A Low-Power Field-Programmable Analog Array for Wireless Sensing

Brandon Rumberg and David W. Graham
Lane Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, WV 26506
Email: brumberg@mix.wvu.edu, david.graham@mail.wvu.edu

***Abstract*—We present a field-programmable analog array (FPAA) for sensor interfacing and information extraction in wireless, resource-constrained applications. Ultra-low-power operation is achieved through low-overhead reprogramming and an efficient processing architecture. We demonstrate reconfigurability and performance through the synthesis of a temperature sensor, heart-rate alarm, and audio spectrum normalizer, which have measured full-system power consumptions of 12μW, 20μW, and 17.25μW, respectively.**

I. Introduction

The proliferation of sensors within mobile devices and wireless sensing systems complicates application development by increasing componentry and energy demands. In these wireless devices, energy is limited to what can be stored within the device or harvested from the environment. Systems based around the "sensor hub" concept, depicted in Fig. 1(a), simplify application design by using dedicated hardware to extract contextual information and handle low-level sensor tasks. Such systems have proven effective for inertial sensing applications with bandwidths below 100Hz. However, for signal bandwidths greater than 100Hz, such as audio, maintaining low power throughout the signal chain remains elusive. Lowering power consumption is becoming increasingly important, particularly for battery-powered devices.

To ease the development of resource-constrained sensing applications, we present the architecture in Fig. 1(b), wherein a sensing-oriented field-programmable analog array (FPAA) efficiently performs sensor interfacing and information extraction. FPAAs have received increasing attention in attempts to bring the advantages of FPGAs (e.g. rapid prototyping) to traditional analog applications, such as filtering and sensor interfacing [1], [2]. Further work has exploited the complex large-signal behavior of transistors to efficiently map signal processing and classification algorithms into analog circuits [3], [4].

This potential convergence of rapid sensor-interface design with low-power signal processing makes FPAAs enticing for resource-constrained sensing applications. However, the cost of dense analog parameter storage—either high quiescent power to refresh volatile storage or high infrastructure overhead to write nonvolatile storage—limits the use of large-scale FPAAs in low-power systems.

In this work, we present an FPAA for low-power sensing applications. A low-overhead, highly-integrated programming architecture has been developed. As a result, only 1–48mJ is required to reprogram, depending on the design which is loaded into the FPAA, thus enabling this FPAA to be used in low-resource systems. Additionally, we have designed the signal path architecture specifically for sensing applications to improve performance and resource utilization compared to a general-purpose architecture.

This FPAA is an apt front-end for "persistent sensing" applications, such as contextual awareness and surveillance. In these applications, the system continuously performs low-level data gathering and processing, and only occasionally responds to these low-level operations. By using the FPAA to implement the bulk of the low-level operations—such as sensor interfacing, signal analysis, and event detection—the system design can be simplified and the system power consumption can be reduced.

This material is based upon work supported by the National Science Foundation under Award No. 1148815.

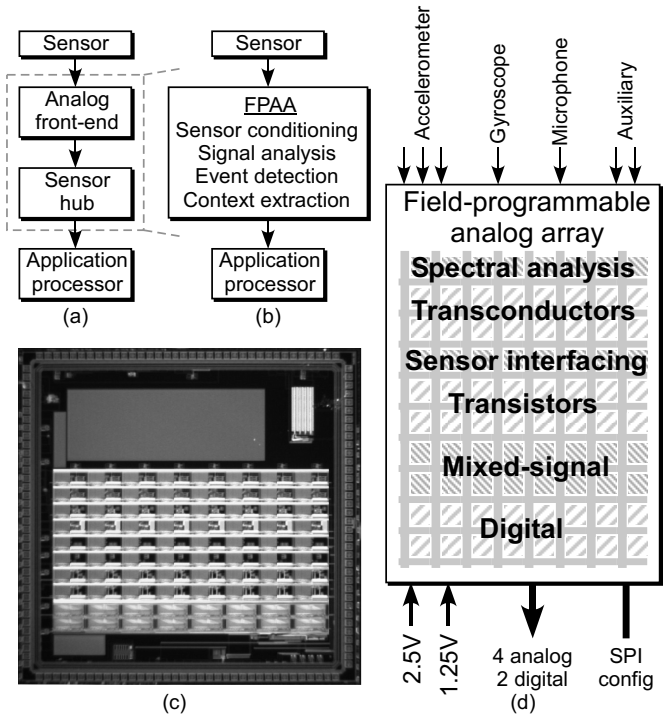

Fig. 1. (a) A conventional sensing system. (b) A sensing system in which an FPAA replaces the sensor interfacing and information extraction blocks. (c) Micrograph of our 25mm^2 FPAA die. (d) Block diagram of our FPAA development board. Our FPAA consists of an 8x10 array of computational blocks: 8 identical channels and 10 function-specific stages.

All plots in this paper depict measurements from our 25mm^2 FPAA integrated circuit—Fig. 1(c)—that was fabricated in a standard 0.35μm CMOS process.

II. Design of the FPAA

A. Signal path architecture

The circuit primitives, or "computational elements," within an FPAA are commonly grouped into a hierarchy of configurable blocks [5], [6]. Each block may contain various computational elements, and different types of blocks may be included. Prior FPAAs have used homogenous arrays of general-purpose blocks, with no more than three unique block types [6], [7]. In our FPAA, diverse computational elements (listed in Table I) are grouped into nine unique and function-specific block types for various stages of sensor interfacing and processing. Since our FPAA is designed for sensor preprocessing and event detection, we have adopted a parallelized architecture—shown in Fig. 1(d)—which consists of 80 computational blocks arranged in an 8-channel by 10-stage signal flow. Signal decomposition is performed early in the chain, after which, data are processed in parallel channels, which allows the remaining processing to be low bandwidth and low power.

The switch architecture of our FPAA is essentially an island-style architecture as is common in FPGAs [8]. A full-crossbar switch matrix connects the 16 terminals of the computational elements at the block level, and a switch box connects 6 routing tracks between neighboring blocks. Additionally, a long track in each stage/channel routes signals globally. All 20,380 switches are implemented using a transmission gate controlled by a local SRAM bit.

The parasitic capacitance caused by the switch matrix was measured to be 203fF for connections in a single block and 400fF for connections to neighboring blocks—comparable to the floating-gate-based FPAA in [6]. These parasitics are significant, and including too many unbuffered switchable nets in the signal path will reduce performance and increase the ac power consumption.

To make our FPAA less sensitive to such parasitics, we have included high-granularity computational elements in the FPAA to reduce the number of switches in the signal path. Our FPAA's computational elements are listed in Table I; these elements represent a variety of granularity and function to improve performance and application fit. The elements are applicable to most signal-processing applications and are grouped by function into stages to streamline audio and vibration applications. These groups are described below.

1) *Spectral analysis*: Contains programmable filters, envelope detectors, and OTAs with reconfigurable bias terminals to synthesize frequency decomposition algorithms.
2) *Transconductors*: Contains a variety of linear and nonlinear transconductance elements for synthesizing G_m–C networks and discriminant functions.
3) *Sensor interfacing*: Contains op-amps and resistors to build reconfigurable sensor interfaces.
4) *Transistors*: Used to synthesize computational elements that are too specialized to include as dedicated elements.
5) *Mixed-signal*: Contains comparators, S/Hs, programmable-width pulse generators, etc. Designed with current-starved and non-overlapping push/pull logic to nullify short-circuit current caused by interfacing with slow signals from the preceding stages.
6) *Digital*: Contains flip flops and lookup tables. Used to add digital control to analog circuits and to generate event-detection and data-ready interrupts for the application processor. Connected to the FPAA's SPI pins for run-time writing/reading of synthesized registers.

TABLE I
Computational Elements

8 BPFs	56 OTAs	8 inverters	16 envelope detectors
8 LPFs	8 multipliers	32 comparators	48 current sources/sinks
56 caps	8 op-amps	8 bump circuits	16 pulse generators
8 PNPs	16 resistors	8 time-to-voltage	16 asymmetric integrators
16 S/Hs	144 FETs	32 JK flip flops	16 6-input 2-output LUTs

B. Memory programming infrastructure

The programmable characteristics of the computational elements—e.g. time constants, G_m's, pulse widths—are controlled by 296 analog nonvolatile memory (NVM) elements. These memory elements consist of programmable current sources that are based upon floating-gate transistors. Floating-gate transistors have no resistive connection to their gate; instead, a "control gate" couples capacitively onto the transistor's "floating gate." As a result, the floating-gate charge, which can be modified using Fowler-Nordheim tunneling [9] and hot-electron injection [10], creates a programmable and nonvolatile threshold-voltage shift from the perspective of the control gate.

We have developed a highly integrated, energy-optimized system—Fig. 2(a)—for programming analog values onto floating gates. The NVM elements and write control circuit, which are shown in detail in Fig. 3, are based upon the continuous-time programming circuitry in [11]. The reprogramming process is shown in Fig. 2(b) and described below.

1) Clear switches & NVM: First, the FPAA is reset by clearing the SRAM that controls the switches and by block erasing the analog floating-gate memory. Block erasure is performed by applying a 10.5V pulse on V_{tun} to tunnel electrons off of all floating gates. This voltage pulse is generated by an on-chip high-voltage charge pump.

2) Write NVM: Next, analog values are written to the NVM. Writing is performed by first raising the memory supply voltage, $V_{dd,NVM}$, to 6V to facilitate injection, and then by sequentially connecting individual elements to the write-control circuit to store V_{targ} into the NVM. V_{targ} is set to the desired value for each element by an on-chip DAC. During the write process, V_{tun} is set to 4.5V using the on-chip regulated charge pump to avoid reverse tunneling through M_{tun}.

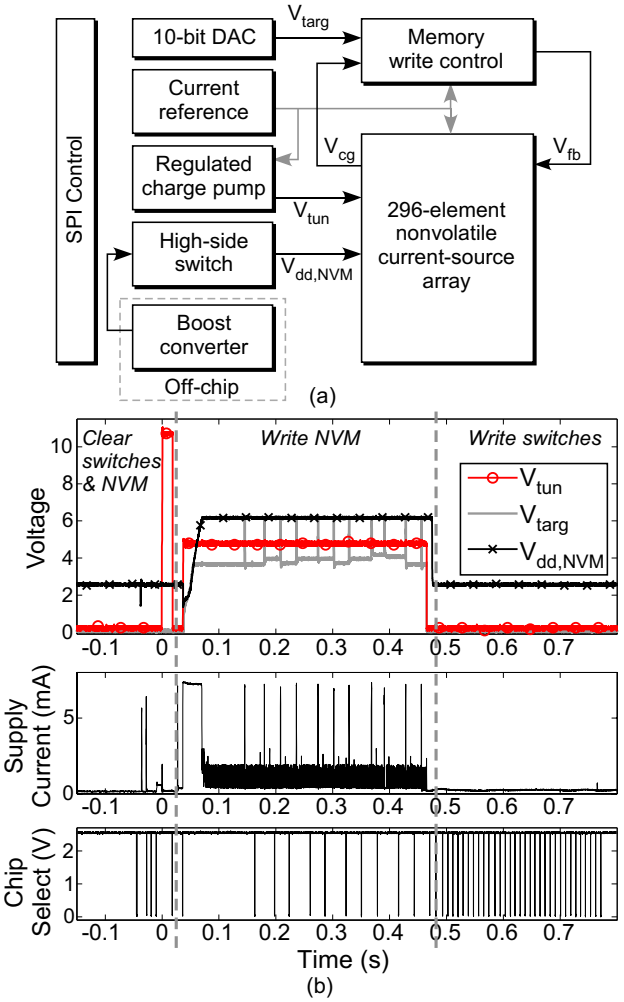

Fig. 2. (a) Block diagram of the FPAA's analog nonvolatile memory (NVM) programming system. (b) Measurement of the process of loading a design into the FPAA. The supply current is measured flowing into the FPAA board. The SPI chip select pin indicates data transfers from the application processor.

3) Write switches: Finally, switches are set so as to wire the FPAA for the desired functionality. We have written a PC program to translate a user-generated netlist into a list of switches to be programmed.

C. Summary of FPAA programming

Analog memory writes are controlled by an on-chip feedback loop, and the only interaction from the application processor is to specify the memory address and DAC codeword. As a result, we incur significantly less overhead than systems which require a processor in the feedback loop [6]. The entire process in Fig. 2(b) consumed 2.35mJ, 33% of which is consumed while starting the external boost converter and can be largely eliminated by generating $V_{dd,NVM}$ with an on-chip charge pump (similar to our V_{tun} charge pump).

In Fig. 2(b), 10 NVM and 33 switches were written. The energy breakdown is approximately 0.12mJ per NVM, 6.4μJ per switch, and a constant 0.94mJ each reconfiguration cycle. These data are used to estimate the energy to load larger designs into the FPAA. For example, the design in Fig. 6 uses 478 switches, 29 LUT bits, and 52 analog NVM, at a total reconfiguration energy of 10.42mJ.

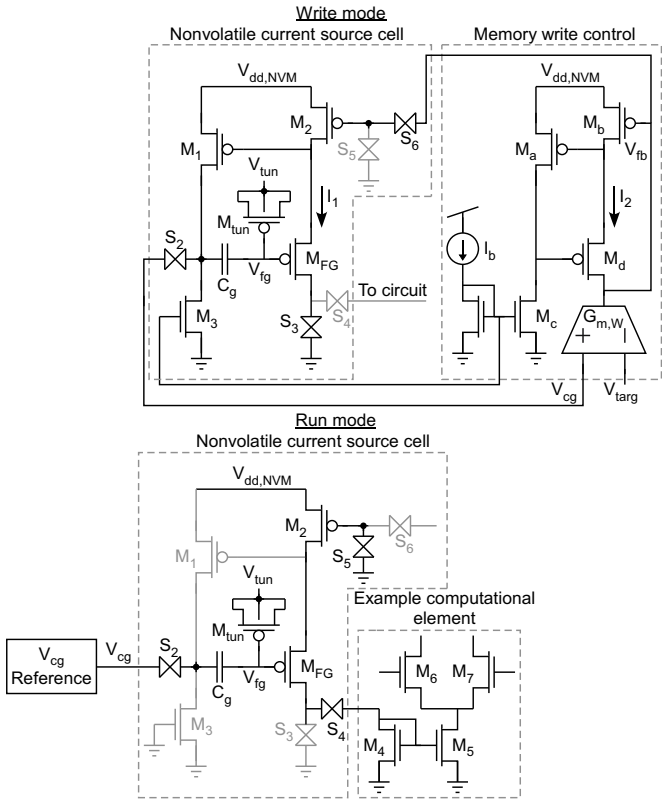

Fig. 3. Detail of the NVM cell. In write mode, a local feedback loop ($M_{1,3}$) around the floating-gate transistor M_{FG} linearizes the exponential injection characteristics. NVM cells are individually connected to a write control circuit which modifies the current injected onto V_{fg} (by modifying I_1) until the charge on V_{fg} causes V_{cg} to match V_{targ}. The regulated-cascode current mirror, M_{a-d}, replicates the memory cell structure to improve matching between I_1 and I_2. In run mode, each NVM cell is connected as a current source to its respective computational element.

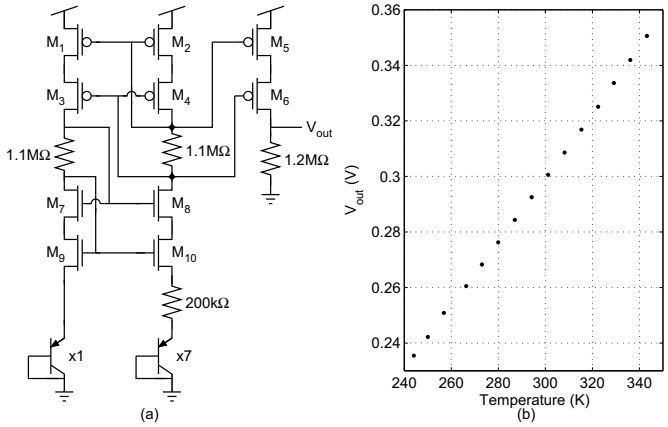

Fig. 4. (a) Temperature sensor which was synthesized using devices in the FPAA. (b) Measured output across a 100K temperature range.

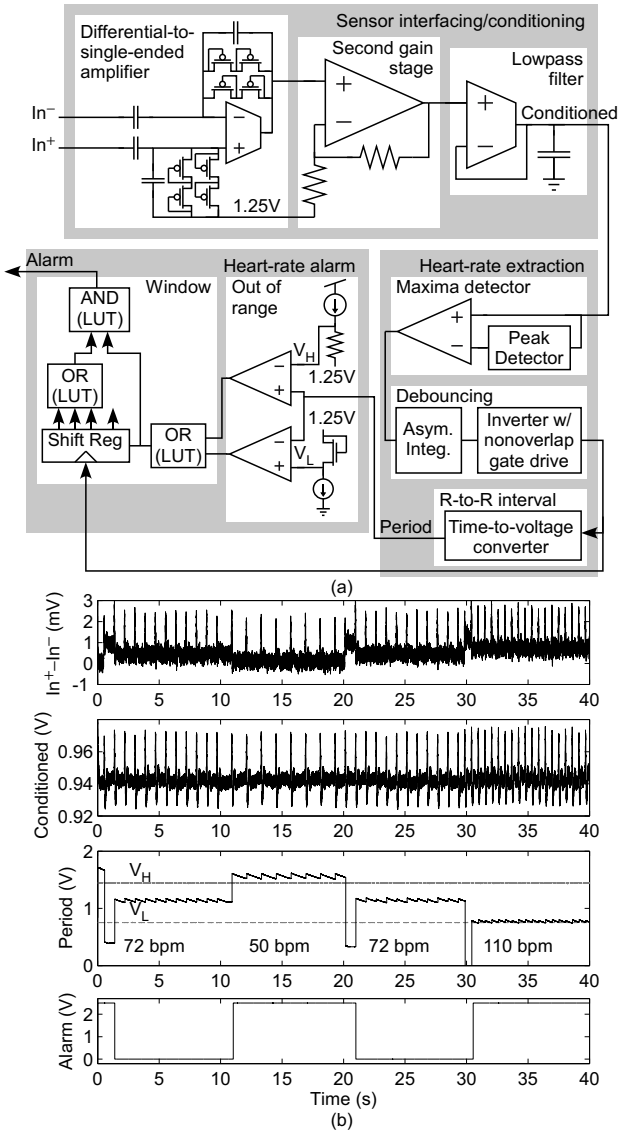

Fig. 5. (a) Heart-rate monitoring system which was synthesized in the FPAA. After passing through the conditioning block, a time-to-voltage converter is clocked by the peaks of the R wave to extract the period. The period is compared with user-defined high/low thresholds. If two recent periods are outside of the safe range, then an alarm is generated. (b) Measured response of the heart-rate monitoring system. The input is a 2mV differential cardiac signal with varying heart rate and 200mV 60Hz common-mode noise. The outputs of the conditioning, extraction, and alarm subsystems are plotted. The bottom plot shows successful detection of out-of-range heart rates.

III. Demonstrations

The following demonstrations validate the FPAA in a variety of applications that are representative of wireless sensing scenarios. All power consumption values (Table II) were obtained by measuring the supply current of the entire FPAA board (at 3V); these values include the power of output buffers, regulators, and references, and are thus representative of the power cost to add the FPAA to an embedded system.

Temperature sensor: The FPAA includes a large number of device-level elements to synthesize circuits that are too specialized to include in the computation blocks. The temperature sensor in Fig. 4(a) was synthesized as a demonstration of a circuit synthesized purely from device-level elements. The ratios of the BJTs and resistors were chosen to yield a 1mV/K output. The circuit's measured temperature response is shown in Fig. 4(b).

TABLE II
Demonstration results

Circuit	Run power	Nets	NVMs	Config energy
Temp. sensor	12μW	23	3 (readout buffer)	1.97mJ
Heart-rate	20μW	55	18	5.52mJ
Audio	17.25μW	89	52	10.42mJ

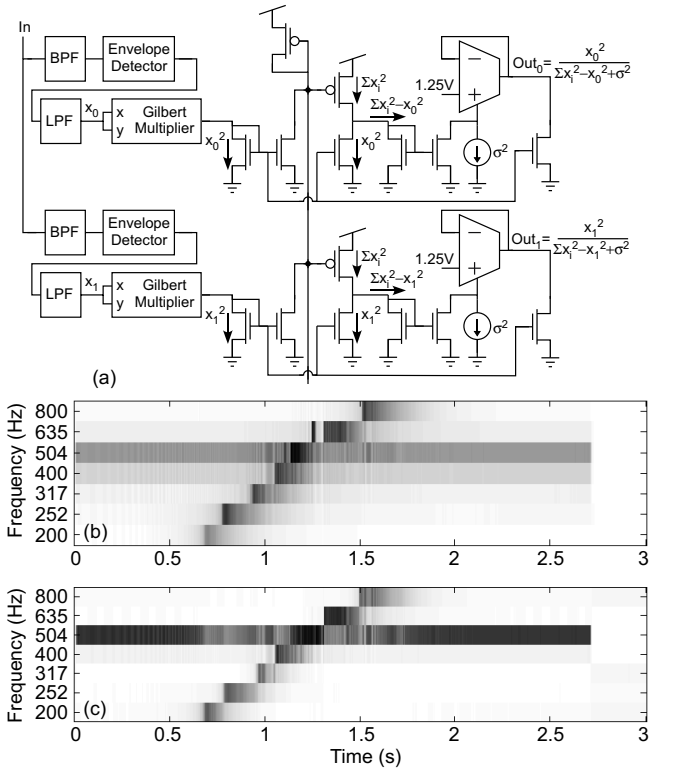

Fig. 6. (a) Audio spectrum normalization system which was synthesized in the FPAA. Only 2 of 7 channels are shown for clarity. (b & c) Measured response of the system to a 500Hz tone and chirp combination. (b) Without normalization: x_i. (c) With normalization: Out_i. Note that normalization reduces leakage of the 500Hz tone into neighboring bands, and observe how the 500Hz band is inhibited during the chirp.

Heart-rate monitor: The FPAA was designed to meet the needs of the first three stages in sensor systems: sensor interfacing, signal analysis, and event detection. These capabilities are demonstrated by synthesizing the heart-rate monitoring system shown in Fig. 5(a). The difference amplifier is based on [12] and the system's back-end was inspired by [13]. Each symbol directly maps to a single computational element (see Table I) with the exception of the shift register, which is a collection of flip flops. As a result of the FPAA's mixed granularity, this relatively large system was mapped onto a small number of elements.

Audio spectrum normalization: The FPAA's architecture is amenable to audio and vibration signal processing using an analog filter bank. Information in filter channels is highly correlated and requires subsequent processing to prepare for classification. In Fig. 6, we present a circuit implementation of the decorrelation algorithm in [14], which was synthesized in the FPAA. The algorithm's nonlinear inhibition of parallel channels maps efficiently into analog circuitry. The circuit sharpens the filter bank frequency responses and normalizes the channels to create scale-invariant features for classification. The power consumption is 17.25μW. For comparison, [3] describes an FPAA implementation of a single channel of a comparable filter bank algorithm which consumed 34.5μW.

IV. Conclusion

A commonly cited application of FPAAs is sensor interfacing. As applications in wirelessly connected sensors and the Internet of Things continue to emerge, the quantity of sensing devices will increase while the energy consumed per device will need to decrease. With this in mind, we have developed a large-scale, low-overhead FPAA for systems which have severe power constraints.

To facilitate the synthesis of large signal processing systems, we have designed the FPAA with a mixture of computational elements and a parallel signal flow topology. The system in Fig. 6 has 52 analog parameters and 89 nets (a synthesized channel readout scanner is not shown). This is one of the largest-scale published systems to be implemented in reconfigurable analog fabric.

Acknowledgment

We would like to thank Brandon Kelly and Spencer Clites for help with the layout of the FPAA.

References

[1] D. Anderson, C. Marcjan, D. Bersch, H. Anderson, P. Hu, O. Palusinski, D. Gettman, I. Macbeth, and A. Bratt, "A field programmable analog array and its application," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, Santa Clara, CA, May 1997, pp. 555–558.

[2] B. Pankiewicz, M. Wojcikowski, S. Szczepanski, and Y. Sun, "A field programmable analog array for CMOS continuous-time OTA-C filter applications," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 2, pp. 125–136, Feb. 2002.

[3] A. Basu, S. Brink, C. Schlottmann, S. Ramakrishnan, C. Petre, S. Koziol, F. Baskaya, C. Twigg, and P. Hasler, "A floating-gate-based field-programmable analog array," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 9, pp. 904–922, Sept. 2010.

[4] D. Fernández, L. Martínez-Alvarado, and J. Madrenas, "A translinear, log-domain FPAA on standard CMOS technology," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 2, pp. 490–503, Feb. 2012.

[5] R. Wunderlich, F. Adil, P. and Hasler, "Floating gate-based field programmable mixed-signal array," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 8, pp. 1496–1505, Aug. 2013.

[6] C. Schlottmann, S. Shapero, S. Nease, and P. Hasler, "A digitally enhanced dynamically reconfigurable analog platform for low-power signal processing," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 9, pp. 2174–2184, Sept. 2012.

[7] S. Peng, G. Gurun, C. Twigg, M. Qureshi, A. Basu, S. Brink, P. Hasler, and F. Degertekin, "A large-scale reconfigurable smart sensory chip," in *IEEE International Symposium on Circuits and Systems*, Taipei, Taiwan, May 2009, pp. 2145–2148.

[8] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and challenges," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, Feb. 2008.

[9] M. Lenzlinger and E. Snow, "Fowler-Nordheim tunneling into thermally grown SiO2," *Journal of Applied Physics*, vol. 40, no. 1, pp. 278–283, Sept. 1969.

[10] P. Hasler, A. Andreou, C. Diorio, B. Minch, and C. Mead, "Impact ionization and hot-electron injection derived consistently from Boltzmann transport," *VLSI Design*, vol. 8, no. 1-4, pp. 454–461, May 1998.

[11] B. Rumberg and D. Graham, "A floating-gate memory cell for continuous-time programming," in *Proceedings of the IEEE Midwest Symposium on Circuits and Systems*, Boise, ID, Aug. 2012, pp. 214–217.

[12] R. Harrison and C. Charles, "A low-power low-noise CMOS amplifier for neural recording applications," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 6, pp. 958–965, June 2003.

[13] H. Abdalla and T. Horiuchi, "An analog VLSI low-power envelope periodicity detector," *IEEE Transactions on Circuits and Systems I, Regular Papers*, vol. 52, no. 9, pp. 1709–1720, Sept. 2005.

[14] O. Schwartz and E. Simoncelli, "Natural signal statistics and sensory gain control," *Nature Neuroscience*, vol. 4, pp. 819–825, Aug. 2001.