

# FEATS

## Framework for Explorative, Analog Topology Synthesis

Dissertation  
zur Erlangung des Doktorgrades  
der Naturwissenschaften

vorgelegt beim Fachbereich Informatik und Mathematik  
der Goethe-Universität  
in Frankfurt am Main

von

Markus Meissner  
aus Heydebreck

Frankfurt (2015)  
(D 30)

---

## ABSTRACT

The presented work inside this thesis aims to raise the degree of automation in analog circuit design. Therefore, a framework was developed to provide the necessary mechanisms in order to carry out a fully automated analog circuit synthesis, i.e., the construction of an analog circuit fulfilling all previously defined (electrical) specifications.

Nowadays, analog circuit design in general is a very time consuming process compared to a digital design flow. Due to its discrete nature, the digital design process is highly automated and thus very efficient compared to analog circuit design. In modern Very-Large-Scale integration (VLSI) circuits the analog parts are mostly just a small portion of the overall chip area. Although this small portion is known to consume a major part of the needed workforce. Paired with product cycles which constantly get shorter, the time needed to develop the analog parts of an integrated circuit (IC) becomes a determinant factor. Apart from this, the ongoing progress in semiconductor processing technologies promises more speed with less power consumption on smaller areas, forcing the IC developers to keep track with the technology nodes in order to maintain competitiveness. Analog circuitry exhibits the inherent property of being hard to reuse, as porting from one technology node to another imposes critical changes for operating conditions (e.g., supply voltage) - mostly leading to a full redesign for most of the analog modules. This *productivity gap* between digital and analog design resembles the primary motivation for this thesis.

Due to the availability of commercial sizing tools, this work deliberately focuses on the construction of circuit topologies in distinction to parameter synthesis, which can be obtained with a dedicated sizing tool. The focus on circuit construction allows the development of a framework which allows a full design space exploration. This thesis describes the needed concepts and methods to realize a deterministic, explorative analog synthesis framework. Despite this, a reference implementation is presented, which demonstrates the applicability in current analog design flows.

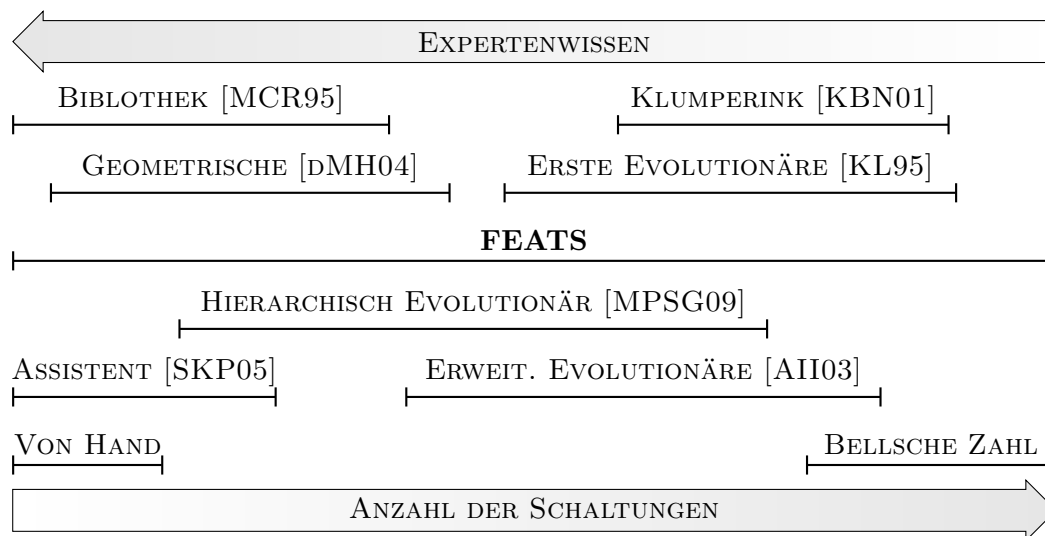


## ZUSAMMENFASSUNG (GERMAN ABSTRACT)

Die in dieser Dissertation vorgestellten Arbeiten verfolgen das Ziel, den Grad der Automatisierung des Entwurfs von integrierten analogen Schaltungen zu erhöhen. Hierfür wurde ein Framework entwickelt, welches die benötigten Mechanismen bereitstellt, um eine voll automatisierte analoge Schaltungssynthese durchführen zu können, d.h. die Konstruktion einer analogen Schaltung, welche alle zuvor definierten (elektrischen) Spezifikationen erfüllt.

Der analoge Entwurfsprozess ist heutzutage ein sehr zeitintensives Unterfangen, insbesondere im Vergleich mit dem digitalen Entwurfsprozess. Durch seine diskrete und damit etwas abstraktere Natur ist der digitale Entwurfsprozess sehr effizient, u.a. da dem Designer Werkzeuge zur Verfügung stehen, die ein hohes Maß an Automatisierung ermöglichen. In modernen integrierten Schaltungen mit einem hohen Integrationsgrad machen die analogen Schaltungsteile zumeist nur einen kleinen Anteil der gesamten Fläche aus. Trotzdem sind der Aufwand und somit die Kosten des Entwurfs unverhältnismäßig groß verglichen mit den digitalen Teilen. Einhergehend mit immer kürzer werdenden Entwurfszeiten wird der analoge Teil auf einem Mikrochip zunehmend der beherrschende Kosten- und Zeitfaktor. Darüber hinaus verspricht der Fortschritt der Prozesstechnologien höhere Geschwindigkeiten mit geringerem Energieverbrauch bei kleinerer Fläche, was die Hersteller dazu zwingt, Schritt zu halten, um weiterhin konkurrenzfähig zu bleiben. Analoge Schaltungen haben die inhärente Eigenschaft schwer wiederverwertbar zu sein, da das Portieren von einem Technologieknoten zum Nächsten nicht selten einher geht mit veränderten Betriebsbedingungen, wie beispielsweise verringerten Versorgungsspannungen. Dies führt zumeist zu einem vollständigen Neuentwurf der meisten analogen Schaltungsteile. Diese daraus entstehende Produktivitätslücke bei der Synthese zwischen analogen und digitalen Schaltungen ist die Hauptmotivation für diese Arbeit. In Letzterer werden neuartige, *deterministische* Verfahren zur *vollautomatischen Synthese von analogen Schaltungen* vorgestellt und demonstriert. Dabei konzentriert sich das hier vorgestellte Framework mit dem Namen FEATS überwiegend auf die Topologiesynthese.

Die in Abbildung 2 dargestellte Relation zwischen *eingeblichem Expertenwissen* und der *Anzahl der Schaltungen* ist eine weitere Kernmotivation für den Entwurf des hier vorgestellten Frameworks. Hierbei werden verschiedene vorgestellte Konzepte in Bezug zueinander gesetzt. Wichtig ist insbesondere der Handentwurf von Schaltungen, der sich am linken Rand der Abbildung befindet; dabei kommt ausschließlich Expertenwissen in Form eines Analogdesigners zum Einsatz. Dem gegenüber steht der absolut naive Ansatz (Bellsche Zahl), welcher *alle* möglichen



**Abbildung 1:** Illustration des Zusammenhangs zwischen eingebrachtem Expertenwissen und der Anzahl der generierten Schaltungen. FEATS bezeichnet das hier vorgestellte Syntheseframework.

Schaltungen generiert und die *richtige* in dieser Menge zu finden versucht. Dieser rein theoretische Ansatz wird ausführlich beleuchtet und definiert zusätzlich den sogenannten *Entwurfsraum für Schaltungen*. Die Abbildung soll darüber hinaus das Ziel veranschaulichen, welches in dieser Dissertation verfolgt wird. Eine Diskussion der Konzepte, welche sich darüber hinaus in der Abbildung befinden, ist auch der vorliegenden Arbeit zu entnehmen.

Im Mittelteil werden die wesentlichen Konzepte und Algorithmen präsentiert, die Verwendung finden. Dabei werden zunächst die benötigten Eingabedaten beleuchtet:

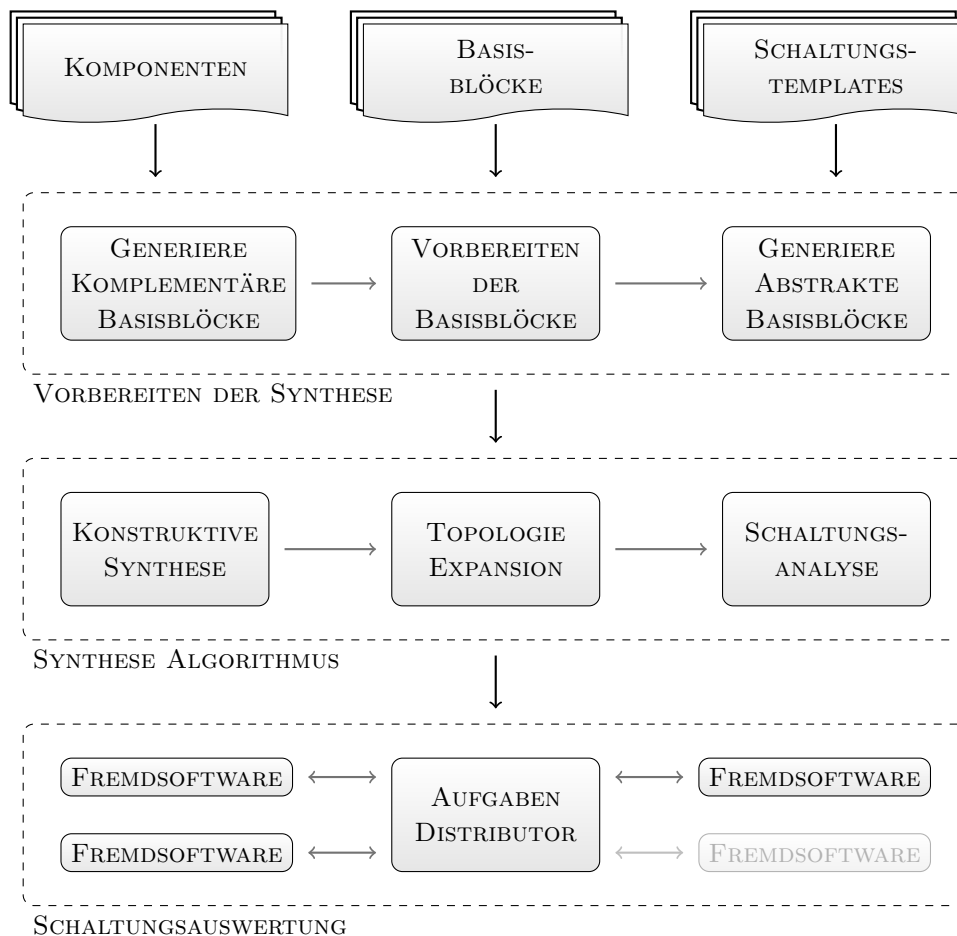
**Komponenten** entsprechen den atomaren Bauteilen einer elektrischen Schaltung. Diese werden in einer abstrakten Form beschrieben (z.B.: hochohmiger Widerstand, präzise Kapazität oder Schalttransistor) und für die Synthese mit einem entsprechend Bauteil aus der Zieltechnologie verknüpft.

**Basisblöcke** bestehen aus einer oder mehreren Komponenten und können auch als *funktionale Blöcke* gesehen werden. Diese können ebenfalls vom Benutzer frei angepasst, hinzugefügt und ersetzt werden.

**Schaltungstemplates** repräsentieren die Hierarchien in dem vorgestellten Framework. Dieses sehr ausdrucksstarke Element des Framework erlaubt die Beschreibung von beliebig tiefen Hierarchien, hierbei liegt der Fokus auf der Wiederverwendbarkeit, d.h. ein zuvor entwickeltes Schaltungstemplate kann ohne weitere Anpassungen in ein anderes Schaltungstemplate eingebettet werden, um so beliebig komplexe analoge Module zu beschreiben und zu synthetisieren.

Der Syntheseablauf ist im Wesentlichen in drei Schritte unterteilt, welche man in Abbildung 2 sehen kann. Zunächst werden die Eingabedaten für die Synthese vorbereitet. Hierbei

werden primär die Basisblöcke betrachtet, die zunächst als Grundlage zum Generieren weiterer Basisblöcke benutzt werden. Daraufhin werden abschließende Konsistenzprüfungen an den Basisblöcken vorgenommen und zusätzliche (automatisch zu generierende) Informationen in eben diesen untergebracht. Schließlich folgt noch ein wichtiger letzter Schritt: Die Klassifizierung der vorliegenden Basisblöcke in sogenannte *abstrakte Basisblöcke*. Es werden mehrere Basisblöcke zu einem abstrakten Basisblock zusammengelegt, abhängig von dessen Eingangs- bzw. Ausgangsspezifikationen. Dieser Schritt zielt darauf ab, die Berechnungskomplexität des Synthesealgorithmus zu verringern und entspricht im weitesten Sinne einem Verhaltensmodell getriebenen Analogentwurf.



**Abbildung 2:** Vollständiger Syntheseablauf mit allen wesentlichen Bestandteilen

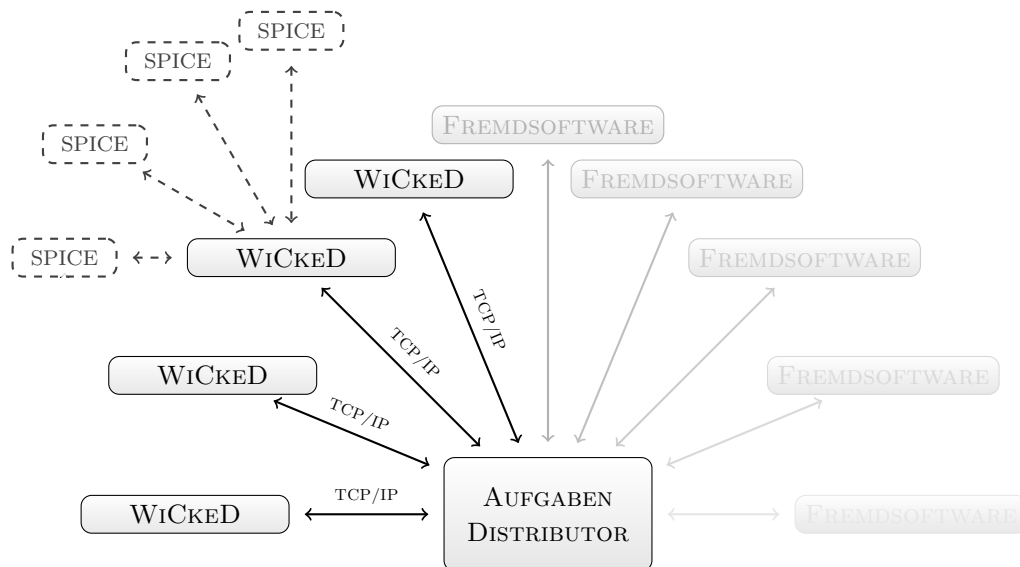
Der Synthesealgorithmus sowie dessen Schritte auf dem Weg zu einer fertigen Schaltung werden ausführlich besprochen, um dem Leser einen genauen Einblick in die internen Algorithmen und Konzepte zu ermöglichen. Hierbei werden (*Synthese-*)*Regeln* eingesetzt, die verschiedene Entwurfsschritte (möglichst abstrakt) realisieren und darüber hinaus eine konsistente Schnittstelle für die Manipulation und Analyse von Schaltungen darstellen.

**Konstruktive Synthese** benutzt *konstruktive Regeln* sowie *destruktive Regeln* um Topologien zu generieren, welche ausschließlich aus abstrakten Basisblöcken bestehen. Der gesamte Entwurfsraum wird dabei exploriert und wird in seiner Ausdehnung nur von den destruktiven Regeln begrenzt.

**Topologie Expansion** ist der nächste Schritt, in dem die zuvor generierten Topologien zu realen Schaltungen expandiert werden.

**Schaltungsanalyse** realisiert einen sehr wichtigen Vorauswahlschritt. Hierbei ist das Ziel die Schaltungen möglichst frühzeitig aus der Synthese auszuschließen, falls sich Eigenschaften finden lassen, die diese mit einer hohen Wahrscheinlichkeit als untauglich klassifizieren.

Der Letzte dieser drei Schritte ist ebenso im Detail beschrieben. Dabei handelt es sich einerseits um einen Isomorphiealgorithmus, welcher zuverlässig Schaltungen aussortiert, die mehrfach vorhanden sind. Andererseits wird eine schnelle Analyse präsentiert, welche Schaltungen mit invertierter Verstärkung identifiziert und von der folgenden Dimensionierung ausschließt. Darüber hinaus werden in diesem Schritt die propagierten elektrischen Nebenbedingungen in ein System von Ungleichungen zusammengefasst. Dieses lässt sich im Folgenden auf nicht-Lösbarkeit überprüfen und liefert in diesem Fall eine zuverlässige Voraussage, ob dies bei einer tatsächlichen Realisierung der Schaltung ebenso eintreten wird.



**Abbildung 3:** Illustration der asynchronen Aufgabendistribution.

Die automatisierte Dimensionierung der Schaltungen ist unabdingbar um Analogsynthese zu betreiben. In FEATS wird dies durch ein kommerziell verfügbares Produkt namens WiCKeD realisiert. Letzteres ermöglicht eine automatisierte, Skript-getriebene Dimensionierung einer Schaltung. Die hierfür benötigten Daten werden vollautomatisch vom Framework erzeugt und bereitgestellt. Ein wichtiges Merkmal des realisierten Konzeptes ist die asynchrone Verteilung von (Dimensionierungs-) Aufgaben an eine beliebige Anzahl von Fremdsoftwareinstanzen, d.h. der



*Aufgaben Distributor* dient nicht nur der Evaluierung respektive, Dimensionierung der Schaltungen, sondern ist wie in Abbildung 3 zu sehen auch noch für das transparente Skalieren der verfügbaren Softwarekapazitäten verantwortlich.

Abschließend werden umfangreiche Analysen und Ergebnisse des vorgestellten Frameworks präsentiert. Hierbei wird die Möglichkeit ergriffen, anhand von realen Schaltungssynthese Beispielen den Ablauf des Syntheseprozesses im Detail zu beleuchten. Die hier gewählten Beispiele sollen eine möglichst breite Abdeckung der möglichen Einsatzgebiete von FEATS repräsentieren:

**Operationsverstärker** dienen als Referenzschaltung, entsprechend wird demonstriert wie das Framework mit Leichtigkeit eine Vielzahl von Lehrbuchschaltungen sowie ungewöhnliche Schaltungen generiert und erfolgreich bzgl. gewünschter Spezifikationen dimensioniert.

**Elliptischer Tiefpass Filter 3. Ordnung** präsentiert die Skalierbarkeit des vorgestellten Frameworks. Dabei wird die Synthese in mehreren Hierarchien ausgeführt, um in der höchsten Hierarchie eine fertige Schaltung zusammensetzen zu können. Das hierbei vollständig automatisiert synthetisierte Analogmodul enthält mehr als 200 Transistoren und präsentiert eindrucksvoll die Leistungsfähigkeit der hier vorliegenden Methode.

Zusammengefasst läßt sich feststellen, dass die hier vorgestellten Konzepte die Möglichkeit bieten, den Entwurf von analogen Schaltungen um signifikante Größenordnungen zu beschleunigen. Darüber hinaus wird demonstriert, dass die analoge Schaltungssynthese während des Entwurfsprozesses zahlreiche Vorteile bieten kann, die dazu führen, dass *schneller bessere* Schaltungen erzeugt werden, die *Produktivität* des Designers wird massiv erhöht und schließlich kann dieser seine Fähigkeiten gezielter einsetzen und auf die echten Probleme des Analogentwurfs richten, anstatt Tage oder sogar Wochen auf den manuellen Schaltungsentwurfprozess zu verschwenden.



## LIST OF ABBREVIATIONS

<b>AAHS</b> analog artificial hormone system	<b>FEATS</b> framework for explorative analog topology synthesis
<b>ABB</b> abstract basic block	
<b>AC</b> alternating current, also analysis type in SPICE simulators	<b>GI</b> graph isomorphism complexity class
<b>ADC</b> analog to digital converter	<b>I/O</b> input and output
<b>API</b> application programming interface	<b>IBR</b> initial block rule
	<b>IC</b> integrated circuit
<b>BB</b> basic block	<b>IEEE</b> Institute of Electrical and Electronics Engineers
<b>BLR</b> block length rule	<b>IIP3</b> third-order intercept point
	<b>ISO</b> (circuit) isomorphism rule
<b>CAP</b> capacitor	<b>ITRS</b> International Technology Roadmap for Semiconductors
<b>CAS</b> computer algebra system	<b>IVR</b> input voltage range
<b>CMFB</b> common-mode feedback amplifier	
<b>CMOS</b> a technology or configuration utilizing nMOS and pMOS devices	<b>LIB</b> library rule
<b>CMRR</b> common-mode rejection ratio	<b>LU decomposition</b> algorithm to solve a square system of linear equations
<b>CNT</b> carbon nanotubes	
<b>CPU</b> central processing unit	<b>MOS</b> metal–oxide–semiconductor field-effect transistor
<b>CTR</b> circuit template rule	
	<b>NB2R</b> no block twice rule
<b>DAC</b> digital to analog converter	<b>nMOS</b> a MOS device with a channel containing mostly electrons
<b>DC</b> directed current, also analysis type in SPICE simulators	<b>NP</b> nondeterministic polynomial time complexity class
<b>DUT</b> device under test	
	<b>OFET</b> organic field-effect transistor
<b>EDA</b> electronic design automation	<b>OMR</b> output match rule
<b>EER</b> elementary electric rule	<b>OP</b> single-ended operational amplifier
<b>ELIPLP</b> elliptical low pass filter	
<b>EXPROPS</b> extract properties rule	
<b>FD</b> fully differential	
<b>FDA</b> fully differential operational amplifier	

## LIST OF ABBREVIATIONS

---

<b>OPCORE</b> operational amplifier core	<b>SoC</b> system on chip
<b>OTA</b> operational transconductance amplifier	<b>SPICE</b> simulation program with integrated circuit emphasis
<b>OTFT</b> organic thin film transistor	<b>SYM EER</b> symmetric elementary electric rule
<b>OVR</b> output voltage range	<b>SYM FEAS</b> symbolic feasibility rule
<b>P</b> polynomial time complexity class	<b>SYM GAIN</b> symbolic gain rule
<b>P1db</b> 1db gain compression point	<b>SYM PRE</b> pre symbolic rule
<b>pMOS</b> a MOS device with a channel containing mostly holes	<b>TCP/IP</b> transmission control protocol/internet protocol
<b>PSRR</b> power supply noise rejection ratio	<b>TIR</b> topology isomorphism rule
<b>RC-net</b> resistor and capacitor network, mostly in the context of (active) filters	<b>TRAN</b> transient analysis in SPICE simulators
<b>RES</b> resistor	<b>VLSI</b> very-large-scale integration
<b>SKLP</b> Sallen-Key low pass filter	<b>VVOER</b> voltage-to-voltage only at end rule
<b>SNDR</b> signal-to-noise and distortion ratio	<b>w.l.o.g.</b> without loss of generality

## LIST OF SYMBOLS

<b>abstract basic block</b> ( $abb$ ) a black box containing one or more BBs sharing the identical I/O characteristic	<b>invariant property</b> ( $p$ ) a property associated with a specific $IV^i$
<b>basic block</b> ( $bb$ ) an electrical basic building block representation with an actual transistor-level implementation	<b>invariants</b> ( $\mathbb{IV}$ ) an unordered set of invariants
<b>big O</b> ( $\mathcal{O}$ ) landau symbol to describe asymptotic behavior of algorithms	<b>label</b> ( $L(x)$ ) label associated with object $x$
<b>circuit</b> ( $c$ ) a circuit representing a physical realization	<b>nets</b> ( $N$ ) an unordered set of nets
<b>circuits</b> ( $C$ ) an unordered set of circuits	<b>parameter space</b> ( $PS$ ) $n$ -dimensional space spanned by the number of parameters $n$
<b>design space</b> ( $DS$ ) design space seen as cross-product of $PS$ and $STS$	<b>partition</b> ( $P^c$ ) a partition of circuit $c$ into subsets $s$
<b>devices</b> ( $M$ ) an unordered set of devices	<b>performance space</b> ( $PERF$ ) $n$ -dimensional space spanned by the number of performances $n$
<b>equivalence class</b> ( $eqcls$ ) an unordered set of objects sharing the combination set of invariant properties	<b>power set</b> ( $\mathcal{P}(x)$ ) power set of a set $x$
<b>equivalence classes</b> ( $EQCLS(c)$ ) set of equivalence classes associated with the circuit $c$	<b>structure space</b> ( $STS$ ) a set of circuit structures
<b>graph</b> ( $G$ ) a representation of object vertices connected through edges	<b>subset</b> ( $s$ ) subset, which may contain vertices
<b>ground</b> ( $V_{GND}$ ) name of the reference net or simply ground	<b>supply</b> ( $V_{DD}$ ) name of the net the supply voltage is made available
<b>hash</b> ( $hash(x)$ ) hash associated with object $x$	<b>threshold voltage</b> ( $V_{TH_{p/n}}$ ) CMOS threshold voltage for either $p$ or $n$ type devices
<b>integer</b> ( $\mathbb{N}$ ) integer numbers	<b>topologies</b> ( $T$ ) an unordered set of topologies
<b>invariant properties</b> ( $EQPROPS(x)$ ) set of invariant properties associated with $x$	<b>topology</b> ( $t$ ) a topology (containing ABBs exclusively)
	<b>vertex</b> ( $u$ ) another vertex inside a graph
	<b>vertex</b> ( $v$ ) a vertex inside a graph
	<b>vertices</b> ( $V$ ) an unordered set of vertices



# CONTENTS

<b>Abstract</b>	<b>i</b>
<b>Zusammenfassung (German Abstract)</b>	<b>iii</b>
<b>List of Abbreviations</b>	<b>ix</b>
<b>List of Symbols</b>	<b>xi</b>
<b>Table of Contents</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 (Traditional) Analog Circuit Design Flow . . . . .	2
1.2 Contributions and Publications . . . . .	4
1.3 Overview . . . . .	5
<b>2 Analog Circuit Representation as a Graph</b>	<b>7</b>
<b>3 Analog Circuit Synthesis</b>	<b>9</b>
3.1 State-of-the-Art in Analog Synthesis . . . . .	10
<b>4 Introspection of the Analog Design Space</b>	<b>15</b>
4.1 Enumerating the Design Space Using the Bell Number . . . . .	15
<b>5 A Framework for Explorative Analog Topology Synthesis</b>	<b>21</b>
5.1 Methodological Considerations and Design Targets . . . . .	22
5.2 Inputs for a Circuit Synthesis . . . . .	23
5.2.1 Components . . . . .	23
5.2.2 Basic Blocks . . . . .	25
5.2.3 Circuit Templates . . . . .	26
5.3 Preparing the Synthesis Process . . . . .	27
5.3.1 Complementary-Symmetric Basic Blocks . . . . .	27
5.3.2 Deduction of Additional Basic Block Properties . . . . .	28
5.3.3 Abstract Basic Blocks . . . . .	29

5.4	Circuit Synthesis Engine . . . . .	30
5.4.1	Constructive Synthesis . . . . .	30
5.4.2	Topology Expansion . . . . .	32
5.4.3	Circuit Analysis . . . . .	32
5.5	Rules – Circuit Manipulation API . . . . .	33
5.5.1	Constructive Rules . . . . .	33
5.5.2	Destructive Rules . . . . .	35
5.5.3	Topology Expansion Rules . . . . .	35
5.5.4	Analyzing Rules . . . . .	36
<b>6</b>	<b>Fast Analyzing Techniques for Huge Circuit Amounts</b>	<b>37</b>
6.1	Circuit Isomorphism . . . . .	37
6.1.1	Graph Isomorphism . . . . .	38
6.1.2	From Graph to Circuit Isomorphism . . . . .	40
6.1.3	Filtering Phase . . . . .	42
6.1.4	Core Circuit Isomorphism Algorithm(s) . . . . .	44
6.1.4.1	Backtracking-based Isomorphism Algorithm . . . . .	46
6.1.4.2	Labeling-based Isomorphism Algorithm . . . . .	47
6.2	Fast Semi-Symbolic Preselection . . . . .	49
6.2.1	Electrical Constraints . . . . .	50
6.2.1.1	Feasibility Analysis . . . . .	51
6.2.1.2	Output Voltage Range . . . . .	51
6.2.2	System Analysis . . . . .	52
6.2.2.1	Gain Approximation . . . . .	52
<b>7</b>	<b>Unattended Circuit Sizing</b>	<b>55</b>
7.1	Task Allocation . . . . .	55
7.2	Sizing . . . . .	56
<b>8</b>	<b>Presentation of Selected Circuit Classes</b>	<b>59</b>
8.1	Utility Circuit Templates . . . . .	60
8.1.1	Bias Circuit . . . . .	60
8.1.2	Voltage-Averaging Circuit . . . . .	62
8.1.3	Common-Mode-Feedback-Amplifier Circuit . . . . .	64
8.2	Circuit Templates for Complex Circuit Classes . . . . .	66
8.2.1	Single-Ended Operational Amplifier (OP) . . . . .	66
8.2.2	Core Operational Amplifier (COREOP) . . . . .	69
8.2.3	Single-Ended Operational Transconductance Amplifier (OTA) . . . . .	71
8.2.4	Fully Differential Operational Amplifier (FDA) . . . . .	72
8.2.5	Sallen-Key 2nd-Order Low-Pass Filter (SKLP) . . . . .	74
8.2.6	Elliptical 3rd-Order Low-Pass Filter (ELIPLP) . . . . .	76



---

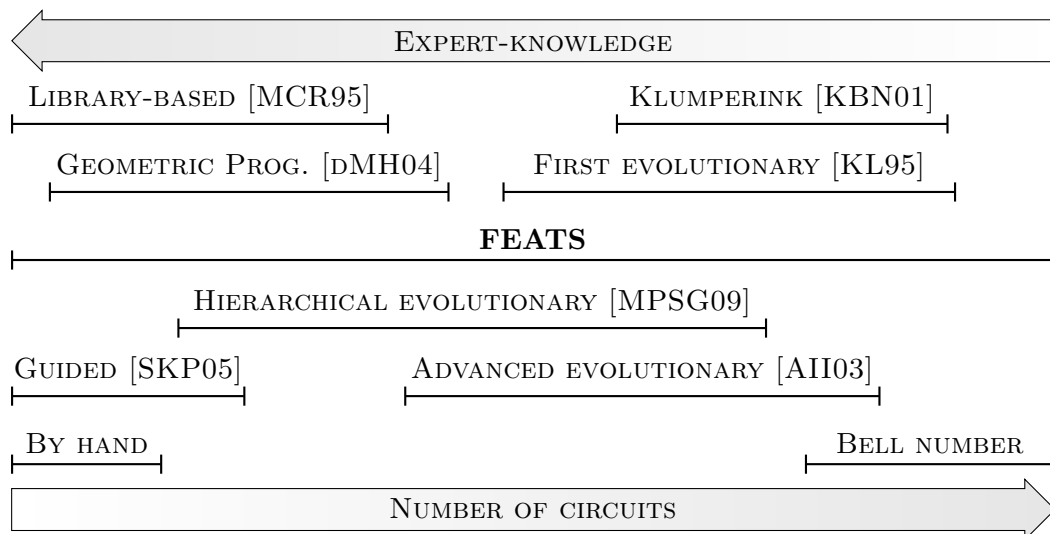
<b>9</b>	<b>Fields of Application for Analog Synthesis</b>	<b>81</b>
9.1	Single-ended Operational Amplifiers . . . . .	81
9.2	Application Inside a System Design . . . . .	85
9.3	Elliptical 3rd-Order Low-Pass Filters . . . . .	86
9.4	Computed Creativity—Current Stealer Design Pattern . . . . .	89
<b>10</b>	<b>Conclusion</b>	<b>91</b>
10.1	Outlook . . . . .	92
	<b>Appendices</b>	<b>95</b>
<b>A</b>	<b>Circuit Class Testbenches</b>	<b>97</b>
<b>B</b>	<b>Basic Block Libraries</b>	<b>99</b>
	<b>List of Tables</b>	<b>103</b>
	<b>List of Figures</b>	<b>106</b>
	<b>List of Algorithms</b>	<b>107</b>

## CONTENTS

---

INTRODUCTION

Analog circuit synthesis is inevitably a long term goal for the industry. To accomplish this, various ideas have been investigated and brought to publications (see Section 3.1). The ongoing advances in processing technologies introduces this very need. The current scientific community around analog circuit synthesis may merely be divided into *circuit construction*, *parameter synthesis* and *layout generation*. The presented methodology focusses on the construction of circuits, which involves the generation of the circuits themselves (see Chapter 5) and despite that, the preselection (see Chapter 6) of feasible circuits for the following sizing.



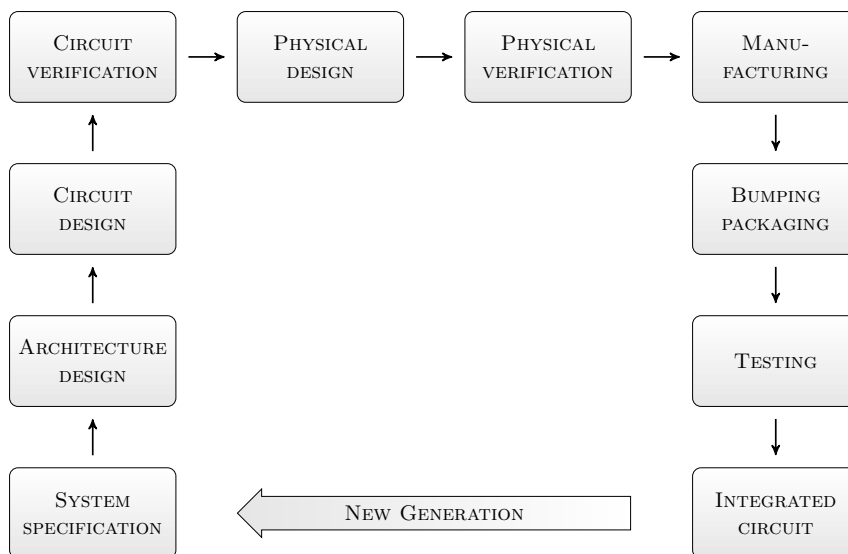
**Figure 1.1:** Illustration of the trade-off between expert-knowledge and circuit count—due to the nature of the underlying problem this is a subjective view and based on the best knowledge of the author. The bell number is a theoretical synthesis approach described in Chapter 4.

The ideas for the construction of circuits span a very wide range and are further illuminated in Section 3.1. In this work the presented methodology aims to deliver a maximum amount of expandability and flexibility in terms of circuit sizes, circuit classes and potential applica-

tions. This originates in a theoretical analysis of the analog design space (see Chapter 4), which illustrates the non-applicability of a brute-force approach in synthesis by enumerating all possible circuits for a given number of components. This approach resembles the naive synthesis without blending in any (expert-) knowledge and thus may be found on the rightmost side in Figure 1.1. On the opposing side of Figure 1.1 the traditional manual design—in Section 1.1 further discussed—is to be found. FEATS introduces the necessary concepts to include and selectively apply *expert-knowledge*, thus determine the amount of generated circuits. By utilizing an extendable *circuit analysis* (see Section 5.4.3) circuits are reliably discarded in an early stage of the synthesis. This not only reduces the runtimes of the whole synthesis, but furthermore the rate of *good* circuits (see Chapter 9).

## 1.1 (Traditional) Analog Circuit Design Flow

The current practice in analog circuit design, especially compared to the digital flow, is mainly driven by highly experienced engineers and mostly lacks automated design steps. Figure 1.2 presents a simplification of a typical design flow for an analog circuit module. The extremely time consuming task *circuit design* consists of several tasks as illustrated in Figure 1.3, which both from an automation point of view contain nearly no automatic steps. The tedious *modify parameters, simulate circuit, evaluate results* iteration—which easily eats up days to several weeks—still dominates the landscape.

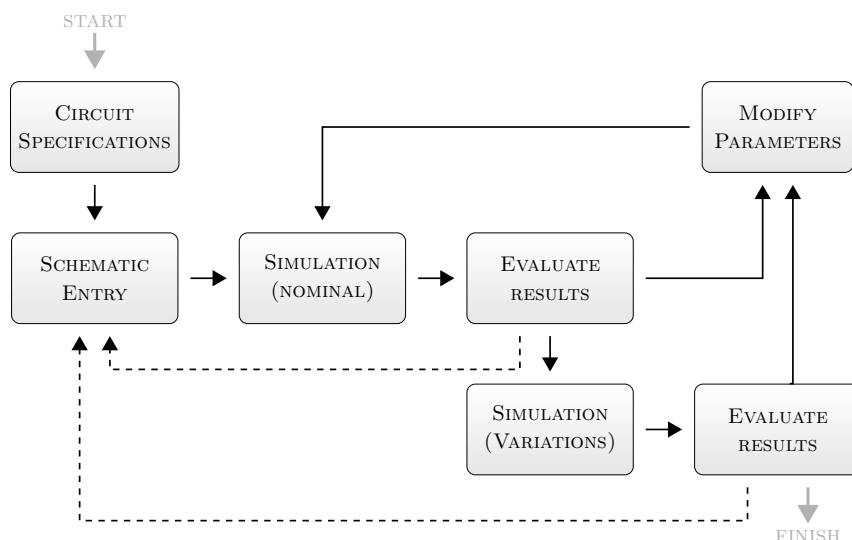


**Figure 1.2:** A (simplified) illustration of the design flow for integrated circuits. The arrows do not imply a constant advancing towards this part—nowadays the step back may be of any distance, thus the backwards pointing arrows were omitted.

The digital design flow is widely dominated by design tools delivering high levels of automation and abstraction, which support the designer towards a finished design. This leads to a less repetitive design flow, where the designer may focus on the more important (higher-

level) tasks and thus avoids the overly tedious circuit design tasks. In particular the digital designer mostly focuses on Register-Transfer-Level design, which hides the used circuits into more abstract building blocks.

The challenges of designing an analog circuit inherently exhibit a much higher level of (computational) complexity due to the continuous nature of analog circuits. Designing an analog circuit from scratch commonly starts with the *circuit construction*, which usually involves a software for schematic entry, testing and evaluation, i.e., simulation. The schematic entry imposes the time consuming task of realizing an initial guess of the circuit by drawing and placing symbols or wires. Once an initial guess was realized as a schematic, the designer starts with the *parameter synthesis* in order to reach the specified performances for the circuit, while the former step (circuit construction) resembles a more or less discrete process: include the device? Interconnect net A with net B? The amount of combinations for a given set of devices is clearly countable and computable (see Chapter 4), thus the amount of possible circuits is still very huge. Contrary to this, the parameter synthesis (sizing) of an analog circuit totally lacks its counterpart inside the digital design flow. Digital modules are designed once and during synthesis, cloned thousands of times, leaving the parameter synthesis to the analog cell designer. Analog designers need to develop a new set of device parameters for each new analog circuit. This is mostly done with uncountable iterations of *modify parameter* and *simulate circuit*, which involves the interpretation of the results (see Figure 1.3). The results are mostly delivered through waveforms, which additionally means, it is not easily adaptable for others except the designer himself. For the parameter synthesis—one of the *big challenges* of analog circuit synthesis—some commercial tools lately started to successfully sustain inside the analog design software ecosystem. Multi-objective-optimization with additional constraints and a very high dimensional parameter space has widely been addressed with various numerical optimization techniques. Section 3.1 gives a short overview regarding the current state of parameter synthesis.



**Figure 1.3:** Simplified analog circuit design flow—zooming into the CIRCUIT DESIGN node from Figure 1.2.

Eventually, the designer identifies a set of parameters, which exhibit the required specifications. But, increasingly often the designer has to forfeit and restart from scratch, as the chosen circuit topology may not always be sufficiently parametrized towards the specified performances. This problem is constantly growing bigger up to a point at which the analog designer is forced to do *statistical massaging*, in other words: repeatedly (fine-) tune the parameters to get the best possible yield, while still fulfilling the specifications, provided he has found a suitable (nominal) circuit topology. Remarkably, the question whether another, better circuit topology - in terms of yield and/or reached specifications - exists or even could outperform the taken circuit in all aspects, is rarely asked.

In the final stage of development a (mostly dedicated) engineer receives the circuit in form of a schematic to realize the circuit on the physical level. This *layout generation* process requires as much additional information as possible, which could be extremely beneficial for the layout engineer. But usually it is carried out ranging from a handwritten sheet of paper to an informative phone call between the designer and the layouter. This traditionally grown *handing over* has recently shown its flaws. The rising complexity, mean variation, shrinking supply voltages and device sizes of recent technology nodes nowadays forces the layout engineer to create a layout, which involves by far more experience and insight into the circuit, as ever before. Matching devices, folding devices using various methods and different priorities of reliability for (groups of) devices lead to reduced process variations, if they are applied correctly.

For sake of completeness the *specification*, deliberately including the evaluation and storing of (simulation) results, recently gets increased awareness. Machine readable, standard-driven specifications, results and the evaluation of those do nearly not exist in industrial environments. Many analog design flows still rely on unstructured, *design documents* providing no insight and documentation about an ongoing or passed design project. Unfortunately, the development in this field is crucial for the applicability of analog circuit synthesis. Most of the analog circuit design automation tools suffer from the fact that almost no machine readable specifications are available, leaving the designer the responsibility to formalize and enter all necessary information into an analog circuit design automation tool.

## 1.2 Contributions and Publications

The first ideas towards the current methodology were published in [WH06, MMH11c, MMH11d], which are used as a blueprint for the core synthesis algorithm, although all rules were reformulated, some of the basic concepts still exist inside the presented work.

The synthesis core engine was one of the first addressed issues due to the limited string-based approach. The realized concepts were first published in [MMH11b, MMH11a] and are now based on graphs as circuit representations, which lead to a necessary increase in flexibility.

The whole methodology was further improved and extended to a more generic approach, while still maintaining a reasonable amount of generated circuits. Due to the application of a highly sophisticated isomorphism algorithm the methodology was enhanced and published in [MMLH12]. With the introduction of *circuit templates*, hierarchical synthesis gets feasible and opens up a wide range of possibilities for the methodology, which was published in [MH15].

Finally, various application studies also have been published. The impact of aging on different operational amplifier topologies was studied in [SHM13] and the applicability of ASDEX, a machine readable specification standard, on automated synthesis has been analyzed in [MMH12]. The (silicon proven) realization of a whole system has also been investigated in [vRMH15]. Furthermore, the methodology was presented in the form of invited talks at various conferences [Mei14, HM13].

### 1.3 Overview

After providing a short introduction into the integrated circuit analog design process in this chapter, the following Chapter 2 provides the necessary formal representation used for circuits inside this thesis. Chapter 3 illustrates the analog circuit synthesis in general and the current scientific landscape—followed by Chapter 4 a theoretical analysis of the (analog) design space, which additionally introduces some important terms and distinctions for synthesis-driven analog design.

A technical in depth presentation of FEATS is given in Chapter 5 to pinpoint the most important concepts and features. Chapter 6 continues to describe the framework internals by illuminating one of the most distinguishing features, the preselection concepts. The middle part is closed with the inspection of the mechanisms for sizing (see Chapter 7) used inside the framework.

A selection of basic blocks and testbenches used throughout the whole framework is shown in Appendix B respectively Appendix A, together with the presentation of selected circuit classes (see Chapter 8) the framework is enabled to synthesize circuits. The application of the latter to actually generate circuits is demonstrated in Chapter 9. Finally, the thesis is finished with a conclusion in Chapter 10.





## ANALOG CIRCUIT REPRESENTATION AS A GRAPH

A circuit is represented through devices and nets—to encode them into an algorithmically processable representation—first we denote the set  $N$  as the *nets* and the set  $M$  as the *devices* of the circuit:

$$N := \{\text{net}_1, \text{net}_2, \dots, \text{net}_n\} \quad (2.1)$$

$$M := \{\text{device}_1, \text{device}_2, \dots, \text{device}_m\} \quad (2.2)$$

The devices and nets are interconnected through  $k$  not directed edges.

$$E := \{e_1, e_2, \dots, e_k\} \quad (2.3)$$

$$e = (u, v) \Leftrightarrow (v, u) : e \in E, u \in M, v \in N \quad (2.4)$$

The sets  $N$  and  $M$  are disjoint and summarized into  $V$ :

$$V = N \cup M : N \cap M = \emptyset \quad (2.5)$$

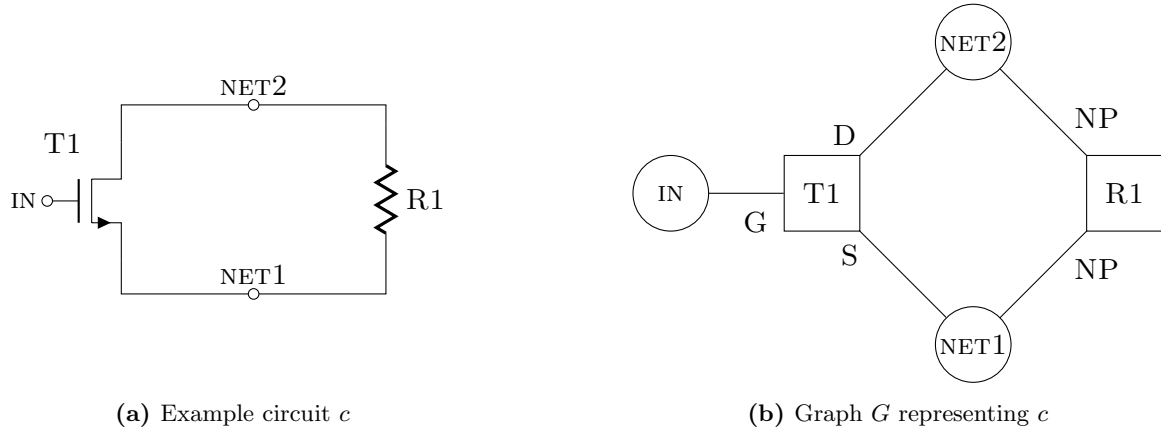
This allows the definition of the strict, bipartite graph  $G$  [Asr98] using Equation 2.1 to Equation 2.5:

$$G = (V, E) \quad (2.6)$$

The property *strict* is enforced by the fact that all nodes are only connected to their respective counterparts:

$$\nexists e = (u, v) : u \in M, v \in M \wedge \quad (2.7)$$

$$\nexists e = (u, v) : u \in N, v \in N \quad (2.8)$$



**Figure 2.1:** A circuit  $c$  represented as a bipartite-graph  $G$

A set of basic blocks  $D^{bb}$  is also referred as a *basic block library*, similarly a set of abstract basic blocks  $D^{abb}$  is denoted as a *abstract basic block library* (see Section 5.2.2).

$$D^{bb} := \{bb_1, bb_2, \dots, bb_n\} \quad (2.9)$$

$$D^{abb} := \{abb_1, abb_2, \dots, abb_m\} \quad (2.10)$$

Throughout this contribution a circuit  $c$  is a graph containing only basic blocks and their interconnections as defined in Equation 2.1 to Equation 2.6:

$$c = (V, E) : V = N \cup D^{bb} \quad (2.11)$$

A topology  $t$  solely consists of abstract basic blocks and resembles a more generic representation of (possibly) multiple circuits and is denoted as:

$$t = (V, E) : V = N \cup D^{abb} \quad (2.12)$$

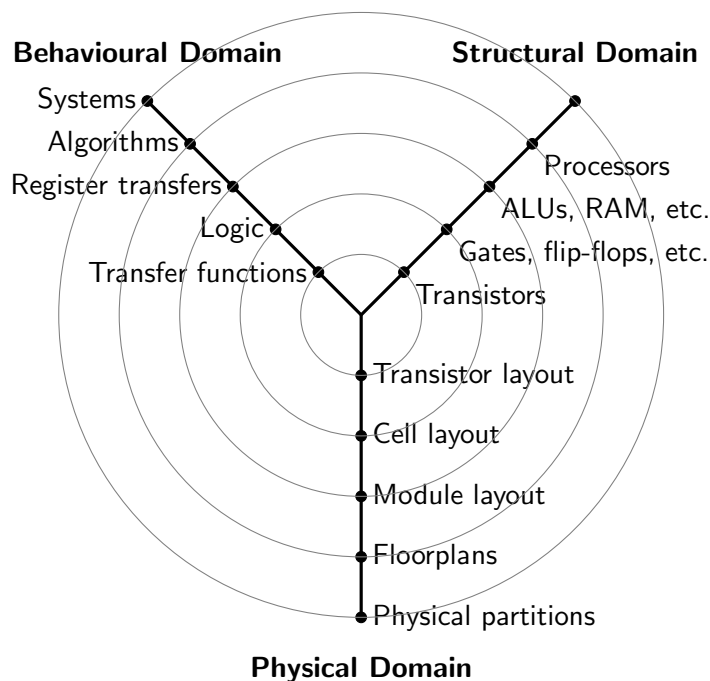
Finally, a set of circuits  $C$  or topologies  $T$  will be named as followed:

$$C := \{c_1, c_2, \dots, c_n\} \quad (2.13)$$

$$T := \{t_1, t_2, \dots, t_m\} \quad (2.14)$$

## ANALOG CIRCUIT SYNTHESIS

Analog circuit synthesis is a widely used term and therefore often also misused. Formally *synthesis* describes a step towards the middle inside the Y-diagram (see Figure 3.1), which leads to a reduction of the abstraction level and therefore a less virtual, more physical representation of the module is generated. While technically the understanding of synthesis in digital design significantly differs from the analog design approach. The general idea still remains identical: An (automated) design step, which converts the module one step further towards the physical realization.



**Figure 3.1:** Gajski-Kuhn Y-chart (from [YCh], license [LPP], unchanged)

The analog synthesis landscape differs massively compared to the digital. While the digital market is near saturation and highly heterogeneous in terms of (automated) digital circuit design software, the typical analog designer workstation nearly has not changed a little during the last years. The established analog design flow has not changed significantly since the advent of the SPICE simulator. According to various scientific contributions during the last years, one might get the impression, the problem is already *solved* [Rut10]. But a closer look reveals that *parameter synthesis* slowly achieves more acceptance and is used by a much bigger user base as it was only some years ago. But opposed to this, persistent tools, which allow automated *circuit construction* or *layout generation* could not sustain inside the semiconductor ecosystem yet. One of the reasons for this development may be the lack of flexibility among the synthesis tools seen on the market. Section 3.1 provides an extensive overview about the evolution of analog circuit synthesis in science and industry during the last years. As previously stated, the high specialization of the analog synthesis tools could be one of the reasons for the still very low acceptance inside the analog design community. Noticeably, the existing commercial *parameter synthesis* tools may *technically* be applied to any circuit in order to optimize the circuit for performance and/or yield, without inherent limitations in circuit size and classes.

### 3.1 State-of-the-Art in Analog Synthesis

In analog circuit design one might split the process into the *construction of circuits* and the *sizing*. This distinction has always existed (see [GR00]), but publications in analog circuit synthesis did not always fulfill this—admittedly this was not possible for a long time due to the lack of commercial sizing tools, which nowadays have successfully sustained inside the analog design ecosystem (e.g., [Mun, Cad]). Analog (structure) synthesis tools did not reach this state for various reasons, which are illuminated inside this section.

Early concepts for analog synthesis can be found in the late 80s, early 90s [HRC89, KSG90, MCR95, AB95]. Mostly the term “analog circuit synthesis” was used to actually describe circuit selection based ideas, which were closely coupled to their circuit libraries and carried out sizings based on pre-assembled parameter to performance rules or equations. While this approach lead to circuits which deliver a high degree of trust, these methods rise and fall with their accompanied libraries and sizing templates. Each new process node will always force a refinement of the library and its sizing templates. This refinement inherently leads to an enormous amount of work, which must be done upfront. The development of such libraries, i.e., the circuits and their individual sizing templates, may only be done by highly experienced analog design engineers—it is by far more difficult to design a *generic* circuit and its sizing templates than a *regular* circuit design towards a specific set of constant performances.

These methods retained a few process nodes, but as the effects of Moore’s law get stronger, the aforementioned refinement gets increasingly complicated in terms of consumed time. Furthermore, the operating conditions have massively changed, i.e., the supply voltage has dropped significantly. This by itself is highly desirable—mainly due to the reduced power consumption—paired with a reduced voltage range, which will inevitably lead to changed circuit structures. Particularly this means various circuit constructions may not be used anymore, e.g., (multiple)

casccodes. This, together with 2nd-order effects due to the ongoing geometrical scaling (i.e., short channel effects) has largely lead to the fact that most of the research in this field has stalled. However, the selection, or library-based circuit synthesis approaches delivered very *trustful* circuits, inherently by the fact that solely circuit topologies are used which were previously designed from scratch by experienced analog designers.

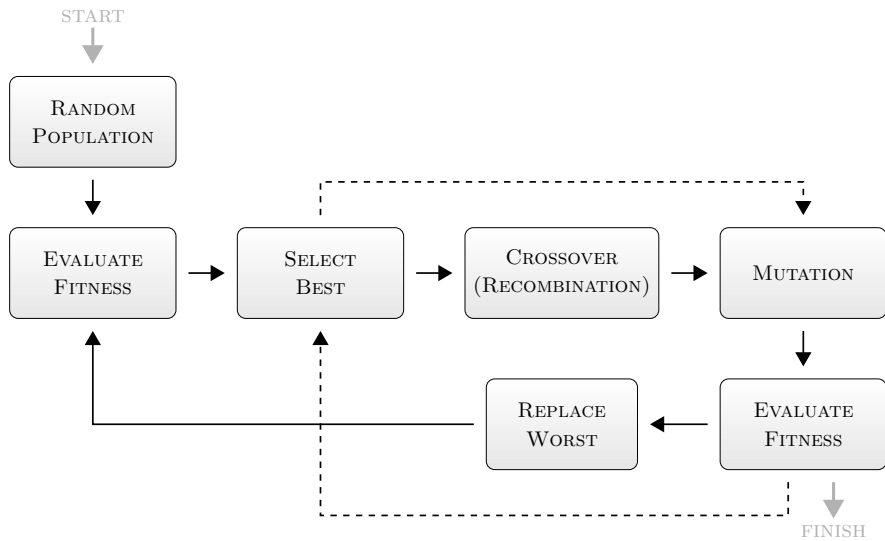
Consequently, the natural next step is the opposing extreme. This was primarily done with brute-force like approaches by Klumperink et al. [KBN01, BKN00]. Remarkably, this approach has an explorative nature, which is maybe the first of its kind. The circuit generation is based on the idea of modeling a MOS-device as a voltage-controlled current-source, which seems reasonable as long as one solely aims to generate circuits containing exclusively MOS-devices in saturation and strong inversion. This approach already encounters the challenges of exploration based circuit synthesis. By generating all possible interconnections between a very limited number of MOS-devices this approach generates a vast amount of circuits already for device counts of three. Notably, this methodology exposed the challenges of exploration-based analog circuit synthesis to the scientific community:

- Vast amounts of circuits with very small component counts show the need for a preselection methodology
- Even well performing circuits lack trust
- Sizing has to be considered during analog circuit synthesis
- Analog circuit synthesis without blending in any (expert) knowledge will never be feasible (see Chapter 4)

While the explorative idea itself looks very promising, the computational effort to actually apply a full design exploration—without introducing (knowledge-based) bounds for the design space—is huge.

During the same time the *evolutionary algorithms* (see Figure 3.2) were first applied for circuit synthesis. Evolutionary algorithms try to mimic the evolution—inspired by nature itself. Therefore, a chromosome represents the circuit (or the building instructions for a circuit), which is also called the *genotype* of the circuit, whilst the incarnation of the latter is noted as *phenotype*. An initial *population*, i.e., a set of chromosomes, is generated randomly. For each of them the *fitness* is calculated by a specific fitness function. Based on this, the best individuals are chosen (parents) which are now being used for *crossover* and/or mutation in order to generate a new generation. The fitness function is now applied to the new individuals. Finally, the least-fit population is replaced with new random individuals and the process is repeated until a specified end condition is reached. This could be a maximum generation count or a desired fitness function target.

Kruiskamp et al. [KL95] presented *DARWIN*, which introduced a matrix representation of the circuits as in Figure 3.3a to be synthesized. In particular the rows and columns of the matrix represent a component and each cell is either true or false, which translates to a connection between those two components. This method introduces an approach which allows easy insight



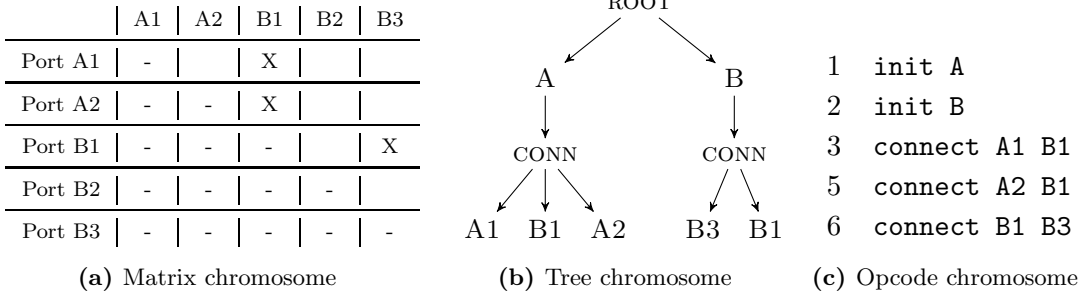
**Figure 3.2:** The typical—biological inspired—evolutionary algorithm flow. Various modifications are known to this flow—dashed lines resemble alternate flows.

into the evolutionary process, as crossover and mutation operations may be visualized with ease. This method introduces various very circuit class specific properties, which may not be easily adapted for other circuit classes, thus it inherently restricts the application scope.

Koza et al. [KBL<sup>+</sup>97] proposed a *tree structure* chromosome as shown in Figure 3.3b to represent circuits. Starting with a so-called *embryonic circuit* the circuit gets evolved using *component-creating functions*, *connection-modifying functions* and *arithmetic-performing subtrees* which may solely be applied to components. Each group has several specific functions realizing design steps, which are highly specialized on specific components, which significantly reduces the flexibility and expandability of this approach. Furthermore, the memory footprint and thus the computational overhead is significant (see [AII03]). Despite this, these properties implicitly introduce restrictions for the size of the circuits to be synthesized.

Another approach tailors the genotype description of the circuit by means of *opcodes* (see Figure 3.3c) similar to what happens during translation of assembler code. Lohn et al. [LC99] thus represent a circuit as a set of chronologically executed operations, which finally construct a circuit. This approach is quite similar to what has been done by Ando et al. [AII03], but is here explicitly named as *variable length chromosome*. Strictly seen, except for the matrix approach (see [KL95]), most evolutionary synthesis methodologies use some kind of variable length chromosomes. Although it is worth to be mentioned, as the natural blueprint is obviously based on a constant length chromosome. On the other side, a variable length chromosome for circuit synthesis is clearly the correct representation due to the fact that circuits themselves have a variable length, respectively size.

A major drawback for the previously discussed methods is the integrated sizing mechanism. While on the first view the integration of sizing into circuit construction allows a higher circuit throughput due to the tighter coupling between circuit construction and sizing, the comparability and the reliability of the methodology strongly correlates with the quality of the sizing methods



**Figure 3.3:** Simplified illustrations of various used chromosomes for evolutionary synthesis for two example devices A and B with two, respectively three ports.

and algorithms. Admittedly this was the only way to generate any presentable results, as commercial sizing software emerged during the last ten years at least, making it impossible to use a state-of-the-art sizing tool during the time of these contributions. Another issue arises once a closer look is taken at the generated circuits, which are often very uncommon and thus lack trust, leading to a reduced acceptance of these among the analog design community.

A widespread solution for the lack of trust is the concept of basic blocks. First introduced by [DCR05] and Wang et al. [WH06] the concept has been widely used (see [MPSG09, DV08]) to overcome the trust drawbacks and to reduce the computational complexity of analog circuit synthesis.

The latest incarnations of the evolutionary methods are resembled through the contributions by Das et al. [DV08] and McConaghy et al. [MPSG09]. While the former employs hierarchies indirectly, the latter explicitly uses an hierarchical approach to propagate properties of building blocks from bottom to top. This leads to so-called *flexible* and *compound blocks*, which may contain several *atomic blocks*. This approach exhibits similar properties, in terms of full design space exploration, but limits itself as it is not truly exploring the design space, due to the stochastic nature of the underlying evolutionary algorithm.

Furthermore the analog circuit synthesis landscape is full of (semi-)automated methodologies aiming to be applicable for specific circuit classes only. Starting with passive filter synthesis (see [DV07, CHS06]), switched capacitor filter synthesis [AEBD00], low noise amplifier synthesis [TB08, BKN04, TB05], low drop out circuits [DM09] to  $\Delta\Sigma$ -synthesis [TD06]. Assistant based tools were also investigated, which support and guide the designer through the development of an analog circuit [SKP05]. Inherently those methodologies suffer from their limited applicability, which may be one of the primary reasons why, to the best knowledge of the author, none of them emerged into an actual commercial tool.

In contrast to the analog circuit synthesis, the analog parameter synthesis, i.e., automated sizing has successfully reached and sustained inside the industrial analog design ecosystem. The most prominent and successful ones are based on *worst-case distances* (see [AGW94, AEG<sup>+</sup>00]). Further ideas have been developed in this field based on *support vector machines* [DBNV05], geostatistics performance modeling [YL07], downhill-driven stochastic parallel recombination [KPH<sup>+</sup>01], discretized sizing [JCK12], spline center and range methods [BKV09] and hybrid evolutionary-driven methods like presented in Lourenço et al. [LH12]. Especially, the concepts

based on *geometric programming* have been applied in various forms to the parameter synthesis problem. Even selection-based synthesis methods—by sizing a library of circuits and returning the *best* one—have been brought to publications. Generally, a geometric program is an optimization problem of the form:

$$f_i(x) \leq 1 \quad i = 1, \dots, m \quad (3.1)$$

$$h_i(x) = 1 \quad i = 1, \dots, p \quad (3.2)$$

$f_0, \dots, f_m$  are *posynomials* and  $h_0, \dots, h_p$  are *monomials*. By formulating the specifications to be reached as posynomial functions the geometric program may be converted into a convex optimization problem. This is done by taking the logarithm of the input variables, objectives and constraints. The very same posynomial formulations are the self induced limitations as the handwritten posynomials mostly rely on first-order models as in [MV01, dMH04]. Geometric programming promises a very fast calculation of a global optimum using interior point methods. Still, the applicability is questionable due to the fact that most posynomials are handwritten and not easily deduced in an automated process. An approach to overcome these drawbacks is presented by Aggarwal et al. [AO07], they focus on the automated generation of posynomial equations. Though the current ideas are applied only on transistor model parameters (implicitly assuming saturation and inversion), they do not (yet) address full circuit level specifications. Eventually, geometric programming has also been applied to model and include layout-dependent effects into the sizing process [ZLY<sup>+</sup>12].

Generating a layout from a given circuit is the obvious final step to tackle in order to accomplish a higher degree of automation in analog circuit design. Graupner et al. [GJW11] presented a *generator based approach*, which is based on *executable design flow descriptions*. This translates to a programming inspired analog circuit design approach by providing a set of procedures and functionalities, which may be used to describe the steps to be taken in order to generate a target circuit. Although this method allows automated circuit and layout generation, the description of design steps using a programming language is not the most intuitive way for an analog design engineer. Other methodologies to automatically layout an analog circuit have been proposed (like [LVGH06, ESL<sup>+</sup>11]), which mostly have to pre-process the target circuit by means of symmetry, matching and partitioning. This inherently is a hard challenge due to the fact that the previously included design knowledge during circuit sizing and construction gets lost and has to be extracted from the circuit schematic. Thus an automated layout would benefit massively from the (formalized) information, which could be delivered by an automated circuit generation methodology.

Finally, this work addresses the problem of isomorphic circuits during circuit construction. This is, by the best knowledge of the author, the first approach handling this inevitable problem. There have been contributions addressing the automated extraction of similarities and symmetries (see [LVGH06, ESL<sup>+</sup>11, Eic13, EG12]) but isomorphism by itself was never illuminated. Generally a similar problem occurs during the layout vs. schematic design step and is fully solved inside the commercial design automation environments. First contributions in this field are found in the 90s and serve as a base for current realizations [Ebe88, OE93].



## INTROSPECTION OF THE ANALOG DESIGN SPACE

The increasingly often abused term *design space* is mostly used in conjunction with circuit synthesis. Thus sometimes misused in the context of sizing, where obviously *parameter space* should have been used. The aim of this chapter is to provide an exact enumeration of *all possible circuits for a given number of ports*, and furthermore to put this into relation with the presented analog synthesis framework. Finally the following excerpts should provide a precise distinction between the terms *structure space*, *parameter space*, *design space*, and *performance space*.

### 4.1 Enumerating the Design Space Using the Bell Number

Before analyzing, or even enumerating, the structure space, it has to be defined. As a preparation, it is worth taking a look at nets and components. In fact the components, i.e., the type of the component has, despite of its number of connections (i.e., ports), no influence on the *size* of the structure space. Translating this formalism to reality means, a circuit containing two nMOS components, opens up a structure space, which has exactly the same size as the resulting structure space for three simple (bipole) resistors—in distinction to the design space, which includes the parameter space and thus differs.

While this seems unintuitive, this is a direct consequence of the absence of an (easily) computable mapping between functionality and circuit structure in analog circuits. The following investigations of the analog design space is thus a very important step to further understand the challenges of analog circuit synthesis. Furthermore the possibility to state the opposing: “It is impossible to synthesize the specified circuit.”<sup>1</sup>

The following Lemma will support these thoughts and first denote an important property of the structure space:

**Lemma 4.1** (Structure Space Size/Dimensions). *The structure space has w.l.o.g. only one dimension, this translates to an unordered set of items, all without any particular order or direct*

---

<sup>1</sup>*Impossible* is to be understood as a simplification for “Mathematically not computable, using all computing power on earth for the next 1,000,000 years.”

relation to each other. The size of structure space  $STS$  will be noted as  $|STS|$  and is strictly defined by the number of circuits inside the structure space.

Once the size  $|STS|$  is well defined, the circuits inside  $STS$  can be further illuminated. A *hypothetical circuit synthesizer*, which fills the structure space with circuits, does w.l.o.g. not take any circuits' functionalities or electrical properties into account during circuit synthesis, thus making no assumptions about the (potential) functions of a circuit.

In contrast, there are various structural characteristics, which will be defined in the following:

**Definition 4.2** (Circuits Inside The Structure Space). *For each circuit  $c$  associated with a structure space  $STS$  the following applies:*

- a) *For each circuit  $c$  there is no other circuit  $c'$  inside the analog design space  $DS_A$ , which is structurally identical.*
- b) *Outer pins for each circuit are omitted.*
- c) *All circuits have the exact same amount of component ports.*

Given these preconditions, the *structure space* starts to get manageable. But first a quick explanation, why these definitions are very helpful without introducing unnecessary simplifications, i.e., they do not reduce the complexity of the analyzed problem. Item a) is pretty self-explanatory, as identical circuits inside the structure space set would not be helpful for a formal analysis. Furthermore Item b) is, at least, easily approximated in terms of how many circuits are forked from one. Omitting the outer pins sounds more infringing as it actually is. Given  $n$  outer ports and  $m$  nets inside a specific circuit  $c_i$  (with  $m$  being the total number of nets and  $n$  a portion of it), the number of forked circuits  $|C_{c_i}|$  is easily calculated, assuming no outer port is shorted with another outer port.

$$|C_{c_i}| = \frac{m!}{(m-n)!} \quad : \quad m > n, \quad m \in \mathbb{N}^+, \quad n \in \mathbb{N}_{\geq 0}^+ \quad (4.1)$$

During analog synthesis the outer pins are actually very important and ignoring them would generate enormous amounts of useless circuits. Nevertheless in this context, the amount of circuits generated through the Equation 4.1 is compared to the structure space, just a little fraction. Especially, from the complexity point of view, i.e., the asymptotic behavior is very good natured. Equation 4.1 would usually rise in a very steep fashion with respect to  $m$  and  $n$ —fortunately only  $m$  is getting (significantly) bigger with larger circuits. In contrast to  $n$ , which represents the number of outer ports of the circuit. For analog circuitry the outer pins can be *nearly* seen as constant in the context of asymptotic behavior. Additionally, Equation 4.1 rises at the fastest rate if the denominator gets smaller, until it reaches 1, what resembles the nonexistence of the denominator, thus the equation would rise equally fast as any other factorial. Summarized, this means Item b) from Definition 4.2 may safely be applied. Finally, Item c) has no actual impact on the methodology or the analysis, moreover it emphasizes the fact that in the context of analog design space analysis the number of component ports inside a circuit determines the complexity and dominates the growth.

Given the previous observations the following definition of a structure space may be given:

**Definition 4.3** (Structure Space). *Let  $K$  be a specific set of components and  $P(K)$  the set of all ports provided by the components. The number of ports will be denoted as  $|P(K)|$  with  $f \mapsto \mathbb{N}$  being the function to determine the actual size. This allows the definition of the structure space for a given number of ports:*

$$STS(K) := f(|P(K)|)$$

The following challenge is to actually determine  $f$ . Therefore a simple reinterpretation of the circuit and its contents is needed. Let's assume the *circuit* consists solely of a tripole device (e.g., an nMOS-device) with three pins named  $D$ ,  $G$  and  $S$ . As previously stated, the outer connections of this circuit are omitted, thus the task is to determine how the three pins may be interconnected. For this example all possible interconnections are easily enumerated:

$$\{D, G, S\} \rightarrow \{(D, G, S)\} \quad (4.2)$$

$$\rightarrow \{(D), (G, S)\} \quad (4.3)$$

$$\rightarrow \{(G), (S, D)\} \quad (4.4)$$

$$\rightarrow \{(S), (D, G)\} \quad (4.5)$$

$$\rightarrow \{(D), (G), (S)\} \quad (4.6)$$

A net—the set of shorted pins—is described by all pins inside one pair of parenthesis (right hand side). Although this notation is very uncommon in electrical engineering, it perfectly matches the common notation of *partitions of a set*. As the name suggests: on the left side of Equation 4.2 is the set and the partitions on the right Equation 4.2 to Equation 4.6 resemble all possible interconnections of the pins.

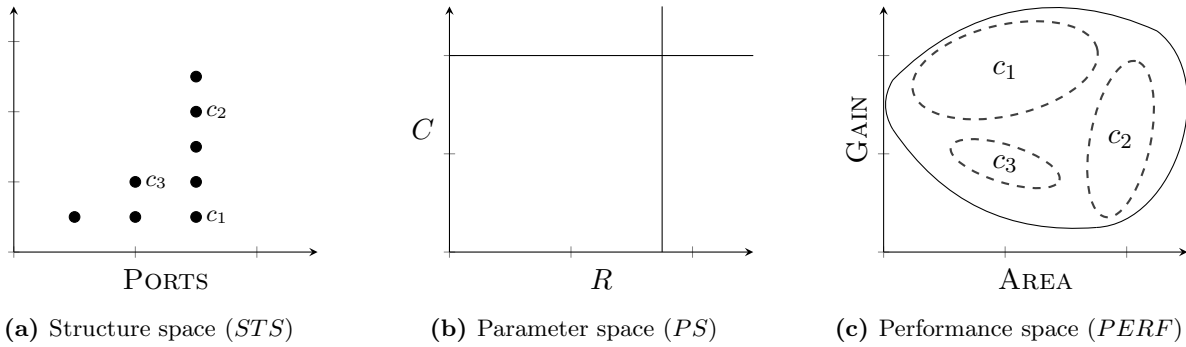
In math *partitions of a set* is a widely known and analyzed concept. Furthermore, the number of partitions of a set can easily be calculated using the *Bell number* [Aig99]. Knowing this, it is now possible to define  $f$  from Definition 4.3 and thus the exact size of the structure space for a given set of components respectively pins.

Each circuit resembled as a dot in Figure 4.1a, furthermore spans a parameter space, which dimensionality is equal to the number of freely adjustable design variables. The example in Figure 4.1b therefore shows a  $C$  (capacity) and an  $R$  (resistance) with their boundaries. It is worth mentioning that the parameter space, opposing to the design space it is nearly continuous. *Nearly* because in real process technologies there is a manufacturing grid, which technically allows only discrete steps for a particular design variable (component parameter).

**Definition 4.4** (Parameter Space). *Each circuit  $c$  spans its own parameter space  $PS(c)$ . The degrees of freedom for this parameter space is determined by the number of design variables for this particular circuit  $c$ .*

**Definition 4.5** (Design Space). *The cross product of the structure space  $STS$  with the parameter space  $PS$  resembles all possible circuits including the sizing.*

$$STS(K) \times PS(c) = DS(K)$$



**Figure 4.1:** An illustration of the degrees of freedom within a circuit development process.

Noticeably, the design space by itself is not bound and grows infinitely. To restrict it and to provide understandable limits, the Definition 4.3 may and should be applied in order to get a manageable enumeration of the design space. Eventually, the design space is mapped into the performance space ( $PERF$ ) as shown in Figure 4.1c.

**Definition 4.6** (Performance Space). *Let  $DS(K)$  be the design space spawned by the structure space  $STS(K)$  and the parameter space  $PS(c)$ . Each (sized) circuit inside  $DS(K)$  maps to a point inside the performance space  $PERF$ . The dimensionality of the performance space is determined by the number of performances.*

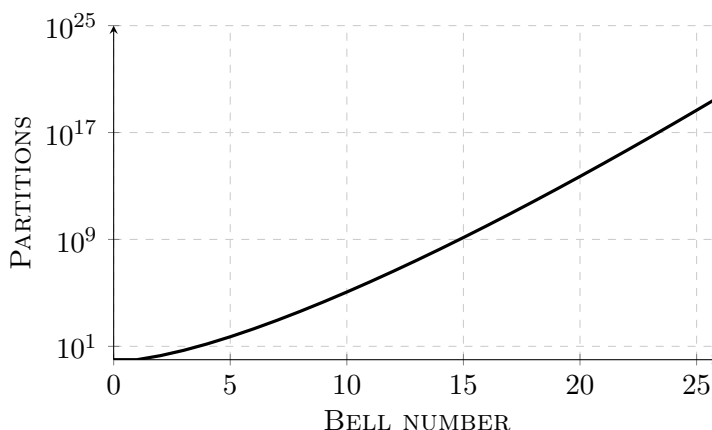
$$DS(K) \mapsto PERF$$

Unfortunately, the mapping is not bijective, i.e., it is not possible to directly derive the parameters and the design of a circuit from a given set of performances. So the only way to fulfill a given set of performances is to *go from left to right*.

The following Table 4.1 should provide an orientation for the reader how complex this task may be without including *expert knowledge* into the process. One may instantly see the very step increase of the number of possible circuits, which undeniably leads to the fact that circuit synthesis without blending in expert knowledge is unfeasible (see Chapter 1). This should moreover serve as a motivation to include this aforementioned expert knowledge into the framework.

TABLE 4.1  
BELL NUMBERS FROM 1 TO 16

PORTS	$B_N$	PORTS	$B_N$
1	1	9	21,147
2	2	10	115,975
3	5	11	678,570
4	15	12	4,213,597
5	52	13	27,644,437
6	203	14	190,899,322
7	877	15	1,382,958,545
8	4,140	16	10,480,142,147



**Figure 4.2:** Graph showing Bell numbers

Table 4.1 and Figure 4.2 impressively show the steep rise of the bell number. The Bell number may be calculated using a recursive expression, with  $B_0 = 0$  being defined previously.

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k \quad (4.7)$$

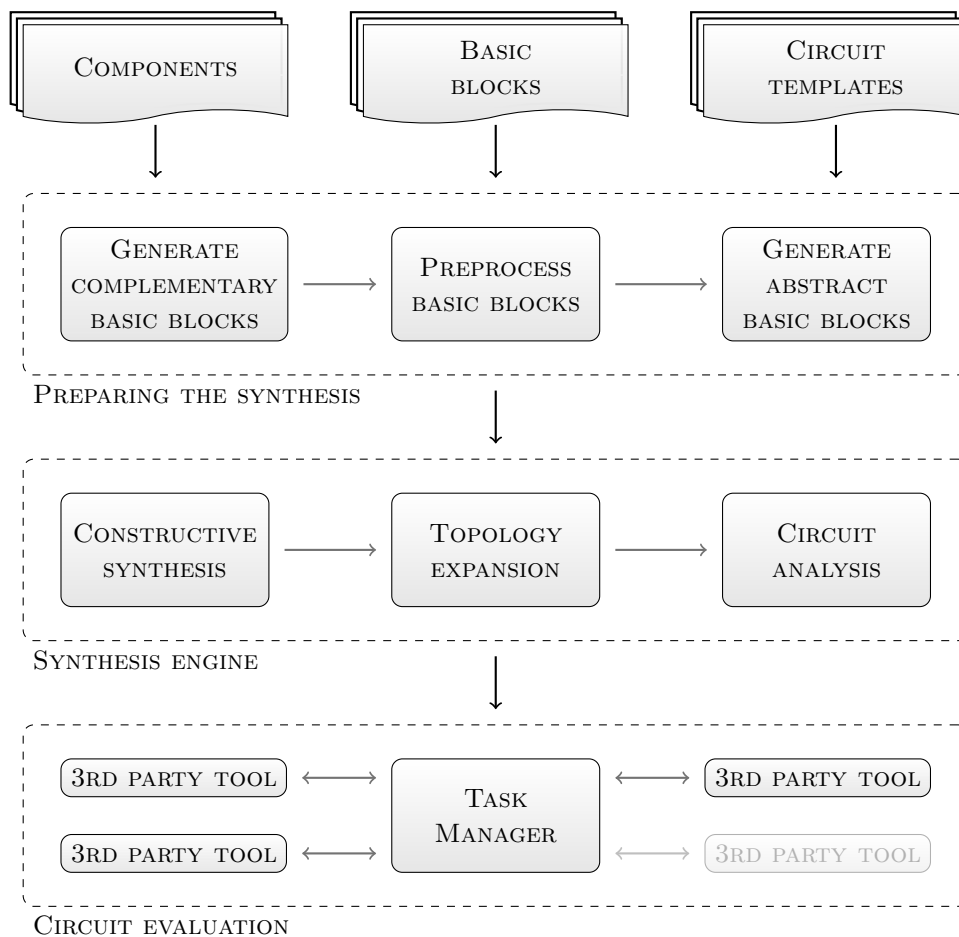
There are various other possibilities to calculate the Bell number and even more approximations, which are not further investigated, as they deliver nothing beneficial for the current analysis.

Nevertheless, these numbers impose a suitable—moreover, even a precise upper boundary for the number of structures to be generated from a given set of components. Fortunately, this is a *theoretical* analysis and is primarily presented to highlight the capabilities of an analog design engineer. Based on *experience*, *simple calculations* and *elementary electrical rules* for voltage and current, a designer develops circuits without even remotely exploring the full *structure space*. In other words, the designer utilizes *expert knowledge* (see Chapter 1) to implicitly discard the vast majority of the structure space. Some, but not all, are directly applicable to the here presented net-based circuit representation, e.g., the amount of ports connected to a net is, except for the supply and ground nets, rarely above four.

By imitating and formalizing the methods the analog circuit designer uses to develop circuits, the presented framework is able to reduce the number of circuits to be evaluated down to a reasonable amount—Chapter 9 aims to provide empirical proof for this statement.



## A FRAMEWORK FOR EXPLORATIVE ANALOG TOPOLOGY SYNTHESIS



**Figure 5.1:** FEATS' top-level flow

The framework consists of several steps, which are visualized in Figure 5.1. Inside the current chapter the various steps and inputs of the synthesis are explained in detail to allow a clear view into the internals of the presented methodology.

As discussed in Chapter 4, the *design space* for analog circuits grows rapidly for rising numbers of used components. Although an analog design engineer is able to build circuits with large component counts, one might get the impression an algorithmic approach may never get even near to this point. The actual number of somehow useful circuits inside this huge design space resembles only a tiny portion.

The primary design goal of the presented explorative topology synthesis engine is the possibility to precisely control how much knowledge is included into the synthesis process. As described in Chapter 3, the process of *circuit construction* may range from pure brute force approaches (see Chapter 4) to hand crafted, experience based circuit creation. The presented framework aims to provide both extrema and additionally anything in between. Clearly the framework itself may not—out of the box—provide this functionality, but moreover it provides the, literally spoken, *framework of tools and hooks* to allow the user to tweak and configure the software in order to get the desired results.

## 5.1 Methodological Considerations and Design Targets

The primary *design targets* for an analog synthesis framework are easily enumerated:

- Flexibility / genericity
- Process independence
- Circuit(-class) independence
- Arbitrary hierarchy
- (Hardware) platform independent
- Optimizer / simulator independent
- Good-natured scaling
- Semantically rich

While the claimed *flexibility* and *genericity* are more a product of the whole framework itself and are not so easily explained and pinpointed. The other points may be directly matched with features provided by the framework.

The previous enumeration does not imply any kind of ordering, but *process independence* is most likely one of the most important points to fulfill. A synthesis framework without a strong emphasis on process independence will most likely be obsolete at the moment the target process will be replaced. The presented synthesis framework considers changing process parameters and other process dependent changes throughout the whole implementation. Starting with the *components* (see Section 5.2.1) the framework already abstracts the atomic building block. In application this means there is an arbitrary count of different atomic components to be used inside the synthesized circuits. Preparing the framework for current (and following) technology nodes, which already exhibit various different components like resistances, capacitances, and a wide variety of MOS-components, which mostly target specific use cases. By giving a component a *semantically rich* description, like: *driving MOS-component*, *high capacity capacitor*, *high precision resistance* etc. the component may easily be mapped to an equivalent component



provided by a specific technology node. By additionally measuring different properties of the target process node, various *numerical properties* are extracted, which are taken into account during the semi-symbolic circuit evaluation (see Chapter 6) executed by the *analyzing rules* (see Section 5.5).

A *circuit class independence* goes hand-in-hand with the need to have the possibility to describe *arbitrary hierarchies*. The presented synthesis framework allows hierarchies of arbitrary depths with *circuit templates*, these may contain any number of circuit templates themselves, which allows the user to create circuits and hierarchies of any depth. By further allowing to assign different *libraries of basic blocks*, as shown in Appendix B, to any circuit template type—the user moreover gets the possibility to combine different types of circuits together to construct circuits of arbitrary complexity and size.

By *platform independence* it is not specifically meant to be independent from ARM, x86 or any other hardware architecture (although it is), moreover the aim is to not be dependent on big computing capacitances. In simple terms, the framework should work with a reasonable speed on a decent enterprise scale workstation. But additionally, it should scale well to an arbitrary number of servers and/or workstations. Therefore various levels of abstraction were realized, which are partly observable as clearly distinguished inputs (see Section 5.2), but additionally the synthesis framework internally does an extensive preprocessing of the inputs in order to further reduce the needed computational effort (see Section 5.3).

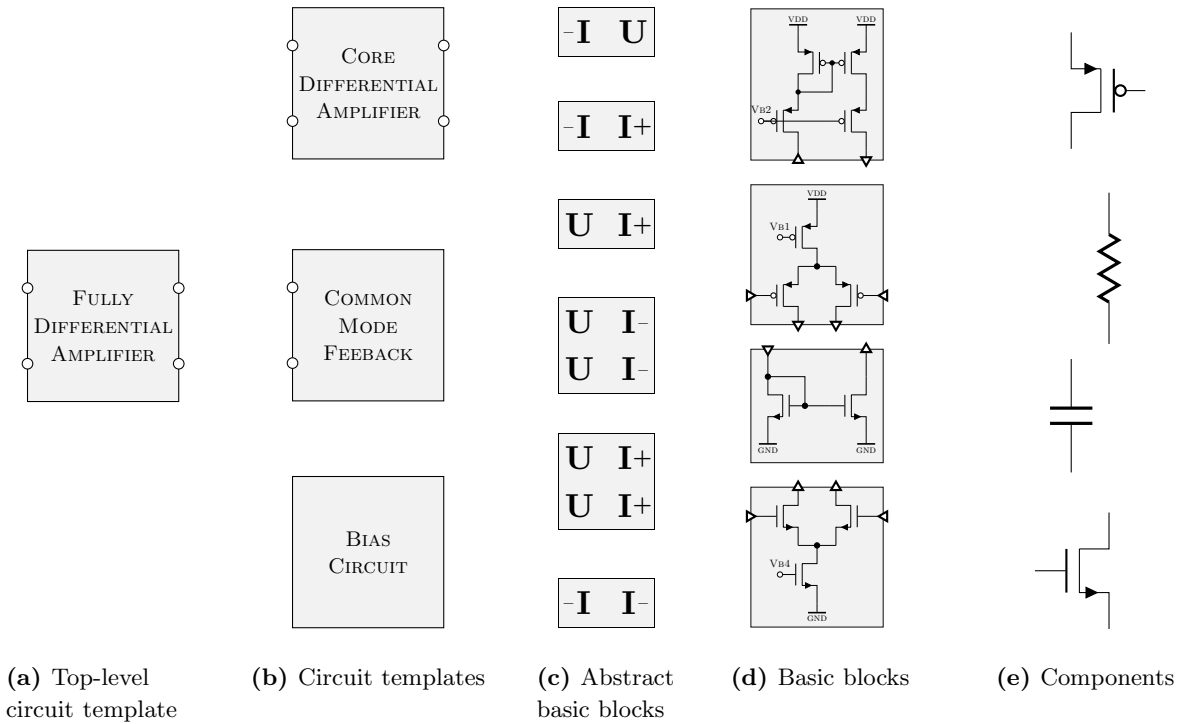
*Optimizer independence* is ultimately also essential for a synthesis framework, to state flexibility and genericity. The framework provides a construct named *application server* (see Chapter 7), which allows the execution and control of any tool, which may perform any desired circuit evaluation. This includes the *simulator independence*, as this is a part of the aforementioned. Furthermore this realizes the *good-natured scaling*, as there may be any number of application servers, which are all handled by the *task manager*. This manager (see Chapter 7) is additionally able to distribute evaluation tasks in an asynchronous manner, which enables the framework to scale very efficiently across any number of servers and/or workstations.

## 5.2 Inputs for a Circuit Synthesis

Apart from a configuration file for the synthesis process, which handles the administrative details of the synthesis and the configuration files for the various application servers, there are exactly three types of inputs. Each of these serving a specific degree of abstraction. The framework clearly divides those three types of inputs, in order to maintain a clean and consistent hierarchy. Figure 5.2 illustrates the supported levels of abstraction, which will be described in the following sections.

### 5.2.1 Components

To describe the *atomic* building elements of circuits, *components* are used inside the framework. The here presented concept of component abstraction was not yet published and represents



**Figure 5.2:** Examples for the various levels of the hierarchy used in FEATS. From left to right, each one representing a level descent. Further descriptions of each level are to be found in this chapter.

another recent consistency addition to the framework to further increase the genericity. The following enumeration lists a selection of possible components and their variants:

- Current source (independent)
- Voltage-controlled current source
- Current-controlled current source
- Voltage source (independent)
- (High-precision) resistance
- (Low-precision) capacity
- Switching nMOS/pMOS device
- Driving nMOS/pMOS device
- nMOS/pMOS device (in saturation)
- (Zener) diode
- Two/three port short

Each component may provide a variety of different properties to actually describe the component's behavior. The following attributes may be used to describe a component:

- *Unique name* (type of the component)
- Number and naming of *connections*
- *Constraints* as inequalities
- Process dependent attributes (e.g., *min/max/typical dimensions, area approximation factor, manufacturing grid*)

Components serve as the atomic building elements of each synthesized circuit, thus this leads to some implications. A component may never be divided somehow and it always resembles exactly one (physical) *part*, which may be mapped directly to a counterpart provided by the technology node. In particular, components do not exhibit any kind of behavioral description or functionality.

### 5.2.2 Basic Blocks

*Functional* building elements inside the framework are characterized through *basic blocks*, which may also be called basic functional blocks. These serve as building blocks inside a circuit, which exhibit functional properties. They exclusively contain components, which are interconnected and annotated in order to describe a specific functionality. A brief selection of some predefined—but easily expandable—basic blocks follows:

- nMOS/pMOS current mirror
- nMOS/pMOS differential stage
- Rail-to-rail differential stage
- (Cascode) current source
- nMOS/pMOS common-source stage
- Inverter
- Push-pull output stage
- (Four level) bias circuit
- nMOS/pMOS gain-booster stage

In contrast to components, basic blocks may contain additional information of functional nature. Primarily these are attributes related to the I/O characteristics and generic port attributes. Additionally, relations between the components contained inside the basic block, but also global relations which can later be used as hints for a successful sizing.

- Number of ports and their properties
  - *Input, output, local reference, or global reference*
  - Nature of the port: *current or voltage*
  - If applicable, the bias current direction: *positive or negative*
- Sizing hints
  - Local sizing variable
  - Global sizing variable
  - Fixed numerical value
  - Fixed maximum/minimum value

Especially, the distinction of the (*global*) *reference* and *local reference* port property needs some explanation. Generally any port which does not carry a signal may be handled as a reference, therefore FEATS transparently adds reference ports to the next higher hierarchy once they are needed. This feature comes very handy for smaller circuit templates, as long as there is no other instance of a single circuit template inside the same level. Assuming both contain e.g., an enable circuit, or a bias circuit, which differ in both instances of the circuit template. Once inserted into the top-level circuit template, implicit interconnections between those two instances occur, as obviously their *reference* ports' names match. This challenge can easily

be mastered by introducing *local reference* ports, which automatically rename themselves once there are identically named *local reference* ports inside the same hierarchy-level.

### 5.2.3 Circuit Templates

The highest level of abstraction is expressed through the *circuit templates*, which allow the user to construct hierarchical structures of arbitrary depth and complexity. Circuit templates may not only contain basic blocks, but moreover they may contain other circuit templates themselves. The user gets an extremely expressive construct, which may be precisely configured in order to express a smaller circuit, but also whole analog modules. The following enumeration provides a very small portion of what could possibly be described using circuit templates.

- Voltage controlled current source
- Schmitt-Trigger
- Integrator
- Differentiator
- Instrumentation amplifier
- 2-stage operational amplifier
- Differential amplifier
- Comparator
- Operational amplifier input stage
- Active filters
- Fully-differential amplifier
- Voltage-controlled oscillator
- Sigma-delta modulator
- Operational amplifier output stage
- Common-mode feedback circuit

The expressive strength of circuit templates originates inside the fine granulated configuration options, which allow the user to precisely tweak the method the circuits are actually generated. Circuit templates are the main reason why the presented synthesis framework may claim a full coverage of the design space as described in Chapter 1 and illustrated in Figure 1.1. The interface for the user is kept plain and easy to understand by providing the following attributes to be set.

- *I/O characteristics* similar to the basic blocks
- *Electrical performances* to be reached during sizing
- *Synthesis rules* to be applied (see Section 5.5)
- *Structural hierarchy*, i.e., other circuit templates, if applicable

As circuit templates may be encapsulated into arbitrary hierarchies and levels, FEATS interprets each circuit template as a (possibly embedded in a bigger) circuit on its own. This allows to place *any* circuit template inside any other without the need to make adjustments. Essentially this not only reduces the effort needed to set up a higher level synthesis—moreover the typical design flows (bottom-top, top-bottom, meet-in-the-middle) are all perfectly suited to fit into this methodology. The actual top-level circuit template is simply chosen by passing

it, by name, to FEATS. Any circuit template below this one will be synthesized on demand as demonstrated in Section 9.3.

In this context the *construction types* for circuit templates must be shortly illuminated. It is possible to include multiple instances of one circuit template inside another (higher level) circuit template. Consequently, the relation of these to each other must be clearly described. Therefore the two properties NAME and SCOPE for circuit templates are available and may be provided inside the containing circuit template. These two properties implicitly set one of the following three *construction types*:

**Unique structure, private scope** span their own scope in terms of design variables and furthermore do not share the structure with any other circuit template. This is accomplished through a unique NAME property. (This is the default, if neither NAME or SCOPE are set.)

**Shared structure, private scope** share the structure with other circuit templates—determined through identical NAME properties—but the design variables are still private, thus the same structure may exist with different sizings. The latter is enforced by omitting SCOPE, or setting it to “private”.

**Shared structure, shared scope** share the structure with circuit templates exhibiting the identical NAME and share the design variables with all other circuit templates with equal SCOPE names.

## 5.3 Preparing the Synthesis Process

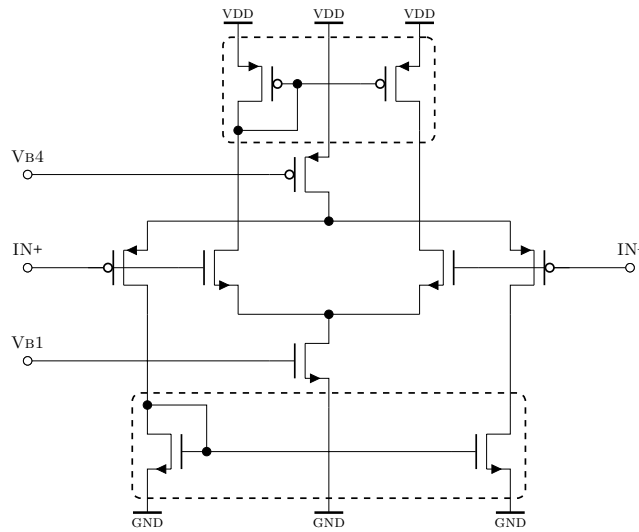
To finally start the synthesis engine in order to generate all hierarchies of circuits, the framework applies three very important preprocessing steps. First all available basic block libraries are analyzed to create so-called *complementary-symmetric basic blocks*. Afterwards the basic block libraries are once again processed to realize an important refactoring of the available basic blocks. All basic blocks are categorized into groups of *abstract basic blocks*, which serve as topological building blocks to reduce the computational effort of the synthesis process. This realizes the widely used top-down design methodology, translated to a more computational approach.

### 5.3.1 Complementary-Symmetric Basic Blocks

Recent research [EG12], [ESL<sup>+</sup>11] has shown the inevitable need of symmetry information for analog blocks in order to primarily accelerate sizing and how important a highly symmetric circuit is for all steps of an analog module design. An automated sizing may benefit from the deducted constraints, especially during the layout creation—the process variations may be significantly reduced using this information. As the framework generates all circuits, the aim is to primarily *generate symmetric circuits*, rather than analyzing them and discard the asymmetric ones. Nevertheless, the definition of symmetry in analog circuits is not always a natural, or more precisely, mathematical one.

In electrical engineering the concept of complementary components (i.e., nMOS vs. pMOS) is justifiably widely understood as symmetry. E.g., a rail-to-rail input stage—there are four

branches, two originating from an nMOS differential pair and two originating from a pMOS differential pair. An analog designer would insert a symmetric load, by using a simple current mirror for the nMOS and the pMOS branch as to be seen in Figure 5.3. While this would clearly be classified as symmetric, a formal symmetry does not apply in this case. The presented framework handles this often occurring construct by generating the aforementioned *complementary-symmetric basic blocks*. Furthermore, by exploiting the developed isomorphism algorithm (see Section 6.1) the framework may generate these, fully unattended, based on the provided basic block libraries.



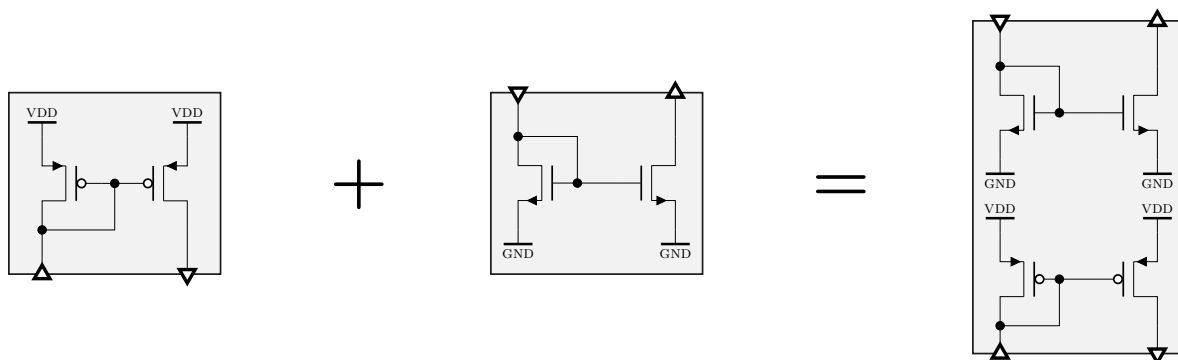
**Figure 5.3:** A rail-to-rail input stage with marked complementary loads.

The algorithm to acquire all complementary-symmetric basic blocks is very straightforward. Roughly outlined, the algorithm inspects each of the available basic blocks  $bb_i$ . After replacing all appropriate components and ports with their complementary counterparts (this leads to a basic block  $bb_i^{comp}$ ), the remaining basic blocks are now searched for isomorphic basic blocks. For each found isomorphic block  $bb_j$  a new *merged basic block* is constructed, which contains the original basic block  $bb_i$  and the complementary-symmetric basic block  $bb_j$ . An example for this process is shown in Figure 5.4.

The result of this preprocessing are several new blocks inside the inspected basic block library, which all exhibit the complementary-symmetric property. These added basic blocks enable the *symmetric elementary electric rule* (see Constructive rule 3) to construct circuits containing complementary symmetries.

### 5.3.2 Deduction of Additional Basic Block Properties

During this very straightforward step, all basic blocks undergo a simple processing step inside which the following properties are extracted for each of the basic blocks in order to annotate the basic blocks accordingly.



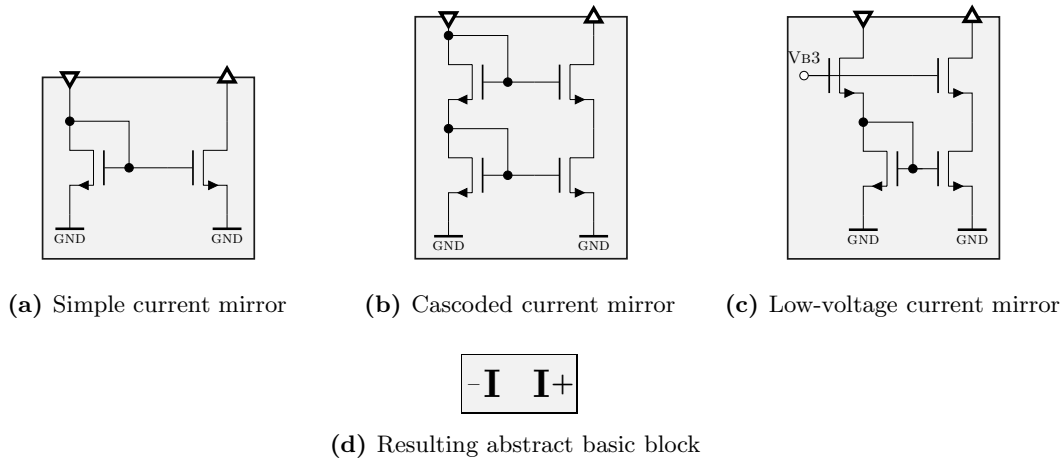
**Figure 5.4:** Two complementary current mirrors forming a new complementary basic block.

**Signal identity** is generated for basic blocks, which exhibit more than one signal path (i.e., have more than one input or output port). This leads to the annotation of the input and output ports according to their signal identity.

**Symmetric property** is set for a basic block, if it exhibits more than one signal path. This may be deliberately assumed for all basic blocks with multiple signal paths due to the fact that these blocks are by definition symmetric, if they were merged with the aforementioned complementary-symmetric method. Further, all provided basic blocks, which match the former property are also flagged as symmetric by default. This is an obvious assumption, but it may be overruled by the user for specific basic blocks, which for some reason exhibit multiple signal paths without being symmetric.

### 5.3.3 Abstract Basic Blocks

Another distinctive feature of the presented framework is the grouping of the provided basic blocks to *abstract basic blocks*. This quite simple, thus very powerful concept allows the construction of *topologies*. Topologies are in the context of this work clearly distinguished from circuits, as they represent the interconnection of abstract basic blocks instead of plain basic blocks. This semantically richer representation is a fundamental part of the framework (see Section 5.4). In order to construct topologies, all basic blocks, which exhibit identical I/O characteristics, are grouped together to abstract basic blocks. A typical example for an abstract basic block and its included basic blocks is shown in Figure 5.5. While an abstract basic block may be seen as a *black box* containing different (transistor level) implementations of a specific I/O characteristic. The side effect of this approach are groups of mostly functionally equivalent basic blocks represented by one abstract basic block. Furthermore, the reduction of the number of basic blocks leads to a reduction of the needed computational effort during the topology, respectively circuit generation process. A wide selection of the (abstract) basic blocks currently used inside the framework is to be found in Appendix B.



**Figure 5.5:** Three nMOS current mirror variants (Figure 5.5a, 5.5b and 5.5c) forming an abstract basic block shown in Figure 5.5d.

## 5.4 Circuit Synthesis Engine

The synthesis engine, or in other words the *circuit generation algorithm*, aims to provide an engine, which upfront does not make any assumptions about the circuit class or the circuit structure it should synthesize. This is absolutely mandatory and desirable due to the design targets presented in Section 5.1. Therefore a *circuit manipulation* API in the form of rules (see Section 5.5) has been developed. These allow a consistent and by the principle of locality driven, manipulation of circuits. This section explains this approach in depth by using the aforementioned *abstract basic blocks* and the definitions made in Chapter 2.

Algorithm 1 presents the actual core synthesis algorithm at a glance. Three phases can clearly be distinguished inside the algorithm. These are namely *constructive synthesis*, *topology expansion* and *circuit analysis* as also seen in Figure 5.1. The synthesis engine is executed for all found circuit templates (see Section 5.2.3) recursively. This operation is realized as a *rule*, as all operations on topologies and circuits, and is further explained in Section 5.5. The following sections describe the phases as distinct parts but the implementation—as shown in Algorithm 1—further encapsulates the latter two phases in order to reduce the memory footprint of the process, following the design targets stated in Section 5.1.

### 5.4.1 Constructive Synthesis

The constructive synthesis is *the starting point* for the synthesis engine—each hierarchical descent realized through Constructive rule 5 (see Section 5.5.1) means the instantiation of an independent synthesis engine, initialized with the circuit template and its associated basic block library. The synthesis engine behaves identical for each provided circuit template—whether it is a top-level template or any below—each one starts exactly here.



**Algorithm 1** Synthesis Engine

---

```

1: function SYNTHESISENGINE( $t_0, D^{abb}$ )
2:    $Q \leftarrow \{t_0\}$            // next topologies
3:    $T_G \leftarrow \emptyset$        // good topologies
4:    $R_c \leftarrow \text{GETACTIVERULES}(\text{constructive})$ 
5:    $R_d \leftarrow \text{GETACTIVERULES}(\text{destructive})$ 
6:   while not empty( $Q$ ) do
7:      $t_B \leftarrow \text{pop}(Q)$ 
8:     for all  $r_c \in R_c$  do // constructive step
9:        $T_{new} \leftarrow \text{apply}(r_c, t_B, D^{abb})$ 
10:      for all  $t_i \in T_{new}$  do
11:         $Q \leftarrow \{t_i\}$ 
12:        if  $\forall r_d \in R_d : \text{apply}(r_d, t_i)$  then // destructive step
13:           $T_G \leftarrow \{t_i\}$ 
14:        end if
15:      end for
16:    end for
17:  end while
18:
19:   $r_e \leftarrow \text{GETEXPANSIONRULE}()$ 
20:   $R_a \leftarrow \text{GETACTIVERULES}(\text{analyzing})$ 
21:   $C_G \leftarrow \emptyset$  // accepted circuits
22:  for all  $t \in T_G$  do
23:     $C_A \leftarrow \text{apply}(r_e, t)$  // expand circuits
24:    for all  $c \in C_A$  do // circuit analysis
25:      if  $\forall r_a \in R_a : \text{apply}(r_a, c)$  then
26:         $C_G \leftarrow c$ 
27:      end if
28:    end for
29:  end for
30:  return  $C_G$ 
31: end function

```

---

The synthesis engine starts with a circuit template  $t_0$  and its library of *abstract basic blocks*  $DD^{abb}$  to incrementally generate all possible topologies based on the activated *constructive* (see Section 5.5.1) and *destructive* (see Section 5.5.2) rules. Starting with the empty topology  $t_0$  inside the work queue  $Q$ , which only defines the I/O characteristics of the final topology. The algorithm pops the first topology out of the work queue, which is treated as *base topology*  $t_B$  during this iteration. This base topology  $t_B$  and the library  $DD^{abb}$  are now passed to each activated constructive rule, which leads to a set of generated topologies  $T_{new}$ . The latter is thus a result of the application of a single constructive rule on  $t_B$  using the provided abstract basic block library  $DD^{abb}$ . While it is possible that  $T_{new}$  is empty  $|T_{new}| = 0$  (the application of the

constructive rule in conjunction with the abstract basic block library lead to no newly generated topologies), usually  $T_{new}$  contains new topologies. These are now passed, one by one, to all activated destructive rules in order to either be classified as a *good topology*, if all destructive rules accept the topology, or be classified as a *bad topology*, if at least one destructive rule declines the topology. Good topologies  $T_G$  are kept for the following steps, while bad topologies are discarded. Furthermore *all* topologies  $t_i \in T_{new}$  are inserted into the work queue  $Q$ , following the assumption that each generated topology may be a base topology for further possibly good topologies. This explains the incremental nature of the algorithm, as each topology is build step by step and may be the base topology for the next, bigger topology. This approach would, without additional constraints, lead to an infinite number of topologies—Destructive rule 1 is the responsible rule to restrict the growth here.

The principle of locality is clearly visible through the fact that a specific constructive rule operates strictly on a single base topology  $t_B$  using the provided abstract basic block library  $DD^{abb}$  to construct a new set of topologies  $T_{new}$ , by applying the rule once. Looking at the construction history of a specific good topology  $t_j \in T_G$  one will observe the application of different constructive rules, each one adding at least one abstract basic block.

### 5.4.2 Topology Expansion

The previous step generated a set of good topologies  $T_G$ , which all consist of abstract basic blocks. As described in Section 5.3.3, each abstract basic block resembles possibly multiple different basic blocks. Thus the topology itself is a semantically richer representation of a whole set of different circuits. So, to further advance in the process of circuit generation, the generated topologies have to be expanded to their circuit representations. This expansion may take place based on different methods, which add an additional level of symmetry and are further explained in Section 5.5.3. This phase leads to a set of (transistor-level) circuits  $C_A$ , which usually consist by far more items as the originating set of topologies  $T_G$ :

$$|C_A| \gg |T_G| \tag{5.1}$$

In contrast to the constructive synthesis or circuit analysis, this synthesis phase is carried out by exactly one expansion method, as the most naive expansion method leads to the maximum count of circuits and the other methods thus resemble subsets of this naive expansion.

### 5.4.3 Circuit Analysis

At this point the synthesis engine generated a vast amount of circuits (see Chapter 9 for detailed numbers). Evaluating, or better sizing, all these circuits would lead to enormous synthesis run times. Thus it is quite obvious that most of the circuits inside  $C_A$  should not be fully evaluated for various reasons. The following enumeration explains the most obvious of these:

**Duplicates** Due to the incremental and explorative nature of the synthesis algorithm, paired with the possibility to introduce own basic blocks, the process will inevitably lead to duplicated circuits as demonstrated in Figure 6.1.

**Inverted phase** In case of a signal manipulating circuit, the synthesis engine will nearly always generate a circuit, which exhibits the correct (positive) phase at the output port(s) but also the circuit, which inverts the phase at the output.

**Electrical constraints** The explorative approach will generate circuits, which do not fulfill even the most basic electrical constraints. For instance this would be too many stacked MOS-devices, which obviously is process dependent.

So, this phase should apply circuit analyzing methods, which are tailored to handle these vast amounts of circuits, to finally reduce this initially big set of circuits  $C_A$  down to a reasonable set of circuits  $C_G$ , which may afterwards be passed to the full evaluation respectively sizing step. The therefore developed algorithms are operating on the circuits through analyzing rules (see Section 5.5.4) and resemble one of the biggest challenges for a modern analog synthesis framework as stated previously in Chapter 1.

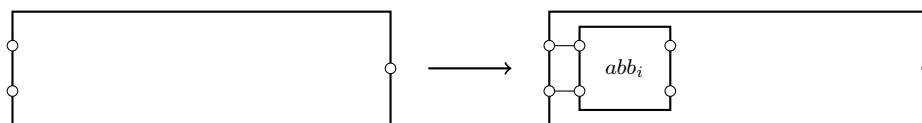
## 5.5 Rules – Circuit Manipulation API

Rules inside the presented framework represent one of the key principles in software design: *a well-defined interface for operations on data*. The data in this context is represented by the topologies, respectively circuits to be manipulated and in particular for the latter, to be analyzed. Furthermore, driven by the principle of locality the designated goal is to allow the development of operations on topologies and circuits, which are widely independent from the other parts of the framework. The previous section has shown how and in which order the rules are applied during the synthesis. This section focuses on the rules themselves, which carry out the actual operations.

### 5.5.1 Constructive Rules

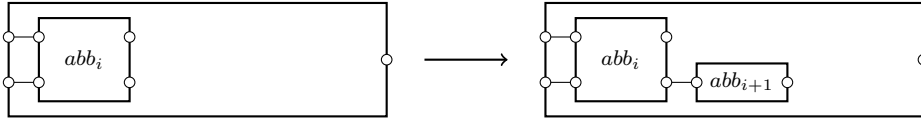
The concept of constructive rules originates in the first publications of this methodology in [WH06, MMH11c]. These were fundamentally reformulated towards genericity during the migration to the graph based approach first presented in [MMH11b, MMH11a] and are in its current evolution fully independent from specific basic blocks, thus allow the constructive synthesis to use any basic block provided by the user as published in [MH15].

**Constructive Rule 1** (InitialBlockRule (IBR)). *Works on empty topologies exclusively. Insert a single block from the given abstract basic block library, which matches the input specifications of the topology. Matching: Two ports match if their properties are identical.*



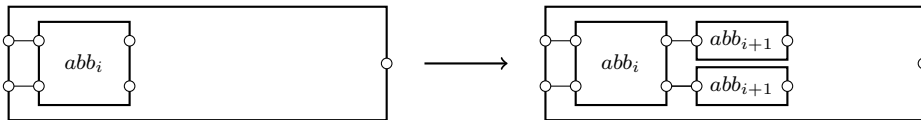
**Figure 5.6:** Initial block rule (IBR)

**Constructive Rule 2** (Elementary Electrics Rule (EER)). *Works on any topology with open inner ports. Insert a single block from the abstract basic block library, which matches any number of the open inner ports. Matching: Two ports match if they both are either current or voltage ports. Additionally if they are current ports the bias current has to match, i.e., having the same direction.*



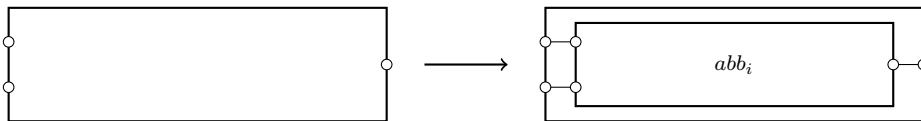
**Figure 5.7:** Elementary electric rule (EER)

**Constructive Rule 3** (Symmetric Elementary Electrics Rule (SYM EER)). *Realizes topologies being symmetric by construction. Instead of placing only a single block inside the topology, the symmetric EER tries to place multiple identical blocks inside the topology if the topology is in symmetric state and there are enough open inner ports. Matching: Same as EER*



**Figure 5.8:** Symmetric elementary electric rule (SYM ERR)

**Constructive Rule 4** (Library Rule (LIB)). *Works on empty topologies exclusively. Each block inside the abstract block library is placed once inside the topology without any matching. Matching: There is no matching*



**Figure 5.9:** Library rule (LIB)

The following constructive rule realizes the *structural hierarchy* as stated in Section 5.2.3. The previous constructive rules may all be combined inside one circuit template—in contrast to the following *circuit template rule*, which may only be used exclusively.

**Constructive Rule 5** (Circuit Template Rule (CTR)). *Realizes the hierarchical descent into the next (lower) level. This means the execution of the full synthesis engine for the current block. Once the synthesis engine has finished, the resulting circuits are handled as basic blocks (variants) for the current abstract basic block.*

### 5.5.2 Destructive Rules

Destructive rules are during the constructive synthesis (see Section 5.4.1) the counterpart for the constructive rules shown in the last section. In particular without the destructive rules the topology count would rise indefinitely, thus each destructive rule realizes a concept to restrict this growth.

**Destructive Rule 1** (Block Length Rule (BLR)). *Restricts the maximum block count inside a topology. Takes one argument: The block threshold  $t_B$ . May be set to either allow only topologies with an exact block count of  $t_B$  or less or equal blocks than  $b_T$ .*

**Destructive Rule 2** (Output Match Rule (OMR)). *Discards any topology, whose inner open ports do not match the output specifications of the topology. Matching: Two ports match if their properties are identical*

**Destructive Rule 3** (No Block Twice Rule (NB2R)). *Discards any topology, which contains two identical blocks connected directly to each other.*

**Destructive Rule 4** (V2V Only At End Rule (VVOER)). *Scans the topology to find a block with the input and output ports having the voltage property (v2v block). If there is any v2v block, which is not the last block inside the topology, the topology is discarded.*

**Destructive Rule 5** (Topology Isomorphism Rule (TIR)). *Verifies that the topology is unique in terms of structure compared to the already accepted topologies. If the topology is isomorphic to one of the accepted topologies it is discarded.*

### 5.5.3 Topology Expansion Rules

The expansion has a major influence on the amount of generated circuits. The previously generated topologies are expanded to their circuits, by replacing all abstract basic blocks with their basic blocks containing actual transistor-level circuitry. The following rules describe how the topologies  $T$ , consisting of multiple abstract basic blocks  $abb_i$ , delivered by the constructive and destructive rules, are expanded to their respective circuits  $C$  containing exclusively basic blocks  $bb_i$ . In the following the number of basic blocks resembling an abstract basic block is denoted as  $|abb_i|$ .

**Expansion Rule 1** (Asymmetric Expansion Rule). *Resembles the straightforward approach for expansion. Each abstract basic block inside the topology is expanded to all its basic block representations.*

$$|C| = \prod_{abb_i \in T} |abb_i|$$

**Expansion Rule 2** (Symmetric Expansion Rule). *Expands the topology with the condition that equal abstract basic blocks, get equal basic block representations exclusively.*

$$|C_{Sym}| = \prod_{abb_i \in T} z$$

$$z := \begin{cases} |abb_i| & \text{first occurrence in topology} \\ 1 & \text{else} \end{cases}$$

#### 5.5.4 Analyzing Rules

Finally, the last steps inside the synthesis engine to get circuits for sizing, is called the circuit analysis step. Analyzing rules may apply any operation on the currently analyzed circuit to determine whether the circuit should be passed to the next rule or simply discard it. If all rules are passed successfully by one circuit, it is called *accepted*.

**Analyzing Rule 1** (Extract Properties Rule (EXPROPS)). *Discards no circuits, but calculates various computational cheap properties of the circuit, which may be used for later analysis.*

**Analyzing Rule 2** (Isomorphism Rule (ISO)). *This rule determines the uniqueness of the circuit, by comparing it to all other already accepted circuits to calculate isomorphism. The therefore used algorithm(s) are further explained in Section 6.1.*

**Analyzing Rule 3** (Pre Symbolic Rule (SYM PRE)). *Similar to the extract properties rule this rule prepares the circuit for the following semi-symbolic analyzing rules. In particular this means setting up the testbench for the circuit and creating necessary data structures.*

**Analyzing Rule 4** (Symbolic Gain Rule (SYM GAIN)). *This rule checks for an inversed gain between the input(s) and the output(s) of a circuit and is further illuminated in Section 6.2.2.1.*

**Analyzing Rule 5** (Symbolic Feasibility Rule (SYM FEAS)). *This rule checks for the feasibility in terms of electrical constraints and despite this approximates the output voltage range, if applicable. This process is described in detail in Section 6.2.1.*

## FAST ANALYZING TECHNIQUES FOR HUGE CIRCUIT AMOUNTS

As motivated in Chapter 1, the analyzing methods used to early distinguish the feasible from the infeasible circuits is one of the most important parts of a modern circuit synthesis framework.

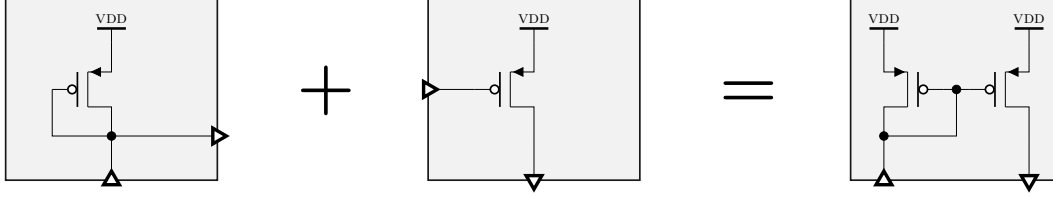
Generally spoken, the main goal must be to drastically reduce the number of generated circuits down to an amount, which can reasonably be passed on to the most expensive (in terms of wall clock time and monetary expenses) evaluation—the SPICE accurate sizing. The following sections describe the developed methodologies to allow a blazing fast selection with great scalability, but with a marginal amount of *false positives* i.e., circuits which are discarded and would have reached the specifications during sizing.

The illustrated methods inside this chapter are applied inside the synthesis framework through analyzing rules, which technically resemble the interface for circuit manipulation and analysis inside the framework. Although the described methods are widely known and applied in smaller scale circuit analyzers—the here introduced methods focus on the demands of a modern analog circuit synthesis environment, which fundamentally differs from the regular applications for circuit analysis. The most crucial of these: the vast amount of circuits, which must be handled. Handling maybe thousands of circuits—even without the analyzing overhead—by themselves sets very high requirements for the software and its resource management e.g., main memory or CPU-time. Despite this, a circuit synthesis must not occupy a computing cluster, moreover it should be usable on a decent desktop or workstation.

### 6.1 Circuit Isomorphism

A basic block based circuit synthesis inevitably leads to the generation of multiple identical circuits. The basic blocks may contain as many components (and other basic blocks) as the user wants to, thus there are constellations in which a single basic block may be constructed from two or more other basic blocks. Hence two (structural) identical circuits may be constructed in different ways. A simple example for this is illustrated in Figure 6.1. Additionally, to maintain the promised flexibility it is self-evident to allow the user of the synthesis framework to introduce

own, new basic blocks, which obviously further enforces the need of a circuit isomorphism to reduce the number of generated circuits to the truly unique ones.



**Figure 6.1:** An example how the building block concept inherently may lead to isomorphic circuits—the construction of a (existing) basic block by using two smaller basic blocks.

In a real usage scenario, the reduction of the generated circuits, down to the unique ones is easily translated into *saving time and money*. More generated circuits means more circuits to be evaluated (sized), which leads to a higher consumption of simulator and sizing resources. This obviously directly correlates to not only higher synthesis times, but also to higher costs.

This section provides a detailed overview over the first of the two presented fast analyzing techniques for huge circuit amounts. Starting with a brief sketch of the graph isomorphism in general, the circuit isomorphism is differentiated from the seemingly more generic problem. After formalizing the properties of a circuit isomorphism and analyzing its applicability, the developed circuit isomorphism algorithm and its various modifications towards handling vast amounts of circuits are presented. A fully annotated example of the labeling based isomorphism described in Section 6.1.4.2 can be found in [MH15].

### 6.1.1 Graph Isomorphism

The *graph isomorphism* problem is one of the few problems, which is known to be in  $NP$ , but neither it was successfully classified to be an  $NP$ -hard problem nor it has been shown whether it is in  $P$  [GJ79]. Although the generalized problem of *subgraph isomorphism* is classified as  $NP$ -hard, very efficient graph isomorphism algorithms have been developed and analyzed in the past years [Pre09]. Various variants of the graph isomorphism problem are polynomially classified as equivalent inside the  $GI$  complexity class and are accordingly  $GI$ -hard [BSC77]. This seems a reasonable classification, as the existence of a polynomial time algorithm would directly proof  $P = NP$ .

Besides this, the algorithm has a further challenge to overcome. As the synthesis engine generates a vast amount of circuits, each newly generated circuit has to be checked against all other, already as unique identified, circuits. This leads to  $\mathcal{O}(|C|^2)$  calls of the algorithm, with  $|C|$  being the number of generated circuits. This multiplies the already challenging problem.

**Definition 6.1** (Isomorphism Property). *A graph  $G_1 = (V_1, E_1)$  is called isomorphic to another graph  $G_2 = (V_2, E_2)$ , if a bijection  $\phi : V_1 \rightarrow V_2$  exists:*

$$\forall u, v \in V_1 : \{u, v\} \in E_1 \Leftrightarrow \{\phi(u), \phi(v)\} \in E_2 \quad (6.1)$$



**Definition 6.2** (Notation). *Two graphs  $G_1 = (V_1, E_1)$ ,  $G_2 = (V_2, E_2)$  are called isomorphic or structurally equivalent, if both fulfill Equation 6.1. This will be written as:*

$$G_1 \simeq G_2 \quad (6.2)$$

In the context of this work, if not stated otherwise, only *undirected, bipartite* graphs  $G = (V, E)$  - as defined in Chapter 2 - are used, containing vertices  $v \in V$  and edges  $e \in E$ , which have an arbitrary number of associated *invariant properties*.

**Definition 6.3** (Invariant property). *An invariant  $IV^i \in \mathbb{IV}$  contains a finite number of easily computable (invariant) properties  $p_j^i \in IV^i$ , which may efficiently be compared to verify equivalence, but there is no strict ordering relation. The following notations will be used:*

$$IV^i \in \mathbb{IV} \quad (\text{an invariant}) \quad (6.3)$$

$$p_j^i \in IV^i \quad (\text{an invariant property}) \quad (6.4)$$

$$IV^i = \{p_1^i, p_2^i, \dots, p_n^i\} \quad (\text{all properties of a single invariant}) \quad (6.5)$$

The following three definitions formally describe the relation between the graph, its vertices and the invariant properties as defined in Definition 6.3. The reader is encouraged to use the annotated example shown in Figure 6.2 and despite this the Table 6.1 to ease the understanding of the introduced terms and their implications.

**Definition 6.4** (Object Invariant). *Any object  $x \in \{V \cup E\}$  inside a graph  $G = (V, E)$ , may exhibit any number of object invariants  $IV^i \in IV(x)$ , while each  $p_j^i = IV^i(x)$  means this object  $x$  exhibits exactly this property  $p_j^i$ . Each object  $x$  may strictly only exhibit exactly one single property  $p_j^i$  for each invariant  $IV^i$ . The following notations will be used:*

$$IV(x) = \{IV^1, IV^2, \dots, IV^m\} \quad (\text{all invariants associated with object } x) \quad (6.6)$$

$$IV^i(x) = p_j^i \quad (\text{property for } IV^i \text{ associated with object } x) \quad (6.7)$$

**Definition 6.5** (Equivalence Class). *Let  $eqcls_i \in EQCLS$  be a equivalence class for a graph  $G = (V, E)$ . All vertices  $v \in V$  are classified into equivalence classes  $eqcls_i \in EQCLS$ . Furthermore,  $|eqcls_i|$  will be denoted as the number of vertices inside the equivalence class with each vertex being associated to exactly one equivalence class.*

$$eqcls_i = \{v_1, v_2, \dots, v_n\}, \text{ with} \quad (6.8)$$

$$\cup_i eqcls_i = V \quad (6.9)$$

$$\forall i \neq j : eqcls_i \cap eqcls_j = \emptyset \quad (6.10)$$

**Definition 6.6** (Equivalence Class Properties). *A set of equivalence class properties  $EQPROPS$  is associated with all combinations of the available invariant properties and is unique for each  $eqcls$ . Therefore  $p_j \in EQPROPS(eqcls)$  denotes the unique combination of*

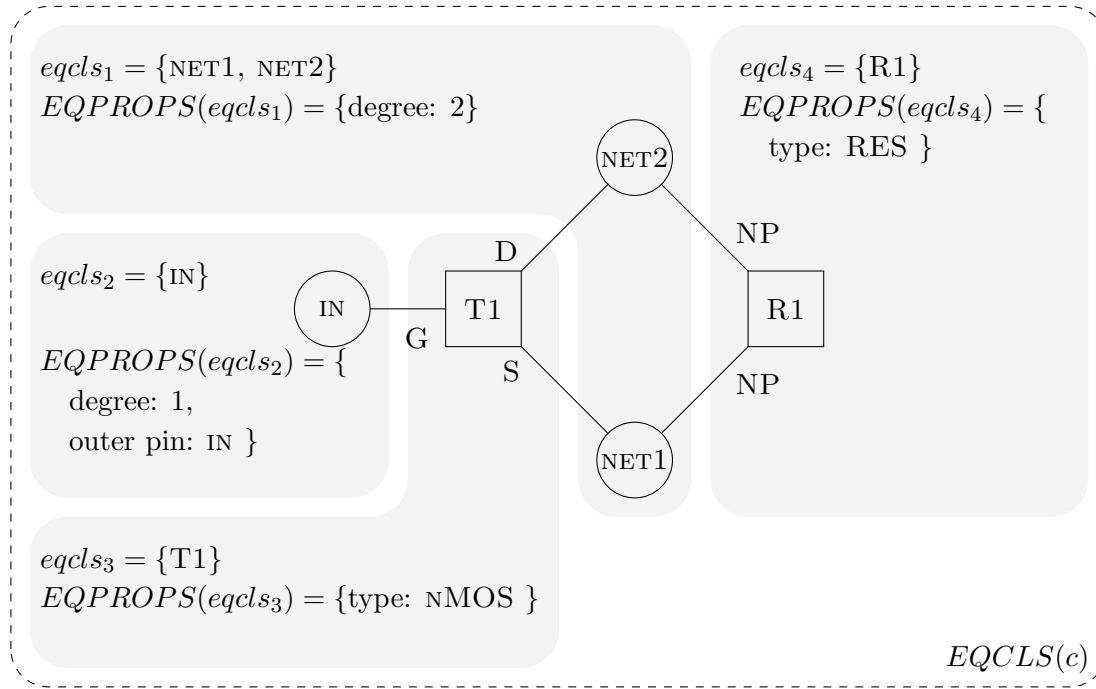
invariant properties, which characterize this equivalence class. The following notations will be used:

$$EQPROPS(eqcls) := \{p_i^1, p_j^2, \dots, p_k^m\} \quad (6.11)$$

$$\text{with } p_k^h \in IV^h \quad (6.12)$$

Furthermore, each vertex inside an equivalence class carries the same properties:

$$\forall u \in eqcls \ EQPROPS(u) := EQPROPS(eqcls) \quad (6.13)$$



**Figure 6.2:** A fully annotated example graph (taken from Chapter 2) using Definition 6.3 to 6.6 and 6.7 applied to a circuit. The invariant properties used here are listed in Table 6.1. Note that the edge invariants are only implicitly handled—by being included into the type at this point.

Already for small amounts of vertices, the mapping  $\phi$  is even for humans not trivial to find. Even small graphs may exhibit a non-trivial isomorphism relation to a structurally equivalent graph. Fortunately, circuit isomorphism compared with graph isomorphism allows to reduce the computational complexity due to various restrictions that apply for circuits represented as graphs.

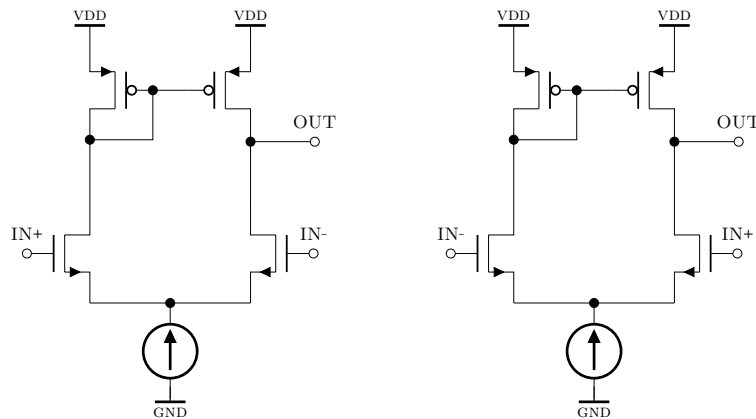
### 6.1.2 From Graph to Circuit Isomorphism

The graph definition is far to generic for a representation of a circuit. As defined in Chapter 2 the fact that the graphs used for circuit representation are *strictly bipartite* might lead to an

reduction of the computational complexity through the reduced number of candidates for a single vertex to be isomorphically mapped to. However, the theoretical computational complexity still remains inside *GI*-hard for a wide variety of graph classes, including the *bipartite* and *labeled* graphs [ZKT85]. This means that in application, there might be cases in which a circuit isomorphism algorithm might perform only in exponential time complexity. Despite the fact that the complexity itself may not be reduced, there *are* several inherent properties, which may be exploited to increase the efficiency of the algorithm.

The most computing resources preserving property of a circuit is its small average vertex degree, compared to the number of vertices. A circuit graph may, mostly be not classified as a planar graph (i.e., no crossing edges), but it is far away from being complete (i.e., each vertex has edges to all other vertices). In simple terms, the fact that many circuits may be easily drawn on a sheet of paper with just a minimal number of drawn wire-crossings, illustrates the claimed.

Furthermore, several vertex invariants, introduced in Definition 6.3, provide very valuable information for the isomorphism algorithm. This partly originates in the explorative analog synthesis methodology itself, as the synthesis engine, in its effort to generate all possible circuits, also may generate structurally identical circuits, which only differ by their pin labellings. The most common case are the simply swapped input pins, which is illustrated in Figure 6.3.



**Figure 6.3:** Structurally equal circuits with swapped pins, which should not be classified as isomorphic in the context of analog synthesis.

Another very useful circuit property in comparison to generic graphs, is the fact that all edges are named—due to the association of the vertices with actual components. These electrical components mostly exhibit different types of connections, i.e., edges. Some examples are presented in Table 6.1.

Especially, this is not only helpful for the isomorphism, but moreover this is a necessary property to be taken into account, even if two circuits are structurally equivalent, there may be a pin swapped component (e.g., a MOS-device). This would obviously lead to different behaviors of the both circuits, so both circuits should be classified as not isomorphic. Although, there may be components (e.g., resistors) which behave equal, regardless of how they are connected. The algorithm handles this by simply allowing components to have equally named pins, leading to

equally named edges. The very same applies for the vertices. While the net vertices are labeled, their label does not count as an invariant property as changing them would neither change the circuit structure nor any other (electrical) property. The exception are the nets, which describe (circuit) pins, their labels are actually invariant properties as to be seen in Table 6.1. On the other hand the component vertices, as they resemble a physical component, exhibit their *type* as an invariant property, which clearly can be used to distinguish one vertex from another.

The vertex invariants shown in Table 6.1 are used to accelerate and support the isomorphism algorithm. Typically a vertex invariant can be calculated in a negligible amount of time and therefore does have nearly no impact on the runtime of the algorithm.

TABLE 6.1  
THE USED INVARIANTS  $IV^i$  AND A SELECTION OF THEIR PROPERTIES  $p \in IV^i$  FOR THE AVAILABLE OBJECT TYPES.

OBJECT TYPE	INVARIANT ( $IV^i$ )	TYPICAL PROPERTIES ( $p$ )
Component vertex	Type	NMOS, PMOS, CAP, RES, ...
Edge	Connections	D, G, S, P, N, NP, ...
Net vertex	Degree	$p \in [0 \cdots \infty]$
	Representing a pin	IN <sub>+</sub> , IN <sub>-</sub> , VDD, VBIAS <sub>1</sub> , OUT <sub>+</sub> , OUT <sub>-</sub> , GND, ...

One is very much inclined to agree with the efficiency of the algorithm comparing two circuits against each other. Unfortunately, the challenging part is to actually compute a *set of unique circuits* out of a set of unknown circuits. Granted, the isomorphism algorithm itself will most likely perform better as expected, nevertheless the huge number of circuits would render it useless, if used in the obvious pairwise comparison approach - leading to  $\mathcal{O}(|C|^2)$  comparisons.

Let  $c_1$  be a circuit, which has just been expanded and is now ready for the isomorphism analysis. The synthesis engine maintains the set of unique  $C_G$  circuits and grants access to it—the isomorphism algorithm uses it to verify circuit  $c_1$  is either isomorphic to any  $c_i \in C_G$  or not, thus being added to  $C_G$ .

To overcome this drawback, the realization of the circuit isomorphism algorithm is divided into two phases, whilst the first phase is designed to reduce the number of circuits to be compared against. Therefore the first phase of the isomorphism searches a  $C_C \subseteq C_G$  with  $|C_C| \ll |C_G|$ . Once this reduction of potential isomorphism candidates is done, (one of) the actual isomorphism algorithms must be applied for all remaining pairs:  $(c_1, c_i) \forall c_i \in C_C$ .

### 6.1.3 Filtering Phase

The first phase of the algorithm has the important task to preprocess all circuits  $C_G$  in order to minimize and categorize the number of circuits. The algorithm solves this challenge by calculating several trivial properties of the circuit and additionally categorizing each of the vertices (i.e., components and devices) into their appropriate *equivalence classes*. The latter is done according

to Definition 6.7 (see Figure 6.2 for an example), which from now on is an important distinction property for this circuit and is also used by the labeling-based isomorphism.

**Definition 6.7** (A circuit's equivalence classes). *Let  $EQCLS(c)$  be the equivalence class associated with the circuit  $c$  represented as a graph  $G = (V, E)$ . Applying the Definition 6.4 and 6.6 on the circuit, there is an finite, unique and deterministic computable set of equivalence classes,*

$$EQCLS(c) := \{eqcls_1, eqcls_2, \dots, eqcls_n\} \quad (6.14)$$

*each one containing all vertices, whose invariant properties are equivalent. The equivalence classes are all pairwise disjoint*

$$\forall eqcls_i, eqcls_j \in EQCLS(c) : i \neq j \rightarrow eqcls_i \cap eqcls_j = \emptyset \quad (6.15)$$

*and each one contains at least one vertex:*

$$\forall eqcls_i \in EQCLS(c) : |eqcls_i| \geq 1 \quad (6.16)$$

Until now the algorithm has just visited each vertex and edge once to gather the information. Eventually, the algorithm has to compute a so-called *circuit hash*, which serves as the most important figure to possibly distinguish one circuit from another.

Hashing is a well known concept in computer science to reduce any data to a remarkably smaller representation in order to compare (i.e., show equivalence) or even verify the original data. Recent work has also shown that the class of non-encrypting hashing functions not only behave comparable good in terms of collisions compared to complex ones, but additionally deliver a decent performance due to the mostly simple calculations [ESRI14]. Exactly those are the two most important properties for a preprocessing step, as a slow preprocessing could easily eat up the potentially saved CPU-time.

Thus, following this reasoning an appropriate hashing function  $h(c)$  has been chosen to tackle the problem:

**Definition 6.8** (Circuit Hashing).

$$h(c) = \prod_{EQCLS(c)}^{eqcls_i} L(eqcls_i)^{|eqcls_i|} \pmod{(2^{32} - 5)} \quad (6.17)$$

The hashing consists of a multiplication of all equivalence classes' labels  $L(x)$  each one to the power of the number of vertices inside the current equivalence class, modulo  $2^{32} - 5$  enabling the simple and fast storage of the hash. The therefore used labeling function  $L(x)$  is defined as follows:

**Definition 6.9** (Equivalence Class Label).

$$L : \mathcal{P} \left( \bigcup_i IV^i \right) \rightarrow [1, 2, 3, \dots, 2^{32}] \quad (6.18)$$

$$\forall p, q \in \mathcal{P} \left( \bigcup_i IV^i \right) : p \neq q \rightarrow L(p) \neq L(q) \quad (6.19)$$

**Definition 6.10** (Vertices' Equivalence Class Label).

$$u \in \mathcal{P}(V) : L(u) = L\left(\bigcup_{v \in u} EQPROPS(v)\right) \quad (6.20)$$

The defined labeling function, delivers a (bijective and unique) integer representation of a specific combination of invariant properties, as defined in Definition 6.6. In order to reduce collisions, relevant literature [ESRI14, Pre09] proposes that  $L$  may be tailored to uniformly distribute the property labels into the codomain  $[1, 2, 3, \dots, 2^{32}]$ . Experiments have shown that this approach delivers very good and reliable results, even if  $L$  is badly chosen.

The underlying math is easily explained: The hashing function  $h$  in combination with the modulo  $2^{32} - 5$  operation forms a so-called *prime modulo multiplication group*, which exhibits the important property of being free of zero-divisors. Using  $2^{32} - 5$ , known to be the largest unsigned 32bit prime number, for the modulo operation leads to a *cyclic group*, which ensures these properties. In simple terms, this property allows the labeling function to return any integer, the multiplication will always lead to a non-zero hash. A hash equal to zero is obviously undesirable, as any further multiplication would not modify the hash anymore.

Due to the definitions for vertex invariants and their properties (see Definition 6.4 to 6.7), including the hashing function from Definition 6.8, it is easy to see that the equivalence of two hashes is for sure a necessary but not sufficient property for the isomorphism between the two circuits represented through their hashes. In the scope of this work this will be referred as the *hashing property*. Using this hash, a smart chosen data-structure (i.e., unordered, associative, lookup complexity  $\mathcal{O}(1)$ ) for the underlying implementation [ISO12] allows an extremely fast filtering of vast amounts of circuits.

Once the first phase of Algorithm 2 has finished its filtering phase (see line 3), a strongly reduced number of circuits  $c_i \in C_C : |C_C| \ll |C_G|$ , which will be denoted as *isomorphic candidates*, is passed on to the next and final phase (line 11), i.e., either the backtracking based isomorphism presented in Section 6.1.4.1 or the labeling based isomorphism presented in Section 6.1.4.2, to determine whether one of the isomorphic candidates has an isomorphism mapping. In line 7 and 8 the starting sets, as defined in Definition 6.11, are generated. To disprove isomorphism for two circuits it is necessary to execute the core isomorphism algorithm (see Section 6.1.4) for each pair  $(v_1, u_i) \forall u_i \in s_{init}^{new}$ . Both starting sets have—due to the previous filtering—the same amount of members, leading to a maximum of  $|s_{init}^{new}|$  iterations, i.e., calls to the core isomorphism algorithm. Note, that at this point the vast majority of the circuits were already excluded, thus verified as not being isomorphic due to the hashing property. Eventually, all truly unique circuits are to be found inside  $C_U$ .

#### 6.1.4 Core Circuit Isomorphism Algorithm(s)

In this section the focus shifts towards the actual isomorphism algorithm processing two circuits  $c_1, c_2$ . The algorithm should—in a reasonable time—determine whether the circuits are isomorphic. Formally, in the context of circuit isomorphism, the distinctions between *homomorphisms* (a one-to-many mapping from one graph to another) and *automorphisms* (a stronger formula-

**Algorithm 2** Top-level Find Uniques Algorithm

---

```

1: function TOPLEVELISOMORPHISM( $C_G$ )
2:    $C_U \leftarrow \emptyset$ 
3:    $C_C \leftarrow \text{FILTERINGPHASE}(C_G)$ 
4:   for all  $c_{new} \in C_C$  do
5:      $iso \leftarrow false$ 
6:     for all  $c_i \in C_U$  do
7:        $s_{init}^{new} \leftarrow \text{STARTINGSET}(c_{new})$ 
8:        $s_{init}^i \leftarrow \text{STARTINGSET}(c_i)$ 
9:        $v_1 \leftarrow pop(s_{init}^{new})$ 
10:      for all  $u_i \in s_{init}^i$  do
11:         $iso \leftarrow iso$  AND ISOMORPHISMLABELING( $c_{new}, c_i, v_1, u_i$ )
12:      end for
13:    end for
14:    if  $iso == false$  then
15:       $C_U \leftarrow c_{new}$ 
16:    end if
17:  end for
18: end function

```

---

tion of isomorphism to an identical graph) deliver no additional help or implications, thus they are deliberately omitted during this analysis.

A very important decision for most graph isomorphism algorithms is the set of vertices  $s_{init}^c$  from which the algorithm starts to explore the circuit  $c$  represented as graph. The number of vertices inside this starting set is significant for the worst-case runtime of the algorithm. Let  $v_1 \in s_{init}^{c_2}$  be the first of these vertices from the starting set of circuit  $c_2$ . To find the corresponding, isomorphic vertex inside the other circuit  $c_1$ , the algorithm has to be applied for each of the vertices  $u_i \in s_{init}^{c_2}$ , as the isomorphic vertex is obviously not known upfront. Relevant literature proposes the *most unique* vertex to be chosen, which usually leads to choosing the vertex with the rarest degree in both graphs. The following algorithms start with the very same strategy, expect that they extensively make use of  $EQCLS(c_1)$  respectively  $EQCLS(c_2)$ , which were generated during the *filtering phase* as defined in Definition 6.7.

**Definition 6.11** (Starting Set). *Let  $c$  be a circuit with its corresponding graph representation  $G = (V, E)$ . After Definition 6.7 was applied, there is a set  $EQCLS(c)$  containing all equivalence classes. The starting set  $s_{init}^c$  is defined as:*

$$s_{init}^c := eqcls_i : |eqcls_i| \leq |eqcls_j| \quad \forall eqcls_i, eqcls_j \in EQCLS(c) \quad (6.21)$$

*If two or more  $eqcls_i$  contain an equal number of members, the one with the smallest label  $L(eqcls_i)$  using Definition 6.9 is chosen.*

Two different algorithms were developed and evaluated to be presented in this work. The first one is a classic, easy to understand, backtracking based algorithm (see Section 6.1.4.1),

this class of algorithms is widely used as a reference in graph analysis. It primarily delivers a reference as it will always find an isomorphism, if there is one—including the drawback of not being very fast. Especially high degrees of regularity (i.e., a large starting set) and symmetry may draw the runtime complexity towards an exponential behavior. The other proposed algorithm (see Section 6.1.4.2) is inspired by [Ebe88, OE93] and was first published in [MMLH12]. It is motivated by the second big group of isomorphism algorithms, which search for a *canonical representation* of the graph. *Motivated*, is an important term here, as a true canonical representation as described in e.g., [HZCH14, She14] would either exhibit an exponential behavior or force a specific ordering (for starting sets  $|s_{init}| > 1$ ) as the algorithm would have to decide at which vertex to start.

The latter, labeling based algorithm, massively reduces the computational effort to verify isomorphism on two graphs, as it may discard non isomorphic circuits earlier than a backtracking based algorithm by being heuristic in its nature. Its approach is a hashing like idea based on the refinement of the initial  $EQCLS(c_1)$ . In application this translates to the possibility to generate *false positives*, i.e., classify two not isomorphic circuits as isomorphic. This is introduced by the fact that there may be collisions during the refinement, as the target codomain—used to *label* vertices into—is not infinitely large.

#### 6.1.4.1 Backtracking-based Isomorphism Algorithm

It is assumed that two circuits  $c_1$  and  $c_2$  should be tested for isomorphism and both have passed the filtering phase, which means both have an equal set of equivalence classes as to be seen in Algorithm 2.

In simple terms, the backtracking algorithm tries to walk through both graphs in parallel. As the set of equivalence classes is equal  $EQCLS(c_1) = EQCLS(c_2)$ , the minimal starting set  $s_{new}^{c_1}$  contains the same number of vertices as  $s_{new}^{c_2}$ . Therefore the algorithm starts at  $v_1^{c_1} \in s_{new}^{c_1}$  respectively at  $v_2^{c_2} \in s_{new}^{c_2}$  (see Algorithm 2 line 7 – 12 and Figure 6.4). As usual for a backtracking based algorithm the general idea is to walk a legal path through the graph until either all vertices and edges are visited, which leads to the proof of isomorphism, or all legal paths were inspected, but none of them leads to the case were all vertices and edges were successfully visited, thus the circuits are not isomorphic.

First, the types of the vertices are compared, those have to be equal (see Equation 6.22) or the routine will return a mismatch. Afterwards the routine starts to inspect the possible edges to be used in order to reach the next vertex. This happens for both circuits by setting up a set of possible paths by simply collecting all connected edges  $E(v^{c_1})$  and  $E(v^{c_2})$ . For these sets and the current vertices  $v^{c_1}, v^{c_2}$  the following characteristics must be assured, with  $N(e)$  being the edge's  $e$  name (e.g., one from Table 6.1), and  $T(v)$  being the vertex's type invariant (as seen in Table 6.1).



$$T(v^{c_1}) = T(v^{c_2}) \quad (6.22)$$

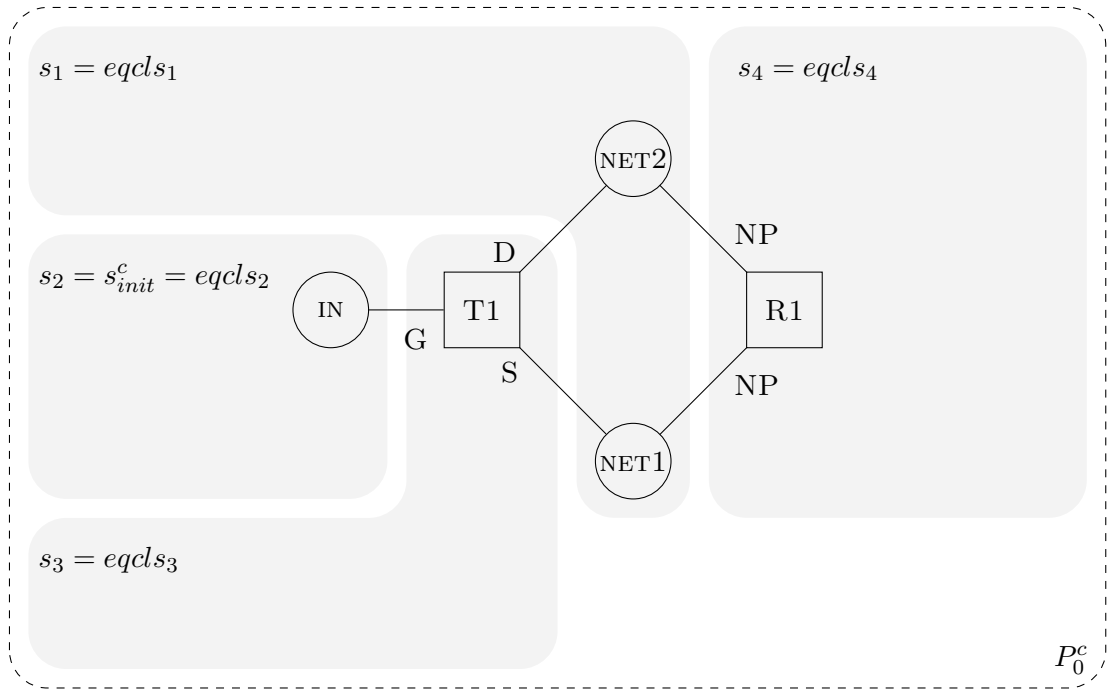
$$|E(v^{c_1})| = |E(v^{c_2})| \quad (6.23)$$

$$\forall e_1 \in E(v^{c_1}) \exists e_2 \in E(v^{c_2}) : N(e_1) = N(e_2) \quad (6.24)$$

$$\forall e_1 \in E(v^{c_2}) \exists e_2 \in E(v^{c_1}) : N(e_1) = N(e_2) \quad (6.25)$$

If any of Equation 6.23, 6.24, 6.25 or 6.22 are violated, the routine returns. Otherwise the routine may proceed and chooses an edge to walk inside circuit  $c_1$  and an equally named edge in  $c_2$ .

#### 6.1.4.2 Labeling-based Isomorphism Algorithm



**Figure 6.4:** The graph for circuit  $c$  from Figure 6.2 correctly annotated to serve as a starting point for the labeling based isomorphism. The starting set (see Definition 6.11) is explicitly marked, assuming its label has the smallest label.

The general idea of the labeling algorithm is to partition a graph according to its vertices' invariants. The first labeling is done according to Definition 6.7 and Definition 6.9, additionally the starting set  $s_{init}$  is determined using Definition 6.11 as shown in Figure 6.4. This initial partitioning for a circuit  $c$  is denoted as  $P_0^c$ —exhibiting the following properties:

$$P_k^c := \{s_1^k, s_2^k, \dots, s_i^k, \dots, s_n^k\} \quad (6.26)$$

$$\text{with } s_{init} \in P_k^c \quad (6.27)$$

$P_k^c$  solely contains  $n$  disjunctive subsets:

$$\forall s, s' \in P_k^c \rightarrow s \cap s' = \emptyset \quad (6.28)$$

The index  $k$  denotes the  $k$ -th refinement of the partition starting with  $k = 0$ . Each  $P_k^c$  resembles a mapping  $f$  for all vertices  $V$ :

$$f : V \rightarrow P_k^c \quad (6.29)$$

The refinement is repeated until the partition solely contains singletons, with a singleton being a subset containing only one vertex:

$$|s_i| = 1 \quad (6.30)$$

If this refinement is done simultaneously for two circuits  $c_1$  and  $c_2$ , one may state:

$$\exists \text{ bijection } g : P_i^{c_1} \rightarrow P_i^{c_2}, s \mapsto s' \Leftrightarrow L_i(s) = L_i(s') \quad (6.31)$$

$$\forall s \in P_i^{c_1}, \forall s' \in P_i^{c_2} : |s| = |s'| = 1 \quad (6.32)$$

$$\Rightarrow c_1 \text{ is isomorphic to } c_2$$

During the refinement, all resulting singleton subsets are constantly checked for equality, once the singleton subsets for two circuits during a refinement differ, it may be stated that the circuits are *not isomorphic*.

Algorithm 3 present the methodology in depth. The function ISOMORPHISMLABELING returns either *true* or *false* for two circuits  $c_1$  and  $c_2$  to state isomorphism, respectively disprove isomorphism.  $v_{init}^1$  and  $v_{init}^2$  are the vertices inside the starting sets from the initial partitioning as defined in Definition 6.11. In particular the outer while statement represents a full iteration through the circuit, whilst the inner implements the *breadth-first search* through both circuits in parallel. GETNEIGHBORS( $u$ ), as the name suggest, acquires all neighbors from a given vertex  $u$ , which are afterwards relabeled within the function RELABELVERTICES( $u$ ). This relabeling is done depending on the type of the vertex  $u$ . If the vertex  $u$  resembles a net the following labeling is applied:

$$L_{n+1}(u) = L_n(u) + \sum_{v \in B(u)} L(v) \quad (6.33)$$

With  $B(u)$  being the neighbor vertices. Accordingly, if the vertex resembles a component, the following equation is used to relabel the neighbors.

$$L_{n+1}(u) = L_n(u) + \sum_{v \in B(u)} \sum_{e \in E(u)} L(v) \cdot L(e) \quad (6.34)$$

The labels for all vertices are kept inside an associative data structure, which is not explicitly mentioned inside Algorithm 3 as it is only used inside RELABELVERTICES, MATCHSINGLETONS and ONLYSINGLETONS. Using an associative data structure eases the realization of the operations done within MATCHSINGLETONS and ONLYSINGLETONS. The former, as the name suggests, matches all singletons for circuit  $c_1$  against all singletons for circuit  $c_2$  and returns *true*, if they match or *false* otherwise. The latter checks all subsets and returns *true*, if all subsets are singletons. Using both, one can state isomorphism, (only) if both return *true* as to be seen in line 22, according to Equation 6.31.

**Algorithm 3** Labeling Isomorphism Algorithm

---

```

1: function ISOMORPHISMLABELING( $c_1, c_2, v_{init}^1, v_{init}^2$ )
2:   while true do
3:      $curQ \leftarrow \{v_{init}^1\}$ 
4:      $compQ \leftarrow \{v_{init}^2\}$ 
5:      $curVisit \leftarrow \emptyset$ 
6:      $compVisit \leftarrow \emptyset$ 
7:     while  $|curQ| > 0$  do
8:        $cur \leftarrow pop(curQ)$ 
9:        $comp \leftarrow pop(compQ)$ 
10:       $curVisit \leftarrow cur$ 
11:       $compVisit \leftarrow comp$ 
12:       $curN \leftarrow GETNEIGHBORS(cur, c_1)$ 
13:       $compN \leftarrow GETNEIGHBORS(comp, c_2)$ 
14:      RELABELVERTICES( $cur, curN$ )
15:      RELABELVERTICES( $comp, compN$ )
16:       $curQ \leftarrow curN$ 
17:       $compQ \leftarrow compN$ 
18:      if not MATCHSINGLETONS( ) then
19:        return false // isomorphism disproved
20:      end if
21:    end while
22:    if MATCHSINGLETONS( ) and ONLYSINGLETONS( ) then
23:      return true // isomorphism verified
24:    end if
25:  end while
26: end function

```

---

## 6.2 Fast Semi-Symbolic Preselection

As mentioned earlier, to facilitate an analog circuit synthesis, a method for a fast pre-selection of viable circuits is crucial. After the isomorphism, this is the second important step to reduce the number of circuits.

Generally, during this step precision is traded for speed. Therefore the currently used technology node has to be analyzed upfront to deduct some parameters, which may be used during the following analyzes. This includes supply voltages, threshold voltages, parasitic capacities and various other technology dependent constants. All these are tailored to deliver optimistic approximations in order to avoid false negatives during this analysis (e.g., minimal threshold voltages). The following sections describe the developed methods to preselect candidates for the sizing step from possibly huge circuit counts.

### 6.2.1 Electrical Constraints

Nowadays, constraint-driven circuit design starts to become established among analog design engineers. The benefits during a regular design sometimes seem hard to explain to the designer, as it forces him to annotate each functional block (e.g., MOS-devices) by hand, which may not always outweigh the benefits.

Contrary to this, during an automated synthesis these constraints are extremely beneficial and lead only to a marginal overhead. The latter originates in the basic block concept. Each basic block (as seen in Section 5.2.2) already contains all necessary electrical constraints. Additionally each component introduces its own constraints, which will be propagated from bottom to top during circuit synthesis.

The used electrical and structural constraints are widely known as *sizing constraints*—based on various publications (e.g., see [SEGA99, GZEA01, MGS08]), the general idea is to bring all devices into their targeted state (e.g., switching, inversion, or saturation). Once all devices inside a circuit reach their target state (i.e., fulfill their constraints), the circuit is called *feasible*. The major difference between the aforementioned publications and the application inside the framework is the fact that usually one needs to extract the structural information from a given circuit. Inside the framework the structures are all known, as they are provided as basic blocks upfront. This allows a precise control of the constraints to be included by either providing them directly in the form of inequalities as described in Section 5.2.1 or implicitly inside the provided basic blocks (see Section 5.2.2) by controlling the design variables and their relations.

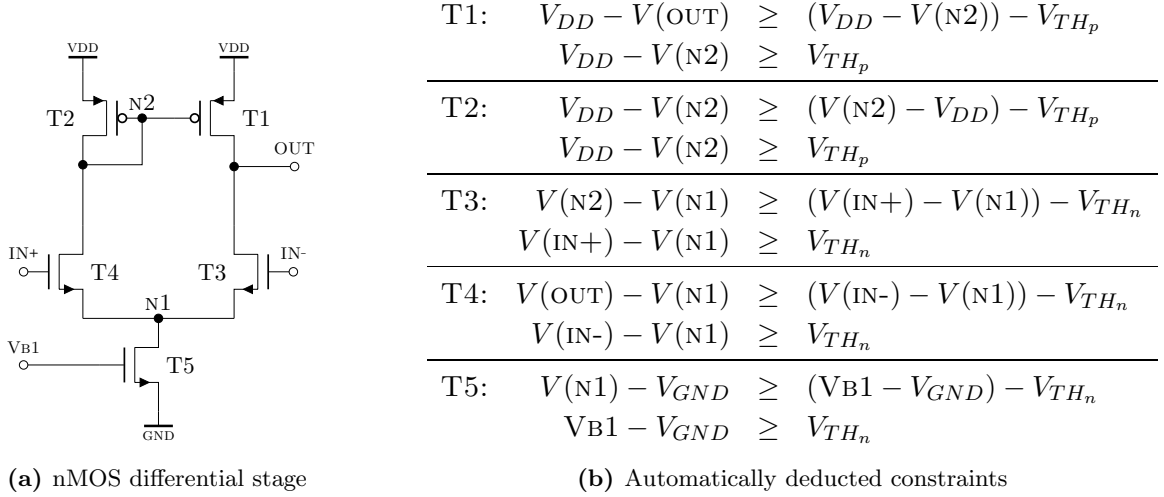
The constraints used in this analysis are the same as the ones used for the sizing process. The important difference is the evaluation of the constraints, instead of simulating the circuit with a SPICE accurate simulator, the framework verifies that all constraints together are feasible. Without using a circuit simulator this method delivers a best-case approximation for the feasibility of the circuit.

This approach is fully technology independent, as the components, thus their constraints, are described in a generic way. The following enumeration gives some examples for the used constraints across basic blocks and components.

- Basic nMOS / pMOS component
  - Saturation
  - (Strong) inversion
- Switching nMOS component
  - $V_{DS} \geq 0$
  - $V_{GS} \geq 0$
- Simple current mirror
  - Equal lengths for all MOS-devices
- Rail-to-Rail input stage
  - Equal lengths/widths for all nMOS-devices
  - Equal lengths/widths for all pMOS-devices

The framework maintains a maximum amount of flexibility by providing an easy and intuitive way to introduce any type of constraint. One may simply add the equation resembling the constraint(s) to any basic block and/or component.

Once a constraint is propagated to the top level circuit, it delivers very useful information for the whole synthesis process. They get passed to the sizing step to force bounds during the circuit sizing. But additionally, they are exploited for the fast pre-selection. This is done by setting up a linear system of inequalities—an minimal example is show in Figure 6.5 containing the saturation and inversion constraints propagated from bottom (e.g., MOS-device) to the top-level. Despite the inequalities shown in Figure 6.5 further ones are considered, e.g., input voltage range constraints, or bias voltage constraints.



**Figure 6.5:** Generation of a circuit’s inequality system—the inversion and saturation constraints.

### 6.2.1.1 Feasibility Analysis

The generated system of inequalities is now evaluated using an—from scratch developed—optimized simplex algorithm. This simplex applied to the system of inequalities provides a very fast feasibility analysis of the circuit and its constraints. To be more precise: if the system of inequalities resulting from a specific circuit has no feasible solution, then the circuit may be directly discarded. This decision strongly depends on the technology node which is used, as a lot of the constraints are formed with technology dependent constants like the respective threshold voltage or supply voltage.

### 6.2.1.2 Output Voltage Range

The previously set up system of inequalities can furthermore be used to provide a very optimistic over approximation for a performance widely used in (operational) amplifier design: the *output voltage range*.

The simplex algorithm allows to define the direction in which any free variable should be optimized after a feasible solution has been found. Namely this means maximizing or minimizing a variable to its optimal value. Translating the output voltage range performance, to the difference

between the maximum and the minimum output voltage, leads to the application of the simplex algorithm to obtain these performances. The difference between these resembles a strong upper bound for the output voltage range. By using the previously extracted technology dependent constants paired with the actual circuit structure, this approximation allows the framework to discard any circuit, which does not reach the specified performance. This method has been used for various technology nodes and has shown a very robust behavior with a minimal ( $< 2\%$ ) amount of false positives.

### 6.2.2 System Analysis

To obtain a reasonable approximation of the (DC-)gain between two net nodes, the full system matrix is generated using the modified nodal analysis. This system description can be easily obtained from inside the framework and is again highly optimized and developed from scratch to maintain a maximum amount of flexibility paired with an outstanding performance due to non existing communication barriers—like a third-party tool connected through a character based (pipe-)interface would impose.

Even though there is a rudimentary Newton-Raphson solver for implicit defined equation systems, this is neither the main priority nor does the world need another new SPICE-accurate simulator. But having the possibility to natively analyze the system opens up a very important door for further fast analyzing methods. Flexibility, is one of the main design goals, which leads to an highly versatile analysis toolbox. The customization possibilities include: the test-benche(s), the used components, technology nodes, basic blocks and component models to be used. Additionally, the target topology or circuit may be analyzed in multiple hierarchies, this even opens up the possibility to apply behavioral models to specific modules in order to accelerate the preselection.

#### 6.2.2.1 Gain Approximation

The main focus of this work is the extremely fast gain approximation, which allows the framework to at least discard 50% of all generated circuits, as the synthesis engine generates each circuit with an inverted gain.

Unfortunately, to calculate the gain between two arbitrary nets inside an electrical network one needs to employ an AC-analysis, which implies the calculation of the DC operating point for the system. Even with a state-of-the-art analog simulator, the calculation of thousands of DC operating points would eat up an considerable amount of resources and wall clock time. The tool startup, netlist read-in, results evaluation and writing and finally the simulation can easily eat up several seconds, which sums up to hours, without considering the waste of these expensive, mostly licensed resources.

Thus, a simplification of the calculation model for the MOS-devices was introduced in order to tremendously reduce the computational effort while still delivering a robust approximation of the (DC-)gain. As the component models of all devices inside the synthesis are easily exchangeable a very simple so-called *static model* for the MOS-devices was realized. This static model assumes the MOS-device to be in saturation and (strong) inversion and deliberately sets an

average transconductance  $g_m$  for each MOS-device without actually calculating it. This reduces the computational complexity of the system from an implicit system of equations down to a quasi linear system of equations, which can directly be solved with e.g., an LU-decomposition.

The assumption that all MOS-devices are in saturation respectively inversion is absolutely feasible, as most MOS-devices operate in saturation. Additionally, the fact that the framework provides the possibility to define and modify all components, their electrical constraints and function inside the circuit is extremely helpful here. E.g., switching MOS-devices are marked and are simply not considered during this analysis. Eventually, the inserted transconductances for the MOS-devices are actual *real* values as the technology nodes was extensively analyzed in an automated process, which simulates thousands of different configurations to obtain a reasonable average transconductance.

Using the static model and an appropriate open-loop testbench the AC-solution is easily computed by multiplying the frequency as a scalar with the AC-admittance matrix and adding the result to the DC-admittance matrix. Once inverted, a reasonable gain (sign) approximation can be taken from the resulting matrix. Some examples of this approach are presented in Chapter 9, which shows evidence that this approach delivers an extremely robust approximation for the gain inversion, while being fast enough to easily handle thousands of circuits.





## UNATTENDED CIRCUIT SIZING

In accordance with the promoted flexibility and scalability (see Section 5.1), the circuit evaluation is carried out by the task manager and an arbitrary number of application servers. The task manager realizes the handling and distribution of the evaluation tasks using any scriptable tool, which has to be encapsulated into an application server. Section 7.1 introduces the concepts realized to communicate between task manager and the application servers, followed by Section 7.2, which describes the actual sizing process in detail.

### 7.1 Task Allocation

To fully utilize the available computing resources an asynchronous task distribution system was developed and integrated into FEATS. The evaluation of (synthesized) circuits is perfectly suited for parallelization due to the inherent independence of the circuits. In other words, a parallelization may easily scale close to linear as each instance of an evaluation tool raises the overall synthesis speed without affecting any of the other instances. In a real usage scenario this linear scaling will surely be affected by hardware limitations, but from the software point of view the overhead for communication, result handling and task allocation is less than 2%.

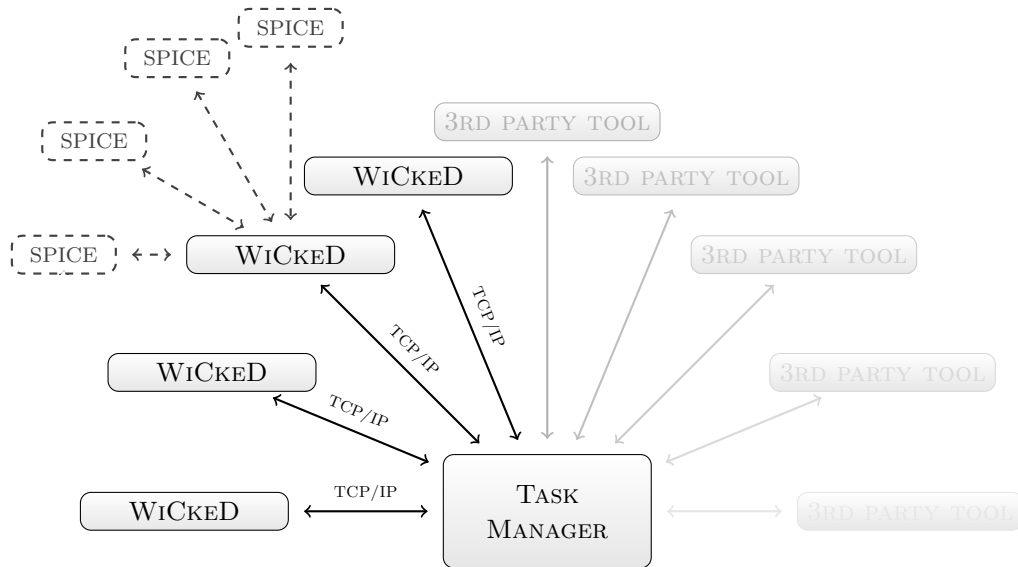
As to be seen in Figure 7.1, the task manager connects to each application server through TCP/IP using a protocol developed from scratch in order to keep the overhead of the communication as low as possible. The application servers may either be started by FEATS during the framework's startup or may be started directly by the user to maintain a maximum of flexibility. The developed communication protocol implements the following capabilities:

**Specification transport** of the current circuit class (circuit template).

**Circuit transport** to deliver the target circuit to the application server.

**Results transport** back to the task manager including any further information generated by the underlying 3rd party application.

**Control commands** manage, initialize and control the application server.



**Figure 7.1:** Illustration of the asynchronous task distribution

FEATS uses application servers to encapsulate any desired 3rd-party software into the framework. Any software may be easily wrapped with a set of tools provided by FEATS. Once the 3rd-party software is encapsulated into an application server, an arbitrary number may be started and used during the synthesis process. Different application servers for various 3rd-party tools have been realized and successfully applied, e.g., WiCkeD [Mun] or Maple [Map].

## 7.2 Sizing

The sizing step is facilitated by a commercial sizing tool named WiCkeD from MunEDA [Mun]. This is one of the persistent, automated, analog sizing tools as described in Section 3.1. It provides a very important and distinctive feature compared to other similar tools, it is fully scriptable, means it may be run in a *batch mode* and (nearly) all operations may be done from a provided script. This allows to fully automate the sizing process, enabling the framework to apply the sizing on an arbitrary number of circuits unattended, without any interaction of a human.

WiCkeD provides a tool, which may be used to properly set up the sizing environment for a given circuit. Although the tool may also be scripted and used in batch mode, the generated environment is far to generic and filled with comfort and help functionalities, which are of no use inside an analog synthesis framework like FEATS. But the environment and especially the file specifications, which are used by WiCkeD are very well documented. Allowing the generation of the environment directly by FEATS, which leads to a massive improvement in terms of overhead, performance and complexity. Despite the latter the framework gets full control of the sizing process and the underlying mechanics, e.g., the used SPICE simulator. This even lead to the prototypical realization of the sizing process with spectreMDL [Cad], which is not supported

by WiCkeD natively. Ultimately, this is crucial for the framework in order to state simulator independence as promoted in Section 5.1.

The following enumeration names the essential parts of the sizing environment, which are generated by the framework.

**Testbenches** especially tailored for each circuit class, thus template-driven. Some of the actually used testbenches are shown in Chapter 8.

**Measurements** for each specification, in order to calculate the performance for the circuit. Especially all propagated (electrical) constraints are automatically generated as measurements by the framework.

**Sizing process control file** realizes the actual sizing steps to be taken.

**Technology node files** for the used SPICE simulator in its native format.

**Technology parameters** extracted once upfront, like the parameter grid, minimal/maximal component dimensions and rough active area approximations with respect to the component dimensions.

**Simulator control files** including simulator start-up files, extraction scripts in order to extract the performances from generated simulator output and finally some error handling routines to enable the user to debug problems.

**(Flattened) netlist** of the currently sized circuit including anything needed by the simulator.

**Design variables** to be used for the parameter synthesis during the sizing.

One of the major challenges inside FEATS is the construction of the *testbenches*, *measurements* and the *sizing process control file*. The former are further discussed in Chapter 8 including the measurements. The sizing process control file defines which steps are taken in order to realize a sizing on a circuit which comes without any initial sizing. Generally, the sizing as a whole consists of two distinct steps:

**Deterministic Feasibility Optimization**, which is applied to center the design in terms of electrical constraints. For most circuits this translates to bringing all (appropriate) MOS components in saturation and (strong) inversion. In particular this mostly leads to a sizing, which already exhibits fundamental functionality. Furthermore, this step is relatively fast and if not successful a strong argument to discard the circuit directly.

**Deterministic Nominal Optimization** facilitates the actual sizing towards the previously defined specifications. All specifications during this first optimization step are handled as simple *bounds*, in other words the optimizer tries to generate a set of parameters, which lead to specifications that simply fulfill the defined specifications and does explicitly no maximization respectively minimization.

A successfully finished deterministic nominal optimization automatically leads to a *good circuit*, i.e., all defined specifications are fulfilled. While this circuit is now functional in terms of performances, there is still potential to further optimize the circuit. Usually the specification implicitly contains performances, which should be further optimized towards a direction or even an explicit target. E.g., the area should usually be minimized and an offset should be optimized towards a systematic offset of exactly zero. FEATS therefore provides the possibility to further optimize the circuit by enabling the user to set various other optimization goals for each of the specifications. The following enumeration shows the realized capabilities:

**Priorities** provide a mechanism to prioritize specific specifications. A performance which exhibits a higher priority is handled and optimized first and all other performances with a lower priority are only evaluated and optimized, if the ones with a higher priority already fulfill the specifications.

**Scaling** allows the user to specify the impact of a specific performance inside the optimizer's cost function. Usually the optimizer calculates the scaling automatically, but there may be cases for which the user wants to overrule this mechanism, e.g., to increase the impact of the area.

**Optimization direction** realizes a maximization respectively minimization or target optimization. This is particularly useful if one aims to size the circuit towards a pareto optimal point.

Despite the realized capabilities, all other optimization methods provided by WiCkeD are applicable. Worth mentioning here is the yield optimization, which is currently not fully supported, but could easily be included into FEATS using already available mechanisms and tools.

## PRESENTATION OF SELECTED CIRCUIT CLASSES

In this chapter the primary goal is to demonstrate the capabilities of the presented methodologies on actual circuit design cases. A set of representative examples was chosen in order to present the flexibility and the scalability of the framework. While the first, smaller analog modules presented, focus on the presentation of the concept as a whole, the final example shifts the focus towards the increase of size and complexity of the synthesized circuits.

Furthermore, the developed circuit classes, respectively circuit templates, are illuminated, especially the used testbenches and performances to evaluate the circuits are presented in detail. The synthesized circuits are additionally analyzed in terms of industrial applicability and scientific competitiveness by providing comparisons to recent publications to provide all information to allow the reader to classify the synthesized circuits in a greater context. This chapter aims to provide precise information about the synthesized circuits, but not every circuit template used, is illuminated, as e.g., RC-networks can easily be calculated using freely available tools or CAS-systems.

To maintain a high degree of applicability and comparability the presented circuits are all synthesized inside a real 350nm bulk CMOS process node from austriamicrosystems using a supply voltage of 3.3V. The wall clock time taken for the synthesis engine including preselection is usually below three minutes—thus the sizing is the determining factor for the synthesis' runtime. The framework implements an asynchronous task distribution to allow a parallelized sizing using an arbitrary number of sizer instances (see Section 7.2).

The here encouraged approach to design analog circuits is highly influenced by the proposed framework and therefore differs—not fundamentally—but primarily in terms of chosen abstractions. Most analog literature and mainstream analog design methodologies widely match in their approach to design analog circuits by often introducing implicit hierarchies, i.e., even (medium sized) analog modules mostly end up with less hierarchy levels than a purely computational approach would suggest. Humans tend to understand and handle redundancy far better than strict hierarchies, especially if these aim to be clean and *computational consistent*, they start to be overwhelming. Remarkably, in relational database design a similar constellation called *normalization* occurs, which is often little liked (e.g., see [For06]). This design principle—possibly due to the fact that database structures are designed by humans—recently suffers.

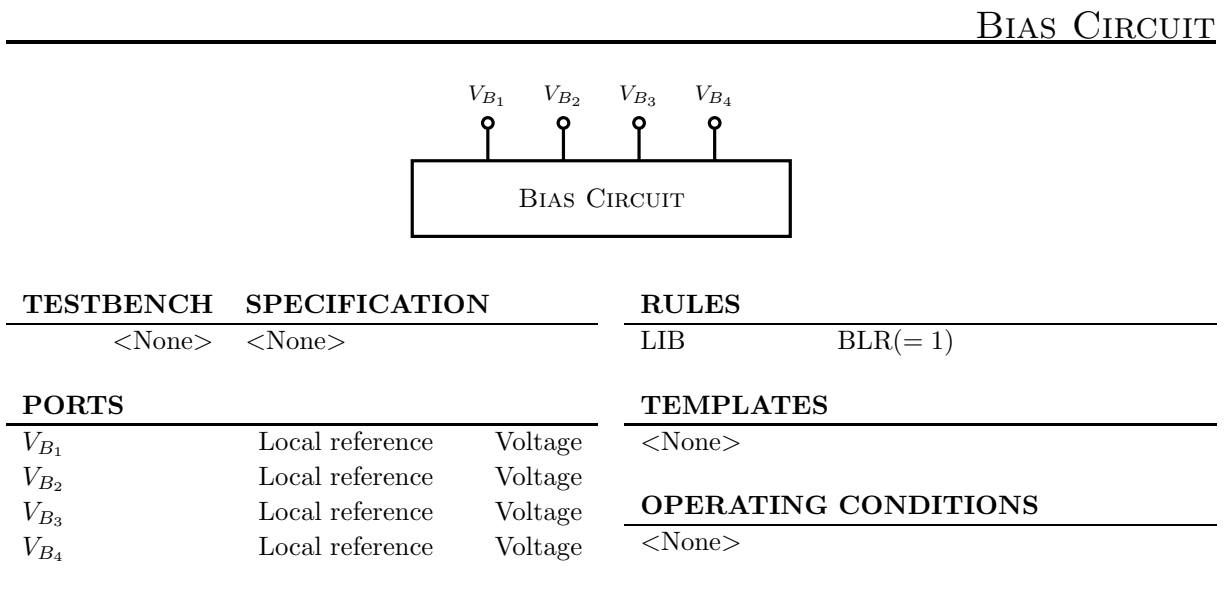
FEATS aims to provide full automation capabilities, including all advantages of a highly consistent, unattended, and computational methodology. Coupled with an interface, which still fits into the current design practice by *speaking the same language* as an electrical engineer does, giving him tools to apply his knowledge and experience using familiar concepts. This chapter provides the necessary information to underpin this statement.

## 8.1 Utility Circuit Templates

Although it is possible to synthesize each circuit template by using small basic blocks, it is not truly feasible for all applications. The framework—as promoted earlier—allows to transparently blend between the basic blocks approach and a library approach for synthesis. For instance the following circuit templates are provided as they are without further modifications in terms of structure.

The utility circuit templates presented in the following are (currently) not tailored to be synthesized by themselves. Thus there are no testbenches for them and in particular they are only used in conjunction with other circuit templates and are evaluated together with them as a whole.

### 8.1.1 Bias Circuit



**Figure 8.1:** Bias Circuit

Before focusing on the actual function of this circuit class, there are various fundamental properties for this particular circuit class presented in Figure 8.1:

- No (signal) inputs and processing is done

- Provides  $n$  voltages, each named like:

$$V_{B_i} : i \in \{1 \dots n\} \quad (8.1)$$

- All *bias-voltages* must not be identical with the ground or supply voltage, i.e., somewhere in between those two potentials:

$$V_{GND} < V_{B_i} < V_{DD} : i \in \{1 \dots n\} \quad (8.2)$$

An *ideal* bias circuit should furthermore maintain the adjusted bias voltages or voltage current ratios under all circumstances, this includes drifts in *temperature*, *supply voltage* or any other *operating condition*. But notably the provided bias-voltages should be connected exclusively to high impedance nodes, i.e., MOS gate pins, in particular this means to never connect a resistive load, as this would have a negative impact on the provided voltage.

### Bias Circuit Implementation

The *bias circuit* is an important part of nearly all linear analog circuits and even for many analog circuits in general. The primary task of a bias circuit is to generate bias voltages for the functional part of the analog circuit. The analog literature provides a large number of possible bias circuits, but most—if not all—are based on the voltage divider concept. Furthermore, one should not confuse a bias circuit with a bandgap reference or constant current reference as they form the base of most bias circuits. The bias circuits used for the presented set of examples is shown in Figure 8.2, it contains an ideal current source, which may be set to a fixed directed current value to resemble a constant current source as provided by most process technologies. Alternatively an user may realize a bandgap or similar constant current generator directly inside the bias circuit, but this usually does not outweigh the unnecessary increase of simulation times. Thus this *behavioral* representation was chosen.

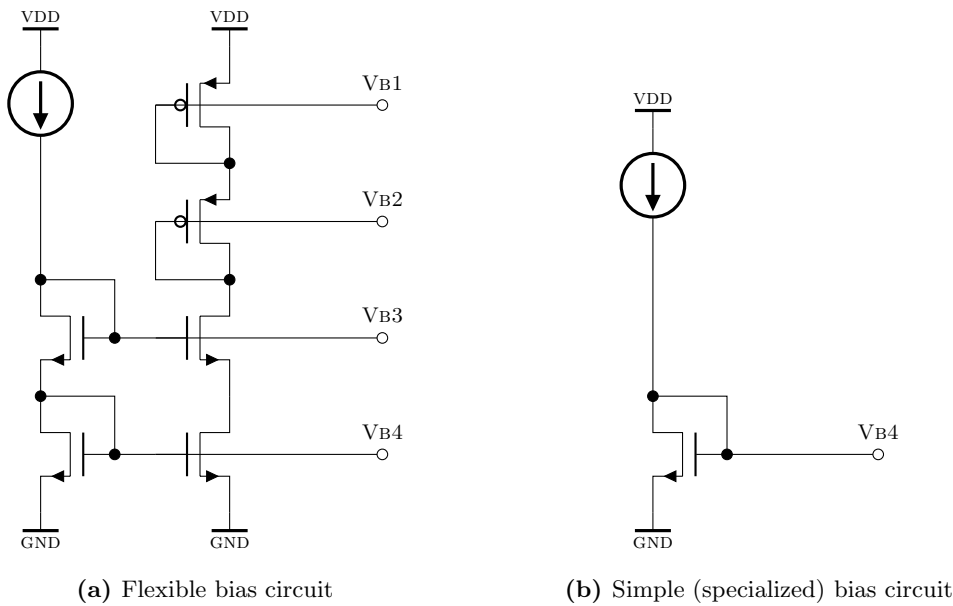
The directed current provided through the ideal current source is fixed for all the following circuit synthesis examples as it will be simply replaced by the constant current source generator from the technology node. Besides this, the bias circuit provides three degrees of freedom to be used as design parameters inside the sizing:

**WBias** The width of all pMOS and nMOS transistors.

**LBias** The length of all pMOS and nMOS transistors.

**WCascMult** A multiplier used for the widths of the inner two bias voltages ( $V_{B_2}$  and  $V_{B_3}$ ).

It is important to mention that although the pMOS and the nMOS devices' widths are based on the same design variable (WBias), the actual width of the pMOS devices differ, as their widths get multiplied by a variable named *pfac*, which compensates the difference of the *charge-carrier mobility* between nMOS and pMOS devices, this value usually is between two and four. This originates in the fact that the *holes* have to move by detaching electrons from one atomic



**Figure 8.2:** Various bias circuit realizations

nucleus and reattach to the next atomic nucleus, this process takes significantly more time than an electron being passed through, as inside an nMOS channel.

Another, not to be underestimated, design decision for this block is its robust behavior against process variations. All devices inside this block are realized with an equal length, enabling to build a very robust layout. This is furthermore encouraged through the fact that all biased nMOS and pMOS devices inside the biased circuit also get the same design variable assigned for their lengths. This leads not only to a reduced amount of design variables, an easily adjustable bias current (simply using the width as a multiplier for the bias device) but moreover resembles a widely used design methodology used by analog designers. Nevertheless, due to the template based approach (see Section 5.2.3) this block may easily be replaced by a more sophisticated biasing block, i.e., to compensate for manufacturing deviations between nMOS and pMOS devices. Furthermore, the automatic topology generation process allows to select the best alternatives automatically.

### 8.1.2 Voltage-Averaging Circuit

A voltage-average-generation circuit calculates—as the name suggests—the average voltage  $V_{avg}$  from two voltages  $V_1$  and  $V_2$ , formally this means:

$$V_{avg} = \frac{V_1 + V_2}{2} \quad (8.3)$$

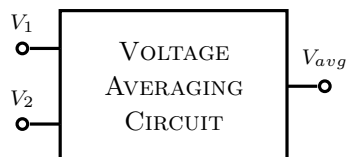
The theory is trivial, but an actual implementation has to address various challenges. Analog circuit design literature mostly does not explicitly deal with this inconspicuous circuit, moreover it is presented as a part of common-mode-feedback circuits (see Section 8.1.3), for the measurement of supply voltage drift or simple power dissipation measurements. This circuit template



serves as a typical example for a hierarchy which usually would be a (redundant) part inside a CMFB implementation.

- The sensed voltages  $V_1$  and  $V_2$  should not be affected by the measurement itself, in particular this means:
  - Resistive load, as high as possible
  - Capacitive load, as low as possible
- Sensing should work for any voltage between ground and supply voltage as linear as possible
- React to changes of  $V_1$  and/or  $V_2$  as fast as possible
- A though area vs. performance trade-off

### VOLTAGE-AVERAGING CIRCUIT



TESTBENCH		SPECIFICATION		RULES	
<None>		<None>		LIB	BLR(= 1)
PORTS			TEMPLATES		
$V_1$	Input	Voltage	<None>		
$V_2$	Input	Voltage			
$V_{avg}$	Output	Voltage	OPERATING CONDITIONS		
			<None>		

**Figure 8.3:** Voltage-Averaging Circuit

As with most ideal behavior descriptions a real implementation has to compromise, mostly not because the possibly best solution can not be realized, but more because the solution may be not known upfront. Facing complex process technologies and the inherent mutual interference of analog circuits, most designs today rely on experience and what was used before. Exactly at this point FEATS can support the design process significantly. By enabling the user to provide several implementations for a given circuit template and let the synthesis framework do the trial and error work.

### Passive Common-Mode-Sensing Circuit

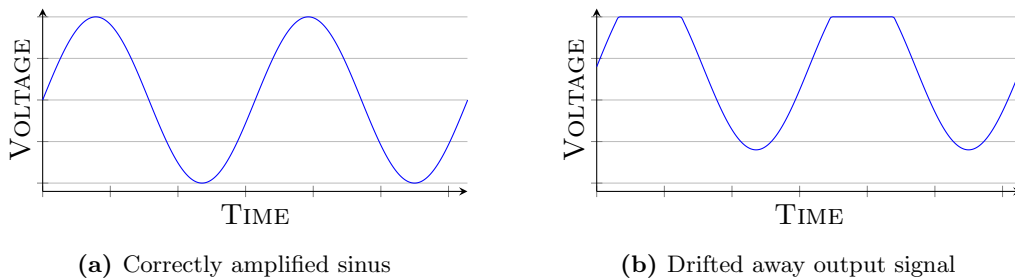
A passive realization resembles the most primitive thus still useful circuit for common-mode-sensing and is shown in Figure 8.6. If both resistors and capacities have equal dimensions

$R_1 == R_2$  and  $C_1 == C_2$ , then the voltage dividing property of this circuit generates an average voltage  $V_{avg}$  between both input voltages leading to the desired behavior. Although this implantation provides the necessary functionality, it has to suffer from some drawbacks. In fact, one might nevertheless keep this circuit inside the library, as it has the distinctive property that its behavior is very good natured, allowing to focus on more complex parts of the analog module. This neglects most layout related issues as the resistances have to be sized with very big resistances.

### 8.1.3 Common-Mode-Feedback-Amplifier Circuit

A common-mode-feedback-amplifier takes a very important role once an amplifier design is developed to provide a fully differential operating mode. To avoid common mode drift, even-order harmonics or even clock feed-through, the differential output nodes DC voltages have to be regulated towards a predefined  $V_{CM}$ , which is usually set to the middle voltage:

$$V_{CM} = \frac{V_{DD} - V_{SS}}{2} \quad (8.4)$$

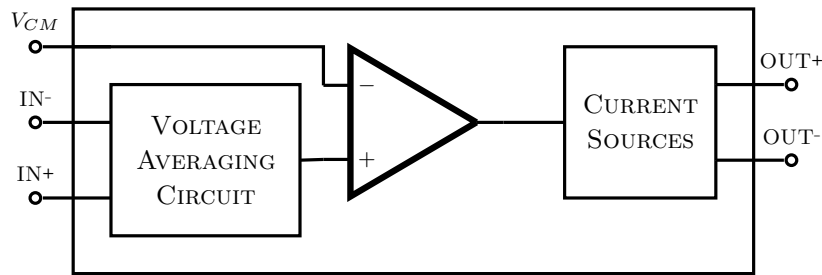


**Figure 8.4:** Fully differential output behavior without common mode feedback

Figure 8.4 shows the typical behavior of differential output nodes without an appropriate common-mode-feedback. The uncontrolled output nodes tend to either drift towards the high or the low end, fully disabling the next input stage. Despite the necessity of regulating the output nodes, a well designed common-mode-feedback may also contribute positively to the circuits electrical properties, e.g., by providing additional gain, stability, or speed.

The block-level diagram of the common-mode-feedback reveals this property, as the realization practically consist of a single-ended operational amplifier comparing the averaged voltage with the target voltage  $V_{CM}$  and driving two current sources to inject current into each of the *floating output nodes*, thus forming a closed control loop. In other words, the mean voltage between the differential nodes is constantly forced towards the defined target voltage  $V_{CM}$ .

## COMMON-MODE-FEEDBACK CIRCUIT

**PORTS**

$IN+$	Input	Voltage
$IN-$	Input	Voltage
$V_{CM}$	Input	Voltage
$OUT+$	Output	Current
$OUT-$	Output	Current

**RULES**

CTR	BLR(= 3)
-----	----------

**TEMPLATES**

Voltage-averaging circuit
Single-ended operational amplifier
Current source

**OPERATING CONDITIONS**

&lt;None&gt;

**TESTBENCH SPECIFICATION**

&lt;None&gt; &lt;None&gt;

**Figure 8.5:** Common-Mode-Feedback Circuit**Simple Common-Mode-Feedback Amplifier Realization**

Although it would be possible to synthesize the single-ended operational amplifier using the basic block approach, the necessity is not given (yet). In real application scenarios, an exploration of more sophisticated amplifiers—compared to the used simple differential stage—would be feasible by replacing the amplifier block as seen in Figure 8.5 with a single-ended operational amplifier circuit template as described in Section 8.2.1. To be more precise, this could be done by changing a single line inside the circuit template provided for the common-mode-feedback circuit (class). Figure 8.6 shows the current transistor-level implementation used for the here presented examples. Although the circuit is simple, it is widely used in industrial applications. Furthermore, with the template based approach, itself could be the target of a full hierarchical structure synthesis run, to improve the circuit in the context of surrounding circuitry. Obviously, this circuit is neither optimized for area nor for providing additional gain, but the flexibility provided by the synthesis framework allows to apply *rapid prototyping* development methods, known from computer science, in analog circuit design, i.e., aim for an early working prototype and decide afterwards where and how the design has to be further optimized. The explorative nature of FEATS encourages the user to investigate the design space, without being afraid of the effort to even do multiple redesigns.

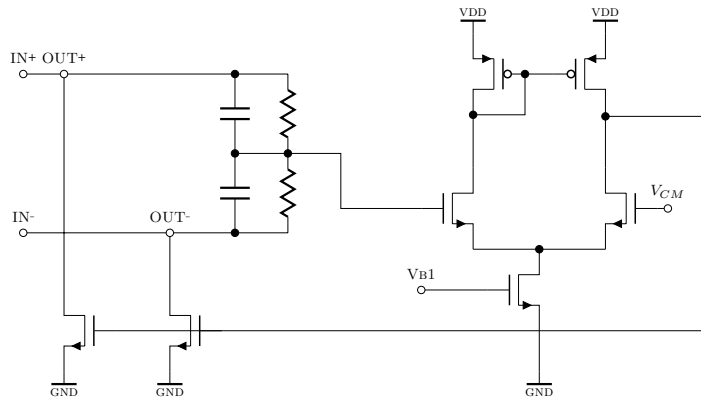


Figure 8.6: Simple Common-mode-feedback amplifier realization

## 8.2 Circuit Templates for Complex Circuit Classes

In the previous section simple utility circuit templates were presented to give the reader an impression how the library-based synthesis (see Section 3.1) approach integrates transparently into FEATS. Amplifiers, in particular, operational amplifiers are inherently complex. Multiple stages, cascoding devices, folding, compensation, symmetry, biasing are just the basic design decisions to take towards an operational amplifier fulfilling the required specifications. Thus *creativity* is of great importance during the design of operational amplifiers. But even the inevitable progress in computing capacities and speed will most likely not solve the complexity problem, as discussed in Chapter 4. On the other side, it is even less likely that the analog designer will ever be replaced by a piece of software.

The following examples and their analysis will demonstrate the latter. Using the framework as presented, the designer has to think in a higher level of abstraction, leaving the responsibility and important (design) decisions in his hands. Ultimately it is crucial for a successful synthesis to know the concepts, to choose the right set of basic blocks and to activate the right rules for the current project. In general FEATS realizes the concepts and provides the necessary tools to also create circuit structures, which may have not been seen before. Nevertheless, the main area of application should be a significant increase in productivity accompanied with the implicit gain in reliability leading to robust system designs. This enables the analog designer to focus on the creative part of the process instead of repetitive trial and error iterations and reduces the overall (economic) risks inherent to analog design.

### 8.2.1 Single-Ended Operational Amplifier (OP)

In analog circuit development, one of the most used circuits, especially in signal processing and amplifying. There is a wide variety in electrical specifications for this circuit class, several important ones are listed in Figure 8.7. In analog literature (e.g., [Raz00, Ndj11, San06]) a vast amount of different topologies may be found, one may even find recent literature featuring OPs exclusively as [Deh13, Hui11]—originating in exploiting advantages in process technologies (e.g., reduced supply voltage) and constantly changing requirements in specifications. The avail-

able analog literature clearly shows how well-suited *amplifier* circuits are for creative synthesis methodologies in general due to its inherent creative component.

An ideal OP amplifies the voltage difference between  $IN+$  and  $IN-$ , using the gain  $A$  and making it available at the  $OUT$  pin. A wide variety of outer circuitry is known (see [Raz00]), especially feedback configurations provide useful functionalities for many analog applications. Different analog applications have varying requirements, leading to diverging specifications. Usually an ideal OP exhibits the following properties:

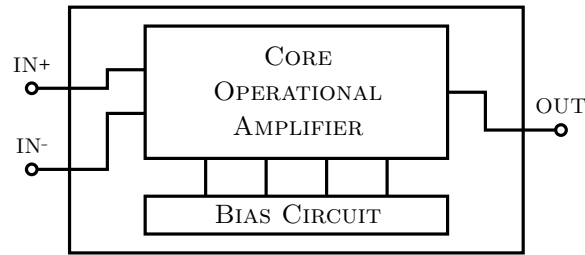
- Infinite
  - Open-loop gain  $A$
  - Input impedance  $R_{in}$
  - Output voltage range
  - Common-mode rejection ratio (CMRR)
  - Power supply rejection ratio (PSRR)
  - Slew rate
- Zero
  - Input current
  - Input offset voltage
  - Output impedance
  - Noise
  - Phase shift

These specifications may naturally not be reached by a real OP, despite this, a real OP is massively influenced by its outer circuitry. In particular the loads, which have to be driven, play an important role while developing a real implementation. All these (as shown in Figure 8.7) may be provided in order to set up a synthesis run. Although it is possible to omit most of the specifications and operating conditions from Figure 8.7, the user is encouraged to provide as much specifications as possible. The more specific the user is in its description of the synthesis target, the more reliable and robust the final synthesized circuit(s) will be. This implicitly forces the user to think about the specifications left out during his first ideas. Simply providing realistic estimates for as many specifications as possible, already prevents the circuit from behaving unpredictable in common application scenarios.

FEATS aims to describe analog circuits based on their signal flows. Therefore the OP consists of several inner circuit templates, realizing the behavioral-first top-down design methodology in a very strict and consistent fashion. The here presented OP in Figure 8.7 demonstrates a reduced approach. More complex examples will be presented in the following sections, additionally various utility circuit classes, as shown in Section 8.1, may be added at any point providing further functionalities or enhancing its electrical specifications.

While designing an analog circuit, one has to set up several different testbenches, each one aiming to measure a specific set of performances. Although most simulators, and in general analog design environments, provide components that enable switching between different outer circuitries, loads, or similar—usually *best practice* is to develop a separate testbench for each test scenario.

## SINGLE-ENDED OPERATIONAL AMPLIFIER



TESTBENCH	SPECIFICATION	PORTS		
Operating Point	Offset	IN+	Input	Voltage
	Power	IN-	Input	Voltage
	Area	OUT	Output	Voltage
Open Loop	Gain	<b>RULES</b>		
	Gain bandwidth product	CTR	BLR(= 2)	
	Phase margin	<b>TEMPLATES</b>		
	Transit frequency	Bias circuit		
	Cutoff frequency	Core operational amplifier		
CMRR	CMRR	<b>OPERATING CONDITIONS</b>		
PSRR	PSRR	VDD	Supply voltage	
		VREF	1/2 Supply voltage	
		IVR	Input voltage range	
		CLOAD	Capacitive load	
		RLOAD	Resistive load	

Figure 8.7: Single-Ended Operational Amplifier

Figure 8.7 already reveals that FEATS may consider multiple testbenches for each circuit with ease. Despite the inherent consistency of the *best practice* approach, the testbenches are easier to maintain and to port to other simulators, if they are clearly separated. The framework may not enforce this, but encourages the analog (testbench) designer to do so, by providing mechanics to allow an easy addition of new testbenches.

The testbenches shown in Figure A.1 are well known in analog literature and target the OP's resilience to common mode operation on the input pins respectively the ability of the circuit to reject power supply noise. Notably the loads are omitted and the PSRR is only determined for the positive power supply noise. Further testbenches are tailored to determine the operating point (see Figure A.1c), the transient behavior (see Figure A.1d) and the open-loop behavior (see Figure A.1e) of the device under test (DUT). All of the latter contain a resistive and a capacitive load. Neither of them has a fixed (i.e., resistance or capacity) value inside their testbenches set explicitly—conveniently both loads may be adjusted inside the circuit template

specification instead. This exposes an important property of the testbenches used inside FEATS: all testbenches may be parametrized and dynamically generated, based on their circuit template specification. This does not only include the loads, but moreover the supply voltage, a reference voltage (e.g., the used virtual ground and/or a specific middle voltage) and any value the user provides through the specification, i.e., the circuit template. This is a very important feature, as this realizes one of the promoted flexibilities, as discussed in Section 5.1, to accomplish a true process technology independence.

The generic testbenches ensure not only the process technology independence—each enhancement of a testbench will, without further user interaction, directly migrate to any other process technology, which was made available inside FEATS. This enables the user to work and design inside a testbench, which has proved its strength in previous designs without the need to *reinvent the wheel* over and over again. This *automated reuse* of analog intellectual property is one of the key strengths of FEATS and has been a hot topic in various research projects (e.g., [SyE]), publications as [MMH12], and inside the ITRS [ITR]: knowing that, the currently used testbench(es) are not only valid and usable for this single circuit, further this encourages the improvement of the testbenches for a circuit template. Despite the significant productivity gain of such an approach, the robustness and reliability of the developed circuits is undeniably increasing.

Nevertheless, not all standard textbook testbenches may be directly used inside an analog synthesis framework. Usually the DUT for a testbench exhibits at least some fundamental functionality of the targeted circuit class, which may not be assumed for each circuit explored during synthesis. For an OP this could be e.g., a reasonable step response. A typical design flow might first parametrize the OP towards a systematic offset of zero and *afterwards* the open-loop testbench is set up to measure the theoretical gain using an AC analysis. These implicit assumptions are regularly found during the automation of processes, which were previously carried out by humans. Specifically for the OP this may be observed inside the open-loop testbench as shown in Figure A.1e. Usually one would assume a literal open-loop configuration, but this might introduce a not negligible error during the gain calculation due to the voltage at the negative input pin being optimal (i.e., middle voltage) instead of shifted by the OP's systematic offset.

Samples and some cherry picks of synthesized single-ended operational amplifier circuits, schematics and measurements are presented in Chapter 9.

### 8.2.2 Core Operational Amplifier (COREOP)

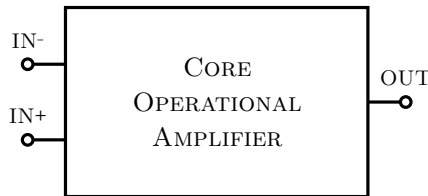
This circuit template stands for the most creative approach in circuit synthesis. While the previously discussed circuit templates either realize hierarchies and/or library based synthesis, this circuit template utilizes a wide range of synthesis rules (use Section 5.5 as a reference) to realize an extensive design space exploration up to a given restriction of the design space (see rules in Figure 8.8). The therefore used basic blocks (see Appendix B), may be freely chosen by the user.

Including, excluding and creating basic blocks is similar to the (creative) decision process the user performs while designing a circuit from scratch. The decisions made during a classic analog design, can directly be transferred into the synthesis-driven analog design. E.g., is a rail-to-rail input stage needed? Is a compensated stage useful or even mandatory? Is there enough voltage headroom to use cascoding structures? Does the targeted load enforce high output currents? All these questions lead to a decision. This decision leads to a functional idea or concept, which can be realized using several differing implementations. Due to the fact that classic analog design means designing a *single* circuit, a single realization for the functional idea is chosen and included into the circuit.

Excluding the final step, the identical reasoning may be applied to a synthesis driven analog design. Instead of including a single chosen realization directly into the schematic, one might take *all realizations of the functional concept* and simply include them as basic blocks in order to let the synthesis engine explore *all combinations* of those—resulting in a much greater chance to find the circuit exhibiting the required properties. Moreover the opposing fact: the synthesis lead to not a single circuit fulfilling the specifications, allows to conclude that the specifications may be too tough and possibly the requirements have to be readjusted. Although the latter may also happen in classic analog design, the important difference is: either argue with *a few* circuit structures, which failed to achieve the specifications or with *hundreds* of circuit structures, which have been explored systematically.

**CORE OPERATIONAL AMPLIFIER**

---



TESTBENCH	SPECIFICATION	RULES		
<None>	<None>	IBR	BLR( $\leq 4$ )	EXPROPS
<b>PORTS</b>		SYM EER	OMR	ISO
IN+	Input		NB2R	SYM PRE
IN-	Input		VVOER	SYM GAIN
OUT	Output		TIR	SYM FEAS
$V_{B_1}$	Local reference		<b>TEMPLATES</b>	
$V_{B_2}$	Local reference		<None>	
$V_{B_3}$	Local reference		<b>OPERATING CONDITIONS</b>	
$V_{B_4}$	Local reference		<None>	

---

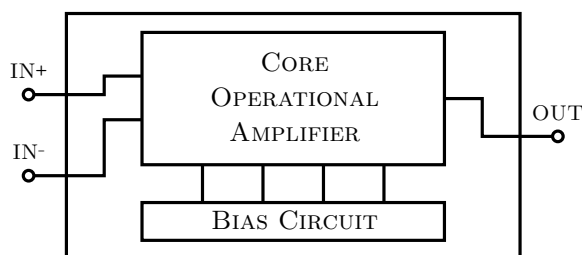
**Figure 8.8:** Core Operational Amplifier



The user may freely add arbitrary basic blocks and leave out any unwanted basic blocks to steer the exploration towards the direction(s) he is targeting. In particular it is an ease to precisely target a specific circuit. This somehow exploits the synthesis, but once this specific circuit is build from more than just one basic block, FEATS will most likely generate: structural similar circuits, subsets, and supersets of the targeted circuit— e.g., by creating complementary merged blocks (see Section 5.3.1), introducing new symmetries (see Constructive rule 3)—according to the enabled rules inside the current circuit template. There is no need to care about *supporting circuitry* like bias, enable or start-up circuits. These are clearly separated from the functional (signal processing) part of the circuit and later appended in higher hierarchies. A wide variety of synthesized circuits is presented inside Chapter 9.

### 8.2.3 Single-Ended Operational Transconductance Amplifier (OTA)

#### SINGLE-ENDED OPERATIONAL TRANSCONDUCTANCE AMPLIFIER



TESTBENCH	SPECIFICATION	PORTS		
Operating Point	Offset	IN+	Input	Voltage
	Power	IN-	Input	Voltage
	Area	OUT	Output	Current
Open Loop	Gain	<b>RULES</b>		
	Transconductance	CTR	BLR(= 2)	
	Output resistance	<b>TEMPLATES</b>		
	Transit frequency	Bias circuit		
	Cutoff frequency	Core operational amplifier		
Open Loop Phase	Phase margin	<b>OPERATING CONDITIONS</b>		
Linearity	Input voltage range	VDD	Supply voltage	
	Unity	IVR	Input voltage range	
	Output voltage swing	CLOAD	Capacitive load	
	Slew rate (fall/rise)	RLOAD	Resistive load	
	Overshoot (fall/rise)			
	Settling time (fall/rise)			

**Figure 8.9:** Single-Ended Operational Transconductance Amplifier

In contrast to the OP, the OTA *converts* a given voltage difference  $V_{diff}$  to a current  $I_{out}$ , using its transconductance  $g_m$ :

$$I_{out} = V_{diff} * g_m \tag{8.5}$$

This section will not provide an extensive technical overview over OTAs as given (for the OPs) inside Section 8.2.1. Instead, the reuse and expansion of existing pieces inside the framework is demonstrated in order to synthesize a circuit which differs in its functionality.

This circuit template was mainly chosen for presentation because of the fact that the *core operational amplifier* is the same as the one inside the OP as shown in Section 8.2.1. The reason the core may be reused here originates in the realized strict hierarchical concept—the COREOP circuit template is totally decoupled and independent from the OP circuit template, thus it may be used inside any other circuit template. An application of this template inside an actual system design is presented in Section 9.2.

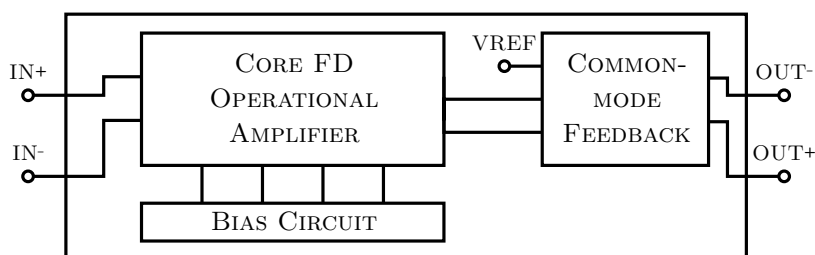
#### 8.2.4 Fully Differential Operational Amplifier (FDA)

The design of an FDA turns out to be very complex and time consuming. This will most likely be induced by the inherent challenges—namely the common-mode feedback (as discussed in Section 8.1.3), the stability to gain trade-off, and the mandatory, highly symmetric structures. Pole zero analysis also suddenly becomes crucial during the design process in order to assure stability for a wide range of applications. Additionally, the testbenches which have to be used, resemble tough design tasks by themselves. Nevertheless, these challenges are worth the hassle as an FDA provides some unique intrinsic properties. The excellent common-mode and power supply noise rejection characteristics are perfectly suited for mixed-signal applications, as they tend to have a noisy supply voltage due to the constant switching of the digital parts inside the circuit. The most prominent examples are for sure (high-precision) ADCs, which increasingly often force the designer to input the analog signal in a differential manner to reduce the influence of the negative and/or positive supply voltage noise, and other noise sources like crosstalk.

This virtually perfect decoupling of the power supply from the transported signal is inherently important for low-voltage applications, especially in modern highly integrated applications i.e., SoCs. Especially high requirements for the signal processing in e.g., mobile radio frequency applications can nowadays only be achieved with FDAs.

Interestingly the presented analog synthesis framework is perfectly suited for the design of highly symmetric analog circuits. In a recent evolution, prior to the inclusion of the FDA circuit template, the symmetric elementary electric rule (see Constructive rule 3) was developed to increase the amount of symmetric circuits inside the generated circuit structures. In fact the actual synthesis happens in a hierarchy one level deeper, similar to the OP. This circuit template is not listed here, as it is despite the second output exactly the same as the COREOP (see Section 8.2.2). Although the generated topologies differ fundamentally, the only thing the user has to change is to add the second output node.

## FULLY DIFFERENTIAL OPERATIONAL AMPLIFIER



TESTBENCH	SPECIFICATION	PORTS		
Operating Point	Offset	IN+	Input	Voltage
	Power	IN-	Input	Voltage
	Area	OUT+	Output	Voltage
Open Loop	Gain	OUT-	Output	Voltage
	Gain bandwidth product			
	Phase margin			
	Transit frequency			
	Cutoff frequency			
CMRR	CMRR			
PSRR	PSRR			
Unity	Output voltage swing			
	Slew rate (fall/rise)			
	Overshoot (fall/rise)			
	Settling time (fall/rise)			
	Position 1st pole			
		RULES		
		CTR	BLR(= 3)	
		TEMPLATES		
		Bias circuit		
		Core fully differential operational amplifier		
		Common-mode feedback circuit		
		OPERATING CONDITIONS		
		VDD	Supply voltage	
		VREF	1/2 Supply voltage	
		IVR	Input voltage range	
		CLOAD	Capacitive load	
		RLOAD	Resistive load	

Figure 8.10: Fully Differential Operational Amplifier

The actual challenge to synthesize FDAs is the development of the testbenches and the formulation of the accompanied measurements. While the common-mode rejection ratio testbench shown in Figure A.2b is very similar to the one used for the OP as shown in Figure A.1, the open-loop (Figure A.2a) and unity gain (Figure 8.11) testbenches are not so easily deduced from the ones used for the single-ended variant. Namely any feedback resembles a challenge, as the inputs have to be decoupled from the ideal voltage sources used for the input stimuli.

To compensate the offset of the DUT the open-loop testbench is also a *faked* open-loop, featuring the same method as the open-loop testbench for the OP. Furthermore the unity gain testbench shown in Figure 8.11 provides a specialized part, which is used to test the common-mode feedback circuit, which has to be included during the sizing of the circuit and is also optimized by the sizer.

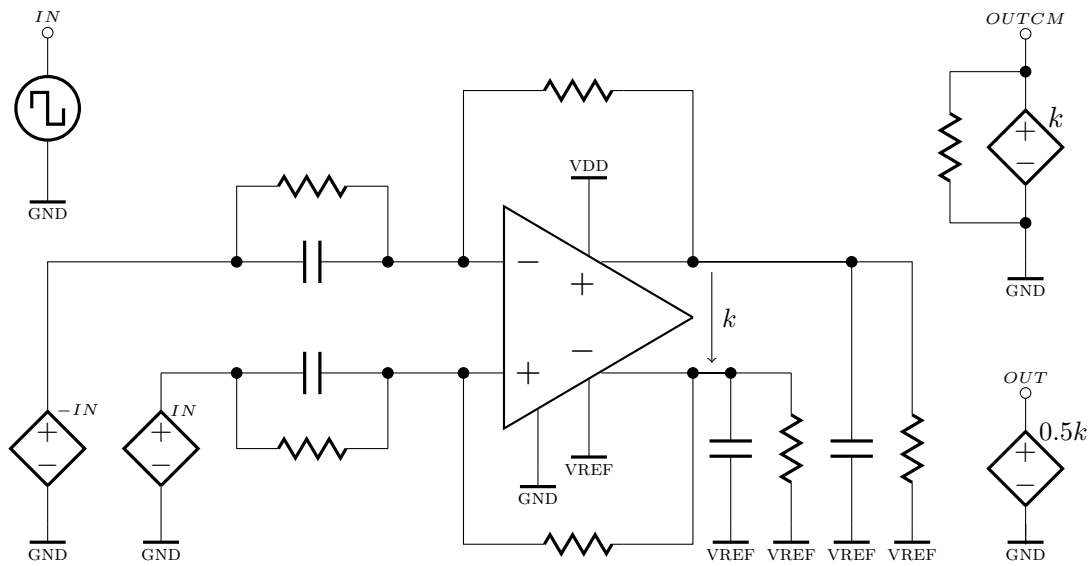


Figure 8.11: Unity testbench

### 8.2.5 Sallen-Key 2nd-Order Low-Pass Filter (SKLP)

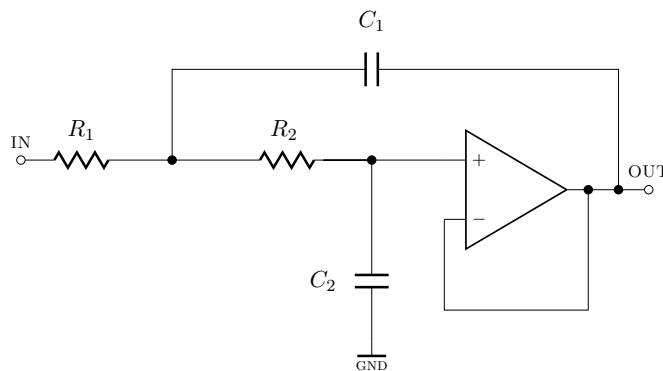


Figure 8.12: Classic implementation of the RC-network for a Sallen-Key filter architecture.

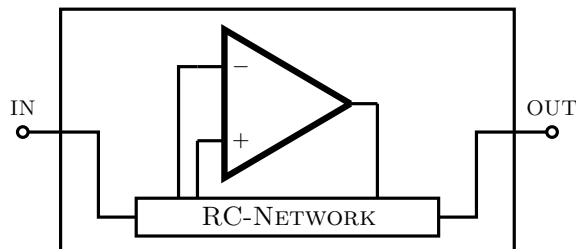
This circuit template increases the hierarchy depth by one and thus demonstrates how the framework may be used to synthesize filters in general.

The circuit shown in Figure 8.12 is well known among analog experts as the Sallen-Key realization of a 2nd-order low-pass filter. It is surely one of the less complex filter implementations known, but still used fairly often in analog design. The inherently low passband ripple and the small area requirements, due to the fact that only one OP has to be used, are its important strengths. Audio applications, anti-aliasing filters for ADC inputs and AC-DC converters are possibly the most prominent ones.

Inside the synthesis framework the synthesis of this circuit template is done in multiple steps. First a set of sized OPs is synthesized using the embedded OP circuit template, which is described in Figure 8.7. Therefore a set of specifications for the OP also have to be provided,

but they should not be seen as the final specifications. Moreover, they should be quite relaxed and in particular the user should use these to control the amount of different OPs, which will later be considered inside the top-level circuit template sizing step.

### SALLEN-KEY 2ND-ORDER LOW-PASS FILTER



TESTBENCH	SPECIFICATION	PORTS		
Operating Point	Offset	IN	Input	Voltage
	Power	OUT	Output	Voltage
	Area			
Open Loop	Gain	RULES		
	Passbandripple	CTR	BLR(= 3)	
	Stopbandripple	TEMPLATES		
	Cutoff frequency	Single-ended operational amplifier		
	Signal-to-Noise ratio	Sallen-Key 2nd-order RC-Network		
PSRR	PSRR	OPERATING CONDITIONS		
Unity	Output voltage swing	VDD	Supply voltage	
	Overshoot (fall/rise)	VREF	1/2 Supply voltage	
	Gain (lower/uppper)	IVR	Input voltage range	
Distort	Total harmonic distortion	CLOAD	Capacitive load	
		RLOAD	Resistive load	

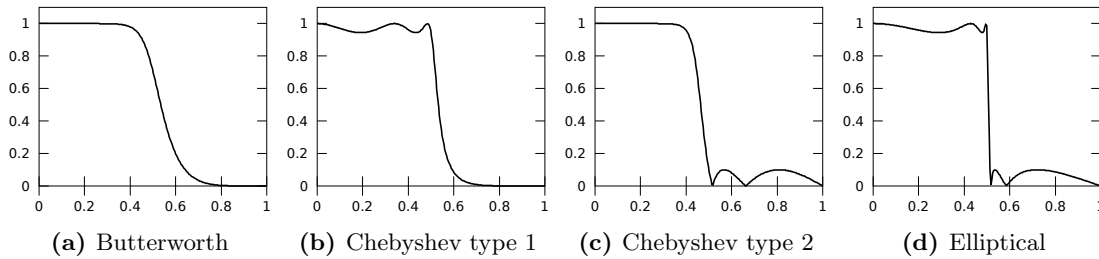
**Figure 8.13:** Sallen-Key 2nd-Order Low-Pass Filter

Once the framework delivers a set of OPs, it returns back one hierarchy to the top-level circuit template of the SKLP. At this point each of the sized OPs are inserted into the top-level circuit and passed again to the sizer to finally synthesize the SKLP. The previously synthesized OPs are used with their sizing as initial values for all design variables, but still allowing the sizer to further improve the underlying OPs sizing. This is a rather uncommon approach as usually the OPs used inside SKLPs are not further optimized inside the used RC-network. In particular this leads to an improvement of the top-level specifications. Furthermore it is important to mention that the RC-network itself is not touched by the sizer as the appropriate values for the resistors and capacities may easily be calculated using standard filter design methods or any of the widely available tools for filter design. Although it would be possible to allow the sizer to change the RC-network, this is undesirable—changing the outer circuitry would lead to

an *undefined* filter, which could expose an undefined behavior and thus reduce the necessary trustworthiness inherent to analog circuits.

### 8.2.6 Elliptical 3rd-Order Low-Pass Filter (ELIPLP)

This circuit template serves as *the* most complex example investigated for the presentation of the technical capabilities of FEATS. It brings together all of the previously illustrated concepts, utilities, and nearly all realized features.



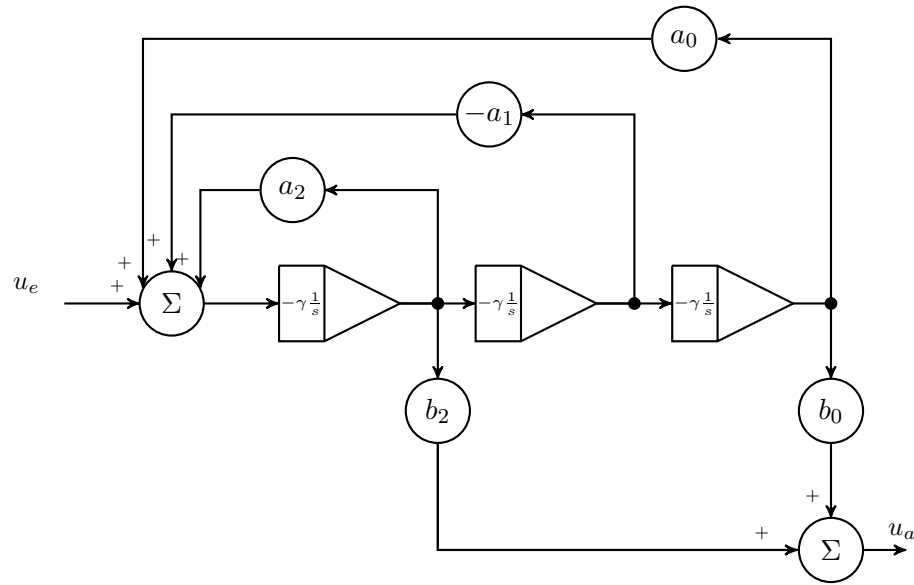
**Figure 8.14:** Typical linear filter transfer functions (source [Act], license [CCB], changed).

Elliptical filters, sometimes also called Cauer filters, intrinsically exhibit various distinctive properties, which render them as very useful for applications like channel select filters. Chebyshev filters allow ripples in the pass band, thus they are able to obtain a better selectivity compared to Butterworth filters. The elliptical filter further improves this by allowing ripples in the stop band. This enables the elliptical filter to be the sharpest compared to all other common filter architectures as shown in Figure 8.14.

The here presented circuit template is intentionally inspired by recent scientific work published by Krishnapur et al. [KAS11]—where a 3rd-order elliptical filter was designed and measured for WiFi applications inside the 2.4GHz band. The aim of this work is not to compete with the published design, but moreover to generate a certain level of comparability, in order to allow the reader to estimate the practical impact of the synthesis framework on his own.

Despite the structural and scaling related challenges, designing an ELIPLP introduces major electrical engineering difficulties, which may not be underestimated. The first and most obvious one is the inherent need for FDAs as the underlying amplifiers. Secondly, a robust FDA design does not automatically lead to an overall functional ELIPLP. In particular the stability paired with the desired fundamental electrical properties force the user to design the target testbenches and measurements with great care and experience—but apparently this work has only to be done once.

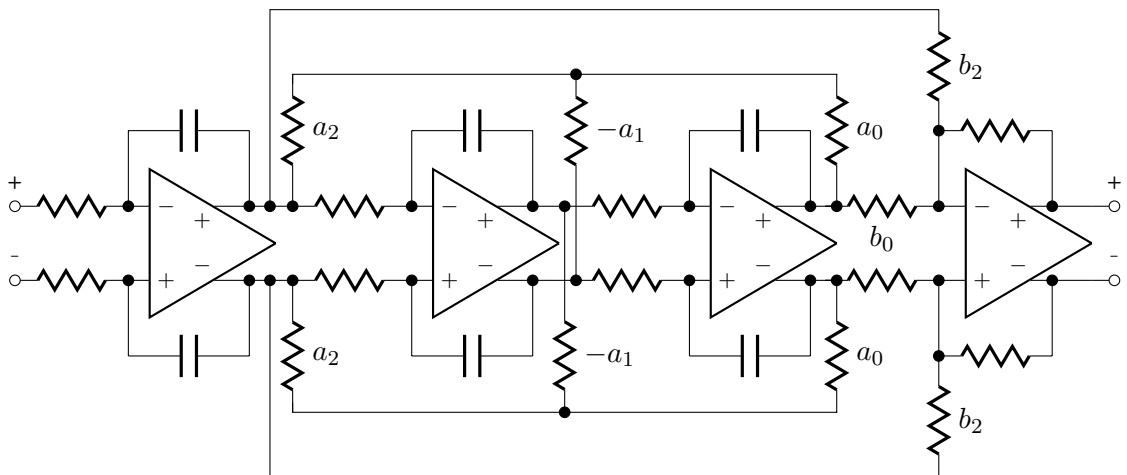
The block-level diagram as shown in Figure 8.15 may be taken from the textbook and can, similar to the SKLP, afterwards be used to calculate the RC-network used in the actual transistor level implementation of the circuit. Thus, as before, the RC-network is not subject to the optimization. The summation factors are calculated according to the desired cutoff frequency and are from thereon given and constant. While this has been done upfront using a computer algebra system, it would be unimaginable to realize such a calculation as an analyzing rule (see Section 5.5.4) in order to fully automate the RC-network creation. This simple addition would



**Figure 8.15:** The block-level diagram of the synthesized elliptical 3rd-order filter visualizing the conceptual architecture and the involved filter coefficients.

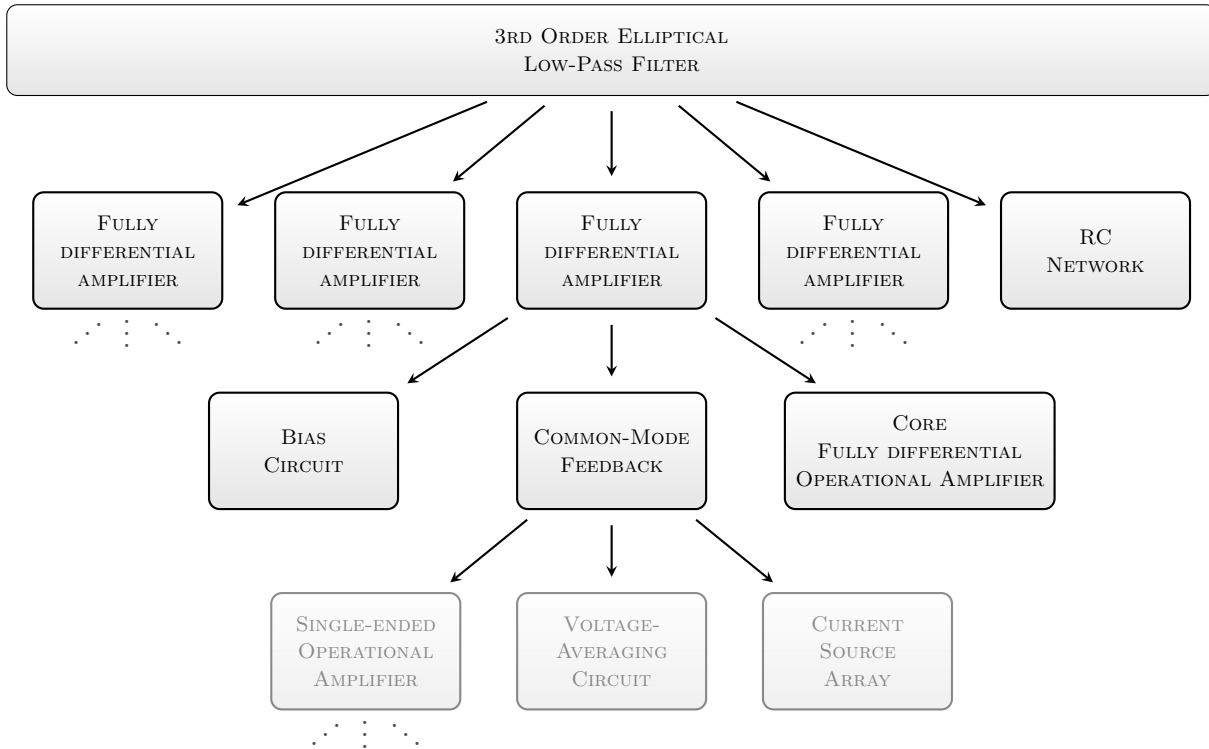
enable the framework to synthesize an ELIPLP for any given cutoff frequency—furthermore a design variable representing a scaling factor for the RC-network could be introduced to optimize the RC-network towards e.g., minimal area, power dissipation, or any other available performance as shown in Figure 8.18.

Based on the block-level diagram shown, the actual filter configuration may be deduced. It consists of three FDAs as integrators and a single FDA as summing amplifier. The schematic shown in Figure 8.16 shows the result and is—in terms of hierarchy—identical with the circuit template shown in Figure 8.18.



**Figure 8.16:** The ELIPLP configuration deduced from Figure 8.15, including the summation factors.

Figure 8.16 showcases the complexity and large scale of the here targeted circuit (class). The support for *arbitrary levels of hierarchies* is one of the key features of the presented framework—without a consistent hierarchical concept, a circuit of this scale could not be synthesized automatically. Despite the latter, hierarchies are very useful in delivering trustful circuits (see Section 3.1). Trust and thus a conceptual understanding of the delivered circuit is nowadays one of the most important requirements—not only for an analog synthesis tool, but moreover for almost every EDA software.

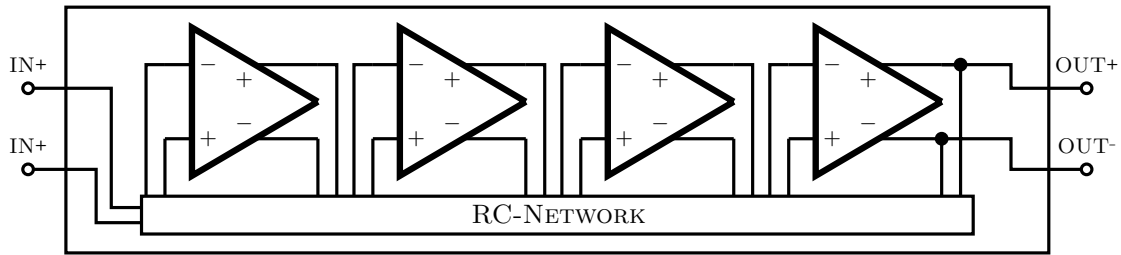


**Figure 8.17:** Visualization of the various hierarchy levels during the synthesis of an ELIPLP. To increase readability the levels below the FDAs are only showed once.

FEATS realizes a very consistent hierarchy, which allows the user to easily introduce new circuit classes. But more importantly, the user may chose to include arbitrary circuit templates into other circuit templates *without any further modifications* of the included. This encourages the user to consequently divide the developed circuit into smaller parts and thus *reuse* already developed circuit templates. Looking closer at the Figure 8.17 reveals an hierarchical depth of *five*, if the common-mode feedback module is configured to also be hierarchical, instead of the quick and easy method—simply provide a basic block containing a whole common-mode feedback circuit as e.g., the one shown in Figure 8.6. Once again, the synthesis engineer has the possibility to precisely tweak the trade-off between *expert knowledge* and *number of generated circuits* as illustrated in Figure 1.1. E.g., a working CMFB stage may be available and thus be provided, but the framework may easily synthesize the CMFB with its full hierarchy.



## ELLIPTICAL 3RD-ORDER LOW-PASS FILTER



TESTBENCH	SPECIFICATION	PORTS	
Operating Point	Offset	IN+	Input Voltage
	Power	IN-	Input Voltage
	Area	OUT+	Output Voltage
Open Loop	Gain	OUT-	Output Voltage
	Gain bandwidth product	<b>RULES</b>	
	Phase margin	CTR	BLR(= 3)
	Transit frequency	<b>TEMPLATES</b>	
	Cutoff frequency	Full differential operational amplifier	
	Signal-to-Noise ratio	Elliptical 3rd-order RC-network	
	Position 1st Pole	<b>OPERATING CONDITIONS</b>	
CMRR	CMRR	VDD	Supply voltage
PSRR	PSRR	VREF	1/2 Supply voltage
		IVR	Input voltage range
Unity	Output voltage swing	CLOAD	Capacitive load
	Slew rate (fall/rise)	RLOAD	Resistive load
	Overshoot (fall/rise)		
	Settling time (fall/rise) (+)		
	Settling time (fall/rise) (-)		
IP3	IIP3		
Distort	Total harmonic distortion		
	1dB compression point		

**Figure 8.18:** Elliptical 3rd-Order Low-Pass Filter

The hierarchy for the ELIPLP as presented in Figure 8.17 illustrates the various levels—starting at the top-level circuit template (ELIPLP) the synthesis engine descends one level every time a circuit template is found, which contains other circuit templates as described in Section 5.2.3. The framework comes without any intrinsic limitations how many levels of hierarchy may be used inside a synthesis run. Furthermore, the *local references* (see Section 5.2.2) and *construction types* (see Section 5.2.3) become crucial for a successful synthesis. E.g., if the chosen construction type is *shared structure*, *shared scope* the synthesis engine descends only once into the FDAs circuit template—the now on demand synthesized circuits are inserted inside this hierarchy into all its occurrences.



## FIELDS OF APPLICATION FOR ANALOG SYNTHESIS

Inside this final chapter the focus is clearly set on the actual transistor level circuits generated by FEATS. Starting with a wide selection of OP circuit structures in Section 9.1 the following Section 9.3 shifts the focus towards a more complex circuit. A recent publication (see [KAS11]) was used as a reference to synthesize an ELIPLP comparable to the published one. In Section 9.2 a system design is presented, which is strongly supported by the framework to demonstrate the applicability and how beneficial a synthesis driven analog development flow may be in real usage scenarios. Eventually, this chapter closes with the inspection of the creative capabilities of the presented framework. Therefore an uncommon circuit structure—generated by the framework—is discussed in Section 9.4.

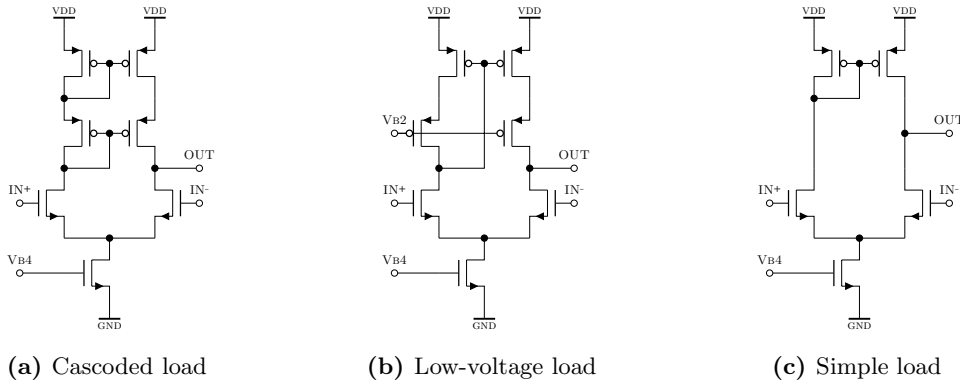
Despite the here chosen examples, the framework was used in various application scenarios—either to demonstrate the benefits of a synthesis driven analog design flow, or to exploit its distinctive explorative nature in order to e.g., analyze the impact of aging on differing OP structures as shown in [SHM13]. Other publications as [MH15, MMH12, vRMH15] can be studied to find further examples of synthesized circuits.

The presented results are based on a 350 nm CMOS bulk technology node from austriamicrosystems with a supply voltage of 3.3 V—the synthesis is carried out by an E5520 Xeon *workhorse*, providing 16 CPU cores and 36GB RAM to easily handle larger amounts of parallel processes. Granted, the CPU generation is from the late 2009’s—it is still very well suited for circuit synthesis. Nonetheless, this *infrastructure* is rather tiny, compared to an industrial scale server (infrastructure).

### 9.1 Single-ended Operational Amplifiers

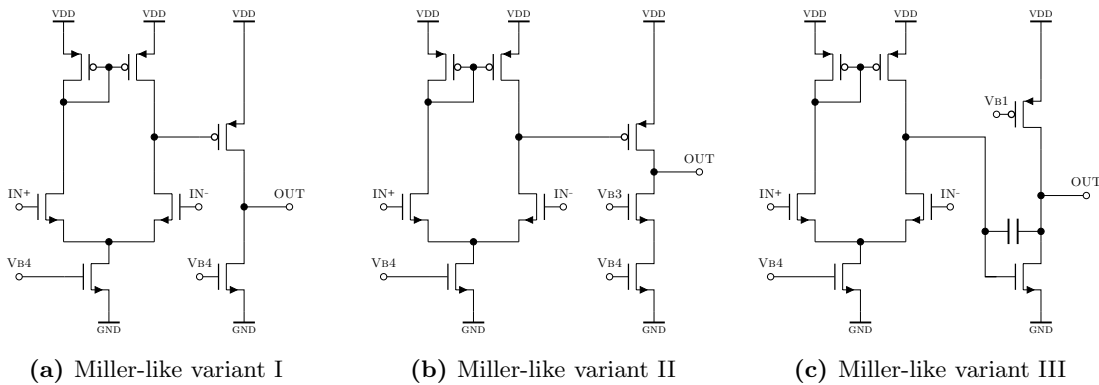
As already mentioned in Section 8.2.1 the OP in general features a wide variety of structures dependent on the target process technology and/or the field of application. The presented framework is able to synthesize the *average* textbook circuit out of the box. Textbooks tend to describe concepts rather than all transistor level realizations of the concept—following this idea

slightly here, the presented circuits resemble just a small portion of what is actually synthesized by the engine.



**Figure 9.1:** nMOS differential stages variants.

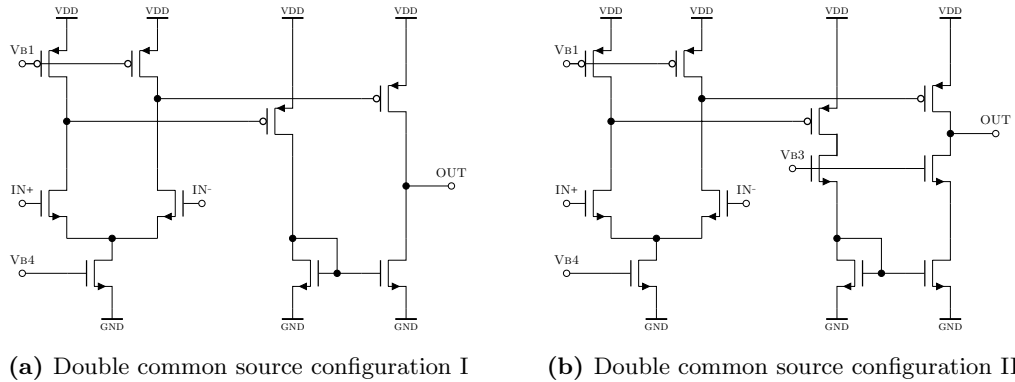
Already the basic differential stage exhibits several differing transistor level implementations. Figure 9.1 shows three variants using an nMOS differential pair. Obviously, the complementary pMOS variants are also generated, leading to at least six variants. Further variants are imaginable e.g., a cascoded current source or other loads. Trying all of these variants in manual design could already be very daunting—but in fact this is a tiny portion of the actual *design space* as described in Chapter 4. To be precise, the structure space (see Chapter 4) containing the circuit from Figure 9.1c, for the five components and thus 15 ports, has 1,382,958,545 other members (see Table 4.1). An analog designer, and especially the framework manage to implicitly discard the majority of these in order to generate the targeted using the mechanisms discussed in Chapter 5 and Chapter 6.



**Figure 9.2:** Various nMOS miller variants.

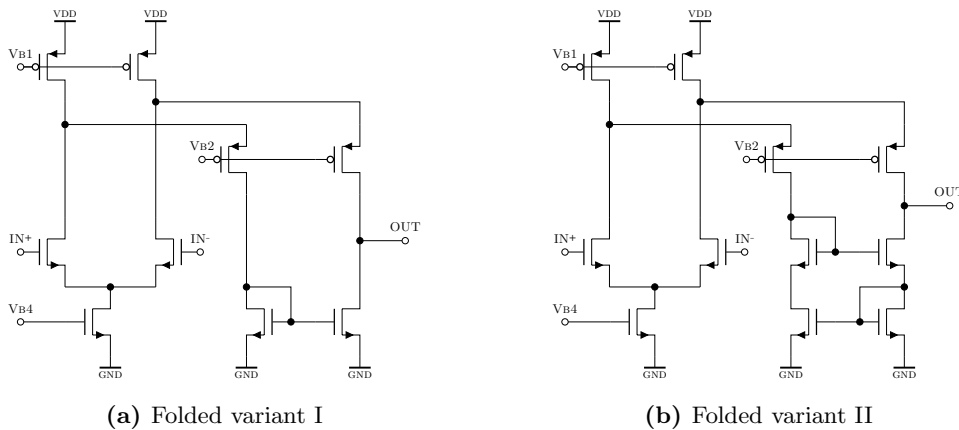
Another widely known concept is the miller amplifier, which formally simply adds a nMOS or pMOS transistor in common source configuration and a current source or sink, in order to boost the maximum gain of the amplifier. Figure 9.2 shows different realizations of this concept with identical differential stages. At this point it is worth mentioning that each miller variant

shown in Figure 9.2 may be generated with every differential stage shown in Figure 9.1 and discussed thereafter.



**Figure 9.3:** Various nMOS double common source variants.

The circuits shown in Figure 9.3 could easily be confused with the ones shown in Figure 9.4. The latter are mostly known as folded (cascode) circuits and especially useful e.g., to increase the output voltage range. Even though the circuits shown in Figure 9.3 share obvious structural similarities with the ones from Figure 9.4, the *double common source configuration* is usually not found in analog literature—maybe due to the inherently limited power supply noise rejection.



**Figure 9.4:** Various nMOS *folded* variants.

Another popular circuit structure regularly used in OP and OTA applications is to be found in Figure 9.5—mostly also named OTA. The structure allows a lot variations by choosing different current mirror realizations.

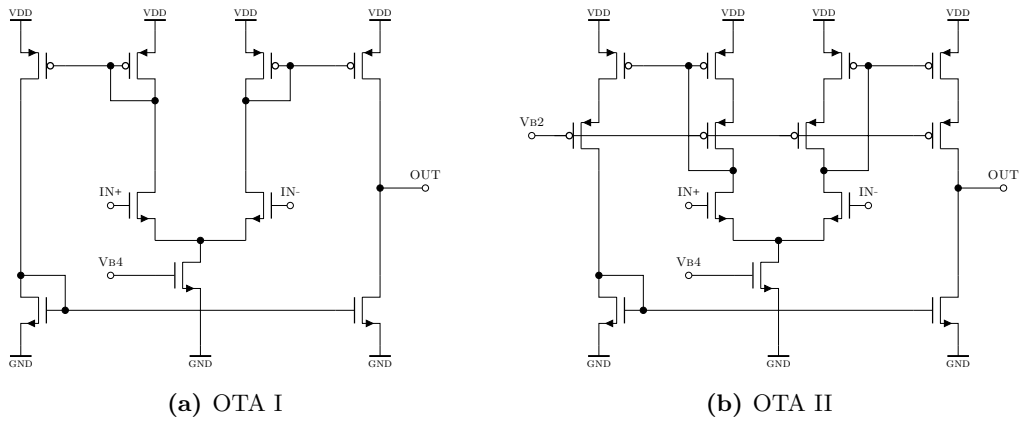


Figure 9.5: Various OTA realizations.

FEATS generates all of the above circuits by using the basic blocks shown in Appendix B and the OP circuit template (see Section 8.2.1). Despite these, the synthesis engine generates *all possible* variants and furthermore mixes, supersets, and subsets of the previously presented circuits.

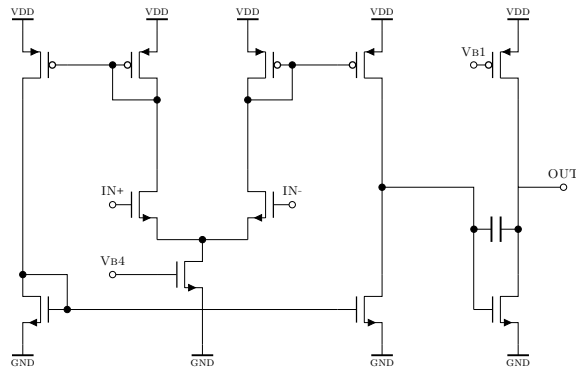


Figure 9.6: OTA mixed with a common source stage as driver.

An example of such a mix of concepts can be seen in Figure 9.6, where an OTA circuit (see Figure 9.5) is connected to a common source stage as to be found in a miller configuration as shown in Figure 9.2. This leads to sometimes uncommon combinations, which surprisingly perform not as bad as one might expect. Finally, Table 9.1 provides some topology and circuit counts—*mentioned* circuits are all within the enumerated circuits. The two rightmost columns in Table 9.1 describe how many circuits *survived*, i.e., were not discarded during the circuit analysis in Section 5.4.3.

TABLE 9.1  
GENERATED OP CIRCUITS FOR DIFFERENT MAXIMUM BLOCK THRESHOLD VALUES (NOT CUMULATIVE).

No. BLOCKS	TOPOLOGIES	CIRCUITS	ACCEPTED ISOMORPHISM	ACCEPTED GAIN	ACCEPTED FEASIBILITY
2	0	0	0	0	0
3	4	24	18	6	6
4	12	144	108	64	60
5	36	600	449	182	168
6	112	2,424	1,829	802	754
7	324	9,072	6,879	3,802	2,938

## 9.2 Application Inside a System Design

During the development and realization of a so-called *analog artificial hormone system* (AAHS), as presented in [vRSH<sup>+</sup>15], the opportunity was taken to realize the system in two different versions. One driven by voltage and thus using OPs as the critical system components and another, current-driven version, which builds its functionality with the foundation of OTAs. Although the circuit structures for OTAs do not differ fundamentally compared to OPs, the electrical properties do significantly. Described through the specification and measured with the appropriate testbenches. The expandability of FEATS, allowed the synthesis of the OTA circuits within a few hours.

TABLE 9.2  
SYNTHESIZED OPs FOR THE AAHS.

SPECIFICATIONS	TARGETS	OUTER ADDER	INNER ADDER
$R_{Load}$	=	12.5 k $\Omega$	100 k $\Omega$
$C_{Load}$	=	10 fF	2.5 pF
Gain	$\geq$ 26.2 dB	35.2 dB	27.3 dB
Output voltage range	$\geq$ 1.95 V	2.48 V	2.6 V
Overshoot	$\leq$ 0.03 %	0.017 %	0.015 %
Undershoot	$\leq$ 0.03 %	0.009 %	0.027 %
Slew Rate (fall)	$\leq$ 27.5 V $\mu$ s <sup>-1</sup> $\geq$ 27.5 V $\mu$ s <sup>-1</sup>	14.1 V $\mu$ s <sup>-1</sup>	74.78 V $\mu$ s <sup>-1</sup>
Slew Rate (rise)	$\leq$ 27.5 V $\mu$ s <sup>-1</sup> $\geq$ 27.5 V $\mu$ s <sup>-1</sup>	13.6 V $\mu$ s <sup>-1</sup>	64.45 V $\mu$ s <sup>-1</sup>
Offset	$\leq$ $\pm$ 40.3 mV	36.6 $\mu$ V	-2.06 $\mu$ V
Phase margin	$\geq$ 35 $^\circ$	70.8 $^\circ$	66.1 $^\circ$

Notably, this was the first approach to synthesize circuits for a system, including the generation, importing them into a state-of-the-art analog design environment, and finally produce them on silicon. One of the major advantages was the ability to synthesize the OPs respectively OTAs in an early stage of system development. The system designer gets an early impression, if the realization of the system with the chosen specifications for the underlying OPs and OTAs is feasible—within one day all needed configurations of e.g., the OPs were delivered. This allows to go through several iterations of the whole system design (using different specifications) within one week. Eventually, two different types of OPs (see Table 9.2) were synthesized for the final voltage driven design. Furthermore, three different OTA structures (see Table 9.3) are part of the current driven design—eight OPs, respectively 14 OTAs were finally produced on silicon. The produced silicon performed in predicted boundaries on all (30) measured samples.

TABLE 9.3  
TWO OUT OF THREE SYNTHESIZED OTAs FOR THE AAHS.

SPECIFICATIONS	TARGETS	MEASURE OTA	TARGETS	RES. OTA
$R_{Load}$	=	102.2 k $\Omega$	=	55.2 k $\Omega$
$C_{Load}$	=	500 fF	=	500 fF
Transconductance	$\geq$ 8.9 $\mu$ S $\leq$ 10.1 $\mu$ S	10.0 $\mu$ S	$\geq$ 16.4 $\mu$ S $\leq$ 20.0 $\mu$ S	18.1 $\mu$ S
Slew Rate (fall)	$\leq$ 27.5 V $\mu$ s $^{-1}$	0.41 V $\mu$ s $^{-1}$	$\geq$ 27.5 V $\mu$ s $^{-1}$	80.9 V $\mu$ s $^{-1}$
Slew Rate (rise)	$\leq$ 27.5 V $\mu$ s $^{-1}$	0.42 V $\mu$ s $^{-1}$	$\geq$ 27.5 V $\mu$ s $^{-1}$	80.8 V $\mu$ s $^{-1}$
Input voltage range	$\geq$ 1.1 V	1.18 V	$\geq$ 1.1 V	1.4 V
Output resistance	$\geq$ 7.7 M $\Omega$	40.0 M $\Omega$	$\geq$ 4.2 M $\Omega$	15.2 M $\Omega$
Offset	$\leq$ $\pm$ 450 $\mu$ V	90 $\mu$ V	$\leq$ $\pm$ 240 $\mu$ V	-9.5 $\mu$ V
Phase margin	$\geq$ 35 $^\circ$	81.4 $^\circ$	$\geq$ 35 $^\circ$	68.9 $^\circ$

### 9.3 Elliptical 3rd-Order Low-Pass Filters

As stated in Figure 8.17, the elliptical 3rd-order low-pass filter serves as the most complex example investigated for this thesis. Once the development of the testbenches, the formulation of the measurements, and the overall setup for the framework is finished, the synthesis may be started in order to generate the target circuit fulfilling the required specifications. For this particular circuit this process involves the synthesis of FDAs, which is carried out by the synthesis engine, fully unattended, after descending one level down in the hierarchy, as illustrated in Figure 8.17.

The FDAs are synthesized on-the-fly and are afterwards inserted into the current hierarchy level after ascending back up—once the synthesis engine finished the FDA synthesis. Based on the available time, resources and layout capacities the user may choose one of the possible circuit template *construction types* as described in Section 5.2.3. As their naming already



implies, the impact on the overall runtime of the synthesis run may be severe. But once again, the user gets the full control and has to decide (how much *circuits should be generated*), either be conservative: get a *good*, sized circuit very fast (shared structure, shared scope) and save layout time by having only a single FDA—or aim for a highly optimized, application specific circuit and therefore handle each FDA as a distinct, unrelated circuit—each one with its own design variables and thus the possibility to be further optimized in the context of the surrounding circuitry (unique structure, private scope).

The here presented examples aim to provide a certain level of comparability—given the realization of an identically specified elliptical 3rd-order low-pass filter from [KAS11], the reader essentially gets all needed information to determine the applicability of the approach by himself. To maintain comparability, the last of the three construction types (see Section 5.2.3) is used exclusively during this analysis. Therefore exactly one FDA is instantiated in each available FDA slot inside the ELIPLP circuit template. Once this has been done for each available FDA, an equal number of ELIPLP transistor level circuits has to be further evaluated—i.e., using this *construction type* will generate as much ELIPLP candidates as there are *good* FDA circuits.

The sizing tool now simulates and optimizes the whole circuit, which in numbers means the optimization of an *analog system* containing at least 200 devices, in particular the synthesis framework evaluates multiple of them in parallel with ease, due to the asynchronous TCP/IP driven task allocation system, as described in Chapter 7,

The parameter space is fortunately by far smaller as for an unknown 200 device circuit. The constraints propagated from bottom, i.e., from the components as described in Section 5.2.1, together with the ones introduced through the used basic blocks (see Section 5.2.2), which get interconnected using the symmetric elementary electric rule (see Constructive rule 3), allow the synthesis engine to further deduct constraints and assign identical design variables for symmetrically placed basic blocks. Eventually, this leads to less than 20 design variables for a circuit as shown in Figure 9.7, including the not shown parts inside the schematic, namely the bias circuit and the common-mode feedback circuit.

TABLE 9.4  
GENERATED FDA CIRCUITS FOR DIFFERENT MAXIMUM BLOCK SIZES.

NO. BLOCKS	TOPOLOGIES	CIRCUITS	ACCEPTED ISOMORPHISM	ACCEPTED GAIN	ACCEPTED FEASIBILITY
2	2	4	4	2	2
3	8	32	30	15	15
4	20	148	132	67	63
5	50	612	557	288	280
6	126	2,240	2,056	1,005	986

Given the assumption the ELIPLP consists of four structurally equal FDAs, the following sizing would only include the parameter space of the FDA integrated into the ELIPLP, thus each parameter change is directly applied to all four instances of the used FDA.

TABLE 9.5

OVERVIEW OF THE CIRCUIT COUNTS AFTER THE ASCENSION BACK TO THE TOP-LEVEL.

No. BLOCKS	FDAS SIZED	SUCCESSFUL FDAS	ELIPLPs SIZED	SUCCESSFUL ELIPLPs
4	63	8	8	2

As shown in Figure 8.17 the synthesis engine starts the synthesis with the ELIPLP circuit class itself. From thereon it descends in hierarchy in order to synthesize the parts necessary to assemble the ELIPLP. Table 9.4 shows the circuit counts generated during the synthesis run. The table should be read from left to right, with the rightmost column being the one resembling the number of circuits passed to the evaluation respectively sizing step.

TABLE 9.6

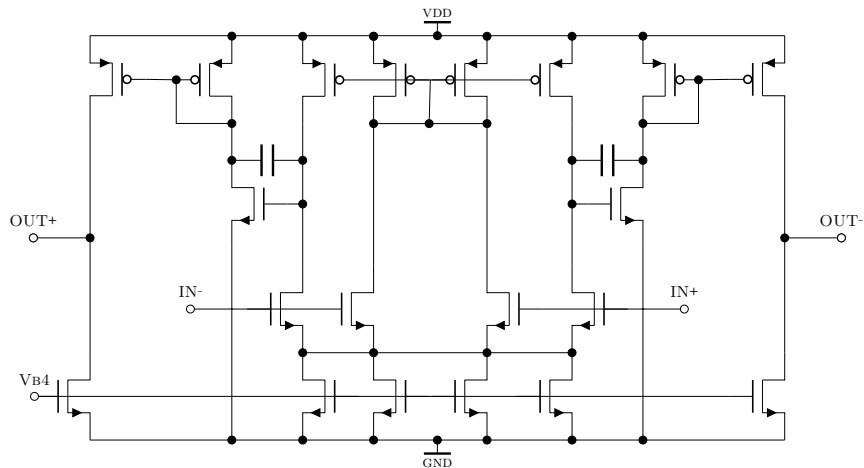
SPECIFICATIONS AND REACHED PERFORMANCES FOR A SUCCESSFULLY SIZED ELLIPTICAL FILTER AND A FAILED VARIANT.

SPECIFICATIONS	TARGETS	GOOD VARIANT	FAILED VARIANT
Filter type	3rd. order Elliptic		
Technology	0.35 $\mu$ m CMOS/3.3V		
$R_{Load}$	=	150 k $\Omega$	
$C_{Load}$	=	1 pF	
Gain	$\approx$ 0 dB	-0.05 dB	-0.16 dB
Cutoff Frequency	$\approx$ 8.5 MHz	8.60 MHz	8.59 MHz
Pass band ripple	$\leq$ 2.14 dB	2.13 dB	2.36 dB
Stop band ripple	$\leq$ -32.0 dB	-32.7 dB	-31.47 dB
SNDR	$\geq$ 40 dB	41.6 dB	43.4 dB
IIP3	$\geq$ 30 dBm	31.5 dBm	41.7 dBm
(Active) area		0.123 mm	0.085 mm
Power		22.4 mW	282 mW

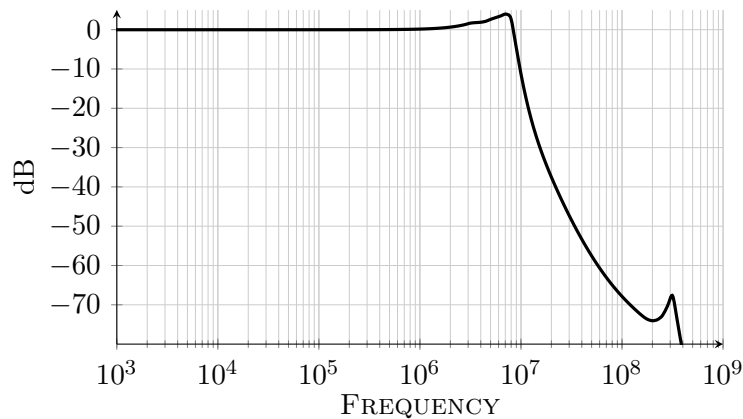
For the presented example a maximum block count of four was chosen and lead to the generation of eight successfully sized FDAs (see Table 9.5), which were then inserted into the top-level circuit template to pass it into the final sizing step.

In Figure 9.7 one of the two FDAs, which are used inside the two successful sized ELIPLPs is shown. This clearly shows that not every successful FDA may also lead to a successful ELIPLP. An example of such a case is shown in the right column inside Table 9.6, which slightly breaks the needed specifications.

Nevertheless, the final ELIP exhibits the typical behavior of an elliptical filter as to be seen in Figure 9.8. Compared to the filter as presented in [KAS11] the here synthesized ELIPLP exhibits mostly comparable performances. The power consumption is the only performance, which differs massively. But this is obviously to be explained with the used process node. The



**Figure 9.7:** The fully differential operational amplifier synthesized inside the elliptical filter



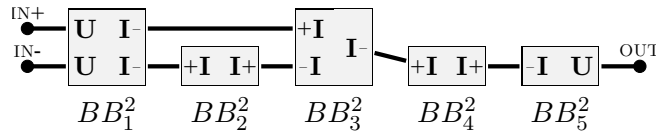
**Figure 9.8:** Simulated nominal transfer function of the synthesized elliptical filter.

process nodes are several generations away from each other, thus an inherent higher power dissipation will be observable. Despite this, [KAS11] does not include any information about the used area of the RC-network—as a larger area translates to less power dissipation in this context, a comparison of the power dissipation may not be applied without further information.

## 9.4 Computed Creativity—Current Stealer Design Pattern

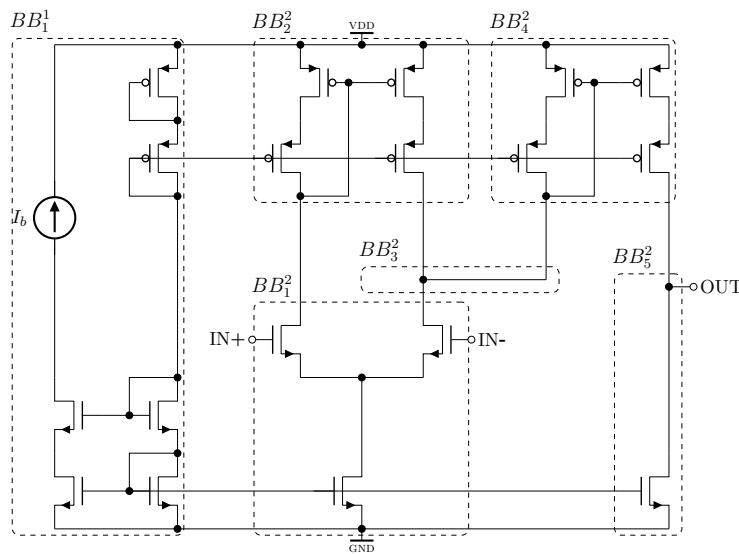
The here presented circuit family is very unfamiliar, even for most people with a decent degree of knowledge in analog design. This design pattern violates one of the most basic design principles found in analog, or more precise, amplifier design. Instead of connecting a high impedance (gate) node to an existing amplifier stage, a low impedance node is appended, which furthermore does not even have a complementary device leading to the naming *current stealer* respectively *current injector*. This asymmetric structure has already caused lively discussions among analog experts, trying to explain the electrical behavior in detail.

Despite the interesting analysis of the underlying electrical concepts, this circuit demonstrates the creative capabilities of FEATS in an accessible and believable fashion. Although this circuit will very likely not be the big leap in analog amplifier design, it allows a peak into the undiscovered areas of the analog design space—its huge size, as shown in Chapter 4, surely hides some undiscovered, useful circuit structures.



**Figure 9.9:** An uncommon operational amplifier circuit structure realizing the current stealing design pattern.

Remarkably, the circuit shown in Figure 9.10 does neither eat up enormous computing resources nor it consists of exotic, or highly sophisticated basic blocks. In fact, by simply replacing the default current merge block (see Appendix B) with the current merge block exhibiting a current port as output instead of a voltage port, the synthesis engine generates circuits based on the current stealer/injector design pattern exclusively. Showcasing the promoted creative freedom, without losing the ability to precisely control the generated quality and quantity of the generated circuits.



**Figure 9.10:** A current stealer circuit annotated with the used basic blocks, for an easy matching with the topology shown in Figure 9.9.

One might be inclined to argue, this circuit could be used in actual designs, e.g., a technology node could be utilized, which provides a much better pMOS or nMOS device compared to its complementing device. An outlook for which applications this could be applied is discussed in Section 10.1.

## CONCLUSION

The synthesis framework presented in this thesis aims to push analog synthesis to the next level by addressing the following important challenges in analog circuit synthesis.

- *Predictability* due to deterministic synthesis algorithms
- Clear, user-defined, *arbitrary depth hierarchies* in the form of *circuit templates*
- User-defined *basic blocks*, enriched with electrical and structural constraints propagated from bottom to top
- Loose binding for the used sizing tool to realize its *expandability* and *changeability*.
- *Fast circuit preselection* trading precision for speed in order to only pass the most promising circuits to the time and cost intensive sizing step.

The presented framework is therefore described in great detail in order to allow the reader to fully understand the internals. First illuminating the necessary inputs, these are especially important to understand the realized hierarchies and their interconnections. Once this is known, the synthesis engine itself may be investigated further, in particular the algorithm is described in detail and the so-called *rules*, which represent the various operations on the topologies respectively circuits are illuminated.

The aforementioned fast preselection algorithms are afterwards analyzed in depth. The framework utilizes a fast circuit isomorphism, necessary due to the basic blocks, which inherently may lead to isomorphic circuits. Furthermore, the framework provides tools to approximate the circuit's feasibility and a potential gain inversion.

The circuits designated for further evaluation are then passed on to the sizing step. This part of the framework aims to provide the methods to utilize any 3rd party software for the evaluation of the circuits. Therefore a *task manager* was developed to asynchronously distribute the tasks among the available *application servers*, which encapsulate the 3rd party software and provide the possibility to be controlled by the task manager across the network using TCP/IP, which enables the easy scaling of the framework.

The presentation of the results starts with the used circuit templates. These circuit templates realize the hierarchies inside the framework and have to be developed upfront. Starting with simple utility circuit templates the complexity is increased step by step. The most complex one is a 3rd-order elliptical low-pass filter.

A selection of use cases and example circuit synthesis runs is finally presented. Starting from a classic single-ended operational amplifier, followed by a 3rd-order elliptical filter, which is in fact the biggest (in terms of devices) that has been synthesized so far using the framework. The latter demonstrates the framework’s capabilities in the context of another recent scientific publication. The overall runtime to synthesize such a complex circuit is below 24 h from specification to a ready for layout transistor level circuit.

In general this thesis aims to showcase that analog transistor-level circuit synthesis can be used during the analog design process for various beneficial reasons.

- Obtaining a design space exploration to collect information about the hardness of the chosen specifications.
- Prototyping a whole system can be done within a fraction of the time needed to design the circuits by hand.
- *Everyday circuits* (e.g., single-ended operational amplifiers) may be synthesized within several hours.
- System feasibility studies may be carried out within days including different specifications.

Once set up the synthesis runs fully unattended, without any user interaction, easily distributed on even heterogeneous server architectures. The analog designer may precisely tweak the quality and quantity of the generated circuits to exactly explore the aimed design-space, whilst still having the opportunity to go for a brute-force approach and inspect hundreds of circuits.

Essentially, the analog designer gets a tool to deliver higher quality circuits, which reduce the uncertainty during an analog design, while consuming less time than a single hand made design. The results demonstrate how beneficial such an approach can be and how the presented methodology can deliver the previously promised flexibility.

## 10.1 Outlook

In the future the main challenges in analog design will inevitably be the ongoing divergence in technology nodes. In particular, there will be technology nodes, which exhibit strongly heterogeneous elementary devices. In fact such technology nodes already exist—organic thin film transistors (OTFTs) are subject to various researchers, good overviews may be found in [Fac07, RRmLB04, Dod06]. Even organic field-effect transistors (OFETs) as shown in [MLWI08] exactly exhibit this property. The nMOS devices in *organic*-based technologies usually have a very small charge carrier mobility of around  $0.1\text{cm}^2/\text{Vs}$ , while pMOS devices easily reach values 100+ times bigger than the nMOS’ charge carrier mobility. This leads to extremely uncommon

design approaches and ideas, as in e.g.: [Dod06]—Uncommon and painful to adapt—a design from scratch is very time consuming. This situation perfectly suits to FEATS’s capabilities. As the *basic block*, respectively the *abstract basic block* concept may be populated with any circuit and/or device type. Regardless of technology node properties, multiple differing devices or even the absence of device types (see Section 5.2.1 and Section 5.2.2) the framework will explore the *design space* as spanned by the available components.

Similar methodologies could also be mandatory for future carbon nanotube (CNT) semiconductor technologies, originating in significant process variations (as described in e.g.: [APRG11]), which further increase the already daunting process variations of recent sub-100nm technology nodes. Silicon technologies have already reached their climax and the question is not *if* silicon will have a competitor, but moreover *when and who it will be*—nevertheless one is assured, the complexity to design and manufacture an (analog) electric circuit will, most likely, not decrease in future technologies.

Despite the aforementioned industrial applications that will increase the need for analog synthesis there is a very wide field for future research based on the proposed synthesis framework:

**Utilizing yield optimization** by introducing the local and/or global mismatch parameters into the component concept and extending the templates used for the netlist generation each circuit could not only be nominal optimized, but moreover a yield optimization, as provided by WiCkeD could be carried out.

**Classifying synthesized circuits** could be accomplished by extending the circuit isomorphism to subcircuit isomorphism. This could provide information as: *circuits containing a specific implementation of a current mirror exhibit a high gain or bandwidth.*

**Construction types** as described in Section 5.2.3 could also be further investigated to create highly complex designs which are consequently tailored to have a minimal area.

**New rules** may easily be developed to realize common design steps, e.g.: dynamic insertion of compensation capacities, bias circuit generation according to the number of actually needed bias voltages, or the calculation of RC-networks to allow the synthesis of arbitrary filters out-of-the-box.

**Even more complex circuits** than the presented 3rd-order elliptical can now be realized making extensive use of the presented circuit template concept.

Nevertheless, one important issue towards a possible product based on FEATS is the integration of the same into a commercial analog development flow. Being a not to be underestimated issue, it is more an engineering task, which was already done before.





# Appendices



CIRCUIT CLASS TESTBENCHES

This appendix lists some of the testbenches that were developed for the analog synthesis framework. In particular they provide proof for not differing fundamentally from regular testbenches—despite being parametrized.

Single-Ended Operational Amplifier Testbenches

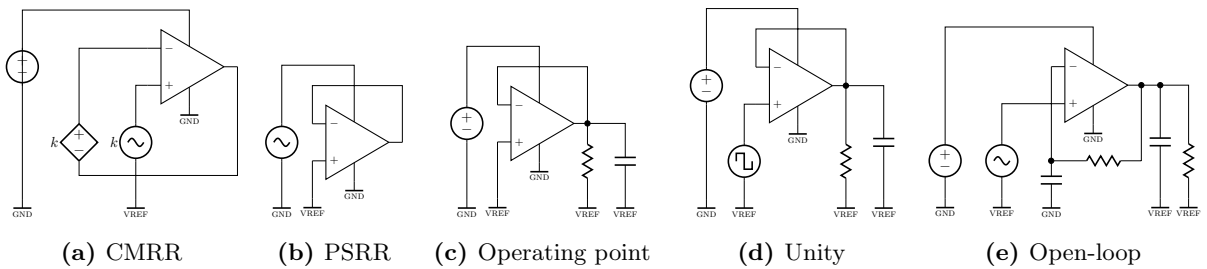


Figure A.1: OP testbenches

Fully Differential Operational Amplifier Testbenches

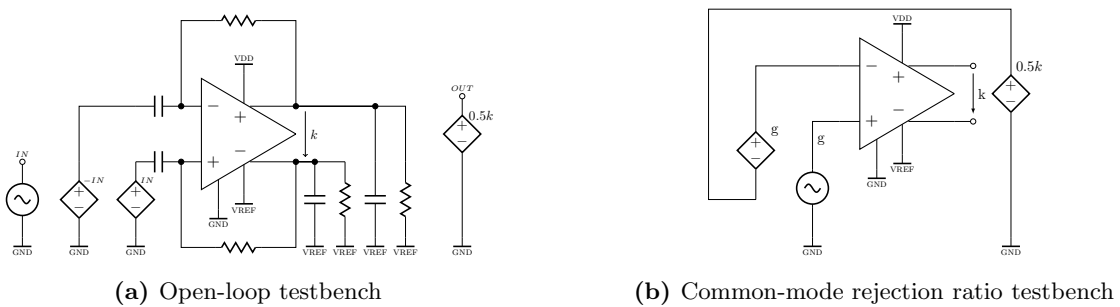
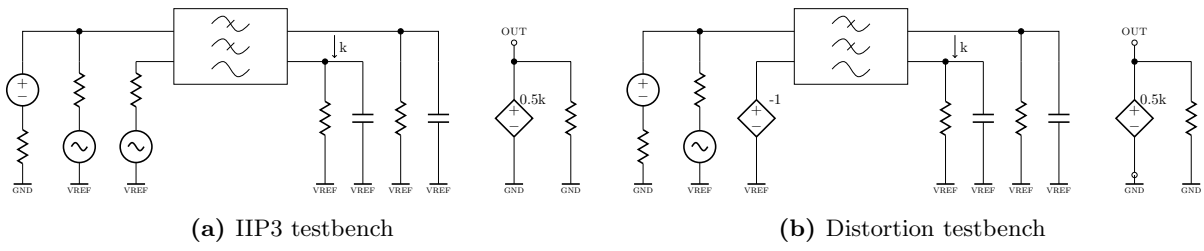
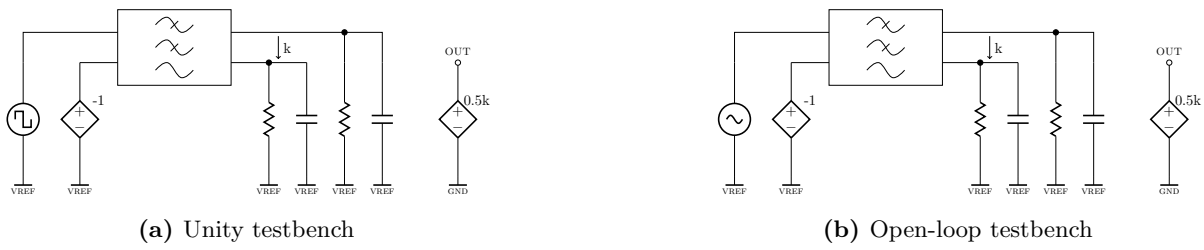


Figure A.2: FDA testbenches

## Active Filter Testbenches



**Figure A.3:** ELIPLP testbenches I

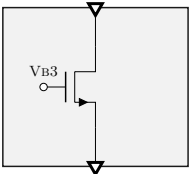
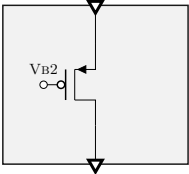
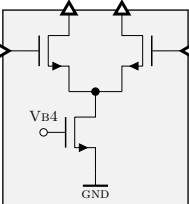
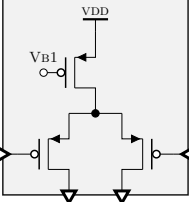
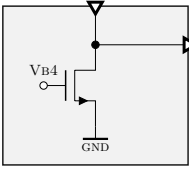
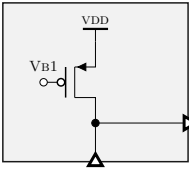
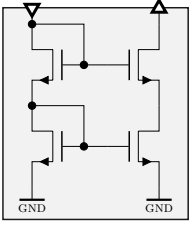


**Figure A.4:** ELIPLP testbenches II

BASIC BLOCK LIBRARIES

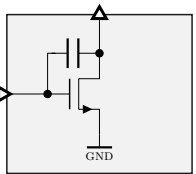
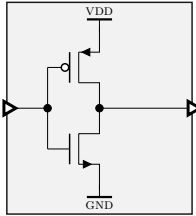
As discussed in Section 5.2.2 basic blocks serve as the elementary building block for circuits. Thus the choice of the *right* basic blocks is crucial for a successful synthesis. Basic blocks do not only provide actual transistor-level (sub-)circuits, but additionally add (electrical) constraints, which are propagated from bottom up to the top-level circuit to provide important hints for the preselection and for sizing.

	BASIC BLOCK	ABSTRACT BASIC BLOCK
NMOS CURRENT MIRROR		$+I \quad I+$
PMOS CURRENT MIRROR		$-I \quad I-$
NMOS COMMON SOURCE		$U \quad I+$
PMOS COMMON SOURCE		$U \quad I-$

	BASIC BLOCK	ABSTRACT BASIC BLOCK
nMOS CASCODE STAGE		$\boxed{+I \quad I-}$
pMOS CASCODE STAGE		$\boxed{+I \quad I-}$
nMOS DIFFERENTIAL PAIR		$\boxed{U \quad I-}$ $\boxed{U \quad I-}$
pMOS DIFFERENTIAL PAIR		$\boxed{U \quad I+}$ $\boxed{U \quad I+}$
nMOS CURRENT SOURCE		$\boxed{-I \quad U}$
pMOS CURRENT SOURCE		$\boxed{+I \quad U}$
nMOS CASCODED CURRENT MIRROR		$\boxed{-I \quad I-}$

	BASIC BLOCK	ABSTRACT BASIC BLOCK
PMOS CASCODED CURRENT MIRROR		$+I \quad I+$
PMOS FOLDING		$+I \quad I+$
NMOS FOLDING		$-I \quad I-$
NMOS LOW VOLTAGE CURRENT MIRROR		$-I \quad I-$
PMOS LOW VOLTAGE CURRENT MIRROR		$+I \quad I+$
NMOS COMMON SOURCE WITH CAPACITY		$U \quad I-$

LIST OF TABLES

	BASIC BLOCK	ABSTRACT BASIC BLOCK
PMOS COMMON SOURCE WITH CAPACITY		$\boxed{U \quad I^+}$
INVERTER		$\boxed{U \quad U}$



## LIST OF TABLES

4.1	Bell Numbers from 1 to 16 . . . . .	18
6.1	The Used Invariants $IV^i$ and a Selection of Their Properties $p \in IV^i$ for the Available Object Types. . . . .	42
9.1	Generated OP Circuits for Different Maximum Block Threshold Values (Not Cumulative). . . . .	85
9.2	Synthesized OPs for the AAHS. . . . .	85
9.3	Two Out of Three Synthesized OTAs for the AAHS. . . . .	86
9.4	Generated FDA Circuits for Different Maximum Block Sizes. . . . .	87
9.5	Overview of the Circuit Counts After the Ascension Back to the Top-level. . . .	88
9.6	Specifications and Reached Performances for a Successfully Sized Elliptical Filter and a Failed Variant. . . . .	88



## LIST OF FIGURES

1	Illustration des Zusammenhangs zwischen eingebrachtem Expertenwissen und der Anzahl der generierten Schaltungen. FEATS bezeichnet das hier vorgestellte Syntheseframework. . . . .	iv
2	Vollständiger Syntheseablauf mit allen wesentlichen Bestandteilen . . . . .	v
3	Illustration der asynchronen Aufgabendistribution. . . . .	vi
1.1	Illustration of the trade-off between expert-knowledge and circuit count. . . . .	1
1.2	A (simplified) illustration of the design flow for integrated circuits. . . . .	2
1.3	Simplified analog circuit design flow. . . . .	3
2.1	A circuit $c$ represented as a bipartite-graph $G$ . . . . .	8
3.1	Gajski-Kuhn Y-chart . . . . .	9
3.2	The typical—biological inspired—evolutionary algorithm flow. . . . .	12
3.3	Simplified illustrations of various used chromosomes for evolutionary synthesis. . . . .	13
4.1	An illustration of the degrees of freedom within a circuit development process. . . . .	18
4.2	Graph showing Bell numbers . . . . .	19
5.1	FEATS' top-level flow . . . . .	21
5.2	Examples for the various levels of the hierarchy used in FEATS. . . . .	24
5.3	A rail-to-rail input stage with marked complementary loads. . . . .	28
5.4	Two complementary current mirrors forming a new complementary basic block. . . . .	29
5.5	Three nMOS current mirror variants forming an abstract basic. . . . .	30
5.6	Initial block rule (IBR) . . . . .	33
5.7	Elementary electric rule (EER) . . . . .	34
5.8	Symmetric elementary electric rule (SYM ERR) . . . . .	34
5.9	Library rule (LIB) . . . . .	34
6.1	Two basic blocks forming another existing. . . . .	38
6.2	A fully annotated example graph. . . . .	40
6.3	Structurally equal circuits with swapped pins. . . . .	41
6.4	Graph for circuit annotated for labeling-based isomorphism. . . . .	47

6.5	Generation of a circuit's inequality system. . . . .	51
7.1	Illustration of the asynchronous task distribution . . . . .	56
8.1	Bias Circuit . . . . .	60
8.2	Various bias circuit realizations . . . . .	62
8.3	Voltage-Averaging Circuit . . . . .	63
8.4	Fully differential output behavior without common mode feedback . . . . .	64
8.5	Common-Mode-Feedback Circuit . . . . .	65
8.6	Simple Common-mode-feedback amplifier realization . . . . .	66
8.7	Single-Ended Operational Amplifier . . . . .	68
8.8	Core Operational Amplifier . . . . .	70
8.9	Single-Ended Operational Transconductance Amplifier . . . . .	71
8.10	Fully Differential Operational Amplifier . . . . .	73
8.11	Unity testbench . . . . .	74
8.12	Classic implementation of the RC-network for a Sallen-Key filter architecture. . .	74
8.13	Sallen-Key 2nd-Order Low-Pass Filter . . . . .	75
8.14	Typical linear filter transfer functions. . . . .	76
8.15	Block-level diagram of a 3rd-order filter. . . . .	77
8.16	The ELIPLP configuration deducted from Figure 8.15, including the summation factors. . . . .	77
8.17	Visualization of the various hierarchy levels during the synthesis of an ELIPLP. To increase readability the levels below the FDAs are only showed once. . . . .	78
8.18	Elliptical 3rd-Order Low-Pass Filter . . . . .	79
9.1	nMOS differential stages variants. . . . .	82
9.2	Various nMOS miller variants. . . . .	82
9.3	Various nMOS double common source variants. . . . .	83
9.4	Various nMOS <i>folded</i> variants. . . . .	83
9.5	Various OTA realizations. . . . .	84
9.6	OTA mixed with a common source stage as driver. . . . .	84
9.7	The fully differential operational amplifier synthesized inside the elliptical filter .	89
9.8	Simulated nominal transfer function of the synthesized elliptical filter. . . . .	89
9.9	Uncommon operational amplifier topology. . . . .	90
9.10	Current stealer circuit schematic . . . . .	90
A.1	OP testbenches . . . . .	97
A.2	FDA testbenches . . . . .	97
A.3	ELIPLP testbenches I . . . . .	98
A.4	ELIPLP testbenches II . . . . .	98

## LIST OF ALGORITHMS

1	Synthesis Engine . . . . .	31
2	Top-level Find Uniques Algorithm . . . . .	45
3	Labeling Isomorphism Algorithm . . . . .	49

## LIST OF ALGORITHMS

---

## BIBLIOGRAPHY

- [AB95] ANTAO, B.AA and BRODERSEN, A.J. *ARCHGEN: Automated synthesis of analog systems*. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 3, no. 2:(1995), pp. 231–244. ISSN 1063-8210. doi:10.1109/92.386223.
- [Act] *Electronic linear filters*. [http://upload.wikimedia.org/wikipedia/commons/5/5c/Electronic\\_linear\\_filters.svg](http://upload.wikimedia.org/wikipedia/commons/5/5c/Electronic_linear_filters.svg).
- [AEBD00] ALPAYDIN, G., ERTEN, G., BALKIR, S. and DUNDAR, G. *Multi-level optimisation approach to switched capacitor filter synthesis*. *Circuits, Devices and Systems, IEE Proceedings -*, vol. 147, no. 4:(2000), pp. 243–249. ISSN 1350-2409. doi:10.1049/ip-cds:20000395.
- [AEG<sup>+</sup>00] ANTREICH, K., ECKMUELLER, J., GRAEB, H., PRONATH, M., SCHENKEL, F., SCHWENCKER, R. and ZIZALA, S. *WiCkeD: analog circuit synthesis incorporating mismatch*. In *Custom Integrated Circuits Conference, 2000. CICC. Proceedings of the IEEE 2000*, pp. 511–514 [2000]. doi:10.1109/CICC.2000.852720.
- [AGW94] ANTREICH, K.J., GRAEB, H.E. and WIESER, C.U. *Circuit analysis and optimization driven by worst-case distances*. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 13, no. 1:(1994), pp. 57–71. ISSN 0278-0070. doi:10.1109/43.273749.
- [Aig99] AIGNER, MARTIN. *A characterization of the Bell numbers*. *Discrete mathematics*, vol. 205, no. 1:(1999), pp. 207–210.
- [AII03] ANDO, SHIN, ISHIZUKA, MITSURU and IBA, HITOSHI. *Evolving Analog Circuits by Variable Length Chromosomes*. In Ashish Ghosh and Shigeyoshi Tsutsui, eds., *Advances in Evolutionary Computing*, Natural Computing Series, pp. 643–662. Springer Berlin Heidelberg [2003]. ISBN 978-3-642-62386-8. doi:10.1007/978-3-642-18965-4\_25.
- [AO07] AGGARWAL, VARUN and O'REILLY, UNA-MAY. *Simulation-based Reusable Polynomial Models for MOS Transistor Parameters*. In *Proceedings of the Conference on*

- Design, Automation and Test in Europe*, DATE '07, pp. 69–74. EDA Consortium, San Jose, CA, USA [2007]. ISBN 978-3-9810801-2-4.
- [APRG11] ARJMAND, O. N., PEDRAM, H., RAJI, M. and GHAVAMI, B. *CNT-count Failure Characteristics of Carbon Nanotube FETs under Process Variations*. *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, vol. 0:(2011), pp. 86–92. ISSN 1550-5774. doi: <http://doi.ieeecomputersociety.org/10.1109/DFT.2011.31>.
- [Asr98] ASRATIAN, ARMEN S. *Bipartite graphs and their applications*. 131. Cambridge University Press [1998].
- [BKN00] BRUCCOLERI, F., KLUMPERINK, E.A.M. and NAUTA, B. *Generating all 2-transistor circuits leads to new wide-band CMOS LNAs*. In *Solid-State Circuits Conference, 2000. ESSCIRC '00. Proceedings of the 26rd European*, pp. 316–319 [sept 2000].
- [BKN04] BRUCCOLERI, F., KLUMPERINK, E.A.M. and NAUTA, B. *Wide-band CMOS low-noise amplifier exploiting thermal noise canceling*. *Solid-State Circuits, IEEE Journal of*, vol. 39, no. 2:(2004), pp. 275–282. ISSN 0018-9200. doi:10.1109/JSSC.2003.821786.
- [BKV09] BASU, S., KOMMINENI, B. and VEMURI, R. *Variation-Aware Macromodeling and Synthesis of Analog Circuits Using Spline Center and Range Method and Dynamically Reduced Design Space*. In *VLSI Design, 2009 22nd International Conference on*, pp. 433–438 [Jan 2009]. ISSN 1063-9667. doi:10.1109/VLSI.Design.2009.51.
- [BSC77] BOOTH, S., KELLOGG and COLBOURN, C. J. *Problems polynomially equivalent to graph isomorphism*.
- [Cad] CADENCE DESIGN FRAMEWORK. [www.cadence.com](http://www.cadence.com).
- [CCB] *Attribution-ShareAlike 3.0 Unported (License)*. <http://creativecommons.org/licenses/by-sa/3.0/legalcode>.
- [CHS06] CHANG, SHOOU-JINN, HOU, HAO-SHENG and SU, YAN-KUIN. *Automated passive filter synthesis using a novel tree representation and genetic programming*. *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 1:(2006), pp. 93–100. ISSN 1089-778X. doi:10.1109/TEVC.2005.861415.
- [DBNV05] DE BERNARDINIS, F., NUZZO, P. and VINCENTELLI, A. SANGIOVANNI. *Mixed Signal Design Space Exploration Through Analog Platforms*. In *Proceedings of the 42Nd Annual Design Automation Conference, DAC '05*, pp. 875–880. ACM, New York, NY, USA [2005]. ISBN 1-59593-058-2. doi:10.1145/1065579.1065808.
- [DCR05] DASTIDAR, TR, CHAKRABARTI, PP and RAY, P. *A synthesis system for analog circuits based on evolutionary search and topological reuse*. *IEEE Transactions on evolutionary computation*, vol. 9, no. 2:(2005), pp. 211–224.



- 
- [Deh13] DEHGHANI, R. *Design of CMOS Operational Amplifiers*. Artech House microwave library. Artech House [2013]. ISBN 9781608071531.
- [DM09] DASGUPTA, S. and MANDAL, P. *An automated design approach for CMOS LDO regulators*. In *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, pp. 510–515 [jan 2009]. doi:10.1109/ASPDAC.2009.4796531.
- [dMH04] DEL MAR HERSHENSON, M. *CMOS analog circuit design via geometric programming*. In *American Control Conference, 2004. Proceedings of the 2004*, vol. 4, pp. 3266–3271 vol.4 [30 2004-july 2 2004]. ISSN 0743-1619.
- [Dod06] DODABALAPUR, ANANTH. *Organic and polymer transistors for electronics*. *Materials Today*, vol. 9, no. 4:(2006), pp. 24–30. ISSN 1369-7021. doi:http://dx.doi.org/10.1016/S1369-7021(06)71444-4.
- [DV07] DAS, ANGAN and VEMURI, RANGA. *GAPSYS: A GA-based Tool for Automated Passive Analog Circuit Synthesis*. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pp. 2702–2705 [may 2007]. doi:10.1109/ISCAS.2007.378519.
- [DV08] DAS, A. and VEMURI, RANGA. *ATLAS: An adaptively formed hierarchical cell library based analog synthesis framework*. pp. 2542–2545. doi:10.1109/ISCAS.2008.4541974.
- [Ebe88] EBELING, CARL. *Gemini II: A Second Generation Layout Validation Tool*. In *Proceedings of the IEEE International Conference on Computer Aided Design (ICCAD-88)* [1988].
- [EG12] EICK, M. and GRAEB, H.E. *MARS: Matching-Driven Analog Sizing*. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 8:(2012), pp. 1145–1158. ISSN 0278-0070. doi:10.1109/TCAD.2012.2190069.
- [Eic13] EICK, MICHAEL. *Structure and Signal Path Analysis for Analog and Digital Circuits*. Verlag Dr. Hut [2013].
- [ESL<sup>+</sup>11] EICK, M., STRASSER, M., LU, KUN, SCHLICHTMANN, U. and GRAEB, H.E. *Comprehensive Generation of Hierarchical Placement Rules for Analog Integrated Circuits*. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 2:(2011), pp. 180–193. ISSN 0278-0070. doi:10.1109/TCAD.2010.2097172.
- [ESRI14] ESTÉBANEZ, CÉSAR, SAEZ, YAGO, RECIO, GUSTAVO and ISASI, PEDRO. *Performance of the most common non-cryptographic hash functions*. *Software: Practice and Experience*, vol. 44, no. 6:(2014), pp. 681–698. ISSN 1097-024X. doi:10.1002/spe.2179.

- [Fac07] FACCHETTI, ANTONIO. *Semiconductors for organic transistors*. *Materials Today*, vol. 10, no. 3:(2007), pp. 28 – 37. ISSN 1369-7021. doi:[http://dx.doi.org/10.1016/S1369-7021\(07\)70017-2](http://dx.doi.org/10.1016/S1369-7021(07)70017-2).
- [For06] FORACHE, MARIN. *Why Normalization Failed to Become the Ultimate Guide for Database Designers?*. <http://dx.doi.org/10.2139/ssrn.905060>.
- [GJ79] GAREY, M.R. and JOHNSON, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of books in the mathematical sciences. W. H. Freeman [1979].
- [GJW11] GRAUPNER, A., JANCKE, R. and WITTMANN, R. *Generator based approach for analog circuit and layout design and optimization*. pp. 1–6. ISSN 1530-1591. doi:10.1109/DATE.2011.5763267.
- [GR00] GIELEN, G. and RUTENBAR, R.A. *Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits*. *Proceedings of IEEE*, vol. 88, no. 12:(2000), pp. 1825–1852.
- [GZEA01] GRAEB, H., ZIZALA, S., ECKMUELLER, J. and ANTREICH, K. *The sizing rules method for analog integrated circuit design*. *ICCAD '01*.
- [HM13] HEDRICH, LARS and MEISSNER, MARKUS. *FAATS, a Fully Automated Analog Topology Synthesis Framework*. *Invited Talk DASS 2013, Dresdner Arbeitstagung Schaltungs- und Systementwurf Dresden*.
- [HRC89] HARJANI, R., RUTENBAR, R.A. and CARLEY, L.R. *OASYS: A Framework for Analog Circuit Synthesis*. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 12:(1989), pp. 1247–66.
- [Hui11] HUIJISING, JOHAN. *Operational Amplifiers: Theory and Design*. Springer Publishing Company, Incorporated, 2nd ed. [2011]. ISBN 9400705956, 9789400705951.
- [HZCH14] HOU, AIMIN, ZHONG, QINGQI, CHEN, YUROU and HAO, ZHIFENG. *A Practical Graph Isomorphism Algorithm with Vertex Canonical Labeling*. *Journal of Computers*, vol. 9, no. 10.
- [ISO12] ISO. *ISO/IEC 14882:2011 Information technology — Programming languages — C++*. International Organization for Standardization, Geneva, Switzerland [Feb 2012].
- [ITR] *ITRS Roadmap - [www.itrs.net](http://www.itrs.net)*.
- [JCK12] JUNG, SEOBIN, CHOI, YUNJU and KIM, JAEHA. *Variability-aware, discrete optimization for analog circuits*. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pp. 536–541 [June 2012]. ISSN 0738-100X.

- [KAS11] KRISHNAPURA, N., AGRAWAL, A and SINGH, S. *A High-IIP<sub>3</sub> Third-Order Elliptic Filter With Current-Efficient Feedforward-Compensated Opamps*. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 58, no. 4:(2011), pp. 205–209. ISSN 1549-7747. doi:10.1109/TCSII.2011.2124571.
- [KBL<sup>+</sup>97] KOZA, J.R., BENNETT, III, F.H., LOHN, J., DUNLAP, F., KEANE, M.A. and ANDRE, D. *Automated synthesis of computational circuits using genetic programming*. In *Evolutionary Computation, 1997., IEEE International Conference on*, pp. 447–452 [apr 1997]. doi:10.1109/ICEC.1997.592353.
- [KBN01] KLUMPERINK, E.A.M., BRUCCOLERI, F. and NAUTA, B. *Finding all elementary circuits exploiting transconductance*. *IEEE transactions on circuits and systems II: analog and digital signal processing*, vol. 48, no. 11:(2001), pp. 1039–1053.
- [KL95] KRUISKAMP, W. and LEENARTS, D. *DARWIN:CMOS opamp Synthesis by means of a Genetic Alorithm*. *DAC '95: Design Automation Conference*, pp. 433–438.
- [KPH<sup>+</sup>01] KRASNICKI, M.J., PHELPS, R., HELLUMS, J.R., MCCLUNG, M., RUTENBAR, R.A and CARLEY, L.R. *ASF: a practical simulation-based methodology for the synthesis of custom analog circuits*. In *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on*, pp. 350–357 [Nov 2001]. ISSN 1092-3152. doi:10.1109/ICCAD.2001.968646.
- [KSG90] KOH, HY, SEQUIN, C.H. and GRAY, P.R. *OPASYN: A compiler for CMOS operational amplifiers*. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 9, no. 2:(1990), pp. 113–125. ISSN 0278-0070.
- [LC99] LOHN, J.D. and COLOMBANO, S.P. *A circuit representation technique for automated circuit design*. vol. 3, pp. 205–219 [sep 1999]. ISSN 1089-778X. doi:10.1109/4235.788491.
- [LH12] LOURENÇO, NUNO and HORTA, NUNO. *GENOM-POF: Multi-objective Evolutionary Synthesis of Analog ICs with Corners Validation*. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, GECCO '12*, pp. 1119–1126. ACM, New York, NY, USA [2012]. ISBN 978-1-4503-1177-9. doi:10.1145/2330163.2330318.
- [LPP] *LaTeX project public license*). <http://latex-project.org/lppl/lppl-1-3c.html>.
- [LVGH06] LOURENGO, N., VIANELLO, M., GUILHERME, J. and HORTA, N. *LAYGEN - Automatic Layout Generation of Analog ICs from Hierarchical Template Descriptions*. pp. 213–216. doi:10.1109/RME.2006.1689934.
- [Map] MAPLESOFT. [www.maplesoft.com](http://www.maplesoft.com).
- [MCR95] MAULIK, P.C., CARLEY, L.R. and RUTENBAR, R.A. *Integer programming based topology selection of cell-level analog circuits*. *Computer-Aided Design of Integrated*

*Circuits and Systems, IEEE Transactions on*, vol. 14, no. 4:(1995), pp. 401–412. ISSN 0278-0070.

- [Mei14] MEISSNER, MARKUS. *Exploring the Design-Space with 'FAATS' to achieve First-Time-Right Silicon in Analog Designs. Invited Talk DATE14, Design, Automation & Test in Europe, Dresden.*
- [MGS08] MASSIER, T., GRAEB, H. and SCHLICHTMANN, U. *The Sizing Rules Method for CMOS and Bipolar Analog Integrated Circuit Synthesis. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 12:(2008), pp. 2209–2222. ISSN 0278-0070. doi:10.1109/TCAD.2008.2006143.
- [MH15] MEISSNER, M. and HEDRICH, L. *FEATS: Framework for Explorative Analog Topology Synthesis. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 34, no. 2:(2015), pp. 213–226. ISSN 0278-0070. doi:10.1109/TCAD.2014.2376987.
- [MLWI08] MANAKA, TAKAAKI, LIU, FEI, WEIS, MARTIN and IWAMOTO, MITSUMASA. *Diffusionlike electric-field migration in the channel of organic field-effect transistors. Phys. Rev. B*, vol. 78:(2008), p. 121,302. doi:10.1103/PhysRevB.78.121302.
- [MMH11a] MEISSNER, M., MITEA, O. and HEDRICH, L. *Graph-based Framework for Explorative Topology Synthesis of Analog Circuits. In FAC11 [2011].*
- [MMH11b] MEISSNER, M., MITEA, O. and HEDRICH, L. *Graphen-basiertes Framework zur explorativen Topologiesynthese von analogen Schaltungen [2011].*
- [MMH11c] MITEA, O., MEISSNER, M. and HEDRICH, L. *Automated Constraint-driven Topology Synthesis for Analog Circuits. In Proc. of the Conference on Design, Automation and Test in Europe [2011].*
- [MMH11d] MITEA, OLIVER, MEISSNER, MARKUS and HEDRICH, LARS. *Topology Synthesis of Analog Circuits with Yield Optimization and Evaluation using Pareto Fronts. In 19th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC) [October 2011].*
- [MMH12] MA, MINGYU, MEISSNER, MARKUS and HEDRICH, LARS. *A Case Study: Automatic Topology Synthesis for Analog Circuit from an ASDeX Specification. In International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design 2012 (SMACD'12), Seville, Spain [September 2012].*
- [MMLH12] MEISSNER, M., MITEA, O., LUY, L. and HEDRICH, L. *Fast isomorphism testing for a graph-based analog circuit synthesis framework. In Design, Automation Test in Europe Conference Exhibition (DATE), 2012, pp. 757 –762 [march 2012]. ISSN 1530-1591.*

- [MPSG09] MCCONAGHY, T., PALMERS, P., STEYAERT, M. and GIELEN, G. G E. *Variation-Aware Structural Synthesis of Analog Circuits via Hierarchical Building Blocks and Structural Homotopy*. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 9:(2009), pp. 1281–1294. ISSN 0278-0070. doi:10.1109/TCAD.2009.2023195.
- [Mun] MUNEDA GMBH. *www.muneda.com*. <http://www.muneda.com>.
- [MV01] MANDAL, P. and VISVANATHAN, V. *CMOS op-amp sizing using a geometric programming formulation*. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 20, no. 1:(2001), pp. 22–38. ISSN 0278-0070. doi:10.1109/43.905672.
- [Ndj11] NDJOUNTCHE, TERTULIEN. *CMOS analog integrated circuits: high-speed and power-efficient design*. CRC Press, Hoboken, NJ [2011].
- [OE93] OHLRICH, M. and EBELING, C. *SubGemini: Identifying SubCircuits using a Fast Subgraph Isomorphism Algorithm*. *DAC '93: Design Automation Conference*, pp. 31–37.
- [Pre09] PRESA, JOSÉ LUIS LÓPEZ. *Efficient Algorithms for Graph Isomorphism Testing*. Ph.D. thesis, Licenciado en Informatica Madrid [2009].
- [Raz00] RAZAVI, B. *Design of analog CMOS integrated circuits*. McGraw-Hill, Inc. New York, NY, USA [2000].
- [RRmLB04] REESE, COLIN, ROBERTS, MARK, MANG LING, MANG and BAO, ZHENAN. *Organic thin film transistors*. *Materials Today*, vol. 7, no. 9:(2004), pp. 20–27. ISSN 1369-7021. doi:[http://dx.doi.org/10.1016/S1369-7021\(04\)00398-0](http://dx.doi.org/10.1016/S1369-7021(04)00398-0).
- [Rut10] RUTENBAR, ROB A. *Analog layout synthesis: what's missing?*. In *ISPD*, p. 43 [2010].
- [San06] SANSEN, WILLY M. C. *Analog Design Essentials (The International Series in Engineering and Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA [2006]. ISBN 0387257462.
- [SEGA99] SCHWENCKER, R., ECKMUELLER, J., GRAEB, H. and ANTREICH, K. *Automating the Sizing of Analog CMOS Circuits by Consideration of Structural Constraints*. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE '99*. ACM, New York, NY, USA [1999]. ISBN 1-58113-121-6. doi:10.1145/307418.307516.
- [She14] SHEN, RU. *Graph mining and module detection in protein-protein interaction networks*. Ph.D. thesis, STATE UNIVERSITY OF NEW YORK AT ALBANY [2014].

- [SHM13] SALFELDER, FELIX, HEDRICH, LARS and MEISSNER, MARKUS. *Evaluating NBTI in Synthesized Operational Amplifiers using an Accurate Ageing Model*. In *ANALOG13 - Aachen* [March 2013].
- [SKP05] STEFANOVIC, DANICA, KAYAL, MAHER and PASTRE, MARC. *PAD: A New Interactive Knowledge-Based Analog Design Approach*. *Analog Integrated Circuits and Signal Processing*, vol. 42, no. 3:(2005), pp. 291–299. ISSN 0925-1030. doi:10.1007/s10470-005-6762-9.
- [SyE] *Syntheseunterstützter Entwurf analoger Schaltungen*. <https://www.edacentrum.de/syena/de/frameset/>.
- [TB05] TULUNAY, G. and BALKIR, S. *Design automation of single-ended LNAs using symbolic analysis*. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 1294–1297 Vol. 2 [May 2005]. doi:10.1109/ISCAS.2005.1464832.
- [TB08] TULUNAY, G. and BALKIR, S. *A Synthesis Tool for CMOS RF Low-Noise Amplifiers*. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 5:(2008), pp. 977–982. ISSN 0278-0070. doi:10.1109/TCAD.2008.917579.
- [TD06] TANG, HUA and DOBOLI, A. *High-level synthesis of Delta; Sigma; Modulator topologies optimized for complexity, sensitivity, and power consumption*. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 3:(2006), pp. 597–607. ISSN 0278-0070. doi:10.1109/TCAD.2005.854633.
- [vRMH15] VON ROSEN, J., MEISSNER, M. and HEDRICH, L. *Semiautomatic Implementation of a Bioinspired Reliable Analog Task Distribution Architecture for Multiple Analog Cores*. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2015* [March 2015]. Accepted for publication.
- [vRSH<sup>+</sup>15] VON ROSEN, JULIUS, SALFELDER, FELIX, HEDRICH, LARS, BETTING, BENJAMIN and BRINKSCHULTE, UWE. *A highly dependable self-adaptive mixed-signal multi-core system-on-chip architecture*. *Integration, the {VLSI} Journal*, vol. 48, no. 0:(2015), pp. 55 – 71. ISSN 0167-9260. doi:http://dx.doi.org/10.1016/j.vlsi.2014.04.001.
- [WH06] WANG, X. and HEDRICH, L. *An approach to topology synthesis of analog circuits using hierarchical blocks and symbolic analysis*. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, p. 705. IEEE Press [2006].
- [YCh] *Gajski-Kuhn Y-chart*. <http://www.texample.net/media/tikz/examples/TEX/gajski-kuhn-y-chart.tex>.
- [YL07] YU, GUO and LI, PENG. *Yield-aware Analog Integrated Circuit Optimization Using Geostatistics Motivated Performance Modeling*. In *Proceedings of the 2007*

*IEEE/ACM International Conference on Computer-aided Design, ICCAD '07*, pp. 464–469. IEEE Press, Piscataway, NJ, USA [2007]. ISBN 1-4244-1382-6.

[ZKT85] ZEMLYACHENKO, V.N., KORNEENKO, N.M. and TYSHKEVICH, R.I. *Graph isomorphism problem*. *Journal of Soviet Mathematics*, vol. 29, no. 4:(1985), pp. 1426–1481. ISSN 0090-4104. doi:10.1007/BF02104746.

[ZLY<sup>+</sup>12] ZHANG, YU, LIU, BO, YANG, BO, LI, JING and NAKATAKE, S. *CMOS op-amp circuit synthesis with geometric programming models for layout-dependent effects*. In *Quality Electronic Design (ISQED), 2012 13th International Symposium on*, pp. 464–469 [march 2012]. ISSN 1948-3287. doi:10.1109/ISQED.2012.6187534.