RAMP: Accelerating Wireless Sensor Hardware Design with a Reconfigurable Analog/Mixed-Signal Platform

Brandon Rumberg David W. Graham Spencer Clites Brandon M. Kelly
Mir Mohammad Navidi Alex Dilello Vinod Kulathumani
Lane Department of Computer Science and Electrical Engineering
West Virginia University
david.graham@mail.wvu.edu

ABSTRACT

The requirements of many wireless sensing applications approach, or even exceed, the limited hardware capabilities of energy-constrained sensing platforms. To achieve such demanding requirements, some sensing platforms have included low-power application-specific hardware—at the expense of generality—to pre-process the sensor data for reduction to only the relevant information. While this additional hardware can save power by reducing the activity of the microcontroller and radio, a unique hardware solution is required for each application, which presents an unrealistic burden in terms of design time, cost, and ease of integration. To diminish these burdens, we present a reconfigurable analog/mixed-signal sensing platform in this work. At the hardware-level, this platform consists of a reconfigurable integrated circuit containing many commonly used signal-processing blocks and circuit components that can be connected in any configuration. At the software level, this platform provides a framework for abstracting this underlying hardware. We demonstrate how to quickly develop new applications on this platform, ranging from standard sensor interfacing techniques to more complicated intelligent pre-processing and wake-up detection. We also demonstrate how to integrate this platform with commonly used wireless sensor nodes and embedded-system platforms.

Categories and Subject Descriptors

B.7 [**Integrated Circuits**]: Miscellaneous; C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems, Signal processing systems

Keywords

Analog Signal Processing; Energy-Efficient; Sensor Networks

IPSN '15, April 14 - 16, 2015, Seattle, WA, USA
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3475-4/15/04 ...$15.00
http://dx.doi.org/10.1145/2737095.2737107

1. INTRODUCTION

With the proliferation of battery-powered sensing devices via wireless sensor networks and the Internet of Things, the ability to gather sensor information in an easy manner while maintaining low-power operation becomes increasingly critical. Custom application-specific hardware can more efficiently collect and process sensor data. In turn, such hardware can help to reduce the overall power consumption of the sensing system by removing compute-intensive tasks from the purview of the more power-hungry general-purpose digital processor. For example, by implementing the early stages of the signal-processing chain in dedicated hardware, sensor data can be efficiently compressed and/or classified into the most relevant information for the system's task. Accordingly, general-purpose digital processors may spend more time in the low-power sleep modes, and radios can be turned on only when necessary. However, this increase in efficiency comes at the cost of flexibility—if the run-time conditions of the system change, it will no longer be able to provide the same level of efficient and/or accurate performance.

In this work, we explore a programmable hardware solution that can provide both efficient processing through customized designs as well as the flexibility to change those designs to meet new system-level requirements. Such flexibility helps developers to save power by implementing the early, and often compute-intensive, stages of their signal-processing chain in "custom" hardware, while also having the option to reprogram the hardware after deployment. Furthermore, when the sensor interfacing hardware is programmable, developers can create efficient custom sensor interfaces for their applications, and adapt them as needed.

1.1 Contributions and Main Results

We present a new reconfigurable analog/mixed-signal platform (RAMP)—illustrated in Fig. 1—that provides the much needed flexibility for ultra-low-power pre-processing hardware. We show how this reconfigurable platform, combined with a flexible abstraction framework, enables quick application design. We discuss methods for integrating this platform with existing systems and show the ability to perform in-the-field reconfiguration. We also provide several example systems that have been built using the RAMP—ranging from simple signal-conditioning circuits to more complicated wake-up detection systems.

The inclusion of analog circuitry in the RAMP is critical to providing significant reductions in power consumption for a number of reasons. First of all, it has been shown that ultra-low-power analog circuitry can actually be lever-

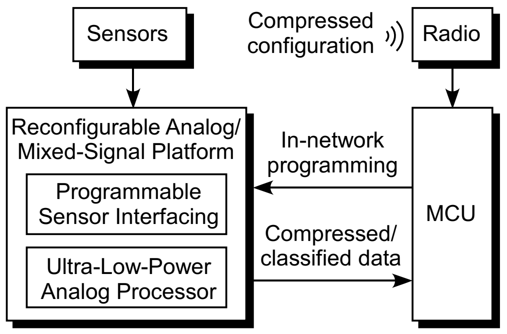

Sensors
Compressed configuration
Radio
Reconfigurable Analog/ Mixed-Signal Platform
Programmable Sensor Interfacing
Ultra-Low-Power Analog Processor
In-network programming
Compressed/ classified data
MCU

Figure 1: Architecture of a wireless sensor with reconfigurable sensor-interface hardware that helps developers implement the early, and often compute-intensive, stages of their signal-processing chain in "custom" low-power hardware, while also having the option to reprogram the hardware after deployment.

aged to provide orders-of-magnitude power savings over all-digital approaches [12, 16]. In this scenario, the analog pre-processing system can remain in an always-on state to monitor the incoming sensor data; meanwhile, the subsequent digital system can remain in a low-power sleep mode until the analog pre-processor wakes it up via an interrupt signal. Additionally, by performing initial classification and compression of the sensor data in the analog domain, only the data that need to be analyzed further will be digitized and processed in a digital microcontroller or digital signal processor. Of note, the process of digitizing the analog signal via an analog-to-digital converter (ADC) is often very costly in terms of power consumption. By pre-processing in the analog domain, less data need to be converted, and oftentimes, they can be converted at a lower resolution (i.e., fewer bits), thereby saving additional power.

The inclusion of digital circuitry alongside the RAMP's analog circuitry provides the ability to control how various analog circuits interact. It also provides a method for crossing the analog/digital divide without necessitating ADCs by interfacing directly with the subsequent system's general I/O pins. Additionally, since the digital portion of the RAMP contains a small FPGA-like framework, it has the ability to synthesize custom circuits that are purely digital.

We have designed and fabricated the RAMP as a 5×5mm integrated circuit (IC) in a standard 0.35μm CMOS process. The baseline quiescent power consumption of this IC and its development board is 4.05μW, and the total power consumption scales up according to what design has been synthesized on the RAMP, with typical designs ranging up to 20μW. In consequence, the addition of this reconfigurable platform adds minimal power consumption to the sensor node, and it has the ability to reduce that sensor node's power consumption considerably by limiting the amount of subsequent digital computation and radio communication.

Furthermore, the RAMP approaches a "one-size-fits-all" solution for all of the analog needs of the sensor node—including conventional signal conditioning (e.g., filtering and amplification of sensor data). Instead of needing a custom standalone analog signal-conditioning circuit for each new sensor, the sensor conditioning along with any pre-processing and/or wake-up circuit can be synthesized on this reconfigurable IC, thereby potentially saving space on a printed circuit board (PCB).

1.2 Related Work

One advantageous use of custom hardware in wireless sensor network (WSN) applications has been to implement wake-up detection for the sensor nodes. Specifically, a custom low-power wake-up detection circuit that is placed immediately after the sensor can significantly reduce the power consumed by a WSN node by permitting the microcontroller (MCU) and the radio to remain in a low-power sleep state. For example, a digital periodicity detector, which was implemented as a custom digital application-specific integrated circuit (ASIC) and consumed only 835nW, was used to generate a wake-up signal for an acoustic surveillance system in [3]. Another example, which used the raw analog signal to generate a wake-up signal, used a comparator circuit within a crack-monitoring application to determine if the amplitude of the sensor's output surpassed a given threshold (with a total power consumption of 16.5μW) [4]. In [7], a further extension of this concept used peak detector circuits to extract the envelope of the signal for comparison with the wake-up threshold (consuming approximately 5μA of current).

Each of these examples were able to increase the percentage of time that the MCU and radio remained in sleep mode, thereby saving power. However, they largely focused on signal timing and/or amplitude levels, which could easily produce false positives in the presence of interference, noise, or other types of unwanted signals. In [12], ultra-low-power analog circuits were used to provide more than amplitude-dependent event detection. Instead, the spectral content of the signal was used to classify the signals based upon spectral templates, meaning that even in the presence of large interfering noise (which would trigger amplitude-dependent wake-up circuits), the subsequent system would remain in sleep mode until a signal that matched a known spectral template was found. The result was a system that produced far fewer false positives than an amplitude-based system and extended the battery lifetimes significantly over an all-digital implementation (by approximately 7.5 years) [12].

In summary, hardware pre-processing systems–especially those using analog signal processing—can provide considerable power savings to a battery-powered wireless sensor by using smart pre-processing and classification. However, custom circuit design is a very time-consuming process, so the intended application must be well described in advance since the circuits offer very little flexibility.

Low-power microcontrollers do offer flexibility, and, similar to the hardware-based wake-up solutions described above, they have been used in smartphones to continuously pre-classify the sensor data and wake up the application processor. For example, [6] used the MSP430 microcontroller as a pre-processor to trigger a speaker identification algorithm when the presence of speech was detected. And in [10], the same microcontroller was used to process multiple motion sensors. In both cases, the continuous background processing was on the order of milliwatts, compared to hundreds of milliwatts for the phone—yielding significant power savings when the phone is otherwise unoccupied. However, since these are the same low-power MCUs that are already used in wireless sensor platforms, using them to pre-process sensor data would not save any power in a sensor network.

To effectively utilize wake-up in wireless sensor platforms, a flexible form of custom low-power signal processing hardware is desired. Given the advantages of ultra-low-power analog circuitry described above, we are seeking to simplify the process of analog circuit design by leveraging the growing field of reconfigurable analog systems [8, 15, 17]. Much like the all-digital field-programmable gate arrays (FPGAs) that have enabled rapid prototyping of digital systems, field-programmable analog arrays (FPAAs) seek to bring that same flexibility and reconfigurability to analog designs. The result is a single IC that can implement diverse analog signal-processing systems, as specified by the designer.

In this work, we present a reconfigurable analog/mixed-signal platform (RAMP) that extends the concept of an FPAA into both the analog and digital domains. While some recent work has investigated the use of mixed-signal design within FPAAs [5, 17], they have been largely restricted to applications in data converters and digitally-assisted analog circuits. Instead, our RAMP has been designed specifically with low-power wireless sensing applications in mind, and as a result, it is able to provide a large assortment of analog and mixed-signal routines for signal conditioning, signal classification, event detection, and wake-up generation. The novel attributes of our RAMP over previously reported reconfigurable analog systems are (1) it was designed from the ground up to implement low-power systems, (2) it is a fully self-contained field-programmable mixed-signal system, (3) it leverages non-linear building blocks for selective decision-making (as opposed to primarily linear blocks for signal conditioning), and (4) it has a signal-flow architecture that lends itself to making decisions. Other large-scale, non-commercial FPAAs (e.g., [15, 17]) offer less variation in the types of available circuit building blocks and have not been targeted for low-power and/or battery-powered applications. Commercially available reconfigurable analog systems, such as the Cypress PSoC systems and the Anadigm FPAAs, offer far less analog reconfigurability and focus on opamps, switched-capacitor circuits, data converters, and custom peripheral interfaces for connecting to a microcontroller—in short, they do not provide the same flexibility in the analog domain and cannot achieve as low power as the RAMP.

1.3 Outline of the Paper

The rest of this paper is organized as follows. In Section 2, we discuss the capabilities of our reconfigurable platform and describe the hardware that enables individual components to be connected together in user-defined configurations. Then, in Section 3, we describe the development environment that we have created to simplify the process of working with this platform and creating custom applications. In Section 4, we show how to include this platform within wireless and embedded sensing systems by describing 1) a custom printed circuit board for integration with common wireless sensor nodes, 2) custom code for interfacing with those sensor node platforms, and 3) a compression strategy for transmitting configuration files for in-the-field reconfiguration. We then provide several examples to demonstrate the functionality of our RAMP platform in Section 5, and finally, we draw conclusions in Section 6.

2. RECONFIGURABLE ANALOG/MIXED-SIGNAL PLATFORM

To facilitate the development of applications that employ custom low-power circuitry, we present the reconfigurable analog/mixed-signal platform (RAMP) integrated circuit (IC). A die photograph is shown in Fig. 2. The RAMP IC contains a mixed-signal FPAA containing many different analog and digital circuits, along with various interface and control circuits that enable it to be controlled by a wireless sensor node or other embedded platform. This RAMP and its constituent parts were designed from the ground up to emphasize low power consumption and ease of reconfiguration/programming.

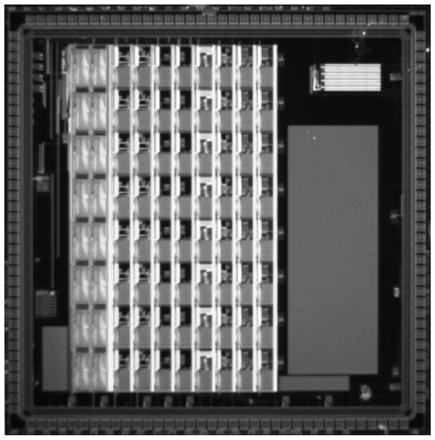


Figure 2: Die photograph of our RAMP IC.

The RAMP's mixed-signal FPAA contains an array of computational analog blocks (CABs) and configurable logic blocks (CLBs), as shown in Fig. 3. Each of these CABs and CLBs are connected to a matrix of switches that allow individual circuits to be connected in any configuration, as specified by the user. A connection box provides a crossbar switch matrix to permit connections among elements within a single CAB or CLB (i.e., intra CAB/CLB). A switch box provides an assortment of 4-way switches that permit connections from one CAB/CLB to another CAB/-CLB (i.e., inter CAB/CLB). Each switch is implemented as an SRAM-controlled transmission gate. A particular configuration (i.e., arrangement of switches as ON/OFF) is loaded into the FPAA using an on-chip serial-peripheral interface (SPI). In total, 20,380 switches are included in this FPAA.

Many analog signal-processing algorithms leverage parallel signal flows to achieve efficient computation [2, 12, 14]. As such, the FPAA consists of an array of the aforementioned CABs and CLBs arranged in a stage/channel configuration. A total of 80 computational blocks are arranged in a 10-stage and 8-channel signal flow, as shown in Fig. 3. The CABs/CLBs in each stage are designed for specific functions (described below), and all 8 channels are identical. This architecture allows an efficient mapping of parallelized algorithms in the FPAA; however, it is not necessary to leverage this parallelism.

To enable the design of a large variety of operations, we have included multiple types of circuit building blocks, ranging in "granularity" from basic circuit elements to complete circuits. At the low-complexity end, circuit elements such as resistors, capacitors, and transistors are included so that virtually any circuit may be synthesized. At the higher-complexity end, circuits that have been designed for a specific task (e.g., filters, envelope detectors, amplifiers) are included to both simplify the routing between circuit elements and to also reduce unwanted parasitic resistances and capacitances in sensitive circuits. This varying granularity of

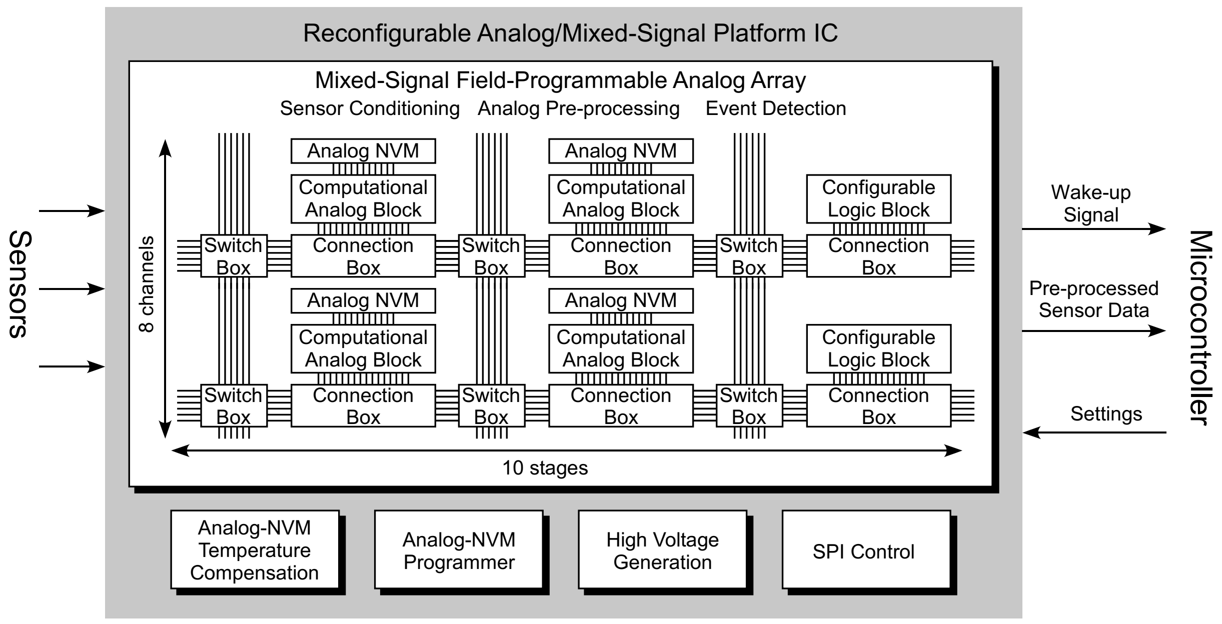


Figure 3: Architecture of our RAMP integrated circuit.

Table 1: Computational Elements in the RAMP

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

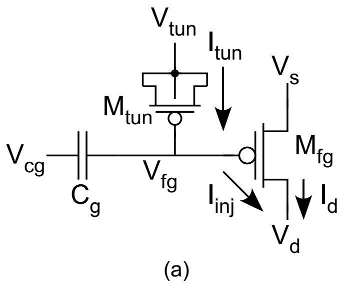
building blocks provides a balance between the degree of flexibility and also the complexity of the systems that can be implemented. These circuit building blocks were chosen to be sufficient to build a wide variety of applications and can be used in a hierarchical configuration to simplify the design of more complex elements, as described in Section 3.2. These different circuit building blocks are listed in Table 1 and have been lumped together by category in five different CAB types, as described below.

1. *Spectral analysis*: Contains programmable filters, envelope detectors, and other circuits for frequency decomposition algorithms.
2. *Transconductors*: Contains a variety of linear and nonlinear transconductance elements for synthesizing amplifiers, discriminant functions, and filters.
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 comparator circuits, sample-and-hold circuits, programmable-width pulse generators, and other circuits that operate at the boundary between analog and digital.

The CLBs consist of flip flops and look-up tables. In essence, the CLBs form a small FPGA to be used within the larger mixed-signal FPAA. These digital elements can be used to provide control signals for the analog circuits in the CABs or in any other scenario that requires sequential and combinational logic, such as for generating interrupt signals based upon the results of the analog pre-processing circuits. These digital elements are connected directly to the SPI block for configuration. A summary of the included digital elements is included in Table 1.

Many of the analog circuits require a precise bias voltage or current to set their parameters, such as a filter's bandwidth or an amplifier's transconductance. With such a large number of circuit elements, careful consideration in how to provide these biases must be made in order to not overwhelm the power consumption of the overall system. For example, using a digital-to-analog converter to set each bias value is unfeasible due to the large overhead in terms of both power consumption and chip area.

Instead, we have leveraged the use of "floating-gate transistors" to provide accurate bias values for the circuits with programmable parameters (such as gain, bandwidth, etc.). Floating-gate (FG) transistors are the core element in Flash memory and are able to store charge on their floating node, as shown in Fig. 4. As such, they are able to serve as non-volatile memory (NVM). Whereas Flash memory quantizes the stored charge into digital bits, we use the FGs to store a precise amount of charge. This precise amount of charge can then be used to generate an exact current flowing through a transistor, which results in a highly tunable non-volatile bias value for our circuits, without expending extra energy



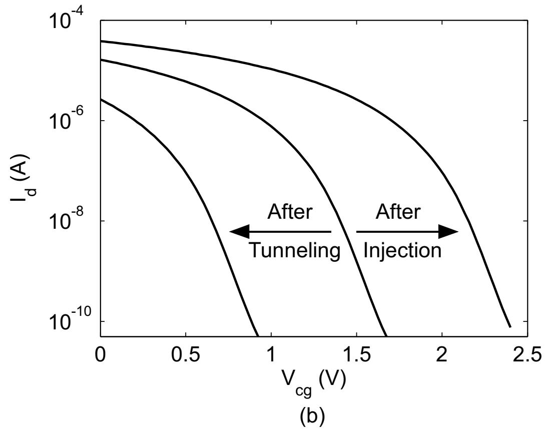


Figure 4: (a) A floating-gate (FG) transistor consists of a standard MOSFET with only capacitive connections to the gate terminal, resulting in a charge that will not leak off. The stored charge can be modified through the quantum-mechanical processes of Fowler-Nordheim tunneling to add charge (via $I_{\mathbf{tun}}$) and hot-electron injection to remove charge (via $I_{\mathbf{inj}}$). (b) The stored charge directly impacts the current flowing through the transistor, which is modeled by a change in the effective threshold voltage of the transistor from the perspective of the capacitively coupled control gate, $V_{\mathbf{cg}}$. The result is an analog nonvolatile memory element capable of providing a precise and programmable bias voltage or current.

once the bias has been established. In essence, we use the FG transistors as nonvolatile *analog* memory.

The analog-NVM programmer and high-voltage generation blocks of the RAMP in Fig. 3 are used for establishing the correct amount of charge on the FG in a fashion similar to [11]. The temperature compensation block then helps the NVM to maintain constant circuit-parameter performance in the face of temperature variations during "run mode" (i.e., when the RAMP is performing its user-defined application). As described in Section 3, the development environment is capable of directly translating a circuit parameter into a specific amount of FG charge so that the use of FG transistors is transparent to the application designer. In total, 296 bias parameters are directly controlled by this analog NVM.

Programming the RAMP consists of setting the appropriate connection switches to be ON/OFF and establishing each of the necessary bias parameters through the analog NVM. In terms of energy consumption for the RAMP, each write to an NVM requires approximately 0.12mJ and each write to a connection switch requires 6.4μJ. A constant 0.94mJ overhead is consumed for each configuration cycle. The total energy consumption to reconfigure is

$$E_{\text{config}} = N_{\text{switch}}(6.4\mu J) + N_{\text{NVM}}(0.12mJ) + 0.94mJ \quad (1)$$

where N_{switch} is the number of switches to be configured and N_{NVM} is the number of NVM elements to be programmed.

3. RAMP DEVELOPMENT ENVIRONMENT

When a user creates a design for the RAMP, that design consists of connections between components, as well as parameters that control the operation of those components. To provide users with powerful abstractions as well as access to full capabilities via low-level control, we allow the notions of "components" and "parameters" to cross multiple levels of abstraction. For example, a component may correspond directly to an individual hardware-primitive in the RAMP, or a component may be a group of primitives that is dynamically determined at compile time. Likewise, a parameter may correspond directly to a programmable bias in the RAMP, or a parameter may determine the relation between programmable biases within—or even the topology of—a group of components.

Given that designs consist primarily of connections between components, a user's design will thus map directly to a signal-flow block diagram. Signal-flow diagrams are often used to visualize signal-processing algorithms, so this form for expressing designs helps users to be productive at developing sensor-processing systems. To specify connections, we use a textual "netlist" representation, wherein each component is listed along with the connections between that component and the "nets" (or nodes) of the system. This textual representation facilitates complex designs by allowing easy creation and inclusion of libraries, commentable netlists, and the ability to write functions which generate portions of netlists at compile time. However, a graphical interface for drawing block diagrams can also be added on top of the tools that we describe in this Section.

In this Section, we describe our development flow from top to bottom, and then we describe the framework that we have developed for incorporating new component abstractions into our flow. We have written our development tools in GNU Octave-/Matlab-compatible code. The Octave or Matlab terminal serves as the user interface for issuing compile and program commands. The development board for our platform works with Arduino and TinyOS platforms. When an Arduino is used, the compiled bitstream is sent directly to the Arduino to program the RAMP. When a TinyOS-based mote is used, the compiled bitstream is printed into a file in a `uint8_t` array, which is then copied into the TinyOS application.

3.1 Compilation Flow

In this Subsection, we describe the compilation flow using a simple example netlist. To simplify this discussion, the example only contains primitive components. In reality, a user works with higher-level components to be more productive. Furthermore, this entire compilation flow is hidden from the user and only one "build" function needs to be run. Each step of the compilation flow, as well as the transformation on the design at each stage, is illustrated in Fig. 5. Below is the example netlist wherein an input signal, `Input`, is split into its 1kHz and 2kHz bands by two bandpass filters.

These bands, which are named Chan1 and Chan2, are then correlated to generate the final output, Out.

```
BPFx Inp=Input Ref=Mid Out=Chan1 fc=1e3 Q=3
BPFx Inp=Input Ref=Mid Out=Chan2 fc=2e3 Q=3
Corr In1=Chan1 In2=Chan2 Out=Out BiaI=50e-9
```

Each line begins with a component type—such as BPFx for bandpass filter or Corr for correlator—and has an arbitrary number of arguments that are parsed as

```
ArgumentName=ArgumentValue
```

Some of the arguments are connections to the component terminals, while other arguments are programmable parameters, such as fc for center frequency and Q for quality factor.

Although programmable analog parameters in the RAMP are tuned using programmable FG current sources, the compilation process offers several options for specifying parameters of primitive components. Users can specify the programmable current or they can specify a simple functional parameter of the circuit, such as the corner frequency, the time constant, the transconductance, or the center frequency and quality factor. To minimize the number of rules that are built into the compiler, we leave the implementation of more complex functional parameters to the abstraction framework that we describe in the next subsection.

When the above netlist is compiled, it is processed in steps that are similar to an FPGA compilation flow. The final outcome of compilation consists of the raw volatile and non-volatile data that are loaded into the RAMP's configuration memory—specifically, these data include the on/off state of each switch, the contents of the digital lookup tables, and the current that is programmed into each analog memory element. This data is analogous to machine code.

3.1.1 Decompose design into primitives

To reach the "machine code" level, the compiler first decomposes component abstractions into primitive component types. This process is described in the next subsection. The example netlist that we are using to describe the development flow does not contain any component abstractions, and could be the result of decomposing a component abstraction.

3.1.2 Place primitives in array

Once the design is decomposed into primitive component types, the next step is to "place" each instance of a primitive type into the RAMP by choosing among the available hardware primitives of that type. Our placement routine is based upon the FPGA-placement algorithm described in [1]. This algorithm uses simulated annealing to minimize the total wiring length of the design. For the sake of fidelity, the "sensitivity" of a net can be specified to ensure that the wiring length of that net is minimized, potentially at the expense of increased wiring length for other nets. The output of the placement routine for the above netlist is shown below.

```
BPFx_S0C6 Input Mid Chan1
BIAS_S0C6 BPFx fc=1e3 Q=3

BPFx_S0C5 Input Mid Chan2
BIAS_S0C5 BPFx fc=2e3 Q=3

Corr_S1C6 Chan1 Chan2 Out
FG_S1C6 Corr_Bia Itar=50e-9
```

The output is again a netlist, but the locations of the primitives within the RAMP are identified by appending the component name with an underscore followed by the stage number and channel number. Furthermore, the parameter values are moved to separate lines now that they have been identified with specific analog memory elements. The placement routine has minimized the total wiring length by placing the filters in adjacent channels (i.e., 5 and 6) and by placing the correlator in the same channel (i.e., 6) as one of the filters. Note that the filters are only located in stage 0 and the correlators are only located in stage 1.

3.1.3 Route connections between primitives

After the primitives have been placed, the next step is to "route" connections between the primitives by determining which switches should be turned on to achieve the desired connectivity. We have developed our own rule-based routing algorithm, which begins by routing the longest net from the CAB that is occupied by the largest number of connections. The algorithm then proceeds to shorter nets and less occupied CABs. The output of the routing routine is shown below. The % symbols denote comments.

```
% Net Input
CB_ST0_CH6 SB5 BPFx_Inp
CB_ST0_CH5 SB5 BPFx_Inp

SB_ST0_CH5 UR5
SB_ST0_CH6 UL5

% Net Mid
CB_ST0_CH6 Mid BPFx_Ref
CB_ST0_CH5 Mid BPFx_Ref

% Net Chan1
CB_ST0_CH6 SB3 BPFx_Out
CB_ST1_CH6 SB3 Corr_Pos

SB_ST0_CH6 UD3

% Net Chan2
CB_ST0_CH5 SB4 BPFx_Out
CB_ST1_CH6 SB4 Corr_Neg

SB_ST0_CH5 UR4
SB_ST0_CH6 UL4
SB_ST0_CH6 UD4

% Net Out
CB_ST1_CH6 Loc Corr_Out

FG_ST0_CH6 BPFx_Low tau=0.00065665
FG_ST0_CH6 BPFx_Hig tau=3.8575e-05
FG_ST0_CH5 BPFx_Low tau=0.00032832
FG_ST0_CH5 BPFx_Hig tau=1.9288e-05
FG_ST1_CH6 Corr_Bia Itar=50e-9
```

This output is the lowest human-readable level in the compilation flow, which makes it analogous to assembly code. Each line maps directly either to a single switch, a programmable analog parameter, or a lookup table (not shown in this example). The bottom of this output includes the programmable analog parameters. Note that at this stage, the interdependent center frequency and quality factor parameters of the bandpass filter have been decomposed into high and low time constants. The top of this output provides a list of switches that should be set. The switches are grouped by net. For example, the Input net connects via

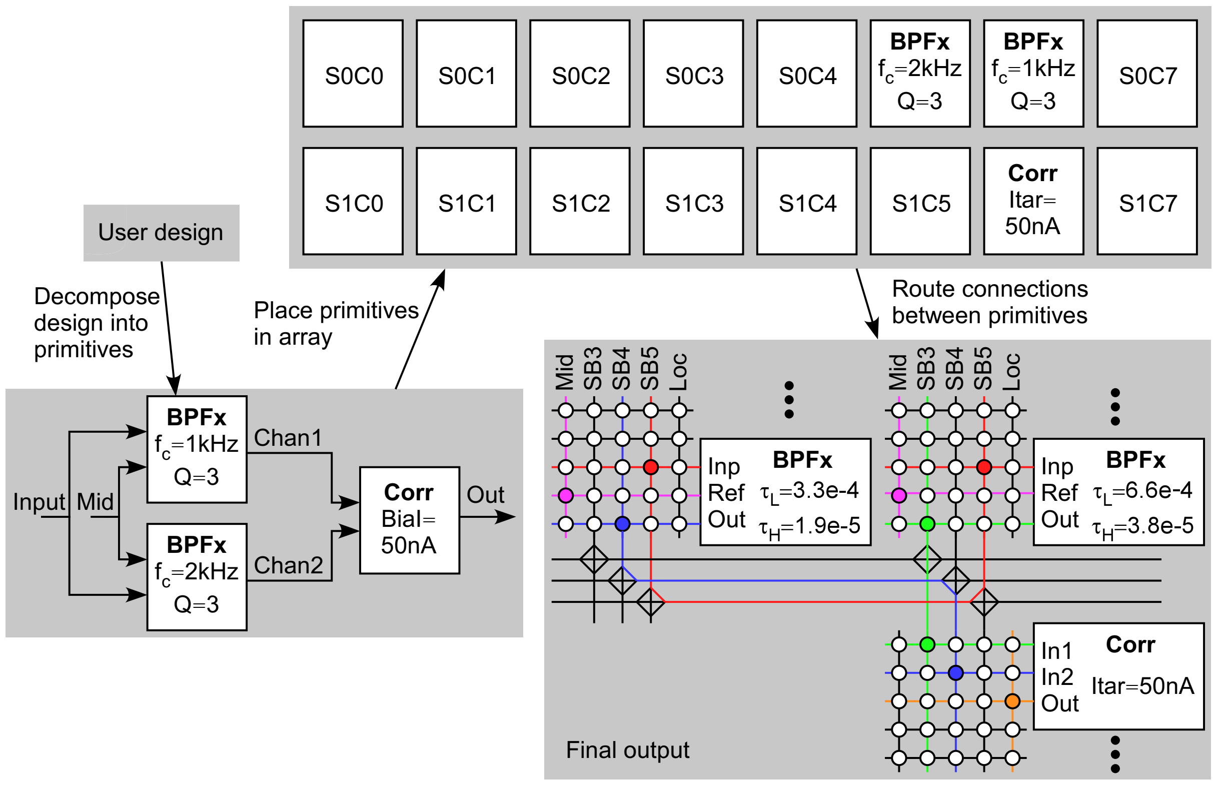


Figure 5: Illustration of the compilation flow for the RAMP. The user design consists of connections between components, and parameters for those components. These components are decomposed into primitive component types. An optimization routine "places" the design by associating the component instances with specific component resources in the RAMP. A heuristic algorithm then "routes" the design by determining which switches should be set to obtain the desired connectivity between components.

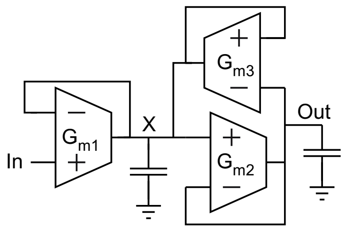
four switches. The syntax for the connection box switches is `CB` followed by the stage number, the channel number, and then the names of the routing track and device terminals that are connected by that switch. The syntax for the switch box switches is `SB` followed by the stage number, the channel number, and then the connection direction on the routing track. For example, `UR5` connects the "up" segment to the "right" segment on routing track number five.

To complete the compilation flow, the above code is translated into matrices of the raw volatile and nonvolatile data that is then loaded into the RAMP's memory. The total data is approximately 3KB, but we have developed a routine, described in Section 4.3, that reduces this configuration to a few hundred bytes.

3.2 Abstraction Framework

To aid hierarchical and reusable design, we have included two mechanisms for defining higher-level components. The simpler mechanism parses a user design for text-substitution macros. In our framework, these macros are generally used to create new components from functional groupings of lower-level components in a hierarchical design. For example, our RAMP provides bandpass filter primitives, but not lowpass filter primitives, so lowpass filters must be constructed from other primitives. An example macro for a second-order lowpass filter is shown in Fig. 6.

Our macro syntax is similar to a subcircuit in the SPICE



```
begin LPF_Order2 In Out Gm1 Gm2 Gm3
   OTAx Pos=<In> Neg=$X Out=$X Gm--=Gm1
   OTAx Pos=$X Neg=<Out> Out=<Out> Gm--=Gm2
   OTAx Pos=$X Neg=<Out> Out=$X Gm--=Gm3
   Capx Top=$X Bot=Gnd
   Capx Top=<Out> Bot=Gnd
end
```

Figure 6: Example of a macro for a second-order lowpass filter component.

circuit simulation language. Only one line is needed to instance this example macro component in a netlist:

LPF_Order2 Input Output 1e−3 1e−3 1e−3

Macro component representations are used when the topology of the circuit is fixed and when the component parameters are determined independently.

To enable dynamic generation of circuit topologies and interdependent biasing, we have also developed a more powerful abstraction mechanism. In this mechanism, we define the keyword `code` to designate that the line should be evaluated from Matlab. In this way, Matlab functions can take arguments from the netlist and then generate the desired lines of netlist code at compile time. This mechanism allows us to provide libraries with more function-oriented abstractions and also allows power users to create their own high-level abstractions. Three examples of the way the `code` framework has been used to dynamically generate circuit topologies and interdependent biasing are:

"Compare to": A simple function performed in many algorithms is comparing a signal to a static reference level. On a circuit level, this requires a comparator as well as a reference-voltage circuit. In our RAMP, reference voltages are generated using the programmable current sources. The most efficient topology for a reference voltage circuit depends upon the desired voltage. This example represents an often-used function which requires a change in topology dependent upon the function parameters. By abstracting this design to Matlab code, a netlist representing a comparator attached to an appropriate reference voltage circuit can be created without the system architect needing an awareness of the circuit-level implementation.

High-order filtering: Filtering operations are another common task within embedded systems. First-order linear filters are easy to implement in a netlisting language, but as the order of the filter increases, so does the overall size of the filter and the netlist that is required to generate it. In theory, this part of the process could be accomplished by chaining together macros of a particular filter topology. But if this route is taken, the user will still need to define the individual biases of each stage to create the filter of their choice—e.g., Butterworth or Chebyshev. Filter creation, therefore, is an example of a task where the scaling of the physical topology is simple, but the biasing can become complicated. Within the `code` framework, flexible Matlab code can be written to generate the bias values. As a result, the user only has to understand and specify the desired filter characteristics, but does not need to understand how to physically implement the filter within the RAMP.

Bit-scalable digital blocks: Finally, scalable digital blocks, such as multiplexers and shift registers, are commonly used. It is tedious to size these blocks specifically for each design, so the development environment should scale them at compile-time according to the number of bits required by the user. This way, the user can simply define their inputs and outputs, and the block will be created with the necessary number of bits. The `code` framework simplifies the creation of such scalable blocks.

With this abstraction framework, we are able to provide high-level, functional components in which another application developer does not need to know the underlying hardware details, but only the functional descriptions of how the blocks work. Since this abstraction layer is user-expandable, it provides a mechanism to promote sharing and reuse of higher-level circuits and signal-processing systems.

4. INTEGRATING THE RAMP INTO EMBEDDED SENSING SYSTEMS

In this Section, we describe three aspects of integrating our RAMP into embedded sensing systems. First, we describe our RAMP development board, which includes a variety of sensors, and which can connect to either a TelosB mote or an Arduino. Second, we describe the microcontroller code, whether TinyOS or Arduino, that is used to control the RAMP. And third, we describe a compression algorithm to enable low-power in-network reconfiguration.

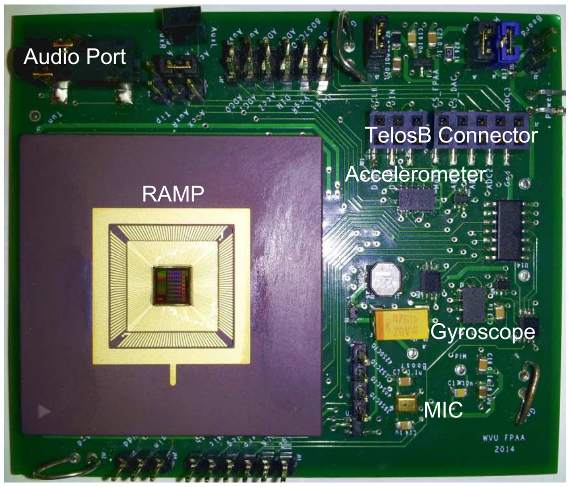


Figure 7: RAMP development board (3.1" x 2.6").

4.1 RAMP Development Board

We have designed a RAMP development board, shown in Fig. 7, for prototyping RAMP-enabled embedded sensing systems. This board includes our RAMP, a variety of sensors, a shift register for enabling/disabling sensors, a 6V boost converter for programming our on-chip nonvolatile analog memory, a current reference for temperature compensation, two 2.5V regulators for analog and digital supplies, a 1.25V reference, and comprehensive power probing.

This circuit board includes a header for connecting to a TelosB mote. This header exposes four analog output pins from the RAMP, two digital output pins from the RAMP (one of which can interrupt the mote), and four SPI pins that allow the mote to control the RAMP board. The mote's control over the board consists of reprogramming the RAMP and enabling the sensors. Alternatively, Arduino-compatible devices can also be connected via a ribbon cable to soften the learning curve for building RAMP-based applications.

The quiescent current draw of the development board when the RAMP is "off" is just 1.35μA: 276nA for the analog supply, 85nA for the digital supply, and the remainder powers the 2.5V regulators and the 1.25V reference. When the RAMP board is connected to a mote, the board and mote are both powered by a battery pack underneath the board. When the RAMP board is connected to an Arduino, the board is powered by the Arduino's supply voltage.

For sensors, we have included the Knowles SPW0430 low-power microphone (240μW), the STMicro LIS352 3-axis accelerometer (900μW), and the STMicro LY3200 1-axis gyroscope (12.6mW)—all of which can be completely turned off using on-board switches. Additional sensor inputs can be provided using a 3.5mm stereo audio jack and a 2-pin female header. This combination of sensors makes the development board useful for prototyping wearable electronics as well as audio/vibration applications.

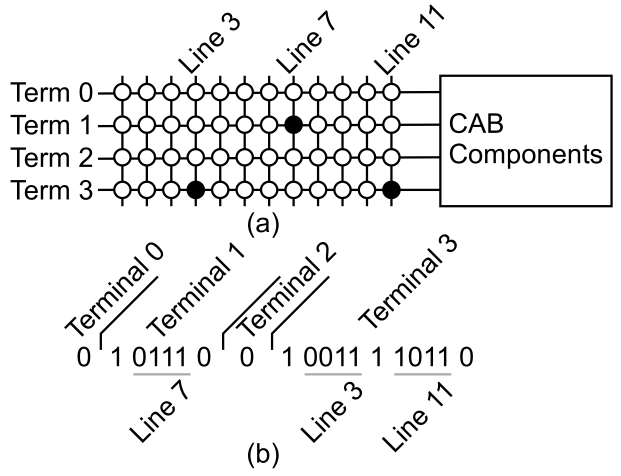


Figure 8: (a) Example of a configuration of switches. (b) Implementation of how this configuration would be transmitted using our compression scheme.

4.2 Microcontroller Code for RAMP Control

It is the responsibility of the microcontroller to initialize the RAMP upon power up, and then reprogram the RAMP when desired. Since the RAMP configuration is compressed, as described in the next subsection, the microcontroller also decompresses the configuration as it programs the RAMP. To facilitate these operations, we have developed code for TinyOS and Arduino environments. Our software interface exposes three functions to the user: 1) reset the RAMP, 2) turn selected sensors on or off, and 3) decompress the configuration/program the RAMP.

4.3 Low-Energy In-Network Reconfiguration with *AZiP*

One advantage that this reconfigurable platform offers for sensor networks is the ability to redefine the analog componentry after the network has been deployed. However, the raw configuration file is approximately 3KB, which presents a challenge to efficiently distributing the configuration to every node in a network. Fortunately, the relatively sparse distribution of "on" switches in a RAMP facilitates high levels of compression. Compression is very important for energy management in sensor networks. A TelosB mote can perform 4,000 cycles of computation for the energy that is needed to transmit/receive one byte of data, and in a multi-hop network, the energy saved by compression can be very significant [13]. Because of the high overhead of transmitting sensor data, work on in-network compression has focused on locally compressing sensor data prior to transmission. In contrast, RAMP configurations are decompressed locally, so the compression algorithm can be complex as long as the decompression algorithm is simple. We have developed an algorithm for compressing RAMP configuration files called *AZiP*—for fast decompression of analog designs—which has a low-complexity decompression routine that unpacks the RAMP settings "just in time" to program so that the expanded configuration is not held in memory.

AZiP is based upon the characteristics of the configuration data. Two types of configuration data are loaded into the RAMP: 1) volatile data, which consists of approximately 1700 12-bit wide registers that control switch on/off states, and 2) nonvolatile analog data, which consists of 296 voltages with 11-bit precision. Most of the volatile registers control the switches that connect routing lines to device terminals, as shown in Fig. 8(a). Noting that it is rare to use multiple switches for a single terminal, we compress the volatile data using a simple entropy coding, wherein empty registers are denoted by a single zero. If a switch is set in the register, then we use a four bit identification number to identify the location of the switch within the register. An example compression is shown in Fig. 8(b), where the 48-bit configuration of Fig. 8(a) is reduced to 19 bits.

By itself, this simple entropy coding compresses the volatile data to a size of $5N_{\mathrm{on}} + N_{\mathrm{reg}}$ bits, where N_{on} is the number of "on" switches and N_{reg} is the total number of volatile registers in the RAMP. To achieve higher compression, *AZiP* pre-processes the configuration prior to applying simple entropy coding. First, since the unused input terminals of hardware primitives are terminated to a supply net, most registers tend to have the same non-zero contents. Consequently, we determine the most common register value and use it as a bitmask. All registers are then encoded as the XOR with this mask, which makes the simple entropy coder more effective since the most common value to encode is zero. This mask is the first word in the compressed bitstream.

The next pre-processing step identifies repeating patterns in the design. Since the RAMP is designed for parallel processing, with each channel containing copies of the same hardware primitives, designs often perform the same function in each channel. In this case, *AZiP* can identify these commonalities so that they are only encoded once.

After these two pre-processing steps, *AZiP* encodes the volatile data one CAB at a time using the simple entropy coder. Then the nonvolatile analog data is compressed. Since the parameters in a parallel architecture will typically either increase monotonically from channel to channel or be constant from channel to channel, the parameters are compressed by encoding the deltas between channels. If the delta is less than four bits, then only the delta is encoded, otherwise the full value is encoded. In this algorithm, the settings are written into the RAMP as they are extracted, so it is not necessary to hold the entire expanded configuration in the mote's memory.

Figure 9 compares *AZiP* to three alternative compression methods. An uncompressed configuration is approximately 3KB. The "Full address for each write" line shows the file size if each register were encoded using it's address and contents. The "Simple entropy" line shows the file size if the scheme in Fig. 8(b) were used without pre-processing. The "LZW" data points show the results of compressing the data using the LZW algorithm that has previously been applied to sensor networks [13]. Note that the "AZiP" and "LZW" points show results for compressing both the volatile and nonvolatile data, whereas the the other lines are theoretical compression sizes for volatile data only. The advantages of *AZiP* for different file types are observed. By pre-processing the data, small design files, such as *i* and *ii*, as well as larger parallelized designs, such as *iv*, are both more efficiently compressed by *AZiP* than by any other single method.

5. APPLICATIONS

In this Section, we illustrate the use of the RAMP in interfacing with, and performing computations on, several different sensor types. Applications shown here utilize the same

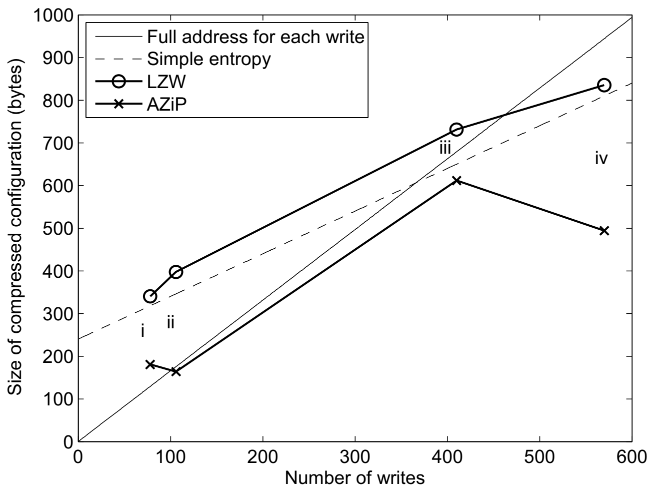


Figure 9: Results of *AZiP* compression on four designs: *i*) accelerometer double-tap detection, *ii*) internal temperature sensor, *iii*) heart-rate alarm, and *iv*) audio spectrum normalization.

RAMP IC, reconfigured for each application. These applications range from implementing the sensor in the RAMP IC itself to using external sensors for more niche applications. In each case, the circuit-level schematic is shown along with measured data from the output of the RAMP.

5.1 Sensing Using RAMP Components

The RAMP IC is actually capable of performing some basic sensing functions itself, without the need of connecting to an external sensor. Fig. 10(a) shows a bandgap-based temperature sensor that has been synthesized using only the components available among the RAMP's FPAA circuit elements. Additionally, this entire design illustrates the synthesis of a circuit using only device-level components—only resistors and transistors are used in this example. While most applications will want to make use of higher-complexity circuits, the inclusion of basic circuit components permits the synthesis of circuits that may be too specialized to have been included in a CAB. The output of this circuit provides a temperature measurement of 1mV/K, as shown in Fig. 10(b), and the entire circuit consumes only 12μW.

5.2 Signal Conditioning

The typical application of analog circuits in sensing systems is to acquire, amplify, and filter a sensor's output to prepare it for digitization—known as sensor conditioning. The RAMP has the ability to perform these basic signal-conditioning needs. Fig. 10(c) illustrates a portion of a typical signal-conditioning chain, focusing on the conversion of a resistance value to an electrical signal, which is typical of many resistance-based sensors such as thermistors and stain-gauge sensors. This circuit uses a Wheatstone bridge synthesized from resistors and op-amps to convert an external resistance sensor into a temperature measurement. This circuit consumes 38.4μW, most of which powers the bridge.

5.3 Body Sensor Network Application

Figure 11 illustrates an example system that reads in

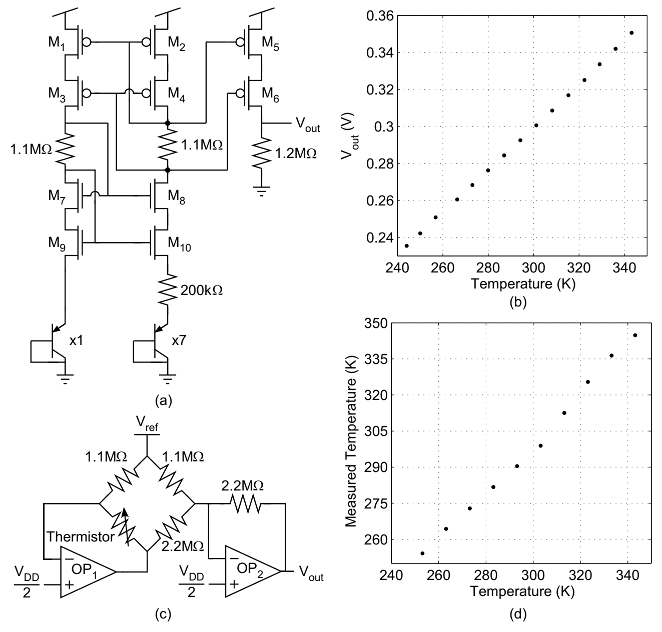


Figure 10: Demonstration of two temperature sensing circuits that were synthesized in RAMP. (a) Completely internal bandgap-based temperature sensor. (b) Measured output of the internal temperature sensor. (c) Wheatstone bridge circuit that was synthesized in RAMP to measure a 1MΩ NTC thermistor. (d) Measured temperature output of the thermistor-based temperature sensor.

a sensor input, collects information, and then generates a wakeup signal based upon the content of the signal. This heart-rate detector amplifies a small pulse signal and determines if the heart rate falls within a range of "healthy" heart rates for the individual. If the pulse rate goes too high or too low, an alarm is generated in the form of an interrupt signal. This system leverages both the analog and digital portions of the RAMP with several non-linear circuits compressing the incoming stream of values from the sensor into a simple difference in time values between subsequent heart beats. This entire system consumes only 20μW.

As a comparison for this body sensor network application, we examine a system that a developer may build with off-the-shelf components. A developer might choose the low-power ADS1191 ECG front-end (335uW) to implement the ECG amplifier, filter, and ADC. The remainder of the processing (i.e., QRS detection, BPM extraction, and alarm) would be done in the sensor node's MCU—for example the ubiquitous MSP430 at a power of 2mW. In this case, the RAMP offers a 100x system power reduction. Equally important, the RAMP simplifies the design—it is not necessary to develop a custom board with an ECG front-end for this one application; rather the RAMP provides a single generic interface that could be used for this and other applications.

5.4 Proximity Detector Application

While many commercially available sensors can be interfaced using linear circuits, such as the resistive-based sensor interface circuit of Fig. 10(c), other sensor applications typically require the overhead of using an MCU in order to

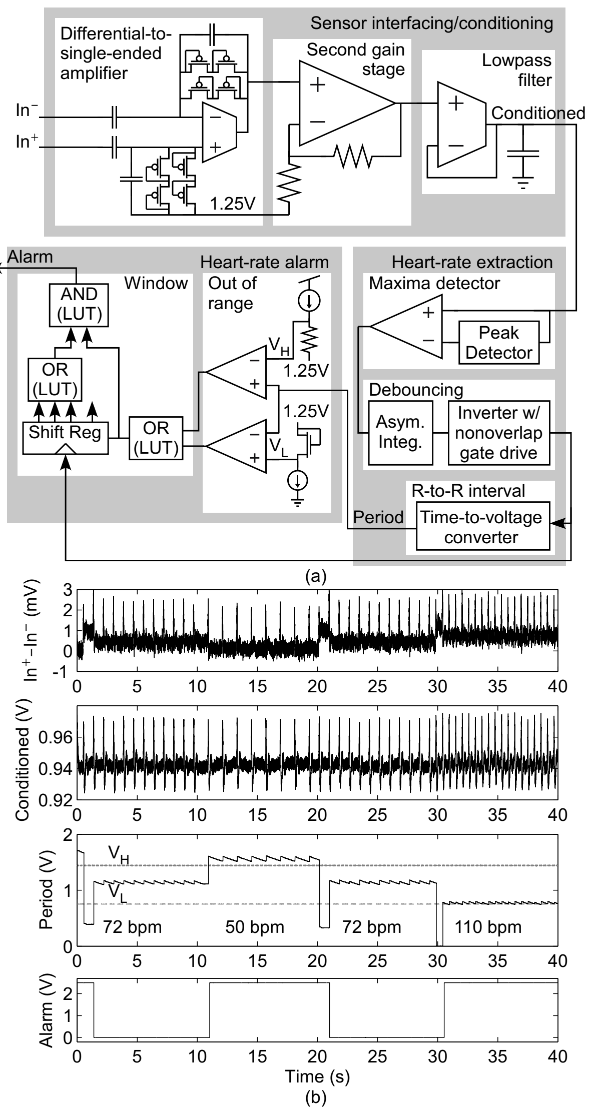


Figure 11: (a) Heart-rate monitoring system synthesized in RAMP. After passing through the conditioning block, a time-to-voltage converter is clocked by the peaks of the R wave to extract the period. The period is compared with user-defined high/low thresholds. If two recent periods are outside the safe range, then an alarm is generated. (b) Measured response. 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.

perform their tasks, thus increasing an application's power budget. However, the RAMP has the ability to control many of these sensor types while the MCU remains inactive. As an example, we constructed a proximity detector using an infrared (IR) emitter and photodetector, and in order to save power, these higher-power devices were pulsed on temporarily. An MCU would commonly be used to control the precise timing, but this same timing generation, as well as the comparison of signals to determine the presence of an object, was implemented in the RAMP, thereby reducing the need for the MCU in this application. Fig. 12 illustrates this application. In this application, the IR emitter is pulsed on, and a circuit compares the output of the photodetector, which measures the bounce-back signal, during times when the IR emitter is turned ON/OFF to determine the presence of an object. A wakeup signal is only generated when an object has been found to be in close proximity. The power consumption for the whole platform—including the IR emitter, which is pulsed with low duty cycle 1mA pulses—is 129μW. Without the IR emitter, the power consumption is 27μW.

6. CONCLUSION

In this paper, we presented a reconfigurable platform that enables the straightforward synthesis of a variety of custom circuits that can be used to improve efficiency and reduce the power consumption of wireless sensing systems. By placing this reconfigurable analog/mixed-signal platform, or RAMP, directly after the sensor and prior to digitization, this RAMP can perform many functions that would typically be done on a microcontroller. As a result, the MCU can remain in a low-power sleep state longer, thereby saving power, or it can be used to do more sophisticated processing since some of its resources have been freed up by the computation being done on the RAMP. Furthermore, since this platform was designed specifically for low-power processing, most applications consume very little power, with many systems operating at 20μW or less, which is less power than a sleeping TelosB mote[9].

We also presented a design environment that helps to separate the application development from the details of the circuit implementation. Using a hierarchical set of abstractions, applications can be developed without a detailed understanding of the underlying circuits. However, the suite of design tools, which incorporate netlisting and automated placement and routing, provide the option to "look under the hood" to observe the exact implementation and fine tune applications, as may be desired. The end result is a design environment that is simple enough that inexperienced users may quickly learn to develop applications, and that more experienced users have the ability to fully optimize their design. The RAMP platform provides a proving ground for demonstrating the capabilities of analog/mixed-signal synthesis of circuits, while maintaining ease of use through a flexible and extensible design environment.

7. ACKNOWLEDGMENTS

Portions of this material were based upon work supported by the National Science Foundation under Award Number 1148815 and by the Department of Energy under Award Number DE-FE0012383.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise

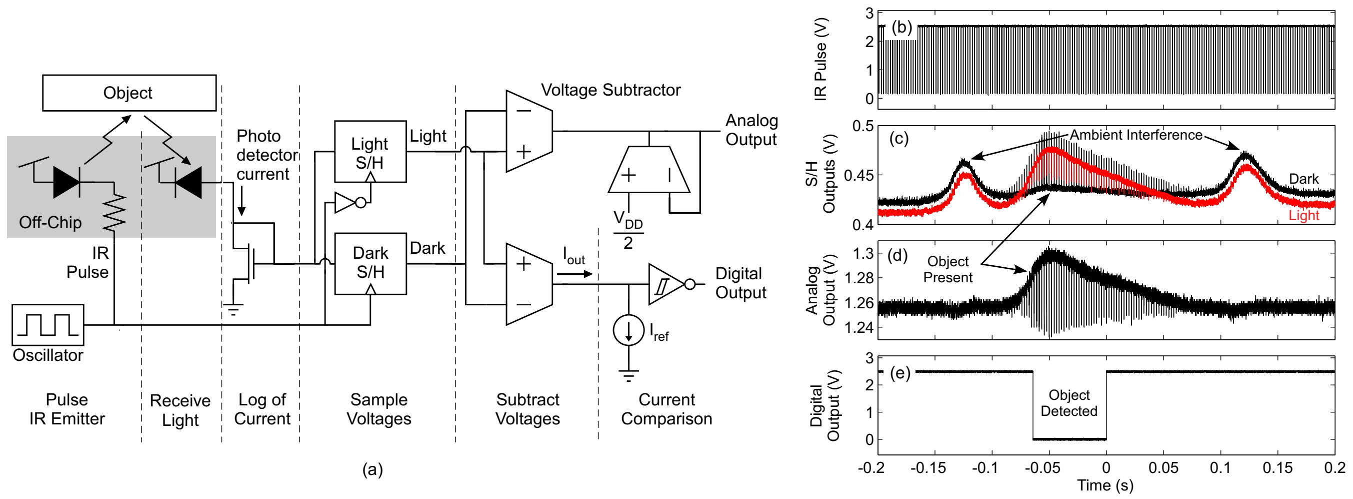


Figure 12: Proximity detector which was synthesized in the RAMP. (a) Functional block diagram of the signal flow. (b) Low duty-cyle pulse train that triggers the IR emitter. (c) Sampled "Light" and "Dark" outputs in the presence of ambient interference. An object is placed in front of the detect at approximately -0.075s. (d) Analog output shows rejection of ambient interference. (e) Falling-edge interrupt wakes the mote when an object is detected.

does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

8. REFERENCES

[1] V. Betz and J. Rose. VPR: A new packing, placement and routing tool for FPGA research. In *Field-Programmable Logic and Applications*, pages 213–222, 1997.

[2] R. Ellis, H. Yoo, D. Graham, P. Hasler, and D. Anderson. A continuous-time speech enhancement front-end for microphone inputs. In *Proc. IEEE ISCAS*, volume 2, pages 728–731, May 2002.

[3] D. Goldberg, A. Andreou, P. Julian, P. Pouliquen, L. Riddle, and R. Rosasco. A wake-up detector for an acoustic surveillance sensor network: Algorithm and VLSI implementation. In *Proc. IPSN*, pages 134–141, Berkeley, CA, 2004.

[4] S. Jevtic, M. Kotowsky, R. Dick, P. Dinda, and C. Dowding. Lucid dreaming: reliable analog event detection for energy-constrained applications. In *Proc. IPSN*, pages 350–359, Cambridge, MA, 2007.

[5] P. Lajevardi, A. Chandrakasan, and H. Lee. Zero-crossing detector based reconfigurable analog system. *IEEE J. Solid-State Circuits*, 46(11):2478–2487, 2011.

[6] H. Lu, A. Brush, B. Priyantha, A. Karlson, and J. Liu. Speakersense: Energy efficient unobtrusive speaker identification on mobile phones. In *Pervasive Computing*, pages 188–205. 2011.

[7] M. Malinowski, M. Moskwa, M. Feldmeier, M. Laibowitz, and J. Paradiso. Cargonet: A low-cost micropower sensor node exploiting quasi-passive wakeup for adaptive asynchronous monioring of exceptional events. In *Proc. ACM SenSys*, pages 145–159, Sydney, Australia, 2007.

[8] 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 *Proc. IEEE ISCAS*, pages 2145–2148, 2009.

[9] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proc. IPSN*, pages 364–369, Los Angeles, CA, 2005.

[10] B. Priyantha, D. Lymberopoulos, and J. Liu. Littlerock: Enabling energy-efficient continuous sensing on mobile phones. *IEEE Pervasive Computing*, 10(2):12–15, 2011.

[11] B. Rumberg and D. Graham. A floating-gate memory cell for continuous-time programming. In *Proc. IEEE MWSCAS*, pages 214–217, Boise, ID, August 2012.

[12] B. Rumberg, D. Graham, V. Kulathumani, and R. Fernandez. Hibernets: Energy-efficient sensor networks using analog signal processing. *IEEE JETCAS*, 1(3):321–334, Sept. 2011.

[13] C. Sadler and M. Martonosi. Data compression algorithms for energy-constrained devices in delay tolerant networks. In *Proc. ACM SenSys*, pages 265–278, 2006.

[14] R. Sarpeshkar. Analog versus digital: Extrapolating from electronics to neurobiology. *Neural Computation*, 10:1601–1608, Oct. 1998.

[15] C. Schlottmann, S. Shapero, S. Nease, and P. Hasler. A digitally enhanced dynamically reconfigurable analog platform for low-power signal processing. *IEEE J. Solid-State Circuits*, 47(9):2174–2184, Sept. 2012.

[16] E. Vittoz. Future of analog in the VLSI environment. In *Proc. IEEE ISCAS*, volume 2, pages 1372–1375, May 1990.

[17] R. Wunderlich, F. Adil, and P. Hasler. Floating gate-based field programmable mixed-signal array. *IEEE Trans. on VLSI Systems*, 21(8):1496–1505, 2013.