A GP Algorithm for Efficient Synthesis of Gm-C Filters on a Hexagonal FPAA Structure

Stanis Trendelenburg, Joachim Becker, Fabian Henrici, Yiannos Manoli
Department of Microsystems Engineering, IMTEK, Albert-Ludwigs-University
Georges-Koehler-Allee 102, 79110 Freiburg, Germany
trendele@imtek.de

ABSTRACT

This work presents an approach based on Genetic Programming for the synthesis of continuous-time analog filters on a field-programmable analog array. A multi-objective algorithm is used to synthesize both the topology and parameter values of G_m-C filter structures to be instantiated on the FPAA based on a given filter specification. The presented algorithm is highly adapted to the underlying hardware platform, with the goal of making an efficient crossover of high-quality building blocks possible without biasing certain types of schemata. By manipulating of the program tree in the instantiation phase, it is assured that the resulting synthesized structure fits within the physical constraints of the underlying hardware platform.

Categories and Subject Descriptors

J.2 [**Computer Applications**]: Electronics; I.2.8 [**Artificial Intelligence**]: Heuristic Methods

General Terms

Algorithms

1. INTRODUCTION

Since many years, the idea of reconfigurable analog hardware platforms, analogous to FPGAs in the digital domain, has been a topic of active research. The goal of these Field-Programmable Analog Arrays (FPAAs) is to bring the flexibility of their digital counterparts to the analog world, allowing analog designers to use rapid prototyping techniques to shorten the typically full-custom, expensive and time-consuming analog design process. Evolutionary algorithms offer a solution by which filters can be synthesized directly from the specification, possibly directly onto the target hardware, taking into account all available reconfigurable elements to build custom filter solutions.

2. FPAA ARCHITECTURE

The underlying hardware architecture is a field programmable analog array specifically designed for high-frequency, continuous-time filter applications. On the highest level, the structure consist of hexagonal Configurable Analog Blocks (CABs) laid out in a honeycomb grid.

The CABs themselves contain 7 digitally programmable transconductance amplifiers (G_m-cells), one in a self-feedback configuration and the other 6 forming the connections to the neighboring CABs. The transconductors are digitally configurable and, together with the parasitic capacitance at the center of each CAB, provide the necessary elements for the instantiation of G_m-C type analog filters. Previous work ([1]) has shown that an evolutionary approach to filter synthesis on the presented architecture is feasible and leads to good results.

A problem that arises from the hexagonal structure of the array is that locality in the search space only corresponds to locality in the problem space under very specific conditions. There is no straight-forward way to map the position of genes in the genome which correspond to G_m-cell settings of the array in a way that neighboring genes correspond to neighboring G_m-cells.

The reason is that the concept of what constitutes a "neighboring" cell is different depending on the topology a filter, and cannot be fairly represented in a traditional genome string or array without favouring some types of high-quality building blocks in respect to others regarding the possibility of a successful crossover. Since the series and parallel connection of low-order functional "building blocks" is a fundamental technique of filter design, it is highly desirable to map the swapping of these functional blocks between individuals as good as possible to the crossover operator.

3. GP-BASED APPROACH

The genome of the filter is represented by a program tree of functions that can be interpreted to build up a filter on the FPAA. Tab. 1 summarizes the functions used and their terminals. For instantiation of the corresponding individual, the tree is traversed in depth-first order, applying the function at each node to enable selected connections between CABs on the initially empty array.

The root node of any tree is always the "input" function, which has the setting of the I/O cell (also a programmable G_m-cell) as a terminal. It sets the CAB it connects to as the "active" CAB. The "set" function sets the feedback cell of the currently active CAB to the setting specified by its terminal. Subsequent calls to this function overwrite the

Table 1: Functions

Name	Symbol	Terminals
Input	i	cell setting ($\pm$[1..6])
Set	s	cell setting ($\pm$[1..6])
Connect	c	direction [1..6], cell setting ($\pm$[1..6])

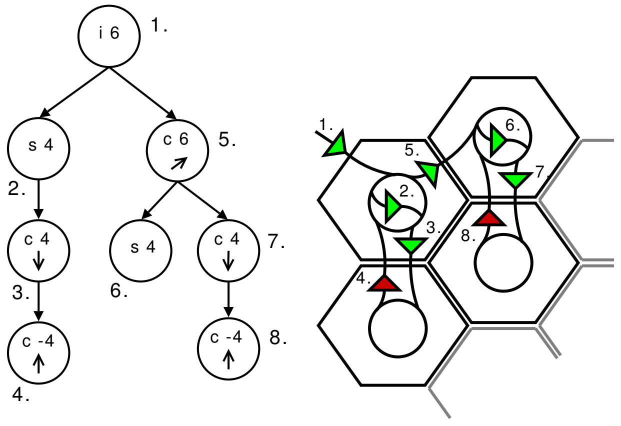

Figure 1: Example tree and resulting structure

previous setting as long as the active CAB is not changed. The "connect" function set the G_m-cell that branches off to the neighbouring CAB in the direction specified by its first terminal, to the setting specified by its second terminal.

Any node in the tree can have between zero and three child nodes. For example, a linear tree of n "connect" functions represents a structure of n CABs connected in series, while any node with 2 or more children of type "connect" represents a branch in the structure: Each child connects to a different neighbouring CAB, causing the structures to be built up by each subtree to diverge in different directions. Fig. 1 shows an example of a program tree next to the resulting structure on the hexagonal FPAA. Terminals representing directions in the tree are depicted by corresponding arrows. The G_m-cell settings of the individual connections in the FPAA structure have been omitted for clarity.

The advantage of this tree representation lies in the fairness of tree-based crossover with regard to the different building blocks. Compact, high-quality filter building blocks can be represented as independent subtrees, and there is no inherent bias in the genome representation for favouring one over the others. For example, the subtree starting at node 2 represents a typical "gyrator" feedback structure (in the lower left of the array), and can be swapped between two trees in a single crossover operation. The same holds true for subtrees representing linear structures of cascaded elements.

The fixed FPAA layout presents an additional constraint: any individual must result in a filter structure that can be instantiated on the array. The case of filter structures created by different subtrees "colliding" on the array is handled by removing the children of any subtree that connects to a CAB of the array which has already been connected to. The same is done with nodes trying to place connections that would lie outside of the array borders. Regarding tree growth, this trimming technique effectively presents an upper limit to the tree size, without the need to introduce further constraints or optimization objectives into the algorithm.

The algorithm includes a crossover operator that selects a random node in each parent tree and swaps the subtrees starting at those nodes. The tree mutation operator, if applied, selects one random node in the tree (including the root node) and replaces it with a randomly generated subtree. An additional mutation operator visits each node in the tree and permutes the values of the integer-encoded terminals specifying G_m-cell settings.

Filter structures are evaluated by instantiation on the FPAA model and subsequent analysis of the resulting transfer function characteristics. The two main criteria for the fitness function are filter stability and matching of the desired specification, which are the two independent objectives of the algorithm. Fit to the specifications is measured by sampling the filters transfer characteristic at a fixed number of points, and calculating the mean squared error. For the stability, the error is the number of unstable poles. The algorithm uses non-dominated sorting and crowding distance parameter, as proposed by Deb et al. in [2], to favour solutions spread evenly over the non-dominated front instead of clusters of very similar solutions.

Filter stability has been represented as an additional objective instead of a constraint. This allows filters with only a small number of unstable poles to occur during the optimization process. Those poles have a chance to be moved to the stable region by parameter variation before the end of the optimization process, where all individuals with remaining unstable poles are discarded. Discarding those individuals earlier would severely limit the ability of the algorithm to explore new configurations. The resulting individuals of the last generation are then compared according to their fit to the specification.

4. RESULTS

When compared to the GA in [1], the presented GP algorithm reaches a higher fitness in the same number of evaluations. Trials were run with both algorithms for two selected example filter specifications. Fig. 2 shows the comparison of mean best fitness over 10 subsequent runs for both algorithms applied to both of the test problems. The number of evaluations was originally set to 1200, but could be reduced to 400 for the lowpass problem without a significant reduction of the solution quality in case of the GP algorithm.

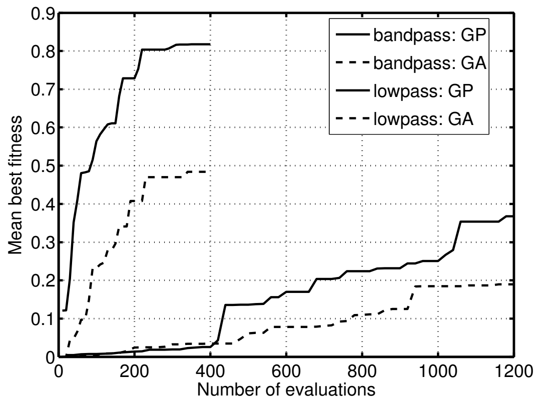

Figure 2: Fitness for selected goal filters

5. REFERENCES

[1] J. Becker, S. Trendelenburg, F. Henrici, and Y. Manoli. Synthesis of analog filters on an evolvable hardware platform using a genetic algorithm. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 190–197, New York, NY, USA, 2007. ACM.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, Apr 2002.