Technische Universität München
Department of Electrical Engineering and Information Technology
Institute for Electronic Design Automation

Structure and Signal Path Analysis for Analog and Digital Circuits

Michael Eick

Vollständiger Abdruck der von der Fakultät für Elektro- und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzende:		Univ.–Prof. Dr. rer. nat. Doris Schmitt–Landsiedel
Prüfer der Dissertation:	1.	Priv.–Doz. Dr.–Ing. Helmut Gräb
	2.	Univ.–Prof. Dr.–Ing. habil. Lars Hedrich *Johann Wolfgang Goethe Universität*, *Frankfurt am Main*

Die Dissertation wurde am 17. Januar 2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektro- und Informationstechnik am 8. Mai 2013 angenommen.

This thesis was also published as book (ISBN 978–3–8349–1114–6) by Verlag Dr. Hut, Munich, Germany.

Abstract

This thesis presents a new method for the automatic structural analysis of analog and digital integrated circuits. The method provides automatic recognition of basic analog and digital building blocks, automatic identification of analog and digital signal paths as well as symmetry computation. Applications in the context of automatic analysis of digital standard cells as well as automatic sizing and placement of analog circuits are presented.

Kurzfassung

Diese Arbeit stellt eine neue Methode zur automatische Strukturanalyse von analogen und digitalen integrierten Schaltungen vor. Sie umfasst eine automatische Erkennung analoger und digitaler Grundblöcke, eine automatische Bestimmung analoger und digitaler Signalpfade sowie die Berechnung von Symmetrien. Die Arbeit stellt Anwendungen im Bereich der automatischen Analyse von digitalen Standardzellen sowie der automatischen Dimensionierung und Platzierung analoger Schaltungen vor.

Preface

This thesis summarizes my research work at the Institute for Electronic Design Automation. Therefore, I would like to thank the head of this institute, Prof. Dr.–Ing. Ulf Schlichtmann, for giving me this opportunity. A special thanks to PD Dr.–Ing. Helmut Graeb for being my advisor and all the fruitful discussions. These discussion were of great importance for quality and extend of this thesis and the related publications. I also would like to thank Prof. Dr.–Ing. Lars Hedrich for his work as second examiner and his attention to my work. Furthermore, I thank Prof. Dr. Doris Schmitt–Landsiedel for taking the chair of my committee and Prof. Dr.–Ing. Maurits Ortmanns for being my mentor in the TUM Graduate School.

Special thanks go to all my former colleagues at the institute and the members of the so–called analog group in particular. I will never forget the cooperativeness and great atmosphere. Especially, I would like to thank Dr.–Ing. Martin Strasser for his great support of my experiments on placement constraints. I would like to thank Gertraude Kallweit, Susanne Werner, Werner Tolle and Hans Ranke for keeping the institute running. I was able to find some great students who were willing to explore my ideas, whom I would like to thank at this point.

Furthermore, I would like to thank the people from the Infineon Technologies AG for their support, especially Jens Bargfrede and Bernhard Lippmann. I would like to commemorate Michael Mirbeth, who died in 2009. He gave me great support as well.

Thanks go also to the researchers from the Austin Research Laboratory of IBM for two very interesting weeks in Austin, particularly, Dr. Sani Nassif and Jente Benedict Kuang, PhD.

Without my parents' support I would never have gotten that far. Therefore, I would like to thank them. Finally, a special thanks to thank my friends for the great time outside university.

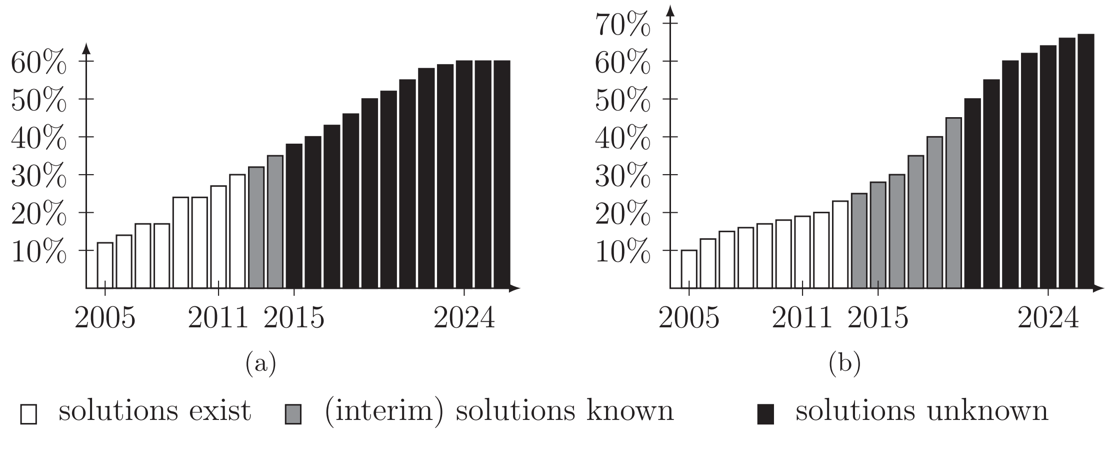
Munich, June 2013

Contents

1. Introduction

Analog and mixed-signal circuits play an important role in today's electronic systems. The *International Technology Roadmap for Semiconductors* lists, among others, consumer, communication and automotive applications (ITRS 2011, System Driver Chapter). It defines analog or mixed-signal circuits to be circuits *that at least partially deal with input signals whose precise values matter* (ITRS 2011, System Driver Chapter, p.16). These circuits are mainly required for analog to digital and digital to analog conversion to read out sensors or control actuators. It lists *shortage of design productivity* (ITRS 2011, System Driver Chapter, p.25) emerging from poor automation as one of the main challenges for future mixed-signal systems.

In the design chapter of the same work, more detailed goals are formulated. Figure 1.1a shows the degree of automation required for analog components in comparison to the degree of automation for digital circuits. Figure 1.1b illustrates the amount of circuits that should be synthesized. For the years starting from 2011 data is taken from ITRS (2011, design chapter, Tables DESN2a, DESN2b, DESN4). For years 2005 to 2011 data from ITRS (2005, design chapter, Tables 13a and 15a) is used. Both, ITRS (2005) and ITRS (2011) state that in 2011 analog automation reaches 27% of



(a)

(b)

Figure 1.1.: Requirements for analog design automation as formulated by the ITRS (ITRS 2005, ITRS 2011). (a) Degree of automation for analog designs in percent of automation for digital designs. (b) Percentage of synthesized analog content.

digital automation and 19% of synthesized analog content. Therefore, it is assumed that the intermediate goals from 2005 to 2011 were met.

Some amount of this increase might be due to the better integration of constraint–driven design and some basic design automation algorithms in major CAD tools. The idea is that analog and mixed–signal designs are heavily dependent on so–called constraints. Constraints describe all conditions that must be held for an analog circuit to work correctly but are normally not part of the specification. Examples for constraints are matching conditions, operating region conditions (e.g., saturation) or symmetry conditions. An example for a commercial software enabling constraint–driven design is Cadence® Virtuoso® 6 (Cadence Design Systems 2008). It contains a constraint manager to edit and store constraints in the design database. It provides basic methods to generate constraints using automatic recognition of building blocks like simple current mirrors and symmetry analysis. The constraints are then automatically considered in the layout editor. In order to be able to use design automation for sizing (Martens & Gielen 2008) or placement (Graeb 2011) without excessive setup, machine readable storage of constraints is necessary. In this context three problems can arise,

1. Traditionally, constraints were rarely documented. Maybe, important constraints were noted in some annotation layer.

2. Modern design software allows to store the constraints in machine–readable form with the design. However, entering all constraints for an existing or new design is tedious.

3. In the case that the constraints of a design were stored in machine–readable form, completeness becomes an issue. For sizing and placement tools, completeness of the constraints is important. Otherwise, they may find inappropriate solutions which an experienced designer would exclude from the beginning.

Automatic generation of constraints is an approach to tackle all these three problems.

1.1. Research Needs in Constraint-Driven Design

For the degree of analog automation, Fig. 1.1 shows a long term goal of 60%. For the amount of synthesized content the long term goal is 67%. However, today's technology, including research results, will limit the degree of automation to 35% which should be reached in 2014. The amount of synthesized content is limited to 45% which should be reached in 2019. Thus, further research in the area of analog design automation is necessary to reach the long term goals.

Further progress is expected trough interactive design aids (ITRS 2011, design chapter). Such an interactive design aid must fulfill two major requirements. First, it must be well integrated with the overall design environment. Second, the required initial set-up must be low to allow the designer a quick start and refine settings afterward. This is especially true for the constraints, which in turn requires an enhanced constraint-driven design.

Necessary enhancements in constraint–driven design are discussed in the following. According to Jerke et al. (2011) constraint–driven design requires a constraint management system, the possibility to generate or derive constraints from design objectives, the possibility to transform higher level constraints into lower level constraints, constraint verification and constraint sensitivity analysis. Constraint generation or constraint extraction is also identified as key issue by Rutenbar (2012). Enhancement of the state of the art is possible for most components:

- **Constraint Management/Representation**
  In current constraint management systems, constraints are often stored in a tool–specific way. This applies to the types of constraints provided as well as the parameters available for some specific constraint. This becomes a problem when various tools shall use the same set of constraints. In most systems, additional constraints can be defined by the user. However, a generic constraint set, applicable to different tools and circuits is not yet known.

- **Constraint Generation**
  Multiple approaches to automatic constraint generation were discussed in the literature (see Section 2.3). Most algorithms analyze the topology of the circuit for building blocks and/or symmetry. For example, the transistors of a simple current mirror always have to work in saturation region and must be matched. Symmetry is a widely used principle, where a circuit is designed to consist of two identical halves. A typical example are fully-differential circuits. For these circuits, high CMRR and low-offset error can only be obtained if the corresponding symmetry constraints for sizing and layout are completely available and are obeyed. Consequently, circuit symmetry should be computed from behavioral considerations. State–of–the–art algorithms focus on the topology of the circuit and the sizing and cannot handle multiple symmetrical signal paths that cross each other.

- **Constraint Transformation**
  Constraint transformation was considered little in research so far. Some work can be found for AC constraint transformation and in the field of hierarchical synthesis (see Section 2.3). The symmetry computation problem mentioned above can also be interpreted as constraint transformation of symmetry constraints on behavioral level to symmetry constraints on transistor level.

- **Constraint Verification**
  Two different problems occur in the context of constraint verification. On the one hand, the completed design must be checked for constraint violations. On the

> other hand, the constraints must be checked for completeness and contradictions. This problem is closely related to the constraint generation problem. Thus, methods developed for the constraint generation problem will push constraint verification forward as well.

This thesis will present new solution approaches to above problems. In particular, it investigates different approaches for constraint generation for analog circuits using structural analysis. It presents a new symmetry computation that is based on constraint transformation. In addition, a data structure for generic representation of constraints for different applications is presented.

1.2. Structural Analysis for Digital Circuits

The use of structural analysis methods is not limited to constraint generation. For digital circuits, structural analysis is a well established method (see Section 2.4). In this work, it will be shown that structural analysis for analog and digital circuits can be performed using the same methods. This is beneficial for circuits containing analog and digital components at the same time. Examples are analog circuits with some digital gates controlling power-down mode or mixed-signal circuits.

Timing characterization for digital standard cells determines the delay of a cell in dependence of the switching pattern and cell load. To perform this automatically, several input data like the logic function of the cell is required. Information about the internal structure of a cell becomes important for advanced techniques. For example the current–source modeling approach of Knoth et al. (2009) as well as the aging analysis approach of Lorenz et al. (2010) require a decomposition of the standard cell into single stage gates. This thesis will show that the suggested method can generate this information.

1.3. Contributions of this Work

The structural analysis method presented in this thesis is suitable but not limited to the applications discussed above. In particular, these are constraint generation for analog circuits and generation of structural and behavioral information for timing characterization of digital standard cells.

An overview of the proposed method is given in Fig. 1.2. Starting point is the circuit topology which is typically a netlist as well as a description of the inputs and outputs of the circuit. Based on the circuit topology a building block analysis is performed, which recognizes analog building blocks like simple current mirrors or digital building blocks like pass gates. After that, a structural signal path analysis is performed which is based on a structural and qualitative behavioral model of the circuit, the

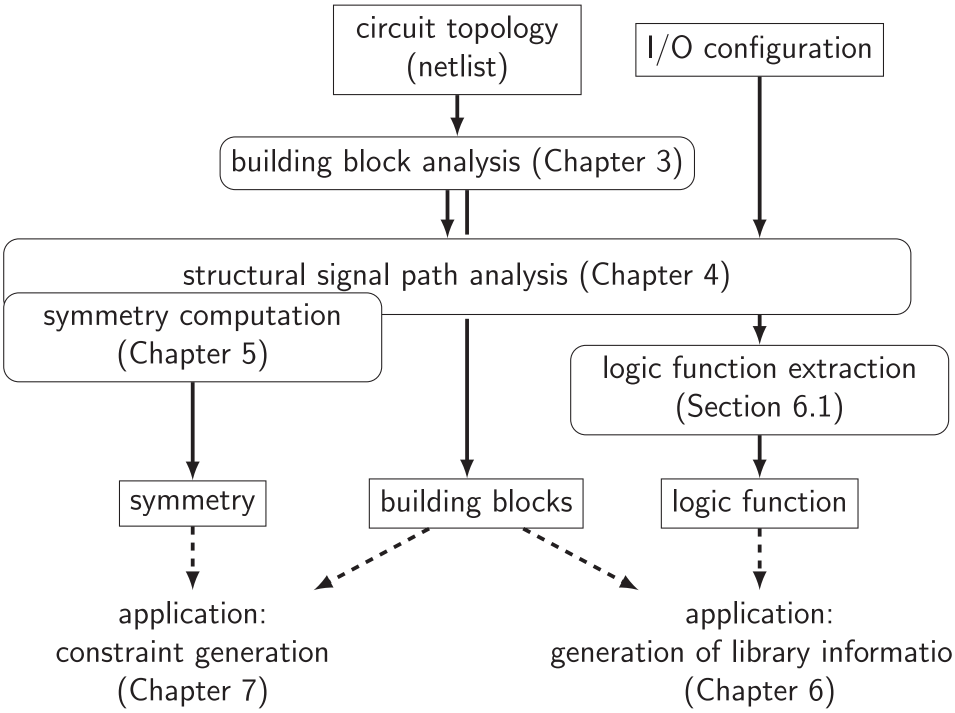

circuit topology (netlist)
I/O configuration
building block analysis (Chapter 3)
structural signal path analysis (Chapter 4)
symmetry computation (Chapter 5)
logic function extraction (Section 6.1)
symmetry
building blocks
logic function
application: constraint generation (Chapter 7)
application: generation of library information (Chapter 6)

Figure 1.2.: Overview of the proposed method.

so–called ESFG. The analysis includes an automatic identification of the analog and digital part of a circuit as well as the core and bias part, a computation of true pass gate directions, automatic breaking of feedback loops and a symmetry computation. For digital circuits, the results of the structural signal path analysis are then used to compute the overall logic function of the circuit.

The method includes the following contributions compared to state of the art (see Chapter 2).

1. **Building Block Analysis**
   Traditional approaches for building block analysis either focus on analog or digital circuits. This is the first method that can handle analog and digital building blocks simultaneously. In addition, it contains several runtime improvements compared to the basic algorithm (Massier 2010).

2. **Structural Signal Path Analysis**
   The structural signal path analysis is based on a new graph model, the ESFG. It is the first model, that includes the structure of the circuit described by its building blocks as well as its qualitative behavior. This thesis presents the following analyses based on that model:

   a) a method to automatically identify the core and bias part of a circuit,

   b) a method to automatically identify the analog and digital part of a circuit,

   c) a method to detect the true directions of pass gates within the digital circuit part,

   d) a method to break feedback loops within the digital circuit part.

3. **Symmetry Computation**
   In addition, a new automatic symmetry computation method is presented. This method shows a significant improvement of the symmetry computation for analog circuits, because

   - it provides an improved handling of asymmetries, and

   - it is the only method that can handle multiple, overlapping differential signal paths correctly.

4. **A Consistent Constraint Generation Method**
   Existing constraint generation approaches were revised and generalized. The resulting method can provide consistent constraints for the complete analog design flow with a focus on sizing and placement.

Parts of this work have been published in Eick et al. (2009), Eick, Lu & Graeb (2010), Eick, Strasser, Graeb & Schlichtmann (2010), Strasser et al. (2011), Eick et al. (2011), Eick & Graeb (2011a), Eick & Graeb (2011b), Eick & Graeb (2012a), Eick & Graeb (2012b), Eick & Graeb (2012c), Eick & Graeb (2013), Eick, Sridharan & Graeb (2013) and Eick, Strasser & Lu (2013). Some subproblems and provisional results were investigated in the following student projects: Lu (2009), Stolberg-Stolberg (2009), Stolberg-Stolberg (2010), Tag (2010), Tschöpe (2010), Tsonev (2010), Youssef (2010), Zhang (2011), Guo (2012), Jongudomkarn (2012) and Sridharan (2013).

The remainder of this thesis is organized as follows. The state of the art is discussed in Chapter 2. Chapter 3 discusses the enhanced building block analysis, Chapter 4 introduces the structural signal path analysis and Chapter 5 describes the new symmetry computation method. After that, two different applications are presented. Chapter 6 describes the application to automatic generation of input data of digital standard cells. Chapter 7 discusses the application to automatic constraint generation for analog circuits. The thesis is concluded by Chapter 8.

2. State of the Art

This chapter discusses the state of the art in fields related to the topic of the thesis. First, existing structural and behavioral symbolic analysis methods for analog circuits are discussed. Next, sizing and placement constraint generation methods for analog circuits are covered. After that, structural and functional analysis methods for digital circuits are discussed. Some of these methods are capable of determining the logic function out of the circuit structure.

2.1. Structural Analysis for Analog Circuits

Two structural analyses types with relevance for this thesis can be identified. Building block analysis computes basic building blocks like simple current mirrors. The symmetries within a circuit are computed by a symmetry analysis.

2.1.1. Building Block Analysis

Building block analysis is the problem of finding subcircuits with a fixed topology within a circuit. From a mathematical point of view, this is the problem of finding subgraph isomorphism. Hence, the problem is related to comparing two complete circuits using isomorphism methods. Such methods are applied in layout–versus–schematic (LVS) checks (Barke 1984). The relatively small number and size of analog building blocks allows to develop more specialized methods, which are discussed in the following.

Chen & Sheu (1992) describe an algorithm to recognize differential pairs, simple current mirrors, common-gate devices and cascode current mirrors by identifying four types of special circuit nodes.

Arsintescu (1996) suggests a weighted bipartite matching algorithm to identify simple current mirrors and level shifters (called biasing groups).

Mahmoud (1998) describes a method using pattern rules to recognize certain template circuits. The extent of these templates is unclear. The example shown covers simple current mirrors and a differential pair.

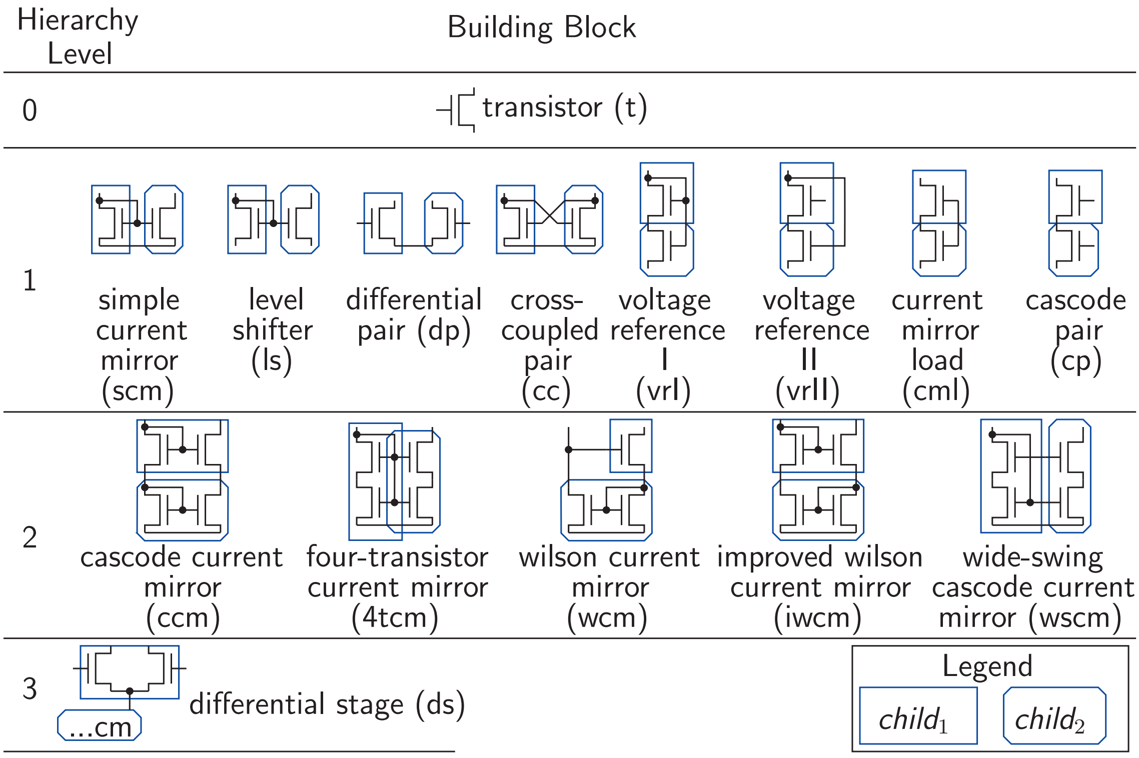

Hierarchy Level
Building Block
0
transistor (t)
1
simple current mirror (scm)
level shifter (ls)
differential pair (dp)
cross-coupled pair (cc)
voltage reference I (vrI)
voltage reference II (vrII)
current mirror load (cml)
cascode pair (cp)
2
cascode current mirror (ccm)
four-transistor current mirror (4tcm)
wilson current mirror (wcm)
improved wilson current mirror (iwcm)
wide-swing cascode current mirror (wscm)
3
...cm
differential stage (ds)
Legend
child$_1$
child$_2$

Figure 2.1.: Library of CMOS building blocks (Massier 2010, Fig. 3.2).

The Sizing Rules Method

The Sizing Rules Method described by Massier (2010)[1] includes a recognition algorithm that compares a hierarchical building block library to a given netlist. It is the most complete approach with respect to the covered types of building blocks and it is the only approach that can handle CMOS and bipolar circuits. Since this algorithm is the basis for the building block analysis presented in Chapter 3, a more detailed description of this approach is given in the following.

Figure 2.1 shows the CMOS part of the building block library. It consists of four hierarchy levels 0 to 3. Hierarchy level 0 contains a single *MOSFET transistor*. Hierarchy level 1 contains the pair building blocks *simple current mirror*, *level shifter*, *differential pair*, *cross–coupled pair*, *voltage reference I*, *voltage reference II*, *current mirror load* and *cascode pair*. The transistors forming the pair are called children of the building block. They are numbered to allow referencing. In Fig. 2.1 the child with index 1 is marked by a blue rectangle and the child with index 2 is marked by a blue rectangle with beveled corners. Hierarchy level 2 contains building blocks that are built using blocks from hierarchy levels 0 and 1: The *cascode current mirror* consists of a *simple*

[1] preliminary works in (Graeb et al. 2001, Zizala 2001, Massier 2002, Massier et al. 2003, Eick 2006, Massier et al. 2008a, Massier et al. 2008b, Massier & Graeb 2008)

```
$\mathcal{B} \leftarrow \emptyset$
for $t \leftarrow \texttt{scm}, \texttt{ls}, \texttt{dp}, \ldots, \texttt{ccm}, \texttt{4tcm}, \ldots, \texttt{ds}$
    $X \leftarrow \{(c_1, c_2) \in (\mathcal{B} \cup \mathcal{D})^2 | \mathsf{leftTypeMatches}(t, c_1) \wedge \mathsf{rightTypeMatches}(t, c_2)\}$
    for $(c_1, c_2) \in \{(c_1, c_2) \in X | \mathsf{patternMatches}(t, c_1, c_2)\}$
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{\mathsf{newBuildingBlock}(t, (c_1, c_2))\}$
$\mathcal{B} \leftarrow \mathsf{resolveConflicts}(\mathcal{B})$
$\mathcal{B} \leftarrow \mathsf{removeUncertainBuildingBlocks}(\mathcal{B})$
```

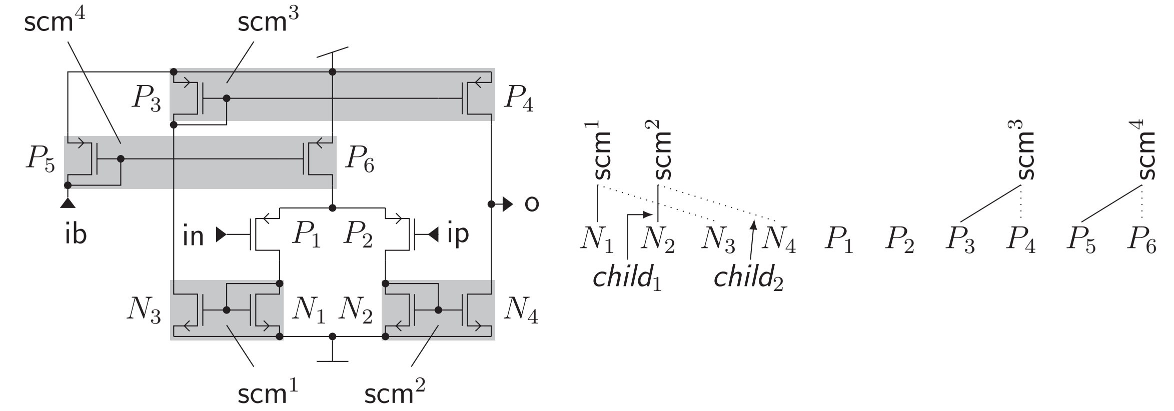
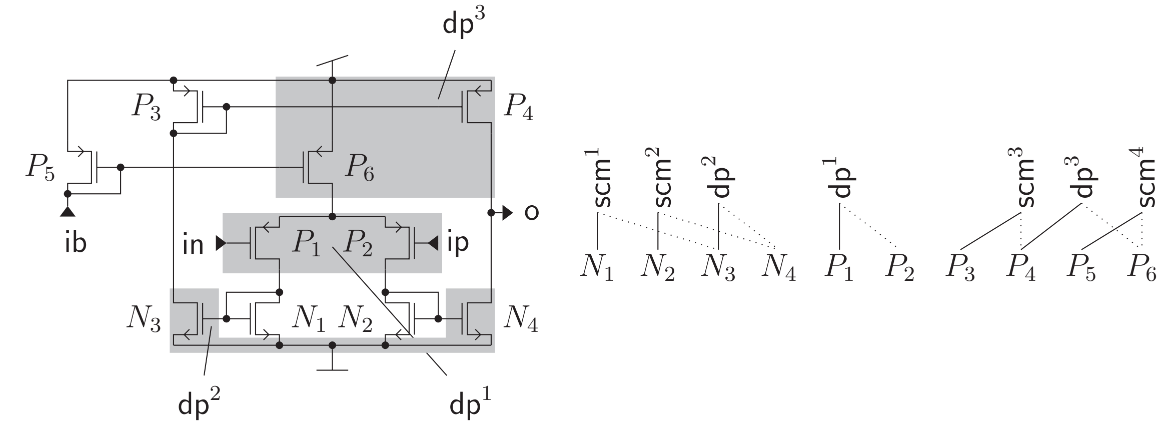
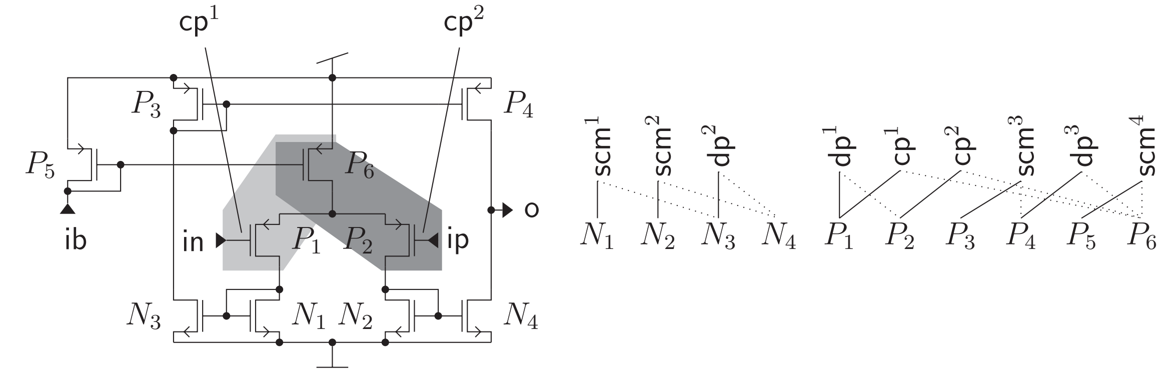
Figure 2.2.: Building block analysis algorithm according to Massier (2010).

current mirror and a *level shifter*. The *four–transistor current mirror* is formed by a *voltage reference I* and a *current mirror load*. The *wilson current mirror* consists of a single transistor and a *simple current mirror*. The *improved wilson current mirror* is formed by a *simple current mirror* and a *level shifter*. The *wide–swing cascode current mirror* of a *voltage reference II* and a *cascode pair*. Hierarchy level 3 contains the *differential stage* which is formed by a *differential pair* and some current mirror.

An outline of the analysis algorithm is given in Fig. 2.2. Set $\mathcal{B}$ includes all recognized building blocks and set $\mathcal{D}$ denotes the devices (e.g., transistors) of the circuit. The algorithm iterates over all defined building blocks of hierarchy level 1, 2 and 3. For each iteration, pairs $(c_1, c_2) \in X$ with matching types are determined first. For example, for the recognition of a *simple current mirror*, pairs of transistors are formed. For the recognition of a *cascode current mirror*, pairs (c_1, c_2) are formed, where c_1 is a *simple current mirror* and c_2 is a *level shifter*. Next, the connection patterns of all pairs in X are compared with the connection pattern of the current building block. If the pattern matches, a new building block is created.

Table 2.1 illustrates the course of the algorithm for the symmetrical OTA from Sansen (2007, Silde 0711). In the iteration for t = scm, four *simple current mirrors* scm^1, scm^2, scm^3 and scm^4 are recognized. They consist of transistors N_1 and N_3, N_2 and N_4, P_3 and P_4 as well as P_5 and P_6, respectively. The search for *differential pairs* (t = dp) yields *differential pair* dp^1 consisting of P_1 and P_2, *differential pair* dp^2 consisting of N_3 and N_4 as well as *differential pair* dp^3 consisting of P_4 and P_6. The latter two are false recognitions that will be removed in later steps. Iteration t = cp finds two (false) *cascode pairs* cp^1 and cp^2 consisting of P_1 and P_6 as well as P_2 and P_6. Finally, one *differential stage* ds^1 consisting of dp^1 and scm^4 is found in iteration t = ds.

In the example, transistor N_3 is part of scm^1 and dp^2. The question arises, which of these building blocks is valid and which is wrong. Two building blocks are said to be in conflict if they overlap at least at one device. Formally this can be written as follows. The set of all devices $D_\star(x) \subseteq \mathcal{D}$ of a building block contains all devices that form the

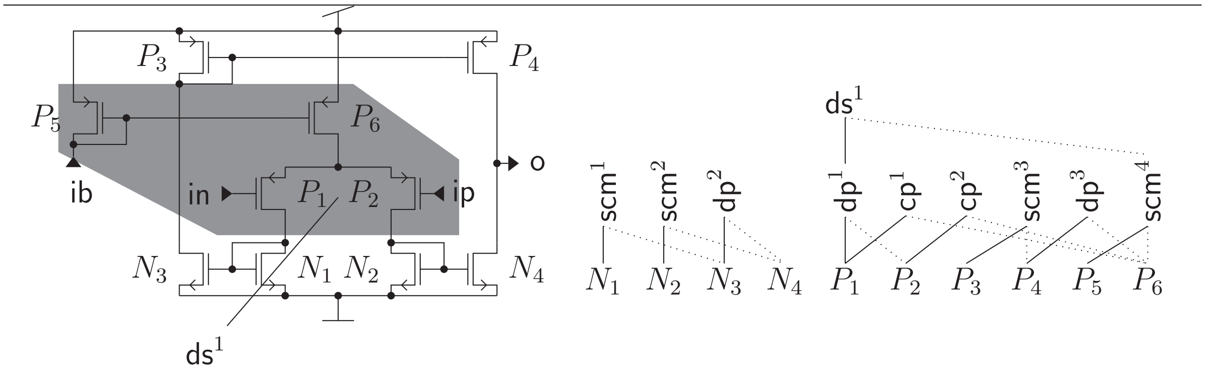
Iteration t=scm
 scm^4 scm^3 P_3 P_4 P_5 P_6 ib in P_1 P_2 ip o N_3 N_1 N_2 N_4 scm^1 scm^2 scm^1 scm^2 scm^3 scm^4 N_1 N_2 N_3 N_4 P_1 P_2 P_3 P_4 P_5 P_6 child$_1$ child$_2$
Iteration t=dp
 dp^3 P_3 P_4 P_5 P_6 ib in P_1 P_2 ip o N_3 N_1 N_2 N_4 dp^2 dp^1 scm^1 scm^2 dp^2 dp^1 scm^3 dp^3 scm^4 N_1 N_2 N_3 N_4 P_1 P_2 P_3 P_4 P_5 P_6
Iteration t=cp
 cp^1 cp^2 P_3 P_4 P_5 P_6 ib in P_1 P_2 ip o N_3 N_1 N_2 N_4 scm^1 scm^2 dp^2 dp^1 cp^1 cp^2 scm^3 dp^3 scm^4 N_1 N_2 N_3 N_4 P_1 P_2 P_3 P_4 P_5 P_6

Continued on next page . . .

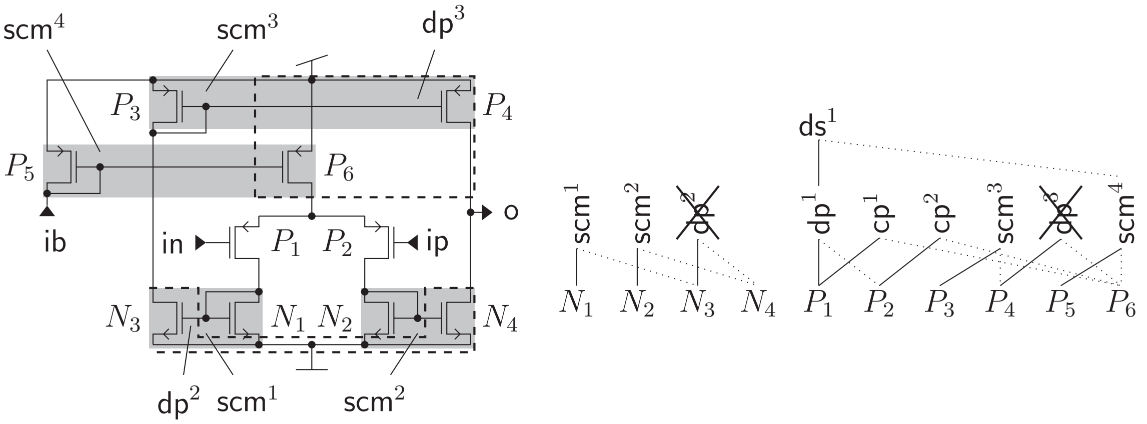
Table 2.1.: Flow of the recognition algorithm for a symmetrical OTA (Sansen 2007, Slide 0711).

Continued from previous page ...

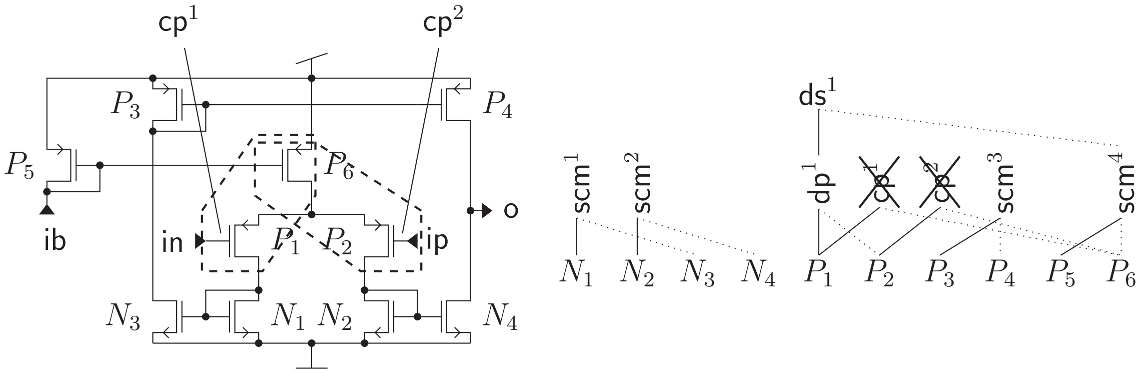
Iteration t=ds



conflict resolution



removal of uncertain building blocks



Continued on next page ...

Table 2.1.: Flow of the recognition algorithm for a symmetrical OTA (Sansen 2007, Slide 0711).

Continued from previous page ...

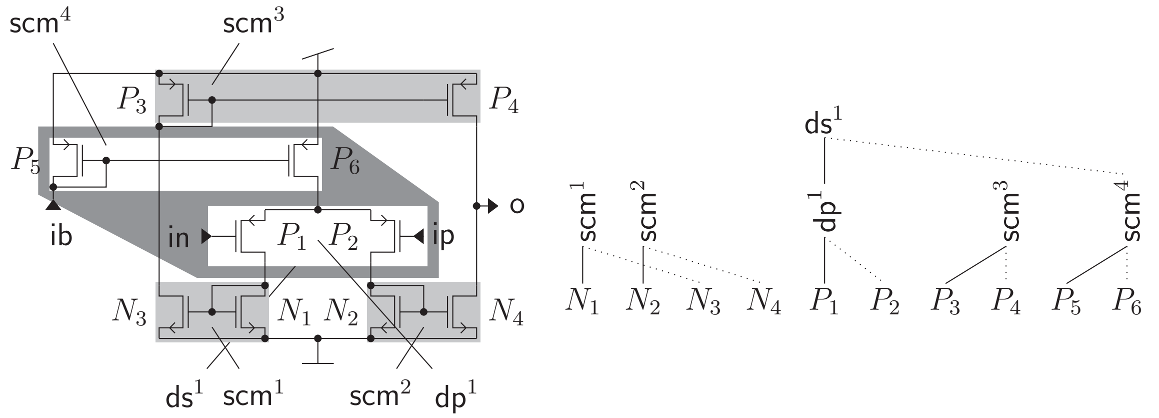
final result


Table 2.1.: Flow of the recognition algorithm for a symmetrical OTA (Sansen 2007, Slide 0711).

block. The devices can either be direct children, grandchildren, great–grandchildren, etc. Set $D_\star(x)$ is recursively defined as follows,

$$D_\star(x) = \begin{cases} \bigcup_{y \in x.children} D_\star(y) & x \in \mathcal{B} \\ \{x\} & x \in \mathcal{D} \end{cases} , \tag{2.1}$$

where $x.children \subseteq \mathcal{D} \cup \mathcal{B}$ denotes the set of all children of x. In some cases, it is necessary to distinguish the devices that belong to the first child $x.child_1$ or the second child $x.child_2$ of a pair. For these cases, sets $D_1(x)$ and $D_2(x)$ are defined as follows,

$$D_1(x) := D_\star(x.child_1) \qquad D_2(x) := D_\star(x.child_2) \ . \tag{2.2}$$

Index $\star$ can also be interpreted as 1 or 2 because a pair has exactly two children and

$$D_\star(x) = D_1(x) \cup D_2(x) \ . \tag{2.3}$$

In the example, sets $D_1(\mathrm{scm}^1)$, $D_2(\mathrm{scm}^1)$ and $D_\star(\mathrm{scm}^1)$ of *simple current mirror* scm^1 are,

$$D_1(\mathrm{scm}^1) = \{N_1\} \qquad D_2(\mathrm{scm}^1) = \{N_3\} \qquad D_\star(\mathrm{scm}^1) = \{N_1, N_3\} \ . \tag{2.4}$$

For *differential pair* dp^2, sets $D_1(\mathrm{dp}^2)$, $D_2(\mathrm{dp}^2)$ and $D_\star(\mathrm{dp}^2)$ are,

$$D_1(\mathrm{dp}^2) = \{N_3\} \qquad D_2(\mathrm{dp}^2) = \{N_4\} \qquad D_\star(\mathrm{dp}^2) = \{N_3, N_4\} \ . \tag{2.5}$$

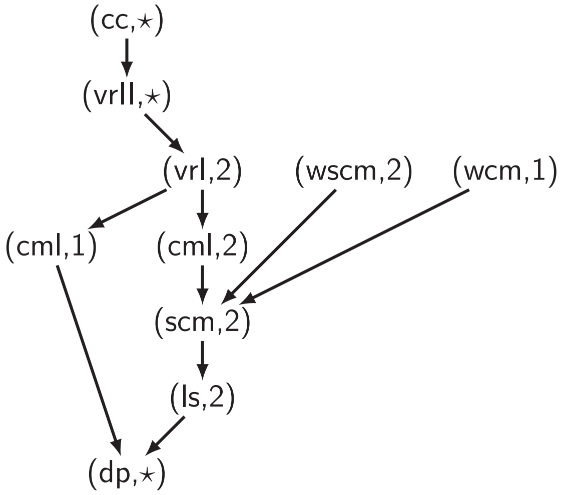


Figure 2.3.: Dominance graph G_D for the library shown in Fig. 2.1 (Massier 2010, Fig. 4.7).

The assignment of N_3 and N_4 to $D_1(\mathrm{dp}^2)$ and $D_2(\mathrm{dp}^2)$ is arbitrary because the *differential pair* is symmetric. However, set $D_\star(\mathrm{dp}^2)$ is not influenced.

According to Massier (2010), a conflict is resolved by determining a dominant building block that is kept and a dominated building block that is removed using the dominance graph shown in Fig. 2.3. Precisely,

$$\begin{aligned}&x_2 \text{ dominates } x_1 :\Leftrightarrow \\ &\exists_{(i,j)\in\{1,2,\star\}^2} \big(D_i(x_1) \cap D_j(x_2) \neq \emptyset\big) \wedge \big((x_1.\mathit{type}, i) \text{ reachable from } (x_2.\mathit{type}, j)\big)\ , \qquad (2.6)\end{aligned}$$

where $x_1.\mathit{type}$ and $x_2.\mathit{type}$ denote the type of building blocks x_1 and x_2, respectively. Cut–set $D_i(x_1) \cap D_j(x_2)$ is non-empty if x_1 and x_2 share one device. If $i = 1$ and $j = 2$ this device is part of the first child of x_1 and the second child of x_2. If $i = \star$ and $j = 2$ this device is part of the first or second child of x_1 and the second child of x_2. Relation *reachable from* is true for μ and ν iff there is a path in G_D from ν to μ. A recognition result described through $\mathcal{B}$ is said to be conflict free, if there is no building block $x_2 \in \mathcal{B}$ that dominates some building block $x_1 \in \mathcal{B}$,

$$\underset{(x_1,x_2)\in\mathcal{B}^2}{\forall} \overline{x_2 \text{ dominates } x_1}\ . \qquad (2.7)$$

The notation used here differs from the notation used by Massier (2010), but Eq. (2.7) can be shown to be equivalent to Massier (2010, 4.11) (see Appendix B).

In the example, the conflict between scm^1 and dp^2 can now be resolved using Eq. (2.6).

For $x_1 = \mathrm{dp}^2$, $x_2 = \mathrm{scm}^1$, $i = \star$ and $j = 2$, the following holds,

$$\Big(D_\star(\mathrm{dp}^2) \cap D_2(\mathrm{scm}^1) = \{N_3\} \neq \emptyset\Big) \wedge \Big((\mathrm{dp}, \star) \text{ reachable from } (\mathrm{scm}, 2)\Big) \\ \Leftrightarrow \mathrm{scm}^1 \text{ dominates } \mathrm{dp}^2\ . \quad (2.8)$$

The conflict is resolved by removing dp^2. In the example, there is a second conflict between scm^3, scm^4 and dp^3. The conflict is resolved by removing dp^3 (see Table 2.1).

The building blocks *differential pair* and *cascode pair* are connected at one point only, e.g., the two transistors of a *differential pair* are only connected at their source pins. These building blocks are considered as uncertain and are removed if they are not part of a larger building block. In the example, *differential pair* dp^1 is kept because it is part of *differential stage* ds^1. In contrast, *cascode pairs* cp^1 and cp^2 are removed because they are not part of a larger building block (Table 2.1). The final recognition result for the example is shown at the bottom of Table 2.1.

2.1.2. Symmetry Analysis

Symmetry analysis or symmetry computation is the problem of identifying two parts in the netlist of a circuit that should work equally or exactly opposed. The following describes how different authors tackled that problem for different applications in the context of analog circuits.

Charbon et al. (1993) present a symmetry computation to generate matching constraints for layout. It works on an undirected graph where nodes are devices and edges are nets. They propagate pairs of symmetric nodes trough the circuit while considering the results of a sensitivity analysis. The propagation is stopped when a real or virtual ground is reached. According to the experimental results, they can handle one fully-differential or single-ended signal path. This signal path is subdivided into multiple symmetry axes. The algorithm needs a sized netlist to compute the sensitivities.

Kole et al. (1994) describe a symmetry analysis with applications in symbolic analysis and behavioral modeling. They use a hypergraph where nets are nodes and devices are edges. Starting from each node and edge of the circuit their algorithm tries to create two isomorphic trees. The algorithm is limited to fully symmetric circuits. They suggest to handle so-called near symmetries by transforming the circuit to a fully symmetric one. However, it is not explained how this can be automated.

Arsintescu & Spanoche (1996) and Arsintescu (1996) suggest a symmetry computation method for placement constraint generation and visualizing netlists of analog circuits. Their data structure is a bipartite graph with the nets and transistor pins as nodes. Starting from every net in the circuit, they try to propagate pairs of symmetric nodes trough this graph. Their algorithm includes the sizing, i.e., two transistors are only detected as symmetric if they have the same sizing. They claim that the algorithm can handle circuits that are not fully symmetric. However, they do not describe how the algorithm achieves this.

Yi et al. (2003), Hao et al. (2004) and Zhou et al. (2005) present a symmetry computation with application in placement constraint generation. They use a bipartite graph and a labeling algorithm similar to the Gemini II algorithm (Ebeling 1988). Asymmetries in the circuit are handled by ignoring the gate connections and an initial transformation of the netlist. However, it is unclear if this can be automated in a reliable way.

Bhattacharya et al. (2004) describes an algorithm to extract symmetry constraints from a given layout for retargeting of analog circuits.

2.1.3. More Structural Analysis Methods

Other structural analysis methods were suggested in the context of structural synthesis. The purpose of these methods is to detect unrealistic circuit configurations and evaluate the results. Examples can be found in Sripramong & Toumazou (2002), Ferent & Doboli (2010) and Meissner et al. (2012).

2.2. Behavioral Analysis

Behavioral analysis methods can be divided into qualitative and quantitative methods. Qualitative methods will give information, e.g., on which path an input signal propagates trough the circuit but will not include information about the exact magnitude of the signal at the output. Quantitative methods will give information about signal magnitudes.

One of the earliest works on qualitative analysis was published in the field of artificial intelligence (de Kleer 1984). It uses causal analysis to generate a qualitative model of a circuit starting from qualitative models of the devices as well as Kirchhoff's Laws. The result is represented as so-called mechanism graph, which represents a sequence of deductions leading to a certain input/output behavior. An algorithm to compute possible current flows trough a circuit is investigated by Hao et al. (2004) and Zhou et al. (2005). They call this signal flow analysis and use it to identity the signal processing part (core part) of the circuit as well as to generate layout constraints. Another so-called signal flow analysis is patented by Zhang et al. (2008). They model possible signal flows on transistor level using a graph. They claim, that this can be used to partition the circuit into a digital part, an analog biasing part and an analog core part. However, they do not present their exact method.

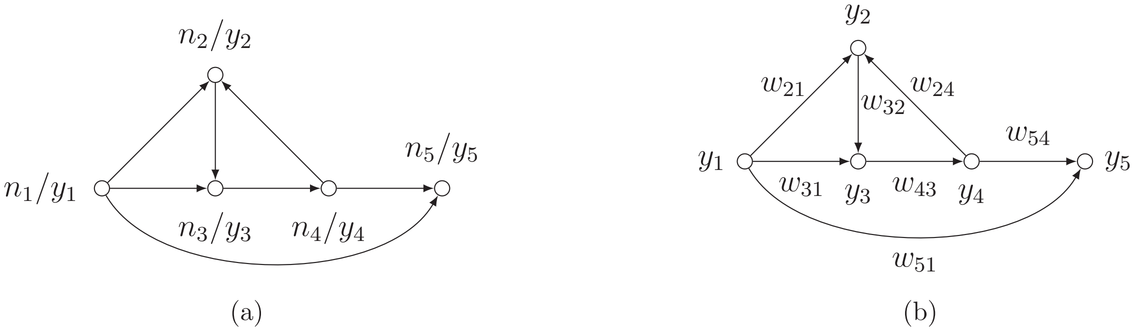


Figure 2.4.: Examples of signal flow graphs. (a) General signal flow graph. (b) Linear signal flow graph.

Quantitative methods can be further subdivided into numerical and symbolic methods. Numerical methods like SPICE simulation are applicable to a wide range of circuits but they do not give insight how a circuit works. Symbolic methods try to find analytical equations to describe the behavior of the circuit (Gielen & Sansen 1991). These models are typically linear and limited to small-signal behavior. A small and simple model may give good insight but have bad accuracy, because important parameters were neglected. A large model may have good accuracy but provide less insight, because it is too complex.

Signal Flow Graphs

Some symbolic methods use so–called signal flow graphs, which are related to data structures used in this thesis. Signal flow graphs have wide applications in all fields of electrical engineering, e.g., control theory and signal processing. They were first introduced by Mason (1953). The work describes general properties of signal flow graphs as well as linear signal flow graphs. The idea is to represent possible ways of signal propagation between variables represented as nodes of a directed graph. An example is given in Fig. 2.4a. If an edge e connects nodes n_i and n_j it means that a signal can propagate from n_i to n_j. Thus, in the example a signal can propagate from node n_1 to node n_2 but not from n_4 to n_1. Every node n_i processes the input signals to generate one output signal y_i using a function f_i. Thus, the example can be written as,

$$y_2 = f_2(y_1, y_4) \qquad y_3 = f_3(y_1, y_2) \qquad y_4 = f_4(y_3) \qquad y_5 = f_5(y_1, y_4)\ . \tag{2.9}$$

No expressions for the functions are considered so far. For linear signal flow graphs the node functions f_i are linear. For the example this could be,

$$y_2 = w_{21}\, y_1 + w_{24}\, y_4 \quad y_3 = w_{31}\, y_1 + w_{32}\, y_2 \quad y_4 = w_{43}\, y_3 \quad y_5 = w_{51}\, y_1 + w_{54}\, y_4\ . \tag{2.10}$$

For convenience the edges of the graph can be labeled with the weights of the corresponding inputs and the nodes can be labeled with the variables (Fig. 2.4b). The total transfer function or gain from node n_i to another node n_j can then be computed using simplification steps (Mason 1953) or Mason's Rule (Mason 1956). The latter allows to compute the transfer function by analyzing loops and forward paths. Forward paths connect input to output and do not contain any node twice. If there is only one feedback loop with loop gain T, Mason's Rule can be written as,

$$\frac{\sum_i G_i + \sum_i H_i\,(1-T)}{(1-T)}\ , \tag{2.11}$$

where G_i is the gain of the i-th forward path touching the loop and H_i is the gain of the i-th forward path not touching the loop. A more general formulation for multiple loops can be found in Mason (1956). Using this, the gain from node y_1 to node y_5 can be calculated for the example. The graph has one loop consisting of nodes y_2, y_3 and y_4 with loop gain $T_1 = w_{32}\, w_{43}\, w_{24}$. The graph has three forward paths. Paths y_1, y_2, y_3, y_4, y_5 and y_1, y_3, y_4, y_5 touch the loop. Their gains are $G_1 = w_{21}\, w_{32}\, w_{43}\, w_{54}$ and $G_2 = w_{31}\, w_{43}\, w_{54}$, respectively. Path y_1, y_5 does not touch the loop. Its gain is $H_1 = w_{51}$. In this case Mason's Rule evaluates to

$$\begin{aligned}\frac{y_5}{y_1} &= \frac{G_1 + G_2 + H_1(1-T)}{1-T} \\ &= \frac{w_{31}\, w_{43}\, w_{54} + w_{21}\, w_{32}\, w_{43}\, w_{54} + w_{51}(1 - w_{32}\, w_{43}\, w_{24})}{1 - w_{32}\, w_{43}\, w_{24}}\ .\end{aligned} \tag{2.12}$$

Linear signal flow graphs are used by many symbolic analysis approaches for analog circuits (Gielen & Sansen 1991). Since the nodes of the signal flow graph are the variables of the problem, structural information is lost. This is different for the graph proposed by Wei & Doboli (2008). The nodes of this so–called coupled building–block behavioral (CBBB) model are the nets of the circuit. The edges and the corresponding linear expressions are generated based on the building blocks of the circuit. However, structural information is not considered during subsequent analysis. They present also a variant called uncoupled building–block behavioral (UBBB) model, where feedbacks are removed and a method to handle weakly non–linear circuits.

2.3. Constraint Generation for Analog Circuits

Constraint generation and constraint management has been discussed in several publications.

Approach	Analyses				Generated Constraints	
	building blocks	symmetry	sensitivities	current flows	sizing	layout
Chen & Sheu (1992)	×		×		—	matching net length net spacing
Charbon et al. (1993) Choudhury & Sangiovanni-Vincentelli (1993) Malavasi et al. (1996)		×	×		—	matching parasitics layout symmetry
Arsintescu & Spanoche (1996)	×	×			—	matching layout symmetry
Graeb et al. (2001) Massier et al. (2008b) Massier (2010)	×				operating region matching	—
Yi et al. (2003) Hao et al. (2004) Zhou et al. (2005)	×	×		×	—	matching layout symmetry core/bias identification

Table 2.2.: Overview of constraint generation approaches for analog circuits

General concepts for so-called constraint–driven design were discussed in the introduction. Krinke & Lienig (2011) present a formal approach to constraint definition using so–called ontologies. Malavasi & Kao (1997) discuss different challenges for constraint–driven physical design, including noise, delay, parasitics, symmetries, matching and yield. Malavasi et al. (1998) present a general constraint management system.

Some approaches to automatic constraint generation were published together with the structural analysis methods discussed in Section 2.1. They are summarized in Table 2.2. It can be observed that various approaches exist to generate layout constraints. They utilize different combinations of building block, symmetry, sensitivity and current flow analyses. The constraints generated cover matching, net length, net spacing, bounds for parasitics, layout symmetry and special restrictions for the core part. Apart from that, there is one approach (Massier 2010) to generate operating region and matching constraints for sizing by analyzing building blocks.

A method to identify critical and less critical constraints during constraint transformation was presented by Arsintescu & Otten (1998). Arsintescu et al. (1998) present a method to transform AC constraints between different abstraction levels. Makris & Toumazou (1995) present a method based on qualitative reasoning to correct a circuit in case the specification is violated. Some approaches to transform constraints from one level of a design hierarchy to another were investigated in the context of analog circuit synthesis. BLADES (El-Turky & Perry 1989) and OASYS (Harjani et al. 1989) suggest knowledge–based approaches.

2.4. Structural Analysis for Digital Circuits

Structural analysis of digital circuits was broadly covered by research in the past. The following discusses recognition of digital building blocks (e.g., inverters), computation of true signal flow of switches and regularity extraction.

For the recognition of digital building blocks two classes of approaches exist: library based approaches and algorithmic approaches.

Library based approaches typically model circuit and building blocks in a library using graphs. Subgraph isomorphism algorithms are applied to compare the building blocks from the library to the circuit (Watanabe et al. 1983, Huang & Overhauser 1995, Rubanov 2003, Rubanov 2006, Zhang & Wunsch II 2006). These methods are independent of a specific design style like CMOS. However, their recognition ability is limited to building blocks contained in the library.

Algorithmic approaches use the design rules of a specific design style like CMOS logic to interpret a given netlist. Some approaches for CMOS logic investigate series and parallel connection in the pull-up and pull-down network (Takashima et al. 1982, Boehner 1988, Yokomizo et al. 1990, Dagenais 1991, Kim & Shin 1998). Bryant (1991) extends this concept using four valued logic considering high impedance and unknown states. Other approaches investigate paths of serial transistors ending at the power rails (Laurentin et al. 1992, Hübner et al. 1997). All approaches compute the logic function of the gate. These approaches can cover one design style (typically CMOS logic) completely, but cannot be used for other design styles.

A hybrid approach is presented by Yang & Shi (2003). They speed-up a subgraph isomorphism approach by preprocessing the netlist using an algorithmic approach to identify simple gates.

In case sequential circuits are investigated it must be considered that states are saved internally. The variables in the resulting logic function can refer to different time steps, leading to so–called temporal logic (Ari et al. 1983). Pandey et al. (1995) and Jain et al. (1995) show how state machines can be extracted using symbolic simulation.

The problem of determining the correct signal flow within switches, e.g., pass gates or pass transistors, occurs for example in switch–level simulation and timing verification.

The approach of Jouppi (1983) relies on a set of local rules. Ousterhout (1985) suggest to search paths between the switch and a strong signal source. Other approaches represent the circuit as graph and classify the nodes according to different criteria (Blaauw et al. 1990, Lee et al. 1990, Baba-ali & Farah 1996). Next, switch directions are determined using local rules as well as reachability analysis on the graph.

Many EDA algorithms for digital circuits can be speeded up by exploiting regularities and symmetries. Regularity extraction tries to find repetition of the same subcircuit within one circuit. The extent of this subcircuit is not known in the beginning. Regularity extraction algorithms work in two steps: First, subcircuit candidates are identified using heuristics. Second, repetitions of these subcircuits within the circuit are searched using the subgraph isomorphism techniques described above (Rao & Kurdahi 1993, Hassoun & McCreary 1999). Symmetry extraction was investigated based on general automorphism approaches (Aloul et al. 2002) as well as using special heuristics (Wang et al. 2003, Chai 2009). None of these approaches is suitable for analog circuits because appropriate heuristics are not known and the general approaches can not handle cases that are not fully symmetric.

3. Enhanced Building Block Analysis

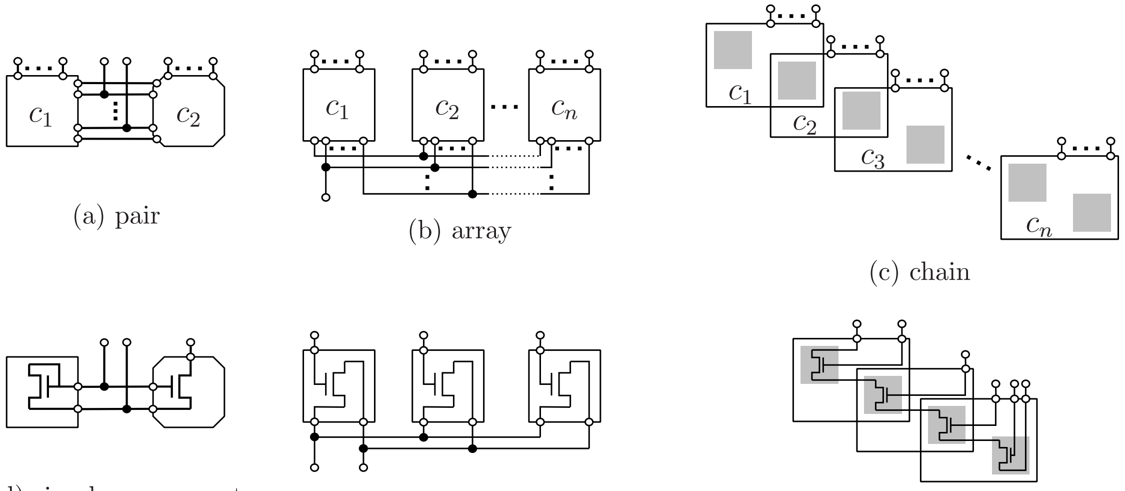
The idea of building block analysis is that most integrated analog and digital circuits are composed out of a few basic building blocks of fixed topology. Although these building blocks are small they make up the essential circuit functions. Typical examples of analog building blocks are current mirrors and differential pairs. Typical examples of digital building blocks are inverters or NAND gates. Building block analysis tries to identify such building blocks in a given circuit. The results can then be fed to other analyses allowing, e.g., automatic constraint generation.

In the following, a new building block analysis method is presented. It consists of a new building block library containing analog and digital building blocks, the corresponding ambiguity resolution graph and a new algorithm to handle this library. The chapter is concluded by a discussion of the contributions compared to the state of the art and by experimental results.

3.1. Library of Analog and Digital Building Blocks

The library is shown in Fig. 3.2. The library is organized in a hierarchical way. Each building block in the library consists of devices (e.g., transistors) or other building blocks. Each building block and each device has a number of pins that connect to the nets of the circuit. The building blocks can be distinguished according to the following criteria:

- *Hierarchy level*
  The library is partitioned into n_L hierarchy levels. The building blocks from one hierarchy level are built out of building blocks from lower hierarchy levels. The overall number of hierarchy levels n_L depends on the size of the logic gates used. The presented algorithm will determine the correct value of n_L automatically.
- *Generic building block type*
  The building blocks in the library are of three different generic building block types:
  - *Pair*
    A pair is formed by exactly two children c_1, c_2, which are building blocks or devices of different or equal type (Fig. 3.1a). At least one pin of child c_1 must be connected to one pin of child c_2. The pins of the pair can be formed by connected and unconnected pins of c_1 and c_2. An example is a *simple*



(a) pair

(b) array

(c) chain

(d) simple current mirror

(e) logic array

(f) stack chain

Figure 3.1.: Generic building block types ((a) to (c)) and corresponding examples ((d) to (f)).

current mirror (Fig. 3.1d). Child c_1 is a diode–connected transistor (more specifically a *diode transistor array*) and child c_2 is a transistor without self connections (more specifically a *normal transistor array*). Both children are connected at their gate and source pins. The input pin of the overall *simple current mirror* is connected to the gate pins of both children. The source pin of the *simple current mirror* is connected to the source pins of both children. The output pin of the *simple current mirror* is connected to the drain pin of the second child.

– *Array*
An array is formed by n children c_1 to c_n which are building blocks or devices of equal type (Fig. 3.1b). All children are connected in parallel by one or more nets. The pins of the pair can be formed by connected and unconnected pins of c_1 to c_n. An example is a *logic array* with three children (Fig. 3.1e) from level 3 of the library in Fig. 3.2. It consists of transistors without self connections (*normal transistor arrays*). Drain pins and source pins of all three transistors are connected together and form pins of the logic array. The gates of the transistors form three additional pins of the logic array.

– *Chain*
A chain is formed by n children c_1 to c_n which are pairs of equal type (Fig. 3.1c). For two children c_i, c_{i+1}, $i = 1 \ldots (n-1)$ the following applies: The second child of c_i is the first child of c_{i+1}, i.e., c_i and c_{i+1} share one child. The pins of the chain are formed by the pins of c_1 to c_n. An example is a stack chain with three children (Fig. 3.1f) from level 4 of the library in

Fig. 3.2. It consists of three *stacks on level 3* that overlap at one transistor. The pins of the stack chain are formed by all gate pins, the drain pin of the first transistor and the source pin of the last transistor.

- *Signal type*
  Building blocks that are only used for analog signals are shown in the unshaded part of the library. The gray shaded part contains building blocks that are only used for digital signals. Building blocks that can be used in both cases are part of the striped part.

- *Subtype*
  For all building blocks except *pass gate*, *logic gate* and *tristate control block* an NMOS and a PMOS subtype exists. This denotes whether the building block is built out of NMOS or PMOS transistors. In general, only the NMOS subtype is shown in Fig. 3.2.

A description of the different hierarchy levels follows.

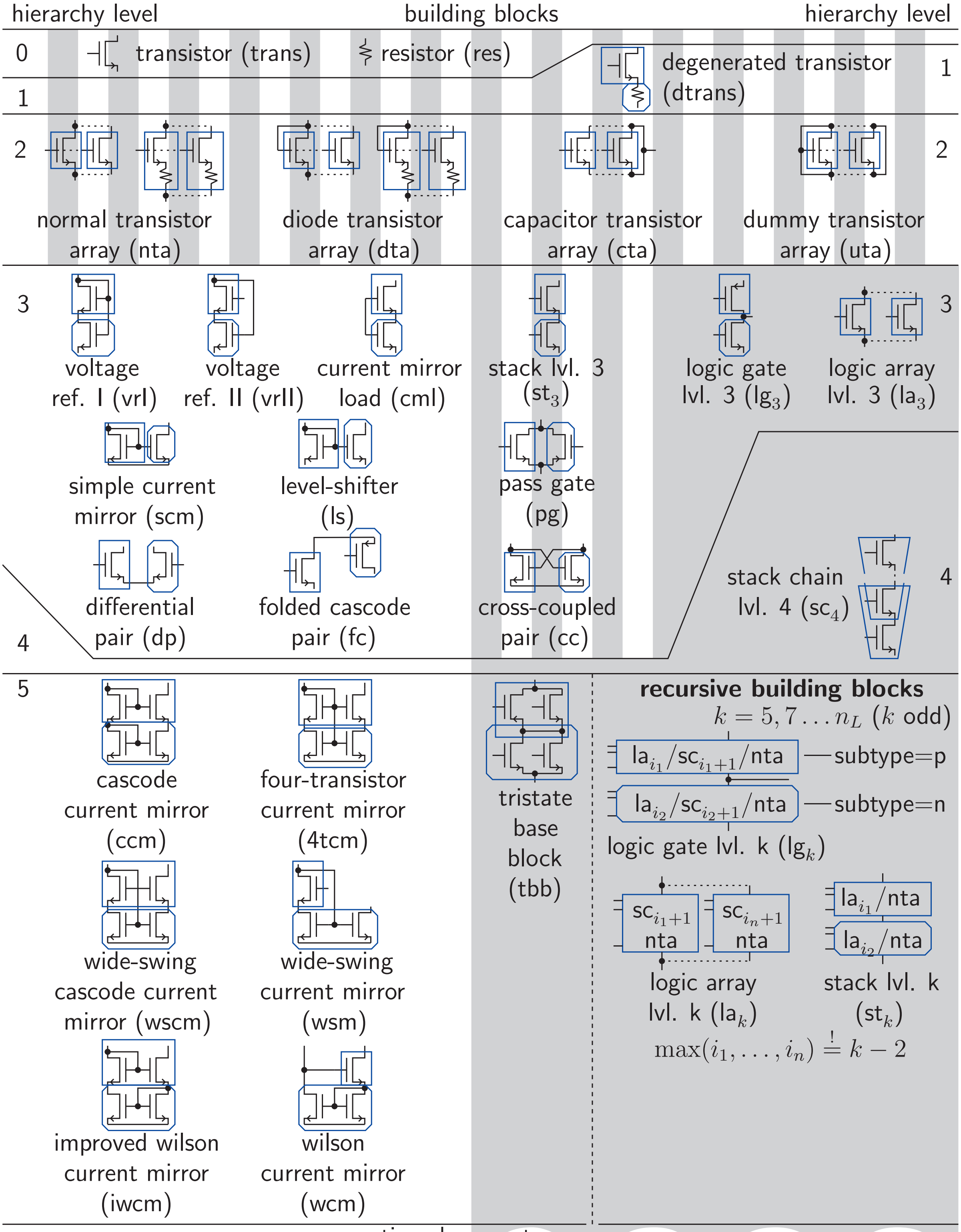
3.1.1. Hierarchy levels 0 and 1

Hierarchy level 0 contains the devices *transistor* (trans) and *resistor* (res). It also can contain other devices like capacitors or inductors but they are not shown in Fig. 3.2.

Hierarchy level 1 contains one pair consisting of a *transistor* and a *resistor*. Such a pair is known as *degenerated transistor* (dtrans) (Johns & Martin 1997).

3.1.2. Hierarchy level 2

Hierarchy level 2 contains four kind of arrays that pool parallel transistors together based on the connection of the transistor with itself. The *normal transistor array* (nta) consists of *transistors* or *degenerated transistors* (dtrans), where all pins connect to different nets. *Transistors* or *degenerated transistors* (dtrans), where gate and drain are connected, form *diode transistor arrays* (dta). A *capacitor transistor array* (cta) contains *transistors* where drain and source are connected. *Transistors* where all pins are connected, form *dummy transistor arrays* (uta). All of these array types are allowed to have one child only, i.e., they are also used for single transistors, which have the required connections.


hierarchy level
building blocks
hierarchy level
0
transistor (trans)
resistor (res)
1
degenerated transistor (dtrans)
1
2
normal transistor array (nta)
diode transistor array (dta)
capacitor transistor array (cta)
dummy transistor array (uta)
2
3
voltage ref. I (vrI)
voltage ref. II (vrII)
current mirror load (cml)
stack lvl. 3 (st_3)
logic gate lvl. 3 (lg_3)
logic array lvl. 3 (la_3)
3
simple current mirror (scm)
level-shifter (ls)
pass gate (pg)
differential pair (dp)
folded cascode pair (fc)
cross-coupled pair (cc)
stack chain lvl. 4 (sc_4)
4
4
5
cascode current mirror (ccm)
four-transistor current mirror (4tcm)
tristate base block (tbb)
recursive building blocks
$k = 5, 7 \ldots n_L$ (k odd)
$\mathsf{la}_{i_1}/\mathsf{sc}_{i_1+1}/\mathsf{nta}$
subtype=p
$\mathsf{la}_{i_2}/\mathsf{sc}_{i_2+1}/\mathsf{nta}$
subtype=n
logic gate lvl. k (lg_k)
sc_{i_1+1} nta
sc_{i_n+1} nta
$\mathsf{la}_{i_1}/\mathsf{nta}$
$\mathsf{la}_{i_2}/\mathsf{nta}$
logic array lvl. k (la_k)
stack lvl. k (st_k)
$\max(i_1, \ldots, i_n) \overset{!}{=} k - 2$
wide-swing cascode current mirror (wscm)
wide-swing current mirror (wsm)
improved wilson current mirror (iwcm)
wilson current mirror (wcm)

continued on next page

Figure 3.2.: Building block library

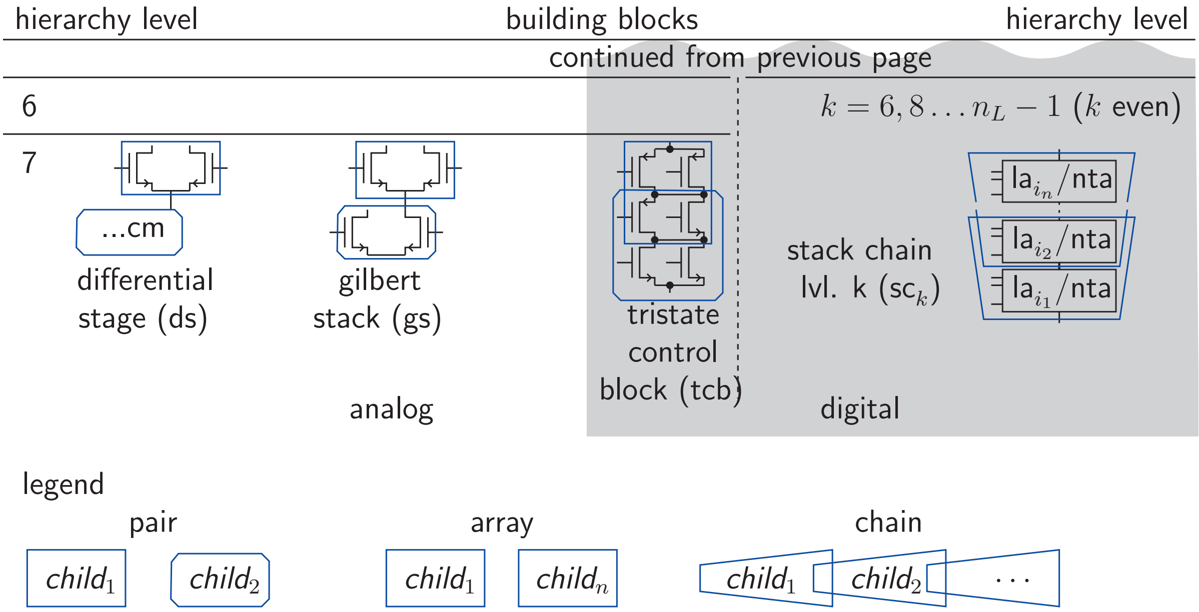


Figure 3.2.: Building block library

3.1.3. Hierarchy level 3

The analog part of hierarchy level 3 contains the pairs *voltage reference I* (vrI), *voltage reference II* (vrII), *current mirror load* (cml), *simple current mirror* (scm), *level shifter* (ls), *differential pair* (dp) and *folded cascode pair* (fc). They are built of *normal transistor arrays* and *diode transistor arrays*. The pairs *stack on level 3* (st_3), *pass gate* (pg) and *cross–coupled pair* (cc) can be used in analog and digital circuits. They consist of *normal transistor arrays*. Pairs *logic gate on level 3* (lg_3) and *logic array on level 3* (la_3) are used in digital circuits only.

3.1.4. Hierarchy level 4

Hierarchy level 4 contains a *stack chain on level 4* (sc_4), which is constructed from *stacks* from level 3 that overlap at one transistor. It is used for digital circuits only.

3.1.5. Hierarchy level 5 (non–recursive part)

The analog part of hierarchy level 5 contains several larger current mirrors. A *cascode current mirror* (ccm) consists of a *simple current mirror* and a *level shifter*. A *four–transistor current mirror* (4tcm) is built out of a *voltage reference I* and a *current mirror load*. A *wide–swing cascode current mirror* (wscm) and a *wide–swing current* *mirror* (wsm) are formed by a *voltage reference II* and a *cascode pair* or *normal transistor array*, respectively. An *improved wilson current mirror* (iwcm) and a *wilson current mirror* (wcm) consist of a *simple current mirror* plus a *level shifter* or *normal transistor array*, respectively. The digital part of level 5 contains the *tristate base block* (tbb), consisting of a *pass gate* and a *logic array*. It is required to handle one type of tristate cells correctly.

3.1.6. Hierarchy level 7 (non–recursive part)

The analog part of hierarchy level 7 contains the *differential stage* (ds), which consists of some current mirror plus a *differential pair*, as well as the *Gilbert stack* (gs), which consists of two *differential pairs*. It is required for Gilbert–type mixers (Vallee & Masry 1994). The digital part contains the *tristate control block* (tcb), consisting of two complementary *tristate base blocks*.

3.1.7. Hierarchy levels $5, 7, \ldots, n_L$ (recursive part)

Some of the digital building blocks in all hierarchy levels starting from level 5 are defined recursively. This allows to handle all CMOS circuits without explicitly specifying all possible variants. These building blocks are *logic gate*, *logic array*, *stack* and *stack chain*.

A *stack* (st_k) on one of the odd hierarchy levels $k = 5, 7, 9, \ldots$ can be formed out of *logic arrays* or *normal transistor arrays*. At least one of the children must be from level $k - 2$, while the other can be from any level less than k. The same principle applies to *logic arrays* (la_k), which are built of *stack chains* or *normal transistor arrays*, as well as *logic gates* (lg_k), which are formed by *stack chains*, *logic arrays* or *normal transistor arrays*. One of these building blocks must be of PMOS subtype and one must be of NMOS subtype.

3.1.8. Hierarchy levels $6, 8, \ldots, n_L - 1$ (recursive part)

Even hierarchy levels $k = 6, 8, 10, \ldots$ contain *stack chains* (sc_k) which are formed by *stacks* from level $k - 1$.

3.1.9. Example

Figure 3.3 illustrates how a compound gate (Weste & Harris 2005, Fig. 1.18) can be represented using the library. The gate consists of PMOS transistors P_1 to P_4 and NMOS transistors N_1 to N_4. Each transistor is represented by *normal transistor arrays*

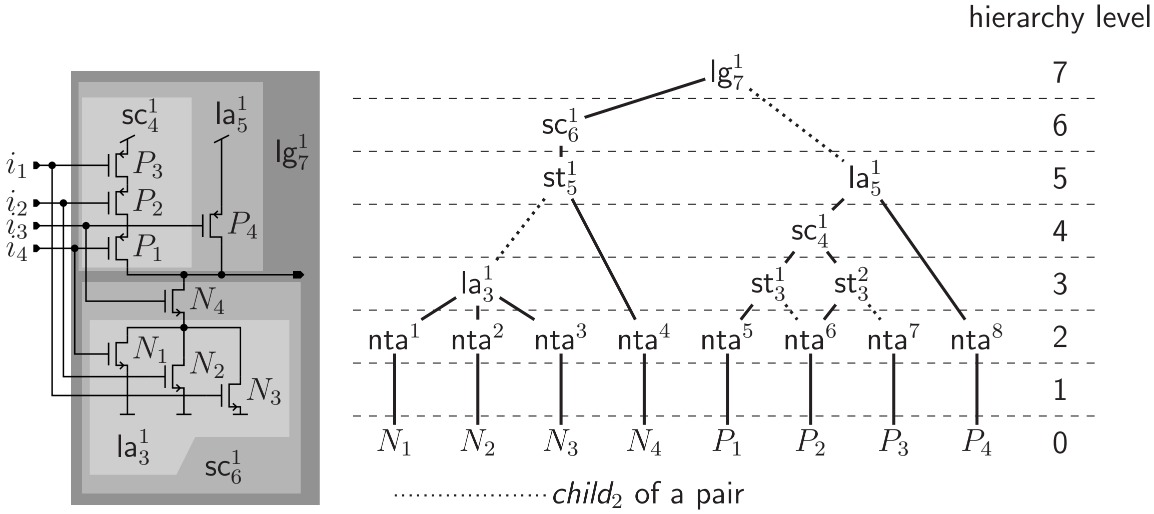


Figure 3.3.: Compound gate (Weste & Harris 2005, Fig. 1.18) with recognized building blocks.

nta^1 to nta^8 on hierarchy level 2. On hierarchy level 3, *logic array* la$^1_3$ consisting of N_1, N_2 and N_3 is formed. *Stack* st$^1_3$ consists of P_1 and P_2. *Stack* st$^2_3$ consists of P_2 and P_3. Since st$^1_3$ and st$^2_3$ share P_2, they form *stack chain* sc$^1_4$ on hierarchy level 4. On hierarchy level 5, sc$^1_4$ and P_4 form *logic array* la$^1_5$. *Stack* st$^1_5$ is built of la$^1_3$ and nta^4. It becomes *stack chain* sc$^1_6$ on hierarchy level 6. Finally, *logic gate* lg$^1_7$ is formed by sc$^1_6$ and la$^1_5$ on hierarchy level 7.

3.1.10. Dominance graph

Ambiguities can occur when comparing the library to the circuit. One transistor can become part of multiple, conflicting building blocks. In these cases it must be decided which building block is the correct one. For this purpose, the ambiguity resolution concept of Massier (2010) (see Section 2.1.1) is used. However, a new dominance graph is required for the extended library. The new dominance graph is shown in Fig. 3.4. The analog part of this graph essentially matches the dominance graph proposed by Massier (2010) (Fig. 2.3). A small modification is made to include the *folded cascode pair*. It dominates the *level shifter* part of a *cascode current mirror* and the output part of a *wide–swing cascode current mirror*. This avoids that transistors of folded cascodes are hidden in *cascode current mirrors* and *wide–swing cascode current mirrors*. This is preferable for the structural signal path analysis discussed in Chapter 4.

Inside each level of the digital, recursive part of the library the following holds: The first device or building block of a *stack* dominates a *logic gate*. In the example of Fig. 3.3 this, e.g., prevents a false inverter consisting of P_1 and N_4. A *logic array*

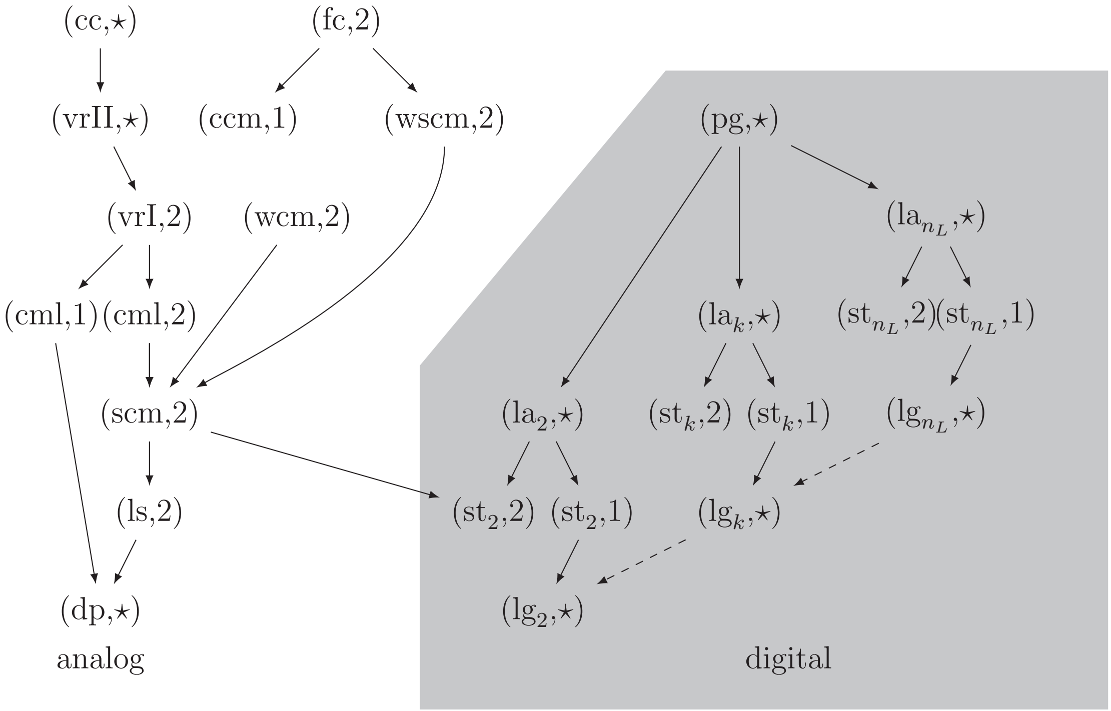


Figure 3.4.: Dominance graph for the building block library from Fig. 3.2.

dominates a *stack*. In addition, *logic gates* from higher hierarchy levels dominate *logic gates* from lower hierarchy levels. This means, in case different *logic gates* of different sizes can be formed, the largest one is selected. Transistors that are part of a *pass gate* must not be part of any other building block.

Currently, there is only one connection between the analog and the digital part. The *simple current mirror* dominates a *stack*. However, further research in the area of mixed–signal circuits might reveal more dependencies.

3.2. Enhanced Algorithm

Below, a formal notation of a circuit is introduced and the algorithm is described. Figure 3.5 gives an overview of the data structures using a pseudo object–oriented description. A circuit consists of a set of devices $\mathcal{D}$, a set of building blocks $\mathcal{B}$ and a set of nets $\mathcal{N}$. Building blocks and devices can be abstracted as components. They share the attributes listed under component in Fig. 3.5. In the following, attributes are denoted in a pseudo object–oriented manner, e.g., *c.type* denotes the type attribute *type* of component *c*. In case a component *c* is a device, its type *c.type* is in the set $T_\mathcal{D}$ of device types,

$$c.\mathit{type} \in T_\mathcal{D} = \{\text{trans}, \text{res}, \ldots\}\,, \tag{3.1}$$

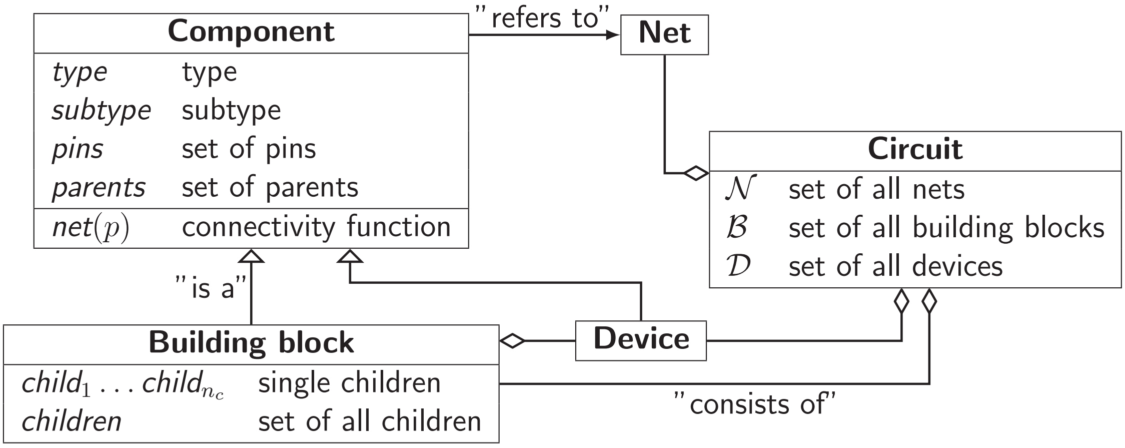

Component
type type
subtype subtype
pins set of pins
parents set of parents
net(p) connectivity function
"refers to"
Net
Circuit
$\mathcal{N}$ set of all nets
$\mathcal{B}$ set of all building blocks
$\mathcal{D}$ set of all devices
"is a"
Building block
child$_1$ ... child$_{n_c}$ single children
children set of all children
Device
"consists of"

Figure 3.5.: Data structures representing a circuit.

where *trans* stands for transistor and *res* stands for resistor. In case a component c is a building block, its type $c.type$ is in library–dependent set $T_\mathcal{B}$ of all building block types,

$$c.type \in T_\mathcal{B} \,. \tag{3.2}$$

The subtype $c.subtype$ of a component c denotes whether it is NMOS or PMOS, i.e.,

$$c.subtype \in \{—, \mathrm{n}, \mathrm{p}\} \,, \tag{3.3}$$

where n stands for NMOS, p stands for PMOS and — stands for none. The set of pins $c.pins$ lists all pins of a component c. It depends on the type of the component. For example, $N_1.pins = \{\mathrm{sc}, \mathrm{gt}, \mathrm{dn}\}$ for a transistor N_1. The relation between nets and pins of a component c is given by connectivity function

$$c.net(p) : c.pins \to \mathcal{N} \,. \tag{3.4}$$

Pin p of component c connects to net n iff $c.net(p) = n$.

A building block $b \in \mathcal{B}$ is formed by several components. In the following, these components will be denoted as children. They are denoted by attributes $b.child_1, b.child_2, \ldots,$ $b.child_{n_c}$. The set of all children of a building block b is given by,

$$b.children = \{b.child_1, b.child_2, \ldots, b.child_{n_c}\} \subseteq (\mathcal{D} \cup \mathcal{B}) \tag{3.5}$$

If a component a is child of a building block b then b is denoted as parent of a. It holds,

$$a \in b.children \Leftrightarrow b \in a.parents \qquad a.parents \subseteq \mathcal{B} \,, \tag{3.6}$$

```
1   $i \leftarrow 0;\ \mathcal{B} \leftarrow \emptyset$
2       $i \leftarrow i+1;\ B_i \leftarrow \emptyset$
3       for all $t \in L_i$
4           $t$ is a
5           pair:
6               $B_i \leftarrow B_i \cup \mathsf{findPairs}(t)$
7               see Section 3.2.1
5           array:
6               $B_i \leftarrow B_i \cup \mathsf{findArrays}(t)$
7               see Section 3.2.2
5           chain:
6               $B_i \leftarrow B_i \cup \mathsf{findChains}(t)$
7               see Section 3.2.3
8       $B'_i \leftarrow \{b \in B_i | \overline{\exists_{a \in \mathcal{B}}\ a\ \mathsf{dominates}\ b}\}$
9       $B_{\mathsf{dom}} \leftarrow \{c \in \mathcal{B} | \exists_{b \in B'_i}\ b\ \mathsf{dominates}\ c\}$
10      $\mathcal{B} \leftarrow (\mathcal{B} \cup B'_i) \setminus B_{\mathsf{dom}}$
11  until $\big(\{b \in \mathcal{B} | b.\mathit{type} \in L_i\} = \emptyset\big) \wedge \big(i = 7, 9, \ldots\big)$
12  $\mathcal{B} \leftarrow \mathsf{removeUncertainBuildingBlocks}(\mathcal{B})$
```

Figure 3.6.: Enhanced building block analysis algorithm.

where $a.parents$ is the set of all parents of a.

The enhanced building block analysis algorithm is shown in Fig. 3.6. The outer loop (lines 2 to 11) iterates over the different hierarchy levels i of the building block library. Set L_i contains the building block types of level i,

$$L_i \subseteq T_{\mathcal{B}}\ . \tag{3.7}$$

For example L_2 for hierarchy level 2 is,

$$L_2 = \{nta, dta, cta, uta\}\ . \tag{3.8}$$

The outer loop ends if no new building blocks were found in the current hierarchy level. The termination condition is checked in level seven for the first time. This guarantees that all non–recursive building blocks are recognized. After that, the condition is evaluated at all odd hierarchy levels because we expect to find a *logic gate* at the very end. *Logic gates* are part of the odd hierarchy levels $3, 5, \ldots$ (see Fig. 3.2).

Inside each hierarchy level, the inner loop (lines 3 to 7) iterates over all building block types belonging to that level. Depending whether the current type is a pair, array or chain, corresponding functions are called to find new building blocks (line 6). These functions are described in detail in Sections 3.2.1 to 3.2.3.

After that, the dominance relation is evaluated with respect to the found building blocks. Line 8 checks whether the new building blocks are already dominated by some existing building block. Line 9 computes the set of all existing building blocks that are dominated by the new building blocks. Relation *dominates* corresponds to Eq. (2.6). Finally, the set of all building blocks is updated.

After termination of the main loop all uncertain building blocks that do not have parents are removed similar to the algorithm of Massier (2010) (see Section 2.1.1).

3.2.1. Recognition of Pairs

In the following, the algorithm to find pairs is described. It is controlled by a rule function $\rho_t : (\mathcal{D} \cup \mathcal{B})^2 \to \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$ is the set of Boolean numbers. Two components x_1 and x_2 are recognized as pair of type t if ρ_t is true. The available conditions cover required and forbidden connections, required types and subtypes as well as existence of parents. For example, the rule function ρ_{scm} for a *simple current mirror* reads as follows.

$$\begin{aligned}\rho_{\text{scm}}(x_1, x_2) = \Big[&\underbrace{\big(x_1.\mathit{type} = \text{dta}\big) \wedge \big(x_2.\mathit{type} = \text{nta}\big)}_{\text{type}} \wedge \underbrace{\big(x_1.\mathit{subtype} = x_2.\mathit{subtype}\big)}_{\text{same subtype}} \\ \wedge &\underbrace{\big(x_1.\mathit{net}(\text{gt}) = x_2.\mathit{net}(\text{gt})\big) \wedge \big(x_1.\mathit{net}(\text{sc}) = x_2.\mathit{net}(\text{sc})\big)}_{\text{required connections}} \wedge \underbrace{\big(x_1.\mathit{net}(\text{gt}) \neq x_2.\mathit{net}(\text{dn})\big)}_{\text{forbidden connection}}\Big]\end{aligned} \tag{3.9}$$

The *type* conditions require the first component to be a *diode transistor array* and the second component to be a *normal transistor array*. Both components must have the *same subtype*. The *required connection* conditions require the gates and sources of both components to be connected. The *forbidden connection* condition forbids a connection between the gate of the first component, which is the drain at the same time, and the drain of the second component.

In order to speed up the recognition algorithm, the rule function is split up into three partial rule functions, for the first child, for the second child and the pair

$$r_{1,t} : \mathcal{N} \times (\mathcal{D} \cup \mathcal{B}) \to \mathbb{B} \quad r_{2,t} : \mathcal{N} \times (\mathcal{D} \cup \mathcal{B}) \to \mathbb{B} \quad r_{12,t} : (\mathcal{D} \cup \mathcal{B})^2 \to \mathbb{B} \tag{3.10}$$

such that

$$\rho_t(x_1, x_2) \Leftrightarrow \exists_{n \in \mathcal{N}} \big[r_{1,t}(n, x_1) \wedge r_{2,t}(n, x_2)\big] \wedge r_{12,t}(x_1, x_2)\ . \tag{3.11}$$

For example the rule function ρ_{scm} for a *simple current mirror* can be rewritten to

$$r_{1,\text{scm}}(n, x_1) \Leftrightarrow \big(x_1.\mathit{type} = \text{dta}\big) \wedge \big(x_1.\mathit{net}(\text{gt}) = n\big) \tag{3.12}$$

$$r_{2,\text{scm}}(n, x_2) \Leftrightarrow \big(x_2.\mathit{type} = \text{nta}\big) \wedge \big(x_2.\mathit{net}(\text{gt}) = n\big) \tag{3.13}$$

$$\begin{aligned} r_{12,\text{scm}}(x_1, x_2) \Leftrightarrow{} & \big(x_1.\mathit{subtype} = x_2.\mathit{subtype}\big) \wedge \big(x_1.\mathit{net}(\text{gt}) \neq x_2.\mathit{net}(\text{dn})\big) \\ & \wedge \big(x_1.\mathit{net}(\text{sc}) = x_2.\mathit{net}(\text{sc})\big) \end{aligned} \tag{3.14}$$

In general, the rule functions for any pair type t can be determined from the library in Fig. 3.2 as follows:

1. One of the connections between both children is selected as characteristic connection. If possible, connections should be avoided that correspond to a power net.
2. Function $r_{1,t}$ for the first child is formed by a type rule for the first child plus the corresponding part of the required connection rule for the characteristic connection.
3. Function $r_{2,t}$ for the second child is formed by a type rule for the second child plus the corresponding part of the required connection rule for the characteristic connection.
4. All other conditions form function $r_{12,t}$:
    a) A required connection rule for every connection shown in Fig. 3.2 except the characteristic connection.
    b) A forbidden connection rule for every connection not shown in Fig. 3.2.
    c) A same subtype rule if only NMOS transistors are shown in Fig. 3.2. If the building block consists of NMOS and PMOS transistors, corresponding subtype rules are added.
    d) For all recursive building blocks, a rule is added that forbids that children already have parents.

Based on the formulation of recognition rules as function $r_{1,t}$, $r_{2,t}$ and $r_{12,t}$ the algorithm can be implemented as shown in Fig. 3.7. A set X of candidate pairs is determined by iterating over all nets (lines 2 to 5). For each net, a candidate set X_1 is computed that contains only components where $r_{1,t}$ is true. Afterward, X_2 is computed analogously. This can be implemented efficiently by keeping an appropriate index for each net. For all resulting pairs in $X_1 \times X_2$ and thus for all pairs in X the existence condition in Eq. (3.11) is fulfilled. Thus, a new pair can be created for all pairs in X where $r_{12,t}$ is fulfilled (lines 6 and 7).

```
findPairs($t$)
1 $B \leftarrow \emptyset$; $X \leftarrow \emptyset$
2 for all $n \in \mathcal{N}$
3     $X_1 \leftarrow \{x_1 \in \mathcal{D} \cup \mathcal{B} | r_{1,t}(n, x_1)\}$
4     $X_2 \leftarrow \{x_2 \in \mathcal{D} \cup \mathcal{B} | r_{2,t}(n, x_2)\}$
5     $X \leftarrow X \cup (X_1 \times X_2)$
6 for all $(x_1, x_2) \in \{(x_1, x_2) \in X | r_{12,t}(x_1, x_2)\}$
7     $B \leftarrow B \cup \{\text{newPair}(t, x_1, x_2)\}$
8 return $B$
```

Figure 3.7.: Algorithm to find pairs.

For the worst–case and average runtime complexity of the algorithm, the following considerations can be made. In lines 3 and 4, sets X_1 and X_2 are computed for one net n. In the worst case all components $\mathcal{D} \cup \mathcal{B}$ are connected to n. This computation can be made fast by using an appropriate index, that is based on red–black trees (Sedgewick 1988), yielding logarithmic complexity for the worst–case runtimes $t_3^{\max}$ and $t_4^{\max}$,

$$t_3^{\max} = t_4^{\max} \in \mathcal{O}(\log|\mathcal{D} \cup \mathcal{B}|) \ , \tag{3.15}$$

In line 5, every new pair requires one insert operation into set X. Assuming that X is implemented using red–black trees and $|X_1| = |X_2| = |\mathcal{D} \cup \mathcal{B}|$ in the worst case (Sedgewick 1988), this yields,

$$t_5^{\max} \in \mathcal{O}(|\mathcal{D} \cup \mathcal{B}|^2 \log|\mathcal{D} \cup \mathcal{B}|^2) = \mathcal{O}(|\mathcal{D} \cup \mathcal{B}|^2 \log|\mathcal{D} \cup \mathcal{B}|) \ . \tag{3.16}$$

Thus, the worst–case runtime t_{2-5} for the upper loop is,

$$t_{2-5}^{\max} = |\mathcal{N}|(t_3^{\max} + t_4^{\max} + t_5^{\max}) \in \mathcal{O}(|\mathcal{N}| \cdot |\mathcal{D} \cup \mathcal{B}|^2 \cdot \log|\mathcal{D} \cup \mathcal{B}|) \ . \tag{3.17}$$

The lower loop is executed $|X|$ times, which is $|\mathcal{D} \cup \mathcal{B}|^2$ in the worst case. In addition, it must be assumed that $r_{12,t}$ is true for every pair and consequently $|\mathcal{D} \cup \mathcal{B}|^2$ new pairs are created, i.e.,

$$|B|_{\text{findPairs}}^{\max} = |\mathcal{D} \cup \mathcal{B}|^2 \ . \tag{3.18}$$

Creating a new pair has logarithmic complexity with respect to the overall number of components, since the index mentioned above must be updated. Overall, the complexity for the worst–case runtime $t_{6-7}^{\max}$ for lines 6 and 7 is,

$$t_{6-7}^{\max} \in \mathcal{O}(|\mathcal{D} \cup \mathcal{B}|^2 \log|\mathcal{D} \cup \mathcal{B}|) \ . \tag{3.19}$$

The runtime complexity for the upper loop is dominant, yielding the following complexity for the worst–case runtime $t_{\text{findPairs}}^{\max}$ of the complete function,

$$t_{\text{findPairs}}^{\max} \in \mathcal{O}(|\mathcal{N}| \cdot |\mathcal{D} \cup \mathcal{B}|^2 \cdot \log|\mathcal{D} \cup \mathcal{B}|) \; . \tag{3.20}$$

However, these assumptions are hardly realistic. In a real circuit, there are many nets with only few devices connected. In addition, it can be assumed, that $r_{1,t}$ and $r_{2,t}$ are only true for a number of devices that is approximately constant. This yields the following typical runtime complexity t_{2-5}^{typ} of the upper loop,

$$t_{2-5}^{\text{typ}} \in |\mathcal{N}| \cdot (\mathcal{O}(1) + \mathcal{O}(1) + \mathcal{O}(\log|\mathcal{D} \cup \mathcal{B}|) = \mathcal{O}(|\mathcal{N}| \cdot \log|\mathcal{D} \cup \mathcal{B}|) \tag{3.21}$$

Experiments show that the number of created new pairs $|B|$ is approximately linear to the circuit size,

$$|B|^{\text{typ}} \in \mathcal{O}(|\mathcal{D} \cup \mathcal{B}|) \; . \tag{3.22}$$

Consequently the typical runtime t_{6-7}^{typ} has the following complexity,

$$t_{6-7}^{\text{typ}} \in \mathcal{O}(|\mathcal{D} \cup \mathcal{B}| \log|\mathcal{D} \cup \mathcal{B}|) \; , \tag{3.23}$$

yielding the following typical runtime complexity for the function,

$$t_{\text{findPairs}}^{\text{typ}} \in \mathcal{O}\Big((|\mathcal{N}| + |\mathcal{D} \cup \mathcal{B}|) \log|\mathcal{D} \cup \mathcal{B}| \Big) \; . \tag{3.24}$$

3.2.2. Recognition of Arrays

Next, the algorithm to find arrays is discussed. Theoretically, an array of type t can be found by evaluating a rule function $\rho_t(x_1, x_2, \ldots, x_n)$ for n–tuples $(x_1, x_2, \ldots, x_n)$ of potential children as follows,

$$\begin{aligned}&\rho_t(x_1, x_2, \ldots, x_n) \Leftrightarrow \\ &n \geq n_{\min,t} \wedge \underset{i \in \{1,2,\ldots,n\}}{\forall} r_{\text{chld}}(x_i) \wedge \underset{\nu_1, \nu_2, \ldots, \nu_{m_t} \in \mathcal{N}}{\exists} \; \underset{i \in \{1,2,\ldots,n\}}{\forall} r_{\text{con},t}(x_i, \nu_1, \nu_2, \ldots, \nu_{m_t}) \; .\end{aligned} \tag{3.25}$$

Constant $n_{\min,t}$ is the minimum number of components for an array of type t. Function

$$r_{\text{chld}} : \mathcal{D} \cup \mathcal{B} \to \mathbb{B}, \tag{3.26}$$

describes the properties of each child. It can contain conditions about type, subtype, existence of parents as wells as required and forbidden connections of the child with itself. Function,

$$r_{\text{con}} : (\mathcal{D} \cup \mathcal{B}) \times \mathcal{N}^{m_t} \to \mathbb{B} \tag{3.27}$$

describes the connectivity between the children and does only contain conditions about required connections. The size m_t of the tuple of nets $\nu_1, \nu_2, \ldots, \nu_{m_t}$ is type specific and constant.

In addition, the tuple must be the largest possible one. This means, it must not be possible to extend a valid tuple $(x_1, x_2, \ldots, x_n)$ by a child x_{n+1}, such that ρ_t is still valid, i.e.,

$$\overline{\underset{x_{n+1}}{\exists}\ \rho_t(x_1, x_2, \ldots, x_n, x_{n+1})}\ . \tag{3.28}$$

Based on Eq. (3.25), Eq. (3.28) can be rewritten to,

$$\begin{aligned}
&\overline{\underset{x_{n+1}}{\exists} \Big[n+1 \geq n_{\text{min},t} \wedge \underset{i\in\{1,2,\ldots,n+1\}}{\forall} r_{\text{chld}}(x_i)} \\
&\qquad \overline{\wedge \underset{\nu_1,\nu_2,\ldots,\nu_{m_t}\in\mathcal{N}}{\exists}\ \underset{i\in\{1,2,\ldots,n+1\}}{\forall} r_{\text{con},t}(x_i, \nu_1, \nu_2, \ldots, \nu_{m_t}) \Big]} \\
\Leftrightarrow\ &\overline{\underset{x_{n+1}}{\exists} \Big[\underbrace{n+1 \geq n_{\text{min},t}}_{1 \text{ if } \rho_t(x_1,x_2,\ldots,x_n)=1} \wedge \underbrace{\underset{i\in\{1,2,\ldots,n\}}{\forall} r_{\text{chld}}(x_i)}_{1 \text{ if } \rho_t(x_1,x_2,\ldots,x_n)=1} \wedge r_{\text{chld}}(x_{n+1})} \\
&\qquad \overline{\wedge \underset{\nu_1,\nu_2,\ldots,\nu_{m_t}\in\mathcal{N}}{\exists}\ \underset{i\in\{1,2,\ldots,n+1\}}{\forall} r_{\text{con},t}(x_i, \nu_1, \nu_2, \ldots, \nu_{m_t}) \Big]}
\end{aligned} \tag{3.29}$$

Under the assumption that $\rho_t(x_1, x_2, \ldots, x_n) = 1$ the first and second term are always true. This yields,

$$\overline{\underset{x_{n+1}}{\exists} \Big[r_{\text{chld}}(x_{n+1}) \wedge \underset{\nu_1,\nu_2,\ldots,\nu_{m_t}\in\mathcal{N}}{\exists}\ \underset{i\in\{1,2,\ldots,n+1\}}{\forall} r_{\text{con},t}(x_i, \nu_1, \nu_2, \ldots, \nu_{m_t}) \Big]}\ . \tag{3.30}$$

This means, there may be no child x_{n+1} that fulfills $r_{\text{chld},t}$ and is connected in parallel.

For example, for a *normal transistor array* the rule functions and type–specific constants are,

$$\begin{aligned}
n_{\text{min,nta}} &= 1 && (3.31)\\
m_{\text{nta}} &= 3 && (3.32)\\
r_{\text{chld,nta}}(x) &\Leftrightarrow \underbrace{\big(x.\mathit{type} = \text{trans}\big)}_{\text{type}} \wedge \underbrace{\big(x.\mathit{net}(\text{gt}) \neq x.\mathit{net}(\text{dn})\big)}_{\text{forbidden connection}}\\
&\quad \wedge \underbrace{\big(x.\mathit{net}(\text{gt}) \neq x.\mathit{net}(\text{sc})\big) \wedge \big(x.\mathit{net}(\text{sc}) \neq x.\mathit{net}(\text{dn})\big)}_{\text{forbidden connections}} && (3.33)\\
r_{\text{con,nta}}(x, \nu_1, \nu_2, \nu_3) &\Leftrightarrow \underbrace{(x.\mathit{net}(\text{gt}) = \nu_1) \wedge (x.\mathit{net}(\text{dn}) = \nu_2) \wedge (x.\mathit{net}(\text{sc}) = \nu_3)}_{\text{required connections}} && (3.34)
\end{aligned}$$

The minimum size for a *normal transistor array* is one. The children of a *normal transistor array* have three parallel connections, thus m_{nta} is three. The rule function $r_{\text{chld,nta}}$ enforces x to be of *type* transistor. It *forbids connections* between any of the pins of the transistor. The rule function $r_{\text{con,nta}}$ *requires* all children to be connected at gate, drain and source.

In general, the constants and rule functions can be determined from the library in Fig. 3.2 as follows.

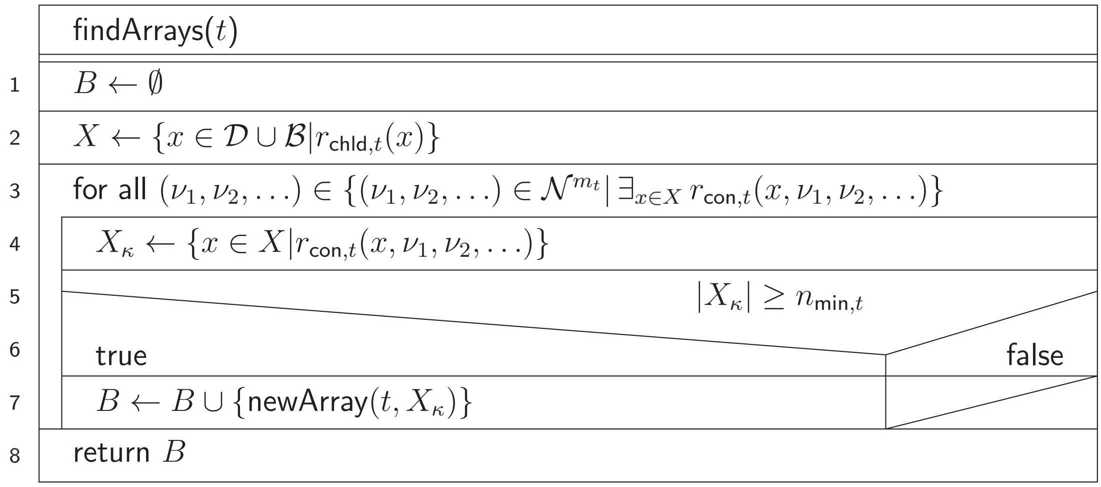
1. Constant $n_{\text{min},t}$ is 1 for the *diode transistor array*, *normal transistor array*, *capacitor transistor array* and *dummy transistor array*. Constant $n_{\text{min},t}$ is 2 for the *logic array*.

2. The nets shown as parallel connected in Fig. 3.2 form required connection conditions in $r_{\text{con},t}$. Constant m_t is equal to the number of nets used for the parallel connections.

3. Rule function $r_{\text{chld},t}$ is formed by a type condition. In addition, connection shown between the pins of one child result in required connection conditions. Connections not shown between the pins of one child result in forbidden connection conditions.

Figure 3.8 shows a recognition algorithm that is based on Eq. (3.25). First, the set X of all components matching rule function $r_{\text{chld},t}$ is computed (line 2). Next, it iterates over all tuples $(\nu_1, \nu_2, \ldots)$ of nets that fulfill $r_{\text{con},t}$ for at least one component from x (line 3). After that, the algorithm finds all components that connect to nets $\nu_1, \ldots, \nu_{m_t}$ (line 4). This ensures that there is no component x_{n+1} left that could fulfill Eq. (3.30). Lines 3 and 4 can be implemented efficiently by using, e.g., maps based on binary trees. Finally, a new array is created in case X_κ is greater than the minimum array size for t (lines 5 to 7).

The runtime complexity of this algorithm is dominated by the searches in lines 3 and 4. In case this is implemented state–of–the–art binary tree implementations (e.g.,



```
findArrays($t$)
$B \leftarrow \emptyset$
$X \leftarrow \{x \in \mathcal{D} \cup \mathcal{B} | r_{\mathsf{chld},t}(x)\}$
for all $(\nu_1, \nu_2, \ldots) \in \{(\nu_1, \nu_2, \ldots) \in \mathcal{N}^{m_t} | \exists_{x \in X}\, r_{\mathsf{con},t}(x, \nu_1, \nu_2, \ldots)\}$
    $X_\kappa \leftarrow \{x \in X | r_{\mathsf{con},t}(x, \nu_1, \nu_2, \ldots)\}$
    $|X_\kappa| \geq n_{\mathsf{min},t}$
    true
        $B \leftarrow B \cup \{$newArray$(t, X_\kappa)\}$
    false
return $B$
```

Figure 3.8.: Algorithm to recognize arrays.

red–black trees), worst–case and typical runtimes are,

$$t^{\mathrm{typ}}_{\mathrm{findArrays}} = t^{\mathrm{max}}_{\mathrm{findArrays}} \in \mathcal{O}(|\mathcal{D} \cup \mathcal{B}| \log |\mathcal{D} \cup \mathcal{B}|) \ , \tag{3.35}$$

respectively. Every component can only become part of one array. The maximum number of arrays $|B|^{\mathrm{max}}_{\mathrm{findArrays}}$ is the number of components,

$$|B|^{\mathrm{max}}_{\mathrm{findArrays}} \leq |\mathcal{D} \cup \mathcal{B}| \ . \tag{3.36}$$

3.2.3. Recognition of Chains

The algorithm to find chains is discussed next. Theoretically, a chain can be found by evaluating a rule function

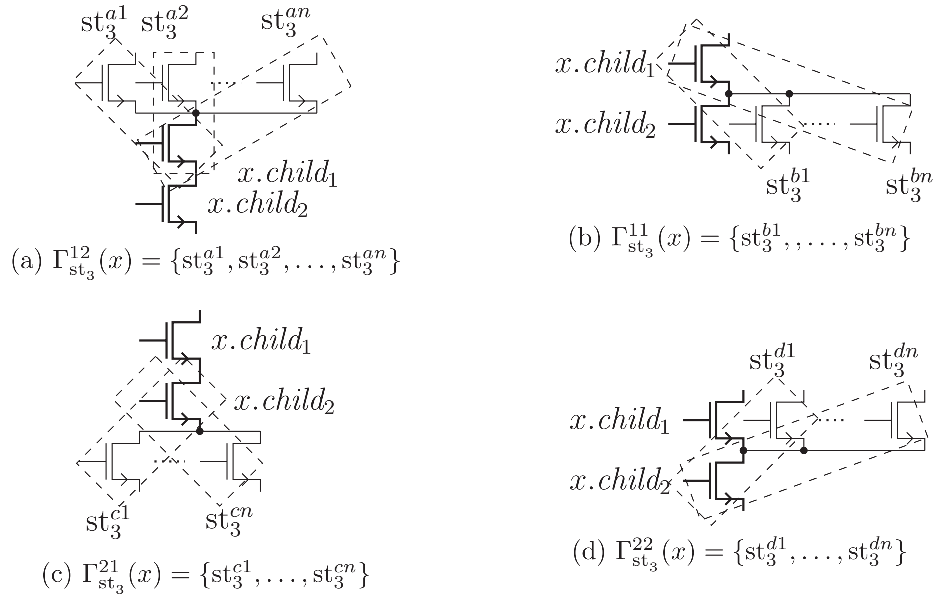
$$\rho_t(x_1, x_2, \ldots, x_n) : (\mathcal{D} \cup \mathcal{B})^n \rightarrow \mathbb{B} \tag{3.37}$$

where n is the (unknown) length of the chain. For a chain of type t, ρ_t is,

$$\begin{aligned} \rho_t(x_1, x_2, \ldots, x_n) :\Leftrightarrow \underset{i=2,\ldots,n}{\forall} & \Big[\Gamma^{21}_t(x_{i-1}) = \{x_i\} \wedge \Gamma^{12}_t(x_i) = \{x_{i-1}\}\Big] \\ & \wedge \underset{i=1,\ldots,n}{\forall} \Big[\Gamma^{22}_t(x_i) = \emptyset \wedge \Gamma^{11}_t(x_i) = \emptyset\Big] \ . \end{aligned} \tag{3.38}$$

where function $\Gamma^{ij}_t(x)$ returns all building blocks x' of type t, where the j–th child is equal to the i–th child of x,

$$\Gamma^{ij}_t(x) = \{x' \in \mathcal{B} | (x'.type = t) \wedge x.child_i = x'.child_j\} \ . \tag{3.39}$$



(a) $\Gamma^{12}_{\mathrm{st}_3}(x) = \{\mathrm{st}_3^{a1}, \mathrm{st}_3^{a2}, \ldots, \mathrm{st}_3^{an}\}$

(b) $\Gamma^{11}_{\mathrm{st}_3}(x) = \{\mathrm{st}_3^{b1}, , \ldots, \mathrm{st}_3^{bn}\}$

(c) $\Gamma^{21}_{\mathrm{st}_3}(x) = \{\mathrm{st}_3^{c1}, \ldots, \mathrm{st}_3^{cn}\}$

(d) $\Gamma^{22}_{\mathrm{st}_3}(x) = \{\mathrm{st}_3^{d1}, \ldots, \mathrm{st}_3^{dn}\}$

Figure 3.9.: Examples for $\Gamma^{ij}_{\mathrm{st}_3}(x)$.

Function $\Gamma^{ij}_{\mathrm{st}_3}(x)$ is illustrated by Fig. 3.9. Function $\Gamma^{12}_{\mathrm{st}_3}(x)$ consists of *stacks* st_3^{a1}, st_3^{a2} to st_3^{an} that are formed by $x.child_1$ and transistors connected to the drain pin of $x.child_1$ by their source pin (Fig. 3.9a). Function $\Gamma^{11}_{\mathrm{st}_3}(x)$ consists of *stacks* st_3^{b1} to st_3^{bn} that are formed by $x.child_1$ and transistors connected to the source pin of $x.child_1$ by their drain pin (Fig. 3.9b). Function $\Gamma^{21}_{\mathrm{st}_3}(x)$ consists of *stacks* st_3^{c1} to st_3^{cn} that are formed by $x.child_2$ and transistors connected to the source pin of $x.child_2$ by their drain pin (Fig. 3.9c). Function $\Gamma^{22}_{\mathrm{st}_3}(x)$ consists of *stacks* st_3^{d1} to st_3^{dn} that are formed by $x.child_2$ and transistors connected to the drain pin of $x.child_2$ by their source pin (Fig. 3.9d).

Equation (3.38) can be interpreted as follows. Two consecutive building block x_{i-1}, x_i, $i = 2 \ldots n$ have the property that,

1. building block x_i is the only building block of type t, where the first child is equal to the second child of x_{i-1}, and

2. building block x_{i-1} is the only building block of type t, where the second child is equal to the first child of x_i.

In addition, every building block x_i in the chain must have the property that,

1. there is no other building block of type t, where the first child is the first child of x_i, and

2. there is no other building block of type t, where the second child is the second child of x_i.

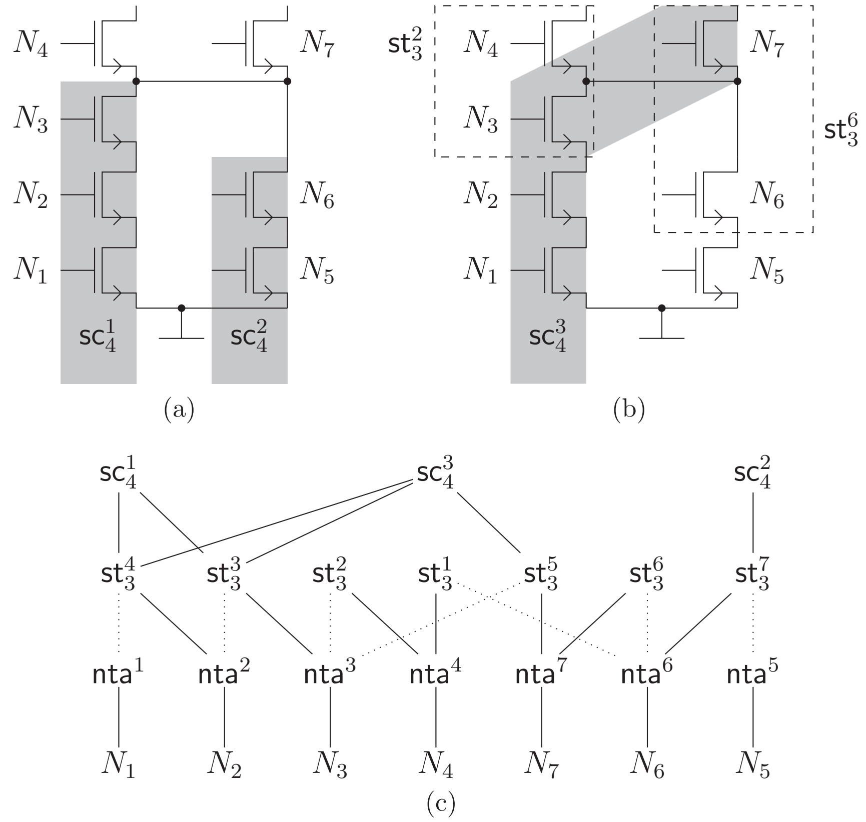


Figure 3.10.: Examples for two valid stack chains sc_4^1 and sc_4^2 (a), an invalid stack chain sc_4^3 (b) as well as the corresponding building block hierarchy (c).

The only chain in the library from Fig. 3.2 is the *stack chain*. The conditions are illustrated by Fig. 3.10. It shows an example consisting of seven NMOS transistors N_1 to N_7. These transistors form *stacks* st_3^1 to st_3^7 (Fig. 3.10c). Stacks st_3^3 and st_3^4 form a valid chain sc_4^1 (Fig. 3.10a), because

$$\begin{aligned}
&\Gamma^{21}_{\mathrm{st}_3}(\mathrm{st}_3^3) = \{\mathrm{st}_3^4\} && \Gamma^{12}_{\mathrm{st}_3}(\mathrm{st}_3^4) = \{\mathrm{st}_3^3\}\\
&\Gamma^{11}_{\mathrm{st}_3}(\mathrm{st}_3^3) = \emptyset && \Gamma^{22}_{\mathrm{st}_3}(\mathrm{st}_3^3) = \emptyset\\
&\Gamma^{11}_{\mathrm{st}_3}(\mathrm{st}_3^3) = \emptyset && \Gamma^{22}_{\mathrm{st}_3}(\mathrm{st}_3^3) = \emptyset\\
\Rightarrow\ &\rho_{\mathrm{sc}_4}(\mathrm{st}_3^3, \mathrm{st}_3^4) = 1\ .
\end{aligned} \tag{3.40}$$

Stacks st_3^3 and st_3^4 share transistor N_2. There are no branches at stack st_3^3 or stack st_3^4. The same is true for sc_4^2 formed by stack st_3^7, because

$$\begin{aligned}
&\Gamma^{11}_{\mathrm{st}_3}(\mathrm{st}_3^7) = \emptyset && \Gamma^{22}_{\mathrm{st}_3}(\mathrm{st}_3^7) = \emptyset\\
\Rightarrow\ &\rho_{\mathrm{sc}_4}(\mathrm{st}_3^7) = 1\ .
\end{aligned} \tag{3.41}$$

Stacks st_3^5, st_3^3 and st_3^4 do not form a valid chain sc_4^3 (Fig. 3.10b), because,

$$\Gamma^{11}_{\mathrm{st}_3}(\mathrm{st}_3^5) = \{\mathrm{st}_3^6\} \qquad \Gamma^{22}_{\mathrm{st}_3}(\mathrm{st}_3^5) = \{\mathrm{st}_3^2\}\ . \tag{3.42}$$

This chain would have two branches and is therefore invalid.

In addition, the chain must be as large as possible. There must be no x_0 that can be added at the beginning and there must be no x_{n+1} that can be added at he end, i.e.,,

$$\overline{\underset{x_0}{\exists}\,\rho_t(x_0, x_1, \ldots, x_n)} \wedge \overline{\underset{x_{n+1}}{\exists}\ \rho_t(x_1, \ldots, x_n, x_{n+1})} \tag{3.43}$$

From the first term of Eq. (3.43), conditions for the first child x_1 are derived in the following. The term can be rewritten to

$$\overline{\underset{x_0}{\exists}\Big[\rho_t(x_1, \ldots, x_n) \wedge \Gamma_t^{21}(x_0) = \{x_1\} \wedge \Gamma_t^{12}(x_1) = \{x_0\} \wedge \Gamma_t^{11}(x_0) = \emptyset \wedge \Gamma_t^{22}(x_0) = \emptyset\Big]}\ . \tag{3.44}$$

Thus set $\Gamma_t^{12}(x_1)$ of a valid start point x_1 must not have exactly one element. Together with the conditions from ρ_t, this yields the following conditions for a valid start point x_1,

$$x_1 \text{ valid start point} \Leftrightarrow \Big[|\Gamma_t^{12}(x_1)| \neq 1 \wedge \Gamma_t^{11}(x_1) = \emptyset \wedge \Gamma_t^{22}(x_1) = \emptyset\Big]\ . \tag{3.45}$$

This allows to formulate the algorithm as shown in Fig. 3.11. The algorithm first determines all building blocks of type t (line 2). After this, valid chains are searched, starting from all components that are valid start points (line 3). A component x_1 is a valid start point, if Eq. (3.45) is fulfilled. The inner loop (lines 4 to 7) runs as long as the chain can be extended. In each iteration, the only element of $\Gamma^{21}(x_i)$ is stored as x_{i+1} and i is incremented. At the end, a new chain is created for x_1 to x_{i-1}. This algorithm can be implemented efficiently by modeling components x_i as graph.

Setting up such a data structure requires iterating over all elements of X and storing their children in, e.g., an adjacency list. The complexity of the worst–case runtime $t_{\mathrm{setup}}^{\max}$ and of the typical runtime $t_{\mathrm{setup}}^{\mathrm{typ}}$ for this is,

$$t_{\mathrm{setup}}^{\max}, t_{\mathrm{setup}}^{\mathrm{typ}} \in \mathcal{O}(|\mathcal{D} \cup \mathcal{B}| \log|\mathcal{D} \cup \mathcal{B}|), \tag{3.46}$$

assuming that insert operations for the adjacency list are logarithmic in time. The procedure described in lines 3 to 10 is similar to a depth–first search (Sedgewick 1988) which is known to be linear in the sum of nodes and edges. In this case the number

```
     findChains($t$)
1    $B \leftarrow \emptyset$
2    $X \leftarrow \{c \in \mathcal{B} | x.\mathit{type} = t\}$
3    for all $x_1 \in \{x \in X \mid x$ valid start point$\}$
4        $i \leftarrow 1$
5        while $(|\Gamma_t^{21}(x_i)| = 1) \wedge (\Gamma_t^{11}(x_i) = \emptyset) \wedge (\Gamma_t^{22}(x_i) = \emptyset)$
6            $x_{i+1} \leftarrow$ only element of$\left(\Gamma_t^{21}(x_i)\right)$
7            $i \leftarrow i + 1$
8        $i > 1$
9        true                                                    false
10           $B \leftarrow B \cup \{$newChain$(t, (x_1, \ldots, x_{i-1}))\}$
11   return $B$
```

Figure 3.11.: Algorithm to recognize chains.

of nodes and edges is of the same size, yielding the following worst–case and typical runtimes,

$$t^{\max}_{\text{3-10}}, t^{\text{typ}}_{\text{3-10}} \in \mathcal{O}(|\mathcal{D} \cup \mathcal{B}|). \tag{3.47}$$

Consequently, the complexity of the worst–case and typical runtime for this function is,

$$t^{\max}_{\text{findChains}}, t^{\text{typ}}_{\text{findChains}} \in \mathcal{O}(|\mathcal{D} \cup \mathcal{B}| \log|\mathcal{D} \cup \mathcal{B}|)\ . \tag{3.48}$$

Every component can only become part of one chain. The maximum number of chains $|B|^{\max}_{\text{findChains}}$ is the number of components,

$$|B|^{\max}_{\text{findChains}} \leq |\mathcal{D} \cup \mathcal{B}|\ . \tag{3.49}$$

3.3. Discussion

In the following, first the overall complexity is calculated. Afterward, differences to the approach of Massier (2010) and pattern recognition approaches are discussed.

In order to calculate the worst–case complexity it is assumed that all hierarchy levels of the library uniformly consist of pairs, arrays and chains. The worst–case for the size of the set of all components after the first iteration is dominated by the worst–case of found new pairs Eq. (3.18), as follows

$$|\mathcal{D} \cup \mathcal{B}|_1^{\max} = |\mathcal{D}| + |L_1||\mathcal{D}|^2 \; . \tag{3.50}$$

Similar the worst–case size after the second iteration is,

$$\begin{aligned} |\mathcal{D} \cup \mathcal{B}|_2^{\max} &= |\mathcal{D} \cup \mathcal{B}|_1^{\max} + \Big(|\mathcal{D} \cup \mathcal{B}|_1^{\max} \Big)^2 && (3.51) \\ &= |\mathcal{D}| + |L_1||\mathcal{D}|^2 + |L_2| \Big(|L_1||\mathcal{D}|^2 + |\mathcal{D}| \Big)^2 && (3.52) \end{aligned}$$

In general, the worst–case for the size of the set of all components after k iterations is,

$$\begin{aligned} |\mathcal{D} \cup \mathcal{B}|_k^{\max} &= |\mathcal{D}| + \sum_{i=1}^{k} |B_i|^{\max} && (3.53) \\ &= |\mathcal{D}| + |L_1||\mathcal{D}|^2 + |L_2| \big(|L_1||\mathcal{D}|^2 + |\mathcal{D}| \big)^2 + \cdots \in \mathcal{O}(|\mathcal{D}|^{2^k}) && (3.54) \end{aligned}$$

The complexity of the runtime $t_{\text{bba}}^{\max}$ of the overall algorithm is then dominated by the runtime to recognize pairs in level n_L, yielding,

$$t_{\text{bba}}^{\max} \in \mathcal{O}(|\mathcal{N}| \cdot |\mathcal{D}|^{2 \cdot 2^{n_L}} \log |\mathcal{D}|^{2^{n_L}}) = \mathcal{O}(|\mathcal{N}| \cdot |\mathcal{D}|^{2^{n_L+1}} \cdot 2^{n_L} \cdot \log |\mathcal{D}|). \tag{3.55}$$

In the typical case, only a number of new components that is linear with the number of existing components is found by all three functions (see Eqs. (3.22), (3.36) and (3.49)). This results in,

$$\begin{aligned} |\mathcal{D} \cup \mathcal{B}|_k^{\text{typ}} &= |\mathcal{D}| + \sum_{i=1}^{k} |B_i|^{\text{typ}} \in |L_1| \mathcal{O}(|\mathcal{D}|) + |L_2| \mathcal{O}(|\mathcal{D}| + |\mathcal{D}|) + \cdots && (3.56) \\ \Rightarrow |\mathcal{D} \cup \mathcal{B}|_k^{\text{typ}} &\in \mathcal{O}(k \cdot |\mathcal{D}|) \; . && (3.57) \end{aligned}$$

The complexity for functions findPairs, findArrays and findChain is given by Eqs. (3.24), (3.35) and (3.48). The complexity of the runtime $t_{\text{bba}}^{\text{typ}}$ of the overall algorithm is then,

$$t_{\text{bba}}^{\text{typ}} \in \sum_{k=1}^{n_L} |L_k| \mathcal{O}((k-1) \cdot |\mathcal{D}| \log |\mathcal{D}|) = \mathcal{O}(n_L^2 \cdot |\mathcal{D}| \log |\mathcal{D}|) \; . \tag{3.58}$$

Overall, the algorithm shows polynomial behavior in the worst case and nearly linear behavior in the typical case if the library is fixed.

For the ESFG generation algorithm (Section 4.1) only the top–most level of the building block hierarchy is regarded. These building blocks represent complete functional blocks. These building blocks do not have parents, i.e.,

$$\mathcal{B}_{\text{top}} = \{b \in \mathcal{B} | b.\mathit{parents} \backslash \{\text{ds}, \text{gs}\} = \emptyset\}. \tag{3.59}$$

Differential stages and *Gilbert stacks* are ignored because these building blocks are only required to check the correctness of detected *differential pairs*. Two building blocks from $\mathcal{B}_{\text{top}}$ are disjoint, this means they do not share devices. The only exceptions are current mirrors that share an input stage. Overall, it can be said that,

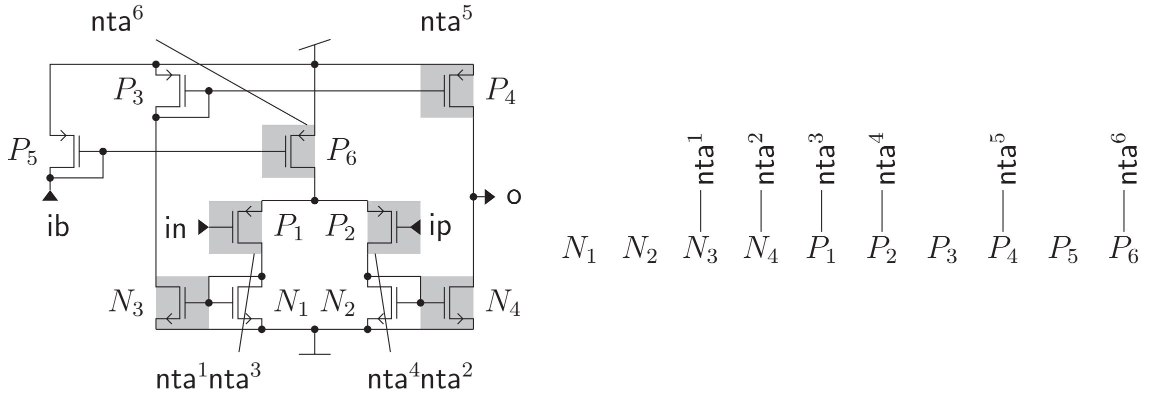
$$|\mathcal{B}_{\text{top}}| \leq |\mathcal{D}| \ . \tag{3.60}$$

The enhanced building block analysis method differs from the building block recognition of Massier (2010) (see Section 2.1.1) in the following points:

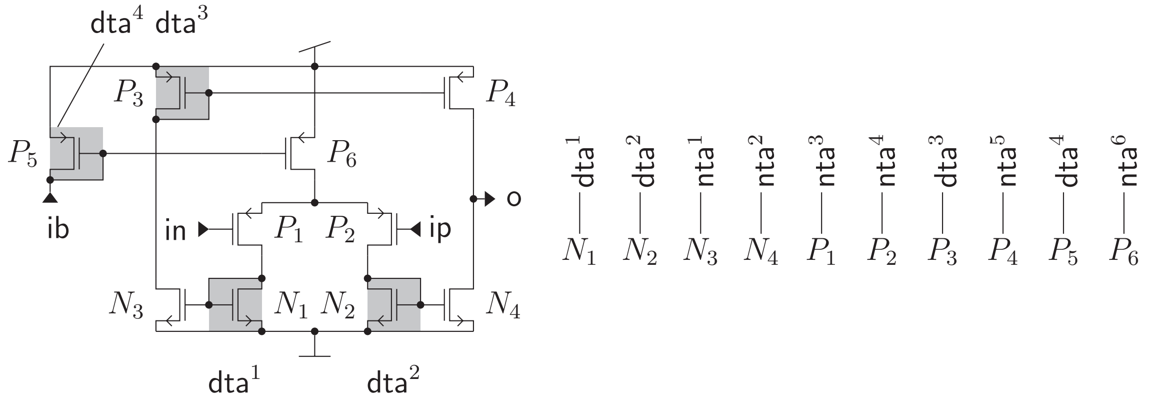
- The algorithm was extended by introducing the concepts of arrays and chains to handle digital building blocks. The library and dominance relation were extended accordingly.
- The library now contains the new hierarchy level 2 to detect *normal transistor arrays* and *diode transistor arrays*. This adopts the principle suggested by Arsintescu (1996) to detect *simple current mirrors* and *level shifters*. However, the library used in this work is more comprehensive. This principle is beneficial because less rules must be evaluated during the recognition of pairs.
- The recognition of pairs was speeded up because less candidate pairs are generated compared to previous approaches. The approach of Massier (2010) used all pairs of devices or building blocks of the correct type. The approach of Zizala (2001) used all pairs of devices or building blocks that are connected to the same net and one rule function per type. The new rule functions suggested in this thesis, only yield pairs that are connected to the same net by the correct pin and that are of correct type.
- The conflict resolution is performed after each hierarchy level. This has the advantage that the overall number of building blocks is kept low, which in turn results in another speed–up. Furthermore, the recognition of chains requires that wrong stacks are removed beforehand.

This is illustrated by the iterations of the enhanced algorithm for the symmetrical OTA from Sansen (2007, Silde 0711) shown in Table 3.1. The analog part of the library is used. Compared to Table 2.1, it contains an extra iteration for hierarchy level 2 to detect *normal transistor arrays* and *diode transistor arrays*. In hierarchy level 3 the

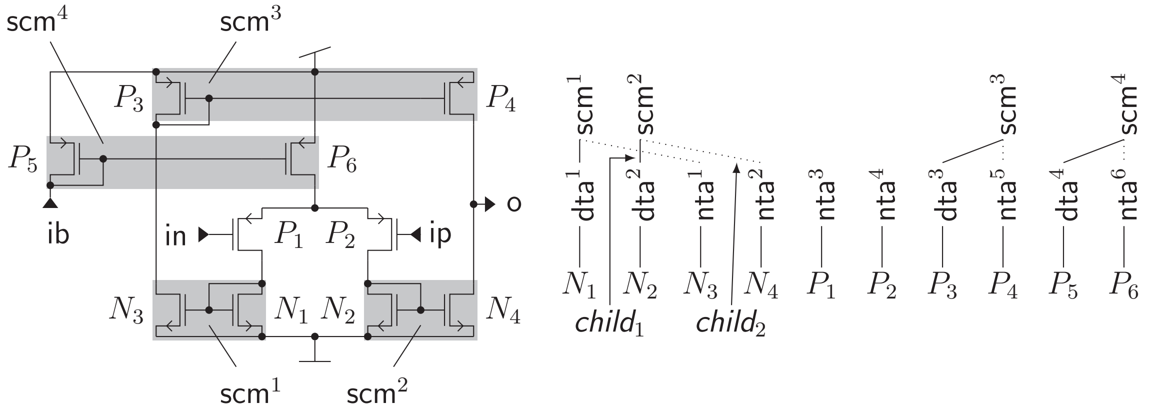
Hierarchy Level 2 — t=nta



Hierarchy Level 2 — t=dta



Hierarchy Level 3 — t=scm

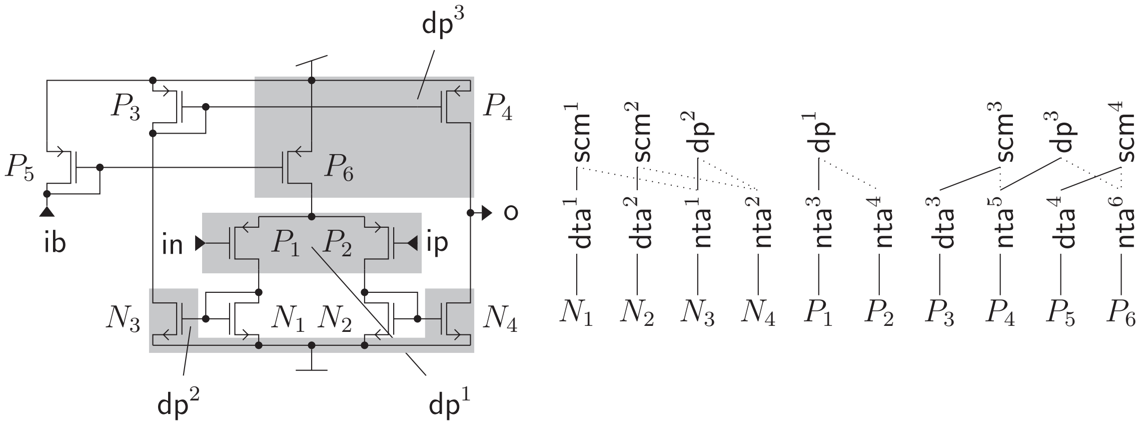


Continued on next page . . .

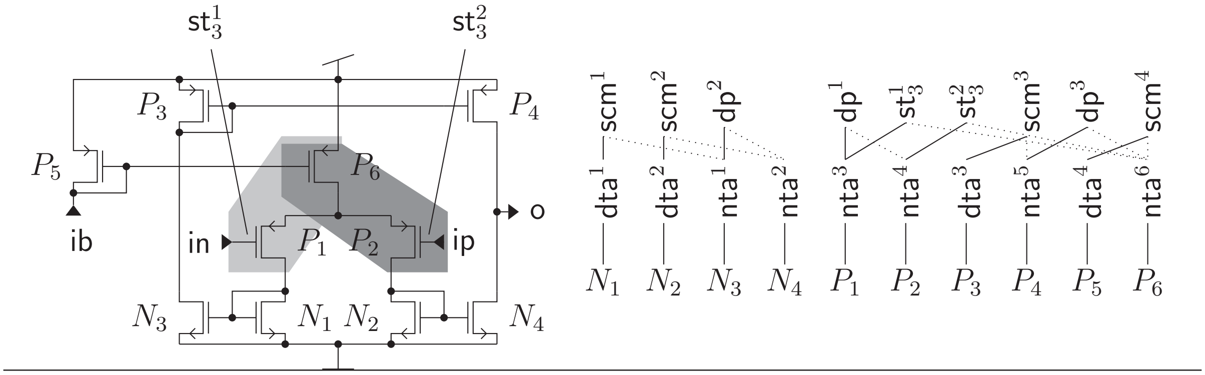
Table 3.1.: Flow of the enhanced algorithm for a symmetrical OTA (Sansen 2007, Silde 0711).

Continued from previous page ...

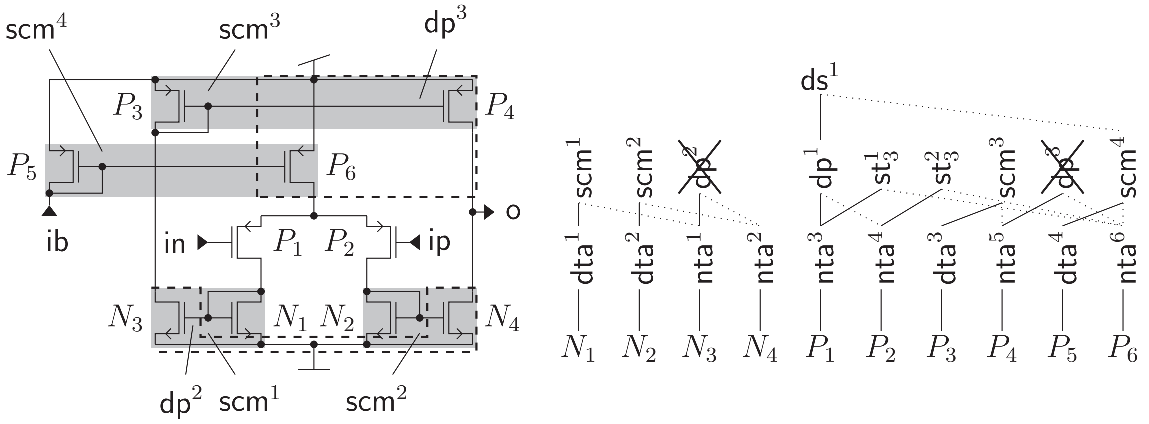
Hierarchy Level 3 — t=dp


dp^3
P_3
P_4
P_5
P_6
ib
in
P_1 P_2
ip
o
N_3
N_1 N_2
N_4
dp^2
dp^1
scm^1 scm^2 dp^2 dp^1 scm^3 dp^3 scm^4
dta^1 dta^2 nta^1 nta^2 nta^3 nta^4 dta^3 nta^5 dta^4 nta^6
N_1 N_2 N_3 N_4 P_1 P_2 P_3 P_4 P_5 P_6

Iteration Hierarchy Level 3 — t=st$_3$


st$_3^1$
st$_3^2$
P_3
P_4
P_5
P_6
ib
in
P_1 P_2
ip
o
N_3
N_1 N_2
N_4
scm^1 scm^2 dp^2 dp^1 st$_3^1$ st$_3^2$ scm^3 dp^3 scm^4
dta^1 dta^2 nta^1 nta^2 nta^3 nta^4 dta^3 nta^5 dta^4 nta^6
N_1 N_2 N_3 N_4 P_1 P_2 P_3 P_4 P_5 P_6

Hierarchy Level 3 — conflict resolution

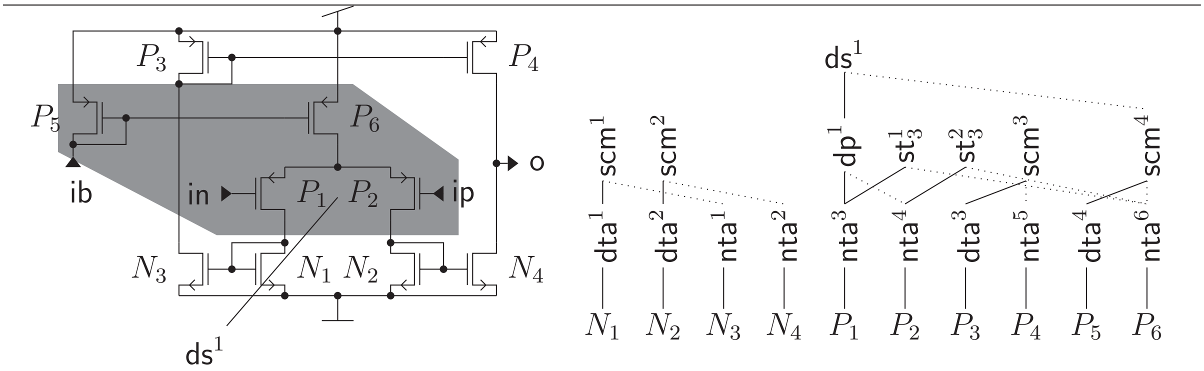

scm^4
scm^3
dp^3
P_3
P_4
P_5
P_6
ib
in
P_1 P_2
ip
o
N_3
N_1 N_2
N_4
dp^2 scm^1
scm^2
ds^1
scm^1 scm^2 dp^2 dp^1 st$_3^1$ st$_3^2$ scm^3 dp^3 scm^4
dta^1 dta^2 nta^1 nta^2 nta^3 nta^4 dta^3 nta^5 dta^4 nta^6
N_1 N_2 N_3 N_4 P_1 P_2 P_3 P_4 P_5 P_6

Continued on next page ...

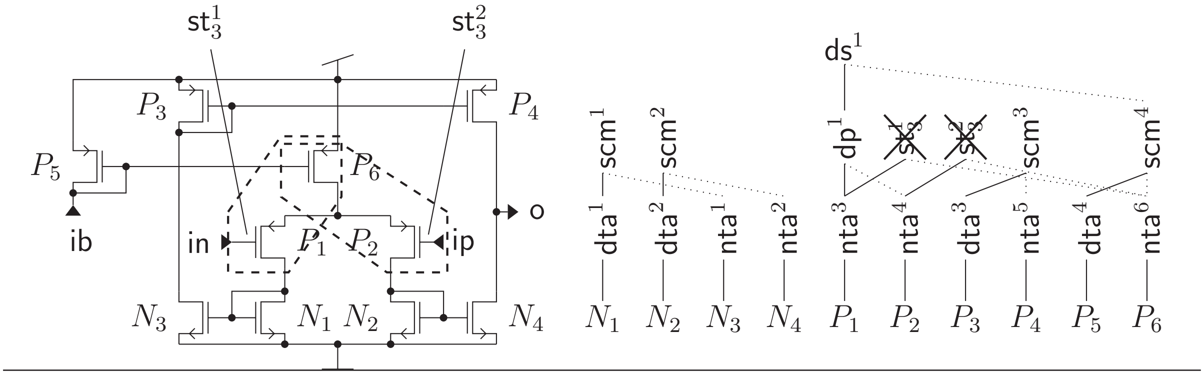
Table 3.1.: Flow of the enhanced algorithm for a symmetrical OTA (Sansen 2007, Silde 0711).

Continued from previous page ...

Hierarchy Level 7 — t=ds



removal of uncertain building blocks



final result

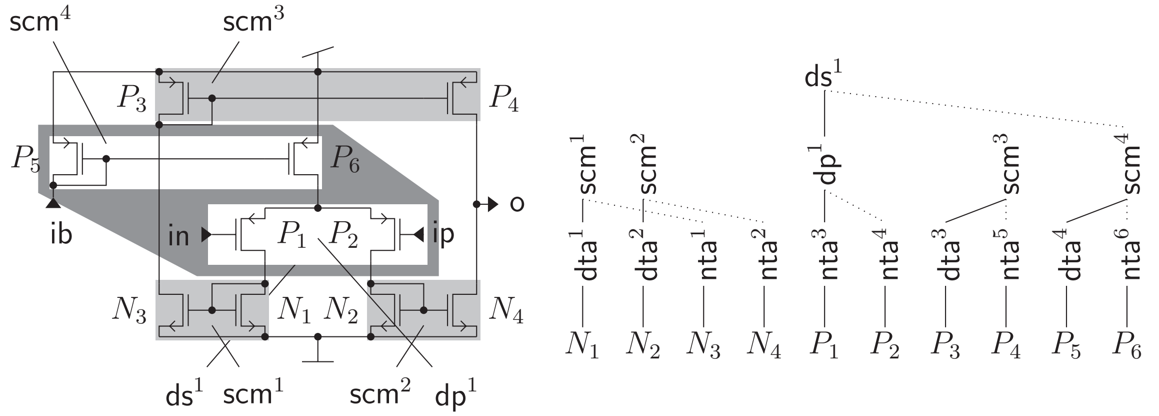


Table 3.1.: Flow of the enhanced algorithm for a symmetrical OTA (Sansen 2007, Silde 0711).

circuit		class		no. devices
analog	1	SE-OTA	(Laker & Sansen 1994)	22
	2	FD-OTA	(Galdi et al. 2008)	30
	3	FD-OTA	(Johns & Martin 1997, Cadence Design Systems 2010)	26
	4	FD-Mixer	(Lee 2004)	12
	5	SE-OTA	(Sansen 2007, Ohri & Callahan 1979, Roh et al. 2008)	12
	6	SE-OTA	(Vallee & Masry 1994)	17
	7	SE-OTA	(Laker & Sansen 1994)	18
digital		standard cells		2 – 96

Table 3.2.: Test cases for building block analysis.

cascode pair was renamed to *stack on level 3* (st$_3$) to allow integration with the digital library part. The wrongly detected *differential pair* dp^2 is immediately removed after hierarchy level 3. This is because the conflict resolution is run after each hierarchy level. No building blocks are found in hierarchy levels 4, 5 and 6. Therefore these levels are not shown in Table 3.1.

Some of the principles discussed in the context of graph matching for pattern recognition (Conte et al. 2004) can be found in the algorithm. This can be seen if components and nets are not regarded as objects but as nodes in a bipartite graph. In addition, a building block in the library would be represented by a so–called object graph and the circuit would be the subject graph. The algorithm can then be interpreted as two step approach. First, a characteristic node of the object graph is searched in the subject graph. Second, the algorithm tries to match the remaining object graph starting from the characteristic node. A strong relation exists also to the algorithm of Messmer & Bunke (1998). They suggest to decompose larger graphs in a preprocessing step and perform a hierarchical matching afterward. For the algorithm used here this preprocessing step is replaced by a knowledge–based decomposition. However, it is important to notice that none of these algorithms can handle arrays and chains.

3.4. Experimental Results

In the following, experimental results are shown for a set of test cases consisting of seven analog circuits and one digital standard cell library. After that, detailed results are shown for two of the analog test cases, a digital latch and two mixed–signal circuits.

Table 3.2 lists details about the test cases. The analog test cases consist of four single–ended (SE) operational transconductance amplifiers (OTA), two fully–differential (FD)

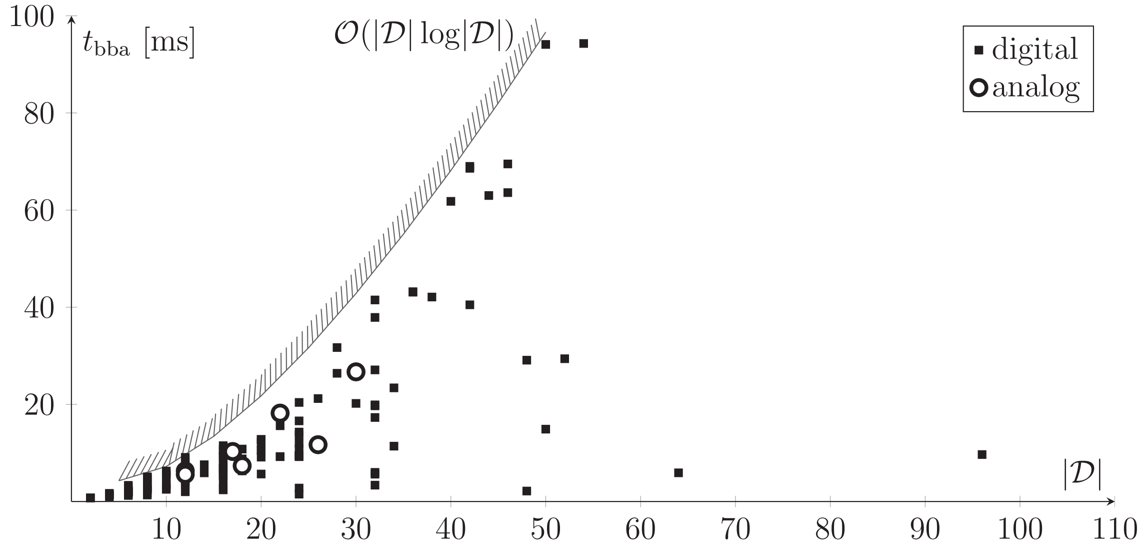
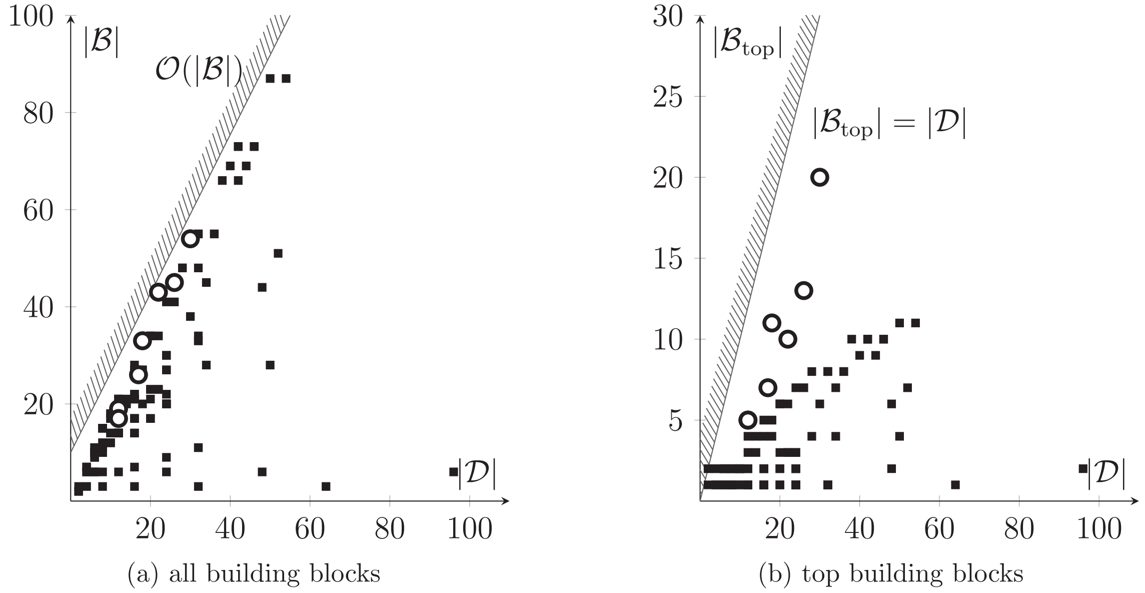


Figure 3.12.: Time needed for building block analysis with respect to circuit size.

OTAs and one fully–differential mixer. In particular, Circuit 1 is a single–ended, folded–cascode OTA from Laker & Sansen (1994). Circuit 2 is the fully–differential OTA presented by Galdi et al. (2008). Circuit 3 is the fully–differential OTA discussed in Johns & Martin (1997) and Cadence Design Systems (2010). Circuit 4 is a fully–differential Gilbert cell mixer from Lee (2004). Circuit 5 is a single–ended OTA with a negative resistance to increase gain (Sansen 2007, Ohri & Callahan 1979, Roh et al. 2008). It is depicted in Fig. 3.14. Circuit 6 is the complementary folded–cascode OTA from Vallee & Masry (1994). It is shown in Fig. 3.15. Circuit 7 is a symmetrical cascode OTA based on Laker & Sansen (1994).

The digital test case is the Nangate standard cell library, which consists of 134 standard cells. The library includes various combinatorial circuits, latches and registers of different driver strength. It can be seen that the number of transistors in these test circuit varies between 2 and 96. For the analog test cases netlists generated from the schematic are used together with the analog part of the library shown in Fig. 3.2. For the digital test cases netlists extracted from the layout are used together with the digital part of the library shown in Fig. 3.2.

Figure 3.12 shows a plot of the runtime required for building block analysis over the circuit size. The times were measured on a Intel® Core™2 Duo CPU running at 3.00 GHz. The analog circuits are marked by circles, while the library cells from the digital test case are marked by squares. It can be seen that there is a larger variation of the runtime for the same circuit size. This is because of the different internal structure of the cells. The runtime t_{bba} of the building block analysis appears to be bounded by $\mathcal{O}(|\mathcal{D}|\log|\mathcal{D}|)$. This corresponds to the derived typical complexity (Eq. (3.58)). It is significantly better than the derived worst–case complexity (Eq. (3.55)).



(a) all building blocks

(b) top building blocks

Figure 3.13.: Number of recognized building blocks with respect to circuit size.

Figure 3.13 shows the number of recognized building blocks in dependency of the circuit size. Figure 3.13a shows the final size of set $\mathcal{B}$ and Fig. 3.13b shows the size of set $\mathcal{B}_{\text{top}}$. Set $\mathcal{B}$ appears to grow linearly with the circuit size for the analog and digital test cases. This corresponds to Eq. (3.54). The size of set $\mathcal{B}_{\text{top}}$ of the test cases is always less than the theoretical bound $|\mathcal{B}_{\text{top}}| = |\mathcal{D}|$ (Eq. (3.59)). For the digital test case, the complexity appears to be better than linear. Thus, the ratio of $|\mathcal{B}_{\text{top}}|$ and $|\mathcal{D}|$ is higher for analog circuits than for digital circuits.

In the following, detailed results are presented for analog circuits 5 and 6. These were selected because they exhibit all typical analysis results.

Figure 3.14 shows detailed results for Circuit 5 from table Table 3.2. The circuit is based on Sansen (2007), Ohri & Callahan (1979) and Roh et al. (2008). It consists of four *simple current mirrors* scm^1 to scm^4, one *differential pair* dp^1 and one cross–coupled pair cc^1.

Figure 3.15 shows detailed results for Circuit 6 from Table 3.2, which is a complementary folded cascode amplifier (Vallee & Masry 1994). It consists of eight NMOS transistors N_1 to N_8, eight PMOS transistors P_1 to P_8 and one resistor R_1. The algorithm recognizes four *simple current mirrors* scm^1 to scm^4, two *differential pairs* dp^1 and dp^2, two *level shifters* ls^1 and ls^2 as well as *differential stages* ds^1 and ds^2 (Fig. 3.15a). In addition, the method recognizes *folded cascode pairs* fc^1 and fc^2 (Fig. 3.15b). These pairs suppress the recognition of a *cascode current mirror* ccm^1 consisting of scm^4 and ls^2. Similarly, *folded cascode pairs* (P_3, N_7) and (P_4, N_8) suppress a *cascode current mirror* consisting of scm^2 and ls^1.

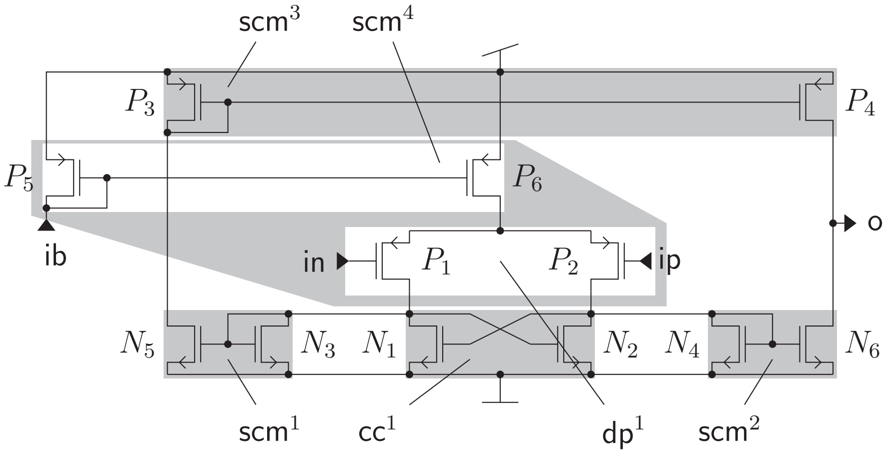


Figure 3.14.: Circuit 5: A modified symmetrical OTA (Sansen 2007, Ohri & Callahan 1979, Roh et al. 2008) and the recognized building blocks.

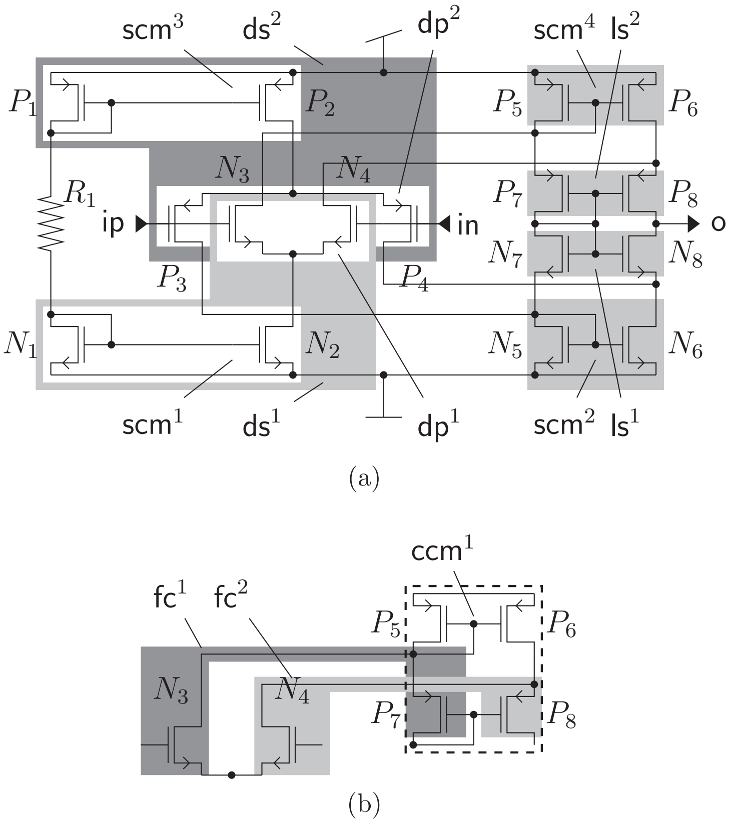


Figure 3.15.: Circuit 6: Complementary folded cascode amplifier (Vallee & Masry 1994) and recognized building blocks (a) and recognized *folded cascode pair* (b).

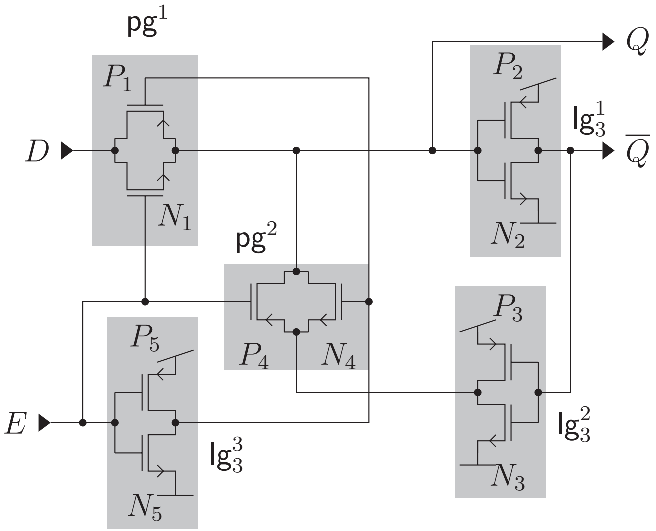


Figure 3.16.: Result of building block analysis for a latch (Weste & Harris 2005).

Figure 3.16 shows the result for a digital latch (Weste & Harris 2005). It consists of two *pass gates* pg^1 and pg^2 as well as three logic gates on level 3 lg_3^1 to lg_3^3. Further results for the structural analysis of complete standard cell libraries can be found in Section 6.4.

Figure 3.17 shows the result for a charge–pump. The circuit is based on Rhee (1999). It consists of two logic gates on level 2 lg_3^1 and lg_3^2, which are digital building blocks. It also consists of three *simple current mirrors* scm^1 to scm^3, which are analog building blocks. From the structural point of view, P_5,P_6 and N_6,N_7 form *differential pairs* but their inputs D and U are digital. Therefore, these *differential pairs* are suppressed by certain additional rules. Although, this information is not always available from the beginning, it can be obtained by the method for automatic identification of analog and digital part described in Section 4.2.

Figure 3.18 shows the result for a voltage–controlled ring oscillator (Retdian et al. 2002, Fig. 8). It consists of seven *simple current mirrors* scm^1 to scm^7 and five logic gates *logic gates on level 3* lg_3^1 to lg_3^5. From the structural point of view N_3, N_6 and P_3, P_6 could form *stacks on level 3* leading to a *logic gate on level 5* covering all four transistors. But this would contradict the analog functionality of N_3 and P_3. Therefore, the recognition of the *stacks on level 3* is prevented by the dominance relation from Fig. 3.4. The second child of a *stack on level 3* is dominated by the second child of a *simple current mirror*.

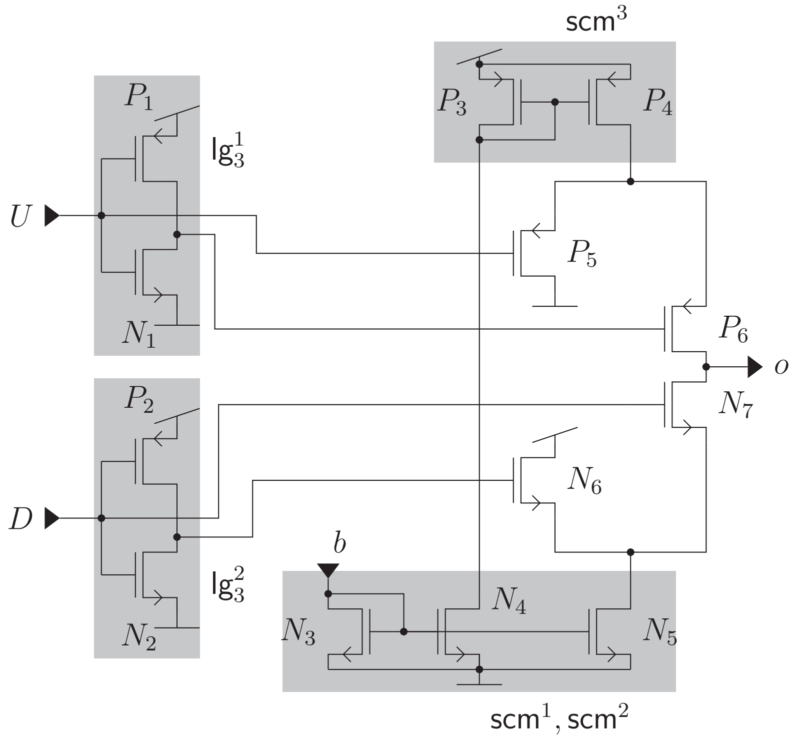


Figure 3.17.: Result of building block analysis for a charge–pump (Rhee 1999).

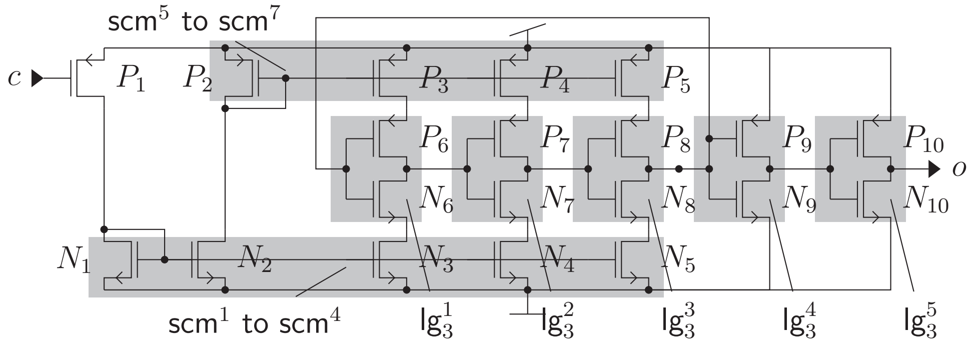


Figure 3.18.: Result of building block analysis for a voltage–controlled ring oscillator (Retdian et al. 2002, Fig. 8).

4. Structural Signal Path Analysis

This chapter describes the structural signal path analysis method. Figure 4.1 shows the overall flow. The method consists of five different steps and starts from the building blocks generated by the analysis presented in Chapter 3. First, the so–called enhanced structural signal flow graph (ESFG) is generated, which is a structural and qualitative behavioral model of the circuit. In case of a mixed–signal circuit, this enhanced structural signal flow graph (ESFG) is then used to automatically identify the analog and digital part. The analog part is then further subdivided into a core part, which does the signal processing, and a bias part, which provides bias voltages and currents. For the digital part, the true operating directions of pass gates are determined next. Finally, feedback loops in the ESFG are automatically broken up, leading to the temporal ESFG, which models time behavior in addition.

All of these steps are detailed in the following.

4.1. Generation of Enhanced Structural Signal Flow Graphs

Electronic circuits usually process some kind of signal which is a voltage or a current. The possible ways of signal propagation inside a given circuit are determined by its

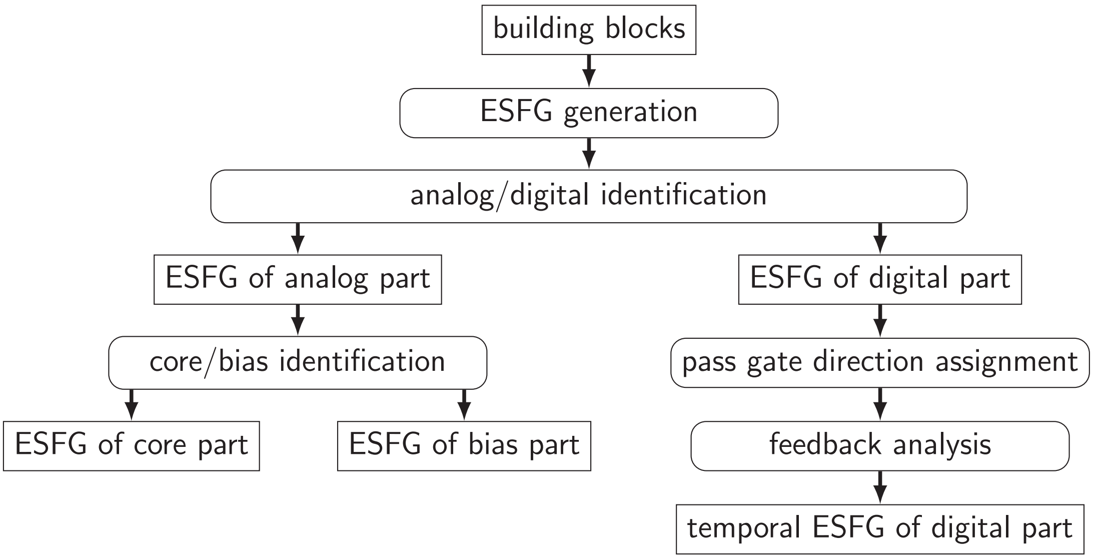


Figure 4.1.: Flow of structural signal path analysis method.

structure. ESFGs aim to model this relationship. They use the concept of signal flow graphs (Mason 1953) to model the qualitative behavior of the circuit. They extend the signal flow graphs by introducing structural attributes that are specific to integrated circuits. This and the next chapter will show that this unique combined structural and behavioral model allows to analyze the circuit to a large extent.

In this work qualitative approaches are preferred over quantitative approaches (see Section 2.2) out of the following reasons. Qualitative methods can be applied to circuits that can not be simulated because they are not sized properly. They are not limited to small–signal domain and avoid the accuracy–complexity trade–off.

In the following, ESFGs are formally defined and the generation method is described together with the corresponding library.

4.1.1. Formal Definition of ESFGs

The definition of an ESFG is based on the definition of terminals and the general definition of digraphs (see Appendix A.1).

Definition 4.1 (terminal)
A terminal connects a circuit block to external circuitry. The set of all terminals is denoted by $\mathcal{K}$. The set of all input terminals is $\mathcal{K}_i$, the set of all output terminals is $\mathcal{K}_o$ such that

$$\mathcal{K} = \mathcal{K}_i \cup \mathcal{K}_o \; . \tag{4.1}$$

Definition 4.2 (Enhanced structural signal flow graph (ESFG))
An ESFG is an attributed, directed graph $G_E = (N_{G_E}, E_{G_E}, \varphi_{G_E}, \alpha_{G_E})$ with the following properties.

1. Nets $\mathcal{N}$ and Terminals $\mathcal{K}$ of the circuit form the set of nodes

$$N_{G_E} \subseteq \mathcal{N} \cup \mathcal{K} \; . \tag{4.2}$$

2. The set of edges is denoted by E_{G_E}. An edge $e \in E_{G_E}$ models possible ways of signal propagation from its start node $\varphi^{-}_{G_E}(e)$ to its end node $\varphi^{+}_{G_E}(e)$,

$$\varphi^{-}_{G_E}(e) : E_{G_E} \rightarrow N_{G_E} \qquad \varphi^{+}_{G_E}(e) : E_{G_E} \rightarrow N_{G_E} \; . \tag{4.3}$$

The incidence function φ_{G_E} is

$$\varphi_{G_E}(e) = (\varphi^{-}_{G_E}(e), \varphi^{+}_{G_E}(e)) \; . \tag{4.4}$$

3. An edge points from each input terminal node $i \in \mathcal{K}_i$ to the associated net node $n_i \in \mathcal{N}$. An edge points to each output terminal node $o \in \mathcal{K}_o$ from the associated net node $n_o \in \mathcal{N}$.

4. Each edge $e \in E_{G_E}$ refers to a component or terminal x given by the component attribute function

$$\alpha^C_{G_E}(e) = x \qquad x \in \mathcal{D} \cup \mathcal{B} \cup \mathcal{K} \; . \tag{4.5}$$

5. The structural attributes of an edge e are given by function

$$\alpha^S_{G_E}(e) = \begin{cases} (t, s, p_1, p_2) & \varphi_{G_E}(e) \in \mathcal{N}^2 \\ \text{terminal} & \text{otherwise} \end{cases} \tag{4.6}$$
$$t \in T_\mathcal{D} \cup T_\mathcal{B} \quad s \in \{\text{—}, \text{n}, \text{p}\} \quad p_1, p_2 \in \alpha^C_{G_E}(e).pins$$

   For edges that connect two net nodes, attributes t and s denote type and subtype of the component represented by this edge, respectively. Attributes p_1 and p_2 represent the pins associated with this edge. For example if an edge e_1 represents the signal flow from the input i to the output o of a NMOS–*simple current mirror* then $\alpha^S_{G_E}(e_1) = (\text{scm}, n, i, o)$ holds. For edges connecting from or to a terminal node, the structure attribute has the special value *terminal*.

6. The attribute function for an edge e is composed from the component attribute function and the structural attribute function:

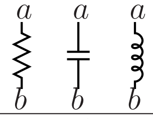
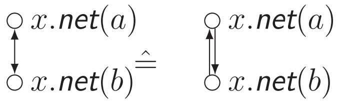
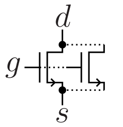
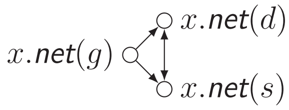
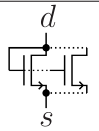
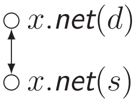
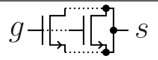
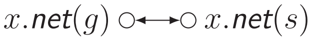
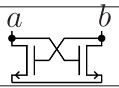
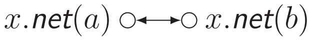
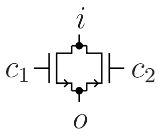
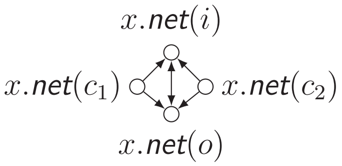
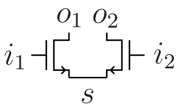
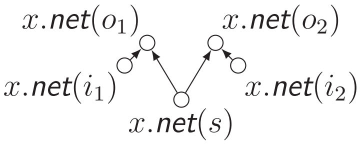
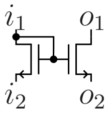
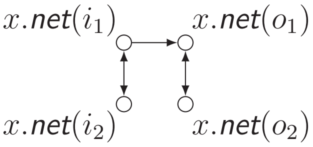
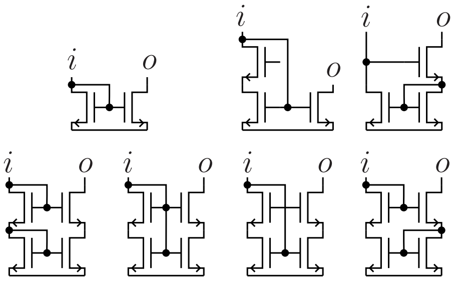
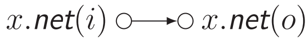
$$\alpha_{G_E}(e) = (\alpha^C_{G_E}(e), \alpha^S_{G_E}(e)) \; . \tag{4.7}$$

   This partitioning is required for the algorithm to identify analog and digital circuit parts (Section 4.2), the pass gate direction assignment (Section 4.4) and the symmetry computation algorithm (Chapter 5).

7. Two edges e_1 and e_2 are equal if they connect the same nodes and have the same attributes, i.e.,

$$(e_1 = e_2) :\Leftrightarrow \big(\varphi_{G_E}(e_1) = \varphi_{G_E}(e_2)\big) \wedge \big(\alpha_{G_E}(e_1) = \alpha_{G_E}(e_2)\big) \tag{4.8}$$

   This implies that two edges e_1, e_2 can connect the same two nodes in parallel if $\alpha_{G_E}(e_1) \neq \alpha_{G_E}(e_2)$.

		type of building block x	Sub-ESFG for x
analog or digital	R/ C/ L		 x.net(a) x.net(b) ≙ x.net(a) x.net(b)
	nta		
	dta		
	cta		
	cc		
	pg		 x.net(i) x.net(c_1) x.net(c_2) x.net(o)
analog	dp		 x.net(o_1) x.net(o_2) x.net(i_1) x.net(i_2) x.net(s)
	ls		 x.net(i_1) x.net(o_1) x.net(i_2) x.net(o_2)
	scm/ wsm/ wcm/ ccm/ wscm/ iwcm		

Continued on next page ...

Table 4.1.: Library of building blocks and corresponding ESFG.

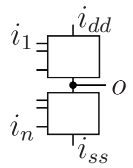
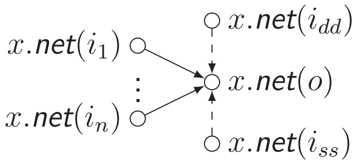
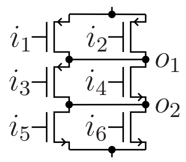
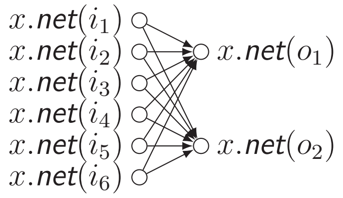
		type of building block x	Sub-ESFG for x
		Continued from previous page ...	
digital	lg		 $x.net(i_{dd})$ $x.net(i_1)$ $x.net(o)$ $x.net(i_n)$ $x.net(i_{ss})$
	tcb	 i_1 i_2 o_1 i_3 i_4 o_2 i_5 i_6	 $x.net(i_1)$ $x.net(i_2)$ $x.net(o_1)$ $x.net(i_3)$ $x.net(i_4)$ $x.net(i_5)$ $x.net(o_2)$ $x.net(i_6)$

Table 4.1.: Library of building blocks and corresponding ESFG.

4.1.2. Library–based Generation

In the following it is assumed that every building block from the building block library (Fig. 3.2) has a typical signal flow, which can be stored in a second library. Thus a sub-ESFG can be generated for each recognized building block. For the ESFG of the complete circuit, only components $b \in \mathcal{B}_{\text{top}}$ are considered, where $\mathcal{B}_{\text{top}}$ is defined according to Eq. (3.59). The ESFG of the complete circuit is generated by combining the subgraphs $G_{E,b}$ for all building blocks b in $\mathcal{B}_{\text{top}}$ and subgraphs $G_{E,k}$ for all terminals in $\mathcal{K}$ as follows,

$$G_E = \bigcup_{b \in \mathcal{B}_{\text{top}}} G_{E,b} \cup \bigcup_{k \in \mathcal{K}} G_{E,k} \,. \tag{4.9}$$

The graph union operation is defined according to Definition A.7.

The library of sub-ESFGs for all building blocks of the library shown in Fig. 3.2 is shown in Table 4.1. The library does only contain sub-ESFGs for building blocks that can occur in set $\mathcal{B}_{\text{top}}$ of a valid recognition result. Among others, it does not contain entries for the *differential stage* and *Gilbert stack* because they are explicitly not part of set $\mathcal{B}_{\text{top}}$ (Eq. (3.59)). A device k, which is a *resistor* (R), *capacitor* (C) or *inductor* (L), is modeled using two anti–parallel edges (drawn as one bidirectional edge) between the nets $x.net(a)$, $x.net(b)$ connecting to pins a, b, respectively. A *normal transistor array* (nta) can transmit signals from gate to drain and source and between drain and source. Thus, it is represented by edges from the gate to drain and source and by two anti–parallel edges between drain and source. Similarly, a *diode transistor array* (dta) is represented by two anti–parallel edges between drain and source. For a *capacitor transistor array* (cta), an additional edge is added from drain/source pin s to the gate because it acts as capacitor. Consequently, the

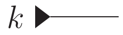
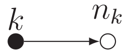
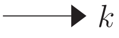
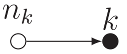
type of terminal k		Sub–ESFG for k
input		
output		

Table 4.2.: Sub-ESFGs for circuit terminals.

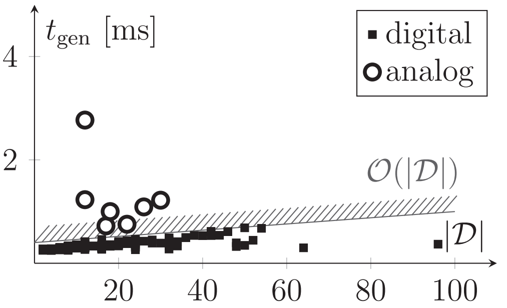


Figure 4.2.: Time needed for ESFG generation with respect to circuit size.

assumption that the signal only flows from gate to drain or source is no longer true. The *dummy transistor array* (uta) is not in the library because it has no function in the circuit and therefore no signal flow. A *cross–coupled pair* (cc) is represented by two anti–parallel edges between pins a and b. The sub-ESFG for a *pass gate* (pg) is the combination of the sub-ESFGs for a PMOS and an NMOS *normal transistor array*. An analog *differential pair* (dp) transmits signals from the inputs i_1 and i_2 to outputs o_1 and o_2, respectively, as well as from pin s to both outputs. Theoretically, there is a signal flow from i_1 to o_2 and from i_2 to o_1. But modeling this signal flow brings no benefit for the subsequent methods working on the graph because neither the symmetry nor the border between core and bias part does change. An analog *level shifter* (ls) transmits signals along drain and source of both transistors and from input i_1 to output o_1. All types of analog *current mirrors* are modeled by an edge from input i to output o. A digital *logic gate* (lg) is represented by edges from all inputs i_1 to i_n to the output o. In case the logic gate is not directly connected to the supply rails, there are additional edges from supply inputs i_{dd} and i_{ss} to output o. For the digital *tristate control block* (tcb) a signal flow exists from every input to very output.

The library of sub–ESFGs for terminals are shown in Table 4.2. An input terminal is represented by an edge from a terminal k to the corresponding net n_k. An output terminal is represented by an edge from the corresponding net n_k to a terminal node k.

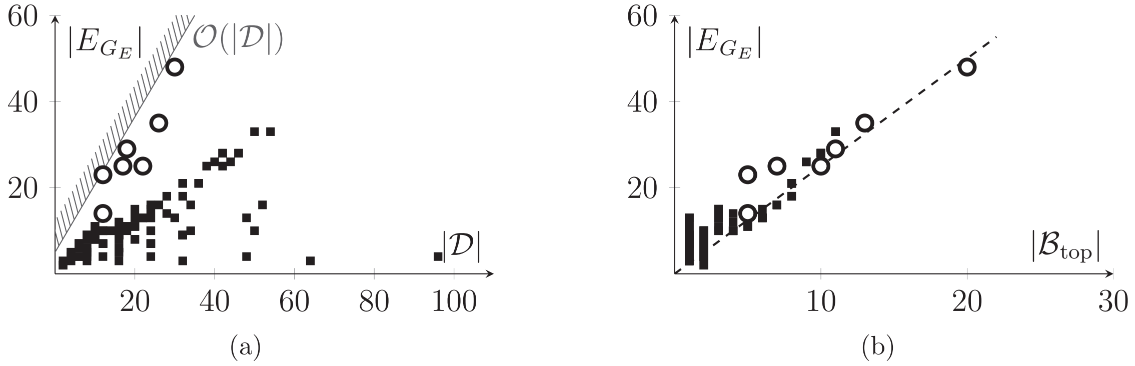


Figure 4.3.: Number of generated edges with respect to circuit size (a) and number of top building blocks (b).

4.1.3. Discussion

Although ESFGs are based on the general concept of signal flow graphs (Mason 1953), there are important differences to the approaches based on signal flow graphs that are discussed in Section 2.2. All other approaches aim for a quantitative model. Thus, the edges are annotated with symbolic information about device properties. Our model is purely structural and qualitative. Therefore, the edges are annotated with structural information only. A second difference is that the nodes represent nets in our approach but represent variables in all other approaches except Wei & Doboli (2008). Compared to the patent of Zhang et al. (2008) the graph is constructed from building blocks and not directly from transistors. This has the benefit of a smaller graph, because some building blocks like current mirrors can be represented by one edge only. In addition, the ESFG has structural attributes that are used to guide the algorithms presented later on, e.g., the identification of analog and digital part in Section 4.2 and the symmetry computation in Chapter 5.

Concerning the complexity of the algorithm, the following considerations can be made. The size of the ESFG G_E can be estimated as follows,

$$|N_{G_E}| \leq |\mathcal{N}| + |\mathcal{K}| \qquad |E_{G_E}| \in \mathcal{O}(|\mathcal{B}_{\text{top}}|) = \mathcal{O}(|\mathcal{D}|)\ . \tag{4.10}$$

Since every node is exactly one net or terminal, the number of nodes is limited by the number of nets and terminals. Since the sub–ESFGs for every building block $b \in \mathcal{B}_{\text{top}}$ are fixed trough the library, a fixed number of edges is generated for every b resulting in linear complexity with the number of devices. The worst–case runtime t_{gen} for the ESFG generation has linear complexity with the number of devices and terminals as follows,

$$t_{\text{gen}}^{\max} \in \mathcal{O}(|\mathcal{D}| + |\mathcal{K}|)\ . \tag{4.11}$$

4.1.4. Experimental Results

In the following, experimental results for the runtime of the algorithm are presented. After that detailed results are shown for a pure analog case, for a pure digital case and two mixed–signal cases.

Figure 4.2 shows the runtime of the algorithm for the test cases from Table 3.2. For the digital test case, the runtime appears to be bounded linearly, which corresponds to Eq. (4.11). No clear regularity can be observed for the investigated analog test cases, probably the test cases are too small and too few. It can also be observed that the time needed for analog circuits is slightly larger than the time needed for digital circuits. The reasons for that are revealed by Fig. 4.3. It shows the size of the edge set E_{G_E} versus the number of devices $|\mathcal{D}|$ (Fig. 4.3a) and building blocks $|\mathcal{B}_{\text{top}}|$ (Fig. 4.3b). The number of edges appears to be linearly bounded by the number of devices. It can be seen that the number of generated edges per device is higher for analog circuits. This is because there are more top building blocks generated per device for analog circuits. Consequently, the relation between $|E_{G_E}|$ and $|\mathcal{B}_{\text{top}}|$ appears to be linear.

An example for the ESFG generation is depicted in Fig. 4.4. The ESFG generated for the circuit shown in Fig. 4.4a consists of port nodes ip, in, ib and o, net nodes n_1 to n_4, n_{ip}, n_{in}, n_{ib} and n_o. The attributes for edges e_1 to e_{10} are shown in Fig. 4.4c.

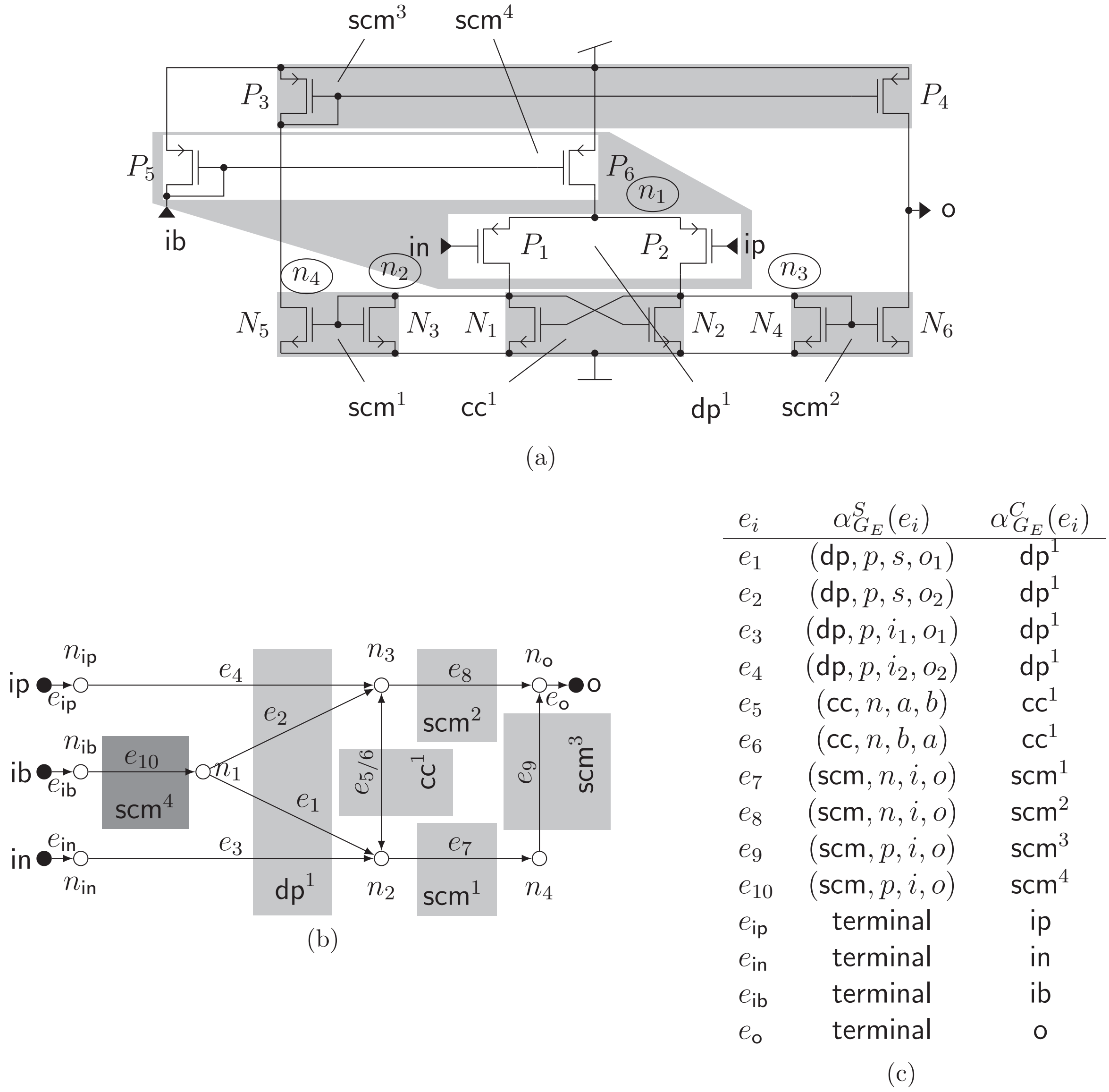
Differential pair dp^1 generates edges e_1 to e_4, thus the component attribute function is dp^1. The structural attribute of edge e_1 is (dp, p, s, o_1) because dp^1 is a *differential pair* with PMOS subtype and the edge represents the connection from s to o_1. Similar applies for edges e_2 to e_4. Anti–parallel edges e_5 and e_6 are generated by cross–coupled pair cc^1. Edges e_7 to e_{10} are generated by *simple current mirrors* scm^1 to scm^4, respectively. Edges e_{ip}, e_{in} and e_{ib} model input terminals *ip*, *in* and *ib*, respectively. Edge e_o models output terminal *o*.

Figure 4.5 shows the ESFG for the latch circuit from Fig. 3.16. Edges e_1 to e_5 represent the *logic gates*. Edges e_{10} to e_{15} and e_{20} to e_{25} represent the two *pass gates*.

Figure 4.6 shows the ESFG for the charge–pump shown in Fig. 3.17. Edges e_1 to e_3 represent the *simple current mirrors*. Edges e_6 to e_{12} represent the analog switches. Edges e_{13} and e_{14} represent the *logic gates*.

Figure 4.7 shows the ESFG of the voltage–controlled ring oscillator shown in Fig. 3.18. Edge e_1 represents the input transistor. Edges e_2 to e_8 represent scm^1 to scm^7. Edges e_9, e_{10} and e_{11} represent lg_3^1. Edges e_{12}, e_{13} and e_{14} represent lg_3^2. Edges e_{15}, e_{16} and e_{17} represent lg_3^3. Edges e_{18} and e_{19} represent lg_3^4 and lg_3^5, respectively.

Up to now, analog and digital building blocks are known, but it is not known whether the nodes in the graph represent analog or digital signals. This makes it impossible



(a)

(b)

e_i	$\alpha^S_{G_E}(e_i)$	$\alpha^C_{G_E}(e_i)$
e_1	(dp, p, s, o_1)	dp^1
e_2	(dp, p, s, o_2)	dp^1
e_3	$(\mathsf{dp}, p, i_1, o_1)$	dp^1
e_4	$(\mathsf{dp}, p, i_2, o_2)$	dp^1
e_5	(cc, n, a, b)	cc^1
e_6	(cc, n, b, a)	cc^1
e_7	(scm, n, i, o)	scm^1
e_8	(scm, n, i, o)	scm^2
e_9	(scm, p, i, o)	scm^3
e_{10}	(scm, p, i, o)	scm^4
e_{ip}	terminal	ip
e_{in}	terminal	in
e_{ib}	terminal	ib
e_{o}	terminal	o

(c)

Figure 4.4.: ESFG for Circuit 5 from Fig. 3.14 (analog): (a) Schematic with building blocks. (b) Generated ESFG. (c) Attributes of edges.

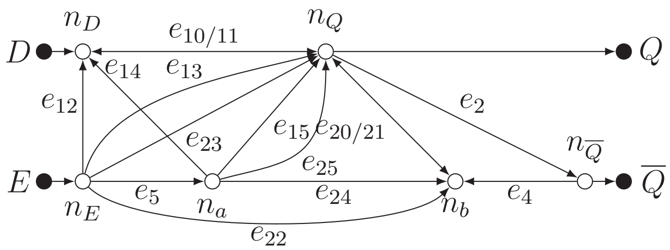


Figure 4.5.: ESFG of the digital latch from Fig. 3.16.

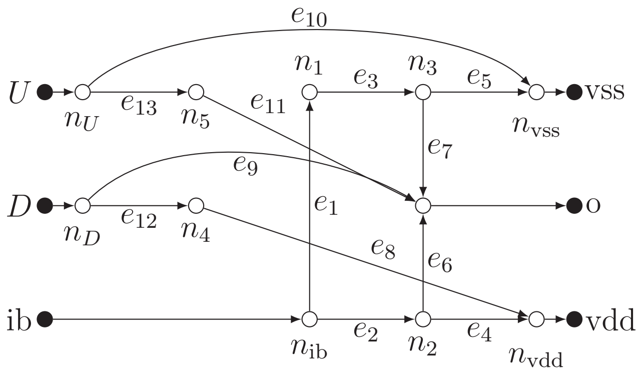


Figure 4.6.: ESFG of the charge–pump from Fig. 3.17 (mixed–signal).

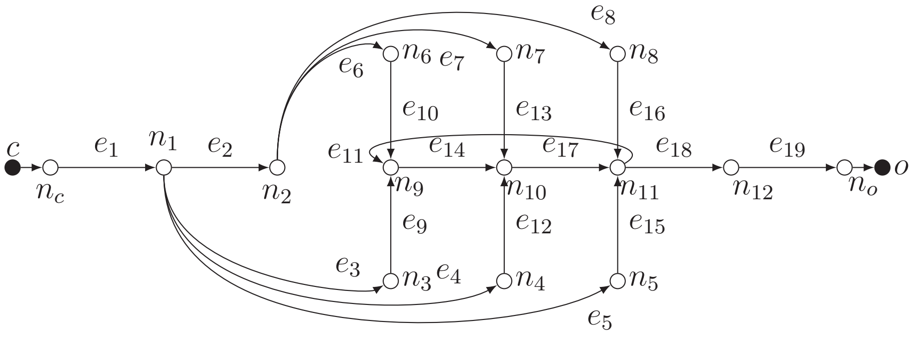


Figure 4.7.: ESFG of the voltage–controlled ring oscillator from Fig. 3.18 (mixed–signal).

to determine the logic function of the digital circuit parts. Therefore an algorithm to identify the analog and digital part of the circuit is presented next.

4.2. Automatic Identification of Analog and Digital Part

The pass–gate direction assignment algorithm in Section 4.4, the feedback analysis in Section 4.5 and the logic function extraction in Chapter 6 are specialized analyses for digital circuits. The algorithm in Section 4.3 to identify core and bias part is a specialized analysis for analog circuits. If a circuit is not purely analog or purely digital but consists of an analog and a digital part, these parts are identified beforehand to apply the algorithms listed above. In the following, an algorithm that identifies the analog and digital part of the ESFG is presented. In addition, the algorithm detects the interface part that represents the transition between analog and digital signals. The algorithm is based on the structural information about the building blocks of the circuit stored in the ESFG. The method uses the ESFG to set up a constraint satisfaction problem (CSP). A CSP consists of a set of Boolean variables, integer variables or set variables and a set of constraints that must be satisfied by the variables. Possible constraints are predicate logic expressions as well as linear and specific non–linear equalities and inequalities. The methods to solve a CSP are called constraint programming.

The following CSP partitions the ESFG G_E in an analog ESFG $G_{E,A}$, a digital ESFG $G_{E,D}$ and an interface ESFG $G_{E,I}$.

$$\begin{aligned} & G_E = G_{E,A} \cup G_{E,I} \cup G_{E,D} \\ \wedge\ & N_{G_{E,A}} \cap N_{G_{E,D}} = \emptyset \\ \wedge\ & \forall_{e \in E_{G_{E,I}}} \varphi_{G_{E,I}}(e) \in \left(N_{G_{E,A}} \times N_{G_{E,D}} \cup N_{G_{E,D}} \times N_{G_{E,A}}\right) \\ \wedge\ & (\forall_{k \in \mathcal{K}_A} k \in N_{G_{E,A}}) \wedge (\forall_{k \in \mathcal{K}_D} k \in N_{G_{E,D}}) \\ \wedge\ & \forall_{e \in E_{G_E}} c(e, N_{G_{E,A}}, N_{G_{E,D}}) \end{aligned} \tag{4.12}$$

The first three constraints describe the partitioning of G_E. Subgraphs $G_{E,A}$ and $G_{E,D}$ must not share any node. Subgraph $G_{E,I}$ has the property that every edge connects a node of $G_{E,A}$ with a node of $G_{E,D}$. The fourth constraint forces analog terminals $k \in \mathcal{K}_A$ to be part of the analog graph and digital terminals $k \in \mathcal{K}_D$ to be part of the digital graph. The last constraint sets up one or more constraints per edge using Boolean function c. Depending on the structural attributes $\alpha^S_{G_E}(e)$ of edge e the function corresponds to one of the four generic functions presented in the following.

1. **fixed analog end**: the end node of the edge e must be in the analog subgraph,

$$c_{\text{fixed–analog–end}}(e, N_{G_{E,A}}, N_{G_{E,D}}) :\Leftrightarrow (\varphi^+_{G_E}(e) \in N_{G_{E,A}}) \ . \tag{4.13}$$

2. **fixed digital end**: the end node of the edge e must be in the digital subgraph,

$$c_{\text{fixed–digital–end}}(e, N_{G_{E,A}}, N_{G_{E,D}}) :\Leftrightarrow (\varphi^+_{G_E}(e) \in N_{G_{E,D}}) \ . \tag{4.14}$$

structural attribute $\alpha^S_{G_E}(e)$	$c(e, N_{G_{E,A}}, N_{G_{E,D}})$
pass gate	
(pg, —, i, o), (pg, —, o, i)	$c_{\text{prop}}(e, N_{G_{E,A}}, N_{G_{E,D}})$
(pg, —, $c_1/c_2, i/o$)	$c_{\text{null}}(e, N_{G_{E,A}}, N_{G_{E,D}})$
normal transistor array	
(nta, n/p, sc, dn), (nta, n/p, dn, sc)	$c_{\text{prop}}(e, N_{G_{E,A}}, N_{G_{E,D}})$
(nta, n/p, gt, dn), (nta, n/p, gt, sc)	$c_{\text{null}}(e, N_{G_{E,A}}, N_{G_{E,D}})$
all pure analog building blocks	
pure analog types	$c_{\text{fixed–analog–end}}(e, N_{G_{E,A}}, N_{G_{E,D}})$
all pure digital building blocks	
pure digital types	$c_{\text{fixed–digital–end}}(e, N_{G_{E,A}}, N_{G_{E,D}})$
terminals	
terminal	$c_{\text{prop}}(e, N_{G_{E,A}}, N_{G_{E,D}})$

Table 4.3.: Mapping from the structural attribute of an edge e to the corresponding generic constraint type.

3. **propagation**: start and end node of the edge e must be in the same subgraph,

$$\begin{aligned} c_{\text{prop}}(e, N_{G_{E,A}}, N_{G_{E,D}}) :\Leftrightarrow & \Big((\varphi^+_{G_E}(e) \in N_{G_{E,A}}) \leftrightarrow (\varphi^-_{G_E}(e) \in N_{G_{E,A}}) \Big) \\ \wedge & \Big((\varphi^+_{G_E}(e) \in N_{G_{E,D}}) \leftrightarrow (\varphi^-_{G_E}(e) \in N_{G_{E,D}}) \Big) \; . \end{aligned} \tag{4.15}$$

4. **null**: there is no condition for this edge,

$$c_{\text{null}}(e, N_{G_{E,A}}, N_{G_{E,D}}) :\Leftrightarrow 1 \; . \tag{4.16}$$

Table 4.3 shows the correspondence of these four generic functions and function c for a specific edge e. For edges representing a *pass gate* (pg) the following holds. Start and end node of the edges running from pin i to pin o and vice versa must be in the same subgraph (Eq. (4.15)). No conditions exists for the edges running from the control inputs c_1 and c_2 to i and o.

For edges e representing a *normal transistor array* (nta) the following holds. Start and end nodes of edges between drain and source must be in the same subgraph. No conditions exists for the edges running from the gate to drain and source. In case edge e represents a pure analog building block the output must always be analog. In case edge e represents a pure digital building block the output must always be digital.

4.2.1. Examples

The following constraints are generated for the charge–pump example from Fig. 3.17 and the ESFG from Fig. 4.6.

$$
\begin{aligned}
& G_E = G_{E,A} \cup G_{E,I} \cup G_{E,D} \\
& \wedge\ N_{G_{E,A}} \cap N_{G_{E,D}} = \emptyset \\
& \wedge\ \forall_{e \in E_{G_{E,I}}} \varphi_{G_{E,I}}(e) \in \left(N_{G_{E,A}} \times N_{G_{E,D}} \cup N_{G_{E,D}} \times N_{G_{E,A}}\right) \\
& \wedge (U \in N_{G_{E,D}}) \wedge (D \in N_{G_{E,D}}) \wedge (\mathrm{ib} \in N_{G_{E,A}}) \wedge (o \in N_{G_{E,A}}) \\
& \wedge c_{\text{fixed–analog–end}}(e_1, N_{G_{E,A}}, N_{G_{E,D}}) \wedge c_{\text{fixed–analog–end}}(e_2, N_{G_{E,A}}, N_{G_{E,D}}) \\
& \wedge c_{\text{fixed–analog–end}}(e_3, N_{G_{E,A}}, N_{G_{E,D}}) \wedge c_{\text{prop}}(e_4, N_{G_{E,A}}, N_{G_{E,D}}) \\
& \wedge c_{\text{prop}}(e_5, N_{G_{E,A}}, N_{G_{E,D}}) \wedge c_{\text{prop}}(e_6, N_{G_{E,A}}, N_{G_{E,D}}) \\
& \wedge c_{\text{prop}}(e_7, N_{G_{E,A}}, N_{G_{E,D}}) \wedge c_{\text{null}}(e_8, N_{G_{E,A}}, N_{G_{E,D}}) \\
& \wedge c_{\text{null}}(e_9, N_{G_{E,A}}, N_{G_{E,D}}) \wedge c_{\text{null}}(e_{10}, N_{G_{E,A}}, N_{G_{E,D}}) \\
& \wedge c_{\text{null}}(e_{11}, N_{G_{E,A}}, N_{G_{E,D}}) \wedge c_{\text{fixed–digital–end}}(e_{12}, N_{G_{E,A}}, N_{G_{E,D}}) \\
& \wedge c_{\text{fixed–digital–end}}(e_{13}, N_{G_{E,A}}, N_{G_{E,D}}) \wedge c_{\text{prop}}(e_U, N_{G_{E,A}}, N_{G_{E,D}}) \\
& \wedge c_{\text{prop}}(e_D, N_{G_{E,A}}, N_{G_{E,D}}) \wedge c_{\text{prop}}(e_{\mathrm{ib}}, N_{G_{E,A}}, N_{G_{E,D}}) \\
& \wedge c_{\text{prop}}(e_o, N_{G_{E,A}}, N_{G_{E,D}})
\end{aligned} \tag{4.17}
$$

Edges e_1, e_2 and e_3 are generated by *simple current mirrors* (scm), which are analog building blocks. Therefore the end nodes n_1, n_2 and n_3 of these edges must be part of the analog subgraph. Edges e_4 to e_7 represent drain–source channels, therefore a propagate constraint is created for these edges. The corresponding gate–drain connections represented by edges e_8 to e_{11} create a null constraint. Edges e_{12} and e_{13} represent logic gates, therefore the end nodes n_4 and n_5 are constrained to be digital. Edges e_D, e_U, e_{ib}, e_o represent the edges connecting port nodes to net nodes. For these edges a propagate constraint is created.

The solution of this problem is depicted in Fig. 4.8. The analog ESFG $G_{E,A}$ consists of edges e_1 to e_7, the interface ESFG $G_{E,I}$ consists of edges e_8 to e_{11} and the digital part consists of edges e_{12} and e_{13}.

Figure 4.9 shows the identified analog, digital and interface part for the voltage–controlled ring oscillator from Fig. 3.18. The analog sub–ESFG $G_{E,A}$ consists of edges e_1 to e_8 representing the input transistor and the *simple current mirrors*. The interface sub–ESFG $G_{E,I}$ consists of edges e_9, e_{10}, e_{12}, e_{13}, e_{15} and e_{16} which model the supply inputs of lg_3^1, lg_3^2 and lg_3^3. The interface sub–ESFG $G_{E,D}$ consists of edges e_{11}, e_{14}, e_{17}, e_{18} and e_{19} modeling lg_3^1 to lg_3^5.

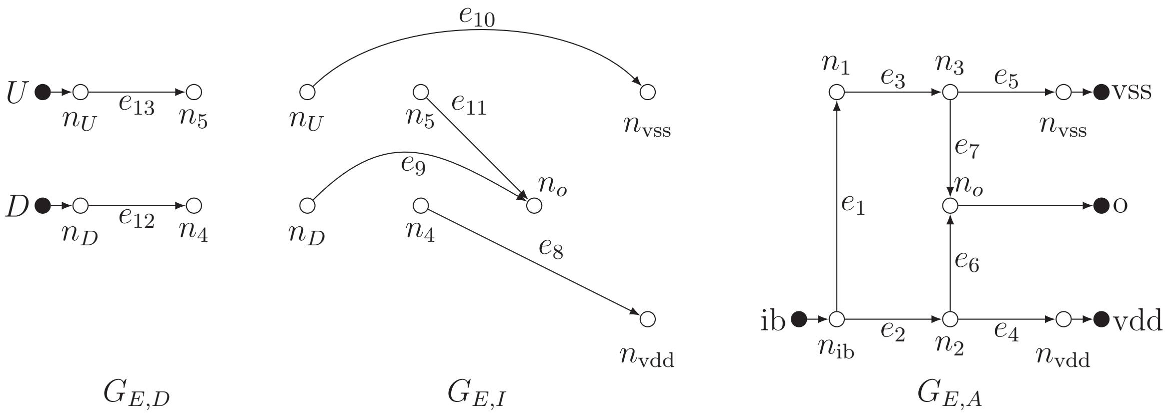


Figure 4.8.: Resulting digital sub–ESFG $G_{E,D}$, interface sub–ESFG $G_{E,I}$ and analog sub–ESFG $G_{E,A}$ of the ESFG (Fig. 4.6) of the charge–pump (Fig. 3.17).

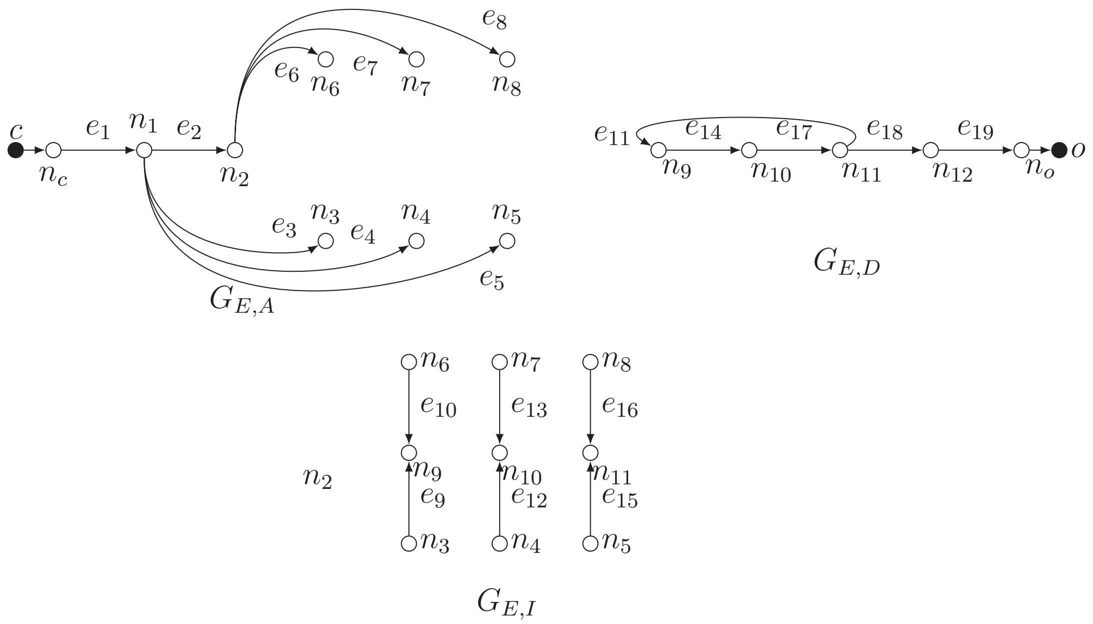


Figure 4.9.: Resulting digital sub–ESFG $G_{E,D}$, interface sub–ESFG $G_{E,I}$ and analog sub–ESFG $G_{E,A}$ of the ESFG (Fig. 4.6) of the voltage–controlled ring oscillator (Fig. 3.18).

4.2.2. Conflicts and Conflict Resolution

In all of the experiments done so far there was exactly one solution. In some cases, above CSP could not be solved in the first place because the constraints contradicted each other. This occurs if an analog building block is found in the digital or interface circuit part or vice versa. A typical example is a *differential pair* that is detected in the interface part, but the transistors work as pass transistors in reality. Such conflicts are resolved in the following steps:

1. The CSP from Eq. (4.12) is reformulated into an optimization problem. The second constraint is changed into an objective function, that minimizes the size of the cut set $N_{G_{E,A}} \cap N_{G_{E,D}}$. All nodes remaining in the cut set after solution are conflict nodes. For all other nodes the assignment is assumed to be certain.

2. The building block analysis is repeated. The recognition rules are updated, such that no pure analog building block has outputs at nodes that are certainly known to be digital and vice versa.

3. The ESFG is regenerated.

4. The above method for identification of the analog and digital part is repeated. The CSP has less conflicts because wrongly detected building blocks were removed by the second step. For all circuits investigated so far, one iteration was enough to resolve all conflicts.

4.2.3. Discussion

The algorithm partitions the ESFG G_E into the three graphs $G_{E,I}$, $G_{E,A}$ and $G_{E,D}$. Thus, these graphs are not larger than G_E, i.e.,

$$|N_{G_{E,I}}|, |N_{G_{E,A}}|, |N_{G_{E,D}}| \leq |N_{G_E}| \leq |\mathcal{N}| + |\mathcal{K}| \tag{4.18}$$

$$|E_{G_{E,I}}|, |E_{G_{E,A}}|, |E_{G_{E,D}}| \leq |E_{G_E}| \Rightarrow |E_{G_{E,I}}|, |E_{G_{E,A}}|, |E_{G_{E,D}}| \in \mathcal{O}(|\mathcal{D}|) \; . \tag{4.19}$$

The runtime is defined trough the CSP solver. The CSP solver used in this work, Gecode (Schulte et al. 2010), combines (local) constraint propagation techniques with exhaustive search. Local constraint propagation techniques solve small, fixed–sized subproblems of the overall CSP (Frühwirth & Abdennadher 2003). The solution of these sub–problems is then added as new constraint to the overall CSP. For example constraints $x > y$ and $y > 2$ imply $x > 2$. Runtime complexity of constraint propagation is strongly dependent on the used constraints. In general, local constraint propagation can be expected to have polynomial time complexity (Frühwirth & Abdennadher 2003). In case a CSP can not be fully solved by constraint propagation exhaustive search is required. The algorithm successively selects different values

```
1  $N_i \leftarrow \emptyset$; $N_o \leftarrow \emptyset$
2  for all $k_i \in \mathcal{K}_{ic}$
3      $N_i \leftarrow N_i \cup \text{forwardNodes}(G_{E,A}, k_i)$
4  for all $k_o \in \mathcal{K}_{oc}$
5      $N_o \leftarrow N_o \cup \text{backwardNodes}(G_{E,A}, k_i)$
6  $G'_{E,C} \leftarrow \text{inducedGraph}(G_{E,A}, N_i \cap N_o)$
7  $G'_{E,B} \leftarrow \text{spannedGraph}(G_{E,A}, E_{G_E} \backslash E_{G'_{E,C}} N_i \cap N_o)$
8  $G_{E,C} \leftarrow \text{insertInputTerminals}(G'_{E,C}, N_{G'_{E,C}} \cap N_{G'_{E,B}})$
9  $G_{E,B} \leftarrow \text{insertOutputTerminals}(G'_{E,B}, N_{G'_{E,C}} \cap N_{G'_{E,B}})$
```

Figure 4.10.: Algorithm for automatic identification of core and bias part.

for a variable or adds constraint based on some heuristics. Constraint propagation is repeated for every value or constraint. In general, search has exponential runtime complexity (Frühwirth & Abdennadher 2003). In practice, better performance is observed if the right heuristic is chosen.

For above problem, Gecode was configured to select variables based on the largest accumulated failure count. This chooses the variable with the highest number of failing propagators (Schulte et al. 2010). In addition, $N_{G_{E,A}} = N_{G_E}$ and $N_{G_{E,D}} = \emptyset$ is used as initialization.

4.3. Automatic Identification of Core and Bias Part

Analog circuits typically consist of a core part that does the signal processing and a bias part that provides bias voltages and currents for the core part. In the following, a new method is presented that identifies both parts automatically. The method assumes that the user specifies for each terminal if it belongs to the core or bias part. The rationale of the method is to trace the signal flow between core inputs and core outputs and construct the core part from that. The remainder of the ESFG is the bias part. In the following, the algorithm and an example are discussed.

4.3.1. Algorithm

Figure 4.10 shows the algorithm. Sets $\mathcal{K}_{ic}$ and $\mathcal{K}_{oc}$ denote the set of core input terminals and the set of core output terminals. In lines 2 and 3 the set N_i of all nodes

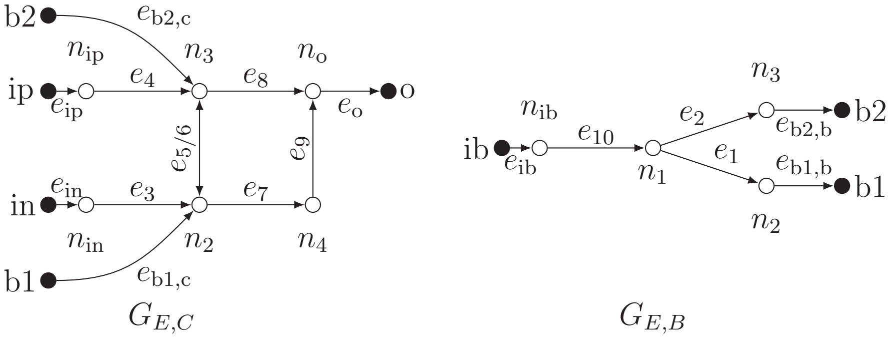


Figure 4.11.: Core ESFG $G_{E,C}$ and bias ESFG $G_{E,B}$ for the OTA circuit from Fig. 3.14 with the ESFG shown in Fig. 4.4.

reachable from a core input terminal k_i is computed. Function forwardNodes of a node n of graph G is recursively defined as follows,

$$\begin{aligned} \text{forwardNodes}(G,n) &:= \{n\} \cup \bigcup_{m \in \Gamma_G^+(n)} \text{forwardNodes}(G,m) \\ \text{where} \quad \Gamma_G^+(n) &:= \{m \in N_G \mid \exists_{e \in E_G} \big(\varphi_G(e) = (n,m)\big)\} \ . \end{aligned} \tag{4.20}$$

This matches a depth–first search which traverses all edges in forward direction. Similarly, the set of all nodes N_o from which a core output terminal can be reached is computed in lines 4 and 5. Function backwardNodes of a node n of graph G is recursively defined as follows,

$$\begin{aligned} \text{backwardNodes}(G,n) &:= \{n\} \cup \bigcup_{m \in \Gamma_G^-(n)} \text{backwardNodes}(G,m) \\ \text{where} \quad \Gamma_G^-(n) &:= \{m \in N_G \mid \exists_{e \in E_G} \big(\varphi_G(e) = (m,n)\big)\} \ . \end{aligned} \tag{4.21}$$

This matches a depth–first search which traverses all edges in backward direction. In line 6, the subgraph $G'_{E,C}$ of $G_{E,A}$ induced by the intersection of N_i and N_o is computed as preparatory step for the core graph. It contains all nodes that are part of N_i and N_o as well as the edges between them (see Definition A.8). In line 7, the subgraph $G'_{E,B}$ spanned by all edges of $G_{E,A}$ that are not part of $G'_{E,C}$ are determined as preparatory step for the bias graph. It contains all such edges and their start and end nodes (see Definition A.9). In lines 8 and 9 graphs $G'_{E,C}$ and $G'_{E,B}$ are completed to core graph $G_{E,C}$ and bias graph $G'_{E,B}$, respectively, by adding port nodes for the interface between both graphs.

4.3.2. Example

In the following the detailed flow of the algorithm for Circuit 5 (Figs. 3.14 and 4.4) is shown. The following terminal sets are used:

$$\mathcal{K}_{ic} = \{\text{ip}, \text{in}\} \qquad \mathcal{K}_{oc} = \{\text{o}\} \tag{4.22}$$

Since the circuit has no digital part, the ESFG from Fig. 4.4 is the analog ESFG. The algorithm computes the following sets,

$$N_i = \{\text{ip}, \text{in}, \text{o}, n_{\text{in}}, n_{\text{ip}}, n_o, n_2, n_3, n_4\} \tag{4.23}$$

$$N_o = \{\text{ip}, \text{in}, \text{o}, n_{\text{in}}, n_{\text{ip}}, n_{\text{ib}}, n_o, n_1, n_2, n_3, n_4\}\ . \tag{4.24}$$

The edge set of the graph $G'_{E,C}$ is

$$E_{G'_{E,C}} = \{e_{\text{ip}}, e_{\text{in}}, e_{\text{o}}, e_3, e_4, e_5, e_6, e_7, e_8, e_9\}\ . \tag{4.25}$$

Graph $G'_{E,B}$ has the following edge set $E_{G'_{E,B}}$ and node set $N_{G'_{E,B}}$,

$$E_{G'_{E,B}} = \{e_{\text{ib}}, e_1, e_2, e_{10}\} \qquad N_{G'_{E,B}} = \{\text{ib}, n_{\text{ib}}, n_1, n_2, n_3\} \tag{4.26}$$

Finally, additional terminal nodes *b1* and *b2* are added to $G'_{E,B}$ and $G'_{E,C}$ for

$$N_{G'_{E,B}} \cap N_{G'_{E,C}} = \{n_2, n_3\} \tag{4.27}$$

resulting in the graphs shown in Fig. 4.11.

4.3.3. Discussion

The runtime t_{cb} of the analysis is dominated by the time needed to compute sets N_i and N_o. For both together, $|\mathcal{K}_{ic}| + |\mathcal{K}_{oc}|$ depth–first searches are executed. Since a depth–first search is known to be linear with the number of nodes and edges, this yields,

$$t_{\text{cb}} \in \mathcal{O}((|\mathcal{K}_{ic}|+|\mathcal{K}_{oc}|)\cdot(|N_{G_{E,A}}|+|E_{G_{E,A}}|)) = \mathcal{O}((|\mathcal{K}_{ic}|+|\mathcal{K}_{oc}|)\cdot(|\mathcal{N}|+|\mathcal{K}|+|\mathcal{D}|)) \tag{4.28}$$

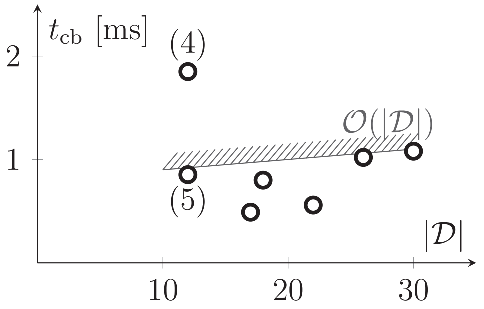


Figure 4.12.: Time needed for automatic identification of core and bias part with respect to circuit size.

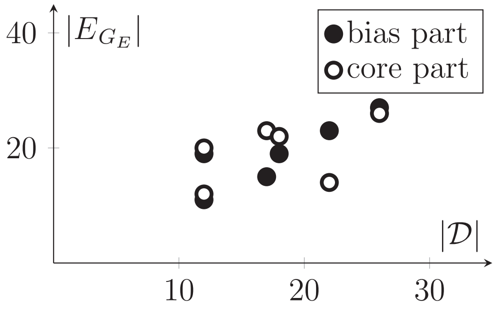


Figure 4.13.: Size of core and bias part with respect to circuit size.

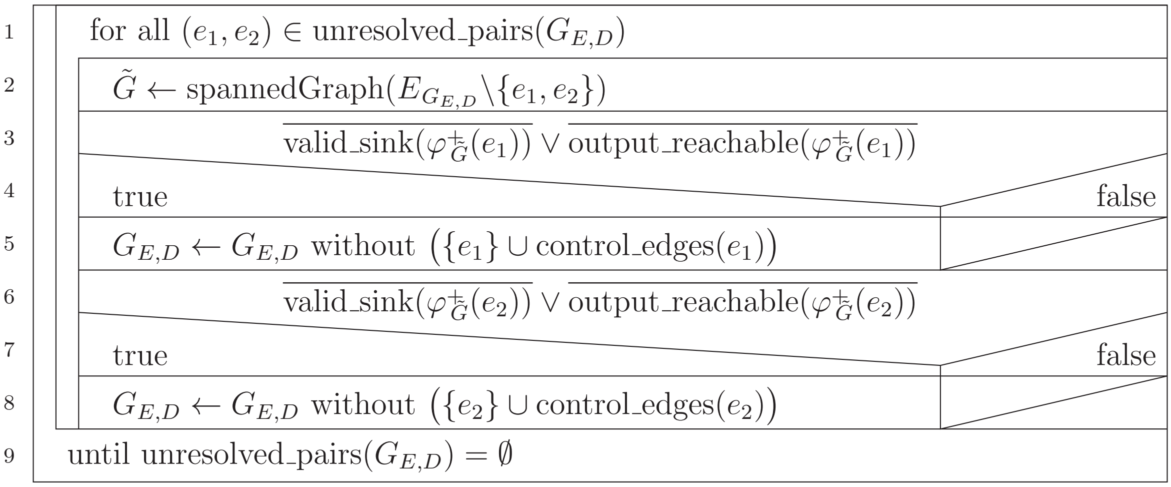
4.3.4. Experimental Results

In the following experimental results are presented for the analog test cases from Table 3.2. Figure 4.12 show the runtime needed for the automatic identification of core and bias parts. Corresponding to Eq. (4.28) the runtime appears to be bounded linearly for all circuits except Circuit 4. For Circuit 4 the runtime is longer than for the other circuits but still below 2 ms. One explanation might be that the core part of Circuit 4 has two differential inputs and one differential output, i.e., $|\mathcal{K}_{\mathrm{ic}}| + |\mathcal{K}_{oc}| = 6$ and Eq. (4.28) depends on $|\mathcal{K}_{\mathrm{ic}}| + |\mathcal{K}_{oc}|$. In contrast, the core part of Circuit 5 has one differential input and one single–ended output, i.e., $|\mathcal{K}_{\mathrm{ic}}| + |\mathcal{K}_{oc}| = 3$. Figure 4.13 shows the edge count for core and bias part with respect to circuit size. For most circuits the size of core and bias part are approximately equal.

4.4. Pass Gate Direction Assignment

So far, the channel of *pass gates* and *normal transistor arrays* has been represented by two anti–parallel edges (see Section 4.1.2). These two edges form a cycle, but the ESFG must be acyclic to compute the logic function of digital circuits later on (see Section 6.1). In most real applications, the signal flow through the *pass gates* is unidirectional. Consequently, one of the anti–parallel edges can be removed to break the cycle. An algorithm to do this automatically is presented in the following.

The method is based on a local and a global analysis per pass gate. The local analysis removes one of the edges if there is no proper signal sink on the end node of the edge. A node has a proper signal sink if edges of a *logic gate* or *pass gate* starts at this node and no edges of *logic gates* end at this node. The global analysis removes one of the edges if there is no path from the end node of the edge to the output. Next, the algorithm is presented together with an example.



```
for all $(e_1, e_2) \in$ unresolved_pairs$(G_{E,D})$
    $\tilde{G} \leftarrow$ spannedGraph$(E_{G_{E,D}} \backslash \{e_1, e_2\})$
    $\overline{\text{valid\_sink}(\varphi^{+}_{\tilde{G}}(e_1))} \vee \overline{\text{output\_reachable}(\varphi^{+}_{\tilde{G}}(e_1))}$
        true
            $G_{E,D} \leftarrow G_{E,D}$ without $\big(\{e_1\} \cup \text{control\_edges}(e_1)\big)$
        false
    $\overline{\text{valid\_sink}(\varphi^{+}_{\tilde{G}}(e_2))} \vee \overline{\text{output\_reachable}(\varphi^{+}_{\tilde{G}}(e_2))}$
        true
            $G_{E,D} \leftarrow G_{E,D}$ without $\big(\{e_2\} \cup \text{control\_edges}(e_2)\big)$
        false
until unresolved_pairs$(G_{E,D}) = \emptyset$
```

Figure 4.14.: Algorithm for pass gate direction assignment.

4.4.1. Algorithm

The algorithm (Fig. 4.14) iterates over all pairs (e_1, e_2) of anti–parallel edges of unresolved *pass gates* or *normal transistor arrays*. It holds,

$$\text{unresolved\_pairs}(G_{E,D}) := \left\{ (e_1, e_2) \in E^2_{G_{E,D}} \,\middle|\, \begin{array}{r} \big(\varphi^{-}_{G_{E,D}}(e_1) = \varphi^{+}_{G_{E,D}}(e_2)\big) \\ \wedge\big(\varphi^{-}_{G_{E,D}}(e_2) = \varphi^{+}_{G_{E,D}}(e_1)\big) \\ \wedge\big(\alpha^{C}_{G_{E,D}}(e_1) = \alpha^{C}_{G_{E,D}}(e_2)\big) \end{array} \right\} . \tag{4.29}$$

The end node of edge e_1 must be the start node of edge e_2 and vice versa. Both edges e_1 and e_2 must represent the same *pass gate* or *normal transistor array*.

The algorithm investigates graph $\tilde{G}$, which is ESFG $G_{E,D}$ without edges e_1 and e_2. Edge e_1 is removed from $G_{E,D}$ if

- the end node is no valid sink, i.e., the edge of another logic gates ends at it (local condition), or
- no output port is reachable from the end node (global condition).

At the same time, all control edges of the *pass gate* or *normal transistor array* are removed. These are all edges of the same component that share the end node with e_1:

$$\text{control\_edges}(e_1) := \{e' \in E_{G_{E,D}} | (\varphi^{+}_{G_{E,D}}(e') = \varphi^{+}_{G_{E,D}}(e_1)) \wedge (\alpha^{C}_{G_{E,D}}(e') = \alpha^{C}_{G_{E,D}}(e_1))\} \tag{4.30}$$

The same applies to e_2. Sometimes, not all *pass gates* or *normal transistor arrays* can be resolved at once. In this case, the above procedure is repeated until there are no unresolved edges left. If at least one pair of edges is assigned per iteration, the

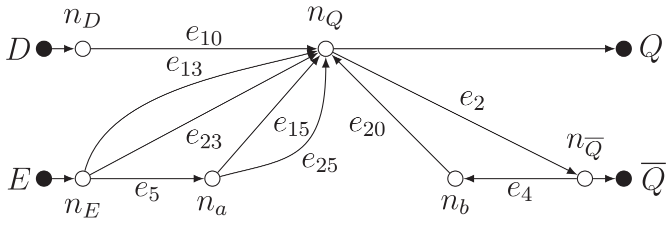


Figure 4.15.: ESFG of the latch circuit from Fig. 3.16 with assigned pass gate directions.

algorithm takes n_{pg} iterations at maximum, where n_{pg} is the number of *pass gates*. Since there are less *pass gates* than devices, the runtime $t_{\text{pg}}^{\max}$ of the analysis has linear complexity in the worst–case,

$$t_{\text{pg}}^{\max} \in \mathcal{O}(\{\mathcal{D}\}) \ . \tag{4.31}$$

4.4.2. Example

The ESFG of the latch circuit from Fig. 3.16 contains two pairs of anti–parallel edges: (e_{10}, e_{11}) and (e_{20}, e_{21}) (Fig. 4.5). The algorithm removes edge e_{11} pointing from n_Q to n_D because no output is reachable from n_D without e_{10} and e_{11}. Consequently, control edges e_{12} and e_{14} are also removed. For the second pass gate, edge e_{21} pointing from n_Q to n_b is removed together with e_{22} and e_{24}. The resulting graph is shown in Fig. 4.15.

4.4.3. Discussion

The problem of determining *pass gate* directions is similar to the problem of determining the signal flow directions of transistors in switch–level simulation (Blaauw et al. 1990). In switch–level simulation each transistor is modeled as switch. In contrast, in the ESFG generated by the approach in this thesis, only edges representing *pass gates* and *normal transistor arrays* must be handled. This allows to use the algorithm discussed above.

The Nangate library which is used as test case in this chapter does not contain *pass gates*. Thus, no experimental results can be presented. However, summarized experimental results for libraries including *pass gates* are presented in Chapter 6.

4.5. Feedback Analysis

A second origin of cycles in the ESFG are feedback loops. In digital circuits, feedback loops are used to store states between clock cycles. Such feedback loops can be broken up by introducing a time concept into the graph. This allows to model the circuit as Mealy machine (Mealy 1955). This state machine runs with a virtual clock frequency which is twice the real clock frequency, thus the real clock can be sampled according to Shannon (1949).

Next, an algorithm is presented that breaks up feedback loops automatically. It is based on the following rationale. For each existing feedback loop, one node is selected to represent the state of the loop. This node is then split up into two nodes. One node represents the state of the node in the previous time step. This node gets all edges pointing from the original node into the loop. The other node represents the state of the node in the current time step. This node gets all other edges. The split node is selected such that the direct transmission from input to output is not changed.

The resulting ESFG is denoted as temporal ESFG. A temporal ESFG is defined as follows:

Definition 4.3 (temporal ESFG)
A temporal ESFG is an ESFG $G_t = (N_{G_t}, E_{G_t}, \varphi_{G_t}, \alpha_{G_t}, \tau_{G_t})$ with additional time function $\tau_{G_t} : N_{G_t} \to \mathbb{N}$. If the time function of a node n is zero $\tau_{G_t}(n) = 0$ this means the node refers to the current time step. If the time function of a node n is one $\tau_{G_t}(n) = 1$ this means the node refers to the previous time step.

4.5.1. Algorithm

The algorithm is depicted in Fig. 4.16. First, result graph G_t is initialized as copy of the digital ESFG $G_{E,D}$. Next, strongly connected components are found in G_t using the algorithm of Tarjan (1972) (line 2). Strongly connected components are subgraphs of G_t with the property that each node inside such a subgraph can be reached from all nodes within the subgraph (see Definition A.5). In case no strongly connected components are found G_t is acyclic and the algorithm terminates (line 3). Otherwise, possible break nodes are determined inside the largest component G (line 4). Therefore, all input nodes N_{in} of G are determined (line 5). This are all nodes where an edge e ends that is not part of G. Similarly, all output nodes N_{out} of G are computed (line 6), which are all nodes, where an edge e starts that is not part of G. Next, all nodes N_{direct} are computed that are between one input and one output node (lines 7 to 9). These nodes and the input nodes can not be selected as break nodes because this would change the behavior of the direct path. In case $N_{\text{direct}} \cup N_{\text{in}}$ is different from N_G there is a valid break node n_x. The graph is then split at this node, i.e., a new node n'_x is created and the edges starting at n_x to a node inside the loop are moved to n'_x (left side of lines 12 and 13). Node n'_x represents a previous time step,

```
1   $G_t \leftarrow G_{E,D}$
2   $\mathcal{G} \leftarrow$ find_strongly_connected_components($G_t$)
3   while $\mathcal{G} \neq \emptyset$
4       $G \leftarrow$ largest_component($\mathcal{G}$)
5       $N_{\text{in}} \leftarrow \{n \in N_G | \exists_{(e \in E_{G_t} \setminus E_G)} \ \varphi^{+}_{G_{E,D}}(e) = n\}$
6       $N_{\text{out}} \leftarrow \{n \in N_G | \exists_{(e \in E_{G_t} \setminus E_G)} \ \varphi^{-}_{G_{E,D}}(e) = n\}$
7       $N_{\text{direct}} \leftarrow \emptyset$
8       for all $(n_i, n_o) \in N_{\text{in}} \times N_{\text{out}}$
9           $N_{\text{direct}} \leftarrow N_{\text{direct}} \cup$ nodes_between($G_t, n_i, n_o$)
10      $N_G \neq (N_{\text{direct}} \cup N_{\text{in}})$
11      true                                                      false
12      $n_x \leftarrow$ select_first($N_G \setminus (N_{\text{direct}} \cup N_{\text{in}})$)   |   $n_x \leftarrow$ select_first($N_{\text{out}}$)
13      $G_t \leftarrow$ split($G_t, n_x$)                             |   $G_t \leftarrow$ duplicate($G_t, n_x$)
14      $\mathcal{G} \leftarrow$ find_strongly_connected_components($G_t$)
```

Figure 4.16.: Algorithm for feedback analysis.

i.e., $\tau_{G_t}(n'_x) = \tau_{G_t}(n_x) + 1$. In addition, an output port node is created for n_x and an input port node is created for n'_x. In case $N_{\text{direct}} \cup N_{\text{in}}$ is equal to N_G the loop can not be broken without changing the direct path from input to output. In this case, the direct path must be partially duplicated. Therefore, one of the output nodes is selected (node n_x) and a new node $\tilde{n}_x$ is created (right side of line 12 and 13). All edges ending at n_x are duplicated for node $\tilde{n}_x$. All edges starting at n_x to a node outside the strongly connected component are moved to $\tilde{n}_x$. Finally, the computation of strongly connected components is repeated (line 14).

4.5.2. Example

For the latch circuit from Fig. 3.16 the flow of the algorithm is as follows. It starts from the ESFG shown in Fig. 4.15. The graph has one strongly connected component G,

$$N_G = \{n_Q, n_{\overline{Q}}, n_b\} \qquad E_G = \{e_2, e_4, e_{20}\} \ . \tag{4.32}$$

The sets of input and output nodes of G are,

$$N_{\text{in}} = \{n_Q\} \qquad N_{\text{out}} = \{n_Q, n_{\overline{Q}}\} \ . \tag{4.33}$$

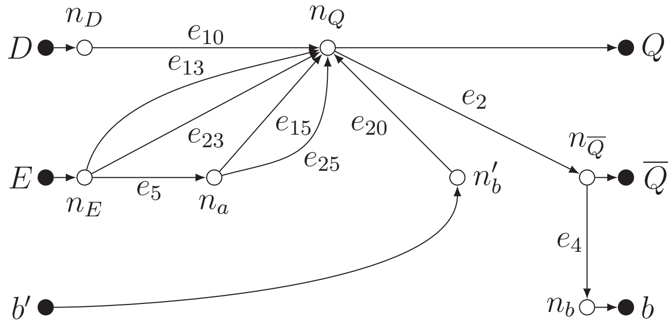


Figure 4.17.: Temporal ESFG of the latch circuit from Fig. 3.16.

There are no nodes between the input and output nodes of G, so set N_{direct} is empty. Candidates for break nodes are,

$$N_G \backslash (N_{\text{direct}} \cup N_{\text{in}}) = \{n_b, n_{\overline{Q}}\} \ . \tag{4.34}$$

The algorithm arbitrarily selects one of these node, e.g., node n_b. The resulting temporal ESFG is shown in Fig. 4.17. It has a new node n'_b representing n_b at the previous time step. Edge e_{20} was changed to start at n'_b. It also has two additional port nodes b and b'. Since this graph is acyclic, $\mathcal{G}$ is empty after the iteration and the algorithm ends.

4.5.3. Discussion

In order to examine the convergence of the algorithm, the following cases can be distinguished.

1. A valid node for the split operation is found and one strongly connected component is broken. This is the most common case.

2. A valid node for the split operation is found but the strongly connected component is not broken. This happens in case the loop is closed by multiple nodes or if the strongly connected component contains more than one loop. In this case the analysis is repeated for the same component in the next iteration. Since a strongly connected component can only contain a limited number of loops this finally will lead to the first case.

3. No valid break node is found. This can happen in case a loop has multiple input and output nodes. In this case, one of the output nodes is duplicated and a valid node for the split operation is found in the next iteration. Thus, one of the first two cases applies.

Cases 2 and 3 lead back to case 1, which breaks a strongly connected component. If one strongly connected component is broken the algorithm will handle the next strongly connected component in the next iteration. If no strongly connected components are left, the algorithm terminates.

The complexity of the worst–case runtime of this analysis is derived in the following. Strongly connected components can be computed using Tarjan's algorithm in linear time (Sedgewick 1988). The maximum of the number of nodes in graph G and sets N_{in} and N_{out} is the number of nodes in $G_{E,D}$,

$$|N_G|^{\max}, |N_{\text{in}}|^{\max}, |N_{\text{out}}|^{\max} \leq |N_{G_{E,D}}| \ . \tag{4.35}$$

Based on that, the computation of N_{direct} needs $|N_{G_{E,D}}|^2$ iterations at maximum. Since a depth first search is executed in each iteration, the overall runtime $t_{8-9}^{\max}$ for this loop is,

$$t_{8-9}^{\max} \in \mathcal{O}(|N_{G_{E,D}}|^3) \ . \tag{4.36}$$

In order to compute the overall complexity the number of iterations for the outer loop must be estimated. Graph $G_{E,D}$ can have multiple strongly connected components. However, every node can only be subject of one duplicate operation and one split operation. Thus the number of iterations is linear with the number of nodes in $G_{E,D}$. For the worst–case runtime $t_{\text{fb}}^{\max}$ of the overall algorithm this yields the following complexity,

$$t_{\text{fb}}^{\max} \in \mathcal{O}(|N_{G_{E,D}}| \times |N_{G_{E,D}}|^3) = \mathcal{O}(|N_{G_{E,D}}|^4) = \mathcal{O}((|\mathcal{N}| + |\mathcal{K}|)^4) \ . \tag{4.37}$$

For the case of standard cells, the number of strongly connected components is either 0 (combinatorial cells), 1 (latches) or 2 (registers). This number n_{loop} is independent of the circuit size. Similarly, sets N_{in} and N_{out} only contain one or two nodes, i.e.,

$$|N_{\text{in}}|^{\text{typ}}, |N_{\text{out}}|^{\text{typ}} \in \mathcal{O}(1) \Rightarrow t_{8-9}^{\text{typ}} \in \mathcal{O}(|N_{G_{E,D}}|) \tag{4.38}$$

The algorithm then searches for strongly connected components $n_{\text{loop}} + 1$ times, yielding,

$$t_{\text{fb}}^{\text{typ}} \in \mathcal{O}((n_{\text{loop}} + 1) \cdot |N_{G_{E,D}}|) = \mathcal{O}(n_{\text{loop}} \cdot (|\mathcal{N}| + |\mathcal{K}|)) \ . \tag{4.39}$$

In the typical case the complexity of the runtime of the feedback analysis is linear with the number of loops and the number of nets and terminals.

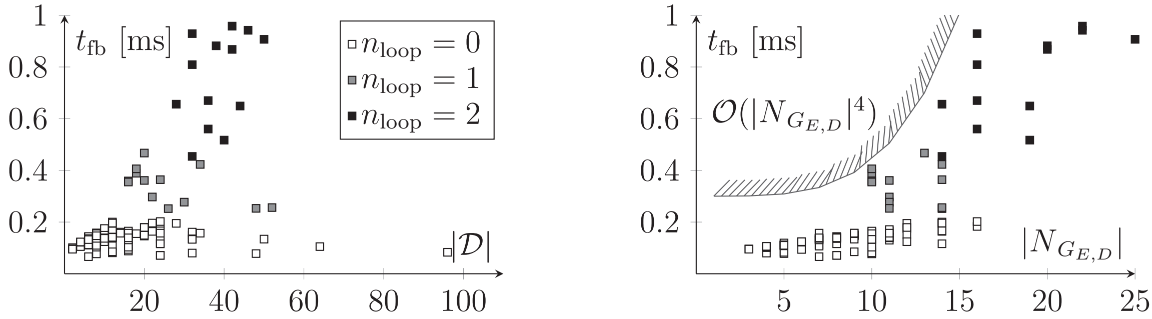


Figure 4.18.: Time needed for feedback analysis with respect to circuit size $|\mathcal{D}|$ and size of the ESFG $|N_{G_{E,D}}|$.

4.5.4. Experimental Results

Figure 4.18 shows the runtime of the algorithm over the circuit size and the number of nodes in the digital ESFG. In both plots, the data points are marked based on the number of loops in the ESFG. Circuits without loop are represented by a white square, circuits with one loop are represented by a gray square and circuits with two loops are represented by a black square. It appears that the dependency on the number of cycles suggested by Eq. (4.39) is the dominant effect. Corresponding to Eq. (4.39) the runtime appears to be linearly dependent on the number of nodes in the ESFG for circuits with the same number of cycles. The runtime seems to be bounded by $\mathcal{O}(|N_{G_{E,D}}|^4)$ as suggested by Eq. (4.37).

5. Symmetry Computation

Symmetry is a widely used design principle in analog circuits. For example, the positive and the negative path of a fully–differential circuit should behave equally. But also for single–ended opamps, it is expected that the positive and the negative input influence the output equally. In general, these conditions are realized by choosing a symmetrical topology. This allows to ensure the desired symmetrical behavior in the presence of process variations and changing operating conditions by using appropriate layout techniques.

In the following, some preliminary considerations are made for the behavioral circuit level (Section 5.1). This will lead to a number of symmetry conditions for different circuit configurations. After this, Section 5.2 will show that these conditions can be transformed to symmetry conditions on the ESFG. Next, the overall symmetry computation algorithm is presented. The algorithms includes transformation of above conditions back to conditions on the structural level. Finally, experimental results will be presented.

5.1. Preliminary Considerations

In the following, single terminals and symmetrical terminal pairs are distinguished.

Definition 5.1 (symmetrical terminal pair, single terminal)
A pair of terminals $k = (k_1, k_2) \in \mathcal{K}^2$ is called symmetrical if

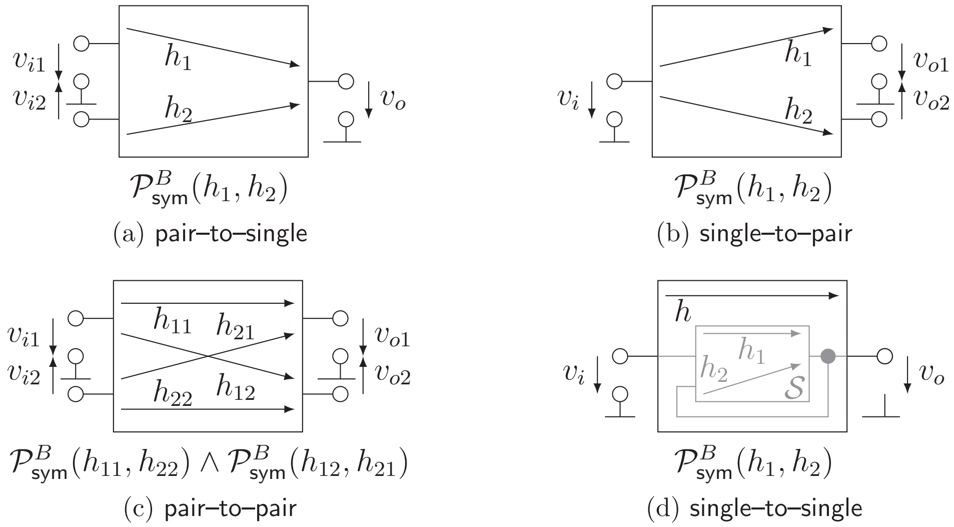
- equal currents are flowing trough terminals k_1 and k_2 or
- terminals k_1 and k_2 have equal voltages with respect to ground.

All other terminals are called single terminals.

This allows to define a signal path of a circuit.

Definition 5.2 (signal path)
A signal path $P = (k_i, k_o)$ describes the transfer behavior between one input k_i and one output k_o. Input k_i and output k_o can either be a single terminal or a symmetrical terminal pair, as follows,

$$k_i \in \mathcal{K}_i \cup \mathcal{K}_i^2 \qquad k_o \in \mathcal{K}_o \cup \mathcal{K}_o^2 \,. \tag{5.1}$$



$\mathcal{P}^B_{\mathsf{sym}}(h_1, h_2)$

(a) pair–to–single

$\mathcal{P}^B_{\mathsf{sym}}(h_1, h_2)$

(b) single–to–pair

$\mathcal{P}^B_{\mathsf{sym}}(h_{11}, h_{22}) \wedge \mathcal{P}^B_{\mathsf{sym}}(h_{12}, h_{21})$

(c) pair–to–pair

$\mathcal{P}^B_{\mathsf{sym}}(h_1, h_2)$

(d) single–to–single

Figure 5.1.: Four cases of signal paths and the associated behavioral symmetry problem.

Based on that, the four possible cases shown in Fig. 5.1 can be identified: pair–to–single signal paths (Fig. 5.1a), single–to–pair signal paths (Fig. 5.1b), pair–to–pair signal paths (Fig. 5.1c) and single–to–single signal paths (Fig. 5.1d). In the following it is assumed that the signal is encoded in the voltages but the same considerations apply to currents as well.

5.1.1. Pair–to–Single Signal Paths

A pair–to–single signal path (Fig. 5.1a) transfers data from a symmetrical terminal pair $(i1, i2)$ to a single output o. In the following it is assumed that output o can be calculated by superposition of an arbitrary transfer function h_1 from input $i1$ to output o and an arbitrary transfer function h_2 from input $i2$ to output o as follows,

$$v_o = h_1(v_{i1}) + h_2(v_{i2}) \; . \tag{5.2}$$

Two different cases leading to different behavioral symmetry conditions are considered in the following. The first case assumes that the output is influenced by both inputs in the same way, yielding,

$$h_1(x) = h_2(x) \quad \text{for } v_{i1} = v_{i2} = x \; . \tag{5.3}$$

In the second case, the output voltage v_o is assumed to be zero, resulting in

$$v_o = 0 \;\Rightarrow\; h_1(x) = -h_2(x) \quad \text{for } v_{i1} = v_{i2} = x \; . \tag{5.4}$$

This is a common condition in the small signal domain. For an ideal single–ended opamp Eq. (5.4) results in low offset error and high CMRR.

For real circuits, it is not always possible to match h_1 and h_2 exactly because parasitics might be different. In addition, the circuit topology is not always fully symmetrical. For example, additional circuitry might be required in the positive or negative signal path to convert the symmetrical signal into a single signal. Therefore, Eq. (5.3) and Eq. (5.4) are reformulated to the following minimization problem $\mathcal{P}^B_{\text{sym}}$,

$$\boxed{\mathcal{P}^B_{\text{sym}}(h_1, h_2) := \min\big||h_1(x)| - |h_2(x)|\big|} \ . \tag{5.5}$$

In the following, $\mathcal{P}^B_{\text{sym}}$ is denoted as behavioral symmetry problem. It covers Eq. (5.3) and Eq. (5.4) because the absolute values of h_1 and h_2 are considered.

5.1.2. Single–to–Pair Signal Paths

A single–to–pair signal path (Fig. 5.1b) transfers data from a single input i to a symmetrical terminal pair $(o1, o2)$. Since $(o1, o2)$ is a symmetrical terminal pair, it holds,

$$v_{o1} = v_{o2} \quad \Rightarrow \quad h_1(x) = h_2(x) \text{ for } v_i = x \quad \hat{=} \quad \boxed{\mathcal{P}^B_{\text{sym}}(h_1, h_2)} \ . \tag{5.6}$$

Functions h_1 and h_2 are arbitrary transfer functions from input i to outputs $o1$ and $o2$, respectively.

5.1.3. Pair–to–Pair Signal Paths

A pair–to–pair signal path (Fig. 5.1c) transfers data from a symmetrical terminal pair $(i1, i2)$ to a symmetrical terminal pair $(o1, o2)$. It holds,

$$v_{o1} = h_{11}(v_{i1}) + h_{21}(v_{i2}) \qquad v_{o2} = h_{12}(v_{i1}) + h_{22}(v_{i2}) \ . \tag{5.7}$$

Functions h_{11} and h_{21} are arbitrary transfer functions from inputs $i1$ and $i2$ to output $o1$, respectively. Functions h_{12} and h_{22} are arbitrary transfer functions from inputs $i1$ and $i2$ to output $o2$, respectively. For practical applications, condition $v_{o1} = v_{o2}$, can only be fulfilled if

$$h_{11}(x) = h_{22}(x) \qquad \wedge \qquad h_{21}(x) = h_{12}(x) \qquad \text{for } x = v_{i1} = v_{i2} \tag{5.8}$$

$$\hat{=} \quad \boxed{\mathcal{P}^B_{\text{sym}}(h_{11}, h_{22}) \qquad \wedge \qquad \mathcal{P}^B_{\text{sym}}(h_{21}, h_{12})} \ . \tag{5.9}$$

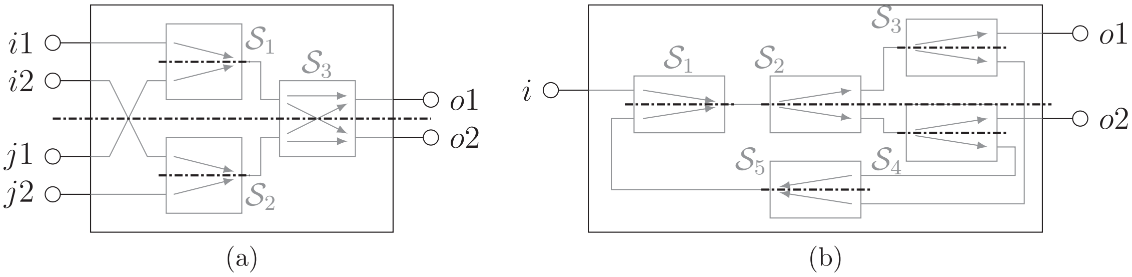


Figure 5.2.: Example for a circuit with multiple, overlapping signal paths (a) and nested signal and feedback paths (b).

5.1.4. Single–to–Single Signal Paths

A single–to–single signal path (Fig. 5.1d) transfers data from a single terminal i to a single terminal o. In general, this does not yield any symmetry condition. An exception are cases where the signal path contains an internal feedback loop. In general, four different cases of connections between the feedback network and the remaining circuit are distinguished (Sansen 2007). In the following, series–shunt feedback is considered. The gray part of Fig. 5.1d shows a Subcircuit $\mathcal{S}$, that loops back output voltage v_o. It is connected in parallel at v_o. Subcircuit $\mathcal{S}$ implements a series connection at the input. It holds,

$$v_o = h_1(v_i) + h_2(v_o) \; , \tag{5.10}$$

where transfer functions h_1 and h_2 correspond to the transfer functions of a pair–to–single signal path. This signal path does not show an offset in case $v_o = v_i = x$, resulting in

$$x - h_2(x) = h_1(x) \quad \hat{=} \quad \boxed{\mathcal{P}^B_{\text{sym}}(h_1, h_2)} \; . \tag{5.11}$$

This is again similar to the problem of minimizing the error between h_1 and h_2, yielding the behavioral symmetry problem $\mathcal{P}^B_{\text{sym}}(h_1, h_2)$.

5.1.5. Multiple Signal and Feedback Paths

The considerations made in Sections 5.1.1 to 5.1.4 are valid for circuits consisting of one signal path. Real circuits can have multiple signal and feedback paths or signal and feedback paths that overlap. Figure 5.2 shows two examples.

The first example (Fig. 5.2a) has two symmetrical input terminal pairs $(i1, i2)$, $(j1, j2)$ and one symmetrical output terminal pair $(o1, o2)$. Internally, it consists of Subcircuits $\mathcal{S}_1$ to $\mathcal{S}_3$. Examination of the pair–to–pair signal path from $(i1, i2)$ to $(o1, o2)$

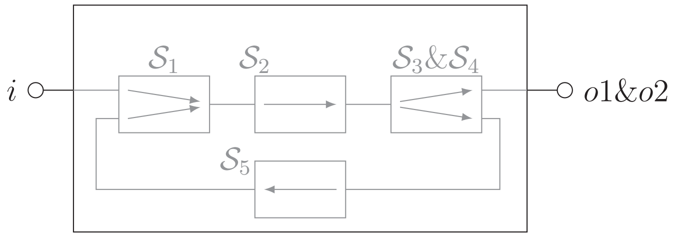


Figure 5.3.: Symmetry equivalent circuit for the circuit from Fig. 5.2b.

yields symmetry of $\mathcal{S}_3$ and between $\mathcal{S}_1$ and $\mathcal{S}_2$. The circuit has a second pair–to–pair signal from $(j1, j2)$ to $(o1, o2)$ that overlaps with the first signal path at $\mathcal{S}_3$. However, the second signal path does not yield additional symmetry conditions in this case.

The second example (Fig. 5.2b) shows a symmetrical signal path with feedback. The internal structure is outlined by Subcircuits $\mathcal{S}_1$ to $\mathcal{S}_5$. The single–to–pair signal path between input i and symmetrical output terminal pair $(o1, o2)$ requires symmetry of $\mathcal{S}_2$, $\mathcal{S}_5$ and between $\mathcal{S}_3$ and $\mathcal{S}_4$. For the feedback path, symmetry of $\mathcal{S}_1$, $\mathcal{S}_3$ and $\mathcal{S}_4$ is required.

In the following, the concepts of multipath symmetry and symmetry equivalence transformation are introduced to handle such cases correctly.

Multipath Symmetry

In general, a circuit has a set of symmetrical input terminal pairs or single input terminals K_i and a set of symmetrical output terminal pairs or output terminals K_o,

$$K_i \subseteq \mathcal{K}_i \cup {\mathcal{K}_i}^2 \quad K_o \subseteq \mathcal{K}_o \cup {\mathcal{K}_o}^2 \; . \tag{5.12}$$

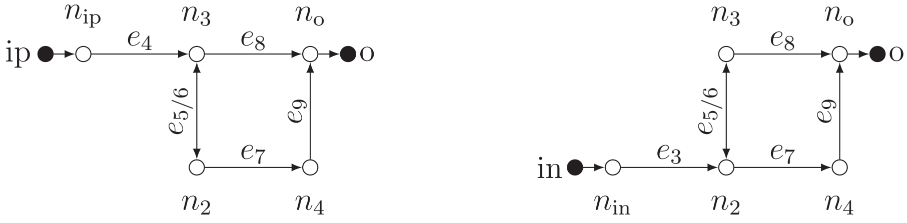
The set $\mathcal{P}$ of all possible signal paths is then,

$$\mathcal{P} = K_i \times K_o \; . \tag{5.13}$$

A circuit has multiple signal paths in case it has more than one symmetrical input terminal pair or more than one symmetrical output terminal pair. Symmetry conditions from different signal paths cannot contradict. Therefore, it is possible to analyze each possible signal path $p \in \mathcal{P}$ independently.

Symmetry Equivalence Transformation

The feedback path of the second example circuit (Fig. 5.2b) does not correspond to any of the cases presented in Sections 5.1.1 to 5.1.4. The symmetry equivalence



$G_{N1} = NCFG(G_{E,C}, \mathrm{ip}, \mathrm{o})$ $G_{N2} = NCFG(G_{E,C}, \mathrm{in}, \mathrm{o})$

Figure 5.4.: NCFGs for the circuit from Fig. 3.14.

transformation presented in the following will allow to explain this case as single–to–single signal path with feedback.

For this purpose, the circuit is transformed to a circuit that is equivalent with respect to symmetry by combining symmetrical components to one component. The signals are transformed accordingly. The result for the example is shown in Fig. 5.3. Terminals $o1$ and $o2$ have been combined as well as subcircuits $\mathcal{S}_3$ and $\mathcal{S}_4$. As a consequence $\mathcal{S}_2$ and $\mathcal{S}_5$ are changed as well. The resulting symmetry equivalent circuit has a single input terminal and a single output terminal. It is a single–to–single signal path with feedback.

5.2. Symmetry Problem on ESFG Level

The previous section showed that the behavioral symmetry conditions for the four different types of signal paths can be formulated in terms of the general symmetry problem $\mathcal{P}^B_{\mathrm{sym}}$. This section will provide an equivalent formulation of the symmetry problem for the ESFG.

5.2.1. Network components of ESFGs

The formulation is based on the network component of an ESFG (NCFG) which models the influence of one input i to one output o. It is defined as follows,

Definition 5.3 (Network component of an ESFG (NCFG))
An NCFG $G_N = \mathrm{NCFG}(G_E, i, o)$ is the maximal subgraph of ESFG G_E that is a network graph (see Definition A.6) with input node i and output node o.

As explained in Section 4.1, ESFGs are based on signal flow graphs (Mason 1953). Mason (1956) presented a method to compute transfer functions from linear signal flow graphs by determining all possible paths and adjacent loops (see Section 2.2). An

NCFG contains exactly these paths and adjacent loops. Therefore, NCFGs can be regarded as structural model for the transfer function.

Figure 5.4 shows the NCFGs G_{N1} and G_{N2} of the OTA circuit from Fig. 3.14. They are subgraphs of the ESFG shown in Fig. 4.4. NCFG G_{N1} represents the transfer behavior between input ip and output o. It consists of edges e_4 to e_9 and nodes n_{ip}, n_2, n_3, n_4 and n_{o}. NCFG G_{N2} represents the transfer behavior between input in and output o. It consists of edges e_3 and e_5 to e_9 as well as nodes n_{in}, n_2, n_3, n_4 and n_{o}.

5.2.2. Symmetry Problem on ESFG Level

In a circuit, two transfer functions are equal if the corresponding circuit parts are built equally. This can either be achieved by building exactly the same topology twice or by using the same physical devices. In the following, the first case will be denoted as symmetry and the second one as identity.

Symmetry and identity are formally defined based on the following definition of isomorphism for sub–NCFG. It extends the general definition of isomorphism (see Definition A.10) by considering the structural attributes of the edges in addition.

Definition 5.4 (Isomorphism of two subgraphs of NCFGs)
Let Q_N be a set of node pairs and Q_E a set of edge pairs,

$$Q_N \subseteq N_{G_1} \times N_{G_2} \qquad Q_E \subseteq E_{G_1} \times E_{G_2}, \tag{5.14}$$

where G_1 and G_2 are sub–NCFGs. Sets Q_N and Q_E represent mappings of the nodes and edges of G_1 and G_2, respectively. Sub–NCFG G_1 is isomorphic to sub–NCFG G_2 with respect to Q_N and Q_E iff the following holds,

$$
\begin{array}{ll}
\multicolumn{2}{l}{G_1 \underset{Q_N,Q_E}{\cong} G_2 :\Leftrightarrow} \\
\left.\begin{array}{l}\underset{(e_1,e_2)\in E_{G_1}\times E_{G_2}}{\forall} (e_1,e_2)\in Q_E \leftrightarrow \begin{bmatrix} \alpha^S_{G_1}(e_1)=\alpha^S_{G_2}(e_2) \\ (\varphi^-_{G_1}(e_1),\varphi^-_{G_2}(e_2))\in Q_N \\ (\varphi^+_{G_1}(e_1),\varphi^+_{G_2}(e_2))\in Q_N \end{bmatrix}\end{array}\right\} & \text{mapping of nodes and edges} \\
\left.\begin{array}{l}
\wedge \underset{n_1\in N_{G_1}}{\forall} \left|\{n_2\in N_{G_2} | (n_1,n_2)\in Q_N\}\right| = 1 \\
\wedge \underset{n_2\in N_{G_2}}{\forall} \left|\{n_1\in N_{G_1} | (n_1,n_2)\in Q_N\}\right| = 1 \\
\wedge \underset{e_1\in E_{G_1}}{\forall} \left|\{e_2\in E_{G_2} | (e_1,e_2)\in Q_E\}\right| = 1 \\
\wedge \underset{e_2\in E_{G_2}}{\forall} \left|\{e_1\in E_{G_1} | (e_1,e_2)\in Q_E\}\right| = 1
\end{array}\right\} & Q_N,\ Q_E \text{ bijective} \\
\left.\begin{array}{l}
\wedge \underset{e_1,e_2\in E_{G_1}\cap E_{G_2}}{\forall} (e_1,e_2)\in Q_E \leftrightarrow (e_2,e_1)\in Q_E \\
\wedge \underset{n_1,n_2\in N_{G_1}\cap N_{G_2}}{\forall} (n_1,n_2)\in Q_N \leftrightarrow (n_2,n_1)\in Q_N \,.
\end{array}\right\} & \text{symmetry of } Q_N,\ Q_E
\end{array}
\tag{5.15}
$$

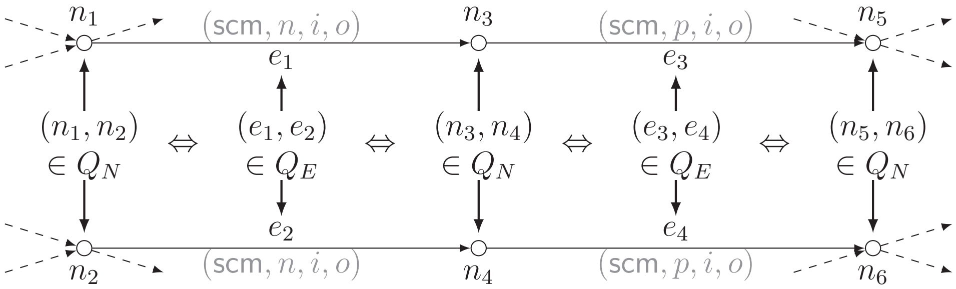


Figure 5.5.: Isomorphism of two sub–NCFGs (Definition 5.4).

Two edges must be mapped together with their start and end nodes and their structural attributes must be equal. Each node or edge of G_1 must be mapped to exactly one node or edge of G_2, i.e., the mapping Q_N must be bijective. The same applies to mapping Q_E. Mappings Q_E and Q_N must be symmetrical with respect to edges and nodes that are part of G_1 and G_2.

Figure 5.5 illustrates Definition 5.4. It shows a part of an ESFG G_1 consisting of nodes n_1, n_3, n_5 and edges e_1 and e_3 as well as an ESFG G_2 consisting of nodes n_2, n_4 and n_6 and edges e_2 and e_4. It is assumed, that edges e_1 and e_2 were generated from an NMOS *simple current mirror*. Consequently, their structural attributes are $\alpha^S_{G_1}(e_1) = \alpha^S_{G_2}(e_2) = (\text{scm}, n, i, o)$. For edges e_3 and e_4 structural attributes $\alpha^S_{G_1}(e_3) = \alpha^S_{G_2}(e_4) = (\text{scm}, p, i, o)$ are assumed. If $(e_1, e_2) \in Q_E$ this requires $(n_1, n_2) \in Q_N$ and $(n_3, n_4) \in Q_N$. If $(n_3, n_4) \in Q_N$ this requires $(e_3, e_4) \in Q_E$ which in turn requires $(n_5, n_6) \in Q_N$.

Based on Definition 5.4, symmetry of two sub-NCFG is defined as follows.

Definition 5.5 (Symmetry of two Sub–NCFGs)
Sub–NCFG G_1 is symmetrical to sub–NCFG G_2, $G_1 \mathrel{\hat{=}} G_2$, iff G_1 is isomorphic to G_2 and mapped edges represent different physical devices, i.e., the edges are not equal.

$$G_1 \underset{Q_N,Q_E}{\mathrel{\overset{\circ}{=}}} G_2 :\Leftrightarrow G_1 \underset{Q_N,Q_E}{\cong} G_2 \wedge \underset{(e1,e_2)\in Q_E}{\forall} e_1 \neq e_2 \tag{5.16}$$

Identity is defined as follows.

Definition 5.6 (Identity of two sub–NCFGs)
Sub–NCFG G_1 is identical to sub–NCFG G_2, $G_1 \equiv G_2$, iff G_1 is isomorphic to G_2 and mapped edges represent the same physical device, i.e., the edges are equal.

$$G_1 \equiv G_2 :\Leftrightarrow G_1 \underset{Q_N,Q_E}{\cong} G_2 \wedge \underset{(e1,e_2)\in Q_E}{\forall} e_1 = e_2 \tag{5.17}$$

The symmetry problem on ESFG–Level tries to find as much symmetry and identity as possible. This yields a partitioning of NCFG G_1 and G_2 into three subgraphs G_1^S, G_1^C, G_1^I and G_2^S, G_2^C, G_2^I, respectively, as follows:

1. Subgraph G_1^S is symmetrical to subgraph G_2^S. In a real circuit, that corresponds to the differential part.
2. Subgraph G_1^I is identical with subgraph G_2^I. In a real circuit, this corresponds to the part with only one signal.
3. Subgraph G_1^C connects G_1^S and G_1^I. Subgraph G_2^C connects G_2^S and G_2^I. In a real circuit, that corresponds to the part that converts a differential signal into a single signal.

Formally, the symmetry problem on ESFG–Level $\mathcal{P}_{\text{sym}}^E(G_1, G_2)$ for NCFGs G_1 and G_2 is defined as follows:

$$\boxed{\begin{array}{lll}
 & \mathcal{P}_{\text{sym}}^E(G_1, G_2) :\Leftrightarrow & \\
 & \min\limits_{\substack{G_1^S, G_1^C, G_1^I \\ G_2^S, G_2^C, G_2^I}} \alpha_C |G_1^C \cup G_2^C| + \alpha_I |G_1^I \cup G_2^I| \text{ s.t.} & \}\text{objective} \\
\wedge & G_1 = G_1^S \cup G_1^C \cup G_1^I \wedge G_2 = G_2^S \cup G_2^C \cup G_2^I & \multirow{2}{*}{$\left.\right\}$ partitioning} \\
\wedge & \forall\limits_{\substack{\mu \neq \nu \\ \mu,\nu \in \{S,C,I\}}} (E_{G_1^\mu} \cap E_{G_1^\nu} = \emptyset \wedge E_{G_2^\mu} \cap E_{G_2^\nu} = \emptyset) & \\
\wedge & (G_1^S \underset{Q_N, Q_E}{\overset{\circ}{=}} G_2^S) & \}\text{symmetry} \\
\wedge & (G_1^I \equiv G_2^I) & \}\text{identity} \\
\wedge & R_E = E_{G_1^C} \cup E_{G_2^C} & \}\text{conversion} \\
\wedge & \forall\limits_{\substack{n \in (N_{G_1^S} \cap N_{G_1^I}) \\ \wedge n \in (N_{G_2^S} \cap N_{G_2^I}) \\ \wedge n \notin (N_{G_1^C} \cup N_{G_2^C})}} \ \exists\limits_{(e_1,e_2) \in Q_E} \left[\begin{array}{ll} & \varphi_{G_1^S}^-(e_1) = \varphi_{G_2^S}^-(e_2) = n \\ \vee & \varphi_{G_1^S}^+(e_1) = \varphi_{G_2^S}^+(e_2) = n \end{array}\right] & \left.\right\}\text{interface} \\
\wedge & \left[(k_i \in \mathcal{K}) \rightarrow (k_i \in N_{G_1^I})\right] \wedge \left[(k_o \in \mathcal{K}) \rightarrow (k_o \in N_{G_1^I})\right] & \\
\wedge & \left[(k_i \in \mathcal{K}^2) \rightarrow (k_i \in N_{G_1^S} \times N_{G_2^S})\right] & \left.\right\}\text{terminals} \\
\wedge & \left[(k_o \in \mathcal{K}^2) \rightarrow (k_o \in N_{G_1^S} \times N_{G_2^S})\right] &
\end{array}} \tag{5.18}$$

The primary optimization objective is to minimize the size of the conversion part to allow as much equality as possible. For the specific problem, it is beneficial to have as much symmetry as possible because circuit robustness is increased. Therefore, minimization of the size of the identity part is a secondary optimization criterion, i.e., $\alpha_C \gg \alpha_I$.

The *partitioning* constraints describe the partitioning of G_1 and G_2. Subgraphs G_1^S, G_1^C and G_1^I of G_1 must not share an edge. The same holds for subgraphs G_2^S, G_2^C and G_2^I

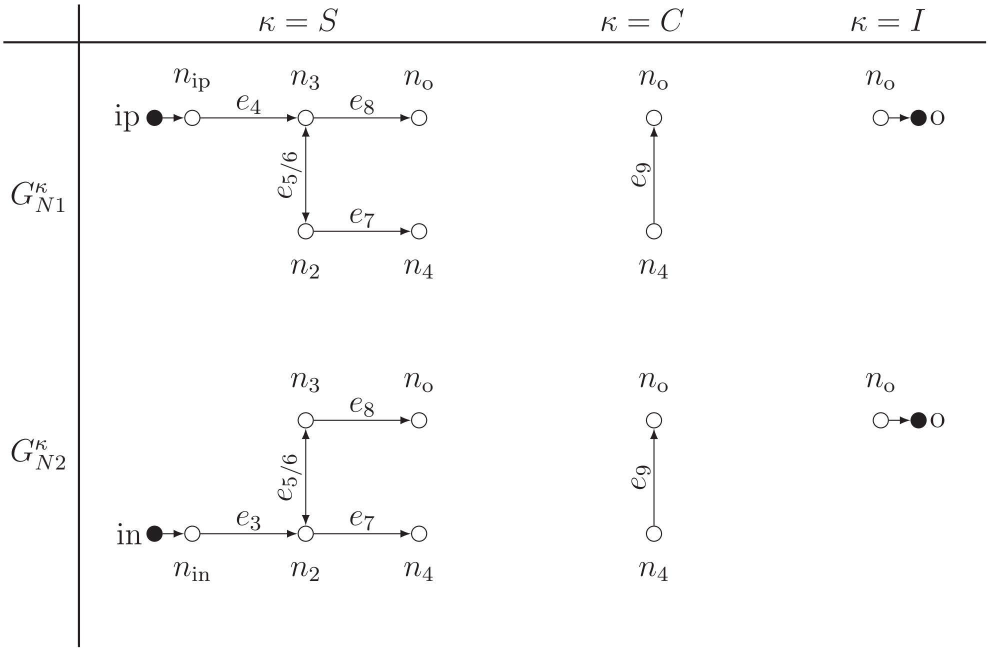


Figure 5.6.: Subgraphs of the NCFGs from Fig. 5.4.

of G_2. The *symmetry* constraints ensures that G^S_1 is symmetrical to G^S_2. The *identity* constraint ensures that G^I_1 is identical with G^I_2. The *conversion* constraint constructs set R_E out of the edges of conversion graphs G^C_1 and G^C_2. The *interface* constraints restricts transitions from the symmetrical part to the identity part. Such transitions are only possible without conversion part if there is a pair of edges mapped by Q_E that start or end at the interface node. An example follows. The *terminal* constraints force single terminals to be in the identity subgraphs G^I_1, G^I_2 and symmetrical terminal pairs to be in the symmetry subgraphs G^S_1, G^S_2.

Solution of the problem yields the node and edge mappings Q_N and Q_E of the symmetry parts G^S_1 and G^S_2 as well as set R_E which contains all edges from conversion parts G^C_1 and G^C_2.

Figure 5.6 shows the resulting subgraphs for the circuit from Fig. 3.14 and the NCFGs from Fig. 5.4. Subgraph G^S_{N1} consists of edges e_4 to e_8 and nodes *ip*, n_{ip}, n_{o}, n_2, n_3, n_4. Subgraph G^S_{N2} consists of edge e_3 and edge e_5 to e_8 and nodes *in*, n_{in}, n_{o}, n_2, n_3, n_4. The mappings Q_N and Q_E to satisfy the symmetry condition are

$$Q_N = \{(\text{ip}, \text{in}), (n_{\text{ip}}, n_{\text{in}}), (n_3, n_2), (n_2, n_3), (n_o, n_4), (n_4, n_o)\} , \tag{5.19}$$

$$Q_E = \{(e_{\text{ip}}, e_{\text{in}}), (e_4, e_3), (e_5, e_6), (e_6, e_5), (e_7, e_8), (e_8, e_7)\} . \tag{5.20}$$

Subgraphs G^I_{N1} and G^I_{N2} both consists of nodes n_{o} and *o* as well as the edge connecting them. Subgraphs G^C_{N1} and G^C_{N2} both consists of nodes n_4, n_o and edge e_9. Set R_E,

```
1  $\mathcal{Q}_N^C, \mathcal{Q}_E^C, \mathcal{R}_E^C \leftarrow \mathsf{computeSymmetry}(G_{E,C}, K_{ic} \times K_{oc})$
2  $K_{ob} \leftarrow \mathsf{createBiasOutputs}(\mathcal{Q}_N^C)$
3  $\mathcal{Q}_N^B, \mathcal{Q}_E^B, \mathcal{R}_E^B \leftarrow \mathsf{computeSymmetry}(G_{E,B}, K_{ib} \times K_{ob})$
4  $\mathcal{Q}_N \leftarrow \mathcal{Q}_N^C \cup \mathcal{Q}_N^B;\ \mathcal{Q}_E \leftarrow \mathcal{Q}_E^C \cup \mathcal{Q}_E^B;\ \mathcal{R}_E \leftarrow \mathcal{R}_E^C \cup \mathcal{R}_E^B$
5  $\mathcal{S} \leftarrow \mathsf{backAnnotateToDeviceLevel}(\mathcal{Q}_N, \mathcal{Q}_E, \mathcal{R}_E)$
```

Figure 5.7.: Overall symmetry computation method.

consisting of the edges of G_{N1}^C and G_{N2}^C is,

$$R_E = \{e_9\}\ . \tag{5.21}$$

Edge e_9 is not assigned to the identity part because of the interface constraint.

Another valid solution of the constraints would be to assign edges e_3 and e_4 to the symmetrical part, edges e_5 and e_6 to the conversion part and all other edges to the identity part. However, the value of the objective function for this solution is worse because the identity part is larger.

5.3. Algorithm

Figure 5.7 shows the overall symmetry computation method resulting from the principles discussed in Sections 5.1 and 5.2. It takes the sets of core input and output terminals and symmetrical terminal pairs K_{ic}, K_{oc} as input as well as the set of bias input terminals and symmetrical terminal pairs K_{ib}. Other inputs are the core and bias ESFG resulting from the method described in Section 4.3. First, symmetries inside the core part are computed for all possible signal paths $\mathcal{P} = K_{ic} \times K_{oc}$ (line 1). The resulting set of symmetrical node pairs $\mathcal{Q}_N^C$ is then used to determine symmetrical terminals K_{ob} within the output terminals of the bias part of the circuit (line 2). This allows to compute symmetries within the bias part in the next step (line 3). After that, final result sets $\mathcal{Q}_N$, $\mathcal{Q}_E$ and $\mathcal{R}_E$ are constructed from the result of the symmetry computation for the core and bias part (line 4). Finally, the overall results are back annotated to the device level, resulting in a set $\mathcal{S}$ of symmetrical device pairs (line 5). These steps are detailed in the following.

5.3.1. Symmetry Computation for Core Part

The details of the symmetry computation method for the core part are shown in Fig. 5.8. The method implements the concept of multipath symmetry (Section 5.1.5)

```
computeSymmetry$(G, \mathcal{P})$
    $\mathcal{Q}_N \leftarrow \emptyset; \mathcal{Q}_E \leftarrow \emptyset; \mathcal{R}_E \leftarrow \emptyset$
    for all $(k_i, k_o) \in \mathcal{P}$
        $k_i \in \mathcal{K}_i{}^2$
        true
            $(k_{i1}, k_{i2}) \leftarrow k_i$
            $k_o \in \mathcal{K}_o{}^2$
            true
                $(k_{o1}, k_{o2}) \leftarrow k_o$
                $G_{11} \leftarrow$ NCFG$(G, k_{i1}, k_{o1})$
                $G_{12} \leftarrow$ NCFG$(G, k_{i1}, k_{o2})$
                $G_{21} \leftarrow$ NCFG$(G, k_{i2}, k_{o1})$
                $G_{22} \leftarrow$ NCFG$(G, k_{i2}, k_{o2})$
                $Q_N^1, Q_E^1, R_E^1 \leftarrow$ solve$(\mathcal{P}_{\mathsf{sym}}^E(G_{11}, G_{22}))$
                $Q_N^2, Q_E^2, R_E^2 \leftarrow$ solve$(\mathcal{P}_{\mathsf{sym}}^E(G_{12}, G_{21}))$
                $\mathcal{Q}_N \leftarrow \mathcal{Q}_N \cup Q_N^1 \cup Q_N^2$
                $\mathcal{Q}_E \leftarrow \mathcal{Q}_E \cup Q_E^1 \cup Q_E^2$
                $\mathcal{R}_E \leftarrow \mathcal{R}_E \cup R_E^1 \cup R_E^2$
            false
                $G_1 \leftarrow$ NCFG$(G, k_{i1}, k_o)$
                $G_2 \leftarrow$ NCFG$(G, k_{i2}, k_o)$
                $Q_N^1, Q_E^1, R_E^1 \leftarrow$ solve$(\mathcal{P}_{\mathsf{sym}}^E(G_1, G_2))$
                $\mathcal{Q}_N \leftarrow \mathcal{Q}_N \cup Q_N^1$
                $\mathcal{Q}_E \leftarrow \mathcal{Q}_E \cup Q_E^1$
                $\mathcal{R}_E \leftarrow \mathcal{R}_E \cup R_E^1$
        false
            $k_o \in \mathcal{K}_o{}^2$
            true
                $(k_{o1}, k_{o2}) \leftarrow k_o$
                $G_1 \leftarrow$ NCFG$(G, k_i, k_{o1})$
                $G_2 \leftarrow$ NCFG$(G, k_i, k_{o2})$
                $Q_N^1, Q_E^1, R_E^1 \leftarrow$ solve$(\mathcal{P}_{\mathsf{sym}}^E(G_1, G_2))$
                $\mathcal{Q}_N \leftarrow \mathcal{Q}_N \cup Q_N^1$; $\mathcal{Q}_E \leftarrow \mathcal{Q}_E \cup Q_E^1$
                $\mathcal{R}_E \leftarrow \mathcal{R}_E \cup R_E^1$
            false
                $G_N \leftarrow$ NCFG$(G, k_i, k_o)$
                $G_1 \leftarrow$ forwardPart$(G_N)$
                $G_2 \leftarrow$ loopPart$(G_N)$
                $Q_N^1, Q_E^1, R_E^1 \leftarrow$ solve$(\mathcal{P}_{\mathsf{sym}}^E(G_1, G_2))$
                $\mathcal{Q}_N \leftarrow \mathcal{Q}_N \cup Q_N^1$; $\mathcal{Q}_E \leftarrow \mathcal{Q}_E \cup Q_E^1$
                $\mathcal{R}_E \leftarrow \mathcal{R}_E \cup R_E^1$
    $\mathcal{Q}_E \neq \emptyset$
    true
        $Q_N, Q_E, R_E \leftarrow$ compSymForEquivalentESFG$(G, \mathcal{P}, \mathcal{Q}_N, \mathcal{Q}_E)$
        $\mathcal{Q}_N \leftarrow \mathcal{Q}_N \cup Q_N; \mathcal{Q}_E \leftarrow \mathcal{Q}_E \cup Q_E; \mathcal{R}_E \leftarrow \mathcal{R}_E \cup R_E$
    false
    return $\mathcal{Q}_N, \mathcal{Q}_E, \mathcal{R}_E$
```

Figure 5.8.: Algorithm to compute symmetries within ESFG G using signal paths $\mathcal{P}$.

by iterating over all possible signal paths $\mathcal{P}$ of ESFG G (line 2). Inside the loop, in lines 3, 6 and 18, the algorithm distinguishes into pair–to–pair signal paths (left side of lines 7 to 16), pair–to–single signal paths (right side of lines 7 to 12), single–to–pair signal paths (left side of lines 19 to 24) and single–to–single signal paths (right side of lines 19 to 24). In each case, the symmetry problem on ESFG–level corresponding to the symmetry problems derived in Sections 5.1.1 to 5.1.4 is set up.

Function *NCFG* computes the NCFG between an input i and an output o. It determines the subgraphs containing all nodes that can be reached from the input and from which the output is reachable,

$$\begin{aligned}\mathrm{NCFG}(G, i, o) := {} & \mathrm{inducedGraph}(G, \\ & \mathrm{forwardReachableNodes}(G, i) \cap \mathrm{backwardReachableNodes}(G, o)) \ . \end{aligned} \tag{5.22}$$

Functions inducedGraph, forwardReachableNodes and backwardReachableNodes are defined according to Definition A.8 and Eqs. (4.20) and (4.21), respectively.

The symmetry problem is then formulated as constraint satisfaction problem and solved using the constraint programming package Gecode (Schulte et al. 2010). In the algorithm this is symbolized by function *solve*. The results are then used to update the set of symmetrical node pairs $\mathcal{Q}_N$, the set of symmetrical edge pairs $\mathcal{Q}_E$ and the set of conversion edges $\mathcal{R}_E$.

For the case of single–to–single signal paths (right side of lines 19 to 24) some extra effort is necessary to identify the feedback. The sub–NCFG corresponding to transfer function h_1 which describes the direct transmission from the input to the output is found by the forwardPart method. It computes all paths between the input and output of the NCFG that contain no node more than once. The sub–NCFG corresponding to transfer function h_2 which describes the feedback loop is found by the loopPart method. Similar to the feedback analysis (Section 4.5), it uses the algorithm of Tarjan (1972) to compute strongly connected components (Sedgewick 1988). In contrast to the feedback analysis, it does not break the loop and assumes that there is only one strongly connected component.

Finally, the symmetry equivalence transformation is performed in case symmetries were found (lines 25 to 28). The details are shown in Fig. 5.9. First, the equivalent ESFG is computed. Each symmetrical node pair is represented as one node. Each symmetrical edge pair is represented as one edge. After that, a similar transformation is done for the signal paths. Next, the symmetry computation is repeated using the equivalent ESFG and signal paths. This results in sets $\tilde{\mathcal{Q}}_N$, $\tilde{\mathcal{Q}}_E$ and $\tilde{\mathcal{R}}_E$. These sets are then transformed back into set Q_N, Q_E and R_E.

Figure 5.10 shows the symmetry–equivalent ESFG for the ESFG from Fig. 4.11 and sets $\mathcal{Q}_N$ and $\mathcal{Q}_E$ given in Eqs. (5.19) and (5.20), respectively. All symmetrical node pairs and all symmetrical edge pairs are represented by one node, e.g., nodes n_2 and

```
compSymForEquivalentESFG($G, \mathcal{P}, \mathcal{Q}_N, \mathcal{Q}_E$)
$\tilde{G} \leftarrow$ createEquivalentESFG($G, \mathcal{Q}_N, \mathcal{Q}_E$)
$\tilde{\mathcal{P}} \leftarrow$ createEquivalentSignalPaths($\mathcal{P}, \mathcal{Q}_N$)
$\tilde{\mathcal{Q}}_N, \tilde{\mathcal{Q}}_E, \tilde{\mathcal{R}}_E \leftarrow$ computeSymmetry($\tilde{G}, \tilde{\mathcal{P}}$)
$Q_N \leftarrow$ transformBack($\tilde{\mathcal{Q}}_N$)
$Q_E \leftarrow$ transformBack($\tilde{\mathcal{Q}}_E$)
$R_E \leftarrow$ transformBack($\tilde{\mathcal{R}}_E$)
return $Q_N$, $Q_E$, $R_E$
```

Figure 5.9.: Algorithm for symmetry equivalence transformation

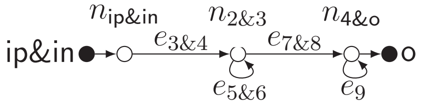


Figure 5.10.: Symmetry equivalent ESFG of Fig. 4.11.

n_3 are represented by node $n_{2\&3}$. No additional symmetries are found for the example circuit.

Overall, the symmetry computation for the core part results in the sets listed in Eqs. (5.19) to (5.21) as follows,

$$\mathcal{Q}_N^C = \{(\text{ip}, \text{in}), (n_{\text{ip}}, n_{\text{in}}), (n_3, n_2), (n_2, n_3), (n_o, n_4), (n_4, n_o)\} \tag{5.23}$$

$$\mathcal{Q}_E^C = \{(e_{\text{ip}}, e_{\text{in}}), (e_4, e_3), (e_5, e_6), (e_6, e_5), (e_7, e_8), (e_8, e_7)\} \tag{5.24}$$

$$\mathcal{R}_E^C = \{e_9\} \; . \tag{5.25}$$

5.3.2. Symmetry Computation for Bias Part

The symmetry computation for the bias part is based on the symmetry computation for the core part. First, symmetrical output terminal pairs and single output terminals K_{ob} of the bias part are determined. After that, the symmetry computation method from above is repeated for the bias part.

In Section 4.2, extra terminal nodes for the interface between the bias part and the core part were introduced. For the core part, symmetry of these terminal nodes results from the symmetry computation. In the bias part, a pair of these terminal nodes is symmetrical if the corresponding pair in the core part is symmetrical. All other terminals are single terminals.

In the ESFGs from Fig. 4.8 extra terminal nodes b1 and b2 were introduced to describe the interface between core and bias part. These nodes are a symmetrical node pair (b1, b2) because they connect to nodes n_2 and n_3, which are a symmetrical node pair in the symmetry computation result from Eq. (5.23). This yields,

$$K_{ob} = \{(\mathrm{b1}, \mathrm{b2})\} \; . \tag{5.26}$$

The symmetry computation for the bias part results in

$$\mathcal{Q}_N^B = \{(n_2, n_3), (\mathrm{b1}, \mathrm{b2})\} \qquad \mathcal{Q}_E^B = \{(e_1, e_2), (e_{\mathrm{b1},b}, e_{\mathrm{b2},b})\} \qquad \mathcal{R}_E^B = \emptyset \; . \tag{5.27}$$

The overall symmetry computation results are,

$$\begin{aligned} \mathcal{Q}_N &= \mathcal{Q}_N^C \cup \mathcal{Q}_N^B \\ &= \{(\mathrm{ip}, \mathrm{in}), (n_{\mathrm{ip}}, n_{\mathrm{in}}), (n_3, n_2), (n_2, n_3), (n_o, n_4), (n_4, n_o), (\mathrm{b1}, \mathrm{b2})\} && (5.28) \\ \mathcal{Q}_E &= \mathcal{Q}_E^C \cup \mathcal{Q}_E^B \\ &= \{(e_{\mathrm{ip}}, e_{\mathrm{in}}), (e_4, e_3), (e_5, e_6), (e_6, e_5), (e_7, e_8), (e_8, e_7), (e_1, e_2), (e_{\mathrm{b1},b}, e_{\mathrm{b2},b})\} && (5.29) \\ \mathcal{R}_E &= \mathcal{R}_E^C \cup \mathcal{R}_E^B = \{e_9\} \; . && (5.30) \end{aligned}$$

5.3.3. Back Annotation to the Device Level

Device level symmetries are determined in two steps. First, the sets of symmetrical nodes and edges $\mathcal{Q}_N$ and $\mathcal{Q}_E$ as well as the set of conversion edges $\mathcal{R}_E$ are transformed into a set of symmetrical building blocks $\mathcal{Q}_\mathcal{B}$ and a set of so–called self–symmetrical building blocks $\mathcal{R}_\mathcal{B}$. Next, sets $\mathcal{Q}_\mathcal{B}$ and $\mathcal{R}_\mathcal{B}$ are transformed into a set of symmetrical devices $\mathcal{S}$.

Two different building blocks x_1 and x_2 are symmetrical if corresponding edges are symmetrical, i.e.,

$$\mathcal{Q}_\mathcal{B} = \{(x_1, x_2) \in \mathcal{B}^2 | (x_1 \neq x_2) \wedge \underset{(e_1,e_2)\in\mathcal{Q}_E}{\exists} (\alpha_{G_E}^C(e_1) = x_1) \wedge (\alpha_{G_E}^C(e_2) = x_2)\} \; , \tag{5.31}$$

where $\mathcal{Q}_\mathcal{B}$ is the resulting set of all symmetrical building block. A building block x is called self symmetrical if two of the included devices are symmetrical to each other. Self–symmetrical building blocks are determined in two ways:

1. Two edges belonging to the same building block x are symmetrical,

$$\mathcal{R}_{\mathcal{B},1} = \left\{ x \in \mathcal{B} \middle| \left(\underset{(e_1,e_2)\in\mathcal{Q}_E}{\exists} \alpha^C_{G_E}(e_1) = \alpha^C_{G_E}(e_2) = x \right) \right\} . \tag{5.32}$$

2. Observation shows that building blocks in the conversion part are self–symmetrical in general,

$$\mathcal{R}_{\mathcal{B},2} = \left\{ x \in \mathcal{B} \middle| \left(\underset{e\in\mathcal{R}_E}{\exists} \alpha^C_{G_E}(e) = x \right) \right\} . \tag{5.33}$$

The overall set of self–symmetrical building blocks is,

$$\mathcal{R}_{\mathcal{B}} = \mathcal{R}_{\mathcal{B},1} \cup \mathcal{R}_{\mathcal{B},2} . \tag{5.34}$$

These two cases are transformed differently to device level. In the first case, symmetry is recursively translated trough the hierarchy of the building blocks until device level is reached. The set of all such symmetrical devices $\mathcal{S}_1$ is,

$$\mathcal{S}_1 = \bigcup_{(x_1,x_2)\in\mathcal{Q}_{\mathcal{B}}} \mathrm{baSym}(x_1, x_2) \tag{5.35}$$

$$\mathrm{baSym}(x_1, x_2) = \begin{cases} \bigcup_{i=1}^{|x_1.children|} \mathrm{baSym}(x_1.child_i, x_2.child_i) & x_1, x_2 \in \mathcal{B} \\ (x_1, x_2) & x_1, x_2 \in \mathcal{D} \end{cases} \tag{5.36}$$

Function baSym recursively descends trough the extracted hierarchy of building blocks until the devices level is reached. Children with the same index in x_1 and x_2 then form a new pair of symmetrical devices.

For the second case, a special function selfSymmetry is defined per building block type t in a library. It returns a set of symmetrical device pairs. The set of all such symmetrical devices $\mathcal{S}_2$ is,

$$\mathcal{S}_2 = \bigcup_{x\in\mathcal{R}_{\mathcal{B}}} \mathrm{selfSymmetry}_{x.type}(x) . \tag{5.37}$$

The library is organized as follows. For all building blocks consisting of two devices only, a symmetry pair is formed out of these devices. For four transistor current mirrors like the *cascode current mirror* or the *wide–swing cascode current mirror* this yields symmetry between the stacked transistors at the input and the stacked transistors at the output.

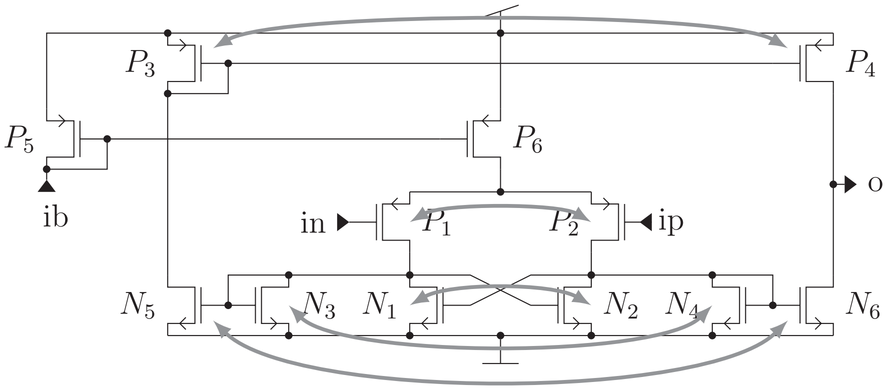


Figure 5.11.: Circuit 5: Results of symmetry computation.

The overall set of device symmetries $\mathcal{S}$ is,

$$\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \; . \tag{5.38}$$

For the example the following sets are found for the building blocks,

$$\mathcal{Q}_{\mathcal{B}} = \{(\mathrm{scm1}, \mathrm{scm2})\} \qquad \mathcal{R}_{\mathcal{B}} = \{\mathrm{dp1}, \mathrm{cc1}, \mathrm{scm3}\} \; . \tag{5.39}$$

This results in the following symmetrical device pairs,

$$\mathcal{S} = \{\underbrace{(N_3, N_4), (N_5, N_6)}_{\mathcal{S}_1}, \underbrace{(P_1, P_2), (N_1, N_2), (P_3, P_4)}_{\mathcal{S}_2}\} \; . \tag{5.40}$$

Set $\mathcal{S}$ is illustrated by Fig. 5.11.

5.3.4. Discussion

The runtime complexity of the symmetry computation is dominated by the runtime complexity to solve the CSP. As discussed in Section 4.2.3, worst–case runtime complexity is exponential but the complexity observed in practice is better. For this problem, Gecode is configured to select search variables based on the largest accumulated failure count (see Section 4.2.3). For initialization, it is assumed that all edges of NCFGs G_1 and G_2 belong to the symmetry part, i.e., $E_{G_1^S} = E_{G_1}$, $E_{G_2^S} = E_{G_2}$.

A limitation of the current method is that it can not handle all possible types of feedback paths like shunt–shunt feedback or series–series feedback. In particular, it is limited to series–shunt feedback configurations.

circuit	class	linear	no. devices	no. symmetry pairs			
				EXP	COM	PROP	CP
1	SE-Opamp	✓	22	5	$3 + 3F$	5	5
2	FD-Opamp	✓	30	10	10	10	10
3	FD-Opamp	✓	26	11	8	9	11
4	FD-Mixer	×	12	7	3	–	7
5	SE-Opamp	✓	12	5	$3 + 2F$	5	5
6	SE-Opamp	✓	17	6	6	6	6
7	SE-Opamp	✓	18	6	6	6	6

EXP: expected; COM: commercial tool; PROP: propagation approach; CP: constraint programming approach presented in this chapter

Table 5.1.: Overview of symmetry computation results for different circuits.

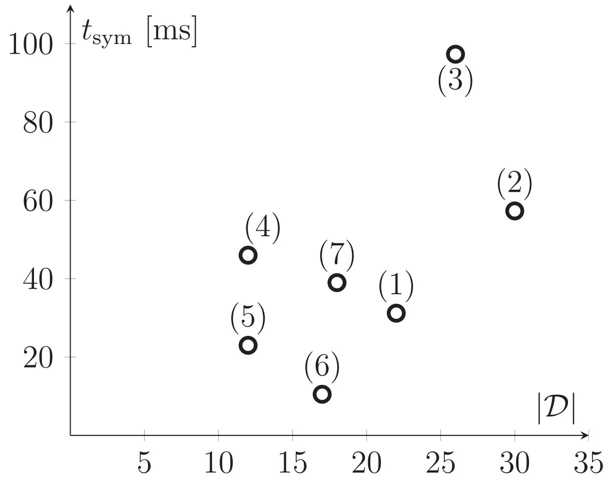


Figure 5.12.: Time needed for the symmetry computation of approach CP with respect to circuit size.

5.4. Experimental Results

In the following, a number of experimental results are presented for the analog test cases from Table 3.2. They cover six linear opamps and one non–linear mixer. Table 5.1 summarizes the results. Besides class, linearity and number of devices it lists the number of expected symmetry pairs (column EXP) and computation results for three different approaches. Approach COM is a commercial tool. Approach PROP is an earlier version of the approach presented in this thesis, which can not handle overlapping signal paths and is based on backtracking instead of constraint programming. Details can be found in Lu (2009) and Eick, Lu & Graeb (2010). Approach CP is the approach presented in this chapter.

Approach COM handles Circuit 2, 6 and 7 correctly. For Circuit 1 it misses two symmetry pairs and detects three false pairs. For Circuit 3 it misses three pairs and for Circuit 4 four pairs are missing. For Circuit 5, which is the example circuit from above, it misses two pairs and detects the two false pairs (N_3, N_2) and (N_1, N_4) .

Approach PROP handles Circuits 1, 2, 5, 6, and 7 correctly. For Circuit 3 symmetry pairs are only found in a part of the circuit. For Circuit 4 the experiment was not performed because of the known limitations of the algorithm in handling overlapping signal paths.

Approach CP handles all circuits correctly. Compared to approach COM it shows improved handling of the asymmetries in Circuits 1 and 5 and of the overlapping signal paths in Circuits 3 and 4. Compared to approach PROP the handling of overlapping signal paths was improved. Figure 5.12 shows the runtime of the method for the test cases. There is a large variation between the circuits because the method is strongly dependent on the internal structure of the circuit. Detailed results for Circuits 3, 4 and 6 are shown in the following.

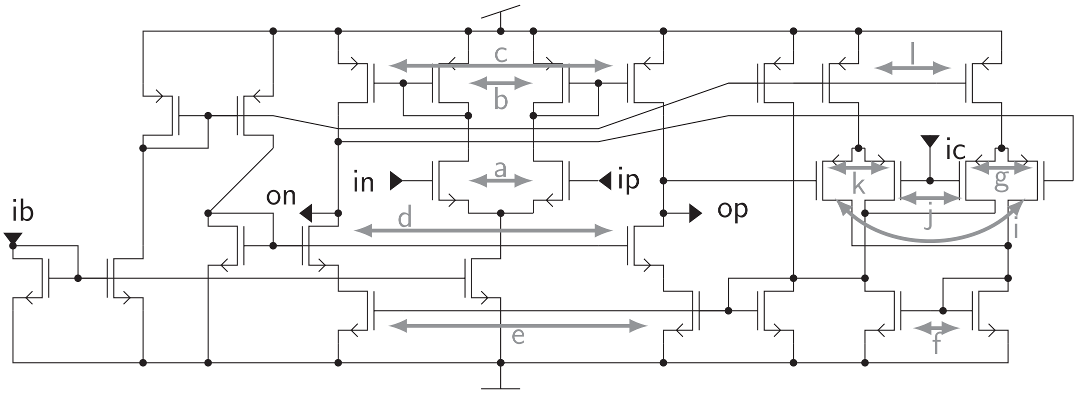


Figure 5.13.: Circuit 3 (Johns & Martin 1997, Cadence Design Systems 2010): Results of symmetry computation using method CP.

Detailed results for Circuit 3 are shown in Fig. 5.13. Circuit 3 is a fully differential OTA with differential inputs (ip, in), differential outputs (op, on), input for the common mode voltage ic and bias current ib. For the experiment, sets K_{ic}, K_{oc} and K_{ib} were,

$$K_{ic} = \{(ip, in), ic\} \quad K_{oc} = \{(op, on)\} \quad K_{ib} = \{ib\} \ . \tag{5.41}$$

The symmetry pairs shown in Fig. 5.13 are computed in the following steps,

1. Symmetry is computed for the pair–to–pair signal path from (ip, in) to (op, on). This yields symmetry pairs *a* to *e*.
2. Symmetry is computed for the single–to–pair signal path from ic to (op, on). This yields symmetry pairs *i* and *j*.
3. Symmetry equivalence transformation is performed based on Step 2. From the feedback path, symmetry pairs *f*, *g* and *k* are found.
4. Symmetry computation for the bias part yields symmetry pair *l*.

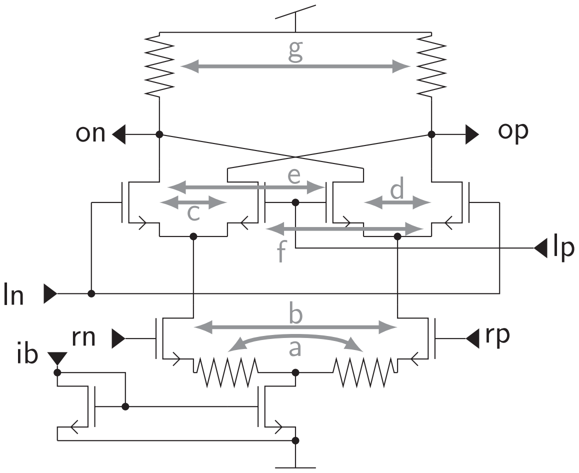


Figure 5.14.: Circuit 4 (Lee 2004): Results of symmetry computation using method CP.

Figure 5.14 shows detailed results for Circuit 4. It is a fully differential mixer with two differential inputs (lp, ln) and (rp, rn) as well as differential output (op, on). For the experiment, sets K_{ic}, K_{oc} and K_{ib} were,

$$K_{ic} = \{(lp, ln), (rp, rn)\} \quad K_{oc} = \{(op, on)\} \quad K_{ib} = \{ib, vdd, gnd\} \ . \tag{5.42}$$

The symmetry pairs shown in Fig. 5.14 are computed in the following steps,

1. Symmetry is computed for the pair–to–pair signal path from (lp, ln) to (op, on). This yields symmetry pairs c and d.
2. Symmetry is computed for the pair–to–pair signal path from (rp, rn) to (op, on). This yields symmetry pairs a, b, e and f.
3. Symmetry is computed for the bias part. This yields symmetry pair g.

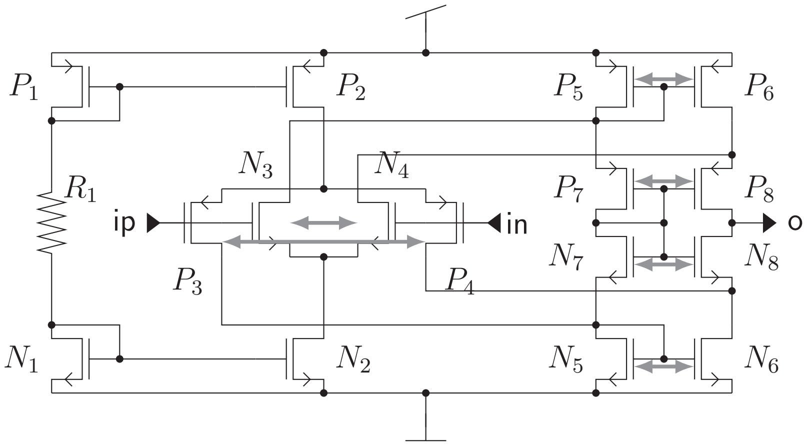


Figure 5.15.: Circuit 6: Results of symmetry computation using method CP.

Figure 5.15 shows detailed results for Circuit 6. It is a single–ended OTA with differential input (ip, in) and single output o. All symmetry pairs show in Fig. 5.15 are found by computing symmetry for the pair–to–single signal path from (ip, in) to o.

6. Application: Input Data Generation for Timing Characterization of Digital Standard Cells

Software for automatic timing analysis requires input data about the characteristics of each element of the standard cell library. This includes a list of input and output pins, the logic function and the delay of the cell for a single switching input. This delay is determined separately for every pin, rising and falling edges and all possible combinations of input levels of the other inputs. This data is referred to as timing vectors. The characterization of standard cells can be done automatically using extensive numerical circuit simulation, e.g., using SPICE. This simulation determines the delay, but again requires input and output pins, logic function and a list of timing vectors as input. In the following it is explained how this data can be generated automatically from the structural analysis results. An additional requirement for the method was to analyze the internal structure of the cell in terms of single–stage logic gates. This feature is required, e.g., by the methods of Knoth et al. (2009) and Lorenz (2012), which build current–source and aging models for multi–stage gates out of the models for single–stage gates.

An overview of the generation flow is given in Fig. 6.1. First, logic functions are computed based on the results of the building block analysis (see Chapter 3) and structural signal path analysis (see Chapter 4). The logic function is then used to detect the type of the cell, e.g., And, Or, Nand, Nor, Flip Flop, Latch, etc. The logic function is also used to generate the timing vectors. Number and direction of pins as well as the internal structure of the cell are determined from the ESFG and temporal ESFG. The analysis steps are described in the following. After that, experimental results are presented.

6.1. Logic Function Extraction

The logic function extraction is done in three steps. First, the logic function of the building blocks is computed. Second, the overall logic function of the complete cell is computed. Third, the overall logic function of sequential cells is transformed to a representation that is based on trigger conditions.

All logic function computations are based on ROBDDs (Bryant 1986) and use a four–valued logic with 0, 1, Z (high impedance) and U (unknown) according to Bryant (1991).

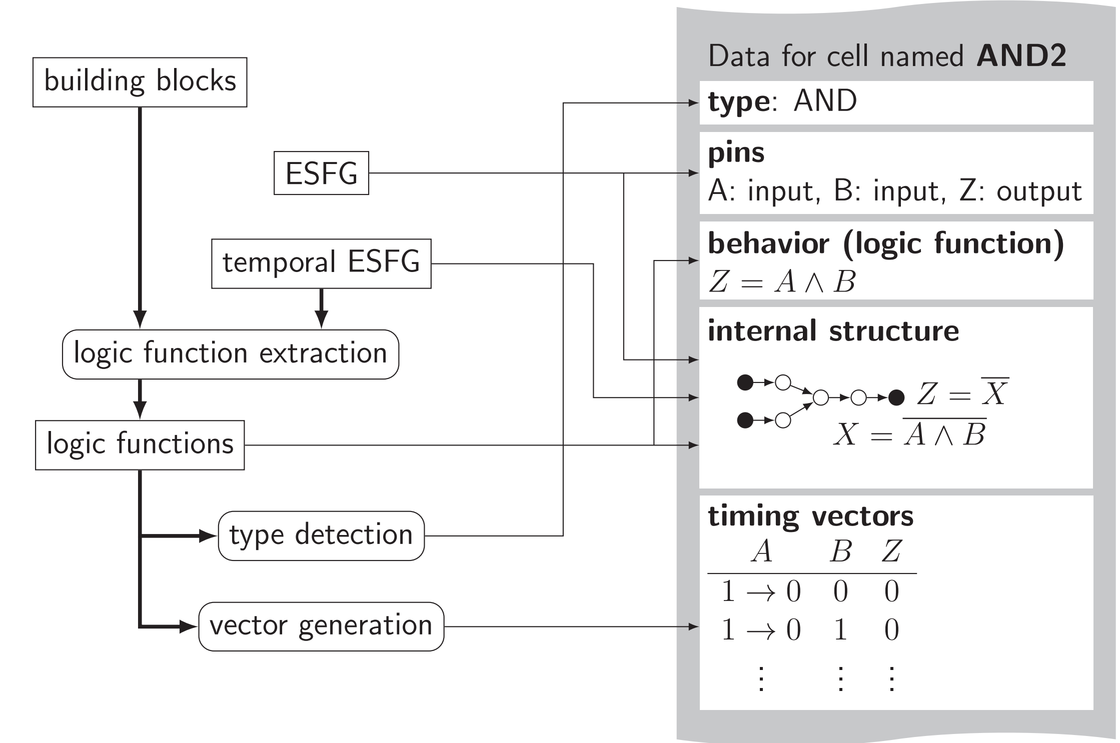


Figure 6.1.: Flow of input data generation

x	$\overline{x}$
0	1
1	0
Z	Z
U	U

(a)

			b		
	$\oplus$	0	1	Z	U
	0	0	U	0	U
	1	U	1	1	U
a	Z	0	1	Z	U
	U	U	U	U	U

(Bryant 1991)

(b)

			b		
	$\diamond$	0	1	Z	U
	0	0	U	0	0
	1	U	1	1	1
a	Z	0	1	Z	U
	U	0	1	U	U

(c)

			s		
	$\odot$	0	1	Z	U
	0	Z	Z	Z	Z
	1	0	U	Z	U
g	Z	U	U	Z	U
	U	U	U	Z	U

(d)

Table 6.1.: Operators for four valued logic.

6.1.1. Logic Function Extraction of Building Blocks

In general, CMOS circuits are composed of a pull–up and a pull–down network (Weste & Harris 2005). The logic function of these networks can be computed by investigating serial and parallel connections (Bryant 1991). The overall logic function is then computed by combining the logic functions of the pull–up and pull–down network.

In the method presented here, serial and parallel connections are already known from the building block analysis step (Chapter 3). Thus, the logic functions for single logic

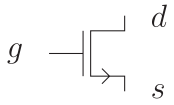
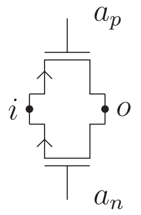
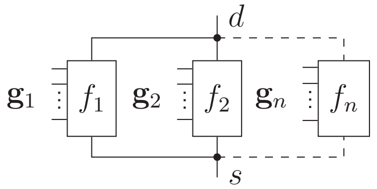
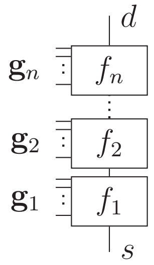
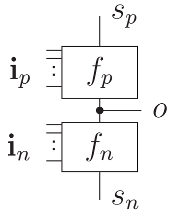
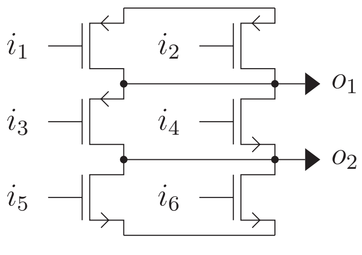
type of building block x		logic functions for x
trans		$d = f_{\mathrm{trans,n}}(g, s) = g \odot s$ nmos subtype $d = f_{\mathrm{trans,p}}(g, s) = \overline{\overline{g} \odot \overline{s}}$ pmos subtype
pg		$o = f_{\mathrm{pg}}(i, a, b) = \begin{cases} i & (a_n = 1) \wedge (a_p = 0) \\ Z & (a_n = 0) \wedge (a_p = 1) \\ U & \text{otherwise} \end{cases}$
la	 d $\mathbf{g}_1$ f_1 $\mathbf{g}_2$ f_2 $\mathbf{g}_n$ f_n s	$d = f_{\mathrm{la}}\big([\mathbf{g}_1\ \mathbf{g}_2 \cdots \mathbf{g}_n], s\big)$ $\quad = f_1(\mathbf{g}_1, s) \diamond f_2(\mathbf{g}_2, s) \diamond \cdots \diamond f_n(\mathbf{g}_n, s)$
sc	 d $\mathbf{g}_n$ f_n $\mathbf{g}_2$ f_2 $\mathbf{g}_1$ f_1 s	$d = f_{\mathrm{sc}}\big([\mathbf{g}_1\ \mathbf{g}_2 \cdots \mathbf{g}_n], s\big)$ $\quad = f_n(\mathbf{g}_n, \cdots f_2(\mathbf{g}_2, f_1(\mathbf{g}_1, s)) \cdots)$
lg	 s_p $\mathbf{i}_p$ f_p o $\mathbf{i}_n$ f_n s_n	$o = f_{\mathrm{lg}}\big([\mathbf{i}_p\ \mathbf{i}_n], s\big)$ $\quad = f_p(\mathbf{i}_p, s_p) \oplus f_n(\mathbf{i}_n, s_n)$
tcb	 i_1 i_2 o_1 i_3 i_4 o_2 i_5 i_6	$o_1 = \begin{cases} 0 & (i_4 = 1) \wedge \big((i_5 = 1) \vee (i_6 = 1)\big) \\ 1 & (i_1 = 0) \vee (i_2 = 0) \\ Z & (i_1 = 0) \wedge (i_2 = 0) \wedge (i_3 = 0) \wedge (i_4 = 1) \\ U & \text{otherwise} \end{cases}$ $o_2 = \begin{cases} 0 & (i_5 = 1) \vee (i_6 = 1) \\ 1 & (i_3 = 0) \wedge \big((i_1 = 0) \vee (i_2 = 0)\big) \\ Z & (i_5 = 1) \wedge (i_6 = 1) \wedge (i_3 = 0) \wedge (i_4 = 1) \\ U & \text{otherwise} \end{cases}$

Table 6.2.: Library of building blocks and corresponding logic functions.

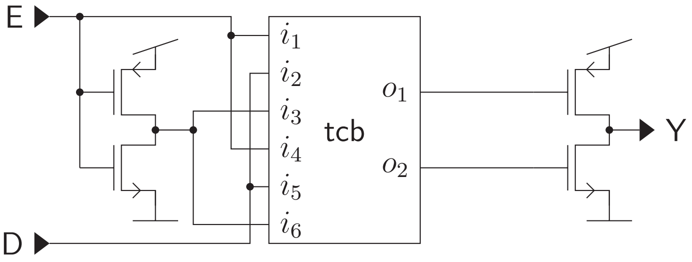


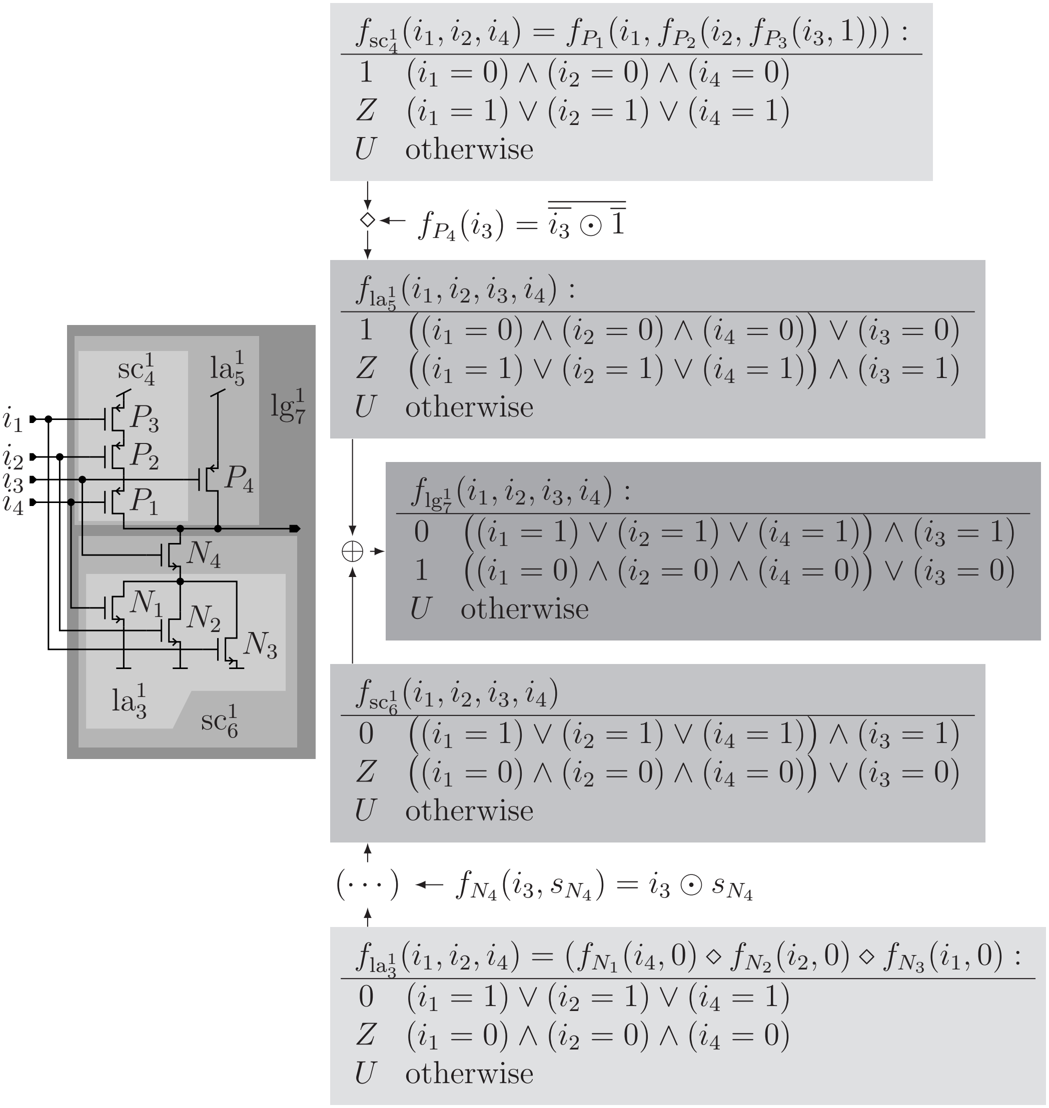
Figure 6.2.: Usage of tristate control block to implement a tristate buffer (Weste & Harris 2005, Fig 12.22).

gates can be computed. Table 6.2 illustrates the correspondence between building blocks and logic functions.

The truth tables of the used operators are given in Table 6.1. The bar operator defines a negation for four valued logic (Table 6.1a). Logic values 0 and 1 are swapped but the other values remain constant. Operator $\oplus$ is the merge operator defined by (Bryant 1991) (Table 6.1b). It is a or b in case one of them is in high impedance state or both have the same value. In all other cases it is U. Operator $\diamond$ is a variant of operator $\oplus$ (Table 6.1c). It considers that unknown states can be suppressed, e.g., in parallel connections. Operator $\odot$ models the behavior of the drain of an NMOS transistor in dependence of gate g and source s (Table 6.1d). It is 0 if the gate pin is 1 and the source pin is 0. It is Z if the gate pin is 0 or the source pin is Z. In all other cases it is U.

The logic function for the drain of a PMOS transistor is obtained by inverting inputs and outputs (see Table 6.2). The drain is 0 if the gate pin is 0 and the source pin is 1. The logic function for a *pass gate* is defined in the library. It transfers the value from the input to the output in case $a_n = 1$ and $a_p = 0$. In case of $a_n = 0$ and $a_p = 1$ nothing is transferred, i.e., it has a high–impedance output. In all other cases it is of unknown state. The logic function for a *logic array* is obtained by applying the $\diamond$ operator to its children. Vectors $\mathbf{g}_1$ to $\mathbf{g}_n$ denote the logic functions at the gate pins. The logic function for a *stack chain* is obtained by substituting the source input of logic function f_{i+1} of the $(i+1)$-th child by logic function f_i of the i-th child for $i = 1, 2, \ldots, n-1$. The logic function for a *logic gate* is obtained by combining the logic function f_p of the p–block and logic function f_n of the n–block. A *tristate control block* implements a certain type of tristate buffer by connecting the PMOS transistor of a logic gate to output o_1 and the NMOS transistor of a logic gate to output o_2 (Fig. 6.2). It holds, $i_1 = i_4 = E$, $i_3 = i_6 = \overline{E}$ and $i_2 = i_5 = D$. For $E = 0$ output $o_1 = 1$, output $o_2 = 0$ and $Y = Z$, i.e., output Y is in high impedance state. For $E = 1$ output $o_1 = o_2 = \overline{D}$ and $Y = D$. More details about the implementation can be found in (Eick & Graeb 2013).

Figure 6.3 illustrates the extraction process for the compound gate (Weste & Harris



$$\begin{array}{ll} \multicolumn{2}{l}{f_{\mathrm{sc}_4^1}(i_1, i_2, i_4) = f_{P_1}(i_1, f_{P_2}(i_2, f_{P_3}(i_3, 1))) :} \\ \hline 1 & (i_1 = 0) \wedge (i_2 = 0) \wedge (i_4 = 0) \\ Z & (i_1 = 1) \vee (i_2 = 1) \vee (i_4 = 1) \\ U & \text{otherwise} \end{array}$$

$$\diamond \leftarrow f_{P_4}(i_3) = \overline{\overline{i_3} \odot \overline{1}}$$

$$\begin{array}{ll} \multicolumn{2}{l}{f_{\mathrm{la}_5^1}(i_1, i_2, i_3, i_4) :} \\ \hline 1 & \big((i_1 = 0) \wedge (i_2 = 0) \wedge (i_4 = 0)\big) \vee (i_3 = 0) \\ Z & \big((i_1 = 1) \vee (i_2 = 1) \vee (i_4 = 1)\big) \wedge (i_3 = 1) \\ U & \text{otherwise} \end{array}$$

$$\oplus \rightarrow$$

$$\begin{array}{ll} \multicolumn{2}{l}{f_{\mathrm{lg}_7^1}(i_1, i_2, i_3, i_4) :} \\ \hline 0 & \big((i_1 = 1) \vee (i_2 = 1) \vee (i_4 = 1)\big) \wedge (i_3 = 1) \\ 1 & \big((i_1 = 0) \wedge (i_2 = 0) \wedge (i_4 = 0)\big) \vee (i_3 = 0) \\ U & \text{otherwise} \end{array}$$

$$\begin{array}{ll} \multicolumn{2}{l}{f_{\mathrm{sc}_6^1}(i_1, i_2, i_3, i_4)} \\ \hline 0 & \big((i_1 = 1) \vee (i_2 = 1) \vee (i_4 = 1)\big) \wedge (i_3 = 1) \\ Z & \big((i_1 = 0) \wedge (i_2 = 0) \wedge (i_4 = 0)\big) \vee (i_3 = 0) \\ U & \text{otherwise} \end{array}$$

$$(\cdots) \leftarrow f_{N_4}(i_3, s_{N_4}) = i_3 \odot s_{N_4}$$

$$\begin{array}{ll} \multicolumn{2}{l}{f_{\mathrm{la}_3^1}(i_1, i_2, i_4) = (f_{N_1}(i_4, 0) \diamond f_{N_2}(i_2, 0) \diamond f_{N_3}(i_1, 0) :} \\ \hline 0 & (i_1 = 1) \vee (i_2 = 1) \vee (i_4 = 1) \\ Z & (i_1 = 0) \wedge (i_2 = 0) \wedge (i_4 = 0) \\ U & \text{otherwise} \end{array}$$

Figure 6.3.: Computation of the logic function of the compound gate (Weste & Harris 2005, Fig. 1.18) from Fig. 3.3

2005, Fig. 1.18) from Fig. 3.3. In the p–block, logic function $f_{\mathrm{sc}_4^1}$ for serial chain sc_4^1 is computed by combining the logic functions for P_1, P_2 and P_3. Next, logic function $f_{\mathrm{sc}_4^1}$ is combined with logic function f_{P_4} of PMOS transistor P_4, yielding $f_{\mathrm{la}_5^1}$ for *logic array* la_5^1. In the n–block, logic function $f_{\mathrm{la}_3^1}$ for *logic array* la_3^1 is computed by combining the logic functions of N_1, N_2 and N_3. Next, logic function $f_{\mathrm{la}_3^1}$ is combined with logic function f_{N_4} of NMOS transistor N_4, yielding $f_{\mathrm{sc}_6^1}$ for *stack chain* sc_6^1. Finally, logic functions $f_{\mathrm{sc}_6^1}$ and $f_{\mathrm{la}_5^1}$ are combined to logic function $f_{\mathrm{lg}_7^1}$ for the complete *logic gate*.

For the latch from Fig. 3.16 the method computes the following logic functions. *Pass gates* pg^1 and pg^2 show the data from input i at the output in case the control inputs a_n and a_p are set properly,

$$f_{\mathrm{pg}_1}(i, a_p, a_n) = f_{\mathrm{pg}_2}(i, a_p, a_n) = \begin{cases} i & (a_n = 1) \wedge (a_p = 0) \\ Z & (a_n = 0) \wedge (a_p = 1) \\ U & \text{otherwise} \end{cases} . \tag{6.1}$$

All logic gates are inverters with logic function:

$$f_{\mathrm{lg}_1}(i_1) = \cdots = f_{\mathrm{lg}_4}(i_2) = \begin{cases} 0 & i_1 = 1 \\ 1 & i_1 = 0 \\ U & (i_1 = U) \vee (i_1 = Z) \end{cases} . \tag{6.2}$$

6.1.2. Overall Logic Function

The overall logic function is computed using temporal logic. This is done by assigning logic variables to each node of the temporal ESFG. Similar to the nodes of the temporal ESFG, the logic variables refer to different time steps. Next, the logic functions of the building blocks are substituted into each other by traversing the graph in topological order. This means every node is visited after all nodes it depends on. For this analysis, it is assumed that the inputs of the cell are in a defined logic state. The logic function of one node can be composed out of the logic function of multiple building blocks by using the resolution operator from Bryant (1991). Due to the correspondence of the temporal ESFG to a Mealy machine (Mealy 1955), the general logic function is,

$$\mathbf{z}(t) = \mathbf{f}_o\big(\mathbf{x}(t), \mathbf{s}(t-1)\big) \tag{6.3}$$

$$\mathbf{s}(t) = \mathbf{f}_s\big(\mathbf{x}(t), \mathbf{s}(t-1)\big) . \tag{6.4}$$

Vector $\mathbf{x}$ describes the input variables, vector $\mathbf{s}$ contains the state variables and vector $\mathbf{z}$ contains the output variables. Function $\mathbf{f}_o$ is the output function and function $\mathbf{f}_s$ is the transition function.

For the latch from Fig. 3.16 this yields the following logic functions.

$$a(t) = f_a(E(t)) = \begin{cases} 0 & E(t) = 1 \\ 1 & E(t) = 0 \end{cases} \tag{6.5}$$

$$\begin{aligned} Q(t) &= f_Q(D(t), E(t), b(t-1)) \\ &= \begin{cases} 0 & ((E(t) = 1) \wedge (D(t) = 0)) \vee ((E(t) = 0) \wedge (b(t-1) = 0)) \\ 1 & ((E(t) = 1) \wedge (D(t) = 1)) \vee ((E(t) = 0) \wedge (b(t-1) = 1)) \\ Z & ((E(t) = 0) \wedge (b(t-1) = Z)) \\ U & ((E(t) = 0) \wedge (b(t-1) = U)) \end{cases} \end{aligned} \tag{6.6}$$

$$\begin{aligned} \overline{Q}(t) &= f_{\overline{Q}}(D(t), E(t), b(t-1)) \\ &= \begin{cases} 1 & ((E(t) = 1) \wedge (D(t) = 0)) \vee ((E(t) = 0) \wedge (b(t-1) = 0)) \\ 0 & ((E(t) = 1) \wedge (D(t) = 1)) \vee ((E(t) = 0) \wedge (b(t-1) = 1)) \\ U & ((E(t) = 0) \wedge (b(t-1) \in \{U, Z\})) \end{cases} \end{aligned} \tag{6.7}$$

$$\begin{aligned} b(t) &= f_b(D(t), E(t), b(t-1)) \\ &= \begin{cases} 0 & ((E(t) = 1) \wedge (D(t) = 0)) \vee ((E(t) = 0) \wedge (b(t-1) = 0)) \\ 1 & ((E(t) = 1) \wedge (D(t) = 1)) \vee ((E(t) = 0) \wedge (b(t-1) = 1)) \\ U & ((E(t) = 0) \wedge (b(t-1) \in \{U, Z\})) \end{cases} \end{aligned} \tag{6.8}$$

$$\begin{aligned} \mathbf{f}_o(\mathbf{x}(t), \mathbf{s}(t-1)) &= \begin{bmatrix} f_Q(D(t), E(t), b(t-1)) \\ f_{\overline{Q}}(D(t), E(t), b(t-1)) \end{bmatrix} \text{ with } \mathbf{s} = \begin{bmatrix} b \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} D \\ E \end{bmatrix} \\ \mathbf{f}_s(\mathbf{x}(t), \mathbf{s}(t-1)) &= \begin{bmatrix} f_b(D(t), E(t), b(t-1)) \end{bmatrix} \end{aligned} \tag{6.9}$$

Logic function f_a of node a is the inverted input E. The logic function can not be U or Z because the input is assumed to be in a valid logic state. The logic functions of outputs Q and $\overline{Q}$ as well as of state b depend on inputs D and E and the value of state b in the previous time step. Output Q is D if $E(t) = 1$, else it is b in the previous time step. Output $\overline{Q}$ is the inverted of output Q. State b equals to Q but cases Z and U of Q are mapped to case U of b through the inverter.

6.1.3. Transformation to a Triggered Representation

In general, the expression for a triggered logic function looks like this (Accellera 2000, Section 5.2),

$$@g\big(\mathbf{x}_t\big)\mathbf{z} = \mathbf{f}(\mathbf{x}_c, \mathbf{z}) \; . \tag{6.10}$$

The outputs in vector $\mathbf{z}$ are equal to logic function $\mathbf{f}$ iff g is true. The inputs are partitioned into the inputs $\mathbf{x}_c$ of logic function f and inputs $\mathbf{x}_t$ of trigger function g.

```
1 $\Gamma \leftarrow \emptyset$
2 $\mathbf{F}\big(\mathbf{x}(t), \mathbf{x}(t-1)\big) \leftarrow \mathbf{f}_o\big(\mathbf{x}(t), \mathbf{f}_s\big(\mathbf{x}(t-1), \mathbf{U}\big)\big)$
3 $\mathbf{F}\big(\mathbf{x}_t(t), \mathbf{x}_c(t), \mathbf{x}_t(t-1), \mathbf{x}_c(t-1)\big) \leftarrow \text{findTriggerInputs}\big(\mathbf{F}(\mathbf{x}(t), \mathbf{x}(t-1))\big)$
4 for all $\mathbf{x}_{t1}, \mathbf{x}_{t2} \in \mathbb{B}^{n_t}$
5   $\tilde{\mathbf{F}}(\mathbf{x}_c(t), \mathbf{x}_c(t-1)) \leftarrow \mathbf{F}\big(\mathbf{x}_t(t) = \mathbf{x}_{t1}, \mathbf{x}_c(t), \mathbf{x}_t(t-1) = \mathbf{x}_{t2}, \mathbf{x}_c(t-1)\big)$
6   alwaysDefined($\tilde{\mathbf{F}}$)
7   true                                                        false
8     $\Gamma \leftarrow \Gamma \cup \{(\mathbf{x}_{t1}, \mathbf{x}_{t2}, \tilde{\mathbf{F}})\}$
9 buildTriggerFunctions($\Gamma$)
```

Figure 6.4.: Algorithm to compute the triggered logic function.

The trigger can contain conditions for specific levels, e.g., $x_1 = 1$ or specific edges $x_1 = 0 \rightarrow 1$; The algorithm computes the triggered representation in four steps (see Fig. 6.4).

First, logic function $\mathbf{F}$ is constructed that describes the output after two time steps (line 2). This allows to handle edge triggered and level triggered latches. The state variables are assumed to be undefined at any time before $t-1$. Next, this logic function is used to divide the inputs $\mathbf{x}$ into trigger input $\mathbf{x}_t$ and combinatorial inputs $\mathbf{x}_c$ using the following heuristic (line 3). Vector $\mathbf{x}_t$ is formed by all logic variables that appear in the logic function for $F_i = U$, where F_i is any of the components of $\mathbf{F}$. After that, the remaining combinatorial logic function $\tilde{\mathbf{F}}$ is computed for all possible assignments of $\mathbf{x}(t)$ and $\mathbf{x}(t-1)$ (lines 4 to 8). In case, $\tilde{F}_i = U$ can not be satisfied for any component $\tilde{F}_i$ of $\tilde{\mathbf{F}}$, the logic function is stored together with the assignment in set Γ. Finally, the triggered logic function is build out of set Γ (line 9). In doing so, equivalent entries are merged.

Lines 3 to 8 are implemented such that the assignment vectors $\mathbf{x}_{t1}$ and $\mathbf{x}_{t2}$ are generated incrementally. This speeds up cases where a partial assignment is already enough to decide about this assignment. At the same time, the necessary merge operations can be handled right away.

For the example from above, the algorithm yields the following results.

$$\mathbf{f}_s(\mathbf{x}(t-1), \mathbf{U}) = \begin{cases} 0 & ((E(t-1) = 1) \wedge (D(t-1) = 0)) \\ 1 & ((E(t-1) = 1) \wedge (D(t-1) = 1)) \\ U & (E(t-1) = 0) \end{cases} \tag{6.11}$$

$$
\begin{aligned}
F_1(\mathbf{x}(t), \mathbf{x}(t-1)) &= f_Q(D(t), E(t), \mathbf{f}_s(\mathbf{x}(t-1), \mathbf{U})) \\
&= \begin{cases} 0 & ((E(t) = 1) \wedge (D(t) = 0)) \vee \\ & ((E(t) = 0) \wedge (E(t-1) = 1) \wedge (D(t-1) = 0) \\ 1 & ((E(t) = 1) \wedge (D(t) = 1)) \vee \\ & ((E(t) = 0) \wedge (E(t-1) = 1) \wedge (D(t-1) = 1) \\ U & ((E(t) = 0) \wedge (E(t-1) = 0) \end{cases} \qquad (6.12) \\
F_2(\mathbf{x}(t), \mathbf{x}(t-1)) &= f_{\overline{Q}}(D(t), E(t), \mathbf{f}_s(\mathbf{x}(t-1), \mathbf{U})) \\
&= \begin{cases} 0 & ((E(t) = 1) \wedge (D(t) = 1)) \vee \\ & ((E(t) = 0) \wedge (E(t-1) = 1) \wedge (D(t-1) = 1) \\ 1 & ((E(t) = 1) \wedge (D(t) = 0)) \vee \\ & ((E(t) = 0) \wedge (E(t-1) = 1) \wedge (D(t-1) = 0) \\ U & ((E(t) = 0) \wedge (E(t-1) = 0) \end{cases} \qquad (6.13)
\end{aligned}
$$

This means Q will show the current value of D if E is set or the previous value of D if E was set in the first time step and cleared in the second. Expressions $F_1(\mathbf{x}(t), \mathbf{x}(t-1)) = U$ and $F_2(\mathbf{x}(t), \mathbf{x}(t-1)) = U$ depend only on E, which is selected as trigger input. The four possible assignments for E(t) and E(t-1) yield,

$$
\begin{aligned}
&\mathbf{E(t) = E(t-1) = 0} \\
&\qquad \tilde{F}_1(\mathbf{x}(t), \mathbf{x}(t-1)) = U \qquad \tilde{F}_2(\mathbf{x}(t), \mathbf{x}(t-1)) = U \qquad (6.14) \\
&\mathbf{E(t) = E(t-1) = 1} \\
&\qquad \tilde{F}_1(\mathbf{x}(t), \mathbf{x}(t-1)) = D(t) \qquad \tilde{F}_2(\mathbf{x}(t), \mathbf{x}(t-1)) = \begin{cases} 0 & D(t) = 1 \\ 1 & D(t) = 0 \end{cases} \qquad (6.15) \\
&\mathbf{E(t) = 0 \wedge E(t-1) = 1} \\
&\qquad \tilde{F}_1(\mathbf{x}(t), \mathbf{x}(t-1)) = D(t-1) \quad \tilde{F}_2(\mathbf{x}(t), \mathbf{x}(t-1)) = \begin{cases} 0 & D(t-1) = 1 \\ 1 & D(t-1) = 0 \end{cases} \qquad (6.16) \\
&\mathbf{E(t) = 1 \wedge E(t-1) = 0} \\
&\qquad \tilde{F}_1(\mathbf{x}(t), \mathbf{x}(t-1)) = D(t) \qquad \tilde{F}_2(\mathbf{x}(t), \mathbf{x}(t-1)) = \begin{cases} 0 & D(t) = 1 \\ 1 & D(t) = 0 \end{cases} \qquad (6.17)
\end{aligned}
$$

The assignment shown in Eq. (6.14) yields always unknown outputs and is therefore discarded. Assignments Eqs. (6.15) to (6.17) show that the latch is transparent in case E is set. Therefore, the complete triggered logic function is,

$$
@(E) \begin{bmatrix} Q \\ \overline{Q} \end{bmatrix} = \begin{bmatrix} D \\ \overline{D} \end{bmatrix} . \qquad (6.18)
$$

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

Table 6.3.: Results for different standard cell libraries.

6.2. Type Detection

The generic type of a cell describes whether the cell is an And, Or, Nand, Nor, Flip Flop, Latch, etc. It is detected by using the following criteria (Youssef 2010),

- **Structure**:
  A Flip Flop is detected if the ESFG of a cell has two or more cycles. A Latch is detected if the ESFG of a cell has exactly one cycle.
- **Logic function**:
  For combinatorial cells the computed logic functions are compared to a library of generic logic functions. Difficulties arise because the mapping between in- and outputs of computed and generic logic function is unknown.

6.3. Vector Generation

The vectors are generated based on a logic function for two time steps. The algorithm generates logic expressions for all combinations of switching input, rising or falling input edge, all outputs and all possible transitions at the output. Solution of this problem creates the required values for the remaining inputs.

6.4. Experimental Results

Table 6.3 summarizes the results for different standard cell libraries. Library 1 is the standard cell library included in the FreePDK (Stine et al. 2007). 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. The size of these libraries was between 30 and 850 cells in different driver strengths. The 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. They were always below one minute. The coverage column lists the amount of correctly handled cells.

This was determined by automatically checking if the result is reasonable, using the following criteria,

- the structural analysis software finished without error,
- there are no transistors that are not part of a *logic gate* or *pass gate*,
- the overall logic function exactly uses all inputs.

It can be seen, that this coverage is always above 95 %. The uncovered cells were not designed according to standard CMOS principles or require a more accurate modeling of the time behavior. A further limitation of the current method are differential logic cells. A more detailed investigation for different latch types was done by Guo (2012).

7. Application: Constraint Generation for Analog Circuits

Most EDA tools for the design steps analog sizing, placement and routing require so–called constraints as input (see Chapter 1). Examples for such constraints are inequalities for the electrical properties of transistors to ensure the correct operating point during sizing, or equalities for the geometrical parameters during placement to ensure symmetry. These constraints are not independent of each other. For example it is only possible to fulfill all constraints for matching during placement if the circuit was sized appropriately. Therefore, a new system of constraints is proposed in the following. This system is called generic constraints. It is based on the idea of capturing the intention of constraints rather than the exact mathematical equation. These generic constraints allow to generate the specific constraints for each tool automatically.

The overall constraint generation flow is depicted in Fig. 7.1. It starts from the circuit topology given as unsized netlist. Next, the structural analysis method described in Chapters 3 to 5 is executed yielding building block and symmetry information. This information is then used together with the netlist to generate the generic constraints. Based on these generic constraints, sizing constraints for the sizing tool and placement constraints for the placement tool are generated. The generation of routing constraints for a routing tool remains future work.

In the following, generic constraints are introduced. Next, the overall constraint generation flow for generic constraints as well as derived sizing and placement constraints are described. Finally, experimental results are presented.

7.1. Generic Constraints

In the following, generic constraints are described in an object–oriented manner. This results in the classes shown in Fig. 7.2. The general generic constraint is described by class *GenericConstraint*. It affects one or more devices. Based on the sizing constraints described by Eckmüller (1998), Zizala (2001) and Massier (2010) as well as the layout constraints described by Hastings (2001) and Eick (2008), the following classes of generic constraints can be identified:

Proximity The affected devices should be as close as possible in the layout.

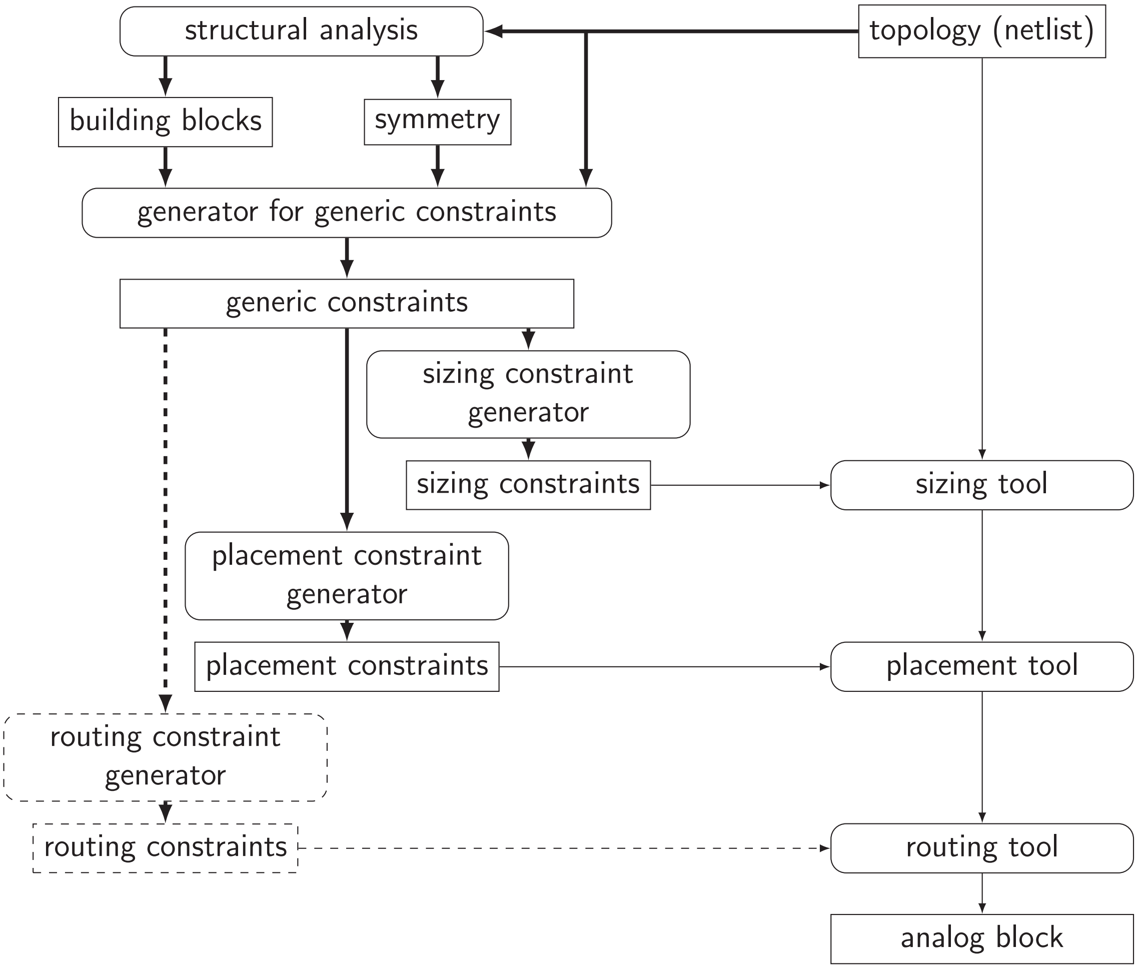


Figure 7.1.: Flow of constraint generation method.

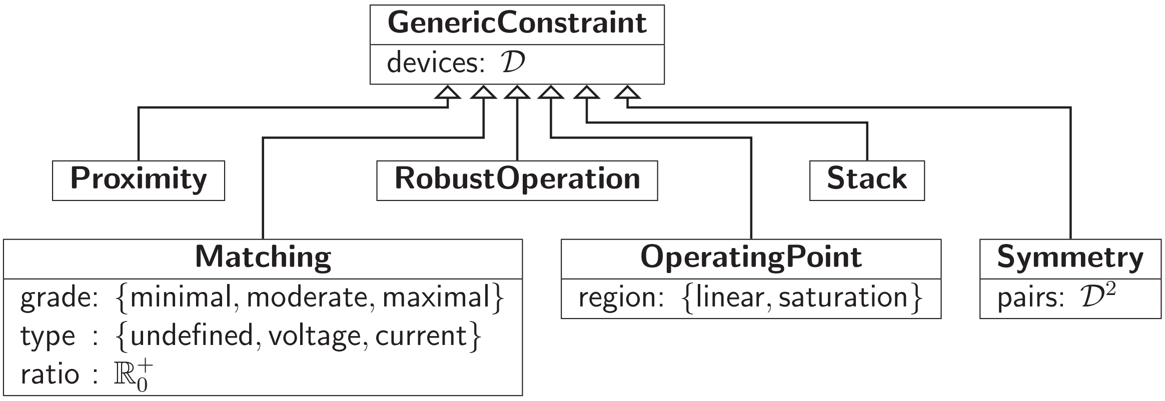


Figure 7.2.: Different classes of generic constraints.

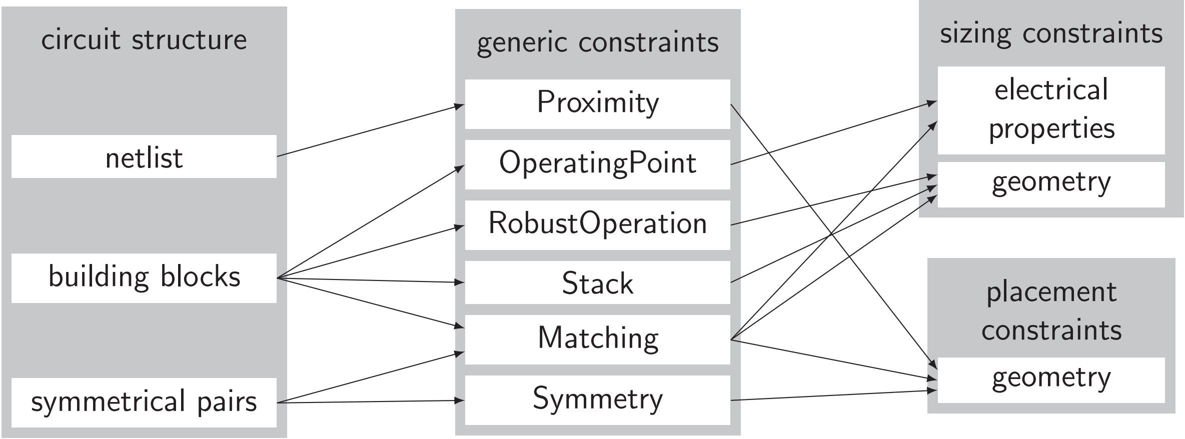


Figure 7.3.: Details of the constraint generation process

Matching Two devices should be matched, resulting in conditions for sizing and layout. Matching can be described in more detail through the attributes *grade*, *type* and *ratio*. The *grade* attribute determines how good the matching should be. The *type* attribute describes whether currents or voltages should be matched. The *ratio* attribute can be used to define a fixed ratio between the devices. Matched devices without fixed ratio are indicated by a value of 0.

RobustOperation A *RobustOperation* constraint describes conditions that are set for a single device to make its operation robust. A typical example are minimum area constraints.

OperatingPoint An *OperatingPoint* constraint determines the operating point of a single device. The attribute *region* describes the appropriate operating region.

Stack A *Stack* constraint is used to indicate that two transistors form a stack and can be laid out in a special way in case they are sized appropriately.

Symmetry The *Symmetry* constraint describes that multiple devices should be laid out with respect to a symmetry axis. The symmetry pairs are given by the *pairs* attribute.

7.2. Constraint Generation

Figure 7.3 shows the constraint generation process in more detail. The generation of generic, sizing and placement constraints is described in the following.

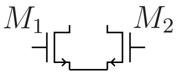
7.2.1. Generic Constraints

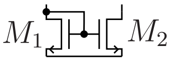
Generic constraints are generated from the topology of the circuit given by the netlist, the building blocks computed by the method from Chapter 3 and the symmetry computation described in Chapter 5. The netlist is used to generate proximity constraints for every pairs of devices that is connected.

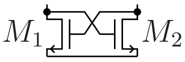
The building blocks found by the building block analysis are evaluated to generate generic constraints of classes *OperatingPoint*, *RobustOperation*, *Stack* and *Matching*. The details are given in Table 7.1. For a *differential pair* (dp), both transistors work in saturation region, have a very good voltage matching and a fixed width and length ratio. For a *simple current mirror* (scm), both transistors work in saturation and have current matching. For a *cross–coupled pair* (cc) both transistors work in saturation and must be matched with a fixed width and length ratio. For a *level shifter* (ls) both transistors work in saturation and have current matching. For a *four–transistor current mirror* (4tcm) the upper transistors work in saturation and have current matching. The lower transistors work in the linear region and also have current matching. For the *cascode current mirror* (ccm), the *wide–swing cascode current mirror* (wscm), *wide–swing current mirror* (wsm), the *improved wilson current mirror* (iwcm) and the *wilson current mirror* (wcm) the following holds: All transistors work in saturation. Transistors with the gate connected to the same net have current matching. A stack constraint can only be generated for the *wide–swing cascode current mirror* (wscm) and *four–transistor current mirror* (4tcm).

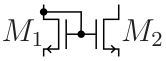
The symmetry information is used as follows. For each pair of symmetrical devices, a moderate matching constraint with a fixed ratio of one is generated. In addition, one symmetry constraint is generated for all symmetry pairs of the same symmetry axis.

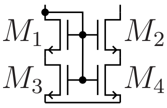
type of building block	generic constraints

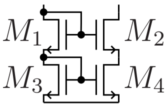

dp
	OperatingPoint(M_1,region=saturation)
	OperatingPoint(M_2,region=saturation)
	RobustOperation(M_1); RobustOperation(M_2)
	Matching(M_1,M_2,grade=maximal,type=voltage,ratio=1.0)


scm
	OperatingPoint(M_1,region=saturation)
	OperatingPoint(M_2,region=saturation)
	RobustOperation(M_1); RobustOperation(M_2)
	Matching(M_1,M_2,grade=minimal,type=current)


cc
	OperatingPoint(M_1,region=saturation)
	OperatingPoint(M_2,region=saturation)
	RobustOperation(M_1); RobustOperation(M_2)
	Matching(M_1,M_2,grade=minimal,ratio=1.0)


ls
	OperatingPoint(M_1,region=saturation)
	OperatingPoint(M_2,region=saturation)
	RobustOperation(M_1); RobustOperation(M_2)
	Matching(M_1,M_2,grade=minimal,type=current)

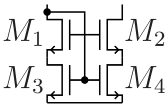

4tcm
	OperatingPoint(M_1,region=saturation)
	OperatingPoint(M_2,region=saturation)
	OperatingPoint(M_3,region=linear)
	OperatingPoint(M_4,region=linear)
	RobustOperation(M_1); RobustOperation(M_2)
	RobustOperation(M_3); RobustOperation(M_4)
	Matching(M_1,M_2,grade=minimal,type=current)
	Matching(M_3,M_4,grade=minimal,type=current)
	Stack(M_1,M_3); Stack(M_2,M_4)

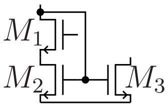

ccm
	OperatingPoint(M_1,region=saturation)
	OperatingPoint(M_2,region=saturation)
	OperatingPoint(M_3,region=saturation)
	OperatingPoint(M_4,region=saturation)
	RobustOperation(M_1); RobustOperation(M_2)
	RobustOperation(M_3); RobustOperation(M_4)
	Matching(M_1,M_2,grade=minimal,type=current)
	Matching(M_3,M_4,grade=minimal,type=current)

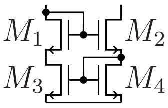
Continued on next page ...

Table 7.1.: Generic constraints generated depending on the building block type.

type of building block	generic constraints
	Continued from previous page ...


wscm
	OperatingPoint(M_1,region=saturation)
	OperatingPoint(M_2,region=saturation)
	OperatingPoint(M_3,region=saturation)
	OperatingPoint(M_4,region=saturation)
	RobustOperation(M_1); RobustOperation(M_2)
	RobustOperation(M_3); RobustOperation(M_4)
	Matching(M_1,M_2,grade=minimal,type=current)
	Matching(M_3,M_4,grade=minimal,type=current)
	Stack(M_1,M_3); Stack(M_2,M_4)


wsm
	OperatingPoint(M_1,region=saturation)
	OperatingPoint(M_2,region=saturation)
	OperatingPoint(M_3,region=saturation)
	RobustOperation(M_1); RobustOperation(M_2)
	RobustOperation(M_3)
	Matching(M_2,M_3,grade=minimal,type=current)


iwcm
	OperatingPoint(M_1,region=saturation)
	OperatingPoint(M_2,region=saturation)
	OperatingPoint(M_3,region=saturation)
	OperatingPoint(M_4,region=saturation)
	RobustOperation(M_1); RobustOperation(M_2)
	RobustOperation(M_3); RobustOperation(M_4)
	Matching(M_1,M_2,grade=minimal,type=current)
	Matching(M_3,M_4,grade=minimal,type=current)

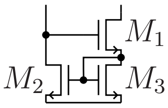
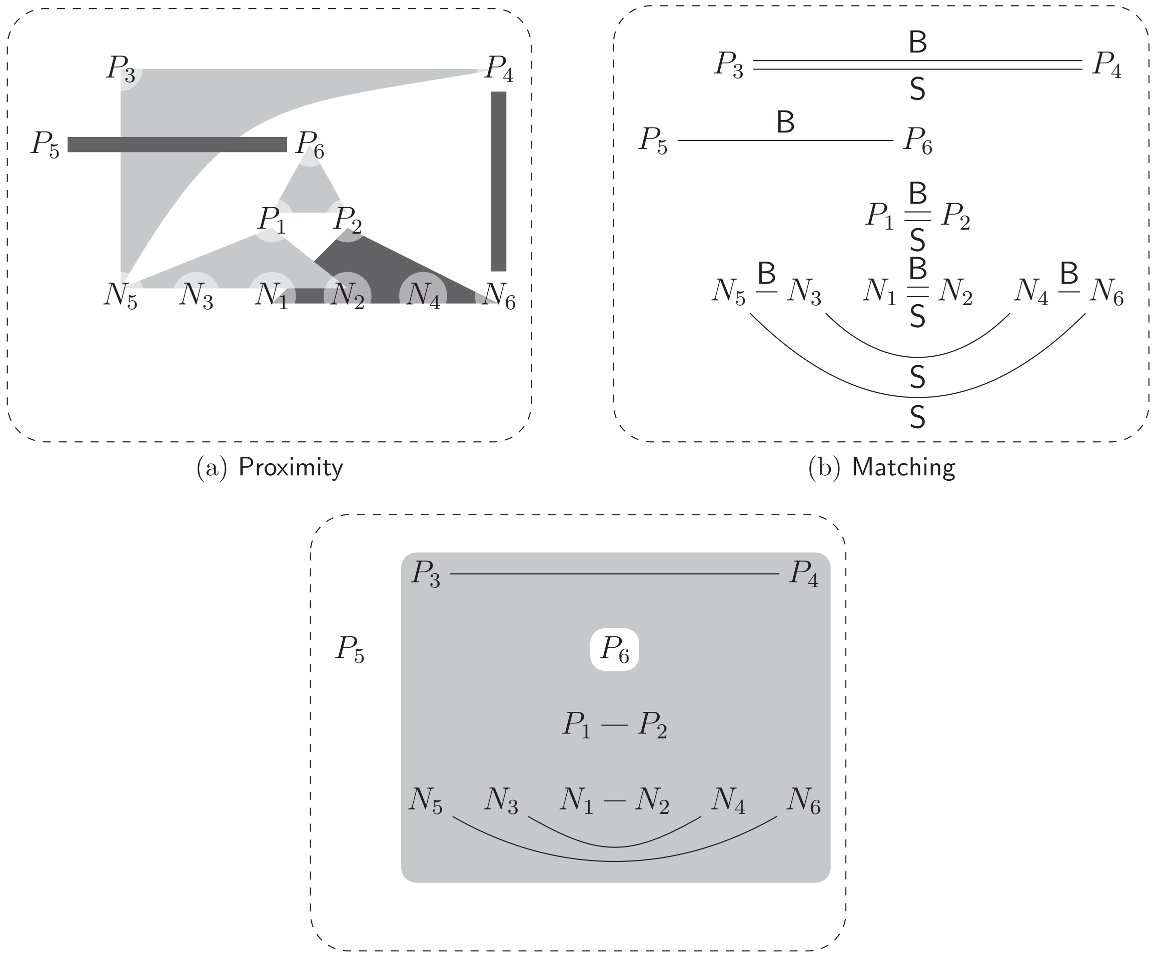

wcm
	OperatingPoint(M_1,region=saturation)
	OperatingPoint(M_2,region=saturation)
	OperatingPoint(M_3,region=saturation)
	RobustOperation(M_1); RobustOperation(M_2)
	RobustOperation(M_3)
	Matching(M_2,M_3,grade=minimal,type=current)

Table 7.1.: Generic constraints generated depending on the building block type.



(a) Proximity

(b) Matching

(c) Symmetry

Figure 7.4.: Generic constraints generated for the example circuit from Fig. 3.14.

The generated generic constraints for the example circuit from Fig. 3.14 are shown in Fig. 7.4. Each shaded area in Fig. 7.4a symbolizes one proximity constraint generated from the netlist. Each line in Fig. 7.4b symbolizes a matching constraint. The lines marked with B are generated from building blocks (see Fig. 3.14) and the lines marked with S are generated from symmetrical device pairs (see Eq. (5.40)). Transistors N_1 and N_2, P_1 and P_2 as well as P_3 and P_4 are subject to a *Matching* constraint from a building block and from a symmetrical device pair. For such cases, a new *Matching* constraint could be generated. The attributes of this constraint could be selected such that both constraints are covered. The gray area in Fig. 7.4c marks the generated symmetry constraint. The lines symbolize the symmetry pairs.

generic constraint			sizing constraints	
OperatingPoint(M)				
	region =	saturation	$v_{\mathrm{ds},M} - (v_{\mathrm{gs},M} - V_{\mathrm{th},M}) \geq V_{\mathrm{sat,min}}$	(7.1)
			$v_{\mathrm{ds},M} \geq 0$	(7.2)
			$v_{\mathrm{gs},M} - V_{\mathrm{th},M} \geq 0$	(7.3)
		linear	$(v_{\mathrm{gs},M} - V_{\mathrm{th},M}) - v_{\mathrm{ds},M} \geq V_{\mathrm{lin,min}}$	(7.4)
			$v_{\mathrm{ds},M} \geq 0$	(7.5)
			$v_{\mathrm{gs}} - V_{\mathrm{th}} \geq 0$	(7.6)
RobustOperation(M)			$W_M \cdot L_M \geq A_{\mathrm{min}}$	(7.7)
			$W_M \geq W_{\mathrm{min}}$	(7.8)
			$L_M \geq L_{\mathrm{min}}$	(7.9)
Matching(M_1,M_2)			$L_{M_1} = L_{M_2}$	(7.10)
			$\lvert v_{\mathrm{gs},M_1} - v_{\mathrm{gs},M_2} \rvert \leq \Delta V_{\mathrm{gs,max}}$	(7.11)
			$\lvert v_{\mathrm{ds},M_1} - v_{\mathrm{ds},M_2} \rvert \leq \Delta V_{\mathrm{ds,max}}$	(7.12)
	grade =	minimal	$\Delta V_{\mathrm{gs,max}}, \Delta V_{\mathrm{ds,max}}$ large	
		moderate	$\Delta V_{\mathrm{gs,max}}, \Delta V_{\mathrm{ds,max}}$ small	
		maximal	$\Delta V_{\mathrm{gs,max}}, \Delta V_{\mathrm{ds,max}}$ very small	
	type =	current	$\lvert v_{\mathrm{gs},M_1} - V_{\mathrm{th},M_1} \rvert \geq V_{\mathrm{ov,min}}$	(7.13)
			$\lvert v_{\mathrm{gs},M_2} - V_{\mathrm{th},M_2} \rvert \geq V_{\mathrm{ov,min}}$	(7.14)
		voltage	$\lvert v_{\mathrm{gs},M_1} - V_{\mathrm{th},M_1} \rvert \leq V_{\mathrm{ov,max}}$	(7.15)
			$\lvert v_{\mathrm{gs},M_2} - V_{\mathrm{th},M_2} \rvert \leq V_{\mathrm{ov,max}}$	(7.16)
	ratio $\neq$	0	ratio $\cdot W_{M_1} = W_{M_2}$	(7.17)
Stack(M_1, M_2)			$W_{M_1} = W_{M_2}$	(7.18)

Table 7.2.: Sizing constraints generated from generic constraints.

7.2.2. Sizing Constraints

In the following, the generation of sizing constraints from generic constraints is described. Therefore all generic constraints of classes *OperatingPoint*, *RobustOperation*, *Matching* and *Stack* are evaluated. The generation rules are summarized in Table 7.2.

For the *OperatingPoint* constraint, inequalities for the saturation and linear region are generated depending on the value of the *region* attribute. These inequalities can be found in every circuit design book, e.g., Sansen (2007).

For the *RobustOperation* constraint, lower limits for width W_M, length L_M and area $W_M \cdot L_M$ of transistor M are generated. This is based on Pelgrom et al. (1989) and general considerations about mismatch (Eckmüller 1998).

For the *Matching* constraint, an equal length constraint for transistors M_1 and M_2 is generated independent of the attributes set. The difference between the gate–source and drain–source voltages of M_1 and M_2 is limited to $\Delta V_{\mathrm{gs,max}}$ and $\Delta V_{\mathrm{ds,max}}$, respectively. Constants $\Delta V_{\mathrm{gs,max}}$ and $\Delta V_{\mathrm{ds,max}}$ are chosen based on the *grade* attribute and the technology. In case of current matching, the gate–source voltages of M_1 and M_2 must exceed the threshold voltage at least by $V_{\mathrm{ov,min}}$. In case of voltage matching, the gate–source voltages of M_1 and M_2 must not exceed the threshold voltage by more than $V_{\mathrm{ov,max}}$. In case the *ratio* is non–zero, a corresponding constraint for the widths is generated. These conditions are based on Eckmüller (1998) and Hastings (2001).

The *Stack* constraint yields a sizing constraint about equal width of both transistors. This is based on Eckmüller (1998).

Based on Tables 7.1 and 7.2, the following constraints are generated for a *simple current mirror*:

OperatingPoint(M_1,region=saturation)

$$v_{\mathrm{ds},M_1} - (v_{\mathrm{gs},M_1} - V_{\mathrm{th},M_1}) \geq V_{\mathrm{sat,min}} \qquad v_{\mathrm{ds},M_1} \geq 0 \tag{7.19}$$
$$v_{\mathrm{gs},M_1} - V_{\mathrm{th},M_1} \geq 0 \tag{7.20}$$

OperatingPoint(M_2,region=saturation)

$$v_{\mathrm{ds},M_2} - (v_{\mathrm{gs},M_2} - V_{\mathrm{th},M_2}) \geq V_{\mathrm{sat,min}} \qquad v_{\mathrm{ds},M_2} \geq 0 \tag{7.21}$$
$$v_{\mathrm{gs},M_2} - V_{\mathrm{th},M_2} \geq 0 \tag{7.22}$$

RobustOperation(M_1)

$$W_{M_1} \cdot L_{M_1} \geq A_{\mathrm{min}} \qquad W_{M_1} \geq W_{\mathrm{min}} \tag{7.23}$$
$$L_{M_1} \geq L_{\mathrm{min}} \tag{7.24}$$

RobustOperation(M_2)

$$W_{M_1} \cdot L_{M_1} \geq A_{\mathrm{min}} \qquad W_{M_1} \geq W_{\mathrm{min}} \tag{7.25}$$
$$L_{M_1} \geq L_{\mathrm{min}} \tag{7.26}$$

Matching(M_1,M_2,grade=minimal,type=current)

$$L_{M_1} = L_{M_2} \tag{7.27}$$
$$|v_{\mathrm{gs},M_1} - v_{\mathrm{gs},M_2}| \leq \Delta V_{\mathrm{gs,max}} \qquad |v_{\mathrm{ds},M_1} - v_{\mathrm{ds},M_2}| \leq \Delta V_{\mathrm{ds,max}} \tag{7.28}$$
$$|v_{\mathrm{gs},M_1} - V_{\mathrm{th},M_1}| \geq V_{\mathrm{ov,min}} \qquad |v_{\mathrm{gs},M_2} - V_{\mathrm{th},M_2}| \geq V_{\mathrm{ov,min}} \tag{7.29}$$

This complies with the constraints formulated by Eckmüller (1998), Zizala (2001) and Massier (2010) except the constraint for the difference between the gate–source voltages. However, this constraint is always fulfilled since $v_{\mathrm{gs},M_1} = v_{\mathrm{gs},M_2}$ for the *simple current mirror*.

generic constraint			sizing constraints	
Proximity(M_1,M_2)			distance(M_1, M_2) $\leq d_{\max}$	(7.30)
Stack(M_1, M_2)			merge transistors	
Matching(M_1,M_2)			$W_{f,M_1} = W_{f,M_2}$	(7.31)
			$\varphi_{M_1} = \varphi_{M_2}$	(7.32)
	grade =	minimal	align M_1 , M_2	(7.33)
		moderate	$n_{f,M_1} = n_{f,M_2}$	(7.34)
			$n_{p,M_1} > 1 \quad n_{p,M_2} > 1$	(7.35)
			common–centroid M_1 , M_2	(7.36)
		maximal	$n_{f,M_1} = n_{f,M_2}$	(7.37)
			$n_{p,M_1} > 1 \quad n_{p,M_2} > 1$	(7.38)
			common–centroid M_1 , M_2(quadratic)	(7.39)
	ratio $\neq$	0	ratio $\cdot\, n_{f,M_1} \cdot n_{p,M_1} = n_{f,M_2} \cdot n_{p,M_2}$	(7.40)
Symmetry($(M_1, M_1'), (M_2, M_2'), \ldots, (M_n, M_n')$)				
			$\forall_{i,j \in \{1,\ldots,n\}}\, x_{M_i} + x'_{M_i} = x_{M_j} + x'_{M_j}$	(7.41)
			$\forall_{i \in \{1,\ldots,n\}}\, y_{M_i} = y_{M_i'}$	(7.42)

Table 7.3.: Placement constraints generated from generic constraints.

7.2.3. Placement Constraints

Similar to the sizing constraints, the placement constraints are directly generated from the generic constraints. The generation rules are summarized in Table 7.3.

A *Proximity* constraint yields an upper limit of the distance of transistors M_1 and M_2. A *Stack* constraint should result in a transistor with a double gate if placement method and the process development kit support this (Eckmüller 1998).

A *Matching* constraint results in several conditions for the placement (Hastings 2001). It is assumed that every transistor M can be divided into a number $n_{p,M}$ of parallel sub–transistors. Each of these sub–transistors consists of $n_{f,M}$ fingers of width $W_{f,M}$. The total width of transistor M is then calculated as,

$$W_M = n_{p,M} \cdot n_{f,M} \cdot W_{f,M} \tag{7.43}$$

In all cases, the finger width W_{f,M_1} of M_1 and W_{f,M_2} of M_2 should be equal, as well as the orientations φ_{M_1} and φ_{M_2}. *Minimal* matching is achieved if all sub–transistors of M_1 and M_2 are aligned in one direction. For *moderate* matching, a common–centroid configuration should be used. This requires that the number of fingers is equal for all sub–transistors of M_1 and M_2. In addition, M_1 and M_2 must be split into more than one sub–transistor. For *maximal* matching, this common–centroid should be as quadratic as possible. In case the *ratio* attribute is set to a non–zero value, the ratio should again be reflected in the ratio of total width of M_1 and M_2. Since both finger widths are equal, this translates to the ratio of the products $n_{p,M_1} \cdot n_{f,M_1}$ and $n_{p,M_2} \cdot n_{f,M_2}$.

A *Symmetry* constraint is always defined for a number n of symmetry pairs (M_1, M_1') to (M_n, M_n'). Table 7.3 lists the constraints for a vertical axis. The coordinate of this axis is calculated by $(x_{M_i} + x_{M_i'})/2$, where x_{M_i} and $x_{M_i'}$ are the x–coordinates of the center of gravity of all sub–transistors of M_i and M_i', respectively. This axis coordinate must be equal for all symmetry pairs. In addition, the center of gravity of M_i and M_i' must be aligned in direction of the y–coordinate.

For the *simple current mirror* example from above, this results in the following placement constraints:

Proximity(M_1,M_2)

$$\text{distance}(M_1, M_2) \leq d_{\max} \tag{7.44}$$

Matching(M_1,M_2,grade=minimal,type=current)

$$W_{f,M_1} = W_{f,M_2} \qquad \varphi_{M_1} = \varphi_{M_2} \tag{7.45}$$

$$\text{align } M_1, M_2 \tag{7.46}$$

7.3. Experimental Results

In the following, experimental results for the generation of generic constraints are shown. After that, the effect of different constraint sets to automatic sizing and placement methods is investigated.

7.3.1. Generic Constraints

Table 7.4 lists the number of generic constraints generated for the test cases from Table 3.2 using the method described in Section 7.1. The number of constraints is broken down to the constraint classes *Proximity*, *Matching*, *RobustOperation*, *OperatingPoint*, *Stack* and *Symmetry*. For the class *Matching*, the distribution to the different values of attributes *grade*, *type* and *ratio* is shown. For the class *OperatingPoint*, the distribution to the different values of the *region* attribute is shown. In some cases, the method described in Section 7.1 generates a constraint of the same class more than once for the same set of devices. For example, if two *simple current mirrors* share one input transistor, an *OperatingPoint* and *RobustOperation* constraint is generated for each *simple current mirror*. Similar, two *Matching* constraints are

item		circuit						
		1	2	3	4	5	6	7
Proximity		16	22	15	12	6	12	12
Matching		19	32	27	11	11	14	18
grade=	minimal	13	21	13	1	5	6	11
	moderate	5	10	11	7	5	6	6
	maximal	1	1	3	3	1	2	1
type=	undefined	5	10	11	7	6	6	6
	current	13	21	13	1	4	6	11
	voltage	1	1	3	3	1	2	1
ratio=	undefined	13	21	13	10	4	6	11
	1	6	11	14	1	7	8	7
RobustOperation		30	44	32	8	10	16	25
OperatingPoint		30	44	32	8	10	16	25
region=	linear	2	4	0	0	0	0	0
	saturation	28	40	32	8	10	16	25
Stack		2	4	0	0	0	0	0
Symmetry		1	1	1	1	1	1	1

Table 7.4.: Number of generated generic constraints for the analog test cases from Table 3.2.

generated for two transistors that form a *simple current mirror* and a symmetrical device pair. Because of this, the number of listed *OperatingPoint* and *RobustOperation* constraints is greater than the number of devices in Table 7.4.

Furthermore, the generic constraints generated from symmetrical device pairs can be clearly identified in Table 7.4. These constraints have *type undefined*, *grade minimal* and a *ratio* of 1. *Differential pairs* contribute *Matching* constraints of *type voltage*, *grade maximal* and *ratio 1*. Thus, the overall number of *Matching* constraints with a *ratio* of 1 is given by the number of *differential pairs* plus the number of symmetrical device pairs.

7.3.2. Sizing

In order to evaluate the influence of different constraint sets on quality and speed of numerical circuit optimization, Circuits 1 to 4 from Table 3.2 are selected as representative circuits. The following experiments were performed,

1. Three different constraint sets are generated:

   COM Set COM is generated using a commercial tool.

item	set	circuit							
		1		2		3		4	
constraint generation									
building blocks									
no. eq. / ineq.	COM	2/	0	2/	0	8/	0	4/	0
	SRM	21/	144	22/	188	19/	182	3/	24
	NEW	21/	144	22/	188	19/	182	3/	24
symmetry pairs									
no. eq. / ineq.	COM	10/	0	18/	0	10/	0	2/	0
	SRM	0/	0	0/	0	0/	0	0/	0
	NEW	10/	20	20/	40	22/	44	12/	24
total									
no. eq. / ineq.	COM	12/	0	20/	0	18/	0	6/	0
	SRM	21/	144	22/	188	19/	182	3/	24
	NEW	31/	164	42/	228	41/	226	15/	48
feasibility optimization									
no. sim. / time [h:min]	SRM	25/	0:03	342/	0:51	74/	0:10	37/	0:02
	NEW	25/	0:03	62/	0:10	76/	0:11	21/	0:01
speed–up	NEW		1×		5×		1×		2×
nominal optimization									
specification fulfilled	SRM		✓		failed		✓		✓
	NEW		✓		✓		✓		✓
no. sim./time [h:min]	SRM	555/	0:42	2802/	4:47	1301/	1:45	794/	0:54
	NEW	1/	0:01	450/	0:32	567/	0:48	241/	0:17
speed–up	NEW		40×		6×		2.3×		3.3×
yield optimization									
worst–case distance	SRM		3.3σ		0.0σ		3.5σ		1.1σ
	NEW		5.2σ		3.6σ		4.1σ		3.9σ
achieved yield	SRM		99.97%		0.0%		99.98%		85.7%
	NEW		> 99.99%		99.99%		> 99.99%		99.99%
no. sim. [1000] /	SRM	9.3/	13:25	—/	—	16/	20:51	9.5/	48:15
time [h:min]	NEW	4.2/	5:29	16/	20:51	7/	7:18	4.1/	12:26
speed–up	NEW		2.4×		∞×		2.9×		3.9×

Table 7.5.: Results of sizing experiments.

SRM Set SRM is generated by using above generation flow but generating only constraints from building blocks. This corresponds to the constraints generated by the sizing rules method (Massier 2010).

NEW Set NEW is generated by using above generation flow.

2. Setups for the circuit optimization tool WiCkeD (WiCkeD 6.3 2010, Antreich et al. 2000) are created for sets SRM and NEW.

3. Nominal sizing is performed for sets SRM and NEW.

4. Yield optimization is performed for both sets.

The results are summarized in Table 7.5. It can be observed that the commercial tool (COM) generate equalities only, while the sizing rules method (SRM) and the new approach (NEW) generate equalities and inequalities. Approaches COM and NEW generate constraints from symmetry pairs while method SRM generates constraints for building blocks only. Overall, approach NEW always generates the highest number of constraints for all test cases.

The effect of this can be seen in the subsequent experiments. They compare quality and speed of feasibility optimization, nominal optimization and yield optimization for the constraint sets generated by SRM and NEW. Feasibility optimization tries to find a sizing of the circuits where all sizing constraints hold true. Nominal optimization tries to find a sizing of the circuit, such that all performance specification are fulfilled and all sizing constraints hold true. Yield optimization tries to optimize parametric yield.

For both constraint sets feasibility optimization was successful. Runtime and simulation count are similar for both methods and Circuits 1, 3 and 4. For Circuit 2 the feasibility optimization takes significantly longer if set SRM is used.

Nominal optimization was successful for both constraint sets and all circuits, except Circuit 2, where no solution is found for Circuit 2 if constraint set SRM is used. In contrast to feasibility optimization, nominal optimization always took less time and less simulations if constraint set NEW was used. Overall, this results in a speed–up of up to 40 times for Circuit 1, where the specification was already fulfilled after feasibility optimization for set NEW.

The quality of the resulting circuit after yield optimization is measured using the parametric yield and the worst–case distance (Graeb 2007). For Circuit 2 no yield optimization was done using constraint set SRM because the preceding nominal optimization failed. A worst–case distance of 3σ means that the circuit corresponds to a 3σ design. A larger worst–case distance means that the circuit is more robust. It can be observed that yield and worst–case distance are always highest for circuit sized using constraint set NEW. For example, for Circuit 4 yield is 99.99% instead of 85.7% and the achieved worst–case distance is 3.9σ instead of 1.1σ The runtime is always smaller if constraint set NEW is used. For Circuit 4 the runtime is reduced from 2 days to half a day. The minimum speed–up is 2.4, which was achieved for Circuit 1.

7.3.3. Placement

For the evaluation of the influence of different constraint sets on placement quality, a set of five test circuits, Circuit L1 to Circuit L5, is used. This set differs to the previous experiments because a different technology was used to realize the circuits. Circuit L1 is a Miller amplifier (Laker & Sansen 1994), Circuit L2 is a symmetrical OTA (Laker & Sansen 1994), Circuit L3 is a fully differential amplifier similar to the circuit published by Galdi et al. (2008). Circuit L4 is a folded cascode amplifier (Laker & Sansen 1994). Circuit L5 is similar to the buffer amplifier published by Fisher & Koch (1987). They have a size between 9 and 42 devices. The symmetry computation method used for constraint generation is approach PROP from Section 5.4. This early version of the approach used in this thesis can not handle all of the test circuits used above. For the commercial tool an earlier version is used than for the experiments above. However, the results still give a good idea about the influence of different constraint sets to the post–layout performances.

For each of these circuits, a placement was generated using the method of Strasser (2011). This placement was then routed using a general purpose router. After that DRC faults were fixed manually and parasitics were extracted. Finally, post–layout performances were simulated and the yield was determined using Monte–Carlo simulation. For each circuit this was done for three different constraint sets:

unconstrained No constrained were generated. The hierarchical partitioning required for the method of Strasser (2011) was generated randomly.

COM Constraints were generated using a commercial tool. The hierarchical partitioning required for the method of Strasser (2011) was generated manually based on the generated constraints and the rules provided in Eick et al. (2011) and Strasser et al. (2011).

NEW Constraints were generated using the approach described in this thesis. Common–centroid structures were used for all matching constraints. The hierarchical partitioning required for the method of Strasser (2011) was generated according to Eick et al. (2011) and Strasser et al. (2011).

The sizes of the *generated* constraint sets are listed in Table 7.6. It can be seen that method COM generates fewer constraints than the new method. A closer investigation shows that method COM misses some important matching constraints due to its insufficient structural analysis. In some cases, method COM generates constraints that are in conflict. For example, a symmetry constraints with a vertical axis was generated for two devices, implying vertical alignment of these devices. But the method also generated a horizontal alignment constraint for the same two devices.

item	set			circuit		
		L1	L2	L3	L4	L5
no. devices		9	10	30	22	42
constraints						
total	COM	8	18	20	13	34
	NEW	13	19	35	38	53
conflicts	COM	1	1	1	0	0
	NEW	0	0	0	0	0
layout properties						
area [$10^4 \mu m^2$]	unconstrained	0.94	1.6	1.2	0.41	6.1
	COM	0.90	1.6	1.4	0.44	6.9
	NEW	0.94	1.6	1.3	0.50	6.5
total netlength [mm]	unconstrained	2.1	2.8	7.2	3.2	25
	COM	1.9	2.7	7.2	3.0	24
	NEW	2.3	3.4	10.8	3.7	30
simulated performance errors (post–layout vs. pre–layout)						
$\epsilon^{(\text{abs})}_{V_{\text{offset}}}$ [mV]	unconstrained	-7.8	-4.6	-0.21	-0.45	-3.4
	COM	-0.011	-1.1	-0.029	-1.55	-0.81
	NEW	-0.016	-1.1	-0.0085	-0.31	-0.34
$\epsilon^{(\text{rel})}_{\text{CMRR}}$	unconstrained	−3.4%	−6.1%	−72%	−25%	−21%
	COM	−1.1%	+6.1%	−72%	−12%	−8.3%
	NEW	−1.1%	+6.1%	−63%	−9.7%	±0%
$\epsilon^{(\text{rel})}_{A_0}$	unconstrained	−1.1%	+1.0%	−1.4%	+1.8%	+5.3%
	COM	±0%	+1.0%	±0%	+1.8%	±0%
	NEW	±0%	±0%	−1.4%	+1.8%	+6.5%
$\epsilon^{(\text{rel})}_{f_T}$	unconstrained	±0%	−33%	±0%	+5.3%	+36%
	NEW	−2.2%	±0%	+4.5%	+3.5%	−7.1%
	HPR	−2.2%	±0%	±0%	+3.5%	+36%
yield						
	COM	99.9%	99.6%	92%	84 %	88 %
	NEW	99.9%	99.6%	96%	99 %	93 %

Table 7.6.: Results of placement experiments.

After placing and routing of the circuits, area and total netlength are determined. It can be observed, that the total areas are similar. Constraint set NEW defines common centroid structures, but constraint set COM does not. Therefore the total netlength is largest because additional routing is required in the common centroid structures.

After parasitic extraction, absolute value of the offset voltage $|V_{\text{offset}}|$, the common mode rejection ratio (CMRR), static gain A_0 as well as transit frequency f_T were determined by simulation and compared to the pre–layout values. Offset error and CMRR are known to be mismatch sensitive (Laker & Sansen 1994). Therefore, a large influence of the layout on offset error and CMRR are to be expected.

For the offset error, Table 7.6 lists the absolute error $\epsilon^{(\text{abs})}_{V_{\text{offset}}}$,

$$\epsilon^{(\text{abs})}_{V_{\text{offset}}} = |V_{\text{offset,pre}}| - |V_{\text{offset,post}}| \ , \tag{7.47}$$

where $V_{\text{offset,pre}}$ is the offset error simulated for the schematics, and $V_{\text{offset,post}}$ is the offset error simulated for the netlist extracted from the generated layout. Thus, a negative value means a degradation of the offset error. It can be seen, that there is a degradation for all constraint sets, but for all circuits except Circuit 2, this degradation is lowest for constraint set NEW.

For CMRR, A_0 and f_T the respective relative errors $\epsilon^{(\text{rel})}_{\text{CMRR}}$, $\epsilon^{(\text{rel})}_{A_0}$ and $\epsilon^{(\text{rel})}_{f_T}$ are listed,

$$\epsilon^{(\text{rel})}_{\text{CMRR}} = \left(\frac{\text{CMRR}_{\text{post}}}{\text{CMRR}_{\text{pre}}} - 1\right) \cdot 100\% \qquad \epsilon^{(\text{rel})}_{A_0} = \left(\frac{A_{0,\text{post}}}{A_{0,\text{pre}}} - 1\right) \cdot 100\% \tag{7.48}$$

$$\epsilon^{(\text{rel})}_{f_T} = \left(\frac{f_{T,\text{post}}}{f_{T,\text{pre}}} - 1\right) \cdot 100\% \ . \tag{7.49}$$

Values CMRR_{pre}, $A_{0,\text{pre}}$ and $f_{T,\text{pre}}$ refer to the simulated values for the schematics. Values $\text{CMRR}_{\text{post}}$, $A_{0,\text{post}}$ and $f_{T,\text{post}}$ refer to the simulated values for the netlist extracted from the generated layout. Again, a negative value means a degradation of the respective performance and a positive value means an improvement. For the CMRR it can observed, that the NEW constraint set results in the best values for CMRR. In the case of Circuit L3 the degradation is large because the pre–layout value goes towards infinity. For the gain the error is small in all cases. The picture is similar for the transit frequency. The error is low for all experiments. Exceptions are Circuit L2 if no constraints are used and Circuit L5 if no constraints or the new constraints are used.

Parametric yield was computed for constraint sets COM and NEW and all circuits by performing Monte–Carlo Analysis with 5000 simulations runs on the extracted netlists. Parameter variations were modeled by ten global statistical parameters and two local statistical parameters per transistor. It turns out that the standard deviation of the single performances is only slightly influenced by the constraint set. In contrast, the mean value is influenced similar to the nominal values presented above. As a consequence, there are large yield differences for Circuits 3 to 5. In all of these cases, the best results are achieved for constraint set NEW.

8. Conclusion

The design of analog blocks is still done mostly manual and only few supporting software is available. Therefore analog blocks are still a significant cost factor for modern integrated circuits. This is because analog designers usually consider many different rules during the design process to ensure function in the presence of changing operating conditions, variations of the manufacturing process and parasitic effects. Theses rules are usually referred to as constraints. They must be available in machine readable form to enable the usage of automatic sizing and placement methods for analog circuits. However, this machine readable form is seldom available. Therefore, this thesis proposed a new method to generate the constraints automatically from the topology of analog circuits.

Beyond that, the thesis showed that similar structural analysis methods can be used to analyze digital circuits as well. The main application shown in this context is the generation of input data for automatic standard cell characterization. The method presented in this thesis is the first method that can handle circuits with analog and digital parts, e.g., mixed–signal circuits.

In particular, this thesis contains the following contributions compared to the state of the art:

- An enhanced method for building block analysis, which is the first method that can handle circuits with analog and digital parts. In addition, it contains several runtime improvements compared to the basic algorithm (Massier 2010).
- A new structural signal path analysis, introducing a new graph model of the circuit, as well as analyses to automatically identify the analog core part, analog bias part and digital part of a circuit, detect the true directions of *pass gates* and break feedback loops within the digital circuit part.
- A new symmetry computation for analog circuits, which can handle multiple, overlapping differential signal paths correctly.
- A new consistent constraint generation method, which generalizes the constraint generation for sizing and placement.

Experimental results to demonstrate the effectiveness of these methods were presented. The capabilities of the method for digital circuits were investigated for five digital standard cell libraries. For all libraries including industrial libraries the method was able to analyze more than 95% of the cells correctly. The capabilities of the method for analog circuits were investigated for four single–ended OTAs, two fully–differential OTAs and a fully–differential mixer.

It was shown, that a commercial tool cannot handle all of these circuits correctly. In contrast, the presented symmetry computation method can handle all of these circuits correctly. Furthermore, it was shown that symmetry related constraints have a big influence on speed and result quality of sizing and placement tools. In the experiments, nominal optimization and yield optimization of a circuit turned out to be up to 40 times and 3.9 times faster, respectively, if symmetry related constraints are used. At the same time yield was increased up to 14% and the worst–case distance improved up to 2.7σ. Sizing failed in one case if no symmetry related constraints were used.

For placement, different layouts were generated for three different constraint sets but the same sizing. It was shown that mismatch critical performances can be improved by a large amount if the constraints are generated using our new method. In one case, yield improved from 84% to 99%.

A. Graph Theoretical Definitions

Different authors from the field of graph theory use different notations and slightly differing definitions (Harary 1969). This appendix summarizes the definitions and notation used in this thesis. They are mostly based on (Harary 1969, Zwillinger 1996).

A.1. Graphs and Subgraphs

Definition A.1 (digraph)
A digraph or directed graph G is a triple $G = (N_G, E_G, \varphi_G)$, where N_G is the set of nodes or vertices and E_G is the set of edges. Incidence function $\varphi_G : E_G \to N_G^2$ describes the relation between nodes and edges. An edge e points from node μ to node ν iff $\varphi_G(e) = (\mu, \nu)$. (Harary 1969, Zwillinger 1996)

Definition A.2 (path, loop)
A path in a graph is an alternating sequence of length k of nodes n_i and edges e_i: $n_0, e_1, n_1, \ldots, e_k, n_k$. Edge e_i must be incident with node n_{i-1} and node n_i, i.e., $\varphi_G(e_i) = (n_{i-1}, n_i)$. In short, a path can be described as a sequence of edges: $e_1, e_2, \ldots e_n$. In case start node n_0 and end node n_k are equal, i.e., $n_0 = n_k$, the path is called loop. (Harary 1969, Zwillinger 1996)

The following special terms for graphs exist (Zwillinger 1996):

simple A graph is called simple if it has no parallel edges and no loops, i.e., φ_G is one–to–one.

multigraph A multigraph has parallel edges but no loops.

pseudograph A directed pseudograph has parallel edges and loops.

Definition A.3 (subgraph)
A graph $G' = (N_{G'}, E_{G'}, \varphi_{G'})$ is called subgraph of $G = (N_{G'}, E_{G'}, \varphi_{G'})$, if $N_{G'} \subseteq N_G$, $E_{G'} \subseteq E_G$ and $\varphi_{G'}(e) = \varphi_G(e)$ for all $e \in E_{G'}$.(Zwillinger 1996)

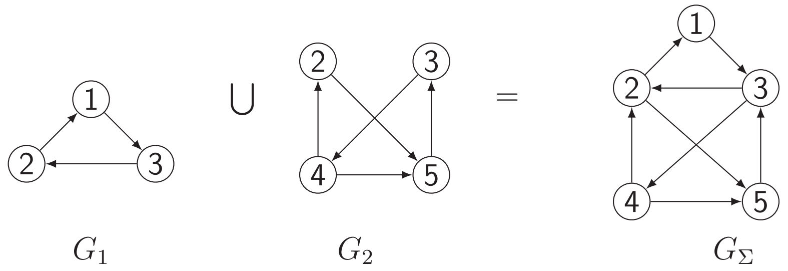


Figure A.1.: Union graph $G_\Sigma = G_1 \cup G_2$.

A.2. Properties of Graphs

Definition A.4 (reachability)
A node n_2 is reachable from node n_1 if there is a path from n_1 to n_2.(Harary 1969)

Definition A.5 (strongly connected)
A digraph G is called strongly connected if every node $n_1 \in N_G$ can be reached from every node $n_2 \in N_G$.(Harary 1969)

Usually this means the graph has a cycle.

Definition A.6 (network graph)
A digraph is called network graph if,

- it has one input node, where no edges end, and,
- it has one output node, where no edges start, and,
- all other nodes can be reached from the input node, and,
- the output node can be reached from all other nodes.

(Robinson & Foulds 1980)

A.3. Operations on Graphs

Definition A.7 (union graph)
The graph $G_\Sigma = (N_\Sigma, E_\Sigma, \varphi_\Sigma)$ created by the union $G_1 \cup G_2$ of graphs $G_1 = (N_1, E_1, \varphi_1)$ and $G_2 = (N_2, E_2, \varphi_2)$ consists of all nodes and edges of G_1 and G_2. It holds (Harary 1969),

$$G_\Sigma = G_1 \cup G_2 := (N_{G_1} \cup N_{G_2}, E_{G_1} \cup E_{G_2}, \varphi_{G_\Sigma}(e)) \quad \varphi_{G_\Sigma}(e) := \begin{cases} \varphi_{G_1}(e) & e \in E_1 \\ \varphi_{G_2}(e) & e \in E_2 \end{cases} . \tag{A.1}$$

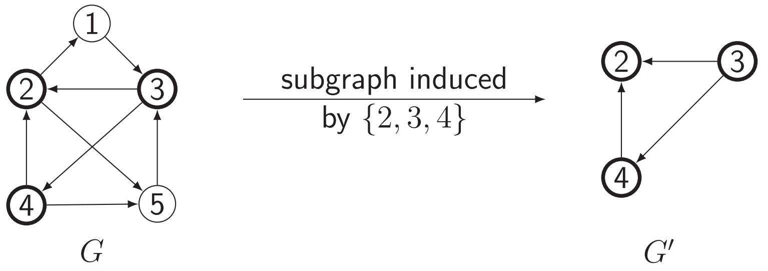


Figure A.2.: Subgraph G' induced by node set $\{2, 3, 4\}$ from Graph G.

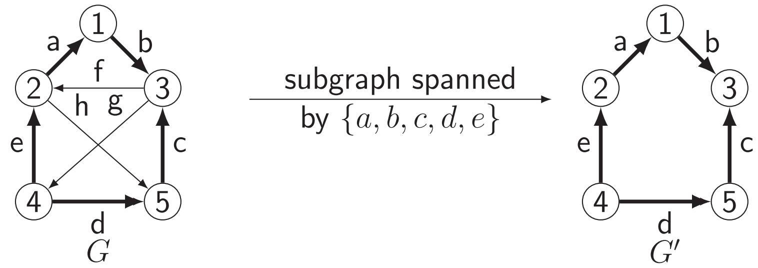


Figure A.3.: Subgraph G' spanned by edge set $\{a, b, c, d, e\}$ from Graph G.

Figure A.1 shows an example for this definition.

Definition A.8 (induced subgraph)
A subgraph $G' = (N_{G'}, E_{G'}, \varphi_{G'})$ of graph G is induced by node set N if $N_{G'} = N$ and $E_{G'} = \{e \in E_G | \varphi_G(e) \in N^2\}$.(Golumbic 1980)

Figure A.2 shows an example for Fig. A.2.

Definition A.9 (spanned subgraph)
A subgraph $G' = (N_{G'}, E_{G'}, \varphi_{G'})$ of graph G is spanned by edge set E if $E_{G'} = E$ and $N_{G'}$ is the minimum set such that $\varphi_G(e) \in N^2_{G'}$ for all $e \in E$.(Golumbic 1980)

Figure A.3 shows an example for Definition A.9.

A.4. Isomorphism

Definition A.10 (isomorphism)
Two graphs G and H are called isomorphic $G \cong H$ if there are one–to–one mappings $f_n : N_G \to N_H$ and $f_e : E_G \to E_H$, such that the incidence function is preserved, i.e.,

$$\underset{e \in E_G}{\forall} (\mu, \nu) \in \varphi_G(e) \;\leftrightarrow\; [\varphi_H(f_e(e)) = (f_n(\mu), f_n(\nu))] \;, \tag{A.2}$$

(Zwillinger 1996).

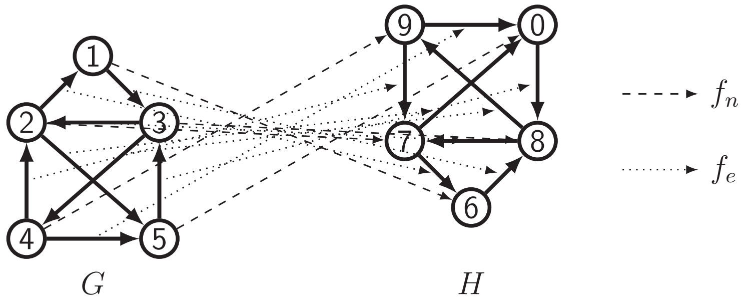


Figure A.4.: Two isomorphic graphs G and H.

Figure A.4 shows an example for Definition A.10.

B. Equivalence of Conflict Resolution Techniques for Building Block Analysis

This thesis and Massier (2010) use a different notation to describe the conflict resolution used during building block analysis. In the following it is shown that both notations are mathematically equivalent.

Formula (4.11) in Massier (2010) gives a propositional form to describe an analysis result that is conflict free. Formula (4.11) literally reads as follows:

$$\begin{aligned}\underset{m_\mu \in M}{\forall}\ \underset{m_\kappa, m_\lambda \in \mathrm{des}(m_\mu)_R}{\forall}\ \underset{i,j \in \{1,2\}}{\forall} \Big[& ((m_\kappa.\mathrm{structype}, i), (m_\lambda.\mathrm{structype}, j)) \in S \wedge \\ & \underbrace{\underset{x \in \mathrm{des}^\star(m_\mu)_R}{\exists} (x, m_\lambda) \in R_j \rightarrow}_{(\dagger)} \overline{\underset{y \in \mathrm{des}^\star(m_\mu)_R}{\exists} (y, m_\kappa) \in R_i} \Big]\end{aligned} \tag{B.1}$$

Set M is the set of all devices, i.e., $M = \mathcal{D}$. Relations R_1, R_2 describe the parent child relation between different components. In case $(x, y) \in R_1$ this means that x is the first child of y. Similarly, $(x, y) \in R_2$ means that x is the second child of y. Relation R is the union of R_1 and R_2. This maps to the notation used in this thesis as follows,

$$(x, y) \in R \Leftrightarrow x \in y.children \tag{B.2}$$

$$(x, y) \in R_1 \Leftrightarrow x = y.child_1 \tag{B.3}$$

$$(x, y) \in R_2 \Leftrightarrow x = y.child_2\ . \tag{B.4}$$

Set $\mathrm{des}(m_\mu)$ consists of all components that contain m_μ, either as child, grandchild, great–grandchild, etc. Using set $D_\star(x)$ from Eq. (2.1) this can be written as,

$$\mathrm{des}(m_\mu) = \{x \in \mathcal{B} | m_\mu \in D_\star(x)\}\ . \tag{B.5}$$

In addition $\mathrm{des}^\star(m_\mu)$ is defined as,

$$\mathrm{des}^\star(m_\mu) = \mathrm{des}(m_\mu) \cup \{m_\mu\} = \{x \in \mathcal{D} \cup \mathcal{B} | (m_\mu \in D_\star(x)) \vee (m_\mu = x)\}\ . \tag{B.6}$$

Relation S describes the dominance graph similar to graph G_D from Fig. 2.3. Graph G_D differs from S that the star symbol $\star$ is used as short notation for 1 or 2. In addition attribute *structype* is denoted as *type* attribute in this thesis. It holds,

$$\begin{aligned} &((m_\kappa.\text{structype}, i), (m_\lambda.\text{structype}, j)) \in S \\ &\qquad\Leftrightarrow \\ &\qquad((m_\kappa.\mathit{type}, i) \text{ reachable from } (m_\lambda.\mathit{type}, j)) \\ &\qquad\vee ((m_\kappa.\mathit{type}, \star) \text{ reachable from } (m_\lambda.\mathit{type}, j)) \\ &\qquad\vee ((m_\kappa.\mathit{type}, i) \text{ reachable from } (m_\lambda.\mathit{type}, \star)) \\ &\qquad\vee ((m_\kappa.\mathit{type}, \star) \text{ reachable from } (m_\lambda.\mathit{type}, \star)) \ . \end{aligned} \tag{B.7}$$

Next, Eq. (B.1) will be rewritten step–by–step. The term (†) reads as follows in the notation of this thesis,

$$\underset{x \in \{x \in \mathcal{D} \cup \mathcal{B} | (m_\mu \in D_\star(x)) \vee (m_\mu = x)\}}{\exists} x = m_\lambda.\mathit{child}_j \ . \tag{B.8}$$

This can be rewritten to,

$$\underset{x \in \mathcal{D} \cup \mathcal{B}}{\exists} \big((m_\mu \in D_\star(x)) \vee (m_\mu = x)\big) \wedge (x = m_\lambda.\mathit{child}_j), \tag{B.9}$$

because the term inside the quantifier is false for all elements of $\mathcal{D} \cup \mathcal{B}$ that are not part of the original set. This is equivalent to,

$$\big[m_\mu \in D_\star(m_\lambda.\mathit{child}_j) \cup \{m_\lambda.\mathit{child}_j\}\big] \Leftrightarrow m_\mu \in D_j(m_\lambda), \tag{B.10}$$

where $D_j(x)$ is defined according to Eq. (2.2).

In short, Eq. (B.1) can be denoted as,

$$\underset{m_\mu \in M}{\forall} \ \underset{m_\kappa, m_\lambda \in \mathrm{des}(m_\mu)_R}{\forall} f(m_\kappa, m_\lambda, m_\mu), \tag{B.11}$$

where $f(m_\kappa, m_\lambda, m_\mu)$ symbolizes the rest of the equation. Using Eq. (B.5) this can be rewritten to,

$$\underset{m_\mu \in \mathcal{D}}{\forall} \ \underset{m_\kappa, m_\lambda \in \{x \in \mathcal{D} \cup \mathcal{B} | m_\mu \in D_\star(x)\}}{\forall} f(m_\kappa, m_\lambda, m_\mu) \tag{B.12}$$

$$\Leftrightarrow \underset{m_\mu \in M}{\forall} \ \underset{m_\kappa, m_\lambda \in \mathcal{D} \cup \mathcal{B}}{\forall} \overline{m_\mu \in (D_\star(m_\kappa) \cap D_\star(m_\lambda))} \vee f(m_\kappa, m_\lambda, m_\mu) \tag{B.13}$$

$$\Leftrightarrow \underset{m_\kappa, m_\lambda \in \mathcal{D} \cup \mathcal{B}}{\forall} \ \underset{m_\mu \in D_\star(m_\kappa) \cap D_\star(m_\lambda)}{\forall} f(m_\kappa, m_\lambda, m_\mu) \tag{B.14}$$

$$\Leftrightarrow \underset{m_\kappa, m_\lambda \in \mathcal{D} \cup \mathcal{B}}{\forall} \ \overline{\underset{m_\mu \in D_\star(m_\kappa) \cap D_\star(m_\lambda)}{\exists} \overline{f}(m_\kappa, m_\lambda, m_\mu)} \tag{B.15}$$

Substituting $f(m_\kappa, m_\lambda, m_\mu)$ by Eq. (B.7) yields,

$$
\begin{aligned}
&\forall_{m_\kappa,m_\lambda\in\mathcal{D}\cup\mathcal{B}} \overline{\exists_{m_\mu\in D_\star(m_\kappa)\cap D_\star(m_\lambda)} \overline{\forall_{i,j\in\{1,2\}}}}\\
&\qquad\overline{\overline{\Big(((m_\kappa.\mathit{type}, i) \text{ reachable from } (m_\lambda.\mathit{type}, j))}}\\
&\qquad\overline{\overline{\vee\,((m_\kappa.\mathit{type}, \star) \text{ reachable from } (m_\lambda.\mathit{type}, j))}}\\
&\qquad\overline{\overline{\vee\,((m_\kappa.\mathit{type}, i) \text{ reachable from } (m_\lambda.\mathit{type}, \star))}}\\
&\qquad\overline{\overline{\vee\,((m_\kappa.\mathit{type}, \star) \text{ reachable from } (m_\lambda.\mathit{type}, \star))\Big)}}\\
&\qquad\qquad\qquad\overline{\overline{\wedge m_\mu \in D_j(m_\lambda) \rightarrow \overline{m_\mu \in D_i(m_\kappa)}}}\;. \qquad \text{(B.16)}
\end{aligned}
$$

Applying de Morgan's law results in,

$$
\begin{aligned}
&\forall_{m_\kappa,m_\lambda\in\mathcal{D}\cup\mathcal{B}} \overline{\exists_{m_\mu\in D_\star(m_\kappa)\cap D_\star(m_\lambda)} \exists_{i,j\in\{1,2\}}}\\
&\qquad\overline{\Big(((m_\kappa.\mathit{type}, i) \text{ reachable from } (m_\lambda.\mathit{type}, j))}\\
&\qquad\overline{\vee\,((m_\kappa.\mathit{type}, \star) \text{ reachable from } (m_\lambda.\mathit{type}, j))}\\
&\qquad\overline{\vee\,((m_\kappa.\mathit{type}, i) \text{ reachable from } (m_\lambda.\mathit{type}, \star))}\\
&\qquad\overline{\vee\,((m_\kappa.\mathit{type}, \star) \text{ reachable from } (m_\lambda.\mathit{type}, \star))\Big)}\\
&\qquad\qquad\qquad\overline{\wedge m_\mu \in D_j(m_\lambda) \wedge m_\mu \in D_i(m_\kappa)}\;. \qquad \text{(B.17)}
\end{aligned}
$$

This is equivalent to,

$$
\begin{aligned}
&\forall_{m_\kappa,m_\lambda\in\mathcal{D}\cup\mathcal{B}} \overline{\exists_{m_\mu\in D_\star(m_\kappa)\cap D_\star(m_\lambda)}}\\
&\overline{\Big(\exists_{i,j\in\{1,2\}} ((m_\kappa.\mathit{type}, i) \text{ reachable from } (m_\lambda.\mathit{type}, j)) \wedge m_\mu \in D_j(m_\lambda) \cap D_i(m_\kappa)\Big)}\\
&\overline{\vee\Big(\exists_{i,j\in\{1,2\}} ((m_\kappa.\mathit{type}, \star) \text{ reachable from } (m_\lambda.\mathit{type}, j)) \wedge m_\mu \in D_j(m_\lambda) \cap D_i(m_\kappa)\Big)}\\
&\overline{\vee\Big(\exists_{i,j\in\{1,2\}} ((m_\kappa.\mathit{type}, i) \text{ reachable from } (m_\lambda.\mathit{type}, \star)) \wedge m_\mu \in D_j(m_\lambda) \cap D_i(m_\kappa)\Big)}\\
&\overline{\vee\Big(\exists_{i,j\in\{1,2\}} ((m_\kappa.\mathit{type}, \star) \text{ reachable from } (m_\lambda.\mathit{type}, \star)) \wedge m_\mu \in D_j(m_\lambda) \cap D_i(m_\kappa)\Big)}\;.\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(B.18)}
\end{aligned}
$$

This can be further rewritten using

$$\begin{aligned}
&\underset{i\in\{1,2\}}{\exists} m_\mu \in D_j(m_\lambda) \cap D_i(m_\kappa)\\
\Leftrightarrow & m_\mu \in \big(D_j(m_\lambda) \cap D_1(m_\kappa)\big) \vee m_\mu \in \big(D_j(m_\lambda) \cap D_2(m_\kappa)\big)\\
\Leftrightarrow & m_\mu \in D_j(m_\lambda) \cap \big(D_1(m_\kappa) \cup D_2(m_\kappa)\big)\\
\Leftrightarrow & m_\mu \in D_j(m_\lambda) \cap D_\star(m_\kappa)
\end{aligned} \tag{B.19}$$

and

$$\underset{j\in\{1,2\}}{\exists} m_\mu \in D_j(m_\lambda) \cap D_i(m_\kappa) \Leftrightarrow m_\mu \in D_\star(m_\lambda) \cap D_i(m_\kappa) \;, \tag{B.20}$$

yielding

$$\begin{aligned}
&\underset{m_\kappa, m_\lambda \in \mathcal{D}\cup\mathcal{B}}{\forall} \overline{\underset{m_\mu \in D_\star(m_\kappa)\cap D_\star(m_\lambda)}{\exists}}\\
&\overline{\Big(\underset{i,j\in\{1,2\}}{\exists} ((m_\kappa.\mathit{type}, i) \text{ reachable from } (m_\lambda.\mathit{type}, j)) \wedge m_\mu \in D_j(m_\lambda) \cap D_i(m_\kappa)\Big)}\\
&\overline{\vee\Big(\underset{j\in\{1,2\}}{\exists} ((m_\kappa.\mathit{type}, \star) \text{ reachable from } (m_\lambda.\mathit{type}, j)) \wedge m_\mu \in D_j(m_\lambda) \cap D_\star(m_\kappa)\Big)}\\
&\overline{\vee\Big(\underset{i\in\{1,2\}}{\exists} ((m_\kappa.\mathit{type}, i) \text{ reachable from } (m_\lambda.\mathit{type}, \star)) \wedge m_\mu \in D_\star(m_\lambda) \cap D_i(m_\kappa)\Big)}\\
&\overline{\vee\Big(((m_\kappa.\mathit{type}, \star) \text{ reachable from } (m_\lambda.\mathit{type}, \star)) \wedge m_\mu \in D_\star(m_\lambda) \cap D_\star(m_\kappa)\Big)}
\end{aligned} \tag{B.21}$$

This can be simplified to,

$$\begin{aligned}
\underset{m_\kappa, m_\lambda \in \mathcal{D}\cup\mathcal{B}}{\forall} & \overline{\Big(\underset{i,j\in\{1,2,\star\}}{\exists} ((m_\kappa.\mathit{type}, i) \text{ reachable from } (m_\lambda.\mathit{type}, j))}\\
& \qquad \overline{\wedge \underset{m_\mu \in D_\star(m_\kappa)\cap D_\star(m_\lambda)}{\exists} m_\mu \in D_j(m_\lambda) \cap D_i(m_\kappa)\Big)} \;.
\end{aligned} \tag{B.22}$$

The last condition says that the cut set must not be empty. This allows to simplify Eq. (B.22) to,

$$\begin{aligned}
\underset{m_\kappa, m_\lambda \in \mathcal{D}\cup\mathcal{B}}{\forall} & \overline{\Big(\underset{i,j\in\{1,2,\star\}}{\exists} ((m_\kappa.\mathit{type}, i) \text{ reachable from } (m_\lambda.\mathit{type}, j))}\\
& \qquad \overline{\wedge D_j(m_\lambda) \cap D_i(m_\kappa) \neq \emptyset\Big)} \;.
\end{aligned} \tag{B.23}$$

This matches Eq. (2.6) for $x_1 = m_\kappa$ and $x_2 = m_\lambda$.

List of Figures

List of Tables

Glossary

Symbols

α attribute function of a graph. 54, 55, 74

α^C component attribute function of a graph. 55, 61, 72, 93, 94

α^S structural attribute function of a graph. 55, 61, 63, 64, 85, 86

φ incidence function of a graph. 54, 55, 63, 65, 69, 74, 133–135

φ^- incidence function of a graph (start node only). 54, 64, 72, 75, 85, 87

φ^+ incidence function of a graph (end node only). 54, 63, 64, 72, 75, 85, 87

τ time function of a temporal ESFG. 74, 75

4tcm four–transistor current mirror. 9, 25, 116

A

attribute .

- ***child***$_i$ i-th child of a building block. 8, 10, 12, 25, 27, 29, 37, 38, 44, 94, 137, 138
- ***children*** set of all children of a building block. 12, 29, 94, 137
- ***net*** connectivity function. 29, 31, 32, 36, 56, 57
- ***parents*** set of all parents of a component. 29, 30, 43
- ***pins*** set of pins of a component. 29, 55
- ***subtype*** subtype of a component, e.g., nmos or pmos. 29, 31, 32
- ***type*** type of a component, e.g., trans for a transistor. 13, 28–32, 36, 37, 41, 94, 138–140, 148

B

$\mathbb{B}$ set of Boolean numbers, i.e., $\mathbb{B} = \{0, 1\}$. 31

$\mathcal{B}$ the set of all building blocks. 9, 29

$\mathcal{B}_{\mathbf{top}}$ set of all building blocks without parents. 57

building block basic functional block of a circuit; formally, a number of devices with a fixed topology (=connection). 7–9, 13, 14, 17–19, 21–23, 26–32, 40, 43, 47, 49, 53, 56, 57, 59, 60, 93–95, 101–104, 106, 113, 115, 116, 119, 126, 142, 147, 148, 150

- **analog** a building block which only occurs in analog circuit parts. 7, 51, 64, 65, 67
- **analysis** a method to recognize building blocks based on the netlist of a circuit. 5, 7, 8, 21, 30, 43, 48, 67, 101, 102, 116, 131, 137
- **digital** a building block which only occurs in digital circuit parts. 19, 43, 51, 64
- **library** . 8, 21, 30

type See attribute *type*. 30, 94

C

cc cross–coupled pair. 8, 25, 50, 58, 60, 61, 116
ccm cascode current mirror. 8, 9, 25, 27, 49, 50, 94, 116
cml current mirror load. 8, 9, 25
component a device or building block. 28, 29, 33, 40, 47, 147
constraint programming general term for methods to solve CSP. 63, 96
cp cascode pair. 8, 9, 14, 26, 47
CSP constraint satisfaction problem: consists of linear and specific non–linear equalities and inequalities for integer variables and predicate logic expressions for Boolean variables. 63, 67, 95, 148
cta capacitor transistor array. 23, 30, 36, 57

D

D set of all devices belonging to a building block. 9, 12–14, 137–140
$\mathcal{D}$ the set of all devices of a circuit. 9, 29
device the most basic elements of a circuit such as transistors, resistors, etc.. 21, 22, 28, 29, 43, 94, 147, 148, 150
dn drain pin of a MOSFET. 29
dp differential pair. 8, 9, 12–14, 25, 26, 43, 47, 49–51, 58, 60, 61, 67, 116, 124
ds differential stage. 9, 14, 26, 43, 49, 57
dta diode transistor array. 22, 23, 25, 30, 31, 36, 43, 57
dtrans degenerated transistor. 23

E

E set of edges of a graph. 54, 55, 59, 60, 63, 65, 67–72, 74, 75, 85, 87, 95, 133, 135
ESFG enhanced structural signal flow graph: a graph representing structure and qualitative behavior of the circuit. 53, 54, 56–63, 65–70, 72–76, 78, 79, 84–87, 89–93, 101, 102, 106, 110, 142, 145, 147, 149, 150

F

fc folded cascode pair. 25, 27, 49, 50, 141

G

generic constraint describes circuit constraint in terms of generic conditions like symmetry, matching, etc.. 113
gs Gilbert stack. 26, 43, 57
gt gate pin of a MOSFET. 29

I

iwcm improved wilson current mirror. 9, 26, 116

K

$\mathcal{K}$ the set of all terminals of a circuit. 54, 55, 57, 59, 63, 67, 68, 70, 71, 77, 79, 87
$\mathcal{K}_i$ the set of all input terminals of a circuit. 54, 55, 83, 90
$\mathcal{K}_o$ the set of all output terminals of a circuit. 54, 55, 83, 90

L

la logic array. 22, 25–27, 36, 103, 104, 106
lg logic gate. 23, 25–28, 30, 51, 58, 60, 71, 103, 104, 106, 111
ls level shifter. 8, 9, 25–27, 43, 49, 58, 116

N

$\mathcal{N}$ set of all nets. 29
N set of nodes of a graph. 54, 59, 63–65, 67–70, 74–78, 85, 87, 133–135, 142
NCFG network component of an ESFG: a special ESFG that represents the influence of one input to one output. 84–88, 91, 95, 142, *see* ESFG
net connects one or more components. 28, 29, 47, 54, 149
nta normal transistor array. 22, 23, 25–27, 30, 31, 35, 36, 39, 43, 57, 58, 64, 71–73

P

pg pass gate. 23, 25, 26, 28, 51, 58, 60, 64, 71–73, 103, 104, 106, 111, 131
placement constraint constraint for automatic placement. 113
$\mathcal{P}^B_{\mathbf{sym}}$ behavioral symmetry problem. 80–82, 84
$\mathcal{P}^E_{\mathbf{sym}}$ symmetry problem for ESFGs. 87, 90

S

$\mathcal{S}$ set of symmetric device pairs. 93–95
sc stack chain. 25–27, 39, 103, 104, 106
sc source pin of a MOSFET. 29
scm simple current mirror. 8, 9, 12–14, 21, 22, 25, 26, 28, 31, 32, 43, 49–52, 55, 60, 61, 65, 86, 116, 121, 123, 124
signal path a signal path describes the transfer behavior between a single input terminal or a symmetrical input terminal pair and a single output terminal or a symmetrical output terminal pair. 79–82, 142
sizing constraint constraint for automatic sizing. 113
st stack. 23, 25–28, 38, 39, 47, 51
structural signal path analysis an analysis method based on structural and qualitative behavioral considerations. 5, 27, 53, 101, 131, 141

T

$T_{\mathcal{B}}$ Set of all building block types. 29
tbb tristate base block. 26
tcb tristate control block. 23, 26, 58, 103, 104
$T_{\mathcal{D}}$ Set of all device types. 28
temporal ESFG a ESFG with additional time concept. 74, 101, 102, 106, 147
terminal external connection of a circuit. 54, 149

U

uta dummy transistor array. 23, 30, 36, 58

V

vrI voltage reference I. 8, 9, 25
vrII voltage reference II. 8, 9, 25, 26

W

wcm wilson current mirror. 9, 26, 116
wscm wide–swing cascode current mirror. 9, 25, 27, 94, 116
wsm wide–swing current mirror. 25, 26, 116

Bibliography

Accellera (2000): Advanced Library Format (ALF) Reference Manual, version 2.0, Los Gatos, CA, USA.

Aloul, Fadi, Ramani, Arathi, Markov, Igor L. & Sakallah, Karem (2002): Solving difficult SAT instances in the presence of symmetry, ACM/IEEE Design Automation Conference (DAC), DAC '02, ACM, New York, NY, USA, pp. 731–736.

Antreich, Kurt, Eckmueller, J., Graeb, Helmut, Pronath, Michael, Schenkel, Frank, Schwencker, R. & Zizala, S. (2000): WiCkeD: Analog Circuit Synthesis Incorporating Mismatch, IEEE Custom Integrated Circuits Conference (CICC), pp. 511–514.

Ari, Mordechai Ben, Pnueli, Amir & Manna, Zohar (1983): The temporal logic of branching time, Acta Informatica **20**: 207–226: 10.1007/BF01257083.

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

Arsintescu, Bogdan G., Charbon, Edoardo, Malavasi, Enrico & Kao, William (1998): AC constraint transformation for top-down analog design, Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on, Bd. 6, pp. 126–130vol.6.

Arsintescu, Bogdan G. & Otten, Ralph H. J. M. (1998): Constraints space management for the layout of analog IC's, Design, Automation and Test in Europe, 1998., Proceedings, pp. 971–972.

Arsintescu, Bogdan G. & Spanoche, S.A. (1996): Global stacking for analog circuits, Design Automation Conference, 1996, with EURO-VHDL '96 and Exhibition, Proceedings EURO-DAC '96, European, pp. 392–397.

Baba-ali, A.R. & Farah, A. (1996): An efficient algorithm for signal flow determination in digital CMOS VLSI, European Design and Test Conference, 1996. ED TC 96. Proceedings, pp. 288–293.

Barke, Erich (1984): A Network Comparison Algorithm for Layout Verification of Integrated Circuits, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **3**(2): 135–141.

Bhattacharya, Sambuddha, Jangkrajarng, Nuttorn, Hartono, Roy & Shi, C-J. Richard (2004): Hierarchical Extraction and Verification of Symmetry Constraints for Analog Layout Automation, Asia and South Pacific Design Automation Conference, pp. 400–405.

Blaauw, David T., Saab, Daniel G., Long, J. & Abraham, Jacob A. (1990): Derivation of signal flow for switch-level simulation, ACM/IEEE Design Automation Conference (DAC), pp. 301–305.

Boehner, Michael (1988): LOGEX — an automatic logic extractor from transistor to gate level for CMOS technology, 25th ACM/IEEE Design Automation Conference DAC-88, pp. 517–522.

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

Bryant, Randal E. (1991): Extraction of gate level models from transistor circuits by four-valued symbolic analysis, Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference on, pp. 350–353.

Cadence Design Systems (2008): Virtuoso Custom Design Platform XL Datasheet: Viewed 27 June 2012, `http://www.cadence.com/products/cic/schematic_editor/pages/resources.aspx`.

Cadence Design Systems (2010): Functional verification of a differential operational amplifier: http://w2.cadence.com/whitepapers/FVofDiffOpAmp.pdf.

Chai, Donald (2009): Circuit Symmetries in Synthesis and Verification, PhD thesis, EECS Department, University of California, Berkeley.

Charbon, E., Malavasi, E. & Sangiovanni-Vincentelli, A. (1993): Generalized constraint generation for analog circuit design, IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 408–414.

Chen, D. J. & Sheu, B. J. (1992): Generalised approach to automatic custom layout of analogue ICs, Circuits, Devices and Systems, IEE Proceedings G **139**(4): 481–490.

Choudhury, U. & Sangiovanni-Vincentelli, A. (1993): Automatic generation of parasitic constraints for performance-constrained physical design of analog circuits, IEEE/ACM International Conference on Computer-Aided Design and Manufacture of Electronic Components,, pp. 208–224.

Conte, D., Foggia, P., Sansone, C. & Vento, M. (2004): Thirty Years of Graph Matching in Pattern Recognition, International Journal of Pattern Recognition and Artificial Intelligence **18**(03): 265–298.

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

de Kleer, Johan (1984): How circuits work, Artificial Intelligence **24**: 205–280.

Ebeling, Carl (1988): GeminiII: A Second Generation Layout Validation Program, IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 322–325.

Eckmüller, Josef (1998): Zur rechnergestützten Dimensionierung analoger integrierter Schaltungen unter besonderer Berücksichtigung von Struktureigenschaften, PhD thesis, tum.

Eick, Michael (2006): Automatische Parameterextraktion und Strukturanalyse analoger Schaltungen im Hinblick auf diskrete Optimierverfahren, Bachelor's thesis, Technische Universiät München.

Eick, Michael (2008): Layoutsynthese analoger integrierter Schaltungen mit automatisch generierten Platzierungsregeln, Master's thesis, Technische Universität München, München.

Eick, Michael & Graeb, Helmut (2011a): Sizing of Analog Integrated Circuits using Enhanced Symmetry Analysis, Technical Report TUM-LEA-11-1, Technische Universität München.

Eick, Michael & Graeb, Helmut (2011b): Unified Generation of Analog Sizing and Placement Constraints: Frontiers in Analog Circuit (FAC) Synthesis and Verification,http://www.async.ece.utah.edu/FAC2011/.

Eick, Michael & Graeb, Helmut (2012a): A Versatile Structural Analysis Method for Analog, Digital and Mixed-Signal Circuits, Int. Conf. Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD).

Eick, Michael & Graeb, Helmut (2012b): Automatische Dimensionierung von Analogschaltungen unter Berücksichtigung von Schaltungssysmmetrien, GMM/GI/ITG-Fachtagung Zuverlässigkeit und Entwurf.

Eick, Michael & Graeb, Helmut (2012c): MARS: Matching-driven Analog Sizing, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **31**(8): 1145–1158.

Eick, Michael & Graeb, Helmut (2013): Towards Automatic Structural Analysis of Mixed-Signal Circuits, in M. Fakhfakh, E. Tlelo-Cuautle & R. Castro-Lopez (eds), Analog/RF and Mixed-Signal Circuit Systematic Design, Springer, Kapitel 1: To be published.

Eick, Michael, Lu, Kun & Graeb, Helmut (2010): Sizing of Analog Integrated Circuits using Symmetry Recognition, Technischer Bericht TUM-LEA-10-1, Lehrstuhl für Entwurfsautomatisierung.

Eick, Michael, Sridharan, Devanathan & Graeb, Helmut (2013): Symmetry Computation for Hierarchical Analog Designs: Frontiers in Analog CAD (FAC), To be published.

Eick, Michael, Strasser, Martin, Graeb, Helmut & Schlichtmann, Ulf (2009): Automatische Generierung hierarchischer Platzierungsregeln für analoge integrierte Schaltungen, GMM/GI/ITG-Fachtagung Zuverlässigkeit und Entwurf, VDE Verlag GMBH, pp. 107–114.

Eick, Michael, Strasser, Martin, Graeb, Helmut & Schlichtmann, Ulf (2010): Automatic Generation of Hierarchical Placement Rules for Analog Integrated Circuits, ACM/SIGDA International Symposium on Physical Design (ISPD), pp. 47–54.

Eick, Michael, Strasser, Martin & Lu, Kun (2013): Automatische Generierung von Platzierungsregeln für analoge integrierte Schaltungen, Mechatronik (1-2): To be published.

Eick, Michael, Strasser, Martin, Lu, Kun, Schlichtmann, Ulf & Graeb, Helmut (2011): 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.

El-Turky, F. & Perry, E. (1989): BLADES: An artificial intelligence approach to analog circuit design, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **8**: 680–692.

Ferent, Cristian & Doboli, Alex (2010): Measuring the uniqueness and variety of analog circuit design features, INTEGRATION - the VLSI journal **44**(1): 39–50.

Fisher, J. & Koch, R. (1987): A Highly Linear CMOS Buffer Amplifier, IEEE Journal of Solid-State Circuits SC **22**: 330–334.

Frühwirth, Thom & Abdennadher, Slim (2003): Essentials of Constraint Programming, Springer-Verlag, Berlin, Heidelberg.

Galdi, Ivano, Bonizzoni, Edoardo, MALCOVATI, Piero, Manganaro, Gabriele & Maloberti, Franco (2008): 40 MHz IF 1 MHz Bandwidth Two-Path Bandpass ΣΔ Modulator With 72 dB DR Consuming 16 mW, IEEE Journal of Solid-State Circuits SC **43**(7): 1648–1656.

Gielen, G. & Sansen, W. (1991): Symbolic Analysis for Automated Design of Analog Integrated Circuits, Kluwer Academic Publishers, Dordrecht.

Golumbic, M. C. (1980): Algorithmic graph theory and perfect graphs, Academic Press, New York.

Graeb, Helmut (2007): Analog Design Centering and Sizing, Springer.

Graeb, Helmut (2011): Analog Layout Synthesis - A Survey of Topological Approaches, Springer.

Graeb, Helmut, Zizala, S., Eckmueller, J. & Antreich, Kurt (2001): The Sizing Rules Method for Analog Integrated Circuit Design, IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 343–349.

Guo, Yan (2012): Automatic Structural Analysis of Multiple Latch Architectures, Bachelor's thesis, Technische Universität München, Munich.

Hao, Qinsheng, Dong, Sheqin, Chen, Song, Hong, Xianlong, Su, Yi & Qu, Zhiyi (2004): Constraints generation for analog circuits layout, 2004 International Conference on Communications, Circuits and Systems **2**: 1339–1343 Vol.2.

Harary, Frank (1969): Graph Theory, Addison-Wesley series in mathematics.

Harjani, R., Rutenbar, Rob A. & Carley, L. (1989): OASYS: A Framework for Analog Circuit Synthesis, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **8**: 1247–1266.

Hassoun, Soha & McCreary, Carolyn (1999): Regularity extraction via clan-based structural circuit decomposition, IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 414–418.

Hastings, Alan (2001): The Art of Analog Layout, Prentice-Hall.

Huang, Kai-Ti & Overhauser, David A. (1995): A Novel Graph Algorithm for Circuit Recognition, IEEE International Symposium on Circuits and Systems (ISCAS), Bd. 3, pp. 1695–1698.

Hübner, U., Vierhaus, H. T. & Camposano, Raul (1997): Partitioning and Analysis of Static Digital CMOS Circuits, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **16**(11): 1292 – 1310.

ITRS (ed.) (2005): International Technology Roadmap for Semiconductors: 2005 Edition, ITRS: [online] available at www.itrs.net (May 2012).

ITRS (ed.) (2011): International Technology Roadmap for Semiconductors: 2011 Edition, ITRS: [online] available at www.itrs.net (May 2012).

Jain, Samir, Bryant, Randal E. & Jain, Alok (1995): Automatic Clock Abstraction from Sequential Circuits, 25th ACM/IEEE Design Automation Conference DAC-88, pp. 707–711.

Jerke, Göran, Lienig, Jens & Freuer, Jan B. (2011): Constraint-driven design methodology: A path to analog design automation, in H. E. Graeb (ed.), Analog Layout Synthesis, Springer, New York, Kapitel 7, pp. 269–297.

Johns, David A. & Martin, Ken (1997): Analog Integrated Circuit Design, John Wiley & Sons.

Jongudomkarn, Jonggrist (2012): Sizing Rules for Mixed-Signal Circuits, Bachelor's thesis, Technische Universität München.

Jouppi, Norman P. (1983): Timing Analysis for nMOS VLSI, ACM/IEEE Design Automation Conference (DAC), pp. 411–418.

Kim, Wonjong & Shin, Hyunchul (1998): Hierarchical LVS based on hierarchy rebuilding, ACM/IEEE Design Automation Conference (DAC), pp. 379–384.

Knoth, Christoph, Kleeberger, Veit Benedikt, Nordholz, Petra & Schlichtmann, Ulf (2009): Fast and Waveform Independent Characterization of Current Source Models, IEEE/VIUF International Workshop on Behavioral Modeling and Simulation (BMAS), pp. 90–95.

Kole, M. E., Smit, J. & Herrmann, O. E. (1994): Modeling symmetry in analog electronic circuits, IEEE International Symposium on Circuits and Systems (ISCAS), pp. 315–318.

Krinke, Andreas & Lienig, Jens (2011): An Ontology for Constraints in Custom IC Design, European Conference on Circuit Theory and Design (ECCTD), pp. 343–345.

Laker, Kenneth R. & Sansen, Willy (1994): Design of Analog Integrated Circuits and Systems, McGraw-Hill, New York.

Laurentin, M., Greiner, A. & Marbot, R. (1992): DESB, a functional abstractor for CMOS VLSI circuits, Design Automation Conference, 1992. EURO-VHDL '92, EURO-DAC '92. European, pp. 22–27.

Lee, K. J., Gupta, R. & Breuer, M.A. (1990): A new method for assigning signal flow directions to MOS transistors, Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on, pp. 492–495.

Lee, Thomas H. (2004): The Design of CMOS Radio-Frequency Integrated Circuits, second, Cambridge University Press, Cambridge.

Lorenz, Dominik (2012): Aging Analysis of Digital Integrated Circuits, PhD thesis, Technische Universität München.

Lorenz, Dominik, Barke, Martin & Schlichtmann, Ulf (2010): Aging analysis at gate and macro cell level, IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 77–84.

Lu, Kun (2009): Symmetry Recognition for Automatic Sizing of Analog Integrated Circuits, Master's thesis, Technische Universität München.

Mahmoud, Imbaby I. (1998): Performance constraints generation for analog circuit layout, Radio Science Conference, 1998. NRSC '98. Proceedings of the Fifteenth National, pp. C36/1–C36/7.

Makris, C. A. & Toumazou, Christofer (1995): Analog IC design automation. II. Automated circuit correction by qualitative reasoning, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **14**(2): 239–254.

Malavasi, Enrico, Charbon, Edoardo, Arsintescu, Bogdan G. & Kao, William (1998): A constraint management system for IC physical design, Integrated Circuit Design, 1998. Proceedings. XI Brazilian Symposium on, pp. 240–243.

Malavasi, Enrico, Charbon, Edoardo, Felt, Eric & Sangiovanni-Vincentelli, Alberto L. (1996): Automation of IC Layout with Analog Constraints, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **15**(8): 923–942.

Malavasi, Enrico & Kao, William H. (1997): Current issues in a constraint-driven mixed signal physical design flow, Circuits and Systems, 1997. ISCAS '97., Proceedings of 1997 IEEE International Symposium on, Bd. 1, pp. 133–136vol.1.

Martens, Ewout & Gielen, Georges (2008): Classification of analog synthesis tools based on their architecture selection mechanisms, INTEGRATION - the VLSI journal **41**(2): 238–252.

Mason, Samuel J. (1953): Feedback Theory – Some Properties of Signal Flow Graphs, Proceedings of the IRE **41**: 1144–1156.

Mason, Samuel J. (1956): Feedback Theory – Further Properties of Signal Flow Graphs, Proceedings of the IRE **44**: 920–926.

Massier, Tobias (2002): Automatische Extraktion von Dimensionierungsregeln analoger Schaltungen, Master's thesis, Technische Universiät München.

Massier, Tobias (2010): On the Structural Analysis of CMOS and Bipolar Analog Integrated Circuits, PhD thesis, Technische Universität München.

Massier, Tobias & Graeb, Helmut (2008): Dimensionierungsregeln für analoge Bipolarschaltungen, ITG/GMM-Fachtagung Entwurf von analogen Schaltungen mit CAE-Methoden (ANALOG), VDE, VDE Verlag GmbH, pp. 107–112.

Massier, Tobias, Graeb, Helmut & Schlichtmann, Ulf (2008a): Sizing Rules for Bipolar Analog Circuit Design, Design, Automation and Test in Europe (DATE), pp. 140–145.

Massier, Tobias, Graeb, Helmut & Schlichtmann, Ulf (2008b): 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.

Massier, Tobias, Stehr, Guido & Graeb, Helmut (2003): Ein Beitrag zur Automatisierung der Strukturanalyse und der impliziten Spezifikation von analogen integrierten Schaltungen, 7. GMM/ITG Diskussionssitzung Entwurf von Analogschaltungen (ANALOG '03) pp. 31–36.

Mealy, George H. (1955): A method for synthesizing sequential circuits, The Bell Systems Technical Journal .

Meissner, Markus, Mitea, Oliver, Luy, Linda & Hedrich, Lars (2012): Fast isomorphism testing for a graph-based analog circuit synthesis framework, Design, Automation Test in Europe Conference Exhibition (DATE), 2012, pp. 757–762.

Messmer, Bruno T. & Bunke, Horst (1998): A new algorithm for error-tolerant subgraph isomorphism detection, Pattern Analysis and Machine Intelligence, IEEE Transactions on **20**(5): 493–504.

Ohri, K.B. & Callahan, M.J. (1979): Integrated PCM codec, Solid-State Circuits, IEEE Journal of **14**(1): 38–46.

Ousterhout, John K. (1985): A Switch-Level Timing Verifier for Digital MOS VLSI, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **4**(3): 336–349.

Pandey, M., Jain, A., Bryant, Randal E., Beatty, D., York, G. & Jain, S. (1995): Extraction of finite state machines from transistor netlists by symbolic simulation, Computer Design: VLSI in Computers and Processors, 1995. ICCD '95. Proceedings., 1995 IEEE International Conference on, pp. 596–601.

Pelgrom, M., Duinmaijer, A. & Welbers, A. (1989): Matching properties of MOS transistors, IEEE Journal of Solid-State Circuits SC **24**: 1433–1440.

Rao, D. S. & Kurdahi, F. J. (1993): On clustering for maximal regularity extraction, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **12**(8): 1198–1208.

Retdian, N., Takagi, S. & Fujii, N. (2002): Voltage controlled ring oscillator with wide tuning range and fast voltage swing, ASIC, 2002. Proceedings. 2002 IEEE Asia-Pacific Conference on, pp. 201 – 204.

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

Robinson, David F. & Foulds, L. R. (1980): Digraphs: Theory and Techniques, Gordon and Breach Science Publishers, New York, London, Paris.

Roh, Jeongjin, Byun, Sanho, Choi, Youngkil, Roh, Hyungdong, Kim, Yi Gyeong & Kwon, Jong Kee (2008): A 0.9-V 60- uW 1-Bit Fourth-Order Delta-Sigma Modulator With 83-dB Dynamic Range, Solid-State Circuits, IEEE Journal of **43**(2): 361–370.

Rubanov, Nikolay (2003): SubIslands: the probabilistic match assignment algorithm for subcircuit recognition, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **22**(1): 26–38.

Rubanov, Nikolay (2006): 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.

Rutenbar, Rob A. (2012): Analog Circuit Synthesis (and Verification) Revisited: What's missing, Talk at SMACD: `http://www2.imse-cnm.csic.es/~smacd2012/SMACD_2012/HOME.html`.

Sansen, Willy M. C. (2007): Analog Design Essentials, Springer.

Schulte, Christian, Tack, Guido & Lagerkvist, Mikael Z. (2010): Modeling and Programming with Gecode, www.gecode.org/doc/3.4.0/MPG.pdf.

Sedgewick, Robert (1988): Algorithms, Addison-Wesley Publishing Company.

Shannon, Claude E. (1949): Communication in the presence of noise, Proceedings of the IRE **37**(1): 10–21.

Sridharan, Devanathan (2013): Automatic Structural Analysis for Hierarchical Designs, Master's thesis, Technische Universität München: To be submitted.

Sripramong, Thanwa & Toumazou, Christofer (2002): The Invention of CMOS Amplifiers Using Genetic Programming and Current-Flow Analysis, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems .

Stine, J.E., Castellanos, I., Wood, M., Henson, J., Love, F., Davis, W.R., Franzon, P.D., Bucher, M., Basavarajaiah, S., Oh, J. & Jenkal, R. (2007): FreePDK: An Open-Source Variation-Aware Design Kit, Microelectronic Systems Education, 2007. MSE '07. IEEE International Conference on, pp. 173–174.

Stolberg-Stolberg, Ludwig (2009): Automatische Gatterextraktion für digitale CMOS Schaltungen, Studienarbeit, Technische Universität München.

Stolberg-Stolberg, Ludwig (2010): Strukturelle Blockdiagramme für analog integrierte Schaltungen, Master's thesis, Technische Universität München, München.

Strasser, Martin (2011): Deterministische hierarchische Platzierung analoger integrierter Schaltungen, PhD thesis, Technische Universität München.

Strasser, Martin, Eick, Michael, Graeb, Helmut & Schlichtmann, Ulf (2011): Deterministic Analog Placement by Enhanced Shape Functions, in H. Graeb (ed.), Analog Layout Synthesis, Springer, Kapitel 3, pp. 95–145.

Tag, Andreas (2010): Symmetrieerkennung für die automatische Dimensionierung von Analogschaltungen, Bachelor's thesis, Technische Universität München.

Takashima, M., Mitsuhashi, T., Chiba, T. & Yoshida, K. (1982): Programs for Verifying Circuit Connectivity of MOS/LSI Mask Artwork, Design Automation, 1982. 19th Conference on, pp. 544–550.

Tarjan, Robert E. (1972): Depth-First Search and Linear Graph Algorithms, SIAM Journal on Computing **1**(2): 146–160.

Tschöpe, Werner (2010): Integration von Dimensionierungsregeln für Analogschaltungen in Cadence Virtuoso(TM), Bachelor's thesis, Technische Universität München.

Tsonev, Dobroslav (2010): Generation of Graph Structures with Application to Structure Synthesis of Analog Integrated Circuits, Master's thesis, Technische Universität München, Munich.

Vallee, R.E. & Masry, E.I. El (1994): A very high-frequency CMOS complementary folded cascode amplifier, Solid-State Circuits, IEEE Journal of **29**(2): 130–133.

Wang, Guoqiang, Kuehlmann, Andreas & Sangiovanni-Vincentelli, Alberto L. (2003): Structural detection of symmetries in Boolean functions, Computer Design, 2003. Proceedings. 21st International Conference on, pp. 498–503.

Watanabe, Takashi, Endo, Makoto & Miyahara, Norio (1983): A New Automatic Logic Interconnection Verification System for VLSI Design, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **2**(2): 70–82.

Wei, Ying & Doboli, Alex (2008): Structural Macromodeling of Analog Circuits Through Model Decoupling and Transformation, IEEE Trans. on CAD of Integrated Circuits and Systems **27**(4): 712–725.

Weste, Neil H. E. & Harris, David (2005): CMOS VLSI Design - A Circuits and Systems Perspective, Pearson Education, Inc.

WiCkeD 6.3 (2010): MunEDA GmbH.

Yang, Lei & Shi, C.-J. Richard (2003): FROSTY: A Fast Hierarchy Extractor for Industrial CMOS Circuits, IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 741–746.

Yi, Su, Dong, Sheqin, Hao, Qingsheng, He, Xiangqing & Hong, Xianlong (2003): Automated Analog Circuits Symmetrical Layout Constraint Extraction by Partition, ASIC, 2003. Proceedings. 5th International Conference on, Bd. 1 von 1, pp. 166–169.

Yokomizo, G., Yoshida, C., Miyama, M., Motono, Y. & Nakajo, K. (1990): A new circuit recognition and reduction method for pattern based circuit simulation, Proceedings of the IEEE Custom Integrated Circuits Conference 1990, pp. 9.4/1–9.4/4.

Youssef, Abdallah (2010): Automatische Strukturanalyse extrahierter Standardzellen, Master's thesis, Technische Universität München.

Zhang, Lin (2011): Evaluation eines automatischen Verfahrens zur Plazierung analoger integrierter Schaltungen, Bachelor's thesis, Technische Universität München.

Zhang, Nian & Wunsch II, Donald C. (2006): Speeding up VLSI Layout Verification Using Fuzzy Attributed Graphs Approach, Fuzzy Systems, IEEE Transactions on **14**(6): 728–737.

Zhang, Pengfei, Zhang, Xisheng & Wu, Yuping (2008): Signal Flow Driven Circuit Analysis and Partitioning Technique, Patent US 7,448,003 B2.

Zhou, Zhe, Dong, Sheqin, Hong, Xianlong, Hao, Qingsheng & Chen, Song (2005): Analog constraints extraction based on the signal flow analysis, ASIC, 2005. ASICON 2005. 6th International Conference On, Bd. 2, pp. 825–828.

Zizala, Stephan (2001): Numerische Verhaltensmodellierung analoger CMOS-Schaltungen unter Berücksichtigung von Dimensionierungsregeln, PhD thesis, Technische Universität München.

Zwillinger, Daniel (1996): CRC Standard Mathematical Tables and Formulae, 30th ed., CRC-Press, Boca Raton, FL, US.