Topology Synthesis of Analog Circuits Based on Adaptively Generated Building Blocks *

Angan Das and Ranga Vemuri
Department of Electrical and Computer Engineering
University of Cincinnati
Cincinnati, Ohio 45221-0030, USA.
{dasan, ranga} @ececs.uc.edu

ABSTRACT

This paper presents an automated analog synthesis tool for topology generation and subsequent circuit sizing. Though sizing is indispensable, the paper mainly concentrates on topology generation. A new kind of GA is developed, where a fraction of the offsprings in each generation is built from building blocks or cells obtained from previous generations. The cells are stored in a hierarchically arranged library that also contains information on the preferred neighborhood of each cell. The adaptively formed cell library starts only with basic elements and gradually includes functionally useful and bigger blocks, pertinent to the design. The techniques have been applied to synthesize an operational amplifier and a ring oscillator design. Results show that with reasonable computational effort, topologies have evolved that are designer understandable.

Categories and Subject Descriptors

J.6 [**COMPUTER-AIDED ENGINEERING**]: Computer-aided design (CAD).

General Terms

Algorithms, Design.

Keywords

Automated design, topology generation, genetic algorithm.

1. INTRODUCTION

In today's world, with ever increasing design complexity and constantly shrinking device sizes, the microelectronics industry faces the need to develop an entire system on a single chip (SoC) [11]. Although efficient CAD techniques exist for handling digital sections on a SoC, automated synthesis tools for the irreplaceable analog sections are still immature and incomplete. Circuit-level analog synthesis comprises of two steps — Topology formation and Sizing of the topology [11]. Though sizing has attained considerable success in both industry as well as academia, the former is still not so fortunate in this regard. Topology selection and topology generation are two approaches, though entirely different, to topology formation. Research in topology selection [4, 6, 8] has almost been discontinued owing to its heavy designer-dependency and huge set up effort, specific to each different kind of application.

On the other hand, with the advent of evolutionary techniques like Genetic Algorithm (GA) and Genetic Programming (GP), topology generation gained popularity. *Evolutionary synthesis* techniques were introduced in [5, 7]. But they required substantial computational overhead and produced functionally correct but *unfamiliar* circuits. To overcome this limitation, Dastidar et al. [2] and Wang et al. [12] blended some design knowledge, although minimal, with the evolutionary aspects. They used a fixed set of user-defined building blocks and connectivity rules to evolve analog circuits.

Unfortunately, in [2, 12], the building block library used is specific to the class of circuits designed. Hence it needs to be rebuilt for each different kind of design. In one of our earlier works, ATLAS [1], we have used adaptively generated building blocks to eliminate this drawback. But in ATLAS, unlike good design practice, these blocks are placed randomly with respect to each other while generating a circuit. Moreover, the tool makes and breaks solutions in every generation. This approach does produce good building blocks, but not always good circuits. In summary, ATLAS somewhat lacks the advantages of a typical GA or GP.

In an attempt to alleviate all of the above drawbacks, we introduce a GA-based framework for topology synthesis of analog circuits. It incorporates the advantages of both ATLAS and that of a conventional GA/GP. The primary contributions of this work are:

- Circuit topologies are generated through well defined rules, with the help of building blocks contained in a hierarchically arranged block (also termed as cell) library.
- The library initially consists of basic elements only. It gradually extends to include bigger and functionally useful cells, specific to the design, as the synthesis progresses.
- Each library block also contains a list of its neighboring blocks. During circuit formation, *good* blocks and their *good* neighbor blocks are selected, adjacently placed, and connected.
- A GA is developed, where reproduction occurs in two ways. One fraction of the solutions is produced conventionally, while the rest is produced afresh from the better library blocks.
- A library generated out of a synthesis run can be reused in the future to speed up synthesis for the same class of designs.
- The tool generates analog circuit topologies that match handcrafted designs, and are thereby designer understandable.

The rest of the paper is organized into the following sections. Section 2 introduces the synthesis methodology. Section 3 describes the GA reproduction mechanisms. Section 4 deals with the adaptive library formation procedure. Section 5 discusses the SA-based sizer and associated fitness evaluation. Section 6 demonstrates the application of our techniques to synthesize an opamp and a ring oscillator design. Section 7 finally concludes the work.

*This work was supported by National Science Foundation under award numbers CCF-0429717 and CNS-0421092.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC 2008, June 8–13, 2008, Anaheim, California, USA
Copyright 2008 ACM 978-1-60558-115-6/08/0006…5.00

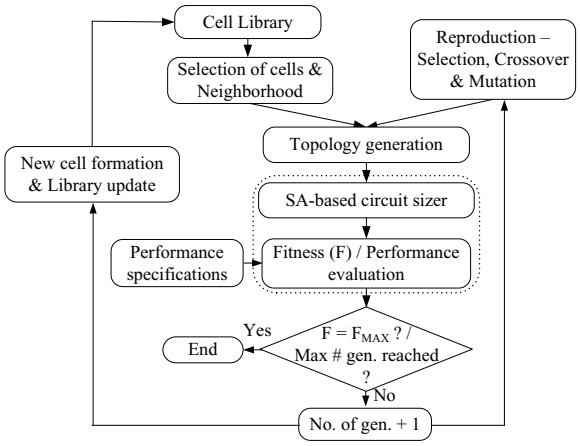


Fig. 1: Synthesis flow

2. SYNTHESIS METHODOLOGY

2.1 Synthesis Flow

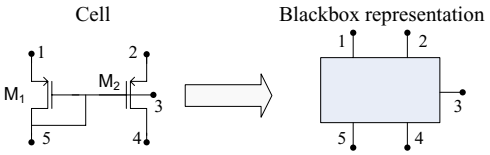
Fig. 1 gives the overall flow of the GA-based synthesis framework. As in a GA, one generation of solutions (chromosomes) gives birth to the next generation. Initially, the various GA run parameters and design specifications are provided by the user. A hierarchical building block or cell library is maintained throughout the synthesis process. The library initially contains basic elements like PMOS, NMOS and current sources only. It gradually includes bigger and more design-pertinent cells as generations progress. A topology is formed by connecting these cells, based on some structural constraints. The topology is subsequently sized with a SA-based sizer and the performance is evaluated. A new generation is formed and the procedure continues till the desired fitness is achieved, for the allowable number of generations. In this respect, here, more and more solutions need to be generated only after the library is adequately formed. A better formed library always provides better cell choices. So we gradually increase the number of chromosomes or solutions with the advancement of generations.

After each generation, the cell library is updated with necessary information acquired from the present set of circuits. Thereby, present cells and their neighboring cells are ranked in the form of a roulette wheel, and new cells are added. From the second generation onwards, two parents reproduce through selection, crossover and mutation to give rise to two new children. These constitute a certain fraction of offsprings. The remaining offsprings are produced afresh (like the first generation) by selecting and joining better library cells. But this time onwards, the library being extended with new cells, the chosen cells are not limited to simple PMOS and NMOS only. Moreover, the positioning of cells with respect to each other i.e. cell neighborhoods, are also no more random. Fitter adjacent cells for each chosen cell are selected from the library along with the original cell and are interconnected appropriately.

2.2 Building Block: Blackbox Approach

In this work, a block or cell represents a moderately sized circuit level netlist of basic elements like NMOS, PMOS and current sources. The cell is considered as a blackbox as shown in Fig. 2, with the following terminal information —

- *Number of terminals*.
- *Terminal signal type:* The signal is of 3 types: voltage only (like gate of a transistor); current only (like current source terminals); and mixed (like drain or source of a transistor).



Terminal_no	Type	Direction
1	Mixed	In
2	Mixed	In
3	Voltage	-
4	Mixed	Out
5	Mixed	Out

Fig. 2: Blackbox representation of an example cell

- *Terminal signal direction:* For current only and mixed terminals, the three directions of current are 'in' (like PMOS source), 'out' (like PMOS drain), and 'in + out' (like junction of PMOS and NMOS drains).

The blackbox also stores information on whether the cell contains input and output (i/o) nodes, and power connections (Vdd, Gnd). The former helps us to soon identify a potential set of circuit i/o gateway configurations, while the latter aids us to avoid connecting terminals to Vdd/Gnd forcefully during circuit formation.

2.3 Hierarchical Cell Library

The building blocks or cells are stored in a hierarchically arranged library. The term hierarchy pertains to the pattern of arrangement of cells within the library. The levels are defined based on the presence (1) or absence (0) of the aspects — input node(s), output node(s) and power rails (Vdd, Gnd, etc.) in a cell, apart from the cell elements. The different levels, as shown in Fig. 3, are:

1. *Level_1:* The top-level library is divided on the basis of various combinations of i/o nodes present. For e.g., for a 2-input 1-output design, there will be $2^3 = 8$ level_1 libraries.
2. *Level_2:* The level_1 library is further subdivided based on the power rail content. For e.g., for only Vdd and Gnd rails, there will be $2^2 = 4$ level_2 libraries for each level_1 library. All level_1 and level_2 libraries also contain fitness-based ranked lists (roulette maps) of the cells contained within them.
3. *Level_3:* Each level_2 library in turn has variable number of sub-libraries, depending on the combination of basic elements, like PMOS and NMOS, present within a cell.
4. *Level_4:* The lowest level contains the cell elements and their terminal information. Each cell also contains an occurrence count and a fitness, signifying the number of times it is used in circuit formation and its appropriateness for the same. Also, at this level, each cell contains a ranked list of its own adjacent or neighboring cells.

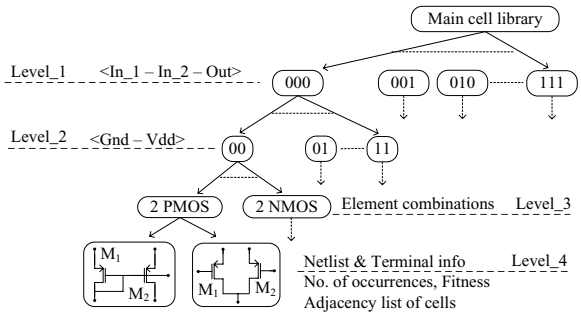


Fig. 3: Hierarchical cell library (shown for a 2-inp 1-out design)

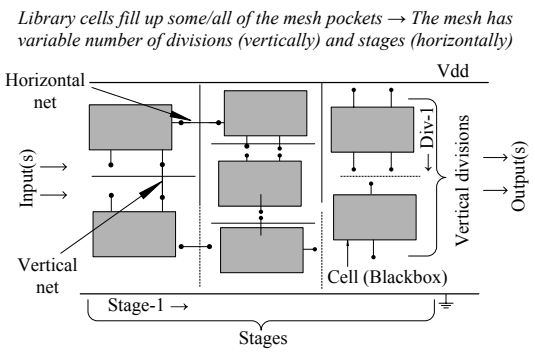


Fig. 4: Extendible mesh structure for placing cells

This hierarchical nature of the library provides some distinct advantages. First, during circuit formation, it helps us in choosing cells as per power rail and i/o requirements. We decide on a level and eventually select a cell from the roulette map at that level. Secondly, once a cell is chosen and placed, its neighborhood may be populated with cells from its own adjacency list within the library. Finally, the hierarchy helps in verifying the presence or absence of a cell (say test cell) during library extension, to avoid its redundant inclusion. Required flag variables for levels 1-3 are generated for the test cell and we search up to level_3 (if it exists) trivially. If found, then only a level_4 exhaustive search is executed.

2.4 Circuit Formation

Circuit formation is an important part of the synthesis framework. In this aspect, we follow a procedure similar to [1], with minor changes regarding the choice of cells from the library. Thus the topology is generated through the following steps —
1) Mesh structure for placing cells: As shown in Fig. 4, a very simple mesh structure consisting of successive stages, with each stage having vertical divisions, serves as the backbone for the circuit. Stages are interconnected through horizontal nets while divisions through vertical nets. The mesh dimension is user-defined. Cells fill up some or all of the mesh pockets during circuit formation.
2) Combination sequence of i/o containing cells: A one time tree is formed containing all the possible i/o containing sequences. For eg., for 2-input (say In1 and In2) 1-output (say Out) design, the various paths of the tree are (In1)-(In2)-(Out), (In1+Out)-(In2), etc.
3) Placing i/o cells: For the i/o sequence selected, the input cells fill up the pockets starting with stage-1 forwards, and output cells vice-versa. Regarding vertical positioning, Vdd cells are placed starting from the upper division downwards, and Gnd ones vice-versa.
4) Placing other cells: The remaining empty pockets are filled with non-i/o cells. There is no preference for horizontal positioning, but the vertical positioning follows the same guideline as for i/o cells.
5) Choices of cell selection from library: A cell can be selected either from the library in general i.e. level_1 and level_2 maps or from the adjacency list of already placed cells. In the initial generations, we do more of the former and gradually shift to the latter when the library has enough information on desired neighborhoods.
6) Cell terminal connections: The cell terminals are interconnected amongst themselves in ways governed by terminal types. Voltage terminals are joined either to mixed or to voltage terminals of previous stages. Current terminals are connected to mixed terminals of other cells in same stage, depending on the direction and availability. Mixed terminals are connected to current or to mixed terminals in the same stage. These nodes are also preserved for connection to voltage nodes of succeeding stages. In all cases, however, unconnected terminals get a preference over the already utilized ones.

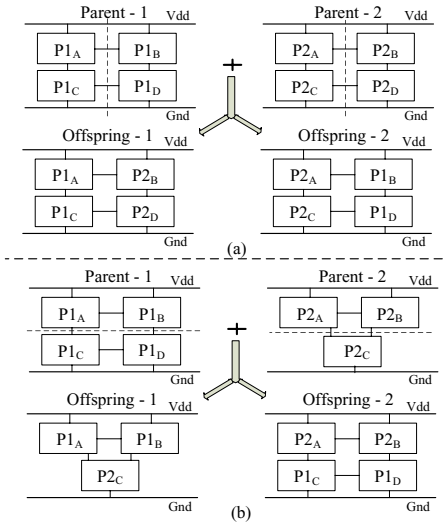


Fig. 5: Crossover - (a) Horizontal crossover swaps stages; (b) Vertical crossover swaps vertical divisions

7) Treatment of floating terminals: In case of floating terminals, voltage terminals are biased with an external V_{source}. For current terminals, the lone current source is deleted. Mixed terminals with only 'in' or 'out' are connected to appropriate power rails. 'In + out' ones are undisturbed, since they are technically non-floating.
8) Checking stage inter-connections: Despite being simulatable, if there is no path from input to output, we simply discard the circuit.

3. REPRODUCTION MECHANISMS

3.1 Crossover

From the second generation onwards, a certain number of new circuits are built through mating of parent circuits, obtained from the previous generation. The parents are selected as per the roulette wheel strategy [3], and are subsequently subjected to crossover. Crossover involves exchange of certain circuit portions or subcircuits between the two participating parent circuits to produce two new offspring circuits, as shown in Fig. 5. In this respect, the two types of crossover mechanisms, applied with equal probability, are:

- *Horizontal crossover:* Here, the subcircuit is a randomly chosen complete stage or group of adjacent stages of the parent circuit. As shown in Fig. 5(a), stage-2 of of parent-1 consisting of blocks $P1_B$ and $P1_D$ is swapped with stage-2 of parent-2 having blocks $P2_B$ and $P2_D$, to produce two offsprings. This process is analogous to one-point crossover.
- *Vertical crossover:* Likewise, here the subcircuit comprises of all the blocks of all stages contained in any randomly chosen vertical division or a group of adjacent divisions of the parent circuit. Fig. 5(b) clearly demonstrates the process.

In both the above mechanisms, if the number of terminals differ for the two subcircuits swapped, then we connect the floating terminals in ways as already discussed above.

3.2 Mutation

Mutation introduces diversity. The mutation mechanism adopted herein comprises of — adding a cell; deleting a cell; replacing a cell with another library cell; and swapping two cells either between two vertical divisions, or between two horizontal stages. All of these mechanisms are applied with equal probability. Alike crossover, floating terminals are adjusted accordingly.

4. ADAPTIVE FORMATION OF LIBRARY

A certain number of offsprings in each generation are produced from the *better* library cells. The betterness of a cell is judged as per its appropriateness to the design under consideration. This requires continuous evaluation of the existing cells and addition of new cells, as per the required design, i.e. adaptive update of the library. Adaptivity emerges from the fact that the fitness of the present circuit decides on how well its building blocks and their placement qualify to be prospective cells for future generations of circuits. The procedure is shown in Alg. 1 and is explained below:

4.1 Impact of Subcircuit

Here, we assume that the appropriateness or merit of subcircuit has an impact on the total circuit performance. This is especially true for analog circuits [11]. This impact is quantitatively analogous to the fitness of the subcircuit. Therefore, when a parent produces an offspring, the change in fitness from parent to offspring owes to the difference of two factors — a) Impact of the outgoing subcircuit and b) Impact of the incoming subcircuit. For e.g., in Fig. 5(a), the fitness difference between offspring-1 and parent-1 is raised due to incoming $P2_B$ and $P2_D$, and lowered owing to outgoing $P1_B$ and $P1_D$. In this way, the impact or fitness of all the subcircuits (F_{subckt}) are computed after each crossover operation.

4.2 Division of Fitness

The process of manually forming a topology in analog design comprises of two sequential steps, viz.

- Selection of correct building blocks.
- Placing the blocks appropriately with respect to each other and interconnecting them.

Hence the fitness of a subcircuit containing some cells is proportional to two factors — what are the cells present, and how are these cells placed. Now, in this CAD framework, since we are unaware of all the different design heuristics and equations, so we assume flat proportionality constants for the above factors. Thereby, the total subcircuit impact or fitness (F_{subckt}) is divided into two parts — one owing to presence of blocks (F_{presence}) and the other owing to their adjacency ($F_{\text{adjacency}}$). Quantitatively,

$$F_{\text{subckt}} = F_{\text{presence}} + F_{\text{adjacency}}$$

$$F_{\text{presence}} = \alpha * F_{\text{subckt}}, \;\; F_{\text{adjacency}} = \beta * F_{\text{subckt}}; \;\; \text{where } \alpha + \beta = 1$$

4.3 Update of Existing-Cell Parameters

The fitness F_{presence} is assumed to be equally contributed by all the participating cells. The required updates for each such cell are:

- *No. of occurrences (N_{cell}):* The cell's occurrence count in the main library increments by unity, each time the cell is used.
- *Gross Fitness ($Gross\_F_{cell}$):* The average gross fitness of a cell denotes the appropriateness of the cell for the design. This quantity is updated through the cell's equal share of the corresponding differential fitness.
- *Net Fitness (Net_F_{cell}):* For ranking candidates, we use net fitness. This measure gives weightage to both $Gross\_F_{\text{cell}}$ and N_{cell}, with γ and δ being the proportionality constants.
- *Adjacency list of cells:* The adjacency list for each cell contains three parameters for each adjacent cell. They are — the number of times they have been placed adjacently, connectivity information, and adjacency fitness measure. The count and fitness for each connectivity gets updated as per the same governing equations given above for a cell. But this time the contribution comes from $F_{\text{adjacency}}$, being equally divided among the adjacent cells of the cell under consideration.

Algorithm 1: Cell library update for each reproduction

Input: Gen-(n) cell library, gen-(n) parents and offsprings
Output: Updated cell library: For gen-(n+1) formation
procedure Cell library update
Update of existing-cell parameters
forall *subcircuit portions* $\in$ *parents* P_1, P_2 **do**
  F_{subckt} = `corr_fitness_diff` $(P_i, O_i); i \in \{1, 2\}$
  forall *cells* $\in$ *the subckt* **do**
    Presence based fitness
    $F_{\text{cell}} = F_{\text{presence}}/X$; // $X \rightarrow$ No. of cells in subckt
    $Gross\_F_{\text{cell}} = \frac{Gross\_F_{\text{cell}} * N_{\text{cell}} + F_{\text{cell}}}{N_{\text{cell}}+1}$;
    $N_{\text{cell}} = N_{\text{cell}} + 1$;
    $Net\_F_{\text{cell}} = \gamma * Gross\_F_{\text{cell}} + \delta * N_{\text{cell}}$; // $\gamma + \delta = 1$
    Adjacency based fitness
    $F_{\text{adjacent cell}} = F_{\text{adjacency}}$ / (No. of adj. cells for the cell);
    Calculate $Gross\_F_{\text{adjacent cell}}$ and $Net\_F_{\text{adjacent cell}}$ in the same way as that for the cell;
  end
end
Addition of new cells
Extract new cells out of the two offsprings;
$F_{\text{new cell}} = F_{\text{offspring - parent}} * \eta *$ gen_no/ (No. of new cells);
Rank all cells and their adjacent cells as per net fitness values;

4.4 Formation of New Cells

New cells are extracted out of a circuit formed and they serve to extend the library. To this end, the connections surrounding each node of each element are evaluated, element combinations are formed, and the combination qualifies to represent a new cell if certain conditions are satisfied [1]. These conditions prevent the inclusion of otherwise functionally meaningless element combinations as cells. Subsequently, it is checked to ensure that the cell is not already present in the library. If absent, it is appended to it with a generation proportionate fitness (scaling factor - η). This step ensures that as the generations progress, new cells get included with appreciable fitness. Adjacency lists are also extended in a similar fashion. Fig. 6 shows how two simple cells in gen-1 gradually combine with other cells to produce bigger and functionally useful new cells. For eg., Fig. 6(a) shows an NMOS producing a differential pair, which further produces that with a constant current biasing.

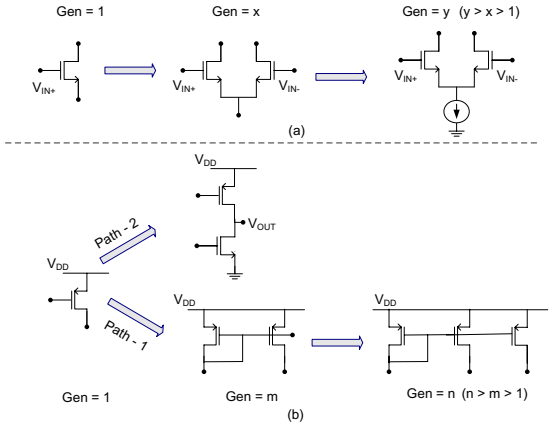


Fig. 6: Results showing (a) an NMOS and (b) a PMOS gradually giving rise to bigger and functionally meaningful new cells

5. SIZER AND FITNESS EVALUATION

5.1 Sizing of the Topology

The merit of a topology can never be estimated before sizing it properly. The circuit sizer used in this work is based on the simulated annealing (SA) algorithm. Further details could not be furnished owing to the size constraints of the paper. Despite several other sizing approaches including recent ones [9] in place, the SA is implemented primarily because it is appreciably fast in sizing hundreds of unknown circuits; it could be executed in the batch mode; and it requires minimal set-up effort, only in the form of tuning. The inputs to the sizer comprise of the range and granularity of the design variables like dimensions of transistors, voltage source values and current source values. The circuit performance parameters are measured through HSPICE simulations. The related cost function for fitness evaluation is described below in section 5.2.

5.2 Fitness Evaluation

Fitness signifies the quality of a solution. Now, most of the analog circuit design problems are multi-objective optimization problems. To obtain the globally optimal solution to this problem, we adopt the widely popular normalized weighted sum approach for multi-criteria optimization. The weighted differences between the obtained values and expected values for each criterion are summed up and the result is normalized to a value between zero and unity. All performance specifications are denoted by four values (increasing in the given order) viz. absolute minimum (min_p), lower limit ($lower\_p$), upper limit ($upper\_p$) and absolute maximum (max_p). Quantitatively, the normalized error function ε for objective i in a design involving P objectives is given by:

$$\varepsilon_i = \begin{cases} \frac{lower\_p_i - obt\_p_i}{lower\_p_i - min\_p_i} & \text{when } obt\_p_i < lower\_p_i, \\ \frac{obt\_p_i - upper\_p_i}{max\_p_i - upper\_p_i} & \text{when } obt\_p_i > upper\_p_i, \\ 0 & \text{when } lower\_p_i <= obt\_p_i <= upper\_p_i. \end{cases}$$

The total deviation (E) and the fitness function (F) are given by:

$$F = \frac{1}{1+E}, \text{ where } E = \sum_{i=1}^{P} \omega_i * \varepsilon_i \text{ and } \sum_{i=1}^{P} \omega_i = 100\%$$

6. EXPERIMENTS AND RESULTS

The synthesis tool has been coded in C++. The program was run on a Sun Workstation having two 750 MHz UltraSPARC-3 processors and 2GB RAM, with Solaris 10 OS. The technology used is 0.18μ, though it is capable of running equally well on all technologies. The various proportionality constants used are $\alpha = 0.75, \beta = 0.25, \gamma = 0.95, \delta = 0.05$, and $\eta = 0.05$. The range of variables as input to the sizer comprises of (in format - min:granularity:max)

Table 1: Design Specifications and Obtained values

PARAMETERS	ω (%)	Specifications	Obtained values
Operational amplifier			
DC Gain	40	$\geq$ 60 dB	60.28 dB
Bandwidth	30	$\geq$ 700 MHz	741.7 MHz
3dB frequency	30	$\geq$ 2.5 MHz	2.53 MHz
Slew rate	–	–	8.19 V/μs
3-stage ring oscillator			
Oscillation frequency	70	$\geq$ 2.0 GHz	2.1 GHz
Amplitude (Vdd = 2.5V)	30	$\geq$ 1.75V	1.82V

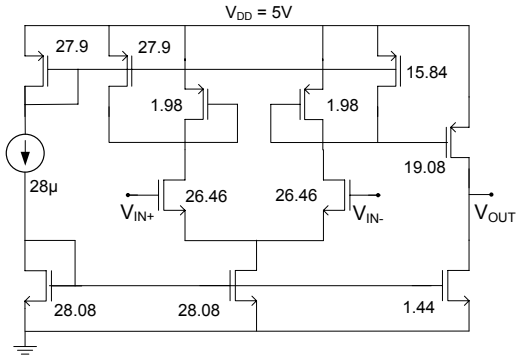


Fig. 7: Operational amplifier circuit generated for design experiment-1 (All transistor widths are in μ, L = 0.36μ)

W_tran - 0.36:0.18:36μ; V_{source} - 0.5:0.1:5.0V; and I_{source} - 0.1:0.1:40μA, the symbols carrying their usual meanings. L_tran = 0.36μ is fixed for sizing convenience. In this work, the number of solutions should increase progressively as discussed earlier. So we start with 100 chromosomes, that is increased by 10 after each set of 10 generations. Also, 20% of the circuits in each generation are formed using library cells and rest 80% through GA reproduction.

Two different design benchmarks, an operational amplifier and a three stage ring oscillator, were synthesized using the framework. The design specifications and the relative weightage given to each objective are provided in Table 1. These parameters are decided after carefully considering several such hand designed circuits [10]. For the opamp, slew rate is not used for evolution purposes. It is only a performance metric reported for the obtained design. For simulation purposes, HSPICE is used owing to three advantages. First, it gives maximum accuracy. Second, unlike symbolic methods, it does not require any setup overhead. Finally, this helps us to integrate the tool into any existing industrial framework.

6.1 Design I: Operational Amplifier

An operational amplifier or opamp is a very commonly used analog circuit having a wide range of applications comprising of comparators, filters, pre-amplification stages, signal converters and so on [10]. The opamp designed is a single-ended opamp. A load capacitance of 5pF is provided at the output node for slew rate performance. During the synthesis run, the tool soon started evolving formidable designs from the 100th generation onwards. However, the totally compliant design was obtained in the 152nd generation and is shown in Fig. 7. Table 1 shows that all the specifications were satisfied by the generated design.

A careful observation of Fig. 7 shows that the circuit actually

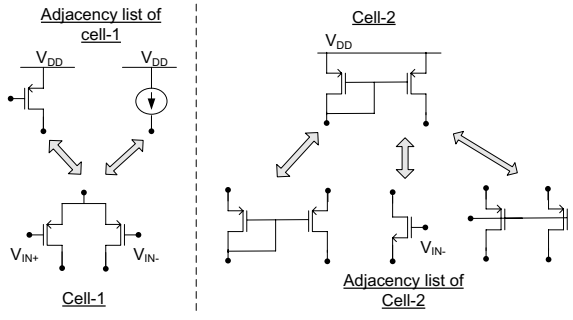


Fig. 8: Preferred adjacent cells of some highly-ranked cells

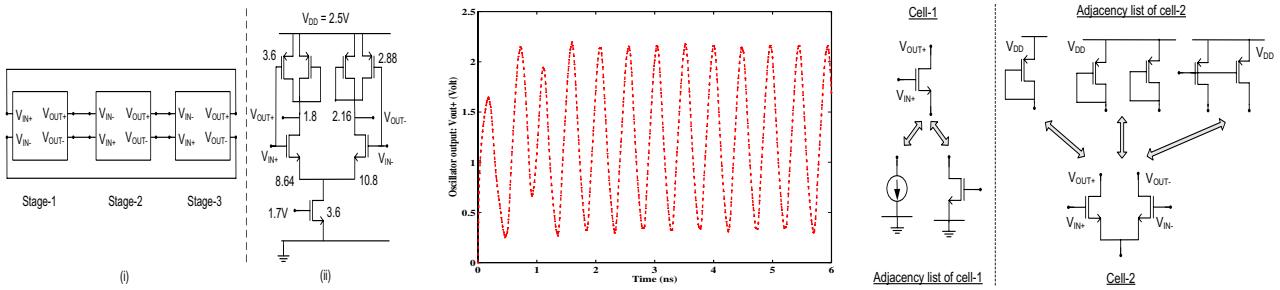


Fig. 9: Three stage ring oscillator design → (a) (i) Prototype circuit (ii) Synthesized circuit for a single stage (All transistor widths are in μ, L = 0.36μ); (b) Output waveform; (c) Some of the fitter library cells and their adjacent cells

consists of building blocks easily identifiable by the analog designer. It includes a differential pair, current mirror and diode connected load among others. This helps in perceiving the operation of the circuit. Out of the 105 new cells formed, some of the fittest cells and their adjacent cells are shown in Fig. 8. Fig. 8 clearly shows that the adjacency list together with the cells consists of well-known and understandable blocks that range from differential pair with current source, to current mirrors, cascode formation, etc.

In order to compare with previous works that synthesize with known building blocks, a second run of the experiment was conducted, this time with a *good* library obtained after the above first run. A fixed 100 chromosomes were used for each generation. The compliant circuit was obtained in the 26th generation, i.e. 2600 designs needed to be explored. This is better than [12] where an almost similar design was obtained after exploring 2732 designs. Also, [12] synthesized the circuit with all-correct building blocks, obtained from a standard text book. More to add, the functional constraints for inter-connecting the blocks were also designer certified. This easily proves the efficacy of our tool over previous works.

A synthesis run was also performed with a normal GA (i.e. no adaptive formation) with the offsprings produced only through cross-over and mutation. The required design was not achieved even within 500 generations. This experiment establishes the merit of our adaptive technique of offspring formation over normal GAs.

6.2 Design II: Three-stage Ring Oscillator

A three stage ring oscillator design is set as our second benchmark. An oscillator is used in clock generation, as prototype circuits to test new semiconductor processes, and so on [10]. Now, it is known that a ring oscillator is basically a cascade of similar stages where the output of one feeds to the input of the next, as shown in the prototype system level circuit in Fig. 9(a)(i). Hence, designing only a single stage of the whole circuit suffices the purpose.

Starting only with MOS transistors and current sources, the fully compliant circuit was obtained in the 73rd generation. The corresponding circuit for a stage is shown in Fig. 9(a)(ii). With the initial voltage at V_{out+} of stage-1 set to 0V, the output waveform obtained is given in Fig. 9(b). After an initial settling transient, it keeps on repeating the same pattern. The circuit has an oscillation frequency of 2.1 GHz. [2] synthesized a similar design with well-defined blocks after 63 generations, each consisting of 600 chromosomes. Our technique thereby converged faster compared to [2]. Also, the final design obtained is designer perceivable. It comprises of identifiable blocks like differential pair and diode-connected loads. Moreover, the block library formed consists of well known blocks like differential pairs adjacent to diode loads, set of biased transistors, etc., as shown in Fig. 9(c).

Table 2: Synthesis Results

Results	Design-I	Design-II
No. of generations required	152	73
No. of library cells produced	105	68

7. CONCLUSION

A GA-based topology synthesis framework has been introduced that is capable of synthesizing human understandable analog designs from the very basic elements. A building block library is used that has been generated adaptively, thus alleviating the need to supply a circuit-specific block library initially. The library not only contains information about the required blocks, but also their desired position and connectivity, with respect to other blocks. The framework has been able to synthesize an opamp and a three stage ring oscillator design. For industrial relevance, the tool has minimal application dependency, uses minimal design knowledge, and is flexible enough to be applied to any existing framework.

8. REFERENCES

[1] A. Das and R. Vemuri. An automated passive analog circuit synthesis framework using genetic algorithms. In *Proc. of IEEE International Symposium on Circuits & Systems (ISCAS)*, May 2008.

[2] T. R. Dastidar, P. P. Chakrabarti, and P. Ray. A synthesis system for analog circuits based on evolutionary search and topological reuse. *IEEE Transactions on Evolutionary Computation*, 9(2), Apr. 2005.

[3] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Prof., 1 edition.

[4] R. Harjani, R. A. Rutenbar, and L. R. Carley. OASYS: A framework for analog circuit synthesis. *IEEE Trans. on CAD*, 8(12), Dec. 1989.

[5] J. R. Koza, F. H. Bennett III, D. Andre, M. A. Keane, and F. Dunlap. Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Trans. on Evolutionary Comp.*, 1(2), 1997.

[6] W. Kruiskamp and D. Leenaerts. DARWIN: CMOS Opamp synthesis by means of a genetic algorithm. In *Proc. of DAC*, Jun. 1995.

[7] J. D. Lohn and S. P. Colombano. A circuit representation technique for automated circuit design. *IEEE Trans. on Evol. Comp.*, 1999.

[8] P. C. Maulik, L. R. Carley, and R. A. Rutenbar. Integer programming based topology selection of cell-level analog circuits. *IEEE Transactions on CAD*, 14(4), 1995.

[9] T. McConaghy, P. Palmers, G. Gielen, and M. Steyaert. Simultaneous multi-topology multi-objective sizing across thousands of analog circuit topologies. In *Proc. of Design Automation Conference*, 2007.

[10] B. Razavi. *Design of Analog CMOS Integrated Cir.* McGraw-Hill.

[11] R. A. Rutenbar, G. G. E. Gielen, and B. A. Antao. *Computer-Aided Design of Analog Integrated Circuits and Systems*. Wiley-IEEE Press, May 2002.

[12] X. Wang and L. Hedrich. An approach to topology synthesis of analog circuits using hierarchical blocks and symbolic analysis. In *Proc. of Asia and South Pacific Design Automation Conf.*, Jan. 2006.