DARWIN: CMOS opamp Synthesis by means of a Genetic Algorithm

Wim Kruiskamp and Domine Leenaerts
Eindhoven University of Technology, Faculty of Electrical Engineering
P.O. Box 513, 5600 MB Eindhoven, the Netherlands
e-mail: M.W.Kruiskamp@ele.tue.nl

Abstract—DARWIN is a tool that is able to synthesize CMOS opamps, on the basis of a genetic algorithm. A randomly generated initial set of opamps evolves to a set in which the topologies as well as the transistor sizes of the opamps are adapted to the required performance specifications. Several design examples illustrate the behavior of DARWIN.

I. Introduction

The analog part of a complex mixed-signal system is often small compared to the digital part. However, the design of the analog part is often the most time consuming part in the entire design. The reason for this is that analog library cells are usually not suitable for all required analog functions [1]. The remaining analog circuits have to be designed by hand or preferable with the help of analog synthesis tools.

Analog circuit synthesis can be divided into two strongly related tasks: The selection of a suitable circuit topology and circuit sizing. During the topology selection, it is very useful if the topologies are already sized. The sized topologies, with known bias currents, capacitor values and transistor dimensions can then be evaluated and the best can be selected. An example of this approach is IDAC [2]. IDAC sizes several topologies and the user selects the sized circuit with the best performance. The main drawback of this approach is the fact that a lot of topologies have to be sized completely, of which only one will be used. To avoid this computational overhead, other tools, like OASYS [3] and OPASYN [4], select on forehand one topology, based on heuristics. If the tool can not size the selected topology correctly, other heuristics are used to redo the topology selection. Unfortunately, these heuristics are very difficult to create and there is a risk that a non-optimal topology is selected. In SEAS [5], the knowledge intensive topology selection heuristics are avoided by using an evolution algorithm to modify the topology. However, the several intermediate topologies still have to be sized completely in SEAS, using a time consuming simulated annealing algorithm. The first methodology that handles topology selection and circuit sizing simultaneously was presented by P. Maulik [6]. In this approach, topology selection is embedded in the circuit sizing problem. Therefore the risk of selecting a non-optimal topology is reduced, without the need of sizing many topologies. However, a lot of expert design knowledge is required to generate the essential design equations.

In this paper, we present a prototype synthesis tool DARWIN, which is based on a genetic algorithm [7, 8]. The genetic algorithm maintains a population of possible solutions, of which the topologies as well as the parameters gradually evolve to the final solution. The main characteristics of DARWIN are:

- Simultaneous topology selection and circuit sizing.
- Topologies are build up from basic building blocks by the program itself.
- Topology selection and circuit sizing are performed by means of a genetic algorithm.
- Only a little amount of expert design knowledge is required in the program.
- At the beginning of the synthesis, a set of constraints is solved for each building block to ensure that all intermediate generated circuits behave properly.

The organization of the rest of the paper is as follows. Section 2 describes the circuit description that is used in the genetic algorithm. Section 3 describes the genetic algorithm itself. Section 4 illustrates the behavior and performance of DARWIN by means of several design examples. Concluding remarks are provided in section 5.

II. Circuit Representation

Genetic algorithms are based on evaluation and generation processes, applied on a certain problem representation. When a sized netlist of a circuit is available, the evaluation of that circuit can be performed by means of simulations. The generation of new circuits however, will be much more complicated. For example in the domain of opamps, hundreds of suitable topologies are known, but the number of topologies that will not be able to behave as an opamp is countless. Furthermore, an unlucky chosen set of transistor dimensions and bias currents can make a circuit useless as an opamp. Therefore some restrictions must be defined to ensure that all intermediate generated circuits behave as an opamp. In section A, the topology aspects of this problem will be discussed, while the sizing aspects will be discussed in section B.

A. Topology description

The topology of an opamp can be separated into three building blocks: an input stage, an optional second gain

32nd ACM/IEEE Design Automation Conference ®

stage and an optional output buffer. Several different topologies are possible for each building block.

On the basis of these three sets of building blocks, a large variety of topologies can be constructed. However, only a limited set of these topologies will be useful as opamp topology. By means of the connection matrix of Fig. 1, valid topologies can be recognized. A cross in entry (*i*, *j*) of the matrix indicates that block *i* might be succeeded by block *j*. As can be seen in the connection matrix, there are currently available: four input stages (element 2 ... 5), four second stages (element 6 ... 9), and three output buffers (element 10 ... 12). Element 1 and element 13 represent the input connection respectively the output connection of the opamp. Together, this results in 24 valid topologies, varying from just a simple input stage {1-2-13} as depicted in Fig. 4, to an opamp consisting of a folded cascode input stage, a second gain stage, and a class AB output buffer {1-4-8-12-13}. It is obvious that the connection matrix is easily to extend with new building blocks.

B. Building block description

In order that each building block behaves correctly, a set of simple constraints can be derived for the separate building block topologies, on the basis of the process parameters and the required specifications. An example of such a constraint is that all transistors in the input stage have to operate in saturation mode. Furthermore, some simple building block constraints can be derived out of the overall opamp specifications. For example, if the overall DC-gain has to be at least 80 dB, an input stage with a DC-gain of only 30 dB will not likely be part of an optimal design. Therefore when we require the DC-gain of the input stage to be at least 30 dB, we will limit the design freedom for the circuit parameters without loosing an optimal solution. Furthermore, since the proposed constraints will only be defined for the separate building blocks, the constraints will be much simpler than constraints for an entire opamp, as used in [6].

For the solution method we use, equations may only be linear. However, most of the equations that map the circuit parameters on the building block specifications are composed out of multiplications and square roots of the parameters. These constraints can therefore be written as linear combinations of the logarithms of the parameters. The only problems are combinations of products and summations. However, when the resultingspace is convex, this problem can be overcome by approximating the (nonlinear) constraint by a piecewise linear space.

	2	3	4	5	6	7	8	9	10	11	12	13
1	×	×	×	×								
2						×			×			×
3					×					×		×
4							×			×		×
5								×	×			×
6									×	×	×	×
7									×	×	×	×
8									×	×	×	×
9									×	×	×	×
10												×
11												×
12												×

1 = input
2 = nMOS diff. pair (simple)
3 = pMOS diff. pair (simple)
4 = nMOS folded cascode
5 = pMOS folded cascode
6 = nMOS Com. Source (CS)
7 = pMOS CS
8 = nMOS CS with level shift
9 = pMOS CS with level shift
10 = nMOS-source follower
11 = pMOS-source follower
12 = class AB output buffer
13 = output

Fig. 1. Connection matrix

The expression for the DC-gain of a simple differential input stage with nMOS input transistors, as depicted in Fig. 4, is derived as follows:

$$\mathrm{DCgain} \geq 30\mathrm{dB} \rightarrow \begin{cases} \sqrt{\dfrac{I_{ss}}{L_1 \cdot W_1}} \leq \dfrac{\sqrt{K_n}}{31.62 \cdot \lambda_n} \\ L_1 \cdot \lambda_p = L_2 \cdot \lambda_n \end{cases} \Rightarrow \quad . \tag{1}$$

$$\begin{cases} \frac{1}{2}\log(I_{ss}) - \frac{1}{2}\log(L_1) - \frac{1}{2}\log(W_1) \leq \log\left(\dfrac{\sqrt{K_n}}{31.62 \cdot \lambda_n}\right) \\ \log(L_1) - \log(L_2) = \log(\lambda_n / \lambda_p) \end{cases}$$

As in (1), all constraints which ensure that a particular building block operates correctly, can derived. This results in a set of linear constraints for each building block in the format:

$$\begin{cases} \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b} \\ \mathbf{C} \cdot \mathbf{x} = \mathbf{d} \end{cases} \quad \text{, with } \mathbf{x} = \left(\log(W_1), \log(L_1), \cdots\cdots, \log(I_{ss})\right) \quad . \tag{2}$$

The solution of this set of constraints is a convex space of which the corner points $\mathbf{p}_i$ can be calculated with the algorithm of Tschernikow [9, 10]. The convex space given by (2) can then completely be described in the format:

$$\mathbf{x} = \left(\sum_{i=1}^{k} \alpha_i \cdot \mathbf{p}_i\right) \Bigg/ \sum_{i=1}^{k} \alpha_i \quad , \quad \underset{i}{\forall} \alpha_i \geq 0 \cdot \tag{3}$$

Expression (3) states that any k-dimensional vector α without negative entries α_i results in a vector **x** that meet all the requirements of (2). In DARWIN, buildingblocks are described by their name and a non-negative k-dimensional vector $\boldsymbol{\alpha}$. On behalf of computational simplicity, the values for the elements α_i are restricted to the values '0' and '1'. Due to this restriction, only 2^k points out of the solution space can be reached for each stage. In most cases however, this number will be large enough for satisfactory synthesis results. An example of this description is depicted in (4):

$$\begin{pmatrix} W_1 \\ L_1 \end{pmatrix} = \frac{\alpha_1 \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \alpha_2 \cdot \begin{pmatrix} 3 \\ 1 \end{pmatrix} + \alpha_3 \cdot \begin{pmatrix} 4 \\ 4 \end{pmatrix} + \alpha_4 \cdot \begin{pmatrix} 3 \\ 5 \end{pmatrix} + \alpha_5 \cdot \begin{pmatrix} 1 \\ 3 \end{pmatrix}}{\sum_i \alpha_i} \quad , \quad \forall \alpha_i \in \{0,1\}$$

$$\boldsymbol{\alpha} = (0,\ 1,\ 1,\ 1,\ 0) \ \Rightarrow \qquad (W_1,\ L_1) = \left(3\tfrac{1}{3},\ 3\tfrac{1}{3}\right) \tag{4}$$

III. GENETIC ALGORITHM

Genetic algorithms or evolution programs [7, 8] can be used to find a near optimal solution for a wide variety of problems. Genetic algorithms maintain a population of individuals **P**(t) for iteration t. Each individual is an abstract representation of a potential solution of the problem at hand. During each generation, all individuals are evaluated to give some measure of their fitness. On the basis of this fitness, part of the population is selected to maintain for the next generation. In this selection, the more fit individuals have more chance to survive. The vacant places in generation *t+1* are filled up by new individuals, generated by means of cross-over applied on the selected individuals. After a number of generations, the population converges to a population in which the best individual hopefully represents the optimal solution. This flow is depicted in Fig. 2.

In the synthesis tool DARWIN, this basic algorithm is implemented as follows. To ensure that the initial population and each following population contain at least consistent solutions to the problem (but not all of them are necessarily the best in context to the fitness function), a set of simple constraints is solved which ensure that all transistors operate in their proper region and that the transistor sizes are between maximal and minimal values. The solution is for each building block a set of vectors **p**, as discussed in section B. This prevents the algorithm to come up with circuits which are not behaving as opamps.

In the initial population **P**(t_0), the topology for each of the opamps is picked randomly out of the 24 possible opamp topologies. The entries in the vectors α for each opamp are set randomly to the value '0' or '1' (each entry in the vectors α corresponds to a vector **p**, as discussed in section B). A circuit description for each opamp can be obtained by a linear combination of the vectors **p**. All circuits in the initial population will now behave as opamps and the separate stages in each opamp will meet a minimum set of requirements. Nevertheless, the overall specifications of the opamps can still be far away from the required specifications.

It is now easy to derive the sized netlist for each opamp out of the circuit description. The performance of each opamp is then evaluated by means of a small signal equivalent circuit and analytical expressions. A fitness value is assigned to each opamp. When a specification of an opamp does not meet the requirements, this results in a negative contribution to the fitness of that opamp, proportional to the gap between specification and requirement. When all requirements are satisfied, the fitness function is defined by an optimization function based on power dissipation.

When all opamps are evaluated, their fitness values are linearly scaled in a way that the maximal fitness is equal to 1 and the minimal fitness is equal to 0. Several of the opamps are then removed out of the population. The number of removed opamps is equal to the population size, multiplied by the cross-over rate. For each opamp, the chance to survive is proportional to its fitness. Newly generated opamps fill up the resulting vacant places in the population. Each new opamp is generated out of two ‘parent’ opamps which are selected with a chance that is again proportional to their fitness. The new opamp is made by means of the cross over operation, as depicted in Fig. 3. The new topology is constructed by mixing the parents’ building blocks. The first block of parent *A* is copied to the new topology. If, according to the connection matrix of Fig. 1, the second block of parent *B* might be placed behind the first block of *A*, this will be done. Finally, the last building block will be copied from the parent who did not deliver the second building block, if allowed by the connection matrix. Furthermore, the circuit parameters of the two parents are mixed if the parents contain similar building block(s). This is the case for the input stage of the example in Fig. 3. In that case, the α-vectors of the two parents are mixed. The α-vectors are cut at two random points and the separate parts are use in the new α-vector. In the case that the two parents have three stages in common (the same type of input stage, second stage and buffer stage),

```
procedure evolution program{
        t = 0;
        initialize P(t);
        evaluate P(t);
        while (not finished){
                t = t + 1;
                select P(t) from P(t-1);
                make new members with cross-over;
                insert new members in P(t);
                mutate P(t);
                evaluate P(t);
        }
}
```

Fig. 2. Genetic algorithm

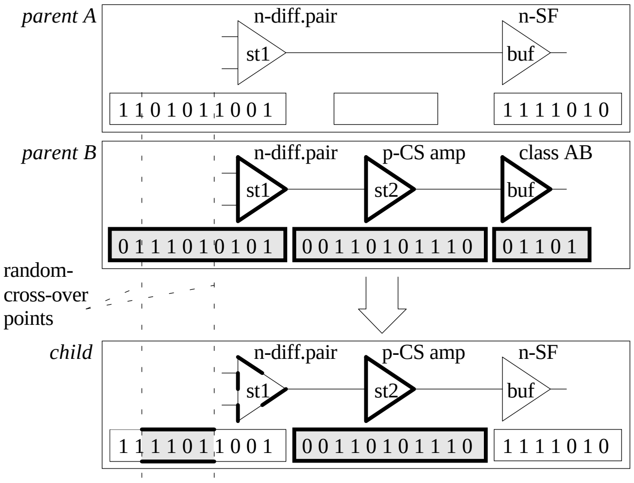

Fig. 3. Cross-over operation

this cross-over operation is performed on each of the three pairs of α-vectors.

The last modification process is mutation. This process changes the value of a small number of the entities of each α-vector. Each entity (a ‘0’ or a ‘1’ in Fig. 3) has a chance equal to the mutation rate to be changed in value.

It is very important here to notice that because we apply the genetic operators ‘cross-over’ and ‘mutation’ on the α-vectors, we always generate correct opamps in terms of the design constraints. Therefore, we do not have to verify whether a certain mutation is legal or not, this in contrast to genetic algorithms as in [11]. When we would perform the genetic operators directly on the circuit parameters (transistor widths and lengths), meaningless circuits can be generated, for example opamps of which the input transistors are out of saturation.

This evolution loop continues until a stop criterion has reached. In DARWIN, we use the number of generations as the stop criterion.

IV. EXPERIMENTAL RESULTS

To illustrate the synthesis process and to verify that DARWIN can synthesize functional opamps, we will discuss several circuits that were generated by DARWIN. For all opamps, we used the MIETEC 2.4 µm n-well CMOS process parameter set. The three design examples (A, B, and C) differed in the required specification for the DC gain. The optimization function was defined by the power dissipation. Table I depicts the performance specifications, the obtained results as simulated with the build in simulator and the SPICE simulation results. The component area in table I is the size of the transistors in addition to the size of the miller capacitance but without wiring and component spacing. For the SPICE simulations in table I, we used the ‘typical’ transistor parameters for examples A and C and the ‘slow’, ‘typical’ and ‘fast’ transistor parameters for example B. The sized schematics of the design example A, B, and C are depicted in Fig. 4, 5 and 6 respectively. The paracitic capacitor in Fig. 5 and 6 represent the paracitic capacitance due to the miller capacitor.

As can be seen in table I, the synthesized opamps meet all specifications according to the estimations of DARWIN and most of the specifications according to the SPICE simulations. The differences between the results as predicted by DARWIN and the SPICE simulation results are mainly due to the simplified transistor models that are used by DARWIN. The open loop transfer function of example B, as derived by DARWIN and by SPICE, is plotted in Fig. 7. In Fig. 7, ‘slow’, ‘typical’ and ‘fast’ refer to SPICE simulations, while ‘darwin’ refers to estimated transfer function by DARWIN. The estimations are good enough to be useful in DARWIN, while at the same time the evaluation of a single opamp only requires in the order of tens of milliseconds on a HP 700 workstation. In this way it is possible to evaluate an entire population of hundred opamps in only a few seconds, resulting in a complete opamp synthesis in a few minutes. When we should have used SPICE as the simulator for the intermediate evaluations, a small improvement in accuracy would have been achieved at the expense of an enormous increase in computation time.

We used the genetic algorithm with a population size of 100 opamps, a maximum of 150 generations, a cross-over rate of 0.2, and a mutation rate of 0.01. Due to the genetic algorithm, the inintial populations evolved to populations in which all opamps were of the same topology. This ‘topology selection’ can be seen in the area charts of Fig. 8, where the topology distribution is depicted during the first tens of generations of the synthesis process. The abbreviations that are used in the legend of Fig. 8 are explained as follows: 1st = one gain stage, 2st = two gain stages, fc = folded cascode, sf = source follower, AB = class AB buffer, n = nMOS input transistor, p = pMOS input transistor.

In design example A, the nMOS differential pair (1st_n, area marked with the number 1) dominated the population. This is not surprisingly, since the pMOS differential pair is

TABLE I.
REQUIRED SPECIFICATIONS, DARWIN ESTIMATIONS, AND SPICE SIMULATION RESULTS

Parameter	example A (60 dB)			example B (80 dB)					example C (100 dB)		
	Spec	darwin	SPICE (typ.)	Spec	darwin	SPICE (slow)	SPICE (typ.)	SPICE (fast)	Spec	darwin	SPICE (typ.)
DC-gain [dB]	≥ 60	60	54	≥ 80	93	75	87	96	≥ 100	144	133
gain bandwidth [MHz]	≥ 3	4.8	4.3	≥ 3	3.0	4.4	4.7	3.7	≥ 3	3.1	3.7
unity gain freq. [MHz]	≥ 3	3.8	5.3	≥ 3	3.2	10.1	7.4	6.8	≥ 3	3.1	6.8
phase margin [°]	≥ 60	72	74	≥ 60	72	30	63	67	≥ 60	60	47
slew rate [V/µs]	≥ 3	3.2	-4.0 +4.0	≥ 3	3.9	-7.9 +2.9	-6.7 +2.1	-5.6 +1.4	≥ 3	7.7	-7.9 +4.6
load capacitance [pF]	2	2	2	2	2	2	2	2	2	2	2
supply voltage [V]	± 2.5	± 2.5	± 2.5	± 2.5	± 2.5	± 2.5	± 2.5	± 2.5	± 2.5	± 2.5	± 2.5
input voltage range [V]	± 1	-1.3 +1.9	-1.2 +2.5	± 1	-1.8 +1.0	-2.2 +0.7	-2.2 +1.0	-2.2 +1.3	± 1	-1.0 +1.8	-1.0 +2.5
output voltage range [V]	± 1	-1.9 +2.1	-1.3 +2.2	± 1	-2.2 +2.0	-2.4 +2.4	-2.4 +2.4	-2.4 +2.4	± 1	-2.2 +2.0	-2.4 +2.4
power dissip. [µW]	low	31	52	low	89	157	124	96	low	251	308
component area [µm^2]	-	6500	-	-	3600	-	-	-	-	8100	-
technology	MIETEC 2.4 µm n-well CMOS										

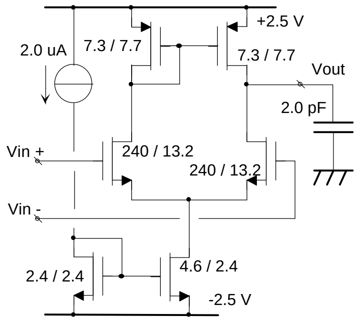

Fig. 4. Sized circuit schematic of example A

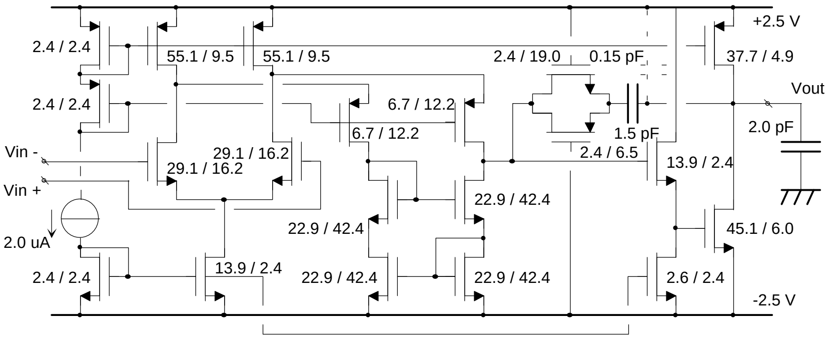

Fig. 6. Sized circuit schematic of example C

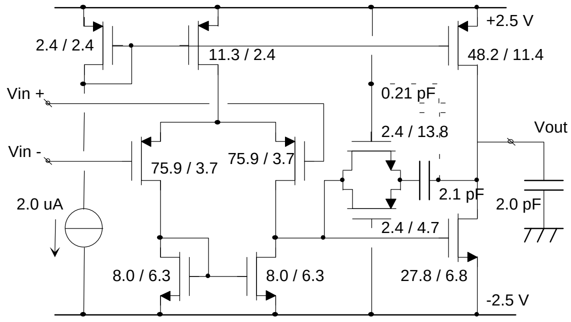

Fig. 5. Sized opamp schematic of example B

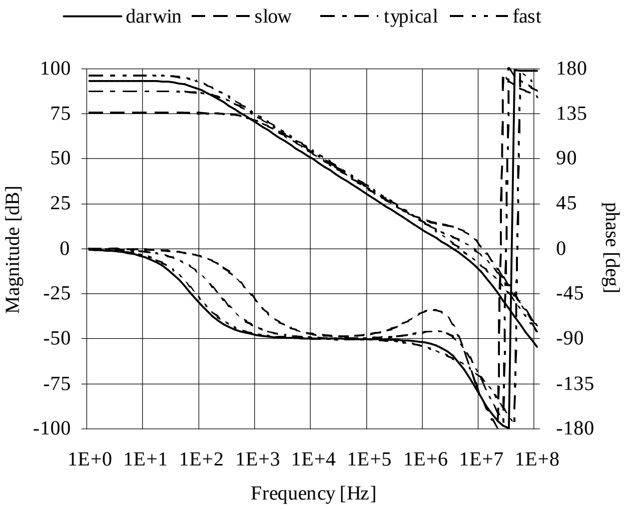

Fig 7 Bode plot of the open loop transfer function of example B

not able to have a DC-gain of 60 dB (in combination with the other specifications), and all other topologies have more stages and are therefore likely to dissipate more power.

In design example B, the required DC-gain was increased to 80 dB. All one-stage opamps died out in the first 20 generations, but several different topologies seemed to be good candidates after 20 generations. From that moment on, the two-stage opamp with pMOS input transistors (2st_n, area marked with the number 2) appeared to be slightly fitter than the other remaining topologies and the number of opamps with that specific topology increased until they dominated the population. Again the final choice of the topology is logical, since the DC-gain can not be achieved by a single differential pair, while at the other hand a folded cascode topology is not really required. Another interesting event in design example B is the fact that some topologies disappeared out of the population and reappeared some generations later. This happened for example with the topology, marked with the number 6. The fact that a new opamp can have a different topology than its two parents makes the algorithm robust, since a died-out topology can get a second chance to prove its suitability.

In design example C, the required DC-gain was further increased to 100 dB. An opamp with such a set of specification requirements can only be achieved by using a cascoded gain stage. It is therefore logical that the opamps with two gain stages, of which the first is a nMOS folded cascode stage, appeared to be more suitable than the other opamps. The areas in Fig. 8, marked with the numbers 3, 4, 5, and 6 all had a nMOS folded cascode input stage followed by a second gain stage. They only differed in their output buffer. of those four topologies, the version without a buffer became the eventual winner.

Simultaneously with this topology selection, the transistor sizes change as well during the synthesis process. The genetic algorithm will increase the maximal as well as the average fitness of the population during the synthesis process, indicating that the transistor sizes are adjusted in a way that the opamps in the population become more suitable to meet the required specifications.

The three design examples show that DARWIN is capable to select suitable topologies and to size the transistor sizes, without much design knowledge.

V. Conclusions

We described a way to synthesize analog circuits, based on genetic algorithms. An initial population of circuits evolves to a population in which the circuits are adapted to the required performance specifications. During each generation, all circuits are evaluated by means of rough simulations. On the basis of this evaluation, a fitness value is assigned to

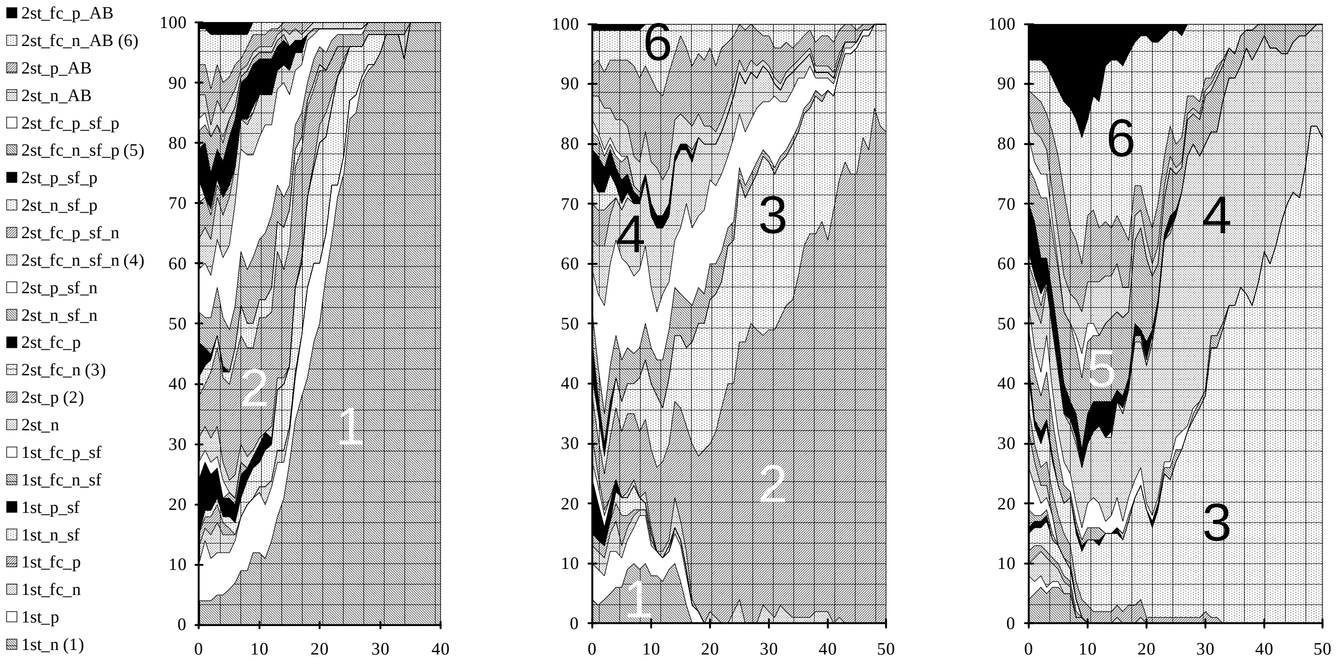

Fig. 8. Topology distribution versus generation, for design example A (left), B (centre), and C (right)

each of the circuits. Some of the ill-suited circuits are removed out of the population, while new circuits are generated out of well-suited circuits. Furthermore, the circuits are slightly changed by means of a mutation operation. The genetic algorithm does not require much design knowledge, but is nevertheless able to synthesize suitable circuits for a wide range of performance specifications.

The circuit representation that is used in the genetic algorithm allows us to put restrictions to the circuits' building blocks. In this way, it is ensured that all generated circuits posses certain qualities, required for a correct functioning of the circuit. A prototype tool, DARWIN, which is based on the proposed algorithm, can synthesize CMOS opamps from performance specifications and technology parameters. Several divergent design examples, which are verified by means of SPICE simulations, indicate that the program is capable of synthesizing the desired circuits.

Particularly in the beginning of the synthesis process, when a topology is selected and the transistor dimensions are roughly sized, the genetic algorithm showed to be very useful. A large number of different topologies can be explored, without the need of time-consuming transistor sizing algorithms. In the final stage of the opamp synthesis, gradient based local search algorithms will probably be more efficient in optimizing the transistor dimensions, although it is possible with genetic algorithms. We believe that the proposed approach is specially suited for the synthesis of circuits or systems that can have many different types of topologies and for which it is difficult to capture the topology selection knowledge in rules or equations.

References

1. R. A. Ruthenbar, 'Analog design automation: Where are we? Where are we going?', *Proc. 15 th IEEE CICC*, San Diego, CA, 1993, IEEE, New York, pp. 13.1.1–13.1.8 (1993).
2. M. Degrauwe, et al., 'IDAC: An interactive design tool for analog integrated circuits', *IEEE J. Solid-State Circuits*, **SC-22**, 1106–1116 (1987).
3. R. Harjani, R. A. Rutenbar, and L. R. Carley, 'OASYS: A framework for analog circuit synthesis', *IEEE Trans. on Computer Aided Design*, **CAD-8**, 1247–1266 (1989).
4. H. Y. Koh, C. H. Sequin, and P. R. Gray, 'OPASYN: A compiler for MOS operational amplifiers', *IEEE Trans. on Computer Aided Design*, **CAD-9**, 113–125 (1990).
5. Z. Ning, M. Kole, T. Mouthaan, and H. Wallinga, 'Analog circuit design automation for performance', *Proc. 14 th IEEE CICC*, Boston, MA, 1992, IEEE, New York, pp. 8.2.1–8.2.4 (1992).
6. P. C. Maulik, L. R. Carley, and R. A. Rutenbar, 'A mixed-integer nonlinear programming approach to analog circuit synthesis', *Proc. 29 th ACM / IEEE DAC*, Anaheim, CA, 1992, IEEE, Los Alamitos, pp. 698–703 (1992).
7. Z. Michalewicz, *Genetic algorithms + data structures = evolution programs*, Springer Verlag, New York, 1992.
8. D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, New York, 1989.
9. D. M. W. Leenaerts, 'Applications of interval analysis to circuit design', *IEEE Trans. on Circuits and Systems*, **CAS-37**, 803–807 (1990).
10. D. M. W Leenaerts, and J. A. Hegt, 'Finding all solutions of piecewise linear functions and the application to circuit design', *Int. J. Cir. Theor. Appl.*, **19**, 107–123 (1991).
11. Z. Michalewicz, and C. Z. Janikow, 'Handling constraints in genetic algorithms', *Proc. 4 th int. conf. on Genetic Algorithms*, San Diego, CA, 1991, Morgan Kaufmann publishers, San Mateo, 1991, pp. 151–157 (1991).