

Exploration-Based High-Level Synthesis of Linear Analog Systems Operating at Low/Medium Frequencies

Alex Doboli, *Member, IEEE*, and Ranga Vemuri, *Senior Member, IEEE*

Abstract—This paper presents a methodology for high-level synthesis of continuous-time linear analog systems. Synthesis results are architectures of op-amps, sized resistors and capacitors such that their ac behavior and total silicon area are optimized. Bounds for op-amp dc gain, unity-gain frequency, input, and output impedances are found as a byproduct of synthesis. Subsequently, a circuit synthesis tool can be used to synthesize the op-amps of an architecture. The paper details the architecture generation technique. Architecture generation produces alternative architectures for a system specification using the tabu search heuristic. Its main advantages over traditional methods is that it is application independent, does not require a library of block connection patterns, and is simple to implement. The paper also discusses the hierarchical, two-step parameter optimization that guides architecture generation. Experiments showed that linear analog systems operating at low/medium frequencies (like telecommunication systems and filters) can be synthesized in a reasonably long time and with reduced effort.

Index Terms—Analog synthesis, high-level synthesis, system architectures, optimization.

I. INTRODUCTION

AUTOMATED synthesis of analog systems is an important research problem. Any real-life application requires an analog interface to its outer environment. Because of fewer synthesis tools, designing 10% of an overall mixed-signal system represented by analog circuits requires 90% of the overall effort [30]. Depending on the performed tasks, analog synthesis can be categorized into three types: 1) high-level synthesis; 2) transistor (circuit)-level synthesis; and 3) layout synthesis. High-level synthesis is the task of automatically producing optimized system architectures from abstract specifications with a hardware description language (HDL). Transistor (circuit)-level synthesis [20], [30], [32] finds transistor dimensions that optimize circuit performance (i.e., gain, bandwidth, CMRR, SR, etc.). Layout design tools [3] place transistors on a silicon die, and route signal and power interconnections to minimize total area and performance degradations due to layout parasitics. As compared to the other two synthesis kinds, high-level synthesis offers the advantages of shorter design cycles, reduced design effort, and ease of use by those unfamiliar with analog circuits [2]. High-level analog synthesis is still in its infancy, as little research has been performed [2], [13].

This paper is one of the first attempts to perform analog high-level synthesis from functional specifications with an HDL. Fig. 1 shows the top-down exploration-based synthesis methodology. Targeted applications are linear continuous-time analog systems operating at low/medium frequencies characterized by performance in the frequency domain. In our methodology, synthesis results are architectures (net-lists) that consist of op-amps, sized resistors, and capacitors such that their ac behavior and total silicon area are optimized. Bounds for op-amp minimum input impedance, dc gain, dominant pole, and

Manuscript received January 20, 2002; revised November 18, 2002. This work was supported in part by the IEEE and by the U.S. Air Force Research Laboratories, Wright Patterson Air Force Base under Contract F33615-96-C-1911.

A. Doboli is with the State University of New York, Stony Brook, NY 11794-1611 USA.

R. Vemuri is with the Department of Electrical and Computer Engineering and Computer Science, University of Cincinnati, Cincinnati, OH 45221 USA.

Digital Object Identifier 10.1109/TCAD.2003.818374

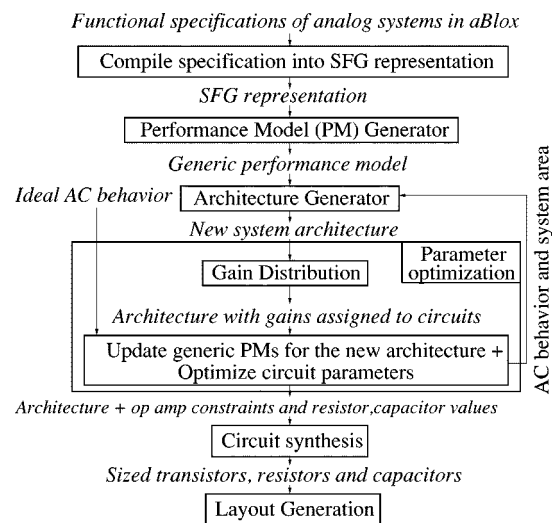


Fig. 1. High-level analog synthesis flow.

maximum output impedance are also produced as a byproduct of high-level synthesis. A circuit synthesis tool (such as [5], [19], and [21]) can be further employed for finding op-amp topologies and sizing their transistors.

The main benefits of the presented high-level synthesis methodology are designing analog circuits at a low cost and in a short time. This is important for the redesign, reengineering, and upgrade of analog legacy systems, which are part of products with very long life cycles such as airplanes, ships, and missiles [36]. Obsolete analog systems need to be rapidly redesigned at affordable costs to address discontinuities in their manufacturing process or improvements in their fabrication technology. Redesign starts from technology-independent specifications [36]. High-level synthesis tools can take these specifications as inputs and automatically produce optimized implementations. Rapid prototyping for upgrading obsolete parts is another possible usage of synthesis. Further, the discussed methods can be used for designing the analog interfaces of digital embedded systems, such as the interfaces to microphones, wire lines, sensors, etc. [13], [17]. Designers with a reduced experience in analog circuits can generate these interfaces by using high-level synthesis tools. Finally, analog implementations of consumer electronics such as headphones, hearing aids, hearing protection gears, amplifiers, etc. offer lower costs and energy consumptions as compared to digital signal processor (DSP)-based solutions, which are currently popular [17], [22], [27].

Functional specifications are mathematical relationships between input, output, and state variables of a design [9], [13]. Conservation laws at the ports of the connected blocks are not necessarily satisfied in the specification [13]. Various performance requirements annotate the blocks of a system [9]. Functional specifications do not explicitly indicate the kind, interconnection and parameters of analog circuits in the final implementation. Hence, a high-level synthesis flow must generate different implementation architectures for a specification, and then search for the architecture parameters (like circuit gains, poles, etc.) that optimize system performance such as ac and transient behavior, power consumption, and area. To achieve its goal, high-level analog synthesis performs three tasks [18]:

- 1) *architecture generation* for producing system topologies and architectures from specifications;
- 2) *performance model generation* for relating architecture parameters to circuit parameters;

3) *optimization* of system parameters using the obtained performance models.

This paper focuses on an original **architecture generation** method. Using tabu search (TS) [34], the architecture generator incrementally changes signal types from currents to voltages and vice versa, and then applies a set of rules to build alternative, specification-oriented architectures for a system. This method is different from traditional approaches based on pattern libraries and pattern-matching algorithms [6], [10], [12]. Patterns describe mappings of block clusters to circuits, and are partially specific to an application. As a result, pattern-matching-based methods are limited to a given class of applications like filters and decimators [2], [10]. New patterns have to be added to tackle different applications. Also, setting up pattern libraries is tedious, as hundreds of patterns are needed. The suggested algorithm considers a lower level of granularity than traditional techniques. It handles basic building blocks (such as op-amps, resistors, and capacitors) and not clusters of such blocks. As a result, new designs can be synthesized without requiring application-specific knowledge. The task of setting up large pattern libraries is also avoided. Finally, the proposed approach uses the simple and efficient TS algorithm instead of more complex pattern-matching methods.

The paper also presents the **hierarchical parameter optimization** method used for optimizing the ac behavior and overall silicon area of a system architecture. As shown in Fig. 1, its outputs guide the exploration loop of the architecture generator. More details on parameter optimization are given in [7]. The technique used for producing **symbolic performance models** for ac gain and phase of a system is discussed in [8].

This work is part of the VHDL-AMS synthesis environment (VASE) developed at the University of Cincinnati, Cincinnati, OH. The goal of the VASE tools is to reduce design cost of mixed-signal and analog circuits by performing an integrated, top-down synthesis process from high-level specifications. This paper presents the high-level front end, which creates op-amp-level architectures from abstract specifications. Next, topologies and transistor sizes of the op-amps are optimized using a hierarchical genetic algorithm and equation-based performance estimation [5]. Performance such as gain, power, unity-gain frequency, and slew rate are tackled during op-amp synthesis.

The paper is organized as seven sections. Section II discusses related work. Section III introduces the specification notation for synthesis. Section IV describes the architecture generation algorithm. Parameter optimization is detailed in Section V. Section VI provides experimental results and discussion of the suggested synthesis methods. Finally, Section VII offers our conclusions.

II. RELATED WORK

Most of the analog synthesis work focuses on optimizing the parameters of a given architecture. An efficient method is to analyze (using a simulator) a large number of parameter values inside an optimization loop [20], [30], [32]. Optimization loops implement heuristic algorithms such as simulated annealing [30] or genetic algorithms [5], [19], [21]. The main limitation is the large synthesis time required for bigger circuits [20]. An alternative is to use exact algorithms, e.g., branch-and-bound [25] or geometric programming [24], to solve an equational formulation of the synthesis problem. Besides computational complexity, the difficulty also consists in the tedious process of calculating the equations for the solvers.

Several techniques are proposed for architecture generation. The intuitive solution is to consider a library of block structures and their mappings to analog circuits, and then use the identity of connected blocks in a specification to find architectures. This is a technology-mapping problem and pattern-matching algorithms can be applied [2], [6], [10],

[12]. Antao *et al.* [2] propose ARCHGEN, a filter synthesis tool. Architectures in controllable, observable, and ladder forms are obtained for the state-space description of a filter. There is no performance optimization phase in ARCHGEN. Block parameters in the specification are directly mapped to the corresponding circuits. Fino *et al.* [10] describe a greedy pattern-matching method that uses a library of circuits described as signal-flow graphs (SFG) to find an implementation for a specification also expressed as an SFG. A similar approach is proposed by Ganesan *et al.* [12]. The limitation of pattern-matching methods is that a library does not optimally address the block structure of a system, as it enforces implementations only in terms of its predefined circuits. Addressing new system types might need introducing of new patterns into the library to accommodate new block structures. Very large pattern libraries are required. Veselinovic *et al.* [41] present an approach that selects architectures by applying filters to a static library of architectures. Given the constraints for an application, the feasibility of a topology is studied by analytical filtering based on boundary checking, interval analysis, and rule-based filtering. A similar approach is studied by Konczykowska *et al.* [18]. A very interesting method is suggested by Koza *et al.* for circuit synthesis [19]. The method employs a genetic algorithm that evolves hardware structures built of transistors, resistors, and capacitors connected in a limited number of ways. This paper proposes a new exploration-based architecture generation method. It can handle new systems without setting up a pattern library or adding application-specific patterns.

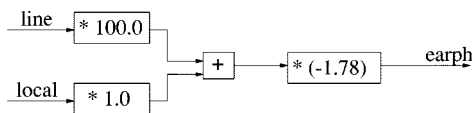
Synthesis tools are widely employed for designing continuous-time and switched capacitor filters [16]. Filter synthesis tools include the following steps [16]:

- 1) approximation of the overall transfer function;
- 2) decomposition of the transfer function into a sequence of cascaded, low-order stages (such as biquad, ladder, etc.);
- 3) mapping the stages to given circuit implementations;
- 4) computing the circuit parameters to meet imposed specifications.

The very large number of possible decompositions is handled by applying designer knowledge to prune low-quality decomposition alternatives. However, in the general case, it is difficult to relate different decompositions [16]. Optimal implementations are proposed for low-order stages using different implementation styles. Murali *et al.* [29] and Sanchez-Sinencio *et al.* [35], among others, discuss the implementation of switched capacitor filters out of biquad and ladder stages. Wu and Elmasry [43] present the design of current mode ladder filters. Recently, Ray *et al.* [33] proposed a method for realizing operational transconductance amplifier (OTA)-based filters starting from a filter decomposition into cascaded first- and second-order stages. Guaranteeing the optimality of the overall filter design is difficult, provided that strong electrical interactions exist between stages. Also, the analytical formulas have to be recalculated whenever the implementation technology is changed. The methodology presented in the paper can be used for continuous-time filter synthesis. The proposed architecture generator is general, and does not assume a certain decomposition of the transfer function. This allows exploring more filter structures to find a better solution for a specification.

III. SYNTHESIS-ORIENTED SPECIFICATION

An important paradigm of our synthesis method is that analog system functionality is expressed by describing *how* continuous-time analog behavior is achieved in terms of signal flow and processing. SFGs [31] accommodate well this description style, and they are specified as *aBlox* programs [9] in our environment. If system functionality is presented at a higher level of abstraction such as transfer functions or differential and algebraic equations, then techniques discussed by Antao



a) Signal flow and processing of telephone receiver

```

macro receiver is continuous_time
  inputs
    line is voltage
      with range 0-1.0 V;
    local is voltage
      with range 0-1.0 V;
  outputs
    earph is voltage
      with range 0-1.5 V
      with impedance 280.0 Ohms;
  attributes
    noise <= 80 dB;
    bandwidth is range 300-3600 Hz;
  arch receiver is
    output = (100.0 * line + local) * -1.78;
  end arch;
end macro;

```

b) Input specification of telephone receiver

Fig. 2. Synthesis-oriented specification of an analog system.

et al. [2] can be used to obtain SFG system descriptions. The case study in Section VI exemplifies this process.

We refer to a simplified version of the receiver module of a telephone set [38] for illustrating our specification notation for synthesis. More details about the specification notation are given in [9]. The functionality of the receiver is described by the block diagram in Fig. 2(a). The receiver provides an audible output signal to the earphone of the telephone set. It amplifies with different gains the signal transmitted from the calling part (signal *line*) and that produced locally by its own microphone amplifier (signal *local*). The specification imposes that port signals *line*, *local*, and *earph* are voltages and that their value ranges are [0.0, 1.0] V for inputs and [0.0, 1.5] V for the output. The output load is 280 Ω .

Fig. 2(b) depicts the aBlox program for the receiver module. aBlox programs consist of *macro* definitions having associated an architecture description through constructs *arch-end arch*. Next are the defining elements of aBlox notation.

- aBlox variables indicate either the *voltage* or the *current* facet of analog signals. Variables can be of three types: *voltage*—when they only correspond to voltages in implementations, *current*—when they are “realized” only as currents, and *unspecified*—when both voltage and current alternatives are acceptable in an implementation. In order to maintain a higher level of abstraction, only one of the facets (voltage or current) can be used for a variable in a specification. Otherwise, the variable corresponds to a physical point, hence eliminating the possibility for architecture generation.
- Statements such as assignments, if-statements, and macro calls describe how macro outputs relate to macro inputs. The following arithmetic operations are available: addition, subtraction, multiplication by a constant, and integration. This is a complete operator set for a linear system [31], and can be implemented with electronic circuits.
- Macro calls permit hierarchical specifications. This is useful for improving the effectiveness of specification and synthesis (such as Performance Model Generation) [8].
- System port signals are denoted as voltages or currents. Value ranges or impedances for ports can be described. These attributes are useful for sizing interface circuits.

- *Attributes* constructions fix constraints for macro implementations (like noise level or bandwidth) or any equational performance model that can be used for synthesis (for example, op-amp models for area). This is important as system components have heterogeneous performance. For example, the analog part of the telephone set includes two modules, a receiver and a transmitter, with different noise constraints [38]. In this example, *noise* and *bandwidth* are keywords for the tool, even though aBlox has a flexible mechanism to define attributes [9].
- *Generic parameters* are used for indicating the generic elements of a macro such as constant values, operators (i.e., addition, subtraction, etc.), block identities, and performance attributes. Generic parameters are useful for expressing regular and hierarchical macro structures. Each macro call instantiates concrete values for the generics. The two stages of the fourth-order filter in Fig. 3(a) are distinguished only by different filter constants that are specified as generics in the aBlox program in Fig. 3(b). Operators can be passed as arguments to macro calls for describing stages built of distinct blocks but connected in similar connection patterns.

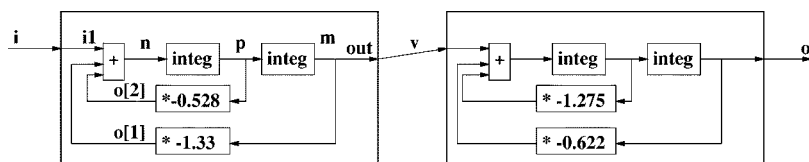
For accepting inputs in a standard mixed-signal modeling language, we proposed a VHDL-AMS [1] subset for synthesis. The subset includes *quantities* for describing abstract analog signals, *simple simultaneous*, *simultaneous if/case*, and *procedural* statements for expressing functionality and *port* annotations to denote attributes of the interfacing signals. Compiling rules were developed to translate VHDL-AMS programs into aBlox descriptions. Details on the VHDL-AMS subset and the compiling rules are offered in [6].

IV. AUTOMATED ARCHITECTURE GENERATION

The proposed architecture generation technique exploits the observation that analog signal processing can be obtained with signals that are either currents or voltages. If the input and output signals of an SFG block have their type fixed (as current or voltage), then networks (*RC* or circuit networks) that realize the block operation can be easily identified. This straightforward observation is important for architecture generation, as different net lists simply result by changing SFG signal types. Basically, in its exploration loop, the architecture generator incrementally changes signal types from voltage to current and vice versa, and applies a set of conversion rules to generate architectures specific for a system. Types of the system input and output signals are assumed to be known.

If all input/output signals of an aBlox block have known types (currents or voltages), then a *conversion rule* indicates the corresponding hardware for implementing the operation of the block. Hardware structures can be op-amp-based active circuit (i.e., gain stage, summing amplifier, etc.), passive *R* or *RC* network, or even a common connection point of two wires. Fig. 4 exemplifies two of our conversion rules, and illustrates how two distinct architectures result for the SFG depicted in Fig. 4(a). We assume that inputs *v1* and *v2* and output *out* are voltages. Inferring architectures for the SFG means deciding the type of signal *aux*, as *aux* can be either current or voltage.

- 1) If signal *aux* is a current, then the structure in Fig. 4(b) is obtained by applying conversion rules for the specific signal types and block operations. The summing operation of *block1* is obtained by first “converting” voltages *v1* and *v2* into currents and then connecting them to the same point *A*. Because of Kirchhoff’s current law, the summation operator implicitly results, if the voltage of point *A* is 0 (otherwise currents *i1* and *i2* are not dependent only on voltages *v1* and *v2*). The need of point *A* being virtual ground is satisfied by *block2*.



(a)

```

macro stage
  inputs
    i1;
  outputs
    out;
  generics
    constants a1, a2;
  arch controlable is
    variables
      m, n, p;
      o is array[2];
      o[1] = a1 * m;
      o[2] = a2 * p;
      n = i1 + o[1] + o[2];
      p = integ(n);
      m = integ(p);
      out = m;
    end arch;
  end macro;

```

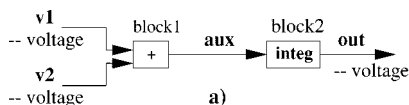
```

macro filter
  inputs
    i is voltage;
  outputs
    o is voltage;
  arch two_stage_filter is
    variables
      v;
      v = stage.controlable(
        i, generics are
          -1.33, -0.528);
      o = stage.controlable(
        v, generics are
          -0.622, -1.275);
    end arch;
  end macro;

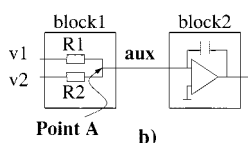
```

(b)

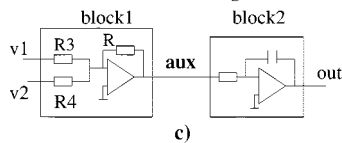
Fig. 3. aBlox specification of a fourth-order filter.



a)

(1) *aux* is current

b)

(2) *aux* is voltage

c)

Fig. 4. Example of a conversion rule.

- 2) If signal *aux* is a voltage then the architecture in Fig. 4(c) results. For these signal types, *block1* has to be a summing amplifier and *block2* an integrator. No special constraints are needed, e.g., point *A* being virtual ground.

Fig. 5(a) and (b) illustrates two implementations for the receiver module obtained by our method.

Table I exemplifies some of the conversion rules embedded in the architecture generation method. The first four columns indicate the operation of an aBlox block and the types of block inputs and outputs. Column 5 describes required conditions for connecting the circuit output, such as the output must be connected to the ground. Column 6 presents circuit characteristics exploited for circuit interconnection (like inputs with virtual ground). Interconnection constraints (Columns 5 and 6) are useful for speeding up the architecture generation process. They eliminate architectures that are infeasible for getting the SFG functionality or are redundant, as they are equivalent with other produced architectures. These constraints are set up based on designer knowledge. Architecture generation is still valid without interconnection constraints, if the resultant behavior is verified by the parameter optimization step, and infeasible architectures are discarded. Nevertheless, it involves a cumbersome analysis to detect some obviously

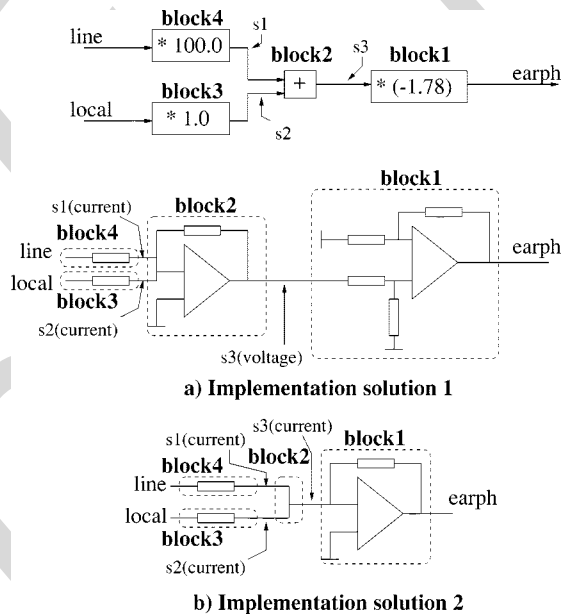


Fig. 5. Samples of architectures generated for the receiver module.

incorrect solutions. Finally, the last column presents the corresponding circuit for a block.

The exploration strategy embedded in the architecture generator is TS [34]. We decided to use TS because it is a deterministic algorithm with a short convergence time. The pseudocode of the TS algorithm is depicted in Fig. 6. It starts from an initial architecture solution where each of the aBlox blocks is mapped to a distinct op-amp-based circuit. Then, for each iteration, subroutine *Find_Best_Move* (Line 5 in Fig. 6)

TABLE I
CONVERSION RULES FOR ARCHITECTURE GENERATION

Block Operation (1)	Input Types		Output Type (4)	Interconnection Constraints (5)	Circuit Characteristics (6)	Circuit (7)
	Inp1 (2)	Inp2 (3)				
Multiply by constant	voltage	-	voltage	-	Has an input point at virtual ground	gain stage
Multiply by constant	voltage	-	current	Output connected to virtual ground	gain > 0	resistor
Adder	voltage	voltage	voltage	-	Has an input point at virtual ground	summing amplifier
Adder	voltage	voltage	current	Output connected to virtual ground	-	resistor pair
Integrator	voltage	-	voltage	-	Has an input point at virtual ground	integrator
Integrator	current	-	voltage	Output connected to virtual ground	-	capacitor

```

procedure NetListGenerator () is begin
(1)   call Produce_Initial_NetList ();
(2)   find area and AC behavior by calling Par_Optim();
(3)   save initial solution as best solution;
(4)   while (NetList-exploration is not finished) do begin
(5)     current_signal = Find_Best_Move ();
(6)     if current_signal is NULL then
(7)       current_signal = Aspiration_Criteria1 ();
      end if
(8)     if current_signal is NULL then
(9)       current_signal = Aspiration_Criteria2 ();
      end if
(10)    if current_signal is NULL then
(11)      current_signal = Find_Closest_Tabu_Expiration ();
      end if
(12)    current_solution = call Execute_Move ();
(13)    find area and AC behavior by calling Par_Optim();;
(14)    if (current solution is best so far) then
(15)      save current solution as best solution;
      end if
(16)  end while
end procedure

```

Fig. 6. TS heuristic for architecture generation.

individually considers all signals in the SFG, and attempts to change their type to the opposite one. For example, if a signal is a current, then its type is modified to voltage. Changing a signal type also implies modifying the circuits for realizing the operations of the source and target blocks. A new architecture results by applying appropriate conversion rules for the new situation. Area and ac behavior of the resulting architecture are found with the parameter optimization module *Par_Optim*. The signal with the best performance improvement is returned. Subroutine *Execute_Move* (Line 12) performs the best signal-type change, and the resulting architecture is the current solution for the next iteration. If a signal type is modified, the signal enters a *tabu* status, meaning that its change is prohibited for a precise number of iterations (*tabu tenure*). This avoids repeated changes of the same signal. For converging to a local optima, the algorithm first accepts only moves that improve the quality of the current solution. However, if a *tabu* move results in a solution better than the best so far, then it is accepted as a *tabu aspiration*. The exploration algorithm ends after performing a number of N_{max} iterations, or if N_{max}^{rbest} iterations passed since the best solution was found.

The algorithm uses three strategies for escaping from local optima. First, it attempts to keep the search locally by selecting the move that previously was improving, and offers the least performance degradation (subroutine *Aspiration_Criteria1* at Line 7). If this does not work, then the search attempts a local diversification by trying to move to a

structurally distinct solution point, but which does not worsen the solution quality too much. Routine *Aspiration_Criteria2* in Line 9 uses the following cost function to decide the signal to be changed:

$$\text{Cost} = \Delta[(\text{Area}_{\text{new}} - \text{Area}_{\text{current}}) * (N/N_{\text{iter}})^2]$$

where N is the number of iterations the signal was of type current and N_{iter} is the total number of iterations so far. Structural changes of the explored solutions are encouraged by squaring N/N_{iter} . If still no signal is selected, then the signal closest to its *tabu* status expiration is chosen (subroutine *Find_Closest_Tabu_Expiration* in Line 11).

This approach to architecture generation is different from pattern-matching-based methods [6], [10], [12], where possible structures are always selected using the fixed set of patterns in a library. A pattern indicates how a group of connected SFG blocks is mapped to electronic circuits [6]. Therefore, a pattern reflects not only the kind of involved SFG blocks but also their connection structure. Instead, the proposed architecture generation technique considers a lower level of granularity. Conversion rules show how individual SFG blocks are mapped to electronic circuits; thus, they correspond to the possible building blocks in an architecture. Rules do not incorporate any information on the connection structure of blocks. Predefined interconnection constraints prohibit the creation of incorrect structures. As a result, the number of conversion rules is much smaller than the number of patterns in traditional

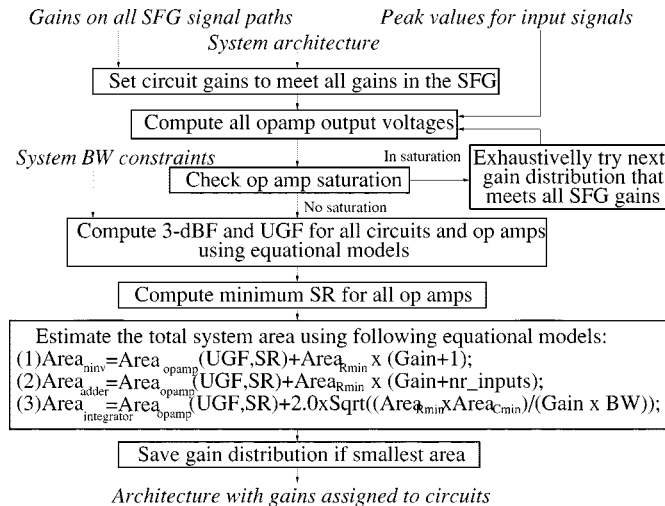


Fig. 7. Gain distribution algorithm.

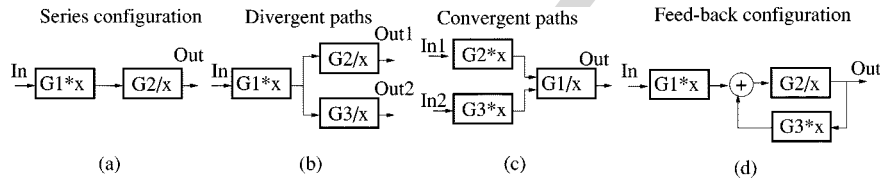


Fig. 8. Rules for gain distribution.

methods. Also, conversion rules are application independent as opposed to the need to define application-specific patterns (specific to the block structure of the current design) for the pattern-matching-based methods. Finally, the proposed formulation of the architecture generation problem is similar to the bipartitioning problem for which exist simple and effective algorithms [4]. Pattern-matching algorithms are much more cumbersome.

The current implementation of the architecture generator allows only one circuit implementation for one SFG block with known signal types at its ports. In reality, multiple implementations are possible for a block. The architecture generation algorithm can be extended to incorporate this capability. Then, for each SFG block with known signal types, subroutine *Find_Best_Move* explores all possible implementations and selects the one with the best performance. The method also assumes that any internal signal can be either voltage or current. To address certain applications such as phase-locked loops, the algorithm can be adapted to handle signals with a fixed type. Routine *Find_Best_Move* will avoid exploring these signals.

V. PARAMETER OPTIMIZATION

The parameter optimization module finds values for the block parameters such that: 1) the obtained ac behavior is within an acceptable error margin from the ideal behavior and 2) the overall system area is minimized. The ideal ac behavior assumes ideal blocks (with infinite gain, bandwidth, and input impedance and zero output impedance), and can be found with well-known simulation tools like MATLAB. Using proper evaluation methods [39], [40], the proposed parameter optimization method can be extended to incorporate other important performance such as power and noise. Parameter optimization was organized as a two-step task (gain distribution followed by the actual parameter optimization) to address the huge solution spaces characteristic to analog synthesis [7].

1) *Gain Distribution*: Fig. 7 presents the gain distribution algorithm for an architecture. The algorithm conducts gain-bandwidth

(BW) and BW-slew rate (SR) tradeoff explorations [23]. Using equational models, it decides the dc gains for each circuit so that overall gains on different signal paths are met (as specified in the aBlox programs), no saturation occurs, and the resulting area is potentially the smallest. In a fast process, constraints are set for some of the external resistors and capacitors in an architecture. This reduces the exploration space by eliminating about 20% of the free variables.

The gain distribution algorithm traverses an architecture, and exhaustively tries out possible gain allocations. It first handles the paths with a higher gain per active circuit because these will impose tighter constraints on gain distribution. The assigned gains are in the range $[1.0, G_o]$, where G_o is the overall gain of the path the circuit belongs to. Fig. 8 depicts gain distribution for different structures. For example, in Fig. 8(a), if the first circuit gets its gain multiplied by x , the next circuit will have its gain divided by x . Then, op-amp output voltages are calculated for each gain distribution based on the allocated gains and the peak values of the input signals. Depending on the used technology, the algorithm checks that no op-amp saturates. BW requirements are deduced for each op-amp depending on the architecture structure and circuit types. Knowing the 3-dB bandwidth for a nonfeedback sequence of n circuits, the 3-dB bandwidth of a circuit is evaluated as

$$3\text{-dB frequency}_{\text{circuit}} = 3\text{-dB frequency}_{\text{overall}} \times \sqrt{2^{\frac{1}{n}} - 1}.$$

This formula captures the cumulative influences of circuits in a sequence. It holds if all circuits have the same 3-dB frequency. If an overall 3-dB bandwidth is imposed for a feedback structure, the algorithm assumes that: 1) the direct path has the same 3-dB bandwidth as the feedback structure and 2) the feedback path has one decade more. This assumption avoids the introduction of zeros in the transfer function so that pole-zero cancellation is needed. However, it overestimates the areas of the op-amps in the direct paths. This overestimation is less important for analog designs [40]. Next, minimum op-amp SRs are evaluated using the previously calculated BWs and output voltages

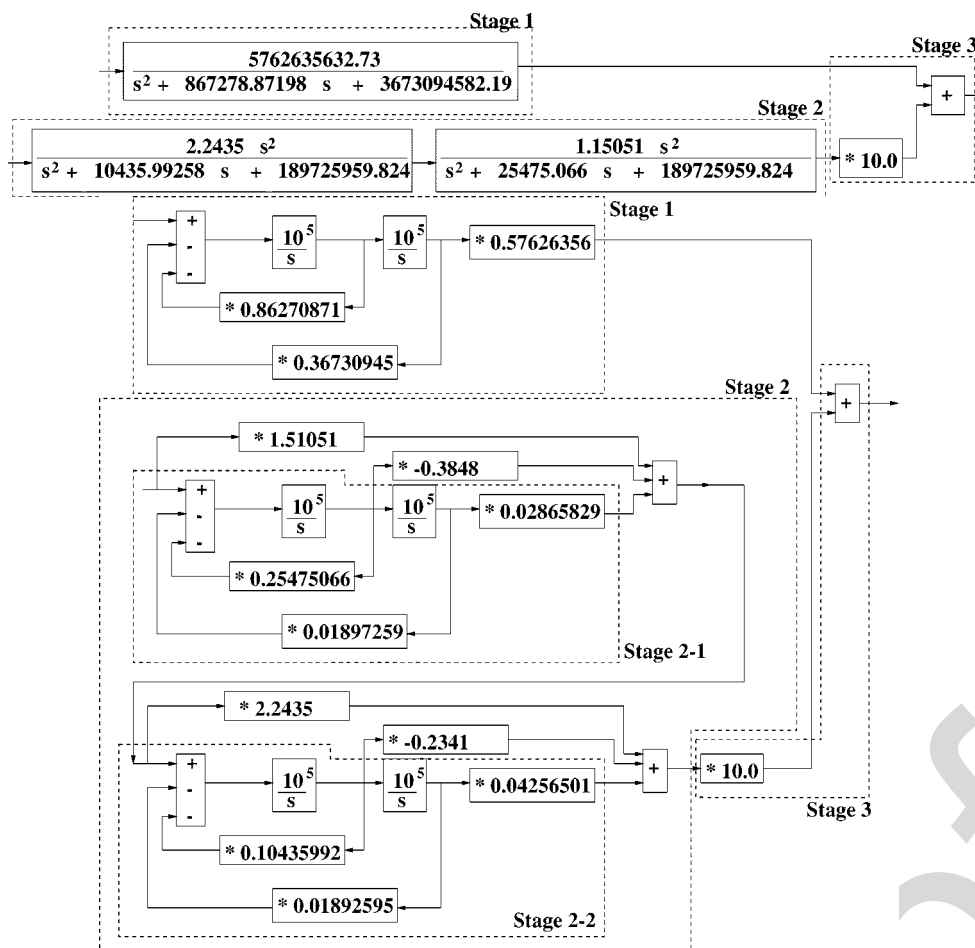


Fig. 9. Block and SFG structure of the optimized filter.

(BW–SR tradeoff [23]). Finally, using regression models for op-amps [7], the circuit area is estimated as a function of gain (Gain), unity-gain frequency (UGF), and SR.

2) *Parameter Optimization for ac Behavior and Area*: The second step sizes the external resistors and capacitors and finds feasible bounds for op-amp parameters (input and output impedance, open-loop gain, and dominant pole) so that: 1) the ac behavior of the system at any considered frequency is within an acceptable relative error from the ideal behavior and 2) the total area is minimized. Values for the free design parameters are searched in their feasibility range for CMOS technology [15]. This step considers the previously set dc gains. The parameter optimization module was developed as a TS algorithm, similar to the one depicted in Fig. 6. Following cost function models the two optimization criteria

$$\text{Cost} = \sum_{\text{all resistors}} \text{resistor area} + \sum_{\text{all capacitors}} \text{capacitor area} \\ + \sum_{\text{all op amps}} \text{op amp area} + \beta \sum_{\text{all frequency values}} \text{error}$$

The term $\sum \text{error}$ indicates the amount by which the real ac behavior differs from the ideal behavior. β is a scaling factor. The ac behavior of the system output is calculated for each frequency using automatically produced symbolic models [8]. System stability is checked by verifying that the phase margin is lower than 60° when the gain value is 1.0 [23].

The main advantage of the two-step parameter optimization technique is finding good solutions in an acceptable length of time. In contrast, a flat optimization needs long optimization times [5] or involves

expensive computing resources [20]. The total silicon area might be higher for the two-step method than for flat optimization. Silicon area is, however, less important for analog designs [40]. The two-step approach is also useful in tackling constraint-satisfying design scenarios, where performance must be met and not optimized.

A different optimization alternative is to model the system as a set of analytical equations and inequalities, and to optimize it using exact techniques such as the steepest descent method [26]. This is difficult because symbolic expressions are large, involving around 100 variables. Such large models approach or exceed the capacity of the optimization tools. Before solving, the symbolic models can be approximated to reduce their sizes. This introduces inaccuracies in the design. Other difficulties relate to finding local and not global optima, the Hessian matrix may be indefinite or singular, and poor convergence of the methods [26].

VI. EXPERIMENTAL RESULTS, DISCUSSION, AND LIMITATIONS

The proposed algorithms were implemented in 9500 lines of C/C++ code. Experiments were run on a SUN Sparc5 workstation running at 170 MHz. First, we discussed a real-life inspired case study designed with the proposed synthesis method. It shows that the methodology is useful in reducing the designer effort. A second set of experiments studied the computational aspects of the techniques. Limitations of the approach are discussed at the end.

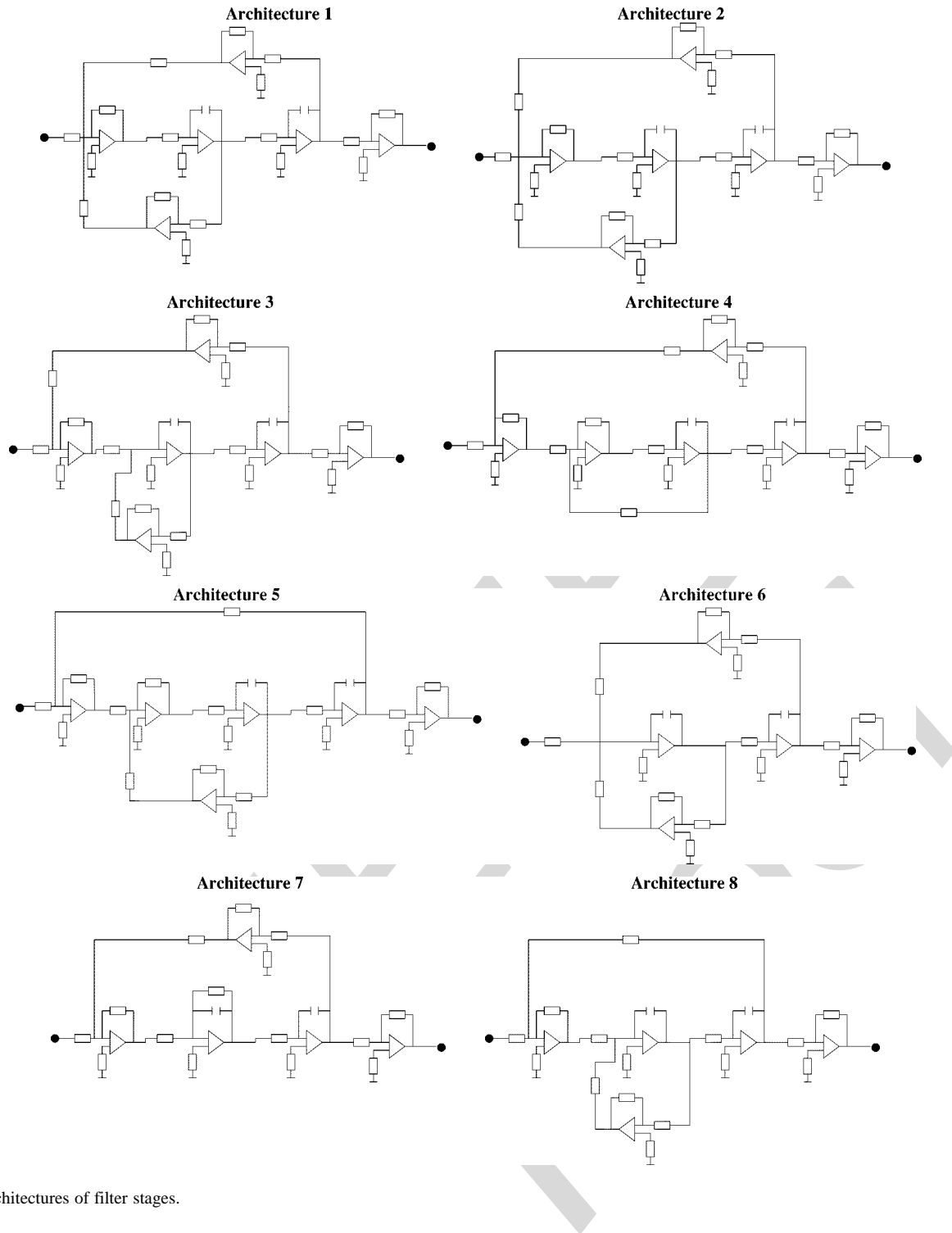


Fig. 10. Architectures of filter stages.

A. Experiments and Discussion

1) *Case Study:* The considered application is the optimized filter (OF) of the Eartalk system [11]. The top part of Fig. 9 presents the block structure of the OF. The OF consists of the following parts.

- The low-frequency band filter for transmitting a portion of the spoken signal (*Stage 1* in Fig. 9).
- The high-frequency band filter for transmitting a second portion of the spoken signal (*Stage 2* in Fig. 9).
- The signal mixing circuitry combines the signals from the two filters (*Stage 3* in Fig. 9).

The first step was to simulate in MATLAB the transfer functions of *Stage 1* and *Stage 2*, and the entire OF, and collect sample points (for gain and phase) for the ideal ac behavior at different frequencies. The second step was to convert the transfer functions of *Stage 1* and *Stage 2* into their SFGs in controllable forms [2], [31]. We applied transformations techniques described by Ogata *et al.* [31] and Antao *et al.* [2]. The bottom part of Fig. 9 depicts the resulting SFG structure for the OF. Using the observable or ladder forms [2] for the two stages could have been other possibilities for obtaining different SFG structures. Then, the SFG in controllable form was used to develop the aBlox description of the OF. This description together with the sampled ac points constituted the inputs for the high-level synthesis tools.

Each of the stages *Stage 1*, *Stage 2-1*, *Stage 2-2*, and *Stage 3* was separately synthesized using the proposed synthesis methodology. As *Stage 1*, *Stage 2*, and *Stage 3* have similar SFG structures, the same kind of architectures were generated by the architecture generator for each stage. Fig. 10 depicts some of the automatically created architectures.

After producing each architecture, automated gain distribution was performed. Parameter optimization ended the high-level synthesis step. Optimization criteria were that: 1) the error of the real ac behavior is within 10% from the ideal ac behavior obtained with MATLAB and 2) the filter area is minimized. Based on the synthesis results, we decided to use *Architecture 3* for *Stage 1*, as it had the smallest area, and its error was larger only by an insignificant amount as compared to *Architecture 1*, which provided the smallest error from the ideal ac behavior. Solution *Architecture 5* was selected for *Stage 2-1*, as it had the smallest area and its ac output error was small. Finally, solution *Architecture 1* was chosen for *Stage 2-2*, as it was the only one with a small ac error. Note that even though the SFG structure of the three stages is identical, different architectures are the best for each stage.

The next step was circuit synthesis. Op-amp constraints for gain, dominant pole, input, and output impedance were given as input to a circuit synthesis tool [5] that decided the topologies and sized the transistors of the op-amps. The SPICE description of the optimized filter was given as input to KOAN/ANAGRAM placement and routing tools [3]. After layout design and parasitic extraction, the circuit was simulated again. Resulting SPICE simulations for ideal (MATLAB), transistor, and layout level are provided in Fig. 11. Simulations show that the ac behavior of the OF design including layout parasitics satisfies the imposed constraints. This happens because the influence of parasitic capacitances was negligible for the considered frequency range.

2) *Experiments Based On Synthetic Examples*: The second set of experiments observed the following computational aspects of the proposed architecture generation and parameter optimization techniques: solution-space complexity, variety of explored solutions, memory, and time overhead. Nine synthetic examples were used. Examples differ by their number of blocks and signals, number of signal paths, and structural topology. According to their topology, we categorized systems into three classes: nonfeedback systems, pure feedback systems, and mixed nonfeedback/feedback systems. No cyclic signal paths occur in a nonfeedback system. Such systems are employed for data acquisition, signal processing [17], and telecommunication [38]. Pure feedback systems have only cyclic signal paths. They refer to filters [37] and control systems [31]. Mixed nonfeedback/feedback systems include both acyclic and cyclic signal paths. They are used for applications like analog computers [42].

Table II describes the nine examples and their characteristics. The first three examples are the nonfeedback systems. Examples t1, t11, t12 correspond to the telephone receiver module [38] and two scaled versions of it. Block descriptions for examples t11 and t12 are provided in Fig. 12(a) and (b). Examples f2s, f3s, f5s, and f5o are second-, third-, and fifth-order filters. The first three filters are in cascaded form, and the last one is in observable form [2]. Examples chem and auto are analog computers for solving differential equations [42]. Fig. 12(c) and (d) show their descriptions.

Fig. 13 illustrates the variety of architectures produced for the second-order filter f2s. All implementations were built only with gain stages, inverting summing amplifiers, integrators, and external resistors and capacitors. Architectures for the other examples were generated in a similar way. Fig. 13(a) depicts the SFG representation of the filter. Fig. 13(b), (c), and (d) present three different architectures and the corresponding signal types for inferring these architectures. Note that Fig. 13(d) actually represents the well-known Tow–Thomas biquad implementation [37]. If a difference circuit were also used, then the architecture in Fig. 13(c) can be further refined into the

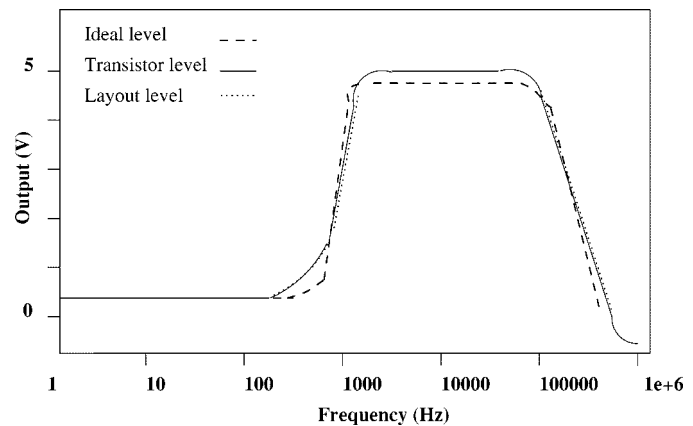


Fig. 11. Transistor and layout level simulation of the optimized filter.

TABLE II
EXAMPLE SET

	Example (1)	# of blocks (2)	# of signals (3)	# of distinct signal paths (4)	Topology type (5)
1	t1	4	6	2	non-feedback
2	t11	12	18	6	non-feedback
3	t12	20	28	9	non-feedback
4	f2s	6	9	2	pure feedback
5	f3s	10	14	3	pure feedback
6	f5s	16	18	5	pure feedback
7	f5o	16	22	5	pure feedback
8	chem	9	14	3	mixed
9	auto	25	36	11	mixed

Kelwin–Huelsman–Newcomb implementation [37]. This example shows that distinct implementations result for a system by exploring for possible signal types (voltages or currents) and circuit kinds for realizing the operations in an SFG. No pattern libraries are required.

Table III offers experimental results for architecture generation and gain distribution. Column 2 indicates the number of different architectures generated. Columns 3 and 4 describe the convergence of architecture generation: Column 3 shows the total number of moves and Column 4 the move where the best solution was encountered.¹ Column 5 presents an estimated number of patterns needed in a pattern-matching-based method to produce the same architectures as the suggested technique. Columns 6 and 7 express the complexity of the gain distribution process as the minimum and maximum number of gain distributions for all architectures of a system. The last column presents the CPU time for gain distribution alone, as it is much longer than architecture generation.

The experiments demonstrate that a significant number of architectures can be produced for an SFG specification. This is due to the many alternatives of implementing the operations in an SFG and to the many circuit variants for achieving the same operation (such as using inverting and noninverting summing amplifiers, inverting and noninverting gain stages). Table III shows that the architecture generation technique tends to produce more architectures for systems including more gain and summing operations, i.e., examples t11 and t12 than for systems having more integration blocks, i.e., examples f3s and f5s. The reason is that systems of the first kind allow more alternatives for signal types as voltages and currents (thus, more architectures) than the latter. Some of the produced architectures are incorrect because the resulting signals do not have the same sign as in the input specification. This is crucial for feedback systems where failing to realize the correct sign

¹By best solution we mean the best solution found by the algorithm and not the optimal solution.

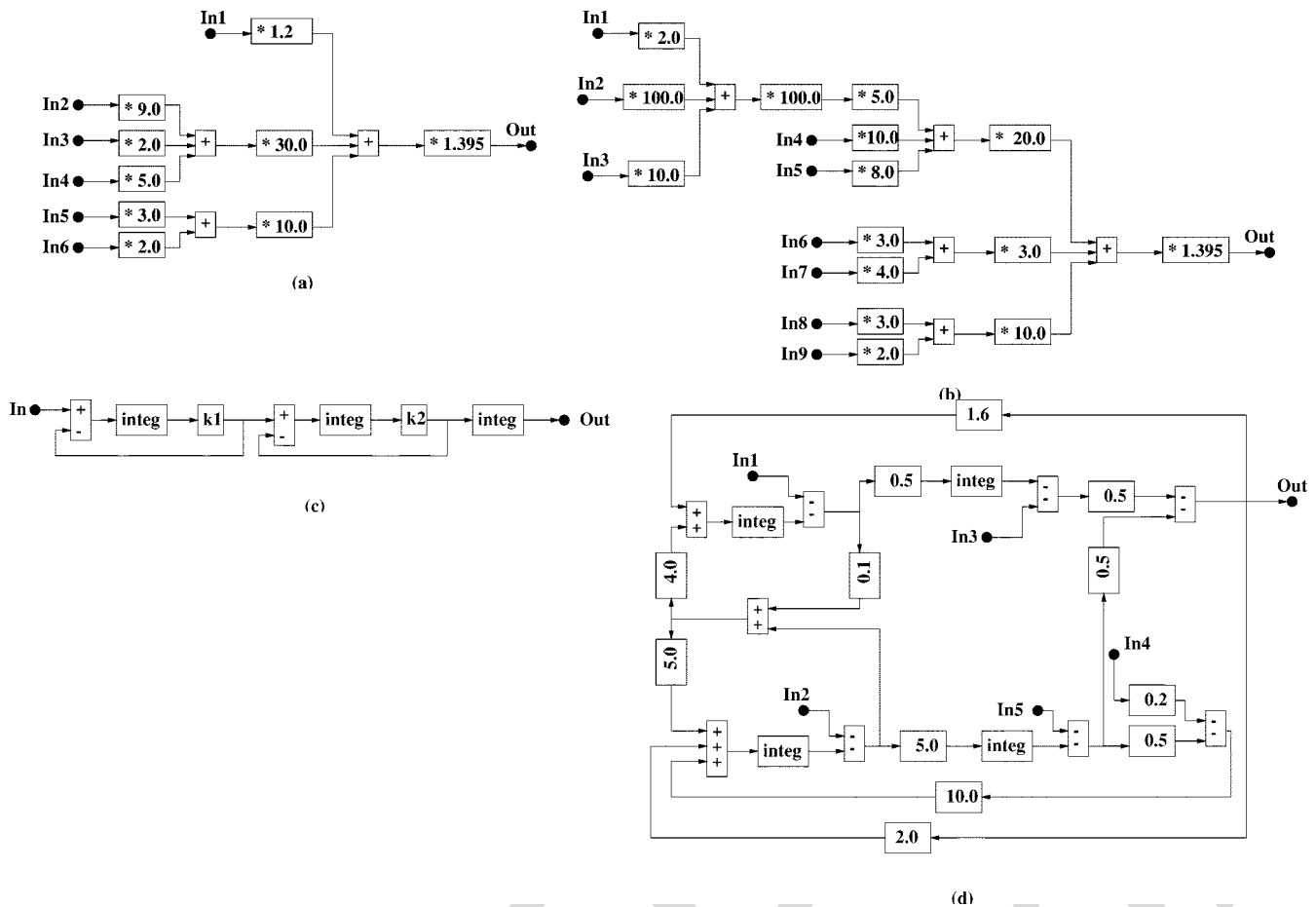


Fig. 12. Example set.

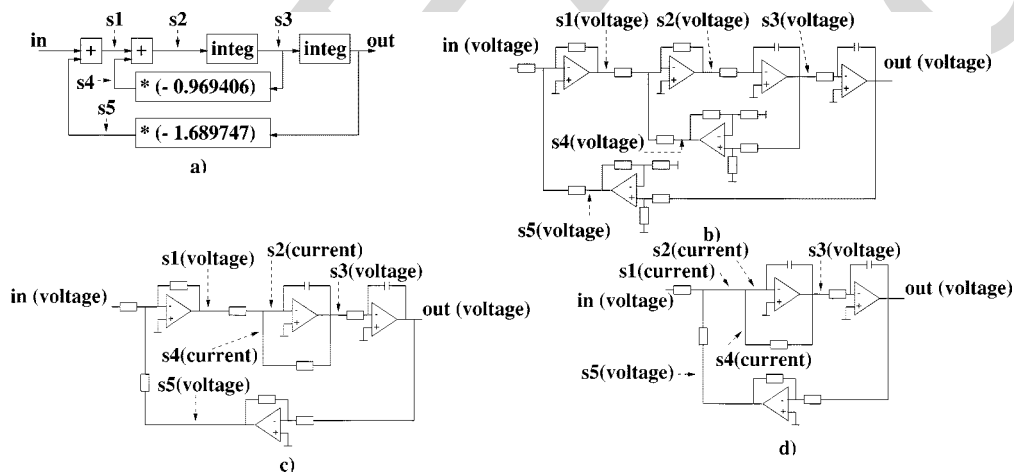


Fig. 13. Architecture generation for second-order filter.

for a signal path leads to instability. Such incorrect solutions can be filtered out by the architecture generator. For keeping architecture generation simple, we decided to shift the filtering task to the parameter optimization module. Columns 3 and 4 indicate that the TS-based architecture generator rapidly converges toward the best solutions. Column 5 shows that the proposed method saves the considerable effort needed for setting up the pattern library for pattern-matching-based architecture generation. A limitation of our architecture generation technique is that it does not perform any restructuring of the SFG representation. As a consequence, the architecture generator could not produce

for example t1 the implementation with the minimum op-amp number. This solution does not correspond to the input SFG. SFG restructuring can be incorporated into our technique by moving, swapping, and clustering blocks in an SFG.

Results on architecture generation also show that it is hard to decide *a priori* what architecture is the best for a system. This decision is possible only after optimizing parameters of the system architecture and analyzing the resulting performance attributes. For example, it rarely happened in our experiments that a solution with a minimum op-amp

TABLE III
ARCHITECTURE GENERATION AND GAIN DISTRIBUTION ALGORITHMS

	Ex. (1)	# diff. archit. (2)	Total # moves (3)	Best sol. (4)	Estim. # patterns (5)	# gain distrib.		Time (sec) (8)
						Min. (6)	Max. (7)	
1	tl	8	22	2	92	4	41	0.08
2	tl1	317	35	5	92	101	791	2.91
3	tl2	694	45	15	92	251	126k	225.7
4	f2s	16	24	4	68	27	73	0.18
5	f3s	48	38	8	68	454	1507	3.65
6	f5s	115	41	11	68	862	175k	364.4
7	f5o	141	60	35	616	41	371k	314.3
8	chem	17	23	3	-	19	106	0.18
9	auto	161	42	22	-	56	13k	252.1

number was found the best. Design parameters of such solutions are more constrained, and, thus, have less opportunities for optimization.

Columns 6 and 7 in Table III explain that gain distribution is a very complex task due to the many ways of allocating gains to the circuits. Accordingly, Column 8 shows that high execution times result for gain distribution. Solution alternatives exponentially grow with the number of active circuits in an implementation and the number of signal paths in a system. Thus, it is worthwhile to separate gain distribution from the rest of the parameter optimization process.

Fig. 14 presents results for the actual parameter optimization task of a fifth-order low-pass Chebyshev filter with the cutoff frequency of 100 kHz (example f5s). For the filter architecture with a maximum number of op-amps, the figure depicts the ideal ac behavior and the transfer functions at different moves. We decided to present results for this architecture, as it had the largest solution space (highest number of free variables) among all architectural alternatives. The filter was optimized in 1647 moves, and the best solution corresponded to move number 1587. The optimized response almost overlaps with the ideal ac behavior.

B. Limitations

The following limitations need to be addressed to extend the applicability of the proposed methodology and algorithms to other types of continuous-time systems.

- *Extend the variety of system architectures:* The architecture generator produces architectures with the same signal flow and processing as that in the abstract specification. For filters, the method creates state-variable architectures (like the KHN filter [37]) and biquad filters (such as the Tow-Thomas filter [37]). State-variable architectures produce simultaneously multiple outputs (high pass, low pass, and band pass). They offer independent control over gain and good-quality factors. However, their total harmonic distortion and noise characteristics are not very good [37]. Other structures such as KRC circuits [37] or two op-amp filter stages are not produced by the method, as they do not directly correspond to the SFG description. A pattern-matching-based approach can produce such architectures by expressing them as patterns. For high-performance applications, the designer knowledge and experience is critical in finding the best implementation. Architecture generation can be also used for producing architectural alternatives for the linear part of some nonlinear systems like analog-to-digital converters (ADC). We created multiple implementations for the linear part of a second-order sigma-delta ADC [28]. Modulator architectures resulted by connecting the linear architectures with a quantizer and a digital-to-analog converter.
- *Address different implementation styles:* Generating OTA [23] based implementations is also a possible continuation of this work. OTAs offer a simple design and an inherent wide band

capacity. To produce OTA-based architectures, the conversion rules for the architecture generator need to be replaced. Using the parameter optimization module, we designed a third-order elliptic filter [33] having its 3-db bandwidth around 12 MHz. We had to change all modeling aspects such as the OTA models for gain distribution and feasibility ranges.

- *Synthesis of systems with tight electrical interactions:* Certain architectures (like the winner-takes-it-all circuit in [17]) do not realize their functionality by composition of operations of op-amp-based circuits. Instead, they rely on the tight interaction of lower level blocks, such as current sources, and gain stages. To address such architectures, the interconnection constraints and circuit characteristics aspects of the conversion rules (see Table I) need to be extended. Genetic programming-based techniques are also a promising approach [19].
- *Parameter optimization for systems with tight electrical interactions:* Parameter optimization assumes that system performance can be expressed using circuit parameters (like gain and dominant pole) that are independent of the circuit connections. This assumption is valid for systems such as filters and converters. For sigma-delta converters, Medeiro *et al.* [28] present expressions for signal-to-noise ratio (SNR) and dynamic range (DR) as functions of integrator dc gains. These expressions can be used to allocate dc gains such that SNR and DR meet specifications. Then, the rest of the parameters are optimized to meet constraints like harmonic distortion and overloading [28]. If tight electrical interactions characterize a system then a flat optimization has to be conducted instead. Due to its computational complexity, flat optimization is reasonable for small/medium-sized circuits.
- *Reduced modeling of layout parasitics (including resistive, capacitive, and inductive parasitics):* Improving the modeling of layout parasitics for system and circuit-level design is considered to be a timely topic [14]. Current research efforts concentrate on building models and techniques that predict parasitic values to accommodate them early in the design process. This will permit increasing the frequency range of the designs.
- *Limited capability to evaluate transient and nonlinear performance attributes:* Using symbolic modeling for gain and phase offered fast performance evaluation. However, it restricted the application domain to linear systems described in the frequency domain. Using numeric simulators like MATLAB and SPICE would allow to contemplate transient and nonlinear performance attributes during high-level synthesis. However, the optimization process would be cumbersome. Alternatives are to extend symbolic performance model generation to transient and nonlinear analysis by using piecewise linear modeling or Pade approximations [13]. Noise evaluation (white noise, $1/f$ noise, etc.) has to be also considered. Then, other types of applications such as ADCs and phase-locked loops can be synthesized using the proposed methodology.

VII. CONCLUSION

This paper presents a methodology for high-level synthesis of continuous-time analog systems from functional HDL specifications. Synthesis results are architectures that consist of op-amps and sized resistors and capacitors such that their ac behavior and total silicon area are optimized. The proposed methodology is used in conjunction with a circuit synthesis tool, as bounds for op-amp dc gains, UGFs, input, and output impedances are found as a byproduct of synthesis.

The paper focuses on architecture generation, one of the main high-level synthesis tasks. Architecture generation explores distinct architectures for a system using the TS heuristic. Compared to traditional

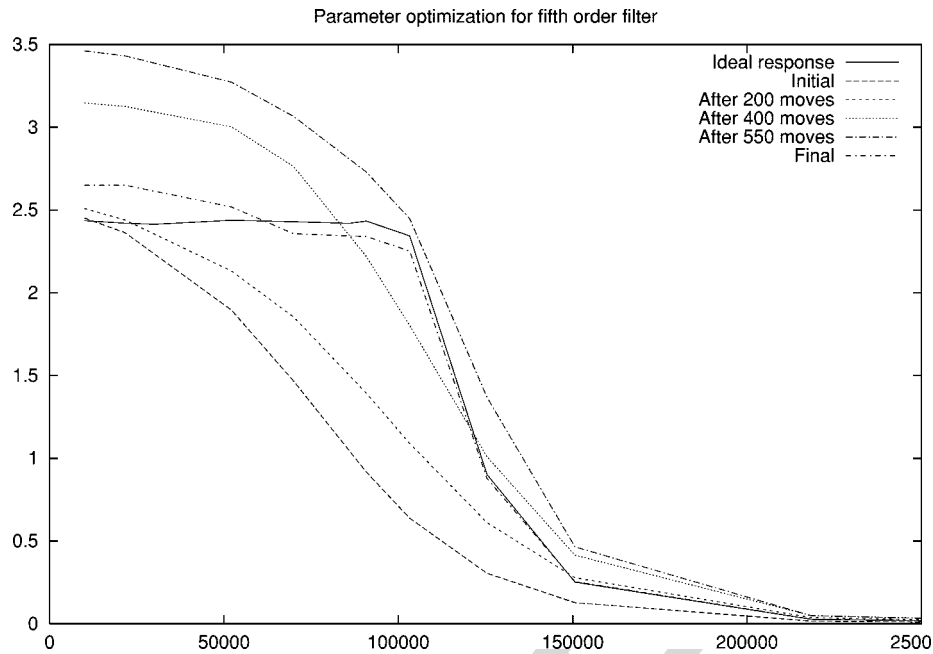


Fig. 14. Parameter optimization for filter examples.

pattern-matching-based methods, the proposed technique is application-type independent, and does not require the setup of a large pattern library with hundreds of patterns. The TS exploration is simple to implement and has short execution times. The paper also discusses a hierarchical, two-step parameter optimization to optimize the ac behavior and area of an architecture. Parameter optimization guides the exploration loop of architecture generation. A significant number of optimized architectures were produced for each experimented example, including some well-known solutions.

Experiments showed that linear analog systems operating at low/medium frequencies can be automatically synthesized in a reasonably long time and with reduced designer effort. Because of a linear ac performance modeling without any layout parasitics, the synthesized designs can correctly operate at low/medium frequencies up to several hundreds of kilohertz.

ACKNOWLEDGMENT

The authors would like to thank the Associate Editor and the anonymous reviewers for their very useful comments and suggestions that have helped improve this paper.

REFERENCES

- [1] *IEEE Standard VHDL Language Reference Manual (Integrated With VHDL-AMS Changes)*, IEEE Std. 1076.1.
- [2] B. Antao and A. Brodersen, "ARCHGEN: Automated synthesis of analog systems," *IEEE Trans. VLSI Syst.*, vol. 3, pp. 231–244, June 1995.
- [3] J. Cohn, D. Garrod, R. Rutenbar, and L. Carley, "KOAN/ANAGRAM II: New tools for device-level analog placement and routing," *IEEE J. Solid-State Circuits*, vol. 26, pp. 330–342, Mar. 1991.
- [4] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.
- [5] N. R. Dhanwada *et al.*, "Hierarchical constraint transformation using directed interval search for analog system synthesis," in *Proc. Design, Automation and Test Eur. Conf.*, 1999, pp. 328–335.
- [6] A. Doboli and R. Vemuri, "A VHDL-AMS compiler and architecture generator for behavioral synthesis of analog systems," in *Proc. Design, Automation and Test Eur. Conf.*, 1999, pp. 338–345.
- [7] —, "Hierarchical optimization for synthesis of linear analog systems," presented at the Int. Symp. Circuits and Systems, 2001.
- [8] —, "A regularity-based hierarchical symbolic analysis method for large-scale analog networks," *IEEE Trans. Circuits Syst. II*, vol. 48, pp. 1054–1068, 2001.
- [9] —, "A functional specification notation for co-design of mixed analog-digital systems," presented at the Design, Automation and Test Eur. Conf., 2002.
- [10] H. Fino, J. Franca, and A. S. Garcao, "Symbolic signal flow graph methods in switched-capacitor design," in *Symbolic Analysis Techniques*, F. Fernandez, A. Rodriguez-Vazquez, J. Huertas, and G. Gielen, Eds. New York: IEEE-Wiley, 1997.
- [11] J. Franks, C. Sizemore, and D. Dunn, "Ear based hearing protector/communication system," U.S. Patent 5 426 719, June 20, 1995.
- [12] S. Ganesan and R. Vemuri, "Technology mapping and retargeting for field programmable analog arrays," in *Proc. Design, Automation and Test Eur. Conf.*, 2000, pp. 58–64.
- [13] G. Gielen and R. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proc. IEEE*, vol. 88, pp. 1825–1852, Dec. 2000.
- [14] G. Gielen, "Panel: When will the analog design flow catch up with digital methodologies," presented at the Design Automation Conference, 2001.
- [15] R. Gregorian and G. Temes, *Analog MOS Integrated Circuits for Signal Processing*. New York: Wiley, 1986.
- [16] R. K. Henderson, L. Ping, and J. I. Sewell, "Analog integrated filter compilation," *Analog Integr. Circuits Signal Process.*, vol. 3, pp. 217–218, 1993.
- [17] M. Ismail and T. Fiez, *Analog VLSI. Signal and Information Processing*. New York: McGraw-Hill, 1994.
- [18] A. Konczykowska and M. Bon, "Structural synthesis and optimization of analog circuits," in *Symbolic Analysis Techniques*, F. Fernandez, A. Rodriguez-Vazquez, J. Huertas, and G. Gielen, Eds. New York: IEEE-Wiley, 1997.
- [19] J. Koza *et al.*, "Automated synthesis of analog electrical circuits by means of genetic programming," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 109–128, July 1997.
- [20] M. Krasnicki, R. Phelps, R. Rutenbar, and L. Carley, "MAELSTROM: Efficient simulation-based synthesis for custom analog cells," in *Proc. Design Automation Conf.*, 1999, pp. 945–950.
- [21] W. Kruiskamp and D. Leenaerts, "DARWIN: CMOS opamp synthesis by means of a genetic algorithm," in *Proc. Design Automation Conf.*, 1995, pp. 433–438.
- [22] M. Kucic, A. Low, and P. Hasler, "A programmable continuous-time floating-gate fourier processor," *IEEE Trans. Circuits Syst. II*, vol. 48, pp. 90–99, Jan. 2001.
- [23] K. Laker and W. Sansen, *Design of Analog Integrated Circuits and Systems*. New York: McGraw Hill, 1994.

- [24] M. del Mar Hershenson, S. Boyd, and T. Lee, "GPCAD: A tool for CMOS op-amp synthesis," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 296–303.
- [25] P. Maulik, R. Carley, and R. Rutenbar, "A Mixed-integer nonlinear programming approach to analog circuit synthesis," in *Proc. Design Automation Conf.*, 1992, pp. 698–703.
- [26] G. McCormick, *Nonlinear Programming*. New York: Wiley, 1983.
- [27] L. McIlrath, "A low-power analog correlation processor for real-time camera alignment and motion computation," *IEEE Trans. Circuits Syst.*, vol. 47, pp. 1353–1364, Dec. 2000.
- [28] F. Medeiro, A. Perez-Verdu, and A. Rodriguez-Vazquez, *Top-Down Design of High-Performance Sigma-Delta Modulators*. Norwell, MA: Kluwer, 1999.
- [29] A. Muralt, P. Zbinden, and G. Moschytz, "CAD tools for synthesis and layout of SC filters and networks," *Analog Integr. Circuits Signal Process.*, vol. 3, pp. 229–242, 1993.
- [30] E. Ochotta, R. Rutenbar, and R. Carley, "Synthesis of high-performance analog circuits in ASTRX/OBLX," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 273–294, Mar. 1996.
- [31] K. Ogata, *Modern Control Engineering*. Englewood Cliffs, NJ: Prentice-Hall, 1990.
- [32] R. Phelps, M. Krasnicki, R. Rutenbar, L. Carley, and J. Hellums, "Anacoda: Simulation-based synthesis of analog circuits via stochastic pattern search," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 703–717, June 2000.
- [33] B. N. Ray, P. P. Chaudhuri, and P. K. Nandi, "Efficient synthesis of OTA network for linear analog functions," *IEEE Trans. Computer-Aided Design*, vol. 21, pp. 517–533, May 2002.
- [34] C. Reeves, *Modern Heuristic Techniques for Combinatorial Problems*. New York: Wiley, 1993.
- [35] E. Sanchez-Sinencio and J. Ramirez-Angulo, "AROMA: An area optimized CAD program for cascade SC filter design," *IEEE Trans. Computer-Aided Design*, vol. CAS-14, pp. 296–303, 1985.
- [36] R. Stogdill, "Dealing with obsolete parts," *IEEE Des. Test Comput.*, pp. 17–25, Apr.–June 1999.
- [37] K. Su, *Analog Filters*. London, U.K.: Chapman & Hall, 1996.
- [38] J. Trontely, L. Trontelj, and G. Shenton, *Analog Digital ASIC Design*. New York: McGraw-Hill, 1989.
- [39] G. Van Der Pas *et al.*, "EsteMate: A tool for automated power and area estimation in analog top-down design and synthesis," *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 139–142, 1997.
- [40] G. Van Der Plas *et al.*, "AMGIE—A synthesis environment for CMOS analog integrated circuits," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 1037–1058, Sept. 2001.
- [41] P. Veselinovic *et al.*, "A flexible topology selection program as part of an analog synthesis system," presented at the Eur. Design and Test Conf., 1995.
- [42] R. Weyrick, *Fundamentals of Analog Computers*. Englewood Cliffs, NJ: Prentice-Hall, 1969.
- [43] J. Wu and E. El-Masry, "Design of current-mode linear filters using coupled-biquads," *IEEE Trans. Circuits Syst. II*, vol. 45, pp. 1445–1454, Nov. 1998.