A High-Level Simulink-Based Tool for FPAA Configuration

Craig R. Schlottmann, *Student Member, IEEE*, Csaba Petre, and Paul E. Hasler, *Senior Member, IEEE*

***Abstract*—With the rapid increase in size and complexity of analog systems being implemented on field-programmable analog arrays (FPAAs), the need for synthesis tools is becoming a necessity. In this paper, we present Sim2spice, a tool that automatically converts analog signal-processing systems from Simulink designs to a SPICE netlist. This tool is the top level of a complete chain of automation tools. It allows signal-processing engineers to design analog systems at the block level and then implement those systems on a reconfigurable-analog hardware platform in a matter of minutes. We will provide detailed descriptions of each stage of the design process, elaborate on a custom library of analog signal-processing blocks, and demonstrate several examples of systems automatically compiled from Simulink blocks to analog hardware.**

***Index Terms*—Field-programmable analog array (FPAA), Simulink, SPICE.**

I. Introduction

TRADITIONALLY, engineers interested in implementing signal-processing algorithms in hardware rely on digital systems such as digital signal processors (DSPs) and field-programmable gate arrays (FPGAs). These platforms benefit from a large body of work and software tools to simplify the compilation to hardware. These software tools use well-developed and intuitive interfaces to allow engineers who may have little-to-no familiarity with ASIC design to benefit from the advantages of dedicated hardware. However, since it is often real-world *continuous* signals that are being processed, the overall system performance can be seriously limited by the conversion power and speed.

One option for increasing the power efficiency of signal-processing systems is to utilize analog in coordination with, or in place of, digital hardware in certain applications. For instance, in audio processing, a bank of analog $G_m - C$ filters can be used on the front end of the digital system to subband the incoming signal, thus reducing the performance demands of the analog–digital converters (ADCs) and the overall power consumed by the system. Analog systems have even been shown to perform certain computations extremely efficient [1], such as scaling and summing, and could find uses as a powerful coprocessor. Given this and the many other advantages of analog signal processing, the use of a *reconfigurable* analog signal processor would create a more accessible design environment (reducing the barrier to entry) as well as decrease the time to market for such a system.

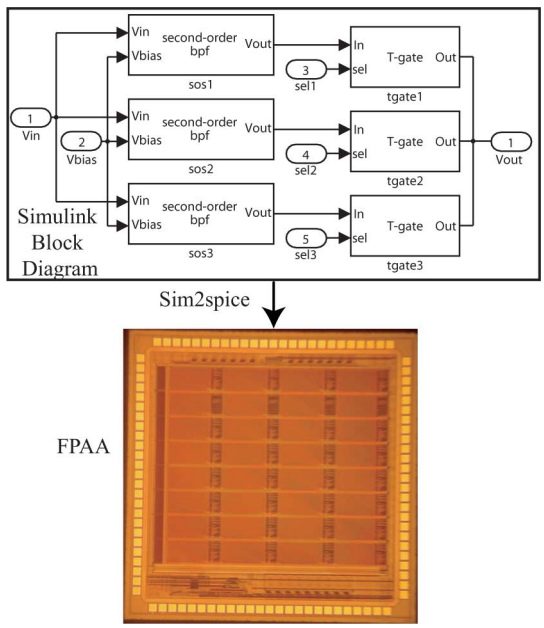

Fig. 1. Sim2spice, the software tool for compiling systems in Simulink block-diagram form down to FPAA analog hardware.

The problem addressed in this work is how to make the benefits of reconfigurable analog hardware available to the broad signal-processing community in the same way that DSPs and FPGAs have done for digital. We propose a software tool, Sim2spice [2], for compiling high-level designs in Simulink onto field-programmable analog arrays (FPAAs), a concept illustrated in Fig. 1. In Section II, we describe the current state of reconfigurable analog systems and the need for synthesis tools. Section III details Sim2spice, the tool for compiling Simulink systems to SPICE, with Section IV finishing up by taking SPICE netlists down to the hardware. We demonstrate a few example systems in Section V and provide final remarks in Section VI.

II. FPAA Technology and Analog Synthesis

FPAAs are reconfigurable VLSI arrays of analog circuit components. The core of such a system is composed of two parts: the computational analog blocks (CABs), which contain the analog primitives, and the routing fabric, which connects the CABs. These devices allow rapid prototyping and implementation of

Manuscript received January 28, 2010; revised June 16, 2010, August 20, 2010; accepted October 17, 2010. Date of publication December 17, 2010; date of current version December 14, 2011.

C. R. Schlottmann and P. E. Hasler are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332-250 USA (e-mail: cschlott@gatech.edu; phasler@ece.gatech.edu).

C. Petre is with Brain Corporation, San Diego, CA 92121 USA (e-mail: csaba.petre@braincorporation.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2010.2091687

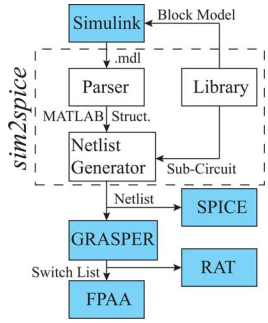

Fig. 2. Complete tool set is comprised of Sim2spice, which converts a Simulink design to a SPICE netlist, and GRASPER, which converts a SPICE netlist to a set of switches for programming on the FPAA.

signal-processing functions in analog hardware. Our group has developed an entire line of FPAAs, with different CAB components in different chips, particularly suited for different applications.

While early versions of FPAAs were modest enough to route circuits by hand using fuse charts, this is quickly becoming impossible. The more recent FPAAs, such as the Reconfigurable Analog Signal Processor (RASP) 2.8 [3], have 50 000 switches and 32 CABs, and they are only getting larger. In addition, newer FPAAs are far more robust than their predecessors and can support larger and more complicated signal-processing systems. With this increase in size and complexity of FPAA systems, higher level synthesis tools are a necessity—hand routing is no longer a reasonable option.

We have chosen Mathworks Simulink as our high-level design space because it allows for the implementation of signal-processing systems in software with an intuitive graphical interface, not to mention the fact that many DSP engineers are already familiar with the program. The user connects together functional blocks and has the ability to simulate the system using a variety of simulation tools. Digital designers have already realized the power of Simulink and have developed software tools to compile Simulink block diagrams to reconfigurable digital hardware on an FPGA, as in [4] and [5]. On the analog front, there have also been tools investigated to allow support for certain FPAAs designed to be done in VHDL-AMS [6]. However, we chose Simulink over AMS as our high-level design space because the graphical block-based user interface is an important complement to the text-based SPICE netlist.

Our tool, Sim2spice, is the top-level compiler of an entire tool chain for configuring FPAAs, a diagram of which is shown in Fig. 2. This tool allows users to utilize our custom library of analog signal-processing blocks to create designs in Simulink and then compile that design down to a SPICE netlist. From there, the GRASPER tool is used to place-and-route the netlist to the FPAA and the Routing and Analysis Tool (RAT) is used to visualize and modify the routing. This complete set of tools now provides a useful interface for engineers outside the analog circuit design field to implement their signal-processing systems and ideas directly in analog hardware.

III. From Simulink to Spice: Sim2spice

The Sim2spice tool is the front end of the toolset; it takes in a model (.mdl) file from Simulink and generates a SPICE netlist, ready for simulation or place-and-route. The program is

```
Block {
  BlockType        Reference
  Name             "VMM 1"
  Ports            [2, 1]
  Position         [105, 88, 305, 152]
  SourceBlock      "ASP_library/VMM"
  SourceType       "VMM"
  elements         "[1 0 0; 0 1 0; 0 0 1]"
  tau              "1us"
  diff_in          "off"
  diff_out         "off"
}
Block {
  BlockType        Reference
  Name             "WTA1"
  Ports            [1, 1]
  Position         [330, 103, 440, 137]
  SourceBlock      "ASP_library/WTA"
  SourceType       "WTA"
  size             "3"
}
```

Fig. 3. Example piece of model (.mdl) file in Simulink. This example shows a representation of the VMM and WTA system, which is described in the Examples section of this paper.

composed of two main parts, shown along with the library in the dashed box of Fig. 2: the Simulink model parser and the SPICE netlist generator.

A. Simulink Model Parser

The parsing of the Simulink model file is done in MATLAB and involves the use of a custom script written in Python. The script is packaged as an executable file, allowing it to be called directly from MATLAB without the installation of Python. The input to the program is the Simulink model file. An example piece of an .mdl is shown in Fig. 3 that displays the vector-matrix multiplier (VMM) and winner-take-all (WTA) blocks. The output of the parser is a MATLAB structure containing the blocks and connections that comprise the model as well as all associated block parameters and values. Python was chosen as the language for the parser due to the ease of parsing text with built-in parsing libraries such as PyParsing [7].

B. SPICE Netlist Generator

The netlist generator takes as input the structure created by the parser from the Simulink model file and converts it to a SPICE circuit netlist, utilizing the circuit netlist elements associated with each block contained in the component library.

First, the netlist generator reads a list of block types from the component library, and then it reads a description file associated with each block type. The description file lists attributes of the block type, such as user-specified block parameters and input/output port parameters. At this point, the parser is called to read the model file and search for blocks and associated parameters. The structure created by the parser contains an array of blocks and an array of lines, or connections between blocks.

Next, the netlist generator makes several passes over the parser's output arrays, finding and naming common circuit nets between blocks. At this point, the lines between blocks can be of a variety of forms: vectorized, single-ended or differential. During compilation, no exhaustive DRC is performed; however, the netlist generator will check if vectorized signals are of the

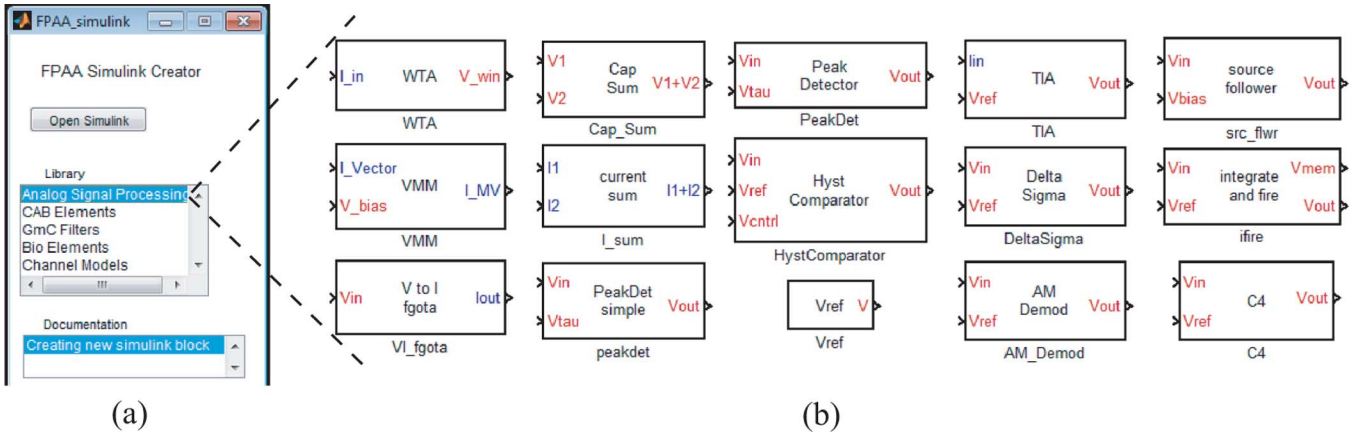

Fig. 4. Component library. (a) The MATLAB GUI presents the user with all of the available component libraries. (b) Expansion of the Analog Signal Processing library. Blocks are continuously being added to the library as other users share their functional blocks.

same dimension between blocks. We also use a color coding scheme in the blocks to match signal type, such as blue for current-mode and red for voltage-mode signals.

The final step involves assembling the actual netlist. The program calls a user-defined MATLAB script, the build file, for each block. The build file receives—as input from the netlist generator—the user-specified parameters for that specific instance of that block type, and returns an array of strings, representing individual lines of the SPICE circuit netlist for that block. The netlist generator combines the netlists for each block in one netlist and keeps global net names unique by making use of subcircuits to encapsulate each block instance. The inport and outport blocks from Simulink become input and output pins in the SPICE netlist, to be converted to I/O pins on the FPAA when placed-and-routed.

C. Component Library

The Simulink block library is our collection of the predefined blocks. This library allows one user to create a functional block and share it with the other users of the Simulink tool. This enables us to design systems with blocks that are already tested on the FPAA, without having to redesign them. In this sense, the library is very much "open source"; anyone who creates a working system is encouraged to build the corresponding Simulink block so that others can utilize the design. Additionally, any component built into the Simulink toolbox is available to the user to help optimize designs. Any such block that is given a circuit equivalent in the library will be realized in the SPICE netlist.

The analog Simulink blocks are defined by level-2 M-file S-functions and their corresponding circuit netlist elements. We have designed the system to make it as easy as possible to add new blocks to the component library. When adding a component to the library, there are four main files that must be written, which are given here.

1) **S-function Simulink block**. The first step is to create the physical block as a Simulink S-function block. The number of ports and their names, as well as the input parameters, are defined here. There is also a field for the designer to write a description about the functionality of the block, which is useful for future users.
2) **Simulink (.m) behavior file**. This file defines the behavior of the block in Simulink simulations. Here, the designer describes what mathematical function the block will perform on the incoming signal. The behavior can be as detailed or as high level as the user wants. For instance, for the WTA block, the user can simply allow the block to output a high or low corresponding to the "winner" or "losers," or try to accurately describe all of the transistor subtleties. Since a true transistor level simulation can be done in SPICE, in the essence of design and simulation time, it is recommended that this block be made as ideal as allowable. A further discussion on macromodeling is given in Section III-D.
3) **MATLAB (.m) build script**. This file tells MATLAB how to build the netlist that corresponds to the block. In general, this file consists of loops that print SPICE subcircuits according to the *size* parameter. The name of this file must be the same as the definition file, with "build_" appended at the beginning.
4) **Description file**. The .desc file defines the list of parameters that the parser will look for when it reads the model file.

Our library now comprises over 60 different parts and is growing constantly as users continue to add new blocks. Currently, we have several libraries of blocks in five different subcategories of functionality including analog signal processing blocks, basic CAB circuit elements, $G_m - C$ filters, bio elements, and neuron channel models.

Fig. 4 shows the GUI which organizes the library into sublibraries. In this figure, the analog signal-processing sublibrary is broken out to show the available components. Fig. 5 shows the dialog box that appears for a specific block, the VMM, allowing the user to edit the parameters of the block. In this case, the user is allowed to edit the matrix elements, the time constant, and the signal representation. During the build cycle, the resulting VMM will consist of floating-gate transistors tiled and programmed according to the *elements* parameter, with the bias current set according to the *tau* parameter. An example of a Simulink system using this VMM is demonstrated in Section V-C of this paper.

One interesting aspect of building an analog component library is that it allows us to define what we consider analog signal processing blocks. This area, surprisingly, is not as defined as it

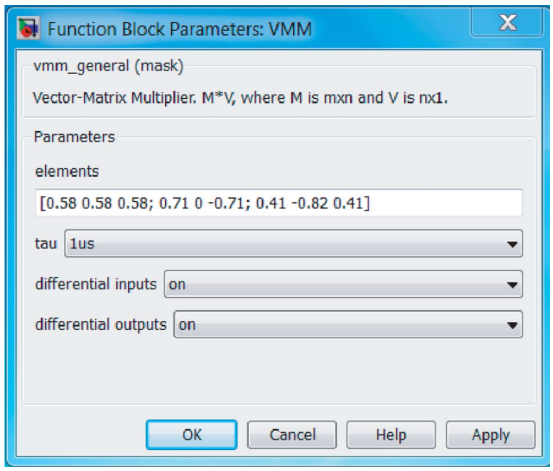

Fig. 5. Block parameter window for the VMM provides a brief statement about the usage of the block and asks the user to specify several needed parameters: the matrix elements, the desired time constant, and whether the inputs and outputs are to be single-ended or differential.

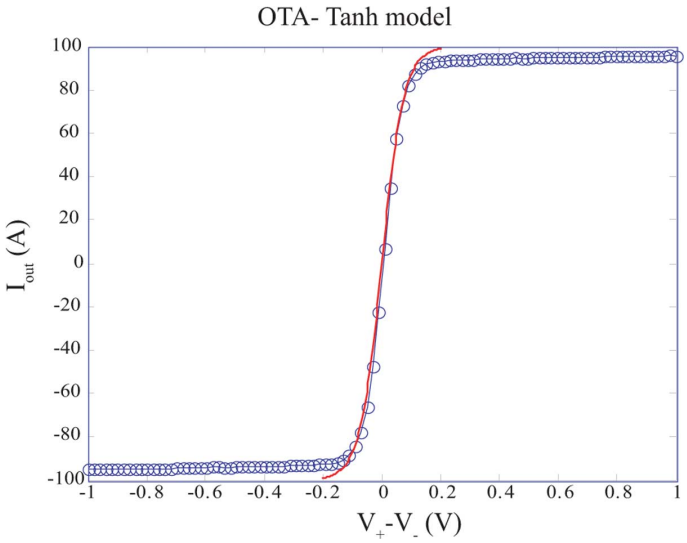

Fig. 6. OTA tanh model with hardware data. The smooth curve is a tanh function which can be used to model the OTA. The bubbles are data taken from the FPAA hardware.

could be. Although there have been books written on the area, notably [8], most analog signal processing is still done at the custom level, without the use of premade blocks. This gives us the freedom to invent the blocks that fit our needs with the hope that once they are built, the community will be inclined to use them as standard analog blocks.

D. Macromodeling

The need for a Simulink behavior file brings up the issue of macromodeling. Macromodeling is the design of a functional block that captures the desired performance, without being overly detailed. While there have been previous discussions on the varieties of macromodels for operational amplifiers (Op Amps) [9], the same design process needs to be extended to the other analog signal processing blocks. In order to design large systems with the Simulink tool, we need to have blocks that are accurate enough to demonstrate that the system works, without being overly complicated that it makes the simulation time too long or muddles the results.

One example of a macromodel design is for systems utilizing many operational transconductance amplifiers (OTAs). As a first pass over the system, the user would want to test the simple functionality. In order to test first-order functionality, as quickly as possible, the user would use the basic tanh function to characterize the OTA. This model is sufficient for testing the dc characteristics containing many OTAs. Fig. 6 shows how closely a simple tanh matches to a real sweep.

Once the general functionality of a system has been established, the models can be made more detailed to show certain design metrics. One metric important to most signal-processing systems is signal-to-noise ratio (SNR). In order to test this, a noise component needs to be added to the blocks. For the OTA we can easily do this by adding MATLAB's noise function to the model. This noise source can be made to have the same power spectral density as the amplifier we expect to use. By using these simple models, with only the features we are directly looking to test, we can get a better feel for the way the system as a whole is working, with a much shorter simulation time.

IV. From SPICE to Analog Hardware

The next step in the process chain is to compile the SPICE netlist onto the analog hardware. The GRASPER tool is used to place-and-route the netlist onto the FPAA, and the Programming & Evaluation board, along with the programming interface tool, is used to target the hardware. Along the way, the systems can be simulated in SPICE, using custom sub-circuits, and the routing can be viewed and edited with the RAT tool.

A. Place-and-Route

GRASPER, developed by Baskaya *et al.*, is the place-and-route tool used for targeting SPICE netlists to the FPAA [10]. The output is a list of switch addresses and the values to which they should be programmed, given in the format: (row, column, prog value). The (row, col) address refers to the desired floating gate's location in the crossbar matrix [11]. The programmed value indicates if this floating gate is intended as a switch or a computational element. A programmed value above approximately 30 μA will result in a switch that is all the way closed (we often simply use the value 1) and any value below this will describe the amount of current that the FG-pFET is programmed to pass with its source at V_{DD}. This list of switches can then be targeted directly onto the RASP family of FPAAs. In the GRASPER netlist input file, the particular FPAA is specified by the device (.dev) file. This .dev file describes all of the important attributes of a given FPAA such as: number and type of horizontal and vertical lines, CAB elements, and I/O lines. By including this file, GRASPER will be compatible with future generations of FPAAs and routing structures. Fig. 7 shows the flow of a circuit being mapped to the FPAA and the corresponding object code.

B. SPICE Library

Although the FPAA's CABs contain predefined circuits, in order for SPICE to accurately simulate a design, these CAB elements need to be implemented as subcircuits. Most of these

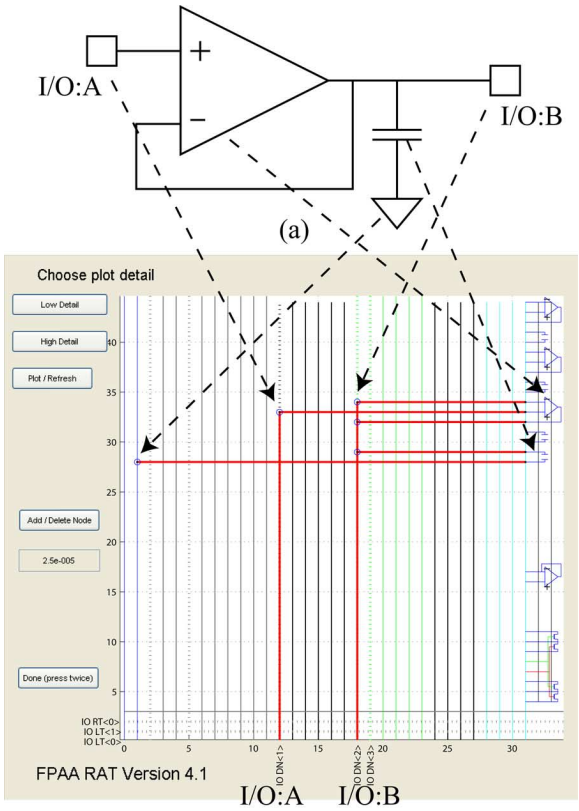

Fig. 7. RAT design. (a) Schematic of a first-order $G_m - C$ filter. This system is complied into an object code that can be programmed onto the FPAA. (b) RAT displaying the connectivity of the filter. The CAB elements are illustrated on the right, with the switch matrix on the left. The nets are highlighted in red to indicate what rows and columns the switches have connected. (c) Object code for the filter consists of a list of the switches that were displayed in the RAT GUI. The first five entries in the list are fully programmed switches, and the last entry is the OTA bias current.

are straightforward one-to-one mappings, such as the MOS elements, 500-pF capacitors, and T-gates. However, several CAB elements use floating gates as programmable current sources, in particular the OTAs. For these, an ideal current source is used in the subcircuit for simulation purposes and that value is then passed to the FPAA as the floating-gate target value.

One problem that arises is how to model the floating-gate elements when they are explicitly used in the circuit. Examples of this occurrence are the floating-gate input OTAs, the MITEs, and the switch elements used for computation. The problem is a result of the rule that in SPICE each node needs a dc path to ground; therefore, gates cannot be left floating.

One popular model is to simply place a dc voltage source on the gate through a large resistor (10^{26} Ω). Although this achieves reasonable results, the dc voltage does not intuitively

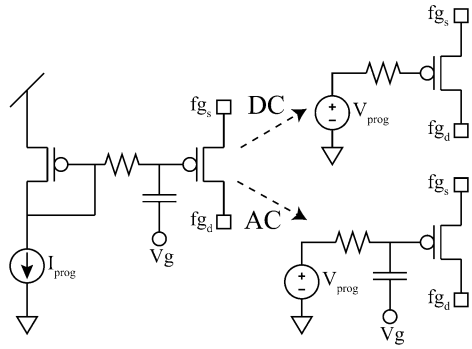

Fig. 8. Floating-gate SPICE model. This model is needed because SPICE cannot implement a true floating-gate transistor. The model closely resembles the indirect programming structure used on the FPAA. Under dc circumstances, the floating gate model resembles the upper equivalent circuit: a normal pFET with a fixed potential on the gate. Under ac conditions, the circuit will act as the lower circuit: the gate programming voltage will be coupled onto the floating node.

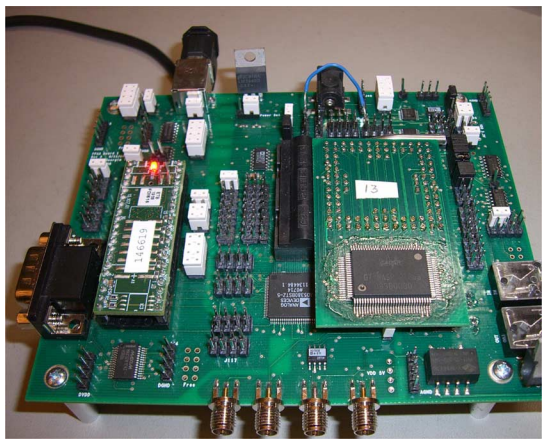

Fig. 9. Evaluation and programming board. This board communicates with MATLAB over a USB connection. The board contains a microcontroller for controlling the programming algorithms, a 40-channel DAC and a four-channel ADC for creating and reading test signals, and all necessary power management circuitry.

translate to the programming current. The model we chose was to force a dc current through an indirect transistor, as shown in Fig. 8. This model was chosen because it allows for the programming current value to be directly used in SPICE and the circuit closely resembles the actual schematic of the indirect programming system [12].

C. RAT Visualization Tool

RAT, developed by Koziol and Abramson, provides a graphical way to view and edit the compiled circuits [13]. This visualization tool has proven invaluable when designing and debugging on the FPAA because it has eliminated the need to draw fuse charts. The input to the RAT is a programming (.prg) file that includes the switch list in the form output by GRASPER. By running the command FPAA_RAT_main(filename.prg), the GUI of Fig. 7(b) is launched. This window shows a zoomable image of the FPAA routing structure and CAB elements. The routing lines are color-coded by type and the I/O ports are clearly labeled. The switches from the input list appear as large black dots connecting the corresponding horizontal and vertical lines, and if a particular switch is used

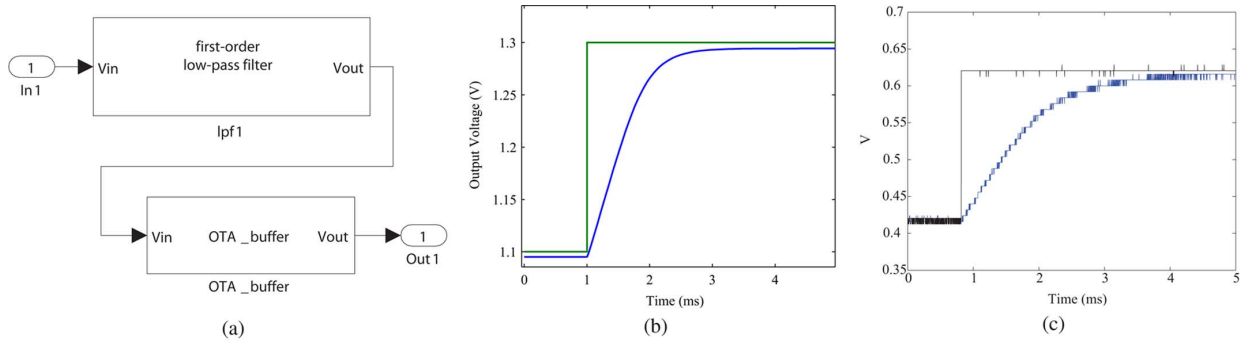

Fig. 10. (a) Simulink model of the low-pass filter. (b) Step response in SPICE simulation. (c) Step response on FPAA.

as a computational element, it is shown with a green circle around it. The lines connecting elements are highlighted in red to easily follow the connectivity of a particular net.

In addition to being able to view a circuit, modifications can be made to it. Switches can be added or deleted and the connectivity highlighting will be updated accordingly. Once modification is complete, the new design can be output into a new .prg file that has the same file name as the input file, but with "_out" appended to the end of it.

D. Evaluation and Programming Board

The custom four-layer printed circuit board (PCB) in Fig. 9 was built to program, communicate with, and test the RASP family of FPAAs. This evaluation board communicates over and is fully powered by USB, but it also has the capability to be powered by a 5-V dc supply and communicate over a serial connection. The board is controlled by an ATMEL ARM microcontroller for handling instructions from the computer using MATLAB commands. It also includes a 40-channel 14-b digital–analog converter (DAC), a four-channel 8-b ADC, audio input/output amplifiers and jacks, and all of the programming circuitry not already on-chip. In order to have maximum control and flexibility, almost every signal is pinned out to a header: all 52 FPAA I/O (4 to SMA connectors), the 40 DAC channels, four ADC channels, and many of the microcontroller and programming lines. The AV_{DD} plane is jumpered so power measurements can be taken.

E. Current FPAA Chips

At the bottom level of the chain, and really the heart of the system, is the FPAA itself. The most recent and advanced line of FPAAs is the RASP 2.8, which was designed in a 350-nm double-poly CMOS process. This new FPAA offers several drastic improvements over its predecessors [14].

With a die size of 3 mm $\times$ 3 mm, the RASP 2.8 was able to contain 32 CABs and incorporate multilevel routing. This new system of routing maintains the flexibility of the previous FPAAs while also providing more dedicated lines. These dedicated routing lines connect each CAB to its four nearest neighbors: top, bottom, left, and right. By providing this nearest neighbor connection, the lines are made shorter and thus have less parasitic capacitance.

Another advancement is the movement of most of the programming infrastructure on-chip [15]. By moving the control DACs, current measurement systems, and the ADCs on chip, we have seen a drastic speed up in programming time. This on-chip migration has also allowed the form factor of the entire system to shrink. Whereas the previous system was the size of a shoebox and contained three separate boards [16], with the on-chip programming we only need the 4.6 in $\times$ 5.6 in board discussed in the previous section.

In addition to the architectural advancements, the RASP 2.8 line was developed with four different varieties of chips for targeted applications. The different models of the RASP 2.8 line all use the same routing and programming infrastructure, but vary by their CAB components.

1) The general-purpose FPAA: RASP 2.8a [17]. This is the most commonly used FPAA and contains common analog building-blocks in its CABs: OTAs, n/pFETs, capacitors, T-gates, floating-gate elements, and Gilbert multipliers.
2) The bio-FPAA [18]: RASP 2.8b. The bio FPAA contains neuron and synapse models in its CABs. By using the reconfigurable nature of the FPAA, the neurons and synapses can be arranged into computational neural networks.
3) The sensor-FPAA [19]: RASP 2.8c. This FPAA contains a specialized sensor interface and capacitive sensor CAB elements.
4) The MITE-FPAA [20]: RASP 2.8d. This FPAA contains multiple-input translinear elements (MITEs) as its computational primitive. The CABs are made up of translinear loops of MITEs which can perform various mathematical operations.

V. Example Systems

In order to demonstrate the capabilities of our high-level Simulink tool, we have constructed the following three example systems. The first, a first-order low-pass filter, illustrates the use of abstracting the circuit parameters away from the user and simply prompting them for functionality-based parameters. The second example, a spiking neuron system, demonstrates the use of a specialized bio-FPAA, RASP 2.8b. The third example system, a VMM-WTA system, demonstrates the use of the switch matrix as a computational element. For these example systems, we highlight the three phases of system verification: Simulink system-level simulations, SPICE transistor-level simulations, and FPAA hardware-level results.

A. Low-Pass Filter

One simple example system we have constructed and tested in Simulink, SPICE, and the FPAA is a low-pass filter block. This particular filter is a first-order $G_m - C$ block and was previously

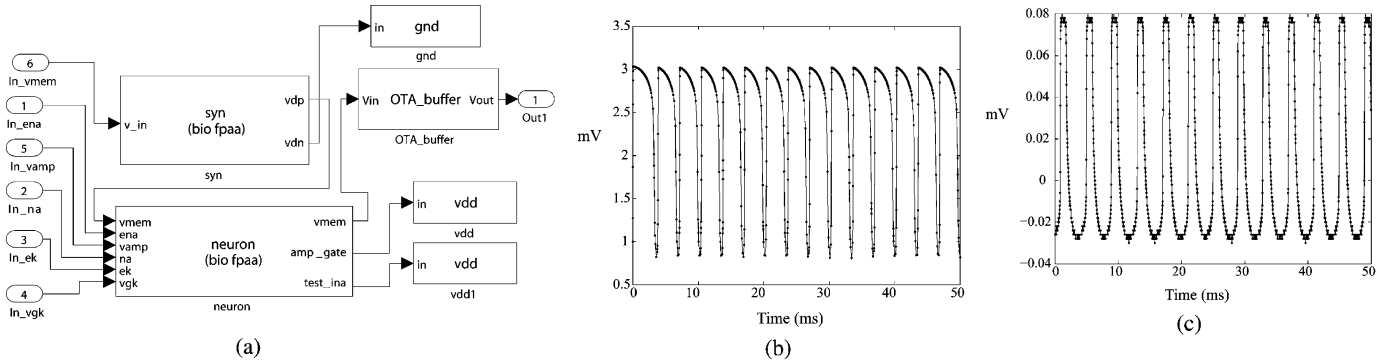

Fig. 11. (a) Simulink model of the spiking neuron system. (b) Simulink simulation of the spiking. (c) Spiking as compiled on FPAA. The complete example is a spiking Hodgkin-Huxley neuron, input block, and output buffer.

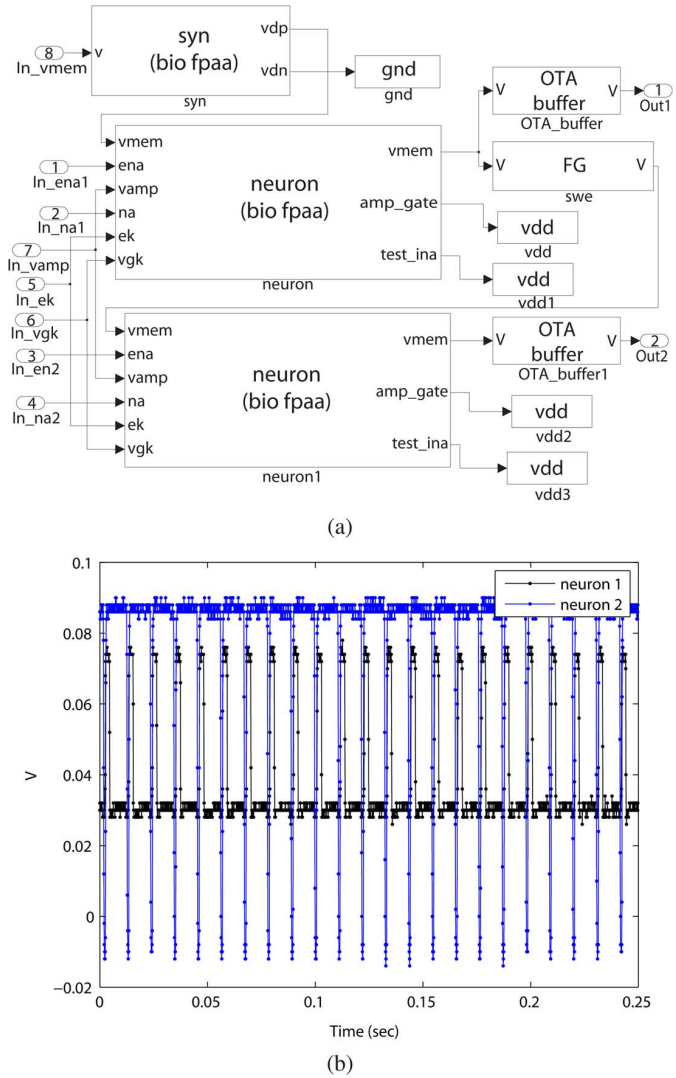

Fig. 12. (a) Simulink model of two neurons and a synapse model. (b) Spiking activity of the two neurons. The pair of neurons demonstrates synchronized spiking due to the coupling through the synapse circuit element.

shown in Fig. 7(a). The transfer function for this filter is given as

$$\frac{V_{\text{out}}}{V_{\text{in}}} = \frac{1}{\tau s + 1} \tag{1}$$

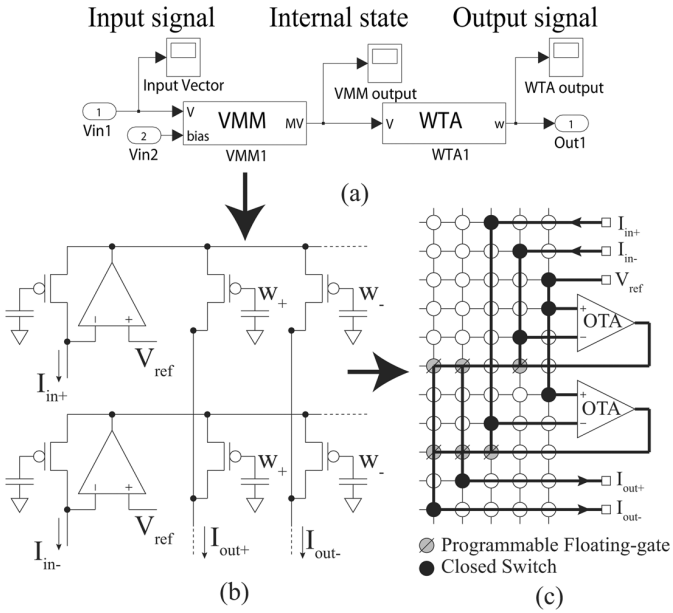

Fig. 13. Three phases of implementation for the VMM-WTA system. (a) The Simulink design. Each of the two blocks, the VMM and WTA, will be compiled based on the custom analog library. The in and out ports will be compiled to I/O pins on the FPAA. The scopes in the figure are for simulation purposes and are not compiled to hardware; they represent the locations of the outputs in Fig. 14. (b) A subblock circuit to which Sim2spice will compile the VMM block. The netlist for this circuit will be tiled in the horizontal and vertical directions according to the parameters specified by the user. (c) The corresponding switch representation to the 2×2 VMM subblock that RASPER will compile. This VMM uses the floating-gate routing fabric for both switches and weight storage.

where τ is the time constant. In the circuit implementation, this time constant is set by (C/G_m), where

$$G_m = \frac{\kappa I_{\text{bias}}}{2U_T} \tag{2}$$

is the gain of the amplifier. In order to abstract the design, in the parameter box, the user is only asked to specify the time constant. By using a fixed-capacitor design, the block will translate this time constant into the gain of the amplifier. In the resulting netlist, the gain is set by the bias current of the OTA, as described in (2). This illustrates the way non-analog designers can take advantage of this tool: they specify a filter of a certain order and its poles, and the compiler will create amplifiers with bias currents. The process of parameter abstraction detailed here can be

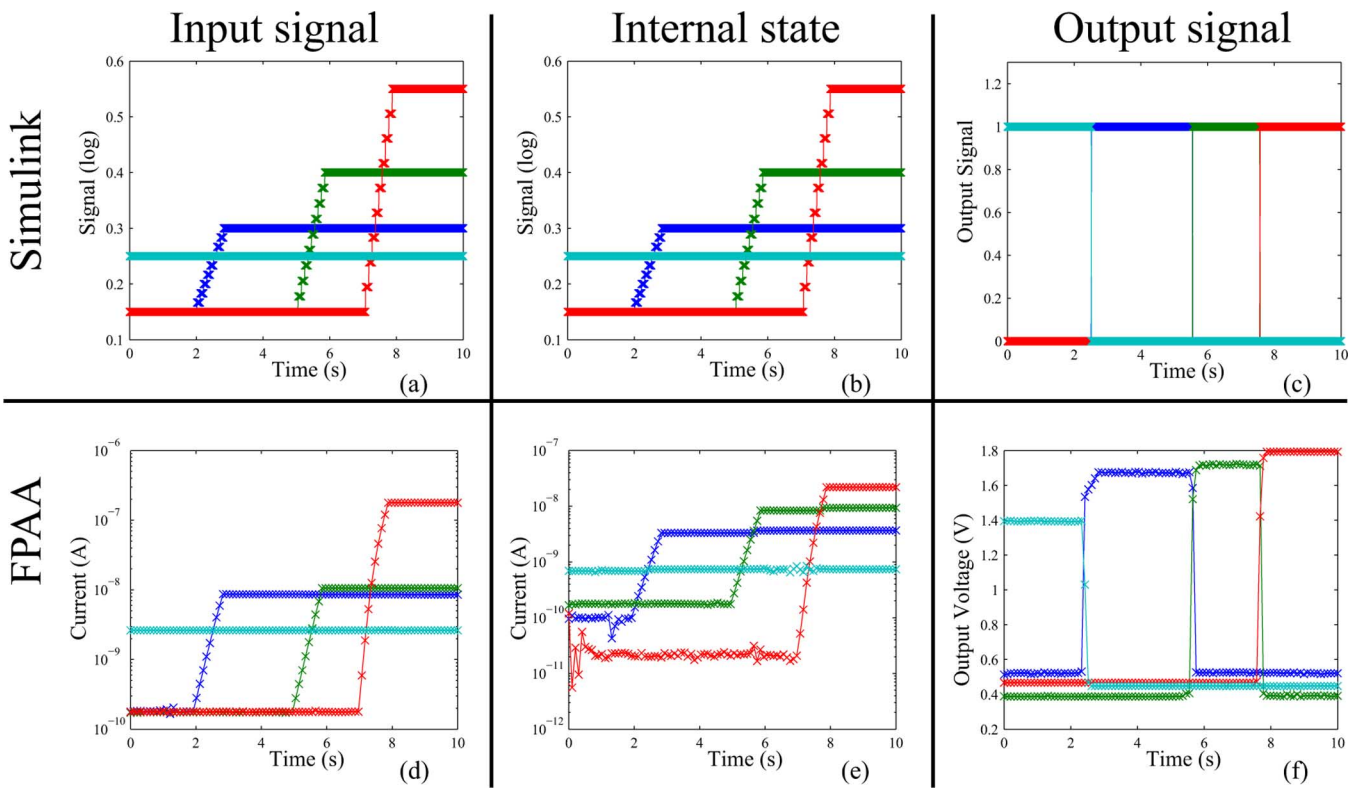

Fig. 14. Output of the VMM-WTA system is compared between the Simulink response and the real hardware data from the RASP 2.8a FPAA. The first row of plots corresponds to the output of the Simulink simulation, and the second row is from the FPAA. The columns represent a common state of the system between the Simulink simulation and the FPAA. The columns correspond to the scopes in Fig. 13(a). (a) The input vector evolves in such a way that each component has a time having the highest value. Here, the inputs are dimensionless signals. (b) The Simulink VMM matrix in this example is the identity, so the output of the VMM is equal to its input. (c) The output of the WTA shows the boolean output corresponding to the input that is highest at a given time. (d) The input vector for the hardware VMM-WTA system is the same as the Simulink example. The input to a hardware system must have physical dimensions; in this, case we used currents. (e) As in the Simulink case, the output of the identity VMM is equal to the input. (f) The output of the hardware WTA demonstrates the same boolean trend as shown in the simulation.

extended to any block. Fig. 10 shows the Simulink block-level diagram, SPICE step simulation, and FPAA step response.

B. Neuron

Another example system we have constructed using Simulink, compiled to SPICE netlist, and targeted to a special purpose neuromorphic FPAA (RASP 2.8b) is a spiking Hodgkin–Huxley-type neuron block. The circuit for this spiking neuron follows the model described in [21]. The Simulink block diagram of the neuron block and necessary input and output blocks, such as an output buffer, is shown in Fig. 11(a). Spiking data from the Simulink model can be seen in Fig. 11(b) and from the FPAA in Fig. 11(c). The neuron block worked as expected in both Simulink level simulations and implemented in analog hardware on the FPAA.

We also compiled a system of two coupled neurons to demonstrate synchronized firing. The Simulink diagram for this system is shown in Fig. 12(a). Two neurons are shown with one feeding into the other trough a programmable synapse. This synapse is a floating-gate transistor that can be programmed strongly or weakly depending on the desired weight. The output of this coupled system is shown in Fig. 12(b). The two neurons are shown to be in sync, with the first one coupling onto the second.

C. VMM-WTA

The third example system is shown in Fig. 13(a). The system consists of a VMM feeding into a WTA. Fig. 13(b) shows a sample 2×2 circuit that will result when the VMM block is compiled to SPICE [22]. The circuit implementation contains design fields such as the number of transistors in the array, floating-gate program values, and amplifier biases. To aid the inexperienced analog designer, these fields are all presented as functional parameters to the user. Referring back to Fig. 5, the block-level design choices include matrix coefficients and time constant.

Fig. 13(c) shows what the sample 2×2 VMM will look like after compiled by RASPER. The diagram shows the same two OTAs that were in the previous schematic, along with the connection switches and floating-gate elements. It should be noted that one benefit of the RASP architecture is that any floating-gate elements can be synthesized right into the switch fabric [23]. This allows for much denser routing and the ability to create larger structures.

The results from the VMM-WTA system are shown in Fig. 14, where (a)–(c) are from the Simulink simulation and (d)–(f) are real data from the RASP 2.8a FPAA. The three plots for each correspond to the output at the scopes in Fig. 13(a).

The two rows of output plots in Fig. 14 match almost identically. The input vector (first column of plots) is designed in

such a way that each element input has a time being the largest. After the matrix multiplication with an identity matrix (the middle column), the magnitudes are preserved. The output after the WTA (last column) correctly decided which element of the vector was the largest.

Of note, a major dissimilarity between Simulink and Hardware is the concept of what constitutes a "signal." In Simulink, the signals are simply numerical vectors. This is sufficient to prove the functionality of the system. In real hardware, however, we are dealing with currents and voltages. In this example, we made the Simulink vector log values in order to map easily to the exponential nature of the input currents. This is an easy conversion to make, but it is something that the user should be aware of when dealing with analog signal processing.

VI. Discussion

We have developed Sim2spice, a configuration tool that allows for the conversion of signal-processing systems defined as interconnected blocks in MATLAB Simulink to a SPICE circuit netlist. This netlist can then be compiled to a targeting code for reconfigurable switches in an FPAA with an existing tool called GRASPER. A user can quickly and effectively compile relatively complex systems directly into analog hardware, even without a thorough understanding of analog circuit design. This tool flow opens a new world of design and rapid prototyping in analog to the wider signal-processing community.

References

[1] *Analog VLSI and Neural Systems*. : Addison Wesley, 1989, Carver Mead.

[2] C. Petre, C. Schlottmann, and P. Hasler, "Automated conversion of simulink designs to analog hardware on an FPAA," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2008, pp. 500–503.

[3] A. Basu, S. Brink, C. Schlottmann, S. Ramakrishnan, C. Petre, S. Koziol, F. Baskaya, C. Twigg, and P. Hasler, "A floating-gate-based fieldprogrammable analog array," *IEEE J. Solid-State Circuits*, vol. 45, no. 9, pp. 1781–1794, Sep. 2010.

[4] B. Sbarcea and D. Nicula, "Automatic conversion of MATLAB/Simulink models to HDL models," in *Proc. Int. Conf. Optimization Electr. Electron. Equipment*, 2004 [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.3077&rep=rep1&type=pdf

[5] C. Chang, J. Wawrzynek, and B. Brodersen, "From Bee to Bee2: Development of supercomputer-in-a-box," BerkeleyWireless Research Center, Univ. of California, Berkeley, 2004, Tech. Rep..

[6] G. D. Asensi, J. S. Gomez-Diaz, J. Martinez-Alajarin, and R. R. Merino, "Synthesis on programmable analog devices from vhdl-ams," in *Proc. IEEE Med. Electrotech. Conf.*, May 2006, pp. 27–30.

[7] P. McGuire, Pyparsing [Online]. Available: http://pyparsing.wikispaces.com/

[8] S.-C. Liu, J. Kramer, G. Indiveri, and T. Delbruck, *Analog VLSI: Circuits and Principles*. Cambridge, MA: MIT Press, 2002.

[9] P. E. Allen and D. R. Holberg, *CMOS Analog Circuit Design*. Oxford, U.K.: Oxford Univ. Press, 2002.

[10] F. Baskaya, S. Reddy, S. K. Lim, and D. V. Anderson, "Placement for large-scale floating-gate field-programable analog arrays," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 8, pp. 906–910, Aug. 2006.

[11] J. D. Gray, C. M. Twigg, D. N. Abramson, and P. Hasler, "Characteristics and programming of floating-gate pFET switches in an fpaa crossbar network," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2005, vol. 1, pp. 468–471.

[12] D. W. Graham, E. Farquhar, B. Degnan, C. Gordon, and P. Hasler, "Indirect programming of floating-gate transistors," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 5, pp. 951–963, May 2007.

[13] S. Koziol, C. Schlottmann, A. Basu, S. Brink, C. Petre, B. Degnan, S. Ramakrishnan, P. Hasler, and A. Balavoine, "Hardware and software infrastructure for a family of floating-gate based FPAAs," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. 2794–2797.

[14] C. M. Twigg and P. Hasler, "A large-scale reconfigurable analog signal processor (RASP) IC," in *Proc. IEEE Custom Integr. Circuits Conf.*, Sep. 2006, pp. 5–8.

[15] A. Basu and P. E. Hasler, "A fully integrated architecture for fast and accurate programming of floating gates over six decades of current," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, no. 99, 2010.

[16] C. M. Twigg and P. E. Hasler, "Incorporating large-scale FPAAs in analog design courses," in *Proc. IEEE Int. Conf. Microelectron. Syst. Education*, Jun. 2007, pp. 171–172.

[17] A. Basu, C. M. 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 *Proc. IEEE Custom Integr. Circuits Conf.*, Sep. 2008, pp. 213–216.

[18] A. Basu, S. Ramakrishnan, C. Petre, S. Koziol, S. Brink, and P. E. Hasler, "Neural dynamics in reconfigurable silicon," *IEEE Trans. Biomed. Circuits Syst.*, vol. 4, no. 5, pp. 311–319, Oct. 2010.

[19] S.-Y. Peng, G. Gurun, C. M. Twigg, M. S. Qureshi, A. Basu, S. Brink, P. E. Hasler, and F. L. Degertekin, "A large-scale reconfigurable smart sensory chip," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2009, pp. 2145–2148.

[20] C. Schlottmann, D. Abramson, and P. Hasler, "A mite-based translinear FPAA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, to be published.

[21] E. Farquhar and P. Hasler, "A bio-physically inspired silicon neuron," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 3, pp. 477–488, Mar. 2005.

[22] C. Schlottmann, C. Petre, and P. Hasler, "Vector matrix multiplier on field programmable analog array," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Mar. 2010, pp. 1522–1525.

[23] C. M. Twigg, J. D. Gray, and P. E. Hasler, "Programmable floating gate fpaa switches are not dead weight," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 169–172.

Craig R. Schlottmann (S'09) received the B.S. degree in electrical engineering from the University of Florida, Gainesville, in 2007, and the M.S. degree in electrical engineering from the Georgia Institute of Technology (Georgia Tech), Atlanta, in 2009, both in electrical engineering. He is currently working toward the Ph.D. degree in electrical engineering at Georgia Tech.

His research interests include low-power analog signal transmission, multiple-input translinear elements, floating-gate transistor circuits, and mixed-signal IC design.

Mr. Schlottmann was the recipient of the Best Live Demo Award at ISCAS 2010.

Csabe Petre received the B.S. degree from the University of California, Los Angeles, in 2007, and the M.S. degree from the Georgia Institute of Technology, Atlanta, in 2009, both in electrical engineering.

He is currently with Brain Corporation, San Diego, CA.

Paul E. Hasler (SM'04) received the B.S.E. degree in electrical engineering and M.S. degree from Arizona State University, Tempe, in 1991, and the Ph.D. degree in computation and neural systems from the California Institute of Technology, Pasadena, in 1997.

He is an Associate Professor with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta. His current research interests include low-power electronics, mixed-signal system ICs, floating-gate MOS transistors, adaptive information processing systems, "smart" interfaces for sensors, cooperative analog–digital signal processing, device physics related to submicrometer devices or floating-gate devices, and analog VLSI models of on-chip learning and sensory processing in neurobiology.

Dr. Hasler was the recipient of the National Science Foundation CAREER Award in 2001, and the Office of Naval Research YIP Award in 2002. He was also the recipient of the Paul Raphorst Best Paper Award from the IEEE Electron Devices Society in 1997, the CICC Best Student Paper Award in 2006, ISCAS Best Sensors Paper Award in 2005, and a Best Paper Award at SCI 2001.