reduced incidence matrix of the graph, which can be obtained from the standard incidence matrix by deleting the row which corresponds to the ground node. The orientation of the edges is given by the assumed direction of the branch currents.

Let A be the reduced incidence matrix of a given circuit. Then Kirchhoff's current law for the whole circuit can be written as

$$A\,\imath = 0. \tag{2.6a}$$

Analogously, Kirchhoff's voltage law for the whole circuit states

$$A^T v = u, \tag{2.6b}$$

relating the node voltages v to the branch voltages u. If we split the circuit into the subgraphs that are induced by the different module types, then the incidence matrix A and accordingly the vectors $\imath$ and u can be subdivided into

$$\begin{aligned} A &= [A_C, A_R, A_L, A_V, A_I]\,, \\ \imath &= \left[\imath_C^T, \imath_R^T, \imath_L^T, \imath_V^T, \imath_I^T\right]^T, \\ u &= \left[u_C^T, u_R^T, u_L^T, u_V^T, u_I^T\right]^T. \end{aligned} \tag{2.7}$$

Hence, (2.6) can be rewritten as

$$A\,\imath = A_C\,\imath_C + A_R\,\imath_R + A_L\,\imath_L + A_V\,\imath_V + A_I\,\imath_I = 0 \tag{2.8a}$$

and

$$A^T v = \begin{bmatrix} A_C^T v \\ A_R^T v \\ A_L^T v \\ A_V^T v \\ A_I^T v \end{bmatrix} = \begin{bmatrix} u_C \\ u_R \\ u_L \\ u_V \\ u_I \end{bmatrix}. \tag{2.8b}$$

Now, we can insert the vector-valued characteristic equations of the basic elements. The currents of the capacitive branches are given by the time derivative of the charges q, i.e.

$$\imath_C = \dot{q}, \quad q = q_C(t, u_C). \tag{2.9a}$$

The currents of the resistive branches $\imath_R$ are given by

$$\imath_R = g(t, u_R). \tag{2.9b}$$

For the inductive branches, we obtain

$$u_L = \dot{\phi}, \quad \phi = \phi_L(t, \imath_L), \tag{2.9c}$$

where ϕ is the vector of magnetic fluxes through the inductors. The equations for the voltage and current sources can be written as

$$u_V = v_s(t, u, \dot{q}, \imath_L, \imath_V) \tag{2.9d}$$

and

$$\imath_I = \imath_s(t, u, \dot{q}, \imath_L, \imath_V). \tag{2.9e}$$

If the circuit does not contain controlled sources, then the equations can be reduced to $u_V = v_s(t)$ and $\imath_I = \imath_s(t)$, respectively.

Thus, the combination of Kirchhoff's laws (2.8) and the characteristic equations of the different module types (2.9) yields the system of differential and algebraic equations

$$\begin{aligned}
0 &= A_C\, \dot{q} + A_R\, g(t, A_R^T v) + A_L\, \imath_L + A_V\, \imath_V + A_I\, \imath_s(t, A^T v, \dot{q}, \imath_L, \imath_V),\\
0 &= \dot{\phi} - A_L^T\, v,\\
0 &= v_s(t, A^T v, \dot{q}, \imath_L, \imath_V) - A_V^T\, v,\\
0 &= q - q_C(t, A_C^T\, v),\\
0 &= \phi - \phi_L(t, \imath_L).
\end{aligned} \tag{2.10}$$

Eliminating the last two equations of (2.10), we obtain the reduced system

$$\begin{aligned}
0 &= A_C\, \dot{q}_C(t, A_C^T\, v) + A_R\, g(t, A_R^T v) + A_L\, \imath_L + A_V\, \imath_V + A_I\, \imath_s(t, A^T v, \dot{q}_C(t, A_C^T\, v), \imath_L, \imath_V),\\
0 &= \dot{\phi}_L(t, \imath_L) - A_L^T\, v,\\
0 &= v_s(t, A^T v, \dot{q}_C(t, A_C^T\, v), \imath_L, \imath_V) - A_V^T\, v,
\end{aligned} \tag{2.11}$$

where $\dot{q}_C(t, A_C^T\, v) = \frac{d}{dt}\left[q_C(t, A_C^T\, v(t))\right]$ is the vector of currents through the capacitors and $\dot{\phi}_L(t, \imath_L) = \frac{d}{dt}\left[\phi_L(t, \imath_L(t))\right]$ the vector of branch voltages across the inductors.

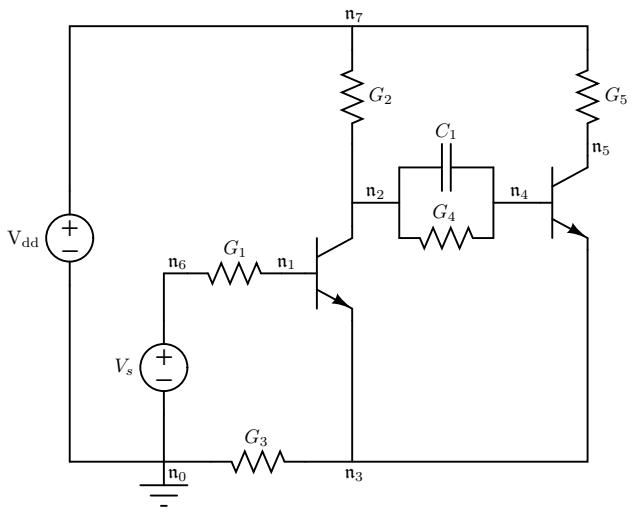


Figure 2.5: Schmitt trigger.

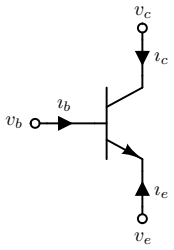
The approach described in (2.10) is called the charge and flux oriented modified nodal analysis, while (2.11) represents the standard modified nodal analysis. Although the standard modified nodal analysis yields a possibly much smaller system of equations, the charge and flux oriented method is usually preferred in industrial circuit simulators since the elimination of the conservation laws may lead to numerical instabilities [Bäc07].

Example 2.14 Consider the Schmitt trigger taken from [GFtM05]. The circuit consists of a linear capacitor with capacitance C_1, five linear resistors with conductances $G_1, \ldots, G_5$, two voltage sources, and two npn bipolar junction transistors, as shown in Figure 2.5. The first-order model of the bipolar transistor is described in Figure 2.6.

The reduced incidence matrix A of the Schmitt trigger is given by

$$A = \left[\begin{array}{c|cccccc|cc|cccc} \multicolumn{1}{c|}{A_C} & \multicolumn{6}{c|}{A_R} & \multicolumn{2}{c|}{A_V} & \multicolumn{4}{c}{A_I} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right].$$

Let $v = [v_1, \ldots, v_7]^T$ be the vector of node potentials and $\imath_V = [\imath_{V_1}, \imath_{V_2}]^T$ the vector of branch currents, where $\imath_{V_1}$ and $\imath_{V_2}$ represent the currents through the voltage sources V_s



$$\begin{aligned}
\imath_b(t) &= g(v_b(t) - v_e(t)) \\
\imath_c(t) &= \alpha\, g(v_b(t) - v_e(t)) \\
\imath_e(t) &= -(1+\alpha)\, g(v_b(t) - v_e(t)) \\
g(v) &= \beta \left(e^{v/v_T} - 1 \right)
\end{aligned}$$

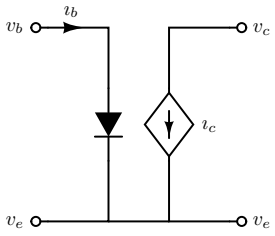


Figure 2.6: The npn bipolar junction transistor and its first-order model.

and $\mathrm{V_{dd}}$, respectively. For the linear capacitor and the linear resistors, we have

$$q_C(t, A_C^T\, v) = C A_C^T\, v, \quad C = C_1,$$

and

$$g(t, A_R^T\, v) = G A_R^T\, v, \quad G = \mathrm{diag}(G_1, \ldots, G_5).$$

The equations of the voltage and current sources are given by

$$v_s(t, A^T v, \dot{q}, \imath_L, \imath_V) = \begin{bmatrix} V_s(t) \\ \mathrm{V_{dd}} \end{bmatrix},$$

$$\imath_s(t, A^T v, \dot{q}, \imath_L, \imath_V) = \begin{bmatrix} g(v_1 - v_3) \\ \alpha\, g(v_1 - v_3) \\ g(v_4 - v_3) \\ \alpha\, g(v_4 - v_3) \end{bmatrix}.$$

Hence, the charge and flux oriented modified nodal analysis (2.10) yields

$$\begin{aligned}
0 &= G_1(v_1 - v_6) + g(v_1 - v_3), \\
0 &= -\dot{q} + G_2(v_2 - v_7) + G_4(v_2 - v_4) + \alpha\, g(v_1 - v_3), \\
0 &= G_3 v_3 - (1+\alpha)\, g(v_1 - v_3) - (1+\alpha)\, g(v_4 - v_3), \\
0 &= \dot{q} - G_4(v_2 - v_4) + g(v_4 - v_3), \\
0 &= G_5(v_5 - v_7) + \alpha\, g(v_4 - v_3), \\
0 &= -G_1(v_1 - v_6) + \imath_{V_1}, \\
0 &= -G_2(v_2 - v_7) - G_5(v_5 - v_7) + \imath_{V_2}, \\
0 &= V_s(t) - v_6, \\
0 &= \mathrm{V_{dd}} - v_7, \\
0 &= q - C_1(v_4 - v_2).
\end{aligned}$$

♢

2.3.2 Numerical solution of the circuit equations

The conventional approach to solve the circuit equations numerically can be split into two major steps. The first step is the computation of consistent initial values, the second the numerical integration of the differential-algebraic equation and the solution of the resulting nonlinear systems. Although controlled sources, for instance, may lead to differential-algebraic equations of index $\nu > 2$, these critical configurations can be detected and regularized in such a way that the resulting equations are typically of index $\nu = 1$ or $\nu = 2$ [Voi06]. Therefore, we restrict ourselves in the following to index-1 and index-2 systems. For the sake of simplicity, we rewrite the system of equations (2.10) originating from the charge and flux oriented modified nodal analysis as

$$0 = \underbrace{\begin{bmatrix} A_C & 0 \\ 0 & I \\ 0 & 0 \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} \dot{q} \\ \dot{\phi} \end{bmatrix}}_{\dot{y}} + \underbrace{\begin{bmatrix} A_R\, g(t, A_R^T v) + A_L\, \imath_L + A_V\, \imath_V + A_I\, \imath_s(t, A^T v, \imath_L, \imath_V) \\ -A_L^T\, v \\ v_s(t, A^T v, \imath_L, \imath_V) - A_V^T\, v \end{bmatrix}}_{f(t,x)},$$

$$0 = \underbrace{\begin{bmatrix} q \\ \phi \end{bmatrix}}_{y} - \underbrace{\begin{bmatrix} q_C(t, A_C^T\, v) \\ \phi_L(t, \imath_L) \end{bmatrix}}_{g(t,x)},$$

with $x = [v, \imath_L, \imath_V]^T$ assuming that the functions $\imath_s$ and v_s do not depend on $\dot{q}$, as described in [GFtM05]. Thus, we obtain a differential-algebraic equation of the form

$$\begin{aligned} 0 &= B\, \dot{y} + f(t, x), \\ 0 &= y - g(t, x). \end{aligned} \tag{2.12}$$

Consistent initial values

If the system is of index one, consistent initial conditions $[x_0, y_0]$ can be found by computing a steady state or DC operating point of the circuit. Let $\dot{y}_0 = 0$, and let x_0 be a solution of the system of nonlinear equations

$$f(t_0, x_0) = 0. \tag{2.13}$$

An operating point of the circuit is then given by $[x_0, g(t_0, x_0)]$. If the index of the system is two, the solution of the steady state problem (2.13) might result in inconsistent initial values since hidden constraints are not taken into account. However, consistent initial

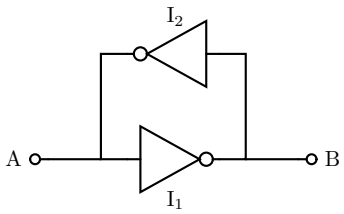


Figure 2.7: Cross-coupled inverters.

values can be obtained from the DC operating point by adding a correction vector which requires only the setup and solution of an additional linear system of equations [Est00].

It is important to note that the solution of the system of nonlinear equations is not necessarily unique. Sequential circuits, for instance, have in general multiple DC operating points. The simplest sequential circuit consists of two cross-coupled inverters [Bak08]. This configuration, which is shown in Figure 2.7, has two stable and one metastable operating point, i.e. $A = 0$ and $B = V_{dd}$, $A = V_{dd}$ and $B = 0$, as well as $A = \frac{1}{2}V_{dd}$ and $B = \frac{1}{2}V_{dd}$. In actual physical circuits, this configuration will never remain in the metastable operating point since any small electrical noise in the circuit will trigger it to one of the stable operating points. Nevertheless, the circuit simulator might as well converge to the metastable solution.

Finding a DC operating point is one of the most important and difficult tasks in electrical circuit simulation [GG05]. The standard approach to solve the system of nonlinear equations is to apply the Newton–Raphson method. Provided that the starting point is sufficiently close to a solution, the Newton–Raphson method is quadratically convergent. However, the location of possible operating points is in general unknown. Especially for large circuits, convergence is often problematic. If the Newton–Raphson method fails to compute an operating point, then usually homotopy methods are used instead.

The idea of homotopy methods is to modify the system of equations by introducing an additional parameter λ in such a way that for $\lambda = 0$ the solution of the system is easy to compute and for $\lambda = 1$ the modified system is equivalent to the original system. After the solution of the system for $\lambda = 0$, the parameter λ is gradually increased and the system is solved at each intermediate step until $\lambda = 1$. Provided that the solution depends continuously on the parameter λ, the solution of the previous step is a good starting point such that the Newton–Raphson iteration converges without problems [Kun95].

There exist different continuation methods tailored to circuit simulation such as the source stepping or the pseudo-transient algorithm. For the source stepping algorithm, all

independent sources are initially set to zero. Then the values of the sources are uniformly scaled up to their nominal values [YG99]. The pseudo-transient analysis can be regarded as a variant of the source stepping algorithm where a transient simulation is carried out for a modified pseudo-circuit.

The homotopy-based methods are more robust but also slower than the plain Newton–Raphson method. We will address the problem of efficiently computing operating points in Chapter 3. Circuit simulators such as SPICE usually allow the user to specify initial values for some or all nodes of the circuit in order to provide a better starting point for the Newton–Raphson iteration or – in case of multiple operating points – to find a particular solution. We will exploit this feature to aid the convergence of the operating point analysis. The aim of this approach is to find a starting point which is sufficiently close to a solution so that the standard Newton–Rapshon method converges.

Numerical integration – the direct approach

After the determination of consistent initial values, different methods can be used to solve the initial value problem numerically. The standard approach is to apply implicit multi-step methods for the time discretization, in particular lower-order BDF schemes or the trapezoidal rule, and to solve the resulting systems of nonlinear equations with the aid of the Newton–Raphson method [GFtM05]. BDF methods, which are described in more detail in Appendix A, can be written as

$$\dot{y}^m = \frac{1}{h} \sum_{i=0}^{s} \alpha_i \, y^{m-i} := \gamma^m \, y^m + r^m. \tag{2.14}$$

With $y^{m-i} = g(t^{m-i}, x^{m-i})$, $i = 0, \ldots, s$, the numerical solution of the differential-algebraic equation (2.12) can be reduced to the repeated solution of the system of nonlinear equations

$$B(\gamma^m \, g(t^m, x^m) + r^m) + f(t^m, x^m) = 0. \tag{2.15}$$

The solution can be computed efficiently with the standard Newton–Raphson method. That is, in order to obtain a new approximation

$$x^{m,k+1} = x^{m,k} + \Delta x^{m,k}, \tag{2.16}$$

the linear system of equations

$$\begin{aligned} \left(\gamma^m \, B \, \frac{\partial g}{\partial x}(t^m, x^{m,k}) + \frac{\partial f}{\partial x}(t^m, x^{m,k}) \right) \Delta x^{m,k} \\ = -B(\gamma^m \, g(t^m, x^{m,k}) + r^m) - f(t^m, x^{m,k}) \end{aligned} \tag{2.17}$$

has to be solved. The vector $x^{m,0} = x^{m-1}$ can be used as a starting point for the iteration. Here, the right-hand side and the Jacobian can be generated at the same time using so-called element stamps. For this purpose, each element of the circuit is processed and the individual terms are added to the corresponding entries of the right-hand side and the Jacobian. In general, the cost of generating both the right-hand side and the Jacobian is only slightly higher than the cost of evaluating the right-hand side only [BGK01, Voi06].

If the differential-algebraic equation is of index two, special care has to be taken to fix a weak instability associated with the system. This can be accomplished using an index monitor which identifies and regularizes critical circuit configurations based on topological and local numerical checks [MEF+03, GFtM05].

Instead of multi-step methods also one-step methods or general linear methods can be applied. If the system exhibits frequent discontinuities, one-step methods are potentially more efficient since multi-step methods must be restarted – usually at low order – after each discontinuity, whereas Runge–Kutta methods can be restarted at a higher order [BCP89]. For a detailed description of one-step and multi-step methods tailored to differential-algebraic equations, we refer to [BCP89, KM06]. General linear methods for the simulation of integrated circuits are studied in [Voi06]. The advantage of the direct approach is that it works for any circuit, provided the index of the differential-algebraic equation is not too high [GFtM05].

Numerical integration – the indirect approach

Relaxation-based circuit simulation techniques such as iterated timing analysis or waveform relaxation, on the other hand, can be used efficiently only for a restricted class of circuits. Both methods aim at exploiting the mainly unidirectional signal flow of digital circuits. Since the convergence of purely node-based relaxation schemes can be very slow if the circuit contains tightly coupled subsystems, it is more efficient to solve strongly connected subcircuits using direct methods and to apply relaxation methods only between weakly coupled blocks [SN90].

The efficiency of the resulting block relaxation methods, which can be regarded as a combination of direct methods and relaxation techniques, depends strongly on the ability to decompose the system into loosely coupled subsystems and to order these subsystems according to the signal flow. For the decomposition of integrated circuits, algorithms based on the partitioning into channel-connected and strongly connected components can be used. We will use similar techniques in Chapter 3 to generate a model of the logic signal flow.

The idea of iterated timing analysis is to discretize the differential-algebraic equation (2.12) using standard integration schemes and to apply relaxation-based methods at the nonlinear equation level [Sal84]. Let p be the number of subsystems. Then x and y can be written as $x = \left[x_1^T, \ldots, x_p^T\right]^T$ and $y = \left[y_1^T, \ldots, y_p^T\right]^T$. Accordingly, the matrix B and the functions f and g can be decomposed in the same way. For the Gauss–Jacobi or the Gauss–Seidel based iteration, define

$$x^l = \begin{bmatrix} x_1^{l-1} \\ \vdots \\ x_{i-1}^{l-1} \\ x_i^l \\ x_{i+1}^{l-1} \\ \vdots \\ x_p^{l-1} \end{bmatrix} \quad \text{or} \quad x^l = \begin{bmatrix} x_1^l \\ \vdots \\ x_i^l \\ x_{i+1}^{l-1} \\ \vdots \\ x_p^{l-1} \end{bmatrix}, \tag{2.18}$$

respectively. Omitting the superscript m for simplicity, at each step of the iterated timing analysis, the set of systems of nonlinear equations

$$B_i(\gamma\, g(t, x^l) + r) + f_i(t, x^l) = 0, \quad i = 1, \ldots, p, \tag{2.19}$$

has to be solved. This can be accomplished using the Newton–Raphson method, i.e.

$$x_i^{l,k+1} = x_i^{l,k} + \Delta x_i^{l,k}, \tag{2.20}$$

with

$$\left(\gamma B_i \frac{\partial g}{\partial x_i}(t, x^{l,k}) + \frac{\partial f_i}{\partial x_i}(t, x^{l,k})\right) \Delta x_i^{l,k} = -B_i(\gamma\, g(t, x^{l,k}) + r) - f_i(t, x^{l,k}). \tag{2.21}$$

Since the p equations can now be solved independently, it is possible to exploit the different rates of activity of the individual subsystems.

The waveform relaxation algorithm, on the other hand, can be viewed as an extension of the Gauss–Jacobi or Gauss–Seidel iteration to function spaces [WOSR85]. That is, rather than vectors as in the linear and nonlinear case, the unknowns are now functions or so-called waveforms on the interval $[0, T]$. The relaxation is directly applied to the system of differential-algebraic equations and reduces the problem of solving the system (2.12) to the problem of solving p subsystems

$$\begin{aligned} 0 &= B_i\, \dot{y}^l + f_i(t, x^l), \\ 0 &= y_i^l - g_i(t, x^l). \end{aligned} \tag{2.22}$$

The individual subsystems are then solved over the whole time interval by means of standard simulation techniques, e.g. implicit multi-step schemes and Newton–Raphson based methods, while the coupling between the subsystems is handled with relaxation methods [LRS82]. The decomposition allows latency to be exploited in a very natural way since each subsystem can be integrated with its own step-size. It was shown that, if the circuit comprises strong feedback loops, the required number of iterations is proportional to the length of the interval [SN90]. Therefore, the interval $[0, T]$ is usually subdivided into smaller time intervals $[0, T_1]$, $[T_1, T_2]$, $\ldots$, $[T_{n-1}, T_n]$.

2.3.3 Further solution techniques

Although the relaxation-based approach is often much faster than the conventional approach, standard circuit simulators usually apply direct methods. This is not only due to the limited fields of application of iterated timing analysis and waveform relaxation, but also due to some lack of accuracy, robustness, and reliability [GFtM05]. Numerous attempts have been made to improve the performance of the conventional approach. The proposed techniques include partitioning and parallelization strategies, multirate integration schemes, and hierarchical methods.

Parallel simulation

In order to enable an efficient parallel simulation of a circuit, the communication between different processors has to be minimized. That is, the circuit needs to be decomposed into weakly coupled subcircuits. At the same time, the individual subcircuits should be of the same computational complexity so that the workload of the processors is evenly balanced. The resulting partitioning problem is NP complete [GJS76, GJ79]. Different heuristics have been developed to efficiently compute appropriate partitions. For a detailed description of partitioning methods tailored to integrated circuits, we refer to [FSF97, FRWZ98, Frö02] and references therein.

Subsequent to the partitioning, each subcircuit can be assigned to a different processor. After the time discretization of the circuit equations, the resulting nonlinear systems can be solved in parallel using so-called multi-level Newton–Raphson methods [WZ96, HRV01, Hon02]. Let again p be the number of subsystems, then the system of nonlinear equations $f(x) = 0$ can be decomposed into

$$\begin{aligned} f_i(x_i, x_E) &= 0, \quad i = 1, \ldots, p, \\ f_E(x_1, \ldots, x_p, x_E) &= 0, \end{aligned} \tag{2.23}$$

where the functions f_i represent the individual subsystems of the circuit and f_E the coupling between these subsystems. The vectors x_i contain the internal unknowns of the respective subsystem, the vector x_E consists of the external unknowns of the interconnect network. The Jacobian J of the partitioned function has the bordered block-diagonal form

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & & & & \frac{\partial f_1}{\partial x_E} \\ & \frac{\partial f_2}{\partial x_2} & & & \frac{\partial f_2}{\partial x_E} \\ & & \ddots & & \vdots \\ & & & \frac{\partial f_p}{\partial x_p} & \frac{\partial f_p}{\partial x_E} \\ \frac{\partial f_E}{\partial x_1} & \frac{\partial f_E}{\partial x_2} & \cdots & \frac{\partial f_E}{\partial x_p} & \frac{\partial f_E}{\partial x_E} \end{bmatrix}. \tag{2.24}$$

The single subsystems can now be solved independently in an inner iteration loop. Afterwards, an outer iteration loop is performed to take into account the interconnection structure. In this way, it is also possible to exploit spatial latency during the iteration process.

The method described above can be extended to an arbitrary number of levels. Nevertheless, usually only two levels are used. The speedup of the parallel simulation depends strongly on the circuit structure and the ability of the partitioning methods to generate a suitable decomposition [GFtM05].

Multirate simulation

During the simulation of highly integrated circuits, only a few elements underlie changing signals, whereas the major part – in general up to 80 % or even 90 % – remains latent [ROTH89, Kao92, CFJS04, GFtM05]. Standard methods discretize the entire circuit with a single step size which is mainly limited by the accuracy requirements of the rapidly changing subcircuits. It is of a particular interest to speed up the simulation without a significant loss of accuracy. By exploiting the latency of the system, only a fraction of the equations has to be generated and solved at a given time point.

Using waveform relaxation, multirate behavior can be exploited directly since the subsystems are decoupled and each system can be integrated with its own step size. If, however, direct methods are used, then the individual subsystems are not decoupled. Different multirate integration strategies have been developed to overcome this limitation. Multirate multi-step methods, for instance, can be found in [GW84, Ske89, KK99, VTB+08]. The most multirate one-step methods so far were tailored to ordinary differential equations [GFtM05]. Recently, also multirate one-step methods for differential-algebraic equations have been proposed in [SG04, SG05, SBG09]. Multirate one-step integration schemes

for ordinary differential equations and signal-flow based methods to exploit the inherent latency will be discussed in Chapter 4.

Hierarchical simulation

A further possibility to speed up the transient simulation is to exploit the given hierarchical circuit structure [Wan02, TFC+03, Cad04, CTCK04]. Integrated circuits are in general composed of subcircuits which in turn are again composed of smaller subcircuits. This inherent subcircuit hierarchy is usually flattened prior to the simulation. Hierarchical simulation methods exploit the nested circuit structure in order to avoid the storing and recomputation of multiple instances of the same subcircuit.

The hierarchical circuit representation can be directly generated from the netlist. Each subcircuit definition including all parameters is only stored once. Instances of the same subcircuit share a single implementation. Since many subcircuits are usually used several times, it is possible to decrease the memory requirements considerably. If instances of the same subcircuit furthermore share a common dynamic state, then isomorphism matching techniques can be applied to skip identical computations. That is, instead of recomputing dynamically equivalent subcircuits, previously computed results are reused. Since in particular digital circuit components exhibit only very few different stable states, hierarchical methods potentially result in a substantially reduced runtime.

3

Approximate operating point analysis

In this chapter, we will describe a graph-based and event-driven algorithm to compute an appropriate initial guess for the operating point analysis. The proposed method is based on the switch-level model introduced by Bryant [Bry81, Bry84]. That is, we utilize the coarser but much faster simulation at the switch level in order to compute an approximate solution at the circuit level. The aim is to provide the Newton–Raphson method with a starting point which is sufficiently close to a solution such that the iteration converges quickly. In this way, it is possible to speed up the operating point analysis significantly.

3.1 Benchmark circuits

All algorithms that we will present in this chapter have been integrated into an industrial circuit simulator. To illustrate the impact of our approach, we will analyze and simulate several different circuits which were provided by a chip manufacturer. Due to the complexity of modern integrated circuits, a detailed simulation of the whole circuit is sometimes impossible or at least extremely time-consuming. To deal with this problem, usually only parts of the circuit such as single word- or bitlines are extracted and simulated in detail [Frö02]. The set of benchmark circuits contains cross sections of memory

Table 3.1: Characteristics of the benchmark circuits.

	Modules	Nodes	Resistors	Capacitors	MOSFETs
$\mathfrak{C}_1$	162	85	1	1	154
$\mathfrak{C}_2$	666	248	34	158	460
$\mathfrak{C}_3$	1 983	1 006	1	10	1 911
$\mathfrak{C}_4$	7 036	3 075	844	1 525	4 598
$\mathfrak{C}_5$	8 812	3 610	119	920	7 651
$\mathfrak{C}_6$	21 439	9 015	1 310	4 127	14 070
$\mathfrak{C}_7$	21 681	13 679	4 498	844	9 788
$\mathfrak{C}_8$	22 685	9 568	1 419	4 227	15 028
$\mathfrak{C}_9$	22 727	9 599	1 417	4 227	15 090
$\mathfrak{C}_{10}$	22 780	3 426	0	17 510	5 196
$\mathfrak{C}_{11}$	52 715	26 097	14 566	15 530	15 628
$\mathfrak{C}_{12}$	71 737	39 163	18 231	31 105	13 607
$\mathfrak{C}_{13}$	71 737	30 811	18 231	31 105	16 391
$\mathfrak{C}_{14}$	230 118	118 118	80 335	121 399	23 028
$\mathfrak{C}_{15}$	367 557	143 906	92 800	149 036	125 670

chips but also converter and large logic circuits including parasitics. Table 3.1 shows the characteristics of these circuits.

3.2 Signal-flow analysis of integrated circuits

To begin with, we introduce methods to analyze the signal flow of integrated circuits. The aim is to gain insight into the structure and functionality of the circuit and to predict the influence of signal changes. The actual signal flow of complicated integrated circuits can be determined accurately only by a detailed simulation. Nevertheless, a good approximation of the signal flow can be obtained by analyzing the circuit topology and the characteristic properties of the different module types [DOR87].

A concept closely related to the signal flow is the directionality. The gate of a MOSFET, for instance, can be regarded as a unidirectional control of the bidirectional current flow in the channel. This simplifying assumption motivates the partitioning of the circuit into channel-connected components [Bry81]. Channel-connected components are subcircuits which are strongly coupled inside but only loosely coupled to other parts of the circuit. With the aid of this decomposition, we will construct a directed graph which describes

the logic signal flow of the circuit. The graph will be used later on to improve the results of the subsequent switch-level simulation.

A different approach to determine the signal flow of integrated circuits is discussed in Chapter 4. In contrast to the topological approach proposed here, a method which exploits properties of the corresponding circuit equations will be presented.

3.2.1 Channel-connected components

The decomposition of MOS circuits into channel-connected components was developed in the early eighties for both the circuit-level and the switch-level simulation [DOR87]. The approach is based on the fact that, neglecting the Miller effect, the gate of a MOSFET is isolated from the channel. To enable the analysis of arbitrary circuits, we use a slightly generalized definition of a channel-connected component.

Definition 3.1 (Channel-connected component) A *channel-connected component* or *CCC* is defined to be a maximal set of modules such that each two modules of this set are connected by a path which exclusively consists of statically conducting modules and drain-source connections.

Now, we have to differentiate the different module types into conducting and nonconducting elements. We use the following classification: Resistors whose resistance is smaller than a predefined threshold R_{th}, inductors, and voltage sources that are used as ammeters are defined to be bidirectionally conducting. Diodes, which let the current pass in one direction and block it in the opposite direction, are defined to be unidirectionally conducting. Statically nonconducting are, for instance, capacitors and current sources. Controlled sources and resistors are treated like the equivalent independent two-port modules. The controlling nodes or elements are regarded as unidirectional inputs. As described above, the channel of a MOSFET is defined to be bidirectionally conducting, unidirectionally controlled by the gate node. The influence of the bulk node is negligible for the decomposition into channel-connected components.

The topology of a circuit $\mathfrak{C}$ can be viewed as a multi-hypergraph. Each module is represented by a hyperedge and each node by a vertex. Let $\mathfrak{M} = \{\mathfrak{m}_1, \ldots, \mathfrak{m}_m\}$ be the set of modules and $\mathfrak{N} = \{\mathfrak{n}_1, \ldots, \mathfrak{n}_n\}$ the set of nodes. Analogously to the notation $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$ for directed graphs, we write $\mathfrak{C} = (\mathfrak{N}, \mathfrak{M})$. The decomposition of a circuit $\mathfrak{C}$ into channel-connected components can be computed efficiently using a depth-first search along the conducting paths of the circuit graph and is thus of linear complexity.

Prior to the partitioning process, the graph has to be modified in order to avoid the coupling of components via input nodes or parasitic elements. Since each transistor is in general connected to either ground or a supply voltage source by a path of conducting channels, we have to split these nodes. That is, the depth-first search has to be stopped whenever such a node is visited. Furthermore, no effect can be transmitted from one node to another through input nodes [Bry87]. Hence, all input nodes and additionally all nodes that are connected to these nodes via paths of parasitic resistors are split in the same way.

After these preparative steps, the channel-connected components can be computed with the aid of the modified circuit graph. However, not all modules can be assigned to a channel-connected component. Resistors with $R > R_{th}$, capacitors, and current sources, for instance, which are defined to be statically nonconducting, will not be visited during the depth-first search.

3.2.2 Component graph

The decomposition of the circuit into channel-connected components can be used to generate a model of the logic signal flow. The gate nodes of the MOSFETs and the controlling nodes of the voltage-controlled elements can be regarded as inputs of the components. Depending on the states of these input nodes, the channel-connected components will either propagate or block signals. This consideration can be utilized to generate a directed graph with the channel-connected components being the vertices and the connections between the components being the edges, as described in [DOR87].

To model the influence of the primary inputs on the channel-connected components, we add a vertex for each signal-input voltage source and directed edges from the vertex to each channel-connected component which contains modules that are controlled by the voltage source. We call this graph the *component graph* $\mathfrak{G}_c$ of the circuit. The component graph enables the analysis of the influence of signal changes and the detection of feedback loops.

To sum up, the following steps are necessary to obtain a model of the logic signal flow of the circuit:

1. Split all nodes that are reachable from ground via paths of voltage sources and parasitic resistors.

2. Partition the circuit into channel-connected components using a depth-first search along the conducting paths.

3. Generate the component graph:
   a) Add a vertex $\mathfrak{v}_i$ for each channel-connected component CCC_i and edges $(\mathfrak{v}_j, \mathfrak{v}_i)$ for each module $\mathfrak{m} \in \mathrm{CCC}_i$ whose input belongs to CCC_j.
   b) Add a vertex for each signal-input voltage source and edges to each channel-connected component of the adjacent circuit elements.

The component graph of a circuit contains in general cycles or feedback loops. Each component of such a configuration potentially depends on the results of all other components. This strong interdependency can lead to oscillations or large undefined regions during the subsequent switch-level simulation. To identify these critical configurations, we compute the strongly connected components of the graph $\mathfrak{G}_c$, as described in Section 2.2.

Example 3.2

i) Consider the clock generation circuit taken from [Bak08]. This circuit converts a given clock signal CLK into two nonoverlapping clock signals ϕ_1 and ϕ_2 such that logically $\phi_1 \, \phi_2 = 0$. It comprises two NAND gates and seven inverters. Figure 3.1a shows the transistor-level schematic of the circuit and the decomposition into channel-connected components, Figure 3.1b the resulting component graph. Each vertex label consists of the component number or name and the number of contained circuit elements. Input nodes are marked by a double circle.

ii) Figure 3.2a shows the layout of a 4-bit ripple-carry adder. The circuit consists of four full adders and computes the sum $s = [s_3 \, s_2 \, s_1 \, s_0]$ of two binary numbers $a = [a_3 \, a_2 \, a_1 \, a_0]$ and $b = [b_3 \, b_2 \, b_1 \, b_0]$. The values $c_0, \ldots, c_4$ are the carry bits. Figure 3.2b shows the implementation of a full adder using NAND gates. The resulting component graph of the 4-bit adder is depicted in Figure 3.2c. Each NAND gate forms a channel-connected component of size four. Hence, the structure of the component graph closely resembles the gate-level description of the circuit. Note that the orientation of the component graph is inverted, that is, the lowest bit is on the left-hand side. The component graph of the 4-bit adder is acyclic.

iii) The component graph of circuit $\mathfrak{C}_3$ is shown in Figure 3.3. The graph comprises 466 channel-connected components and 19 nontrivial strongly connected components. ♢

The example illustrates that even the signal flow of comparably small circuits can be rather complex. Large-scale integrated circuits usually consist of thousands of different channel-connected components. Circuit $\mathfrak{C}_{15}$, for instance, comprises 39 692 channel-connected and 17 804 nontrivial strongly connected components.

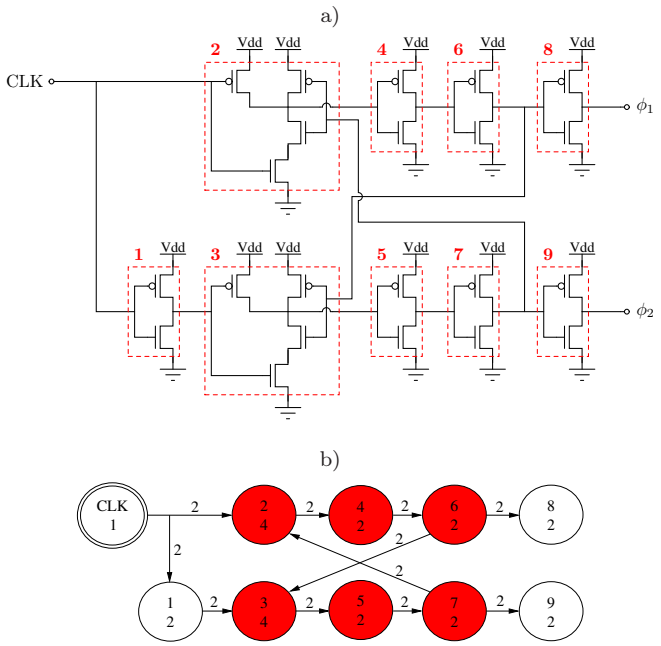


Figure 3.1: Two-phase nonoverlapping clock generation circuit. a) Decomposition into channel-connected components. b) Resulting component graph $\mathfrak{G}_c$. The six red marked vertices form a strongly connected component.

3.3 Switch-level simulation

As described in Section 2.1, a MOSFET can be regarded as a voltage-controlled switch. The voltage at the gate terminal or the corresponding digital value, respectively, controls the connection between drain and source. If the MOSFET is on, then signals flow both from source to drain and from drain to source. Hence, a CMOS circuit can be modeled as a network of nodes connected by ideal switches. If the switch is turned on, then the transistor connects the source and drain node. If, on the other hand, the switch is turned off, then the two nodes are isolated [Mic03]. This bidirectional signal flow of CMOS circuits could not be modeled accurately by existing gate-level simulators and led to the development of switch-level simulators [ROTH89]. The first switch-level techniques were proposed independently by Bryant [Bry81, Bry84] and Hayes [Hay82, Hay84] and enabled the simulation of large MOS circuits without the expense of the time-consuming circuit-level analysis.

The switch-level algorithm developed by Bryant combines methods from graph theory and Boolean algebra and bears several similarities to the model of Hayes. Below, we

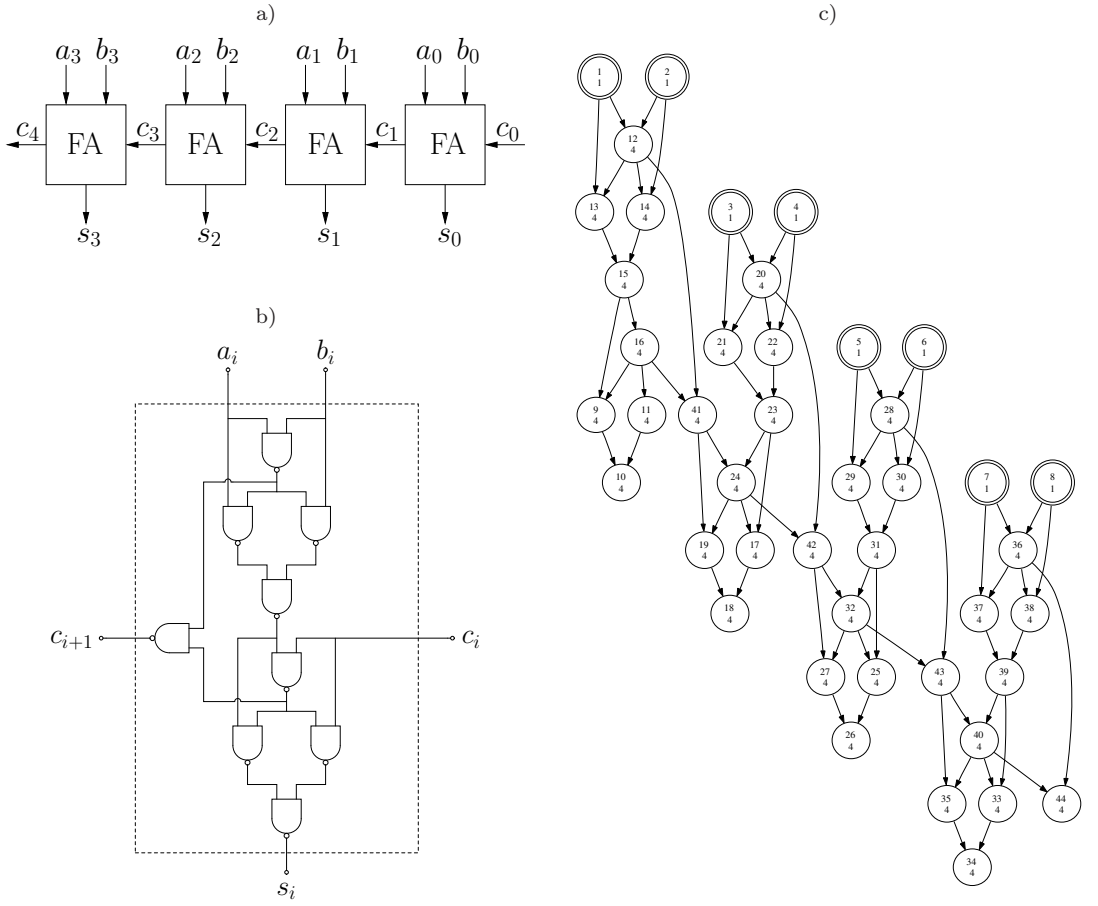


Figure 3.2: 4-bit ripple-carry adder. a) Schematic circuit diagram. b) Gate-level model of a single full adder. c) Component graph $\mathfrak{G}_c$.

present Bryant's switch-level model in detail. Our extended switch-level simulation is a generalization of the standard switch-level simulation in the direction of circuit-level simulation.

The abovementioned decomposition can be used to improve the performance of the switch-level simulation since the signal propagation can be restricted to the respective channel-connected components. Most switch-level simulators decompose the circuit in this way [ROTH89]. The partitioning can be performed either statically prior to the simulation or dynamically during the simulation where additional information on the states of the transistors and nodes can be exploited. The dynamic partitioning results in general in much smaller components and can be used to avoid the reevaluation of inactive regions [Bry87].

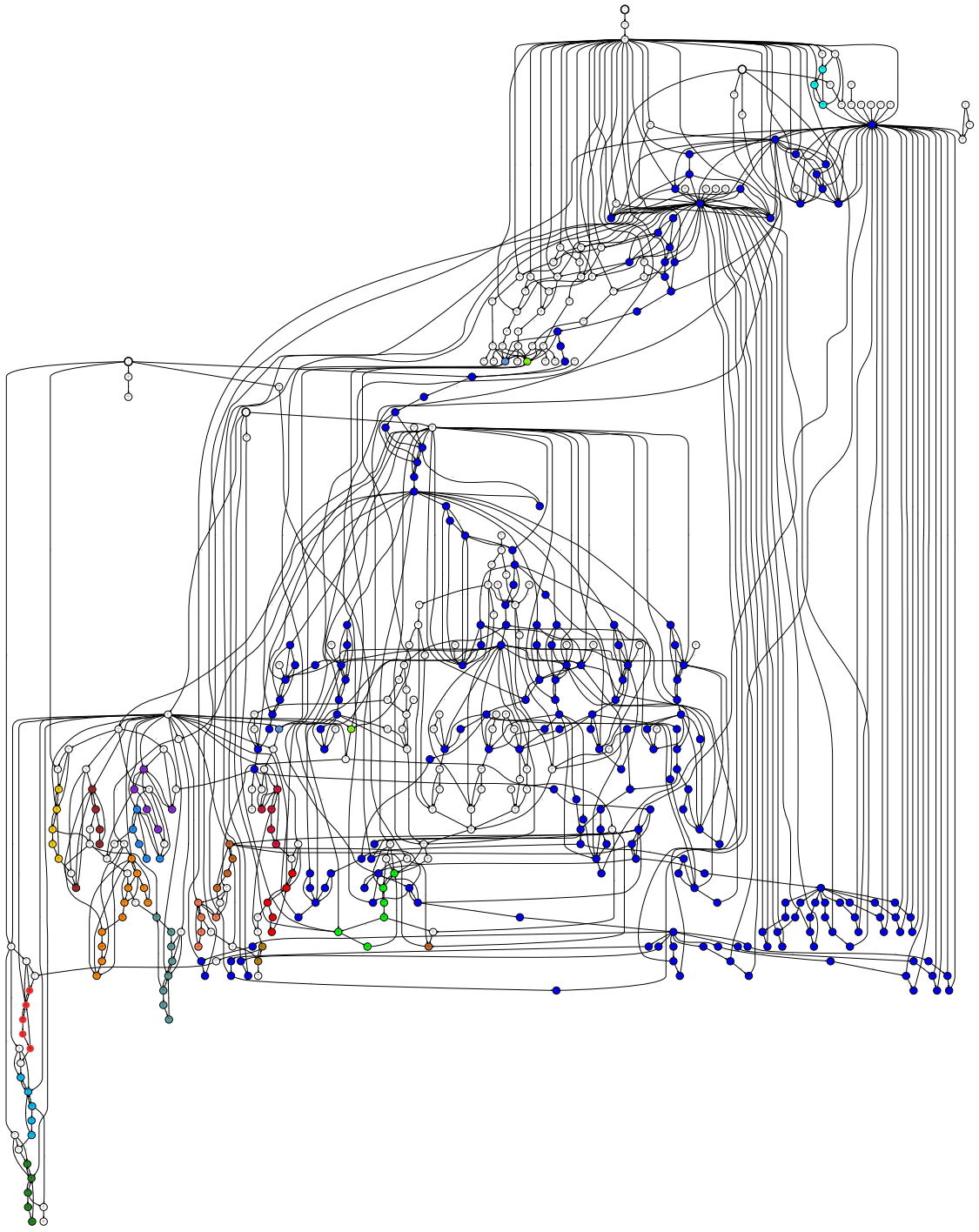


Figure 3.3: Component graph $\mathfrak{G}_c$ of circuit $\mathfrak{C}_3$. Each strongly connected component is displayed in a different color.

3.3.1 The basic network model

Given a circuit $\mathfrak{C} = (\mathfrak{N}, \mathfrak{M})$ consisting of a set of nodes $\mathfrak{N} = \{\mathfrak{n}_1, \ldots, \mathfrak{n}_n\}$ and a set of modules $\mathfrak{M} = \{\mathfrak{m}_1, \ldots, \mathfrak{m}_m\}$, we want to describe the behavior of the circuit in terms of node states $y = (y_1, \ldots, y_n)$ and module states $z = (z_1, \ldots, z_m)$. In contrast to the classical switch-level simulation which is restricted to transistor networks, we allow arbitrary integrated circuits.

Let us be more precise. For each node $\mathfrak{n}_i \in \mathfrak{N}$, we define a state $y_i \in \{0, 1, X, Z\}$, where 1 represents the positive supply voltage V_{dd}, 0 the ground voltage, and X *invalid*, respectively. Additionally, a state Z denoting *initial unknown* is introduced. This state is useful to identify parts of the circuit that are not driven or properly initialized [Dav91].

We have to distinguish between input nodes and storage nodes. Nodes which are connected to supply voltage sources, signal-input voltage sources, or voltage-controlled voltage sources are defined to be input nodes, all remaining nodes are, by definition, storage nodes. Input nodes are assumed to supply unlimited current to the circuit and the provided signal cannot be overwritten by other signals. The state of the storage nodes, on the other hand, is determined by the states of adjacent nodes and modules. To model different signal strengths, we assign each node a *size*. The different sizes have no property other than their ordering [Bry84]. Each input node is assigned the size ω, each internal node a size from the set $\mathcal{K} = \{\kappa_1, \ldots, \kappa_{\max}\}$. Define size : $\mathfrak{N} \mapsto \mathcal{K} \cup \{\omega\}$ to be the function which returns the size of a node.

Moreover, we define a state $z_j \in \{0, 1, X\}$ for each module $\mathfrak{m}_j \in \mathfrak{M}$. Here, 0 denotes *nonconducting*, 1 *conducting*, and X *indeterminate*. The state of a module can be fixed or determined by the states and voltages of controlling nodes or elements. A resistor, for example, is – depending on the resistance – defined to be either conducting or nonconducting. The state of a transistor, on the other hand, depends on the state of the gate node and the transistor type.

Define $\mathfrak{T} \subseteq \mathfrak{M}$ to be the set of all transistors and type : $\mathfrak{T} \mapsto \{p_e, n_e, p_d, n_d, p_c, n_c\}$ to be the function which specifies the transistor type. We distinguish again between n-channel and p-channel MOSFETs and additionally between *enhancement-type*, *depletion-type*, and *capacitor-type* MOSFETs. A depletion-type MOSFET is almost identical to the corresponding enhancement-type MOSFET with one important difference: the conducting channel is not induced but rather physically implanted [Bal03]. That is, the depletion-type transistors possess a conducting channel even for $v_{gs} = 0$. For the switch-level simulation, depletion-type MOSFETs are assumed to be always conducting. This

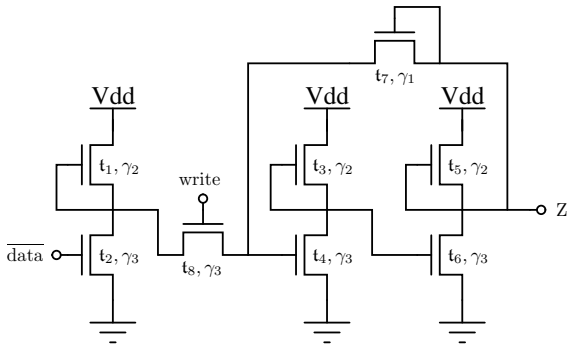


Figure 3.4: Static RAM cell in nMOS technology.

transistor type can be used to model resistors [Lew89], for example. Capacitor-type MOSFETs are defined to be transistors with source and drain tied together. Now, let $\delta : \{0, 1, X, Z\} \times \{n_e, p_e, n_d, p_d, n_c, p_c\} \mapsto \{0, 1, X\}$ be the function that describes the relation between the gate state, the transistor type, and the resulting state of the MOSFET, i.e.

δ	n_e	p_e	n_d	p_d	n_c	p_c
0	0	1	1	1	0	0
1	1	0	1	1	0	0
X	X	X	1	1	0	0
Z	X	X	1	1	0	0

(3.1)

Similar to the size of the nodes, each conducting module is defined to have a *strength* from the set $\Gamma = \{\gamma_1, \ldots, \gamma_{\max}\}$ which describes the relative conductance compared to other modules. Let strength : $\mathfrak{M} \mapsto \Gamma$ be the function that assigns each module its conductance.

Depending on the characteristics of the circuit, the sets $\mathcal{K}$ and Γ can be arbitrarily large. However, in general a few different strengths suffice to describe the switching behavior.

Example 3.3 Figure 3.4 shows a static RAM cell [Bry84, Mic03]. Here, three different transistor strengths are required. To ensure the correct functioning of the nMOS inverters, the strength of the enhancement-type transistors has to be larger than the strength of the depletion-type transistors. Additionally, the feedback loop contains a high-resistance transistor with the strength γ_1 to guarantee that previously stored values can be overwritten by new data which has either strength γ_2 or γ_3. ♢

The dynamic behavior of the network will be described with the aid of the excitation function, which can be defined in terms of the steady-state response function [Bry84].

The conducting channel of the different module types is regarded as a nonlinear resistor whose resistance is controlled by the voltages of the controlling nodes. Assuming that the resistances are fixed and can be controlled independently of the controlling voltages, the circuit represents a network of passive elements with a unique steady-state solution. Define F to be the steady-state response function which computes the steady-state solution for a set of fixed resistances and initial node voltages, but in the digital domain, using the node and transistor states. That is, for given vectors y and z the steady-state response $y' = F(y, z)$ contains the resulting node states under the assumption that the states of the modules are fixed. If the states of the modules are set according to the states of the controlling nodes given by the vector y, i.e. $z = z(y)$, then the function E with

$$E(y) = F(y, z(y)) \tag{3.2}$$

is defined to be the excitation function. Most switch-level simulators use iterative methods to compute the steady-state response [ROTH89]. At each iteration step, the signals are propagated along the conducting channels to the adjacent nodes. Then the signals are compared to the current node signals. If a node state changes, a new event is generated and enqueued. This procedure is repeated until convergence is reached [Bry87].

With the aid of the excitation function, the switch-level simulation can be implemented as follows: Given initial node states y, the excitation states are computed repeatedly until a stable steady state is reached, i.e.

$$y' = \lim_{k \to k_{\max}} E^k(y). \tag{3.3}$$

If no stable state can be reached after $k_{\max}$ steps, for example on account of oscillations, some nodes should be set to X [Mic03].

Below, we present a formal, graph-based definition of the excitation function and an iterative algorithm to compute the steady-state solution.

Definition 3.4 (Signal) Let $\Sigma = \mathcal{K} \cup \Gamma \cup \{\omega\}$ be the set of signal strengths with the ordering $\kappa_1 < \cdots < \kappa_{\max} < \gamma_1 < \cdots < \gamma_{\max} < \omega$, and let $S = \{0, 1, X, Z\} \times \Sigma$. Then a *signal* is defined to be a pair $s = (y_s, \sigma_s) \in S$.

Now, we have to define functions which handle and propagate the signals.

Definition 3.5 (Propagation functions) Let s_1 and s_2 be two signals. Define the *propagation functions* $\bullet : \Gamma \times S \mapsto S$ and $\sqcup : S \times S \mapsto S$ by

$$\gamma \bullet s_1 = (y_{s_1}, \min(\gamma, \sigma_{s_1})) \tag{3.4}$$

and

$$s_1 \sqcup s_2 = \begin{cases} s_1, & \text{if } \sigma_{s_1} > \sigma_{s_2} \vee s_1 = s_2, \\ s_2, & \text{if } \sigma_{s_1} < \sigma_{s_2}, \\ (X, \sigma_{s_1}), & \text{if } \sigma_{s_1} = \sigma_{s_2} \wedge y_{s_1} \neq y_{s_2}, \end{cases} \tag{3.5}$$

as described in [Lew89].

Since arbitrary user-defined module types are permitted – each module is assumed to provide its own behavioral model –, a conducting connection can be established between any number of terminal nodes. Let $\mathcal{P}(\mathfrak{N})$ denote the power set of $\mathfrak{N}$. Define $\text{cond} : \mathfrak{M} \mapsto \mathcal{P}(\mathfrak{N})$ to be the function that returns the set of nodes which are connected by a conducting channel. For a conducting transistor $\mathfrak{t}_j \in \mathfrak{T}$, for instance, $\text{cond}(\mathfrak{t}_j) = \{\text{source}(\mathfrak{t}_j), \text{drain}(\mathfrak{t}_j)\}$.

Definition 3.6 (Conducting path) A sequence $p = (\mathfrak{n}_0, \mathfrak{m}_1, \mathfrak{n}_1, \ldots, \mathfrak{m}_l, \mathfrak{n}_l)$ with $\mathfrak{n}_i \in \mathfrak{N}$, $i = 0, \ldots, l$, and $\mathfrak{m}_j \in \mathfrak{M}$, $j = 1, \ldots, l$, is said to form a *conducting path* of length l if

$$z(\mathfrak{m}_j) = 1 \quad \text{and} \quad \{\mathfrak{n}_{j-1}, \mathfrak{n}_j\} \subseteq \text{cond}(\mathfrak{m}_j) \quad \forall j = 1, \ldots, l. \tag{3.6}$$

Define $\text{root}(p) = \mathfrak{n}_0$ and $\text{dest}(p) = \mathfrak{n}_l$.

In contrast to a conventional path, a conducting path depends not only on the network structure but also on its current state.

Definition 3.7 (Path strength) Let P_c be the set of all conducting paths. For a path $p = (\mathfrak{n}_0, \mathfrak{m}_1, \mathfrak{n}_1, \ldots, \mathfrak{m}_l, \mathfrak{n}_l) \in P_c$, define

$$|p| = \min\Big(\text{size}(\mathfrak{n}_0), \min_{j=1,\ldots,l}\big(\text{strength}(\mathfrak{m}_j)\big)\Big) \tag{3.7}$$

to be the *path strength*.

The path strength is an approximation of the charge that can be supplied along the path [Bry84]. The strength of a conducting path from an input node is determined by the minimum transistor strength. The strength of a path from a storage node, on the other hand, is determined by the size of the storage node [ROTH89]. Input nodes can be regarded as unbounded charge sources which are able to overwrite the limited amount of charge supplied by storage nodes [Lew89]. Thus, some paths arriving at a node will be blocked by stronger paths.

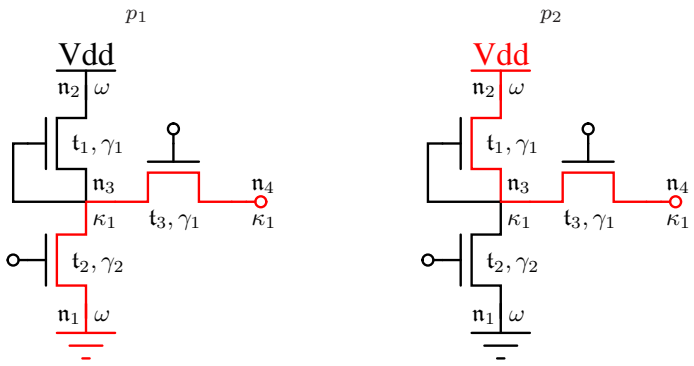


Figure 3.5: Examples of conducting paths.

Definition 3.8 (Blocked path) A path $p = (\mathfrak{n}_0, \mathfrak{m}_1, \mathfrak{n}_1, \ldots, \mathfrak{m}_l, \mathfrak{n}_l) \in P_c$ is said to be *blocked* at $\mathfrak{n}_j$, if a path $p' = (\mathfrak{n}'_0, \mathfrak{m}'_1, \mathfrak{n}'_1, \ldots, \mathfrak{m}'_i, \mathfrak{n}'_i) \in P_c$ exists such that

$$\exists j : \mathfrak{n}_j = \mathfrak{n}'_i \quad \text{and} \quad |p'| > |(\mathfrak{n}_0, \mathfrak{m}_1, \mathfrak{n}_1, \ldots, \mathfrak{m}_j, \mathfrak{n}_j)|. \tag{3.8}$$

That is, a conducting path is blocked if there exists a stronger path to one of its nodes. To determine the new state of a node, only the unblocked paths have to be taken into account. The following example, taken from [Bry84, Mic03], illustrates the above definition.

Example 3.9 Figure 3.5 shows the unblocked path p_1 and the blocked path p_2 with $\text{dest}(p_1) = \mathfrak{n}_4$ and $\text{dest}(p_2) = \mathfrak{n}_4$. Even though the paths p_1 and p_2 have the same strength, p_2 is blocked at node $\mathfrak{n}_3$ by p_1 since the subpath from ground to node $\mathfrak{n}_3$ has strength γ_2. ♢

Moreover, the example demonstrates that the signals with the largest signal strength have to be propagated first. If we propagate the signal from V_{dd} first, then $\mathfrak{n}_3$ and $\mathfrak{n}_4$ are set to $(1, \gamma_1)$. The signal from ground overwrites the previous signal at $\mathfrak{n}_3$ by $(0, \gamma_2)$. This signal is – attenuated by $\mathfrak{t}_3$ – passed as $(0, \gamma_1)$ to node $\mathfrak{n}_4$, where the two contradictory states result in an undefined output. If we, on the other hand, propagate the signal from ground first, then the signal from V_{dd} is blocked at node $\mathfrak{n}_3$ and the output is set to $(0, \gamma_1)$. The inadvertent propagation of wrong signals can be avoided using a priority queue which is ordered according to the signal strengths [SN90].

Definition 3.10 (Excitation function) Let P_u be the set of all unblocked paths and $P_u(\mathfrak{n}_i) = \{p \in P_u \mid \text{dest}(p) = \mathfrak{n}_i\}$. Then the *excitation function* E is defined to assign each

node $\mathfrak{n}_i \in \mathfrak{N}$ a new state y_i as follows:

$$y_i = \begin{cases} 0, & \text{if } \forall p \in P_u(\mathfrak{n}_i) : y(\text{root}(p)) = 0, \\ 1, & \text{if } \forall p \in P_u(\mathfrak{n}_i) : y(\text{root}(p)) = 1, \\ X, & \text{if } \exists p_1, p_2 \in P_u(\mathfrak{n}_i) : y(\text{root}(p_1)) \neq y(\text{root}(p_2)), \\ Z, & \text{otherwise.} \end{cases} \tag{3.9}$$

Subsequently, the states of the modules have to be updated. The new state of a transistor $\mathfrak{t}_j \in \mathfrak{T}$, for instance, is given by $z(\mathfrak{t}_j) = \delta\big(y(\text{gate}(\mathfrak{t}_j)), \text{type}(\mathfrak{t}_j)\big)$. After the update of the module states, the excitation function can be evaluated again to compute the new node states. This procedure is repeated until a stable steady state is reached.

Initially, the state of all input nodes is set to the digital value which corresponds to the supplied voltage. The initial state of all storage nodes is defined to be Z. A detailed description of the switch-level simulation is given in Algorithm 3.1, cf. also [Lew89].

Algorithm 3.1 Basic switch-level simulation.

```
function Circuit::simulate(k_max)
    k = 0
    repeat
        for all m_j ∈ 𝔐 do
            m_j.updateState()
        end for
        for all n_i ∈ 𝔑 do
            s_i = (y_i^(k), size(n_i))
        end for
        active = 𝔑
        while active ≠ ∅ do
            choose the node n_i with the largest signal strength
            n_i.updateSignal()
            active = active \ {n_i}
        end while
        for all n_i ∈ 𝔑 do
            y_i^(k+1) = y_{s_i}
        end for
        k = k + 1
    until y^(k) = y^(k−1) ∨ k > k_max
end function
```

The function Node::updateSignal() notifies all adjacent modules $\mathfrak{M}[\mathfrak{n}_i]$ as described in Algorithm 3.2. Prior to each iteration, the function Module::updateState() determines

Algorithm 3.2 Notification of adjacent modules.

```
function Node::updateSignal()
    n_i = this
    for all m_j ∈ M[n_i] do
        m_j.updateSignal(n_i)
    end for
end function
```

the state of the modules according to the current node states. Module::updateSignal($\mathfrak{n}$) propagates the signals through the conducting channel to the adjacent nodes. Each module type has its own implementation of these functions, the implementation for MOSFETs is presented in Algorithm 3.3. In accordance with the C++ notation, *this* is defined to refer to the current object.

Algorithm 3.3 Update of the module state and signal propagation.

```
function MOSFET::updateState()
    m_j = this
    n_i = gate(m_j)
    z_j^(k) = δ(y_i^(k), type(m_j))
end function

function MOSFET::updateSignal(n_i)
    m_j = this
    if z_j^(k) = 1 ∧ ∃ n_o ∈ N : {n_i, n_o} = {source(m_j), drain(m_j)} then
        ŝ = s_o ⊔ (strength(m_j) • s_i)
        if ŝ ≠ s_o then
            s_o = ŝ
            active = active ∪ {n_o}
        end if
    end if
end function
```

Since the switch-level model was primarily tailored to pure transistor networks, the algorithm presented above yields appropriate results only if the circuit is mainly digital. Static CMOS circuits, for example, can be simulated efficiently and reliably using the basic network model.

Example 3.11 To compute the steady state of circuit $\mathfrak{C}_1$, the switch-level algorithm requires 32 iterations. After the simulation only one node remains undefined. The average difference between the approximated and the actual operating point per node amounts to

approximately $\Delta v = 1.1 \cdot 10^{-9}\,\mathrm{V}$. Using the standard Newton–Raphson method without a previous switch-level simulation, 634 iterations are needed to compute an operating point. If we, on the other hand, use the initial guess provided by the switch-level model, then 26 iterations are sufficient. ♢

3.3.2 The extended network model

If the circuit contains analog or mixed-signal components, then the basic switch-level algorithm will in general not produce reliable results. Therefore, we extend the capabilities of switch-level simulation in the direction of circuit-level simulation in order to enable the approximate operating point analysis of a larger class of integrated circuits.

The first modification is that rather than updating all modules only after each iteration, the state of a module is directly updated if the states of the controlling nodes change. As a result, the number of iterations and accordingly the runtime of the switch-level simulation can be reduced considerably. Moreover, for some of the benchmark circuits, the modified algorithm converges quickly while the standard algorithm results in oscillations.

Since integrated circuits usually work with different supply voltages, the state of a node is not sufficient to represent the corresponding voltage level. In addition to the state y_i of the node $\mathfrak{n}_i$, we now also take into account the corresponding node voltage v_i. That is, the behavior of the network is now described by the node states $y = (y_1, \ldots, y_n)$, the node voltages $v = (v_1, \ldots, v_n)$, and the module states $z = (z_1, \ldots, z_m)$. The voltages can also be used to dynamically compute the supplied voltage of voltage-controlled voltage sources and the strength of voltage-controlled resistors subject to the states and voltages of the controlling nodes.

A further drawback of the basic switch-level model is that it does not produce correct results for arbitrary connections of pass transistors if threshold voltage drops occur [SN90]. Pass transistors are often used to connect and disconnect different subcircuits to a common bus. If a pass transistor is enabled, then it transmits logic signals in both directions and blocks them otherwise. However, nMOS transistors are better suited for transmitting a low signal, while pMOS transistors are better suited for transmitting a high signal. The nMOS transistor passes a logic 1 with a so-called threshold voltage drop. Analogously, the pMOS transistor does not pass an ideal logic 0.

If now a pass transistor transmits a weak signal, then threshold voltage drops can be taken into account using adjusted voltages v_i. Nevertheless, the transmission of a weak logic signal to the inputs of other subcircuits might result in undefined behavior since,

depending on the characteristics of the MOSFETs, both the nMOS and the pMOS transistors might be in a conducting state. To cope with this problem, we replace the pure digital MOSFET model by the Shichman–Hodges model introduced in Section 2.1. With the aid of this analog model, we obtain an approximation of the continuous behavior. Furthermore, different threshold voltages and transistor strengths can be taken into account. This is, for instance, of advantage if a low-voltage signal in the high state is connected to an nMOS transistor whose threshold voltage is greater than the supplied voltage. Using only digital states, the switch-level model would propagate wrong results.

Moreover, we replace the finite set of strengths by arbitrary real-valued resistances. That is, each conducting module is now assigned a resistance. The MOSFET model is of particular importance for the quality of the approximate operating point. In order to estimate the resistance of a conducting channel, we use the following consideration [Bak08]: The channel of an nMOS transistor operating in the triode region can be interpreted as a resistor whose resistance R_{ch} can be approximated by

$$R_{ch}^{-1} = \frac{\partial \imath_{ds}}{\partial v_{ds}} = \beta_n(v_{gs} - v_{th,n}) - \beta_n v_{ds} = \beta_n(v_{ds,sat} - v_{ds}). \tag{3.10}$$

If $v_{ds,sat} \gg v_{ds}$, this can be simplified to

$$R_{ch}^{-1} \approx \beta_n(v_{gs} - v_{th,n}). \tag{3.11}$$

The resistance of pMOS transistors can be approximated in the same way. The advantage of this estimate is that we only need the gate and the source voltage – the source node is here always defined to be the source of the signal – to compute the resistance and not the drain voltage which is in general unknown, in particular at the beginning of the switch-level simulation.

Definition 3.12 (Signal) Define $S = \{0, 1, X, Z\} \times \mathbb{R} \times \mathbb{R}$. A *signal* is defined to be a tuple $s = (y_s, v_s, r_s) \in S$ which consists of a state y_s, a voltage v_s, and a resistance r_s.

The signal states are only used to distinguish between initialized and uninitialized nodes. For the extended network model, we can redefine the propagation functions as follows.

Definition 3.13 (Propagation functions) Let s_1 and s_2 be two signals. Define the *propagation functions* $\bullet : \mathbb{R} \times S \mapsto S$ and $\sqcup : S \times S \mapsto S$ by

$$r_p \bullet s_1 = (y_{s_1}, v_{s_1}, r_p + r_{s_1}) \tag{3.12}$$

and

$$s_1 \sqcup s_2 = \begin{cases} s_1, & \text{if } r_{s_1} < r_{s_2} \vee s_1 = s_2, \\ s_2, & \text{if } r_{s_1} > r_{s_2}, \\ (X, v_{s_1}, r_{s_1}), & \text{otherwise.} \end{cases} \tag{3.13}$$

Initially, the resistance of the ground node is set to $r = 0$, the resistance of all other nodes is set to $r = \infty$. Afterwards, the by definition ideally conducting voltage sources pass the supplied voltages from the ground node to the input nodes. As a result, we do not have to distinguish between input and storage nodes any longer. Furthermore, the modification renders the distinction between signal-input voltage sources and zero-valued voltage sources, which are frequently used to measure currents, unnecessary. The estimates of the resistances can now be used to determine the strength of conducting paths. Define r to be the function which assigns each module and node the corresponding resistance.

Definition 3.14 (Path resistance) Let $p = (\mathfrak{n}_0, \mathfrak{m}_1, \mathfrak{n}_1, \ldots, \mathfrak{m}_l, \mathfrak{n}_l) \in P_c$ be a conducting path. Then the *path resistance* is defined by

$$|p| = r(\mathfrak{n}_0) + \sum_{j=1}^{l} r(\mathfrak{m}_j). \tag{3.14}$$

The extended switch-level simulation is presented in Algorithm 3.4. Here, node $\mathfrak{n}_1$ is defined to be the ground node. The set *active* is a priority queue which is ordered according to the resistances of the signals.

The function Node::updateSignal notifies again all adjacent modules. The difference is that here the modified functions of the modules are called. Prior to the propagation of the signal, the current status of the module has to be evaluated. The implementation of this function for n-channel MOSFETs is described in Algorithm 3.5. The function isValid returns *true* if the state is 0 or 1 and *false* otherwise. The function a2d converts a voltage into the corresponding digital state.

3.3.3 Initialization of undefined subcircuits

It is in general not possible to compute valid states for all nodes. The node between the two n-channel MOSFETs of a NAND gate whose inputs are both zero, for instance, will remain indeterminate after the switch-level simulation since it is only connected to dynamically nonconducting modules. Large undefined regions can also be caused by sequential logic since in contrast to combinational logic the outputs of a sequential subcircuit do not only depend on the current input signals but also on previous input signals.

Algorithm 3.4 Extended switch-level simulation.

```
function Circuit::simulate(k_max)
    k = 0
    repeat
        for all n_i ∈ 𝔑 do
            s_i = (y_i^(k), v_i^(k), ∞)                    // reset all signal strengths
        end for
        s̃_1 = (0, 0, 0)                                    // initialize the ground node
        active.push(n_1, s̃_1)
        while active ≠ ∅ do
            (n_i, s̃_i) = active.top()
            ŝ_i = s_i ⊔ s̃_i
            if ŝ_i ≠ s_i then
                s_i = ŝ_i
                n_i.updateSignal()
            end if
            active.pop()
        end while
        for all n_i ∈ 𝔑 do
            y_i^(k+1) = y_{s_i}
            v_i^(k+1) = v_{s_i}
        end for
        k = k + 1
    until v^(k) = v^(k−1) ∨ k > k_max
end function
```

Example 3.15 Consider the edge-triggered D flip-flop [Bak08] shown in Figure 3.6. If CLK is low, then the transmission gates T_1 and T_4 are on while T_2 and T_3 are off. Thus, the signal D is passed to node $\mathfrak{n}_1$ and the complement $\overline{D}$ to node $\mathfrak{n}_2$. The second stage stores the previous output. If, on the contrary, CLK is high, then T_2 and T_3 are on while T_1 and T_4 are off. Consequently, the first stage captures the previous input and passes it to the second stage. Since in both cases these previous values do not exist, the output remains indeterminate. Furthermore, all subcircuits that depend on the result of this configuration will remain indeterminate. ♢

Definition 3.16 (Coverage) Define the *coverage* of the switch-level simulation to be the number of nodes that could be assigned a value divided by the number of all nodes. Analogously, define the *input coverage* of a channel-connected component to be the number of initialized inputs of the component divided by the number of all inputs.

Algorithm 3.5 Propagation of the signals along the conducting path.

```
function MOSFET::updateSignal(𝔫_i)
    𝔪_j = this
    if ∃ 𝔫_o ∈ 𝔑 : {𝔫_i, 𝔫_o} = {source(𝔪_j), drain(𝔪_j)} then
        𝔫_g = gate(𝔪_j)
        if isValid(y_g) ∧ isValid(y_i) ∧ v_g − v_i > v_th then
            ṽ = min(v_g − v_th,n, v_i)
            r̃ = r(𝔪_j) + r_i
            s̃ = s(a2d(ṽ), ṽ, r̃)
            active.push(𝔫_o, s̃)
        end if
    end if
end function
```

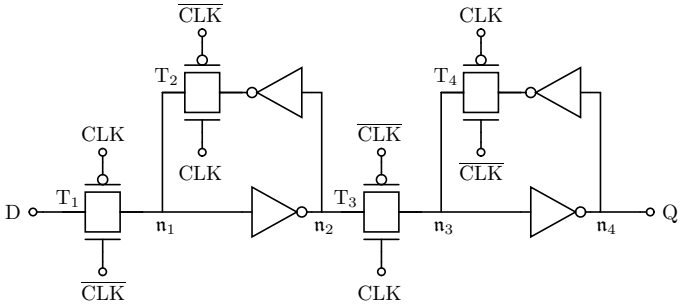


Figure 3.6: An edge-triggered D flip-flop.

To improve the coverage of the switch-level simulation and, as a consequence, the convergence of the nonlinear solver, we propose a method to initialize critical nodes of the circuit and to compute valid states for all subcircuits that depend on these critical nodes. For this purpose, we utilize the component graph $\mathfrak{G}_c$ defined in Section 3.2.

The component graph is a model of the logic signal flow of the circuit and we assume that the output of a channel-connected component is a function of all its inputs. If an input of a channel-connected component is undefined, then all the components whose outputs are undefined inputs of this subcircuit have to be computed first in order to resolve the undefined states. We exploit this interdependency of the undefined subcircuits to generate a new graph as follows: For each channel-connected component, we add a vertex and, if the input coverage of the component is less than one, for each undefined input a directed edge from the component which determines the value of the node to the currently processed component. Each edge is labeled with the corresponding node number

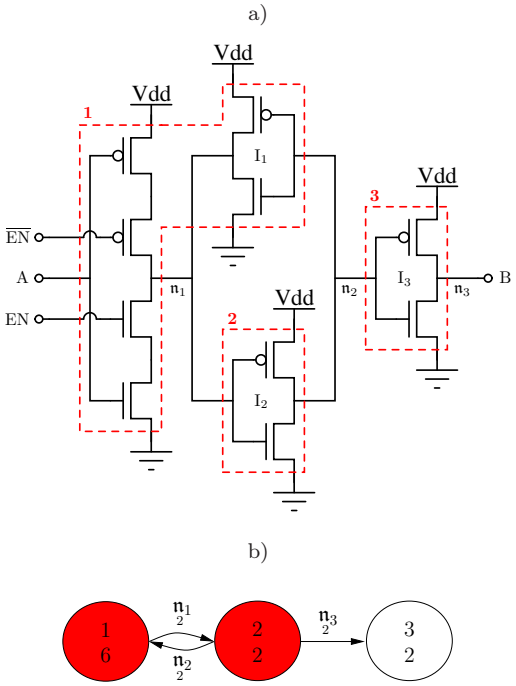


Figure 3.7: A latch with potential deadlocks. a) Decomposition into channel-connected components. b) Deadlock graph $\mathfrak{G}_l$ of the latch for EN = 0.

and the number of connections. We call this graph the *deadlock graph* $\mathfrak{G}_l$ of the circuit. Since two components can be coupled by more than one undefined node, the deadlock graph is in general a directed multigraph.

Example 3.17 Consider the circuit shown in Figure 3.7a. If the input EN of the tri-state inverter is high, then the configuration can be regarded as a normal inverter and the switch-level algorithm yields $B = \overline{A}$. If, on the other hand, the enable input is low, then the output of the tri-state inverter is in the high-impedance state, which effectively disconnects the device from the output. Thus, the result of inverter I_2 depends on the output of inverter I_1 and vice versa. This configuration forms a strongly connected component. Consequently, the nodes $\mathfrak{n}_1$, $\mathfrak{n}_2$, and $\mathfrak{n}_3$ remain undefined after the switch-level simulation. Figure 3.7b shows the deadlock graph for EN = 0. ♢

Each strongly connected component of the deadlock graph represents a configuration in which a circular dependence on undefined nodes occurs. In order to break this circular dependence, one node $\mathfrak{n}_i$ of each component is initialized with the state $y_i = 0$.

Subsequent to the identification and initialization of the critical nodes, the switch-level simulation is restarted. Since the results of the first simulation can be reused, only the critical configurations have to be updated. Hence, a few iterations are in general sufficient to compute a new steady state. The resimulation is of particular importance since in many cases the coverage can be increased significantly and, more importantly, the new initial guess is sufficiently close to a solution of the system of nonlinear equations so that the Newton–Raphson method converges.

It is possible to use different strategies to break the cycles of the deadlock graph in order to maximize the coverage and to minimize the deviation at the same time. Potential strategies could be to break all undefined 2-cycles or to exploit the topological ordering or other properties of the deadlock graph. However, the results turned out to be virtually identical for the benchmark circuits since the critical configurations frequently consist of isolated 2-cycles.

Example 3.18 Consider the latch of Example 3.17 again. To illustrate the convergence difficulties of the Newton–Raphson method, we connect an inverter chain with ten stages to the output of the latch and simulate the circuit with our simulator *signalflow*, which is described in more detail in Chapter 5. Without the initialization and resimulation, the Newton–Raphson method needs 103 iterations to converge to the metastable operating point. The algorithm to initialize the critical nodes will either set $y(\mathfrak{n}_1) = 0$ or $y(\mathfrak{n}_2) = 0$ so that after the resimulation all nodes are well-defined except for one of the internal nodes of the tri-state inverter. Now, two iterations are sufficient to find the corresponding stable operating point. ♢

Let us demonstrate the influence of the initialization and resimulation with the aid of a more complex circuit. To visualize the results of the switch-level simulation, we draw the component graph and color the vertices as follows: If the input coverage of a component is zero, then the corresponding vertex is colored red. If the input coverage is one, then the vertex is colored blue. Otherwise a linear interpolation between these two colors is used. This graph-based representation of the switch-level simulation is also useful to identify and analyze critical configurations such as, for instance, analog subcircuits that cannot be described properly by the extended switch-level model.

Example 3.19 The coverage of the switch-level simulation of circuit $\mathfrak{C}_{10}$ amounts to approximately 20 % and the impact on the DC analysis is negligible. The deadlock graph, which is shown in Figure 3.8, reveals that a significant part of the circuit depends on two

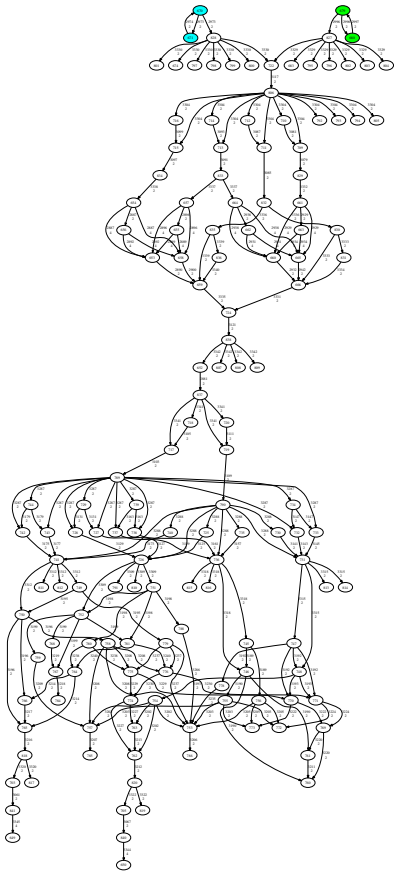


Figure 3.8: Section of the deadlock graph $\mathfrak{G}_l$ of $\mathfrak{C}_{10}$. The undefined regions depend on the two colored critical configurations.

critical configurations. After the initialization of the two critical nodes and the resimulation of the circuit, the coverage is approximately 60 %. Almost all inputs and outputs of the channel-connected components are set to a valid state. The remaining undefined nodes are mostly internal nodes that are only connected to nonconducting modules. Figure 3.9 shows a small fraction of the component graph of circuit $\mathfrak{C}_{10}$ before and after the resimulation. With the new initial guess, the runtime of the operating point analysis can be reduced considerably. Detailed results are presented in Section 3.4. ♢

In summary it can be said that only the combination of the switch-level simulation and the initialization of the critical nodes results in a robust and efficient method to compute an approximate operating point. Without an appropriate initial guess, the Newton–Raphson method usually fails to compute an operating point for large and complex integrated

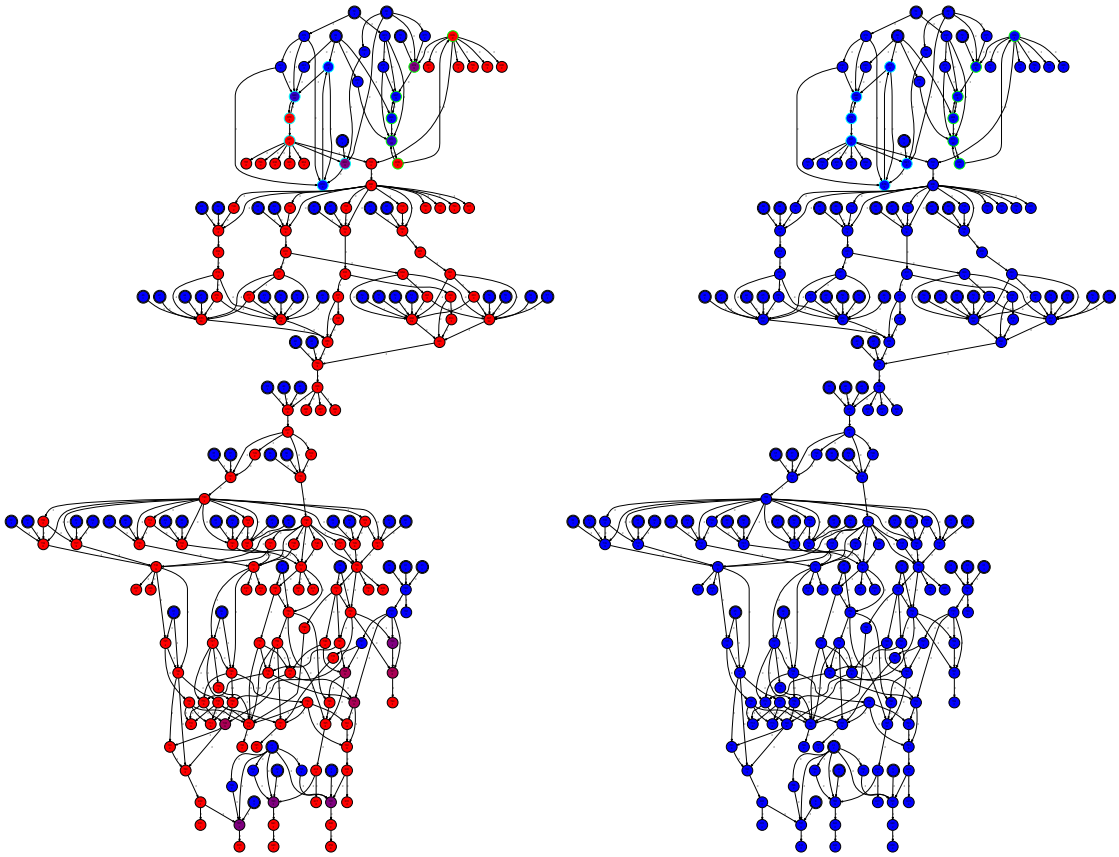
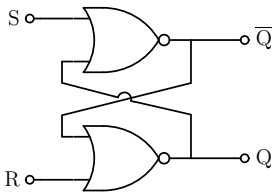


Figure 3.9: Section of the component graph $\mathfrak{G}_c$ of $\mathfrak{C}_{10}$ before and after the resimulation. Before the resimulation only the input voltage sources and a few channel-connected components are defined while the major part remains undefined. After the resimulation the input coverage of all components is 100 %.

circuits so that more time-consuming continuation methods have to be used. The initial guess provided by the extended switch-level simulation helps the circuit simulator to find an operating point. In particular for medium- and large-scale cross sections, this approach leads to a significantly improved performance of the operating point analysis, as we will show in the following section.

3.4 Numerical results

Several circuit simulators enable the user to specify initial values for some or all nodes of the circuit via IC and NODESET statements. These initial values have a big influence



S	**R**	**Q**	$\overline{\mathbf{Q}}$
0	0	Q	$\overline{\text{Q}}$
0	1	0	1
1	0	1	0
1	1	0	0

Figure 3.10: Set-reset latch using NOR gates.

on the convergence of the DC analysis. The provided NODESET statements are used to make a preliminary pass with the specified nodes held to the given voltages [QNPS93]. The NODESETs are then released and the operating point is recomputed. In case of multiple operating points, NODESETs can also be used to obtain a specific solution. The IC statement is for setting transient initial conditions. While NODESETs apply to node voltages only, ICs can also be used to assign initial node charges.

Example 3.20 The SR latch shown in Figure 3.10 has one metastable and two stable operating points. If the inputs S and R are set to 0 V and no NODESETs are specified, SPICE converges to the metastable operating point. If a NODESET for Q or $\overline{\text{Q}}$ is supplied, then the closest stable operating point is found. ♢

With the aid of the NODESET statement, the results of the switch-level simulation can be easily applied. The extended switch-level simulation is carried out prior to the DC analysis and the results are used to generate appropriate NODESETs. The generated NODESETs can be added to the netlist or, since these methods have been integrated into the circuit simulator of our industry partner, directly submitted to the nonlinear solver. There are different approaches to deal with already specified user-defined NODESETs and ICs. Depending on the settings, user-defined NODESETs and ICs can be ignored and possibly overwritten or used as initial conditions for the switch-level simulation.

The impact of this approach depends strongly on the coverage of the switch-level simulation. Only if appropriate initial conditions for a sufficiently large number of nodes are generated, the Newton–Raphson method converges quickly to a solution. In addition to the standard Newton–Raphson method, the abovementioned circuit simulator provides a pseudo-transient analysis. In case of failing Newton–Raphson method, the simulator switches automatically to the pseudo iteration.

To analyze the performance of the signal-flow based approach, we simulate each benchmark circuit introduced in Section 3.1 with eight different settings. First of all, we start a

Table 3.2: Runtimes of the different simulations in seconds.

	OP				TRAN			
	SLS off		SLS on		SLS off		SLS on	
	NR	PR	NR	PR	NR	PR	NR	PR
$\mathfrak{C}_1$	0.05	0.06	0.04	0.07	0.38	0.08	0.02	0.06
$\mathfrak{C}_2$	0.26	2.23	2.94	0.50	0.19	1.73	2.35	0.35
$\mathfrak{C}_3$	2.12	1.74	1.51	4.24	2.64	1.63	4.18	2.54
$\mathfrak{C}_4$	69.00	18.71	69.34	24.39	63.37	15.06	61.37	37.54
$\mathfrak{C}_5$	75.49	43.82	5.49	10.03	83.42	59.80	3.27	2.85
$\mathfrak{C}_6$	238.30	123.81	133.90	29.26	1.16	2.39	1.39	2.95
$\mathfrak{C}_7$	1.05	1.09	2.13	2.20	0.18	0.28	0.37	0.58
$\mathfrak{C}_8$	298.97	129.67	261.04	26.22	473.49	157.23	23.81	39.46
$\mathfrak{C}_9$	294.48	104.40	420.53	26.17	127.92	121.01	27.13	28.54
$\mathfrak{C}_{10}$	44.14	17.97	2.27	2.76	35.44	14.48	0.74	1.34
$\mathfrak{C}_{11}$	443.12	332.53	108.64	36.47	345.77	264.83	23.10	64.48
$\mathfrak{C}_{12}$	599.13	444.12	28.21	32.51	463.32	357.72	18.70	22.50
$\mathfrak{C}_{13}$	556.45	447.44	20.67	35.17	447.25	400.78	11.92	52.96
$\mathfrak{C}_{14}$	911.33	755.90	63.64	56.18	602.71	481.92	38.29	30.29
$\mathfrak{C}_{15}$	2 208.93	340.94	56.56	109.05	206.62	100.31	21.80	59.88

standard DC analysis (OP) without a previous switch-level simulation using the Newton–Raphson method (NR) and the homotopy method (PR), respectively. Afterwards, the simulation is repeated with enabled switch-level simulation. The same settings are then used to compute consistent initial values for the transient simulation (TRAN). Table 3.2 shows the runtimes of the simulator. In case of enabled switch-level simulation, the runtime of the switch-level algorithm itself, which is in general only a negligibly small fraction of the overall runtime, is already included.

To measure the speedup, we compare the fastest simulation with disabled and the fastest simulation with enabled switch-level simulation. Figure 3.11 shows the coverage of the switch-level simulation and the achieved speedup. Whether the standard Newton–Raphson or the homotopy method is better suited, is in general unknown prior to the simulation and depends strongly on the size and the characteristics of the circuit. For large and difficult problems, it is recommended to use the homotopy method. However, if the switch-level simulation is used to compute an appropriate initial guess and the

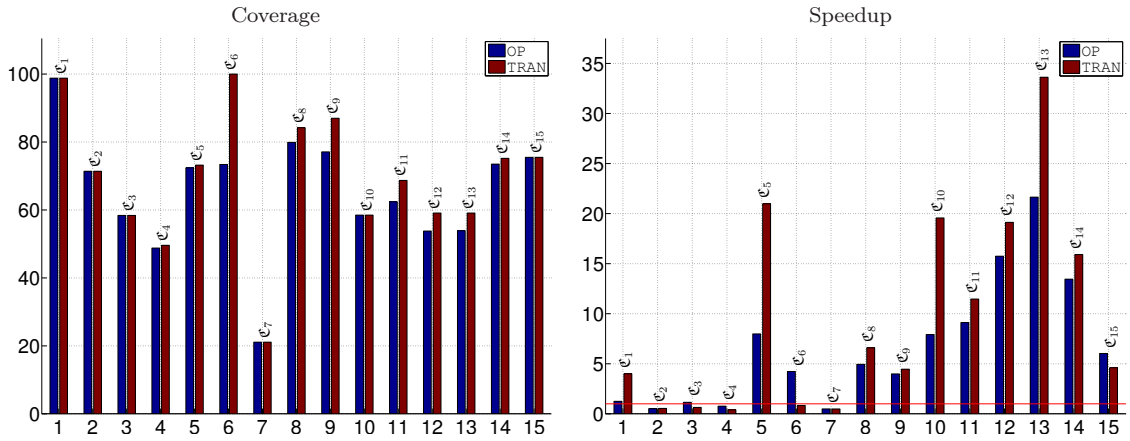


Figure 3.11: Coverage of the switch-level simulation and achieved speedup.

supplied NODESETs are sufficiently close to an operating point of the circuit, then the Newton–Raphson iteration usually converges much faster.

The coverage of the switch-level simulation for the subsequent transient analysis is often slightly higher due to the additional IC statements. As already mentioned above, a coverage of 100 % can in general not be obtained. Depending on the structure of the circuit, a value between 60 % and 80 % usually covers the relevant part of the circuit. Nodes which are only connected to nonconducting modules or whose behavior cannot be determined accurately with the aid the mainly voltage-based switch-level model remain indeterminate.

The netlist of circuit $\mathfrak{C}_6$ contains IC statements with exact initial conditions for all nodes such that the Newton–Raphson method converges directly. Therefore, the switch-level simulation results in a coverage of 100 % without improving the convergence. The simulation of circuit $\mathfrak{C}_7$ is a power-up simulation where all voltage sources are initially set to zero. As a consequence, the operating point is zero and since this is also the commonly used starting point for the Newton–Raphson iteration, the switch-level simulation causes an additional overhead which increases the over-all runtime.

Note that there is a correlation between the size of the circuit – the characteristics of the circuits are listed in Table 3.1 – and the achieved speedup. For small circuits, the Newton–Raphson method usually converges without problems and a preceding switch-level simulation is not necessary. However, the extended switch-level model was in particular tailored to time-consuming medium- and large-scale cross sections. For these circuits, the switch-level simulation leads to a significantly reduced runtime.

3.5 Further applications

The component graph could also be used to generate a signal-flow based partitioning of the circuit for a subsequent parallel simulation. In this way, the communication between partitions could possibly be minimized. Furthermore, the component graph is in general much smaller than the circuit graph so that the partitioning could be carried out more efficiently.

A combination of the signal-flow based partitioning and an extension of the switch-level model which takes into account also dynamic aspects could be employed to adaptively generate a decomposition of the circuit into active and latent parts. The latent parts could then be solved using different integration schemes or different step sizes. Multirate strategies will be discussed in more detail in Chapter 4.

Switch-level simulators can also be designed to provide additional information on the circuit. Bryant [Bry81] suggested that, instead of computing only the state of each node, a more sophisticated program could also supply the paths of the signals, which would greatly aid the user in debugging the circuit design. This feature was incorporated in our switch-level model to analyze and improve the algorithms. Moreover, the information on conducting paths can be used to detect critical configurations and to resolve invalid states.

4

Signal-flow based numerical integration

The previous chapter illustrated that it is possible to speed up the operating point analysis considerably using information on the network structure and the logic signal flow. In this chapter, we will focus on the transient simulation of integrated circuits. During the transient simulation, usually only a few elements are active whereas the major part of the circuit remains latent, as described in Section 2.3. Günther and Rentrop [GR94] suggest that multirate strategies must be based both on the numerical information of the integration scheme and on the topology of the circuit. We will introduce a directed graph which describes the interdependency of the underlying system and propose Runge–Kutta methods that utilize the signal flow of the system in order to identify and exploit inactive regions. Furthermore, an extension of these methods to identify and exploit also periodic subsystems is described.

The graph that we introduced in Chapter 3 can be viewed as a model of the digital, logic signal flow. It is based on the topology of the circuit and the characteristic properties of the different module types. The graph that we will introduce now is a model of the analog signal flow. Here, we will exploit properties of the circuit equations to determine the dependency relations. Nevertheless, it will be shown that these two graphs are closely related for specific circuits.

We will present the signal-flow based methods using mainly the example of CMOS circuits. However, the proposed methods are applicable to arbitrary complex dynamical networks with inherent latency or periodicity. Complex networks appear in a wide range of physical, biological, and engineering systems. Since the coupling of subsystems with varying time scales often results in multirate behavior, signal-flow based integration schemes could also be used to speed up the simulation of such systems.

4.1 Ordinary differential equations

The modified nodal analysis leads in general to a mixed system of differential and algebraic equations. For some circuits, however, the equations can be rewritten as a system of ordinary differential equations. A further possibility to obtain an ordinary differential equation is to regularize the circuit equations by adding parasitic elements. A rule-of-thumb is to insert a capacitor from each node to ground in order to obtain a regular capacitance matrix. This technique is often used to elaborate on the concepts. For a detailed discussion, see for example [GF95, GFtM05]. Here, we consider in particular regularized CMOS circuits which consist of voltage sources, capacitors, and MOSFETs. To describe the behavior of the MOSFETs, we use the Shichman–Hodges model, which was introduced in Section 2.1. For such a circuit, the nodal analysis leads to a system of the form $C\dot{v}(t) + \imath(t, v(t)) = 0$ with a regular capacitance matrix C and thus to an ordinary differential equation.

From now on, we consider ordinary differential equations of the general form

$$\dot{x}(t) = f(t, x(t)), \tag{4.1}$$

with $t \in \mathbb{I} \subseteq \mathbb{R}$, $f : \mathbb{I} \times \mathbb{D}_x \to \mathbb{R}^n$, $\mathbb{D}_x \subseteq \mathbb{R}^n$, and the initial condition

$$x(t_0) = x_0, \quad t_0 \in \mathbb{I}. \tag{4.2}$$

Since it is in general not possible or feasible to solve this initial value problem analytically, numerical methods have to be applied. A fundamental class of numerical solvers are one-step methods of the form

$$x^{m+1} = x^m + h\,\Phi(t^m, x^m, h), \tag{4.3}$$

where Φ is referred to as the *increment function*. Important examples of one-step methods are Runge–Kutta methods [But87, HNW93, HW96].

Definition 4.1 (Runge–Kutta method) A general s-stage *Runge–Kutta method* is given by

$$x^{m+1} = x^m + h\sum_{q=1}^{s} b_q k_q, \tag{4.4a}$$

where

$$k_q = f\big(t^m + c_q h, x^m + h\sum_{r=1}^{s} a_{qr} k_r\big). \tag{4.4b}$$

The coefficients a_{qr}, b_q, and c_q are often arranged in form of the so-called *Butcher tableau*

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} \quad := \quad \begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array} . \tag{4.5}$$

If the matrix A is strictly lower triangular, then the Runge–Kutta method is called explicit. Otherwise, the method is said to be implicit.

Definition 4.2 (Convergence) Let $x(t)$ be the exact solution of the ordinary differential equation (4.1). The *global truncation error* is defined by

$$e_h(t^m) = x(t^m) - x^m. \tag{4.6}$$

A one-step method is defined to be *convergent* if

$$\lim_{h\to 0} e_h(t^m) = 0 \tag{4.7}$$

and *convergent of order* p if $e_h(t^m) = \mathcal{O}(h^p)$.

Theorem 4.3 *If the coefficients a_{qr}, b_q, and c_q of the Runge–Kutta method fulfill the conditions*

$$B(p): \qquad \sum_{q=1}^{s} b_q c_q^{i-1} = \frac{1}{i}, \qquad i = 1,\dots,p, \tag{4.8a}$$

$$C(\eta): \qquad \sum_{r=1}^{s} a_{qr} c_r^{i-1} = \frac{1}{i} c_q^i, \qquad i = 1,\ \dots,\eta,\ q = 1,\dots,s, \tag{4.8b}$$

$$D(\zeta): \qquad \sum_{q=1}^{s} b_q c_q^{i-1} a_{qr} = \frac{1}{i} b_r (1 - c_r^i), \qquad i = 1,\dots,\zeta,\ r = 1,\dots,s, \tag{4.8c}$$

with $p \le 2\eta + 2$ and $p \le \eta + \zeta + 1$, then the method is convergent of order p.

Proof. A proof of this result can be found in [HNW93], for instance. □

Example 4.4

i) The classical explicit fourth-order Runge–Kutta method is defined by

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}.$$

ii) The trapezoidal rule, which is an implicit second-order Runge–Kutta method, can be written as

$$\begin{aligned} x^{m+1} &= x^m + \frac{h}{2}\left(f(t^m, x^m) + f(t^{m+1}, x^{m+1})\right) \\ &= x^m + \frac{h}{2}(k_1 + k_2), \end{aligned}$$

with

$$\begin{aligned} k_1 &= f\big(t^m, x^m\big), \\ k_2 &= f\big(t^{m+1}, x^{m+1}\big) = f\big(t^m + h, x^m + \frac{h}{2}(k_1 + k_2)\big). \end{aligned}$$

Thus, the corresponding Butcher tableau is given by

$$\begin{array}{c|cc} 0 & & \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}.$$ ♢

4.2 Multirate integration

The distribution of the time scales in the solution of the initial value problem has a big influence on the applicability and reliability of numerical integration schemes [GR94]. Nonstiff problems can in general be solved efficiently using explicit schemes, whereas stiff problems involving different time scales necessitate implicit schemes. The application of implicit schemes to nonstiff problems results in an unnecessarily high computing time. The application of explicit schemes to stiff problems, on the contrary, might lead to completely wrong results or prohibitively small step sizes. Different approaches such as partitioning or

multirate strategies have been developed in order to exploit the different rates of activity and thus to reduce the computational complexity.

In [Ren85, GR94, KR99, BGK01, GKR01] it is suggested to partition the vector $x \in \mathbb{R}^n$ into an active part $x_A \in \mathbb{R}^{n_A}$ and a latent part $x_L \in \mathbb{R}^{n_L}$, $n_A + n_L = n$, and to rewrite the ordinary differential equation $\dot{x}(t) = f(x(t))$, which now without loss of generality is assumed to be autonomous, as

$$\begin{aligned} \dot{x}_A(t) &= f_A(x_A(t), x_L(t)), \\ \dot{x}_L(t) &= f_L(x_A(t), x_L(t)). \end{aligned} \tag{4.9}$$

The partitioning can be either performed statically prior to the simulation exploiting characteristic properties of the system or dynamically during the simulation using information provided by the integration scheme.

One possibility to exploit the inactive regions is to use partitioned Runge–Kutta methods [Hof76, HNW93]. A partitioned Runge–Kutta method for the system (4.9) is of the form

$$\begin{aligned} x_A^{m+1} &= x_A^m + h \sum_{q=1}^{s} b_q k_A^q, \\ x_L^{m+1} &= x_L^m + h \sum_{q=1}^{s} \hat{b}_q k_L^q, \end{aligned} \tag{4.10a}$$

with

$$\begin{aligned} k_A^q &= f_A\big(x_A^m + h \sum_{r=1}^{s} a_{qr} k_A^r,\, x_L^m + h \sum_{r=1}^{s} \hat{a}_{qr} k_L^r\big), \\ k_L^q &= f_L\big(x_A^m + h \sum_{r=1}^{s} a_{qr} k_A^r,\, x_L^m + h \sum_{r=1}^{s} \hat{a}_{qr} k_L^r\big). \end{aligned} \tag{4.10b}$$

Now, the nonstiff active part can be solved by explicit schemes and the stiff latent part by implicit schemes. Günther and Rentrop [Ren85, GR94] also combined explicit Runge–Kutta methods and linearly implicit Rosenbrock methods via

$$\begin{aligned} k_A^q &= f_A\big(x_A^m + h \sum_{r=1}^{q-1} a_{qr} k_A^r,\, x_L^m + h \sum_{r=1}^{q-1} \hat{a}_{qr} k_L^r\big), \\ k_L^q &= f_L\big(x_A^m + h \sum_{r=1}^{q-1} a_{qr} k_A^r,\, x_L^m + h \sum_{r=1}^{q-1} \hat{a}_{qr} k_L^r\big) + hDf_L \sum_{r=1}^{q} \gamma_{qr} k_L^r, \end{aligned} \tag{4.10b'}$$

where $Df_L = \frac{\partial f_L}{\partial y_L}(y_A^m, y_L^m)$. The advantage of Rosenbrock type methods, which can be derived from diagonally implicit Runge–Kutta methods, is that rather than systems of nonlinear equations only systems of linear equations have to be solved [HW96]. Moreover, Rosenbrock methods are well designed for an automatic stiffness detection since they always include an embedded standard Runge–Kutta method [Ren85].

Another possibility to exploit the inactive regions is to use different step sizes for the active and the latent part. The active part is integrated with a small step size h, while the inactive part is integrated with a large step size H. A synchronization of both parts is performed after each macro step [KR99]. For convenience of notation, we omit the superscript m and describe the first macro step. Let $H = \mu h$, then the active components x_A are given by

$$\begin{aligned} x_A^{\lambda+1} &= x_A^\lambda + h\sum_{q=1}^{s} b_q k_A^{\lambda,q} \approx x_A(t_0 + (\lambda+1)h), \\ k_A^{\lambda,q} &= f_A\big(x_A^\lambda + h\sum_{r=1}^{s} a_{qr} k_A^{\lambda,r}, \tilde{X}_L^{\lambda,q}\big), \end{aligned} \tag{4.11a}$$

for $\lambda = 0, 1, \ldots, \mu - 1$. Here, $\tilde{X}_L^{\lambda,q} \approx x_L\left(t_0 + (\lambda + c_q)h\right)$, with $c_q = \sum_{r=1}^{s} a_{qr}$. The latent components x_L are then given by

$$\begin{aligned} x_L^1 &= x_L^0 + H\sum_{q=1}^{\hat{s}} \hat{b}_q k_L^q \approx x_L(t_0 + H), \\ k_L^q &= f_L\big(\tilde{X}_A^q, x_L^0 + H\sum_{r=1}^{\hat{s}} \hat{a}_{qr} k_L^r\big), \end{aligned} \tag{4.11b}$$

where $\tilde{X}_A^q \approx x_A(t_0 + \hat{c}_q H)$, with $\hat{c}_q = \sum_{r=1}^{\hat{s}} \hat{a}_{qr}$. In the same way, Rosenbrock type methods can be used, as described in [GR94].

The coupling between the active and the latent part, given by the intermediate stage values $\tilde{X}_L^{\lambda,q}$ and $\tilde{X}_A^q$, can be computed using interpolation or extrapolation schemes. There are mainly two different approaches:

i) *Fastest first strategy:* Perform μ steps of step size h and use extrapolation schemes to obtain the required values of x_L, then perform one step of step size H and interpolate to obtain the values of x_A.

ii) *Slowest first strategy:* Perform one step of step size H and use extrapolation schemes to obtain the required values of x_A, then perform μ steps of step size h and interpolate to obtain the values of x_L.

A shortcoming of these multirate methods which makes the implementation into existing simulation tools challenging is the coupling between the active and the latent part. In [BGK01, GKR01] a so-called compound step which combines the macro step and the first micro step is introduced. Furthermore, dense output formulas are used to obtain the already computed solution on the finer grid.

A further multirate time-stepping strategy is presented in [SHV07]. Rather than decomposing the differential equation explicitly, they compute a first, tentative approximation for the entire system using a Rosenbrock type method with a global step size h. All components for which the estimated local error is larger than a predefined tolerance are then recomputed with the step size $\frac{h}{2}$. The required intermediate values of the remaining components are obtained via interpolation. This procedure is repeated recursively until all components satisfy the accuracy requirements.

We will propose a different approach which is based on the structure of the underlying ordinary differential equation. The latent parts of the system are identified using tools from graph theory. We aim in particular at exploiting variables or subsystems that are temporarily in a steady state. This kind of latency exploitation can be regarded as a special type of multirate integration.

4.3 Time-driven ordinary differential equations

Without loss of generality, the ordinary differential equation (4.1) can be rewritten as

$$\begin{bmatrix} x_E \\ \dot{x}_I \end{bmatrix} = \begin{bmatrix} f_E(t) \\ f_I(x_E, x_I) \end{bmatrix}, \tag{4.12}$$

with external variables $x_E \in \mathbb{R}^{n_E}$ and internal variables $x_I \in \mathbb{R}^{n_I}$. That is, we split the system into two subsystems and introduce additional variables which can be explicitly written as a function of the time t. The dimension of the input vector x_E depends on the number of different time-dependent terms, the dimension of the internal vector x_I is equal to the number of equations of the original system. We introduce this partitioning to measure the influence of the input signals on the internal variables and to generate a model of the signal flow.

From now on, for the sake of simplicity, we will write the system – to which we will refer as a *time-driven ordinary differential equation* – as

$$\begin{bmatrix} x_E \\ \dot{x}_I \end{bmatrix} = f(t, x), \text{ with } x = \begin{bmatrix} x_E \\ x_I \end{bmatrix} \text{ and } f = \begin{bmatrix} f_E \\ f_I \end{bmatrix}. \tag{4.13}$$

Thus, $x_{E,i} = x_i$ and $x_{I,i} = x_{n_E+i}$. Let $n = n_E + n_I$ denote the size of the whole system again.

For a time-driven ordinary differential equation, a one-step method is of the form

$$\begin{bmatrix} x_E^{m+1} \\ x_I^{m+1} \end{bmatrix} = \begin{bmatrix} x_E^m \\ x_I^m \end{bmatrix} + \begin{bmatrix} \Delta x_E^m \\ \Delta x_I^m \end{bmatrix}, \tag{4.14}$$

with

$$\begin{aligned} \Delta x_E^m &= f_E(t^{m+1}) - f_E(t^m), \\ \Delta x_I^m &= h\,\Phi(t^m, x^m, h). \end{aligned} \tag{4.15}$$

The increment function of a Runge–Kutta method can now be rewritten as

$$\Phi(t^m, x^m, h) = \sum_{q=1}^{s} b_q k_I^q, \tag{4.16a}$$

where

$$\begin{aligned} k_E^q &= f_E(t^m + c_q h), \\ k_I^q &= f_I\big(k_E^q, x_I^m + h \sum_{r=1}^{s} a_{qr} k_I^r\big). \end{aligned} \tag{4.16b}$$

4.3.1 Dependency graph

Given a time-driven ordinary differential equation, we want to analyze how changes of the input variables x_E affect the internal variables x_I and how the signals propagate through the system. To this end, we derive a directed graph which represents the structure of the system.

For simplicity, define $\langle n \rangle = \{1, \ldots, n\}$ to be the set of indices. Since in general the functions f_i, $i \in \langle n \rangle$, do not depend on all variables x_j, $j \in \langle n \rangle$, we introduce input and output sets of each variable to describe the dependency on other variables.

Definition 4.5 (Input and output sets) Define the *input set* of x_i, $i \in \langle n \rangle$, to be

$$\bullet\, x_i = \left\{ x_j \;\middle|\; \frac{\partial f_i}{\partial x_j} \not\equiv 0,\, j \in \langle n \rangle \right\}. \tag{4.17}$$

Analogously, define the *output set* to be

$$x_i\, \bullet = \left\{ x_j \;\middle|\; \frac{\partial f_j}{\partial x_i} \not\equiv 0,\, j \in \langle n \rangle \right\}. \tag{4.18}$$

That is, the variable x_i depends on x_j if the value of x_j is required for the evaluation of f_i. The input and output sets induce a directed graph with the vertices being the variables and the edges being the dependency relations between the variables.

Definition 4.6 (Dependency graph) For a given time-driven ordinary differential equation, define the *dependency graph* by $\mathfrak{G}_d(f) = (\mathfrak{V}_d, \mathfrak{E}_d)$, with $\mathfrak{V}_d = \{\mathfrak{v}_1, \dots, \mathfrak{v}_n\}$ and $\mathfrak{E}_d = \{(\mathfrak{v}_i, \mathfrak{v}_j) \mid x_i \in \bullet x_j,\ i, j \in \langle n \rangle\}$.

If it is clear which differential equation is meant, we will simply write $\mathfrak{G}_d$. The dependency graph of large-scale dynamical networks can be very sparse since the subsystems are often strongly coupled inside but only connected to a few other subsystems of the network.

Example 4.7

i) Consider the linear differential equation

$$\dddot{x}(t) = \ddot{x}(t) + \dot{x}(t),$$

which is equivalent to the first-order system

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}}_{A} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix}.$$

The input and output sets are

$$\begin{aligned} \bullet x_1 &= \{x_2\}, & x_1 \bullet &= \varnothing, \\ \bullet x_2 &= \{x_3\}, & x_2 \bullet &= \{x_1, x_4\}, \\ \bullet x_3 &= \{x_4\}, & x_3 \bullet &= \{x_2\}, \\ \bullet x_4 &= \{x_2, x_4\}, & x_4 \bullet &= \{x_3, x_4\}. \end{aligned}$$

The differential equation is an equation of order three in $\dot{x}(t)$. This can also be seen in the dependency graph, which is shown in Figure 4.1, since x_1 depends only on x_2 and can be obtained by integration. Moreover, the transposed system matrix A^T is the adjacency matrix of $\mathfrak{G}_d$, i.e. $\mathfrak{G}_d = \mathfrak{G}(A^T)$.

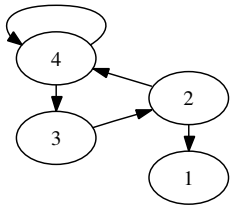


Figure 4.1: Dependency graph $\mathfrak{G}_d$ of the linear system.

ii) Given the inverter chain of length N shown in Figure 4.2, the corresponding circuit equations can be written as a time-driven ordinary differential equation with

$$f(t,v) = \begin{bmatrix} 0 \\ \mathrm{V_{dd}} \\ V_s(t) \\ \hline g(v_1,v_2,v_3,v_4) \\ g(v_1,v_2,v_4,v_5) \\ \vdots \\ g(v_1,v_2,v_{N+2},v_{N+3}) \end{bmatrix}.$$

Here, $n_E = 3$ and $n_I = N$. The function g consists of the characteristic equations of the modules connected to the individual nodes and can be written as

$$g(v_1,v_2,v_{i-1},v_i) = -\frac{1}{C_i}\big(\imath_{ds,n}(v_i,v_{i-1},v_1) + \imath_{ds,p}(v_i,v_{i-1},v_2)\big).$$

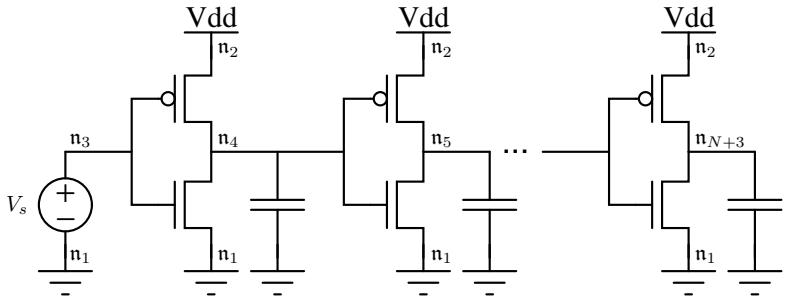


Figure 4.2: Inverter chain of length N.

Although the ground voltage and the positive supply voltage $\mathrm{V_{dd}}$ are constant over time, we introduce additional variables since this assignment leads to a natural correlation between the nodes $\mathfrak{n}_i$ and the vertices $\mathfrak{v}_i$. In addition, it allows for a straightforward

graph-based approach to generate the system of equations and the dependency graph. The Jacobian $\frac{\partial f}{\partial v}$ exhibits the following structure

$$\frac{\partial f}{\partial v} = \left[\begin{array}{ccc|cccccc} & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ \hline * & * & * & * & & & & \\ * & * & & * & * & & & \\ * & * & & & * & * & & \\ \vdots & \vdots & & & & \ddots & \ddots & \\ * & * & & & & & * & * \end{array}\right],$$

where empty places denote partial derivatives identical to zero. Figure 4.3 shows the dependency graph of the inverter chain. Since the constant voltages v_1 and v_2 have no influence on the dynamic signal flow, the corresponding vertices and associated edges have been omitted due to visualization reasons.

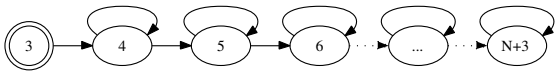


Figure 4.3: Dependency graph $\mathfrak{G}_d$ of the inverter chain.

iii) A semi-discretization of the medical Akzo Nobel problem [LSV96], which describes the penetration of radio-labeled antibodies into tumorous tissue and consists of two partial differential equations

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2} - kuv, \\ \frac{\partial v}{\partial t} &= -kuv, \end{aligned}$$

yields a $(2N+1)$-dimensional time-driven ordinary differential equation. Here, N is a user-defined parameter. The system can be written as

$$f(t,x) = \begin{bmatrix} \varphi(t) \\ g(x_E, x_I) \end{bmatrix},$$

with $n_E = 1$ and $n_I = 2N$. The function g is given by

$$g_i(\xi,\eta) = \begin{cases} \alpha_{\frac{i+1}{2}} \dfrac{\eta_{i+2} - \eta_{i-2}}{2\Delta\zeta} + \beta_{\frac{i+1}{2}} \dfrac{\eta_{i+2} - 2\eta_i + \eta_{i-2}}{(\Delta\zeta)^2} - k\,\eta_{i+1}\eta_i, & \text{if } i \text{ odd}, \\ -k\,\eta_i\,\eta_{i-1}, & \text{if } i \text{ even}, \end{cases}$$

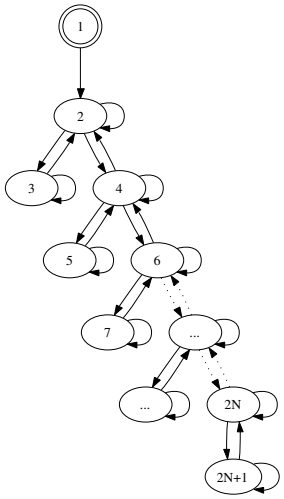


Figure 4.4: Dependency graph $\mathfrak{G}_d$ of the medical Akzo Nobel problem.

where

$$\alpha_i = \frac{2(i\Delta\zeta - 1)^3}{c^2} \quad \text{and} \quad \beta_i = \frac{(i\Delta\zeta - 1)^4}{c^2},$$

for $i = 1, \ldots, 2N$. Furthermore, $\Delta\zeta = \frac{1}{N}$, $\eta_{-1} = \xi$, and $\eta_{2N+1} = \eta_{2N-1}$. The dependency graph of this system is shown in Figure 4.4. ♢

Remark 4.8

i) A similar graph is introduced in [Rei88, Šil91] to analyze control systems. The graph representation of the system is used to investigate important properties such as controllability and observability. Moreover, it is also mentioned that the decomposition of the graph into strongly connected components and the subsequent topological ordering of the resulting condensed graph can be used to partition large-scale control systems into weakly coupled subsystems and to order these subsystems.

ii) An analogous approach to partition a given complex dynamical system into a hierarchy of subsystems is described in [Mez04, VKLM04]. The system is decomposed in such a way that each hierarchy level depends only on the levels below. Additionally, each level itself is further subdivided into separate subsystems whose dynamics are independent of each other. This so-called *horizontal-vertical decomposition* corresponds to the aforementioned partitioning of the graph representation of the system into strongly connected components.

In the following, we often identify x_i with $\mathfrak{v}_i$ and vice versa. Each internal vertex of the dependency graph represents a one-dimensional ordinary differential equation that is

coupled to other one-dimensional systems. Generally speaking, a time-driven ordinary differential equation together with its dependency graph can be regarded as a coupled cell system with additional time-dependent inputs.

Remark 4.9 A *coupled cell system* [GS03, GPS04] is a network of coupled ordinary differential equations. Associated with a coupled cell system is a directed graph. Each vertex of the graph corresponds to a subsystem or cell and each edge represents the coupling between different cells. It can be shown that structural properties, in particular symmetries of the graph, can be used to explain many aspects of pattern-formation in these systems. Rather than analyzing the symmetry and synchrony of dynamical systems, we will focus on the latency inherent in complex networked systems.

In order to generate the dependency graph, it is necessary to compute the structure of the Jacobian $\frac{\partial f}{\partial x}$. This might be infeasible or at least time-consuming for large-scale dynamical systems. For the class of integrated circuits that we consider here, however, the dependency graph can be generated efficiently using the topology of the circuit. Let $\mathfrak{T}_i$ denote the set of all MOSFETs that are connected with their channel to node $\mathfrak{n}_i$. Then Kirchhoff's current law yields

$$C_i \dot{v}_i = \sum_{\mathfrak{t} \in \mathfrak{T}_i} \pm \imath_{ds,p/n}(v_d(\mathfrak{t}), v_g(\mathfrak{t}), v_s(\mathfrak{t})).$$

That is, the function f_i depends on all drain, gate, and source nodes of the MOSFETs contained in $\mathfrak{T}_i$. The dependency graph and the component graph of a CMOS circuit are closely related as the following example shows.

Example 4.10 Consider the 4-bit adder of Example 3.2. The component graph of this circuit is a directed acyclic graph. If we cluster the strongly connected components of the dependency graph, then the resulting condensed graph, which is shown in Figure 4.5, is isomorphic to the component graph. That is, the dependency graph of the 4-bit adder can be viewed as a refinement of the component graph. ♢

Now, we want to estimate the influence of signal changes. If an input signal is switched from low to high or vice versa, then possibly all vertices that are reachable from the corresponding input vertex have to be recomputed. This reachability analysis can be used to predict the resulting active and inactive regions.

Example 4.11 Consider the 4-bit adder again. The size of the reachable sets depends strongly on the corresponding bit. If, for instance, a_0 is switched, then the reachable

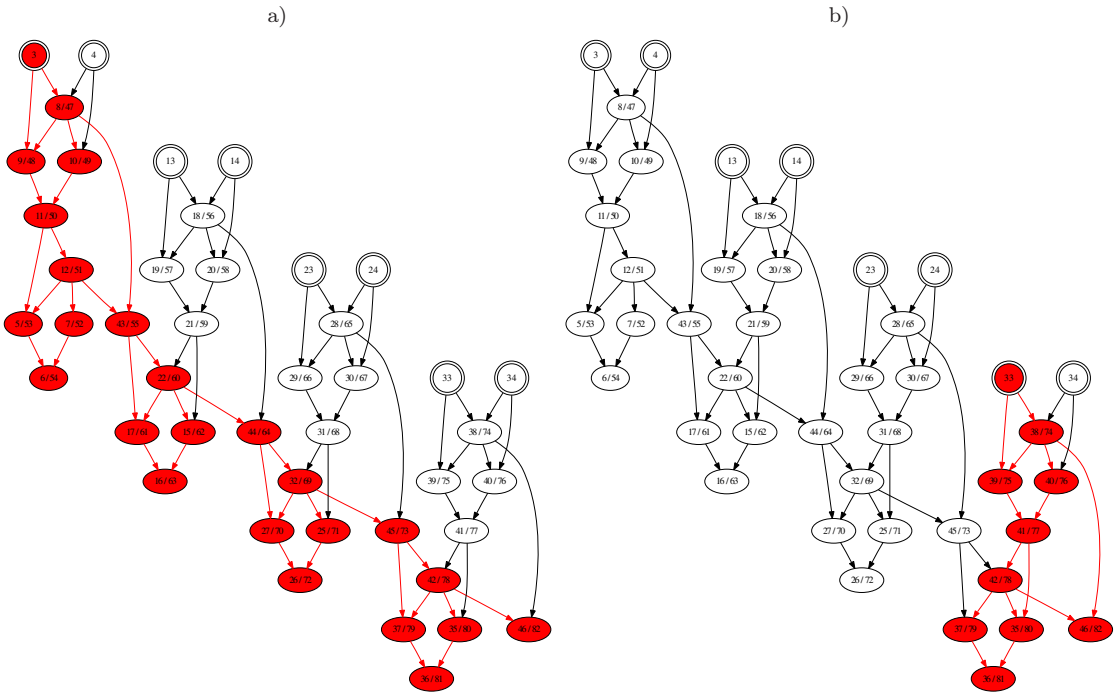


Figure 4.5: Condensed dependency graph of the 4-bit adder. a) Influence of input a_0. b) Influence of input a_3.

set comprises 48 vertices. If, on the other hand, a_3 is switched, then only 18 vertices are reachable. The reachable sets are shown in Figure 4.5. However, the size of the active regions depends also on the summands a and b. While the active region during the computation of $s = a + b$ with $a = [0\,0\,0\,\mathbf{1}]$ and $b = [0\,0\,0\,0]$ will be confined to the first full adder if the bold marked bit a_0 is switched from 0 to 1, the same computation with $a = [0\,0\,0\,\mathbf{1}]$ and $b = [1\,1\,1\,1]$ will activate all full adders through the carry bits. ♢

4.3.2 Algebraic graph theory

The above example illustrates that the computation of the reachable sets yields only a coarse, static estimate of the resulting active regions. The actual active regions will in general be much smaller. We want to use methods from algebraic graph theory [Fie86, GR01, BC08] to dynamically compute the active and inactive regions. As described in Section 2.2, a graph can be represented by various matrices such as the adjacency matrix, the incidence matrix, or the Laplacian. One aim of algebraic graph theory is to determine how the properties of a graph are related to algebraic properties of the corresponding

matrix representation. To begin with, we state a few basic definitions and results. The proofs of the following theorems can be found in [Fie86].

Definition 4.12 (Irreducibility) A matrix $A \in \mathbb{R}^{n\times n}$ is *reducible* if there exists a permutation matrix P such that

$$PAP^T = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}, \tag{4.19}$$

where A_{11} and A_{22} are square matrices of order at least one. If the matrix A is not reducible, then it is said to be *irreducible*.

Theorem 4.13 *A matrix A is irreducible if and only if $\mathfrak{G}(A)$ is strongly connected. Furthermore, any matrix A can be transformed into an upper block triangular matrix by a permutation matrix P, i.e.*

$$PAP^T = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1p} \\ 0 & A_{22} & A_{23} & \dots & A_{2p} \\ 0 & 0 & A_{33} & \dots & A_{3p} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & A_{pp} \end{bmatrix}. \tag{4.20}$$

The diagonal blocks correspond to the strongly connected components of the graph $\mathfrak{G}(A)$ and are uniquely determined up to permutations within the blocks. The ordering of the blocks corresponds to the – not necessarily unique – topological ordering of the components.

Definition 4.14 (Nonnegative matrix) A matrix $A \in \mathbb{R}^{n\times n}$ is defined to be *nonnegative*, if all entries a_{ij} are nonnegative. That is, $A \geq 0 \;\Leftrightarrow\; a_{ij} \geq 0 \;\forall\, i,j \in \langle n\rangle$.

Analogously, a matrix is said to be *positive* if all entries are positive. The sum or product of two nonnegative matrices is again a nonnegative matrix. For powers of nonnegative matrices, we obtain the following result.

Theorem 4.15 *Let $A \in \mathbb{R}^{n\times n}$ be a nonnegative matrix, and let k be a positive integer. The (i,j)-entry of A^k is nonzero, i.e. $\left[A^k\right]_{ij} \neq 0$, if and only if there exists a path of length k from vertex $\mathfrak{v}_i$ to vertex $\mathfrak{v}_j$ in $\mathfrak{G}(A)$.*

Corollary 4.16 *If A is the adjacency matrix of the graph $\mathfrak{G}$, then $\left[A^k\right]_{ij}$ equals the number of paths of length k from $\mathfrak{v}_i$ to $\mathfrak{v}_j$.*

Theorem 4.17 *Given an irreducible nonnegative matrix $A \in \mathbb{R}^{n\times n}$. If c_0, c_1, ..., c_{n-1} are positive numbers, then the matrix $c_0 I + c_1 A + c_2 A^2 + \cdots + c_{n-1} A^{n-1}$ is positive.*

Let us now begin with the analysis of linear or linearized ordinary differential equations of the form $\dot{x}(t) = A\,x(t)$. The solution of such a system can be easily expressed using the matrix exponential $\exp(A)$ – we will also use the alternative notation e^A –, which is given by

$$\exp(A) = \sum_{k=0}^{\infty} \frac{A^k}{k!}. \tag{4.21}$$

It is well known that the linear system of differential equations $\dot{x}(t) = A\,x(t)$, $x(0) = x_0$, has the general solution $x(t) = e^{tA}x_0$.

Lemma 4.18 *The dependency graph of the differential equation $\dot{x}(t) = A\,x(t)$ is given by $\mathfrak{G}_d = \mathfrak{G}(A^T) = \mathfrak{G}(A)^T$.*

Proof. Since

$$f_i = \sum_{j=1}^{n} a_{ij}x_j \;\Rightarrow\; \frac{\partial f_i}{\partial x_j} = a_{ij},$$

it holds that

$$(\mathfrak{v}_j, \mathfrak{v}_i) \in \mathfrak{E}_{\mathfrak{G}_d} \;\Leftrightarrow\; a_{ij} \neq 0 \;\Leftrightarrow\; (\mathfrak{v}_i, \mathfrak{v}_j) \in \mathfrak{E}_{\mathfrak{G}(A)}.$$ □

We want to analyze the influence of the structure of $\mathfrak{G}(A)$ or $\mathfrak{G}_d$, respectively, on the solution of the system. Theorem 4.15 illustrates the relation between paths in the graph $\mathfrak{G}(A)$ and powers of the matrix A. The matrix exponential contains all powers A^k, $k \in \mathbb{N}_0$. That is, for a nonnegative matrix A, the graph $\mathfrak{G}(e^A)$ and hence $\mathfrak{G}(e^{tA})$, $t > 0$, can be obtained from the transitive closure $\mathfrak{G}^*(A)$ by adding self-loops to all vertices. We use this consideration to identify active and latent regions.

Definition 4.19 (Activity sets) Let $A \in \mathbb{R}^{n\times n}$ and $\bar{x} \in \mathbb{R}^n$. For the differential equation $\dot{x}(t) = A\,x(t)$, define

$$\begin{aligned}
\mathfrak{U}_0^l(\bar{x}) &= \{\mathfrak{v}_i \mid \bar{x}_i \neq 0\},\\
\mathfrak{U}_1^l(\bar{x}) &= \mathrm{reach}^l_{\mathfrak{G}_d}\left(\mathfrak{U}_0^l(\bar{x})\right),\\
\mathfrak{U}_2^l(\bar{x}) &= \mathfrak{V} \setminus \left(\mathfrak{U}_0^l(\bar{x}) \cup \mathfrak{U}_1^l(\bar{x})\right)
\end{aligned} \tag{4.22}$$

to be the *activity sets* with respect to $\bar{x}$.

Theorem 4.20 *Given the ordinary differential equation $\dot{x}(t) = A\,x(t)$, $x(0) = x_0$, with a nonnegative matrix A and a nonnegative vector x_0. Then $x_i(t) \equiv 0$ if and only if $\mathfrak{v}_i \in \mathfrak{U}_2^\infty(x_0)$.*

Proof. Assume to the contrary that $x_i(t) \equiv 0$ and $\mathfrak{v}_i \in \mathfrak{U}_1^\infty(x_0) \setminus \mathfrak{U}_0^\infty(x_0)$. Hence, there exist $k \in \mathbb{N}$ and $\mathfrak{v}_j \in \mathfrak{U}_0^\infty(x_0)$ with $\mathfrak{v}_j \xrightarrow[\mathfrak{G}_d]{k} \mathfrak{v}_i$ or $\mathfrak{v}_i \xrightarrow[\mathfrak{G}(A)]{k} \mathfrak{v}_j$, respectively. It follows from Theorem 4.15 that $\left[A^k\right]_{ij} \neq 0$ and thus $\left[e^{tA}\right]_{ij} \neq 0$, $t > 0$. Since $x_{0,j} \neq 0$, also $x_i(t) \neq 0$ contradicting our assumption that $x_i(t) \equiv 0$.

For the other direction, let $\mathfrak{v}_i \in \mathfrak{U}_2^\infty(x_0)$. Therefore, there exists no path $\mathfrak{v}_j \xrightarrow[\mathfrak{G}_d]{k} \mathfrak{v}_i$ or $\mathfrak{v}_i \xrightarrow[\mathfrak{G}(A)]{k} \mathfrak{v}_j$ for a vertex $\mathfrak{v}_j \in \mathfrak{U}_0^\infty(x_0) \cup \mathfrak{U}_1^\infty(x_0)$. Using Theorem 4.15 again, we get

$$\left[A^k\right]_{ij} = 0 \;\forall k \in \mathbb{N} \;\Rightarrow\; \left[e^{tA}\right]_{ij} = \left[\sum_{k=0}^{\infty} \frac{t^k A^k}{k!}\right]_{ij} = 0$$

and

$$x_i(t) = \left[e^{tA} x_0\right]_i = \sum_{j=1}^{n} \left[e^{tA}\right]_{ij} x_{0,j} = 0.$$

The last equation holds since $\left[e^{tA}\right]_{ij} = 0$ for all $\mathfrak{v}_j \in \mathfrak{U}_0^\infty(x_0) \cup \mathfrak{U}_1^\infty(x_0)$ and $x_{0,j} = 0$ for all $\mathfrak{v}_j \in \mathfrak{U}_2^\infty(x_0)$ by definition. □

Roughly speaking, a vertex is active during the simulation if and only if it is reachable from a vertex with a nonzero initial condition. If $\mathfrak{U}_2^\infty(x_0) \neq \varnothing$, then the matrix A is reducible since $\mathfrak{G}_d$ is not strongly connected and the subsystem which corresponds to $\mathfrak{U}_2^\infty(x_0)$ will remain in the steady state. If, on the other hand, A is irreducible and nonnegative, then Theorem 4.17 implies that $e^{tA} > 0$ for $t > 0$.

The result can be extended to Metzler matrices, i.e. matrices whose off-diagonal entries are nonnegative, since any Metzler matrix M can be written as $M = A - \eta I$, with a nonnegative matrix A. Hence, $x(t) = e^{t(A-\eta I)}x_0 = e^{-\eta t}e^{tA}x_0$. The following example shows that the result does not hold for arbitrary matrices or initial conditions.

Example 4.21 Consider the differential equation $\dot{x}(t) = A\,x(t)$, $x(0) = x_0$, with

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

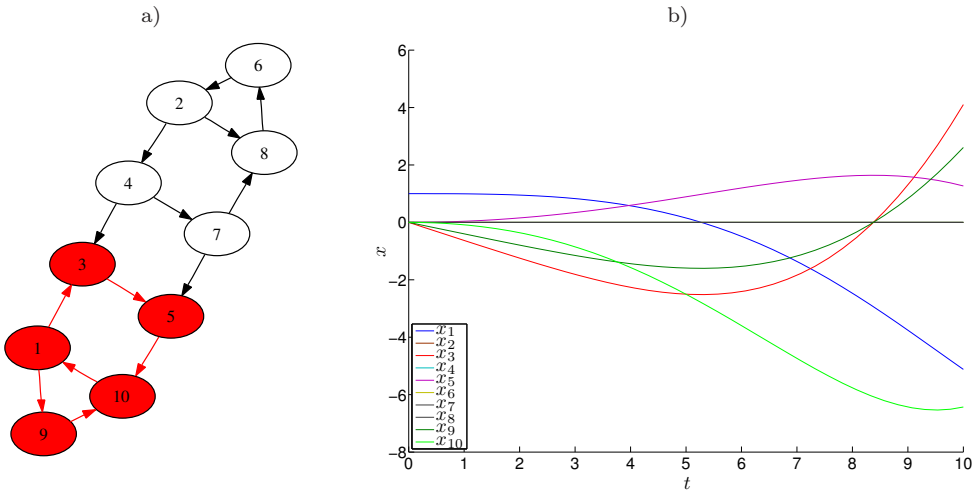


Figure 4.6: Randomly generated linear system. a) Reachable set within the dependency graph. b) Simulation results.

and $x_0 = [1, 0, 0, 0]^T$. Here, $A^k = 0$ for $k \geq 2$ and

$$x(t) = e^{tA} x_0 = (I + tA)\, x_0 = [1, t, 0, t]^T,$$

whereas $\text{reach}^{\infty}_{\mathfrak{G}_d}(\mathfrak{v}_1) = \mathfrak{V}$. ♢

However, for arbitrary linear systems, we get the following corollary.

Corollary 4.22 *Given the ordinary differential equation* $\dot{x}(t) = A\,x(t)$, $x(0) = x_0$. *Then* $x_i(t) \equiv 0$ *if* $\mathfrak{v}_i \in \mathfrak{U}^{\infty}_2(x_0)$.

Example 4.23 Consider the randomly generated linear system $\dot{x}(t) = A\,x(t)$ whose dependency graph is shown in Figure 4.6. Let $x(0) = e_1$, where e_1 denotes the first unit vector. Only the vertices that are reachable from $\mathfrak{v}_1$, namely $\mathfrak{v}_3$, $\mathfrak{v}_5$, $\mathfrak{v}_9$, and $\mathfrak{v}_{10}$, are active during the simulation. The graph consists of two strongly connected components and the permutation matrix P corresponding to the permutation

$$\pi = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 6 & 2 & 7 & 3 & 8 & 9 & 10 & 4 & 5 \end{bmatrix}$$

transforms the matrix into block triangular form (4.20). ♢

In the preceding analysis, we considered the behavior of the analytical solution of linear differential equations. Now, we want to compare these results with the solution obtained by numerical integration schemes. For this purpose, we use results from stability analysis.

Definition 4.24 (Stability function) Let $\dot{x}(t) = A\,x(t)$ be the linear test problem. For a one-step method of the form

$$x^{m+1} = R(hA)\,x^m, \tag{4.23}$$

the function $R(z)$ is defined to be the *stability function* and the set

$$S = \{z \in \mathbb{C} \mid |R(z)| \leq 1\} \tag{4.24}$$

is called the *stability domain*.

Lemma 4.25 *The stability function of a Runge–Kutta method is of the form*

$$R(z) = 1 + zb^T(I - zA)^{-1}\mathbb{1} = \frac{\det(I - zA + z\mathbb{1}b^T)}{\det(I - zA)}, \tag{4.25}$$

where A is now the coefficient matrix of the Runge–Kutta method and $\mathbb{1} = [1, \ldots, 1]^T$.

Proof. See [HW96], for example. □

That is, the stability function can be written as a rational function $R(z) = \frac{P(z)}{Q(z)}$. The polynoms $P(z)$ and $Q(z)$ are of degree less than or equal to s, where s denotes the number of stages. Now, let $R(z)$ be the stability function of a Runge–Kutta method of order p. Since the exact solution of the test problem is e^z, $R(z)$ is a rational approximation to e^z of order p, i.e.

$$R(z) = 1 + z + \frac{z^2}{2} + \cdots + \frac{z^p}{p!} + \mathcal{O}(z^{p+1}) \tag{4.26}$$

or

$$e^z - R(z) = c_{p+1}z^{p+1} + \mathcal{O}(z^{p+2}), \tag{4.27}$$

where c_{p+1} is the error constant of the integration scheme.

If the Runge–Kutta method is explicit, then the coefficient matrix A is strictly lower triangular and $Q(z) \equiv 1$. Thus, $R(z)$ is a polynomial of the form

$$R(z) = 1 + z + \frac{z^2}{2} + \cdots + \frac{z^p}{p!} + r_{p+1}z^{p+1} + \cdots + r_s z^s. \tag{4.28}$$

If, on the other hand, the Runge–Kutta method is implicit, then it is possible to obtain a rational function $R(z)$ with $p = 2s$. In this case, $R(z)$ is the (s, s) Padé approximation to the exponential function [But87]. The stability function of the standard fourth-order Runge–Kutta method, for example, is $R(z) = 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24}$, the stability function of the trapezoidal rule is given by $R(z) = \frac{1+\frac{z}{2}}{1-\frac{z}{2}} = 1 + z + \frac{z^2}{2} + \mathcal{O}(z^3)$.

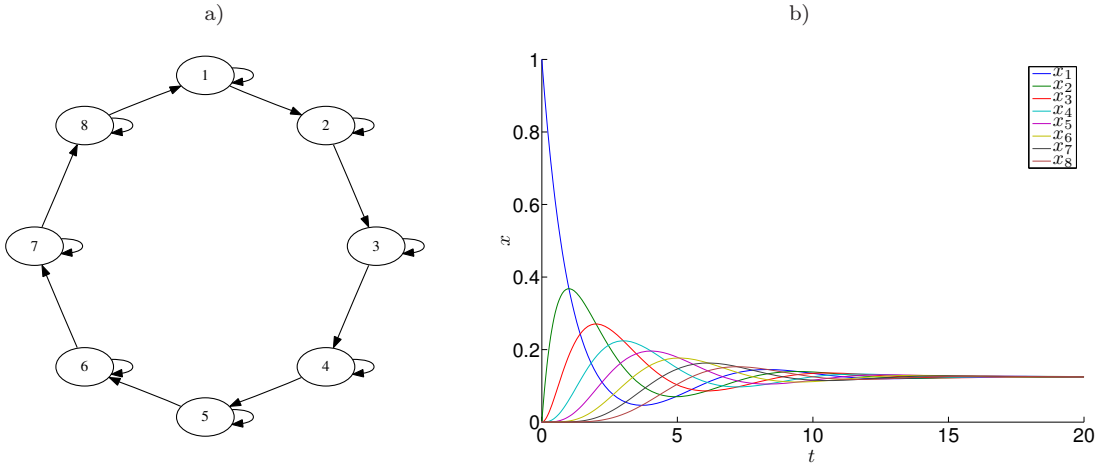


Figure 4.7: Consensus algorithm with eight agents. a) Dependency graph. b) Simulation results.

In Theorem 4.20 and Corollary 4.22, respectively, it was shown that the active regions propagate along the paths of the dependency graph. The following example illustrates the signal propagation with the help of a consensus algorithm.

Example 4.26 Define $\mathfrak{C}_n^T$ to be the transposed directed cycle graph with n vertices and let L be the Laplacian of this graph. Then the linear system $\dot{x}(t) = -L\,x(t)$, $x(0) = e_1$, describes a consensus algorithm for a simple multi-agent networked system [OFM07]. The dependency graph of this system can be obtained from $\mathfrak{C}_n$ by adding self-loops to all vertices. It can be shown that x converges to the equilibrium $x^* = (\alpha, \ldots, \alpha)^T$ with the consensus value $\alpha = \frac{1}{n}$. The dependency graph for $n = 8$ and the solution of the consensus algorithm are shown in Figure 4.7. Since $-L$ is a Metzler matrix and the dependency graph is strongly connected, $x(t) > 0$ for all $t > 0$. That is, all vertices are active. For the numerical solution, this is in general not the case as we will show below. ♢

Lemma 4.27 *Given a linear system $\dot{x}(t) = A\,x(t)$. If $\mathfrak{v}_i \in \mathfrak{U}_2^l(x^m)$, then $\left[A^k x^m\right]_i = 0$ for all $k \leq l$.*

Proof. For $\mathfrak{v}_i \in \mathfrak{U}_2^l(x^m)$, it holds that

$$\left[A^k x^m\right]_i = \sum_{j=1}^{n} \left[A^k\right]_{ij} x_j^m = 0$$

since $\left[A^k\right]_{ij} = 0$ if $\mathfrak{v}_j \in \mathfrak{U}_0^l(x^m)$ and $x_j^m = 0$ if $\mathfrak{v}_j \notin \mathfrak{U}_0^l(x^m)$. □

Corollary 4.28 *Define* $J = \operatorname{diag}(\jmath_1, \ldots, \jmath_n)$ *to be the diagonal projection (cf. [HS09]) on* $\mathfrak{U}_0^l(x^m) \cup \mathfrak{U}_1^l(x^m)$*, i.e.* $\jmath_i = 1$ *if* $\mathfrak{v}_i \in \mathfrak{U}_0^l(x^m) \cup \mathfrak{U}_1^l(x^m)$ *and* $\jmath_i = 0$ *otherwise. Then* $JA^k x^m = A^k x^m$*.*

Lemma 4.29 *Given a linear system* $\dot{x}(t) = A\,x(t)$ *and an explicit s-stage Runge–Kutta method. If* $\mathfrak{v}_i \in \mathfrak{U}_2^s(x^m)$*, then* $x_i^{m+1} = 0$*.*

Proof. The Runge–Kutta method can be written as $x^{m+1} = R(hA)\,x^m$, where $R(z)$ is a polynomial of degree less than or equal to s. □

In contrast to the exact solution

$$x^{m+1} = e^{hA} x^m = \sum_{k=0}^{\infty} \frac{h^k A^k}{k!} x^m \tag{4.29}$$

which contains paths of arbitrary length, explicit Runge–Kutta schemes take into account only paths of finite length. Generally speaking, the region of activity advances at most s vertices per time step and the influence of the paths is decreasing with increasing length.

Theorem 4.30 *Given an explicit s-stage Runge–Kutta method of the form* $x^{m+1} = R(hA)\,x^m$*, define* $\tilde{x}^{m+1}$ *by*

$$\tilde{x}_i^{m+1} = \begin{cases} x_i^m, & \text{if } \mathfrak{v}_i \in \mathfrak{U}_2^s(x^m), \\ \left[R(hA)\,x^m\right]_i, & \text{otherwise.} \end{cases} \tag{4.30}$$

Then $\tilde{x}^{m+1} = x^{m+1}$*.*

Proof. This is a direct consequence of Corollary 4.28 and Lemma 4.29 since

$$\tilde{x}^{m+1} = J(R(hA)\,x^m) + (I - J)\,x^m = R(hA)\,x^m = x^{m+1}.$$ □

As a result, the simulation of the linear system can be restricted to the active part and the part that is reachable from the active part by a path of length $l \leq s$. This observation provides the basis for the analysis of nonlinear time-driven ordinary differential equations and the development of signal-flow based Runge–Kutta methods.

Example 4.31 Consider the consensus algorithm of Example 4.26 again. If we apply the explicit Euler method, then only the first $m + 1$ variables are active at t^m. That is, for each t^m, $m + 1 < n$, we can restrict the simulation to the reduced $(m + 1)$-dimensional linear system. Figure 4.8 shows the activity sets at different time points. Here, the red vertices belong to $\mathfrak{U}_0^1(x^m)$, the yellow vertices to $\mathfrak{U}_1^1(x^m) \setminus \mathfrak{U}_0^1(x^m)$, and the green vertices to $\mathfrak{U}_2^1(x^m)$. ♢

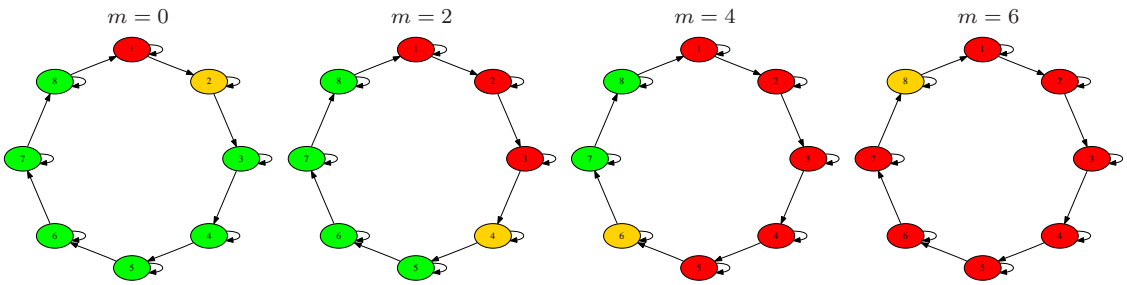


Figure 4.8: The sets $\mathfrak{U}_0^1(x^m)$, $\mathfrak{U}_1^1(x^m)$, and $\mathfrak{U}_2^1(x^m)$ at different time points.

If we apply implicit Runge–Kutta methods, then the simulation cannot be restricted to an equivalent lower-dimensional subsystem. Using the Neumann series, the implicit Euler method, for example, can be written as

$$x^{m+1} = (I - hA)^{-1} x^m = \sum_{k=0}^{\infty} (hA)^k x^m.$$

That is, all possible paths within the dependency graph are taken into account. Consequently, we will not obtain an equivalent signal-flow based method considering only paths of finite length. However, it is possible to construct methods of the same order.

Theorem 4.32 *Given an implicit Runge–Kutta method $x^{m+1} = R(hA)\, x^m$ of order p, define $\tilde{x}^{m+1}$ by*

$$\tilde{x}_i^{m+1} = \begin{cases} x_i^m, & \text{if } \mathfrak{v}_i \in \mathfrak{U}_2^p(x^m), \\ [R(hA)\, x^m]_i, & \text{otherwise.} \end{cases} \tag{4.31}$$

Then $\tilde{x}^{m+1} = x^{m+1} + \mathcal{O}(h^{p+1})$.

Proof. Let J be the diagonal projection on $\mathfrak{U}_0^p(x^m) \cup \mathfrak{U}_1^p(x^m)$. It holds that

$$\begin{aligned} \tilde{x}^{m+1} &= J(R(hA)\, x^m) + (I - J)\, x^m \\ &= J\left(I + hA + \frac{h^2}{2}A^2 + \cdots + \frac{h^p}{p!}A^p + \mathcal{O}(h^{p+1})\right) x^m + (I - J)\, x^m \\ &= x^m + hAx^m + \frac{h^2}{2}A^2 x^m + \cdots + \frac{h^p}{p!}A^p x^m + \mathcal{O}(h^{p+1}) \\ &= \left(I + hA + \frac{h^2}{2}A^2 + \cdots + \frac{h^p}{p!}A^p\right) x^m + \mathcal{O}(h^{p+1}) \\ &= R(hA)\, x^m + \mathcal{O}(h^{p+1}) = x^{m+1} + \mathcal{O}(h^{p+1}). \end{aligned}$$

□

Now, we want to extend the results of this section to arbitrary nonlinear time-driven ordinary differential equations. Our aim is to use the dependency graph as well as numerical information from the integration scheme to identify and take advantage of inactive regions. For this purpose, we will modify Runge–Kutta schemes in such a way that the different rates of activity are automatically taken into account.

4.4 Signal-flow based Runge–Kutta methods

During the simulation of big and loosely coupled networks, different subsystems often exhibit different rates of activity. That is, the values in some parts of the network change rapidly, while in other parts the values change very slowly or do not change at all. The active regions usually vary over time so that a previously inactive region undergoes quick changes and vice versa.

Consider for example the inverter chain. If we apply an input signal, then, generally speaking, this input signal is reversed repeatedly with a small time delay so that it seems to flow continuously through the circuit. The step size control of standard integration schemes depends mainly on the fastest changing variables. As a result, even the inactive signals have to be recomputed at every time step unless multirate integration schemes or other techniques to exploit the latency are used. We will propose an integration scheme which utilizes the underlying structure of the system.

With the definitions in Section 4.3, it is possible to determine which values of x^m are necessary to compute the new values of x^{m+1}, namely, for the update of x_i^m, all values of the variables of the input set $\bullet\, x_i$ are required. Since the external variables $x_{E,i}$, $i \in \langle n_E \rangle$, depend only on the time t, the input sets are empty, i.e. $\bullet\, x_{E,i} = \varnothing$. The update of the internal values $x_{I,i}$, $i \in \langle n_I \rangle$, requires the evaluation of $f_{I,i}$ and thus the values of $\bullet\, x_{I,i}$. To identify latent regions, we have to distinguish between the different vertex types.

Definition 4.33 (Semi-latency) Let t^m be the current time point and t^{m-1} the previous time point.

i) An external variable $x_{E,i}$, $i \in \langle n_E \rangle$, is said to be *semi-latent* at t^m if

$$f_{E,i}(t^m + c_q h) = f_{E,i}(t^{m-1} + c_q h) \tag{4.32}$$

for all $q = 1, \ldots, s$.

ii) An internal variable $x_{I,i}$, $i \in \langle n_I \rangle$, is defined to be *semi-latent* if

$$\Phi_i(t^{m-1}, x^{m-1}, h) = 0. \tag{4.33}$$

The definition implies that $x_{I,i}^m = x_{I,i}^{m-1}$ for all semi-latent internal variables. Whether a vertex is semi-latent at a specific time point is not known until all the values have been evaluated, but since our aim is to reduce the number of function evaluations, we want to mark vertices which need not be recomputed. Therefore, we introduce an additional concept.

Definition 4.34 (Latency) A variable x_i, $i \in \langle n \rangle$, is called *latent of order* 1 if x_i and all variables of the set $\bullet\, x_i$ are semi-latent. Additionally, a latent variable x_i is defined to be *latent of order* ν if all variables in $\bullet\, x_i$ are at least latent of order $\nu - 1$.

For numerical computations, the semi-latency conditions are replaced by $|\Delta x_{E,i}^{m-1}| < \varepsilon$ and $|\Delta x_{I,i}^{m-1}| < \varepsilon$, respectively. Here, ε is a user-defined error tolerance. Let us illustrate the different types of activity with an example.

Example 4.35 If the inverter chain is excited with a given input signal, then this signal flows – reversed at each inverter – through the circuit, as described above. Figure 4.9 shows the voltages and activity states resulting when the circuit is excited with the displayed piecewise linear function. With a view to a better visualization, the respective activity states of the vertices are slightly shifted upward. Clearly, only a few vertices are active at each time point and these active regions flow through the dependency graph. ♢

The example shows that the vertices are latent during the major part of the simulation, but each vertex at a different time. Below, we will propose modified Runge–Kutta methods for time-driven ordinary differential equations which take into account the dependency graph and the signal flow of the underlying system. The aim is to reduce the number of function evaluations without a huge loss of accuracy by exploiting the inherent latency. Since for some applications the function evaluations are time-consuming, whereas the examination of the dependency graph can be accomplished in linear time, this approach offers the possibility to conceivably speed up the simulation.

4.4.1 Explicit Runge–Kutta methods

For the computation of the vectors k_E^q and k_I^q, $q = 1, \ldots, s$, in (4.16), it is necessary to evaluate the functions f_E and f_I, respectively. The functions $f_{I,i}$, $i \in \langle n_I \rangle$, have to be

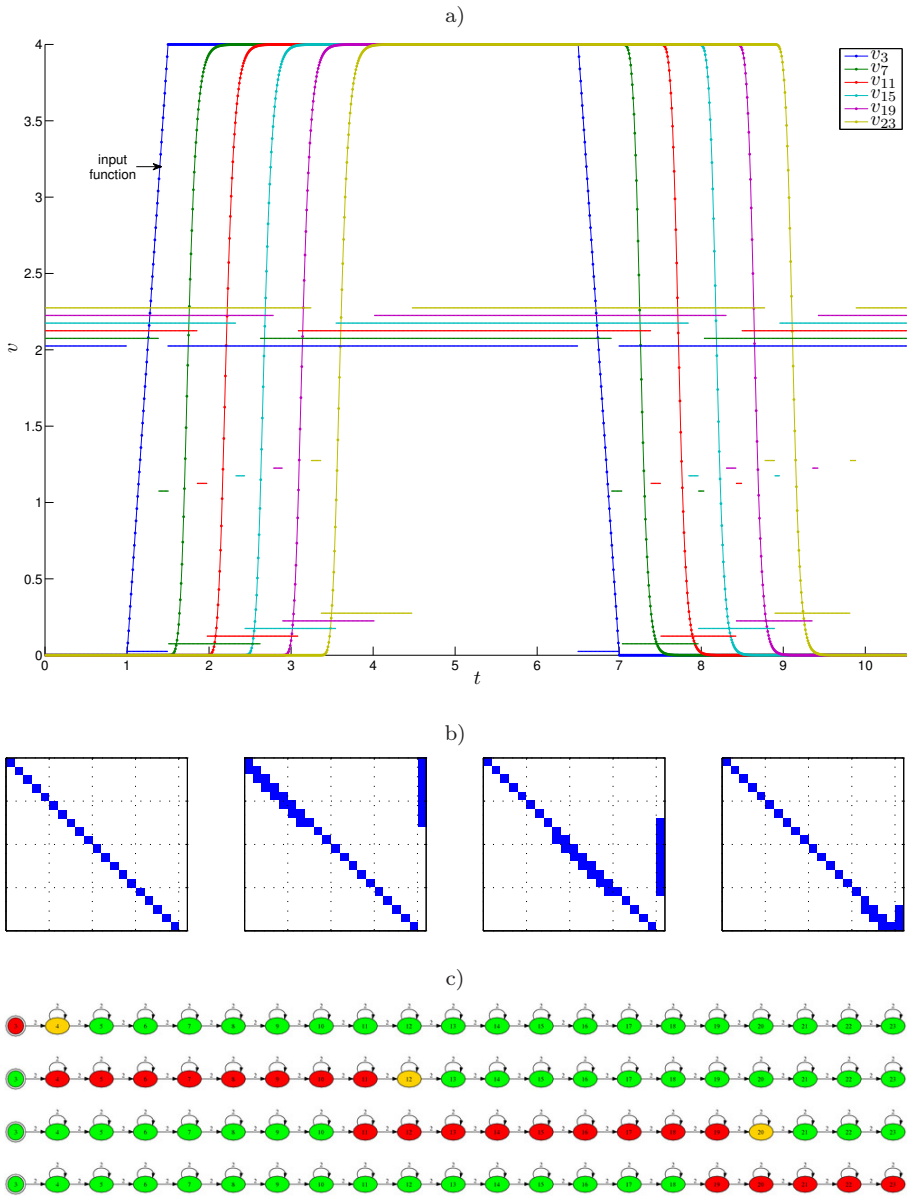


Figure 4.9: Excitation of the inverter chain with a piecewise linear function. a) The dotted trajectories show the input function and the voltages at intermediate vertices, the thin horizontal lines in the corresponding color the activity state. Here, 0 denotes active, 1 semi-latent, and 2 latent, respectively. b) Structure of $\frac{\partial f_I}{\partial x_I}$ and $\dot{x}_I$ at time 1, 2, 3, and 4 for a threshold of 10^{-4}. c) Activity states at time 1, 2, 3, and 4, where red vertices represent active, yellow vertices semi-latent, and green vertices latent regions.

recomputed if only one of the variables of the input set $\bullet x_{I,i}$ is active or semi-latent. If $x_{I,i}$ is latent of a certain order, then we can reuse the previous value.

Definition 4.36 (Signal-flow based Runge–Kutta method) Given a time-driven ordinary differential equation, a *signal-flow based Runge–Kutta method* is defined by

$$\begin{aligned} x_E^{m+1} &= x_E^m + \Delta x_E^m, \\ x_{I,i}^{m+1} &= \begin{cases} x_{I,i}^m, & \text{if } x_{I,i} \text{ is latent of order } s, \\ x_{I,i}^m + \Delta x_{I,i}^m, & \text{otherwise,} \end{cases} \end{aligned} \tag{4.34}$$

for all $i \in \langle n_I \rangle$. Here, s is again the number of stages. The vectors Δx_E^m and Δx_I^m are as defined in (4.15).

Provided that we use exact computation, the following theorem holds.

Theorem 4.37 *The explicit Runge–Kutta methods and the corresponding signal-flow based methods are equivalent.*

Proof. In the proof, we add the superscript m or $m-1$ to the stages to differentiate between the different time points. Let $x_{I,i}$ be latent at t^m, i.e. $\Phi_i(t^{m-1}, x^{m-1}, h) = 0$ and

$$\begin{aligned} f_{E,j}(t^m + c_q h) = f_{E,j}(t^{m-1} + c_q h) \;\Rightarrow\; k_{E,j}^{m,q} = k_{E,j}^{m-1,q} \quad \forall x_{E,j} \in \bullet x_{I,i}, \\ \Phi_j(t^{m-1}, x^{m-1}, h) = 0 \;\Rightarrow\; x_{I,j}^m = x_{I,j}^{m-1} \quad \forall x_{I,j} \in \bullet x_{I,i}. \end{aligned}$$

For $q = 1$, we have $c_1 = 0$ and thus

$$k_{I,i}^{m,1} = f_{I,i}(x_E^m, x_I^m) = f_{I,i}(x_E^{m-1}, x_I^{m-1}) = k_{I,i}^{m-1,1}$$

since $f_{I,i}$ depends only on the values of the input set $\bullet x_{I,i}$ and these values are the same as in the previous time step by definition. Now, assume that $x_{I,i}$ is latent of order 2, i.e. all inputs of $x_{I,i}$ are at least latent of order 1. If follows that

$$\begin{aligned} k_{I,i}^{m,2} &= f_{I,i}(k_E^{m,2}, x_I^m + h\, a_{21} k_I^{m,1}) \\ &= f_{I,i}(k_E^{m-1,2}, x_I^{m-1} + h\, a_{21} k_I^{m-1,1}) = k_{I,i}^{m-1,2} \end{aligned}$$

using the same reasoning again. Furthermore, by induction it can be shown that

$$\begin{aligned} k_{I,i}^{m,q} &= f_{I,i}\big(k_E^{m,q}, x_I^m + h \sum_{r=1}^{q-1} a_{qr} k_I^{m,r}\big) \\ &= f_{I,i}\big(k_E^{m-1,q}, x_I^{m-1} + h \sum_{r=1}^{q-1} a_{qr} k_I^{m-1,r}\big) = k_{I,i}^{m-1,q} \end{aligned}$$

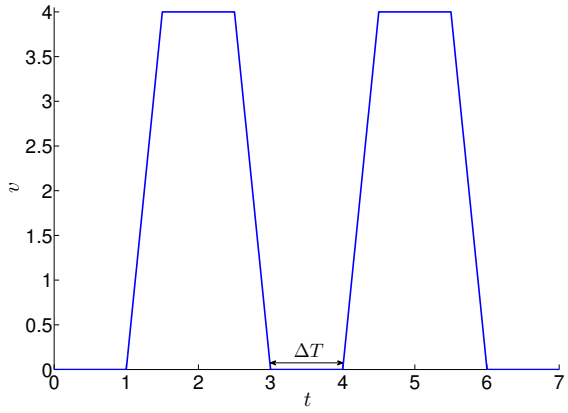


Figure 4.10: Piecewise linear input function with varying delay ΔT to emulate latency.

if $x_{I,i}$ is latent of order q and

$$\begin{aligned} x_{I,i}^{m+1} &= x_{I,i}^{m} + h\,\Phi_i(t^m, x^m, h) \\ &= x_{I,i}^{m} + h \sum_{q=1}^{s} b_q k_{I,i}^{m,q} \\ &= x_{I,i}^{m-1} + h \sum_{q=1}^{s} b_q k_{I,i}^{m-1,q} \\ &= x_{I,i}^{m-1} + h\,\Phi_i(t^{m-1}, x^{m-1}, h) = x_{I,i}^{m} \end{aligned}$$

if $x_{I,i}$ is latent of order s. □

For numerical computations, we do not update a variable if it is latent of order at least one assuming that the influence of longer paths is negligibly small. In the following, we will abbreviate the standard classical fourth-order Runge–Kutta method as RK and the corresponding signal-flow based method as sfRK.

Example 4.38 Consider once again the inverter chain, which is a popular benchmark problem for multirate integration schemes. To analyze the efficiency of the signal-flow based standard Runge–Kutta method, we simulate the inverter chain of length $N = 100$ with variably time-consuming function evaluations and different rates of inherent latency. To vary the amount of latency, we apply periodic input functions with different delays between two adjacent pulse signals, as shown in Figure 4.10. The complexity of the transistor model is increased by artificially adding terms which do not affect the solution of the system.

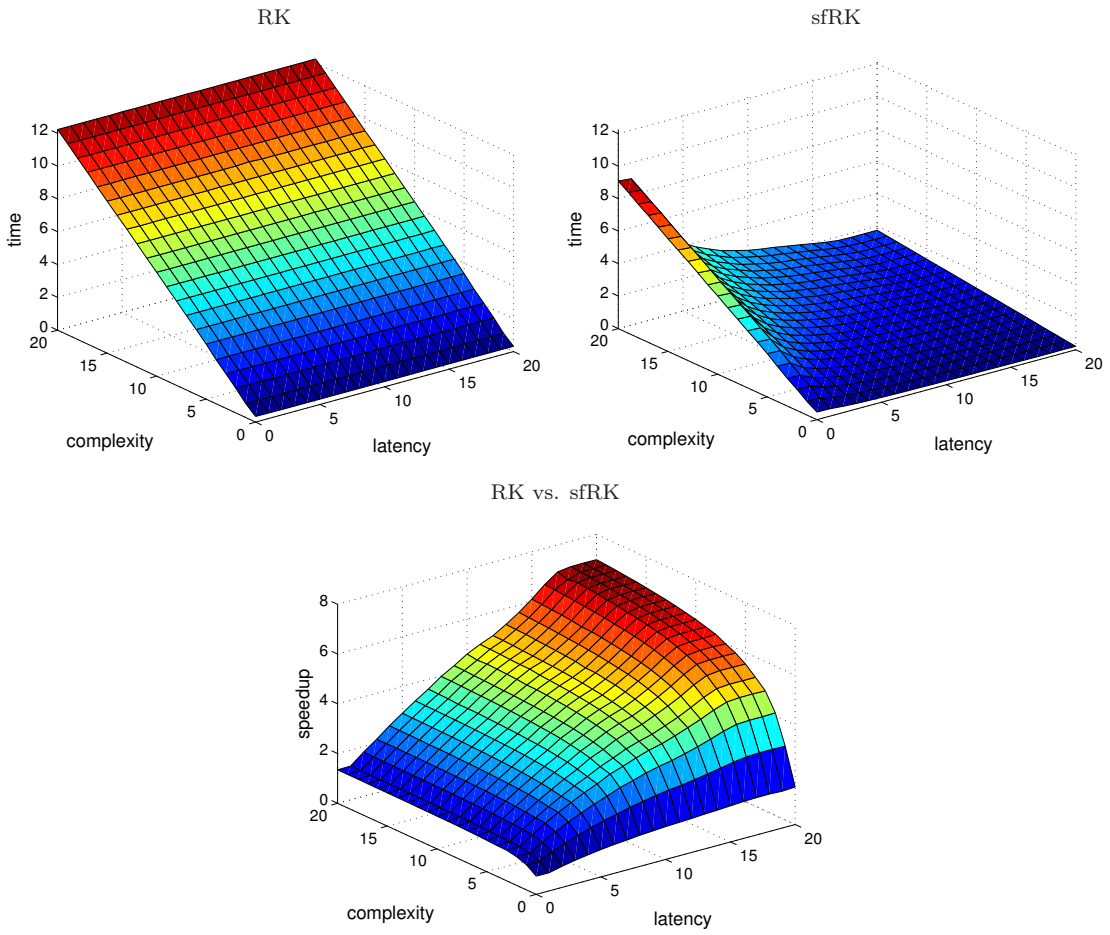


Figure 4.11: Influence of the complexity and latency on the runtime of RK and sfRK.

The runtimes of the simulation with both the standard Runge–Kutta method and the corresponding signal-flow based method for varying model complexities and input functions are shown in Figure 4.11. Here, the time interval is $\mathbb{I} = [0, 40]$, the step size $h = \frac{1}{100}$, and the latency parameter $\varepsilon = 10^{-6}$. While the runtime of RK does not depend on the inherent latency, the runtime of sfRK decreases with increasing latency. Furthermore, the more complex the transistor model is, the larger is the speedup of the signal-flow based integration scheme due to the reduced number of function evaluations. Table 4.1 contains the number of transistor model evaluations for different values of ΔT. The influence of ε on the speedup of sfRK and the average difference per step between RK and sfRK for a fixed delay $\Delta T = 10$ are shown in Figure 4.12.

We can reduce the number of function evaluations even for $\Delta T = 0$ since at the beginning of the simulation the circuit is in a steady state and it takes a short time until the

Table 4.1: Number of transistor model evaluations of RK and sfRK.

ΔT	0	5	10	15	20
RK	3 200 000	3 200 000	3 200 000	3 200 000	3 200 000
sfRK	2 317 152	1 046 664	649 976	479 360	413 024

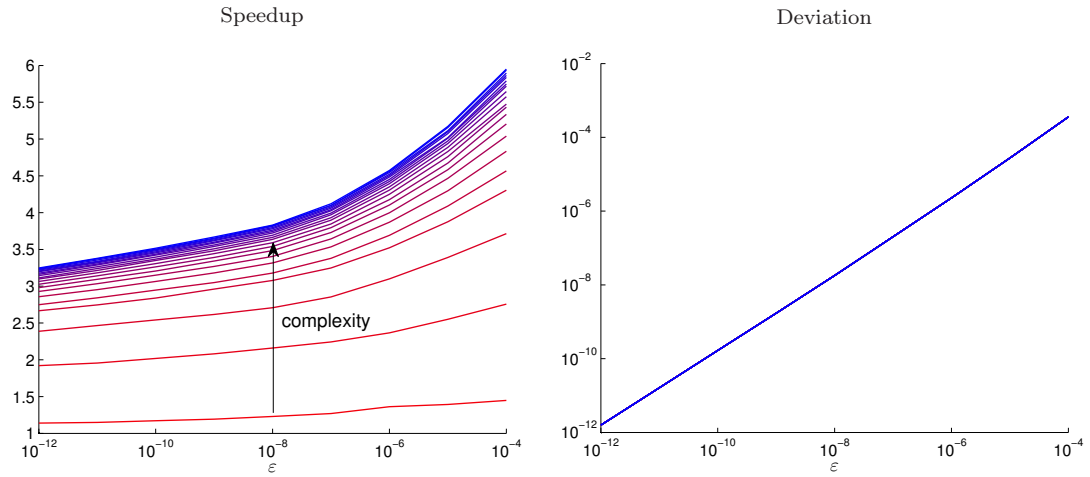


Figure 4.12: Speedup and deviation of sfRK as a function of ε.

input signal reaches the last inverter. During that time, parts of the circuit are inactive and need not be evaluated.

Note that the deviation does not depend on the complexity since only artificial terms were introduced to model different complexities of the transistor model. ♢

The inverter chain contains only one time-dependent voltage source and the efficiency of the signal-flow based integration scheme depends mainly on the shape of the input waveform. If a circuit contains several time-dependent inputs, then the obtainable speedup depends also on the size of the reachable sets and the interaction of different input signals.

Example 4.39 The influence of the inputs of the 4-bit adder varies considerably in size, as described in Example 4.11. Whether the signals are actually propagated from one full adder to the next, however, depends on the summands a and b. If we initially set $a = [0\,0\,0\,0]$ and $b = [0\,0\,0\,0]$ and then switch a_0 from 0 to 1, the active regions are limited to the first full adder. If we, on the other hand, set $b = [1\,1\,1\,1]$ and repeat the computation with the same input signal, then all four full adders are activated. The signal-flow based Runge–Kutta method automatically detects these active regions. Figure 4.13 shows the

dependency graph and the resulting partitioning into active and inactive vertices at a fixed time t. The number of function evaluations required for the first simulation can be reduced by a factor of more than seven. Since during the second simulation the major part of the circuit is active, the number of function evaluations can be reduced only by a factor of less than two. ♢

Remark 4.40 Let n_1 and n_2 be the number of function evaluations of the standard and the corresponding signal-flow based method, respectively. Generally speaking, the speedup of the signal-flow based approach is – according to Amdahl's law [Amd67] – limited by $s \leq \frac{n_1}{n_2}$.

4.4.2 Implicit Runge–Kutta methods

The stages of implicit Runge–Kutta methods cannot be evaluated successively. At each time point, a system of nonlinear equations has to be solved. To solve these systems with the Newton–Raphson method, the Jacobian $\frac{\partial f_I}{\partial x_I}$ has to be computed. For the transient analysis of integrated circuits, this can be accomplished efficiently using so-called element stamps, as described in Section 2.3. Every time the right-hand side f_I is evaluated, the Jacobian $\frac{\partial f_I}{\partial x_I}$ – if needed – is generated simultaneously.

However, only the nonlinear equations that correspond to active regions will be solved assuming that the influence of and on the latent regions is negligibly small. Furthermore, it is then only necessary to compute and factorize the fraction of the Jacobian which represents the active part. That is, we can exploit the latency also on the level of the nonlinear and linear systems of equations. In our implementation, a variable is not updated if it is at least latent of order one, the influence of longer paths is neglected again.

In the following, we will consider in particular the trapezoidal rule, which is frequently used for the simulation of integrated circuits. Since the second version of SPICE most circuit simulators apply either the trapezoidal rule or BDF schemes to solve the circuit equations [GFtM05]. We will denote the trapezoidal rule abbreviatory as TR and the signal-flow based trapezoidal rule as sfTR.

The increment function of the trapezoidal rule tailored to time-driven ordinary differential equations can be written as

$$\Phi(t^m, x^m, h) = \frac{1}{2}\left(f_I(x_E^m, x_I^m) + f_I(x_E^{m+1}, x_I^{m+1})\right). \tag{4.35}$$

That is, at each time step a system of nonlinear equations

$$F(z) := z - x_I^m - \frac{h}{2}\left(f_I(x_E^m, x_I^m) + f_I(x_E^{m+1}, z)\right) = 0 \tag{4.36}$$

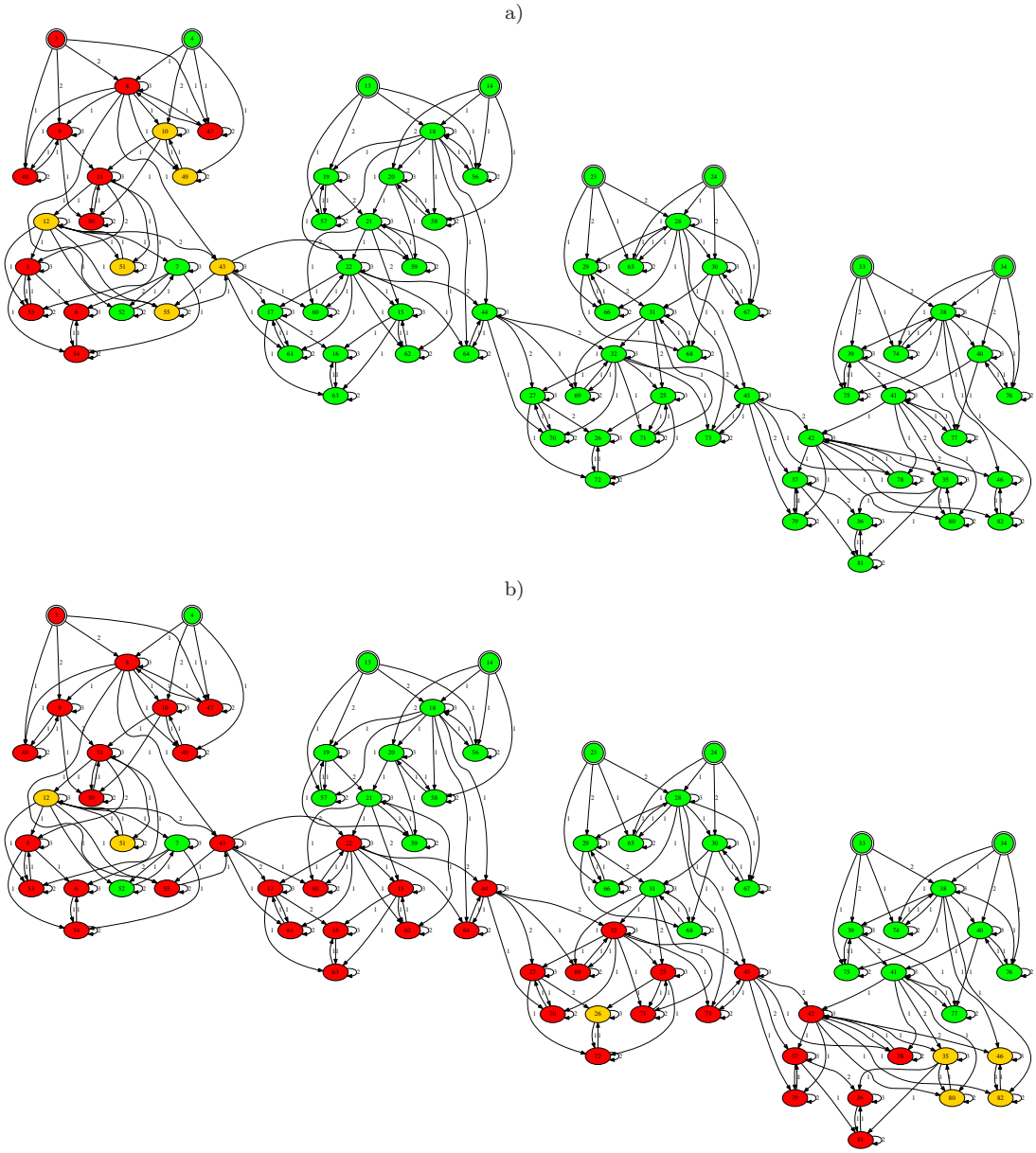


Figure 4.13: Partitioning of the 4-bit adder into active (red), semi-latent (yellow), and latent (green) regions at a fixed time t. a) $a = [0\,0\,0\,\mathbf{1}]$ and $b = [0\,0\,0\,0]$. b) $a = [0\,0\,0\,\mathbf{1}]$ and $b = [1\,1\,1\,1]$.

has to be solved. Using the Newton–Raphson method, this leads to the iteration

$$z_{k+1} = z_k + \Delta z_k, \tag{4.37}$$

where Δz_k is the solution of the linear system of equations

$$\left(I - \frac{h}{2}\frac{\partial f_I}{\partial x_I}(x_E^{m+1}, z_k)\right)\Delta z_k = -z_k + x_I^m + \frac{h}{2}\left(f_I(x_E^m, x_I^m) + f_I(x_E^{m+1}, z_k)\right). \tag{4.38}$$

As a starting point for the iteration, we use $z_0 = x_I^m$.

Example 4.41 To facilitate comparisons of the explicit Runge–Kutta method and the implicit trapezoidal rule, we repeat the simulation of the inverter chain of length $N = 100$ with the settings described in Example 4.38. Figure 4.14 shows the runtimes of the simulation with both the standard trapezoidal rule and the signal-flow based trapezoidal rule for varying model complexities and input functions again. We use the Newton–Raphson method to solve the nonlinear systems and the LU factorization to solve the resulting linear systems of equations. For the signal-flow based simulation, only the active and semi-latent parts of the nonlinear and linear systems of equations are generated and solved. Here, the influence of the model complexity is negligible since the runtime of the LU factorization is dominating. Table 4.2 contains the number of required transistor model evaluations. The influence of ε on the speedup of sfTR and the average deviation per step for a fixed delay $\Delta T = 10$ are shown in Figure 4.15.

If the delay ΔT of the input function is larger than 12 or the period is larger than 14, respectively, then the trapezoidal rule depends on the latency. This is due to the fact that the signal needs approximately this period of time to pass all inverters. For larger values of ΔT, there is a small time interval where all vertices are latent and thus the Newton–Raphson method needs less iterations to converge. ♢

Table 4.2: Number of transistor model evaluations of TR and sfTR.

ΔT	0	5	10	15	20
TR	2 353 600	2 353 600	2 353 600	2 075 200	1 881 600
sfTR	1 736 618	784 214	486 788	357 118	307 582

Here, we did not apply sparse matrix techniques to store and solve the systems of equations. For large-scale examples, however, this is essential. The use of sparse matrix libraries would decrease the runtime required for the LU factorization and therefore increase the speedup of the signal-flow based trapezoidal rule.

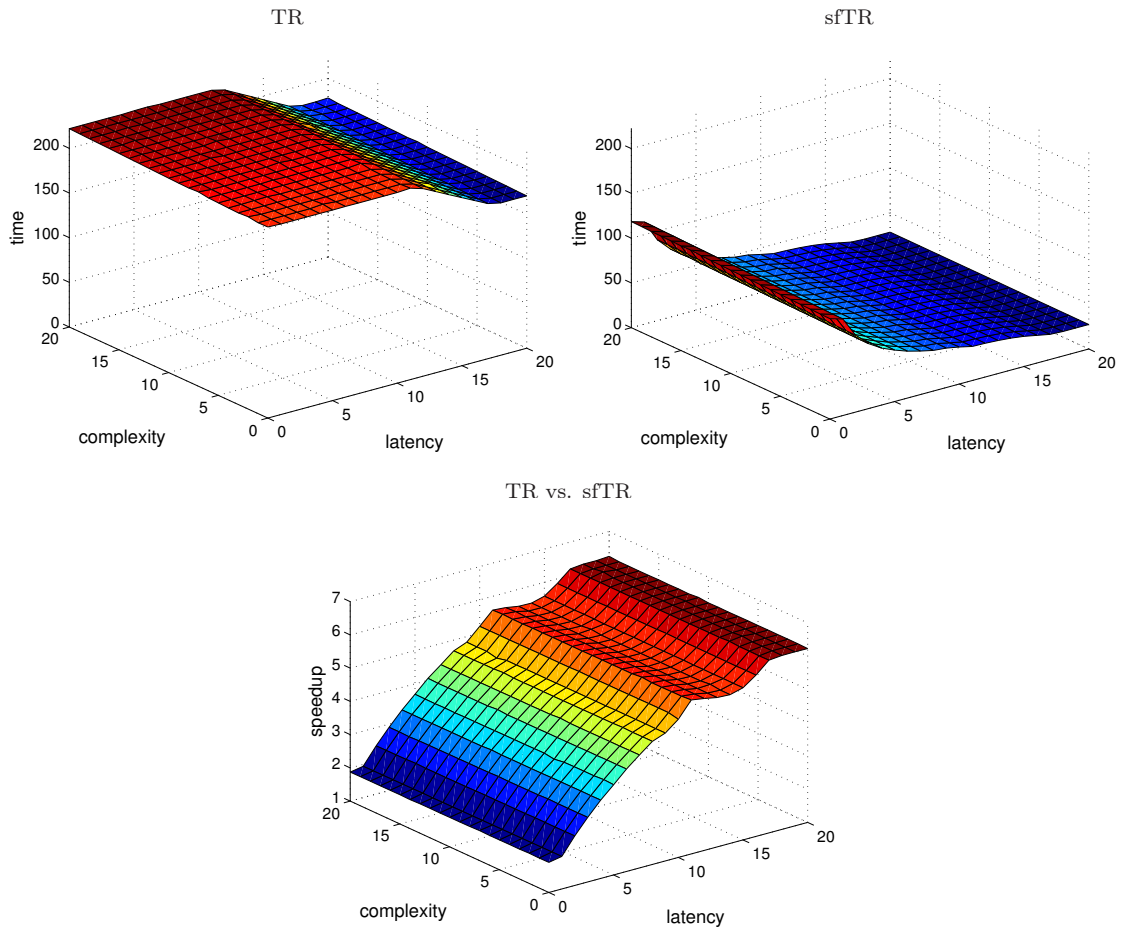


Figure 4.14: Influence of the complexity and latency on the runtime of TR and sfTR.

Example 4.42 As described in Example 4.7, the medical Akzo Nobel problem can be reduced to a stiff time-driven ordinary differential equation. We set $N = 60$, $k = 100$, $c = 4$, $v_0 = 1$, and choose the initial condition $x_{I,0} = [0, v_0, 0, v_0, \ldots, 0, v_0]^T$. The input function is defined to be

$$\phi(t) = \begin{cases} 2, & \text{for } t \in (0, 5], \\ 0, & \text{for } t \in (5, 10]. \end{cases}$$

The solution and the resulting active and inactive regions are shown in Figure 4.16. Note that a region is only marked latent if both u and v are latent at the same time. For $h = 0.001$ and $\varepsilon = 10^{-8}$, the number of function evaluations can be reduced by a factor of 1.8. The obtained speedup is approximately 1.6. ♢

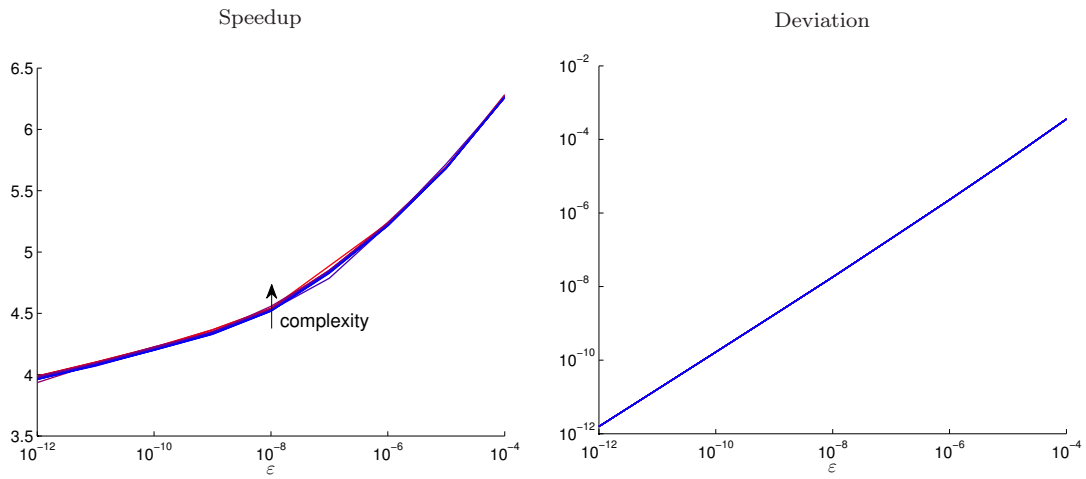


Figure 4.15: Speedup and deviation of sfTR as a function of ε.

Remark 4.43 Let n_1 and n_2 be again the number of function evaluations of the standard and the corresponding signal-flow based method, respectively. In contrast to the explicit Runge–Kutta methods, the speedup of the signal-flow based trapezoidal rule is not necessarily limited by $s \leq \frac{n_1}{n_2}$ since in addition to the number of function evaluations also the size of the resulting nonlinear and linear systems of equations can be reduced.

4.4.3 Generalization to periodic systems

In power electronic circuits, diodes and semiconductor switches are constantly changing their status and a steady state condition is by definition reached when the waveforms are periodic with a time period T which depends on the specific nature of the circuit [MUR95]. The time scales of these circuits may differ by several orders of magnitude and the simulation requires very small step sizes to cover the dynamics of the fastest subsystems. The maximum simulation time, on the other hand, is usually determined by the slowest subsystems. Thus, a detailed simulation of power electronic circuits is in general very time-consuming. The following motivating example illustrates the above definition of a steady state.

Example 4.44 Consider the three-phase diode-bridge rectifier in Figure 4.17a. This configuration is often used in industrial applications to convert the AC input into a DC voltage in an uncontrolled manner [MUR95]. The simulation results are shown in Figure 4.17b. Here, the frequency is 60 Hz and hence $T = \frac{1}{60}$ s. ♢

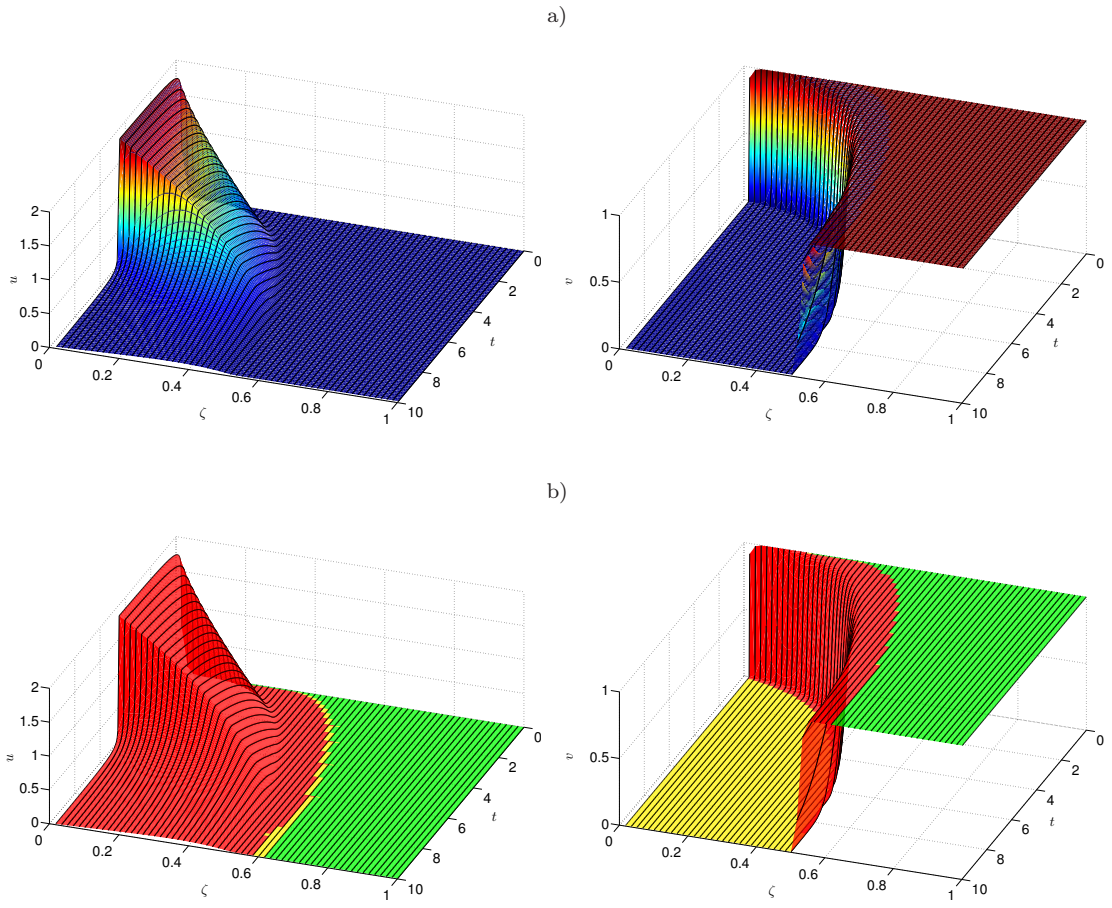


Figure 4.16: Simulation results for the medical Akzo Nobel problem. a) Solution u and v as a function of t and ζ. b) Decomposition of the surface into active (red), semi-latent (yellow), and latent (green) regions.

Now, we want to extend the signal-flow based approach to identify and exploit not the latency but the periodicity of subsystems in order to reduce the runtime of the simulation.

Definition 4.45 (Semi-periodicity) Let T be the fundamental period of the system and $h = \frac{T}{p}$, $p \in \mathbb{N}$, the step size.

i) An external variable $x_{E,i}$, $i \in \langle n_E \rangle$, is said to be *semi-periodic* at t^m if

$$f_{E,i}(t^m + c_q h) = f_{E,i}(t^{m-p} + c_q h) \tag{4.39}$$

for all $q = 1, \ldots, s$.

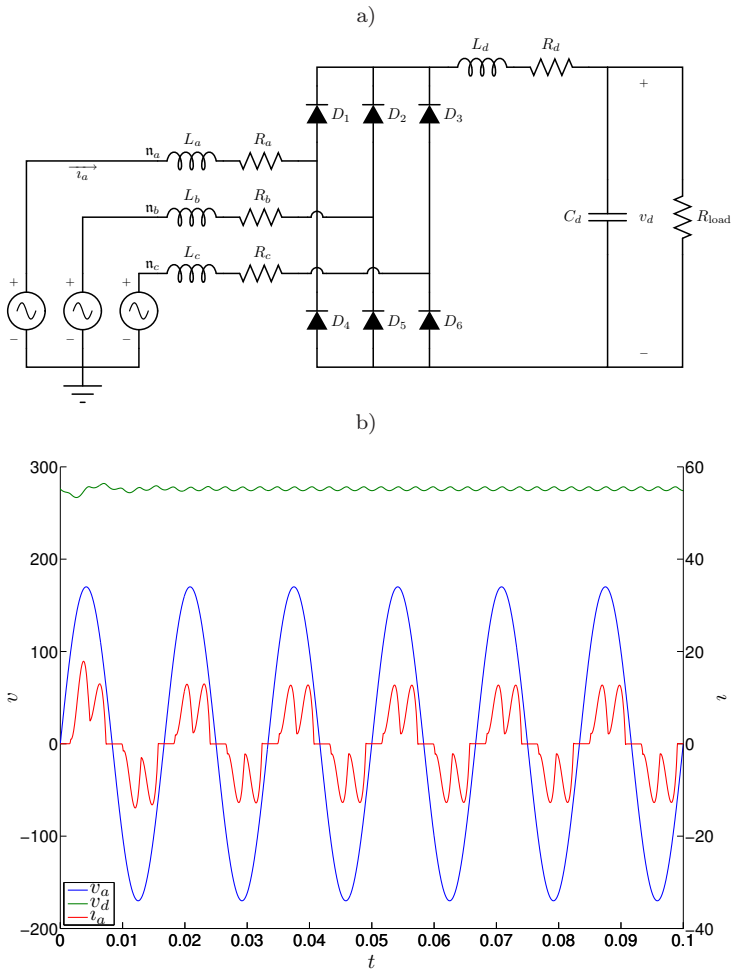


Figure 4.17: Three-phase rectifier. a) Circuit schematics. b) Simulation results.

ii) An internal variable $x_{I,i}$, $i \in \langle n_I \rangle$, is defined to be *semi-periodic* if

$$x_{I,i}^{m} = x_{I,i}^{m-p}. \tag{4.40}$$

In contrast to the definition of semi-latency, the variables are not compared to the previous time step, but to the corresponding time step of the previous period. Roughly speaking, latency can be regarded as a special case of periodicity for which $p = 1$.

Definition 4.46 (Periodicity) A variable x_i, $i \in \langle n \rangle$, is called *periodic of order* 1, if x_i and all variables of the set $\bullet\, x_i$ are semi-periodic. Additionally, a periodic variable x_i is defined to be *periodic of order* ν if all variables in $\bullet\, x_i$ are at least periodic of order $\nu - 1$.

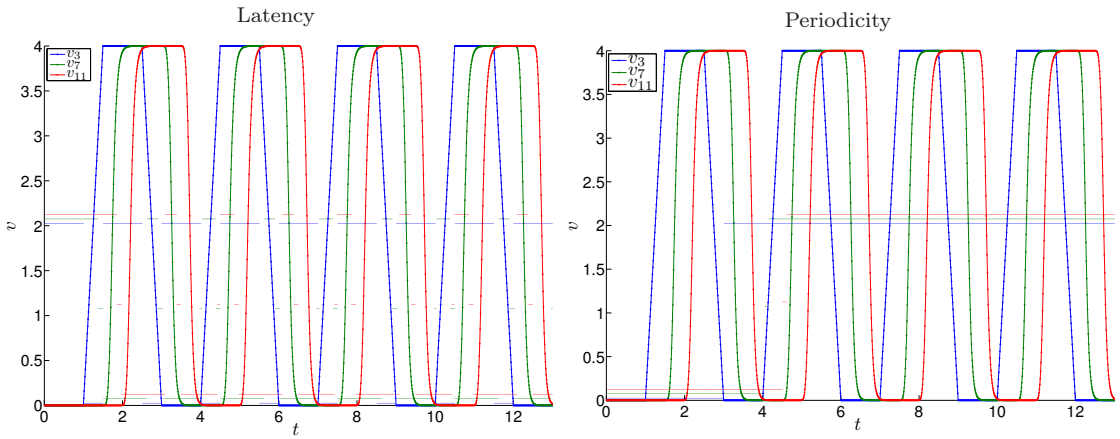


Figure 4.18: Comparison of latency and periodicity. The curves show the node voltages v_3, v_7, and v_{11}, the thin horizontal lines the corresponding states of the variables. Here, 0 denotes active, 1 semi-latent or semi-periodic, and 2 latent or periodic, respectively.

Let ε be again a given error tolerance. For numerical computations, the semi-periodicity conditions are replaced by $|x_{E,i}^m - x_{E,i}^{m-p}| < \varepsilon$ and $|x_{I,i}^m - x_{I,i}^{m-p}| < \varepsilon$, respectively. Analogously to the latency-based methods, we do not update a variable if it is periodic of order one or higher. To illustrate the different activity states, we use the inverter chain.

Example 4.47 The inverter chain is excited with a piecewise linear function which is periodic with $T = 4$ for $t > 1$. The input function and the resulting node voltages at intermediate vertices are shown in Figure 4.18. ♢

Definition 4.48 (Signal-flow based periodic Runge–Kutta method) Given a time-driven ordinary differential equation, an explicit *signal-flow based periodic Runge–Kutta method* is defined by

$$\begin{aligned} x_E^{m+1} &= x_E^m + \Delta x_E^m, \\ x_{I,i}^{m+1} &= \begin{cases} x_{I,i}^{m-p+1}, & \text{if } x_{I,i} \text{ is periodic of order } s, \\ x_{I,i}^m + \Delta x_{I,i}^m, & \text{otherwise,} \end{cases} \end{aligned} \tag{4.41}$$

for $i \in \langle n_I \rangle$.

To exploit the periodicity of subsystems and to reduce the number of function evaluations, we store the vectors $x^{m-p+1}, x^{m-p+2}, \ldots, x^m$ in a circular buffer.

Theorem 4.49 *The explicit Runge–Kutta methods and the corresponding signal-flow based methods for periodic systems are equivalent.*

Proof. The proof is almost identical to the proof of Theorem 4.37. We add again the superscript m or $m-p$ to the stages to differentiate between the time points. Let $x_{I,i}$ be periodic at t^m, i.e. $x_{I,i}^m = x_{I,i}^{m-p}$ and

$$f_{E,j}(t^m + c_q h) = f_{E,j}(t^{m-p} + c_q h) \quad \forall x_{E,j} \in \bullet\, x_{I,i},$$
$$x_{I,j}^m = x_{I,j}^{m-p} \quad \forall x_{I,j} \in \bullet\, x_{I,i}.$$

For $q = 1$, this yields

$$k_{I,i}^{m,1} = f_{I,i}(x_E^m, x_I^m) = f_{I,i}(x_E^{m-p}, x_I^{m-p}) = k_{I,i}^{m-p,1}$$

and hence by induction

$$\begin{aligned} k_{I,i}^{m,q} &= f_{I,i}\big(k_E^{m,q}, x_I^m + h \sum_{r=1}^{q-1} a_{qr} k_I^{m,r}\big) \\ &= f_{I,i}\big(k_E^{m-p,q}, x_I^{m-p} + h \sum_{r=1}^{q-1} a_{qr} k_I^{m-p,r}\big) = k_{I,i}^{m-p,q} \end{aligned}$$

for each variable $x_{I,i}$ which is periodic of order q. Consequently,

$$\begin{aligned} x_{I,i}^{m+1} &= x_{I,i}^m + h\, \Phi_i(t^m, x^m, h) \\ &= x_{I,i}^m + h \sum_{q=1}^{s} b_q k_{I,i}^{m,q} \\ &= x_{I,i}^{m-p} + h \sum_{q=1}^{s} b_q k_{I,i}^{m-p,q} \\ &= x_{I,i}^{m-p} + h\, \Phi_i(t^{m-p}, x^{m-p}, h) = x_{I,i}^{m-p+1}, \end{aligned}$$

for each $x_{I,i}$ which is periodic of order s. □

Let sfpRK denote the signal-flow based standard fourth-order Runge–Kutta method for periodic systems.

Example 4.50 To compare the signal-flow based method for periodic systems with the standard Runge–Kutta method, we simulate the inverter chain as described in Example 4.38. The results are shown in Figure 4.19 and Table 4.3. Here, the number of function evaluations rises with increasing ΔT since the time interval in which the system is periodic according to our definition decreases. ♢

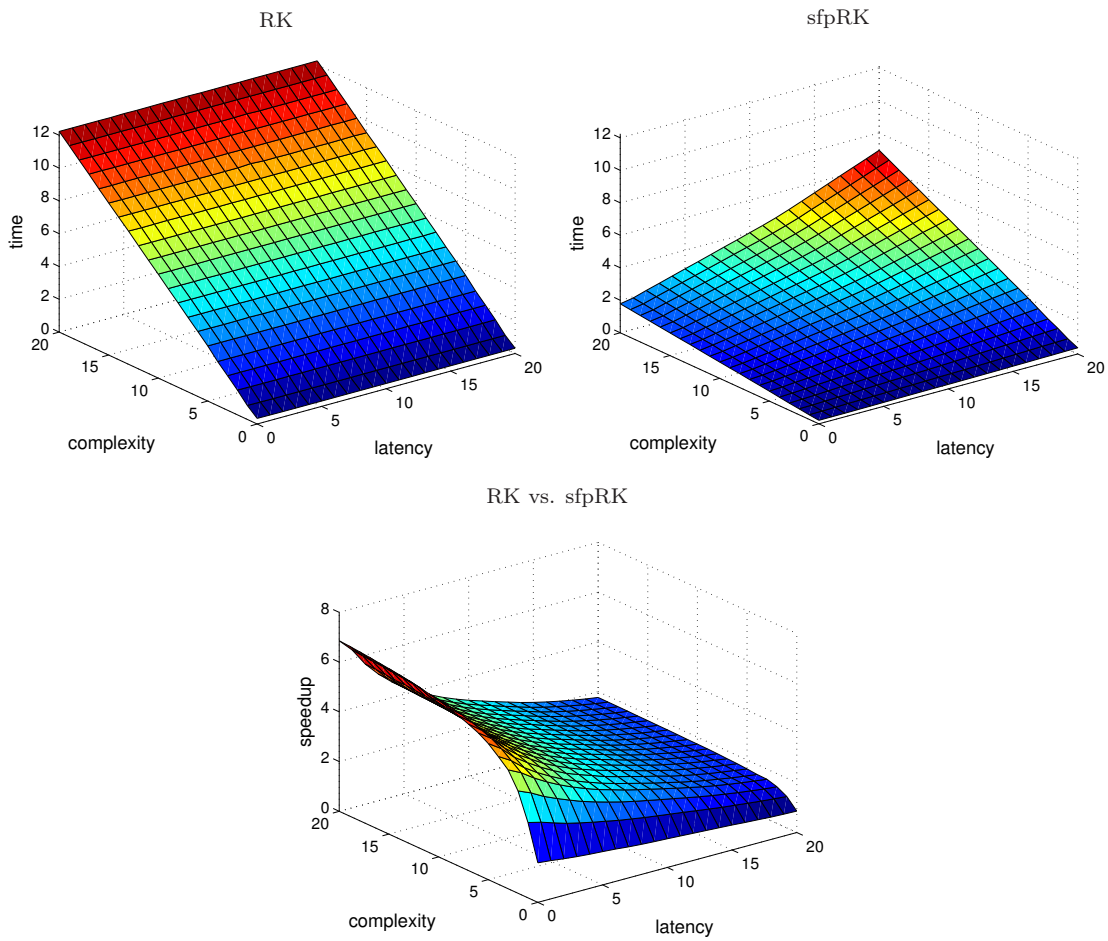


Figure 4.19: Influence of the complexity and latency on the runtime of RK and sfpRK.

4.4.4 Comparison and concluding remarks

The efficiency of the signal-flow based Runge–Kutta methods depends strongly on the characteristic properties of the system. The inverter chain example shows that if during the simulation large parts of the system are latent and function evaluations are comparatively time-consuming, then the signal-flow based methods result in a substantially reduced runtime while introducing only a small deviation compared to the corresponding standard Runge–Kutta methods. If, on the other hand, large parts are periodic with a fundamental period T, then the signal-flow based methods for periodic systems can be used to speed up the simulation. The following example summarizes these results.

Example 4.51 Figure 4.20 shows a comparison of the signal-flow based standard Runge–Kutta method and the corresponding method for periodic systems. If T is small, then the

Table 4.3: Number of transistor model evaluations of RK and sfpRK.

ΔT	0	5	10	15	20
RK	3 200 000	3 200 000	3 200 000	3 200 000	3 200 000
sfpRK	422 328	700 936	999 672	1 360 800	1 760 800

periodicity-oriented Runge–Kutta method is more efficient since the circuit is active most of the time. With increasing T, the latency exploitation becomes more efficient. ♢

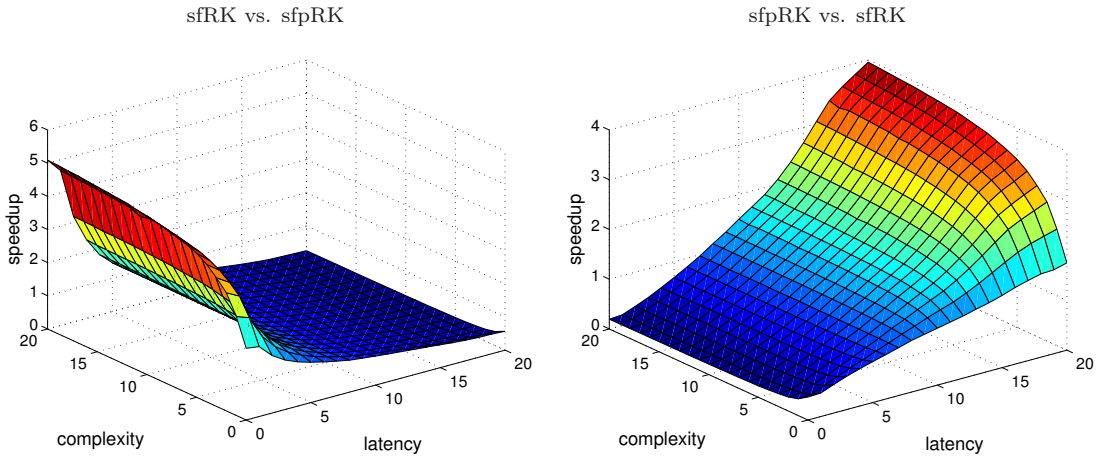


Figure 4.20: Comparison of sfRK and sfpRK.

5 Implementation

In this chapter, we briefly describe the software tool *signalflow*, which was written during the preparation of this thesis as an experimental platform for the analysis and improvement of the newly developed algorithms. All algorithms presented in Chapter 3 and Chapter 4 were implemented using the object-oriented programming language C++. The advantage of C++ is that it enables a flexible, extensible, and yet efficient implementation of numerical algorithms [KM99]. Furthermore, it is in widespread use in industry. Some algorithms, the methods to analyze the signal-flow of integrated circuits and the switch-level based simulation for the approximate computation of operating points, have also been integrated into an industrial circuit simulator.

Figure 5.1 shows a diagram of the program structure. Our software tool consists of three different parts: the input processor, the numerical library, and the circuit simulation kernel. The overall code comprises more than 13 000 lines of code. The modeling of the circuit, the signal-flow analysis, and the switch-level simulation alone cover approximately 8 000 lines of code. Below, we illustrate the capabilities of *signalflow* using the example of the clock generation circuit introduced in Example 3.2.

In order to enable a convenient and efficient specification of integrated circuits, standard gates can be described by predefined subcircuits. The textual description and the

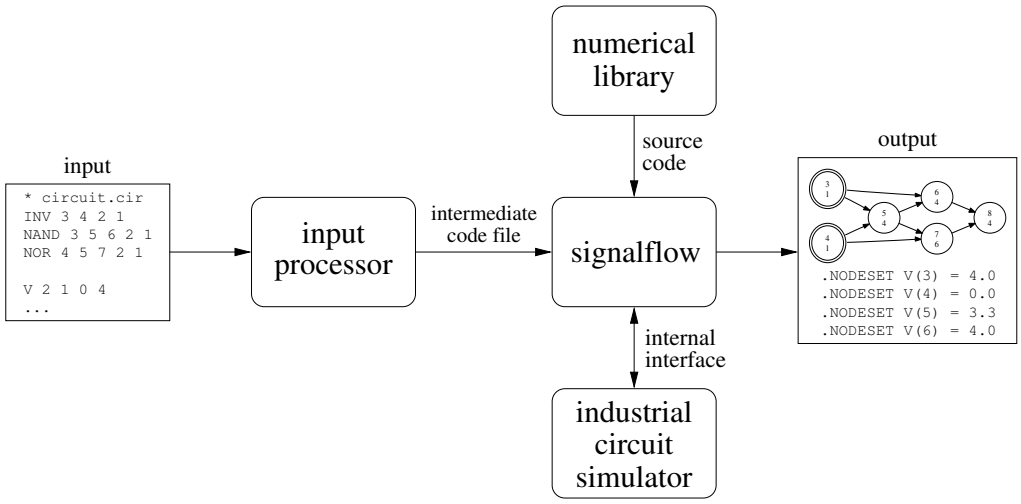


Figure 5.1: Schematic diagram of the program structure.

corresponding schematic diagram of the clock generation circuit are shown in Figure 5.2. The circuit consists of two NAND gates, seven inverters, and two voltage sources. The first voltage source provides the supply voltage V_{dd}, the second voltage source generates a piecewise linear clock signal. Node $\mathfrak{n}_1$ is always defined to be the ground node. The input processor converts this circuit description into an intermediate code file. The intermediate code file is similar to a flattened SPICE netlist and consists of a list of all modules and a specification of their interconnection. This file interface was in particular designed to allow for a simple description of large integrated circuits and to enable the development and optimization of the algorithms independently of the circuit simulator of our industry partner.

```
* clockgen.cir
INV 3 4 2 1
NAND 3 10 5 2 1
NAND 4 9 6 2 1
INV 5 7 2 1
INV 6 8 2 1
INV 7 9 2 1
INV 8 10 2 1
INV 9 11 2 1
INV 10 12 2 1
V 2 1 0 4
V 3 1 1 0 4 5 2.5 2.5 10 25
```

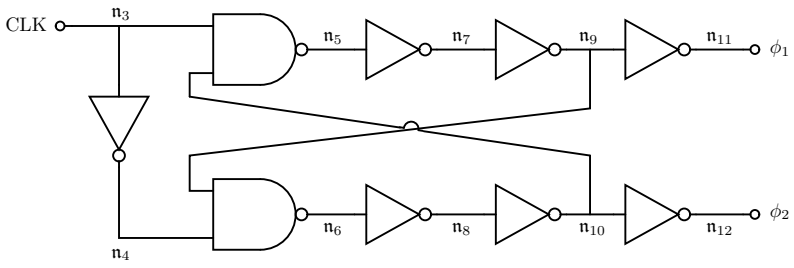


Figure 5.2: Gate-level description of the clock generation circuit.

The circuit simulation kernel parses the intermediate code file and builds the corresponding circuit hypergraph. In order to bypass the parsing of the textual circuit description, parts of *signalflow* were directly integrated into the aforementioned industrial circuit simulator so that now also a more efficient internal interface can be used to generate the circuit structure.

After the assembly of the circuit hypergraph, all algorithms described in Chapter 3 can be applied. It is, for instance, possible to decompose the circuit into channel-connected and strongly connected components and to compute the component graph. The results of the signal-flow analysis can be written into supplementary text files. Furthermore, it is also possible to generate a graphical representation of the component graph. For the graph visualization, we use the open source software *graphviz*[1]. The clock generation circuit comprises nine channel-connected components and one signal-input voltage source. The resulting component graph contains only one nontrivial strongly connected component. Figure 3.1 shows the results of the partitioning and signal-flow analysis.

Another main feature of *signalflow* is the possibility to compute an initial guess for the operating point analysis with the aid of the extended switch-level simulation. The results of the switch-level simulation can be either directly submitted to the nonlinear solver of the circuit simulator or added to the netlist in form of additional NODESET statements. Moreover, it is possible to visualize the coverage and to display the critical configurations of the circuit. Examples of such graphs are shown in Figure 3.8 and Figure 3.9. The switch-level simulation of the clock-generation circuit yields the results shown in Figure 5.3. The nodes $\mathfrak{n}_{13}$ and $\mathfrak{n}_{14}$ are the internal nodes of the two NAND gates. Here, $\mathfrak{n}_{13}$ remains uninitialized after the switch-level simulation since it is only connected to nonconducting modules. The clock generation circuit does not contain critical configurations, all inputs and outputs of the channel-connected components are well-defined.

In addition to the switch-level simulation, we developed a numerical library for the transient simulation of regularized CMOS circuits in order to verify and improve the signal-flow based integration schemes described in Chapter 4. The numerical library is independent of the circuit-related part an can be used for arbitrary complex dynamical networks with inherent latency or periodicity. It provides several different explicit and implicit Runge–Kutta schemes and the corresponding signal-flow based counterparts as well as methods for an adaptive step-size control. To assemble the circuit equations, we implemented a nodal-analysis based algorithm which directly generates the time-driven

[1] www.graphviz.org

```
.NODESET V(1) = 0
.NODESET V(2) = 4
.NODESET V(3) = 0
.NODESET V(4) = 4
.NODESET V(5) = 4
.NODESET V(6) = 0
.NODESET V(7) = 0
.NODESET V(8) = 4
.NODESET V(9) = 4
.NODESET V(10) = 0
.NODESET V(11) = 0
.NODESET V(12) = 4
* .NODESET V(13) = Z
.NODESET V(14) = 0
```

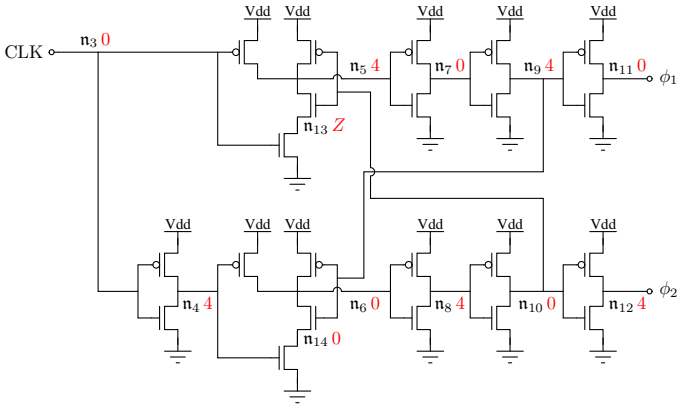


Figure 5.3: NODESETs for the clock generation circuit.

ordinary differential equation (4.13). The dependency graph is computed automatically prior to the simulation. This can be accomplished efficiently using the network topology, as described in Section 4.3. Our software tool also provides a graphical user interface which can be used to plot the voltages and the activity states of the individual nodes. The simulation of the clock generation circuit yields the results shown in Figure 5.4. For the sake of consistency, we replotted the trajectories with MATLAB.

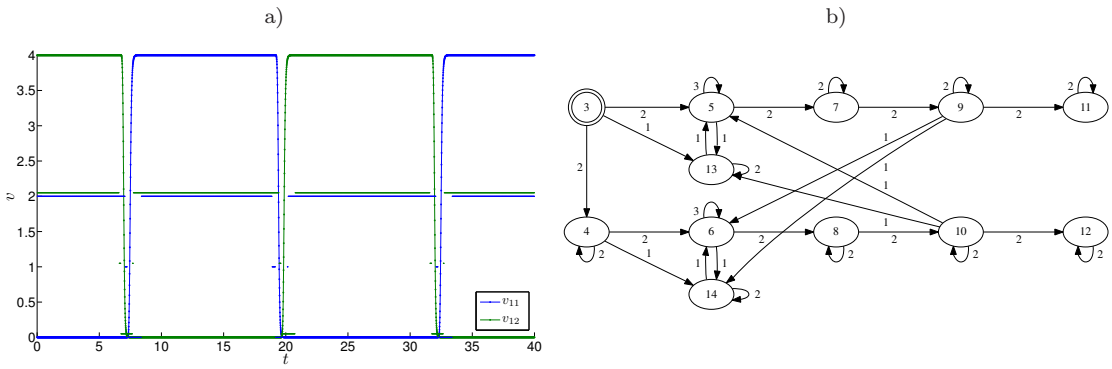


Figure 5.4: Simulation of the clock generation circuit. a) Output trajectories and activity states. b) Dependency graph.

6 Conclusion

Analog circuit simulators such as SPICE play an important role in the design process of integrated circuits. Over the last decades, the complexity and functionality of integrated circuits rose significantly. As a consequence, a detailed simulation of the entire circuit is often very time-consuming. The combination of Kirchhoff's laws and the characteristic equations of the basic circuit elements leads in general to a system of differential and algebraic equations. The standard approach to solve the circuit equations consists of mainly two steps: the computation of consistent initial values and the subsequent integration of the differential-algebraic equation with implicit one-step or multi-step methods. Usually the network topology is only used to generate the circuit equations, but not to solve these equations numerically. Nevertheless, the network topology is implicitly encoded in the circuit equations. In this thesis, we proposed different techniques to speed up the simulation of integrated circuits exploiting the underlying network structure.

We introduced two different directed graphs to model the signal flow of integrated circuits, namely the component graph and the dependency graph. The component graph, which is based on the partitioning into channel-connected components, can be regarded as a model of the logic signal flow. We used this graph to improve the results of the extended switch-level simulation defined in Chapter 3 in order to generate an appropriate

starting point for the computation of consistent initial conditions. To obtain consistent initial conditions, a system of nonlinear equations has to be solved. The Newton–Raphson method usually fails to converge to a solution. Homotopy-based methods, on the contrary, are comparatively slow. If the starting point provided by the switch-level simulation is sufficiently close to a solution of the system of nonlinear equations, then the Newton–Raphson method converges quickly. This leads to a considerably reduced runtime of the operating point analysis.

The dependency graph, which is based on the structure of the circuit equations, on the other hand, is tailored to the numerical integration of the circuit equations. During the simulation, the major part of the circuit is in general inactive. Conventional integration schemes discretize the entire system with a single step size which is mainly dictated by the active subsystems. As a consequence, inactive subsystems are simulated with an unnecessarily high accuracy. We utilize the dependency graph to identify temporarily inactive subsystems. The splitting of the system into active and inactive subsystems is then used to design signal-flow based Runge–Kutta methods which recompute only the active parts of the circuit. With the aid of the adapted integration schemes, the number of function evaluations and thus the time required for the numerical integration can be minimized.

In summary, the following results have been achieved:

1. Generation of signal-flow graphs for both the switch-level and the circuit-level simulation.

2. Computation of operating points or consistent initial values based on an extended switch-level model.

3. Analysis of the influence of the network structure on the dynamic behavior of complex systems.

4. Development of signal-flow based integration schemes for the exploitation of the inherent latency or periodicity.

The proposed methods illustrate that it is possible to speed up the simulation of complex dynamical systems using information on the underlying network structure. This motivates a further development of signal-flow based simulation techniques. There are many possible future directions to extend and generalize the described approach.

Parallel simulation

The dependency graph of a time-driven ordinary differential equation could also be utilized to generate a signal-flow based partitioning of the system for a subsequent parallel simulation. If the dependency graph is not strongly connected, then the horizontal-vertical decomposition [Mez04, VKLM04] can be used to generate a hierarchy of weakly coupled subsystems that preserves the directionality of the signal flow. The strongly connected subsystems in turn could be further decomposed using standard partitioning libraries such as, for instance, PARTY [Pre00]. With the aid of this two-step approach, the system can be decomposed into any number of subsystems. In order to ensure an evenly balanced workload, the decomposition could be updated dynamically using information on temporarily inactive subsystems. Another advantage of such a splitting is that if during the simulation active regions occur which are independent of each other, these subsystems can be integrated fully in parallel. That is, the results of the individual subsystems need not be distributed to the other processors.

Multirate integration

The methods that we presented in Chapter 4 split the system into an active part and a part that is completely inactive. This characteristic property of the considered systems can be regarded as a special type of multirate behavior. The dependency graph, however, could also be used to separate different *frequencies* of the system. It would then be possible to integrate each subsystem with a step size that is adjusted to the individual speed of the inherent dynamics. For this purpose, the dependency graph could be combined with the multirate integration schemes described in Section 4.2. As mentioned in [KR99], the exploitation of information on the neighborhood of active components in the partitioning strategy minimizes the rate of rejected time steps and increases the reliability of multirate integration schemes.

Isomorphism matching

Often several vertices of the dependency graph represent the same subcircuit or subsystem. That is, they share the same input-output behavior. If the values of all inputs of equivalent subsystems are identical, then the resulting output has to be computed only once. For all remaining configurations, the results can be reused. Furthermore, frequently occurring results could be stored in a database so that, for example, also phase-related or delayed signals need not be recomputed. These isomorphism matching techniques could be used to further reduce the number of function evaluations.

Differential-algebraic equations

For differential-algebraic equations, there exists no one-to-one correspondence between the derivatives and the individual functions or equations. The structure of the system is given only implicitly. In order to enable the simulation of differential-algebraic equations with signal-flow based integration schemes, it would be necessary to generalize the dependency graph in such a way that the interconnection structure of these equations can be described appropriately.

Spatial latency

To utilize not only temporal latency, i.e. inactivity over a period of time, but also spatial latency, i.e. inactivity during the Newton–Raphson iterations, similar signal-flow based techniques might be applicable as well. This could, for instance, be used to speed up the DC analysis, exploiting the fact that some parts of the circuit possibly converge quickly to a solution while other parts converge only very slowly.

A

Differential-algebraic equations

If the states of a physical system, whose dynamic behavior is described via differential equations, are constrained by additional algebraic equations, for instance mass and energy conservation laws or Kirchhoff's laws, then the mathematical modeling of this system usually leads to a so-called *differential-algebraic equation* [BCP89, AP98, RR02, KM06]. The most general form of a differential-algebraic equation is given by

$$F\big(t, x(t), \dot{x}(t)\big) = 0, \tag{A.1}$$

with $F : \mathbb{I} \times \mathbb{D}_x \times \mathbb{D}_{\dot{x}} \mapsto \mathbb{R}^n$, where $\mathbb{I} \subseteq \mathbb{R}$ is an interval and $\mathbb{D}_x, \mathbb{D}_{\dot{x}} \subseteq \mathbb{R}^n$ are open subsets. When we additionally require the solution to fulfill

$$x(t_0) = x_0, \tag{A.2}$$

this leads to an initial value problem.

If the Jacobian $\frac{\partial F}{\partial \dot{x}}$ is nonsingular, then system (A.1) is an implicit ordinary differential equation. If, contrariwise, $\frac{\partial F}{\partial \dot{x}} = 0$, then the equation reduces to a system of nonlinear equations. Otherwise it forms a mixed system of differential and algebraic equations [Bäc07]. Note that the meaning of $\dot{x}$ is ambiguous, it denotes both the differentiation of x with respect to the time t and an independent variable of the function F [KM06].

Linear differential-algebraic equations with constant coefficients

Some of the specific characteristics of differential-algebraic equations can be directly illustrated with the aid of the linear constant coefficient system

$$A\dot{x} + Bx = f, \tag{A.3}$$

with $A, B \in \mathbb{R}^{n\times n}$. If the matrix A is regular, then the system can be easily transformed into an explicit ordinary differential equation. If, on the other hand, A is singular, consider the so-called matrix pencil $\lambda A + B$, where λ is a complex parameter. The pencil is defined to be regular if the characteristic polynomial $p(\lambda) = \det(\lambda A + B)$ is not identical to zero. Provided that the pencil is regular, the system is solvable and there exist nonsingular matrices P and Q such that

$$PAQ = \begin{bmatrix} I & 0 \\ 0 & N \end{bmatrix} \quad \text{and} \quad PBQ = \begin{bmatrix} J & 0 \\ 0 & I \end{bmatrix}, \tag{A.4}$$

where J and N are in Jordan canonical form [BCP89]. Moreover, N is a nilpotent matrix. Let k be the nilpotency index of N, i.e. $N^k = 0$ but $N^{k-1} \neq 0$. The transformation $x = Qy$ gives rise to a system of the form

$$\begin{aligned} \dot{y}_1 + Jy_1 &= \tilde{f}_1, \\ N\dot{y}_2 + y_2 &= \tilde{f}_2. \end{aligned} \tag{A.5}$$

The first equation forms an ordinary differential equation and can be solved for any initial condition. Using the Neumann series and the fact that N is nilpotent, the second equation can be written as

$$y_2 = (ND + I)^{-1}\tilde{f}_2 = \sum_{i=0}^{k-1} (-N)^i \tilde{f}_2^{(i)}, \tag{A.6}$$

where $D = \frac{d}{dt}$ is the differential operator. That is, initial values for the second equation are completely prescribed by the unique solution, and the initial condition is only consistent if

$$y_{2,0} = \sum_{i=0}^{k-1} (-N)^i \tilde{f}_2^{(i)}(t_0). \tag{A.7}$$

If $k = 1$, then $y_2(t) = \tilde{f}_2(t)$ and one differentiation suffices to obtain an ordinary differential equation. If $k \geq 2$, the solution depends not only on the input function but also on its derivatives. Furthermore, the constraints of the system are hidden in the higher-index case [GFtM05].

Nonlinear differential-algebraic equations

The analysis of the linear constant coefficient system demonstrates that there are fundamental differences between ordinary differential equations and differential-algebraic equations. Several different index concepts have been developed to classify differential-algebraic equations according to the difficulty of solving the system analytically or numerically. A frequently used index is the differentiation index which, roughly speaking, describes the number of differentiations needed to obtain an ordinary differential equation.

Definition A.1 (Differentiation index) The *differentiation index* of the differential-algebraic equation (A.1) is defined to be the smallest integer ν such that the system of equations

$$\mathcal{F}_\nu \left(t, x, \dot{x}, \ldots, x^{(\nu+1)}\right) := \begin{bmatrix} F(t, x, \dot{x}) \\ \frac{d}{dt} F(t, x, \dot{x}) \\ \vdots \\ \frac{d^\nu}{dt^\nu} F(t, x, \dot{x}) \end{bmatrix} = 0 \tag{A.8}$$

uniquely determines the variable $\dot{x}$ as a continuous function of t and x.

For linear constant coefficient systems of the form (A.3), the differentiation index ν is given by the nilpotency index k of the matrix N. The differentiation index of ordinary differential equations is, according to the definition, zero. Differential-algebraic equations with index zero or one are in general much simpler to understand and solve than higher-index systems [BCP89]. Other index concepts such as the strangeness index, which can be viewed as a generalization of the differentiation index, can be found in [KM06], for example. However, for the most differential-algebraic equations these different concepts yield the same index or an index which differs at most by one [GFtM05].

As shown above, the solution of differential-algebraic equations requires the determination of consistent initial values which fulfill the algebraic constraints and – in the higher index case – also the hidden constraints [Est00].

Definition A.2 (Consistent initial values) An initial condition $x(t_0) = x_0$ is defined to be *consistent* if there exists a solution of (A.1) that passes through x_0.

That is, prior to the the numerical integration of the differential-algebraic equation, consistent initial values have to be computed. Provided that specified information on the initial state of the system, given by algebraic equations of the form $B(t_0, x_0, \dot{x}_0) = 0$,

is sufficient to determine a unique solution, this can be accomplished as described in [BCP89, LPG91]. The proposed method requires the solution of the system of equations

$$\begin{aligned} \mathcal{F}_\nu \left(t_0, x_0, \dot{x}_0, \ldots, x_0^{(\nu+1)} \right) &= 0, \\ B(t_0, x_0, \dot{x}_0) &= 0. \end{aligned} \tag{A.9}$$

The first two components x_0 and $\dot{x}_0$ of the solution $\left(x_0, \dot{x}_0, \ldots, x_0^{(\nu+1)} \right)$ are uniquely determined. In general it is not possible or feasible to compute the derivatives of the system analytically. However, the derivatives can be replaced by one-sided finite difference approximations. The resulting rank-deficient over-determined approximate system may not have an exact solution. Therefore, the equations are solved in a least-squares sense.

After the determination of consistent initial values, the differential-algebraic equation can be solved for example with one-step or multi-step methods. The formulation of Runge–Kutta methods given in Chapter 4 can be directly generalized to differential-algebraic equations of the form (A.1) by

$$x^{m+1} = x^m + h \sum_{q=1}^{s} b_q K_q, \tag{A.10}$$

where

$$F \left(t^m + c_q h, x^m + h \sum_{r=1}^{s} a_{qr} K_r, K_q \right) = 0. \tag{A.11}$$

In the same way, multi-step methods can be applied to differential-algebraic equations. For an ordinary differential equation $\dot{x}(t) = f(t, x(t))$, a general linear multi-step method with the coefficients α_i and β_i is of the form

$$\sum_{i=0}^{s} \alpha_i \, x^{m-i} = h \sum_{i=0}^{s} \beta_i \, f(t^{m-i}, x^{m-i}). \tag{A.12}$$

An important class of multi-step methods are BDF schemes, which can be written as

$$\sum_{i=0}^{s} \alpha_i \, x^{m-i} = h f(t^m, x^m). \tag{A.13}$$

That is, $\beta_0 = 1$ and $\beta_i = 0$ for $i = 1, \ldots s$. The remaining coefficients α_i, $i = 0, \ldots, s$, are chosen such that the scheme is consistent of order s. BDF methods are only stable for $s \leq 6$ [KM06]. The coefficients of these methods are shown in Table A.1.

Table A.1: Coefficients of the BDF methods.

s	α_0	α_1	α_2	α_3	α_4	α_5	α_6
1	1	-1					
2	$\frac{3}{2}$	-2	$\frac{1}{2}$				
3	$\frac{11}{6}$	-3	$\frac{3}{2}$	$-\frac{1}{3}$			
4	$\frac{25}{12}$	-4	3	$-\frac{4}{3}$	$\frac{1}{4}$		
5	$\frac{137}{60}$	-5	5	$-\frac{10}{3}$	$\frac{5}{4}$	$-\frac{1}{5}$	
6	$\frac{147}{60}$	-6	$\frac{15}{2}$	$-\frac{20}{3}$	$\frac{15}{4}$	$-\frac{6}{5}$	$\frac{1}{6}$

Since the BDF methods satisfy

$$\dot{x}(t^m) = \frac{1}{h} \sum_{i=0}^{s} \alpha_i \, x^{m-i} + \mathcal{O}(h^s), \tag{A.14}$$

these discretization schemes can be applied to nonlinear differential-algebraic equations of the form (A.1) via

$$F\left(t^m, x^m, \frac{1}{h} \sum_{i=0}^{s} \alpha_i \, x^{m-i}\right) = 0. \tag{A.15}$$

Due to the advantageous stability properties and the fact that only one function evaluation per step is needed, these integration schemes are most frequently used to solve stiff ordinary differential equations and differential-algebraic equations [Voi06].

The generalization of arbitrary multi-step methods of the form (A.12) is not immediately possible since these methods require several evaluations of a formula which determines $\dot{x}$ in terms of t and x. Such a formula is not directly available. To enable the use of methods of the form (A.12), the differential-algebraic equation has to be rewritten in a semi-explicit form in order to distinguish differential equations from algebraic constraints [BCP89].

There exist several commercial and noncommercial software packages for the numerical solution of differential-algebraic equations such as, for instance, GENDA[1] and DASKR[2], which provide different Runge–Kutta or BDF methods. An overview and comparison of available packages can be found in [KM06].

[1] www.math.tu-berlin.de/numerik/mt/NumMat/Software/GENDA/
[2] www.netlib.org/ode

Bibliography

[Amd67] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the AFIPS conference*, volume 30, pages 483–485, 1967.

[AP98] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, 1998.

[Bäc07] S. Bächle. *Numerical Solution of Differential-Algebraic Systems Arising in Circuit Simulation*. PhD thesis, Technische Universität Berlin, 2007.

[Bak08] R. J. Baker. *CMOS: Circuit Design, Layout, and Simulation (Revised Second Edition)*. Wiley Interscience & IEEE Press, 2008.

[Bal03] M. Balch. *Complete Digital Design: A Comprehensive Guide to Digital Electronics and Computer System Architecture*. McGraw-Hill, 2003.

[BC08] R. A. Brualdi and D. Cvetković. *A Combinatorial Approach to Matrix Theory and Its Applications*. CRC Press, 2008.

[BCP89] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. North-Holland, 1989.

[BGK01] A. Bartel, M. Günther, and A. Kværnø. Multirate methods in electrical circuit simulation. Preprint Numerics 2, Department of Mathematical Sciences, Norwegian University of Science and Technology, 2001.

[Bry81] R. E. Bryant. *A Switch-Level Simulation Model for Integrated Logic Circuits*. PhD thesis, Massachusetts Institute of Technology, 1981.

[Bry84] R. E. Bryant. A switch-level model and simulator for MOS digital systems. *IEEE Transactions on Computers*, C-33(2):160–177, 1984.

[Bry87] R. E. Bryant. A survey of switch level algorithms. *IEEE Design and Test of Computers*, 4(4):26–40, 1987.

[But87] J. C. Butcher. *The numerical analysis of ordinary differential equations: Runge–Kutta and general linear methods.* John Wiley & Sons, 1987.

[Cad04] Cadence Design Systems. Using hierarchy and isomorphism to accelerate circuit simulation, 2004.

[CFJS04] T. Carrisosa, T. Félix, M. Jerónimo, and J. Soares Augusto. SUSANA: a MOS-Mixed-Circuit Simulator Using Logic/ELogic Algorithms. In *DCIS'04: 19th Conference on Design of Circuits and Integrated Systems*, 2004.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (Second Edition).* MIT Press, 2001.

[CO93] G. Chartrand and O. Oellermann. *Applied and Algorithmic Graph Theory.* McGraw-Hill, 1993.

[CTCK04] T. Chen, J. Tsai, C. Chen, and T. Karnik. HiSIM: Hierarchical interconnect-centric circuit simulator. In *IEEE/ACM International Conference on Computer-Aided Design*, 2004.

[DA93] A. Dolan and J. Aldous. *Networks and Algorithms: An Introductory Approach.* John Wiley & Sons, 1993.

[Dav91] A. T. Davis. *Implicit Mixed-Mode Simulation of VLSI Circuits.* PhD thesis, University of Rochester, New York, 1991.

[DOR87] P. Debefve, F. Odeh, and A. E. Ruehli. Waveform techniques. In A. E. Ruehli, editor, *Circuit Analysis, Simulation and Design, Volume 3, Part 2*, pages 41–127. North-Holland, 1987.

[Est00] D. Estevez Schwarz. *Consistent initialization for index-2 differential algebraic equations and its application to circuit simulation.* PhD thesis, Humboldt-Universität zu Berlin, 2000.

[Fie86] M. Fiedler. *Special matrices and their applications in numerical mathematics*. Martinus Nijhoff Publishers, 1986.

[Frö02] N. Fröhlich. *Verfahren zum Schaltungspartitionieren für die parallele Simulation auf Transistorebene*. PhD thesis, Technische Universität München, 2002.

[FRWZ98] N. Fröhlich, B. M. Riess, U. A. Wever, and Q. Zheng. A new approach for parallel simulation of VLSI circuits on a transistor level. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 45(6):601–613, 1998.

[FSF97] N. Fröhlich, R. Schlagenhaft, and J. Fleischmann. A new approach for partitioning VLSI circuits on transistor level. In *Proceedings of the 11th IEEE/ACM/SCS Workshop on Parallel and Distributed Simulation*, 1997.

[GF95] M. Günther and U. Feldmann. The DAE-index in electric circuit simulation. *Mathematics and Computers in Simulation*, 39(5–6):573–582, 1995.

[GFtM05] M. Günther, U. Feldmann, and J. ter Maten. Modelling and discretization of circuit problems. Technical report, OAI Repository of the Technische Universiteit Eindhoven, 2005.

[GG05] L. B. Goldgeisser and M. M. Green. A method for automatically finding multiple operating points in nonlinear circuits. Postprints, paper 867, University of California, Irvine, 2005.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[GJS76] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[GKR01] M. Günther, A. Kværnø, and P. Rentrop. Multirate partitioned Runge–Kutta methods. *BIT Numerical Mathematics*, 41(3):504–514, 2001.

[GPS04] M. Golubitsky, M. Pivato, and I. Stewart. Interior symmetry and local bifurcation in coupled cell networks. *Dynamical Systems*, 19:389–407, 2004.

[GR94] M. Günther and P. Rentrop. Partitioning and multirate strategies in latent electric circuits. In R. E. Bank, R. Burlisch, H. Gajewski, and K. Merten, editors, *Mathematical Modelling and Simulation of Electrical Circuits and Semiconductor Devices*, volume 117. Birkhäuser, 1994.

[GR01] C. Godsil and G. Royle. *Algebraic Graph Theory*. Springer, 2001.

[GS03] M. Golubitsky and I. Stewart. *The Symmetry Perspective: From Equilibrium to Chaos in Phase Space and Physical Space*. Birkhäuser, 2003.

[GW84] C. W. Gear and D. R. Wells. Multirate linear multistep methods. *BIT Numerical Mathematics*, 24(4):484–502, 1984.

[Hay82] J. P. Hayes. A unified switching theory with applications to VLSI design. *Proceedings of the IEEE*, 70(10):1140–1151, 1982.

[Hay84] J. P. Hayes. Fault modeling for digital MOS integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 3(3):200–208, 1984.

[HNW93] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems (Second Revised Edition)*. Springer, 1993.

[Hof76] E. Hofer. A partially implicit method for large stiff systems of ODEs with only few equations introducing small time-constants. *SIAM Journal on Numerical Analysis*, 13(5):645–663, 1976.

[Hof04] K. Hoffmann. *System Integration: From Transistor Design to Large Scale Integrated Circuits*. John Wiley & Sons, 2004.

[Hon02] M. Honkala. *Parallel Hierarchical DC Analysis*. Licentiate thesis, Helsinki University of Technology, 2002.

[HRV01] M. Honkala, J. Roos, and M. Valtonen. New multilevel Newton–Raphson method for parallel circuit simulation. In *ECCTD '01: Proceedings of the 15th European Conference on Circuit Theory and Design*, pages 113–116, 2001.

[HS09] W. Hundsdorfer and V. Savcenco. Analysis of a multirate theta-method for stiff ODEs. *Applied Numerical Mathematics*, 59(3–4):693–706, 2009.

[HW96] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems (Second Revised Edition)*. Springer, 1996.

[Kao92] R. Kao. Piecewise linear models for switch-level simulation. Technical Report CSL-TR-92-532, Departments of Electrical Engineering and Computer Science, Stanford University, 1992.

[KK99] T. Kato and T. Kataoka. Circuit analysis by a new multirate method. *Electrical Engineering in Japan*, 126(4):55–62, 1999.

[KM99] C. E. Kees and C. T. Miller. C++ implementations of numerical methods for solving differential-algebraic equations: design and optimization considerations. *ACM Transactions on Mathematical Software*, 25(4):377–403, 1999.

[KM06] P. Kunkel and V. Mehrmann. *Differential-Algebraic Equations*. EMS Textbooks in Mathematics. European Mathematical Society, 2006.

[KR99] A. Kværnø and P. Rentrop. Low order multirate Runge-Kutta methods in electric circuit simulation. IWRMM Preprint 1, University of Karlsruhe, 1999.

[Kun95] K. S. Kundert. *The Designer's Guide to Spice and Spectre*. Kluwer Academic Publishers, 1995.

[Lew89] K.-D. Lewke. *Ein Modell zur ereignisgetriebenen Simulation von MOS-Transistornetzwerken auf der Schalterebene*, volume 120 of *Fortschrittsberichte VDI: Informatik/Kommunikationstechnik*. VDI-Verlag, 1989.

[LPG91] B. Leimkuhler, L. R. Petzold, and C. W. Gear. Approximation methods for the consistent initialization of differential-algebraic equations. *SIAM Journal on Numerical Analysis*, 28(1):205–226, 1991.

[LRS82] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli. The waveform relaxation method for time-domain analysis of large scale integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1(3):131–145, 1982.

[LSV96] W. Lioen, J. de Swart, and W. van der Veen. Test set for IVP solvers. Technical Report NM-R9615, CWI, Department of Numerical Mathematics, Amsterdam, 1996.

[MEF+03] R. März, D. Estevez Schwarz, U. Feldmann, S. Sturtzel, and C. Tischendorf. Finding beneficial DAE structures in circuit simulation. In W. Jäger and H.-J. Krebs, editors, *Mathematics – Key Technology for the Future*, pages 413–428. Springer, 2003.

[Mez04] I. Mezić. Coupled nonlinear dynamical systems: Asymptotic behavior and uncertainty propagation. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, pages 1778–1783, 2004.

[Mic03] A. Miczo. *Digital Logic Testing and Simulation (Second Edition)*. John Wiley & Sons, 2003.

[MUR95] N. Mohan, T. M. Undeland, and W. P. Robbins. *Power Electronics: Converters, Applications, and Design (Second Edition)*. John Wiley & Sons, 1995.

[OFM07] R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(3):215–233, 2007.

[Pre00] R. Preis. *Analyses and Design of Efficient Graph Partitioning Methods*. PhD thesis, Universität Paderborn, 2000.

[QNPS93] T. Quarles, A. R. Newton, D. O. Pederson, and A. L. Sangiovanni-Vincentelli. *SPICE 3 User's Manual*, 1993.

[Rei88] K. J. Reinschke. *Multivariable Control: A Graph-theoretic Approach*. Springer, 1988.

[Ren85] P. Rentrop. Partitioned Runge–Kutta methods with stiffness detection and stepsize control. *Numerische Mathematik*, 47:545–564, 1985.

[ROTH89] V. B. Rao, D. V. Overhauser, T. N. Trick, and I. N. Hajj. *Switch-Level Timing Simulation of MOS VLSI Circuits*. Kluwer Academic Publishers, 1989.

[RR02] P. J. Rabier and W. C. Rheinboldt. Theoretical and numerical analysis of differential-algebraic equations. In P. G. Ciarlet und J. L. Lions, editor, *Handbook of numerical analysis, Vol. VIII: Techniques of scientific computing (Part 4)*, pages 183–540. Elsevier Science BV, 2002.

[Sal84] R. A. Saleh. Iterated timing analysis and SPLICE1. Technical Report UCB/ERL M84/2, EECS Department, University of California, Berkeley, 1984.

[SBG09] M. Striebel, A. Bartel, and M. Günther. A multirate ROW-scheme for index-1 network equations. *Applied Numerical Mathematics*, 59(3–4):800–814, 2009.

[SG04] M. Striebel and M. Günther. Towards one-step multirate methods in chip design. Preprint BUW-AMNA 04/09, Bergische Universität Wuppertal, 2004.

[SG05] M. Striebel and M. Günther. A charge oriented mixed multirate method for a special class of index-1 network equations in chip design. *Applied Numerical Mathematics*, 53:489–507, 2005.

[SH68] H. Shichman and D. A. Hodges. Modeling and simulation of insulated-gate field-effect transistor switching circuits. *IEEE Journal of Solid-State Circuits*, 3(3):285–289, 1968.

[SHV07] V. Savcenco, W. Hundsdorfer, and J. G. Verwer. A multirate time stepping strategy for stiff ordinary differential equations. *BIT Numerical Mathematics*, 47:137–155, 2007.

[Šil91] D. D. Šiljak. *Decentralized Control of Complex Systems*. Academic Press, 1991.

[Ske89] S. Skelboe. Stability properties of backward differentiation multirate formulas. *Applied Numerical Mathematics*, 5(1–2):151–160, 1989.

[SN90] R. A. Saleh and A. R. Newton. *Mixed-Mode Simulation*. Kluwer Academic Publishers, 1990.

[TFC+03] A. Tcherniaev, I. Feinberg, W. Chan, J.-F. Tuan, and A.-C. Deng. Transistor level circuit simulator using hierarchical data. United States Patent, 2003.

[VKLM04] S. Varigonda, T. Kalmár-Nagy, B. LaBarre, and I. Mezić. Graph decomposition methods for uncertainty propagation in complex, nonlinear interconnected dynamical systems. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, pages 1794–1798, 2004.

[Voi06] S. Voigtmann. *General Linear Methods for Integrated Circuit Design*. PhD thesis, Humboldt-Universität zu Berlin, 2006.

[VTB+08] A. Verhoeven, B. Tasic, T. G. J. Beelen, E. J. W. ter Maten, and R. M. M. Mattheij. BDF compound-fast multirate transient analysis with adaptive step-size control. *Journal of Numerical Analysis, Industrial and Applied Mathematics*, 3(3–4):275–297, 2008.

[Wan02] S. Wang. Delivering a full-chip hierarchical circuit simulation. Technical report, Nassda Corp., 2002.

[WOSR85] J. White, F. Odeh, A. L. Sangiovanni-Vincentelli, and A. E. Ruehli. Waveform relaxation: Theory and practice. Technical Report UCB/ERL M85/65, EECS Department, University of California, Berkeley, 1985.

[WZ96] U. Wever and Q. Zheng. Parallel transient analysis for circuit simulation. In *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*, pages 442–446, 1996.

[YG99] E. Yilmaz and M. M. Green. Applying globally convergent techniques to conventional DC operating point analyses. In *Proceedings of the 32nd Annual Simulation Symposium*, pages 153–158, 1999.