ANALOG IC DESIGN WITH A LIBRARY OF PARAMETERIZED DEVICE GENERATORS

Vincent Bourguet, Laurent de Lamarre and Marie-Minerve Louërat

University of Paris VI, LIP6-ASIM Laboratory,
4, Place Jussieu, 75252 Paris, France
Email: Vincent.Bourguet@lip6.fr

ABSTRACT

The CAIRO+ tool allows the designer of analog circuits to automate his flow, and permits the reuse. However, it doesn't prevent him from controlling the generation. CAIRO+ is based on a library of smart generators which can handle the analog specific constraints, related to the electrical synthesis or to the design rules for a reliable layout. Here we present what are these device generators and how they can be used in a complex design automation. The example of a simple OTA using some of the available generators is shown.
Key Words : Analog design automation, Analog circuit synthesis, Layout generation.

1. INTRODUCTION

Since the review given in [1], many other studies have shown the increasing part of design automation in the design of mixed-signal integrated systems on chip [2, 3, 4, 5, 6, 7, 8]. Although there has been a lot of improvement in the topic of analog CAD, analog designers are still reluctant to use design automation, one of the problem being that there are always some steps of the design that they can not control. Today, migrating an existing function to a new set of specification for the same process, or to a new process with the same or new specifications is considered as a major but very difficult issue. Yet, with the growing part of analog circuitry in manufactured chip, it is compulsory to find how to reuse existing design and knowledge, which means that designers are required to define and use some kind of analog libraries of commonly used analog functions [9]. However, specification and technology migration of an analog circuit is still considered as requiring a lot of designer's tuning.

Here we present both the CAIRO+ language to design analog function generators as well as a library of devices, which consist of matched basic components. With such a language and generators at his disposal, we show that the analog designer is able to create complex hierarchical function generators. Although the proposed method greatly facilitates the design by automating it, it still provides the designer the entire control of the circuit generation. We will also explain how CAIRO+ generators can be independent of the technological process thus enabling process and specification migration.

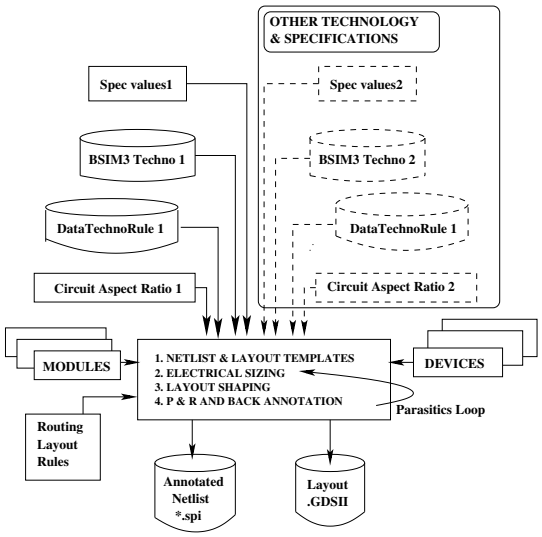


Figure 1: CAIRO+ Module generator featuring specification and technology migration

2. ANALOG IC DESIGN FLOW

2.1. State of the Art

Different approaches to analog design of integrated circuits agree that layout information has to be taken into account as soon as possible in the electrical sizing phase of the circuit [1, 10]. In order to meet this goal various flows [4, 5, 6, 7, 8] as well as various parasitics models(i.e. knowledge based estimation, layout generation with or without external extractor, look up tables of pre-characterized devices) are used. CAIRO+ allows the design to be as fast as layout-aware flows [11, 12, 6, 4, 7, 8] since the layout data are not given by any external extractor, but they are measured while generating the layout. Yet the design can be as accurate as the layout-inclusive [5] flow seeing that the layout generation is actually performed in the sizing loops.

2.2. The Proposed Flow

In the CAIRO+ approach, the circuit is described by a functional tree. The nodes of this tree are called modules and are created by module generators (Fig. 1). Therefore a module is an instance of the hierarchical representation of the circuit and it can instantiate other modules. The leaf cell of the module tree is called a device.
A module is based on three hierarchical templates : the description of the electrical topology (the netlist template), a list of specifications (the specification template) and the physical relative placement of modules (the layout template). The netlist template, as shown on Fig 2-a., is the unsized netlist of the design, described in a hierarchical way. Let us note that a module generator is dedicated to a fixed electrical topology.
In order to predict parasitics resulting from layout, we have chosen an approach using layout templates (Fig 2-b.) with layout device generators. The description of the relative placement of instances inside a module is described by a container tree. A container is composed of abutted containers placed besides each other in a specific order. There exists vertical and horizontal containers. The leaf cell of the container tree corresponds to a device.
Devices are elementary components such as folded MOS transistors, capacitors and resistances but also sets of elementary components that have to be matched (i.e. differential pair, current mirrors, capacitor matrices, matched resistances [13, 14]).
The CAIRO+ module tree, used to represent both the electrical view of the circuit (sized netlist) and the physical view (layout), allows strong interaction between electrical sizing and layout realization.

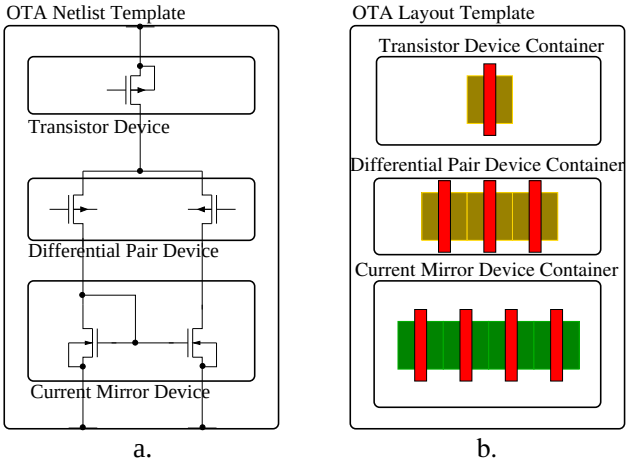


Figure 2: Netlist and layout templates for a simple OTA

The first step of the CAIRO+ flow (Fig 1) consists in creating both the electrical and physical templates. Then, the second step is the design space exploration [15, 16]. Here we check that the specifications can be reached, and if possible, we compute parameters of the sub-level generators to meet them. Once the parameter values are obtained, we proceed to the third step which shapes the circuit layout. In fact, as we will explain in section 3.3, a sized netlist can be realized by several layouts with different shapes. Therefore, the shaping step consists in calculating all the geometrical dimensions that can be taken by a module. Each of these shapes involve different parasitics, and moreover, different performances. After the shape selection by a geometrical constraint, the layout engine also measures the parasitics, with exact knowledge of the device layouts. The fourth step achieves the circuit placement and routing.
Although this flow goes until the layout generation, parasitics are not computed by an external extractor. The synthesis loop is significantly improved by that, and this flow can take place at the frontier of a layout-aware and a layout-inclusive design flow [6].

3. DEVICE GENERATORS

3.1. Introduction

We have seen that the proposed design flow of an analog circuit is based on an existing device library which is made up of both electrical and physical views. These devices are not necessarily electrical basic elements such as transistor, capacitance or resistor [17, 13, 14]. They can also be several elements tied by strong matching constraints, that can only be respected by a dedicated layout [14, 18, 19]. Typically, a differential pair is a device. They are critical components since each of the presented hierarchical steps of the flow is based on device generators. In the following we will explain how the latter handle electrical sizing, shaping and layout generation.

3.2. Electrical Sizing

The proposed method allows the designer to keep control on the sizing of the circuit. Device generators integrate a set of functions which return the electrical characteristics of a device. Then, by questioning the generator on the value of these parameters, the designer has the possibility to write a synthesis procedure, based on his own expertise of the circuit. There are two kinds of functions:

direct functions return the electrical performances of a device. The arguments taken are the biasing voltages and the dimensions of the devices

inverse functions that are able to return a dimension or a biasing voltage in order to satisfy a constraint on the circuit

The electrical synthesis step is seen as a negotiation between two levels of the module hierarchy. After specifying a number of constraints on the generator's parameters, the higher level can ask for the value of the missing parameters. An exception request is threw when the set of specifications is

unrealizable. Caught by the upper level, it is decoded to know which parameter failed. For example the width of a transistor can be computed for a given drain current, length and bias voltages. If the drain current is too high and consequently the largest drawable width cannot drive it, the upper level is informed. The latter may try another configuration.
As an example of useful built-in functions provided by CAIRO+ to the designer, the list of valid questions for the MOS transistor device is given in table 1. To calculate the width for a given drain current, the designer may fix the gate-source voltage V_{gs} or the overdrive voltage $V_{OD} = V_{gs} - V_T$, where V_T is the threshold voltage. A function that computes both W and V_{gs} to achieve I_{ds} and V_{OD} is implemented in the MOS transistor device generator.

dimensions	
W(L,IDS,bias)	gate width
L(W,IDS,bias)	gate length
biasing voltage	
VGS(VOD,W,L,bias)	gate-source voltage
static parameters	
VTH(W,L,bias)	threshold voltage
VDSAT(W,L,bias)	saturation voltage
IDS(W,L,bias)	drain current
Qx(W,L,bias)	charge at node x
STH(W,L,bias)	thermal noise density
S1F(W,L,bias)	$\frac{1}{f}$ noise density
small signal parameters	
GM(W,L,bias)	gate transconductance
GDS(W,L,bias)	drain conductance
GMB(W,L,bias)	substrate transconductance
Cxy(W,L,bias)	capacitor between nodes x et y

Table 1: Questions to the MOS transistor device generator

From the designer's point of view, handling the device generator is process independent. The same model's equations than those used in electrical simulator (BSIM3v3) have been implemented in our device generators.
Parasitics elements are accurately estimated during this step since the value of all geometrical informations, set up by the layout generation step, are known. For a transistor, the number of fingers, the diffusion area and perimeter are taken into account during the synthesis process.

3.3. Shaping and Layout Generation

One of the main difficulty in analog layout automation is to manage the wide range of aspect ratios that can be presented by a given netlist topology. In fact, the layout area of a specific analog function can easily double between two sets of specifications. Of course, migrating this function on a new process leads to the same problem.
The CAIRO+ device generators try to give an answer to that problem by having several layout aspect ratio for electrical known specifications on a target process. For example, a transistor with a gate width W can be drawn as N parallel transistors with a gate width $W_f = W/N$ while W_f respects the minimal width of the process. This technique is well known as transistor folding [14, 18, 19, 13]. These different aspect ratios are represented by a shape function which gives for example the width as function of the height, as presented on Fig 3. Thanks to that, the designer can obtain a compact layout [20].

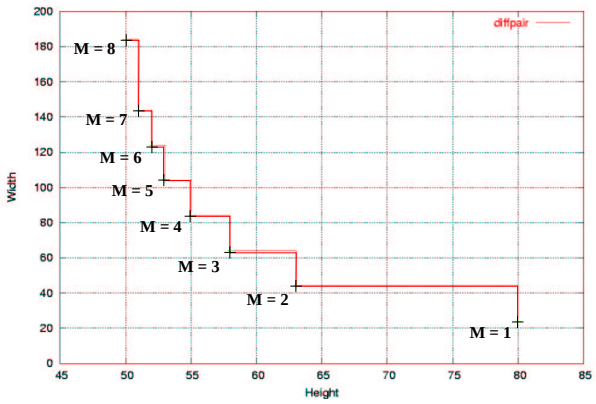


Figure 3: Shape function of a MOS inter-digitate differential pair device generator

The device generators achieve this shaping by building the layout as an abutment of simple patterns. If we consider a folded MOS transistor, patterns would be each of the "little" parallel transistors, also called transistor fingers. If we now consider a capacitor matrix, pattern would be a unit capacitor. Thus, the placement of these patterns gives the global layout aspect ratio. Fig. 4 shows two different aspect ratios -that means different patterns and different placements- of the same transistor, using respectively two and four patterns.

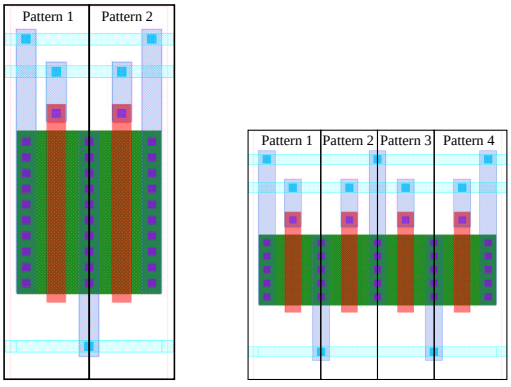


Figure 4: Pattern placement of a MOS transistor device generator

Finally, all device shapes are stored in the shape function which can be bottom-up propagated through the hierarchy to compute higher level module shape functions [20]. Thus,

the whole design layout can also present several shapes. Although the final shape is typically selected by a geometrical constraint, the designer can still impose the shape he wants for one or several devices.

3.3.1. Pattern Generators

Although patterns are not visible from the circuit designer's point of view, presenting them leads to a good understanding of the process migration mechanism. Pattern generators are very complex generators since they enclose all the layout expertise of the design. They are required to be respectful of the process rules, and to guaranty maximal accuracy for a reliable analog layout. We aim to clearly divide this complex achievement into simpler tasks. The idea is that a pattern is composed by rectangles and each rectangle can be defined by five variables : X, Y, DX, DY and the Layer. With this simple set of variables, the whole pattern layout can be drawn. But we have to define the interface between the target process rules, which is a very complex set of rules, and the pattern's simple set of variables. This is done by a set of functions called MAPI. However, MAPI has another issue which consists in separating patterns -and thus device- and the target process, which is a key factor to perform technology migration.
Let's see how MAPI compute the X, Y, DX, DY, and Layer variables. First, we have to understand two things :

1. Each pattern generator has his own MAPI at his disposal. That means that the MAPI is aware of the pattern drawing.
2. Pattern generators are device specific. That implies that they are designed to be used in a known instantiating environment. For example, a pattern knows which kind of other pattern is abutted to it.

Therefore, MAPI knows exactly the process rules it has to be aware of. Other computing such as maximums, or sums in order to get X, Y, DX, DY and Layer of each rectangle are also achieved by MAPI.

3.3.2. Process Information Access

All along the netlist and layout generation, data about the process are required. As patterns have their MAPI, devices have a set of functions, called DAPI, which gives the device direct access to the process data. For example, it is compulsory to get the W_{min} related to the target process, which is involved in both electrical synthesis and shaping computation. However, this strategy relies upon a convenient description of the target process. Therefore, we have introduced the *Device Technological Rules* files (DTR) to describe each available process. The syntax of this file has 3 purposes :

1. Making a census of all rules which may be involved, for available processes.
2. Being easily updated .
3. Being able to express that a specific rule is not involved in the target process.

Figure 5 shows the finally involved hierarchy in a device generator.

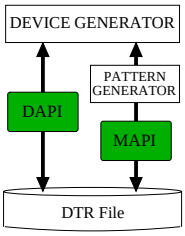


Figure 5: Internal hierarchy of a device

3.3.3. Parasitics Calculation

In order to perform an electrical and layout co-design, it is compulsory to communicate parasitics values resulting from the layout to the electrical sizing engine. To achieve this, the pattern generators return geometrical data corresponding to the layout they draw. For example, the transistor device is informed about the W, L and diffusion dimensions of each finger. Then, the device takes into account the data received from each of its patterns to compute global device data. Therefore, the design space explorer has a back-annotated netlist at his disposal, resulting from actual layout and the electrical models can estimate the resulting performances. This enables to adjust the sizing parameters to meet the specifications (parasitics loop between step 2 and step 4 in Fig. 1).

4. USING CAIRO+

Let's now present how the designer actually uses our tool. Since CAIRO+ is a knowledge based method, it aims to capture the designer's expertise. What is proposed is then a method together with a language which help him in describing his circuit. The method imposes that each node of the hierarchy, that means each module, is defined by four functions. One per step presented in section 2.2:

- CAIRO_CREATE : Template creation.
- CAIRO_GET_PARAM : Design Space Exploration, Electrical synthesis procedure.
- CAIRO_SHAPE : Shape Function Computation.
- CAIRO_LAYOUT : Placement and routing.

All of these functions have to be written by the designer in a hierarchical way. For example, the shaping function consists only in calling lower level bloc shaping function. Then CAIRO+ achieves the computation to get the shape function

of the considered bloc. Leaf functions of this hierarchy are the device generator functions. As explained is Section 3, another CAIRO+ issue is to provide this device library.
Concerning the CAIRO+ language, a set of user functions is provided to help the designer in writing each of the four module functions. Specific functions exist to create the templates, to tradeoff electrical parameters and target performances with lower levels or to achieve a procedural and shape independent routing. Here are some of the provided functions :

- CREATION :
  - CAIRO_CREATE allows instantiation of lower levels (netlist template).
  - CAIRO_LOGICAL_IO : allows the designer to describe the interface of the netlist template.
  - CAIRO_CONNECT : used to describe internal connections of the netlist.
  - CAIRO_CONTAINER : used to stack the sub-containers given as parameters in the new created one (layout template).
- ELECTRICAL SIZING :
  - CAIRO_SET_PARAM : used to specify the value of a sizing parameter to the electrical sizing procedure belonging to a child module.
  - CAIRO_GET_PARAM : used to ask the value of a parameter belonging to a child module.
  - CAIRO_TRY_GET_VALUE : used to ask the value of a constraint or a biasing data from the father level.
- P & R :
  - CAIRO_PHCON : places a physical connector at given coordinates or on a named point.
  - CAIRO_WIREx : draws a specified layer and width metal line composed by x segments from a named point to another. This function computes the parasitic capacitance which is introduced by routing.

When the design is entirely described in the CAIRO+ language, a C main function starts the flow by activating each step. At the end of the direct flow, CAIRO+ outputs a gds layout and a layout annotated sized netlist. It is up to the designer to decide if he wants to introduce a loop to reinject the parasitics involved in the electrical sizing step (Fig. 1).
With this helpful context, the designer is asked to describe his design expertise for a given netlist topology. On one hand, that means that he chooses the hierarchy that suits his wishes in terms of matching or routing complexity. On the other hand, he has to give equations to size his design, as he would have written them down on a paper. This expertise should not depend on the design performances or on the target process. That is why CAIRO+ IPs can be reused since CAIRO+ handles the process dependent informations.

5. APPLICATION : SIMPLE OTA

Here is the example of the generation of a simple OTA. This one is represented by a single-level hierarchy. In fact, the OTA module instantiates a current-mirror device, a differential pair device and a transistor device (Fig. 2). In this example, we show what happens when the unity frequency FT changes. This specification affects the device parameters which are computed by the OTA generator. In both examples, the geometrical constraint is given by $DY_{max} = 300\mu m$. Figure 6 shows the module specifications, generated layout and eldo simulation of the CAIRO+ layout aware netlist for $FT = 65$MHz. Figure 7 present the results for the $FT = 130$MHz case. Layout have of course different aspect ratio and electrical simulations are in good agreement with specifications.

6. CONCLUSION

CAIRO+ provides a method, a language and a library of device generators which allow the automation of analog circuit design. In this flow, the parasitics resulting from the layout are measured while drawing the layout. This allows to introduce a very fast sizing loop including parasitics, since all steps are achieved by CAIRO+. Moreover, the layout generation engine allows the designer to get a compact layout on different target technologies, by performing a shaping step. However, the designer keeps control of every step of the generation seeing that CAIRO+ follows a knowledge-based approach.

7. REFERENCES

[1] Gielen and Rutenbar. Computer Aided Design of Analog and Mixed-signal Integrated Circuits. *Proc IEEE*, 88(12):1825–1852, December 2000.

[2] Iskander, Dessouky, Aly, Magdy, Hassan, Soliman, and Moussa. Synthesis of CMOS Analog Circuits using AMIGO. *Design Automation and Test in Europe, DATE*, March 2003.

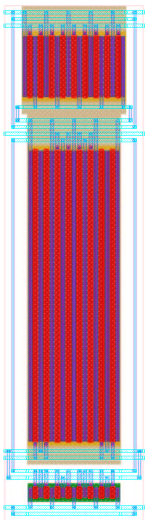
[3] Vanderhaegen and Brodersen. Automated Design of Operational Transconductance Amplifiers using Reverersed Geometric Programming. *DAC 2004*, pages 133–138, June 2004.

[4] Bhattacharya, Jangkrajarng, Hartono, and Shi. Correct-by-Construction Layout-Centric Retargeting of Large Analog Designs. *DAC 2004*, pages 139–144, June 2004.

[5] Ranjan, Verhaegen, Agarwal, Sampath, Vemuri, and Gielen. Fast, Layout-Inclusive Analog Circuit Synthesis using Pre-Compiled Parasitic-Aware Symbolic Performance Models. *Design AUtomation and Test in Europe, DATE*, pages 604–609, February 2004.

Specifications

V_{dd}	3.3V
V_{ss}	0V
C_{load}	5.0pF
V_{incm}	1.2V
V_{outcm}	1.2V
FT	65MHz



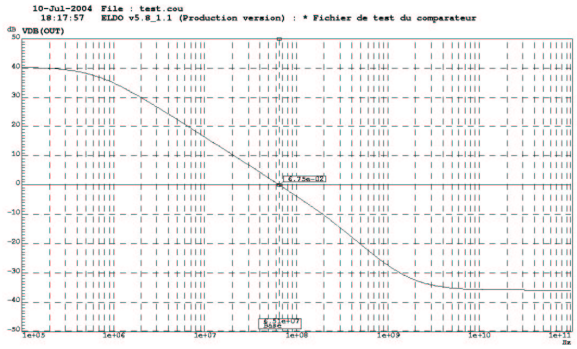
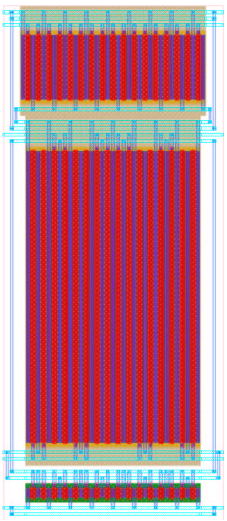


Figure 6: Simple OTA : $FT = 65MHz$

Specifications

V_{dd}	3.3V
V_{ss}	0V
C_{load}	5.0pF
V_{incm}	1.2V
V_{outcm}	1.2V
FT	130MHz



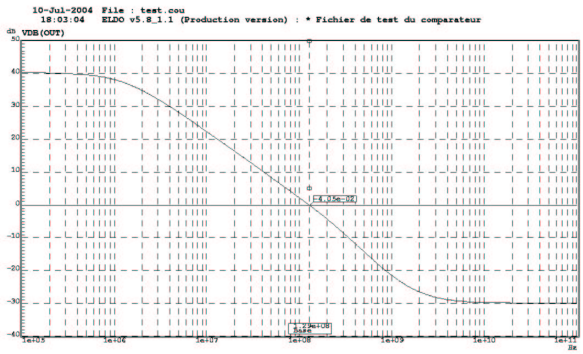


Figure 7: Simple OTA : $FT = 130MHz$

[6] Agarwal, Sampath, Yelamanchili, and Vemuri. Accurate Estimation of Parasitic Capacitances in Analog Circuits. *Design Automation and Test in Europe, DATE*, pages 1365–1366, February 2004.

[7] Xu, Pileggi, and Boyd. ORACLE: Optimization with Recourse of Analog Circuits including Layout Extraction. *DAC 2004*, pages 151–154, June 2004.

[8] Zhang, Dengi, Rohrer, Rutenbar, and Carley. A Synthesis Flow Towards Fast Parasitic Closure For Radio-Frequency Integrated Circuits. *DAC 2004*, pages 155–158, June 2004.

[9] Hershenson, Boyd, and Lee. GPCAD : A Tool for CMOS Op-Amp Synthesis. *ICCAD*, pages 296–303, 1998.

[10] de Ranter, Van der Plas , Steyaert, Gielen and Sansen. CYCLONE : Automated Design and Layout of RF LC-Oscillators. *IEEE TCAD*, pages 1161–1170, October 2002.

[11] Tang, Zhang, and Doboli. Layout-Aware Analog System Synthesis Based on Symbolic Layout Description and Combined Block Parameter Exploration, Placement and Global Routing. *Proc. IEEE Symposium on VLSI, ISVLSI'03*, 2003.

[12] Dessouky, Louërat, and Porte. Layout-Oriented Synthesis of High Performance Analog Circuits. *Design AUtomation and Test in Europe, DATE*, pages 53–57, March 2000.

[13] Bryn R. Owen, R. Duncan, S Jantzi, C. Ouslis, S. Rezania, and K. Martin. BALLISTIC: An Analog Layout Language. *CICC*, 1995.

[14] Naiknaware and Fiez. Automated Hierarchical CMOS Analog Circuit Stack Generation with Intramodule Connectivity and Matching Considerations. *IEEE JSSC*, pages 304–317, March 1999.

[15] Hershenson. Efficient Description of the Design Space of Analog Circuits. *DAC 2003*, pages 970–973, June 2003.

[16] De Smedt and Gielen. WATSON : Design Space Boundary Exploration and Model Generation for Analog and RF IC Design. *Proc IEEE*, 22(2):213–224, February 2003.

[17] Graeb, Zizala, Eckmueller, and Antreich. The Sizing Rule Method for Analog Integrated Circuit Design. *ICCAD*, November 2001.

[18] Pelgrom, Duinmijer, and Welbers. Matching Properties of MOS Transistors. *IEEE JSSC*, pages 1433–1440, October 1989.

[19] Hastings. *The Art of Analog Layout*. Prentice Hall, 2001.

[20] Nguyen Tuong, Louërat, and Greiner. Managing the Shape Function of Analog Devices in a Slicing Tree Floorplan . *Proc. of the 11th Int. Conf. on Mixed Design of Integrated Circuit and Systems, (MIXDES)*, pages 226–229, June 2004.