

# **AIDA-C: Evolutionary Optimization Techniques applied to Analog IC Design**

**André Ricardo Henriques Ferreira**

Thesis to obtain the Master of Science Degree in

**Electrical and Computer Engineering**

Supervisor: Prof. Nuno Cavaco Gomes Horta

Prof. Nuno Calado Correia Lourenço

## **Examination Committee**

Chairperson: Prof. Horácio Cláudio de Campos Neto

Supervisor: Prof. Nuno Calado Correia Lourenço

Member of the Committee Prof. Manuel Fernando Martins de Barros

**May 2016**

## **Abstract**

---

The work presented in this report belongs to the field of Electronic Design Automation (EDA), particularly, to the domain of Analog IC Design Automation. Despite the advances on the state-of-the-art, nowadays most of the automatic synthesis tools address circuit level synthesis and the setup is very much dependent on the designers expertise when specifying, e.g., design search space, design constraints, topology selection, etc., which limits the efficiency of the design automation process. In this work the automatic constraint generation is addressed not only to increase the level of automation but also give more support to the designer. The state-of-the-art shows that just a few approaches addressed the topic of automatic constrain generation, basically, considering two alternatives one using a pattern recognition approach and other a signal flow graph approach. These approaches will be further investigated to enhance AIDA-C, a state-of-the-art circuit level synthesis tool developed at IT (Instituto de Telecomunicações), and validated with a diversified set of examples including circuits such as, LC-VCO, LNA, OpAmps, Bandgaps, etc.

## **Keywords**

---

Analog Integrated Circuits, Electronic Design Automation, , Design Constraints Generation, Signal Flow Graphs, AIDA-C



## Resumo

---

O trabalho apresentado está relacionado com o campo de Automatização de Projecto Electrónico (EDA), com especial ênfase no domínio da automatização de projecção de circuitos integrados analógicos. Apesar dos mais recentes avanços tecnológicos, a maioria das ferramentas de sintetização automática foca-se mais ao nível do circuito, e a configuração depende bastante da experiência e conhecimentos do projectista aquando a especificação de características como por exemplo selecção de topologia, definição do espaço de procura, definição de *constraints*, etc, limitando a eficiência do processo de automatização do projecto. Neste trabalho a geração automática de *constraints* não será só para incrementar o grau de automatização, mas também para assistir o projectista no processo de design. O estado da arte mostra que esta vertente não tem sido seguida, havendo poucos trabalhos a abordar o problema da geração automática de *constraints*, considerando apenas duas alternativas: uma, onde se procuram padrões no circuito, e outra que consiste na análise da propagação do sinal. Estas abordagens foram estudadas mais aprofundadamente para serem integradas e validadas no AIDA-C, uma ferramenta de síntese a nível do circuito desenvolvida no IT (Instituto de Telecomunicações) bastante referenciada com um conjunto diversificado de exemplos, incluindo circuitos como o LC-VCO, LNA, AmpOps, Bandgaps, etc. O método conseguido gera *constraints* baseados em *matching* de transístores, simetrias, e proximidades com sucesso e com um tempo de processamento de alguns segundos.

## Palavras Chave

---

Circuitos Integrados Analógicos, Automação do Projecto de Circuitos, Geração Automática de Restrições Funcionais, Grafos de Fluxo de Sinal, AIDA-C



# Table of Contents

<b>ABSTRACT .....</b>	<b>I</b>
<b>KEYWORDS.....</b>	<b>I</b>
<b>RESUMO .....</b>	<b>III</b>
<b>PALAVRAS CHAVE .....</b>	<b>III</b>
<b>TABLE OF CONTENTS .....</b>	<b>V</b>
<b>1 INTRODUCTION .....</b>	<b>1</b>
1.1 Analog Design Flow.....	1
1.2 Analog IC Design Optimization .....	3
1.3 Motivation .....	3
1.4 Goals.....	4
1.5 Document Structure.....	5
<b>2 STATE-OF-THE-ART .....</b>	<b>7</b>
2.1 Analog IC Design Automation Tools .....	7
2.2 Simulation-based automatic circuit sizing .....	8
2.3 Automatic constraint generation applied to analog circuit sizing .....	9
2.4 Sizing Rules Method.....	11
2.4.1 Automatic Building Block recognition.....	11
2.4.2 Recognition Algorithm .....	12
2.5 Hierarchical Placement Rules' SSFG .....	12
2.5.1 Definition .....	12
2.5.2 SSFG Generation.....	12
2.5.3 Hierarchical Symmetry Assignment.....	13
2.5.4 MARS Enhanced Structural Signal Flow Graph.....	14
2.5.5 ESFG Generation.....	14
2.6 Conclusions.....	14
<b>3 AIDA OVERVIEW.....</b>	<b>17</b>
3.1 AIDA-C Architecture .....	17
3.1.1 AIDA Setup .....	18
3.2 Efficient Setup with Automatic Constraint Extraction.....	21
3.3 Conclusions .....	22
<b>4 AUTOMATED CONSTRAINTS GENERATION MODULE.....</b>	<b>23</b>
4.1 Overview .....	23
4.2 Reading the netlist.....	24
4.3 Pattern configuration, constraints, and subgraphs .....	25

4.3.1	Level 0 .....	25
4.3.2	Level 1 .....	26
4.3.3	Level 2. ....	29
4.4	Pattern Detection .....	31
4.5	Finding symmetry .....	35
4.5.1	Generating the signal flow graph .....	35
4.5.2	Finding symmetric nodes.....	35
4.5.3	Determining symmetric devices.....	36
4.6	Generating constraints.....	36
4.6.1	Sizing and electronic constraints .....	36
4.6.2	Proximity constraints .....	36
4.6.3	Symmetry constraints .....	37
4.7	Designing new patterns .....	37
4.7.1	Connections configuration .....	37
4.7.2	Constraints generation .....	37
4.7.3	Subgraph generation.....	38
4.7.4	Remaining details.....	38
4.8	Integration with AIDA .....	39
4.8.1	Netlist .....	40
4.8.2	Electrical Constraints and Measures .....	40
4.9	Running Example .....	41
4.10	Observations.....	45
<b>5</b>	<b>EXPERIMENTAL RESULTS.....</b>	<b>47</b>
5.1	Folded Cascode .....	47
5.2	Fully Differential two-stage Folded Cascode.....	50
5.3	Fully Differential OTA.....	53
5.4	Folded Cascode Optimization Project.....	54
5.5	Conclusions .....	56
<b>6</b>	<b>CONCLUSIONS AND FUTURE WORK .....</b>	<b>57</b>
6.1	Conclusion.....	57
6.2	Future work .....	57
<b>7</b>	<b>REFERENCES .....</b>	<b>59</b>
	<b>APPENDIX A.....</b>	<b>67</b>
<b>A.1</b>	<b>FOLDED CASCODE .....</b>	<b>67</b>
<b>A.2</b>	<b>FULLY DIFFERENTIAL TWO-STAGE FOLDED CASCODE .....</b>	<b>72</b>
<b>A.3</b>	<b>FULLY DIFFERENTIAL OTA.....</b>	<b>78</b>





# 1 Introduction

Since IC technology has been developed and made accessible, it has seen exponential growth throughout the decades of its development that enabled a multitude of life-changing applications like cellphones, personal computers, and the self-driving cars are expected to be available in the near future. The level of integration in modern very large scale integration technologies (VLSI), enables extremely complex, multi million transistors electronic circuits to be integrated in a few  $\text{mm}^2$  with reduced costs (in mass production), which allowed circuit designers to create IC's that, meet the needs of the demanding microelectronics market, for new functionalities, smaller devices, lower production costs, higher power efficiency, etc.. These complex single IC designs are established in telecommunications, medical and multimedia applications, where blocks of AMS, digital processors and memory blocks appear together. To increase the performance of ICs, i.e. enhance functionalities and/or lower power consumption, there is an exponential increase of transistor density in ICs, as described by Moore's law. This means that the designers deal with the project ICs containing billions of transistors, under extreme competitive market conditions. [1].

Although the analog component of the SoC only occupies approximately 20% of the global circuit area (as shown in Figure 1.1) the design effort is considerably higher when compared to its digital counterpart. In digital design, it is usual to reuse digital projects, leading to an increased productivity of design. By contrast, in analog design there are no mature and well-defined strategies to address a problem, leading to custom solutions that are difficult to reuse. Several Electronic Design Automation (EDA) tools and design methodologies are available for digital IC's that assist the designers in managing the increased complexity systems, as well as keeping up with the fast-paced progress offered by the technology. On the other hand, and despite the developments achieved in recent years in analog design automation, analog design automation keeps lagging behind with practically no automation and very few design reuse appearing in the designers' flow. [2]

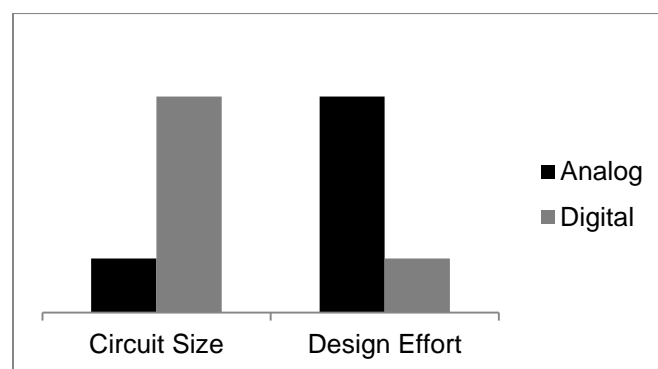
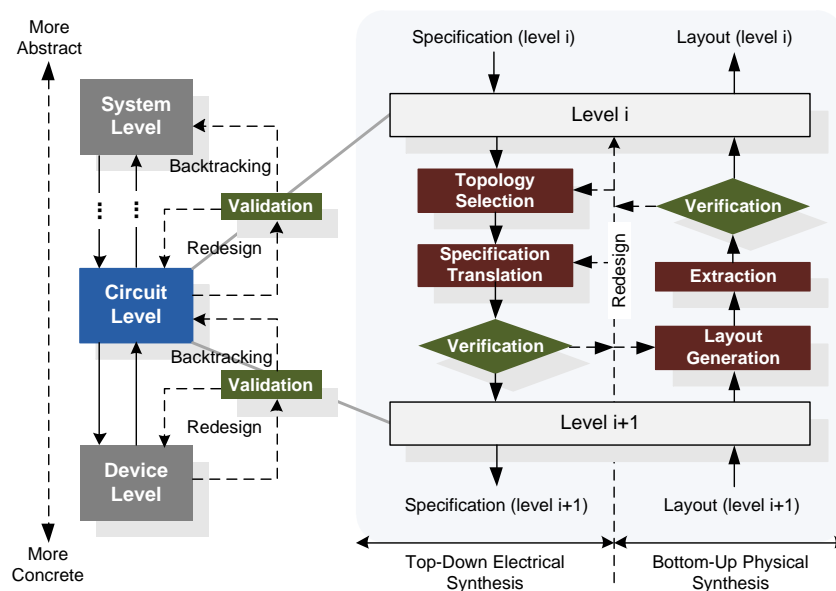


Figure 1.1 - Comparison between Analog and Digital design efforts

## 1.1 Analog Design Flow

Although several design flows for analog circuits can be found, a generally well accepted and used by many analog design automation works developed in the last years follow the design flow introduced by Gielen and Rutenbar [3]. This design flow for AMS IC circuits, illustrated in Figure 1.2, consists of a

series of top-down design steps repeated from the system level to the device-level, and bottom-up layout generation and verification. By adopting a hierarchical top-down design methodology, it is possible to perform system architectural exploration, obtaining a better overall system optimization at a higher level before starting more detailed implementations at device level. Problems are found in the early stages of the design flow, and as a result have a higher chance of first-time success, with fewer or no overall time consuming redesign iterations [4]. The number of hierarchy levels depends on the complexity of the system being handled, and the steps between any two hierarchical levels can be the top-down electrical synthesis, with topology selection, circuit sizing at its lowest level, and design verification. Then, there's the bottom up physical synthesis, which includes layout generation and detailed design verification after extraction.



**Figure 1.2 – Top-down and bottom-up tasks of design flow [5]**

Topology selection is the step of determining the most appropriate circuit topology in order to meet a set of given specifications of the current hierarchy level. This topology can be either chosen from a set of available topologies or be newly created. Specification translation is the step of mapping the high-level specifications for the selected topology block under design into individual specifications for each of the sub-blocks, at the lowest level the sub blocks are single devices, thus becoming circuit sizing. Specifications translation is verified by means of simulations before proceeding down in the hierarchy. Since no device-level sizing is available at higher levels, behavioral simulations are needed and electrical simulations are used at the lowest design hierarchy level. The specifications for each of the blocks are passed to the next level of the hierarchy and the process is repeated until the top-down flow is completed. Some recent works based on Pareto Optimal Fronts (POF) have been very successful exploring the tradeoff during synthesis [6], and have already been applied at system level sizing. To aid in the task, designers resort to several CAD tools that became widely used throughout the years, rendering the standard for IC design editing and evaluation. ADiT, Questa, Eldo [7]; HSPICE [8],

nanosim, HSim [9]; Spectre [10]; ngspice [11] and SMASH [12] are among the most used tools in the latest years.

Layout generation consists of creating the geometrical layout of the block under design at the lowest level in the design hierarchy, or place and route the layouts of the sub-blocks at higher levels. In the presented design flow, it is important to notice the presence of a detailed verification step over the extraction of the layout. In order to ascend to higher hierarchical levels is necessary that no potential problems are detected at the lowest levels and the layout meet the target requirements. When the topmost level verification is complete, the system is considered to be designed. Some of the most used CAD tools available for layout edition are IC Station Layout [7], Galaxy Custom Designer LE [9] and Virtuoso Layout Editor [10]. CALIBRE [7], Hercules [9] and DIVA, Assura [10] can be used for Design Rule Checking (DRC) and layout extraction.

## **1.2 Analog IC Design Optimization**

While design automation of analog IC has advanced greatly in the last couple of decades, it has not improved as much as digital IC design. This issue comes from the analog design IC itself, which has a significantly higher level of complexity, even for small problem sizes, and which lacks a sufficiently comprehensive and exact descriptiveness with conventional approaches.

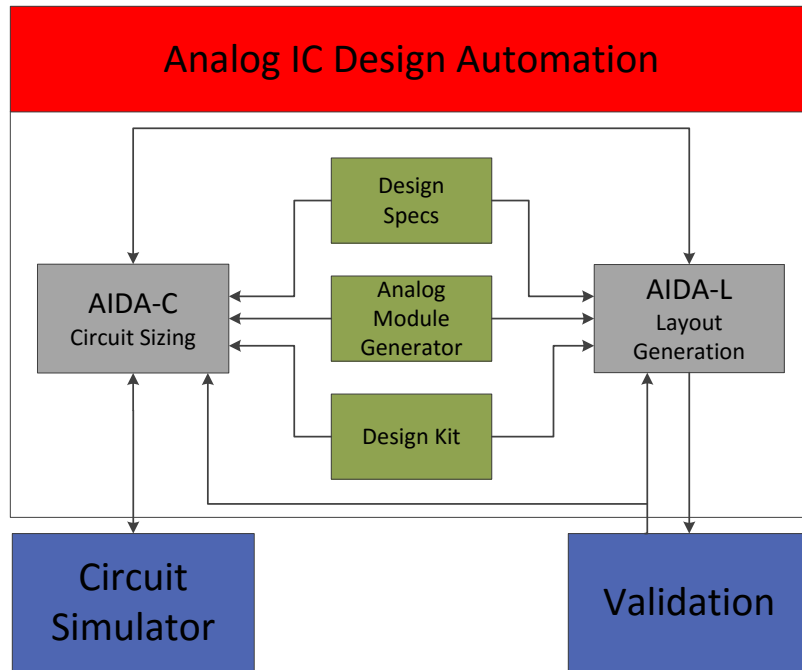
Analog circuit sizing automation is becoming more and more common, and is mostly achieved using optimization techniques, which may or may not use a circuit simulator to aid in performance evaluation of possible solutions during the optimization process. Some commercially available tools, such as Cadence's Virtuoso Custom Design Platform GXL [10], Synopsys Titan ADX [9], or MunEDA-GNO [13] already have this optimization approach implemented. However, these tools focus on a single-objective optimization, providing only one solution, thus giving the designer no other goal to maximize or tradeoff with [14].

AIDASoft is an Analog IC Design automation framework, fully developed at Instituto de telecomunicações' Integrated Circuits Group. AIDA-C, is the frameworks' circuit optimizer, where circuit-level sizing optimization is performed, enhancing the robustness of solutions by considering process and operating variations (i.e. PVT corners), and for circuit's performance figures, they are measured with electrical circuit simulators, like Spectre [10], eldo [7], or HSPICE. Once the optimization process is complete, a Pareto Optimal Front is given, showing all the non-dominated solutions, in which the designer can explore design tradeoffs and choose a specific design. An overview of the AIDA can be seen in Figure 1.3, showing how the design process works.

## **1.3 Motivation**

The quality of a design is generally determined by the degree to which compliance constraints have been met and predefined optimization goals achieved. Due to the lack of identical expression and interpretation of design constraints in the analog design flow context, most of the constraints in analog designs are specified and considered manually by expert designers. Therefore, analog constraints are implicitly based on a designer's experience, rather than being explicitly defined, preventing their

systematic use in design automation and leading to an error prone flow. Progress in analog IC design automation is needed due to increasing design sizes and aggravating challenges (e.g. required robustness), as well as a widening verification gap [15].



**Figure 1.3 - AIDA Architecture and Design Flow [14]**

In the last decades, analog topology synthesis, nominal design optimization, and sizing with respect to tolerances were in the focus of research interest. Certain approaches include equation-based methods, where design equations are derived with the help of symbolic analysis and simulation-based methods. Sizing tasks have a key potential for providing automation support to the designer, particularly when considering mismatch and process and operating tolerances [16].

Specifying circuit design goals (e.g. gain or bandwidth of an amplifier) are often not enough to prevent optimizers from meeting a mathematical solution while not reaching the IC design solution, and when an IC design solution is found, it often has a high sensitivity to process and operating variations and to noise [17]. Adding constraints on lower levels of the IC design process, e.g. transistor sizing and placing, or enforcing certain transistor voltages to keep them in the desired operating region. These constraints (or sizing rules) can be generated as equalities or inequalities, either for physical design (e.g. transistor dimensions) or for electrical design (e.g. drain-source voltage), and both kinds of constraints are desired, as they aid in not only reaching an IC design solution, but to optimize the relation between design performance and process yield.

## 1.4 Goals

This work has as main objective the automatic generation of constraints to be implemented and embedded in the AIDA-C optimizer. Since AIDA-C's method is to search in a given search space, more complex circuits generate many degrees of freedom, and finding a solution requires a lot of processing time. By automatically adding constraints, the search space is reduced and solution finding

takes less time. The constraints generation module will be validated with typical analog building blocks.

## **1.5 Document Structure**

This document is organized as:

Chapter 2 presents state-of-the-art and the latest research in automatic constraint generation.

Chapter 3 presents a more detailed work of AIDA-C and plans for implementation of automatic constraint generation.

Chapter 4 presents the Automated Constraint Generation Architecture, describing the pattern search, symmetry search, and constraint generation.

Chapter 5 presents experimental results, using example circuits and showing the detected patterns, symmetries, and generated constraints, and some optimization results.

Chapter 6 draws the conclusions and presents future work topics.



## 2 State-of-the-art

In the last 25 years, many techniques were approached by the scientific community for the automation of the analog circuit sizing. In this chapter those approaches are briefly surveyed with further emphasis on the methods used to define the design constraints, and a summary describing how current knowledge will be applied in an optimization tool in order to achieve automation in the field of constraint generation.

### 2.1 Analog IC Design Automation Tools

Generally speaking, analog circuit sizing automation techniques are classified in two main groups: the knowledge-based approaches and the optimization based approaches [18]. This classification is based on the fundamental techniques used to address the problem, as illustrated in Figure 2.1.

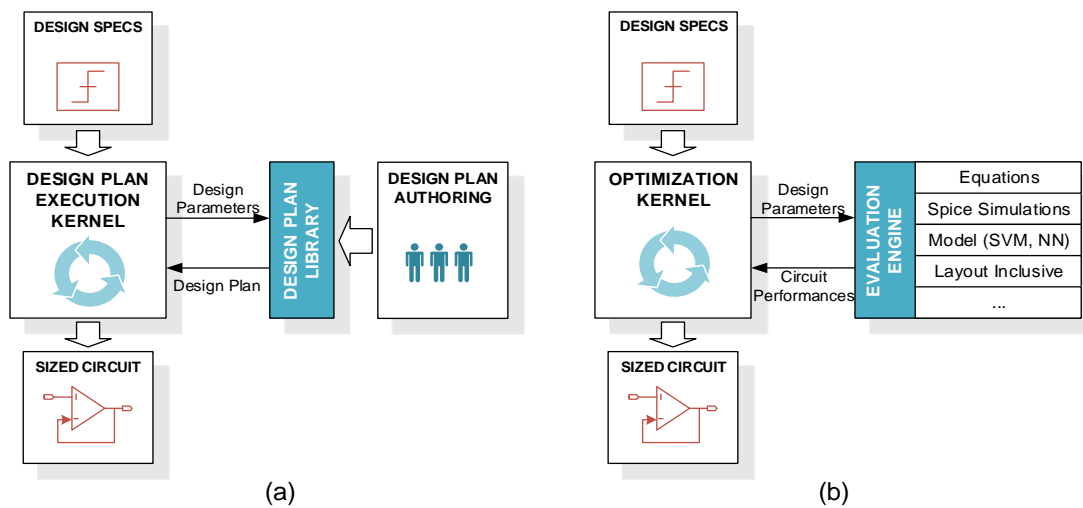


Figure 2.1 - Automatic circuit sizing approaches [18]

(a) knowledge-based (b) optimization-based

Most early automation tools [19, 20, 21, 22] did not use optimization, and tried instead to systematize the design by using a design plan that was originated from expert knowledge. These tools relied on design equations used to build a plan and a design strategy that produces the component sizes that meet the performance requirements. These knowledge-based approaches were applied with some level of success, with short execution time being the main advantage of this knowledge-based approach. However, deriving the design plan is both hard and time consuming, and once the design plan is achieved, requires constant maintenance to keep up with the progress being made, and still does not manage to achieve optimal results. The following generations of sizing tools apply optimization techniques to analog IC sizing, which can be further classified into equation-based or simulation-based, considering the method used to evaluate the circuit's performance.

The equation-based methods use analytic design equations to create a relationship between the circuit's performance goals and the design variables. Different optimization techniques are used, both deterministic and stochastic. Taking advantage of knowing the equations and their properties allows

the use of classical optimization methods. In OPASYN [23] the optimization is performed using steepest descent, and similarly in STAIC [24] is used a successive solution refinements technique.

Maulik et. al. [25, 26] defined the sizing problem as a constrained nonlinear optimization problem using spice models and DC operating point constraints, solving it through sequential quadratic programming. Matsukawa et. al. [27] designed  $\Delta\Sigma$  and pipeline analog to digital converters solving via convex optimization the equations that relate the performance of the converter to the size of the components.

In GPCAD [28] a posynomial circuit model is optimized using Geometrical Programming (GP), the execution time is in the order of few seconds, but the general application of posynomial models is difficult and the time to derive the model for new circuits is still high. Kuo-Hsuan et. al. [29] revisited the posynomial modeling recently, solving the accuracy issue by introducing an additional generation step, where local optimization using simulated annealing (SA) and a circuit simulator is performed. The same strategy is applied in FASY [30, 31], where analytical expressions are solved to generate an initial solution and a simulation-based optimization is performed to further improve the solution.

Other equation based approaches do not limit the problem formulation in order to use a specific optimization technique, relying on heuristic optimization instead. OPTIMAN [32] uses SA applied to analytical models, and, in ASTRX/OBLX [33], a SA optimization is also performed using a cost function defined by equations for dc operation point, and small signal Asymptotic Waveform Evaluation (AWE)-based simulation, this evaluation technique is also used in DARWIN [34], which uses Genetic Algorithms (GA) instead. Doboli et. al. [35] also applies genetic programming techniques to simultaneously derive the sub-blocks specifications, sub-block topology selection and transistor sizing.

The equation-based methods' main advantage is the short evaluation time, making them highly suited to find first-cut designs as the knowledge-based approaches were. On the other hand, despite the advances in symbolic analysis, equations are unable to accurately simulate the behavior of the design characteristics, making the generalization of the method to different circuits very difficult. The approximations introduced in the equations also yield lower accuracy designs as circuit complexity increases, requiring additional work to ensure that the circuit meets the needed specifications.

## **2.2 Simulation-based automatic circuit sizing**

Recent increases in access to processing power and memory capacity allowed simulation based optimization to also develop, being the most common method found in recent approaches, since simulation allows for better accuracy designs. In simulation-based sizing, as in the case of AIDA-C, a circuit simulator, e.g., SPICE [36], is used to evaluate the circuit.

Early approaches to simulation based automatic sizing used local optimization around a “good” solution, where SA [37] is the most common optimization technique used. SA mimics the annealing of material under slow cooling to minimize the internal energy, as the name suggests. In DELIGHT.SPICE [38] the optimization algorithm (phase I-II-III method of feasible directions) is used to perform local design optimization around a user provided starting point. Kuo-Hsuan et. al. [29] and FASY [31, 30] use equation-based techniques to derive an approximate solution, and then use



simulation within a SA kernel to optimize the design. Cheng et al. [39] also uses SA but considers the transistor bias conditions to constrain the problem, and, instead of solving the circuit by finding transistor sizes, the problem is solved by finding the bias of the transistors. FRIDGE [40] aims for general applicability approach by using an annealing-like optimization without any restriction to the starting point. Castro-Lopez et al. [41] use SA followed by a deterministic method for fine-tuning to perform the optimization.

Swarm intelligence algorithms [42] can also be found in the literature applied to analog circuit sizing. The fundament of swarm intelligence algorithms is to use many simple agents that lead an intelligent global behavior, like the one observed in many insect hives. From these methods, the most common are the ant colony optimization (ACO), which was successfully applied in [43, 44], and particle swarm optimization (PSO) that can be found in [45, 46, 47].

Circuit sizing is in its essence a multi-objective multi constraint problem, and the designer often explores the tradeoff between contradictory performance measures. For example, minimizing power consumption while maximizing bandwidth, or maximizing gain and minimizing area of an amplifier. As such, the usage of multi-objective optimization techniques is becoming increasingly common. When considering multiple objectives the output is not one solution, but a set of optimal design tradeoff solutions, usually referred as Pareto Optimal front (POF). Given the multiple elements already present in both evolutionary and swarm intelligence algorithms, these are the natural candidates to implement such approach. In GENOM-POF [48, 49] and MOJITO [50] the evolutionary multi-objective methods are applied, respectively, to circuit sizing and both sizing and topology exploration, whereas in [47] the particle swarm optimization is explored in both single and multi-objective approaches. A different approach is taken by Pradhan and Vemuri in [51], where the multi-objective simulated annealing (MOSA) is used.

From the study of analog circuit sizing and optimization approaches proposed by the scientific community recently, it is clear that there is not a specific trend to consider a single best algorithm, but many were experimented with. In the next section the summary of the surveyed approaches is presented, and finally the objectives for this work are refined, namely the selection of the optimization kernels to be initially included in the platform.

## **2.3 Automatic constraint generation applied to analog circuit sizing**

The analog sizing tools approaches surveyed are summarized in Table 2-1. In the equation-based systems the usage of classical optimization methods is possible. However, models' accuracy and the derivation of such equations strongly limits applicability. This limitation of the equation based systems is overcome at the expense of evaluation time by using accurate circuit simulation to evaluate the performance figures being optimized.

Although most experienced circuit designers have heuristic knowledge when manually creating constraints for a determined circuit, some of the constraints might be overlooked, either because the designer is not aware of the constraint, or the constraint is not considered to be of much relevance to

the end result, making circuit optimizers take longer optimization times to find the solution. An automatic constraint generation would allow to a more complete constraint generation, making the circuit solutions more robust as well as saving setup time and reducing optimization time.

This work will focus on the automatic generation of design constraints in the scope of AIDA-C. In AIDA-C, the circuit sizing and optimization problem, which will be described in detail in chapter 3, is modeled as a multi-objective multi-constraint optimization problem. In this context special relevance is given to the constraints.

From a brief perusal of the tools available in the state of the art, it is noticed that despite the relevance of constraints in the design of analog circuits, there are few approaches to automatically generate such constraints. In the next sections the most relevant approaches are detailed, first the work by Massier et al [16], that focuses on the generation of constraints for circuit sizing, then another work from the same research group, that focuses on layout related constraints, namely placement [52].

**Table 2-1 – Summary of analog IC design automation tools for sizing and optimization**

Tool/Author		Design Plan/Optimization Method	Evaluation	Constraints Definition
IDAC [21]	1987	Design plan plus SA post-optimization	Equations	Manual
DELIGHT.SPICE [38]	1988	Feasible directions Optimization	Simulator	Manual
OPASYN [23]	1990	Steepest descent	Equations	Manual
OPTIMAN [32]	1990	SA	Equations	Manual
STAIC [24]	1992	2 step optimization	Equations	Manual
Maulik et al. [25, 26]	1993	B&B, and Seq. Quadratic Progr.	Equations and BSIM models	Manual
FRIDGE [40]	1994	SA	Simulator	Manual
DARWIN [34]	1995	GA	small signal, analytical expressions.	Manual
ISAID [19, 20]	1995	Qualitative reasoning + post optimization	Equations and Qualitative reasoning	Manual
FASY [31, 30]	1996	SA + Gradient	Simulator	Manual
ASTRX/OBLX [33]	1996	SA	AWE Equations	Manual
Koza [53]	1997	GA	Simulator	Manual
GPCAD [28]	1998	Geometric Programming	Posynomial	Manual
MAELSTROM [54]	1999	GA+SA	Simulator	Manual
ANACONDA [55]	2000	Stochastic pattern search	Simulator	Manual
Sipramong [56]	2002	GA	Simulator	Manual
Alpaydin [57]	2003	Evolutionary strategies + SA	Fuzzy + NN trained with Simulator	Manual
Shoou-Jin [58]	2006	GA	Equations	Manual
Barros [18, 59, 60]	2006	GA	Simulator	Manual
Castro-Lopez [41]	2008	SA + Powels method	Simulator	Manual
Santos-Tavares [61]	2008	GA	Simulator	Manual
MOJITO [50]	2009	NSGA-II	Simulator	Manual
Pradhan [51]	2009	Multi-Objective SA	Layout aware MNA models	Manual
Matsukawa [27]	2009	Convex Optimization	Convex functions	Manual
Cheng [39]	2009	SA	Equations	Manual
Hongying [62]	2010	GA with VDE	Simulator	Manual
Fakhfakh [47]	2010	Multi-objective PSO	Equations	Manual
Pehl [63]	2010	SQP and B&B	Simulation	Automatic
Kuo-Hsuan [29]	2011	Convex optimization; Stochastic Fine Tuning	Posynomial Simulator	Manual
Habal [64]	2011	Deterministic non-linear optimization	Evaluation	Automatic
Roca et al. [65]	2012	NSGA-II	Simulator	Manual
Genom-POF [49, 48]	2012	NSGA-II	Simulator	Manual
AIDA-GM [66, 67, 2]	2013	NSGA-II	Simulator	Manual
Liao et al. [68]	2013	Look-up Table	Equations and Lookup Tables	Automatic
Afacan et al. [69]	2014	MBHO, IMBHO	Equations	Manual
AIDA [14, 70, 71]	2014	NSGA-II	Simulator	Manual

## 2.4 Sizing Rules Method

Automatic sizing has seen slow progress due to often incomplete circuit specifications. Generally, specifying only the circuit performance (e.g. dc gain, bandwidth of an operational amplifier) is not enough for optimizers to reach a solution, and even if a solution is found (i.e. that works according to the specifications), it tends to be very sensitive to process and operation variations and to noise [24]. Additional sizing rules for transistor geometry and voltages have to be considered (i.e. constraints), so the circuit becomes increasingly robust. Research has been conducted when considering the importance that constraints have in finding a solution in an IC design. However, for most works, in the end responsibility falls on the designer when specifying constraints. An automated process that would generate some sizing rules when the circuit is read would aid optimizers in finding a solution with little to no effort required from the designer.

The Sizing Rules Method (SRM) approach is to find certain matching patterns, so that it can generate sizing rules without any aid from the designer. This reduces setup time and effort, as well as optimization times. Sizing rules can efficiently capture design knowledge on the technology-specific level of transistor-pair groups. The Sizing Rules Method also helps in further general circuit sizing, design centering, response surface modeling or analog placement, bysetting the foundations to further create layout-oriented constraints.

### 2.4.1 Automatic Building Block recognition

In order to generate design constraints, some circuit patterns need to be recognized, so constraints can be assigned to their respective transistors in their building blocks.

Much of the electrical circuit design process is composed of several sets of transistors (e.g., a current mirror, an amplifier). This aids the designer in dividing a circuit design into multiple sub-circuit designs (or building blocks), and thus breaking down design into multiple, lower complexity problems. This method allows the designer to define transistor specifications based on its function in the building block it is in. Some building blocks can also be merged to form a new, higher level building block (e.g. a Simple Current Mirror and a Level Shifter can form a Cascode Current Mirror). This allows the generation of additional constraints when considering the necessary relations between the different building blocks. Thus, having a hierarchical system of building blocks, where each level is a combination of multiple building blocks from lower levels, seems to be a good approach to the problem at hand.

The main benefit from the hierarchically organized block building is that new building blocks can be generated by merging existing ones. This simplifies the assignment of constraints by having the new building block inherit the constraints from its' forming blocks. In other words, when a new building block is generated, only a few constraints are added. In a given circuit schematic, the block recognition algorithm detects all building blocks that correspond to the respective elements in the library, starting from simple building blocks on low levels, and going up the hierarchy into more complex blocks, in higher levels. [16]

## 2.4.2 Recognition Algorithm

Set  $M$  starts initialized with all circuit elements contained in the netlist (level 0 modules), and two relations  $R_1$  and  $R_2$  are initialized.

Whenever a new module is found, a new ordered pair will be added to both relations. The upper submodule  $m_\lambda$  (with index 1), and the new module  $m_\mu$ , are stored as an ordered pair in  $R_1$ , while the lower submodule  $m_k$  (index 2) is also stored as a pair with the new module,  $m_\mu$ , in  $R_2$ . These relations are used to handle recognition ambiguities. The algorithm scans through all the library elements above  $L_0$ , i.e. library elements that consist of more than a single transistor. This means that each element in the hierarchical library is only checked once by the algorithm. Each element builds into the  $C_{l(1), l(2)}$  relation. For each element in the library, all possible module pairs  $m_k$  and  $m_\lambda$  are analyzed and compared. If patterns of  $C_{l(1), l(2)}$  and  $C_{k,\lambda}$  match, then  $(m_k, m_\lambda)$  forms a new building block, as well as a new module,  $m_\mu$ , with  $\mu = |M| + 1$ , which is instantiated and added to set  $M$ . Although a new module is created, its' forming sub-modules remain in the set  $M$ , considering that a single module can belong to multiple different building blocks. After a new module is instantiated, its pins are connected to the appropriate nets of its submodules.

After fully analyzing the circuit all the modules have been created (i.e. set  $M$  is complete), sizing rules will be assigned in a top-down order. Should a transistor or a certain building block not be assigned to a hierarchical library element, it becomes uncertain, and no sizing rules are generated for them, except that they must be in saturation. Instead, these modules are provided to the designer for further actions.

## 2.5 Hierarchical Placement Rules' SSFG

SSFG (Structural Signal Flow Graph) representation combines structural and qualitative behavioral information. Through SSFG, the problem size is reduced and its additional structural information prevents the symmetry computation from exploring many infeasible solutions. [52]

### 2.5.1 Definition

A SSFG is a directed graph. Its nodes represent nets of the circuit and the edges indicate the ways through which signal can propagate. An edge pointing from node  $n_i$  to another node  $n_j$  means that a change in voltage and/or current in  $n_i$  can cause a change in voltage and/or current in  $n_j$ .

### 2.5.2 SSFG Generation

Each building block in the library has its corresponding sub-SSFG. Because the current library is small, each sub-SSFG can be manually created.

After assigning a sub-SSFG to each building block of a circuit, a graph union operation merges all the sub-SSFGs into the circuit's SSFG,  $G_s$

$$G_S = \bigcup_{b \in B} G_{S,b} \quad (2.1)$$

$$G(N, E) = \bigcup_i G_i(N_i, E_i) \Leftrightarrow N = \bigcup_i N_i \wedge E = \bigcup_i E_i \quad (2.2)$$

For the follow-up symmetry assignment, each edge is given three attributes: the type of building block which originated the edge, along with the names of the two associated pins. For example, a NMOS simple current mirror (n-scm) from pin  $p_a$  to  $p_b$  gets the attributes (n-scm, a, b). The graph is setup so that there are no parallel edges with equal attributes.

### 2.5.3 Hierarchical Symmetry Assignment

The symmetry assignment algorithm generates pairs of symmetric nodes and edges from the SSFG according to the following definitions. Two nodes,  $n_i$  and  $n_j$  are symmetric if they are differential inputs or outputs, or if symmetric edges  $(e_i, e_j)$  start or end at  $n_i, n_j$ . Two edges,  $e_i$  and  $e_j$ , are symmetric if they start and end at two symmetric nodes  $(n_i, n_j)$  and have identical attributes. It is also important to note self-symmetric edges, which connect two symmetric nodes, confirming the symmetry path possibility that was being checked.

Ambiguities often arise, and this is due to different symmetric edge pairs being formed if said edges have equal attributes and begin at the same node. These ambiguities can be resolved through a backtracking approach, which starts by considering one symmetry assignment to be correct. The search is then continued as normally, and if it remains valid, then it is part of the solution. If not, the search is repeated with a different assignment, starting from the latest ambiguity. A solution may be verified through two processes: In one hand, the search ends at two nodes connected by a self-symmetric edge, or at two nodes without any successors (including when they are differential outputs). On the other hand, any edge starting at  $n_i$  must be symmetric to another edge starting at a symmetric pair node,  $n_j$ , and vice versa.

Symmetrical Building blocks are then generated from the symmetry assignments made to the edges and nodes the SSFG. Two building blocks are determined symmetrical if they belong to the same type and have symmetrical edges. A building block can also be self-symmetrical if its edges are symmetrical and/or self-symmetrical.

Finalizing the top-down symmetry attribution, devices themselves are determined symmetrical through symmetry conditions from building blocks. When two building blocks are symmetrical, it means that their corresponding devices must also be symmetrical between each other. When a building block is self-symmetrical, the device symmetry is previously specified for each building block type. In the end, a set  $S$  is formed, with all the symmetric elements of the circuit

## 2.5.4 MARS Enhanced Structural Signal Flow Graph

The ESFG (Enhanced Structural Signal Flow Graph), as the name indicates, is an improvement of the SSFG. It complements the SSFG by including the handling of input and output terminals, as well as passive, one-port devices [72]. It can be defined as

$$G_E = (N_{G_E}, E_{G_E}, \varphi_{G_E}, \alpha_{G_E}, \beta_{G_E}) \quad (2.3)$$

where

- $N_{G_E}$  is a set of nodes containing all nets  $N$ , and terminals,  $T$  of the circuit

$$N_{G_E} \subseteq N \cup T \quad (2.4)$$

- $E_{G_E}$  is the set of edges, which similarly to the SSFG, represent ways of signal propagation.
- $\alpha_{G_E}$  is a set of attributes which characterize its corresponding edge. This kind of information was already implemented in the original SSFG concept, but it had not been formally characterized. For example, a NMOS simple current mirror (n-scm) from pin  $p_a$  to  $p_b$  is represented by edge  $e_1$ , with  $\alpha(e_1) = (n - scm, p_a, p_b)$ .
- $\beta_{G_E}$  is the set of devices that physically implementing each edge  $e$ . For example, if  $e_1$  is an edge representing a scm (simple current mirror) with transistors  $M_1$  and  $M_2$ , then  $\beta(e_1) = \{M_1, M_2\}$ .

Unlike SSFG, the ESFG of a circuit may have two equal edges, i.e. they make the same connection between two nodes, have equal attributes, and represent the same physical devices.

## 2.5.5 ESFG Generation

The generation of an ESFG is simple once the SSFG generation algorithm has been implemented. First, additional sub-graphs are created to model single-port, passive devices (which were not included in the SSFG method). Then, the set of all subgraphs is generated, similarly to the SSFG method. Then the node set of the ESFG is formed by all nodes that are the the starting or ending of edges.

## 2.6 Conclusions

In this chapter a survey of techniques and approaches to the automation of analog circuit sizing were presented. The different approaches were classified in terms of the techniques used and the most significant aspect observed was the setup and the execution time, as well as the accuracy in the evaluation of the solutions. In this survey were presented several ADA tools and analyzed to better understand the advantages, and, drawbacks that can be improved in the future. It was also possible to identify that a wide range of optimization techniques are considered in the field and new ones are always being introduced.

Although automatic constraint generation is not yet diversified, existing works have contributed significantly to automatic generation of crucial sizing rules in terms of optimization times, and is a very reasonable starting point to implement in AIDA-C.

In this work, Automatic Generation of Analog IC Design Constraints enhances AIDA-C by reducing degrees of freedom when optimizing a circuit. The building block hierarchy is still relatively small, so some future work might consist of adding more building block elements to the hierarchical library, if further patterns are considered to be necessary.





### 3 AIDA Overview

The work developed targets the automation of the constraint definition for analog IC circuit design, and is to be implemented in the AIDA framework, illustrated in Figure 3.1. This chapter starts with an overview of AIDA. A detailed analysis of the tools' setup with special emphasis to the constraint definition is presented, from the study of the tool's setup the contributions to be made in this work are revisited.

#### 3.1 AIDA-C Architecture

AIDA is an analog IC design automation framework with two branches: AIDA-C and AIDA-L. AIDA-C, featured in Figure 3.1, targets optimization of device sizing through state-of-the-art and innovative techniques. It is based in state-of-the-art multi-objective, multi-constraint optimization techniques and targets highly robust designs by considering PVT corner simulations. AIDA-L takes as inputs the device sizes that AIDA-C provides as well as the best floorplan templates, and generates a layout by placing and routing the devices using internal Design-Rule Check (DRC) and Layout-Vs-Schematic (LVS) methods, finishing the circuit design process.

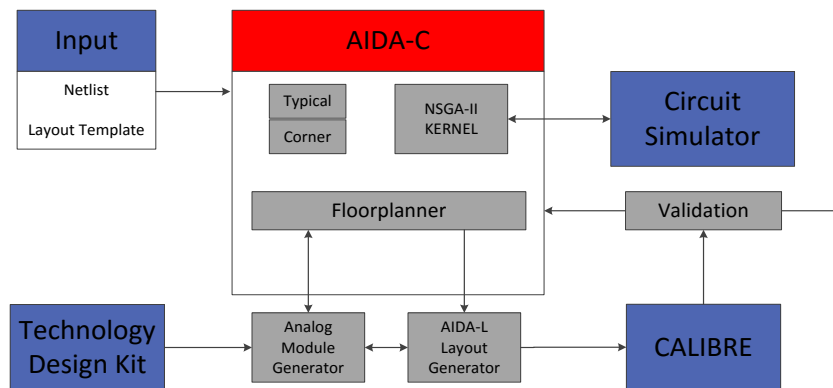
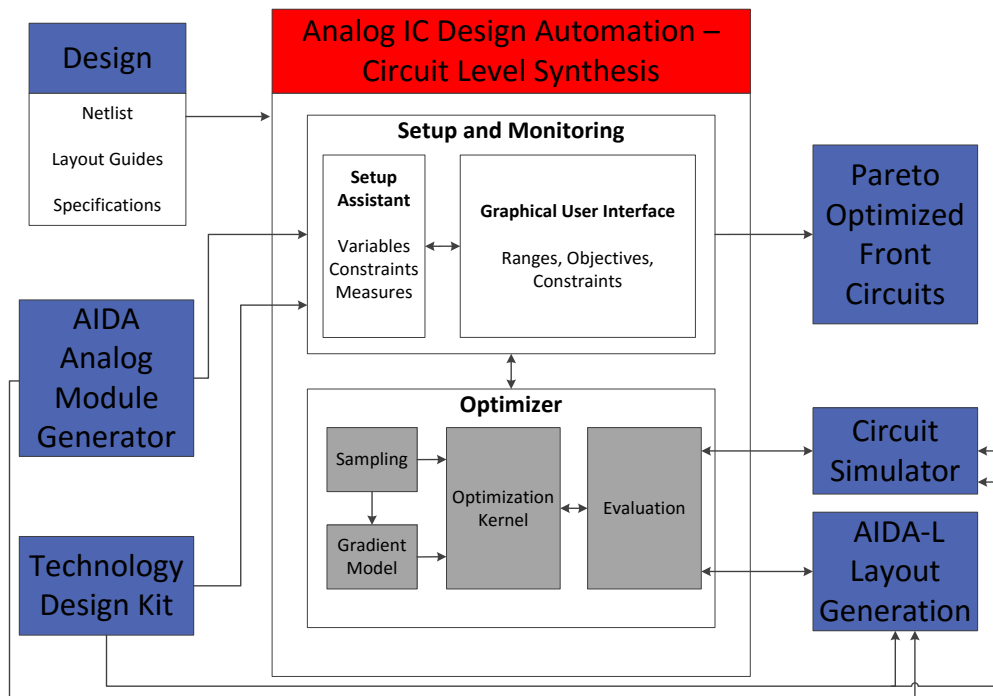


Figure 3.1 – AIDA-C Overview

Figure 3.2 details AIDA-C composed by two main modules: the Setup & Monitoring module, and the Optimizer module. The Setup & Monitoring module assists the designer through the design process and aid in using the circuit optimizer. The Optimizer solves the circuit through multi-objective techniques, where the circuit's performance is measured through industrial circuit simulators for electrical measures, as well as through AIDA-L, for layout-related characteristics. In the scope of this work the Setup Assistant will be analysed with special focus.

The Setup Assistant helps the setup by generating some statements and drafts automatically. For example, for the active devices some default measures and the correspondent constraint statements for overdrives and saturation margin (delta) are generated automatically with the purpose of having all transistors working in the saturation region. The measures for the currents in all circuit nodes are also generated by AIDA-C. In addition, a draft for the layout guides is generated from the netlist directly. Using these aids, the setup productivity is somewhat enhanced, but the level of setup automation is limited.



**Figure 3.2– AIDA-C Architecture**

### 3.1.1 AIDA Setup

In this section, it is explained how to setup all the necessary elements to use AIDA-C. The setup of a design in AIDA-C is mostly made at file level. The inputs from the designer are the circuit and test-bench(es) in the form of spice-like netlist(s). The netlist(s) must have the optimization variables as parameters, and must include means to measure the circuit's performance. Corner's parameter variations are also included in the netlist. In addition, the designer defines ranges for the optimization variables, design constraints, and optimization objectives. If a layout-aware circuit sizing optimization is intended, the AIDA-L's layout guides must also be provided.

Adding a circuit to AIDA-C is a two-step operation. First, the circuit netlist needs to be properly parameterized, and then, using AIDA-C's setup assistant to accelerate the process, AIDA-C design structure, as shown in Figure 3.1, needs to be created. The project is organized by a set of files, with "design.xml" as the main description file. XML is a markup language that defines a set of rules for encoding documents in a format that is both human- and machine-readable. The image is optional, and is used to show the circuit schematic to ease the identification of the devices and parameters when using the tool (commonly a screenshot of the schematic). The folder "layout" contains multiple layout guide files and the folder "tech\_netlist" contains the circuit and testbenches.

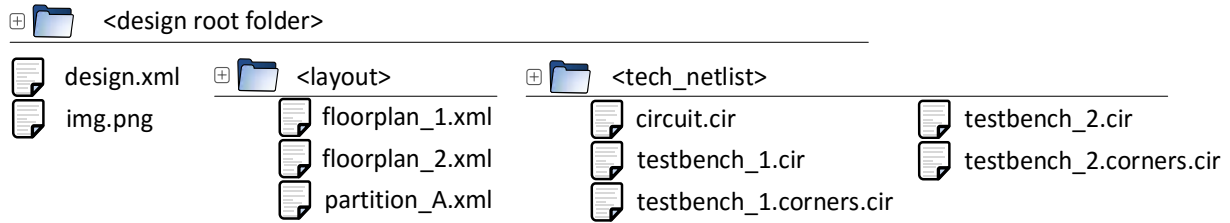


Figure 3.3 - AIDA-C design structure.

Starting from the circuit netlist, designed by hand or exported from a schematic editor, the design variables are defined. This is done by taking into consideration what are the values the optimizer may change (usually the sizes of the devices). In this stage, matching of devices can be enforced. The design variables are defined as parameters and are used in the netlist appropriately. Figure 3.4 shows the parameterized netlist used for the Miller amplifier example.

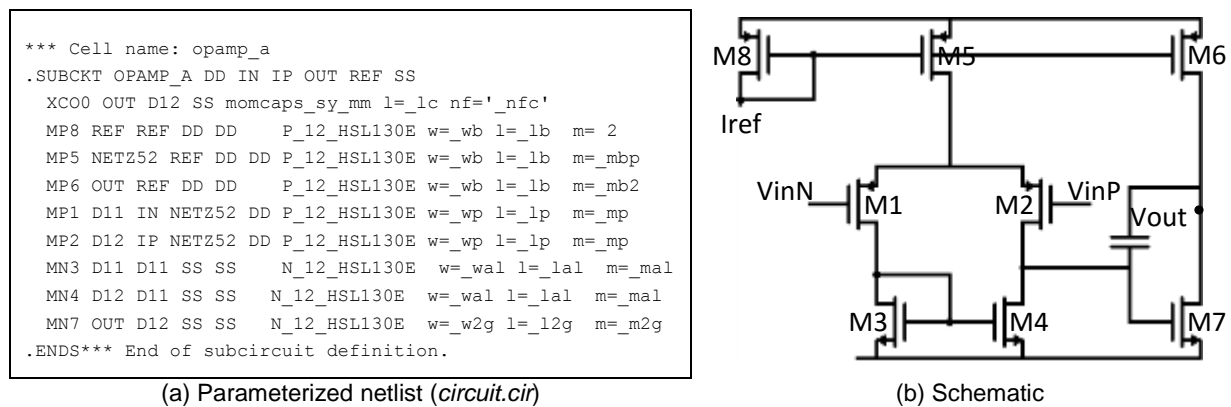


Figure 3.4 - Single-ended two-stage Miller amplifier.

From the parameterized netlist, the setup assistant automatically generates structural measures (overdrives, deltas and active area) for all transistors. In addition, DC current measures for all device terminals, which are needed for AIDA-L's router, are also generated. These measures, shown in Figure 3.5, are not mandatory, and are up to the designer to be included or not in the relevant testbenches.

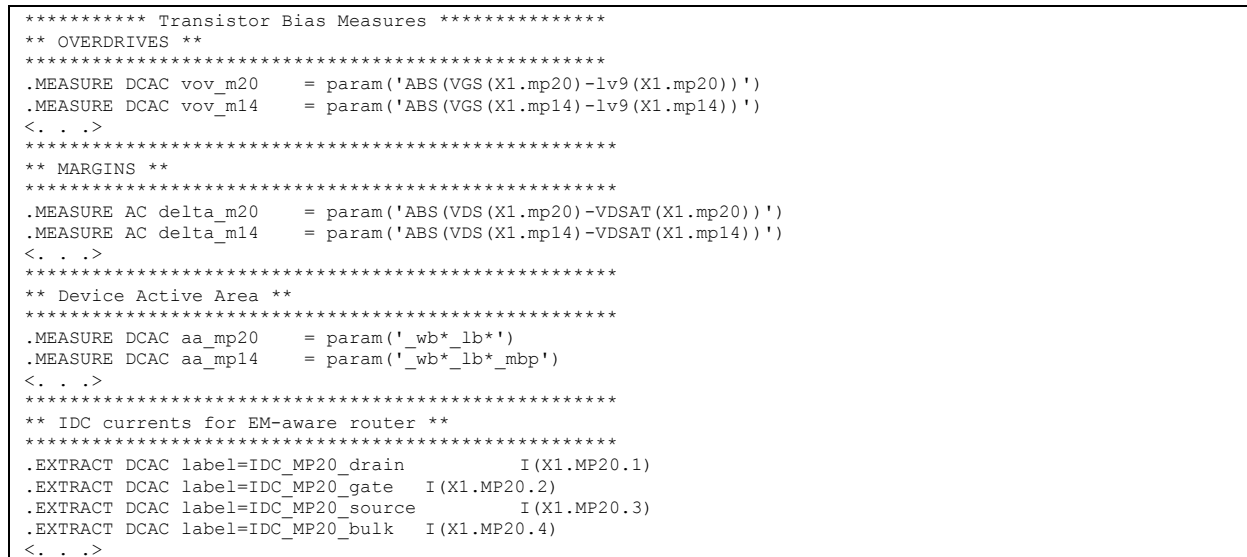


Figure 3.5 - DC measures for ELDO™ AC testbench

At this point, the netlist for the circuit and its testbenches are completed. The next step is to create the “design.xml” file where the circuit setup is described. Again, the “setup assistant” is used to accelerate the setup by generating a draft of the “design.xml” from the data in the netlist and some technology-dependent default values (overdrives, ranges, etc.) that are stored in AIDA’s design kit. The circuit netlist is parsed and the variables are detected, as well as the devices. With this data, the setup assistant generates a partially filled design.xml, where the designer is responsible for adding the missing data fields. The complete setup for the Miller amplifier is shown in Figure 3.6.

```

1: <?xml version="1.0" encoding="ISO-8859-1"?>
2: <!DOCTYPE Design SYSTEM "design-1.0.dtd">
3: <Design name="OPAMP_A">
4:   <Circuit netlist="2stage.cir" techNode="UMC_013">
5:     <Layout template="t1a.xml"/>
6:   ...
16:   <Layout template="t3d.xml"/>
17:
18:   <Var id="_lc" range="4.4e-6:1e-7:1.0e-4"/>
19:   <Var id="_nfc" range="14:2:198"/>
20:   <Var id="_wb_wp_wal_w2g" range="1.00e-6:1e-7:10e-6"/>
21:   <Var id="_lb_lp_lal_l2g" range="0.12e-6:5e-8:10e-6"/>
22:   <Var id="_mbp_mbp2_mp_mal_m2g" range="1:2:1000"/>
23: </Circuit>
24: <TestbenchSetup simulator="eldo" inputMethod="LAM">
25:   <TestCase name="wp_avdd_max_dvdd_max_temp_min_1_vcm_max"/>
26:   <TestCase name="ws_avdd_min_dvdd_min_temp_max_8_vcm_max"/>
27:   <TestCase name="wp_avdd_max_dvdd_max_temp_min_1_vcm_min"/>
28:   <TestCase name="ws_avdd_min_dvdd_min_temp_max_8_vcm_min"/>
29:
30:   <NominalTb netlist=" ac_testbench.cir.eldo">
31:     <Meas name="idd" description="I VDD [A]"/>
32:     <Meas name="gdc" description="Gain DC [Hz]"/>
33:     <Meas name="gbw" description="Unity Gain Frequency [Hz]"/>
34:     <Meas name="pm" description="Phase margin [degrees]"/>
35:     <Meas name="psrr" description="Power Supply Rejection Ratio [Hz]"/>
36:     <Meas name="sr" description="Slew Rate [V/s]"/>
37:     <Meas name="voff" description="Structural Offset [V]"/>
38:     <Meas name="no" description="Noise [V]"/>
39:     <Meas name="sn" description="Noise Density [V/sqrt(Hz)]"/>
40:     <Meas name="device_area" description="Area from net list [m2]"/>
41:     <Meas id="vov_mp20 vov_mp14 vov_mp22 vov_mp11 vov_mp12 vov_mp9 ..." desc="Overdrive [V]"/>
42:     <Meas id="delta_mp20 delta_mp14 delta_mp22 delta_mp11 delta_mp12 ..." desc="Delta [V]"/>
43:   </NominalTb>
44:   <WorstCaseTb netlist=" ac_testbench.cir.eldo.corners">
45:     <Meas importFrom="ac_testbench.cir.eldo"/>
46:   </WorstCaseTb>
47: </TestbenchSetup>
48: <Constraint op="GE" value="0.10" meas="vov_mp20 vov_mp14 vov_mp22 vov_mp11 vov_mp12 vov_mp9 ..."/>
49: <Constraint op="GE" value="0.10" meas=" delta_mp20 delta_mp14 delta_mp22 delta_mp11 ..."/>
50: </Design>

```

**Figure 3.6 – Completed circuit setup**

When layout effects are to be considered, layout guides must be provided. The AIDA-L’s layout guides, which are also described in XML, describe the floorplan and are parameterized to include the design variables. The floorplan is defined using simple rectangular constructs that capture the proximity and topological relations that the designer wishes to impose. The information used for placement is the type and relative placement of the cells, and also, the symmetry, matching and combine requirements. The high level floorplan of each cell is described by a box shape. The size of this box has no meaning, only the relative position between cells (boxes) is of concern. The topological constraints that are enforced by the tool are inferred from the boxes’ placement directly.

Symmetry is specified locally in each floorplan (or sub-floorplan) by two properties: ‘symGroupId’ that identifies a group of cells that share the same symmetry axe; and ‘symCellId’ that identifies a pair of the cells that are to be placed symmetrically in relation to the symmetry axe. By default, cells are self-symmetric and do not share their symmetry axis. Matching is enforced in the device parameterization,

where the devices that the designer deemed to be matched share some or all parameters (in both layout guides and netlist). Finally, combine operations can be used to replace a group of basic cells with more complex layout structures, e.g., merged structures or interdigitated/common-centroid layout styles.

In Figure 3.7 (a) part of the XML description of the hierarchically defined layout guides, and in (b) the corresponding graphical representation is shown. The devices in blue are defined hierarchically in the sub-template for partition P1A. The expected location of the power supply nets, VSS and VDD, are illustrated in the image running on top and bottom of the layout, respectively, although there is no explicit definition in the template.



Figure 3.7- Single ended two-stage amplifier layout guides

- (a) Part of the XML description of the layout guides (*floorplan.xml*), showing the constructs and illustrating the hierarchy, with part of the sub-floorplan file for partition P1A shown inline;  
 (b) Graphical representation of a template showing the relative location of the devices.

## 3.2 Efficient Setup with Automatic Constraint Extraction

After reviewing how the setup is done in AIDA, this work's contribution becomes clear and can be defined with more detail.

- Starting from the netlist shown in Figure 3.4 (a), where all the matching is done manually, using pattern identification techniques can make matching between devices possible to automatize.

- The measures for each device are generated but are not included automatically in the netlist because there is no processing of the circuit other than identifying individual devices. Due to the same reason only device related constraints (overdrive and saturation margin) are automatically considered. With the blocks properly identified, all measures and constraints that are needed can be introduced automatically.
- In terms of the desing.xml, the setup assistant generated only a very simplified version. Using more knowledge will permit the automatic generation of a more complete setup file, leading to considerable savings in terms of setup in addition to preventing potential configuration errors.
- The layout guides are mostly manual, and further automation of the template generation is possible using circuit recognition and rule extraction to define symmetry, grouping and hierarchical partitioning.

### **3.3 Conclusions**

In this chapter AIDA's architecture was reviewed, with special emphasis on constraint definition and circuit sizing. Considering PVT corners analysis and the usage of industrial circuit simulators allows AIDA to find a set of optimal multi-objective sizing solutions denominated Pareto Optimal Front (POF). The resulting sizing solutions represent a trade-off between the two optimized variables, where no solution is strictly better than another. Constraints are determined during optimization setup, so that to aid in the optimization process by restricting the search space, and reducing optimization time, as well as finding more robust solutions. Once the optimization project is finished, one sizing solution can then be sent to AIDA-L for layout generation.

## 4 Automated Constraints Generation Module

In this chapter, the Constraint Generation Module will be thoroughly explained. A brief overview will explain the general working method of the module, and then each stage is analysed with further detail. Then a running example will be presented, using a simple OpAmp, and explaining with detail how each stage will work with the given circuit.

### 4.1 Overview

Figure 4.1 illustrates how the Automated Constraints Generation works. The module receives as input a non-sized circuit netlist, and according to the provided technology specific variables and the library of patterns and their respective constraints and subgraphs, generates outputs according to the desired constraints, in the form of a parameterized netlist, electrical constraints, and their respective measure commands.

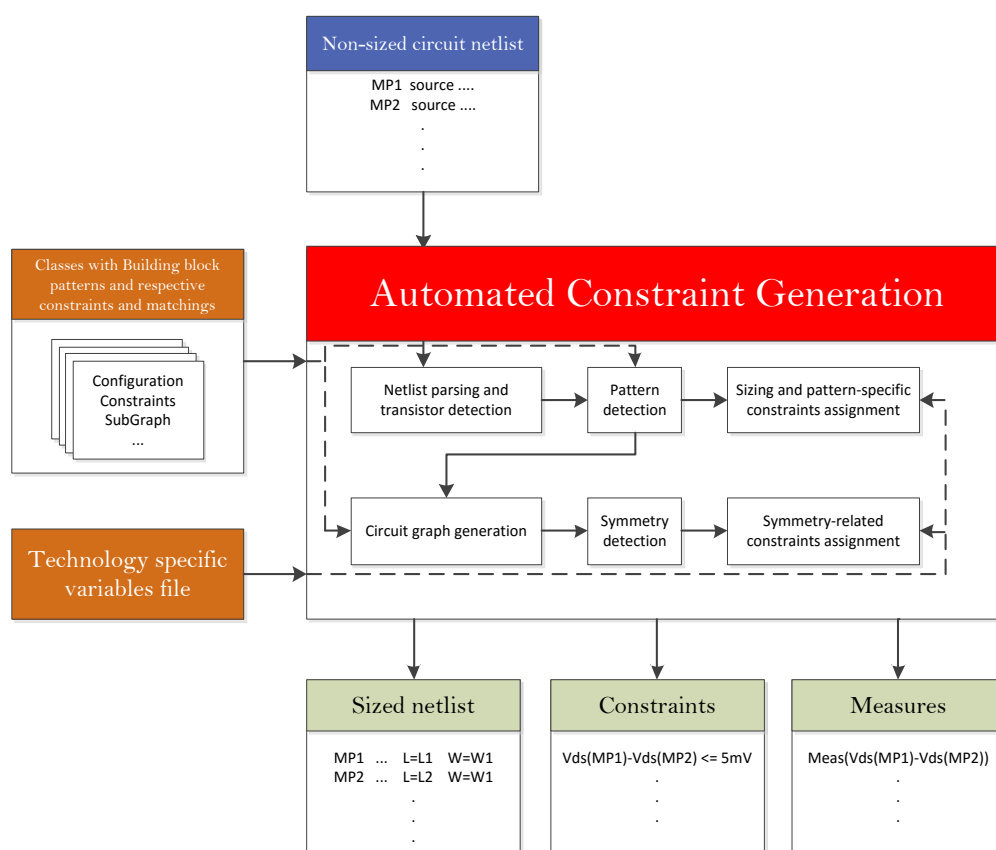


Figure 4.1 – Automated Constraint Generation Architecture

## 4.2 Reading the netlist

The netlist format for transistors is as follows:

Mx          Drain          Gate          Source          Bulk          Model          W=MW          L=ML          M=MM

The transistors are stored in a data structure with several variables as shown in Figure 4.2. Most of the variables used in the structure are Strings, either to store the transistor name and model (Name, Model), the node names connected to the transistor's ports (Drain, Gate, Source, Bulk), or the sizing variables (Width, Length, Fingers). Integers are used to assign symmetry and proximity groups to the transistor (SPID, SGID, GID), as these only utilize numbers. A state variable is also used to determine if the transistor should be working in the saturation or in the linear region (State).

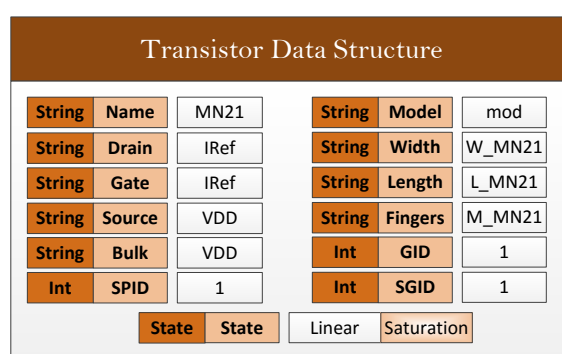
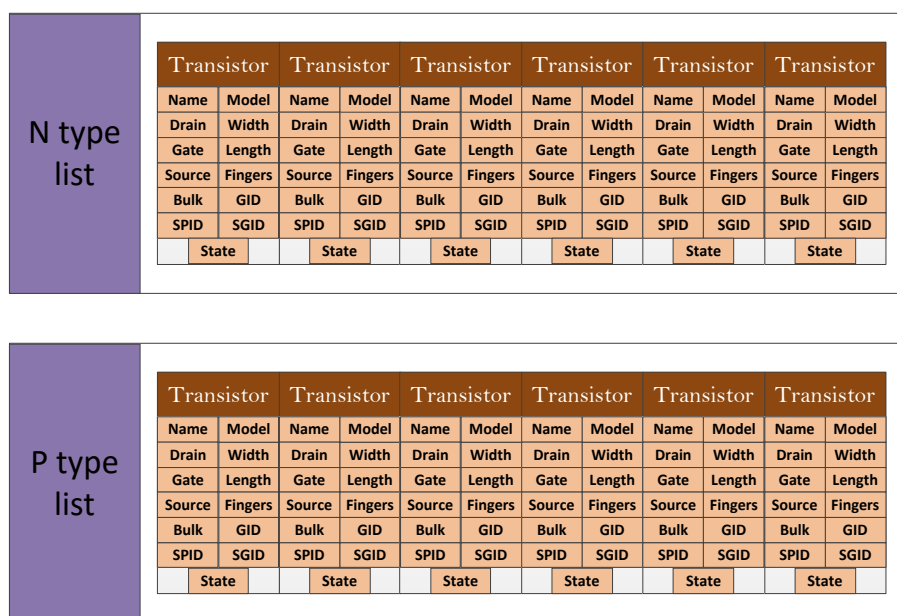


Figure 4.2 – Transistor Data Structure with field examples

Two lists of transistors are created, one list for p-type transistors and the other for n-type transistors, as shown in Figure 4.3. Determining the type of transistor is made by checking the second character of the name, as it is always either P or N. Any line that is not commented nor is a valid transistor is stored in a list of strings to be placed in the output netlist.





**Figure 4.3 – Transistor lists according to type**

Although the original netlist might be parameterized, the module ignores the transistors dimensions, and only reads the transistor's name, model, and port nodes. All information after that is discarded. Once the transistor is read, the w, l, and m variables are assigned according to the name. (e.g. MN6 will have W=W\_MN6, L=L\_MN6, M=M\_MN6).

### 4.3 Pattern configuration, constraints, and subgraphs

Patterns are implemented in files that contain all the information needed to characterize the intended building blocks. These are categorized in two levels, 1 and 2. Level 0 contains the single transistors, which are not patterns.

#### 4.3.1 Level 0

Level 0 building blocks are the individual transistor, required mostly to apply saturation or linear region constraints. Even though these are the simplest building blocks, they are also very common and are where most of the constraints are generated. Should a transistor not be matched into a building block, it is set as being in saturation by default.

##### 4.3.1.1 Transistor

Transistor-assigned constraints come mainly from the state they are in (saturation/linear). The state is assigned by the building block the transistor is in. If the transistor is not found to be in any building block then it is considered to be in saturation by default. The constraints generated to transistors in saturation are as follows:

$$v_{ds} - (v_{gs} - V_{th}) \geq V_{SAT_{min}} \quad (4.1)$$

$$v_{ds} \geq 0 \quad (4.2)$$

$$v_{gs} - V_{th} \geq 0 \quad (4.3)$$

$$w \times l \geq A_{min_{SAT}} \quad (4.4)$$

$$w \geq w_{min_{SAT}} \quad (4.5)$$

$$l \geq L_{min_{SAT}} \quad (4.6)$$

The constraints generated for transistors that are working in the linear region generate the following constraints:

$$(v_{gs} - V_{th}) - v_{ds} \geq V_{lin_{min}} \quad (4.7)$$

$$v_{ds} \geq 0 \quad (4.8)$$

$$v_{gs} - V_{th} \geq 0 \quad (4.9)$$

### 4.3.2 Level 1

Level 1 patterns consist of two transistors that are connected according to a given configuration. These configurations often imply some constraints to aid in transistor matching and improve circuit performance, as well as aiding in a simpler way to represent circuits. Although some patterns do not have any constraints associated, they are used to help detecting higher level patterns.

Level 1 Pattern structure		
String	Name	SCM1
Transistor	A	MN21
Transistor	B	MN22

Figure 4.4 – Level 1 Building Block Data Structure with field examples

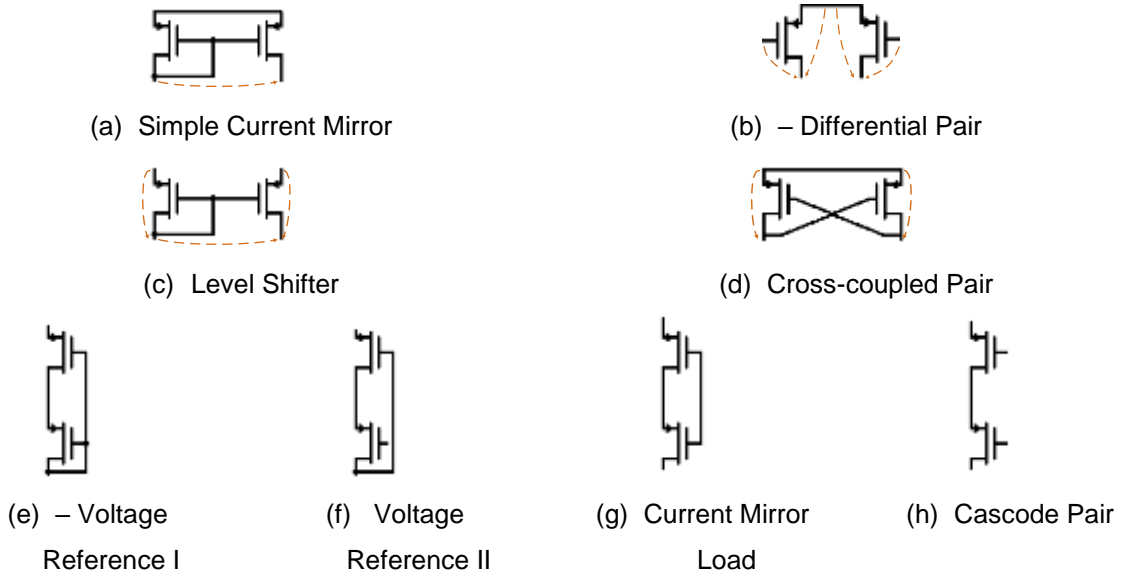


Figure 4.5 – Level 1 Patterns and Subgraphs

#### 4.3.2.1 Simple Current Mirror

The Simple Current Mirror is the basic building block of current mirrors, providing a constant current equal to a reference current multiplied by a desired ratio,  $R$ . The pattern's configuration consists of two transistors with a connection between their sources and another connection between their gates and one of the drains. Constraints will set the transistors to match so that there is enough precision in the output current, and the widths are also matched, leaving the currents' ratio to be set by the number of fingers. Both transistors are set as saturated.

$$l_1 = l_2 \quad (4.10)$$

$$w_1 = w_2 \quad (4.11)$$

$$m_2 = m_1 \times R_{1,2} \quad (4.12)$$

$$v_{gs_{1,2}} - V_{th_{1,2}} \geq V_{gs_{min}} \quad (4.13)$$

The SRM [16] method included a small difference between the Drain-Source voltages of the transistors. The optimizer was not finding good circuit solutions with this constraint, mainly because this forces a voltage at the current source node, which is not intended. The intention of the Simple Current Mirror is to establish a current. This constraint would essentially create a voltage source, rather than a current source.

The subgraph associated with this pattern consists of a single edge pointing from the driver's drain to the driven's drain.

#### 4.3.2.2 Level Shifter

The Level Shifter has a similar configuration as the Simple Current Mirror, with the difference that the sources are not connected. This means that the configuration of a Level Shifter consists of a single connection between the transistors gates as well as one of the transistor's drain. Constraints are also similar.

$$l_1 = l_2 \quad (4.14)$$

$$w_1 = w_2 \quad (4.15)$$

$$m_2 = m_1 \times R_{1,2} \quad (4.16)$$

$$v_{gs_{1,2}} - V_{th_{1,2}} \geq V_{gs_{min}} \quad (4.17)$$

Unlike the Simple Current Mirror, the Level Shifter is not expected to be connected to vdd or vss, so the sub-graph is more elaborate. Although HPR [52] has a much more complex sub-graph, which was not adopted. Although the signal flow is of that complexity, this sub-graph should be enough to find circuit symmetries.

#### 4.3.2.3 Voltage Reference I

The Voltage Reference I pattern does not generate any constraint or subgraph, and its only utility is to assist in determining level 2 building blocks. The configuration consists of two transistors with the drain of one (1) connected to the source of the other (2), as well as a connection between their gates and the remaining drain (2).

#### 4.3.2.4 Voltage Reference II

The Voltage Reference II pattern also does not generate any constraint or subgraph, and its only utility is to assist in determining level 2 building blocks. The configuration consists of two transistors with the drain of one (1) connected to the source of the other (2), as well as a connection between one gate (1) and the remaining drain (2).

#### 4.3.2.5 Current Mirror Load

The Current Mirror Load does not generate any constraint or subgraph, also only having the utility of generating level 2 building blocks. The configuration consists of two transistors with a connection between one drain and one source and another connection between their gates.

#### 4.3.2.6 Cascode Pair

The Cascode Pair is one of the most generic patterns, and is the last one on the list of no-constraints patterns. The configuration consists of a single connection between a drain and a source.

#### 4.3.2.7 Differential Pair

The Differential Pair is the building block in circuits that reads a small differential input that is amplified into a larger voltage differential output. To ensure that both the positive and negative parts of the voltage, the building block should be symmetric, i.e. both transistors need to have the same dimensions. Because this pattern is very generic, false detections are possible, but because this pattern requires the sources to be connected to a current source, a verification of such connection is made once all the building blocks are detected, easing the determination process of false detections. The configuration consists of a single connection between the transistors' sources. Constraints for this pattern ensure that both transistors are working in similar conditions (same dimensions and same working state) so that a symmetric behavior is achieved:

$$l_1 = l_2 \quad (4.18)$$

$$w_1 = w_2 \quad (4.19)$$

$$|v_{ds_2} - v_{ds_1}| \leq \Delta V_{ds_{\max}(dp)} \quad (4.20)$$

$$|v_{gs_2} - v_{gs_1}| \leq \Delta V_{gs_{\max}} \quad (4.21)$$

The sub-graph of this pattern consists of two edges for each transistor, one from the gate to the drain, and another from the source to the gate.

#### 4.3.2.8 Cross-coupled Pair

The Cross-coupled Pair works as a Differential Pair where one input is connected to the opposite side output. This creates an oscillating voltage at the outputs, as this is used mainly in voltage-controlled oscillators. The configuration consists of a connection between two sources, and two symmetric connections between one transistor's gate and the other transistor's drain. This pattern only generates transistor sizing constraints, and no electrical constraints other than setting both transistors as saturated.

$$l_1 = l_2 \quad (4.22)$$

$$w_1 = w_2 \quad (4.23)$$

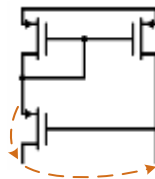
The Cross-coupled Pair has a subgraph consisting of one edge from each transistor's source to its own drain.

### 4.3.3 Level 2.

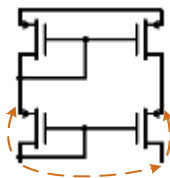
Level 2 Pattern structure		
String	Name	CCM1
Level 1 BB	A	SCM1
Level 1 BB	B	LS1
Transistor	C	MN22

**Figure 4.6 - Level 2 Building Block Data Structure with field examples**

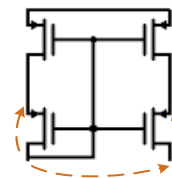
The level 2 patterns consist of current mirrors constructed by three or four transistors. This approach requires level 1 building blocks to generate level 2 building blocks, and creates further constraints. All the level 2 building blocks have the same subgraph, and only the first will be displayed. All the others will not be displayed, as the subgraphs are similar.



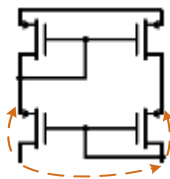
(a) Wilson Current Mirror



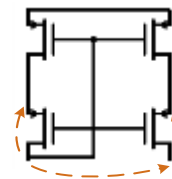
(b) Cascode Current Mirror



(c) 4-Transistor Current Mirror



(d) Improved Wilson Current Mirror



(e) Wide Swing Cascode Current Mirror

**Figure 4.7 – Level 2 patterns and subgraphs**

#### 4.3.3.1 Wilson Current Mirror

The Wilson Current Mirror is the only level 2 pattern consisting of three transistors, working as a regular Simple Current Mirror with an additional transistor connected to the driver transistor. However, the driven and driver transistor roles are reversed. The Wilson Current Mirror consists of a Simple Current Mirror and a single transistor, where the single transistor's source is connected to the SCM's driver's drain, and the single transistor's gate is connected to the SCM's driven's drain. This pattern does not generate any constraint additional to the Simple Current Mirror, with the exception that the single transistor is set as saturated.

This pattern's subgraph consists of an edge from the single transistor's drain to the source, and from the single transistor's drain to the gate.

#### 4.3.3.2 Cascode Current Mirror

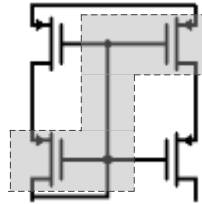
The Cascode Current Mirror is the standard current mirror composed of four transistors. The pattern consists of a Simple Current Mirror and a Level Shifter. The configuration establishes a connection between the SCM's driver transistor drain and the LS's driver transistor source, and a similar connection between both driven transistors. The constraints generated consist of similar Drain-Source voltages on the SCM transistors, so the Level Shifter's transistors are working at similar voltage points.

$$|v_{ds_2} - v_{ds_1}|_{(cm)} \leq \Delta V_{ds_{\max}(cm)} \quad (4.24)$$

The subgraph consists of two edges from the drain to the source of each Level Shifter transistor, as well as another edge from the Level Shifter's driver transistor drain to the driven transistor drain.

#### 4.3.3.3 4-Transistor Current Mirror

The 4-Transistor Current Mirror consists of four transistors with all their gates and one of the "Level Shifter's" drain connected. Although the transistor sizing constraints and subgraph generated are similar to the CCM, their detection is made by matching a Voltage Reference I to a Current Mirror Load, and the two upper transistors are set as working in the linear region. Observing this pattern's configuration shows that there is a Level Shifter false detection shown in Figure 4.8, which must be removed.



**Figure 4.8 – Level Shifter false detection in a 4-Transistor Current Mirror**

The transistors shown in the shade area meet the connection configuration of a Level Shifter. However, the transistors are not to be matched.

#### 4.3.3.4 Improved Wilson Current Mirror

Similar to the Cascode Current Mirror, the Improved Wilson Current Mirror consists of a Simple Current Mirror and a Level Shifter, with the difference that the connections configuration is reversed. However, the constraints and subgraph are similar to the Cascode Current Mirror.

#### 4.3.3.5 Wide Swing Cascode Current Mirror

The Wide Swing Cascode Current Mirror consists of four transistors with a configuration similar to the 4-Transistor Current Mirror, with the difference that the two lower transistor gates are not connected to the upper two transistor gates. Although the transistor sizing constraints and subgraph generated are similar to the CCM, their detection is made by matching a Voltage Reference II to a Cascode Pair, and the two upper transistors are set as working in the linear region.

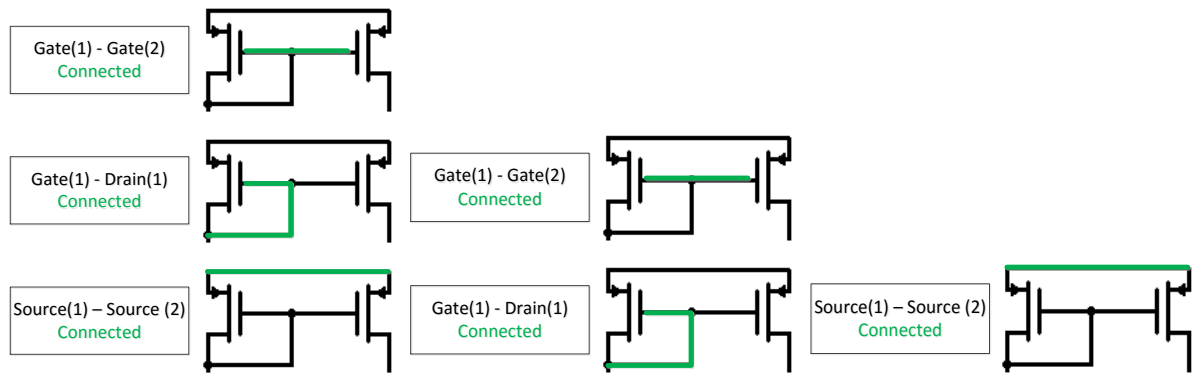
## 4.4 Pattern Detection

All constraints are based on either certain sub-circuits that are commonly found throughout a large variety of analog circuit designs, also known as Building Blocks. These consist mainly of differential pairs and current mirrors. For any given circuit, detecting these building block will aid in generating the desired constraints, as well as a signal flow graph to detect symmetries.

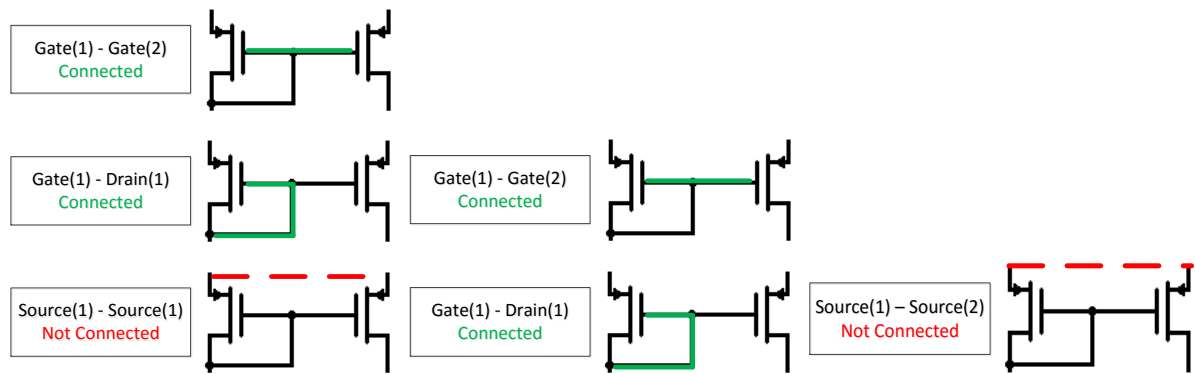
Given the configuration of some patterns, certain transistor pairs that match certain patterns will also match other patterns. Despite this, the SRM [16] approach searches for possible transistor pair matches for each pattern. Considering that a given pair of transistors never becomes more than one building block, and that certain patterns will also match to other patterns (as mentioned previously), this method might not be the most efficient, requiring a follow-up search to remove certain building blocks that share the same transistor pair as other building blocks.

The search method implemented, on the other hand, does not require this after step, because searches are made for possible patterns for each given transistor pair, making some ambiguities easier to deal with. However, the order in which the pattern matching is done for each pair has to be considered, or there is a risk that the transistor pair is attributed the wrong pattern.

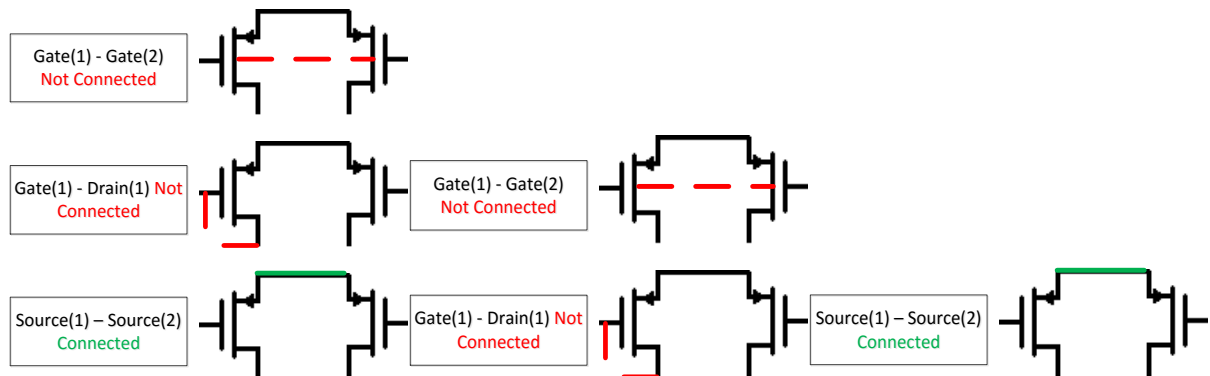
For example, a Level Shifter consists of two transistors with their gates and one of the drains connected. A Differential Pair consists of two transistors with their sources connected. A Current Mirror consists of two transistors with their gates and one of the drains connected, and also their sources connected. This means that if a Differential Pair setup is checked first, all Current Mirrors will be recognized as being Differential Pairs, and the same thing happens if Level Shifter is checked first. To avoid this problem, the Current Mirror setup must be checked before the Level Shifter and the Differential Pair. Figure 4.9 illustrates the problem described. The column on the left shows the connection verifications for the Simple Current Mirror, the middle column shows the connection verifications for the Level Shifter, and the column on the right shows the connection verifications for the Differential Pair.



(a)



(b)



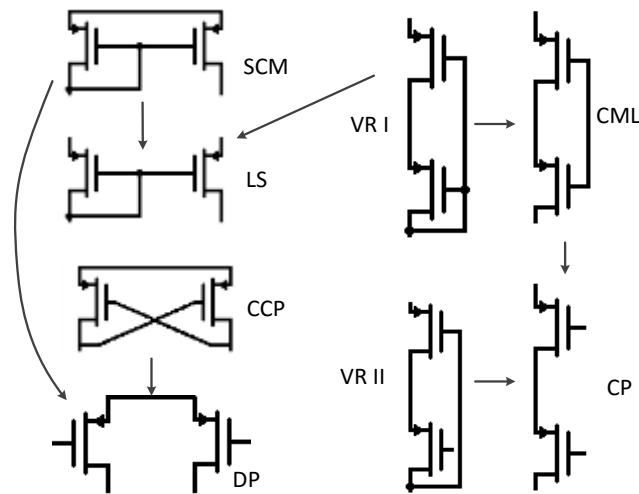
(c)

**Figure 4.9 – Pattern detection ambiguity examples with (a) Simple Current Mirror (b) Level Shifter (c) Differential Pair**

The illustration shows that a Simple Current Mirror will verify all Level Shifter and Differential Pair connections. However, neither the Level Shifter nor the Differential Pair will be verified in the Simple Current Mirror pattern check.

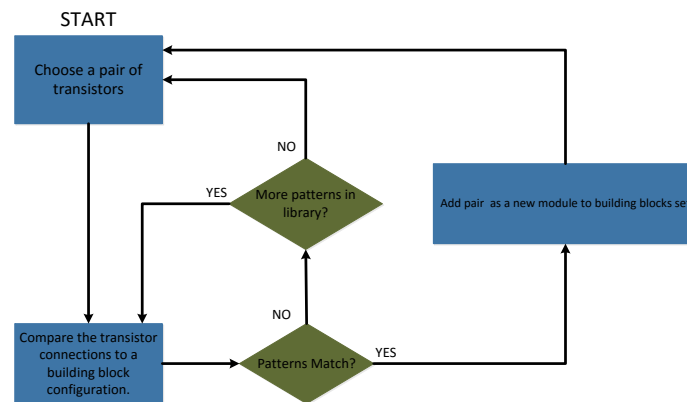
A complete hierarchy is shown in Figure 4.10, showing which patterns must be compared before other patterns. The arrows represent which patterns must be compared beforehand.





**Figure 4.10 – Hierarchy in building block search**

After reading the netlist and establishing the transistor database, a search for patterns begins, comparing each combination of transistor and matching to patterns in the library, generating the first level of building blocks of the circuit.



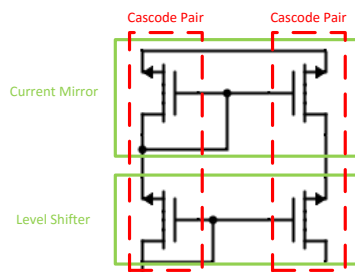
**Figure 4.11 - Flowchart of implemented building block recognition**

Because some level 1 building blocks have a very generic configuration, many detected building blocks may be false positives (for example, the Cascode Pair is just a transistor's drain connected to another transistor's source). To determine the false detections, the level 2 building blocks are determined first.

Once the first-level building blocks of a circuit are determined, the level 2 building blocks are generated through a similar search, where the search is done for each combination of building blocks, or combination of building blocks and transistors, and are matched with the level 2 patterns. Some level 1 ambiguous building blocks can then be determined to be false detections and removed.

Some level 1 patterns do not have constraints associated to them, so their only function is to generate into level 2 building blocks. In the case one of the building blocks does not generate any level 2 building block, nor generates any constraint, then the building block has no utility in the module. Once the level 2 building blocks are detected, a search is made to detect the level 1 building blocks that do

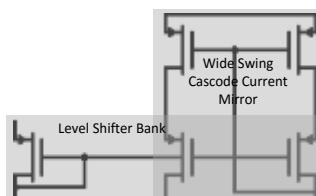
not have constraints and did not generate any higher level building block, and these blocks are removed, considering that they are highly likely to be false detections. For example, as shown in Figure 4.12, a Cascode Current Mirror, the correctly detected level 1 building blocks are the Current Mirror, and the Level Shifter. However, due to the configuration of the transistors, two Cascode Pairs are detected, which is clearly a false detection. Once the level 2 building blocks are found, the program realizes the Cascode Pairs did not generate any level 2 building block and removes them. Figure 4.3 shows a graphical representation of the CCM example, with the correct level 1 building blocks in green, and the incorrect in red.



**Figure 4.12 - Example of false level 1 detections**

Lastly, the Differential Pair is the last level 1 pattern that while having constraints, also has a generic configuration, which is prone to many false detections. But because most Differential Pairs have their sources connected to a current source, each Differential Pair detected is checked if their sources are connected to a port of any kind of current mirror. If no current mirror is found to be connected to their sources, the building block is considered a false detection and is discarded.

Another possible way to detect false detections is by searching building blocks that share transistors. Although this is a possible way of removing false detections, some building blocks are still valid even when sharing transistors with other building blocks, as is shown in Figure 4.13. This method would also not remove invalid building blocks that happen to not share transistors with any other building block.



**Figure 4.13 – Level Shifter Bank sharing transistors with a Wide Swing Cascode Current Mirror**

The above example shows a case where two different building blocks have two transistors in common, both being valid. This configuration will be later found in one of the example circuits in the results section. The Level Shifter driver transistor is being used to polarize the lower transistor gates of the Wide Swing Cascode Current Mirror.

## 4.5 Finding symmetry

### 4.5.1 Generating the signal flow graph

After determining all the building blocks in a circuit, a signal flow graph is created to aid in finding symmetries in the circuit. Each pattern brings an associated graph, representing the signal flow as well as the current flow. Unlike HPR [52] and MARS [72], which symmetries are applied at the block level (p.e, determining two cascode current mirrors as symmetric would mean that they would have two symmetric pairs of transistors), the approach used determines symmetry at the transistor level, reducing the implementation complexity of determining symmetric devices, and potentially increasing flexibility of new building block implementations in finding symmetries.

The nodes of the graph correspond to nodes in the circuit, and edges correspond to the flow of signal or current. To better facilitate symmetry detection, the edges are attributed a string variable to store characteristics: if the edge is just representing signal or is also a current path, the name of the originating pattern, and if the pattern has transistors of n-type or p-type.

Usage of current representing edges will help determine self-symmetric nodes and devices, i.e. devices that are placed or centered on the symmetry axis.

### 4.5.2 Finding symmetric nodes

Search for symmetric nodes is the same as the HPR [52] approach. Starting at the found Differential Pair, the gate nodes of the transistors are considered symmetric.

The search for additional symmetric nodes starts with a pair that is already determined to be symmetric, and each node of the pair has similar edges pointing to (or from) another pair of nodes. The process repeats for the new found symmetric pair.

If a pair of nodes has multiple similar edges each, each combination is tested with a verification if the paths further in the graph are also symmetric. If the paths in the graph merge into a single node then the entire path is considered symmetric. If the paths are similar in their paths but do not merge then are also considered symmetric. Otherwise, they are not considered symmetric and a different combination of edges is tested.

If a symmetric pair of nodes has edges pointing to/from a single node, and the edges represent current pathways, then the node is considered self-symmetric. If the node has further single edges representing current, the self-symmetry propagates. If the node has two similar edges representing current, and diverges to two nodes, then the nodes are considered symmetric and a regular symmetry search continues.

The method used includes backtracking, which was not implemented in HPR[ref]. This aids in determining further symmetries that would otherwise not be detected.

### 4.5.3 Determining symmetric devices

After determining all the symmetric nodes in a circuit, finding symmetry at the device level becomes a simple task. For each possible pair of transistors, the ports are checked if they are connected to the same node, or connected to symmetric nodes. If this condition applies to all the ports of a given pair of devices, they are considered to be symmetric..

## 4.6 Generating constraints

### 4.6.1 Sizing and electronic constraints

Sizing and electronic constraints are determined from the building blocks and the patterns' associated constraints. The main difference is that because AIDA also designs multiple finger transistors, some current mirror building blocks have their transistor widths (per finger) matched (i.e equal), with the number of fingers determining the current ratio. For example, the simple current mirror, constraints are as follows:

$$w_1 = w_2$$

$$\frac{i_2}{i_1} = \frac{m_2}{m_1}$$

Where  $w$  is the finger width,  $i$  is the source-drain current, and  $m$  is the number of fingers of a transistor. A transistor can be considered to have a total width of  $w \times m$ .

Whenever a new sizing match is applied, a search is made throughout the transistor database to assign the new parameter to transistors that share the same parameter to be changed. For example, M2 needs to have the same length as M1, so M2 will have  $L=L\_M1$ . If there is a M3 that also needs to have the same length as M2, then M2 will have  $L=L\_M3$ , and the transistor database will also be searched for transistors that have  $L=L\_M1$ , so transistor M1 will now have  $L=L\_M3$ .

### 4.6.2 Proximity constraints

HPR [52] mentions two levels of proximity: a low priority proximity for any devices that are connected with each other, and a higher priority proximity for certain building blocks.

Because AIDA considers single proximity groups, only the higher priority proximity is considered, because:

-In a given circuit, if a low proximity constraint translates to including two any transistors that are connected in the same group, the whole circuit will become a proximity group.

-There is only a single proximity grouping method, so the highest priority proximity is considered.

The higher proximity constraints refer to devices in the same building.

### 4.6.3 Symmetry constraints

Multiple lists of symmetry pairs might be generated. Each list consists of all the symmetry pairs that share the same symmetry axis. Symmetric transistors are matched to have the same Length, Width and number of fingers (geometric symmetry)

## 4.7 Designing new patterns

The Automated Constraint Generation module allows for new patterns to be added to the library, as well as editing or removing existing ones. The designer has the possibility of creating patterns by indicating the connections configuration, constraints it generates, and the signal and current subgraph.

### 4.7.1 Connections configuration

Because the transistor structure contains the node information stored as a String with the node name, connections are found by comparing two port strings and checking if they match. The module is run in Java, so the checks have the following template:

```
if(transistorA.portA.equals(transistorA.portB) && transistorA.portC.equals(transistorB.portA)):
```

This template checks if transistor A has ports A and B connected, and if transistor A has port C connected to transistor B port A.

One additional detail when checking if a pair of transistors match into a pattern, is if the pattern is symmetrical or not. If the pattern is symmetrical, only one check is necessary, otherwise a second check with the transistors swapped must be made. For example, if transistors M5 and M6 are being checked to see if they form a Simple Current Mirror, with transistor A as driver transistor, and transistor B as driven transistor, then a check is made with M5 as transistor A and M6 as transistor B, and then another check is made with M6 as transistor A and M5 as transistor B.

When dealing with higher level patterns, these are formed by at least other building blocks. The check is similar to only transistors, except the connections require specification of the building block's transistor in addition to the transistor's port, in addition to checking the building block's assigned pattern (e.g when checking for a Cascode Current Mirror, the building blocks are analyzed to check if their patterns are Simple Current Mirrors and Level Shifters). Below is a template featuring patterns of level higher than 1.

```
if(bblockA.namebblockA.equals(scm) && transistorA.portA.equals(bblockB.transistorA.portB && ...)
```

### 4.7.2 Constraints generation

Constraints are saved in arrays of 3 strings: variable name, whether it is a maximum or a minimum constraint, and the limit's value.

Variable name specifies if the constraint is a voltage between two ports in a transistor (e.g. VDS, VGS), voltage differential between two pairs of ports (e.g. psids, psigs, which represent the VDS or VGS difference between two transistors), length, width, or area.

Maximum or minimum is a string that contains either “LE” (Lower or Equal) or “GE” (Greater or Equal). As the name states, it is used to determine if the constraint is an upper or a lower limit.

The limit's value is determined from a variables file which is provided by the designer.

As an example, the constraint associated in a given transistor Mx in saturated mode, with  $\Delta > 0.01$  (where 0.01 is a number representing the lower limit for the delta voltage, and delta is  $v_{ds} - v_{gs}$ ), the constraint generated is as follows:

```
temp_constr[0] = "GE";  
temp_constr[1] = "0.01";  
temp_constr[2] = "delta_Mx"
```

### 4.7.3 Subgraph generation

The subgraph is used to show if a change in current/voltage in a given node will influence current/voltage change in another node. It is important to note that these are for signal flow graph. Edges representing current flow could also be used. These are mostly only between a given transistor's Drain and Source. Edges need to have information, stating whether signal or current is being represented, which N or P type of transistors are involved, and the pattern. The following example shows a signal flow edge added from a Simple Current Mirror

```
adj.addAdj(a.drain, b.drain, "s " + type + "-scm");
```

Another example showing a current and signal flow from a Differential Pair

```
adj.addAdj(a.drain, b.drain, "sc " + type + "-dp");
```

The type is not needed to be specified, as the information is already stored within each building block and is read from it.

### 4.7.4 Remaining details

Besides electrical configuration, constraints, and subgraph, some additional information is required.

The *update* function is used to add or change nodes in the circuit that are considered a current source. For example, a Simple Current Mirror will add the driven transistor's drain node to the list, and a Cascode Current Mirror will remove the Simple Current Mirror's node from the list and add the Level Shifter driven transistor's drain. The Cascode Current Mirror's code lists all the lines involved with this function:

```
srcList.remove(a.b.drain);  
  
if(!srcList.contains(b.b.drain)) srcList.add(b.b.drain);
```

The function *hasSource* returns true or false and is only used for patterns that require a current source to be in a certain node, such as a Differential Pair's source node. Patterns that do not require a current

source will just return *true*, and patterns that do require a current source will need to have a check if the required node is a current source or not, with the following line:

```
return (srcList.contains(a.source));
```

The function `hasConstraints` also returns true or false, depending on whether the pattern generates constraints or not.

If the pattern configuration is composed of two similar components and is symmetric only one of the configuration check is necessary. A second check with the transistor positions switched is not needed.

## 4.8 Integration with AIDA

Constraint input in AIDA varies, depending on the type of constraint. Sizing constraints, where transistor dimensions are parameterized are included in the netlist, so another netlist is created as the Automated Constraint Generation's output, which is fed as input to AIDA. Current and voltage constraints are generated as strings, both to declare the constraint variables, and to generate the measures. Symmetry and proximity constraints are also generated as strings, this time to declare symmetry and proximity groups, and are sent to AIDA-L, considering that these are layout constraints.

Figure 4.14 shows the AIDA interface with the Automated Constraint Generation module included. A new tab with the parameterized netlist is added, and the design XML will show the new constraints added. This allows the designer to verify the sized netlist and the generated constraints, and make changes if needed.

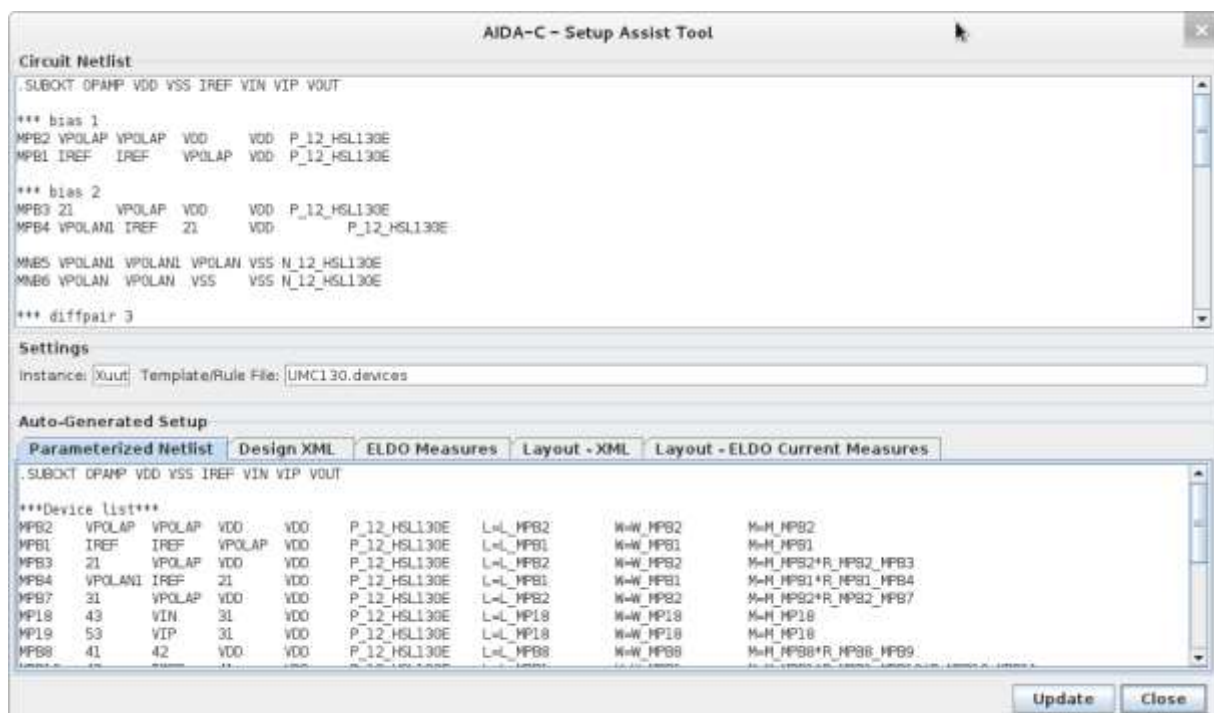


Figure 4.14 – AIDA Interface with the new module included

### 4.8.1 Netlist

As mentioned earlier, any transistor sizing in the netlist is ignored, and the module creates the size variables assigned to each transistor. Any non-empty string that is not a comment nor a transistor is stored in a string table to be reproduced in the output netlist, as it might contain other elements (e.g. capacitance) that are not considered in this module, or contain relevant commands.

The string table is split in two, one table for the strings before the transistor lines, and one table for the strings after the transistor lines. The module is assuming there are no other strings mixed in with the declaration of transistors, and any that is will be placed in the string table after the transistor lines.

Figure 4.15 shows the netlist from the circuit found in Figure 3.1, (a) being the module input netlist, and (b) the output, which is sent to AIDA.

```
*** Cell name: opamp_a
.SUBCKT OPAMP_A DD IN IP OUT REF SS
XCO0 OUT D12 SS momcaps_sy_mm l=_lc nf='_nfc'
MP8 REF REF DD DD P_12_HSL130E
MP5 NETZ52 REF DD DD P_12_HSL130E
MP6 OUT REF DD DD P_12_HSL130E
MP1 D11 IN NETZ52 DD P_12_HSL130E
MP2 D12 IP NETZ52 DD P_12_HSL130E
MN3 D11 D11 SS SS N_12_HSL130E
MN4 D12 D11 SS SS N_12_HSL130E
MN7 OUT D12 SS SS N_12_HSL130E
.ENDS*** End of subcircuit definition.
```

(a) Unparameterized netlist (*circuit.cir*)

```
*** Cell name: opamp_a
.SUBCKT OPAMP_A DD IN IP OUT REF SS
XCO0 OUT D12 SS momcaps_sy_mm l=_lc nf='_nfc'
MP8 REF REF DD DD P_12_HSL130E W=W_MP20 L=L_MP20 M=M_MP20
MP5 NETZ52 REF DD DD P_12_HSL130E W=W_MP20 L=L_MP20 M=M_MP14*R_MP20_MP14
MP6 OUT REF DD DD P_12_HSL130E W=W_MP20 L=L_MP20 M=M_MP22*R_MP20_MP22
MP1 D11 IN NETZ52 DD P_12_HSL130E W=W_MP11 L=L_MP11 M=M_MP11
MP2 D12 IP NETZ52 DD P_12_HSL130E W=W_MP11 L=L_MP11 M=M_MP11
MN3 D11 D11 SS SS N_12_HSL130E W=W_MN9 L=L_MN9 M=M_MN9
MN4 D12 D11 SS SS N_12_HSL130E W=W_MN9 L=L_MN9 M=M_MN9
MN7 OUT D12 SS SS N_12_HSL130E W=W_MN21 L=L_MN21 M=M_MN21
.ENDS*** End of subcircuit definition.
```

(b) Parameterized netlist (*circuit.cir*)

**Figure 4.15 – Unparameterized and parameterized netlists**

The parametrized netlist shows that the module attributes variable names according to the name of the transistor. Using this terminology to determine transistor variables will also aid the designer in finding the desired sizing equalities (e.g. transistor MP14 has  $L=L\_MP20$ , meaning that MP14 has the same length as MP20), in case the netlist needs to be read.

### 4.8.2 Electrical Constraints and Measures

Electrical constraints are required to set each transistor in the desired working state (saturation or linear), and to match transistors so the pairs are in the intended working conditions (e.g. the gate-source voltages are very similar, so the current per transistor width dimension ratios are equal).

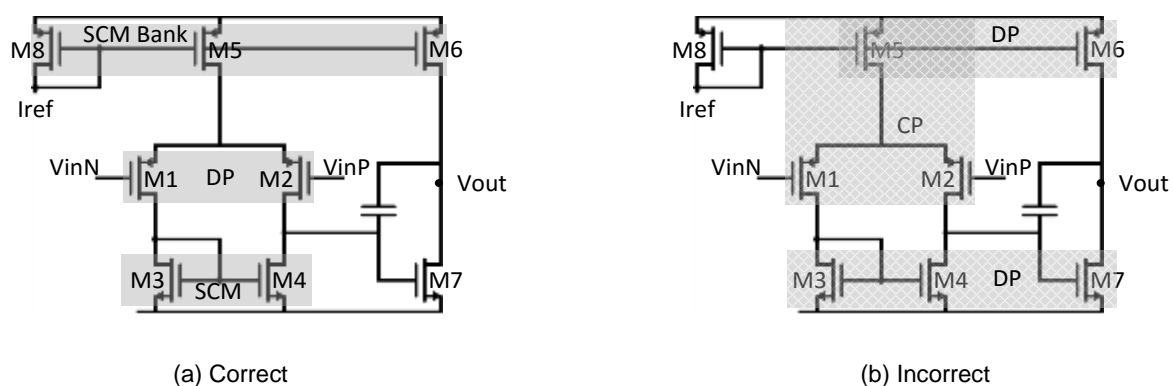


AIDA requires declaration of variables, so constraints can effectively be applied to them. After declaring a variable, it is assigned as a voltage between a pair of nodes, a voltage difference between two pairs of nodes, and the same applies to currents. Then the constraint is set with a lower limit, an upper limit, or both.

## 4.9 Running Example

The two-stage amplifier found in Figure 3.4 is used as an example to describe in detail the constraint generation process, being a circuit with a low degree of complexity and number of transistors.

Once the netlist is sent as input, the transistors are read and stored in their p-mos or n-mos list. Then, the level 1 building blocks are detected. Figure 4.16 shows all the Level 1 detected building blocks, with a detailed list in Table 4-1.



**Figure 4.16 – Two-Stage detected Level 1 building blocks**

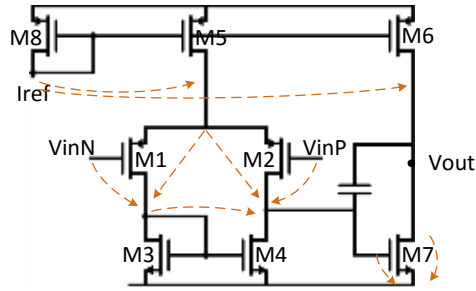
**Table 4-1 – List of Two-Stage detected Level 1 building blocks**

Building Block	Transistor 1	Transistor 2	Correct	Building Block	Transistor 1	Transistor 2	Correct
Simple Current Mirror 1	M8	M5	✓	Differential Pair 3	M3	M7	✗
Simple Current Mirror 2	M8	M6	✓	Differential Pair 4	M4	M7	✗
Simple Current Mirror 3	M3	M4	✓	Cascode Pair 1	M5	M1	✗
Differential Pair 1	M1	M2	✓	Cascode Pair 2	M5	M2	✗
Differential Pair 2	M5	M6	✗				

After determining the level 1 building blocks, the module does not differentiate between correct and incorrect detections, and goes on to search for the level 2 building blocks. However, due to the simplicity of the circuit, there is no level 2 building block to be detected. The search is still executed, but nothing is found.

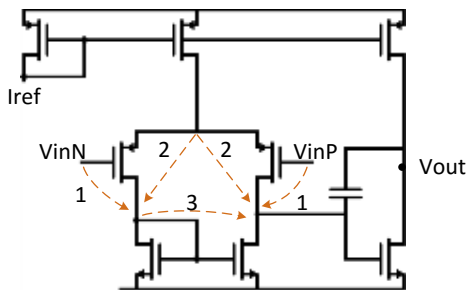
After determining the second level of building blocks, the incorrect detections can be found. Because none of the Cascode Pairs generate into a higher level building block, they are removed. And Differential Pairs 2-4 do not have their transistor sources connected to a current mirror/source, and are removed too.

Now the graph generation process begins. Each building block provides its own subgraph to be added to the circuit graph. Figure 4.17 shows the final circuit graph.

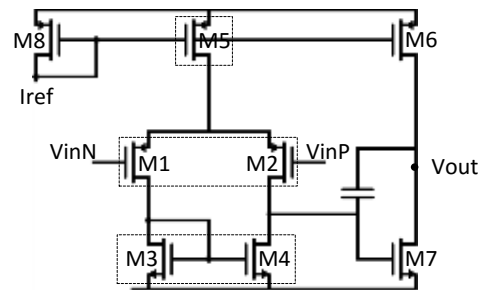


**Figure 4.17 – Two-Stage Signal and Current Flow Graph**

Once the circuit graph is generated, the search for symmetry begins. The search starts at the gate nodes of the Differential Pair, as the building block is assumed to be symmetric. The edges pointing from M4 and M5 gate's to their drains are matched and considered symmetric, and the drain nodes are considered a symmetric pair. Then, the edges from M4 and M5 sources to their drains are considered symmetric, and the source node (M4 and M5 have their sources connected to the same node) is considered self-symmetric. Because these edges merge into a single node, the module checks if they represent current, and considers the symmetry valid if they do. After all the symmetric edges and nodes are found, the device symmetry pairing is determined. Two devices are symmetric if their port nodes are either symmetric or connected to each other. Figure 4.18 shows the two stage edge and device symmetry pairs.



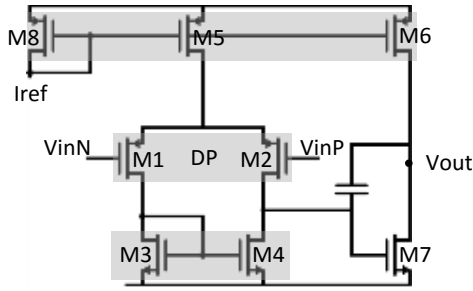
(a) Symmetric edges



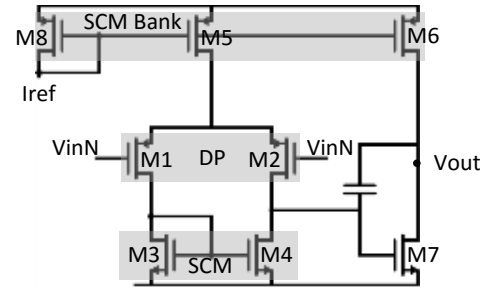
(b) Symmetric Devices

**Figure 4.18 – Two-Stage symmetry pairs**

All the building blocks and symmetric devices are now determined. Transistors are sized, proximity and symmetry groups are created, and electrical constraints are generated. Figure 4.19 shows the proximity groups and the matched devices. Both figures are similar, mostly due to the circuit being simple, and the only detected building blocks were the Simple Current Mirror and Differential Pair, both having matched devices as constraints, as well as proximity groups being directly determined from building blocks, while no building block sharing a transistor with another building block. Additionally, matched transistors have the same Length and Width per finger, but not the same amount of fingers.



(a) Proximity groups



(b) Matched devices

**Figure 4.19 - Two-Stage proximity groups and matched devices**

All constraints are sent in a text format to AIDA

```
<Variable name="W_MP8" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MP8" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MP8" min="1" step="1" max="8"/>
<Variable name="R_MP8_MP5" min="0.5" step="0.1" max="10"/>
<Variable name="R_MP8_MP6" min="0.5" step="0.1" max="10"/>
<Variable name="W_MP1" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MP1" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MP1" min="1" step="1" max="8"/>
<Variable name="W_MN3" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MN3" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MN3" min="1" step="1" max="8"/>
<Variable name="R_MN3_MN4" min="0.5" step="0.1" max="10"/>
<Variable name="W_MN7" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MN7" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MN7" min="1" step="1" max="8"/>
```

```
<MeasureDescription name="vov_MP8" description="Overdrive" units="[V]" />
<MeasureDescription name="vov_MP5" description="Overdrive" units="[V]" />
<MeasureDescription name="vov_MP8" description="Overdrive" units="[V]" />
<MeasureDescription name="vov_MP6" description="Overdrive" units="[V]" />
<MeasureDescription name="psiDS_MP1_MP2" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MP1_MP2" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="vov_MN3" description="Overdrive" units="[V]" />
<MeasureDescription name="vov_MN4" description="Overdrive" units="[V]" />
<MeasureDescription name="delta_MN3" description="VDS - VDSat" units="[V]" />
<MeasureDescription name="A_MN3" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MN3" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="delta_MN4" description="VDS - VDSat" units="[V]" />
<MeasureDescription name="A_MN4" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MN4" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="delta_MN7" description="VDS - VDSat" units="[V]" />
<MeasureDescription name="A_MN7" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MN7" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="vov_MN7" description="Overdrive" units="[V]" />
<MeasureDescription name="rev_delta_MP8" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MP8" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MP8" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="rev_delta_MP5" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MP5" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MP5" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="rev_delta_MP6" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MP6" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MP6" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="rev_delta_MP1" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MP1" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MP1" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="vov_MP1" description="Overdrive" units="[V]" />
<MeasureDescription name="rev_delta_MP2" description="VDSat - VDS" units="[V]" />
```

```

<MeasureDescription name="A_MP2" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MP2" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="vov_MP2" description="Overdrive" units="[V]" />
<MeasureDescription name="psiDS_MP1_MP2" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MP1_MP2" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_MN3_MN4" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MN3_MN4" description="VGS_X1 - VGS_X2" units="[V]" />

```

```

<Constraint op="GE" value="0.1" meas="vov_MP8" />
<Constraint op="GE" value="0.1" meas="vov_MP5" />
<Constraint op="GE" value="0.1" meas="vov_MP8" />
<Constraint op="GE" value="0.1" meas="vov_MP6" />
<Constraint op="LE" value="0.1" meas="psiDS_MP1_MP2" />
<Constraint op="LE" value="0.05" meas="psigs_MP1_MP2" />
<Constraint op="GE" value="0.1" meas="vov_MN3" />
<Constraint op="GE" value="0.1" meas="vov_MN4" />
<Constraint op="GE" value="0.1" meas="delta_MN3" />
<Constraint op="GE" value="6.0E-14" meas="A_MN3" />
<Constraint op="GE" value="0.00" meas="VDS_MN3" />
<Constraint op="GE" value="0.1" meas="delta_MN4" />
<Constraint op="GE" value="6.0E-14" meas="A_MN4" />
<Constraint op="GE" value="0.00" meas="VDS_MN4" />
<Constraint op="GE" value="0.1" meas="delta_MN7" />
<Constraint op="GE" value="6.0E-14" meas="A_MN7" />
<Constraint op="GE" value="0.00" meas="VDS_MN7" />
<Constraint op="GE" value="0.00" meas="vov_MN7" />
<Constraint op="GE" value="0.1" meas="rev_delta_MP8" />
<Constraint op="GE" value="6.0E-14" meas="A_MP8" />
<Constraint op="GE" value="0.00" meas="VDS_MP8" />
<Constraint op="GE" value="0.1" meas="rev_delta_MP5" />
<Constraint op="GE" value="6.0E-14" meas="A_MP5" />
<Constraint op="GE" value="0.00" meas="VDS_MP5" />
<Constraint op="GE" value="0.1" meas="rev_delta_MP6" />
<Constraint op="GE" value="6.0E-14" meas="A_MP6" />
<Constraint op="GE" value="0.00" meas="VDS_MP6" />
<Constraint op="GE" value="0.1" meas="rev_delta_MP1" />
<Constraint op="GE" value="6.0E-14" meas="A_MP1" />
<Constraint op="GE" value="0.00" meas="VDS_MP1" />
<Constraint op="GE" value="0.00" meas="vov_MP1" />
<Constraint op="GE" value="0.1" meas="rev_delta_MP2" />
<Constraint op="GE" value="6.0E-14" meas="A_MP2" />
<Constraint op="GE" value="0.00" meas="VDS_MP2" />
<Constraint op="GE" value="0.00" meas="vov_MP2" />
<Constraint op="LE" value="0.1" meas="psiDS_MP1_MP2" />
<Constraint op="LE" value="0.05" meas="psigs_MP1_MP2" />
<Constraint op="LE" value="0.1" meas="psiDS_MN3_MN4" />
<Constraint op="LE" value="0.05" meas="psigs_MN3_MN4" />

```

## **4.10 Observations**

As shown in a thorough description of the Automated Constraint Generation architecture, this module provides constraints to the designer by automating the constraint determination process. This generation also guarantees consistency between constraints in any given circuit, as different transistors within similar circuit configurations are met with similar constraints.

By reading the circuit netlist, the module reads all the transistors and electrical connections, compares multiple configurations, finds patterns in the circuits, makes a graph to find symmetries, and generates constraints according to the detected patterns and symmetries within the circuit.

An example has been shown to demonstrate how the module works step-by-step, using a simple Two-Stage OpAmp, and displaying the end result of the constraints generated in a compatible format with AIDA.



## 5 Experimental results

This section shows the detected patterns, generated graphs, and detected symmetries for a series of circuits. The circuit netlists were sent as input for the Automated Constraint Generation module, and all the expected building blocks were found, and their respective constraints generated. All circuits are composed of multiple patterns and a variety of level 2 building blocks. The circuits also have non-symmetrical sections to check how the symmetry detection behaves in non-symmetrical components.

### 5.1 Folded Cascode

Figure 5.1 shows the Folded Cascode, an Operational Amplifier that uses several current sources, as well as active loads that use existing current source patterns.

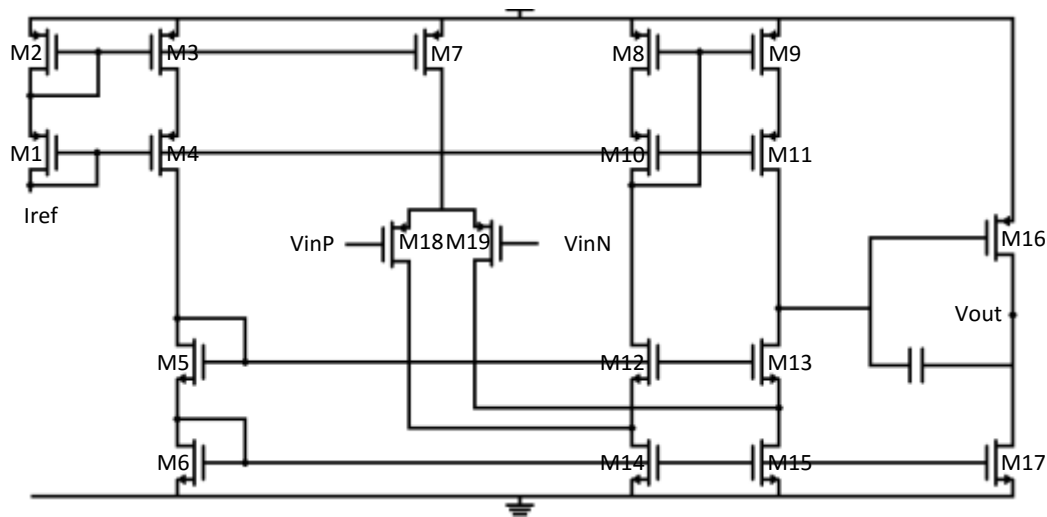
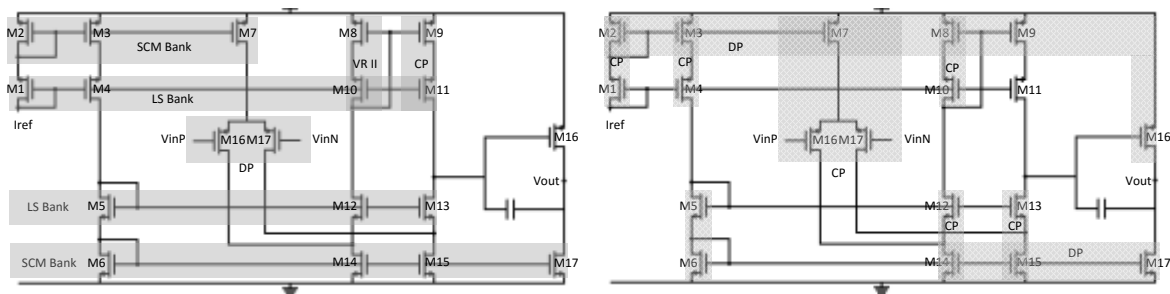


Figure 5.1 – Folded Cascode

Once the module reads the netlist and stores the transistor information, the first level patterns are found. Figure 5.2 shows the level 1 building blocks, with (a) being the correctly detected building blocks, and (b) the incorrect building blocks. In image (b), the Differential Pairs detected are every possible combination of transistors in the shaded group with the exception of detected Simple Current Mirrors, and the Cascode Pair is two building blocks, both being M7 and one for each of the Differential Pair's transistor.

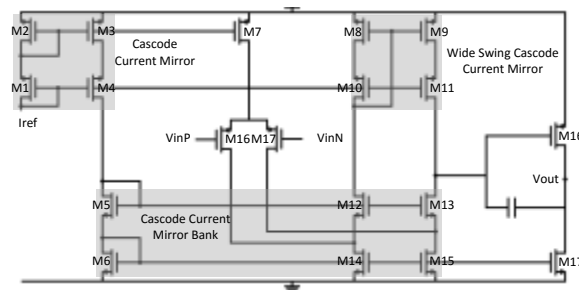


(a) Correct (b) Incorrect  
Figure 5.2 – Folded Cascode detected Level 1 building blocks (a) correct (b) incorrect

**Table 5-1 – Folded Cascode detected Level 1 building blocks**

Building Block	Transistor 1	Transistor 2	Correct	Building Block	Transistor 1	Transistor 2	Correct
Simple Current Mirror 1	M2	M3	✓	Differential Pair 10	M15	M17	✗
Simple Current Mirror 2	M2	M7	✓	Differential Pair 11	M2	M8	✗
Simple Current Mirror 3	M6	M14	✓	Differential Pair 12	M2	M9	✗
Simple Current Mirror 4	M6	M15	✓	Differential Pair 13	M2	M16	✗
Simple Current Mirror 5	M6	M17	✓	Differential Pair 14	M3	M16	✗
Level Shifter 1	M1	M4	✓	Differential Pair 15	M7	M16	✗
Level Shifter 2	M1	M10	✓	Differential Pair 16	M8	M16	✗
Level Shifter 3	M1	M11	✓	Differential Pair 17	M9	M16	✗
Level Shifter 4	M5	M12	✓	Differential Pair 18	M15	M17	✗
Level Shifter 5	M5	M13	✓	Cascode Pair 1	M9	M11	✓
Differential Pair 1	M16	M17	✓	Cascode Pair 2	M7	M16	✗
Differential Pair 2	M3	M7	✗	Cascode Pair 3	M7	M17	✗
Differential Pair 3	M3	M8	✗	Cascode Pair 4	M1	M2	✗
Differential Pair 4	M3	M9	✗	Cascode Pair 5	M3	M4	✗
Differential Pair 5	M7	M8	✗	Cascode Pair 6	M8	M10	✗
Differential Pair 6	M7	M9	✗	Cascode Pair 7	M5	M6	✗
Differential Pair 7	M8	M9	✗	Cascode Pair 8	M12	M14	✗
Differential Pair 8	M14	M15	✗	Cascode Pair 9	M13	M15	✗
Differential Pair 9	M14	M17	✗	Voltage Reference II 1	M8	M10	✓

At this stage, the module does not differentiate between correctly and incorrectly detected building blocks, and moves on to construct the second level on building blocks. The level 2 detections are shown in Figure 5.3.



**Figure 5.3 – Folded Cascode Level 2 detected building blocks**

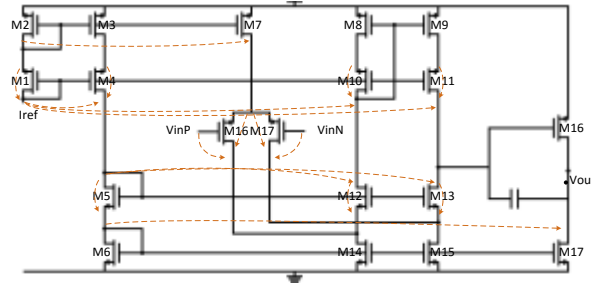
**Table 5-2 – Folded Cascode Level 2 detected building blocks**

Building Block	Building Block 1	Building Block 2
Cascode Current Mirror 1	Simple Current Mirror 1	Level Shifter 1
Cascode Current Mirror 2	Simple Current Mirror 4	Level Shifter 4
Cascode Current Mirror 3	Simple Current Mirror 5	Level Shifter 5
Wide Swing Cascode Current Mirror 4	Cascode Pair 1	Voltage Reference II

Once the level 2 building blocks are found, the false level 1 detections can be determined and removed. Because none of the Differential Pairs in Figure 5.2 (b) (2 to 10) have their transistors' sources connected to any Current Mirror, they are removed, and the Cascode Pairs do not generate any level 2 building block, and are also removed.

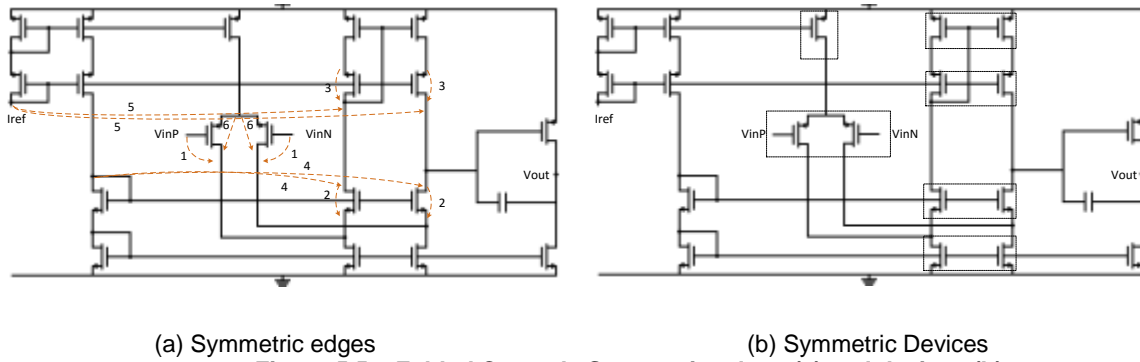
The remaining building blocks are then used to create the circuit's graph. Figure 5.4 shows the signal and current flow graph generated in the Folded Cascode circuit.





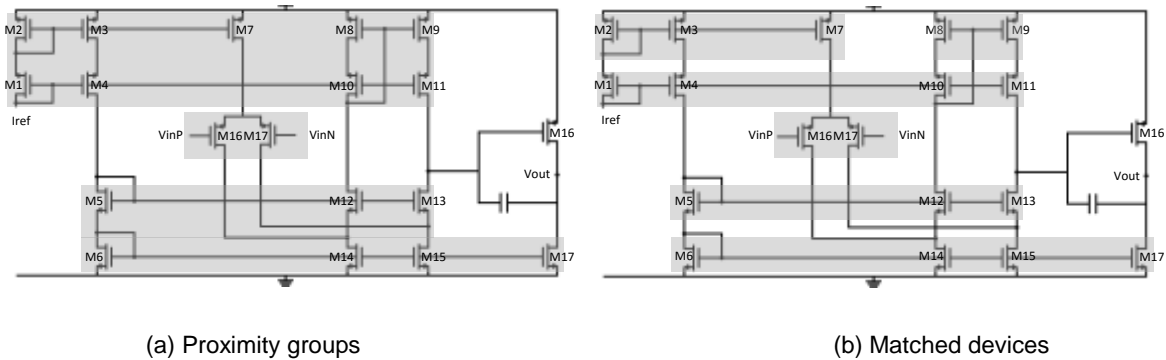
**Figure 5.4 – Folded Cascode Signal and Current Flow Graph**

The Differential Pair assigns the nodes  $V_{inP}$  and  $V_{inN}$  to be symmetric, and the symmetry search begins with those two nodes, following the algorithm explained in section 4. Figure 5.5 (a) shows the symmetric edges found in the circuit graph. Symmetric pairs have the same numbers. Figure 5.5 (b) shows the symmetric transistor pairs. The box with a single transistor means the transistor must be centered in the symmetry axis.



**Figure 5.5 – Folded Cascode Symmetric edges (a) and devices (b)**

Now the constraints are assigned according to the detected patterns and symmetries. The transistors are sized, the electrical constraints are generated for each transistor and for transistor pairs, and the proximity and symmetry groups are assigned for each transistor. Proximity groups are shown in Figure 5.6 (a), and transistors that have the same Widths (per finger) and Lengths are shown in Figure 5.6 (b). Transistors that have the same Widths or Lengths may or may not have different numbers of fingers between them.



**Figure 5.6 – Proximity groups (a) and matched devices (b)**

All constraints have been determined, and are sent to AIDA in the form of text. A thorough list of constraints can be found in Appendix A.1.

## 5.2 Fully Differential two-stage Folded Cascode

Figure 5.7 shows a fully differential two-stage Folded Cascode, found in [73]. This circuit consists of several current mirrors and has a differential section that is symmetrical even though most of the transistors do not fit into any of the building block patterns (M14, M15 and M18 to M23). The following figures show the same steps as the previous projects.

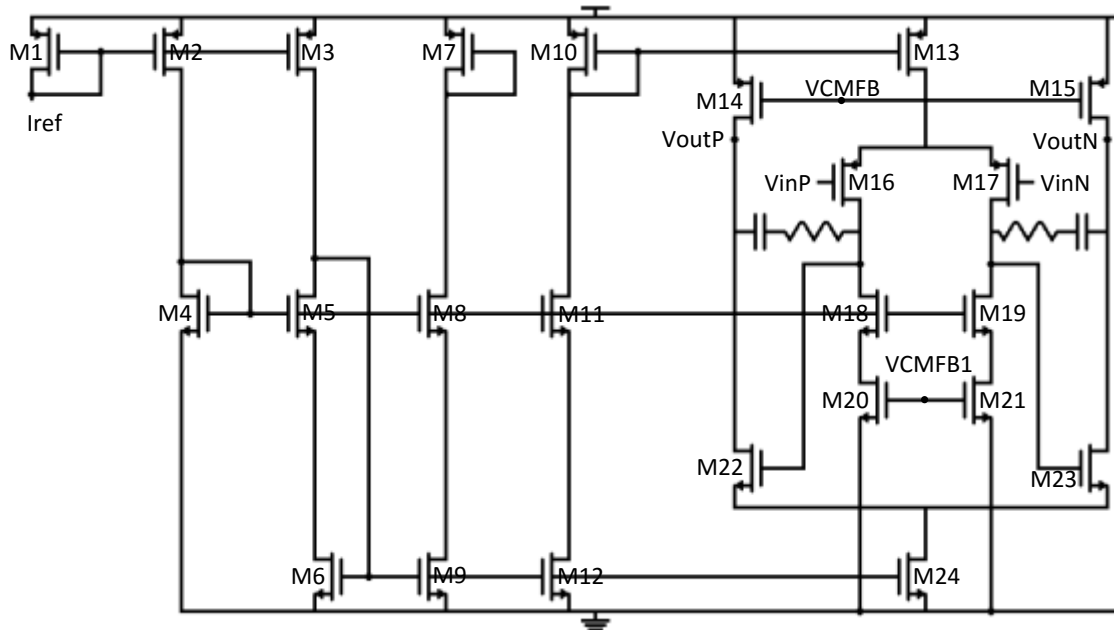
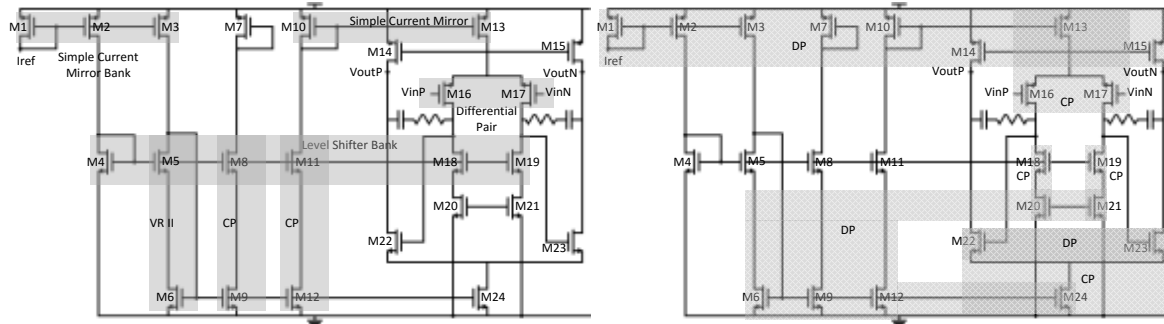


Figure 5.7 – Fully Differential two-stage Folded Cascode



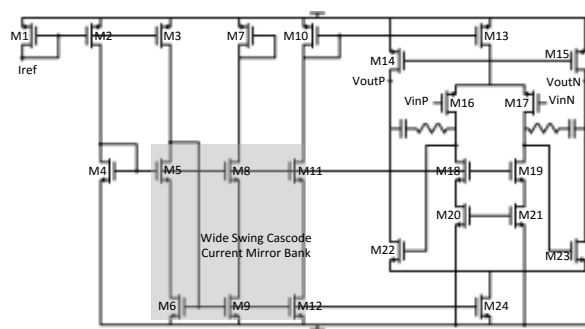
(a) Correct

(b) Incorrect

Figure 5.8 – Fully Differential two-stage Folded Cascode detected Level 1 building blocks (a) correct (b) incorrect

**Table 5-3 – OTA Edinei detected level 1 building blocks**

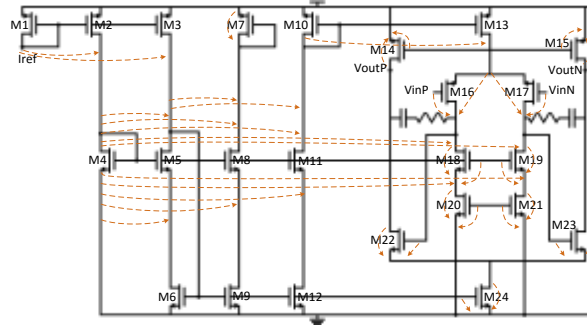
Building Block	Transistor 1	Transistor 2	Correct	Building Block	Transistor 1	Transistor 2	Correct
Simple Current Mirror 1	M1	M2	✓	Differential Pair 22	M10	M15	✗
Simple Current Mirror 2	M1	M3	✓	Differential Pair 23	M13	M14	✗
Simple Current Mirror 3	M10	M13	✓	Differential Pair 24	M13	M15	✗
Level Shifter 1	M4	M5	✓	Differential Pair 25	M14	M15	✗
Level Shifter 2	M4	M8	✓	Differential Pair 26	M6	M9	✗
Level Shifter 3	M4	M11	✓	Differential Pair 27	M6	M12	✗
Level Shifter 4	M4	M18	✓	Differential Pair 28	M6	M20	✗
Level Shifter 5	M4	M19	✓	Differential Pair 29	M6	M21	✗
Differential Pair 1	M1	M7	✓	Differential Pair 30	M6	M24	✗
Differential Pair 2	M1	M10	✗	Differential Pair 31	M9	M12	✗
Differential Pair 3	M1	M13	✗	Differential Pair 32	M9	M20	✗
Differential Pair 4	M1	M14	✗	Differential Pair 33	M9	M21	✗
Differential Pair 5	M1	M15	✗	Differential Pair 34	M9	M24	✗
Differential Pair 6	M2	M3	✗	Differential Pair 35	M12	M20	✗
Differential Pair 7	M2	M7	✗	Differential Pair 36	M12	M21	✗
Differential Pair 8	M2	M10	✗	Differential Pair 37	M12	M24	✗
Differential Pair 9	M2	M13	✗	Differential Pair 38	M20	M21	✗
Differential Pair 10	M2	M14	✗	Differential Pair 39	M20	M24	✗
Differential Pair 11	M2	M15	✗	Differential Pair 40	M21	M24	✗
Differential Pair 12	M3	M7	✗	Differential Pair 41	M22	M23	✗
Differential Pair 13	M3	M10	✗	Cascode Pair 1	M8	M9	✓
Differential Pair 14	M3	M13	✗	Cascode Pair 2	M11	M12	✓
Differential Pair 15	M3	M14	✗	Cascode Pair 3	M18	M20	✗
Differential Pair 16	M3	M15	✗	Cascode Pair 4	M19	M21	✗
Differential Pair 17	M7	M10	✗	Cascode Pair 5	M24	M22	✗
Differential Pair 18	M7	M13	✗	Cascode Pair 6	M24	M23	✗
Differential Pair 19	M7	M14	✗	Cascode Pair 7	M13	M16	✗
Differential Pair 20	M7	M15	✗	Cascode Pair 8	M13	M17	✗
Differential Pair 21	M10	M14	✗	Voltage Reference II 1	M5	M6	✓



**Figure 5.9 – Fully Differential two-stage Folded Cascode detected Level 2 building blocks**

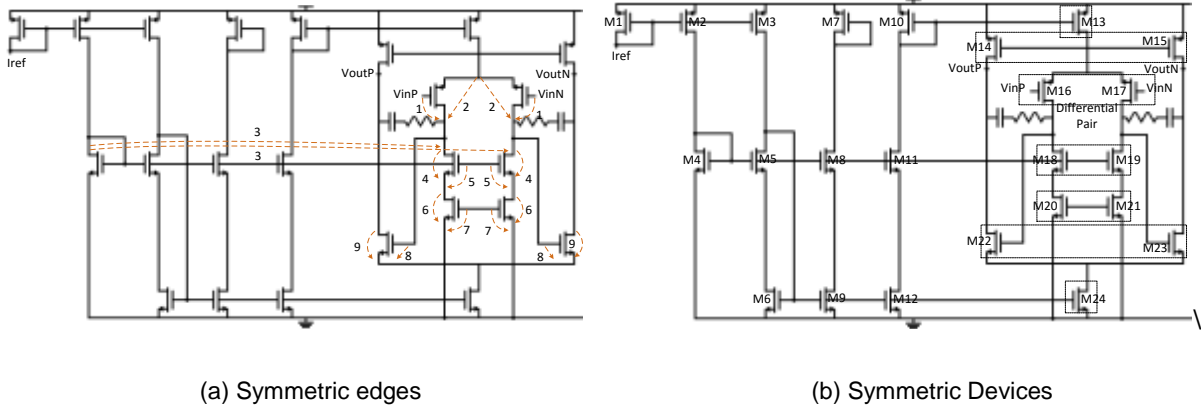
**Table 5-4 – Fully Differential two-stage Folded Cascode detected Level 2 building blocks**

Building Block	Building Block 1	Building Block 2
Wide Swing Cascode Current Mirror 1	Voltage Reference II 1	Cascode Pair 1
Wide Swing Cascode Current Mirror 2	Voltage Reference II 1	Cascode Pair 2



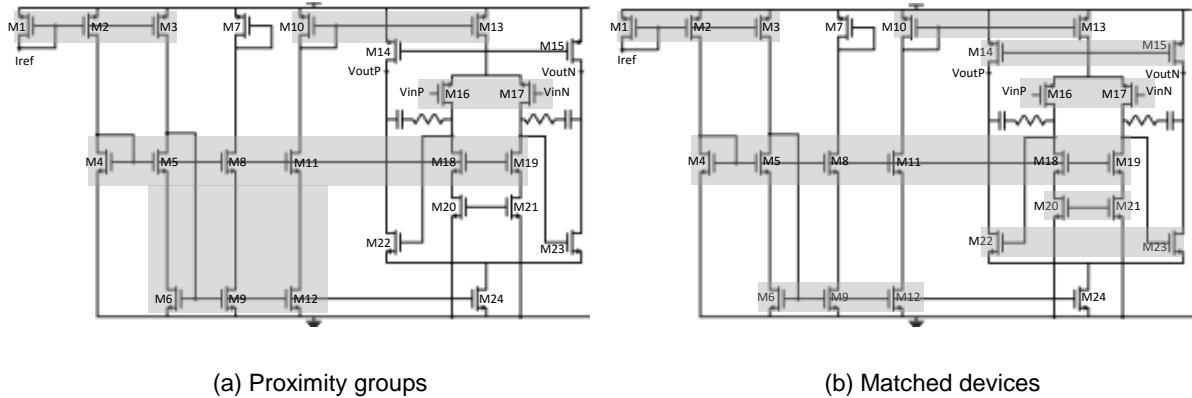
**Figure 5.10 – Fully Differential two-stage Folded Cascode Signal and Current Flow Graph**

Figure 5.11 shows that all transistors in the differential section are determined symmetric. All the unassigned transistors have their subgraph and the graph analysis aids in determining their symmetries.



**Figure 5.11 – Fully Differential two-stage Folded Cascode Symmetric edges (a) and devices (b)**

Transistor M24 should be matched with M6, but according to Figure 5.12 (b) this is not the case. This happens because M24 does not fulfill any pattern configuration. Matching comes from constraints generated by building blocks, and as M24 is not assigned to any building block, the transistor is not matched to any other transistor.



**Figure 5.12 – Fully Differential two-stage Folded Cascode Proximity groups (a) and matched devices (b)**

All constraints have been determined, and are sent to AIDA in the form of text. A thorough list of constraints can be found in Appendix A.2

### 5.3 Fully Differential OTA

Figure 5.13 shows a fully differential circuit, found in [74]. Due to the circuit configuration, the standard Differential Pair cannot be found, and an initial node symmetry must be determined in an alternative way. In this case, inputs  $V_{inP}$  and  $V_{inN}$  were considered as being symmetric nodes and the symmetry search started there. This circuit is used as a running example to show symmetry detection results, as shown in Figure 5.13, Figure 5.14, Figure 5.15, and Figure 5.16 and most of the pattern detection process is not shown, as the circuit also has only two first level building blocks detected.

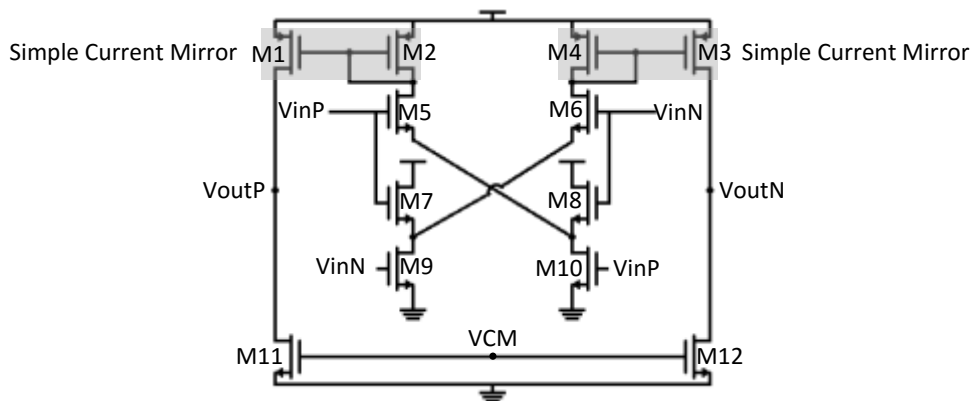
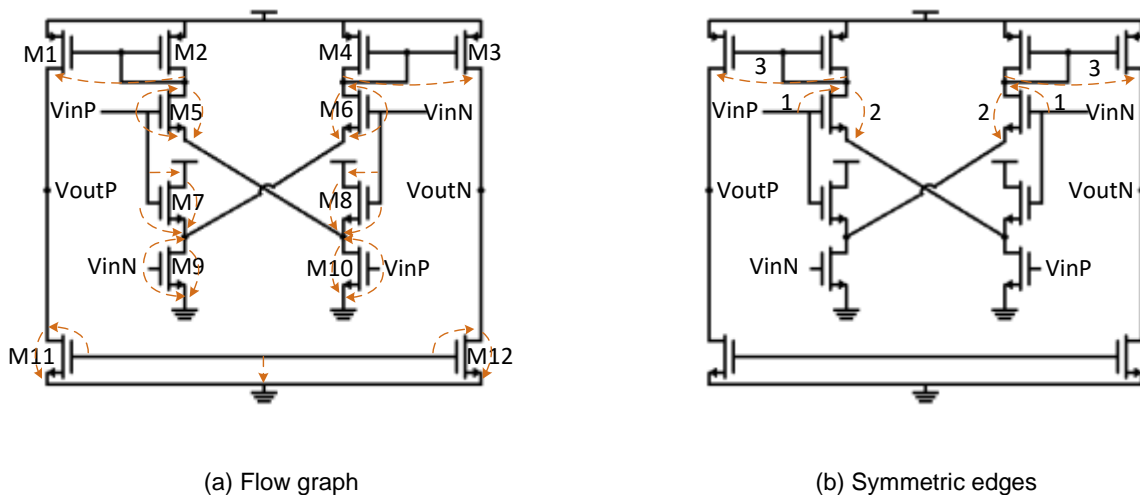


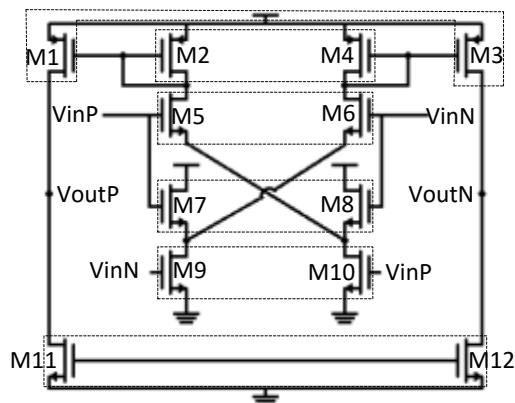
Figure 5.13 – Fully Differential OTA level 1 building blocks

Because most of the circuit's transistors are not assigned a pattern, most of the graph is generated from a transistor subgraph configuration. Figure 5.14 (a) shows the circuit graph. All the edges are generated from a transistor's subgraph with the exception of the edges going from M2 to M1 and M4 to M3. From the initial symmetric nodes  $V_{inP}$  and  $V_{inN}$ , the symmetric edges determined are shown in Figure 5.14 (b). Although not much of the circuit seems to be determined symmetric, most nodes are inter-connected, and are considered symmetric.



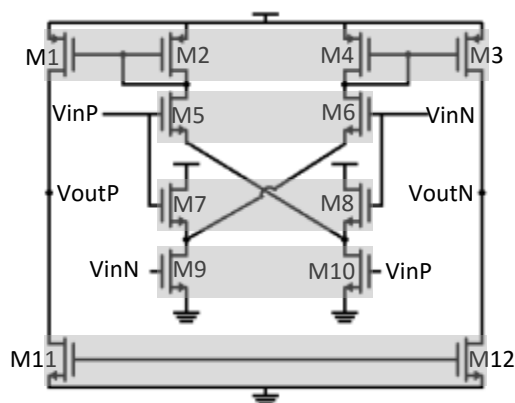
(a) Flow graph (b) Symmetric edges  
Figure 5.14 – Fully Differential OTA Signal and Current Flow Graph (a) and symmetric edges (b)

With a list of symmetric nodes and edges, the module can now find symmetry pairs in the circuit, shown in Figure 5.15.



**Figure 5.15 – Fully Differential OTA symmetric devices**

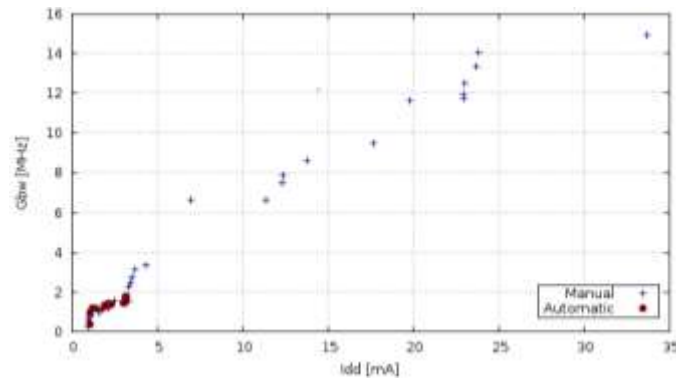
With the symmetries determined, the matching of devices can be determined, either from the symmetry constraint, or from the SCM constraint. Figure 5.16 shows the matched devices in the Fully Differential OTA.



**Figure 5.16 – Fully Differential OTA matched devices**

## 5.4 Folded Cascode Optimization Project

An optimization project was performed for the Folded Cascode shown in Figure 5.1, with manually created constraints used in a previous project, and with the automated constraints generated by the module. Results will be shown as a POF where the current ( $I_{dd}$ ) and gain-bandwidth ( $G_{bw}$ ) are the optimized variables, and then a Monte Carlo analysis is run with some selected solutions. Figure 5.17 shows the POF resulting from an AIDA-C optimization process, with both the manual and automatic constraints. The manually written constraints yield a wider set of solutions, while the automatically generated constraints result in a reduced set of solutions that are very close to the solutions found with manual constraints.



**Figure 5.17 – POF achieved with manual and automatic constraints in the Folded Cascode**

Two sizing solutions, one from each POF, were selected and a Monte Carlo analysis was applied with a sample size of 450. The sizing solutions were determined by selecting the one with the highest DC gain in the respective POF. Then, a second solution was chosen from the manual POF that was close to the automated sizing solution in terms of optimized variables to also be analyzed in Monte Carlo. The analysis results can be seen in Table 5-5 in regards to the optimized variables and the offset voltage, which is the most problematic variable in this project.

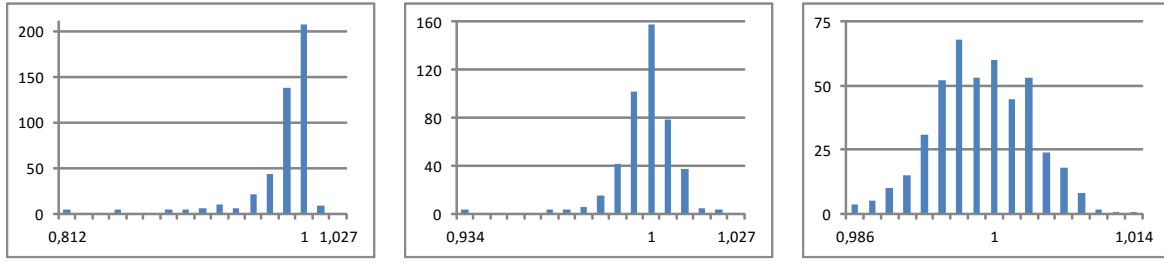
**Table 5-5 – Monte Carlo Analysis for the Folded Cascode**

Simulation	Current Consumption		Gain-Bandwidth		Offset Voltage	
	Nominal [mA]	Std. Deviation	Nominal [MHz]	Std. Deviation	Average [mV]	Std. Deviation
<b>Manual 1</b>	33,71	0.219	14,9	0.330	21,79	16.97
<b>Manual 2</b>	2,486	0.014	1,55	0.013	9,757	7.56
<b>Automated</b>	3,175	0.016	1,77	0.008	3,032	2.25

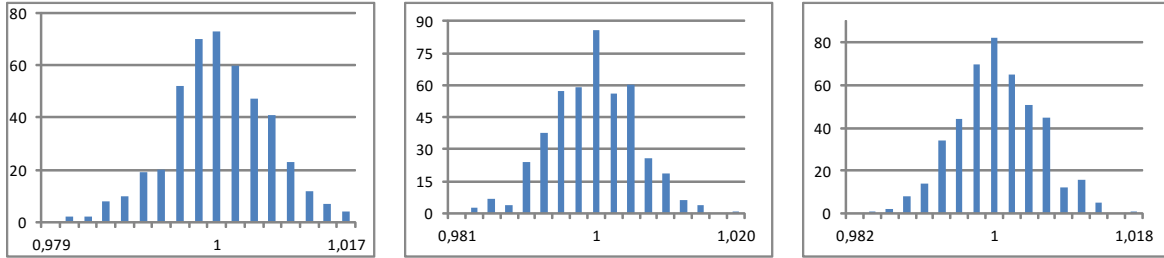
The simulation labeled Manual 1 is the sizing solution with the highest variances. This translates to the solutions towards that area being more sensitive and less robust, and that the constraints generated automatically will filter out the sizing solutions that are highly sensitive.

On the other hand, the Manual 2 simulation has similar nominal values, and is in the same region of the POF as the automated simulation, meaning that this solution is in a region that is “safer” and less sensitive. Although the current variance is similar in both, the gain-bandwidth in the manual 2 has a higher variance than the automated solution. The offset voltage not only has a higher variance, but also has a higher average than the automated solution.

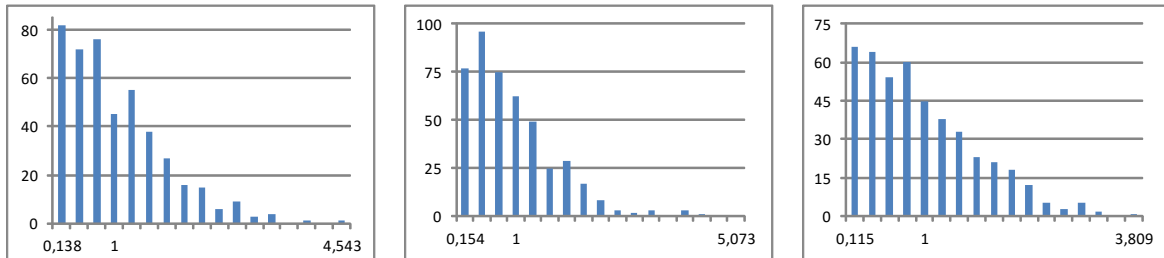
Figure 5.18, Figure 5.19 and Figure 5.20 show histograms resulting from a total of 450 samples each, taking in consideration nominal values for the optimized variables and the average for the offset voltage. Although current consumptions in the automatic solution and the manual 2 show similar levels of dispersion, the gain-bandwidth and offset voltages are less dispersed in the automatic solution. This means that the sizing solution from the automatic POF is less sensitive to the random changes applied by the Monte Carlo analysis, and that the circuit is more robust.



**Figure 5.18 – Gain-Bandwidth histograms for Monte Carlo (a) manual 1 (b) manual 2 (c) automatic**



**Figure 5.19 – Current histograms for Monte Carlo (a) manual 1 (b) manual 2 (c) automatic**



**Figure 5.20 – Offset Voltage histograms for Monte Carlo (a) manual 1 (b) manual 2 (c) automatic**

## 5.5 Conclusions

All the correct building blocks were detected in all three circuits, and the false detections were also correctly detected. The only matching that was not determined is in OTA Edinei's transistor M24, as mentioned previously. This is indicative that the field of automated constraint generation for analog circuits still has room for progress of further determining constraints.

Both the Folded Cascode and OTA Edinei have differential sections but are not fully differential circuits (i.e. have non-differential sections). All symmetries and self-symmetries were detected, and no inadequate symmetry is being detected. One circuit had very few transistors with assigned patterns, but the symmetries were still correctly determined considering the transistors without assigned patterns provided their own subgraphs.

One of the circuits was run for a full optimization project to compare previously written constraints with the automatically generated constraints, and although the automated constraints yield a reduced POF, the found solutions have lower sensitivities to random alterations that happen in circuit manufacturing. This allows the circuit designer to trade between maximizing the nominal solutions or circuit robustness.



## 6 Conclusions and future work

---

### 6.1 Conclusion

Constraint generation is approached in the automation of the field of IC design process. The current state of the art in this field was studied and shown that automated constraint generation has not seen much progress. Considering that constraints are currently being manually determined by the designer for each different circuit, automating the process would reduce setup time and accelerate the optimization process, as well as allow additional constraints that were not being included to be used.

Most of the research found was around finding predetermined patterns in the circuits, assigning constraints according to the patterns found, and find symmetries based on graphs also determined by the patterns found.

A new module based on this approach was implemented in AIDA, an analog IC design automation tool that optimizes circuit sizing. A non-sized netlist is used as input, and the module sends a sized netlist, along with the electrical constraints to the optimizer. This aids the optimizer in finding a set of robust solutions that have lower sensitivity to process variations.

### 6.2 Future work

Generating a more thorough set of constraints will yield more robust circuits, but will also result in a reduced set of solutions. This means that there's a trade-off between nominal performance and robustness. A possible way of dealing with this is if the designer can choose to start with a good set of nominal sizing solutions, or start with solutions that are already robust. Categorizing constraints with different priorities would give the designer flexibility to choose robust solutions, good nominal solutions, or a set of solutions somewhere in the middle. This could be accomplished by defining constraints to be e.g. essential, recommended, or optional, and in a given project the designer would choose to include constraints up to a certain priority.

Another suggestion for future work would be to research more possible patterns and associated constraints, as the current library does not cover every possible configuration, as shown in the fully differential folded cascade.



## 7 References

---

- [1] R. Martins, N. Lourenço and N. Horta, "LAYGEN II - Automatic Layout Generation of Analog Integrated Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 11, pp. 1641-1654, November 2013.
- [2] F. Rocha, R. Martins, N. Lourenço and N. Horta, "Enhancing a Layout-aware Synthesis Methodology for Analog ICs by Embedding Statistical Knowledge into the Evolutionary Optimization Kernel," in *Doctoral Conference on Computing, Electrical and Industrial Systems*, Lisbon, 2013.
- [3] G. Gielen and R. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proceedings of the IEEE*, vol. 88, no. 12, pp. 1825-1854, December 2000.
- [4] G. Gielen, "CAD tools for embedded analogue circuits in mixed-signal integrated systems on chip," *IEE Proceedings on Computers and Digital Techniques*, vol. 152, no. 3, pp. 317-332, May 2005.
- [5] N. e. a. Lourenço, "Automatic Analog IC Sizing and Optimization Constrained with PVT Corners and Layout Effects," 2014.
- [6] E. Roca, R. Castro-López and F. Fernández, "Hierarchical synthesis based on pareto-optimal-fronts," *European Conference on Circuit Theory and Design*, pp. 755-758, August 2009.
- [7] "Mentor Graphics," [Online]. Available: <http://www.mentor.com>.
- [8] Synopsys, "HSPICE® User Guide: Basic Simulation and Analysis, Version G-2012.06-SP2," December 2012.
- [9] "Synopsys," [Online]. Available: <http://www.synopsys.com>.
- [10] "Cadence Design System Inc.," [Online]. Available: <http://www.cadence.com>.
- [11] "gEDA Project," [Online]. Available: <http://www.gpleda.org>.
- [12] "Dolphin Integration," [Online]. Available: <http://www.dolphin.fr>.
- [13] "MunEDA," [Online]. Available: <http://www.muneda.com>.

- [14] N. Lourenço, A. Canelas, R. Póvoa, R. Martins and N. Horta, "Floorplan-aware analog IC sizing and optimization based on topological constraints," *Integration, the VLSI Journal*, vol. 48, no. 1, pp. 183-197, January 2015.
- [15] G. Jerke and J. Liening, "Constraint-driven Design - The Next Step Towards Analog Design Automation," in *Proceedings of the 2009 international symposium on Physical design*, 2009.
- [16] T. Massier, H. Graeb and U. Schlichtmann, "The Sizing Rules Method for CMOS and Bipolar Analog Integrated Circuit Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 12, pp. 2209-2222, December 2008.
- [17] T. Mukherjee, L. R. Carley and R. Rutenbar, "Efficient handling of operating range and manufacturing line variations in analog cell synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 8, pp. 825-839, August 2000.
- [18] M. F. M. Barros, J. M. C. Guilherme and N. C. G. Horta, *Analog circuits and systems optimization based on evolutionary computation techniques*, Berlin: Springer, 2010.
- [19] C. A. Makris and a. C. Toumazou, "Analog IC design automation. II. Automated circuit correction by qualitative reasoning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 2, pp. 239-254, 1995.
- [20] C. Toumazou and C. A. Makris, "Analog IC design automation. I. Automated circuit generation: new concepts and methods," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 2, pp. 218-238, 1995.
- [21] M. G. R. Degrauwe, O. Nys, E. Dijkstra, J. Rijmenants, S. Bitz, B. L. A. G. Goffart, E. A. Vittoz, S. Cserveny, C. Meixenberger, G. V. D. Stappen and H. J. Oguey, "IDAC: an interactive design tool for analog CMOS circuits," *IEEE Journal of Solid-State Circuits*, vol. 22, no. 6, pp. 1106-1116, 1987.
- [22] N. Horta, "Analogue and Mixed-Signal Systems Topologies Exploration Using Symbolic Methods," *Analog Integrated Circuits and Signal Processing*, vol. 31, no. 2, pp. 161-176, 2002.
- [23] H. Y. Koh, C. H. Sequin and P. R. Gray, "OPASYN: a compiler for CMOS operational amplifiers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 2, pp. 113-125, 1990.
- [24] J. P. Harvey, M. I. Elmasry and B. Leung, "STAIC: an interactive framework for synthesizing CMOS and BiCMOS analog circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 11, pp. 1402-1417, 1992.

- [25] P. C. Maulik, L. R. Carley and D. J. Allstot, "Sizing of cell-level analog circuits using constrained optimization techniques," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 3, pp. 223-241, 1993.
- [26] P. C. Maulik, L. R. Carley and R. A. Rutenbar, "Integer programming based topology selection of cell-level analog circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 4, pp. 401-412, 1995.
- [27] K. Matsukawa, T. Morie, Y. Tokunaga, S. Sakiyama, Y. Mitani, M. Takayama, T. Miki, A. Matsumoto, K. Obata and S. Dosho, "Design methods for pipeline & delta-sigma A-to-D converters with convex optimization," in *Asia and South Pacific Design Automation Conference*, Yokohama, 2009.
- [28] M. d. M. Hershenson, S. P. Boyd and T. H. Lee, "GPCAD: a tool for CMOS op-amp synthesis," in *IEEE/ACM International Conference on Computer-Aided Design*, San Jose, 1998.
- [29] M. Kuo-Hsuan, P. Po-Cheng and C. Hung-Ming, "Integrated hierarchical synthesis of analog/RF circuits with accurate performance mapping," in *International Symposium on Quality Electronic Design*, Santa Clara, 2011.
- [30] A. J. Torralba, J. Chavez and L. G. Franquelo, "Fuzzy-logic-based analog design tools," *Micro, IEEE*, vol. 16, no. 4, pp. 60-68, 1996.
- [31] A. Torralba, J. Chavez and L. G. Franquelo, "FASY: a fuzzy-logic based tool for analog synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 7, pp. 705-715, 1996.
- [32] G. G. E. Gielen, H. C. C. Walscharts and W. M. C. Sansen, "Analog circuit design optimization based on symbolic simulation and simulated annealing," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 3, pp. 707-713, 1990.
- [33] E. S. Ochotta, R. A. Rutenbar and L. R. Carley, "Synthesis of high-performance analog circuits in ASTRX/OBLX," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 3, pp. 273-294, 1996.
- [34] W. Kruiskamp and D. Leenaerts, "DARWIN: CMOS opamp synthesis by means of a genetic algorithm," in *Design Automation Conference*, San Francisco, 1995.
- [35] A. Doboli, N. Dhanwada, A. Nunez-Aldana and R. Vemuri, "A two-layer library-based approach to synthesis of analog systems from VHDL-AMS specifications," *ACM Transactions on Design Automation of Electronic Systems*, vol. 9, no. 2, pp. 238-271, 2004.

- [36] L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," EECS Department, University of California, Berkeley, 1975.
- [37] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671-680, 1983.
- [38] W. Nye, D. Riley, A. Sangiovanni-Vincentelli and A. Tits, "DELIGHT.SPICE: an optimization-based system for the design of integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 4, pp. 501-519, 1988.
- [39] L. Cheng-Wu, S. Pin-Dai, S. Ya-Ting and C. Soon-Jyh, "A bias-driven approach for automated design of operational amplifiers," in *International Symposium on VLSI Design, Automation and Test*, Hsinchu, 2009.
- [40] F. Medeiro, F. Fernandez, R. Dominguez-Castro and A. Rodriguez-Vazquez, "A Statistical Optimization-based Approach For Automated Sizing Of Analog Cells," in *International Conference Computer-Aided Design*, 1994.
- [41] R. Castro-Lopez, O. Guerra, E. Roca and F. Fernandez, "An Integrated Layout-Synthesis Approach for Analog ICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1179-1189, 2008.
- [42] S.-C. Chu, H.-C. Huang, J. F. Roddick and J.-S. Pan, "Overview of Algorithms for Swarm Intelligence," in *Computational Collective Intelligence. Technologies and Applications*, Berlin Heidelberg, Springer , 2011, pp. 28-41.
- [43] H. Gupta and B. Ghosh, "Analog Circuits Design Using Ant Colony Optimization," *International Journal of Electronics, Computer & Communications Technologies*, vol. 2, no. 3, pp. 9-21, 2012.
- [44] B. Benhala, A. Ahaitouf, M. Fakhfakh and A. Mechaqrane, "New Adaptation of the ACO Algorithm for the Analog Circuits Design Optimization," *International Journal of Computer Science Issues*, vol. 9, no. 3, pp. 360-367, 2012.
- [45] S. Kamisetty, J. Garg, J. Tripathi and J. Mukherjee, "Optimization of Analog RF Circuit parameters using randomness in particle swarm optimization," in *World Congress on Information and Communication Technologies*, 2011.
- [46] P. P. Kumar and K. Duraiswamy, "An Optimized Device Sizing of Analog Circuits using Particle Swarm Optimization," *Journal of Computer Science*, vol. 8, no. 6, pp. 930-935, 2012.

- [47] M. Fakhfakh, Y. Cooren, A. Sallem, M. Loulou and P. Siarry, "Analog circuit design optimization through the particle swarm optimization technique," *Analog Integrated Circuits and Signal Processing*, vol. 63, no. 1, pp. 71-82, 2010.
- [48] N. Lourenço and N. Horta, "GENOM-POF: Multi-Objective Evolutionary Synthesis of Analog ICs with Corners Validation," in *Genetic and Evolutionary Computation Conference*, Philadelphia, 2012.
- [49] N. Lourenço, R. Martins, M. Barros and N. Horta, "Analog Circuit Design based on Robust POFs using an Enhanced MOEA with SVM Models," in *Analog/RF and Mixed-Signal Circuit Systematic Design*, Berlin, Springer, 2013, pp. 149-167.
- [50] T. McConaghy, P. Palmers, M. Steyaert and G. Gielen, "Trustworthy Genetic Programming-Based Synthesis of Analog Circuit Topologies Using Hierarchical Domain-Specific Building Blocks," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 4, pp. 557-570, 2011.
- [51] A. Pradhan and R. Vemuri, "Efficient Synthesis of a Uniformly Spread Layout Aware Pareto Surface for Analog Circuits," in *International Conference on VLSI Design*, New Delhi, 2009.
- [52] M. Eick, M. Strasser, K. Lu, U. Schlichtmann and H. Graeb, "Comprehensive Generation of Hierarchical Placement Rules for Analog Integrated Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 2, pp. 180-193, February 2011.
- [53] J. R. Koza, F. I. Bennett, D. Andre, M. A. Keane and F. Dunlap, "Automated synthesis of analog electrical circuits by means of genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 2, pp. 109-128, 1997.
- [54] M. Krasnicki, R. Phelps, R. Rutenbar and L. Carley, "MAELSTROM: efficient simulation-based synthesis for custom analog cells," in *Design Automation Conference*, New Orleans, 1999.
- [55] R. Phelps, M. Krasnicki, R. Rutenbar, L. Carley and J. Hellums, "Anaconda: simulation-based synthesis of analog circuits via stochastic pattern search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 6, pp. 703-717, 2000.
- [56] T. Sripramong and C. Toumazou, "The invention of CMOS amplifiers using genetic programming and current-flow analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, pp. 1237-1252, 2002.
- [57] G. Alpaydin, S. Balkir and G. Dunder, "An evolutionary approach to automatic synthesis of high-performance analog integrated circuits," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 240-252, 2003.

- [58] C. Shoou-Jinn, H. Hao-Sheng and S. Yan-Kuin, "Automated passive filter synthesis using a novel tree representation and genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 1, pp. 93-100, 2006.
- [59] M. Barros, J. Guilherme and N. Horta, "Analog circuits optimization based on evolutionary computation techniques," *Integration, the VLSI Journal*, vol. 43, no. 1, pp. 136-155, 2010.
- [60] M. Barros, J. Guilherme and N. Horta, "GA-SVM feasibility model and optimization kernel applied to analog IC design automation," in *ACM Great Lakes symposium on VLSI*, Stresa-Lago Maggiore, 2007.
- [61] R. Santos-Tavares, N. Paulino, J. Higino, J. Goes and J. P. Oliveira, "Optimization of Multi-Stage Amplifiers in Deep-Submicron CMOS Using a Distributed/Parallel Genetic Algorithm," in *International Symposium on Circuits and Systems*, Seattle, 2008.
- [62] Y. Hongying and H. Jingsong, "Evolutionary design of operational amplifier using variable-length differential evolution algorithm," in *International Conference on Computer Application and System Modeling*, Taiyuan Shanxi, 2010.
- [63] M. Pehl, M. Zwerger and H. Graeb, "Sizing analog circuits using an SQP and Branch and Bound based approach," in *17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, Athens, 2010.
- [64] H. Habal and H. Graeb, "Constraint-Based Layout-Driven Sizing of Analog Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 8, pp. 1089-1102, 2011.
- [65] E. Roca, M. Velasco-Jiménez, R. Castro-López and F. V. Fernández, "Context-dependent transformation of Pareto-optimal performance fronts of operational amplifiers," *Analog Integrated Circuits and Signal Processing*, vol. 73, no. 1, pp. 65-76, 2012.
- [66] F. Rocha, N. Lourenço, R. Póvoa, R. Martins and N. Horta, "A New Metaheuristic Combining Gradient Models with NSGA-II to Enhance Analog IC Synthesis," in *EEE Congress on Evolutionary Computation*, 2013.
- [67] F. Rocha, R. Martins, N. Lourenço and N. Horta, *Electronic Design Automation of Analog ICs combining Gradient Models with Multi-Objective Evolutionary Algorithms*, Springer, 2014.
- [68] Y.-C. Liao, Y.-L. Chen, X.-T. Cai, C.-N. Liu and T.-C. Chen, "LASER: layout-aware analog synthesis environment on laker," in *Proceedings of the 23rd ACM international conference on Great lakes symposium on VLSI*, 2013.



- [69] E. Afacan, S. Ay, F. Fernandez, G. Dunder and F. Basckaya, "Model based hierarchical optimization strategies for analog design automation," in *Design, Automation and Test in Europe Conference and Exhibition*, Dresden, 2014.
- [70] R. Póvoa, R. Lourenço, N. Lourenço, A. Canelas, R. Martins and N. Horta, "Synthesis of LC-Oscillators using Rival Multi-Objective/Multi-Constraint Optimization Kernels," in *Performance Optimization Techniques in Analog, Mixed-Signal, and Radio-Frequency Circuit Design*, IGI Global, 2014, p. (in press).
- [71] R. Póvoa, R. Lourenço, N. Lourenço, A. Canelas, R. Martins and N. Horta, "LC-VCO Automatic Synthesis Using Multi-Objective Evolutionary Techniques," in *Proceedings of the International Symposium on Circuits and Systems*, Melbourne, 2014.
- [72] M. Eick and H. Graeb, "MARS: Matching-Driven Analog Sizing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 8, pp. 1145-1158, August 2013.
- [73] E. Santin, L. B. Oliveira, B. Nowacki and J. Goes, "A Fully Integrated and Reconfigurable Architecture for Coherent Self-Testing of High Speed Analog-to-Digital Converters," *Transactions on Circuits and Systems – I*, vol. 58, no. 7, pp. 1531-1541, Jul 2011.
- [74] R. Póvoa, N. Lourenço, N. Horta, R. Santos Tavares and J. Goes, "Single-Stage Amplifiers with Gain Enhancement and Improved Energy-Efficiency employing Voltage-Combiners," in *Very Large Scale Integration (VLSI-SoC), 2013 IFIP/IEEE 21st International Conference*, 2013.
- [75] G. Gielen, T. McConaghy and T. Eeckelaert, "Performance space modeling for hierarchical synthesis of analog integrated circuits," in *Design Automation Conference*, 2005.
- [76] Y.-L. Chen, W.-R. Wu, C.-N. Liu and J.-M. Li, "Simultaneous Optimization of Analog Circuits With Reliability and Variability for Applications on Flexible Electronics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 1, pp. 24-35, 2014.
- [77] Mentor Graphics Corporation, "Eldo user's manual, Release 13.1," 2013.



# Appendix A

## A.1 Folded Cascode

```
***Netlist testing for circuit***
.SUBCKT OPAMP VDD VSS IREF VIN VIP VOUT
***Device list***
MPB2    VPOLAP  VPOLAP  VDD    VDD    P_12_HSL130E  L=L_MPB2 W=W_MPB2 M=M_MPB2
MPB1    IREF    IREF    VPOLAP VDD    P_12_HSL130E  L=L_MPB1 W=W_MPB1 M=M_MPB1
MPB3    21      VPOLAP  VDD    VDD    P_12_HSL130E  L=L_MPB2 W=W_MPB2 M=M_MPB2*R_MPB2_MPB3
MPB4    VPOLAN1 IREF    21      VDD    P_12_HSL130E  L=L_MPB1 W=W_MPB1 M=M_MPB1*R_MPB1_MPB4
MPB7    31      VPOLAP  VDD    VDD    P_12_HSL130E  L=L_MPB2 W=W_MPB2 M=M_MPB2*R_MPB2_MPB7
MP18    43      VIN      31      VDD    P_12_HSL130E  L=L_MP18 W=W_MP18 M=M_MP18
MP19    53      VIP      31      VDD    P_12_HSL130E  L=L_MP18 W=W_MP18 M=M_MP18
MPB8    41      42      VDD    VDD    P_12_HSL130E  L=L_MPB8 W=W_MPB8 M=M_MPB8*R_MPB8_MPB9
MPB10   42      IREF    41      VDD    P_12_HSL130E  L=L_MPB1 W=W_MPB1
M=M_MPB1*R_MPB1_MPB10*R_MPB10_MPB11
MPB9    51      42      VDD    VDD    P_12_HSL130E  L=L_MPB8 W=W_MPB8 M=M_MPB8*R_MPB8_MPB9
MPB11   52      IREF    51      VDD    P_12_HSL130E  L=L_MPB1 W=W_MPB1
M=M_MPB1*R_MPB1_MPB10*R_MPB10_MPB11
MP16    VOUT    52      VDD    VDD    P_12_HSL130E  L=L_MP16 W=W_MP16 M=M_MP16
MNB5    VPOLAN1 VPOLAN1 VPOLAN  VSS    N_12_HSL130E  L=L_MNB5 W=W_MNB5 M=M_MNB5
MNB6    VPOLAN  VPOLAN  VSS    VSS    N_12_HSL130E  L=L_MNB6 W=W_MNB6 M=M_MNB6
MN12    42      VPOLAN1 43      VSS    N_12_HSL130E  L=L_MNB5 W=W_MNB5 M=M_MNB5*R_MNB5_MN13
MN14    43      VPOLAN  VSS    VSS    N_12_HSL130E  L=L_MNB6 W=W_MNB6 M=M_MNB6*R_MNB6_MN15
MN13    52      VPOLAN1 53      VSS    N_12_HSL130E  L=L_MNB5 W=W_MNB5 M=M_MNB5*R_MNB5_MN13
MN15    53      VPOLAN  VSS    VSS    N_12_HSL130E  L=L_MNB6 W=W_MNB6 M=M_MNB6*R_MNB6_MN15
MN17    VOUT    VPOLAN  VSS    VSS    N_12_HSL130E  L=L_MNB6 W=W_MNB6 M=M_MNB6*R_MNB6_MN17
```

```
<Variable name="W_MPB2" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MPB2" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MPB2" min="1" step="1" max="8"/>
<Variable name="W_MPB1" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MPB1" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MPB1" min="1" step="1" max="8"/>
<Variable name="R_MPB2_MPB3" min="0.5" step="0.1" max="10"/>
<Variable name="R_MPB1_MPB4" min="0.5" step="0.1" max="10"/>
<Variable name="R_MPB2_MPB7" min="0.5" step="0.1" max="10"/>
<Variable name="W_MP18" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MP18" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MP18" min="1" step="1" max="8"/>
<Variable name="W_MPB8" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MPB8" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MPB8" min="1" step="1" max="8"/>
<Variable name="R_MPB8_MPB9" min="0.5" step="0.1" max="10"/>
<Variable name="R_MPB1_MPB10" min="0.5" step="0.1" max="10"/>
<Variable name="W_MP16" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MP16" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MP16" min="1" step="1" max="8"/>
<Variable name="W_MNB5" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MNB5" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MNB5" min="1" step="1" max="8"/>
<Variable name="W_MNB6" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MNB6" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MNB6" min="1" step="1" max="8"/>
<Variable name="R_MNB5_MN13" min="0.5" step="0.1" max="10"/>
<Variable name="R_MNB6_MN15" min="0.5" step="0.1" max="10"/>
<Variable name="R_MNB6_MN17" min="0.5" step="0.1" max="10"/>
```

```
<MeasureDescription name="vov_MPB3" description="Overdrive" units="[V]" />
<MeasureDescription name="vov_MPB7" description="Overdrive" units="[V]" />
<MeasureDescription name="vov_MPB1" description="Overdrive" units="[V]" />
```



```

<MeasureDescription name="VDS_MPB9" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="rev_delta_MPB11" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MPB11" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MPB11" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="rev_delta_MPB16" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MPB16" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MPB16" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="vov_MPB16" description="Overdrive" units="[V]" />
<MeasureDescription name="psiDS_MPB18_MPB19" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MPB18_MPB19" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_MPB8_MPB9" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MPB8_MPB9" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_MPB10_MPB11" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MPB10_MPB11" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_MN12_MN13" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MN12_MN13" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_MN14_MN15" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MN14_MN15" description="VGS_X1 - VGS_X2" units="[V]" />

```

```

.MEAS DC vov_MPB3 PARAM = 'LV9(XAMP.MPB3)-VGS(XAMP.MPB3)'
.MEAS DC vov_MPB7 PARAM = 'LV9(XAMP.MPB7)-VGS(XAMP.MPB7)'
.MEAS DC vov_MPB1 PARAM = 'LV9(XAMP.MPB1)-VGS(XAMP.MPB1)'
.MEAS DC vov_MPB4 PARAM = 'LV9(XAMP.MPB4)-VGS(XAMP.MPB4)'
.MEAS DC vov_MPB10 PARAM = 'LV9(XAMP.MPB10)-VGS(XAMP.MPB10)'
.MEAS DC vov_MPB11 PARAM = 'LV9(XAMP.MPB11)-VGS(XAMP.MPB11)'
.MEAS DC psiDS_MPB18_MPB19 = PARAM('ABS(VDS(XAMP.MPB18)-VDS(XAMP.MPB19))')
.MEAS DC psigs_MPB18_MPB19 = PARAM('ABS(VGS(XAMP.MPB18)-VGS(XAMP.MPB19))')
.MEAS DC vov_MNB5 PARAM = 'LV9(XAMP.MNB5)-VGS(XAMP.MNB5)'
.MEAS DC vov_MN12 PARAM = 'LV9(XAMP.MN12)-VGS(XAMP.MN12)'
.MEAS DC vov_MN13 PARAM = 'LV9(XAMP.MN13)-VGS(XAMP.MN13)'
.MEAS DC vov_MN14 PARAM = 'LV9(XAMP.MN14)-VGS(XAMP.MN14)'
.MEAS DC vov_MN15 PARAM = 'LV9(XAMP.MN15)-VGS(XAMP.MN15)'
.MEAS DC vov_MN17 PARAM = 'LV9(XAMP.MN17)-VGS(XAMP.MN17)'
.MEAS DC vov_MPB9 PARAM = 'LV9(XAMP.MPB9)-VGS(XAMP.MPB9)'
.MEAS DC delta_MNB5 PARAM = 'VDS(XAMP.MNB5) - VDSAT(XAMP.MNB5)'
.MEAS DC A_MNB5 = PARAM('L_MNB5*W_MNB5*M_MNB5')
.MEAS DC VDS_MNB5 PARAM = 'VDS(XAMP.MNB5)'
.MEAS DC delta_MNB6 PARAM = 'VDS(XAMP.MNB6) - VDSAT(XAMP.MNB6)'
.MEAS DC A_MNB6 = PARAM('L_MNB6*W_MNB6*M_MNB6')
.MEAS DC VDS_MNB6 PARAM = 'VDS(XAMP.MNB6)'
.MEAS DC vov_MNB6 PARAM = 'LV9(XAMP.MNB6)-VGS(XAMP.MNB6)'
.MEAS DC delta_MN12 PARAM = 'VDS(XAMP.MN12) - VDSAT(XAMP.MN12)'
.MEAS DC A_MN12 = PARAM('L_MNB5*W_MNB5*M_MNB5*R_MNB5_MN13')
.MEAS DC VDS_MN12 PARAM = 'VDS(XAMP.MN12)'
.MEAS DC delta_MN14 PARAM = 'VDS(XAMP.MN14) - VDSAT(XAMP.MN14)'
.MEAS DC A_MN14 = PARAM('L_MNB6*W_MNB6*M_MNB6*R_MNB6_MN15')
.MEAS DC VDS_MN14 PARAM = 'VDS(XAMP.MN14)'
.MEAS DC delta_MN13 PARAM = 'VDS(XAMP.MN13) - VDSAT(XAMP.MN13)'
.MEAS DC A_MN13 = PARAM('L_MNB5*W_MNB5*M_MNB5*R_MNB5_MN13')
.MEAS DC VDS_MN13 PARAM = 'VDS(XAMP.MN13)'
.MEAS DC delta_MN15 PARAM = 'VDS(XAMP.MN15) - VDSAT(XAMP.MN15)'
.MEAS DC A_MN15 = PARAM('L_MNB6*W_MNB6*M_MNB6*R_MNB6_MN15')
.MEAS DC VDS_MN15 PARAM = 'VDS(XAMP.MN15)'
.MEAS DC delta_MN17 PARAM = 'VDS(XAMP.MN17) - VDSAT(XAMP.MN17)'
.MEAS DC A_MN17 = PARAM('L_MNB6*W_MNB6*M_MNB6*R_MNB6_MN17')
.MEAS DC VDS_MN17 PARAM = 'VDS(XAMP.MN17)'
.MEAS DC rev_delta_MPB2 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPB2 = PARAM('L_MPB2*W_MPB2*M_MPB2')
.MEAS DC VDS_MPB2 PARAM = 'VDS(XAMP.MPB2)'
.MEAS DC vov_MPB2 PARAM = 'LV9(XAMP.MPB2)-VGS(XAMP.MPB2)'
.MEAS DC rev_delta_MPB1 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPB1 = PARAM('L_MPB1*W_MPB1*M_MPB1')
.MEAS DC VDS_MPB1 PARAM = 'VDS(XAMP.MPB1)'
.MEAS DC rev_delta_MPB3 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPB3 = PARAM('L_MPB2*W_MPB2*M_MPB2*R_MPB2_MPB3')
.MEAS DC VDS_MPB3 PARAM = 'VDS(XAMP.MPB3)'

```

```

.MEAS DC rev_delta_MPB4 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPB4 = PARAM('L_MPB1*W_MPB1*M_MPB1*R_MPB1_MPB4')
.MEAS DC VDS_MPB4 PARAM = 'VDS(XAMP.MPB4)'
.MEAS DC rev_delta_MPB7 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPB7 = PARAM('L_MPB2*W_MPB2*M_MPB2*R_MPB2_MPB7')
.MEAS DC VDS_MPB7 PARAM = 'VDS(XAMP.MPB7)'
.MEAS DC rev_delta_MP18 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MP18 = PARAM('L_MP18*W_MP18*M_MP18')
.MEAS DC VDS_MP18 PARAM = 'VDS(XAMP.MP18)'
.MEAS DC vov_MP18 PARAM = 'LV9(XAMP.MP18)-VGS(XAMP.MP18)'
.MEAS DC rev_delta_MP19 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MP19 = PARAM('L_MP18*W_MP18*M_MP18')
.MEAS DC VDS_MP19 PARAM = 'VDS(XAMP.MP19)'
.MEAS DC vov_MP19 PARAM = 'LV9(XAMP.MP19)-VGS(XAMP.MP19)'
.MEAS DC rev_delta_MPB8 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPB8 = PARAM('L_MPB8*W_MPB8*M_MPB8*R_MPB8_MPB9')
.MEAS DC VDS_MPB8 PARAM = 'VDS(XAMP.MPB8)'
.MEAS DC vov_MPB8 PARAM = 'LV9(XAMP.MPB8)-VGS(XAMP.MPB8)'
.MEAS DC rev_delta_MPB10 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPB10 = PARAM('L_MPB1*W_MPB1*M_MPB1*R_MPB1_MPB10*R_MPB10_MPB11')
.MEAS DC VDS_MPB10 PARAM = 'VDS(XAMP.MPB10)'
.MEAS DC rev_delta_MPB9 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPB9 = PARAM('L_MPB8*W_MPB8*M_MPB8*R_MPB8_MPB9')
.MEAS DC VDS_MPB9 PARAM = 'VDS(XAMP.MPB9)'
.MEAS DC rev_delta_MPB11 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPB11 = PARAM('L_MPB1*W_MPB1*M_MPB1*R_MPB1_MPB10*R_MPB10_MPB11')
.MEAS DC VDS_MPB11 PARAM = 'VDS(XAMP.MPB11)'
.MEAS DC rev_delta_MP16 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MP16 = PARAM('L_MP16*W_MP16*M_MP16')
.MEAS DC VDS_MP16 PARAM = 'VDS(XAMP.MP16)'
.MEAS DC vov_MP16 PARAM = 'LV9(XAMP.MP16)-VGS(XAMP.MP16)'
.MEAS DC psiDS_MP18_MP19 = PARAM('ABS(VDS(XAMP.MP18)-VDS(XAMP.MP19))')
.MEAS DC psigs_MP18_MP19 = PARAM('ABS(VGS(XAMP.MP18)-VGS(XAMP.MP19))')
.MEAS DC psiDS_MPB8_MPB9 = PARAM('ABS(VDS(XAMP.MPB8)-VDS(XAMP.MPB9))')
.MEAS DC psigs_MPB8_MPB9 = PARAM('ABS(VGS(XAMP.MPB8)-VGS(XAMP.MPB9))')
.MEAS DC psiDS_MPB10_MPB11 = PARAM('ABS(VDS(XAMP.MPB10)-VDS(XAMP.MPB11))')
.MEAS DC psigs_MPB10_MPB11 = PARAM('ABS(VGS(XAMP.MPB10)-VGS(XAMP.MPB11))')
.MEAS DC psiDS_MN12_MN13 = PARAM('ABS(VDS(XAMP.MN12)-VDS(XAMP.MN13))')
.MEAS DC psigs_MN12_MN13 = PARAM('ABS(VGS(XAMP.MN12)-VGS(XAMP.MN13))')
.MEAS DC psiDS_MN14_MN15 = PARAM('ABS(VDS(XAMP.MN14)-VDS(XAMP.MN15))')
.MEAS DC psigs_MN14_MN15 = PARAM('ABS(VGS(XAMP.MN14)-VGS(XAMP.MN15))')

```

```

<Constraint op="GE" value="0.1" meas="vov_MPB3" />
<Constraint op="GE" value="0.1" meas="vov_MPB7" />
<Constraint op="GE" value="0.1" meas="vov_MPB1" />
<Constraint op="GE" value="0.1" meas="vov_MPB4" />
<Constraint op="GE" value="0.1" meas="vov_MPB10" />
<Constraint op="GE" value="0.1" meas="vov_MPB11" />
<Constraint op="LE" value="0.1" meas="psiDS_MP18_MP19" />
<Constraint op="LE" value="0.05" meas="psigs_MP18_MP19" />
<Constraint op="GE" value="0.1" meas="vov_MNB5" />
<Constraint op="GE" value="0.1" meas="vov_MN12" />
<Constraint op="GE" value="0.1" meas="vov_MN13" />
<Constraint op="GE" value="0.1" meas="vov_MN14" />
<Constraint op="GE" value="0.1" meas="vov_MN15" />
<Constraint op="GE" value="0.1" meas="vov_MN17" />
<Constraint op="GE" value="0.1" meas="vov_MPB9" />
<Constraint op="GE" value="0.1" meas="delta_MNB5" />
<Constraint op="GE" value="6.0E-14" meas="A_MNB5" />
<Constraint op="GE" value="0.00" meas="VDS_MNB5" />
<Constraint op="GE" value="0.1" meas="delta_MNB6" />
<Constraint op="GE" value="6.0E-14" meas="A_MNB6" />
<Constraint op="GE" value="0.00" meas="VDS_MNB6" />
<Constraint op="GE" value="0.00" meas="vov_MNB6" />
<Constraint op="GE" value="0.1" meas="delta_MN12" />
<Constraint op="GE" value="6.0E-14" meas="A_MN12" />

```

```
<Constraint op="GE" value="0.00" meas="VDS_MN12" />
<Constraint op="GE" value="0.1" meas="delta_MN14" />
<Constraint op="GE" value="6.0E-14" meas="A_MN14" />
<Constraint op="GE" value="0.00" meas="VDS_MN14" />
<Constraint op="GE" value="0.1" meas="delta_MN13" />
<Constraint op="GE" value="6.0E-14" meas="A_MN13" />
<Constraint op="GE" value="0.00" meas="VDS_MN13" />
<Constraint op="GE" value="0.1" meas="delta_MN15" />
<Constraint op="GE" value="6.0E-14" meas="A_MN15" />
<Constraint op="GE" value="0.00" meas="VDS_MN15" />
<Constraint op="GE" value="0.1" meas="delta_MN17" />
<Constraint op="GE" value="6.0E-14" meas="A_MN17" />
<Constraint op="GE" value="0.00" meas="VDS_MN17" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPB2" />
<Constraint op="GE" value="6.0E-14" meas="A_MPB2" />
<Constraint op="GE" value="0.00" meas="VDS_MPB2" />
<Constraint op="GE" value="0.00" meas="vov_MPB2" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPB1" />
<Constraint op="GE" value="6.0E-14" meas="A_MPB1" />
<Constraint op="GE" value="0.00" meas="VDS_MPB1" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPB3" />
<Constraint op="GE" value="6.0E-14" meas="A_MPB3" />
<Constraint op="GE" value="0.00" meas="VDS_MPB3" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPB4" />
<Constraint op="GE" value="6.0E-14" meas="A_MPB4" />
<Constraint op="GE" value="0.00" meas="VDS_MPB4" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPB7" />
<Constraint op="GE" value="6.0E-14" meas="A_MPB7" />
<Constraint op="GE" value="0.00" meas="VDS_MPB7" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPB8" />
<Constraint op="GE" value="6.0E-14" meas="A_MPB8" />
<Constraint op="GE" value="0.00" meas="VDS_MPB8" />
<Constraint op="GE" value="0.00" meas="vov_MPB8" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPB9" />
<Constraint op="GE" value="6.0E-14" meas="A_MPB9" />
<Constraint op="GE" value="0.00" meas="VDS_MPB9" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPB10" />
<Constraint op="GE" value="6.0E-14" meas="A_MPB10" />
<Constraint op="GE" value="0.00" meas="VDS_MPB10" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPB11" />
<Constraint op="GE" value="6.0E-14" meas="A_MPB11" />
<Constraint op="GE" value="0.00" meas="VDS_MPB11" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPB12" />
<Constraint op="GE" value="6.0E-14" meas="A_MPB12" />
<Constraint op="GE" value="0.00" meas="VDS_MPB12" />
<Constraint op="GE" value="0.00" meas="vov_MPB12" />
<Constraint op="LE" value="0.1" meas="psids_MP18_MP19" />
<Constraint op="LE" value="0.05" meas="psigs_MP18_MP19" />
<Constraint op="LE" value="0.1" meas="psids_MPB8_MPB9" />
<Constraint op="LE" value="0.05" meas="psigs_MPB8_MPB9" />
<Constraint op="LE" value="0.1" meas="psids_MPB10_MPB11" />
<Constraint op="LE" value="0.05" meas="psigs_MPB10_MPB11" />
<Constraint op="LE" value="0.1" meas="psids_MN12_MN13" />
<Constraint op="LE" value="0.05" meas="psigs_MN12_MN13" />
<Constraint op="LE" value="0.1" meas="psids_MN14_MN15" />
<Constraint op="LE" value="0.05" meas="psigs_MN14_MN15" />
```

## A.2 Fully Differential two-stage Folded Cascode

```
***Netlist testing for circuit***
.subckt nova_diff cmfb gnd vdd vin vip von vop
***Device list***
mpm1    von    net15    vdd      vdd      P_HG_33_L130E    L=L_mpm4 W=W_mpm4 M=M_mpm4*R_mpm4_mpm3
mpm2    net15  net15    vdd      vdd      P_HG_33_L130E    L=L_mpm4 W=W_mpm4 M=M_mpm4
mpm3    vop    net024   vdd      vdd      P_HG_33_L130E    L=L_mpm4 W=W_mpm4 M=M_mpm4*R_mpm4_mpm3
mpm4    net024 net024   vdd      vdd      P_HG_33_L130E    L=L_mpm4 W=W_mpm4 M=M_mpm4
mnm5    net15  vip      crossa   crossa   N_HG_33_L130E    L=L_mnm6 W=W_mnm6 M=M_mnm6
mnm7    vdd    vip      crossb   crossb   N_HG_33_L130E    L=L_mnm8 W=W_mnm8 M=M_mnm8
mnm9    crossb vin      gnd      gnd      N_HG_33_L130E    L=L_mnm10 W=W_mnm10 M=M_mnm10
mnm11   von    cmfb    gnd      gnd      N_HG_33_L130E    L=L_mnm12 W=W_mnm12 M=M_mnm12
mnm12   vop    cmfb    gnd      gnd      N_HG_33_L130E    L=L_mnm12 W=W_mnm12 M=M_mnm12
mnm6    net024 vin      crossb   crossb   N_HG_33_L130E    L=L_mnm6 W=W_mnm6 M=M_mnm6
mnm8    vdd    vin      crossa   crossa   N_HG_33_L130E    L=L_mnm8 W=W_mnm8 M=M_mnm8
mnm10   crossa vip      gnd      gnd      N_HG_33_L130E    L=L_mnm10 W=W_mnm10 M=M_mnm10
```

```
<Variable name="W_MP0" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MP0" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MP0" min="1" step="1" max="8"/>
<Variable name="W_MPM17" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MPM17" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MPM17" min="1" step="1" max="8"/>
<Variable name="W_MPM10" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MPM10" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MPM10" min="1" step="1" max="8"/>
<Variable name="R_MPM10_MPM13" min="0.5" step="0.1" max="10"/>
<Variable name="W_MPM15" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MPM15" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MPM15" min="1" step="1" max="8"/>
<Variable name="W_MPM1" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MPM1" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MPM1" min="1" step="1" max="8"/>
<Variable name="R_MPM1_MPM3" min="0.5" step="0.1" max="10"/>
<Variable name="R_MPM1_MPM2" min="0.5" step="0.1" max="10"/>
<Variable name="W_MPM7" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MPM7" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MPM7" min="1" step="1" max="8"/>
<Variable name="W_MN0" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MN0" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MN0" min="1" step="1" max="8"/>
<Variable name="W_MNM4" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MNM4" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MNM4" min="1" step="1" max="8"/>
<Variable name="R_MNM4_MNM19" min="0.5" step="0.1" max="10"/>
<Variable name="R_MNM4_MNM5" min="0.5" step="0.1" max="10"/>
<Variable name="W_MNM20" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MNM20" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MNM20" min="1" step="1" max="8"/>
<Variable name="W_MNM24" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MNM24" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MNM24" min="1" step="1" max="8"/>
<Variable name="W_MNM6" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MNM6" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MNM6" min="1" step="1" max="8"/>
<Variable name="R_MNM6_MNM9" min="0.5" step="0.1" max="10"/>
<Variable name="R_MNM6_MNM12" min="0.5" step="0.1" max="10"/>
<Variable name="W_MNM23" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_MNM23" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_MNM23" min="1" step="1" max="8"/>
```

```
<MeasureDescription name="psiDS_MPM17_MPM16" description="VDS_X1 - VDS_X2" units="[V]" />
```



[illegible]

```

<MeasureDescription name="vov_MNM23" description="Overdrive" units="[V]" />
<MeasureDescription name="rev_delta_MP0" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MP0" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MP0" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="vov_MP0" description="Overdrive" units="[V]" />
<MeasureDescription name="rev_delta_MPM17" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MPM17" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MPM17" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="vov_MPM17" description="Overdrive" units="[V]" />
<MeasureDescription name="rev_delta_MPM16" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MPM16" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MPM16" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="vov_MPM16" description="Overdrive" units="[V]" />
<MeasureDescription name="rev_delta_MPM13" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MPM13" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MPM13" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="rev_delta_MPM10" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MPM10" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MPM10" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="vov_MPM10" description="Overdrive" units="[V]" />
<MeasureDescription name="rev_delta_MPM14" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="W_MPM14" description="Width" units="[m]" />
<MeasureDescription name="L_MPM14" description="Length" units="[m]" />
<MeasureDescription name="A_MPM14" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MPM14" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="vov_MPM14" description="Overdrive" units="[V]" />
<MeasureDescription name="rev_delta_MPM15" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MPM15" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MPM15" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="vov_MPM15" description="Overdrive" units="[V]" />
<MeasureDescription name="rev_delta_MPM3" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MPM3" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MPM3" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="rev_delta_MPM1" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MPM1" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MPM1" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="vov_MPM1" description="Overdrive" units="[V]" />
<MeasureDescription name="rev_delta_MPM2" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MPM2" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MPM2" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="rev_delta_MPM7" description="VDSat - VDS" units="[V]" />
<MeasureDescription name="A_MPM7" description="Area" units="[m²]" />
<MeasureDescription name="VDS_MPM7" description="Drain-Source Voltage" units="[V]" />
<MeasureDescription name="vov_MPM7" description="Overdrive" units="[V]" />
<MeasureDescription name="psiDS_MPM17_MPM16" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MPM17_MPM16" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_MPM14_MPM15" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MPM14_MPM15" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_MNM18_MNM19" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MNM18_MNM19" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_MNM21_MNM20" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MNM21_MNM20" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_MNM22_MNM23" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_MNM22_MNM23" description="VGS_X1 - VGS_X2" units="[V]" />

```

```

.MEAS DC psiDS_MPM17_MPM16 = PARAM('ABS(VDS(XAMP.MPM17)-VDS(XAMP.MPM16))')
.MEAS DC psigs_MPM17_MPM16 = PARAM('ABS(VGS(XAMP.MPM17)-VGS(XAMP.MPM16))')
.MEAS DC vov_MPM13 PARAM = 'LV9(XAMP.MPM13)-VGS(XAMP.MPM13)'
.MEAS DC vov_MPM3 PARAM = 'LV9(XAMP.MPM3)-VGS(XAMP.MPM3)'
.MEAS DC vov_MPM2 PARAM = 'LV9(XAMP.MPM2)-VGS(XAMP.MPM2)'
.MEAS DC vov_MNM4 PARAM = 'LV9(XAMP.MNM4)-VGS(XAMP.MNM4)'
.MEAS DC vov_MNM18 PARAM = 'LV9(XAMP.MNM18)-VGS(XAMP.MNM18)'
.MEAS DC vov_MNM19 PARAM = 'LV9(XAMP.MNM19)-VGS(XAMP.MNM19)'
.MEAS DC vov_MNM5 PARAM = 'LV9(XAMP.MNM5)-VGS(XAMP.MNM5)'
.MEAS DC vov_MNM11 PARAM = 'LV9(XAMP.MNM11)-VGS(XAMP.MNM11)'
.MEAS DC vov_MNM8 PARAM = 'LV9(XAMP.MNM8)-VGS(XAMP.MNM8)'

```

```

.MEAS DC vov_MNM12 PARAM = 'LV9(XAMP.MNM12)-VGS(XAMP.MNM12)'
.MEAS DC vov_MNM9 PARAM = 'LV9(XAMP.MNM9)-VGS(XAMP.MNM9)'
.MEAS DC delta_MN0 PARAM = 'VDS(XAMP.MN0) - VDSAT(XAMP.MN0)'
.MEAS DC A_MN0 = PARAM('L_MN0*W_MN0*M_MN0')
.MEAS DC VDS_MN0 PARAM = 'VDS(XAMP.MN0)'
.MEAS DC vov_MN0 PARAM = 'LV9(XAMP.MN0)-VGS(XAMP.MN0)'
.MEAS DC delta_MNM18 PARAM = 'VDS(XAMP.MNM18) - VDSAT(XAMP.MNM18)'
.MEAS DC A_MNM18 = PARAM('L_MNM4*W_MNM4*M_MNM4*R_MNM4_MNM19')
.MEAS DC VDS_MNM18 PARAM = 'VDS(XAMP.MNM18)'
.MEAS DC delta_MNM19 PARAM = 'VDS(XAMP.MNM19) - VDSAT(XAMP.MNM19)'
.MEAS DC A_MNM19 = PARAM('L_MNM4*W_MNM4*M_MNM4*R_MNM4_MNM19')
.MEAS DC VDS_MNM19 PARAM = 'VDS(XAMP.MNM19)'
.MEAS DC delta_MNM5 PARAM = 'VDS(XAMP.MNM5) - VDSAT(XAMP.MNM5)'
.MEAS DC A_MNM5 = PARAM('L_MNM4*W_MNM4*M_MNM4*R_MNM4_MNM5')
.MEAS DC VDS_MNM5 PARAM = 'VDS(XAMP.MNM5)'
.MEAS DC delta_MNM11 PARAM = 'VDS(XAMP.MNM11) - VDSAT(XAMP.MNM11)'
.MEAS DC A_MNM11 = PARAM('L_MNM4*W_MNM4*M_MNM4*R_MNM4_MNM5*R_MNM5_MNM11')
.MEAS DC VDS_MNM11 PARAM = 'VDS(XAMP.MNM11)'
.MEAS DC delta_MNM8 PARAM = 'VDS(XAMP.MNM8) - VDSAT(XAMP.MNM8)'
.MEAS DC A_MNM8 = PARAM('L_MNM4*W_MNM4*M_MNM4*R_MNM4_MNM5*R_MNM5_MNM8')
.MEAS DC VDS_MNM8 PARAM = 'VDS(XAMP.MNM8)'
.MEAS DC delta_MNM4 PARAM = 'VDS(XAMP.MNM4) - VDSAT(XAMP.MNM4)'
.MEAS DC A_MNM4 = PARAM('L_MNM4*W_MNM4*M_MNM4')
.MEAS DC VDS_MNM4 PARAM = 'VDS(XAMP.MNM4)'
.MEAS DC delta_MNM21 PARAM = 'VDS(XAMP.MNM21) - VDSAT(XAMP.MNM21)'
.MEAS DC W_MNM21 = PARAM('W_MNM20*M_MNM20')
.MEAS DC L_MNM21 = PARAM('L_MNM20')
.MEAS DC A_MNM21 = PARAM('L_MNM20*W_MNM20*M_MNM20')
.MEAS DC VDS_MNM21 PARAM = 'VDS(XAMP.MNM21)'
.MEAS DC vov_MNM21 PARAM = 'LV9(XAMP.MNM21)-VGS(XAMP.MNM21)'
.MEAS DC delta_MNM20 PARAM = 'VDS(XAMP.MNM20) - VDSAT(XAMP.MNM20)'
.MEAS DC A_MNM20 = PARAM('L_MNM20*W_MNM20*M_MNM20')
.MEAS DC VDS_MNM20 PARAM = 'VDS(XAMP.MNM20)'
.MEAS DC vov_MNM20 PARAM = 'LV9(XAMP.MNM20)-VGS(XAMP.MNM20)'
.MEAS DC delta_MNM24 PARAM = 'VDS(XAMP.MNM24) - VDSAT(XAMP.MNM24)'
.MEAS DC A_MNM24 = PARAM('L_MNM24*W_MNM24*M_MNM24')
.MEAS DC VDS_MNM24 PARAM = 'VDS(XAMP.MNM24)'
.MEAS DC vov_MNM24 PARAM = 'LV9(XAMP.MNM24)-VGS(XAMP.MNM24)'
.MEAS DC delta_MNM9 PARAM = 'VDS(XAMP.MNM9) - VDSAT(XAMP.MNM9)'
.MEAS DC A_MNM9 = PARAM('L_MNM6*W_MNM6*M_MNM6*R_MNM6_MNM9')
.MEAS DC VDS_MNM9 PARAM = 'VDS(XAMP.MNM9)'
.MEAS DC delta_MNM12 PARAM = 'VDS(XAMP.MNM12) - VDSAT(XAMP.MNM12)'
.MEAS DC A_MNM12 = PARAM('L_MNM6*W_MNM6*M_MNM6*R_MNM6_MNM12')
.MEAS DC VDS_MNM12 PARAM = 'VDS(XAMP.MNM12)'
.MEAS DC delta_MNM6 PARAM = 'VDS(XAMP.MNM6) - VDSAT(XAMP.MNM6)'
.MEAS DC A_MNM6 = PARAM('L_MNM6*W_MNM6*M_MNM6')
.MEAS DC VDS_MNM6 PARAM = 'VDS(XAMP.MNM6)'
.MEAS DC vov_MNM6 PARAM = 'LV9(XAMP.MNM6)-VGS(XAMP.MNM6)'
.MEAS DC delta_MNM22 PARAM = 'VDS(XAMP.MNM22) - VDSAT(XAMP.MNM22)'
.MEAS DC W_MNM22 = PARAM('W_MNM23*M_MNM23')
.MEAS DC L_MNM22 = PARAM('L_MNM23')
.MEAS DC A_MNM22 = PARAM('L_MNM23*W_MNM23*M_MNM23')
.MEAS DC VDS_MNM22 PARAM = 'VDS(XAMP.MNM22)'
.MEAS DC vov_MNM22 PARAM = 'LV9(XAMP.MNM22)-VGS(XAMP.MNM22)'
.MEAS DC delta_MNM23 PARAM = 'VDS(XAMP.MNM23) - VDSAT(XAMP.MNM23)'
.MEAS DC A_MNM23 = PARAM('L_MNM23*W_MNM23*M_MNM23')
.MEAS DC VDS_MNM23 PARAM = 'VDS(XAMP.MNM23)'
.MEAS DC vov_MNM23 PARAM = 'LV9(XAMP.MNM23)-VGS(XAMP.MNM23)'
.MEAS DC rev_delta_MP0 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MP0 = PARAM('L_MP0*W_MP0*M_MP0')
.MEAS DC VDS_MP0 PARAM = 'VDS(XAMP.MP0)'
.MEAS DC vov_MP0 PARAM = 'LV9(XAMP.MP0)-VGS(XAMP.MP0)'
.MEAS DC rev_delta_MPM17 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPM17 = PARAM('L_MPM17*W_MPM17*M_MPM17')
.MEAS DC VDS_MPM17 PARAM = 'VDS(XAMP.MPM17)'
.MEAS DC vov_MPM17 PARAM = 'LV9(XAMP.MPM17)-VGS(XAMP.MPM17)'
.MEAS DC rev_delta_MPM16 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'

```

```
.MEAS DC A_MPM16 = PARAM('L_MPM17*W_MPM17*M_MPM17')
.MEAS DC VDS_MPM16 PARAM = 'VDS(XAMP.MPM16)'
.MEAS DC vov_MPM16 PARAM = 'LV9(XAMP.MPM16)-VGS(XAMP.MPM16)'
.MEAS DC rev_delta_MPM13 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPM13 = PARAM('L_MPM10*W_MPM10*M_MPM10*R_MPM10_MPM13')
.MEAS DC VDS_MPM13 PARAM = 'VDS(XAMP.MPM13)'
.MEAS DC rev_delta_MPM10 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPM10 = PARAM('L_MPM10*W_MPM10*M_MPM10')
.MEAS DC VDS_MPM10 PARAM = 'VDS(XAMP.MPM10)'
.MEAS DC vov_MPM10 PARAM = 'LV9(XAMP.MPM10)-VGS(XAMP.MPM10)'
.MEAS DC rev_delta_MPM14 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC W_MPM14 = PARAM('W_MPM15*M_MPM15')
.MEAS DC L_MPM14 = PARAM('L_MPM15')
.MEAS DC A_MPM14 = PARAM('L_MPM15*W_MPM15*M_MPM15')
.MEAS DC VDS_MPM14 PARAM = 'VDS(XAMP.MPM14)'
.MEAS DC vov_MPM14 PARAM = 'LV9(XAMP.MPM14)-VGS(XAMP.MPM14)'
.MEAS DC rev_delta_MPM15 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPM15 = PARAM('L_MPM15*W_MPM15*M_MPM15')
.MEAS DC VDS_MPM15 PARAM = 'VDS(XAMP.MPM15)'
.MEAS DC vov_MPM15 PARAM = 'LV9(XAMP.MPM15)-VGS(XAMP.MPM15)'
.MEAS DC rev_delta_MPM3 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPM3 = PARAM('L_MPM1*W_MPM1*M_MPM1*R_MPM1_MPM3')
.MEAS DC VDS_MPM3 PARAM = 'VDS(XAMP.MPM3)'
.MEAS DC rev_delta_MPM1 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPM1 = PARAM('L_MPM1*W_MPM1*M_MPM1')
.MEAS DC VDS_MPM1 PARAM = 'VDS(XAMP.MPM1)'
.MEAS DC vov_MPM1 PARAM = 'LV9(XAMP.MPM1)-VGS(XAMP.MPM1)'
.MEAS DC rev_delta_MPM2 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPM2 = PARAM('L_MPM1*W_MPM1*M_MPM1*R_MPM1_MPM2')
.MEAS DC VDS_MPM2 PARAM = 'VDS(XAMP.MPM2)'
.MEAS DC rev_delta_MPM7 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_MPM7 = PARAM('L_MPM7*W_MPM7*M_MPM7')
.MEAS DC VDS_MPM7 PARAM = 'VDS(XAMP.MPM7)'
.MEAS DC vov_MPM7 PARAM = 'LV9(XAMP.MPM7)-VGS(XAMP.MPM7)'
.MEAS DC psiDS_MPM17_MPM16 = PARAM('ABS(VDS(XAMP.MPM17)-VDS(XAMP.MPM16))')
.MEAS DC psigs_MPM17_MPM16 = PARAM('ABS(VGS(XAMP.MPM17)-VGS(XAMP.MPM16))')
.MEAS DC psiDS_MPM14_MPM15 = PARAM('ABS(VDS(XAMP.MPM14)-VDS(XAMP.MPM15))')
.MEAS DC psigs_MPM14_MPM15 = PARAM('ABS(VGS(XAMP.MPM14)-VGS(XAMP.MPM15))')
.MEAS DC psiDS_MNM18_MNM19 = PARAM('ABS(VDS(XAMP.MNM18)-VDS(XAMP.MNM19))')
.MEAS DC psigs_MNM18_MNM19 = PARAM('ABS(VGS(XAMP.MNM18)-VGS(XAMP.MNM19))')
.MEAS DC psiDS_MNM21_MNM20 = PARAM('ABS(VDS(XAMP.MNM21)-VDS(XAMP.MNM20))')
.MEAS DC psigs_MNM21_MNM20 = PARAM('ABS(VGS(XAMP.MNM21)-VGS(XAMP.MNM20))')
.MEAS DC psiDS_MNM22_MNM23 = PARAM('ABS(VDS(XAMP.MNM22)-VDS(XAMP.MNM23))')
.MEAS DC psigs_MNM22_MNM23 = PARAM('ABS(VGS(XAMP.MNM22)-VGS(XAMP.MNM23))')
```

```
<Constraint op="LE" value="0.1" meas="psiDS_MPM17_MPM16" />
<Constraint op="LE" value="0.05" meas="psigs_MPM17_MPM16" />
<Constraint op="GE" value="0.1" meas="vov_MPM13" />
<Constraint op="GE" value="0.1" meas="vov_MPM3" />
<Constraint op="GE" value="0.1" meas="vov_MPM2" />
<Constraint op="GE" value="0.1" meas="vov_MNM4" />
<Constraint op="GE" value="0.1" meas="vov_MNM18" />
<Constraint op="GE" value="0.1" meas="vov_MNM19" />
<Constraint op="GE" value="0.1" meas="vov_MNM5" />
<Constraint op="GE" value="0.1" meas="vov_MNM11" />
<Constraint op="GE" value="0.1" meas="vov_MNM8" />
<Constraint op="GE" value="0.1" meas="vov_MNM12" />
<Constraint op="GE" value="0.1" meas="vov_MNM9" />
<Constraint op="GE" value="0.1" meas="delta_MN0" />
<Constraint op="GE" value="6.0E-14" meas="A_MN0" />
<Constraint op="GE" value="0.00" meas="VDS_MN0" />
<Constraint op="GE" value="0.00" meas="vov_MN0" />
<Constraint op="GE" value="0.1" meas="delta_MNM18" />
<Constraint op="GE" value="6.0E-14" meas="A_MNM18" />
<Constraint op="GE" value="0.00" meas="VDS_MNM18" />
<Constraint op="GE" value="0.1" meas="delta_MNM19" />
```

[illegible]

```

<Constraint op="GE" value="0.1" meas="rev_delta_MPM14" />
<Constraint op="GE" value="3.0E-7" meas="W_MPM14" />
<Constraint op="GE" value="1.5E-7" meas="L_MPM14" />
<Constraint op="GE" value="6.0E-14" meas="A_MPM14" />
<Constraint op="GE" value="0.00" meas="VDS_MPM14" />
<Constraint op="GE" value="0.00" meas="vov_MPM14" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPM15" />
<Constraint op="GE" value="6.0E-14" meas="A_MPM15" />
<Constraint op="GE" value="0.00" meas="VDS_MPM15" />
<Constraint op="GE" value="0.00" meas="vov_MPM15" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPM3" />
<Constraint op="GE" value="6.0E-14" meas="A_MPM3" />
<Constraint op="GE" value="0.00" meas="VDS_MPM3" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPM1" />
<Constraint op="GE" value="6.0E-14" meas="A_MPM1" />
<Constraint op="GE" value="0.00" meas="VDS_MPM1" />
<Constraint op="GE" value="0.00" meas="vov_MPM1" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPM2" />
<Constraint op="GE" value="6.0E-14" meas="A_MPM2" />
<Constraint op="GE" value="0.00" meas="VDS_MPM2" />
<Constraint op="GE" value="0.1" meas="rev_delta_MPM7" />
<Constraint op="GE" value="6.0E-14" meas="A_MPM7" />
<Constraint op="GE" value="0.00" meas="VDS_MPM7" />
<Constraint op="GE" value="0.00" meas="vov_MPM7" />
<Constraint op="LE" value="0.1" meas="psiDS_MPM17_MPM16" />
<Constraint op="LE" value="0.05" meas="psigs_MPM17_MPM16" />
<Constraint op="LE" value="0.1" meas="psiDS_MPM14_MPM15" />
<Constraint op="LE" value="0.05" meas="psigs_MPM14_MPM15" />
<Constraint op="LE" value="0.1" meas="psiDS_MNM18_MNM19" />
<Constraint op="LE" value="0.05" meas="psigs_MNM18_MNM19" />
<Constraint op="LE" value="0.1" meas="psiDS_MNM21_MNM20" />
<Constraint op="LE" value="0.05" meas="psigs_MNM21_MNM20" />
<Constraint op="LE" value="0.1" meas="psiDS_MNM22_MNM23" />
<Constraint op="LE" value="0.05" meas="psigs_MNM22_MNM23" />

```

### A.3 Fully Differential OTA

```

***Netlist testing for circuit***
.subckt nova_diff cmfb gnd vdd vin vip von vop
***Device list***
mpm1    von    net15    vdd    vdd    P_HG_33_L130E    L=L_mpm4 W=W_mpm4 M=M_mpm4*R_mpm4_mpm3
mpm2    net15  net15    vdd    vdd    P_HG_33_L130E    L=L_mpm4 W=W_mpm4 M=M_mpm4
mpm3    vop    net024   vdd    vdd    P_HG_33_L130E    L=L_mpm4 W=W_mpm4 M=M_mpm4*R_mpm4_mpm3
mpm4    net024 net024   vdd    vdd    P_HG_33_L130E    L=L_mpm4 W=W_mpm4 M=M_mpm4
mnm5    net15  vip    crossa crossa N_HG_33_L130E    L=L_mnm6 W=W_mnm6 M=M_mnm6
mnm7    vdd    vip    crosssb crosssb N_HG_33_L130E    L=L_mnm8 W=W_mnm8 M=M_mnm8
mnm9    crosssb vin    gnd    gnd    N_HG_33_L130E    L=L_mnm10 W=W_mnm10 M=M_mnm10
mnm11   von    cmfb    gnd    gnd    N_HG_33_L130E    L=L_mnm12 W=W_mnm12 M=M_mnm12
mnm12   vop    cmfb    gnd    gnd    N_HG_33_L130E    L=L_mnm12 W=W_mnm12 M=M_mnm12
mnm6    net024 vin    crosssb crosssb N_HG_33_L130E    L=L_mnm6 W=W_mnm6 M=M_mnm6
mnm8    vdd    vin    crossa crossa N_HG_33_L130E    L=L_mnm8 W=W_mnm8 M=M_mnm8
mnm10   crossa vip    gnd    gnd    N_HG_33_L130E    L=L_mnm10 W=W_mnm10 M=M_mnm10

```

```

<Variable name="W_mpm4" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_mpm4" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_mpm4" min="1" step="1" max="8"/>
<Variable name="R_mpm4_mpm3" min="0.5" step="0.1" max="10"/>
<Variable name="W_mnm6" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_mnm6" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_mnm6" min="1" step="1" max="8"/>
<Variable name="W_mnm8" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_mnm8" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>

```

```
<Variable name="M_mnm8" min="1" step="1" max="8"/>
<Variable name="W_mnm10" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_mnm10" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_mnm10" min="1" step="1" max="8"/>
<Variable name="W_mnm12" min="3.0E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="L_mnm12" min="1.5E-7" step="5.0E-7" max="1.0E-4"/>
<Variable name="M_mnm12" min="1" step="1" max="8"/>
```

[illegible]

```

<MeasureDescription name="psiDS_mpm2_mpm4" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_mpm2_mpm4" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_mnm5_mnm6" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_mnm5_mnm6" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_mnm7_mnm8" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_mnm7_mnm8" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_mnm9_mnm10" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_mnm9_mnm10" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_mpm1_mpm3" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_mpm1_mpm3" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_mpm2_mpm4" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_mpm2_mpm4" description="VGS_X1 - VGS_X2" units="[V]" />
<MeasureDescription name="psiDS_mnm11_mnm12" description="VDS_X1 - VDS_X2" units="[V]" />
<MeasureDescription name="psigs_mnm11_mnm12" description="VGS_X1 - VGS_X2" units="[V]" />

```

```

.MEAS DC vov_mpm1 PARAM = 'LV9(XAMP.mpm1)-VGS(XAMP.mpm1)'
.MEAS DC vov_mpm3 PARAM = 'LV9(XAMP.mpm3)-VGS(XAMP.mpm3)'
.MEAS DC delta_mnm5 PARAM = 'VDS(XAMP.mnm5) - VDSAT(XAMP.mnm5)'
.MEAS DC W_mnm5 = PARAM('W_mnm6*M_mnm6')
.MEAS DC L_mnm5 = PARAM('L_mnm6')
.MEAS DC A_mnm5 = PARAM('L_mnm6*W_mnm6*M_mnm6')
.MEAS DC VDS_mnm5 PARAM = 'VDS(XAMP.mnm5)'
.MEAS DC vov_mnm5 PARAM = 'LV9(XAMP.mnm5)-VGS(XAMP.mnm5)'
.MEAS DC delta_mnm7 PARAM = 'VDS(XAMP.mnm7) - VDSAT(XAMP.mnm7)'
.MEAS DC W_mnm7 = PARAM('W_mnm8*M_mnm8')
.MEAS DC L_mnm7 = PARAM('L_mnm8')
.MEAS DC A_mnm7 = PARAM('L_mnm8*W_mnm8*M_mnm8')
.MEAS DC VDS_mnm7 PARAM = 'VDS(XAMP.mnm7)'
.MEAS DC vov_mnm7 PARAM = 'LV9(XAMP.mnm7)-VGS(XAMP.mnm7)'
.MEAS DC delta_mnm9 PARAM = 'VDS(XAMP.mnm9) - VDSAT(XAMP.mnm9)'
.MEAS DC W_mnm9 = PARAM('W_mnm10*M_mnm10')
.MEAS DC L_mnm9 = PARAM('L_mnm10')
.MEAS DC A_mnm9 = PARAM('L_mnm10*W_mnm10*M_mnm10')
.MEAS DC VDS_mnm9 PARAM = 'VDS(XAMP.mnm9)'
.MEAS DC vov_mnm9 PARAM = 'LV9(XAMP.mnm9)-VGS(XAMP.mnm9)'
.MEAS DC delta_mnm11 PARAM = 'VDS(XAMP.mnm11) - VDSAT(XAMP.mnm11)'
.MEAS DC W_mnm11 = PARAM('W_mnm12*M_mnm12')
.MEAS DC L_mnm11 = PARAM('L_mnm12')
.MEAS DC A_mnm11 = PARAM('L_mnm12*W_mnm12*M_mnm12')
.MEAS DC VDS_mnm11 PARAM = 'VDS(XAMP.mnm11)'
.MEAS DC vov_mnm11 PARAM = 'LV9(XAMP.mnm11)-VGS(XAMP.mnm11)'
.MEAS DC delta_mnm12 PARAM = 'VDS(XAMP.mnm12) - VDSAT(XAMP.mnm12)'
.MEAS DC A_mnm12 = PARAM('L_mnm12*W_mnm12*M_mnm12')
.MEAS DC VDS_mnm12 PARAM = 'VDS(XAMP.mnm12)'
.MEAS DC vov_mnm12 PARAM = 'LV9(XAMP.mnm12)-VGS(XAMP.mnm12)'
.MEAS DC delta_mnm6 PARAM = 'VDS(XAMP.mnm6) - VDSAT(XAMP.mnm6)'
.MEAS DC A_mnm6 = PARAM('L_mnm6*W_mnm6*M_mnm6')
.MEAS DC VDS_mnm6 PARAM = 'VDS(XAMP.mnm6)'
.MEAS DC vov_mnm6 PARAM = 'LV9(XAMP.mnm6)-VGS(XAMP.mnm6)'
.MEAS DC delta_mnm8 PARAM = 'VDS(XAMP.mnm8) - VDSAT(XAMP.mnm8)'
.MEAS DC A_mnm8 = PARAM('L_mnm8*W_mnm8*M_mnm8')
.MEAS DC VDS_mnm8 PARAM = 'VDS(XAMP.mnm8)'
.MEAS DC vov_mnm8 PARAM = 'LV9(XAMP.mnm8)-VGS(XAMP.mnm8)'
.MEAS DC delta_mnm10 PARAM = 'VDS(XAMP.mnm10) - VDSAT(XAMP.mnm10)'
.MEAS DC A_mnm10 = PARAM('L_mnm10*W_mnm10*M_mnm10')
.MEAS DC VDS_mnm10 PARAM = 'VDS(XAMP.mnm10)'
.MEAS DC vov_mnm10 PARAM = 'LV9(XAMP.mnm10)-VGS(XAMP.mnm10)'
.MEAS DC rev_delta_mpm1 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC W_mpm2 = PARAM('W_mpm4*M_mpm4')
.MEAS DC L_mpm2 = PARAM('L_mpm4')
.MEAS DC A_mpm1 = PARAM('L_mpm4*W_mpm4*M_mpm4*R_mpm4_mpm3')
.MEAS DC VDS_mpm1 PARAM = 'VDS(XAMP.mpm1)'
.MEAS DC rev_delta_mpm2 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_mpm2 = PARAM('L_mpm4*W_mpm4*M_mpm4')
.MEAS DC VDS_mpm2 PARAM = 'VDS(XAMP.mpm2)'
.MEAS DC vov_mpm2 PARAM = 'LV9(XAMP.mpm2)-VGS(XAMP.mpm2)'

```



```
.MEAS DC rev_delta_mpm3 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_mpm3 = PARAM('L_mpm4*W_mpm4*M_mpm4*R_mpm4_mpm3')
.MEAS DC VDS_mpm3 PARAM = 'VDS(XAMP.mpm3)'
.MEAS DC rev_delta_mpm4 PARAM = 'VDSAT(XAMP.delta) - VDS(XAMP.delta)'
.MEAS DC A_mpm4 = PARAM('L_mpm4*W_mpm4*M_mpm4')
.MEAS DC VDS_mpm4 PARAM = 'VDS(XAMP.mpm4)'
.MEAS DC vov_mpm4 PARAM = 'LV9(XAMP.mpm4)-VGS(XAMP.mpm4)'
.MEAS DC psiDS_mpm2_mpm4 = PARAM('ABS(VDS(XAMP.mpm2)-VDS(XAMP.mpm4))')
.MEAS DC psigs_mpm2_mpm4 = PARAM('ABS(VGS(XAMP.mpm2)-VGS(XAMP.mpm4))')
.MEAS DC psiDS_mnm5_mnm6 = PARAM('ABS(VDS(XAMP.mnm5)-VDS(XAMP.mnm6))')
.MEAS DC psigs_mnm5_mnm6 = PARAM('ABS(VGS(XAMP.mnm5)-VGS(XAMP.mnm6))')
.MEAS DC psiDS_mnm7_mnm8 = PARAM('ABS(VDS(XAMP.mnm7)-VDS(XAMP.mnm8))')
.MEAS DC psigs_mnm7_mnm8 = PARAM('ABS(VGS(XAMP.mnm7)-VGS(XAMP.mnm8))')
.MEAS DC psiDS_mnm9_mnm10 = PARAM('ABS(VDS(XAMP.mnm9)-VDS(XAMP.mnm10))')
.MEAS DC psigs_mnm9_mnm10 = PARAM('ABS(VGS(XAMP.mnm9)-VGS(XAMP.mnm10))')
.MEAS DC psiDS_mpm1_mpm3 = PARAM('ABS(VDS(XAMP.mpm1)-VDS(XAMP.mpm3))')
.MEAS DC psigs_mpm1_mpm3 = PARAM('ABS(VGS(XAMP.mpm1)-VGS(XAMP.mpm3))')
.MEAS DC psiDS_mpm2_mpm4 = PARAM('ABS(VDS(XAMP.mpm2)-VDS(XAMP.mpm4))')
.MEAS DC psigs_mpm2_mpm4 = PARAM('ABS(VGS(XAMP.mpm2)-VGS(XAMP.mpm4))')
.MEAS DC psiDS_mnm11_mnm12 = PARAM('ABS(VDS(XAMP.mnm11)-VDS(XAMP.mnm12))')
.MEAS DC psigs_mnm11_mnm12 = PARAM('ABS(VGS(XAMP.mnm11)-VGS(XAMP.mnm12))')
```

```
<Constraint op="GE" value="0.1" meas="vov_mpm1" />
<Constraint op="GE" value="0.1" meas="vov_mpm3" />
<Constraint op="GE" value="0.1" meas="delta_mnm5" />
<Constraint op="GE" value="3.0E-7" meas="W_mnm5" />
<Constraint op="GE" value="1.5E-7" meas="L_mnm5" />
<Constraint op="GE" value="6.0E-14" meas="A_mnm5" />
<Constraint op="GE" value="0.00" meas="VDS_mnm5" />
<Constraint op="GE" value="0.00" meas="vov_mnm5" />
<Constraint op="GE" value="0.1" meas="delta_mnm7" />
<Constraint op="GE" value="3.0E-7" meas="W_mnm7" />
<Constraint op="GE" value="1.5E-7" meas="L_mnm7" />
<Constraint op="GE" value="6.0E-14" meas="A_mnm7" />
<Constraint op="GE" value="0.00" meas="VDS_mnm7" />
<Constraint op="GE" value="0.00" meas="vov_mnm7" />
<Constraint op="GE" value="0.1" meas="delta_mnm9" />
<Constraint op="GE" value="3.0E-7" meas="W_mnm9" />
<Constraint op="GE" value="1.5E-7" meas="L_mnm9" />
<Constraint op="GE" value="6.0E-14" meas="A_mnm9" />
<Constraint op="GE" value="0.00" meas="VDS_mnm9" />
<Constraint op="GE" value="0.00" meas="vov_mnm9" />
<Constraint op="GE" value="0.1" meas="delta_mnm11" />
<Constraint op="GE" value="3.0E-7" meas="W_mnm11" />
<Constraint op="GE" value="1.5E-7" meas="L_mnm11" />
<Constraint op="GE" value="6.0E-14" meas="A_mnm11" />
<Constraint op="GE" value="0.00" meas="VDS_mnm11" />
<Constraint op="GE" value="0.00" meas="vov_mnm11" />
<Constraint op="GE" value="0.1" meas="delta_mnm12" />
<Constraint op="GE" value="6.0E-14" meas="A_mnm12" />
<Constraint op="GE" value="0.00" meas="VDS_mnm12" />
<Constraint op="GE" value="0.00" meas="vov_mnm12" />
<Constraint op="GE" value="0.1" meas="delta_mnm6" />
<Constraint op="GE" value="6.0E-14" meas="A_mnm6" />
<Constraint op="GE" value="0.00" meas="VDS_mnm6" />
<Constraint op="GE" value="0.00" meas="vov_mnm6" />
<Constraint op="GE" value="0.1" meas="delta_mnm8" />
<Constraint op="GE" value="6.0E-14" meas="A_mnm8" />
<Constraint op="GE" value="0.00" meas="VDS_mnm8" />
<Constraint op="GE" value="0.00" meas="vov_mnm8" />
<Constraint op="GE" value="0.1" meas="delta_mnm10" />
<Constraint op="GE" value="6.0E-14" meas="A_mnm10" />
<Constraint op="GE" value="0.00" meas="VDS_mnm10" />
<Constraint op="GE" value="0.00" meas="vov_mnm10" />
<Constraint op="GE" value="0.1" meas="rev_delta_mpm1" />
<Constraint op="GE" value="3.0E-7" meas="W_mpm2" />
```

```

<Constraint op="GE" value="1.5E-7" meas="L_mpm2" />
<Constraint op="GE" value="6.0E-14" meas="A_mpm1" />
<Constraint op="GE" value="0.00" meas="VDS_mpm1" />
<Constraint op="GE" value="0.1" meas="rev_delta_mpm2" />
<Constraint op="GE" value="6.0E-14" meas="A_mpm2" />
<Constraint op="GE" value="0.00" meas="VDS_mpm2" />
<Constraint op="GE" value="0.00" meas="vov_mpm2" />
<Constraint op="GE" value="0.1" meas="rev_delta_mpm3" />
<Constraint op="GE" value="6.0E-14" meas="A_mpm3" />
<Constraint op="GE" value="0.00" meas="VDS_mpm3" />
<Constraint op="GE" value="0.1" meas="rev_delta_mpm4" />
<Constraint op="GE" value="6.0E-14" meas="A_mpm4" />
<Constraint op="GE" value="0.00" meas="VDS_mpm4" />
<Constraint op="GE" value="0.00" meas="vov_mpm4" />
<Constraint op="LE" value="0.1" meas="psiDS_mpm2_mpm4" />
<Constraint op="LE" value="0.05" meas="psigs_mpm2_mpm4" />
<Constraint op="LE" value="0.1" meas="psiDS_mnm5_mnm6" />
<Constraint op="LE" value="0.05" meas="psigs_mnm5_mnm6" />
<Constraint op="LE" value="0.1" meas="psiDS_mnm7_mnm8" />
<Constraint op="LE" value="0.05" meas="psigs_mnm7_mnm8" />
<Constraint op="LE" value="0.1" meas="psiDS_mnm9_mnm10" />
<Constraint op="LE" value="0.05" meas="psigs_mnm9_mnm10" />
<Constraint op="LE" value="0.1" meas="psiDS_mpm1_mpm3" />
<Constraint op="LE" value="0.05" meas="psigs_mpm1_mpm3" />
<Constraint op="LE" value="0.1" meas="psiDS_mpm2_mpm4" />
<Constraint op="LE" value="0.05" meas="psigs_mpm2_mpm4" />
<Constraint op="LE" value="0.1" meas="psiDS_mnm11_mnm12" />
<Constraint op="LE" value="0.05" meas="psigs_mnm11_mnm12" />

```