An Open-Source ToolSet for FPAA Design

Jennifer Hasler and Aishwarya Natarajan
Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, U.S.A.
jennifer.hasler@ece.gatech.edu

***Abstract*—This open-source toolset enables targeting and design of the fine-grain SoC Large-scale Field Programmable Analog Array (FPAA) family of devices, similar to tools enabling FPGA devices. The SoC FPAA tool framework is presented, following a discussion of the resulting analog abstraction, FPAA infrastructure, and resulting educational and research impact.**

I. The Need for FPAA Targeting Tools

The programmability and configurability of digital computation as seen in FPGAs has enabled ubiquitous digital computation. Widely available digital design tools enables this programmability. FPGA tools come from FPGA manufacturers, open-source tools (e.g. [1]), and tools compiling from higher-level languages to commercial FPGAs.

This effort presents an integrated open-source mixed-signal toolset for compiling the SoC large-scale Field Programmable Analog Arrays (FPAA) (e.g. [2]), potentially enabling ubiquitous analog or mixed-signal low-power sensor to processing devices in a manner similar to FPGA devices. These tools abstract some low-level details enabling system design and application engineers to design on the FPAA using a set of user-friendly, high-level tools (e.g. graphical) for a wide range of energy efficient applications (Fig. 1). Analog design tools are limited, looking at particular space of simulation (e.g. [3]), macromodeling techniques (e.g. [4], [5]), over leveraged digital tools (e.g. [6]), or automation for a minority of analog designers rather than system designers (e.g. [7]).

The hope for programmable and configurable analog and mixed-signal devices has been at least as strong if not stronger than the original drive for digital reconfigurability. Analog computing techniques result in 1000× improvement in power or energy efficiency, and a 100× improvement in area efficiency, compared to digital computation as Mead originally predicted [8]. For example, an SoC FPAA implemented a command-word acoustic classifier (spectral classification) with hand-tuned weights, achieving command-word recognition in less than 23μW with standard digital interfaces [9]. The full classification results in less than 1μJ per classification (or inference), which has 1000× improvement over similar digital neuromorphic solutions requiring roughly 1mJ or higher for just an inference (e.g. [10]). The SoC FPAA interdigitates analog and digital computation in the same routing fabric.

Physical FPAA implementations drove the development for analog and mixed signal design tools (Fig. 1), particularly the SoC FPAA implementation, as well as FPAA ICs leading up to the SoC FPAA devices [11], [12], [13]. These tools give the user the ability to create, model, and simulate analog and digital designs. High-level design tools (Fig. 1) have been essential to the SoC FPAA development (Fig. 1). The SoC FPAA tools and PC board infrastructure are openly available as open-source tools[1]. A standard tool and infrastructure platform enables faster utilization and development of next generation FPAA applications. This tool framework is directly applicable to other FPAA devices (e.g. [14], [15], [16]), and we encourage an open community in these directions.

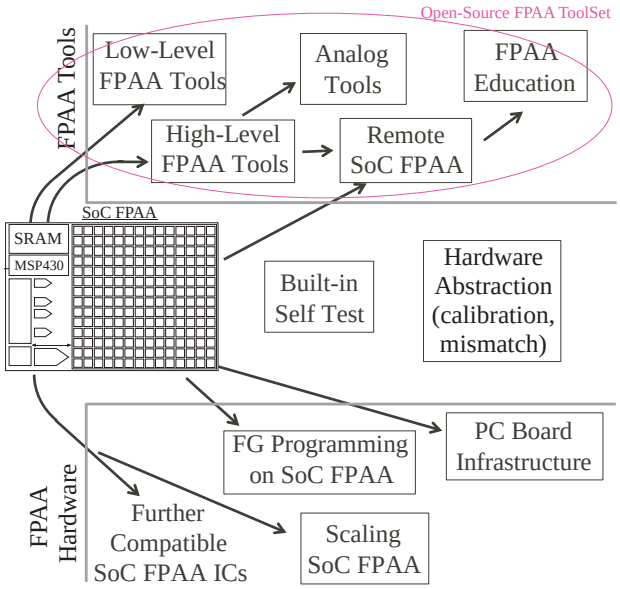

Fig. 1. The open-source FPAA tool infrastructure drives the opportunities and development of the SoC large-scale Field Programmable Analog Array (FPAA) devices. Innovations in FPAA hardware, innovations and developments in FPAA tool structure as well as innovations in the bridges between them are essential for the SoC FPAA infrastructure.

This discussion starts with the SoC FPAA tool infrastructure, including FPAA tool framework (Section II) and Analog Abstraction (Section III). SoC FPAA infrastructure (Section IV) includes FG programming, and PC board infrastructure, through system enabling technologies as calibration and built-in self test methodologies. These tools are currently used at GT for education and research, and the authors would welcome the opportunity to demonstrate these tools at the meeting.

II. Analog Tools and the SoC FPAA Toolset

Tools are essential for FPAA system design. While identifying every switch is easier than IC layout, design, verification,

[1]Tools can be downloaded at http://hasler.ece.gatech.edu/FPAAtool/index.html

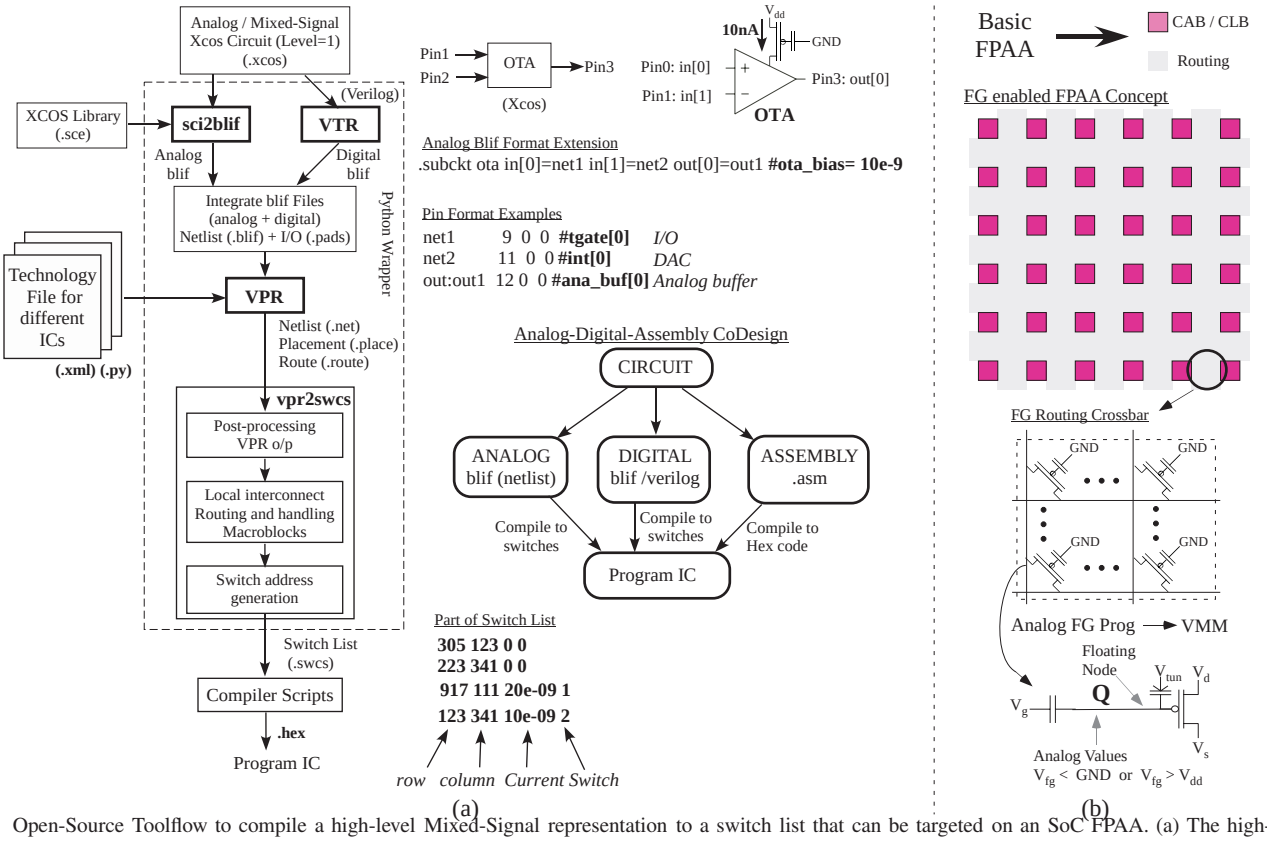

Fig. 2. Open-Source Toolflow to compile a high-level Mixed-Signal representation to a switch list that can be targeted on an SoC FPAA. (a) The high-level representation is defined in Scilab / Xcos, and compiled through x2c tool that uses a number of GT open-source tools as well as other open-source tools (e.g. VPR, VTR). The tools allow for high-level simulation as well as circuit level simulation in the Scilab / Xcos interface. (b) The FPAA computing architecture enables computing in the routing as well as in the CAB (Computational Analog Blocks) or CLB, requiring further tool efforts for targeting and design. A *Basic FPAA* also includes CAB (Computational Analog Blocks) or CLB and Routing. The FG enabled routing crossbar are excellent switches, as well as enabling computing in the routing as a result of analog programming. Unlike FPGAs, all switches in some FG enabled FPAA devices are potential places of computation. FG stores a charge, **Q**, at the floating node, allowing storage of analog voltages that can be inside or outside the power supplies (GND, V_{dd}).

fabrication, and testing for analog IC engineers, system designers will expect higher-level capabilities. The current SoC FPAA [9] utilizes 600,000 analog programmable Floating-Gate (FG) parameters in 350nm CMOS. Most of these designers do not want to know about transistors and analog transistor circuits, and yet the tools must enable efficient use by analog IC designers to enable blocks for application designers. Tools are essential for application-based system design using physical systems, given the modern comfort with structured and automated digital design from code to working application.

This open-source tool platform (Fig. 2) creates an integrated environment running in Scilab/Xcos [30] integrating developed tools with modified open-source digital place and route (*VPR* [1]) tools. *x2c* converts high-level block description by the user to *blif* format, inputs to the modified VPR tool, and utilizes $vpr2swcs$ to a switch list directed by an IC architecture file. An open-source Ubuntu 12.04 Virtual Machine (VM) abstracts the entire tool flow, from Scilab/Xcos, device library files, through *sci2bliff*, *vpr2swcs*, and modified VPR tools. The core tools are updated inside the VM[2].

High-level design tools (Fig. 2), implemented in Scilab / Xcos, enable automated compilation to a *switch list*, the description of the programmed FPAA hardware. The graphical high level tool uses a palette for available blocks that compile down to a combination of digital and analog hardware blocks, as well as software blocks on the resulting processor. The tools are designed to enable a non-circuits expert, like a system applications engineer, to investigate particular algorithms. They enable system level design (level=1) and circuit level design (level=2) (e.g. [13]), including both FPAA targeting and simulation [17]. Tools enable physical noise modeling (e.g. [13]) allowing for simulated prediction of the effect of noise on a compiled system as well as the resulting system SNR. The architecture files specify the analog–digital IC details. The result is a rich set of open analog and digital blocks that is available for wider utilization and contribution.

The SoC FPAA enables computation both in the Computational Blocks (CAB for Analog and CLB for Logic) as well as in the routing fabric, singificantly increasing the fine granularity and resulting FPAA functionality, while increasing the tool complexity. Analog FG devices provide both the FPAA memory elements as well as the FPAA routing elements. The fabric switches use a single FG pFET device that can be programmed in an analog manner, enabling computation in routing fabric (e.g. Vector-Matrix Multiplication) as well

[2]The updated tool core GIT repo: https://github.com/jhasler/rasp30. One can make their own linux-based toolflow with this link

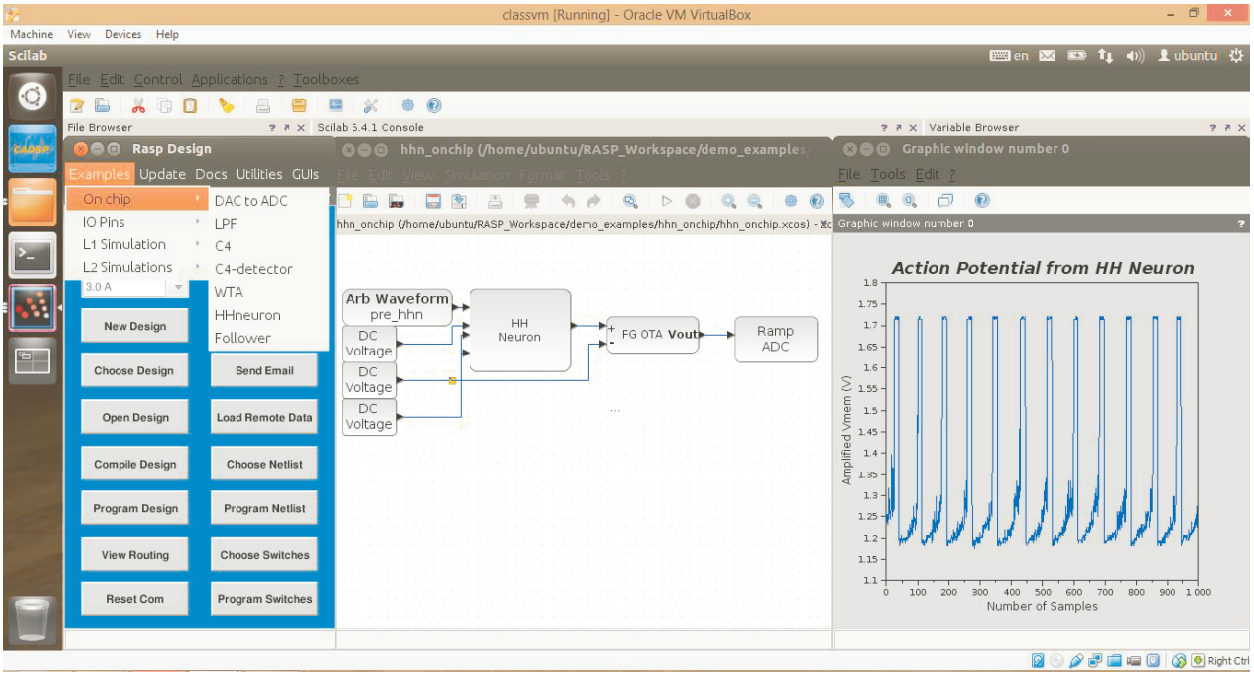

Fig. 3. Illustration of the open-source FPAA tools operating in an Ubuntu 12.04 VM.

as CAB elements [18], Computation in FPAA fabric represents a dramatic departure from classical FPGA architectures (Fig. 2b). *The FPAA computes in the co-located memory space of switches.* Multiple stages of the place and route infrastructure are modified to enable this functionality.

III. Tools Enabling Abstraction

Tools open the space for abstraction. The multiple levels of analog abstraction in a typical implementation (Fig. 4) can be abstracted from the designer who only needs to use higher level blocks (blocks in measurement setup of Fig. 4). Figure 4 shows a typical use of the C^4 block in an acoustic front-end for creating sub banded outputs. The core computational chain, C^4 Bandpass filter + Amp Detect + LPF, all compiles into a single CAB. FG elements (e.g. FG enabled OTA elements), as well as tunable capacitor banks, enable this abstraction and can be tuned around mismatches (e.g. [19]). The abstraction includes computation and testing instrumentation blocks into a single complete compiled system. This measurement illustrates the measure voltage block, effectively a slow speed (200SPS), high-resolution (14-bit) voltage measurement. The structure uses the FG programming circuitry, including the 14-bit measurement ramp ADC, still available in *run* mode. These blocks abstract further at the sub band processing stage as the front-end of an acoustic classifier (e.g. [9]). With the higher level of abstraction, handling the co-design issue between at least analog code, digital code, and μP code becomes immediately apparent. Tools should enable designers to effectively and efficiently design through the large number of open questions in this analog–digital codesign space.

IV. SoC FPAA Hardware Infrastructure

The FPAA structure uses an open-source board design[3] connected (through USB) to a laptop with the Virtual Machine and FPAA tools. The SoC FPAA programs the FG elements from the switch list using the μP. The programming appears as simple as downloading code libraries (e.g. python, Java) to stream the device data through the TCL framework used by the design tools. Programmable subthreshold and above threshold current sources are routinely programmed over six orders of magnitude (e.g. 30pA to 30μA) with better than 1 percent accuracy at all values including subthreshold current levels [20]. The interface board uses a single USB interface for power and resulting infrastructure for SoC FPAA devices [21], [22], enabling a range of user approaches, including FPAA devices that can be powered, programmed, and controlled through Android devices [23], or a remote FPAA infrastructure controlled through a simple unix platform using a POP email server controlled through open-source Python code [24].

V. Summary

The open-source FPAA design tools enable design from high-level synthesis to gate/transistor design as well as compilation to configurable hardware, starting a path well understood today by FPGA devices. The goal is to empower a large community utilizing these FPAA devices, contributing to a number of commercial and open-source communities, empowered through common tool frameworks. The authors want to encourage user communities towards developing a wide library of analog and digital components as well as solving application problems. These tools were developed along-side educational

[3]Board designs are available at http://hasler.ece.gatech.edu/PCboards/index.html

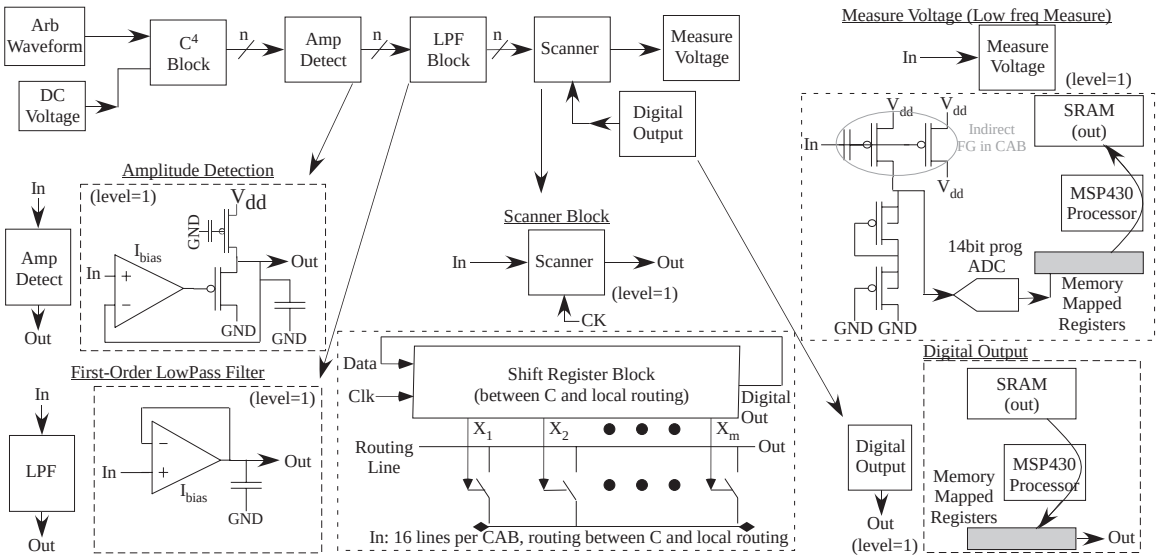

Fig. 4. An example of the analog level abstraction for an acoustic subband computation. In addition to the measurement blocks, each with their own level of abstraction, the computation requires abstractions for a scanner block, measurement infrastructure, compiled digital blocks, and assembly language code.

efforts to enable experimentally measurable mixed-signal design opportunities as well as reducing or eliminating the need for additional benchtop test equipment other than an FPAA board ([25], [26], [27], [22]. These techniques were used in a first undergraduate transistor circuits course [28], [29]. The open-source tools opens further commercial and educational applications upon commercial availability of FPAA devices.

REFERENCES

[1] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," vol. 7, 2014. pp 6:1–6:30.

[2] J. Hasler, "Large-Scale Field Programmable Analog Arrays," *IEEE Proceedings*, vol. 108. no. 8. August 2020. pp. 1283-1302.

[3] S. Ganesan and R. Vemuri, "Digital Partitioning for Field-Programmable Mixed-Signal Systems," *ARVLSI*, 2001, pp.172-185.

[4] G.R. Boyle, B.M. Cohn, D.O. Pederson, and J.E. Solomon, "Macromodeling of Integrated Circuit Operational Amplifier," *IEEE JSSC*, vol. 9, 1974. pp. 353-363.

[5] G. Casinovi, "A macromodeling algorithm for analog circuits," *IEEE TCAD*, vol. 10, no. 2, 1991, pp. 150-160.

[6] S. Liao and M. Horowitz, "A verilog piecewise-linear analog behavior model for mixed-signal validation," *IEEE Transactions CAS I*, vol. 61, no. 8, 2014, pp. 2229-2235.

[7] Barcelona Design, http://www.barcelonadesign.com. Company is not in business any longer (started in 1999); link provided as a reference.

[8] C. Mead, "Neuromorphic electronic systems," *Proc. IEEE*, no. 78, 1990. pp. 1629-1636.

[9] S. George, S. Kim, S. Shah, J. Hasler, M. Collins, F. Adil, R, Wunderlich, S. Nease, and S. Ramakrishnan, "A Programmable and Configurable Mixed-Mode FPAA SoC," *IEEE Transactions on VLSI*, vol. 24, no. 6, 2016, pp. 2253-2261.

[10] P. Blouw, X. Choo, E. Hunsberger, C. Eliasmith, "Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware," arXiv: 1812.01739v2, April 2, 2019.

[11] M. Collins, J . Hasler, and S. George, "An Open-Source Toolset Enabling Analog–Digital Software Codesign," *Journal of Low Power Electronics Applications*, vol. 6, no. 1, February 2016.

[12] C. R. Schlottmann, C. Petre, and P. E. Hasler, "Simulink Framework for Design to and Automated Conversion on Large-Scale FPAA Devices," *IEEE Transactions VLSI*, February 2011.

[13] C. Schlottmann and J. Hasler, "High-level modeling of analog computational elements for signal processing applications," *IEEE Transactions VLSI*, vol. 22, no. 9, 2014. pp. 1945–1953.

[14] G. Cowan, R. Melville, and Y. Tsividis, "A VLSI analog computer/digital computer accelerator," *IEEE JSSC*, vol. 41, no. 1, 2006, pp. 4253.

[15] B. Rumberg and D. W. Graham, "A Low-Power Field-Programmable Analog Array for Wireless Sensing," *ISQED*, 2015.

[16] F. Henrici, J. Becker, S. Trendelenburg, D. De Dorigo, M. Ortmanns, and Y. Manoli, "A field programmable analog array using floating gates for high resolution tuning," *ISCAS*, 2009, pp. 265-268.

[17] A. Natarajan and J. Hasler, "Modeling, simulation and implementation of circuit elements in an open-source tool set on the FPAA," *Analog Integrated Circuits and SP*, vol. 91, no. 1, April 2017. pp. 119-130

[18] C. Twigg, J. Gray, and P. Hasler, "Programmable Floating-gate FPAA switches are not dead weight," *IEEE ISCAS*, May 2007, pp. 169-72.

[19] S. Kim, S. Shah, and J. Hasler, "Calibration of Floating-Gate SoC FPAA System," *IEEE TVLSI*, July 2017.

[20] S. Kim, J. Hasler, and S. George, "Integrated Floating-Gate Programming Environment for System-Level ICs," *IEEE Transactions VLSI*, vol. 24, no. 6, 2016, pp. 2244-2252.

[21] S. Koziol, C. Schlottmann, A. Basu, S. Brink, C. Petre, S. Ramakrishnan, P. Hasler "Hardware and Software Infrastructure for a Family of Floating-Gate FPAAs," *IEEE ISCAS*, June 2010. Winner of the best demonstration paper award.

[22] J. Hasler, S. Kim, S. Shah, F. Adil, M. Collins, S. Koziol, and S. Nease, "Transforming Mixed-Signal Circuits Class through SoC FPAA IC, PCB, and Toolset," *IEEE European Workshop on Microelectronics Education*, Southampton, May 2016.

[23] B. Bolte, S. Shah, S. Kim, P. Hwang and J. Hasler, "FPAA Demonstration Controlled through Android-Based Device," sl IEEE ISCAS, 2016.

[24] J. Hasler, S. Kim, S. Shah, I. Lal, M. Kagle, and M. Collins, "Remote FPAA System Setup Enabling Wide Accessibility of Configurable devices," *JPLEA*, vol. 6, no. 14, 2016, pp. 1-17.

[25] C. Twigg and P. Hasler, "Incorporating Large-Scale FPAAs Into Analog Design and Test Courses," *IEEE Transactions Education*, Vol. 51, No. 3, 2008, pp. 319-324.

[26] P. Hasler, C. Schlottmann, S. Koziol, S. Ramakrishnan, S. Brink, and A. Basu, "FPAA chips and tools as the center of an Design-Based Analog Systems Education," *IEEE MSE*, San Deigo, June 2011, pp. 47-51.

[27] M. Collins, J. Hasler, and S. George, "Analog systems education: An integrated toolset and FPAA SoC boards," *IEEE MSE*, 2015, pp. 32-35.

[28] J. Hasler, A. Natarajan, S. Shah, and S. Kim, "SoC FPAA Immersed Junior Level Circuits Course," *MSE*, May 2017.

[29] J. Hasler, "Circuit Implementations Teaching a Junior Level Circuits Course Utilizing the SoC FPAA," *ISCAS 2018*, Florance, May 2018.

[30] Scilab Enterprises. *Scilab: Free and Open Source software for numerical computation*. Scilab Enterprises, Orsay, France, 2012.