A Generic Reconfigurable Array Specification and Programming Environment (GRASPER)

Faik Baskaya, David V. Anderson, Paul Hasler, and Sung Kyu Lim
School of Electrical and Computer Engineering, Georgia Institute of Technology
Atlanta, GA 30332
{baskaya, dva, phasler, limsk}@ece.gatech.edu

***Abstract*—Modern advances in reconfigurable technologies are allowing analog circuit designers to benefit from the computational flexibility provided by large-scale field-programmable analog arrays (FPAAs). With the component density of these devices, small analog circuits as well as larger analog systems can be synthesized and tested in a shorter time and at a lower cost compared to the full design cycle. However, automated development platforms and CAD tools for these devices are far fewer than the physical synthesis tools for their digital counterparts. One of the major reasons for this is the considerably higher impact of interconnect parasitics on circuit functionality in the analog domain; therefore, performance metrics must be monitored closely. Our goal in this paper is to present a physical synthesis framework with a generic architecture specification interface and a parasitic extractor for verification of the synthesis results. Our synthesis tool can support a wide range of FPAA architectures and our simulations can successfully predict the performance metrics.**

I. Introduction

In the fast pace of today's electronics design world, time to market has become more crucial than ever. Companies that can effectively use all available resources to cut the design cycle are getting an edge against their competitors. Design process, which is inherently iterative, must avoid the costly failures whether incurred as money or time loss. Furthermore, Application Specific Integrated Circuit (ASIC) solutions are economically feasible only for high volume products. These factors made the use of digital reconfigurable systems very attractive in the form of Field Programmable Gate Arrays (FPGA) since their introduction in the mid 1980's. 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 [1]. 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, and implemented on standard CMOS to take advantage of scaled CMOS technology. On the other hand, FPAAs still have not achieved the same success as FPGAs in the digital domain even with the growing 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. The methods that work for the physical design of digital circuits in FPGAs may not work very well for FPAAs. The criteria for a successful design are different and more complex. For example, signal integrity of an analog circuit is more difficult to maintain and can severely impact the functionality of the circuit; therefore, it is of higher priority than achieving the most compact design. Increased segmentation of wires may result in additional capacitances that can result in undesired circuit behavior after routing. Another challenge to be faced is the fact that parasitic impedances not only deteriorate the performance as in digital circuits but may also destroy the functionality completely, which requires monitoring the impact of parasitics on performance metrics during the synthesis steps. A placement and routing solution that satisfies all device and net constraints may not necessarily be a desired one; the performance often has to be optimized as well.

Various approaches have been taken in implementing structural and parametric programmability of analog circuits on FPAAs in the past. A survey paper [2] reviews some of these works in the area of programmable analog and mixed-signal circuits. Recently, use of fine-tunable floating-gate transistors in the switch fabric allowed development of large-scale FPAAs; where switches are used not only for routing, but also for component biasing and as computational elements by themselves [3], [4]. Most existing CAD efforts for FPAAs in the literature focus only on small-scale, switch-capacitor-based designs [5], [6], [7].[1] On the other hand, [3] and [8] use 32 to 72 Computational Analog Blocks (CAB), necessitating a more scalable approach to handle the increased complexity. The work in [9] addresses this need for [3] or similar topologies; however, more generic tools are required to catch up with ever changing and improving FPAA architectures. This work addresses this need and presents a tool that we call as Generic Reconfigurable Array Specification and Programming Environment (GRASPER) to support a wide range of FPAA topologies without compromising on the placement and routing quality. In addition, special challenges resulting from the unique use of floating-gate switches as computational elements are addressed.

II. Generic Reconfigurable Array Specification and Programming Environment (GRASPER)

A. Input Circuit Interface

In digital circuits, each signal can be driven by the output of a single gate since gate outputs cannot be connected together. Popular netlist formats such as BLIF reflects this limitation well. Analog circuits don't have such a limitation; it is not uncommon to connect the output pins of an output transconductance amplifier (OTA) to the output pin of another OTA. Assignment of input/output pins become even more vague when it comes to basic components such as transistors and capacitors. Therefore, SPICE netlist format was found more suitable for analog circuit description and is used to create input files for our tool. SPICE is a widely used EDA tool and is already familiar to a large community of analog designers [10]. Full SPICE compatibility of input circuits allows simulation of designs using the available SPICE distributions as well.

[0]This work was supported in part by the National Science Foundation under Contracts CNS-0411149 and CNS-0347792.

[0]The authors also thank Leung Kin Chiu for his assistance in obtaining the measurement data.

[1]Anadigm, a leading commercial FPAA vendor, is currently supplying FPAAs with 4 (2×2) simple switch-capacitor based CABs.

The placement phase can accept circuits that use only the available FPAA components, so input netlists are limited to components from a list of subcircuits that correspond to the FPAA components. Subcircuit descriptions for FPAA components are maintained in a technology library, so new component types for future FPAA architectures can easily be supported. GRASPER input interface allows nested subcircuit descriptions for larger system specifications with the requirement of having FPAA components as the lowest level netlist entry. To enter GRASPER commands that are incompatible with SPICE, a special text sequence that can be treated as a comment line by SPICE is used: ”∗ >>”. Each circuit element and signal node in the input netlist is stored as a graph vertex (cell) and a hyper-edge (net) in an undirected graph. The point where a net is connected to a cell is modeled as a pin. Figures 1a and 1b depict the equivalent graph representations of various circuit elements and a circuit block. A sample SPICE netlist is presented in Figure 1c. Once a circuit is captured as a graph, the objective is to map each graph cell (i.e., circuit component) to an actual component on the target FPAA architecture and determine the switches that define a path between pins that need to be connected.

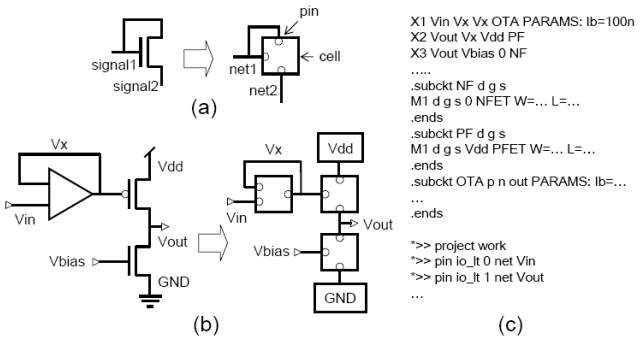

Fig. 1. Graph representation of (a) circuit elements, and (b) a complete circuit with input and output pins, along with (c) corresponding SPICE netlist.

B. Generic Architecture Specification in GRASPER

The placement and routing tool in [9] supports the architectures described in [3]. The changes introduced with new FPAA architectures [8] and possible future architectures to follow dramatically increase the routing possibilities; making it impossible to use the routing resource graph (RRG) and the simultaneous placement and routing method of [9]. Therefore, GRASPER uses a RRG model that depends on capturing the wire-switch relationship as a vertex-edge pair in a simple undirected graph as illustrated in Figure 2a. In a simple undirected graph, edges can exist between two and only two distinct vertices while any number of edges can be connected to a vertex, which reflects the relationship between physical wires and switches perfectly, as each switch has to exist between two and only two wires, and wires don’t have a limitation on how many wires they can connect to via switches. Figure 2b shows how each component pin is paired with a RRG vertex that corresponds to a physical wire connected to the same component pin in the actual FPAA. This is a simple yet effective model that allows many algorithms to be employed easily for placement and routing. As an added benefit, it can be used to specify arbitrary connections via device configuration files allowing most FPAA architectures that use switches as continuous-time pass gates (switched capacitor-based architectures not supported).

A similar model has been used in wireC [11], the triptych FPGA placement and routing [12], and VPR [13]. Since VPR uses a similar

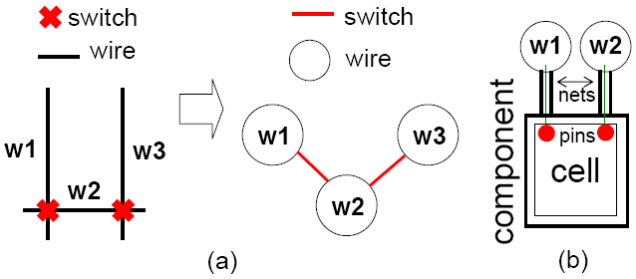

Fig. 2. GRASPER routing resource graph (RRG) fundamentals. (a) Wire-switch relationship on a reconfigurable architecture. GRASPER uses a model where wires are represented as vertices and switches are represent as edges. (b) Wire-component pin connections in GRASPER. This model allows instant determination of terminal wires (source/sink) for routing immediately following the placement of a cell.

RRG model in its core, it also offers flexibility in supported architectures. However, it is designed primarily taking FPGA architectures into account, so does not really use the potentials of the RRG to its full extent. Degrees of freedoms granted when designing the interface took only the cases one would encounter in FPGA design. Although heterogeneous component support, which consists of only additional digital components such as memory and multiplier, was added recently, VPR has only digital delay calculation models rather than transistor level simulation support, and can’t support the CAB types we need. VPR allows only a regular rectangular array of one block per each grid on a regular 2-dimensional array. VPR input interface allows blif and verilog formats; these formats, as already discussed in Section II-A, are not suitable for effectively describing analog circuit netlists. The range of interconnect topologies and CAB array geometries GRASPER interface supports are wider. Although it was not the main objective initially, it is possible to extend GRASPER to place and route digital circuits as well, but VPR can’t support analog circuits.

C. Placement in GRASPER

Because of the increased wire type variety, placement results no longer enforce a unique routing solution in GRASPER. Therefore, placement and routing are divided into separate steps again.

GRASPER placement uses the MHEC-based cell ordering that was also used in [9] to maximize the possibility of local wire use for routing. Unlike [9], GRASPER lets only the components that match the cell type to be visited directly rather than the CABs. Each component that is feasible for placement is ranked for placing the current cell, and the highest rank component is selected for placement after all components are visited.

Pins of CAB components are associated with wires that are connected to the routing resource graph as depicted in Figure 2b. Therefore, once a cell is placed into a CAB component, net information is also transferred to these pin wires. As pins of a net are placed on various locations on the chip surface, a bounding box called as a netbox, begins to form defined by the positions of the wires associated by this net. The larger a net’s netbox becomes, the more distance is likely to be traversed by the wires in the routing phase. In addition, since larger netboxes may occupy more routing resources, they may also increase the chances of congestion. This concern brings another placement objective, which is the minimization of the sum of netbox sizes.

The quality of the placement results can be influenced in a certain direction to achieve different goals. Depending on the placement

priorities, candidate components can be awarded or penalized in different ways during the ranking process. In the current implementation of GRASPER, increased CAB utilization is aimed and awarded. For a different case, this may not be an important issue and a different placement objective can be awarded.

D. Routing in GRASPER

When placement is complete, terminal wires for the circuit nets are determined as the wires connected to the component pins each net is associated with. The objective in routing is to find disjoint minimum spanning trees (MST) within the RRG for all nets where each terminal of a net is also a member of that net's MST. The routing for a sample MST on an undirected graph is illustrated in Figure 3.

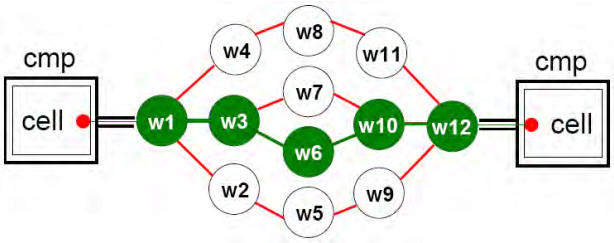

Fig. 3. Grasper routing on the undirected routing resource graph. Source and sink terminals for each net are determined at the end of placement, routing establishes a minimum spanning tree that connects all terminals together.

GRASPER routing uses the maze router approach based on Dijkstra's shortest path algorithm [14]. Maze router guarantees a minimum distance path between two terminals if a path exists. Its weakness is the large memory demands and time consuming operation because of the tendency to radiate toward all directions from a source terminal; however, the limited number of the routing resources and their connections alleviate this effect for the reconfigurable arrays. In addition, the wires corresponding to the vertices in the RRG correspond to longer distances than the equivalent grids in the full custom design, resulting in reduced number of layers or wavefronts that are created at every propagation. This simple method works effectively in GRASPER.

GRASPER routing algorithm consists of two basic steps. In the first step, wires are labeled with their effective distances from a source terminal. This distance can be a cost value that reflects the wire parasitics, such as the number of switches on a wire in our case. The first layer consists of the source terminal only, which is chosen to be the terminal closest to the center of the netbox so that the number of propagations in RRG can be minimized. All vacant wires (i.e. wires that are not used for routing another net) that are connected to the current layer of wires are labeled and added to the next layer of wires. This process is repeated until all terminal wires are labeled.

The second routing step is backtracing through the minimum label wires from the sink terminals to the source terminal. Each wire that is visited during the backtracing is added into the routing path. Backtracing is applied to one sink terminal at a time. Once a routing path is established between the source terminal and the first sink terminal, backtracing the remaining sinks continue until any wire that was previously added to the routing path is encountered.

As mentioned in Section II-C, congestion can cause problems when routing a net. To reduce the chances of net congestion, netbox sizes are minimized during the placement phase. The order nets are chosen for routing also affects the success of routing. If nets that have larger netboxes are routed first, chances of congestion will increase for the subsequent nets. To avoid this, nets are listed in ascending order of size of their netboxes and routed in this order.

In development of GRASPER, specific requirements of the floating-gate based FPAA architectures as well as the opportunities presented by them were also addressed. Traditional switches can be programmed to either on or off positions, whereas floating-gate switches can be programmed to transmit different current values, which correspond to intermediate levels between on and off positions. This unique ability allows use of switch fabric as computational elements in the form of switch elements (SWE) [4]. Physically, A SWE is a routing switch that connects two distinct nets and is programmed to the characteristics desired by the user. In the circuit graph, a SWE is a two-port element associated with a current value connected between two nets. GRASPER can recognize SWEs in the circuit description, and map them to the switch fabric. Since SWEs do not correspond to physical CAB components, they can not be placed during the placement phase. SWE mapping takes place during the routing phase, after routing of all nets in the circuit are complete (except SWE routing, of course). Routing a SWE is establishing a routing path between two disjoint MST of two different nets.

Parasitic capacitances contributed to the circuit by the routing interconnects can be major obstacles to meeting most design specifications. The logical objective in routing would be minimization of wire and switch capacitances to reduce the impact on the circuit performance. On the other hand, capacitors are often used as analog circuit elements, and capacitance contributed by interconnects can be utilized for this purpose as well. This approach not only helps to meet the design objectives but also reduces the demand for on chip drawn capacitors saving much area. GRASPER assigns each net a target capacitance value equivalent to the capacitance between that net and ground as specified in the circuit netlist. After routing each net, total capacitance of wires on the net are computed, and CAB capacitors are added to the net while net capacitance is below the target capacitance.

E. Parasitic Extraction in GRASPER

GRASPER can also extract the parasitic resistances and capacitances contributed to the circuit by adding non-ideal routing wires and switches. Each interconnect is modeled as a subcircuit block which has as many pins as the associated net has. The interconnect subcircuit is constructed using wire segments, which are undivided portions of wires bounded by either an end of the wire or an on switch which connects the wire to another wire, setting the number of wire segments to one more than the number of programmed switches (on-switches) on each wire. Each wire segment is extracted using the information of the wire resistivity, wire capacitance and the capacitance contributions of the off-switches on the wire segment. For accuracy, wire segments are also divided into smaller segments between the grids that a switch may take place, and an RC chain is constructed for the whole segment to account for the distributed parasitic effects. This results in a more accurate network, which comes at a high computational cost for simulators. Applying model order reduction techniques solve this problem by reducing the circuit complexity by preserving a level of accuracy that can be set by the user [15]. Wire segments are connected to other wire segments or the enclosing interconnect subcircuit pins either directly or via the on-switches. On-switch models are also provided in a technology file and can be modified for a trade-off between accuracy and complexity by editing the model.

III. GRASPER Placement and Routing Results

One of the goals in developing GRASPER was to make it independent of a particular architecture, so that different FPAA architectures

exhibiting variations in their wire and CAB topologies can be supported in the future without major updates to the tool. Therefore, we tested 5000 architectures similar to RASP2.8 [8] by varying the number of components and wires, switch matrix densities, and wire segmentation levels. An 8th order butterworth GmC filter was placed and routed on each of these architectures. Parasitic impedances were also extracted to observe the impact of placement and routing. Among these 5000 architectures, 4747 (94.9%) were routed successfully and 4715 (94.3%) demonstrated low-pass filtering functionality with a reasonable deviation from the target frequency response. Simulation results for this experiment are presented in Table I.

TABLE I
PERFORMANCE METRICS FOR AN 8TH ORDER LOW-PASS FILTER BEFORE AND AFTER ROUTING ON 5000 FPAA ARCHITECTURES.

metric	ideal	mean	std
cut-off frequency (fc) [Hz]	9976	10939	1363
pass-band gain (gpass) [dB]	0.585	0.878	0.165
pass-band ripple (rp) [dB]	0.362	3.743	1.683

GRASPER placement and routing results were also tested on an actual circuit and FPAA, using a RASP2.8 FPAA evaluation board powered by HP E3610A power supply, and PCI-DAS 4020/12 scope card to program and measure the response of a 4th order butterworth low-pass filter that is specified using 8 OTAs and 4 capacitors in the circuit netlist. Measurement and simulation results are depicted in Figure 4. In addition to the simulation of the input netlist (pre-pnr), two different post-routing simulations (post-pnr sim1 & sim2) are plotted to demonstrate the effect of the simulation models chosen. In the first case (post-pnr sim1) routing switches are approximated by a 10 $k\Omega$ resistor. In the second case (post-pnr sim2), each routing switch is modeled using a floating-gate transistor; resulting in more complex parasitic interconnects. However, the performance metrics summarized in Table II suggests that this model can simulate the actual circuit performance more accurately.

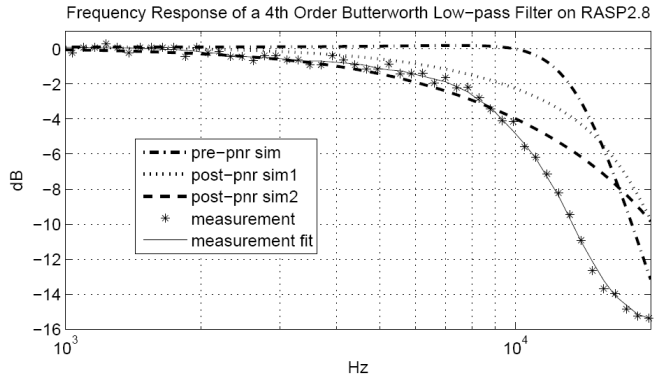

Fig. 4. Simulations of circuits that correspond to the input netlist (pre-pnr sim), parasitic extracted after routing for different routing switch models (post-pnr sim1 & sim2), and measurement (data points and curve fit) for a 4th order butterworth low-pass filter. Measurements have about 2.5 dB gain, which has been removed here for comparison purposes.

TABLE II
PERFORMANCE METRICS OBTAINED FROM MEASUREMENTS AND DIFFERENT SIMULATIONS OF 4TH ORDER LOW-PASS FILTER.

metric	pre-pnr	post-pnr sim1	post-pnr sim2	measurement
fc [Hz]	13804	11641	8222	8500
gpass [dB]	0.097	0.102	0.059	2.5

IV. CONCLUSIONS AND FUTURE DIRECTIONS

GRASPER is currently being used by various research groups, and has been used in workshops and student labs for implementing circuits on the floating-gate based RASP2.8 FPAAs. New architecture configuration files written to support recently manufactured chips will be tested soon. The results summarized in Section III demonstrate the efficiency and effectiveness of GRASPER in supporting a wide range of FPAA topologies. GRASPER is also flexible to accept different levels of simulation models by allowing a trade-off between accuracy and circuit complexity. It finds a good match within the accuracy of the available models and the limitations of the reconfigurable analog devices. As more effective models become available, users will be able to incorporate them into their technology files and improve the quality of the placement and routing results.

REFERENCES

[1] H. Chen, I. Li-Da Huang, M. Liu, and M. Wong, "Simultaneous Power Supply Planning and Noise Avoidance in Floorplan Design," *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*, vol. 24, no. 4, 2005.

[2] D. D'Mello and P. Gulak, "Design Approaches to Field-Programmable Analog Integrated Circuits," *Analog Integrated Circuits and Signal Processing*, vol. 17, no. 1, pp. 7–34, 1998.

[3] T. Hall, C. Twigg, P. Hasler, and D. Anderson, "Developing large-scale field-programmable analog arrays for rapid prototyping," *International Journal of Embedded Systems*, vol. 1, no. 3, pp. 179–192, 2005.

[4] C. Twigg, J. Gray, and P. Hasler, "Programmable Floating Gate FPAA Switches Are Not Dead Weight," in *Circuits and Systems, 2007, ISCAS 2007 IEEE International Symposium on Circuits and Systems*, 2007, pp. 169–172.

[5] H. Wang and S. Vrudhula, "Behavioral synthesis of field programmable analog array circuits," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 7, no. 4, pp. 563–604, 2002.

[6] S. Ganesan and R. Vemuri, "FAAR: A Router for Field-Programmable Analog Arrays," *Proc. Intl. Conf. on VLSI Design*, pp. 556–563, 1999.

[7] ——, "A methodology for rapid prototyping of analog systems," *International Conference on Computer Design*, pp. 482–488, 1999.

[8] A. Basu, C. Twigg, S. Brink, P. Hasler, C. Petre, S. Ramakrishnan, S. Koziol, and C. Schlottmann, "RASP 2.8: A New Generation of Floating-gate based Field Programmable Analog Array," in *IEEE Custom Integrated Circuits Conference, 2008. CICC 2008*, 2008, pp. 213–216.

[9] F. Baskaya, S. Reddy, S. Lim, and D. Anderson, "Placement for large-scale floating-gate field-programable analog arrays," *IEEE transactions on very large scale integration(VLSI) systems*, vol. 14, no. 8, pp. 906–910, 2006.

[10] A. Vladimirescu, "SPICE-the fourth decade analog and mixed-signal simulation-a stateof the art," in *Semiconductor Conference, 1999. CAS'99 Proceedings. 1999 International*, vol. 1, 1999.

[11] L. McMurchie and C. Ebeling, "Wirec 3.2 Tutorial and Reference Manual."

[12] C. Ebeling, L. McMurchie, S. Hauck, and S. Burns, "Placement and routing tools for the Triptych FPGA," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 3, no. 4, pp. 473–482, 1995.

[13] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," *LECTURE NOTES IN COMPUTER SCIENCE*, pp. 213–222, 1997.

[14] C. Lee, "An algorithm for path connection and its application IRE Trans," *Electronic Computer*, vol. 10, pp. 346–365, 1961.

[15] B. Sheehan, "TICER: Realizable reduction of extracted RC circuits," in *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*. IEEE Press Piscataway, NJ, USA, 1999, pp. 200–203.