A Heuristic Technique for System-Level Architecture Generation from Signal-Flow Graph Representations of Analog Systems

Alex Doboli, Nagu Dhanwada and Ranga Vemuri
Digital Design Environments Laboratory, Department of ECECS
University of Cincinnati, Cincinnati, OH 45221-0030
Email: {adoboli, nagu, ranga }@ececs.uc.edu

Abstract

This paper presents a heuristic technique for automatically generating different architectures for an analog system. The architecture generator (AG) starts from a signal-flow graph (SFG) specification of a system. The AG iteratively produces various system net-lists as distinct implementations can realize the signal processing and flow in an SFG. Area and power for resulting net-lists are rapidly evaluated with *High-Level Performance Estimator* (HPE), a simplified estimation module. The AG algorithm is simple to implement. It does not require an extensive pattern library as traditional AG techniques do.

1 Introduction

The complex task of synthesizing analog and mixed analog-digital systems is typically approached using a hierarchically organized top-down design paradigm across three successive levels of abstraction [2]: (1) *system-level*, (2) *circuit-level*, and (3) *layout-level*.

Circuit synthesis [7] [9] assumes a known circuit-topology and searches for physical dimensions of transistors so that circuit-level performances i.e. bandwidth, slew-rate, power, etc. are optimized. Most of the recent analog-circuit synthesis tools rely on *optimization* techniques [2]. They start by producing the circuit performance model that relates performance attributes to design parameters [7]. Next, the model is used by an approximate [9] or exact [2] optimization algorithm to find the physical design parameters.

As opposed to circuit synthesis, *system-synthesis* also considers functionality as a design dimension [1] [5] [8] [14]. Three main synthesis tasks are employed by a system-synthesis methodology: *architecture generation* (for exploring distinct system net-lists for the specified functionality), *constraint transformation* (from system-level to component-level) and *component selection*. Very few work concentrates on system-synthesis. [1] uses C++-like system specifications in the ARCHGEN synthesis tool. Their system relies on state-space methods for deducing net-lists in controllable, observable and diagonal forms. Traditional architecture generation (AG) methods employ pattern matching techniques [1] [4] [5]. [5] presents a synthesis flow from signal-flow graph (SFG) programs. System structures are produced by a greedy pattern-matching method that uses a library of circuits also expressed as SFGs. An exact AG method is presented in [4]. Pattern-matching based methods have a limitation in that big pattern libraries are needed for finding good quality mappings [3].

System-level design decisions, such as selecting a particular system net-list, have a great impact on the overall performances of the final implementation. By selecting a particular filter structure i.e. KHN or Tow-Thomas [12], constraints i.e. Q, ω_0, G sensitivities of the filter are basically determined. However, other performance attributes i.e. area, power, are difficult to predict due to their tight dependency on lower-level design aspects i.e. transistor dimensions, bias currents, circuit loadings. These values are exactly known only at the end of the entire synthesis flow.

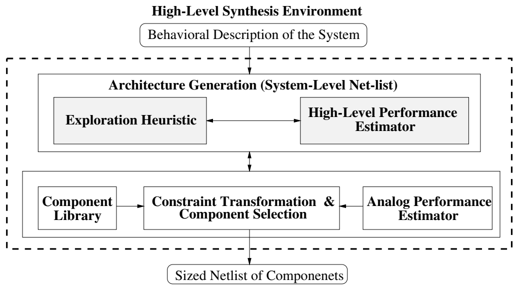

Figure 1: **Exploration based approach to behavioral synthesis**

Considering the previous arguments, we believe that any system-level synthesis task (including architecture generation) should explore as many solution points as possible. This is because none of them can be *a priori* discarded unless a design problem with a specific optimization aspect is considered. Also, for evaluating solution-points quality, a synthesis method must offer an effective way of capturing relevant low-level design/performance attributes. This concludes that any system-level synthesis task must address two distinct aspects: (1) an effective mechanism for *solution space exploration* and thus, for generating various solution points for a problem and (2) a fast and accurate method for *estimating* solution-points quality so that low-level attributes are also considered.

This paper presents a heuristic algorithm based on solution-space exploration for generating architectures for analog systems. The goal of the architecture generator (AG) is to find the net-list of analog circuits that correctly realizes the functionality of the system and satisfies imposed design/performance constraints i.e. bandwidth, power, etc. and minimizes ASIC area. The AG starts from a behavioral specification of a system described as a signal-flow graph (SFG). The SFG explicitly captures port (input/output) signals, signal flows and processing. The AG instantiates SFG signals as either currents or voltages and then, with a set of predefined rules (that link SFG structures to electronic circuits), it generates various system net-lists for an SFG. Area and power of the resulting net-lists are rapidly evaluated with an estimation module that performs gain-bandwidth trade-off analysis. The estimator uses op amp macro-models for capturing circuit-level synthesis aspects. We implemented the AG as a *tabu-search* algorithm [10] due to its deterministic nature and short execution-time. The AG algorithm is simple to implement. Besides, it does not require an extensive pattern library as traditional AG techniques do.

The paper is organized in six sections. Section 2 presents the system synthesis flow. Section 3 details the estimation module and Section 4 discusses the proposed AG method. Section 5 describes experimental results and finally, we provide conclusions.

2 Methodology for Analog System Synthesis

Figure 1 outlines the overall flow of our behavioral synthesis environment for analog systems and its liaison with the work we present in the paper. The environment accepts input specifications in the

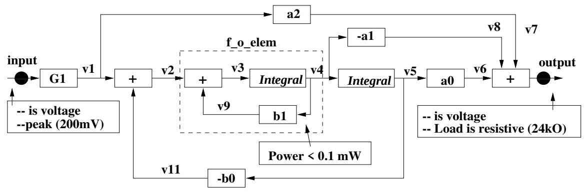

Figure 2: **Signal-flow graph for a second-order filter**

form of an SFG, together with design/performance constraints for the final implementation. Such specifications can be either manually produced or compiled from VHDL-AMS programs [4]. The *Architecture Generator* (AG) produces alternative system net-lists for the SFG. Once a net-list is fixed, the *Constraint transformation & component selection* step picks from a component library real circuit topologies for op amps and calculates physical dimensions for op amp transistors. This complex task is guided by an *Analog Performance Estimator* that computes approximate values for circuit performances. By combining existing lower-level synthesis tools with the AG discussed in this paper, a complete CAD environment for behavioral-synthesis of analog systems is realized.

VHIF (VASE Hierarchical Intermediate Format) [4] is the specification language we use for describing mixed-signal systems. It can express continuous-time behavior at the level of signal-flow graph (SFG) by explicitly indicating flows and processing (operations) of free signals (signals that do not have their type fixed as currents or voltages). For illustrating VHIF, we refer to the SFG of the example in Figure 2. The example is a second-order low-pass filter having the transfer function $H(s) = G \times \frac{a_2 s^2 + a_1 s + a_0}{s^2 + b_1 s + b_0}$ [12]. The corresponding VHIF program encodes the SFG in Figure 2. It explicitly shows the flows and operations on signals by assigning/referring variable values. In/out ports are marked with a black bubble in the figure. As shown in Figure 2, port signals can be annotated with information on their attributes i.e. type (voltage, current), value range, impedances at ports, etc. This knowledge is useful for synthesis as it enables inferring of specific input/output stages that otherwise can not be deduced from a traditional SFG.

3 Fast Estimation for Architecture Generation

Exploration heuristics i.e. simulated annealing (SA) or tabu search (TS) traverse their solution-space in the following manner. For deciding what solution-point to move to from the current solution point, the algorithm produces and analyzes the neighborhood of the current solution. Neighborhood is the set of *tentative* solution-points where each point differs from the current solution by a small change. Performance values for each of the neighborhood points must be fastly estimated and then, the point with best performances is chosen as next solution-point. Obviously, a fast performance-estimation method is needed.

This section presents the *High-Level Performance Estimator* (HPE) that finds system area and power for the AG. First, we describe our op amp macro-models and then, we present how system area and power are quickly estimated with these macro-models.

3.1 Development of Op-amp Macro-model

Our idea of capturing circuit-synthesis aspects for fast system-performance estimation is based on using op amp macro-models for area and power. These models are a realistic approximation under the next comments:

- For the AG, we assume that all op amps have the same structure as they consist of three stages. If after architecture generation, it is shown that some op amp stages are redundant, these are removed by circuit synthesis tools.
- All op amps have a fixed open-loop gain of 60 dB in our system. For having a reasonable good op amp, it is enough if its open-loop gain is beyond a certain limit. If an open-loop gain of 60 dB is not good enough in the final implementation, the successive circuit-synthesis tool will realize a higher gain.
- The loading of an op amp was fixed at 10 pF in our system. The value is sufficiently high for describing circuit-interfacing loadings.

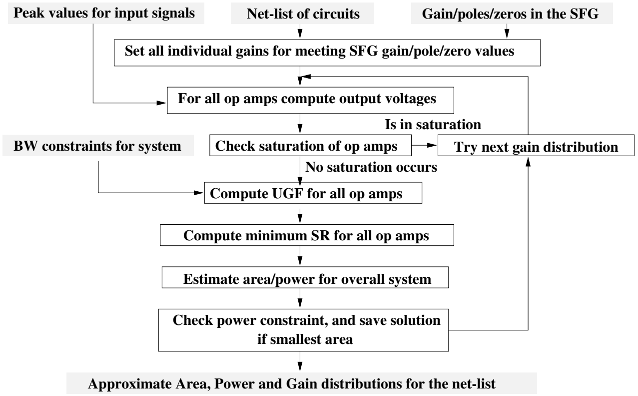

Figure 3: **High-level performance estimation procedure**

Our method for building op amp macro-models relied on a Genetic Algorithm-based circuit synthesis tool [4]. The tool uses SPICE models for calculating transistor dimensions so that the overall op amp area is minimized. We varied UGF in the range 1MHz-9MHz, and slew-rate (SR) in the range 0.8-2.2$\frac{V}{\mu s}$. We collected resulting area and power values for 90 solution-points and then applied linear regression [11] for finding the macro-models. Following relationships between op amp area, power, UGF and SR were obtained:

$$Area_{op-amptransistors} = 171.91 \times UGF^{0.2875} \times SR^{0.1688} \quad [\mu m^2]$$
$$Power_{op-amp} = 0.395027 \times SR^{0.8602} \quad [mW]$$

Table 1 compares some of the area and power values obtained by the circuit-synthesis tool with values resulting from the above models. For transistor area, *R-square* value was 0.8353 and all *p*-values were 0.0001. Similarly, for power, the resulting *R-square* was 0.8289 and all *p*-values were 0.0001. This shows that the regression models for area and power are fair enough to be used for an approximate evaluation of op amps.

3.2 High-Level Performance Estimation

HPE takes a net-list of circuits and system constraints i.e. gains, poles, signal peak values as inputs and returns approximate values for system area and power. HPE implements a set of equations for gain-bandwidth (BW) and BW-SR trade-off analyzes [6] and then uses the op amp macro-models for estimating the resulting system area and power for each *gain-BW-SR* triplet.

Figure 3 presents the global structure of our HPE algorithm. Using values for gains/poles/zeros in the SFG, peak-values for input signals and global BW constraints, the HPE finds a good gain distribution for a net-list (gain-BW trade-off exploration). For each gain distributions, output voltages of op amps are calculated. Also, depending on the net-list structure and circuit types, BW requirements for each op amp in the net-list are deduced. Next, minimum SR-s for op amps are evaluated using previously calculated BW-s and output voltages (BW-SR trade-off analysis). Finally, op amp areas and powers are estimated with the macro-model equations using the previously calculated values for op amp UGF-s and SR-s. The HPE also checks that no op amp output saturates.

Gain distribution and thus, gain-BW trade-off exploration, is one of the traditional activities in analog design [6]. By properly distributing overall gains to the constituent circuits of a net-list, smaller resistor areas and better UGF constraints can outcome for individual circuits. The latter can lead to smaller op amp areas.

Example	Gain (dB)	UGF (MHz)	Transistor Area			Power			Resulting SR
			Synthesis	Macro-model	Error (%)	Synthesis	Macro-model	Error (%)	($\frac{V}{\mu s}$)
Op amp1	1062	1.51	212.0	199.01	-6.1	0.5051	0.4554	-9.97	1.180
Op amp2	1073	3.07	255.8	247.03	-3.1	0.5832	0.4839	-17.01	1.266
Op amp4	1063	6.04	290.2	305.44	5.2	0.4860	0.5302	9.09	1.408
Op amp6	1064	8.51	365.3	347.42	-4.8	0.5510	0.6184	12.23	1.684

Table 1: **Evaluation of Op amp macro-models**

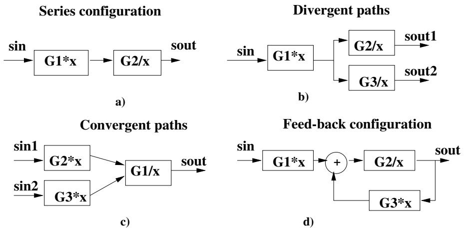

Figure 4: **Rules for gain distribution**

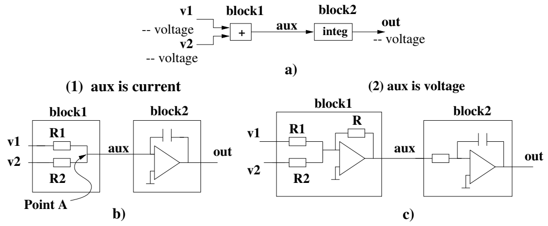

Figure 5: **Example of a conversion rule**

Gains, poles and zeros remain unchanged during this step. HPE traverses the net-list and tries different gain allocations depending on the net-list structure. Figure 4 depicts how gain-distribution takes place for distinct structures. If the first circuit gets its gain multiplied by *x*, the next circuit will have its gain divided by *x*.

Without losing generality, we assume that a system specification introduces all 3-dB bandwidth requirements for non-feedback paths of an SFG (for a feed-back structure, 3-dB bandwidths for both directed and feed-back paths result from pole/zero placement of the global system). Knowing the 3-dB bandwidth for a non-feedback sequence of *n* circuits, the 3-dB bandwidth for each circuit can be directly calculated with the formula:

$$3dB-frequency_{circuit} = 3dB-frequency_{overall} \times \sqrt{2^{\frac{1}{n}} - 1}$$

This formula captures cumulative influences of circuits in a sequence. It holds if all circuits have the same 3-dB frequency. However, there are no reasons for having a sequence of circuits with distinct 3-dB frequencies as the overall frequency is set by the smaller frequencies.

If the 3-dB frequency and the gain of a circuit are known then the UGF of its op amp can be directly calculated (we assumed the dominant-pole model for op amps) [6]. Following equations are used at this step: $Area_{ninv} = Area_{opamp}(UGF, SR) + Area_{Rmin} \times (Gain + 1)$ for a non-inverting amplifier, $Area_{adder} = Area_{opamp}(UGF, SR) + Area_{Rmin} \times (Gain + nrinputs)$ for summing amplifier, or $Area_{integrator} = Area_{opamp}(UGF, SR) + 2 \times \sqrt{\frac{Area_{Rmin} \times Area_{Cmin}}{Gain \times BW}}$ for an integrator. We derived the last formula so that circuit area is minimal for a fixed $\frac{1}{RC}$ value. Finally, knowing the circuit gain (*Gain*), along with *UGF* and *SR* of its op amp, circuit area and power can be evaluated.

4 Exploration Heuristic for Architecture Generation

The system-level architecture generator (AG) produces distinct net-lists of circuits that realize the functionality of an SFG. We described the AG as the problem of searching for feasible realizations of SFG signals as currents or voltages. If input and output signals of an VHIF block have their type fixed (as either currents or voltages) then also the network (RC network or circuit) that realizes the block operation can be determined. Basically, *in its exploration-loop, the AG incrementally changes signal types from voltage to current and vice-versa, applies a set of conversion rules to generate corresponding net-lists and analyzes with HPE the net-list quality.*

4.1 Conversion Rules for Architecture Generation

If all input/output signals of a VHIF block have known types (currents or voltages) then a *conversion rule* indicates the corresponding electronic circuit for implementing the operation of the block. By electronic circuit we mean either an op amp based circuit i.e. gain stage, summing amplifier, etc. or an R or RC network.

Figure 5 exemplifies two of our conversion rules and illustrates how two distinct net-lists result for the simple SFG depicted in Figure 5a. We assume that inputs *v1* and *v2* and output *out* are voltages. Inferring net-lists 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 by applying conversion rules for the specific signal types and block operations, the structure in Figure 5b is obtained. 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 requirement of point *A* being virtual ground is satisfied by *block 2*. (2) If signal *aux* is a voltage, the net-list in Figure 5c results. For these signal types, *block1* has to be a summing amplifier and *block 2* an integrator. No special constraints are needed i.e. point *A* being virtual ground.

Table 2 exemplifies some of our conversion rules as they were embedded in our architecture generator. The first four columns indicate the operation of a VHIF block and the types of block inputs and outputs. Column five describes required conditions for connecting the circuit output i.e. the output must be connected to ground. Column six presents circuit characteristics useful for the AG (i.e. inputs with virtual ground) that can be exploited for circuit interconnections. Finally, the last column presents the corresponding circuit for a block.

4.2 Architecture Generation Algorithm

The exploration strategy embedded in the AG is tabu-search (TS) [10]. We decided to use TS because it has short convergence time and also, it is a deterministic algorithm.

The pseudo-code of the TS algorithm is depicted in Figure 6. It starts from an initial net-list solution, where each of the VHIF blocks is mapped to a distinct op amp based circuit. Then, for each iteration, routine *Find_Best_Move* (line 5 in Figure 6) individually considers all signals in the SFG and attempts to change their type to the opposite one i.e. if a signal is a current than 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 net-list results by applying appropriate conversion rules for the new situation. Next, performances of the resulting net-list are rapidly estimated with HPE. The signal offering the best performance improvement is returned. Routine *Execute_Move* (line 10) performs the best signal-type change and the resulting net-list 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. 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. Escaping from local optima is achieved by accepting degrading moves,

Block Operation	Input Types		Output Type	Interconnection Constraints	Circuit Characteristics.	Circuit
	Inp1	Inp2				
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

Table 2: **Conversion rules for architecture generation**

```
procedure NetListGenerator () is begin
(1)     call Produce_Initial_NetList ();
(2)     call HPE and find performance attributes of solution;
(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_Criteria ();
(8)         if current_signal is NULL then
(9)             current_signal = Find_Closest_Tabu_Expiration ();
(10)        current solution = call Execute_Move ();
(11)        call HPE and find performance attributes of new solution;
(12)        if (current solution is best so far) then
(13)            save current solution as best solution;
(14)    end while;
end procedure
```

Figure 6: **Tabu-search for architecture generation**

also. However, the algorithm attempts to keep the search locally by selecting the move that previously was improving and offers the least performance degradation (routine *Aspiration_Criteria* at line 7). If still no signal is selected, the signal closest to its tabu status expiration is chosen (routine *Find_Closest_Tabu_Expiration* in line 9).

5 Synthesis Results

The AG was experimented with a large set of examples. Due to space limitations, we presented in the paper only results for two small examples. We studied (1) the correctness and variety of net-lists produced by the AG and (2) the impact of gain-BW trade-off analysis on choosing a particular net-list for an SFG. The examples are a 2nd order filter [12] and a telephone receiver module [13].

Filters are multiple feed-back system with a higher number of VHIF blocks and signals. Thus, a more elaborate architecture exploration results. The filter has the transfer function $H(s) = \frac{100.0}{s^2+1.41421s+1.0}$. For architecture generation we used the controllable SFG, similar to the one depicted in Figure 2.

We run the AG for overall BW constraints of 20kHz, 200kHz and 1MHz. Synthesis results (estimated area) are depicted in Figure 7. Eight distinct net-lists were produced, including the well-known KHN and Tow-Thomas structures [12]. Minimal area resulted for Tow-Thomas structure in all three cases. Execution times were 0.091 sec on a SUN Sparc5 workstation.

The receiver module is a non-feedback example. We used it to show how distinct net-lists enable better gain distributions. The main functionality of the receiver is to amplify with gains 10.0, 1.0, and 100.0 three incoming signals and then add them together. The experiment was conducted for an overall BW constraint of 20kHz. Synthesis results (estimated area) are depicted in Figure 7. Twelve distinct net-lists were produced and execution time was 0.05 sec. The best net-list had a gain stage and a summing amplifier and their allocated gains were 11.16 and 12.5.

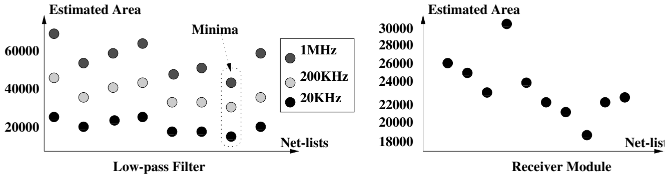

Figure 7: **Synthesis experiments**

By experimenting with the AG, we learned that it is beneficial to simultaneously address architecture generation and gain-BW trade-off analysis. The smallest area for a net-list results only after gain distribution is done. Besides, there is no way of directly selecting the best net-list without performing exploration. For example, a net-list with more op amps can consume more op amp area but resistor area is less due to a better gain distribution. Also, for resistive net-lists, the best gain distribution did not depend on the overall system BW. For filters, at low frequencies, gains also tend to be distributed among integrators and gain stages. At higher frequencies, however, gains are mainly allocated to integrators. As integrators already have a high $\frac{1}{RC}$ due to BW constraints, increasing their gain will less affect their RC area.

6 Conclusions

This paper presents a tabu-search heuristic for producing architectures for analog systems specified as SFGs. The goal is to find the architecture (system net-list) that correctly realizes the system functionality, satisfies imposed system constraints and minimizes ASIC area. The AG iteratively instantiates SFG signal types as either currents or voltages and then, it produces various system net-lists by using a set of pre-defined conversion rules. The AG algorithm is simple to implement. It does not require an extensive pattern library as traditional AG techniques do. Area and power for resulting net-list are rapidly evaluated with our HPE module. It targets gain-BW trade-off analysis as it is important for improving system performance. The AG can be improved by also restructuring SFGs to find better architectures. Ongoing work also attempts to extend the HPE by addressing switching noise during analog synthesis.

References

[1] B. Antao, A. Brodersen, "ARCHGEN: Automated Synthesis of Analog Systems", *IEEE Trans. on VLSI Systems*, vol.3, no.2, pp.231-244, June 1995.

[2] L.R. Carley *et al*, "Synthesis Tools for Mixed-Signal ICs: Progress on Frontend and Backend Strategies", Proc. of 33rd DAC, pp.298-303, 1996.

[3] G. De Micheli, "Synthesis and Optimization of Digital Circuits", McGraw-Hill, 1994.

[4] A. Doboli *et al*, "Behavioral Synthesis of Analog Systems using Two-Layered Design Space Exploration", *Proc. of 36th DAC*, 1999.

[5] H. Fino *et al*, "Symbolic Signal Flow Graph Methods in Switched-Capacitor Design", in *Symbolic Analysis Techniques*, editors: F. Fernandez *et al*, IEEE Press, 1998.

[6] S. Franco, "Design with Operational Amplifiers and Analog Integrated Circuits", McGraw Hill, 1998.

[7] G. Gielen *et al*, "Analog Circuit Design Optimization Based on Symbolic Simulation and Simulated Annealing", *IEEE Trans. on Solid-State Circuits*, vol.25, no.3, pp.707-713, June 1990.

[8] A. Konczykowska *et al*, "Structural Synthesis and Optimization of Analog Circuits", in *Symbolic Analysis Techniques*, editors: F. Fernandez *et al*, IEEE Press, 1998.

[9] E. Ochotta *et al*, "ASTRX/OBLX: Tools for Rapid Synthesis of High-Performance Analog Circuits", *Proc. of 31st DAC*, pp.24-30, 1994.

[10] C. Reeves, "Modern Heuristic Techniques for Combinatorial Problems", J. Wiley, 1993.

[11] S. Ross, "Introduction to Probability and Statistics for Engineers and Scientists", J. Wiley, 1987.

[12] K. Su, "Analog Filters", Chapman & Hill, 1996.

[13] J. Trontely, L. Trontelj, G. Shenton, "Analog Digital ASIC Design", McGraw-Hill, 1989.

[14] P. Veselinovic *et al*, "A Flexible Topology Selection Program as Part of an Analog Synthesis System", *Proc. of the ED & TC*, 1995.