Synthesis of Analog Filters on an Evolvable Hardware Platform using a Genetic Algorithm

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

ABSTRACT

This work presents a novel approach to filter synthesis on a field programmable analog array (FPAA) architecture using a genetic algorithm (GA). First, a Matlab model of the FPAA is created and verified for compliance with transistor-level simulations of the FPAA. Using this model, different filter structures are built using an active-RC approach and evaluated. Secondly, a robust genetic algorithm is implemented in Matlab, which allows synthesis of analog filters on the given structure. Optimal parameters and operators of the genetic algorithm are identified by gradual adaptation and performance evaluation, and the general feasibility is shown. Finally, the GA is used to overcome quantization-limitations of the FPAA structure and find configurations of filters, which would not have been achievable with traditional synthesis methods.

The system is not only a platform for theoretical investigation of filter structures on the given chip structure but also provides a framework for evolution and instantiation of filters on actual chip hardware.

Categories and Subject Descriptors

I.6.3 [**Simulation and Modeling**]: Applications;
I.2.8 [**Artificial Intelligence**]: Heuristic Methods;
J.2 [**Computer Applications**]: Electronics.

General Terms

Algorithms, Design.

Keywords

computer aided design, evolvable hardware, genetic algorithm, hardware realization, microelectronics, synthesis.

1. INTRODUCTION

Full-custom analog design of integrated circuits is time-consuming and their verification expensive. Many attempts have been made to apply rapid-prototyping techniques known from the digital domain to analog designs. Several field programmable analog arrays (FPAAs) have been developed, which differ hugely in complexities and capabilities. Although FPAAs are not comparable to their digital counterparts (FPGAs) with respect to flexibility, development tools, usability, or market share, there are some commercial products available for specialized applications as well as ongoing research at universities.

The goal of the presented work is to develop an FPAA, which is suitable for reconfigurable implementation of analog continuous-time (CT) high-frequency filters on an integrated circuit. This would enable designers to immediately verify the structure and coefficient selection of a filter in hardware and provide a working prototype. When being used as a verification tool for system-level designers, only a prototype would be implemented on the FPAA and deliver the proof-of-concept. The knowledge gained would then be used to accomplish an integrated design for mass-production. Moreover, the FPAA structure could be used as reconfigurable hardware platform in itself, to achieve a filter which is adaptable in the field to varying requirements or environmental conditions.

When comparing the proposed architecture with field programmable analog circuits being reported in literature, the meanings of *field programmability* and *analog array* have to be classified first. The simplest form of field programmability in filters are adjustable filters with fixed structures and programmable coefficients (e.g. [6]). More ambitious designs extend the meaning of programmability to the reconfiguration of circuits beyond the presetting of a structure. Thus, *field programmability* in the context of this paper is to be understood in terms of both reconfigurable and adjustable.

Concerning the number and granularity of analog elements on an FPAA, there are structures with few but large configurable analog blocks (CABs) [3], up to designs with hundreds of very small elements [7, 5]. The CABs have to be designed in a way that they provide both all the basic analog features needed, and allow reasonable partitioning for the desired application. The network of the analog blocks needs to be so generic that designers are enabled to invent new structures and instantiate them on an analog fabric. This work aims for a large number of analog elements so that both the coefficients and the structure of the analog part are freely reconfigurable. An architecture has to be found that features a good compromise between dedicated features for the implementation of CT filters and the freedom to implement new building blocks.

In the following, we shortly review an analog array structure with many degrees of freedom, which will provide the basis for instantiation of arbitrary filter structures. Upon that, basic methods of filter synthesis are applied to the FPAA and feasibility of the array is shown. The main part of this paper describes the use of a genetic algorithm to achieve mapping of given filter specifications to the FPAA structure. Because of the specific design and granularity of the array structure, it is not only possible to use the result of an evolution for instantiation of that filter in hardware; a main feature of the structure is that the description of the filter is close to a human abstraction level. Therefore, results are likely to be easy to analyze so that working principles can be understood and included into new designs.

2. FPAA-STRUCTURE

The original FPAA structure as presented in [2] consists of a two-dimensional array of CABs, which include G$_m$ cells as active transconductors. The arrangement in a hexagonal layout as depicted in figure 1 allows routing of signals throughout the chip. Series and parallel connection as well as feedback of any order is indispensable for instantiation of filters and is thus provided by the structure.

Every CAB has six branches connecting to the respective neighbor CABs and one for self-feedback. Each branch consists of a tunable transconductor, which is built by parallel connection of unit G$_m$ cells (differential transconductance amplifiers). The massive parallel connections of both parasitic input capacitances as well as parasitic output capacitances at the input nodes of each CAB sum up to capacitances in an order of magnitude such that they are suitable as integrating capacitances for the filter [4]. Therefore, each filter stage is made up of the tunable transconductance of one branch and the capacitance of the center node of the next CAB [1].

2.1 Building a MATLAB-Model

In order to investigate the whole FPAA on the system level, it is beneficial to have a model representation assembled from the most basic transfer function blocks. It was chosen to implement the model in Matlab due to the numerous signal processing and filter toolboxes available.

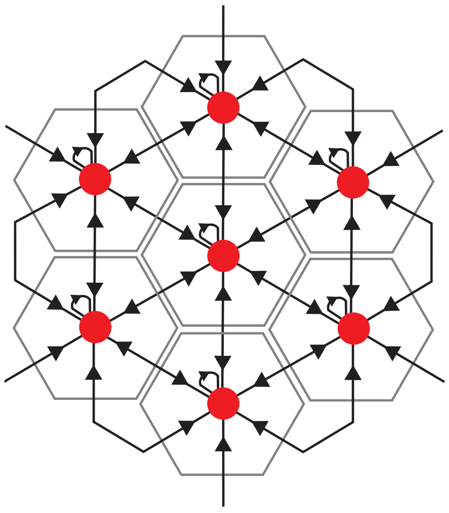


Figure 1: Hexagonal array topology of FPAA.

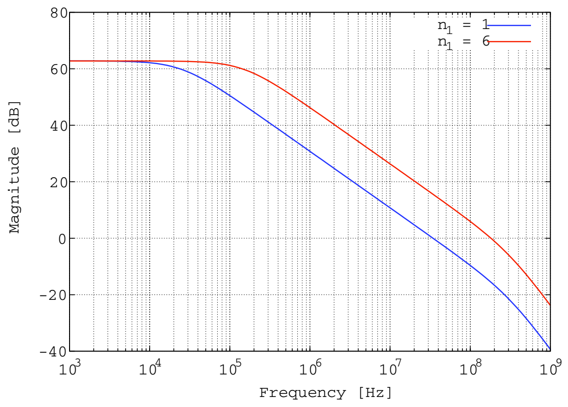


Figure 2: 1st order integrator with different settings of the tunable transconductor.

Through setting of the G$_m$ cell transconductance, filter characteristics can be tuned as figure 2 demonstrates for the case of the simple integrator. The parameter tuned here is the corner frequency, which depends directly on the transconductance tuning setting $n = [1..6]$:

$$T(s) = \frac{A_0}{1 + \frac{s}{\omega_c}} \text{ with } \omega_c = \frac{n \cdot G_m}{C_L \cdot A_0} \text{ and } A_0 = G_m \cdot r_{out} \quad (1)$$

As can be seen in figure 2, the transfer function has a 2nd pole at about 300 MHz. This pole is intrinsic to the G$_m$ cell and stems from its internal realization as differential amplifier with folded cascode. Ideally the frequency of this parasitic pole should be high enough that it does not influence the transfer function of the filters to be realized on the FPAA. The pole is nevertheless included in the FPAA model to have a precise representation of the real FPAA and to be able to determine its influence on the different filter types examined in the next section.

The building blocks can be broken down to the following four basic elements: The G$_m$ cell's transconductance G_m, its output resistance r_{out}, the aforementioned parasitic pole at the frequency ω_2, and the combined input and output capacitances of the 7 outgoing and incoming G$_m$ cells that form the constant load capacitance C_L at the center node of a CAB. Figure 3 shows the realization of a single G$_m$-C stage as electrical circuit and transfer function blocks.

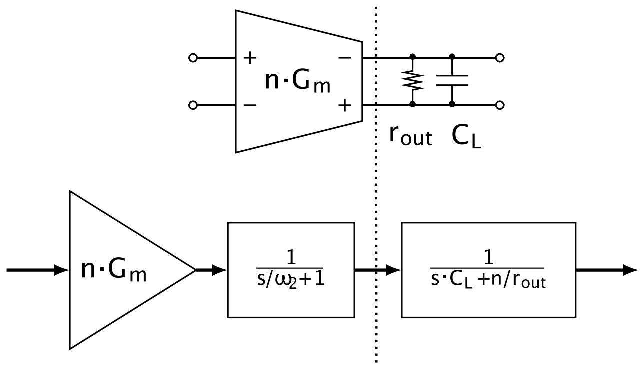


Figure 3: Conversion of the basic electrical CAB elements to transfer function blocks in Matlab.

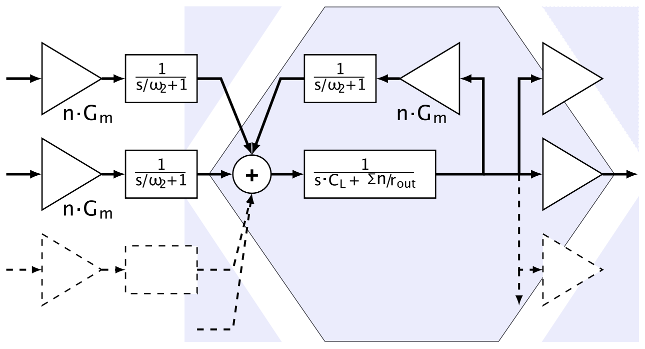


Figure 4: Assembly of the FPAA as state-space model from the basic elements in Matlab.

The parasitic input and output capacitances are not modeled separately, because their exact value has no direct influence on the realized filter function. Only the value of the sum of all parasitic capacitances at the center node that forms C_L has an influence on the filter function. It can be easily calculated from the location of the dominant pole in the simulation of a single stage G_m lowpass.

The G$_m$ cell properties can be modeled as a transfer function block with:

$$T_{G_m}(s) = n \cdot G_m \cdot \frac{1}{\frac{s}{\omega_2}+1} \tag{2}$$

The output resistance, which lies in parallel to the output capacitance, cannot be modeled as a separate transfer function and is accounted for in the C_L transfer function.

When connecting multiple cells in parallel, the output conductances of all G$_m$ cells have to be summed up at the center node. Therefore, the output resistances of all inward facing G$_m$ cells have been combined with the resulting load capacitance C_L to a separate block with:

$$T_{C_L}(s) = \frac{1}{s \cdot C_L + \frac{1}{\sum r_{out}}} \tag{3}$$

The resulting transfer function is:

$$T(s) = n \cdot G_m \cdot \frac{1}{\frac{s}{\omega_2}+1} \cdot \frac{1}{s \cdot C_L + \frac{n}{r_{out}}} \tag{4}$$

The state-space model for a specific FPAA configuration can then be assembled programmatically in Matlab from these building blocks (figure 4), by connecting the corresponding transfer function blocks and keeping account of G_m cell settings and output resistances. The transfer functions are then converted to a single multiple-input/multiple-output (MIMO) model, from which transfer functions between arbitrary inputs and outputs can be extracted.

2.2 Verification and Parameter Extraction

In order to verify that the FPAA model built in the previous section is a valid representation, it was compared with a transistor-level Spectre simulation of the chip. First, the model parameters G_m, C_L, A_0, r_{out}, and ω_2 were extracted. Then, a set of example FPAA filter structures were simulated using both the Matlab model and Spectre, and their results compared.

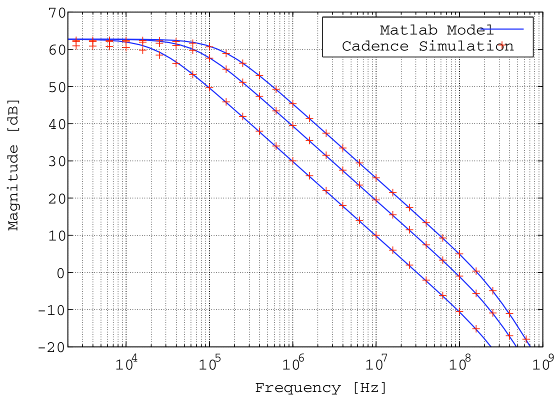


Figure 5: Comparison of the Matlab model with Spectre simulations (1st order lowpass).

The parameter values were extracted from the 1st order lowpass simulation (figure 5). Since the two poles of the transfer function are far enough apart from each other, their frequency can be approximated by the points of −45 and −135 degrees phase shift, respectively. Combining the relations of (1), the equation system has 2 degrees of freedom. A_0 can be extracted from a separate DC simulation. The last parameter (C_L or G_m) was approximated by curve fitting. Final parameter values are summarized in table 1.

G_m	A_0	r_{out}	C_L	ω_2
2.87 mS	62.7 *dB*	480 kΩ	14.75 pF	293 MHz

Table 1: Parameters of the G$_m$ cell.

A cascaded 4th order bandpass was simulated using different configurations in order to verify the model and the extracted parameters against a higher order function. As can be seen in figure 6, the Matlab model fits the transistor-level simulation well, which proves that it is indeed a valid representation of the FPAA, and that all significant effects have been factored in. This allows system-level characterization of the FPAA using the Matlab model while retaining the confidence that the results gained by simulation will hold when applied to the FPAA hardware.

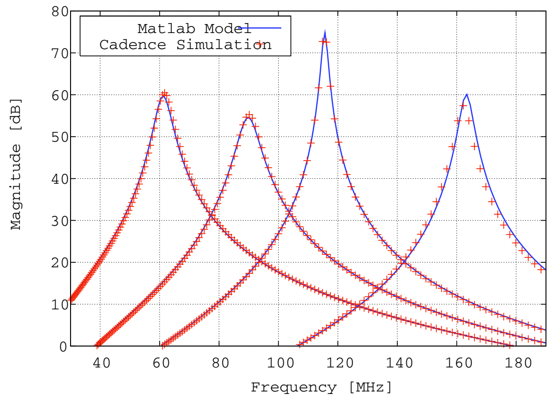


Figure 6: Comparison of the Matlab model with Cadence simulation (4th order bandpass).

The Matlab model is more flexible because - once instantiated for a given configuration - it allows to extract transfer functions between arbitrary inputs and outputs, in contrast to parameter sweeps using Cadence, which only yield the nodes phase and magnitude values. Furthermore, due to the high complexity of the transistor-level simulation, it takes a significant amount of time and CPU resources (several hours) compared to the model assembly in Matlab, which is almost instantaneous (under 1 second).

3. CLASSICAL FILTER SYNTHESIS

A straightforward technique to implement analog filters on the FPAA-structure is the replication of passive RLC-circuits with active elements such as Op-Amps, or in this case, G_m cells. Filters made from passive circuits are well understood and a large number of different approaches for filter design and transformation have been described in literature. Identifying blocks on the FPAA that correspond to passive filter elements would therefore make this knowledge available for filter design.

First of all, 1st and 2nd order building blocks of basic filter functions can be identified, which can then be cascaded to achieve higher order functions. In general, all 2nd or higher order filter functions can be expressed as a product of 1st and 2nd order functions. The basic elements of the FPAA, as described in the latter section, are the G_m cell transconductor and the load capacitance C_L. These elements can be used to build a "library" of simulated passive elements that can be combined to higher order filters.

The most basic filter element of the FPAA is the G_m-C integrator, which corresponds directly to a 1st order R-C lowpass. The corner frequency of the filter is tunable in a range from 22.6 kHz to 135.6 kHz in steps of 22.6 kHz, with a gain bandwidth product between 31.0 MHz and 185.3 MHz. By connecting several of these stages in series, higher order lowpass filters can be created. The corner frequency and GBW of these stay the same for any order, while the number of possible G_m cell combinations increases, and therefore the minimal step size for tuning of both corner frequency and GBW decreases with each additional filter stage, leading to a more fine-grained tunability for higher order filters.

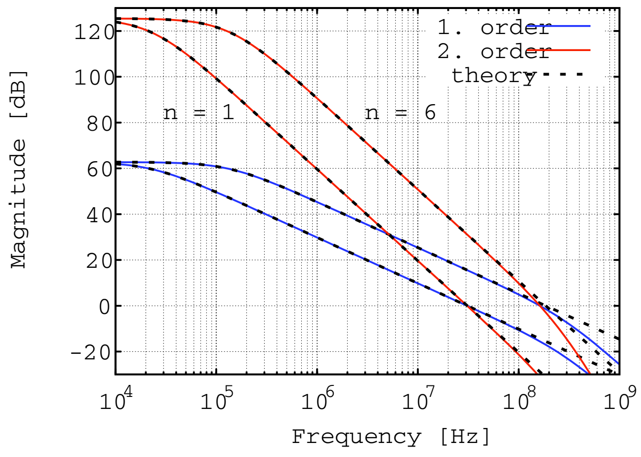


Figure 7: Magnitude plot of 1st / 2nd order lowpass.

Figure 7 shows the frequency response for selected 1st and 2nd order configurations, both for the theoretical (ideal) model, as represented by the transfer function derived above, and the Matlab model of the FPAA which includes the parasitic pole of the G_m cell, thereby providing a more accurate representation of the real hardware. The influence of this pole in the model can be seen above unity-gain frequency, where the steepness of the curve increases. Otherwise, the ideal model is a good approximation.

It can be shown that a number of different lowpass and bandpass filters can be synthesized on the FPAA using this approach. Basic blocks have been identified and their realization on the FPAA proven. A problem that remains with the current architecture is that the G_m cell pole is too close to the desired operating frequency, which makes it more complicated to design higher order filters using classical analog design methods.

Incorporating the parasitic pole as a separate integrator into the G_m cell transfer function, as has been done for the Matlab model of the FPAA, would lead to transfer functions of very high order, and make it cumbersome to find structures that correspond to classical filter designs, due to the absence of simple block connections without integrating behavior. A practical approach to find desired filter configurations is to build up filters using the identified building blocks directly on the FPAA model. Thereby, the resulting transfer function in which all non-ideal effects are already contained can be simulated instantly. The designer can then try to work around the unwanted effects by tuning selected filter parameters.

A filter design GUI has been programmed in Matlab allowing the user to build up higher order filters step-by-step using predefined stages and to analyze the resulting transfer function, which is calculated by instantiating the filter on the Matlab model of the FPAA. The user can then fine-tune the filters properties by changing selected G_m cell parameters, while being able to constantly monitor the influence of the changes on the final filter function.

Another approach, namely employing a genetic algorithm to find suitable filter parameters and topologies for a set of given filter requirements is presented in the following.

4. GA IMPLEMENTATION IN MATLAB

To apply genetic algorithms to filter synthesis, the Matlab model of the FPAA introduced in the previous section was used. It has been shown to be a good representation of the FPAA, so the results gained here are expected to be applicable to the final hardware of the FPAA as well.

4.1 Representation of filter

A filter on the FPAA is represented through the setting of all configuration switches of the chip. Each of the 7 CABs has 7 branches so that the configuration is stored in a 7×7 matrix. Each entry in this matrix is an integer [1 .. 6] for the transconductance parameter, leading sign for inverting/ non-inverting output, or 0 for switched off. In terms of evolutionary algorithms, this matrix defines the genome of an *individual* on the FPAA. Since it is known from simulations in the previous section that the transfer function of a given configuration as calculated by Matlab matches quite well with the behavior of the actual circuit, it is sufficient to analyze how changes in the genome change this transfer function. This is the basis of a fitness function for a GA.

$$\begin{pmatrix} cab1_{\uparrow} & cab2_{\uparrow} & \dots & cab7_{\uparrow} \\ cab1_{\nearrow} & cab2_{\nearrow} & \dots & cab7_{\nearrow} \\ cab1_{\searrow} & cab2_{\searrow} & \dots & cab7_{\searrow} \\ cab1_{\downarrow} & cab2_{\downarrow} & \dots & cab7_{\downarrow} \\ cab1_{\swarrow} & cab2_{\swarrow} & \dots & cab7_{\swarrow} \\ cab1_{\nwarrow} & cab2_{\nwarrow} & \dots & cab7_{\nwarrow} \\ cab1_{\odot} & cab2_{\odot} & \dots & cab7_{\odot} \end{pmatrix} \quad cab \in [-6..+6]$$

Figure 8: Matrix representation of one individual.

4.2 Generation of first population

At first, a population needs to be created. The first generation is built from randomly initialized configurations. A basic approach is to fill the matrix evenly with values from the definition range. Although this is possible, it is more advantageous to include some knowledge about the filter to be implemented into the preparation of the first generation: Looking at the CAB matrix for hand-made, working filter configurations, it is unusual that is is completely filled with active coefficients. Usually, only few CABs are active and route the signal through the array, whereas most transconductors are switched off. This is contradictory to an evenly distributed random configuration and the result of the evolution can be improved, when the first generation consists of sparsely filled matrices rather than fully populated matrices.

4.3 Definition of goal and fitness

In order to compare a configuration with the desired goal definition, a fitness function has to be set up, which allows to rank the individuals within a generation and select the best for further evolution. The fitness function is one of the most important parts of the algorithm because it defines which parameters of a configuration are taken into account for evolution and how they are defined, measured, and weighted. To evaluate the fitness of a configuration, the transfer functions from each input to each output of the chip are calculated and rated. In the following, a list of criteria for instantiation of filters on the FPAA is given.

Goal points

Points in the frequency vs magnitude plot can be specified, which the resulting transfer function curve should match as closely as possible. The mean square error (i.e. the deviation at each point) of the filter's curve is added to the fitness penalty. This allows to provide the general shape of the filter, while still being able to weight critical regions stronger by increasing the number of points per frequency in this region. An exemplary filter description with spline through the goal points and one individual is depicted in figure 9.

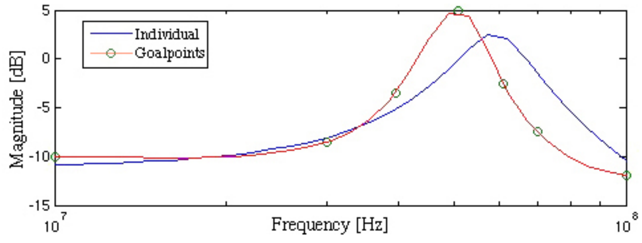


Figure 9: Amplitude plot with goal points.

Frequency windows

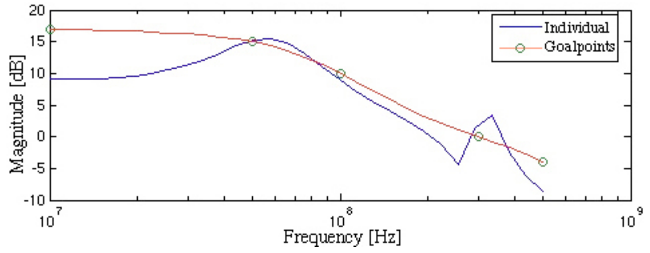
Another way of specifying filter requirements is by marking windows in the frequency vs magnitude plot that the filter transfer function must be above or below of. In terms of the GA, this means the total deviation is summed up and added to the fitness penalty. The lower the overall fitness value for transfer function matching is, the better is the matching of the filter to the goal. It is a main part of the fitness function and sufficient to build a working GA. However, there are some other characteristics of filters that can be added in order to achieve more robust and feasible filters on the FPAA.

Pole-zero placement

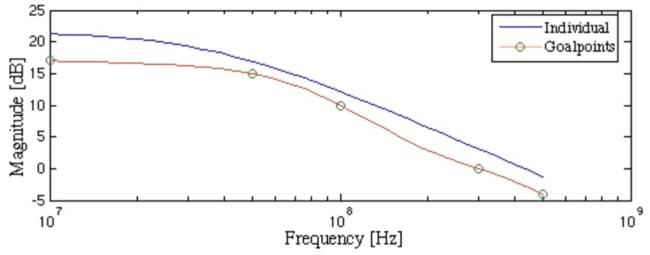
Sometimes a filter function can be better expressed in terms of the placement of its poles and zeroes. This feature allows to specify the locations of the dominant poles and zeros, both real and complex-conjugate. As with the goal points, the fitness penalty in this case is the mean square of the total deviation of the actual poles and zeroes from the specified locations.

Filter order

The filter order is an important parameter, because it influences two major characteristics of the filter: firstly, it is proportional to the number of active transconductors and therefore related to the space- and power-consumption. If the same functionality can be achieved by filters of different order, the lower order configuration is preferred. Secondly, the orders of the nominator and denominator of the transfer function can be read as the number of zeros and poles of a filter, which are a measure for the number of possible inflection points of a transfer function. Figure 10 (a) shows how a function with a high number of poles makes use of sudden inflections to meet the goal very well at the defined points but is very jagged in between. In general it is more desirable to have a filter with smoother shape even if it has a slightly larger variance to the goal. Smoothness can be forced by a lower number of poles as depicted in figure 10 (b).



(a) Filter with high order



(b) Filter with low order

Figure 10: Filters of different order.

Other requirements

Since the genetic algorithm only optimizes for the fitness value, it must include even the most trivial requirements, such as that the filter must be stable. This is achieved by specifying penalty values that get added to the fitness value for every pole on or to the right of the imaginary axis in the pole-zero diagram. Although it might be tempting to set these values as large as possible, one must keep in mind that the fitness value represents the sum of all these requirements. Each parameter must be weighed carefully to prevent the GA from optimizing towards the wrong goal.

4.4 Parameters of the GA

A genetic algorithm is defined by the genetic operators and techniques used, as well as global parameters like population size, crossover and mutation rates. These parameters and their interactions influence the performance of the genetic algorithm in complex ways, which also depend on the problem to which the GA is applied. For some parameters, like the mutation rate, the range of reasonable values can be estimated. Others, like the crossover and selection operators, are more problem dependent and therefore have to be found by experimentation.

To compare the performance of different GA variants applied to the same goal specification, the fitness of the best individual in the population plotted versus the generation number is used. This is a good measure for the GA performance, because it allows to compare both the best fitness reached, as well as the convergence speed. Since the outcomes of a GA may vary greatly between two runs, each GA was run for 20 times in a row and the average and median fitness per generation were compared. To determine the best set of parameters for the specific problem of finding filters on the FPAA, example goal functions were chosen, against which each GA was tested. Numerous different parameter sets were tested against different goal functions and only a summary of the final result can be presented here.

Table 2 summarizes a parameter set for the GA with a fast and yet very robust behavior. It uses tournament selection with tournaments of size 2, a uniform crossover operator and creeping mutation with maximum mutation distance of 2.

In summary, the genetic algorithm proves to be a good alternative to classical methods of filter synthesis, especially since it is, due to its nature, completely unhindered by the non-ideal properties of the FPAA, like for example the parasitic poles. Through the more abstract search-space approach, instead of designing filter functions using ideal assumptions and trying to compensate for non-idealities afterwards, the FPAA is viewed as-is, and properties normally regarded as interference might even contribute positively to the realization of the desired function. This shall be further investigated in the following section.

Population size	50
Initialization	Random, with 50% zeros
Selection operator	Tournament selection with size 2
Crossover operator	Uniform crossover
Mutation operator	Creeping mutation with max. 2
Crossover rate	70%
Mutation rate	5%

Table 2: Parameters for best performance of GA.

5. SYNTHESIS RESULTS

As mentioned in the introduction, the structure of the FPAA has deliberately been set up to allow backtracking of evolved configurations. A current research topic is to create and investigate previously unknown filter structures on the FPAA. In the following section some results will be presented, which outline how the use of a GA as synthesis tool enriches and exceeds the possibilities of traditional techniques.

Figure 11 shows a goal definition with frequency window specifications (red), which give a typical description of a bandpass design-problem. Some transfer functions of configurations found by the GA are plotted in the windows and although they have different shapes and orders, they all fulfill the given specifications within the boundaries. Obviously, the problem description leaves some room for diverse solutions and the designer would have to chose by additional criteria which configuration is the best.

Figure 12 shows a graphical representation of the two individuals with the best fitness value on the FPAA structure. Figure 12 (a) depicts the characteristics of a biquad bandpass configuration with the two gyrators from CABs *n1*/*n4* and *n2*/*n3* being connected in series. This is a configuration, which could have been designed by use of classical filter synthesis and is a well known standard implementation. In figure 12 (b), the function has been realized by the GA using another approach, namely a filter composed of a lowpass stage *n1* and 2 integrators in series (*n4* and *n3*) with a distributed feedback network over *n2*.

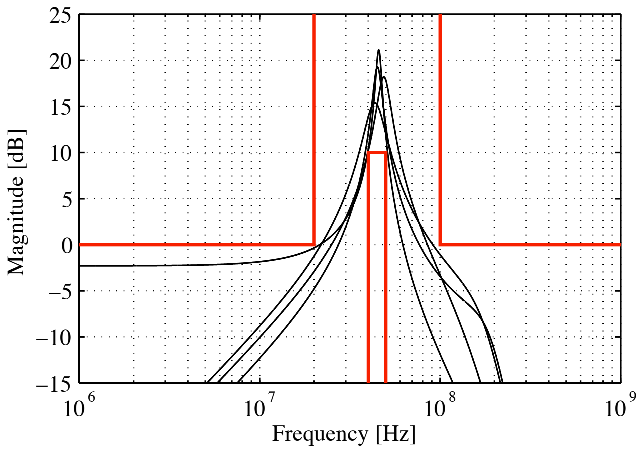
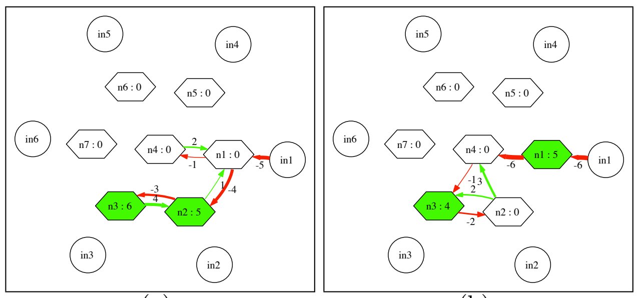


Figure 11: Bandspass goal specifications and results.



(a) (b)

Figure 12: Solutions found for the bandpass goal specification.

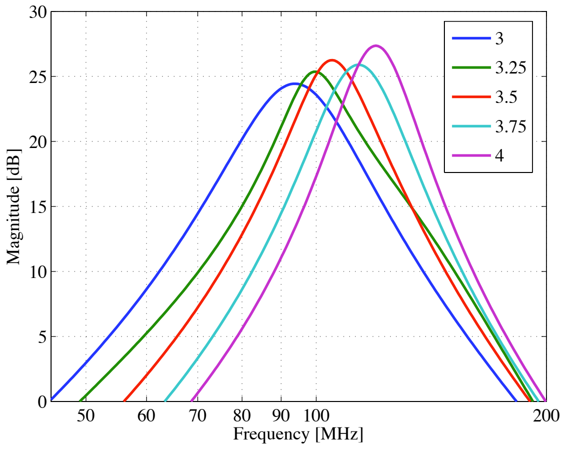


Figure 13: Intermediate values for bandpass.

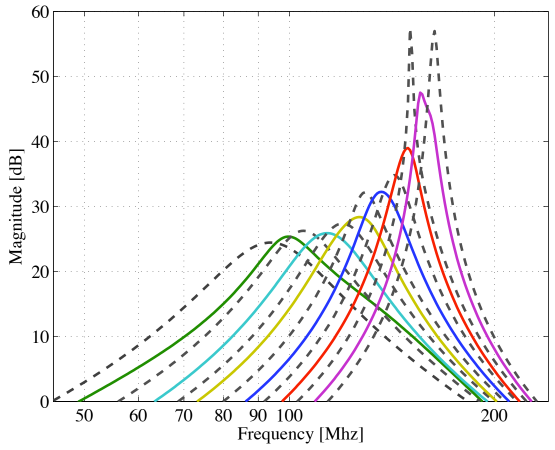


Figure 15: Designspace exploration of one bandpass.

As shown by these very different solutions for the same problem, the genetic algorithm explores the search space in a way that is not biased towards a specific realization, and might well be able to find solutions not within the reach of traditional design methods.

Another interesting field of application is to overcome limitations of parameter quantization by generation of intermediate values. As depicted in figures 2, 5, 6, 7, and equation (2), the quantization of the transconductance parameter of each branch from $n = [1..6]$ leads to a quantization of the achievable transfer functions. Once a filter structure is set, it is very easy to move characteristic poles by sweeping of parameters but the sweep can only be done in discrete steps. It might however be desirable to move e.g. the center of a bandpass to a frequency, which would refer to a non-integer value of a parameter. It is almost impossible to amend a structure by hand to resemble such behavior and the question arises, whether the regions in the design-space between integer values are not accessible at all.

In order to investigate this problem, goal functions have been set up in Matlab, which represent the theoretical filter properties of intermediate values. The GA has then been run in order to find configurations that match these transfer functions. Figure 13 shows transfer-functions of the biquad bandpass configuration from the previous section with values 3 and 4 in the stages of the pure biquad structure. The functions for 3.25, 3.5, and 3.75 have been calculated by Matlab and given to the GA for evolution. The plots in figure 13 are results from the GA, which look very similar to a

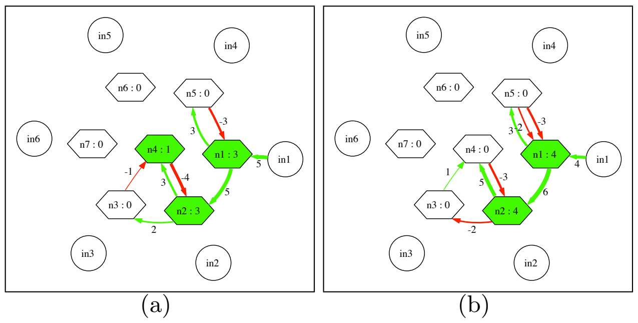


Figure 14: Bandpass solutions for 3.25 and 3.75.

continuous sweep of parameters. However, there are slight variances from the ideal function shape, which can be seen where the lines are not fully in parallel. The line for 3.25 even crosses several other lines for higher frequencies, which is another sign that this is no true parameter sweep.

When examining the best individuals as in figure 14, it can be seen that the biquad structure of cells *n1*/*n5* and *n2*/*n4* has been preserved as dominating shape of the filter. Only slight variations in the parameter values and an additional 2nd order lowpass path over *n3* lead to the amendments necessary to resemble the intermediate values 3.25 and 3.75. This example shows how valuable the enhancement of traditional synthesis methods by genetic algorithms can be. The knowledge gained from the implementations found with the GA can be understood on the system-level and made available as guidelines and inspiration to designers.

Finally, figure 15 shows how the whole design space can be made accessible by approximations to intermediate values. The dotted lines depict integer values $n = [1..6]$ and the solid lines solutions of the GA, which smoothly fill the voids of quantization. The combination of a widely reconfigurable architecture with a robust genetic algorithm is therefore a powerful and versatile design tool for the synthesis of analog filters.

6. OUTLOOK

As mentioned before, the Matlab model of the FPAA structure presented here is derived from a transistor-level model of a microchip recently fabricated in a 130nm CMOS technology. When the basic validation and characterization of the prototypes is finished, the design environment presented is immediately usable with the real hardware. Configuration of the device can be done via USB-interface and fitness evaluation through a test-board and network analyzer. Evolution on hardware will deliver working prototypes of filters, which could be used in analog signal processing.

The experiments to come will have to prove that the results gained from simulation hold for real hardware. Also, it will be interesting to see whether the GA stays with perfect system-level building-blocks or makes use of the fabrication process variations of the individual pieces of hardware. An investigation of these variations would then lead to either deliberate facilitation of those properties or amendments to the GA such that these individual variations get cancelled out. It would be possible to run parallel fitness evaluation on several different chips, in order to average over process variations.

7. CONCLUSION

A reconfigurable platform well suited for artificial evolution of continuous-time analog filters has been presented. It consists of a dedicated hardware in the form of a field programmable analog array, for which both transistor- and system-level models exist. A genetic algorithm implemented in Matlab delivers synthesis on the dedicated hardware structure from behavioral descriptions of filter transfer functions.

It has been shown that the task of synthesizing higher-order filters on the FPAA can not feasibly be done by traditional design techniques. Instead, it can successfully be accomplished using a genetic algorithm. The heuristics-based approach shows a number of very promising properties compared to more traditional methods: It allows for exploration of the design space unhindered of quantization limitations and non-idealities of the hardware, and can be used for automated synthesis of filter structures on the FPAA.

While it is too early to report on new filter structures that were derived from evolved filters, recent research on this subject seems very promising. Ongoing work includes testing of the FPAA prototypes and implementing the complete genetic algorithm in hardware, which will lead to greatly improved performance, and will allow to study how the evolution copes with intrinsic effects of the underlying hardware. The structures evolved in this way will also be directly usable as working hardware samples.

8. REFERENCES

[1] J. Becker, F. Henrici, and Y. Manoli. System-level analog simulation of a mixed-signal continuous-time field programmable analog array. *IEEE International Workshop System-on-Chip for Real-Time Applications (IWSOC)*, 2005.

[2] J. Becker and Y. Manoli. A continuous-time field programmable analog array (fpaa) consisting of digitally reconfigurable g_m-cells. *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2004.

[3] T. S. Hall, C. M. Twigg, J. D. Gray, P. Hasler, and D. V. Anderson. Large-scale field-programmable analog arrays for analog signal processing. *IEEE Transactions on Circuits and Systems I, vol. 52, pp. 2298 - 2307*, 2005.

[4] F. Henrici, J. Becker, A. Buhmann, M. Ortmanns, and Y. Manoli. A continuous-time field programmable analog array using parasitic capacitance g_m-c filters. *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2007.

[5] J. Langeheine, J. Becker, S. Fölling, J. Schemmel, and K. Meier. A CMOS FPTA Chip for Intrinsic Hardware Evolution of Analog Electronic Circuits. In *The Third NASA/DoD workshop on Evolvable Hardware*. IEEE, 2001.

[6] S. Pavan and Y. Tsividis. *High Frequency Continuous Time Filters in Digital CMOS Processes*. Kluwer Academic Publishers, Boston, 2000.

[7] A. Stoica, D. Keymeulen, R. Zebulum, A. Thakoor, T. Daud, Y. Klimeck, R. Tawel, and V. Duong. Evolution of analog circuits on field programmable transistor arrays. In *The Second NASA/DoD workshop on Evolvable Hardware*. IEEE, 2000.