Lecture Notes in Electrical Engineering 233

Mourad Fakhfakh
Esteban Tlelo-Cuautle
Rafael Castro-López
Editors

Analog/RF and Mixed-Signal Circuit Systematic Design

Springer

Mourad Fakhfakh, Esteban Tlelo-Cuautle, and
Rafael Castro-López (Eds.)

Analog/RF and Mixed-Signal Circuit Systematic Design

Springer

Editors
Mourad Fakhfakh
University of Sfax
Sfax
Tunisia

Esteban Tlelo-Cuautle
INAOE
Department of Electronics
Puebla
Mexico

Rafael Castro-López
Instituto de Microelectrónica de Sevilla
IMSE-CNM, CSIC. c/ Américo
Vespucio, s/n
Seville
Spain

ISSN 1876-1100 e-ISSN 1876-1119
ISBN 978-3-642-36328-3 e-ISBN 978-3-642-36329-0
DOI 10.1007/978-3-642-36329-0
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2012956292

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

This book introduces contributions to the topic of systematic design of analog, RF and mixed signal circuits. In my view, the material represents a significant step towards advancing the state-of-the-art in the field of robust analog and mixed signal design automation. Although The topic has been researched for many years primarily for pure analog circuits, it is quite evident that extending the research effort to include mixed signal and RFIC design is very timely and relevant in light of the ever increasing complexity of complete Systems-on-Chip (SoCs) which include analog, RF and mixed signal on the same die with Ultra large scale digital. Such SoCs are also widely recognized as the "More-than-Moore" scaling extending the end of the silicon roadmap for many years to come. The field of systematic circuit design has long been a mature science area for many years for digital circuits but has represented a formidable challenge to analog circuits. However, the analysis and synthesis techniques and flows presented in this book provide practical solutions to meet the challenge. Not only does the material address some of the traditional bottlenecks of analog and RF design automation such as device sizing and layout generation, but also incorporates tools and methodologies to deal with worst case corners and random process variations. This will indeed result in automated and robust design solutions that lend themselves naturally to implementations in deep nanometer process nodes and with enhanced yields. Automation of these More-than-Moore SoCs will also help with meeting narrow market windows and with reducing development costs of complex nano-scale chip sets.

The material is organized in two parts and presented in 16 chapters. It strikes a good balance between theory and practice and includes case studies or design examples to reinforce understanding of basic concepts. The book is highly recommended for mixed signal, RF and SoC design engineers and practitioners in the semiconductor industry as well as researchers and graduate students in electrical and computer engineering with a major in circuit design and design automation.

November 2012 Mohammed Ismail

Mohammed Ismail
Ohio State University, Columbus, USA
Currently with Khalifa University of Science, Technology and Reserach (KUSTAR), UAE

Preface

Advances in electronics technologies have led to a kind of a ‘boom’ in a very wide range of fields, such as, informatics, bioengineering, communications, electronic gadgets, to name a few.

Despites the fact that in the digital domain, designers can take full benefits of IPs and design automation tools to synthesize and design very complex systems, the analog designers’ task is still considered as a ‘handcraft’, cumbersome and very time consuming process. This is mainly due to the lack of support by computer-aided design programs, which has led to a so-called ‘productivity gap’ (difference between what technology can offer and what can be manufactured). Thus, tremendous efforts are being deployed by researchers, R/D engineers, etc. to develop new design methodologies in the analog/RF and mixed-signal domains.

Actually, the analog/RF and mixed signal fields rely on three major areas, namely Synthesis, Design and Optimization. These domains form a trilogy in this realm of analog/RF and mixed-signal circuit and system design. Endeavors are being made to develop new synthesis techniques (building novel active circuits, for instance), design methodologies (proposing new circuits) and sizing/optimization techniques (offering more complex functionalities with advanced performances, higher frequency operating ranges, less power consumption, etc.).

On this basis, this book collects in sixteen Chapters, recent theories, synthesis techniques and design methodologies, as well as new sizing approaches. It highlights their application to the design of high performance analog/RF and mixed-signal circuits and systems. This book is intended to researchers and R/D engineers, as well. The book encompasses two parts: *Methodologies* and *Techniques*.

The first part, *Methodologies*, is composed of seven Chapters, very briefly introduced in the following:

Chapter 1, entitled ‘Towards Automatic Structural Analysis of Mixed-Signal Circuits’, is proposed by *M. Eick and H. Graeb*. It presents a new method for the automatic structural and functional analysis of analog, digital and mixed-signal circuits.

Chapter 2, ‘Efficient Synthesis Methods for mm-wave Frequency Passive Components and Amplifiers’, authored by *B. Liu and G. Gielen*, deals with an efficient high-frequency synthesis methods for integrated passive components as well as for the synthesis of mm-wave-frequency linear amplifiers, using the memetic machine learning-based differential evolution method and the efficient machine learning-based differential evolution method, respectively.

Chapter 3, entitled 'Self-Healing Circuits Using Statistical Element Selection' and proposed by *V. H.-C. Chen, G. Keskin, and L. T. Pileggi*, analyzes the statistical element selection methodology for the implementation of low-power self-healing circuits and systems.

Chapter 4, 'Improving Design Feature Reuse in Analog Circuit Design through Topological-Symbolic Comparison and Entropy-based Classification', authored by *C. Ferent and A. Doboli* introduces a novel circuit synthesis methodology based on concept comparison, combination, learning, and re-use.

Chapter 5 that is entitled 'Graph-based symbolic and symbolic sensitivity analysis of analog integrated circuits' and proposed by *S. Rodriguez-Chavez, A.A. Palma-Rodriguez, E. Tlelo-Cuautle, and S.X.-D. Tan*, describes a graph-based technique for the solution of a system of equations for analog ICs formulated by applying symbolic NA and for symbolic sensitivity analysis.

Chapter 6 titled 'A Designer Centric Analog Synthesis Flow', which is authored by *F. Javid, S. Youssef, R. Iskander, and M.-M. Louërat*, presents a designer centric analog synthesis flow that is fully controlled by the designer and offers an intuitive design approach that is composed of a sizing tool and a layout generation tool.

Chapter 7; 'Analog Circuit Design based on Robust POFs using an Enhanced MOEA with SVM Models' by *N. Lourenço, R. Martins, M. Barros, and N. Horta* highlights a multi-objective design methodology for automatic analog integrated circuits synthesis, which enhances the robustness of the solution by varying technological and environmental parameters, and by the inclusion of corner cases.

The second part of the book, *Techniques*, encompasses the nine following Chapters:

Chapter 8; 'Applications of symbolic analysis in the design of analog circuits' by *F. Grasso, A. Luchetta, and M. C. Piccirilli*, describes the use of symbolic techniques in the realization of efficient automatic tools for designing analog circuits. In particular three phases of the design cycle of an integrated circuit are considered: the simulation phase, the design centering phase and the fault diagnosis phase.

Chapter 9, titled 'Synthesis of Electronically-Controllable Signal Processing/Signal Generation Circuits using Modern Active Building Blocks', is authored by *R. Senani, D. R. Bhaskar, A. K. Singh, and V. K. Singh* focuses on the synthesis of various electronically-controllable signal processing/signal generation circuits. The coverage includes the basics and hardware implementation of various building blocks mentioned above and includes some elegant representative applications using them.

Chapter 10, entitled 'Synthesis of Generalized Impedance Converter and Inverter Circuits Using NAM Expansion' by *A. M Soliman* proposes the use of the nodal admittance matrix expansion technique to generate all possible voltage generalized impedance converter and the current generalized impedance converter circuits, and the realizations of two types of the generalized impedance inverter circuits.

Chapter 11; 'Fractional Step Analog Filter Design', by *T. Freeborn, B. Maundy, and A. Elwakil* outlines the process to design, analyze, and implement continuous-time fractional-step filters, and presents new methods and design equations for the physical realization of these filters using fractional capacitors, SABs, FPAA hardware, and FDNR topologies.

Chapter 12, entitled 'The Flipped Voltage Follower: Theory and applications' and that is authored by *J. Ramirez-Angulo, M. R. Valero-Bernal, A. Lopez-Martin, R. G. Carvajal, A. Torralba, S. Celma-Pueyo, and N. Medrano-Marqués,* exposes and summarizes in a tutorial way, the most relevant information published to date on the FVF, and presents several improved FVF cells and structures and gives a comparison of their performances and characteristics.

Chapter 13, titled 'Synthesis of Analog Circuits using only Voltage and Current Followers as Active Elements', by *R. Senani, D.R. Bhaskar, A.K. Singh, and R.K. Sharma*, presents a brief account of some prominent works done on the analog circuit design using VFs and CFs as active elements, together with the design of VFs and CFs themselves.

Chapter 14; 'Design of Setable Active Lossy Inductors', proposed by *M. Pierzchala, and M. Fakhfakh* is concerned with transformation of passive LC filters into active RC-circuits using signal-flow graphs in the two-graph by using exclusively RC-elements and the newly introduced 'active switches'. The Chapter also deals with the reduction of the complexity of the constructed active circuits.

Chapter 15, entitled 'MIDAS: Microwave Inductor Design Automation on Silicon' by *L. Aluigi, F. Alimenti, L. Roselli, D. Pepe, and D. Zito* emphasizes a methodology to automate the design of microwave inductor on silicon and presents the implementation of an auxiliary CAD tool for Microwave Inductor Design Automation on Silicon.

Chapter 16; 'LC-VCO Design Challenges in the Nano-Era' authored by *P. Pereira, H. Fino, M. Fakhfakh, F. Coito, and M. Ventim-Neves* exposes an optimization based methodology for the design of LC-VCOs whose efficiency is granted by the use of analytical models to characterize the behavior of active and passive elements.

Finally, we want to use this opportunity to thank all the authors for their high quality contributions, and the reviewers for their valuable help. We are also thankful to Prof. Mohamed Ismail (Ohio State University, Columbus, USA. Currently with Khalifa University of Science, Technology and Reserach (KUSTAR), UAE) for writing the foreword of the book. Our thanks go also to the SPRINGER team for his support and assistance.

Mourad Fakhfakh
Esteban Tlelo-Cuautle
Rafael Castro-López

Contents

Part II: Techniques

Part I

Methodologies

Chapter 1
Towards Automatic Structural Analysis of Mixed-Signal Circuits

Michael Eick and Helmut Graeb

Abstract. A new approach for the structural analysis of integrated circuits is presented in this chapter. As a unique feature this approach can handle circuits that contain analog and digital components at the same time. Such a situation occurs, e.g., in mixed–signal circuits. First, the approach analyzes the circuit for basic analog and digital building blocks. Next, a structural signal flow analysis partitions the circuit into an analog and digital part. It is also used to determine true pass–gate directions and break feedback loops. Finally, the logic functions of the building blocks as well as the complete digital circuit part are extracted. The chapter presents application examples for digital standard cell libraries and mixed–signal circuits. For industrial grade standard cell libraries more than 95% of the contained cells are analyzed correctly. The mixed–signal examples include a charge pump as well as voltage–controlled ring oscillator.

1.1 Introduction

Mixed–signal circuits play an important role in most modern integrated circuits. Typical examples are analog–to–digital and digital–to–analog converters, voltage–controlled ring oscillators and charge pumps. Like pure analog circuits, mixed–signal circuits are subject to several constraints, e.g., certain MOSFET transistors must work in saturation region and special layout styles must be applied to some devices to achieve good matching. The availability of such constraints in machine–readable form is an indispensable prerequisite for the automation of design steps such as sizing and layout synthesis. Usually such a machine–readable documentation is not available, which requires algorithms to extract these constraints from the schematic.

Michael Eick · Helmut Graeb
Institute for Electronic Design Automation, Technische Universität München, Munich, Germany
e-mail: {eick,graeb}@tum.de

M. Fakhfakh et al. (Eds.): *Analog/RF & Mixed-Signal Circuit Sys. Design*, LNEE 233, pp. 3–25.
DOI: 10.1007/978-3-642-36329-0_1 © Springer-Verlag Berlin Heidelberg 2013

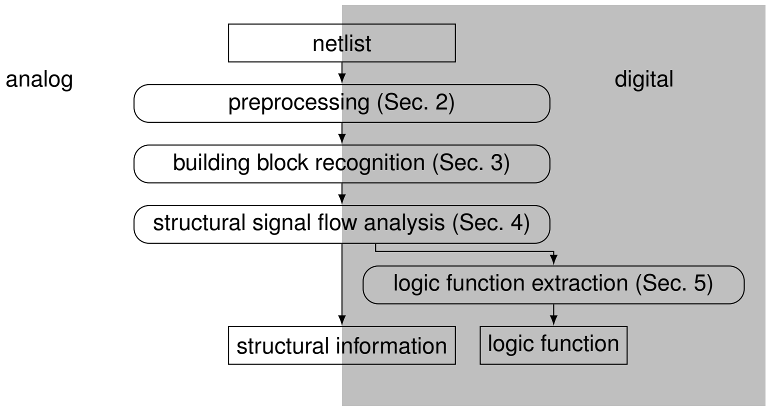

analog
digital
netlist
preprocessing (Sec. 2)
building block recognition (Sec. 3)
structural signal flow analysis (Sec. 4)
logic function extraction (Sec. 5)
structural information
logic function

Fig. 1.1 Overall structural analysis flow

Previous work has shown that such constraints can be generated automatically for analog circuits [6, 7, 14]. The authors of [14] use building block recognition to identify analog blocks such as current mirrors. The available building blocks are defined through a library, which can contain CMOS and bipolar structures. Ambiguities are resolved using a dominance graph. The authors of [6, 7] compute symmetry in analog circuits using the recognized building blocks. Based on detected building blocks and symmetries, constraints for sizing and placement are generated.

These methods cannot be applied to mixed–signal circuits. This is because mixed–signal circuits consist of common analog components, such as current mirrors, common digital components, such as inverters and logic gates, as well as pass–gates and pass–transistors. In addition, continuous time and signal values can be assumed for analog circuits, for mixed-signal circuits time and signal values can be discrete.

Current approaches for the structural analysis of digital circuits can be divided into two classes. The first class assumes a CMOS structure and analyzes the parallel and serial connections of the transistors using special algorithms, e.g., [4, 5, 9, 11, 20]. These approaches can handle nearly all digital CMOS circuits but are limited to this type of circuit, which makes them infeasible for mixed–signal circuits. Some approaches can generate a graph representing the circuit structure, e.g., [11, 20]. The second class compares a netlist to a given library using subgraph isomorphism algorithms, e.g., [13, 16, 21, 22]. They are applicable for a wide range of circuit types but are limited to the provided library. Both approaches can yield a logic function for each identified subcircuit, which in turn allows to compute the overall logic function.

In this book chapter, we will present a new method enhancing the approaches of [7, 14] to handle mixed–signal circuits. The overall analysis flow is shown in Fig. 1.1. First, a netlist is read and some preprocessing is performed. After that, a building block recognition algorithm is executed. Compared to the state of the art, it provides the following new features,

- a versatile building block library for analog, digital and mixed–signal circuits,
- a corresponding dominance relation,
- a new recognition algorithm that can handle this library.

The approach uses a hierarchical library combining the benefits of library based approaches and algorithmic approaches. Next, a structural signal flow analysis is performed. It enhances the analysis presented in [6, 7] for analog circuits to handle digital and mixed–signal circuits. Algorithms to assign pass–gate directions and to break feedback loops are added. Finally, the logic function of the digital circuit part is extracted.

Preprocessing, enhanced building block recognition and structural signal flow analysis are discussed in Sections 1.2, 1.3 and 1.4, respectively. Section 1.5 introduces the logic function extraction algorithm. Application examples are shown in Section 1.6. Section 1.7 concludes the chapter.

1.2 Preprocessing

The netlist can contain parasitic resistors and capacitors which inhibit a correct building block recognition. Therefore parasitic devices are replaced by short–circuits and open–circuits as appropriate.

In addition, the source and drain assignment of MOSFETs in the netlist does not always match the actual assignment during operation. The actual assignment is required for correct building block recognition. It is determined by traversing the netlist from *Vdd*– to *Vss*–nets nets and vice versa.

1.3 Building Block Recognition

In the following, an algorithm is presented that recognizes basic building blocks, e.g., simple current mirrors (in analog circuits) and inverters (in digital circuits). This is done by comparing the circuit netlist to a given library of building blocks. A library for analog, digital and mixed–signal circuits is presented after some formal definitions. Next, a dominance relation is presented, which is used to resolve recognition ambiguities. Finally, the recognition algorithm is discussed.

A circuit consists of several devices such as MOSFETs. The set of all devices is $\mathscr{D}$. Each device $d \in \mathscr{D}$ has several attributes associated with it. We denote these attributes using a pseudo object–oriented notation, e.g., $d.a$ is attribute a of device d. A device d has the following attributes:

type t	The type $d.t \in T_{\mathscr{D}} = \{\text{trans}, \text{res}, \text{cap}, \ldots\}$ describes whether the device is a transistor (trans), a resistor (res), a capacitor (cap), etc.
subtype s	The subtype $d.s \in \{\text{none}, \text{nmos}, \text{pmos}\}$ is used to distinguish between NMOS and PMOS transistors.

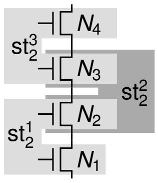


Fig. 1.2 Stack chain consisting of three stacks

pins p	Pins are used to connect the device to nets. The set $d.p$ lists the available pins, e.g., $d.p = \{\mathrm{gt}, \mathrm{dn}, \mathrm{sc}\}$ for a mosfet-transistor with gate (gt), drain (dn) and source (sc).

Definition 1 (building block). A building block $b \in \mathscr{B}$ consists of several devices or other building blocks, where $\mathscr{B}$ is the set of all building blocks. It has the following associated attributes:

children c	A tuple $b.c \in (\mathscr{D} \cup \mathscr{B})^{n_{c,s}}$ listing the included building blocks or devices, where $n_{c,s}$ is the number of children.
type t	A type $b.t \in T_{\mathscr{B}}$ similar to the type defined for devices. The available building block types depend on the used library.
subtype s	A subtype $b.s$ similar to the type defined for devices.
pins p	A set of pins $b.p$ similar to the type defined for devices.

Devices and building blocks connect to the nets $n \in \mathscr{N}$ of the circuit using their pins.

Definition 2 (Connectivity function η)
The connectivity function $\eta(x,p) \in \mathscr{N}, x \in (\mathscr{D} \cup \mathscr{B}), p \in x.p$ describes the connectivity of a circuit. A device or building block x connects to a net n by pin p iff $n = \eta(x,p)$.

1.3.1 Analog, Mixed-Signal and Digital Building Block Library

The recognition algorithm is based on the building block library shown in Fig. 1.3. The unshaded part covers analog building blocks, the gray shaded part covers digital building blocks and the gray striped part covers building blocks used in analog and digital circuits. The figure does not show the complete library of analog building blocks, which can be found in [14].

The library is based on three different generic building blocks.

pair	A pair consists of two building blocks or devices, e.g., a simple current mirror or a stack.
array	An array consists of n building blocks or devices connected in parallel, e.g., a diode transistor array.
chain	A chain consists of pairs y_1, y_2 to y_n, where two pairs $y_i, y_{i+1}, i = 1\ldots(n-1)$ share one child. Figure 1.2 illustrates this for a stack chain consisting of stacks st_2^1 to st_2^3. Stacks st_2^1 and st_2^2 share N_2, stacks st_2^2 and st_2^3 share N_3. A chain can have a single child only.

For a part of the building blocks, all children have the same subtype, i.e., they are either all nmos or all pmos. For these building blocks only the nmos variant is shown in Fig. 1.3. Examples are simple current mirror and stack. Other building blocks consist of children with nmos and pmos type. Examples are logic gate and pass–gate.

The library is organized into different hierarchy levels. Building blocks from one hierarchy level are built out of building blocks from lower hierarchy levels. The lowest hierarchy level, hierarchy level 0, is formed by the transistors from the netlist. The overall number of hierarchy levels n_L depends on the circuit and is automatically determined by the recognition algorithm.

Hierarchy level 1 contains building blocks that group parallel transistors together. For example, a diode transistor array, consists of parallel diode connected transistors.

Hierarchy level 2 contains the analog building blocks simple current mirror (scm), voltage reference II (vrII), differential pair (dp) and level-shifter (ls). Simple current mirror and level–shifter consist of a diode transistor array connected to a normal transistor array. The other building blocks consist of normal transistor arrays only. Stack, pass–gate and cross–coupled pair can be used for analog as well as digital circuits. Logic gate, logic array and stack chain are useful for digital circuits only. The current hierarchy level is used as index for stack, logic gate, logic array and stack chain because they are repeated on higher hierarchy levels. The gate pins of the logic gate can be connected to an inverter or independently controlled.

Hierarchy Level 3 contains a stack chain which is needed for digital circuits only. It is constructed from multiple stack building blocks that overlap at one transistor.

The analog part of *hierarchy level 4* contains the cascode current mirror, which is formed from a simple current mirror and a level–shifter as well as the wide–swing current mirror, which is formed from a voltage reference II and a stack from level 2. For digital circuits the tristate base block is defined. It consists of a pass–gate and a logic array.

For all *even hierarchy levels starting from 4 up to* n_L digital building blocks are defined recursively. A logic array on hierarchy level $k = 4, 6, \ldots$ can be formed by stack chains from lower hierarchy levels as well as normal transistor arrays. At least one of the stack chains must be from hierarchy level $k-1$. The same principle applies to stacks which are formed out of logic arrays and normal transistor arrays. A logic gate combines a logic array, stack chain or normal transistor array with PMOS–subtype and a logic array, stack chain or normal transistor array with NMOS–subtype.

All *odd hierarchy levels starting from 5 up to* $n_L - 1$ contain a stack chain which is formed from stacks from the hierarchy level before.

The analog part of *hierarchy level 6* defines the differential stage, consisting of a current mirror and a differential pair. In addition to the recursively defined building blocks, the digital part of hierarchy level 6 contains the tristate control block, which consists of two tristate base blocks. It is needed to handle one type of tristate buffers correctly.

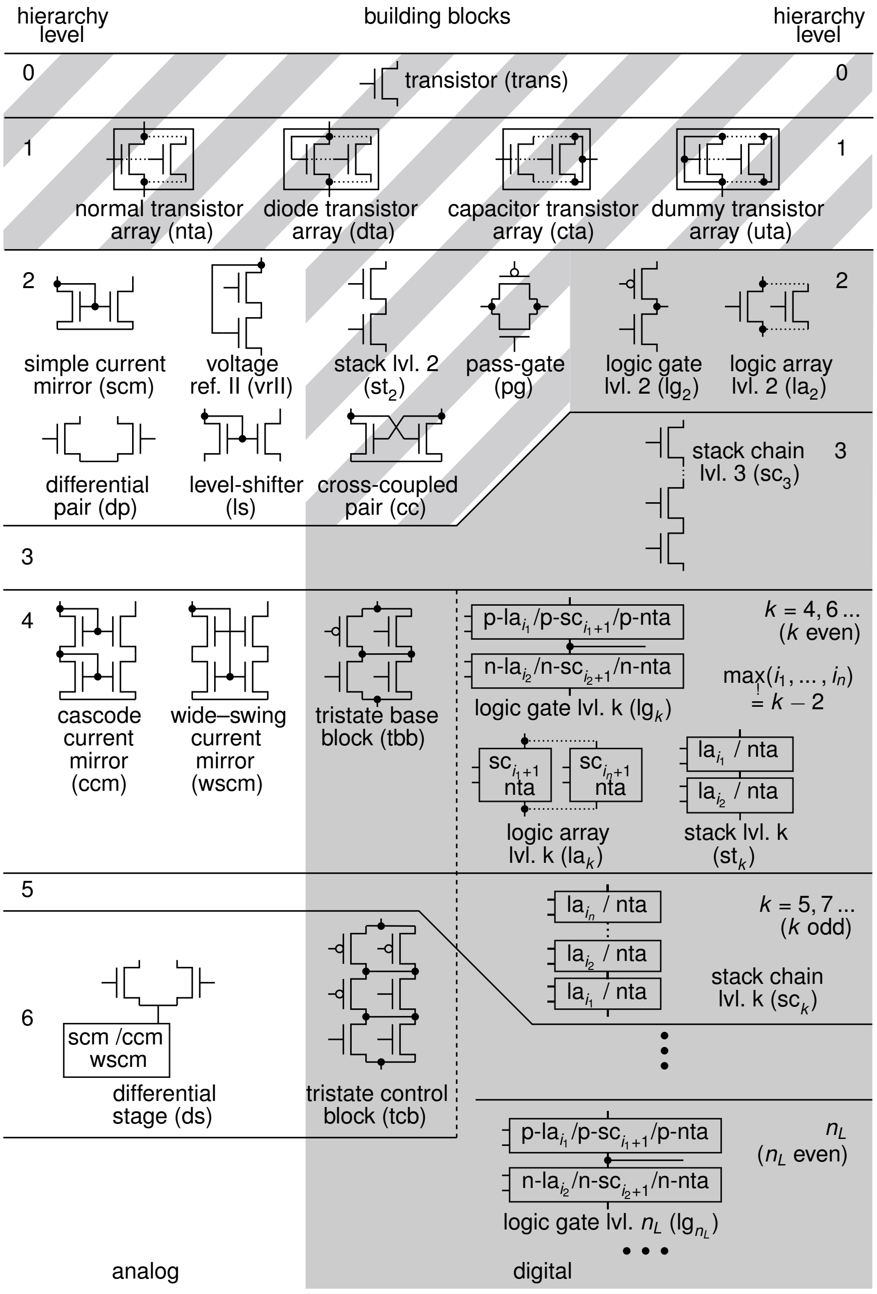

hierarchy level
building blocks
hierarchy level
0
transistor (trans)
0
1
normal transistor array (nta)
diode transistor array (dta)
capacitor transistor array (cta)
dummy transistor array (uta)
1
2
simple current mirror (scm)
voltage ref. II (vrII)
stack lvl. 2 (st$_2$)
pass-gate (pg)
logic gate lvl. 2 (lg$_2$)
logic array lvl. 2 (la$_2$)
2
differential pair (dp)
level-shifter (ls)
cross-coupled pair (cc)
stack chain lvl. 3 (sc$_3$)
3
3
4
cascode current mirror (ccm)
wide–swing current mirror (wscm)
tristate base block (tbb)
p-la$_{i_1}$/p-sc$_{i_1+1}$/p-nta
n-la$_{i_2}$/n-sc$_{i_2+1}$/n-nta
logic gate lvl. k (lg$_k$)
$k = 4, 6 \ldots$ (k even)
$\max(i_1, \ldots, i_n) \overset{!}{=} k - 2$
sc$_{i_1+1}$ nta
sc$_{i_n+1}$ nta
logic array lvl. k (la$_k$)
la$_{i_1}$ / nta
la$_{i_2}$ / nta
stack lvl. k (st$_k$)
5
la$_{i_n}$ / nta
la$_{i_2}$ / nta
la$_{i_1}$ / nta
$k = 5, 7 \ldots$ (k odd)
stack chain lvl. k (sc$_k$)
6
scm /ccm wscm
differential stage (ds)
tristate control block (tcb)
p-la$_{i_1}$/p-sc$_{i_1+1}$/p-nta
n-la$_{i_2}$/n-sc$_{i_2+1}$/n-nta
logic gate lvl. n_L (lg$_{n_L}$)
n_L (n_L even)
analog
digital

Fig. 1.3 Library for building block recognition of analog, mixed-signal and digital circuits. The analog part shows a subset from the library presented in [14].

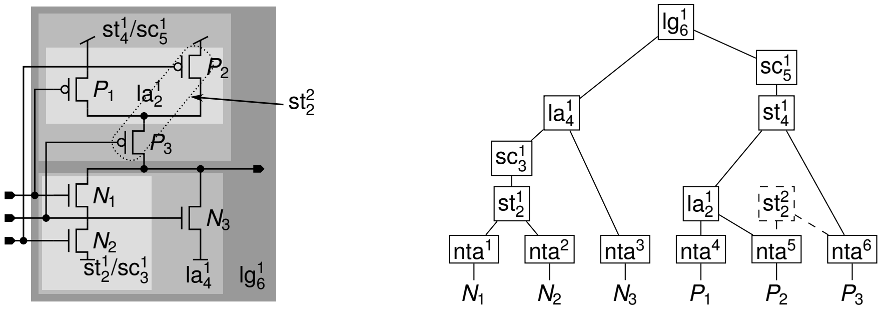


Fig. 1.4 And-nor gate [18] with recognized building blocks

Figure 1.4 illustrates how this library can be used to recognize the building blocks of the and-nor gate from [18]. First, normal transistor arrays nta^1 to nta^6 are recognized for every transistor in the circuit. After that, a stack st$_2^1$ covering N_1, N_2 and a logic array la$_2^1$ covering P_1, P_2 are found. For the third hierarchy level a stack chain (sc$_3^1$) is formed out of stack st$_2^1$. In hierarchy level four a logic array (la$_4^1$) and a stack (st$_4^1$) are recognized. Stack st$_4^1$ becomes part of a stack chain sc$_5^1$ on hierarchy level five. Finally, logic gate lg$_6^1$ is recognized on hierarchy level six.

Comparing the netlist to the library does not unambiguously yield this result. Additional building blocks can be recognized, e.g., the stack st$_2^2$. Normal transistor array nta^5 would be part of la$_2^1$ and st$_2^2$ at the same time. In the following, we will show how such ambiguities can be resolved by determining a dominating building block, i.e., one building block is kept and one is removed.

1.3.2 Recognition Conflicts and Their Resolution

For pairs used in analog circuits an ambiguity resolution concept was presented by [14]. An enhanced version, capable of handling chains and arrays as well, is described in the following.

For ambiguity resolution two building blocks are considered together with their transitive children. The set of transitive children $C_\star(x)$ of a building block x contains the children $x.c$ of x as well as all elements of their sets of transitive children, i.e.,

$$C_\star(x) = \begin{cases} \bigcup_{y \in x.c} \left(\{y\} \cup C_\star(y)\right) & x \in \mathscr{B} \\ \emptyset & x \in \mathscr{D} \end{cases} . \tag{1.1}$$

Set $C_i(x)$ is the set of transitive children limited to the i-th child $x.c_i$ of x, i.e.,

$$C_i(x) = \{x.c_i\} \cup C_\star(x.c_i) \tag{1.2}$$

The ambiguity resolution is based on a dominance graph (Fig. 1.5).

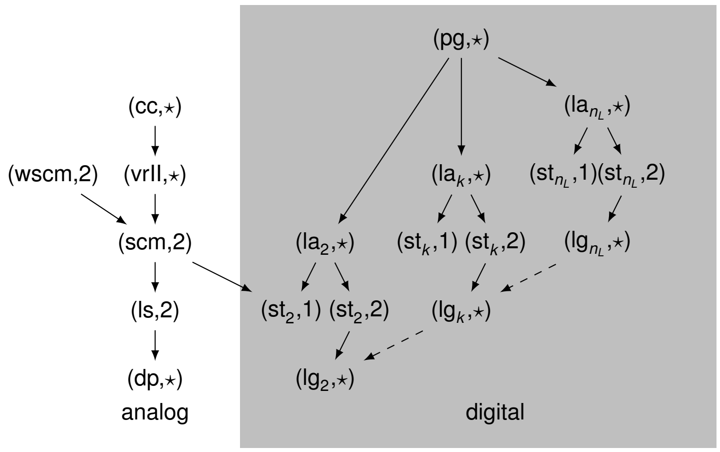


Fig. 1.5 Dominance graph for the library shown in Fig. 1.3. The analog part is based on [14].

Definition 3 (Dominance Graph). A dominance graph G_D is a directed graph $G_D = (N_{G_D}, E_{G_D})$. The nodes are pairs $(t,i) \in N_{G_D} = T_{\mathscr{B}} \times \{1,2,\star\}$, where t is a building block type and i refers to one of the set of transitive children defined above. The edges are pairs of nodes, i.e., $E_{G_D} = N_{G_D}^2$.

Definition 4 (Dominance). A building block x_1 dominates a building block x_2 iff

$$\exists_{(i,j)\in\{1,2,\star\}^2} \left(C_i(x_1) \cap C_j(x_2) \neq \emptyset\right) \wedge \text{reachable}_{G_D}((x_2.t, j), (x_1.t, i)) \,. \qquad (1.3)$$

The first part checks if there is a common transitive child using one of the sets C_1, C_2 and $C_\star$. The second part checks if the node in the dominance graph corresponding to x_2 is reachable from the node corresponding to x_1. Function reachable$_{G_D}(\mu,\nu)$ is true if node μ is reachable in G_D from node ν. This definition is based on [14].

The dominance graph for the building block library for analog, digital and mixed–signal circuits is shown in Fig. 1.5. The left, non–shaded part handles conflicts between analog building blocks. It is based on the graph presented in [14]. The right, gray shaded part handles conflicts between digital building blocks. It has to consider the recursive nature of the library. Inside each hierarchy level the following holds: Transistors that are part of a logic array must not be part of a stack. The upper transistor of a stack must not be part of a logic gate. In the example (Fig. 1.4) this prevents recognition of a false logic gate consisting of N_1 and P_3. A logic gate from a higher hierarchy level will always dominate logic gates from a lower hierarchy level. This means, in case that multiple logic gates are detected only the largest one is kept. This includes transitive relations, e.g., a stack from level 4 will dominate a logic gate from level 2. In case a transistor is part of a pass–gate it must not be part of another building block.

```
$i \leftarrow 0$; $\mathcal{B} \leftarrow \emptyset$
repeat
    $i \leftarrow i + 1$; $B_i \leftarrow \emptyset$
    for $t \in L_i$
        $t$ is a
            pair:  $B_i \leftarrow B_i \cup$ findPairs($t$)
            array: $B_i \leftarrow B_i \cup$ findArrays($t$)
            chain: $B_i \leftarrow B_i \cup$ findChains($t$)
    $\mathcal{B} \leftarrow$ resolveConflicts($\mathcal{B} \cup B_i$)
until $\left((\mathcal{B} \cap B_i) = \emptyset\right) \wedge \left(i = 6, 8, ...\right)$
$\mathcal{B} \leftarrow$ removeBlocksWithoutFunction($\mathcal{B}$)
```

Fig. 1.6 Building block recognition algorithm

This allows to resolve the conflict from the example (Fig. 1.4). Building block nta^5 is a child of la_2^1 and second child of st_2^2, i.e., $C_\star(\text{la}_2^1) \cap C_2(\text{st}_2^2) = \{P_2, \text{nta}^5\}$. Since node $(\text{st}_2, 2)$ is reachable from $(\text{la}_2, \star)$, logic array la_2^1 dominates stack st_2^2.

1.3.3 Recognition Algorithm

The recognition algorithm for analog, digital and mixed-signal circuits is shown in Fig. 1.6. It is based on the algorithm presented in [14]. It was enhanced to handle the recursive library, recognize arrays and chains as well as recognize pairs faster.

The algorithm iterates over all hierarchy levels $L_i \subseteq T_{\mathscr{B}}, i = 1, 2 \ldots n_L$ of the library. In each iteration, pairs, arrays and chains are found by calling functions findPairs, findArrays and findChains, respectively. These functions are discussed below. All building blocks recognized for the current hierarchy level are collected in set B_i. Conflicts are resolved in each hierarchy level, leading to an update of the overall set of recognized building blocks $\mathscr{B}$. In contrast to a conflict resolution at the very end as suggested by [14], this has the benefit that the overall number of building blocks is kept low. Consequently, less components must be considered during subsequent steps. According to Definition 4, it is sufficient to check for each new building block $x_1 \in B_i$,

- if it is dominated by some other building block $x_2 \in B_i \cup \mathscr{B}$, or,
- if it dominates some other building block $x_2 \in B_i \cup \mathscr{B}$.

The outer loop ends if the following two conditions are met,

- no new building blocks were found in this iteration or all found building blocks were dominated, and,
- the current hierarchy level number is even and greater or equal to six.

Finally, building blocks are removed that do not have a function if they are not part of a bigger building block. For example, voltage references II, which are not part of a wide–swing cascode current mirror, are removed.

Fig. 1.7 Function findPairs

```
findPairs($t$)
$B \leftarrow \emptyset$
$X \leftarrow$ candidatePairs($t$)
for $(c_1, c_2) \in \{(c_1, c_2) \in X | r_t(c_1, c_2)\}$
    $B \leftarrow B \cup \{$newPair$(t, c_1, c_2)\}$
return $B$
```

1.3.3.1 Finding Pairs

Function findPairs is shown in Fig. 1.7. First a set of candidate pairs $X \subseteq (\mathscr{B} \cup \mathscr{D})^2$ is determined. Below, this will be described in more detail. Next, a rule function r_t is evaluated for each of these candidate pairs. In case the function is true for a pair, a new pair is created and added to the set of found pairs, which is returned in the end. The rule function r_t is specific for each pair type t. It can contain conditions about type, subtype, required and forbidden connections as well as existence of parents. For example, the rule function $r_{\mathrm{st}_k}(x_1, x_2)$ for stack type st_k on level k contains the following conditions,

$$\begin{array}{lcl}
 & (x_1.t, x_2.t) \in \left\{ \begin{array}{l} (\mathrm{la}_{k-2}, \mathrm{nta}),\ (\mathrm{la}_{k-2}, \mathrm{la}_2),\ \cdots) \\ (\mathrm{nta}, \mathrm{la}_{k-2}),\ (\mathrm{la}_2, \mathrm{la}_{k-2}),\ \cdots) \end{array} \right\} & \text{(type)} \\
\wedge & x_1.s = x_2.s & \text{(same subtype)} \\
\wedge & \eta(x_1, \mathrm{dn}) = \eta(x_2, \mathrm{sc}) & \text{(required connection)} \\
\wedge & \eta(x_1, \mathrm{sc}) \neq \eta(x_2, \mathrm{dn}) & \text{(forbidden connections)} \\
\wedge & \mathrm{parents}(x_1) = \mathrm{parents}(x_2) = \emptyset & \text{(no parents)}
\end{array} \tag{1.4}$$

The *type* condition requires one component to be a logic array from hierarchy level $k-2$. The other component can either be a normal transistor array or another logic array from any hierarchy level. Both components must have the *same subtype*. The *required connection* condition requires the drain of the first building block to connect to the source of the second building block. The *forbidden connection* condition forbids a connection between source of the first building block and drain of the second building block. Both components must have *no parents*.

Runtime of findPairs is dependent on the number of candidate pairs X. This number can be kept low by including some of above conditions in the candidate pair computation. The authors of [14] use all pairs of devices and building blocks that are of correct types. The authors of [8] use all pairs that are at least connected by one net. We combine both methods. For the stack at level k candidate set X is computed as follows,

$$X_1(n) = \left\{x_1 \in \mathscr{D} \cup \mathscr{B} | x_1.t \in \{\mathrm{nta}, \mathrm{la}_2, \mathrm{la}_4, \ldots\}) \wedge \eta(x_1, \mathrm{dn}) = n\right\} \tag{1.5a}$$

$$X_2(n) = \left\{x_2 \in \mathscr{D} \cup \mathscr{B} | x_2.t \in \{\mathrm{nta}, \mathrm{la}_2, \mathrm{la}_4, \ldots\}) \wedge \eta(x_2, \mathrm{sc}) = n\right\} \tag{1.5b}$$

$$X = \bigcup_{n \in \mathscr{N}} X_1(n) \times X_2(n) \tag{1.5c}$$

```
findArrays($t$)
$B \leftarrow \emptyset$
$X \leftarrow \{c \in \mathcal{D} \cup \mathcal{B} | r_t(c)\}$
$K \leftarrow \cup_{x \in X}\{k_t(c)\}$
for $\kappa \in K$
    $X_\kappa \leftarrow \{x \in X | k_t(x) = \kappa\}$
    $|X_\kappa| \geq t.m$
        true
            $B \leftarrow B \cup \{\text{newArray}(t, X_\kappa)\}$
        false
return $B$
```

Fig. 1.8 Function findArrays

```
findChains($t$)
$B \leftarrow \emptyset$
$X \leftarrow \{c \in \mathcal{D} \cup \mathcal{B} | r_t(c)\}$
$y_0 \in \{x \in X | \gamma_X^-(x) \neq 1\}$
    $y \leftarrow \text{unbranchedChain}(x_0, X)$
    $B \leftarrow B \cup \{\text{newChain}(t, y)\}$
return $B$
```

Fig. 1.9 Function findChains

Functions $X_1(n)$ and $X_2(n)$ return candidates for the first and second component for a specific net $n \in \mathcal{N}$, respectively. Pairs are then computed for each net $n \in \mathcal{N}$ by evaluating $X_1(n)$ and $X_2(n)$. The resulting set X only contains pairs where the connection condition and parts of the type condition are fulfilled.

1.3.3.2 Finding Arrays

The algorithm to find arrays is depicted in Fig. 1.8. First, the algorithm creates a set X of candidate children by evaluating a rule function r_t, which is specific for each array type t. It can consist of conditions about type, subtype, connectivity and existance of parents. The rule function $r_{\text{dta}}(x)$ for a diode transistor array contains the following conditions:

$$\underbrace{x.t = \text{trans}}_{\text{(type)}} \wedge \underbrace{\eta(x, \text{gt}) = \eta(x, \text{dn})}_{\text{(required connection)}} \wedge \underbrace{\eta(x, \text{dn}) \neq \eta(x, \text{sc})}_{\text{(forbidden connection)}} \tag{1.6}$$

It enforces type transistor and a gate drain connection. It forbids a connection between drain and source. The key function k_t maps each component in X to a tuple of nets, such that components connected in parallel get the same key. The key function k_{dta} for a diode transistor array is,

$$k_{\text{dta}}(x) = (\eta(x, \text{dn}), \eta(x, \text{sc})). \tag{1.7}$$

This means, for a diode transistor array all transistors are grouped together that connect to the same net at their drain pins and their source pins. If more than a minimum number $t.m$ building blocks are connected in parallel then a new array is created. For the diode transistor array dta.$m = 1$, i.e., an array is always created. Finally, the set B of new arrays is returned.

1.3.3.3 Finding Chains

Function findChains is shown in Fig. 1.9. First, the algorithm computes a set X of candidate children using a rule function r_t, which is specific for each chain type t. It can use the conditions described for arrays. All candidate children must be pairs. The rule function $r_{\mathrm{sc}_k}(x)$ for a stack chain on level k contains the following conditions:

$$x.t = \mathrm{st}_{k-1} \ . \qquad (1.8)$$

It requires x to be a stack on level $k-1$.

Next, all tuples $y = (y_0, y_1, \dots y_{\mathrm{last}})$ with the following properties are found:

- $\gamma^-(y_0) = |\{x \in X | x.c_2 = y_0.c_1\}| \neq 1$, i.e., more than one or no candidate in X share the second child with y_0.
- $y_i.c_2 = y_{i+1}.c_1$ for $y_i \neq y_{\mathrm{last}}$, i.e., the second child of each building block y_i is the first child of the next building block y_{i+1}.
- $|\{x \in X | x.c_1 = y_{\mathrm{last}}.c_2\}| \neq 1$, i.e., the chain can not be continued beyond y_{last}.

Finally, a new chain is created for each y and returned in B.

1.3.3.4 Discussion

The analog building block recognition described in [14] is included in this algorithm. It corresponds to the analog part of the library in Fig. 1.3 and the dominance graph in Fig. 1.5. The algorithm corresponds to the algorithm of Fig. 1.6 when all building blocks are pairs. Consequently, the results obtained for the algorithm of [14] can be transfered to the new algorithm.

The authors of [1] suggested to recognize simple current mirrors and level-shifters by recognizing diode connected transistors first. Application of the principle from [1] to the library from [14] resulted in the new hierarchy level 1 shown in Fig. 1.3. This has the advantage of faster recognition of pairs because less rules must be evaluated.

1.4 Structural Signal Flow Analysis

After applying the building block algorithm from the previous section to a circuit, basic building blocks such as pass-gates or simple current mirrors are known. Figure 1.10 illustrates this for a latch from [19]. It consists of logic gates lg_2^1 to lg_2^3 as well as pass–gates pg_2^1 and pg_2^2. This information is now used to generate the Enhanced Structural Signal Flow Graph [6] (ESFG) of the circuit, which combines qualitative behavioral and structural information. This graph is then used to assign a direction to each pass–gate, partition the circuit into an analog and digital part and to identify feedback loops.

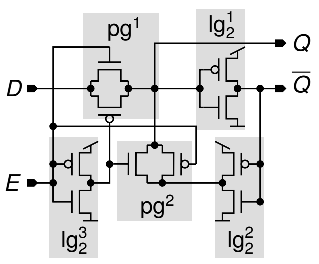


Fig. 1.10 Latch [19] with recognized building blocks

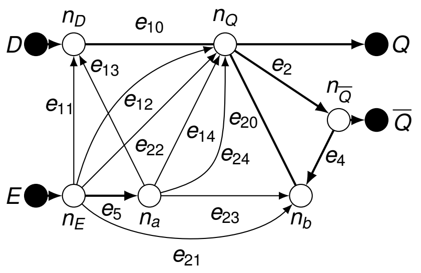


Fig. 1.11 Generated ESFG

1.4.1 Generation

An ESFG [6] is a directed graph. The nodes of the graph are formed by the nets of the circuit. An edge models a qualitative influence from one net to another. An edge from net n_i to n_j means that a change of a branch current or voltage of n_i causes a change of a branch current or voltage of n_j. The relation between edges and the recognized building blocks is modeled by some edge attributes. Only top–level building blocks without parents are considered. The ESFG is generated as follows.

- For a logic gate edges from each input to the output are generated.
- A pass–gate generates an undirected edge from drain to source. Directed edges are generated from both gates to drain and source. These edges are called control edges.
- For analog building blocks the generation is described in [6, 7]. For example, for current mirrors one edge from the input to the output is generated.
- For each port of the circuit a port node is generated and connected to the corresponding net.

For the latch example this is illustrated by Fig. 1.11. Logic gates lg_2^1 to lg_2^3 are represented by edges e_1 to e_3. Pass–gate pg^1 is represented by undirected edge e_{10} and control edges e_{11} to e_{14}. Pass–gate pg^2 is represented by edges e_{20} to e_{24}. Circuit ports E, D, Q, $\overline{Q}$ are represented by port nodes.

1.4.2 Assignment of Pass–Gate Directions

After the generation step, pass–gates are represented as undirected edges, e.g., edge e_{10} in Fig. 1.11. In reality pass–gates are only used in one direction. The problem is related to the problem of determining the signal flow direction of transistors in switch–level simulation [2]. However, only a small part of the ESFG edges is undirected in our case, which allows to used a different approach which is described in the following.

Assume e is an undirected edge connecting nodes ν and μ. It is replaced by a directed edge from μ to ν, if the following conditions hold simultaneously.

- An output node is reachable from ν without traversing e, and,
- no edge representing a logic gate ends at ν.

Simultaneously with the assignment, edges pointing from the control inputs of the pass–gate to μ are removed.

For the undirected edge e_{10}, connecting n_D and n_Q in the example of Fig. 1.11, output node $n_{\overline{Q}}$ is reachable from n_Q but not from n_D without traversing edge e_{10}. Consequently the edge points from n_D to n_Q. The undirected edge e_{20} between n_2 and n_Q is found to point from n_2 to n_Q. The control edges are removed accordingly (Fig. 1.12).

In some cases, it is not possible to assign directions to all pass–gates at once. In these cases above conditions are repeatedly evaluated for all pass–gates without assigned direction. In each iteration at least one pass–gate direction is assigned. The algorithm needs n_{pg} iterations at maximum, where n_{pg} is the number of pass–gates.

The computation of the logic functions (see Section 1.5.1) for building blocks can be done before this step. In this case, it can be checked whether the output of a logic gate can be in high impedance state.

1.4.3 Analog / Digital Partitioning

For further processing, the ESFG must be partitioned into an analog and digital part. Therefore a signal type is assigned to each node. This signal type can be either *unknown*, *analog* or *digital*. The signal type for each node is determined based on the edges of the graph and the building blocks they represent. For each building block type a specific set of conditions for the connected nodes exists. For example, input and output of a current mirror must be of type analog. In addition, the user can specify the signal type of inputs and outputs of the circuit. Overall, we get a set of conditions forming a constraint satisfaction problem which is solved by a constraint programming method, e.g., [17].

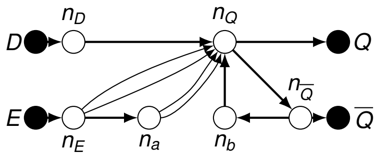


Fig. 1.12 ESFG after assignment of pass-gate directions

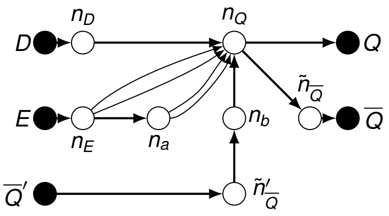


Fig. 1.13 Temporal ESFG

In some cases, this leads to conflicting requirements for a node, i.e., it must be analog and digital at the same time. This happens in case an analog building block was wrongfully recognized in the digital part or vice versa. Such conflicts are resolved by back–annotating the signal type to the nets of the circuit. Next, the building block recognition is rerun using additional rules for the signal types at the pins of a building block.

For a pure analog or digital circuit this step has no effect. Therefore the ESFG in Fig. 1.12 does not change.

1.4.4 Transformation to Temporal ESFG

In case of sequential circuits such as latches, the ESFG contains feedback loops. In order to compute the logic function of such circuits, a temporal ESFG is introduced, which is an acyclic ESFG and adds a time concept.

Definition 5 (Temporal ESFG). A temporal ESFG is an acyclic ESFG. It refers to a virtual normalized clock with clock period 1 that is at least twice the real clock frequency, i.e., the real clock can be sampled. Each node gets an additional clock cycle attribute, indicating if the node belongs to the current or a previous clock cycle.

The transformation from the ESFG to the temporal ESFG is described in the following. All loops of the ESFG are computed by finding strongly connected components in the graph. All nodes, where an edge to a node outside the loop starts, are called output nodes of the loop. All nodes, where an edge from a node outside the loop ends, are called input nodes of the loop. The feedback path of a loop is the path from an output node to an input node that does not contain any other output or input node of the loop. Some node n_s of this path, which is not an input node of the loop, is then selected as node to represent the state of the loop. This node is split up into two nodes $\tilde{n}_s$ and $\tilde{n}'_s$, which represent the state at the current and previous time step, respectively. All edges going from n_s to a node inside the loop are assigned to $\tilde{n}'_s$. All other edges are assigned to $\tilde{n}_s$. In addition, an output port node corresponding to $\tilde{n}_s$ and an input port node corresponding to $\tilde{n}'_s$ is created.

Figure 1.13 illustrates this for the example. The ESFG (Fig. 1.12) contains a loop consisting of n_Q, $n_{\overline{Q}}$ and n_b. Node n_Q is an input node of the loop and nodes n_Q, $n_{\overline{Q}}$ are output nodes. The feedback path is $n_{\overline{Q}}, n_b, n_Q$. Node $n_{\overline{Q}}$ is split up into node $\tilde{n}_{\overline{Q}}$ and $\tilde{n}'_{\overline{Q}}$. Input port $\overline{Q}'$ is created. The resulting temporal ESFG is shown in Fig. 1.13.

1.5 Logic Function Extraction

Based on the temporal ESFG the logic function of the circuit can now be computed. This is done in two steps. First, the logic function for all recognized building blocks is computed. Afterwards, the logic function for the complete circuit is determined.

Unless denoted otherwise, a four valued logic [4] with 0, 1, Z (high impedance), U (unknown) is used in the following. All logic functions are represented using ROBDDs [3].

1.5.1 Computation of Logic Function for Building Blocks

In this step, the logic function of single logic gates is determined. For CMOS circuits this requires in general to evaluate the serial and parallel connections of the pull–up and pull–down network [19]. Algorithmic implementations can be found in, e.g., [4, 5, 9, 11, 20]. Our approach builds on the hierarchical recognition result computed by the algorithm presented in Sec. 1.3.

Table 1.1 lists the logic function associated with each building block from the library for digital circuits. It uses the operators $\oplus$ and $\diamond$, which are defined as follows.

$$a \oplus b :\Leftrightarrow \begin{cases} a & (a = b) \vee (b = Z) \\ b & a = Z \\ U & \text{otherwise} \end{cases} \qquad a \diamond b :\Leftrightarrow \begin{cases} a & b = U \\ b & a = U \\ a \oplus b & \text{otherwise} \end{cases} \tag{1.9}$$

The operator $\oplus$ is the “merge” operator from [4]. The result is a defined logic state, i.e., zero or one, if a and b have the same value or one is high impedance. If both are high impedance the result is “Z”, otherwise the result state is undefined. The operator $\diamond$ considers in addition, that undefined states can be canceled out in case of parallel connections, i.e., the result is a defined logic state in case a or b is 0 or 1 and the other one is undefined.

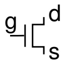
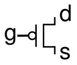
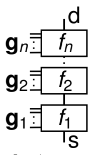
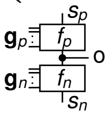
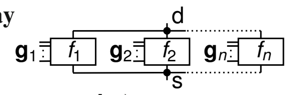
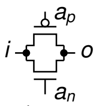
An NMOS transistor for example shows a logic “0” at the drain pin if the gate is at logic “1” (i.e., vdd) and source is at logic “0”. The drain pin is at high impedance state if the transistor is off. This is the case for “0” at the gate or “Z” at the source pin. In all other cases the output is unknown (Table 1.1). The logic function for a PMOS transistor is found analogously.

The logic function at the drain pin of a stack chain is formed out of the logic functions f_1 to f_n of its children. The gate inputs of these children are described by vectors $\mathbf{g}_1$ to $\mathbf{g}_n$. The overall logic function is the logic function f_n with f_{n-1} substituted for the source variable. These substitutions are continued until f_1 is reached.

The logic function at the output o of a logic gate are the logic functions of the p– and the n–block combined by the $\oplus$ operator. This includes that the output becomes high–impedance in case no block is on or unknown in case both blocks are on at the same time.

The logic function of a logic array is the logic functions f_1 to f_n of the children combined by the $\diamond$ operator. The logic function at the output o of a pass–gate is equal to the input i in case the pass–gate is on, otherwise it is “Z”.

Table 1.1 Logic functions for digital building blocks

n–transistor  $d = f_{\text{n-t}}(g,s) = \begin{cases} 0 & (g=1)\wedge(s=0) \\ Z & (g=0)\vee(s=Z) \\ U & \text{otherwise} \end{cases}$	**p–transistor**  $d = f_{\text{p-t}}(g,s) = \begin{cases} 1 & (g=0)\wedge(s=1) \\ Z & (g=1)\vee(s=Z) \\ U & \text{otherwise} \end{cases}$
stack chain  $d = f_{\text{sc}}([\mathbf{g}_1\mathbf{g}_2\cdots\mathbf{g}_n],s) = f_n(\mathbf{g}_n, f_{n-1}(\mathbf{g}_{n-1},\cdots f_1(\mathbf{g}_1,s)\cdots))$	**logic gate**  $o = f([\mathbf{g}_p\mathbf{g}_n],s_p,s_n) = f_n(\mathbf{g}_n,s_n)\oplus f_p(\mathbf{g}_p,s_p)$
logic array  $d = f_{\text{la}}([\mathbf{g_1}\mathbf{g_2}\cdots\mathbf{g_n}],s) = f_1(\mathbf{g}_1,s)\diamond f_2(\mathbf{g}_2,s)\diamond\cdots\diamond f_n(\mathbf{g}_n,s)$	**pass–gate**  $o = f(i,a_p,a_n) = \begin{cases} i & (a_n=1)\wedge(a_p=0) \\ Z & (a_n=0)\wedge(a_p=1) \\ U & \text{otherwise} \end{cases}$

This is illustrated by the example shown in Fig. 1.14. The logic function for the complete gate,

$$f_{\text{NAND}}([a\,b]) = \underbrace{f_{N_2}(a, f_{N_1}(b,0))}_{f_N([a,b],0)} \oplus \underbrace{\left(f_{P_1}(a,1)\diamond f_{P_2}(b,1)\right)}_{f_P([a,b],1)} \tag{1.10}$$

is formed by the logic function f_P of the logic array consisting of P_1 and P_2 as well as the logic function f_N of the stack chain consisting of N_1 and N_2. Logic function f_P is formed by the logic functions of P_1 and P_2 and logic function f_N is formed by the logic functions of N_1 and N_2, yielding

$$f_N([a\,b],0) = \begin{cases} 0 & (a=1)\wedge(b=1) \\ Z & (a=0)\vee(b=0) \\ U & \text{otherwise} \end{cases} \quad f_P([a\,b],1) = \begin{cases} 1 & (a=0)\vee(b=0) \\ Z & (a=1)\wedge(b=1) \\ U & \text{otherwise} \end{cases}. \tag{1.11}$$

Logic function f_N represents the output of the pull–down network, which is the drain of N_1. It is vss ("0") in case both inputs are one, if both inputs are zero it is high-impedance. In case one input is high–impedance or unknown f_N is unknown. Logic function f_P represents the output of the pull–up network at point x_1 in Fig. 1.14.

This results in the following logic function for the complete gate,

$$f_{\text{NAND}}([a\,b]) = \begin{cases} 0 & (a=1)\wedge(b=1) \\ 1 & (a=0)\vee(b=0) \\ U & \text{otherwise} \end{cases}, \tag{1.12}$$

which is the logic function of a NAND gate. The unknown case occurs if one of the inputs is in unknown or high–impedance state. Since the NAND gate is no tristate gate, the overall logic function does not include a high–impedance case.

1.5.2 Computation of Overall Logic Function

The overall logic function is computed by assigning logic variables to each node. We use a temporal logic, i.e., the logic variables refer to different time steps. Next, the temporal ESFG is traversed in topological order, i.e., each node in the graph is visited after all nodes it depends on. During this traversal, the logic functions are substituted into each other. In case two building blocks (e.g., pass gates) have outputs o_1, o_2 on the same node, the logic function for the node is calculated as $o_1 \oplus o_2$. It is assumed, that the inputs of the circuit are in a defined logic state, i.e., they are not "U" or "Z".

For the example circuit from Fig. 1.10 and the temporal ESFG from Fig. 1.13 the assigned logic variables are shown in Fig. 1.15. Logic variables $a(t)$, $b(t)$, $D(t)$, $E(t)$, $Q(t)$ and $\overline{Q}(t)$ refer to the current time step. Logic variable $\overline{Q}(t-1)$ refers to the previous time step. It holds.

$$a(t) = \begin{cases} 0 & E(t) = 1 \\ 1 & E(t) = 0 \end{cases} \qquad b(t) = \begin{cases} 0 & \overline{Q}(t-1) = 1 \\ 1 & \overline{Q}(t-1) = 0 \\ U & \text{otherwise} \end{cases} \tag{1.13}$$

$$Q(t) = \begin{cases} 0 & \left[(E(t)=1) \wedge (D(t)=0)\right] \vee \left[(E(t)=0) \wedge (\overline{Q}(t-1)=1)\right] \\ 1 & \left[(E(t)=1) \wedge (D(t)=1)\right] \vee \left[(E(t)=0) \wedge (\overline{Q}(t-1)=0)\right] \\ U & \text{otherwise} \end{cases} \tag{1.14}$$

$$\overline{Q}(t) = \begin{cases} 0 & \left[(E(t)=1) \wedge (D(t)=1)\right] \vee \left[(E(t)=0) \wedge (\overline{Q}(t-1)=0)\right] \\ 1 & \left[(E(t)=1) \wedge (D(t)=0)\right] \vee \left[(E(t)=0) \wedge (\overline{Q}(t-1)=1)\right] \\ U & \text{otherwise} \end{cases} \tag{1.15}$$

Logic function $a(t)$ can only become "0" or "1" because input $E(t)$ is assumed to be in a defined logic state. No such assumption is made for $\overline{Q}(t-1)$. Consequently, $b(t)$ can become unknown in case $\overline{Q}(t-1)$ is unknown or high-impedance. Overall

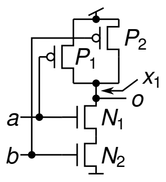


Fig. 1.14 NAND gate

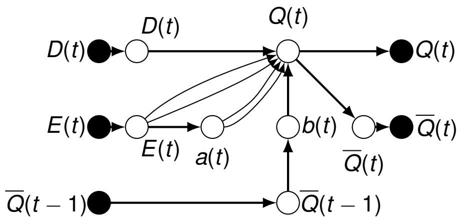


Fig. 1.15 ESFG with assigned logic variables

Table 1.2 Recognition results for different standard cell libraries

Library	No. Cells	Analysis Time	Coverage
Lib 1	32	1 sec.	100.0%
Lib 2	134	4 sec.	100.0%
Lib 3 – Tech 1	$\sim$ 600	18 sec.	97.6%
Lib 3 – Tech 2	$\sim$ 550	13 sec.	99.6%
Lib 3 – Tech 3	$\sim$ 700	27 sec.	99.1%
Lib 4	$\sim$ 850	37 sec.	95.2%

logic function $Q(t)$ is input $D(t)$ in case $E(t)$ is set otherwise it is the inversion of $\overline{Q}(t-1)$. Logic function $\overline{Q}(t)$ is the inversion of $Q(t)$. This corresponds to a latch.

1.6 Application Examples

In the following application to digital standard cell libraries and mixed-signal circuits is discussed including experimental results.

1.6.1 Description Generation for Digital Standard Cell Libraries

The approach is used to automatically generate a library description for digital standard cell libraries. The description includes a decomposition into pass–gates and logic gates, the ESFG, the logic function of the standard cell and a table listing possible single input switching events together with the possible values at the other inputs and the resulting output behavior. These events are a necessary input for automatic timing characterization of digital standard cell libraries. The decomposition into logic gates and pass–gates corresponds to a decomposition of multi–stage gates into single–stage gates. This is a required input for the current–source modeling approach of [10] and the aging analysis approach of [12].

In the experiment, the building block recognition with the digital part of the library is used as well as the structural signal flow analysis and the logic function extraction. Additional post–processing is used to generate the table of all possible single input switching events.

We performed this analysis for 4 different standard cell libraries (Table 1.2). Library 1 is the standard cell library included in the FreePDK presented in [18]. Library 2 is the Nangate open cell library. Library 3 is an industrial standard cell library which was available for three different technology nodes. Library 4 is an industrial standard cell library, too.

Table 1.2 shows that these libraries contained between 30 and 850 cells. In all cases the analysis for the complete library took less than 1 minute. All runtimes were normalized to an Intel® Xeon® 2.33 GHz computer with 4 GB RAM running Ubuntu and using 4 of 8 cores in parallel.

Column four of Table 1.2 gives the recognition coverage of the presented method. For libraries 1 and 2 all cells were recognized correctly. For libraries 3 and 4, the building block analysis was not able to fully decompose all cells into pass–gates and logic gates. Typically, these cells were not designed according to standard CMOS principles. However, these cells can be included by extending the library accordingly. Overall, more than 95% of all cells were correctly recognized for the industrial libraries.

1.6.2 Structural Analysis of Mixed-Signal Circuits

The new mixed–signal capabilities of the structural analysis were evaluated using a voltage–controlled ring oscillator (Fig. 1.16) and a charge–pump (Fig. 1.18).

The voltage–controlled ring oscillator generates a digital clock signal. The frequency of the clock signal can be adjusted by the analog control voltage applied at input c. The building block recognition computed 4 NMOS simple current mirrors, 3 PMOS simple current mirrors and 5 logic gates on level 2, i.e., inverter. It is not possible to get the correct recognition result by computing analog and digital building blocks independently: A logic gate on level 4 consisting of N_3, N_4, P_3, P_4 would be found, which would contradict the current mirrors formed by N_1,N_4 and P_1,P_2.

Fig. 1.17 shows the corresponding ESFG of the voltage–controlled ring oscillator. The partitioning into analog and digital part is symbolized by the node shape. The analog control circuitry as well as the digital feedback loop are clearly visible.

The charge pump shown in Fig. 1.18 is based on [15]. The output is usually connected to the loop filter of a PLL. Digital inputs D and U control the direction of the output current. The building block recognition computed 2 NMOS simple current mirrors, a PMOS simple current mirror and two logic gates. Transistors N_6 and N_7 as well as P_5 and P_6 would match a differential pair. These differential pairs were dropped because they connect to digital inputs D and U. Transistors N_7 and P_6 would form a logic gate, which was dropped because output o is specified as analog. The corresponding ESFG is shown in Fig. 1.19.

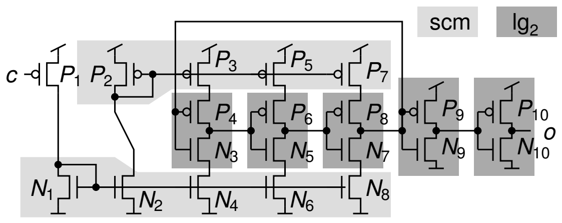
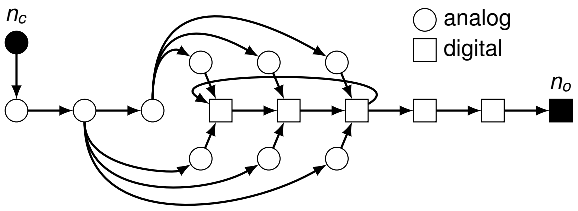


Fig. 1.16 Voltage–controlled ring oscillator with recognized building blocks

Fig. 1.17 ESFG of voltage-controlled ring oscillator



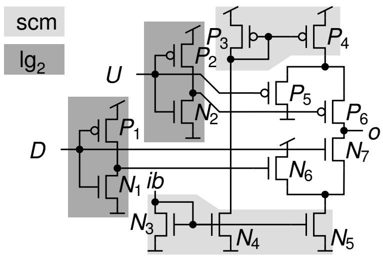


Fig. 1.18 Charge pump with recognized building blocks

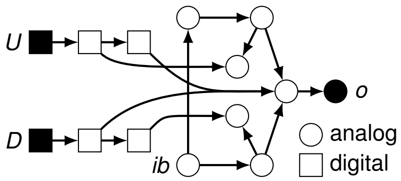


Fig. 1.19 ESFG of the charge pump

1.7 Conclusion

This chapter presented a new method for the automatic structural and functional analysis of analog, digital and mixed-signal circuits. Its first step is the recognition of building blocks such as simple current mirrors and logic gates. These results are then used to generate an Enhanced Structural Signal Flow Graph (ESFG). Based on that, true pass-gate directions are computed and feedback paths are broken up. Finally, the logic function is determined for the digital circuit parts.

Experimental results show successful application of the algorithm to several digital standard cell libraries with more than 95% of correctly recognized cells. Structural analysis of mixed-signal circuits was demonstrated using a voltage-controlled ring oscillator and a charge pump.

Acknowledgements. We would like to thank Jens Bargfrede from Infineon Technologies AG for all his help and encouragement. We would also like to remember Michael Mirbeth († 2009) from Infineon Technologies AG, who gave us great support in this work.

References

[1] Arsintescu, B.G.: A Method for Analog Circuits Visualization. In: IEEE International Conference on Computer Design (ICCD), pp. 454–459 (1996)

[2] Blaauw, D.T., Saab, D.G., Long, J., Abraham, J.A.: Derivation of signal flow for switch-level simulation. In: ACM/IEEE Design Automation Conference (DAC), pp. 301–305 (1990)

[3] Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers 35(8), 677–691 (1986)

[4] Bryant, R.E.: Extraction of gate level models from transistor circuits by four-valued symbolic analysis. In: IEEE International Conference on Computer-Aided Design, ICCAD 1991. Digest of Technical Papers, pp. 350–353 (1991)

[5] Dagenais, M.R.: Efficient algorithmic decomposition of transistor groups into series, bridge, and parallel combinations. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 38(6), 569–581 (1991)

[6] Eick, M., Graeb, H.: MARS: Matching-driven Analog Sizing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2012)

[7] Eick, M., Strasser, M., Lu, K., Schlichtmann, U., Graeb, H.: Comprehensive Generation of Hierarchical Placement Rules for Analog Integrated Circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30(2), 180–193 (2011)

[8] Graeb, H., Zizala, S., Eckmueller, J., Antreich, K.: The Sizing Rules Method for Analog Integrated Circuit Design. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 343–349 (2001)

[9] Kim, W., Shin, H.: Hierarchical LVS based on hierarchy rebuilding, pp. 379–384 (1998)

[10] Knoth, C., Kleeberger, V.B., Nordholz, P., Schlichtmann, U.: Fast and Waveform Independent Characterization of Current Source Models. In: IEEE/VIUF International Workshop on Behavioral Modeling and Simulation (BMAS), pp. 90–95 (2009)

[11] Lester, A., Sabet, P.B., Greiner, A.: YAGLE, a second generation functional abstractor for CMOS VLSI circuits. In: Proceedings of the Tenth International Conference on Microelectronics, ICM 1998, pp. 265–268 (1998)

[12] Lorenz, D., Barke, M., Schlichtmann, U.: Aging analysis at gate and macro cell level. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 77–84 (2010)

[13] Lullau, F., Hoepken, T., Barke, E.: A Technology Independent Block Extraction Algorithm. In: ACM/IEEE Design Automation Conference (DAC), pp. 610–615 (1984)

[14] Massier, T., Graeb, H., Schlichtmann, U.: The Sizing Rules Method for CMOS and Bipolar Analog Integrated Circuit Synthesis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27(12), 2209–2222 (2008)

[15] Rhee, W.: Design of high-performance CMOS charge pumps in phase-locked loops. In: IEEE International Symposium on Circuits and Systems (ISCAS), vol. 2, pp. 545–548 (1999)

[16] Rubanov, N.: A High-Performance Subcircuit Recognition Method Based on the Nonlinear Graph Optimization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25(11), 2353–2363 (2006)

[17] Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling and Programming with Gecode (2010), http://www.gecode.org/doc/3.4.0/MPG.pdf

[18] Stine, J., Castellanos, I., Wood, M., Henson, J., Love, F., Davis, W., Franzon, P., Bucher, M., Basavarajaiah, S., Oh, J., Jenkal, R.: FreePDK: An Open-Source Variation-Aware Design Kit. In: IEEE International Conference on Microelectronic Systems Education, MSE 2007, pp. 173–174 (2007)

[19] Weste, N.H.E., Harris, D.: CMOS VLSI Design - A Circuits and Systems Perspective. Pearson Education, Inc. (2005)

[20] Yang, L., Shi, C.J.R.: FROSTY: A Fast Hierarchy Extractor for Industrial CMOS Circuits. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 741–746 (2003)

[21] Zhang, N., Wunsch II, D.C.: Speeding up VLSI Layout Verification Using Fuzzy Attributed Graphs Approach. IEEE Transactions on Fuzzy Systems 14(6), 728–737 (2006)

[22] Zhang, N., Wunsch II, D.C., Harary, F.: The subcircuit extraction problem. In: Proceeding of IEEE International Behavioral Modeling and Simulation Workshop 2005, vol. 22(3), pp. 22–25 (2003)

Chapter 2
Efficient Synthesis Methods for High-Frequency Integrated Passive Components and Amplifiers

Bo Liu and Georges Gielen

Abstract. Existing design automation methods for RF ICs and microwave passive components often rely on parasitic-aware lumped equivalent circuit models. That framework is difficult to apply to synthesis tasks at high frequencies (e.g. 40GHz and above) due to the distributed effect. When directly embedding the computationally expensive electromagnetic (EM) simulations in the optimization loop, a too long synthesis time results. This chapter presents a new method for high-frequency integrated passive component synthesis, called Memetic Machine Learning-based Differential Evolution (MMLDE), and the first method for mm-wave integrated circuit synthesis, called Efficient Machine Learning-based Differential Evolution (EMLDE), both addressing the problem of obtaining highly optimized design solutions in a very practical time. The common idea of these two methods is the on-line surrogate model assisted evolutionary algorithm (SAEA), where a computationally cheap surrogate model is constructed adaptively in the optimization process to replace expensive EM simulations. The differences between the two algorithms are that a memetic SAEA is built to enhance the optimization ability and efficiency in MMLDE, while a decomposition method is used to address the "curse of dimensionality" of SAEA in EMLDE. Experimental results show the effectiveness and the high efficiency obtainable with MMLDE and EMLDE.

2.1 Introduction

In recent years, design methodologies for high-frequency and mm-wave circuits have attracted a lot of attention. In particular, research and applications on RF building blocks for 40 GHz to 120 GHz and beyond are increasing drastically [1]. Existing RF IC synthesis methodologies, however, focus on low-GHz cases [2,3]. Even till now, the synthesis methodologies for mm-wave frequencies are still

Bo Liu · Georges Gielen
ESAT-MICAS, KU Leuven, Kasteelpark Arenberg 10, B3001, Leuven, Belgium
e-mail: {Bo.Liu,Georges.Gielen}@esat.kuleuven.be,
liubo168@gmail.com

M. Fakhfakh et al. (Eds.): *Analog/RF & Mixed-Signal Circuit Sys. Design*, LNEE 233, pp. 27–52.
DOI: 10.1007/978-3-642-36329-0_2 © Springer-Verlag Berlin Heidelberg 2013

lacking. Designers rely on experience and simulation verifications when designing these circuits. Due to the high-performance and tightening time-to-market requirements, this “experience and trial” method or local optimization is often not good enough.

The reason why existing synthesis methods cannot be extended to mm-wave frequencies is that they all rely on parasitic-aware equivalent circuit models for passive components [2,3,4]. Due to the distributed effects, however, an accurate equivalent circuit model is difficult to find at mm-wave frequencies. The solution is to include electromagnetic (EM) simulation based on the actual layout structure in the optimization loop. However, EM simulation is computationally very expensive. When combining it directly with techniques like evolutionary computation (EC) [5], like at low frequencies, high-quality solutions can be obtained, but the time consumption is extremely large. For example, the synthesis of a transformer typically needs more than 20 hours, and the synthesis of a linear amplifier needs about 10 days. This is not practical for real-world applications.

In this chapter, efficient synthesis method for mm-wave-frequency passive components and linear amplifiers will be introduced. The Memetic Machine Learning-based Differential Evolution (MMLDE) method [6] for the synthesis of integrated passive components will briefly be introduced first. The key idea of MMLDE is the on-line surrogate model-based memetic evolutionary optimization mechanism, whose training data are generated adaptively in the optimization process. By using the Gaussian Process with the expected improvement prescreening method and an artificial neural network with the prediction value in the proposed search mechanism, surrogate models are constructed on-line to predict the performances. Hence, the computationally expensive EM simulations are only used in the necessary part of the design space, which is guided by the prediction and prescreening methods. Compared with directly using EC algorithms, MMLDE can obtain comparable results, and has approximately a tenfold improvement in computational efficiency. The Efficient Machine Learning-based Differential Evolution (EMLDE) method [7] for the synthesis of mm-wave linear amplifiers will then be elaborated next. A decomposition method is used, which separates the design variables that require expensive EM simulations and the variables that only need cheap S-parameter circuit simulations. Hence, a low-dimensional but more complex expensive optimization problem is generated. By the proposed core algorithm integrating adaptive population generation, naive Bayes classification, Gaussian process and differential evolution, the generated low-dimensional expensive optimization problem can be solved efficiently (thanks to the on-line surrogate model), and global search can be achieved (thanks to the evolutionary computation algorithm). A 100GHz three-stage differential amplifier in a 90nm CMOS technology is shown as an example. The power gain reaches 10dB with more than 20GHz bandwidth. The synthesis costs only 25 hours, having a comparable result and a 9 times speed enhancement compared with directly using the EM simulator in combination with a global optimization algorithm.

The remainder of this chapter is organized as follows. Section 2.2 reviews the existing works for RF IC synthesis, and motivates the construction of the EMLDE algorithm. Section 2.3 introduces the basic mathematical and computational intelligence techniques used in this chapter. Section 2.4 briefly introduces the MMLDE method as a first step for EMLDE. Section 2.5 elaborates the EMLDE method. The experimental verifications are in Section 2.6. Section 2.7 concludes the chapter.

2.2 Review of Related Works and Challenges

2.2.1 RF Integrated Circuit Synthesis

Existing RF IC design automation methods focus on low-GHz synthesis [2-4,8-14] by employing lumped equivalent circuit models for passive components (e.g. transformer, inductor). The framework of most of these methods is shown in Figure 2.1. Compared with the low-frequency analog circuit sizing flow, a key part is the generation of the parasitic-aware model of the passive components. In RF IC designs at low-GHz frequencies, a simple lumped model is often extracted to mimic the behavior of the key passive components (transformer, inductor). Regression methods are then used to fit the (calibrated) EM simulation results (S-parameters) to the parasitic-included equivalent circuit models. The generated passive component models are accurate at low-GHz frequencies and computationally efficient.

To make the parasitic-aware model reliable in providing the correct performances for different design parameters, a strictly enforced layout template is often necessary. [10,11] use the parasitic corner, rather than a strict layout template, to improve the flexibility of the generated layout for circuits below 10GHz, yielding good results. In the development of the optimization kernel, evolutionary algorithms (EAs) are introduced in RF IC synthesis to achieve global search, getting very good results. [14] uses Particle Swarm Optimization (PSO) and [13] introduces the non-dominated genetic algorithm (NSGA) to RF IC synthesis in order to achieve multi-objective optimization.

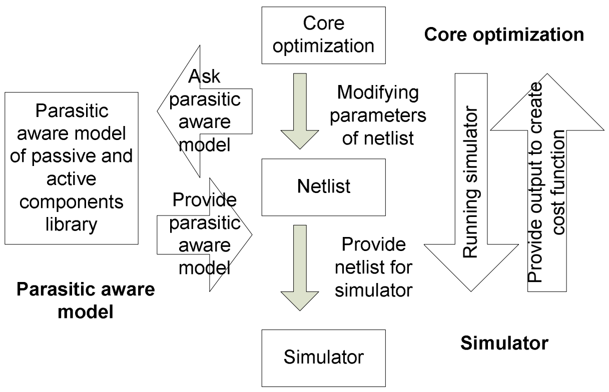

Core optimization
Core optimization
Ask parasitic aware model
Modifying parameters of netlist
Running simulator
Provide output to create cost function
Parasitic aware model of passive and active components library
Netlist
Provide parasitic aware model
Provide netlist for simulator
Parasitic aware model
Simulator
Simulator

Fig. 2.1 Framework of parasitic-aware optimization for RF ICs (from [4])

Parasitic-aware lumped equivalent circuit models for passive components that accurately match the EM simulation results are difficult to find at frequencies between say 40GHz and above 100GHz due to the distributed effects at these mm-wave frequencies [6]. Hence, when employing lumped equivalent circuit models, available RF integrated circuit design automation methods are limited to low-GHz instances. Because the speed enhancement method for RF IC synthesis (using lumped models) cannot be extended to mm-wave integrated circuit synthesis, and because directly including the EM simulations in each performance evaluation is too CPU time intensive, no good efficient method for mm-wave integrated circuit synthesis exists today. The only way left to mm-wave circuit designers is the "experience and simulation verification" method, which is at odds with today's high-performance and tightening time-to-market requirements.

To summarize, the goal of this chapter is to fill the blank of efficient automated design of mm-wave-frequency integrated passive components and integrated circuits (linear amplifiers as an instance), achieving good accuracy while knowing an acceptable CPU time.

2.3 Basic Computational Intelligence Techniques

The methods presented in this chapter are based on computational intelligence techniques, i.e. evolutionary computation and machine learning techniques in specific. In the following, we will introduce three basic techniques: the Differential Evolution (DE) algorithm, the Gaussian Process (GP) machine learning and the Naive Bayes Classifier (NBC), which are the fundamentals for the presented algorithms MMLDE and EMLDE.

2.3.1 Differential Evolution

The DE algorithm [15] is selected as the search engine in MMLDE and EMLDE. The DE algorithm outperforms many other evolutionary computation (EC) algorithms in terms of solution quality and convergence speed. DE uses a simple differential operator to create new candidate solutions and a one-to-one competition scheme to greedily select new candidates.

The i-th candidate solution in the d-dimensional search space at generation t can be represented as

$$x_i(t) = [x_{i,1}, x_{i,2}, \cdots, x_{i,d}] \tag{2.1}$$

At each generation t, the *mutation* and *crossover* operators are applied to the candidate solutions, and a new population arises. Then, *selection* takes place, and the corresponding candidate solutions from both populations compete to comprise the next generation. The operators are now explained in detail.

For each target candidate solution, according to the *mutation* operator, a *mutant vector* is built:

$$V_i(t+1) = [v_{i,1}(t+1), \ldots, v_{i,d}(t+1)] \tag{2.2}$$