Technology Mapping and Retargeting for Field-Programmable Analog Arrays *

Sree Ganesan and Ranga Vemuri

Department of ECECS, ML0030
University of Cincinnati, Cincinnati, OH 45221.
{sganesan,ranga}@ececs.uc.edu

Abstract

Rapid prototyping followed by technology retargeting provides a fast and cost-effective approach to analog system synthesis. Field-programmable analog arrays (FPAAs) enable rapid implementation of a function-compliant prototype, while technology retargeting converts the functional FPAA prototype to an ASIC. We first address the FPAA technology mapping problem. A novel structural approach based on hierarchical pattern matching and covering is employed to map the analog behavior onto the FPAA. We then address issues of technology retargeting and design reuse, and present our FPAA-ASIC retargeting strategy. We present experiments and a design example for FPAA technology mapping and retargeting.

1 Introduction

The VHDL-AMS Synthesis Environment (VASE) being developed at the University of Cincinnati performs synthesis of analog and mixed signal designs. The first step is architecture generation [1]. It produces different system topologies for the given specification. This is followed by component selection and constraint transformation. They involve selection of ASIC components from the library and propagation of system-level constraints to the component-level. For every system topology, transistor sizing is done to determine if the system performance constraints are satisfied. Using the traditional prototype fabrication method to validate the design is both time-consuming and expensive. We propose a fast and cost-effective approach to analog system synthesis in VASE, employing rapid prototyping and retargeting. Rapid prototyping using field-programmable analog arrays (FPAAs) [2, 3, 4] produces a *function-compliant* hardware prototype of the analog system. The retargeting phase rehosts the functional design into an ASIC. Figure 1 shows the design flow. In this paper, we focus on FPAA technology mapping and retargeting of FPAA designs to ASIC.

The rest of this paper is organized as follows. Section 2 gives an overview of FPAA technology and its limitations. We present a method to overcome the FPAA bandwidth limitation for prototyping. Section 3 addresses FPAA technology mapping and presents our algorithm. Section 4 outlines issues in FPAA-ASIC retargeting, and describes our strategy. Section 5 presents experiments and a design example, while Section 6 gives the concluding remarks.

*This work is sponsored by AFRL, Wright Patterson Air Force Base under contract number F33615-96-C-1911

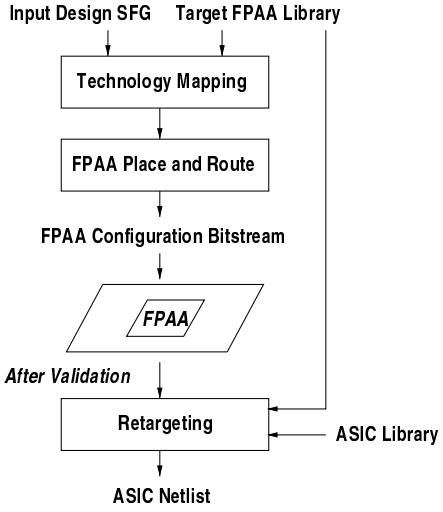

Figure 1. Design Flow

2 FPAA Technology

Field-programmable analog arrays are a family of integrated circuits composed of configurable analog blocks (CABs) and programmable interconnect, that can be configured to implement analog circuits. FPAAs belong to either continuous-time or discrete-time domains. Discrete-time designs are based on switched-capacitor or switched-current technology. Their advantages include wide range of programmability and insensitivity to parasitics of programming switches. The disadvantage is that maximum signal frequencies are limited by the maximum sampling clock frequency. Continuous-time FPAAs are typically designed using transconductors. They have higher bandwidth, but the programmable parameter range is smaller. Voltage-mode designs have limited signal swing while current-mode designs have unlimited signal swing. The Motorola MPAA [2] and IMP EPAC [3] are discrete-time FPAAs with bandwidths of 200kHz and 150kHz respectively, while the Zetex TRAC [4] is a continuous-time FPAA with a 4MHz bandwidth.

Irrespective of whether the FPAA is continuous-time or discrete-time, the programmable parameter range and the bandwidth pose constraints on the circuits realizable on the FPAA. The FPAA technology mapping phase (discussed in Section 3) addresses the former limitation. A method to overcome the FPAA bandwidth limitation is presented next.

2.1 Overcoming the FPAA Bandwidth Limitation

Analog systems that work at frequencies higher than the FPAA bandwidth cannot be implemented on the FPAA. In or-

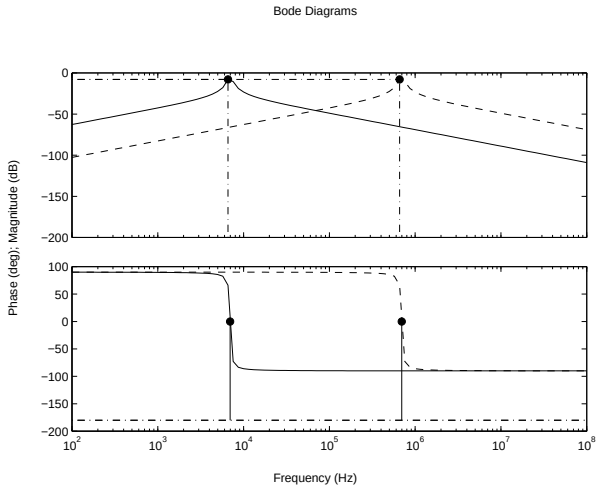

Figure 2. Frequency Scaling

der to overcome this limitation and use the FPAA for prototyping such systems, we propose a strategy based on *frequency scaling*.

Consider a linear network η_1 with behavior specified by transfer function $T_1(s)$. Now we construct another network η_2 with the same topology as η_1. Let its behavior be specified by the transfer function $T_2(s)$ such that

$$T_2(s) = T_1(s/w_s) \tag{1}$$

The factor w_s is referred to as the frequency scaling factor. The implication of Equation 1 is that frequency response of the two networks remains unchanged except that the frequency axis is scaled down for network η_2.

For instance, consider second order Chebyshev bandpass biquad filter that has the following transfer function:

$$T(s) = \frac{1.167e-06s}{1+1.167e-06s+5.214e-10s^2} \tag{2}$$

This design cannot be implemented on the Motorola FPAA [2] due to bandwidth limitation of 200kHz. With a frequency scaling factor w_s = 100, center frequencies before and after are 697kHz and 6.97kHz respectively. Shown in Figure 2 is the frequency response of the bandpass filter before (dashed line) and after (solid line) frequency scaling.

Since the aim of technology mapping is to produce a functional prototype, the high-frequency system transfer function is scaled down appropriately such that the scaled response is achievable on the target FPAA. After this, the frequency-scaled system is prototyped to determine if the circuit has the required functionality. To ensure that the high-frequency system also has the same functionality as well as desired performance, it is important to use a topology designed specifically for the high-frequency system.

3 Library Based FPAA Technology Mapping

Coarse-granularity FPAAs are suitable candidates for system-level design since their CABs can be configured as functional building blocks or macros. Technology mapping transforms the given technology-independent netlist into a netlist of library macros of the target FPAA technology. The approach is to select subnetworks of the input netlist that can each be implemented by one of the available library components, while optimizing the circuit to minimize a cost function that typically incorporates measures of area and performance. This is called *network covering* by library cells.

3.1 System Representation

All linear time-invariant systems can be built using three basic functional elements: Integrator, Gain Amplifier and Summing Amplifier [6]. This set of basic elements is functionally complete. Hence any linear time-invariant system may be represented by a block-diagram or a signal-flow graph (SFG) composed of these basic elements [6]. For nonlinear systems, the set of basic elements also includes elements corresponding to the nonlinear function blocks such as Absolute Value, Maximum, Minimum, Log, Antilog, Greater or Lesser Than, and so on.

For FPAA technology mapping, a list of parameters required to achieve the desired functionality is associated with each basic element. For instance, the Gain element will have a "gain" parameter that determines the actual amplification between the input and output of the element.

The given analog system is specified as a connected network of behavioral blocks. In other words, a signal flow graph representation of the analog system composed of transfer function blocks, nonlinear function blocks and summing nodes. Generation of a structure from the specified behavior is achieved by realizing each of the transfer function blocks using state-space methods [5]. We refer to this SFG as the *design SFG*.

3.2 Library Representation

Each component of the target FPAA library is likewise represented in all possible ways using the set of basic functional elements. This generates a set of *pattern SFGs*. Associated with each pattern SFG is a cost, which in our case is the area (number of CABs) occupied on the FPAA. The underlying assumption of the approach is that the nodes of the SFG do not have any coupling effects. This is a valid assumption for a library composed of isolated components, for instance opamp-level designs where the input impedance is large and output impedance small. The target FPAA library must satisfy this requirement.

3.3 Strategy

The objective of the FPAA library-binding problem is to find a minimum cost covering of the design SFG using pattern SFGs of library components, and ensure that programmable parameters of the FPAA components are within legal ranges. The design SFG is a directed graph, and the problem of finding an optimal cover is NP-complete. Efficient algorithms for using dynamic programming have been developed for tree circuits. One of the techniques used for DAG covering is approximating it by a number of tree covering problems by partitioning it into a forest of trees [7]. Our approach is outlined in Figure 3. Procedure convert_SFG_to_forest() transforms the design SFG from a graph into a forest of SFG trees F by disconnecting all edges at each of the output nodes feeding

```
procedure lib_bind()
Inputs: Design SFG d_sfg, Target Library lib
Output: Best Covering best_sfg_cover
begin
    F : forest of SFG trees
    F ← convert_SFG_to_forest(d_sfg);
    best_sfg_cover ← null;
    foreach τ ∈ F
        C_τ : set of coverings of tree τ
        C_τ ← cover(τ);
        best_tree_cover ← null;
        foreach c ∈ C_τ
            c ← sum_expansion(c);
            c ← gain_chaining(c);
            if cost(c) < cost(best_tree_cover) then
                best_tree_cover ← c;
            end if
        end for
    best_sfg_cover ← best_sfg_cover ∪ best_tree_cover;
    end for
end
procedure hierarchical_bind()
Inputs: Design SFG d_sfg,
    Target Multi-level Library lib_0, lib_1, ..., lib_k
Output: Best Covering best_cover
begin
    g ← d_sfg;
    best_cover ← null;
    for lib = lib_0 to lib_k
        g ← lib_bind(g, lib);
        if cost(g) < cost(best_cover) then
            best_cover ← g;
        end if
    end for
end
```

Figure 3. Library Binding Algorithm

multiple inputs. Thereby feedback loops as well as feedforward paths are converted into branches of trees in the forest.

There are two restrictions on the library: (1) We do not handle multi-output components, and hence the library is restricted to only single-output components i.e. the patterns must have exactly one root node. (2) The pattern SFGs of library components are directed graphs that may contain cycles. Since we perform tree covering, the patterns must be acyclic. In other words, every component must be represented using a *single* pattern tree. This restriction implies that the component corresponding to the pattern may not exhibit fanout except at the primary inputs and output. Only feedback loops from the primary output to intermediate nodes are permissible.

A *cover* is a collection of pattern graphs such that every node of an SFG tree, $\tau \in F$ is contained in exactly one of the pattern graphs. In addition, each input used by a pattern graph must be an output of another pattern graph. Procedure cover() produces the set of possible coverings, $\{C_\tau\}$ implementing each SFG tree by means of *dynamic programming traversal* from the leaf to the root node. In order to detect output feedback, the matching algorithm must also determine if the vertex and corresponding leaf node(s) match. We then perform *sum-expansion* on each covering $c \in \{C_\tau\}$. Summing amplifiers in FPAA library have a fixed number of inputs, say k. For instance, the MPAA [2] has a 2-input inverting summer. The design SFG may have summing nodes with greater than k inputs, and the sum_expansion() step replaces those nodes with an equivalent tree of k-input summing nodes.

Once a match has been found, the parameters of the basic elements are used to determine the programmable parameter values for the FPAA component corresponding to the match. The next step is to verify whether legal values are obtained for the programmable parameters. Except for the gain parameter, we determined that other parameters such as Q-factor, corner frequency (for filter components) and so on are not "flexible". Hence, the component is discarded if there is violation of the parameter bounds. In the case of violation of gain parameter bound, we employ *gain-chaining*. This step achieves the required gain, G by cascading the minimum number, n of legal gains, g such that

$$G = g^n$$

$$\textbf{if } G < g_{min} \quad \textbf{then } n = \lceil \frac{log(G)}{log(g_{min})} \rceil$$

$$\textbf{if } G > g_{max} \quad \textbf{then } n = \lceil \frac{log(G)}{log(g_{max})} \rceil$$

where g_{min} and g_{max} are the minimum and maximum permissible gain parameter values respectively.

Next the minimum area covering, *best_tree_cover* of the SFG trees is obtained. The minimum area covering of the design SFG, *best_sfg_cover* is obtained by putting together the best coverings of the SFG trees and replacing the broken edges at fanout nodes.

3.4 Hierarchical Pattern Matching and Covering

One of the limitations of the *tree* covering approach is the restriction on the pattern library that every component must be representable as a single pattern tree composed of basic functional elements. This greatly restricts the representation of components with feedback. Consider a bandpass biquad filter. Its controllable form representation results in two pattern trees, as shown in Figure 4. Even though the biquad component exists in the library, the occurrence of the former structure in the SFG tree will fail to show a match. [1]

Removal of the restriction on the pattern library can result in a pattern now represented by a forest of trees. Determining a match with such a pattern forest tends to increase the complexity of the algorithm. Instead of incurring this overhead, the remedy we propose is to have a *multi-level pattern library*. Level 1 patterns are composed of basic elements or level 0 patterns. Level 2 patterns are composed of level 1 patterns and basic elements, and so on. As shown in Figure

[1] Unlike the analog domain, this is not of much concern in the digital domain. Even though digital circuits are often sequential and hierarchical in nature, most studied binding problems deal with their combinational components because choice of implementation of registers, inputs/output circuits and drivers is often straightforward and binding is done by direct replacement [8]. As for combinational portions of the circuit, there is little or no feedback exhibited. With AND2/INV decomposition or NAND2 decomposition, only gates such as EXOR and EXNOR exhibit feedback. A match with these patterns exists if the vertex and corresponding leaves match [9].

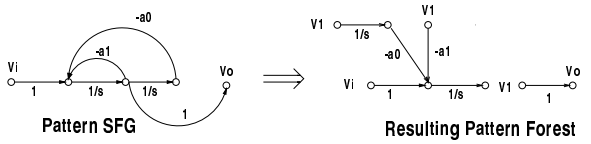

Figure 4. Forest of Patterns

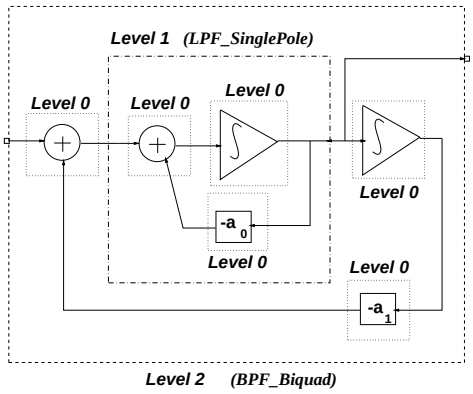

Figure 5. Multi-Level Pattern

5, the bandpass biquad filter can be represented as a level 2 pattern. It is composed of one level 1 pattern (single-pole lowpass) and rest are level 0 patterns (gain, integrator and summing amplifier). Such a hierarchical representation of the library is natural and well-suited to system design since components may be specified across several levels of abstraction.

The library-binding now involves partitioning into forest of trees, followed by hierarchical pattern matching and covering. Suppose the multi-level pattern library has k+1 levels. First the network is covered using level 0 patterns from lib_0. Then patterns of levels 1,2,...,k from lib_1, lib_2, ... and lib_k respectively are used. At each level, the best covering solution obtained is chosen as input to the next level. The best covering encountered among all levels, *best_cover* is returned. The procedure is outlined in Figure 3. One of the advantages of this approach is that if the network cannot be covered using lib_i, it might be possible to find a covering solution using a higher-level library, lib_j (j>i) since the latter contains patterns composed from level 0 to level j-1.

The cost of a library component is less than the sum of the costs of the constituent components used to implement the same function. In such a situation, every match using a higher-level library results in lowering the cost of the current solution. Thus solution quality with a multi-level library will always be better than that obtained with a single pattern library.

Limitations: As with any library-based technique, the size of the pattern library is a concern here. Also, since we consider only single-output components, library components with multiple outputs cannot be utilized. Fanout nodes remain as "observable" nodes of the network since we do not perform covering across tree boundaries. These factors tend to affect the optimality of the covering solution.

4 Retargeting to ASIC

FPAAs are economical when used for the initial prototype and limited production. Nevertheless there are several advantages of an ASIC solution: it allows the use of a smaller package with lesser number of pins, and reduces power consumption. Also, FPAA limitations do not apply to ASIC solution. There is wider range for parameter values, and it is possible to achieve higher bandwidth.

There are two approaches to produce the ASIC solution: (1) re-synthesize the system for ASIC technology, and (2) retargeting the synthesized FPAA design to an ASIC. The former requires a system-level analog synthesis tool and produces a functional equivalent of the FPAA design. The component selection and constraint transformation step in the synthesis process is selects components from the ASIC library and propagates system-level constraints to the component-level. This produces a system-level functional equivalent of the FPAA design. In order to ensure that the desired functionality is obtained, it is essential to incorporate all the design changes made during the rapid prototyping runs.

Since a component-level netlist for the system is already available, retargeting re-uses the existing FPAA component netlist. This saves the time that the synthesis process spends on the search for components and their parameters. Every FPAA component is mapped to a ASIC component that has same functionality. ASIC component parameters are derived from the FPAA component parameters. Thus retargeting produces an ASIC netlist of component-level functional equivalents.

4.1 Representation

The FPAA component has a $pinlist_{fpaa}$, the list of input and output pins of the component. Associated with the component is a behavioral model that captures the terminal behavior. Finally, $parlist_{fpaa}$ consists of the list of programmable parameters and the component bandwidth. The ASIC component interface, namely input and output, is identified by the $pinlist_{asic}$. Associated with the ASIC component is its analytical performance model comprising of performance composition equations. The performance parameters of the model constitute the $parlist_{asic}$.

For instance, consider a non-inverting amplifier. The $parlist_{fpaa}$ contains gain g and bandwidth bw. The $parlist_{asic}$ includes gain G, bandwidth B, bias current I_{bias}, openloop gain A_{dm}, output impedance R_{out}, and gain-width-length product, GWL.

4.2 Library Characterization

In this step, FPAA component bandwidths are determined (either from data sheets provided by the manufacturer or by experimental methods). The $parlist_{fpaa}$ of each FPAA component is updated with the bandwidth found. Unlike the ASIC library, such a characterization is possible for the FPAA library because the components are fabricated and the device technology is known.

4.3 Library Mapping

This involves obtaining a mapping, *libmap* between FPAA components and ASIC components. The FPAA component is identified by the terminal behavior. It is possible to have several ASIC components each with a different topology implementing the same behavior as the FPAA component. Without second order effects such as sensitivity, distortion, noise, offset and so on, the behavioral model cannot be used in selecting a specific topology. Hence one-to-many mappings can now occur. If FPAA component's behavior has no information about second order effects, there are two ways to choose the ASIC component: (1) employ the designer's knowledge to guide the selection, or else (2) select any one of the ASIC components.

Thus $libmap$ is a one-to-one relation between the set of FPAA components and ASIC components. For each FPAA component in the domain of $libmap$, there is exactly one corresponding ASIC component in the range of $libmap$.

In addition to identifying the FPAA-ASIC component map, pin mapping and parameter mapping are also performed. Pin mapping may be one-to-one or one-to-many or many-to-one. If there are pins in $pinlist_{asic}$ that remain unmapped, the ASIC component's analytical model determines whether these pins are tied to Vdd or Gnd. Next, each parameter of $parlist_{fpaa}$ is mapped to a parameter of $parlist_{asic}$. There will be unmapped parameters of $parlist_{asic}$ that correspond to the design parameters of the component (these are determined during netlist generation phase).

For the noninverting amplifier, the library mapping sets up the map between the FPAA noninverting amplifier, and the ASIC noninverting amplifier. Parameter mapping sets up the map between *g* and *bw* of the former to *G* and *B* of the latter. For a given target FPAA library, library mapping and characterization are both one-time design efforts. The rest of the steps in retargeting are discussed below. Figure 6 illustrates the retargeting procedure.

4.4 Netlist Translation/ Constraint Generation

First each FPAA component, *fpaa_comp* is replaced with the ASIC equivalent, *asic_comp* obtained from the library mapping. The netlist generation phase then determines the unmapped parameters of $parlist_{asic}$.

The ASIC component's performance parameter constraints are of two types: *equality* and *inequality constraints*. The former implies that the exact parameter value is required to satisfy the constraint, while the latter implies a lower or upper bound on the parameter value.

The programmable parameter values such as gain, cut-off frequency and so on of the FPAA components are the set of *equality constraint* parameters, C_{set}. The primary *inequality constraint* parameter is the bandwidth lower bound, B_{min}. The FPAA component bandwidth obtained from the FPAA library characterization is set as the lower bound for the ASIC component. Using such a pessimistic lower bound bandwidth constraint can lead to an overhead in the area of the ASIC design.

procedure *retarget()*
Inputs: FPAA netlist fpaa_ckt, Library Mapping lib_map
Output: ASIC netlist asic_ckt
begin
  foreach *fpaa_comp* ∈ *fpaa_ckt*
    asic_comp ← *libmap[fpaa_comp];*
    C_{set} *: equality constraint parameter set*
    B_{min} *: bandwidth lower bound constraint*
    DP_{set} *: design parameter set*
    C_{set} ← *get_constraints(fpaa_comp);*
    B_{min} ← *get_bandwidth(fpaa_comp);*
    $parlist_{asic} \leftarrow \{ C_{set}, B_{min}, DP_{set} \}$
    asic_ckt ← *asic_ckt* ∪ *asic_comp;*
  end for
  asic_ckt ← *optimize_area(asic_ckt);*
  foreach *asic_comp* ∈ *asic_ckt*
    asic_comp ← *optimize_component(asic_comp);*
  end for
end

Figure 6. Retargeting Procedure

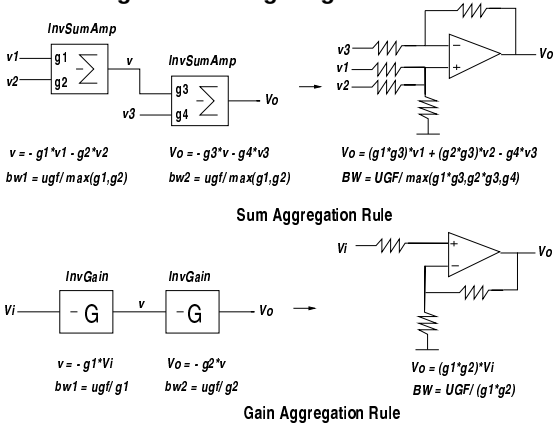

Figure 7. Area Optimization

4.5 Area Optimization

This step involves application of rewriting rules to improve the area of the design. Procedure optimize_area() achieves this by performing *summing-amplifier aggregation* and *gain aggregation*. Blind application of these rewriting rules can result in loss of bandwidth. Hence, we compare the bandwidths before and after the rewriting rule is applied. If there is improvement in bandwidth, the post-rewrite solution is accepted. If there is bandwidth degradation, the post-rewrite solution is accepted if the degradation is within a specified bound.

Sum Aggregation: In the FPAA, there will be a limit on the number of inputs to an adder. In the Motorola FPAA, only 2-input inverting summing amplifiers are available. This limitation does not exist for the ASIC component. In the FPAA, trees of summing nodes will be employed to achieve a k-input add-subtract function. The *sum aggregation* rules identify such summing node trees and replace them with a single adder-subtracter node.

Figure 7 illustrates the rule for combining two Motorola FPAA InvSumAmp (inverting summing amplifier) compo-

nents to produce a 3-input adder-subtracter. The bandwidths of the InvSumAmp and the ASIC adder-subtracter are *bw1*, *bw2* and *BW* respectively. The equivalent bandwidth, bw_{eq} of the 2 InvSumAmps in cascade is *(bw1*bw2)/(bw1+bw2)*. Hence, the rule is applied if $BW > bw_{eq}$ or $BW - bw_{eq} < bound$. Similar rules are written for various summing node trees.

Gain Aggregation: It is possible that the FPAA components may not satisfy all specified requirements. For instance, a gain of 100 may be specified in a design. But the FPAA component's gain must lie within a specified range, for instance [0.001-20] in the Motorola FPAA library. The prototype will be built with a cascade of two gain stages, say x10 followed by x10, while the required gain of 100 can be obtained on the ASIC using a single gain stage during retargeting. This is the basis of the *gain aggregation* rule.

4.6 Component Optimization

A component optimization engine [10] then employs the ASIC component's analytical performance model and determines the unmapped parameters of the $parlist_{asic}$ such that the C_{set} and B_{min} constraints are satisfied. Thus, for the non-inverting amplifier, the optimization engine determines I_{bias}, A_{dm}, R_{out} and GWL such that the desired G and B are obtained.

Limitations: To save time and computing effort, we have resorted to a library of predefined ASIC components. The consequence is that resulting circuits may not be optimal in terms of area and performance. This is a drawback when the main goal is high performance and not fast access to silicon.

5 Experiments

First we studied the performance of the hierarchical pattern matching and covering algorithm. Randomly generated SFGs were provided as input. Figure 8 shows the behavior of the algorithm: improvement in solution quality with the library levels. This is because the cost of a component is smaller that the cost of its constituent patterns. Hence, any match from a higher level library lowers the cost. Figure 9 shows the execution time versus number of levels. We observe that the curves show a linear increase in time with the number of levels. The complexity of covering directed graphs with cycles is controlled by formulating it as a set of tree covering problems (Note: the complexity of tree covering is linear in both the number of nodes and number of patterns.)

A second set of experiments were conducted to study FPAA technology mapping for various input structures of linear systems, as well as the effect of programmable parameter range. The Motorola FPAA [2] was used for these experiments. The following state-space structures were studied: (1) block, (2) observable, (3) controllable, (4) observable staircase, (5) controllable staircase, and (6) cascade form. The Motorola FPAA hierarchical pattern library is composed of 2 levels: Level 1 (L1) includes integrators, gain and summing amplifiers and single-pole low pass filter, and level 2 (L2) consists of biquad filter components.

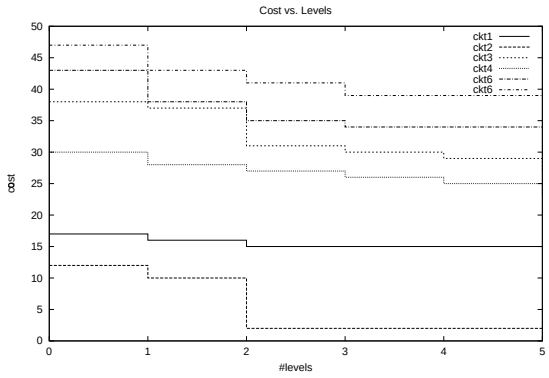

Figure 8. Cost vs. # Levels

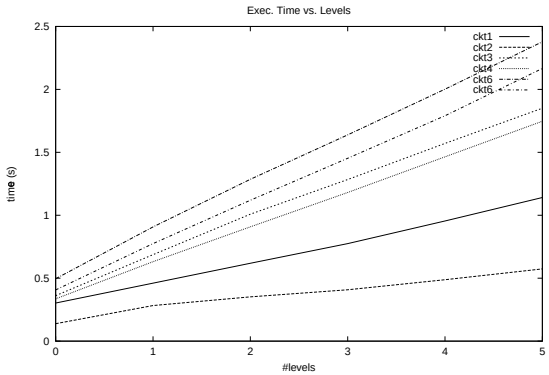

Figure 9. Execution Time vs. # Levels

For the structures (1), (2), (4) and (5), only L1 patterns are used. In structure (3), L2 patterns are used but insignificant in number. For cascade structure (6), several matches were L2 patterns. Since the input is a series of biquad filters and a single-pole/bilinear filter, cascade forms lend themselves well to hierarchical representation. Table 1 shows the costs of the mapped designs, execution times and the number of patterns matched from L1 and L2 for the different structures. In the first case, the cascade form was the best. In the second case, the same filter was used but with much higher Q. We observed that the cascade form could not use the L2 match because of parameter bound violation. The next example showed similar results. In the cascade form, of the six biquad sections, two low-Q biquad sections were realized us-

Design	Structure	Cost	Time(s)	#L1	#L2
3rd	Block	18	0.073	11	0
order	Obs.	10	0.075	7	0
Butter.	Ctrl.	10	0.071	7	0
LPF	Obs. Stair.	17	0.084	12	0
(low Q)	Ctrl. Stair.	15	0.070	12	0
	Cascade	5	0.047	3	1
3rd	Block	18	0.073	11	0
order	Obs.	10	0.075	7	0
Butter.	Ctrl.	10	0.071	7	0
LPF	Obs. Stair.	17	0.084	12	0
(high Q)	Ctrl. Stair.	15	0.070	12	0
	Cascade	14	0.069	12	0
6th	Block	88	0.194	71	0
order	Obs.	51	0.186	41	0
Cheby.	Ctrl.	51	0.187	41	0
LPF	Obs. Stair.	111	0.221	90	0
	Ctrl. Stair.	123	0.202	102	0
	Cascade	48	0.168	38	2

Table 1. Effects of Structure and Prog. Parameters

Function	FPAA Library Macro	Programmable Parameters Gain	Center Freq.(Hz)	Quality Factor
Prefilter	BPF (lo Q)	1.0	1000	0.70
Band	LPF (lo Q)	1.0	966	0.70
Separators	HPF (lo Q)	1.0	1180	0.70
Tone	BPF (hi Q)	1.0	700	20.00
Detectors	BPF (hi Q)	1.0	770	19.84
	BPF (hi Q)	1.0	850	20.00
	BPF (hi Q)	1.0	940	20.00
	BPF (hi Q)	1.0	1210	20.00
	BPF (hi Q)	1.0	1330	20.00
	BPF (hi Q)	1.0	1480	20.00
	BPF (hi Q)	1.0	1630	20.00

Table 2. FPAA Prototype DTMF Decoder

ing level-2 components while the remaining four high-Q sections violated the Q-parameter bounds.

Design Example: DTMF Decoder

In this section, we illustrate the prototyping and retargeting of a Dual-Tone Multi-Frequency (DTMF) decoder [11]. In the DTMF technique, each digit in the telephone is represented by a different pair of frequencies within the voice band. One of the frequencies is from the lower frequency group (697Hz, 770Hz, 852Hz and 941Hz) while the other is from the upper frequency group (1209Hz, 1336Hz, 1477Hz and 1633Hz). The behavior is specified in VHDL-AMS, and compiled into the intermediate format: SFG for the analog part, and CDFG for the digital. The SFG comprises of transfer functions of the various blocks. Technology mapping is performed for the Motorola FPAA [2] library. Table 2 shows the macros and the programmable parameter settings used to implement the various functions. The analog part was implemented on the Motorola MPAA020 FPAA and digital part on a Xilinx XC4010E FPGA, and the prototype was tested for the desired frequency response and also the following system performance requirements: detection time $<$ 40ms, and talk-off $<$ 10 hits.

FPAA-ASIC retargeting was performed next. For the target FPAA, the library characterization and mapping took upto 20 minutes for each macro in the library. Figure 10 illustrates the frequency response of the eight outputs (corresponding to the 697Hz, 770Hz ..., 1663Hz tones) obtained after Spice3gf simulation of the sized ASIC netlist. Table 3 gives the performance obtained for the ASIC components. The average area and power obtained for the filters was 24.8μ and 3.65mW respectively. The deviation of the obtained performance from desired performance was found to be less than 2.5%, 1% and 1.5% for the gain, center frequency and quality factor respectively.

6 Concluding Remarks

In this papaer, we have presented our synthesis methodology using rapid prototyping and technology retargeting that reduces time-to-market and is also cost-effective. First, technology mapping of analog systems was discussed. We employed a novel structural approach based on hierarchical pattern matching and covering to solve the covering problem for directed graphs (with cycles). Complexity is controlled

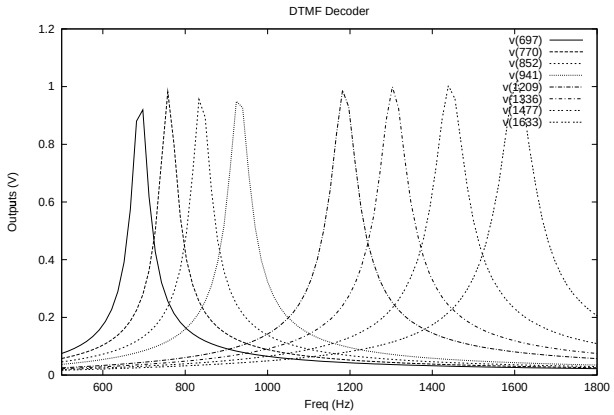

Figure 10. Spice Simulation of Freq. Response

Function	ASIC Component	Frequency Response Gain	Center Freq.(Hz)	Quality Factor
Prefilter	BandPass	0.976	990	0.68
Band	LowPass	0.987	972	0.71
Separators	HighPass	0.991	1153	0.70
Tone	BandPass	0.979	692	20.30
Detectors	BandPass	0.982	765	19.95
	BandPass	0.975	848	20.29
	BandPass	0.979	945	20.22
	BandPass	0.987	1192	20.31
	BandPass	0.999	1320	19.72
	BandPass	1.004	1484	19.97
	BandPass	1.019	1626	20.10

Table 3. Retargeted DTMF Decoder

by formulating it as a set of tree-covering problems. Next, the need for retargeting an FPAA to ASIC was addressed. We have presented a strategy to produce an ASIC equivalent for an FPAA design. Future work will address low-level effects such as noise and parasitics, and extension of the methodology for mixed signal systems.

Acknowledgments: The authors would like to thank Adrian Nunez, Nagu Dhanwada and Alex Doboli of the VASE team, and special thanks to Satish Ganesan.

References

[1] A. Doboli, A. Nunez, N.Dhanwada, S. Ganesan and R. Vemuri, "Behavioral Synthesis of Analog Systems using Two-Layered Design Space Exploration", DAC '99, pp. 950-958.

[2] Inc. Motorola, *Easy Analog Design Software User's Manual*, Motorola Inc., October '97.

[3] H. W. Klein, "The EPAC Architecture: An Expert Cell Approach to Field Programmable Analog Devices", FPGA '96, pp. 94-98.

[4] Inc. Zetex, *TRAC User Guide*, Zetex Inc., July '97.

[5] P. K. Sinha, *Multivariable Control*, Marcel Dekker, New York, 1984.

[6] R. Unbehauen and A. Cichocki, *MOS Switched-Capacitor and Continuous-Time Integrated Circuits and Systems*, Springer Verlag, Berlin, 1989.

[7] K. Keutzer, "DAGON, Technology Binding and Local Optimization by DAG Matching", DAC '87, pp.341-347.

[8] G. De Micheli, "Architecture-Level Synthesis and Optimization" in *Synthesis and Optimization of Digital Circuits*, Mc-Graw Hill, 1994

[9] E. Detjeus and G. Gannot, "Technology Mapping in MIS", ICCAD '87, pp. 116-119.

[10] N. Dhanwada, A. Nunez and R. Vemuri, "Component Characterization and Constraint Transformation based on Directed Intervals for Analog Synthesis", 12th Intl. Conf. VLSI Design '99.

[11] B. J. White, G. M. Jacobs and G. F. Landsburg, "A Monolithic Dual Tone Multifrequency Receiver", IEEE J. Solid State Circuits '79, vol. SC-14(6), pp. 991-997.