FACTA UNIVERSITATIS (NIŠ)
SERIES: ELECTRONICS AND ENERGETICS VOL. 9, NO.1 (1996), 109–131

ADAS: AN INNOVATIVE SYNTHESIS TOOL FOR ANALOG LAYOUT STRUCTURES

Mirko Minnich, Steffen Arlt, Ronald Böttcher, Karsten Pahnke, Reiner Selent and Gerd Scarbata

Abstract: This paper describes a prototype version of an Analog Design Assistance System (ADAS) to synthesize analog layout structures, both CMOS and BiCMOS technologies. We show a flexible cell generator system, which is integrated in a complete automatic layout system for analog circuits. The tool is based on the layout strategy "Telescopic Layout Cell"(TLC). The device abutment strategy is used to achieve maximum performance of analog circuits. It performs automatically and is controlled by both area and wiring minimization. Furthermore, ADAS is characterized by resulting compact layouts, flexible parameterization and integration into commercial design frameworks. An object-oriented strategy leads to a significant speed improvement. The implemented algorithms are described and results are illustrated by an example proving the effectiveness of this approach.

1. Introduction

Both industry and university working on design electronic systems today are experiencing intense competitive pressure. An increasinglycompetitive electronic product market requires quality products designedand delivered in shorter time-scales. Furthermore, product life cycles are becoming shorter. These points have led to an urgency in the drive for moreengineering productivity.

The use of CAE design tools is widely accepted by the industry. Additional tools for special problems are delivered from third party vendors. In the last 5 years, design supporting tools have become a common feature in the development of analog IC's. But even the design of analog IC's is characterized by a contradiction between product introduction and optimal

Manuscript received January 11, 1996.
The authors are with the Technical University of Ilmenau, Faculty of Information Techniques and Electrical Engineering, P.O.B. 327, 98693 Ilmenau, Germany.

design. That means that very often not all desired features of a given design tool are deliverable and a compromise has to be made. If the designer needs to obtain maximum performance or minimum usage of chip area, he decides to use manual design methods. This often will have an impact on the design time. On the other hand, if minimum design time is needed, the designer may choose more automated tools. Doing this he accepts that the final device may be larger and more expansive than the manually designed solution.

We present an approach which is characterized by

- automatic **Generation** of physical layout structures derived from a hierarchical schematic and input specifications,
- an **Abutment** layout representing circuit topology and structuring by parameterized layout modules,
- **Parameterization** providing a flexible configuration and universal sizing of electrical parameters,
- **Flexibility** by consequently using of a hierarchical module generator system,
- improved **Placement and Routing** by using advantages of an abutment concept,
- **Modification** by access to an extensive, expandable library and
- **Integration** into commercial frameworks to use all their features.

This paper is structured as follows. Part II outlines prior approaches on both analog circuit design strategies and analog synthesis concepts. Part III describes the basic ideas, features and related facts of the design system ADAS. The integration of ADAS into a commercial framework will be discussed in Part IV, while Part V explains experimental results of a real synthesized circuit. Finally, Part VI offers concluding points and future enhancements.

2. Background

Today's IC design process incorporates additional tools to automate the process, to improve design quality or to provide better performance values. Besides the commonly known commercial design frameworks (*Cadence* DFW II^{TM}, Mentor Graphics^{R}), there are a number of additional design tools to solve special problems in the analog design automation. The first approach is represented by pre-structured transistors from the logic field of a sea-of-gate gate-array master [1]. The unchangeable W/L ratio of the transistors limits the design flexibility. Furthermore, electrical parameters of the designed

circuits can only be varied in a small range. The use of analog cell libraries, comparable to digital standard cell systems, is another method [2]. The inflexible layout topology which does not allows universal tailoring is the disadvantage of using optimized pre-designed layout cells.

Important tools in the area of analog design automation are characterized mainly by AI-based approaches. Tools like OPASYN [3], BLADES [4] or IDAC [5] work with a selection plan choosing a suitable topology from a data base including a fixed number of topologies. A disadvantage is the time-consuming process to create new circuit topologies. Another group of tools is based on an equation-based design description. OASYS [6] carries out a hierarchical circuit description to obtain re-usable building blocks. The ARIADNE [7] [8] approach clearly separates between design knowledge and design procedures which represents a new step in the analog EDA area. A new approach introduced the synthesis framework ASTRX/OBLX presented in [9]. Starting with an un-sized circuit topology and a set of performance specifications, a complete synthesized circuit is computed automatically. The strategy combines the features of both equation-based and simulation-based approach.

However, we see a strong discontinuity between high-level synthesis results and the real physical implementation. Different techniques are developed to automate placement and routing as the critical points of physical implementation according to specific analog problems like symmetries, matching or parasitics. In ILAC [5] and MIDAS [10] macro cells of a library are placed and routed by digital tools. KOAN/ANAGRAM [11] [12] use a simulated annealing placer and a line expansion router according to symmetry and crosstalk constraints. An interesting and often neglected aspect in the area of analog circuit layout is the generation of device layout structures, especially transistor layouts. Good cell generators are useful to obtaining a minimal chip area and to avoiding typical analog problems.

3. The ADAS project

3.1 Relevant aspects for project ADAS

ADAS was developed to assist designers of analog circuits. Based on digital design strategies developed earlier, we adapted some ideas for the analog design area. Main intention for developing ADAS was to offer a tool which is able to solve monotonous tasks, which are often potential sources of designer errors. Layout generation as a time consuming design task is completely realized by module generators with the concept of TLC. Geometrical

structures of the layout are generated by using parameter and constraint specifications. The result is an error-free, design-rule conform layout.

Because analog design strongly depends on intuition and know-how of the experienced designer, ADAS supports massive interaction on each design level. Interactions are effective and convenient only with adequate performance. For this reason both modelling and implementation are performed object-oriented. The EDA specific class library *WOOD*[1], developed in our department, realizes features like high speed generation, acceptable design times, user support and both vendor- and design tool kit independence.

The demand for a short training period leads to a Graphical User Interface (GUI) with a similar "look-and-feel" like the *Cadence DFW II*^{TM}.

3.1.1. ADAS system architecture

The development of ADAS had two goals: First it has to work as a stand-alone tool with all necessary features for layout generation of analog structures. And secondly it has to save costs and has to be incorporated into commercial design frameworks efficiently. Objectives to reach this goal are

- a **data management** administrating all used data during a session,
- **data conversion** to import strange data into the application and to supply results for other tools, and
- a **visualisation** component providing the graphical presentation of layout and technology data by guaranty of a sensible control of interaction.

Data management: Data structures in an internal data model represent technology data describing design rules, symbolic data describing an abstract circuit view, and geometrical data describing the precise geometrical sizes of device layouts. Complex analytical operations are used to calculate topological and graphical relationships of layout structures. This data model can be layed on a database or an external format. Furthermore, design frameworks offer procedural interfaces to its internal database. By using these procedures, generators could be developed producing layout data directly on the framework database. ADAS uses the observer concept to synchronise ADAS database with *Cadence DFW II*^{TM} database.

Data Conversion: In ADAS input and output of layout data are handled by converters which are able to read and write standard industrial formats like *GDSII*, *EDIF2* and *CIF*. To integrate ADAS into commercial

[1] **W**hy not **O**bject-**O**riented **D**esign

design frameworks the interface formats *SKILL* and *GDTL* are available. A postscript converter provides geometrical information for printing.

Visualization: A graphical user interface summarizes all activities in ADAS. Project sensitive adjustments are easily to do and powerful layout and technology editors are implemented. Furthermore, the interaction with *Cadence DFW II*^{TM} can be controlled by GUI.

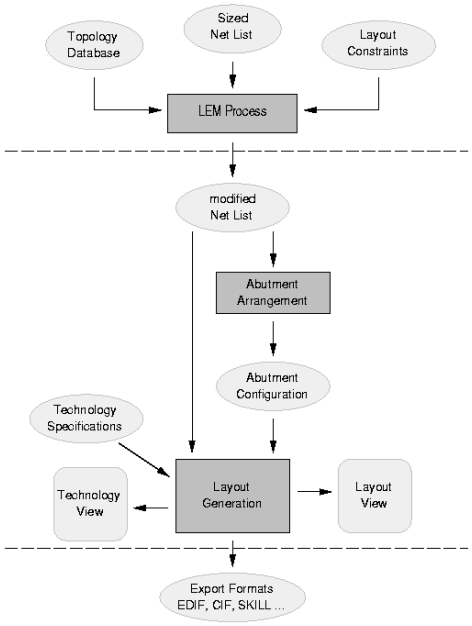

Figure 1. Structure of ADAS

Based on the three above mentioned topics, the program is structured as shown in Fig. 1. It is split into an input part (the first dashed line above), the main part for layout generation, and the export interface including incorporation possibilities.

Details to separate design phases and design procedures will be discussed later.

3.1.2 Design methodology using TLC

[13] and [14] introduce a new approach for analog layout generation developed on the layout strategy "Telescopic Layout Cell" (TLC). TLC represents the gist of ADAS project. The developed layout concept is based on

optimized layout modules. Each layout module includes up to 4 device elements. The entire layout is received by "abutment" of separate layout modules. Module abutment is used to achieve maximum performance of analog circuits. It is performing automatically and it is controlled by both area and wiring minimization. Here an abutment represents an one-dimensional arrangement of layout modules with default values either for width x or height y. The following highlights, suitable for analog layout generation, characterize this approach.

Variable Circuit Topologies: Regarding to topology demands a design system must be able to offer possibilities to add new topologies and to change topologies simply. Furthermore, technology independent layout generators are necessary. By permutation of existing layout modules, different abutment arrangements are resulting. The abutment arrangement are found by analysing a cost function. But it is optional to choose the own arrangement qualified by the designers know-how. Adding new blocks ensures adapted circuit topologies (see Fig. 2). Technology specifications are another aspect to achieve variable topologies. The choice of object-oriented implementation and the resulting strong abstraction make it possible to create nearly technology independent layout generators, which allow to switch between different technologies. The addition of new technologies is limited to the implementation of small technology-dependent leaf-classes (see 3.3.B).

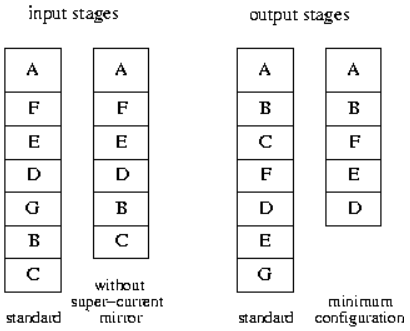

Figure 2. Different circuit topologies

Universal Parameterization: An important component of an analog design system is the free parameterization of all design sensitive parameters onto all design levels. ADAS distinguishes between different kinds of specifications. Based on a sized CDL netlist and special layout constraints made by the designer, abutment specific information with the necessary layout mod-

ules are extracted. Generally, width or height of abutment are prescribed. Layout constraints are optional. Technology information fixing layout topology will be included late in design process. As an example, device generators compute the final layout by means of device specific parameters like device type or terminal list delivered by abstract generators and the electrical values (W/L, R, C).

Adapted Routing Strategy: Prior approaches represented placement and routing algorithms as analog adapted variants of digital place and route tools based on different cost functions. Hereby, digital tools are handled with single devices. The particular cost function was more or less a combination of minimal wiring-length, avoidable cross-talks, complete rout ability and observance of parasitic constraints. Specification and execution of this optimization problem is time-consuming and does not lead always to satisfiing results. The central placement and routing model is described by our abutment concept. Our approach obeyes the above explained facts by consuming less computing time. Routing problem will be solved by stacking of layout modules. Remaining routing wires are generated simply. Fig. [3] shows the structure of a block divided into device area, routing area and terminal points. Terminal points are pre-defined and on fixed position, only depending on height and width of the block. The maximum distance between the block and device terminals and the specification of a net type leads to minimized cross-talk problems. Good routing results are achieved by sufficient routing area, fixed terminal points and the block complexity. The distance between device terminal and block terminal guarantees a minimum wiring length. Supply lines are generally routed by terminal T3 and T7. As the result of this made constraints and the execution on block level (small complexity), the routing algorithm works efficient. Interaction mechanism allows to solve complex routing constellations.

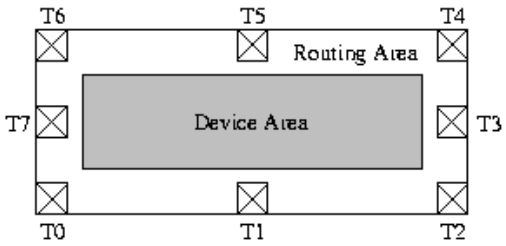

Figure 3. Routing Pins per Block

Compact Layout Results: Several real implementations using TLC

strategy shown a minimization of chip area compared to "cell-compiler" designs and a decrease of design time compared to "full-custom" designs. In comparison to the *Berkeley-Analog-Benchmarks* ("cell-compiler" designs), the chip area of layouts produced by ADAS is reduced up to 30%. Compared to manual full custom designs ADAS layouts show only a small increase of used chip area by keeping the circuit functionality. The evaluation of test transistor structures by measurement does not show negative effects between topology and electrical behaviour.

3.1.3 Implementation strategy

The ADAS is implemented in *C++* . The use of an object oriented programming language provides a number of advantages, e.g. data encapsulation, inheritance and restrictions concerning the usage of member functions. So it is easier to keep all data up to date and to prevent problems caused by the extensive usage of global variables.

Moreover, the use of *C++* results in a significant speed improvement. The results of some runtime comparisons between ADAS and equivalent *Skill* routines are shown in Table [1]

For each device structure a separate generator has been implemented. There are generators for a resistor, a capacitor, several single transistors, and some more complex structures containing up to 4 transistors like a current mirror or a differential pair. All generators are derived from a base class which provides some basic functions and defines some purely virtual functions which have to be implemented in every derived class.

One base concept which is used to implement ADAS is the *Model-View-Controller concept*. What does this concept mean? The *Model-View-Controller concept* ([15]) allows a flexible configuration of a software system with different kinds of control and data views. The model is divided into three components. The *Model* component includes the complete system functionality to manipulate its state. The *Controller* component carries out interactions between software and user. The *View* component generates and manages different views on the software. All components are connected to each other. Because both the *Controller* and the *Views* depend on the *Model* component, the automatic propagation of all *Model* states is guaranteed by using the *change propagation* capabilities. This feature provides the designer every time with an updated state of the system and prevents problems caused by data inconsistencies.

The class library *WOOD* which is used to implement ADAS provides a number of different *Views* to a *Model*. The most common are the *Lay-*

Table 1. Runtime comparisons between *C++* and *Skill* implementations

Operation	*C++*	*Skill*
int	1	350
float	1	700
struct	1	350–700
stact	1	1500
heap	1	50
database	1	100

out View to inspect the physical layout, the *Text View* to get a textual representation of the circuit, the *Technology View* and the *PostScript View* to obtain a printable version of the circuit. Other *Views* produce *C++* , *EDIF2*, *GDSII*, *GDTL*, or *CIF* output.

The principle of the *Model-View-Controller concept* is shown in Fig. 4.

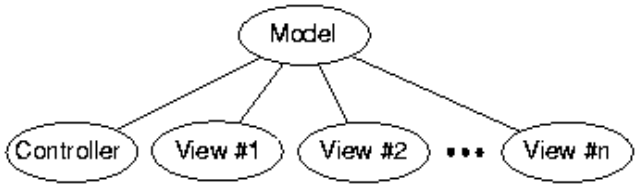

Figure 4. The *Model-View-Controller concept*

3.2 Circuit design phase

3.2.1 Input Specifications

Corresponding to Fig. 1 input specifications made up of three different parts. The most important part is a sized netlist formating in well known CDL notation. By use of a hierarchical netlist an import filter containing in LEM (Library Element Mapping) process converts hierarchy into a flat netlist. This is necessary for the following topology mapping. The topology database contains topology relevant aspects of each optimized layout module in form of a partial netlist. This netlist is represented by a graph. The database is split into five parts corresponding to the devices used in modules: bipolar, CMOS p-channel, CMOS n-channel, CMOS np-channel, and passive. Each part can be expanded simply by the designer.

As an important input specification we see the input of already known layout constraints. Layout typical information like net type, symmetry, crosstalk, topology type, etc. are specified at this point to get an optimum

mapping in the following LEM process. These layout constraints are fed into the process as follows. A parsing process through the netlist provides all containing nets and devices. The designer can assign layout constraints to each element in form of attributes. If no details are specified, proven default values are used. Particularly symmetry and topology type information are necessary information for the following LEM process to get optimum mapping results.

3.2.2 LEM Proces

LEM (**L**ibrary **E**lement **M**apping) means in this context to map a partial netlist, called block pattern, onto a larger netlist, called target netlist. It represents a transformation of circuit hierarchy, which does not influence the function of the circuit. This problem known as "Partial Graph Isomorphism Disease" is a search problem, where block pattern are identified in a flat netlist. Similar to the technology independent algorithm in [16], the block patterns are stored in a library and represented by a small netlists.

The netlist model will be used for both partial netlist and the target net list implemented in the own *C++* class library *WOOD*. The netlist is represented by a bipartite graph consisting of two nodes, *Device* and *Potential*. Every node have to connect with a node of the other type. A node is implemented in class *UGraphNode* from which *Device* and *Potential* are derived. A data element *_followers* consists of all concatenated nodes for this node. Fig. 5 gives an example of such a representation.

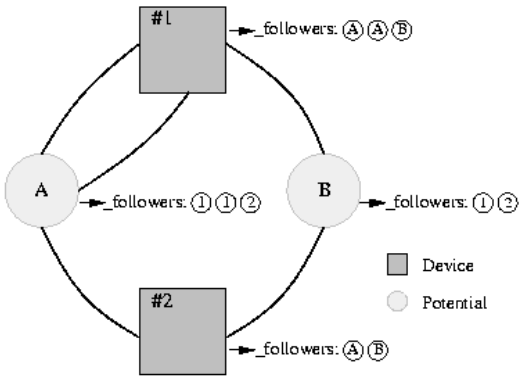

Figure 5. Partial Graph Representation

There are two disjunct partial sets, devices (M_D) and nets (M_P). The circuit graph is bipartite because each edge connects one element of M_D with

one element of M_P. The edges (M_E) and the nodes (M_N) could be allocated with attributes which represent both analog constraints (symmetry, matching or terminal permutability) and designers know-how. To solve the "Graph Isomorphism Disease" a simple finite algorithm [17] will be used. Both by using specific analog qualifications (circuit size) and a defined pattern size of a maximum of 4 devices, acceptable computation time is realized.

A necessary requirement for the isomorphic test of two graphs G(M_N, M_E) and G'(M_N, M_E') is the cardinality of both node set and edge set. By marking of nodes in graph G and G' with $1, \ldots, n$ the concatenated nodes of each pair, with the same marking, are considered. In our approach the target netlist is marked random. By marking of G' with all possible permutations and by testing of concatenated nodes per node pair, $n^2 \cdot n$ computation steps are necessary. But the computation efford is bordered by size of block pattern.

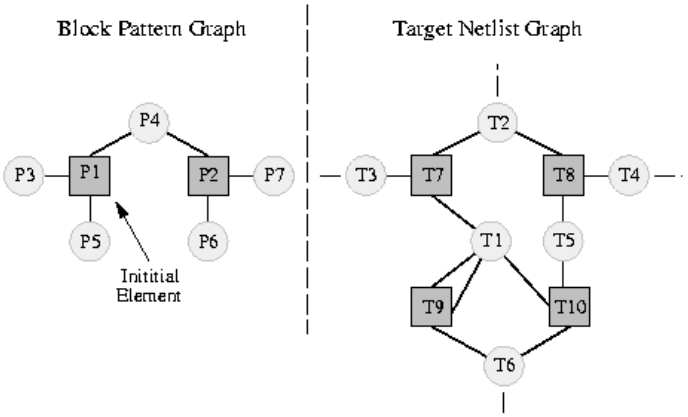

Figure 6. Example of LEM Process

The following example illustrates the LEM process, see Fig. 6. At first, the algorithm chooses an initial element (P1) of the block pattern supported by the designer. The algorithm looks for all suitable items in the target qualified by the same attributes. All *Potentials* and *Devices* arranged around the initial element (P1-T7) must be checked, whether they fit the target. Beginning at the initial element the algorithm proceeds over concatenated *Potentials* (P3, P5 or P4). At the same time same steps will process in target netlist (T3, T2 or T1). If more than 1 choice seems to be suitable, the first-matching element will be used (e.g.T2); T3, T1 remain stored. The stored elements are considered later if one step failes. Each found element in

target is bound by the element in block pattern (P1-T7, P4-T2, ...). There are no corresponding elements the algorithm resorts to the stored elements. The LEM process is successful when all elements in the sub-graph (P) fit the elements in target graph (T). Finally, *Potentials* and *Devices* in the target netlist are replaced by a block symbol (here: differential pair).

In [18] an algorithm based on hierarchical refinement and pattern matching is introduced. This approach could be interesting for a larger block pattern.

The output result of the LEM process is an internal netlist description in form of used block symbols and nets. This description is converted into a hierarchical, adapted CDL netlist. Sub-circuits in the resulting netlist represent block symbols and nets of a specific net type. Each of these sub-circuits is joined by layout specific information added to the parameter list. Thus, different models could be used for each net type (see Table 2). The different net types are characterized by different attributes. For instance a maximal wire length is prescribed for NetType0, while NetType3 is modelled by maximal length and maximal parasitic capacitance. Sub-circuits of layout modules own besides device specific layout information like W/L, R, or C, additional information about position (aT), symmetry (SB) or generator type (gT). We used the CDL netlist format with expanded parameter list, so that simulations can be execute by this netlist.

3.2.3. Abutment Arrangement Process

The previous described annotated netlist contains both all device elements and user specified layout constraints. Starting point is an indiscriminate arrangement of all required layout modules. From a decision space of $(n-1)!$ ($n \ldots nb$ of layout modules) an arrangement, complying with a maximum of a cost function is to find. Our approach is characterized by strictly dividing in fixing a cost function and following optimization process. The exact description of cost function is critical for finding suitable solutions. Costs of each abutment arrangement are computed by

$$C(x) = A_{viol}\frac{C_{viol}}{C_{viol_n}} + A_{wire}\frac{C_{wire}}{C_{wire_n}} + A_{area}\frac{C_{area}}{C_{area_n}}. \qquad (1)$$

A_{viol}, A_{wire} and A_{area} are interpreted as weights depending on design problem and design goal. Designer have to allocate a value in percentage terms in this manner that the sum of all factors is equal to 100. Each part of the cost function covers layout constraints derived from the modified

netlist. In detail: C_{viol} describes the violations of clustering. The clustering value is compound by user defined symmetry and matching constraints of coherent blocks. Fig. 7 shows an simplified abstract of the used algorithm to calculate violation costs. C_{viol_n} is the sum of all possible cluster violations in the abutment to normalize the cluster violation costs. A little bit more difficult to estimate are the net specific costs (C_{wire}).

$$C_{wire} = \sum_{i=0}^{i<w} a_{design}[i] \cdot a[typeof(w_i)] \cdot lenght(w_i). \tag{2}$$

```
for i=nb_of_cluster
    if cluster=ok then Cval=0
        else Cval=1
    sum+=Cval // sum of cluster violations
```

Figure 7. Algorithm for clustering costs

The designer can allocate a priority (a_{design}) for each net. A default value (1) is specified and can be overwritten by the designer input. The term $a[typeof(w_i)]$ describes a wire type specific correction value. Table 2 shows all possible wire types ($typeof(w_i)$) which could be assigned to a wire.

Table 2. Types of wire used in ADAS

nb	net type	realization
0	d.c. sym–non–sens	metal1 (default)
1	d.c. sym–sens	wire near to supply rail, max. distance
2	a.c. sym–non–sens	metal2, min. distance
3	a.c. sym–sens	metal1 or metal2, poly guard
4	d.c. asym–non–sens	metal1, min. distance
5	d.c. asym–sens	wire near to supply rail or subst cont., max. dist.
6	a.c. asym–non–sens	metal2, min. distance
7	a.c. asym–sens	metal1, poly guard connected to ground

The estimation of wire length $lenght(w_i)$ is based on a model of abutment. First a coarse grid are used for a rough calculation. And secondly, a refinement to estimate the nearly exact wire length are performed by real terminal positions.

Furthermore, we work on the inclusion of dynamic description of cost function specified by designer and realized by an interpreter approach.

Result is a cost value for each possible arrangement. There are two ways to find an optimum arrangement. At first, the designer can carry out a rough optimum by means of simple heuristics in a small decision space. Starting with a randomly selected initial block, the most distant path through the abutment graph is chosen. By a complexity of $O(n)$, a solution is easy to find and more than one variants can be examined. A clever selection of initial block reach to a satisfied abutment. To achieve exacter results, standard optimization methods like simulated annealing, threshold method etc. can be used in that process as a second way. Good results could be achieved by using simulated annealing, while genetic algorithms lead to exhaustive performance requirements.

3.3. Physical design phase

3.3.1 Layout generation process

The generation of the physical layout in ADAS based on a hierarchical library of module generators works on three different hierarchy levels. Fig. 8 clarifies the separated levels of generator hierarchy representing simultaneously both the topological hierarchy of generated layout and the dependencies of a generator with one other.

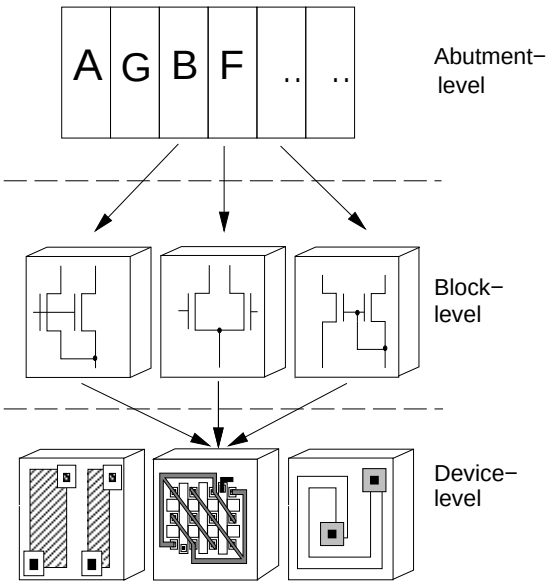

Figure 8. Module Generator Hierarchy

Module generators of each level realise the following topology representation

- **Configuration level**: Generation of abutment arrangement with all blocks contained in the abutment list,
- **Block level**: Generation of block topology, that includes calls of necessary device generators and generation of inter-block connections in form of wires, and
- **Device level**: Layout generation of each device topology (MOS transistors, bipolar transistors, R, C)

The advantage of using a hierarchical form is the short time to topology changing connecting with an high degree of flexibility. As an example, it is possible to substitute the layout of a MOS comb transistor by a waffle-shape transistor or serpentine transistor. To obtain device matching requirements the generation of common centroid and inter-digitized topologies by special module generators are realized. A next advantage of the module generator concept exists in the reduction of costs for technology changing. By using of conforming technologies "technology independence" is nearly reached.

Inputs for the layout generation process are modified netlist, constraints like design-rules (technology) and a global parameter set containing prescribed values for design width and aspect ratio of design height or width. The synthesis flow of a concrete design is essentially formed by an iteration of generator calls. At each hierarchy level all input information is available. Inputs for inferior module generators (e.g. sizing points) are produced by higher module generators.

The adapted router is also based on a generator library. For each type of wire, shown in Table 2, exists a module generator to realize the physical structure. The routing is restricted mainly to create wires between block terminals and device terminals, that means to carry out a mapping of electrical nets of each block to block terminals.

Wires which are not covered by abutment are arranged in an external routing channel. A simple over-the-cell router is implemented for the external routing process. The router acquires input specification from the netlist, the abutment list and layout information for wire specific data. Fig. 9 shows an example of layout realisation of an Op Amp as part of a *Chua Chaos* circuit described in Part 5.

3.3.2 Technology mapping

Programming of technology independent generators for CMOS and

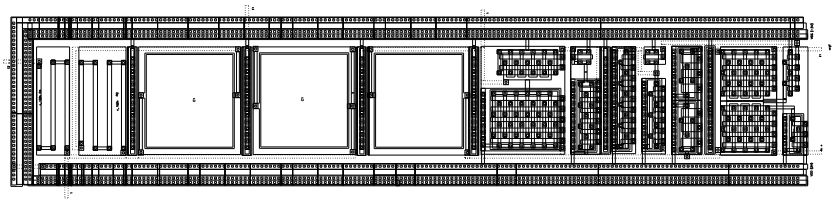

Figure 9. Example of a generated abutment layout

bipolar devices is a complex problem. Analog circuit design is not restricted to CMOS technologies. Different complex bipolar and BiCMOS technologies become more significant. Logically, CAD tools for analog layout design should not have hard coded technology specifications. Besides different transistor topologies, certain transistor components differ in layers, design rules and construction (used layers). For this reason all device structures are assembled by sub generators (leaf-cell-generators). For example, a CMOS transistor is generated by drain/source-, gate-, contact- and substrate generators. Abstract generator classes for device elements (CMOSTranGen, BipolarTranGen, ResGen, CapGen, etc.) carry out the construction of a device. Technology relevant information is used only in leaf-cell-generators.

A general set of design rules describes minimum design sizes (widths, distances, overlapping) for all available layers and supplies detailed information about the electrical parameter of the technology (area resistor, area capacity). This piece of information is sufficient for a design rule conform fine layout generation. For the reason of scaling and calculation, all geometry values are derived from the basic unit λ.

The usage of an abstract technology model with interfaces leads to a good linking between the actually technology and technology aspects in all phases of the layout synthesis. Different classes are implemented to realize this data model. Technology descriptions used in generators represent a mix of pure data for design rules and specific code fragments. The code fragments are implemented in leaf-cell-generators which produce the detailed technology specific layout elements. That is one way to write generators nearly technology portable.

A next advantage of this adaptive method is the expansion of new layout structures. Furthermore, leaf-cell-generators are compact and easy to program by the user. The object-oriented implementation is suitable for such kind of the generator realization.

4. Framework integration

An important aspect of the development of ADAS was to support the designer with a known, user-friendly environment. For several years our design projects has been realized by means of *Cadence DFW II*^{TM}. For this reason we looked for possible ways to integrate ADAS into *Cadence DFW II*^{TM}. In addition to this well-known design framework, we offer an interfaces to *Mentor Graphics*^{R}.

There are three different ways for integration:

- expansion of the *SKILL* interpreter,
- start of separate external programs, and
- start of an external frame program.

The first way could not be realized because we do not have access to the necessary integration package distributed by Cadence. But a concept for such a solution is put forward. To support the integration of external programs, a strategy for automated script-generation for all tools sufficing minimal requirements for parameter mechanism is implemented (way 2). The third way uses a frame program with strong interactions. This program can be invoked by *Cadence DFW II*^{TM} and is described below.

Generally, following problem is indicated: *How it is to distribute and manage generated objects?* A synchronization mechanism must ensure the exchange of data located in system memory between external tool and framework. Our implementation uses a well-known observer concept ([19]). That means, a model may have any number of dependent observers. All observers are notified whenever the model undergoes a change in state (see 3.1.C)). There is a class hierarchy for layout objects. Instances of layout objects are created by a factory class. The factory class instance sends out messages controlled by an observer. For each synchronisation job exists a separate observer. General disadvantageous are higher performance costs. Because ADAS and *Cadence DFW II*^{TM} are side by side processes, a separation of both is possible.

There are different implementation variants for synchronization ADAS with *Cadence DFW II*^{TM}. The system have to offer data to *SKILL* interpreter (see Fig. 10). That are

I direct communication by standard I/O,

II generation of *SKILL* files and down loading in *Cadence DFW II*^{TM},

III the usage of a intercommunication interface (pipes of commands and messages),

IV C database interface ("db.h") integration of *Cadence DFW II*^{TM}

Currently, the first three ways are realized, in course of which way 1 is only used for test purposes. The usage of pipe interface from *Cadence DFW II*^{TM} is about 10 times slower than communication by external files. The intercommunication interface uses channel 3 and 4; ADAS is always child process. Its seems, that an internal polling process testing the command port, is not implemented efficiently. Disadvantages are further the poor test facilities. Parallel to previously explained ways we prepared direct synchronization via procedural C interface of database mechanism. This mechanism should be equivalent in speed to the generation of objects. In contrast to the communication using *SKILL*, functions offered in database interface ("db.h") can be bound in own *C++* sources. The external application assumes control.

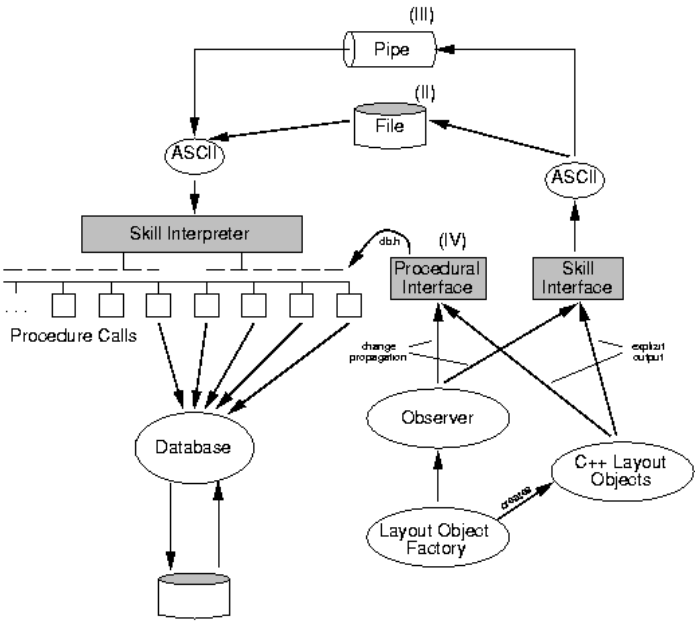

Figure 10. Communication model

5. Application example

5.1. Test Circuit

There is an increasing need for applications in the area of message encryption systems, noise-generating systems and novel sensor systems which

use different types of signal processing functions (conversion, analysis, detection, transmission) by means of noise, periodic or stochastic signals. Several simple, nonlinear circuits are well-known which have a great variety regarding to their dynamic behaviour. Dependent on certain parameters different types of signals can be generated (noise signals, ad-hoc chaotic signals). This paper presents an integrated circuit that realizes the dynamic behaviour of the *Chua Chaos* circuit. The electric equivalent circuit of the *Chua Chaos* circuit is shown in Fig. 11.

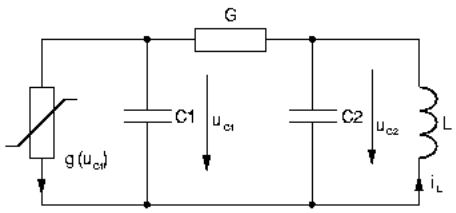

Figure 11. Electric equivalent circuit

Regarding to the electric equivalent circuit the state equations of the circuit were derived and transformed to the basic circuit topology. Fig. 12 shows the block structure of an integrated realization by means of integrators and a feedback network. This set of analog arithmetic units is realized by operational amplifiers (used as integrators), a resistive feedback network and diodes (supply of nonlinear terms).

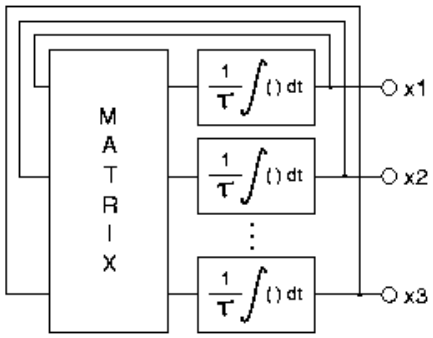

Figure 12: Block structure of *Chua Chaos* circuit

An application example of the *Chua Chaos* integrated circuit is illustrated in Fig. 13 and consists of the IC, external passive devices and switches.

The dynamic behaviour is adjustable by means of three external capacitors which represent the capacitors and the inductance of the original circuit (see Fig. 11). Default is to use three equal capacitors to get the double scroll attractor. It is possible to increase the capacitor corresponding to the original C_1 by switching on S to get the Roessler attractor.

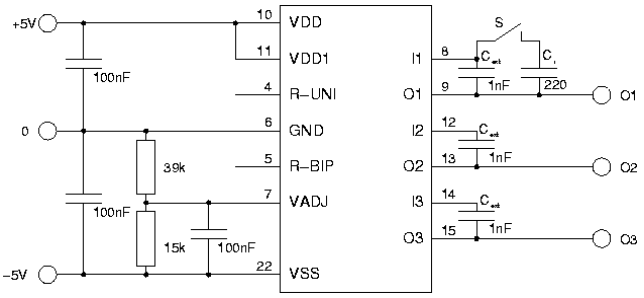

Figure 13. Application of Chua circuit

The diagram in Fig. 14 shows as an example the Lissajous figure of output 2 versus output 1 for $C_{ext} = 1$ nF. Experimental results confirmed the results of simulation and the dynamic behaviour was rather good adjustable over the whole dynamic range.

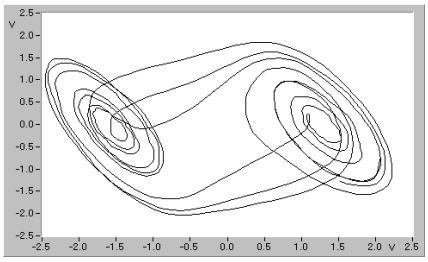

Figure 14. Double scroll attractor

5.2. Chip Layout

The chip implementation of the *Chua Chaos* circuit was realized by a 2.4 μm CMOS technology, see Fig. 15. Based on the block structure (Fig. 12) the whole circuit was partitioned into different sub-circuits (integrators, feedback network). Each sub-circuit represents an own TLC sub-cell in the layout view. The layout generation process was done by means of the ADAS. Certain parameters (W/L-ratio, C-value, R-value) were specified for

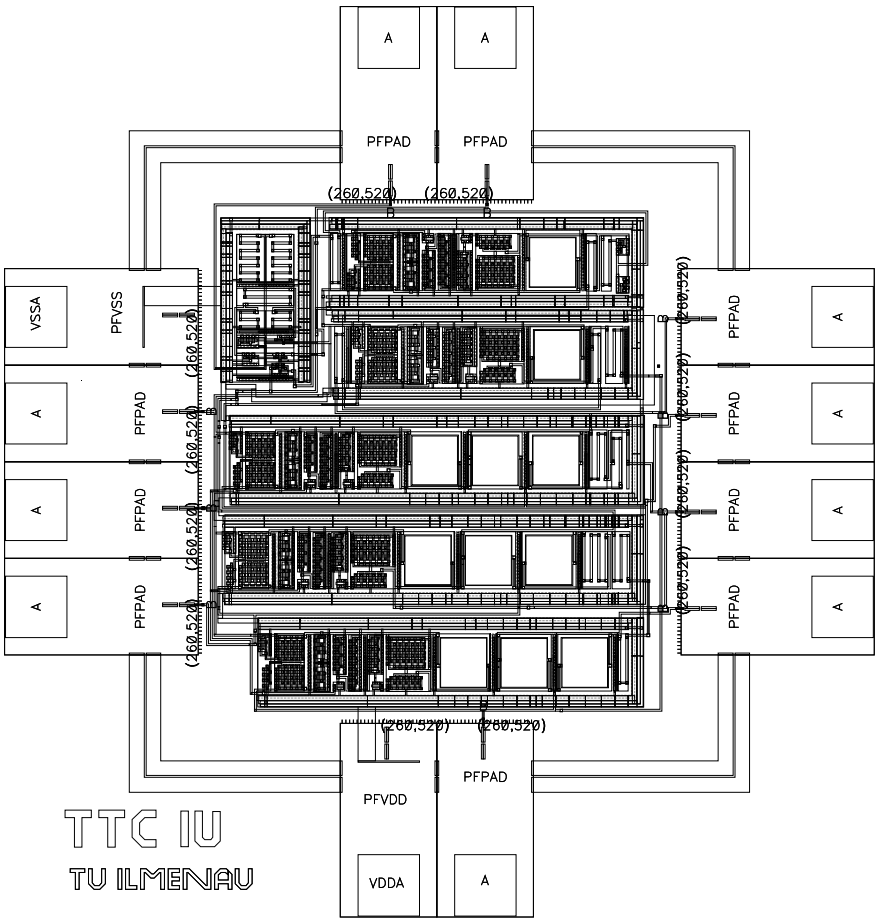

Figure 15. Final layout of *Chua Chaos* circuit

each sub-cell. These parameters controlled the automatic layout generation process based on the layout concept of TLC. The core layout of each sub-cell (see Fig. 9) is surrounded by a guard ring to improve the EMC characteristics. The guard ring is laid out in a special manner to allow a dynamic buffering of the power supply, to minimize the cross-talk between the layout modules and to avoid the propagation of current spikes and glitches. The placement and routing of all sub-cells were performed in two ways to test the performance of the *Cadence Virtuoso Custom Layout Tools*. In the case of manual placement and routing the chip area consumption takes only 70 per-

cent as in case of automatic placement and routing by means of the *Cadence DFW II*^{TM} tools but the loss of time was higher.

6. Conclusions and future work

An approach to synthesize analog layout structures has been reported. A fundamental advantage of our approach is a drastic reduction of design time by using an improved module generator concept based on the layout strategy *Telescopic Layout Cell* and an object-oriented implementation. The TLC strategy is characterized by variable topologies, universal parameterization and an adapted routing strategy which offers a high-performance generation of analog layout structures. A module generator system places all devices in abutment with respect to minimal area and analog-specific constraints.

The program has been used to design circuits in several technologies to evaluate the TLC strategy. Since measurement and simulation results do not differ, the program has proven to be a useful design tool. The integration with a commercial framework makes analog design easy accessible to both novice and expert designers in a well-known environment. Design knowledge can enter on each design level by interactive use.

Further work includes (I) the expansion of a mapping process for more than one hierarchy, (II) the refinement of the cost function and (III) the handling of more layout constraints. Also alternative approaches for optimization algorithms to device abutment will be investigated. An immediate goal is the specification of an interface to integrate high-level analog synthesis ideas published in [20].

Acknowledgements

This research has been supported by a generous grand of Deutsche Forschungsgemeinschaft (DFG). We thank A. Mögel and W. Schwarz of Technical University Dresden for providing the *Chua Chaos* circuit. Furthermore, the authors wish to acknowledge the use of the fabrication facilities at *Alcatel-Mietec* supporting by *Eurochip* to fabricate the circuit in this work.

REFERENCES

1. S. Kawada, Y. Hara, T. Isono and T. Inuzuka: 1.5μ *CMOS Gate Arrays with Analog/Digital Macros Designed Usign Common Base Arrays.* IEEE Journal of Solid-State Circuits, vol. 4, 1989, pp. 985–990.
2. M. Smith, C. Portmann, A. Constantine, P. Tschang, R. Ramu, P. Valdenaire, and Ch. Holly: *Cell Libraries and Assembly Tools for Analog/Digital*

CMOS and BiCMOS Application-Specific Integrated Circuit Design. IEEE Journal of Solid-State Circuits, 1989, pp. 1419–1431.

3. K. Young, S. Carlos, and P. Gray: *OPASYN: A Compiler for CMOS Operational Amplifier.* IEEE Transactions on Computer-Aided Design, vol. 2, 1990, pp. 113–115.
4. E. El-Turkey and E. E. Perry: *BLADES: An Artificial Intelligence Approach to Analog Circuit Design.* IEEE Transactions on Computer-Aided Design, vol. 8, no. 6, June 1989, pp. 680–691.
5. M.G. Degrauwe, O. Nys, E. Dijkstra, J. Rijmenants, S. Bitz, B. Goffart, E. Vittoz, S. Cserveny, Ch. Meixenberger, G. VAN DER sTAPPEN, AND h. Oguey: *IDAC: An Interactive Design Tool for Analog CMOS Circuits.* IEEE Journal of Solid-State Circuits, vol. 1, no. 6, 1987, pp. 1106–1116.
6. R. Harjani, R. Rutenbar, and L. Carley: *OASYS: A Framework for Analog Circuit Synthesis.* IEEE Transactions on Computer-Aided Design, vol. 12, 1989, pp. 1247–1265.
7. G. Gielen, H. Walscharts, and C. Sansen: *Analog Circuit Design Optimization Based on Symbolic Simulation and Simulated Annealing.* IEEE Journal of Solid-State Circuits, vol. 25, June 1990, pp. 707-713.
8. K. Swings and W. Sansen: *ARIADNE A Constraint-Based Approach to Computer-Aided Synthesis and Modeling of Analog Integrated Circuits.* chapter 2, pp. 37–55, Analog Integrated Circuits and Signal Processing. Kluwer Academic Publishers, Boston, 1993.
9. E.S. Ochotta, R.A. Rutenbar, and L.R. Carley: *ASTRX/OBLX: Tools for Rapid Synthesys of High-Performance Analog Circuits.* Proceedings of the 31-st Automation Conference, 1994.
10. G.F.M. Beenker, J.D. Conway, G. Schrooten, and A.G.J. Slenter: *Analog CAD for consumer ICs.* Proc. Advances in Analog Circuit Design, 1992.
11. D. Garrod, R.A. Rutenbar, and R.L. Carley: *Automatic Layout of Custom Analog Cells in ANAGRAM.* Proc. 1988 IEEE Int'l Conf. on CAD, November 1988.
12. J.M. Cohn and et.al.: *KOAN/ANAGRAM II: New Tools for Device-Level Analog Placement and Routing.* IEEE Journal of Solid-State Circuits, vol. 26, no. 3, March 1991.
13. S. Arlt, G. Scarbata, S. Ritter, and C. Wisser: *Telescopic Layout Cells for Analog CMOS Circuits.* in Proceedings of Europian Design and Test Conference, 1992, pp. 139–143.
14. R. Böttcher, S. Arlt, J. Kampe, M. Minnich, and G. Scarbata: *Design Framework for A/D Circuits Based on Generator Concept.* in Proceedings of European Design Automation Conference (Software Exhibition), 1994.
15. M.A. Linton, J.M. Vlissides, and P.R. Calder: *Composing User Interfaces with InterViews.* Computer, vol. 22, no. 2, February 1989, pp. 8–22.
16. F. Luellau, T. Hoepken, and E. Barke: *A Technology Independant Block Extraction Algorithm.* in 21-st Design Automation Conference, 1984, pp. 610–615.
17. R.C. Read and D.G. Corneil: *The graph isomorphism disease.* Journal of Graph Theory, vol. 1, 1977, pp. 339–363.
18. G. Pelz and U. Röttcher: *Pattern Matching and Refinement Hybrid Approch to Circuit Comparism.* in IEEE Trans. on CAD of Integrated Circuits and Systems, 1994.
19. G.E. Krasner and S.T. Pope: *A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80.* Journal of Object-Oriented Programming, vol. 1, no. 3, August/September 1988, pp. 26–49.
20. J. Kampe and G. Scarbata: *Automatisierbare Synthesemethoden für analoge Systemkomponeneten.* in 6. E.I.S. Workshop, November 1993, p. 25./26.