Mapping Algorithm for Large-scale Field Programmable Analog Array[*]

Faik Baskaya, Sasank Reddy, Sung Kyu Lim, Tyson Hall, and David V. Anderson

School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332
{baskaya, sreddy, limsk, tyson, dva}@ece.gatech.edu

ABSTRACT

Modern advances in reconfigurable analog technologies are allowing field-programmable analog arrays (FPAAs) to dramatically grow in size, flexibility, and usefulness. With these advances, analog circuits and systems can be programmable, reconfigurable, adaptive, implemented on standard CMOS to take advantage of scaled CMOS technology, and at a density comparable to digital memories. Our goal in this paper is to develop the first physical design automation toolset for floating-gate based FPAA with focus on minimization of parasitic effects on FPAA interconnect. We provide graph-based analog circuit and FPAA device modeling suitable for efficient mapping. Our FPAA clustering algorithm constructs Computational Analog Blocks (CAB) from analog circuit elements while improving the utilization of the device and reducing its impact on the total number of routing switches used. Experimental results demonstrate the effectiveness of our approach.

Categories and Subject Descriptors

B.7.2 [**Design Aid**]: Placement and routing

General Terms

Algorithms, Design

Keywords

Floating gates, Field Programmable Analog Array, mapping

1. INTRODUCTION

While digital processors can perform the desired functions, there are many cases where an analog design can offer the same functionality at a fraction of the power required for the digital solution. Modern advances in reconfigurable
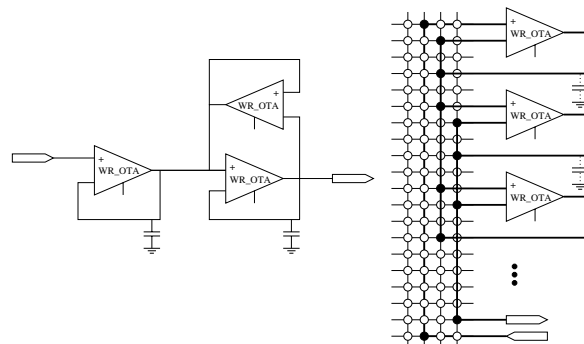

Figure 1: An analog circuit and its mapping onto our floating-gate based FPAA.

analog technologies are allowing field-programmable analog arrays (FPAAs) to dramatically grow in size, flexibility, and usefulness. With these advances, analog circuits and systems can be programmable, reconfigurable, adaptive, implemented on standard CMOS to take advantage of scaled CMOS technology, and at a density comparable to digital memories. Our goal in this paper is to develop the first physical design automation toolset for floating-gate based FPAA with focus on minimization of parasitic effects on FPAA interconnect.

On the otherhand, FPAAs still have not achieved the same success as FPGAs in the digital domain even with the grown interest, availability and use of FPAAs. This results from several factors, including the lack of CAD tools, small circuit density, small bandwidth and layout dependent noise figures. These factors are all related to each other, making the design of a high performance FPAA a multi-dimensional problem. A critical reason behind these difficulties is the non-ideal programming technology, which contributes a large portion of parasitics into the sensitive analog system.

The floating-gate transistors used in our FPAAs are standard pFET devices whose gate terminals are not connected to signals except through capacitors. Because the gate terminal is well insulated from external signals, it can maintain a permanent charge, and thus, it is an analog memory cell similar to an EEPROM cell. We propose to develop the first mapping algorithm for floating-gate based large-scale FPAA. The possibility of programmable analog technology has potential to penetrate the market from simple one-parameter programmable elements to large-scale signal processing front-end systems. A few companies have already

started investigating using analog floating-gate elements as simple trimming elements for analog applications, but these approaches only start to unlock the potential of this technology. An illustration of analog circuit mapping using our FPAA is shown in Figure 1.

The existing CAD works for FPAAs target switch capacitor-based FPAAs (Motorola) and include behavioral synthesis [10, 3], technology mapping [2], and place-and-route [10, 9, 1]. However, these algorithms are designed for small-scale FPAAs and thus are not applicable for our large-scale floating-gate based FPAA. In addition, the placement and routing constraints in our floating-gate based FPAA are radically different from the switch capacitor-based FPAA. Since the major parasitic effects on FPAA chips are due to parasitic resistance and capacitance on FPAA interconnects, our goal is to minimize the overall routing switches used while satisfying various device/wire-related constraints.

The remainer of this paper is organized as follows. Section II presents the analog circuit and FPAA device modeling. Section III presents the problem formulation. Section IV presents the signal degradation modeling. Section V presents our FPAA clustering algorithm. Experimental results are shown in Section VI, and we conclude in Section VII.

2. DEVICE AND CIRCUIT MODELING

2.1 FPAA Device Modeling

A floating-gate element is a polysilicon layer that has no contacts to other layers; this floating-gate can be the gate of a MOSFET and can be capacitively connected to other layers. Charge on the floating-gate is stored permanently, providing a long-term memory, because it is completely surrounded by a high-quality insulator. Since the floating-gate voltage can modulate a MOSFET's channel current, the floating-gate is not only a memory, but also can be an integral part of a computation. The charge on the floating-gate is modified through a combination of hot-electron injection and electron tunneling, and the modifications can occur simultaniously to the computation being performed [4, 6, 7]. The small size and scalability make these approaches ideal for integration with classical analog techniques (e.g. voltage references, filters, ADCs or DACs) as well as larger-scale analog signal processing techniques (e.g. compression or classification for audio or image signal processing). Therefore, with both digital and analog signal processing modalities feasible, more options are now available when designing a signal processing system.

The computational logic in the FPAA is organized in a compact computational analog block (CAB) that consists of op-amps, transistors, multiplier, programmable capacitors, edge detectors and filters. An illustration is shown in Figure 2. CABs are tiled across the chip in a regular mesh-type architecture with busses and local interconnects in-between. The major parasitic effects on FPAA chips are due to parasitic resistance and capacitance on FPAA interconnects. Therefore, the primary objective during mapping is to minimize the total number of wires and switches involved in each interconnect.

We model the given FPAA architecture with an undirected graph $A(V, E)$, where V denotes a set of CABs, I/O cells, local crossbars, and global crossbars, and E denotes a set of local wires, crossbar wires, global wires, and I/O
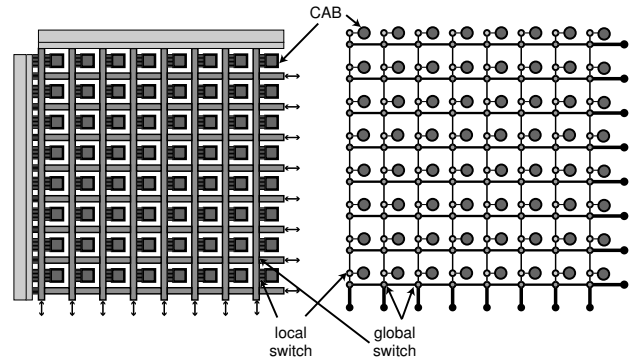

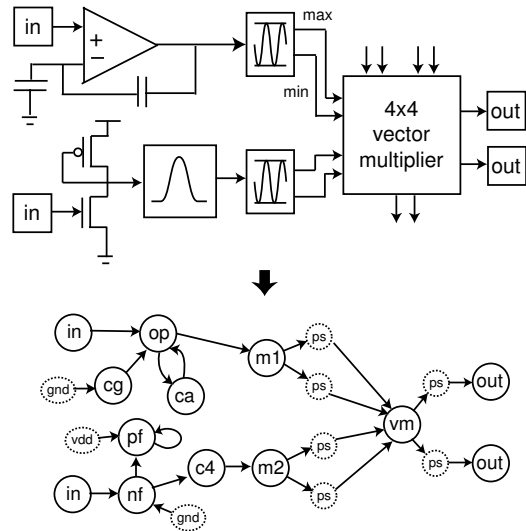Figure 3: Illustration of graph-based FPAA device modeling.

Figure 4: Illustration of analog circuit modeling

wires. We model the given analog circuit with a directed graph $G(V, E)$, where V denotes a set of passive, active, pseudo, and I/O elements in the circuit and E denotes a set of connections among the elements. An illustration of our architecture modeling is shown in Figure 3.

2.2 Analog Circuit Modeling

Analog circuit modeling can be divided into two phases: analog circuit description and element modeling. Analog circuit description, the process in which a circuit that will be analyzed is represented in terms of its topology and element values, has two major challenges. The first challenge is to make the description human readable, and the second challenge is to make it programmatically friendly. In order to satisfy these constraints, a SPICE-like netlist was chosen to describe the circuits. Each element in the circuit is specified by an element line that contains the name and the circuit nodes to which the element is connected to. The element name is an alphanumeric string which begins with a reserved name and ends with a number representing a unique node. Thus, vm1, vm2, cg1, ca1, and op1 are all valid element names representing different nodes in a circuit. Input and output pins are described in a similar fashion.
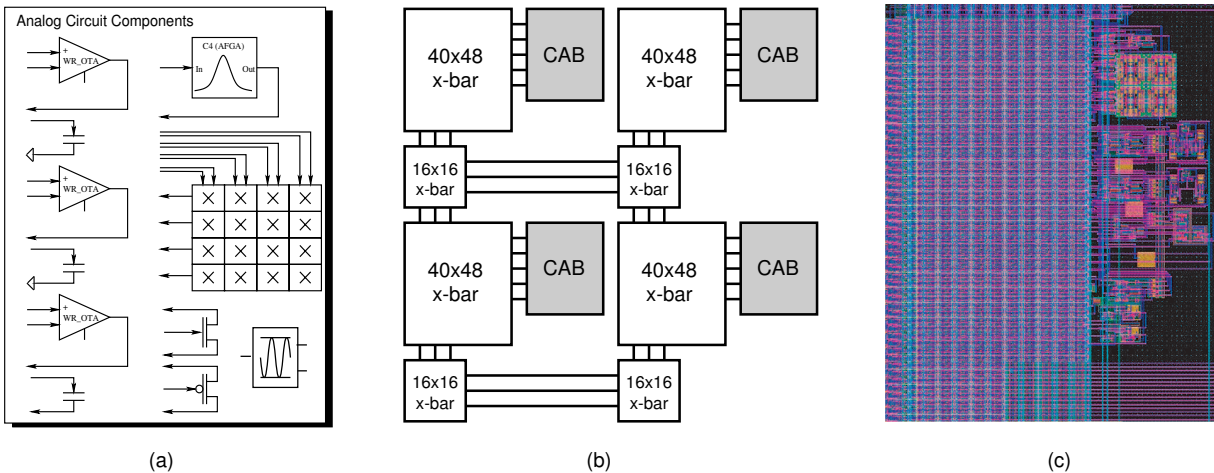
Figure 2: (a) **Computational Analog Block (CAB) for an FPAA based on floating-gate devices, where each CAB contains a four-by-four matrix multiplier, three wide-range operational transconductance amplifiers (OTAs), three fixed–value capacitors, a capcatively coupled current conveyor (C^4), a signal-by-signal multiplier, one pFET, and one nFET. (b) overall block diagram for a large-scale FPAA. The switching interconnects are fully connectable crossbar networks built using floating-gate transistors, (c) layout of a single CAB and its crossbar.**

More complex circuits can be created by using sub-circuit files. Basically, a generic circuit is described with an input/output node interface list and a similar netlist as a regular circuit but without unique numbers representing each element. This sub-circuit then can be used multiple times in a circuit description without having to re-write every element in the sub-circuit description. Element modeling is the next phase after analog circuit description. Each element is modeled as one or more vertices in a directed graph depending on the number of output terminals. Unlike digital gates, analog devices may have multiple output terminals. Also, each vertex must drive only one net. In order to support such a device, pseudo vertices and pseudo nets are introduced into the graph. Figure 4 illustrates a sample analog circuit and its corresponding directed graph.

3. PROBLEM FORMULATION

We divide the FPAA physical synthesis process into three steps, namely, FPAA clustering, FPAA placement, and FPAA routing:

- FPAA Clustering: the input is the netlist along with FPAA architecture. The output is a clustered netlist that satisfies the device-related resource constraints, i.e., number of components and IO ports in each CAB. The primary objective is to minimize the amount of inter-cluster connection, which has positive impact on total number of switches needed after routing. The secondary objective is to pack each cluster to its capacity in order to minimize the number of CABs needed.

- FPAA Placement: the FPAA placement problem is to seek a 1-to-1 mapping between the set of clusters we obtain from FPAA clustering and the set of CABs in the given FPAA architecture. In addition, we map I/O nodes in the given circuit to I/O cells in the FPAA. The primary objective is to minimize the estimated number of switches needed after routing. The sec-

ondary objectives include total/maximum wirelength and reduced congestion (= improved routability).

- FPAA Routing: the FPAA routing problem is to map each edge in G to a set of wires in the given FPAA architecture. Each intra-cluster edge in G may use local wires, whereas each inter-cluster edge may only use vertical and possibly horizontal global wires. The objective is to complete the routing while minimizing the performance degradation. Our goal is to minimize the total parasitics along the interconnect, and this is done by minimizing the number of various types of wires and switches along the interconnects.

The focus of this paper is to develop algorithms for FPAA clustering.

There are three types of possible interconnects between the components. These can be called as intra-CAB, inter-CAB/intra-column and inter-column interconnects. Each of these interconnects are essentially wires which are connected to the components or other types of interconnects via floating-gate transistor switches. Due to the fact that each interconnect type contains different number of switches and the switches still have a loading effect on the circuits even when they are closed, different interconnect types have different effects on the circuit performance.

Intra-CAB interconnects are local interconnects within every CAB. For this reason, they are also referred to as internal nets, or i-nets in this paper. These interconnects correspond to the local wires and several switches on them, which can only be used to connect the components within a CAB. Since they don't go all the way from the top row of CABs to the bottom and contain less total number of switches than other types of interconnects along the wire, the overall parasitic effect of switches on the interconnect is less and is desirable for connections that require less loading. These interconnects can not be used for connecting a component to another component if they are clustered in different CABs.

Inter-CAB interconnects correspond to the vertical global wires on columns of the FPAA. Since these wires (along with their inter-column counterparts to be described below) establish connections between the components clustered into different CABs, they are also called as external nets, or x-nets in this paper. There are also switches enabling access to the inter-column interconnects on these wires. When a connection has been established on an inter-CAB/intra-column interconnect, that wire is dedicated to this connection from the very top row to the bottom. Hence, these interconnects have to be shared between the CABs in a column, and are very valuable resources. An inter-CAB/intra-column interconnect has more loading effect on the circuit compared to an intra-CAB interconnect.

In order to establish connection between the components in different FPAA columns, a horizontal global wire, namely, an inter-column interconnect is required. These wires have accesses to the vertical global wires via floating-gate transistor switches. Since these wires go all the way from the first column to the last, they cause a considerable loading on the circuit as well.

Intra-CAB and inter-CAB/intra-column interconnects are all determined at the end of clustering phase. On the other hand, inter-column interconnects can be determined only after the placement phase.

4. PERFORMANCE DEGRADATION

The impact of interconnect parasitics strongly depends on whether the interconnect is an internal or an external signal path. An *internal path* connects two elements in a single CAB, two CABs, or one CAB and input I/O cell. An *external path* connects a CAB to an output I/O cell. The performance degradation from an external path, denoted D_{ext}, is simply given by the sum of all wires and switches parasitics along the path.

By introducing the *capacitor attenuation parameter* α, the parasitic effects on FPAA interconnects are modeled as capacitor attenuation. In order to estimate the circuit performance degradation caused by the parasitics on an FPAA signal path i, the capacitors that are charged or discharged through the signal path i are attenuated by a factor of α during the circuit analysis. The capacitance attenuation parameter α_i for path i is defined to be the ratio between the capacitance at the destination CAB and the source CAB. The computation of α_i involves computing parasitics along i between the source and sink CABs. Let V_l^{id} denote the ideal voltage of internal paths contributing to the output l. Thus, V_l^{id} is a function of capacitors along the paths: $V_l^{id} = f_l(c_1, c_2, \ldots, c_k)$. Let $S_{c_j}^l$ denote the *sensitivity* of output signal l with respect to capacitor c_j. Then, $S_{c_j}^l = (\partial f_l/\partial c_j) \times (c_j/f_l)$. Finally, the performance degradation from an internal path is given by:

$$D_{int} = \sum_{l=1}^{L} w_l \left| \sum_{j=1}^{k_j} S_{c_j}^l (\alpha_j - 1) \right|$$

where α_j is the capacitor attenuation factor for a path that drives capacitor c_j.

Assuming a single CAB is driving an output cell, the computation of D_{ext} is straightforward. For each output, we compute the total wire resistance between the CAB output and the output cell, which will in turn determine the corre- sponding output attenuation factor. On the other hand, the computation of D_{int} is much harder due to its high runtime and space complexity. For a given output v, we need to examine all possible paths that connect to v. However, there exists an exponential number of directed external paths for a single output in a directed graph G. In addition, for each CAB-to-CAB connection e, we may need to store sensitivity value for all output cells since e may lead to all output nodes. Thus the time complexity is exponential and space complexity is quadratic. During our physical synthesis optimization, therefore, it is desirable to compute estimated values of D_{int} and D_{ext} instead of exact values. The parasitics of wires and switches along the paths to output cells play the major role. Therefore, we focus on minimizing the number of switches on the intra-CAB and inter-CAB interconnect as mentioned in Section 3 for performance degradation minimization.

5. FPAA CLUSTERING ALGORITHM

5.1 Constraints, Objectives and Challenges

There are two main types of structural constraints in the FPAA architectures, namely, device and net constraints. There is a certain number of each device or analog circuit block in each CAB. These numbers determine the device constraints. Furthermore, we don't have infinitely many wires and switches for connecting the devices and circuit blocks to each other. The wires for connecting components of the same CAB to each other are called as internal nets whereas the remaining wires that are used to connect devices from different CAB's to each other are called as external nets. Each type of net has its own limits and all these limits in addition to the device constraints are given in a device file.

In the proposed FPAA clustering algorithm, we accept user defined constraints in addition to the structural constraints. The components specified by the user defined constraints have to be clustered into the same group under all conditions. Due to the hierarchical support of the cell data structure, we can create a new cell which will cover all the cells in this group and treat the whole cluster as a single cell.

Unfortunately, the x-net limits described above are not exclusive to each CAB but they must also be observed by the CABS which will share the same column at the end of the placement phase. This x-net sharing condition shapes our objective during FPAA clustering, which is to minimize the number of used CAB's while keeping the number of i-nets as high and the number of x-nets as low as possible. The constraint on number of x-nets for the proposed FPAA architectures is so hard that at this time we had to give priority to reducing the cut size between the clusters rather than being concerned with the weight of the nets.

Another challenge comes from one of the components in the FPAA structure, which is called as Vector Matrix Multiplier (VM). Since it is a rather large circuit compared to the others, it comes with only a few of all CAB's. So, this component is given higher priority in determining the order for clustering the cells.

This is not the only challenge brought on by VMs. Due to its high number of input and output terminals, a VM can easily violate the x-net limitation by itself. Actually, not only VM but also the cell groups formed by the user defined constraints described above can introduce similar

```
CAB Selection
1:  best = NULL;
2:  for (each ordered cell i)
3:      for (each CAB c)
4:          if (c is available for i)
5:              if (rank(c) > rank(best))
6:                  best = c;
7:      if (best == NULL)
8:          for (each CAB x with x-net violation)
9:              if (x-net_reduce(x))
10:                 best = x;
11: return best;
```

Figure 5: Pseudo code for CAB select algorithm.

```
x-net_reduce(c)
1:  while (x_net(c) > x_net_limit)
2:      for (each neighbor n of c)
3:          if (c is available for n)
4:              pkey[n] = # nets in n not link to c;
5:              skey[n] = # nets in n link to c;
6:      if (no n is available)
7:          return FALSE;
8:      add n with max skey[n] with min skey[n];
9:  return TRUE;
```

Figure 6: Pseudo code for x-net reduction algorithm.

challenges. This would cause problems in a straightforward clustering algorithm since there would be times that there are no available CAB's in sight and the search would stop. This problem is overcome by the x-net reduction algorithm we propose later.

In the following sections, different steps of FPAA clustering algorithm are described.

5.2 Pre-clustering and Cell Ordering

In this step, cells corresponding to the components that have to be in the same cluster under all conditions due to user defined constraints or other possible reasons are grouped together under a higher level cell. The I/O cells are also separated from the remaining cells in this step.

Once the groups are formed, all non I/O cells are put into an order for clustering. The atomic cells already pre-clustered will be exempt from ordering and clustering. Instead their parent cells will be ordered and clustered.

During ordering, the three main groups are treated differently. VM type cells and other cells that include VM type cells are given the highest priority. Then comes the user defined groups of cells. Finally, the remaining cells follow after being ordered according to Modified Hyper Edge Coarsening scheme [5]. The motivation is to give higher priority to the cells in the same net to group together first, thereby reducing the inter-CAB connection.

5.3 CAB Selection

A pseudocode for the CAB selection algorithm is presented in Figure 5. The cells to be clustered are already ordered in the previous step. Next, for each of these cells every CAB is scanned for availability and then ranked. Ranking is done based on the improvement of occupancy of the cab, increase in the number of i-nets and decrease in the number of x-nets when the given cell is assigned to the CAB of interest. The CAB with the highest rank is selected for assignment.

A CAB may be unavailable for a given cell due to device and net constraints. If addition of the cell violates the device constraints or i-net constraints (local switch limits), there is no way that this selection may be feasible. X-net violation, on the other hand, is a different issue. Allowing temporary violation of x-net constraints may later result in an increase in i-nets within acceptable limits. Thus, the cabs which violate only x-nets are kept in a separate list, called as x-net-violate-only cabs in an order such that the cabs with lower overflow value are in front. These cabs are tested for x-net reduction algorithm (line 10) in the case that no available CAB's may be found for assignment. X-net reduction algorithm is explained in the following section.

5.4 X-net Reduction

X-net reduction algorithm described in Figure 6 is inspired by Prim's minimum spanning tree algorithm [8]. The objective of this algorithm is to reduce the number of x-nets of the cluster assigned to the given cab to an acceptable level. This value could be the x-net limit or even lower. Every neighboring cell available for this cluster is given a key, which is the number of nets of this neighbor cell not shared with the cluster. Then the neighbor with the minimum key is selected for assignment to the cab containing the cluster. If there are several cells all having the same minimum key value, then we look at the maximum secondary key holder among these cells. Secondary key is the number of nets of a cell which are also shared with the given cluster.

X-net reduction continues until the number of x-nets are below the desired value (success), or the cab is no more available for neighboring cells (no success).

6. EXPERIMENTAL RESULTS

We implemented our algorithms using C++/STL, compiled with gcc v3.3, and tested on an Apple Powerbook loaded with MAC OS X Panther v10.3, running with a 1.5 GHz PowerPC G4 processor, and containing 1 GB of RAM. The benchmark generation and clustering processes were independent C++/STL programs.

The algorithms were tested in 4 different FPAA architectures of various sizes with 5 benchmark circuits using each architecture. Currently, there are two available CAB types. These types are given in Table 1. Table 2 summarizes different FPAA architectures. Due to the different number of rows in each architecture, each FPAA column may accomodate more CABs. Thus, inter-CAB/intra-column interconnect limitations have to be adjusted accordingly. Therefore, different number of global wires per column are allowed for each architecture. The 20 benchmark circuits being used are displayed in Table 3. Sizes of the benchmark circuits range from 10 components with 17 nets up to 500 components and 707 nets.

Clustering algorithm has been applied to 20 benchmark circuits in 4 different FPAA architectures and the results

Table 3: FPAA clustering results

ckt	benchmark circuits			total usage			average utilization			time
	#comp	#nets	arch	CAB	lsw	gsw	comp	i-nets	x-nets	(ms)
a1	10	17	fpaa1	2	18	18	47.70%	45.00%	40.00%	10
a6	20	27	fpaa1	4	40	19	48.41%	50.00%	22.50%	20
a9	16	23	fpaa1	4	28	20	38.41%	35.00%	22.50%	20
a12	18	22	fpaa1	4	29	17	45.00%	35.00%	20.00%	20
a15	10	18	fpaa1	2	22	16	47.27%	55.00%	35.00%	10
b4	89	128	fpaa2	16	151	119	53.24%	46.25%	33.75%	770
b11	74	115	fpaa2	14	117	136	50.13%	40.71%	41.43%	240
b12	111	160	fpaa2	17	192	144	62.67%	55.29%	38.82%	630
c1	187	253	fpaa2	37	269	262	49.21%	35.68%	32.70%	2060
c5	207	300	fpaa2	47	292	350	42.30%	30.64%	33.19%	8010
d1	267	384	fpaa3	60	399	427	42.68%	32.50%	31.50%	16080
d2	302	434	fpaa3	63	419	503	46.20%	32.70%	36.19%	22420
d3	307	438	fpaa3	62	461	473	47.55%	36.61%	34.03%	22830
d4	297	411	fpaa3	62	415	455	46.51%	32.90%	33.39%	19610
d5	265	371	fpaa3	50	368	431	51.20%	36.00%	38.20%	15290
e1	423	602	fpaa4	92	639	638	44.34%	34.24%	31.20%	103470
e2	396	555	fpaa4	80	563	617	47.98%	34.75%	34.63%	84770
e3	403	567	fpaa4	75	626	568	51.87%	41.07%	34.53%	89150
f4	500	707	fpaa4	101	792	707	47.70%	38.32%	31.68%	163820
f5	474	680	fpaa4	106	734	734	42.96%	33.77%	30.38%	143990

Table 1: FPAA CAB types

comp	cab0	cab1
opamp	3	3
cap	1	1
cap (grounded)	2	2
vm multiplier	0	1
minmax	1	1
pfet	1	1
nfet	1	1
c4 filter	1	1
local wires	10	10

Table 2: FPAA architectures

comp	fpaa1	fpaa2	fpaa3	fpaa4
dimension	4×4	8×8	12×12	16×16
cab0	4	16	36	64
cab1	12	48	108	192
global wires/col	6	10	15	20
total global wires	24	80	180	320

obtained are displayed in Table 3. Depending on the benchmark circuit complexity, number of generated clusters range from 2 up to 106. The number of local switches obtained at the clustering phase will be the final total number of switches whereas the number of global switches given in the table represent only those which connect the CABs to the vertical global wires. The total number of global switches can be known only after the placement is finished and the switches connecting vertical wires to the horizontal wires are determined.

For each cluster, average utilization of components is within 40% to 60% margin, which is acceptable due to the fact that priority is given to reducing the number of external connections and the limited number of internal connections. Utilization of external connections is lower than that of internal connections and components, which is also not a bad thing, for it will make the further steps of mapping easier by giving more room to sharing of external connections in a column. Internal connections, on the other hand could be better utilized than the obtained results, since it would further reduce the number of external connections and this limitation will not be revisited in further steps once the clustering is complete.

7. CONCLUSIONS

In this paper we present the problem formulation and algorithms for clustering targeting floating-gate based FPAA. Since the major parasitic effects on FPAA chips are due to parasitic resistance and capacitance on FPAA interconnects, our goal is to minimize the overall routing switches used while satisfying various device/wire-related constraints. Our ongoing work includes placement and routing as well as path length balancing heuristics. Signal degradation modeling and analysis is also underway. Our FPAA mapping tool will be used to evaluate and improve candidate FPAA architectures for a variety of signal processing designs.

8. REFERENCES

[1] S. Ganesan and R. Vemuri. FAAR: a router for field-programmable analog arrays. In *Proc. Intl. Conf. on VLSI Design*, pages 556–563, 1999.

[2] S. Ganesan and R. Vemuri. Technology mapping and retargeting for field-programmable analog arrays. In *Proc. Design, Automation and Test in Europe*, 2000.

[3] S. Ganesan and R. Vemuri. Behavioral partitioning in the synthesis of mixed analog-digital systems. In *Proc. ACM Design Automation Conf.*, 2001.

[4] P. Hasler, C. Diorio, B. A. Minch, and C. A. Mead. *Advances in Neural Information Processing Systems 7*,

chapter Single transistor learning synapses, pages 817–824. MIT Press, Cambridge, MA, 1995.

[5] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning : Application in VLSI domain. In *Proc. ACM Design Automation Conf.*, pages 526–529, 1997.

[6] M. Kucic, P. Hasler, J. Dugger, and D. V. Anderson. Programmable and adaptive analog filters using arrays of floating-gate circuits. In E. Brunvand and C. Myers, editors, *2001 Conference on Advanced Research in VLSI*, pages 148–162. IEEE Computer Society, March 2001.

[7] M. Kucic, A. Low, P. Hasler, and J. Neff. A programmable continuous-time floating-gate fourier processor. *IEEE Transactions on Circuits and Systems II*, 48(1), 2001.

[8] R. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, 36:1389–1401, 1957.

[9] H. Wang and S. Vrudhula. Performance driven placememnt and routing for field programmable analog arrays. In *Proc. Intl. Conf. on Mixed Design of Integrated Circuits and Systems*, pages 207–212, 2001.

[10] H. Wang and S. Vrudhula. Behavioral synthesis of field programmable analog array circuits. *ACM Trans. on Design Automation of Electronics Systems*, pages 563–604, 2002.