Hierarchical Placement for Large-scale FPAA

I. Faik Baskaya, Sasank Reddy, Sung Kyu Lim, and David Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology
{baskaya, sreddy, limsk, 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. Our goal in this paper is to develop the first hierarchical placement algorithm for large-scale floating-gate based FPAAs with a focus on the minimization of the parasitic effects on interconnects under various device-related constraints. In our hierarchical approach, our FPAA clustering algorithm first groups analog components into a set of clusters so that the total number of routing switches used is minimized and all IO paths are balanced in terms of routing switches used. Our FPAA placement algorithm then maps each cluster to a Computational Analog Block (CAB) of the target FPAA while focusing on routing switch usage again. Experimental results demonstrate the effectiveness of our approach.**

I. 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 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.

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 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.

There currently exist several CAD works for FPAAs in the literature including behavioral synthesis [1], [2], technology mapping [3], [4], and place-and-route [1], [5], [4]. However, these works focus only on small-scale, switch-capacitor based designs. On the other hand, our floating gate-based FPAAs contain up to 256 (16×16) complex CABs, thereby necessitating more scalable approach to handle the complexity. In addition, the device and interconnect constraints in our floating-gate based FPAA are radically different from the switch capacitor-based FPAA. Since the major parasitic effects on our 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.

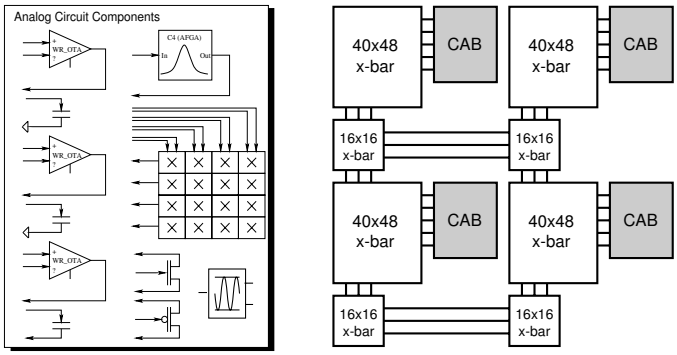

Fig. 1. (a) Computational Analog Block (CAB) for an FPAA based on floating-gate devices, where each CAB contains a four-by-four matrix multiplier, three wide-range operational transconductance amplifiers (OTAs), three fixed–value capacitors, a capacitively coupled current conveyor (C^4), a signal-by-signal 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.

II. Problem Formulation

A. 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 [6] 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-

0-7803-9362-7/05/$20.00 ©2005 IEEE

scale analog signal processing techniques (e.g. compression or classification for audio or image signal processing).

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 1. 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. Unlike digital circuits, these parasitics have a cumulative impact on performance of analog circuits. Therefore, the primary objective during mapping is to minimize the number of switches connected to a signal path and also to balance the different signal paths.

B. FPAA Hierarchical Placement Problem

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 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.

The hierarchical placement problem defined in this graph consists of two parts: The first part is clustering the circuit into several partitions. The input to the FPAA clustering problem is an analog netlist along with a target 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. The second part is placement which is to seek a one-to-one mapping between the set of clusters obtained from FPAA clustering and the set of CABs in the given FPAA architecture. In addition, we map I/O ports in the given circuit to I/O nodes in the FPAA. The primary objective is to minimize the estimated number of switches needed after routing.

III. FPAA Interconnect Analysis

The routing resource in our floating-gate based FPAA is shown in Figure 2. There exist three kinds of routing switch boxes: local, vertical, and horizontal crossbar. The local crossbar is used to establish connection among components in the same CAB. The vertical crossbar is used to connect components from CABs in the same column. Lastly, the horizontal crossbar is used to connect components from CABs in different columns. The routing switches in local, vertical, and horizontal crossbars are respectively called local, vertical, and horizontal switches. Each column in the vertical crossbar extends all the way from top to bottom, while each row in the horizontal crossbar extends all the way from left to right.

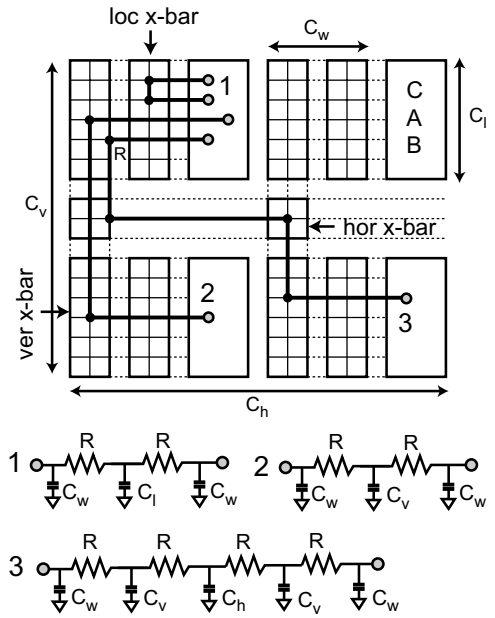

Fig. 2. Routing resource for a 2×2 FPAA that consists of local, vertical, and horizontal crossbrs. Three types of interconnects (1: intra-CAB, 2: inter-CAB/intra-column, 3: inter-column) are also shown.

Each row or column in these crossbars can only be used by a single interconnect. In addition, once a net is occupying a row (or column), the sum of the resistance and capacitance of *all* routing switches in this row (or column) contribute to the delay of the interconnect. Each switch contributes a parasitic diode capacitance from the drain of the switch to ground regardless of the switch being on or off. This is the depletion capacitance, and the worst case (highest) value of this capacitance is when the switch is off [7]. Since there is a number of switch capacitances in parallel on a line, the total capacitance of the line is assumed to be the sum of all these off capacitances. The difference that comes from only one switch being on is neglected. When a switch is turned on, there will also be a resistance in series on the order of 10 kΩ [8]. Each switch contributes a pole when added into the circuit path and has a negative impact on the bandwidth.

The following three kinds of interconnects are possible depending on the placement of components (see Figure 2):

- intra-CAB wires (type 1): these wires connect components in the same CAB using the switches in local crossbar. We model the resistance of local switches that are turned on (= R). The capacitive component of the wire includes all local switches in the entire row (= C_w) and column (= C_l) occupied by the wire. The parasitics of these wires are minimal.
- inter-CAB/intra-column wires (type 2): these wires connect components from different CABs located in the same column. We model the resistance of vertical switches that are turned on (= R). The capacitive component of the wire includes all vertical switches in the entire row (= C_w) and column (= C_v) occupied by the wire. The parasitics of

these wires are between those of type 1 and 3 wires.
- inter-column wires (type 3): these wires connect components from differen CABs located in different columns. We model the resistance of vertical and horizontal switches that are turned on (= R). The capacitance of all vertical and horizontal switches along the rows (= C_w and C_h) and columns (= C_v) occupied by the wires are modeled. The parasitics of these wires are maximal.

We assume that $C_w < C_v$, $C_w < C_h$, and $C_l < C_v$. In case the interconnect contains more than two components (= multi-pin net), each source-to-sink connection can be individually modeled. The type 1 wires are alternatively called *i-nets* in this article, whereas type 2 and 3 are called *x-nets*.

Due the the exclusive usage of routing tracks (= rows and columns in the crossbars) by the interconnect, the number of available routing tracks imposes a strict design constraint. For example, the number of vertical tracks available for each CAB column is 10 in our current 8×8 FPAA design. The reason for this strict constraint is due to the inverse relation between the number of routing switches in each track and bandwidth. Architectural design space exploration can reveal good tradeoff points between the number of routing switches vs bandwidth degradation.

IV. FPAA Clustering Algorithm

A. Overview of the Approach

The primary aim in FPAA clustering is to pack the circuit components that are closely connected to each other within the same CABs. This process is constrained by device, internal net and external net limitations. The objective is to utilize the device and internal net resources of the CABs as much as possible while keeping the external net use lower in order to make room for the placement phase. While CAB selection is carried out by ranking all the CABs and picking the best among them, there are several alternatives that can be taken for cell selection.

B. Constraints and Objectives

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 or i-nets whereas the remaining wires that are used to connect devices from different CAB's to each other are called as external nets or x-nets. Each type of net has its own limits and all these limits in addition to the device constraints are defined in a device file. In the proposed FPAA clustering algorithm, we also accept user defined constraints which tell the components that have to be in the same cluster.

The objectives of clustering is to minimize the inter-CAB connection and maximize the CAB device utilization. In order to minimize the former, increasing the number of i-nets and decreasing the number of x-nets are encouraged for each CAB.

C. Cell Selection

In this step, the cells in the graph corresponding to the components in the circuit netlist are put in an order according to different criteria and selected for clustering in this order. The ordering criteria will be explained below.

1) Pre-clustering: User constraints dictate that certain cells have to be placed in the same cluster under all conditions. In addition, pseudo cells and master cells they are associated with have to be grouped together. All these cells are grouped under higher level cells that will be treated as single cells after this phase. The cells that are not grouped under a higher level cell are called as atomic cells.

The Input-Output (IO) cells and the non-IO cells are also separated from each other in this step. IO cells are not considered in clustering except for their connections to the non-IO cells.

2) Cell ordering: Once the cells are grouped, all non-IO cells are put into an order for clustering. During ordering, three cell types may be encountered. The cell to be clustered may be an atomic cell, a group of pre-clustered atomic cells, or a Vector Matrix Multiplier (VM) cell. VM is a large component, so it is not available in most CABs in the FPAA. Thus, giving the highest priority to this type in ordering is reasonable.

Cells may be ordered using different methods. In this work, two different approaches were used. According to the first approach, VM cells and all other cells are ordered randomly within their own groups. According to the other approach, VM cells and group cells are ordered with respect to their sizes and the atomic cells are ordered according to the Modified Hyper Edge Coarsening scheme [9]. This is a net-driven approach and the motivation is to give higher priority to the cells that belong to the nets containing less components, thereby reducing the inter-CAB connections. Net-driven approach is not implemented alone. In stead, we also take path lengths into consideration and try to balance them.

3) Path length balancing: A directed acyclic graph corresponding to the same circuit is analyzed for determining arrival and required times of each cell and a time slack value is extracted. The less the time slack value of a cell is, the more critical that path is. So, if lower time slacked cells are clustered first, the paths will be expected to be more balanced. This approach is not implemented alone. Instead, time slack values are used as tie breakers when there is an equality in the net-driven approach. Thus, this approach can be called as net/path driven approach.

D. CAB Selection

A pseudocode for the CAB selection algorithm is presented in Figure 3. 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

```
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;
```

Fig. 3. 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 connecting to c;
5:              skey[n] = # nets in n connecting 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;
```

Fig. 4. Pseudo code for x-net reduction algorithm.

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 described in Figure 4 is inspired by Prim's minimum spanning tree algorithm. 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).

```
Constructive FPAA Placement
1:  sort CABs based on xnets;
2:  for (each CAB c)
3:      for (each column n)
4:          if (c.type is in n)
5:              if (c.xnetsused < n.xnetlimit)
6:                  add c to n;
7:                  sort columns based on xnets;
8:          else
9:              break;
10: return n;
```

Fig. 5. Pseudo code for constructive FPAA placement algorithm.

V. FPAA Placement Algorithm

A. Overview of the Approach

FPAA placement is the process of performing a one-to-one mapping between the set of Clustered CABs (CCAB) that are generated after performing the clustering algorithm and Physical CABs (PCAB) that exist in the FPAA architecture. Furthermore, the I/O nodes that appear in the CCABs are mapped to the corresponding I/O nodes, which we call as external nets, or xnets in short, in the PCABs. These xnets may connect several PCABs in the same or in different FPAA columns.

B. Problem Formulation

The placement problem consists of several constraints. First, the type constraints must be met in the PCAB structure. Since the physical FPAA structure can contain several different types of CABs based on architectural differences, the placement algorithm needs to properly correspond the CCAB types to PCAB types. Another constraint of the placement problem is that each column in the physical FPAA structure can only have a certain number of CABs associated with it. Furthermore, each column consists of a certain number of each type of CABs. Thus, the placement algorithm has to meet the type and number constraints for the columns in the physical FPAA structure. The placement algorithm also has to take xnet limitation of the PCABs and columns of PCABs into consideration. Since the xnets on an FPAA column may be shared among its PCABs, number of xnets associated with each CCAB can not be simply added. In stead, the repeated xnets are counted only once.

The objective of this placement is to reduce the loading effects of interconnects and switches. At this time, we try to reduce the total number of switches and this requires reducing the number of columns that each xnet traverses through. This comes from the fact that every connection from a PCAB to another requires one and only one switch to one of the vertical buses and for every wire in other columns to be connected to this xnet, one more switch will be required in a horizontal bus. Thus, the objective is to minimize the repetition of xnets in different columns.

C. Constructive FPAA Placement

The initial process, referred to as Constructive FPAA Placement (CFP) (see Figure 5), is focused around finding a viable solution that can be refined by the Stochastic Refinement process. During the CFP process, the CABs that are formed during the clustering process are arranged in descending order. A column data structure that can contain a set of CABs is created as well. The columns are arranged in ascending order based on number of external nets that are used. Each column data structure also contains type limitations. Since the clustered CABs can be of different types, based on if there is a vector multiplier present or not, the algorithm has to consider the types as the clustered cabs are placed in the columns. The clustered CABs are placed in columns if the type restriction and the external nets limit are satisfied. If one of the restrictions fails, than the next column is considered and the restrictions are applied again. This process repeats until the CAB is placed in a column. If the CAB doesn't get placed then the process fails. If the process succeeds than the next CAB is considered. When the CFP process completes, a viable solution is provided to the Stochastic process.

D. Stochastic Refinement

Constructive placement method is not intended to yield optimum results in terms of number of required horizontal-global switches (hgsw). Simulated annealing is a widely used and well developed probabilistic placement method and is employed as the last step of this hierarchical placement tool. It uses the result of the preceding step as an initial placement and searches better placements swapping the CABs randomly.

For this problem, the cost is a weighted sum of hgsw and maximum of number of x-nets in each column. If the CAB swap operation decreases the cost, this change is kept. If not, it is kept with a probability that decreases at every iteration. This mechanism prevents from getting stuck at a local minimum far from reaching the global minimum. The probability of allowing a cost increasing change is $e^{\frac{-\Delta C}{temp}}$. The temperature changes from 400000 down to 1 and is multiplied by a coefficient every iteration. This coefficient is also a function of temperature (0.95 in the medium temperatures, 0.8 in the high and low temperatures).

VI. Experimental Results

We implemented our algorithms using C++/STL, compiled with gcc v3.3, and tested on a Dell Latitude X300, running on a 1.2 GHz Pentium M processor with 632 MB of RAM. The algorithms were tested in 4 different FPAA architectures of various sizes with 18 benchmark circuits using one of these architectures. Currently, there are two available CAB types: type 0 (cab0) includes 3 op-amps, 1 capacitor, 2 grounded capacitors, 1 vector multiplier, 1 min-max calculator, 1 pFET, 1 nFET, 1 C4 filter, and 10 local wires. The configuration of the type 1 (cab1) is the same except that it does not include the vector multiplier. Table I summarizes different FPAA architectures. Due to the different number of rows in each architecture, each FPAA column may accommodate 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 18 benchmark circuits being used are displayed in Table II. Sizes of the benchmark circuits range from 10 components with 17 nets up to 474 components and 680 nets.

TABLE I
FPAA ARCHITECTURES

	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 (per column)	8	22	41	48
global wires (total)	32	176	492	768

TABLE II
BENCHMARK CIRCUITS

ckt	comp	net	arch	ckt	comp	net	arch
1	10	17	fpaa1	10	74	115	fpaa2
2	20	29	fpaa1	11	187	253	fpaa3
3	14	24	fpaa1	12	218	311	fpaa3
4	20	31	fpaa1	13	267	384	fpaa3
5	16	23	fpaa1	14	423	602	fpaa4
6	97	131	fpaa2	15	396	554	fpaa4
7	89	128	fpaa2	16	403	567	fpaa4
8	96	135	fpaa2	17	385	545	fpaa4
9	95	136	fpaa2	18	474	680	fpaa4

A. FPAA Clustering Results

Clustering algorithm with two different cell ordering methods (baseline random vs. net/path-driven) has been applied to 18 benchmark circuits in 4 different FPAA architectures and the results obtained are displayed in Table III. Depending on the benchmark circuit complexity, number of generated clusters range from 2 up to 108. CAB utilization for both cases is about the same. During clustering, the main objective is to minimize the inter-cluster connection, which will naturally result in utilization of more internal nets for each CAB. Increasing the number of i-nets within allowed limits is actually awarded during the CAB selection process. Obviously, there is a correlation between the i-net and local switch (lsw) utilization. As Table III suggests, net/path-driven method makes better use of the lsw than the baseline method. Use of external nets on the other hand, requires more concern. X-nets will continue to be valuable resources in the following phases, so it would be wise to leave some room for the placement phase. So having less vgsw is desired, which is the case for the net/path-driven method.

Another difference between the two methods can be observed in path balancing. Net/path driven method results in smaller maximum path lengths and less maximum-minimum differences but higher average path lengths. Since these values suggest less variation, path lengths of the circuits clustered using net/path driven method are more likely to be balanced.

B. FPAA Placement Results

Placement phase consists of two steps. Results of both steps are displayed in Table IV for comparison. The primary objective of the placement step is reducing the number of horizontal

TABLE III

FPAA CLUSTERING RESULTS

	random ordered clustering						net/path-driven clustering					
	device usage			path balance			device usage			path balance		
ckt	cab	lsw	vgsw	max	ave	max-min	cab	lsw	vgsw	max	ave	max-min
1	2	20	16	14	11	4	2	20	16	14	11	4
2	4	34	25	23	21	3	4	34	25	20	20	0
3	3	26	24	18	16	3	3	28	22	15	15	0
4	4	36	28	29	27	2	5	36	28	32	31	3
5	3	30	19	22	20	4	3	26	23	22	20	4
6	20	105	173	174	78	149	20	138	140	130	71	112
7	17	132	138	90	80	20	16	145	125	87	77	19
8	22	109	183	74	45	50	22	152	140	53	31	36
9	16	113	171	49	36	24	16	163	121	62	53	20
10	14	119	134	58	33	41	14	117	136	57	32	33
11	37	181	350	135	104	61	37	245	286	124	99	49
12	43	270	391	204	103	192	43	335	326	172	99	160
13	59	357	469	109	57	84	58	398	428	126	104	38
14	92	527	755	894	458	878	93	640	642	799	413	783
15	80	487	704	262	175	181	78	592	599	214	155	162
16	75	519	675	277	259	35	75	574	620	212	211	1
17	77	463	692	456	130	439	75	575	580	411	158	394
18	108	608	859	440	282	406	106	716	751	378	251	329
ratio	1	1	1	1	1	1	1.006	1.158	0.901	0.924	1.027	0.732
time	728.27s						717.91s					

TABLE IV

FPAA PLACEMENT RESULTS

	constructive		refinement	
ckt	maxx	hg-sw	maxx	hg-sw
1	8	4	6	4
2	8	18	8	16
3	8	10	6	10
4	8	18	7	18
5	8	16	7	16
6	22	105	15	89
7	22	100	14	85
8	22	104	15	62
9	22	100	14	77
10	22	108	19	74
11	41	243	22	184
12	41	274	23	180
13	41	355	32	223
14	48	562	34	335
15	48	532	34	375
16	48	556	34	390
17	48	521	32	340
18	48	630	41	362
ratio	1	1	0.704	0.669
time	4.357s		5.438s	

global switches (hgsw). Although we can not observe much improvement in the smaller circuits, the reduction in hgsw for the larger circuits seem quite satisfactory.

During the constructive placement step, the only aim is to find a valid column for every CAB as long as there is enough room for the CAB or its x-nets, so the maximum x-nets observed in any column is equal to the upper limit at the end of the placement for all given circuits. The results suggest that the refinement step can considerably reduce the maximum number of required x-nets. This can be utilized by allowing temporary violation of the x-net constraints in the constructive placement step to compensate for later in the refinement step, so that more circuits can be successfully placed.

VII. 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 routing as well as path length balancing heuristics.

ACKNOWLEDGMENT

This research has been supported by the National Science Foundation under contract CNS-0411149.

REFERENCES

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

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

[3] ——, "Technology mapping and retargeting for field-programmable analog arrays," in *Proc. Design, Automation and Test in Europe*, 2000.

[4] ——, "A methodology for rapid prototyping of analog systems," in *Proc. IEEE Int. Conf. on Computer Design*, 1999.

[5] ——, "FAAR: a router for field-programmable analog arrays," in *Proc. Intl. Conf. on VLSI Design*, 1999, pp. 556–563.

[6] P. Hasler, C. Diorio, B. A. Minch, and C. A. Mead, *Advances in Neural Information Processing Systems 7*. Cambridge, MA: MIT Press, 1995, ch. Single transistor learning synapses, pp. 817–824.

[7] P. E. Allen and D. R. Holberg, Eds., *CMOS Analog Circuit Design*. Prentice-Hall, 2002.

[8] T. Hall, C. Twigg, P. Hasler, and D. V. Anderson, "Application performance of elements built in a floating-gate FPAA," in *Proceedings of the International Symposium on Circuits and Systems*, Vancouver, BC, May 2004.

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