A COORDINATED APPROACH TO RECONFIGURABLE ANALOG SIGNAL PROCESSING

A Dissertation
Presented to
The Academic Faculty

By

Craig R. Schlottmann

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Electrical Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
August 2012

A COORDINATED APPROACH TO

RECONFIGURABLE ANALOG SIGNAL PROCESSING

Approved by:

Dr. Jennifer Hasler, Advisor
Professor, School of ECE
Georgia Institute of Technology
Atlanta, GA

Dr. James H. McClellan
Professor, School of ECE
Georgia Institute of Technology
Atlanta, GA

Dr. David V. Anderson
Professor, School of ECE
Georgia Institute of Technology
Atlanta, GA

Dr. Mark T. Smith
Professor, Dept. of Communications Systems
Kungliga Tekniska Högskolan
Swedish Royal Institute of Technology
Stockholm, Sweden

Dr. Aaron D. Lanterman
Assoc. Professor, School of ECE
Georgia Institute of Technology
Atlanta, GA

Date Approved: June 13, 2012

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Jennifer Hasler, for her support and guidance. I would also like to thank my committee for their insight and helpful comments: David Anderson, Aaron Lanterman, Jim McClellan, and Mark Smith. Of course, this work would have been impossible without my ICE Lab colleagues. I thank them for creating a fun and interesting atmosphere.

I am very fortunate to have a loving and supportive family. I thank my Mom, Dad, and sister, Dawn, for their encouragement. Most of all, I can never repay my ever patient wife and editor, Shannon.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The purpose of this research is to create a solid framework for embedded system design with field-programmable analog arrays (FPAAs). To achieve this goal, we've created a unified approach to the three phases of FPAA system design: (1) the hardware architecture; (2) the circuit design and modeling; and (3) the high-level software tools.

First, we describe innovations to the reconfigurable analog hardware that enable advanced signal processing and integration into embedded systems. We introduce the multiple-input translinear element (MITE) FPAA and the dynamically-reconfigurable RASP 2.9v FPAA, which was designed explicitly for interfacing with external digital systems. This compatibility creates a streamlined workflow for dropping the FPAA hardware into mixed-signal embedded systems.

The second phase, algorithm analysis and modeling, is important to create a useful and reliable library of components for the system designer. We discuss the concept and procedure of analog abstraction that empowers non-circuit design engineers to take full advantage of analog techniques. We use the analog vector-matrix multiplier as an example for a detailed discussion on computational analog analysis and system mapping to the FPAA.

Lastly, we describe high-level software tools, which are an absolute necessity for the design of large systems due to the size and complexity of modern FPAAs. We describe the Sim2Spice tool, which allows system designers to develop signal processing systems in the Simulink environment. The tool then compiles the system to the FPAA hardware.

By coordinating the development of these three phases, we've created a solid unified framework that empowers engineers to utilize FPAAs.

CHAPTER 1

INTRODUCTION TO ANALOG SIGNAL PROCESSING

Cooperative analog-digital signal processing (CADSP) is the design approach whereby the two domains (analog and digital) are used in combination to achieve advanced system performance [1]. In traditional systems, analog processing is used primarily for front-end amplification and data conversion, whereas the digital signal processor (DSP) handles the mathematical operations. By repartitioning the boundary between the processing domains, we stand to take advantage of extreme power and area savings. For instance, the natural physics of the subthreshold transistor can be used to perform many mathematical operations with a fraction of the number of devices required for digital computation [2] and a much lower total current draw.

Figure 1 illustrates Gene's law—a trend fit of the power efficiency progress for DSPs. The y-axis for this plot is in terms of the power required for performing a million multiply accumulate cycles a second (MMACS)—the most enlightening figure of merit for DSP power efficiency [3]. The result is striking when the performance of a *functionally equivalent* analog signal processor (ASP) is added to the plot [4]. Analog's computational efficiency is *several orders of magnitude* greater. This result was a (then) 20-year leap in efficiency, but the analog signal processor's advantage is compounded by the recent observation that the DSP is asymptotically approaching a power efficiency wall [5]. Now, analog processing is no longer simply just a "nice" opportunity for a quick leap, but rather, analog techniques will be *required* to meet future signal processing efficiency needs.

It is the efficient balancing of the analog and digital domains that the highest performance can be achieved. A popular subset of this concept is the notion of *digitally enhanced analog systems*, whereby digital processing is utilized to add resolution to analog blocks [6]. As a broader approach, CADSP additionally promotes the use of analog techniques to increase the power efficiency of digital blocks, as illustrated in the system of

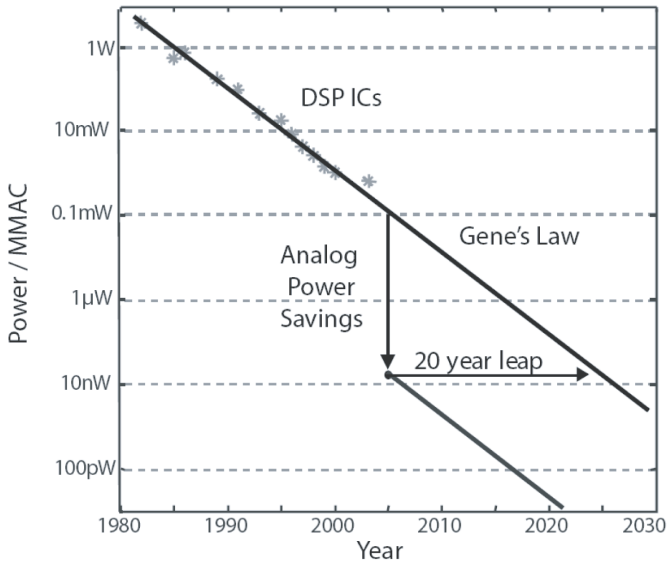


Figure 1: Gene's law shows the power efficiency trend for digital signal processors. Analog signal processing poses a 20-year leap in efficiency.

Figure 2. CADSP techniques have been successfully utilized in such systems as compressive sensing [7] and classifiers [8]. However, two very important hurdles have prevented the wide-spread use of analog computation—the traditional lack of both programmability and intuitive system design tools.

The recent development of large-scale field-programmable analog arrays (FPAAs) has provided a stable platform for programmable analog systems. The reconfigurable analog signal processor (RASP) FPAA is a VLSI system that contains hundreds of configurable analog blocks (CABs) and allows for on-the-fly synthesis of large-scale analog systems.

The FPAA has provided the hardware platform to develop ASP systems, but the remaining fundamental problem is that it is not always easy for the typical DSP engineer to utilize analog techniques. To solve this problem, intuitive design tools are needed to empower the non-circuit designer to take advantage of the FPAA hardware. Simulink has been chosen as the top level design space for analog systems on FPAAs in order to appeal to the broadest audience of DSP engineers. Although this design space is intuitive and makes systems easy to visualize and simulate, there has yet to be an established framework for the proper abstraction of analog design. The development of a high-level framework for abstracting

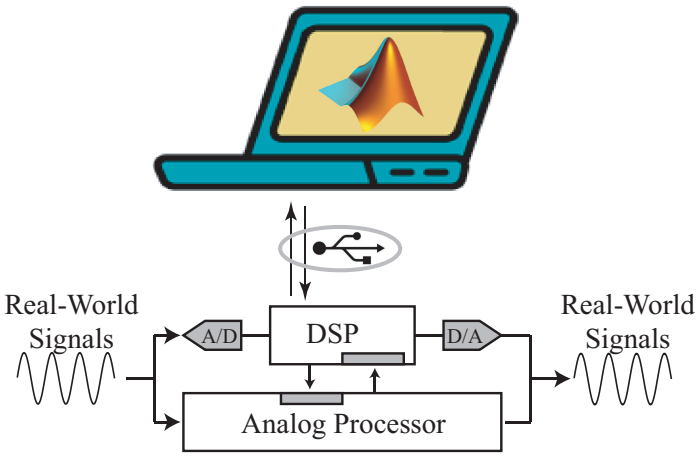


Figure 2: The analog processor embedded with a digital processor provides a power efficient platform. The incoming signal can be processed by the FPAA, the DSP, or by a combination of both. A custom MATLAB toolbox is used to program and control the mixed-mode processor.

analog design and creating behavioral analog blocks is necessary to bridge the analog and digital design gap for the system engineer.

The goal of this work is to define a coordinated approach for FPAA system design. A guiding role model in this endeavor is the Mead-Conway digital revolution of the 1980s [9]. Carver Mead and Lynn Conway helped to spark the VLSI boom by unifying the fields of computer architecture, integrated circuit design, and semiconductor device physics. Their work demystified the process for the system designer and increased the number of engineers entering the field. Their methodology for the merging of disciplines was based on presenting a small set of key concepts from a range of topics in order to carry along the least amount of mental baggage from topic to topic.

With the Mead-Conway approach in mind, we are attempting to unify the three phases of FPAA design: (1) the hardware architecture; (2) the circuit design and modeling; and (3) the high-level software tools. We are attempting to abstract the analog design to allow the system design engineer to take advantage of the benefits of analog technology. This approach is illustrated in Figure 3. The figure shows how the three areas of FPAA design overlap and how this dissertation will proceed.

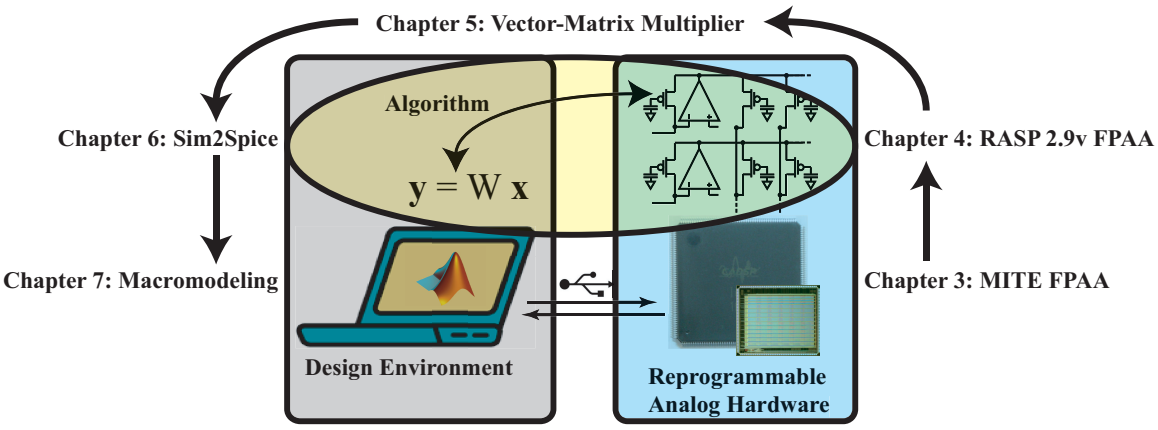


Figure 3: Diagram of the coordinated approach to FPAA design.

Chapter 2 provides a review the fundamental background technology. The two key elements covered are floating-gate transistors and field-programmable analog arrays. The floating gate (FG) is important for analog storage, and the FG transistor is used as a switch as well as for computation. We review the device characteristics—including subthreshold operation—as well as the processes for adding and removing charge from the gate. In the review of FPAA technology, we go over a brief history of their evolution and describe some of the various architectures that have been tried. We then provide a detailed description of the RASP FPAA, which will be the hardware platform used in the rest of this dissertation.

Chapter 3 introduces the multiple-input translinear element (MITE) FPAA. This architecture is based on the MITE as a circuit primitive, which is ideal for computing polynomial equations. There is a robust body of work on the synthesis of high-order static and dynamic equations to systems of MITEs, which makes it an ideal platform to get started with analog signal processing.

Chapter 4 introduces the RASP 2.9v, a next-generation FPAA architecture that is optimized for embedded systems. This architecture was motivated by the need for higher-level digital control for dynamic reconfigurability. The controllability allows the FPAA to be much more easily fielded in embedded electronic systems.

Chapter 5 presents the analog vector-matrix multiplier (VMM) as a bottom-up case

study for the analysis of a computational analog element and its mapping to the FPAA architecture. The VMM is one of "killer apps" of analog computation in that it efficiently computes a very common signal processing function. The circuit design process includes the step-by-step choices motivated by the architecture of the FPAA hardware. The circuit analysis includes a description of the system's power, speed, noise, and temperature dependence.

Chapter 6 introduces the high-level software tools for FPAA configuration. The top-level design space is Simulink, which provides an intuitive platform for designing signal processing systems with functional blocks. The core elements of this system are the component library and the FPAA compilers. The component library contains abstracted analog blocks that allow non-circuit designers to create large systems with analog blocks. The compilers include two main tools: (1) Sim2Spice for converting Simulink designs to a circuit netlist; and (2) the GRASPER tool for converting the netlist to FPAA targeting code.

Chapter 7 discusses the challenges and opportunities of abstracted analog design. By defining a standard interface for the analog block library, designers can treat analog blocks more like their digital counterparts. This standardization provides a low barrier-to-entry for engineers who are skilled in the *problem domain* of signal processing, but may have limited experience in the *solution domain* of analog hardware. A major focus of this work is the creation of accurate, yet elegant, macromodels for the simulation and overall understanding of the function of the analog signal processing blocks.

Lastly, Chapter 8 recaps the overall impact of this dissertation. Personal contributions are noted and some directions are provided for the future of this work.

CHAPTER 2

FUNDAMENTAL BACKGROUND

This chapter provides a brief background on two core elements of technology for reconfigurable analog systems—the floating-gate transistor and the field-programmable analog array. Our intent is to introduce these topics and provide a self-contained document, but not delve too deep onto them (as per the Mead-Conway approach). We've provided a thorough set of references for further reading about each topic.

2.1 Floating-Gate MOSFETs

The fundamental piece of technology for the reconfigurable system is the floating-gate transistor. Originally reported in 1967 [10], these are ordinary transistors whose gates are then entirely surrounded by electrical insulation (i.e., with no DC path to ground), which allows stored charge to be retained. Floating-gate transistors have firmly established themselves in digital circuits as a reliable non-volatile memory storage, used in Flash and EEPROM. Recent research has been exploring their applications in analog circuits such as multiplier weights, neuromorphic synapses, analog memory, bias generation, and offset removal.

An important feature of these devices is that they can be fabricated in a standard CMOS process. The schematic and layout for a floating-gate pFET is shown in Figure 4. The polysilicon (poly 1) gate is shown in red has no direct contacts; it is completely surrounded by oxide. The floating node is actively controlled by two voltage signals capacitively coupled onto it: the tunneling voltage and the gate voltage. A MOS capacitor is used for tunneling because a high quality oxide is needed and a poly-poly capacitor is used for the gate voltage for its linearity.

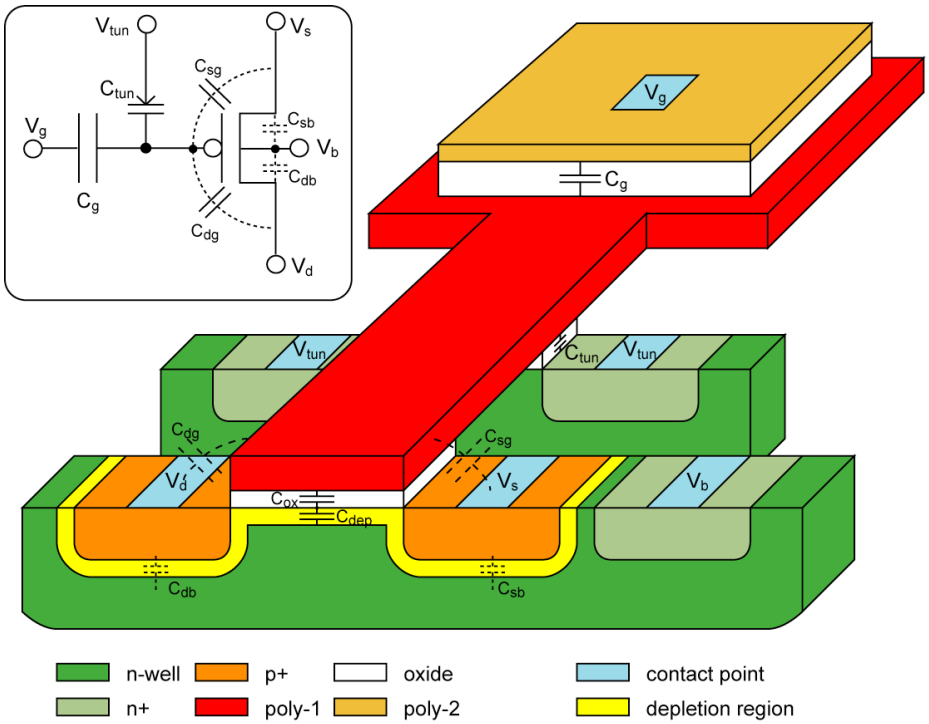


Figure 4: Floating-gate transistor technology. The distinguishing feature is that the polysilicon gate is completely insulated by oxide. The input gate voltage is coupled in through a poly capacitor and the tunneling voltage through a MOS capacitor.

2.1.1 Floating-Gate Transistor Characteristics

To accurately use floating-gate transistors, the I-V relationship along with the stored charge's affects need to be defined. For low power operation, all FETs discussed in this dissertation will be operated in the subthreshold (weak inversion) region. In subthreshold, the drain current (I_D) is given as

$$I_D = I_0 e^{\frac{V_S - \kappa V_{fg}}{U_T}} e^{\frac{V_D}{V_A}}, \tag{1}$$

where κ is the capacitive division between the oxide capacitance and the depletion capacitance $\left[C_{ox}\left(C_{ox} + C_{dep}\right)\right]$, U_T is the thermal voltage (kT/q), and V_A is the Early voltage. Rather than a fixed gate voltage, V_{fg} is the capacitive sum of the control voltages and the stored charge,

$$V_{fg} = \frac{1}{C_T}\left(C_g V_g + C_{tun} V_{tun} + Q\right), \tag{2}$$

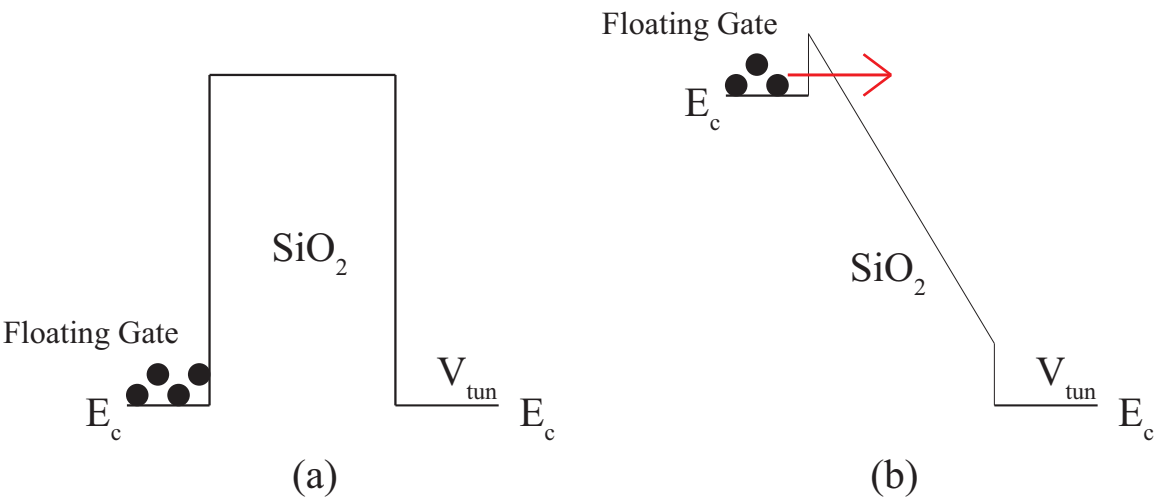


Figure 5: Band diagram of electron tunneling.

where $C_T = C_g + C_{tun}$ is the total capacitance at the gate and Q is the charge trapped on the floating node. Note that all of the voltages are referenced to the bulk, which is V_{DD} for the pFET. For the condition where V_{tun} is 0 V, Equation 2 can be reduced to

$$V_{fg} = \frac{C_g}{C_T} V_g + V_{offset}, \tag{3}$$

where V_{offset} is Q/C_T.

Equation 3 shows that this floating-gate transistor behaves very similar to a traditional pFET, but with a programmable offset on the gate. Precise control over this offset (the stored charge) is what makes floating-gate technology so important as an analog memory.

2.1.2 Floating-Gate Charge Modification

There are two well-documented procedures that are used to accurately modify the charge on a floating-gate transistor. Fowler-Nordheim tunneling is used to remove electrons and hot-electron injection is used for add electrons. With this combination of processes, one can precisely program the transistor to have any stored value at its gate.

Tunneling is the process for removing charge from the floating gate. The charge removal occurs when an electron is made to pass through a barrier rather than following its conduction band. Fowler-Nordheim tunneling is the procedure commonly used to induce this phenomena. The concept of tunneling is shown in Figure 5.

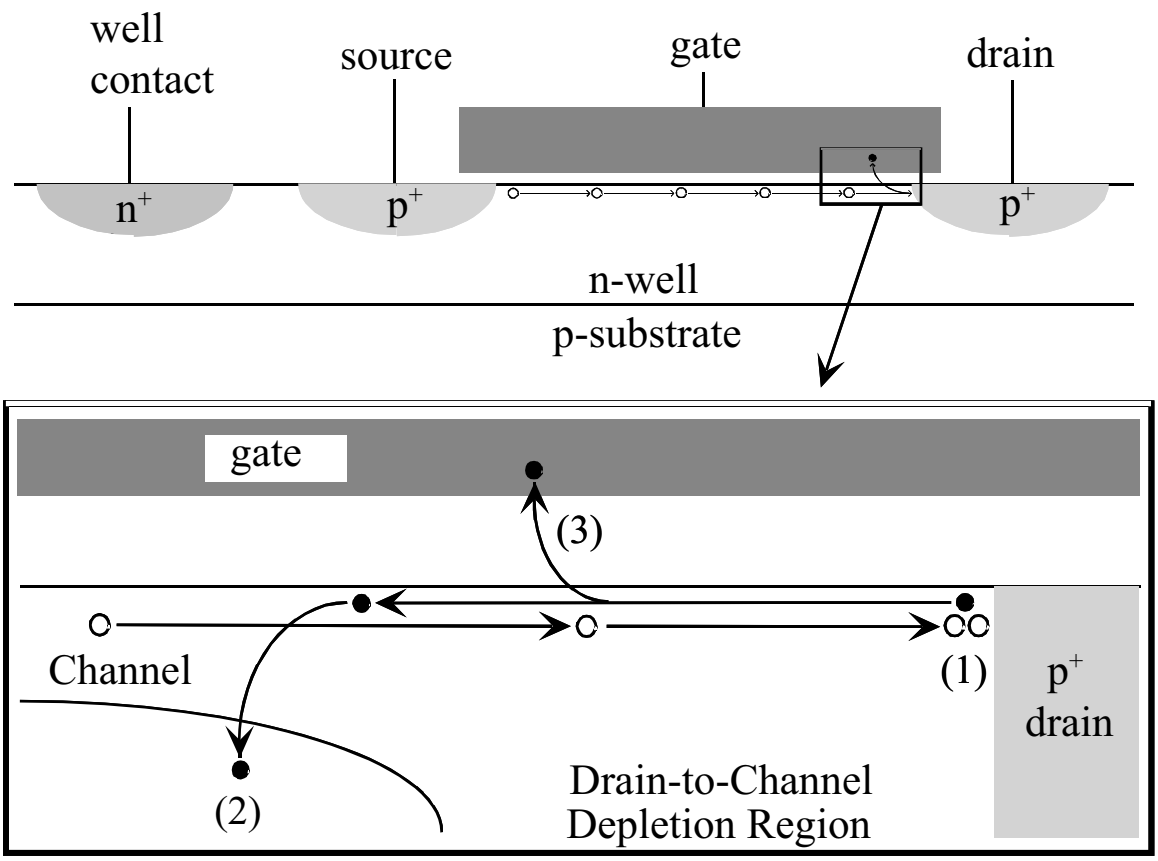


Figure 6: Channel diagram of hot-electron injection.

For a reasonably sized silicon-dioxide insulator, such as the MOS capacitor shown in Figure 5a, the electronic barrier is high enough to prevent conduction under normal conditions. However, when a large electric field is applied across the capacitor, the bands are bent so steeply that the electrons see a much thinner barrier. At a certain point, the barrier is thin enough that the electrons can pass right through it, as shown in Figure 5b. The ultimate result of this process is a decrease in electrons on the floating node and thus an increase in the effective gate voltage by means of V_{offset} in Equation 3.

Hot-electron injection is the process used to add electrons back to the floating gate. This process is shown in Figure 6. Injection is performed by providing two conditions to the floating-gate transistor—a high drain-source potential and a high gate-source potential. The gate potential creates a conducting channel in the device and the high drain potential creates a large field between the drain and source. Under these conditions, when a minority

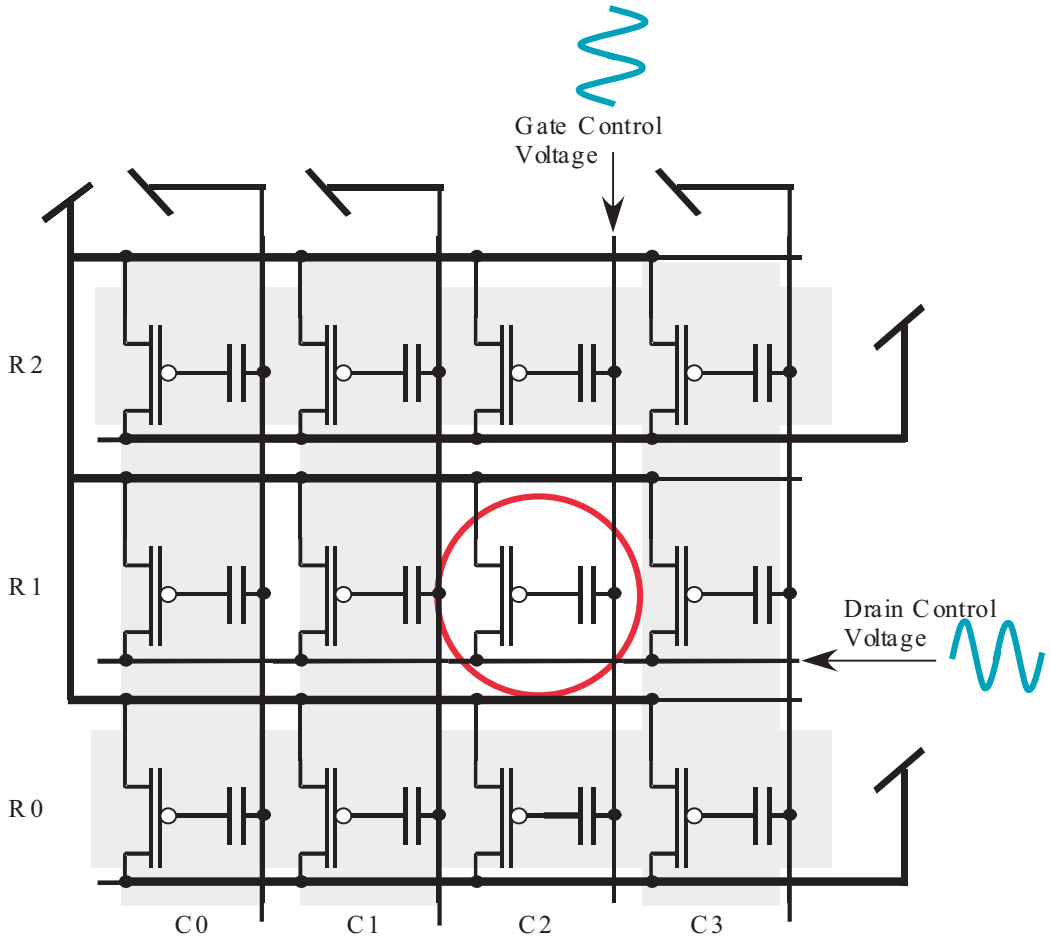


Figure 7: Floating-gate array isolation.

carrier enters the channel, it is accelerated with high energy toward the drain. When this carrier collides with the drain, it impact ionizes and creates an electron-hole pair. At this stage, with the high field from the gate, some of these "hot" electrons have enough energy to pass through the oxide to the gate region. The net effect is an addition of negative charge to the floating gate, lowering the effective gate voltage (V_{offset}).

2.1.3 Array Programming

An intelligent selection procedure has been developed to rapidly program arrays of multiple floating-gate devices [11]. This array programming plays off the requirement for two control voltages to perform injection. By arranging the transistors in a two-dimensional array, shown in Figure 7, each floating-gate transistor can be individually targeted based

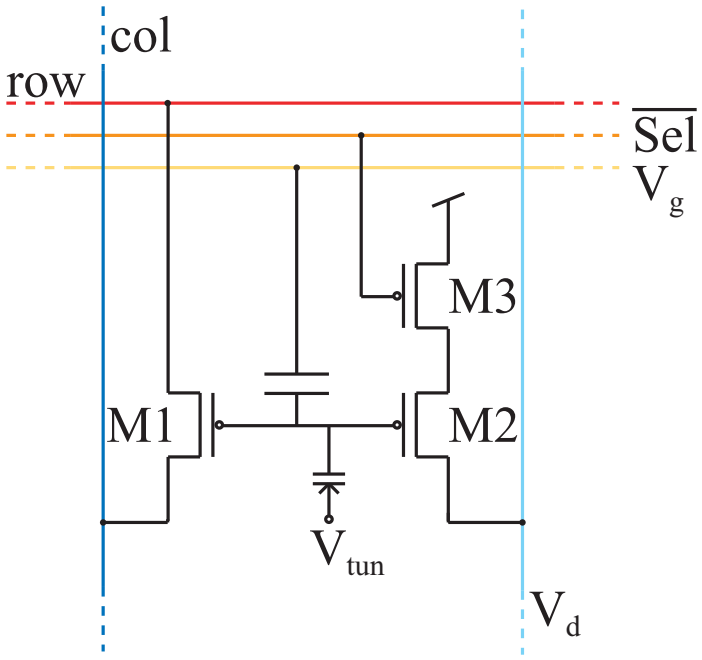


Figure 8: Indirect floating-gate switch cell.

on its row and column address. This two-dimensional addressing lends itself nicely to the two-parameter injection. By tying all of the gates of a particular dimension together and the drains of the other dimension together and then applying an appropriate gate/drain voltage to the desired row/column, only the element at the intersection will be under the right conditions to inject. On the other hand, tunneling only involves one parameter—the voltage coupled across the MOS capacitor. Therefore, in array programming, tunneling is used as a global erase and injection is used to program particular elements.

Figure 8 shows the architecture of a single floating-gate element as it appears in an array. This device illustrates the detail involved to incorporate indirect programming [12]. Transistor M1 is at the intersection of the row and column lines and is therefore in the signal path. If directly programmed, the floating-gate transistor needs to be disconnected from the circuit to provide proper control and isolation to not create unintended injection. To disconnect, a 2-to-1 multiplexer would be needed for each floating-gate transistor's source and drain. The addition of this multiplexer increases the overall switch count and thus, parasitics in the signal path.

By using indirect programming, the in-circuit transistor (M1) does not have to be disconnected, the other pFET (M2) is tied to the drain line and the select circuitry. Transistor M2 shares M1's floating gate so programming one will generate the same bias on the other. Now, injection can be performed on M2, with the resulting charge being deposited on the common floating gate without the need for any disconnection circuitry. In addition to decreasing the parasitics, this layout also decreases the area of the cell and increases the programming speed.

2.2 The Field-Programmable Analog Array

Field-programmable analog arrays (FPAAs) are mixed-signal systems that provide a platform for rapidly implementing analog systems in real hardware. FPAAs consist of a programmable network of switches and routing used to electrically connect analog elements, arranged in computational analog blocks (CABs). The user of such a device can design, program, and test large scale analog systems in a matter of minutes. This platform provides the engineer a considerable decrease in fabrication costs and speedup in time-to-market.

While FPGAs have been developed for commercial use, FPAAs have not had the same success. The chief barrier to commercial success is the lack of a universal block from which analog circuits could be systematically built, as gates are to digital circuits. A survey of the FPAA landscape illustrates this lack of consensus as to the appropriate level of granularity.

2.2.1 FPAA Topologies

Originally reported about twenty years ago, FPAAs have seen several architectural revisions. Most have been of modest proof-of-concept size and contained CAB elements that target particular applications.

An early FPAA work was Gulak's, in which he used "switch blocks" of cross-bar switches and a shift register to control the connections between CABs [13]. The CABs in his design were arranged so that they could only implement one of seven limited functions (variations of comparing and multiplying), which was determined by a three-bit code

applied to it. A later, more fine-grained approach was the field-programmable transistor array (FPTA), which used a sea of single transistor that had to be connected together to realize a system [14].

On the other end of the granularity spectrum are the coarse-grained architectures that target a specific application. The analog ODE co-processor from Tsividis's group is one such chip [15]. His chip is composed of analog mathematical computational blocks like integrators and gains stages for accelerating computer simulations. Although limited in general application, the chip advanced the field of embedded FPAA systems by pushing the importance of having easy digital control from a master device. Another example of a very coarse-grained FPAA is the hexagonal $G_m - C$ architecture from Becker's group [16]. This chip was explicitly targeted to OTA-C filters where precise knowledge of node capacitance is important for high-order filters. His novel approach was to hexagonally arrange 55 OTAs where there are no "switches" between blocks, but systems are synthesized by only turning on certain OTAs. The chip reported good performance for signal continuous filters—its only application.

There have also been a couple commercially available FPAAs. Anadigm's FPAA and their software package, Anadigm Designer, use switched-capacitor circuits to realize circuits [17]. Anadigm demonstrates how important an intuitive tool flow is to a chip's acceptance. They have a graphical top-level design tool that makes systems easy to draw and understand. However, their hardware is not only extremely small (4 CABs), it is really only targeted to switched-capacitor systems. Another player in the commercial market is the Cypress PSoC (programmable embedded system on chip) [18]. Although not a true FPAA—the analog components are limited to a few tunable ADCs, DACs, 4 op-amps, 4 comparators, etc.—it introduces the concept of integrating a microcontroller with configurable analog and digital peripherals.

2.2.2 The Reconfigurable Analog Signal Processor

The largest advancement in FPAA technology, in terms if size and versatility, has been the reconfigurable analog signal processor (RASP) line of FPAAs. The RASP 2.8 is composed of 32 CABs, multilevel routing, and on-chip programming structures. While there are several versions of FPAAs in the 2.8 family—defined by the CAB components—it is the general-purpose RASP 2.8a that is suitable for most applications [19, 20].

Of the 32 CABs in the RASP 2.8a, 28 each contain 16 common analog components: 1 programmable bias operational transconductance amplifier (OTA), 2 programmable input and bias OTAs, 1 OTA buffer, 4 n/pFETs, 2 multiple-input translinear elements, 4 500 fF capacitors, and 1 transmission gate. The other 4 CABs contain elements for signal-by-signal multiplication, i.e., Gilbert multipliers. The routing is a full crossbar switch matrix. There are multiple levels of routing lines to reduce the capacitance nodes: local, nearest neighbor, and global.

The core element that facilitates highly dense FPAA systems is the floating-gate (FG) MOS transistor. No external memory is required when utilizing this switch because it stores its own value. The FPAA switch topologies are shown in Figure 9, which includes an indirect programmed FG [12]. One of the most beneficial features of FG switches is that they can also be used as computational elements [21]. FGs can be programmed to hold any value between ON and OFF, essentially providing a free bias source.

Figure 10 shows the architecture of the RASP 2.8a FPAA and the connections between the CABs, routing, and FG switches. The layout is shown in Figure 11, a die photo is in Figure 12, and relevant parameters are given in Table 1. The chip has been fabricated in a 350 nm double-poly CMOS process through TSMC and is 3 mm $\times$ 3 mm. One advancement in this line of FPAAs is the use of nearest-neighbor routing as an alternative to the existing horizontal/vertical globals and locals. This extra layer of routing creates direct connections to the nearest CAB to the left, right, top, and bottom. A system of bridge switches was also introduced. The bridges allow for more lines to be drawn as locals and

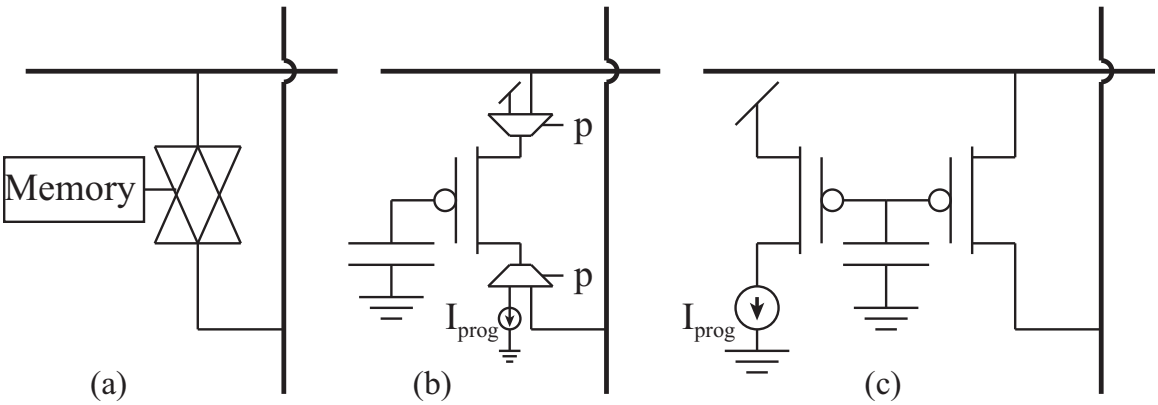


Figure 9: The three types of FPAA switches. (a) The traditional switch with a separate memory element. (b) Floating gate switch elements store their programmed value, allowing for a very area efficient design. (c) Indirect programming reduces the switches in the signal path.

Table 1: RASP 2.8a device parameters.

Process	350 nm
Die Size	3 mm × 3 mm
Power Supply	2.4 V
Injection V_{DD}	5.6 V
Number of CABs	32
Switch programming time	$N_{rows}\times$ 100 μs
Bias programming time	5 ms/element
Programming accuracy and range	9 bits over 100 fA to 10 μA

then connected to the top and bottom local lines if needed. Table 2 shows that by using these shorter connections, the line capacitance is greatly reduced.

Another advantage presented by the RASP 2.8 line of FPAAs is the incorporation of on-chip programming [22]. Moving the programming on chip has allowed for much higher speed operation. The major contributor to this increase in speed comes from the floating-point current ADC. This is a ramp ADC with an adaptive logarithmic I-V converter on the front end, which allows for conversions of seven decades of current in 200 μs, which is much faster than off-chip measurements would take. DACs are also on-chip for setting

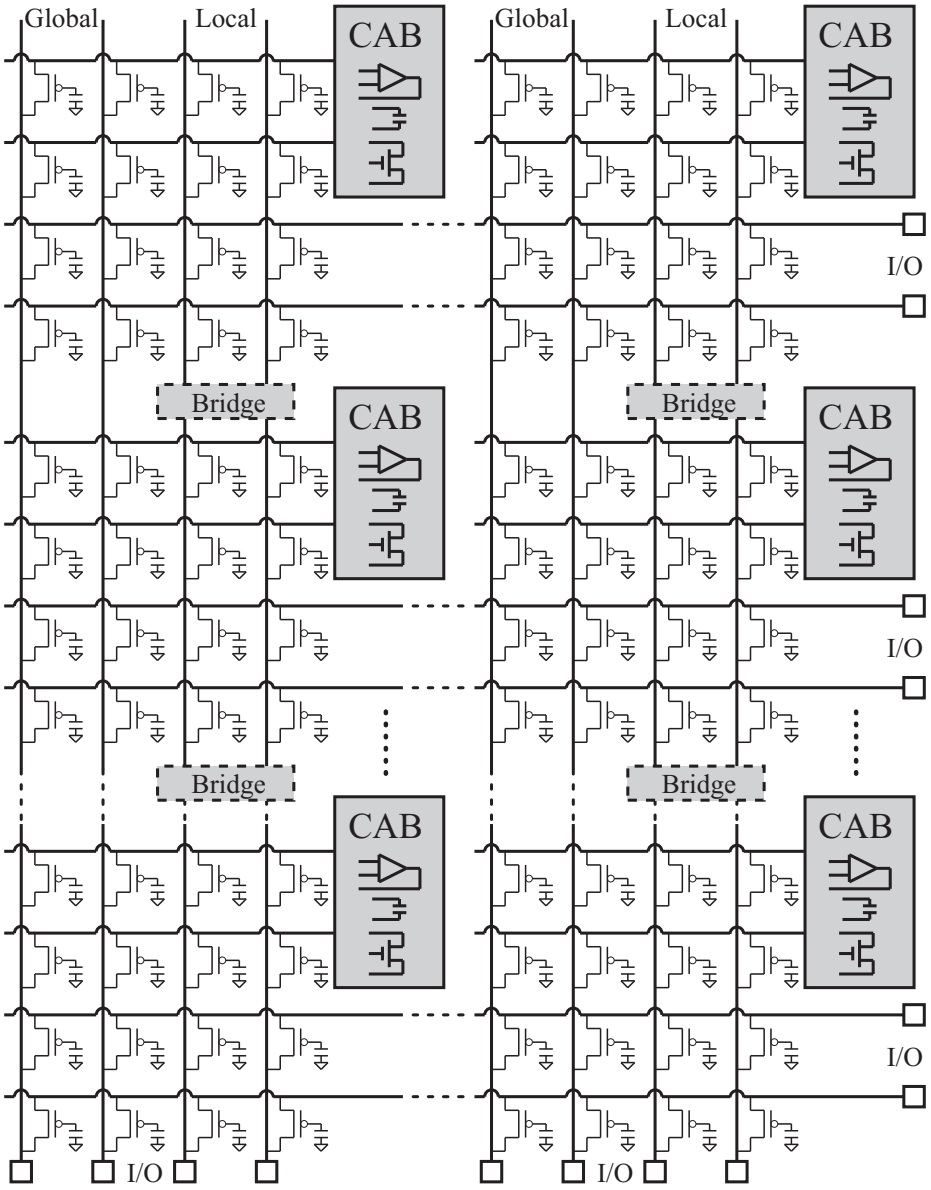


Figure 10: The architecture of the RASP 2.8 FPAA. The routing is a full crossbar switch matrix with floating-gate switches. The FG switches can be used for computation, such as for vector-matrix multiplier (VMM) weights.

Table 2: RASP 2.8a routing line capacitance values.

Nearest neighbor vertical	151 fF
Nearest neighbor horizontal	228 fF
Global	763 fF

the gate and drain voltages during injection. When precisely programming (injecting) a device, the present current value is measured and digitized with the I-to-V ADC. This voltage reading is then sent off-chip to a microcontroller and used to calculate how many and what size pulses are needed to converge on the target current. These values are then passed back to the chip's shift register through an SPI interface. This register is used to control the selection lines as well as the gate/drain DACs. A custom printed circuit board (PCB) was built to control all of the off-chip aspects of the programming, as well as for testing the chip.

To increase the size of synthesizable systems, the RASP 2.9a was recently introduced. This FPAA maintains the exact same architecture of the RASP 2.8a, but on a larger scale. The RASP 2.9a was fabricated in 350 nm CMOS and with a size of 5 mm $\times$ 5 mm was able to hold 84 CABs. By using the same architecture, all of the infrastructure from the RASP 2.8a is able to be utilized. This chip can synthesize much larger systems by virtue of it having more CABs. However, in Chapters 3 and 4 we will introduce two RASP FPAAs with new architectures that change the whole FPAA application space.

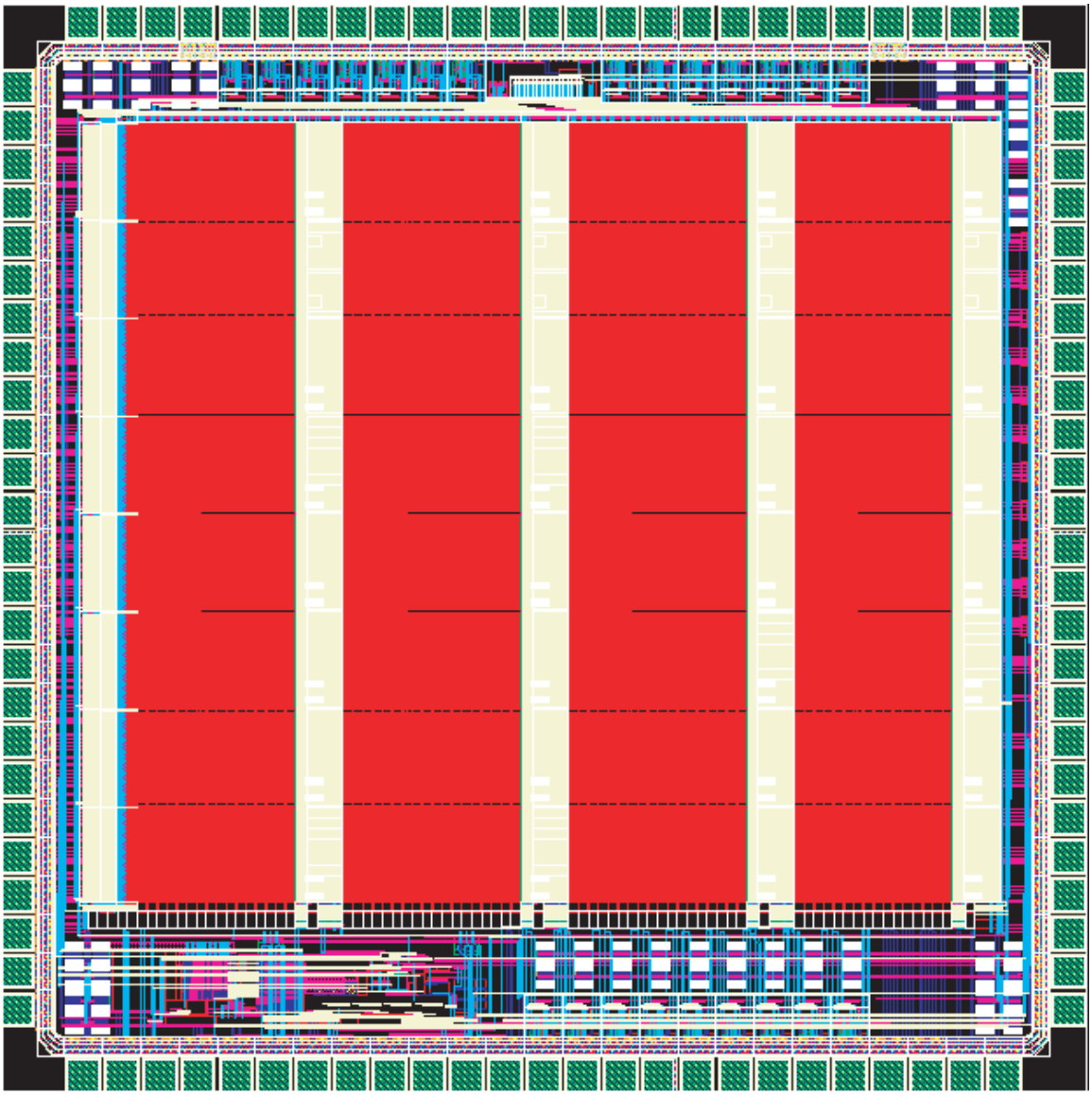


Figure 11: RASP 2.8a layout.

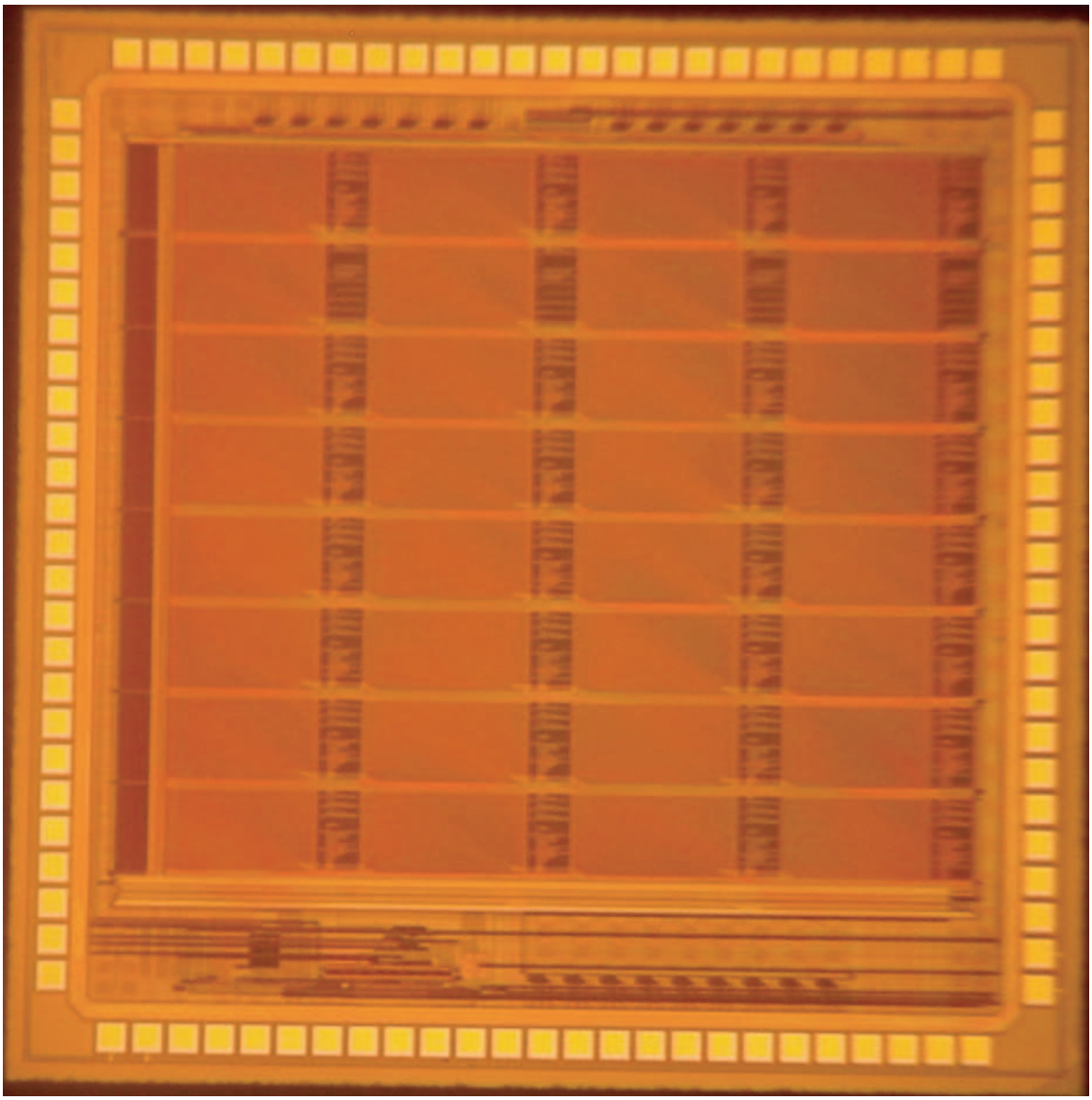


Figure 12: RASP 2.8a die photo.

CHAPTER 3

THE MITE FPAA

The coordinated approach to analog signal processing begins with looking at the right hardware *primitives*. To maximize the computational efficiency of FPAAs, we need to pick a core processing element that can be readily applied to many common signal processing systems. Whereas in FPGAs, it is easy to abstract almost any digital system to look-up tables (LUTs) and flip-flops, the reconfigurable analog processor is much more open ended as to what the most efficient primitive should be.

This chapter presents the MITE FPAA (MFPAA), which utilizes multiple input translinear elements (MITEs) as the core computational unit [23]. This FPAA utilizes a novel MITE unit, which takes advantage of commonly connected nodes while still fitting into a reconfigurable framework. By carefully designing this hardware structure, we can fully utilize existing synthesis algorithms for large-scale MITE systems. This novel architecture allows for a synthesis procedure that is elegant in its simplicity and lets us fully abstract the circuit design for the user. Thus, by using this full-system approach to FPAA design, a complete tool chain is possible: the abstracted software design environment, the place-and-route and programming tools, and the analog hardware. This entire platform will open up MITEs to new audiences as a design tool for implementing low-power signal-processing

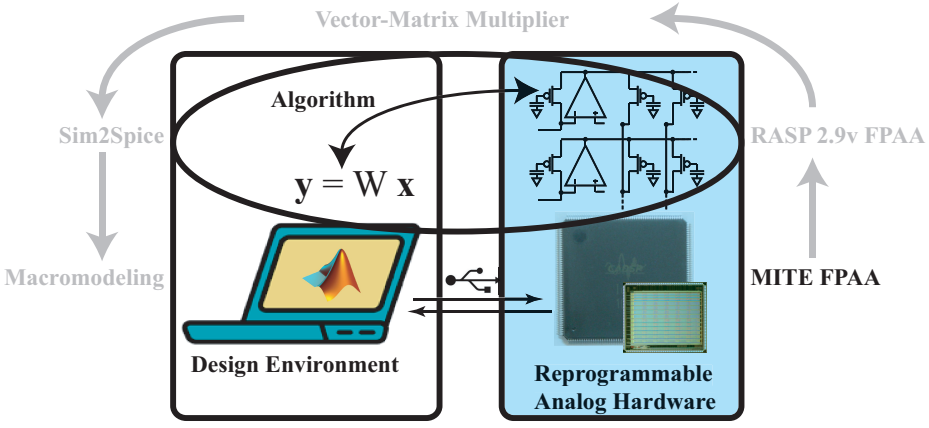


Figure 13: The coordinated approach to FPAA design: MITE FPAA.

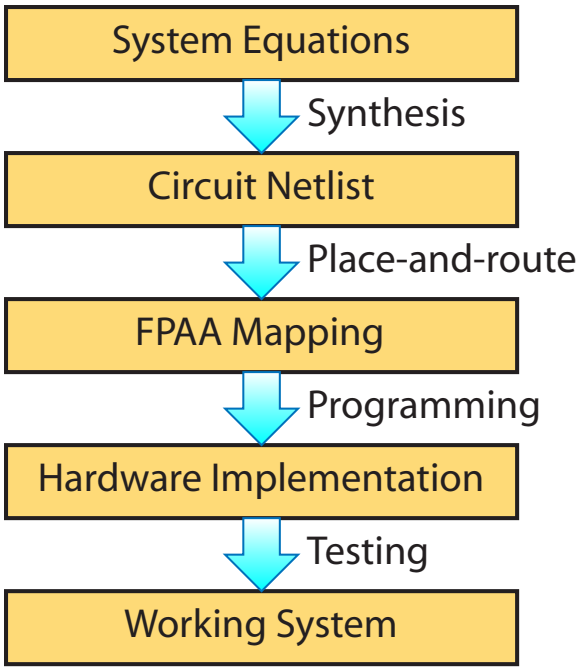


Figure 14: Design flow using a translinear FPAA. Using translinear circuits allows the user to enter a set of equations, which is then netlisted using existing synthesis procedures. The circuit is then place-and-routed, and the system is programmed onto the FPAA.

systems. Figure 13 shows how the MFPAA fits into the coordinated-design framework.

The use of translinear circuits as the universal analog block to reduce the trade off between flexibility and abstraction level has been gaining a lot of recent attention [23, 24, 25]. Using translinear circuits for which known network synthesis procedures exist [26, 27], it is possible to build a system in which the only input necessary is the set of equations that describe the system to be implemented. The translinear FPAA will be able to implement a wide range of circuits, including all linear static equations and most differential equations, while requiring the user to perform no actual analog design. This idea is illustrated by the translinear FPAA design flow, shown in Figure 14. Unlike the traditional FPAA design flow, there are no design or simulation steps required to create the working system. This will allow users with a background in math, controls, physics, or many other fields to easily interact with the FPAA.

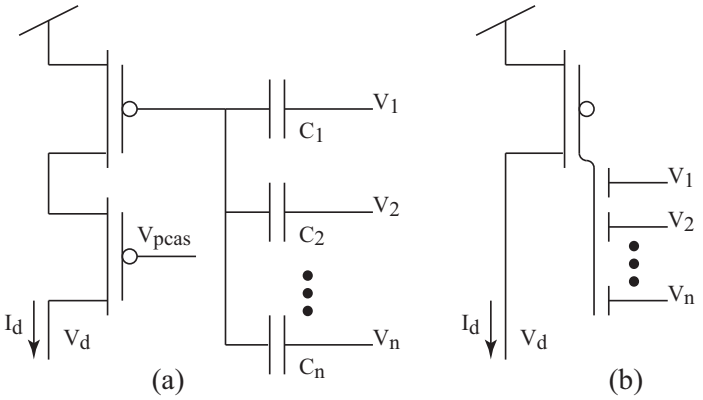


Figure 15: Subthreshold pFET realization of a MITE. (a) Components used to realize a MITE in a standard CMOS process. (b) Symbol used to represent a MITE.

3.1 Multiple Input Translinear Elements

Ideal translinear elements have infinite input impedance and an exponential voltage to current relationship independent of the current level at which they are operating. In addition, any translinear element can be made to have multiple inputs by simply applying resistive or capacitive division at the voltage input. MITEs can thus be built using either subthreshold MOSFETs or BJTs, each of which is stronger in one of the two above specifications [26]. Subthreshold pFETs were chosen for the MITE FPAA to allow the practical implementation in common CMOS process. The pFET has a current that is exponentially related to its gate voltage given by

$$I_d = I_0 e^{\frac{V_s - \kappa V_g}{U_T}} \left(1 - e^{\frac{V_d - V_s}{U_T}}\right), \tag{4}$$

where I_0 is a pre-exponential constant term, κ is the capacitive division between the oxide capacitance and the depletion capacitance, and U_T is the thermal voltage (kT/q). Note that all voltages are referenced to the bulk, which is the well voltage for the pFET. Furthermore, as long as the device is in saturation, $V_{sd} > 100$ mV, the second exponential term can be neglected.

Figure 15 shows the subthreshold pFET realization of a MITE, with capacitive division used for the introduction of multiple inputs. The current-voltage relationship for this

element is given by

$$I_d = I_0 e^{\frac{V_s - \kappa \sum (w_i V_i)}{U_T}}, \tag{5}$$

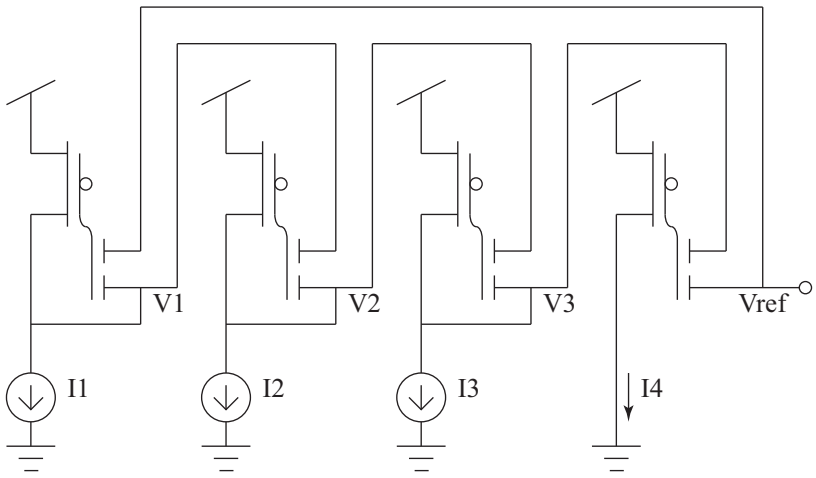
where w_i, the dimensionless weight applied to an input, is given by C_i/C_T, and C_T is the total capacitance at the gate of the pFET. Figure 15b shows the symbol that will be used for this realization of a MITE. Note that while the subthreshold MOSFET does have nearly infinite input impedance, the range in which the relationship between current and voltage is exponential is limited. However, by making the W/L ratio of the MITEs larger, this range can be increased.

To precisely set the charge on the floating node of the floating-gate pFET, a programming method is used that utilizes two quantum processes: Fowler-Nordheim tunneling and hot-election injection. This method of programming is vastly superior to simply removing the charge with UV radiation, because the charge can be precisely set, thus removing any offset between devices. Historically, gain errors induced by charge mismatch between devices have had a crippling affect on large-scale MITE systems [28]. The specialized MITE structure in [29] was developed for it to be compatible with the FPAA programming core. Of particular importance, the use of the on-chip programming core comes at no additional overhead as it is already built in to program the floating-gate switches [22].

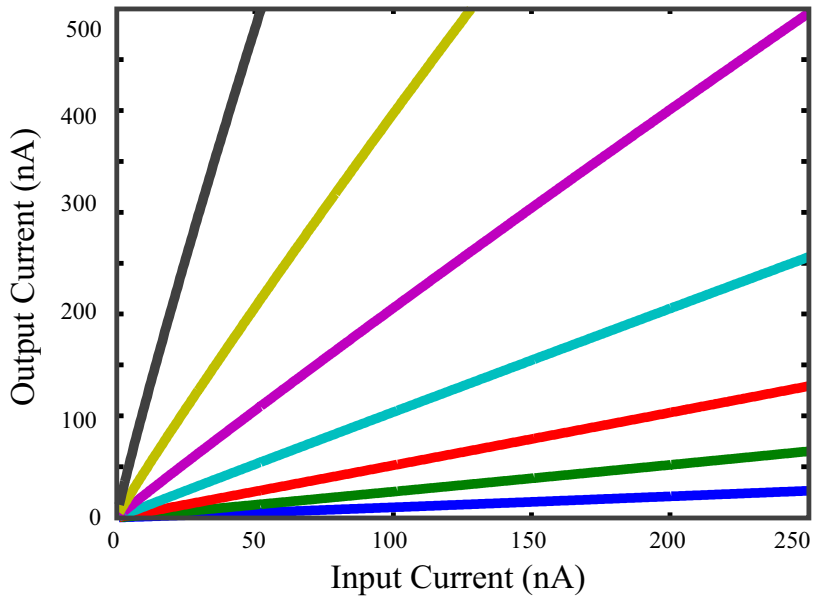
To build complex systems using MITEs, it is necessary to explore what higher level components are commonly used. Translinear loops and log-domain filters will be emphasized because they are commonly used as core elements in most synthesis procedures.

3.1.1 Building Block: Translinear Loops

Translinear loops are well documented building blocks of almost every translinear system [26, 30]. In a reconfigurable system, fixed loops are used to reduce the amount of reconfigurability needed. For the reconfigurable system we will use the translinear loop shown in Figure 16, which can be analyzed by simply solving for each MITE's diode connected voltage. For the analysis, we can assume that the floating gates have an equal amount of



(a)



(b)

Figure 16: MITE implementation of a 2^{nd}-order translinear loop. (a) Schematic of a 2^{nd}-order translinear loop. (b) Simulation results of the translinear loop. The multiplication coefficients were chosen to be $\frac{1}{10}$, $\frac{1}{4}$, $\frac{1}{2}$, 1, 2, 4, and 10.

charge on them and that both of the MITE's input capacitors are equal ($\kappa w_1 = \kappa w_2 \equiv w$). Under these assumptions (with $V_{ref} \equiv V_0$), the equations are

$$V_i = \frac{U_T}{w} \log \frac{I_i}{I_0} - V_{i-1},\ i = 1, 2, 3 \tag{6}$$

$$V_3 + V_0 = \frac{U_T}{w} \log \frac{I_4}{I_0}. \tag{7}$$

Substituting Equation 6 into Equation 7, gives

$$log\frac{I_1}{I_0} + log\frac{I_3}{I_0} = log\frac{I_2}{I_0} + log\frac{I_4}{I_0}, \tag{8}$$

which can be written as

$$I_1 I_3 = I_2 I_4. \tag{9}$$

This circuit is most often used as a multiplier with

$$I_{out} = \frac{I_a I_b}{I_c}. \tag{10}$$

Simulation results of the translinear loop are shown in Figure 16b. Data was taken as I_a was swept and the coefficient I_b/I_c was held constant. For higher coefficients, the trace is not completely straight because the MITEs leave the subthreshold region due to the higher current levels. The dynamic range (DR) for such a system follows the discussion given in [31].

3.1.2 Building Block: Filters

Log-domain filters were included in this system as higher level blocks because they are a building block of almost every dynamic system and are commonly utilized in synthesis procedures. The synthesis of the circuit, found in [26], is similar to the synthesis of the loop, but first the constraint equations are needed. The differential equation for a first-order low-pass filter is

$$\tau \frac{dI_y}{dt} + I_y = I_x, \tag{11}$$

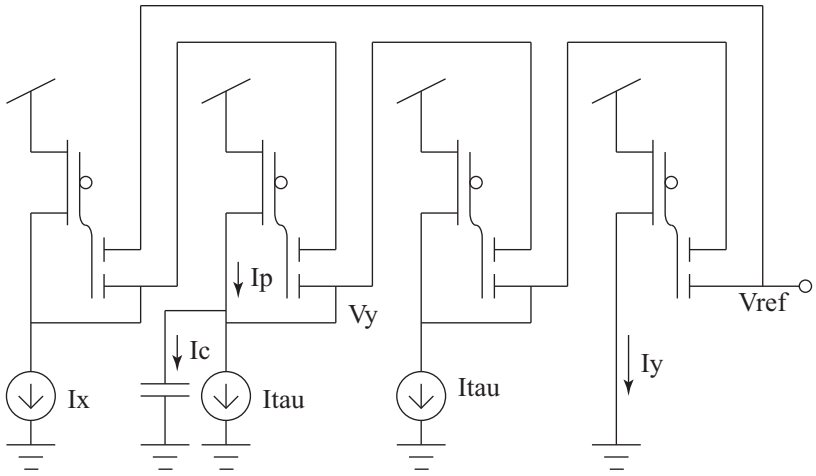


Figure 17: MITE implementation of a 1^{st}-order low-pass log-domain filter. The I_τ bias current connected to the capacitor is used to set the corner frequency of the filter. The second bias current is set to I_τ in order to maintain unity gain.

where I_x is the input current, I_y is the output current, and τ is the time constant of the filter. The chain rule can be applied to the derivative of the current giving

$$\tau \frac{\partial I_y}{\partial V_y} \frac{dV_y}{dt} + I_y = I_x, \tag{12}$$

where V_y is the log compressed voltage associated with I_y. Taking the derivative of the current through the 2-input MITE with respect to a single controlling voltage results in

$$-\tau \frac{w I_y}{U_T} \frac{dV_y}{dt} + I_y = I_x, \tag{13}$$

where w is the weight of the controlling voltage V_y. Noting that CdV_y/dt is a capacitive current (I_c) and $\tau w/U_T C$ can be written as a reciprocal of a bias current (I_τ^{-1}) we can arrange Equation 13 as

$$I_\tau - I_c = \frac{I_x I_\tau}{I_y}. \tag{14}$$

This equation is implemented by the circuit in Figure 17, where the right hand side is the same as the loop derived in Equation 10, and the left hand side is simply the KCL of I_p. In addition, a gain term can be added to the transfer function by multiplying the second I_τ, the bias current for the MITE without the capacitor on its drain, by the coefficient desired.

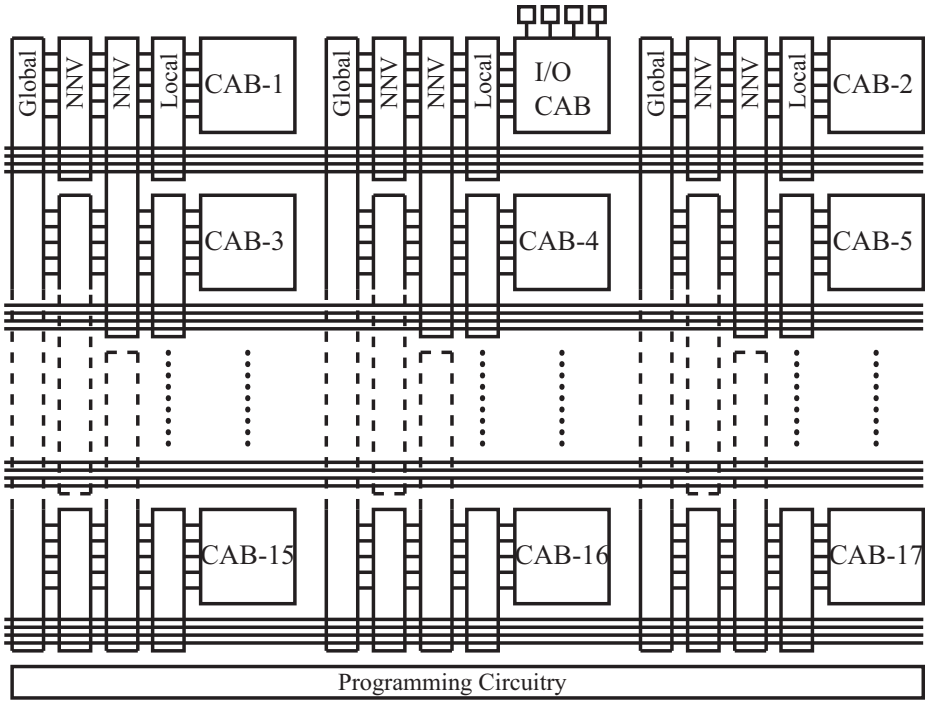


Figure 18: Architecture of the MITE FPAA. The FPAA consists of 17 MITE CABs and a single IO CAB. The vertical routing between CABs is organized into local, nearest-neighbor vertical (NNV), and global. The horizontal routing is only global.

3.2 Reconfigurable Architecture

The MFPAA utilizes the base architecture developed for the general RASP 2.8 line of FPAAs [19]. This results in a system that is a vast advancement over the Reconfigurable Analog Array of MITEs [24] by using a more computationally efficient MITE element, incorporating a more complex routing scheme in order to reduce the parasitic capacitance of the switch matrix, and utilizing on-chip programming [22].

3.2.1 System Architecture

The architecture of the MFPAA is shown in Figure 18. The FPAA is laid out with 18 CABs in a 6×3 array, with 17 being MITE CABS and one being the I/O CAB. The RASP infrastructure incorporates a cross-bar switch matrix for connecting the elements to one another. The connection between the horizontal and vertical lines is controlled by a single floating-gate switch, which allows it to store its own value without a separate memory.

Within each CAB, the vertical routing is organized into 10 global, 20 nearest neighbor

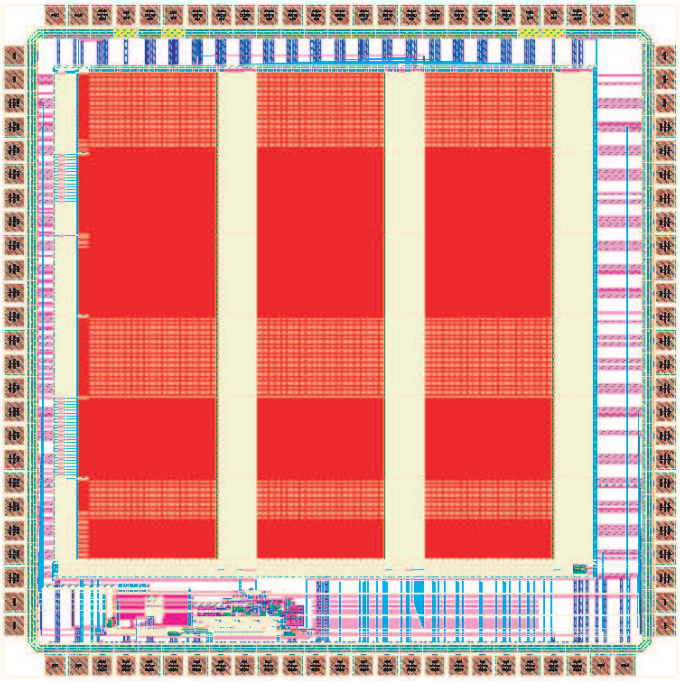


Figure 19: Layout of the MITE FPAA. The FPAA was fabricated in a 350 nm standard CMOS process on a 3 mm × 3 mm die.

(10 up, 10 down) and 10 local lines. The shorter lines are used whenever possible to reduce parasitic line capacitance. Each CAB also has 10 global horizontal lines. At the lower end of the IC is the on-chip programming structure, which selects and programs all necessary floating-gate switches and MITEs. The layout of the MFPAA is shown in Figure 19, which was fabricated in 350 nm standard CMOS with a V_{DD} of 2.4 V.

3.2.2 The MITE CAB

The most significant advancements in the architecture of the MFPAA are within the MITE CAB. To improve the density of computation elements to switch elements, single MITEs must be replaced with computational blocks with less reconfigurability. The computation element in Figure 20 was chosen to avoid losing flexibility by trying to maximize the number of equations the element could implement while minimizing the reconfigurability needed. This structure is similar to the one analyzed in Section 3.1.1, with V_{ref} taken from V_1. Two of these elements, called MITE computational elements (MCEs), are contained in

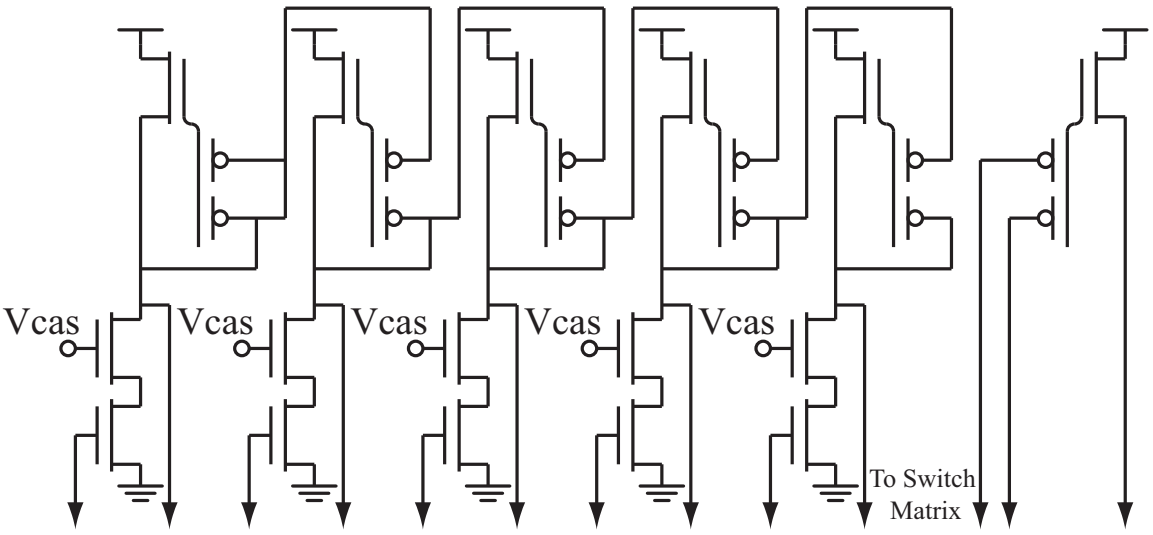


Figure 20: Basic MITE computation element of the MITE FPAA. The computation element consists of 5 input MITEs in a translinear loop configuration and 1 output MITE. The gates of the output MITE are sent into the switch matrix where they are connected to any of the input MITE gate voltages.

each CAB.

The CAB also includes a first-order log-domain filter, shown in Figure 28. This is the same structure discussed in Section 3.1.2, with V_{ref} taken from V_{tau}. Again, this was done to increase the density of the computational elements without losing too much reconfigurability. This also lends itself to implementing previously developed synthesis procedures on the MFPAA, as dynamic functions can be implemented by combining static functions with first-order filters [27]. Both the MCE and filter were drawn with $W/L = 48\,\mu m/1.2\,\mu m$ to increase the subthreshold range.

In addition to the two MCEs and the filter, the CAB includes six bias current generators, six nFET current mirrors, and a cascode-bias generator. The bias currents are programmed with floating-gate current sources and are used for implementing coefficients and scaling currents in the input equations. The current mirrors are used for adding and subtracting as well as signal routing. The cascode-bias generator, based on Minch's design [32], creates all of the cascode biases needed.

3.2.3 The I/O CAB

The input/output (I/O) CAB is the CAB that interfaces the MITE systems to the outside world. This CAB contains input voltage-to-current (V/I) converters, output drivers, and broadcast drivers for inputs. The chip was designed with banks of 10 of each of these components. The V/I converter is necessary because MITEs are mainly current-mode elements, but it is much easier to generate voltage-mode signals off chip, via DACs or function generators. The output driver is a current mirror with a gain factor of 10 to help off-chip current meters read the subthreshold MITE currents. In this system, a current-to-voltage ADC was not incorporated because it was easy enough for to read currents with off-the-shelf instruments. This capability should be pursued in future systems to allow interfacing with a programmable processor. The broadcast driver is equivalent to half of a current mirror to log-compress the current into a gate voltage by a diode connected nFET, which can then be broadcast to many input nFET devices.

The V/I converter on the MFPAA was designed for both accuracy and speed considerations. The V/I must be able to convert currents on the order of nanoamps without sacrificing the speed of the entire system. This requires an extremely low input resistance to compensate for the large capacitance of the bonding pad. This is accomplished by using active feedback, shown in Figure 21, which is similar to the one presented in [33]. The speed of the V/I can be written as

$$f_{-3dB} = \frac{1}{2\pi \left(R \parallel R_{in}\right) C_{pad}} \approx \frac{1}{2\pi R_{in} C_{pad}}, \tag{15}$$

and its accuracy can be written as

$$Error \approx \frac{R_{in}}{R}, \tag{16}$$

where

$$R_{in} \approx \frac{1}{g_{m1}[g_{m2} r_{ds2} \left(A + 1\right) + 1]}. \tag{17}$$

The amplifiers used are simple pFET-input 5 transistor OTAs with a voltage gain of approximately $g_m r_{ds}$. V_{ref} is usually set to 0.4 V and R (off chip) is usually 10 MΩ.

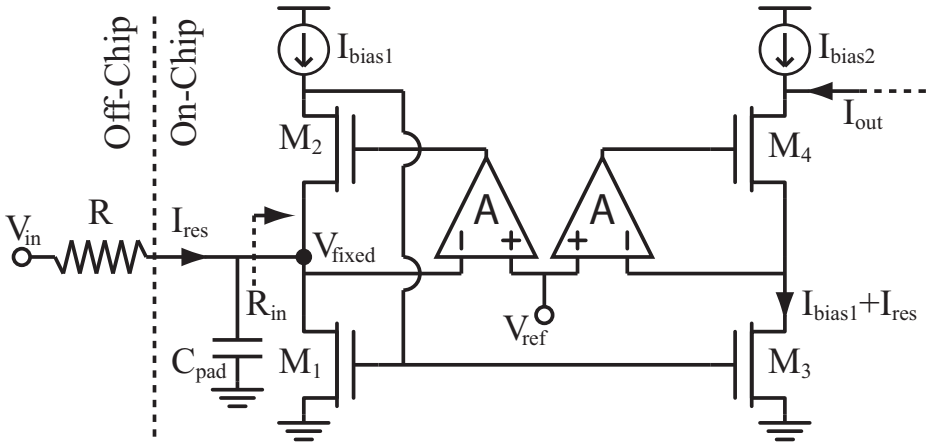


Figure 21: Voltage-to-current converter used in the MITE FPAA. The amplifier on the input side provides an extremely low input resistance allowing for high speed and good accuracy. The amplifier on the output side reduces mismatch between the input and output currents by matching the drain voltages of the mirror transistors. The bias currents are provided by floating gates.

3.3 The Design Flow

An entire software chain has been developed in order to effectively utilize the MFPAA. The collective purpose of this chain is to implement, in hardware, the equation entered by the user. The main components of the chain are network synthesis, place-and-route, visualization and programming.

3.3.1 Network Synthesis

The first step in the software chain is the synthesis of a circuit topology from the input equation. This topic was thoroughly explored in [27]. To take full advantage of this work, a set of MATLAB functions were written to parse the input equation into modules capable of being processed by the MCE. First, the expression is prepared for parsing by expanding it using MATLAB's symbolic toolbox. Since expanding the expression blindly may not lead to optimal use of components in the MFPAA, an option for the user to create sub-blocks was included. This is done by using '[' and ']' instead of parenthesis while entering the equation. Anything included in brackets is treated as its own expression and is replaced by a new variable in the original expression. Once expanded, each expression is split at the

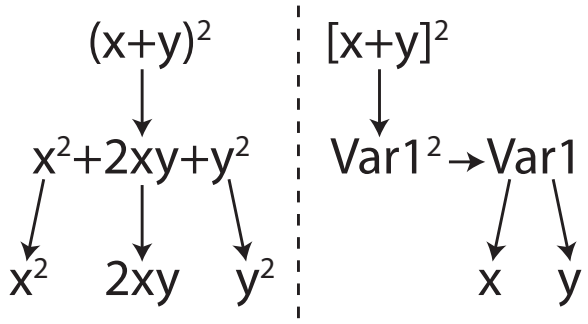


Figure 22: A representation of equation parsing for the MITE FPAA. Equations are split at addition and subtraction signs to create units that will be implemented by MCEs. The user's expression is expanded first in order to create a simple parsing tree (left). However, the user can define sub-blocks by using brackets to replace an expression with an intermediate variable [right].

'+' and '−' signs in order to break it into units containing only multiplication, division, and powers. These ideas are illustrated in Figure 22.

Now that expressions containing only multiplication, division, and powers have been obtained, a few special cases must be checked for and taken care of. One of these cases is an expression that contains fractional exponents. Since MITEs with only two gate capacitors can only implement powers with magnitudes of 1 or 2, the final expression that will be implemented can only have integer exponents. This is accomplished by raising the expression to the lowest integer power that will result in all integer exponents. While the new expression is now capable of being implemented, the output now has an exponent other than one. To correct this, the output signal will be fed back to produce an equation that results in the intended output. An example of this process is shown here:

$$I_{out} = I_1^{1/2} I_2^{1/4} I_3^{1/4} \Rightarrow I_{out}^4 = I_1^2 I_2 I_3 \Rightarrow I_{out} = \frac{I_1^2 I_2 I_3}{I_{out}^3}. \tag{18}$$

Once functions capable of being implemented with the MITEs are obtained, previous work can be leveraged to map the functions to an MCE. As described in [27], the fixed gate connections of the 5 input MITEs contained in each MCE produces a set pattern in the exponents of the expression implemented. This pattern can be altered by changing where the gates of the output MITE are connected. The possible patterns are shown in Table 3.

Table 3: MITE FPAA translinear loop exponent patterns.

Gate Connections	Input Exponent Pattern
1, 3	+1, -1, +1, 0, 0
1, 5	+1, -1, +1, -1, +1
2, 2	-1, +2, 0, 0, 0
2, 4	-1, +2, -1, +1, 0
3, 3	+1, -2, +2, 0, 0
3, 5	+1, -2, +2, -1, +1
4, 4	-1, +2, -2, +2, 0
5, 5	+1, -2, +2, -2, +2

Exponents with a magnitude greater than two must be realized by connecting the input signal to multiple MITEs. For example,

$$I_{out} = \frac{I_1^3 I_2^2}{I_3^4} = \frac{I_1 I_1^2 I_2^2}{I_3^2 I_3^2}. \tag{19}$$

In addition, expressions that cannot be implemented in a single MITE Computation Element must be broken up into multiple elements. For example,

$$I_{out} = \frac{I_1^4 I_2^3}{I_3^5 I_4} = \left[\frac{I_1^2 I_1^2 I_2}{I_3^2 I_3^2}\right] \frac{I_2^2}{I_3 I_4}. \tag{20}$$

While the MITE elements realize the multiplication, division, and powers found in the user's expression, addition and subtraction is done through the use of KCL. Intermediate expressions are summed by simply connecting the current-mode output of each MITE together, and subtracted by connecting the appropriate output of each MITE to different sides of a current mirror.

3.3.2 Place-and-Route

While place-and-route algorithms are an area of active research in both FPGAs [34] and FPAAs [35], the simple algorithm used here is meant to show the possibilities of using a translinear FPAA in simplifying the software algorithms needed. The algorithm, which uses the output of the synthesis function, can be broken into two distinct functions—placement of the components used and routing of the signals between them.

The placement function breaks the input structure into five main categories—inputs, outputs, loops, scaling currents, and mirrors. They are placed in that order by searching for the closest available elements to the I/O CAB. The current biases and mirrors are placed in the same CAB as the MCE they are operating on. The routing is then performed by picking the shortest line between elements. To reduce parasitic capacitance, the local lines have the lowest cost and the globals have the highest cost.

The last major functions in the software chain are visualization and hardware programming. While programming floating-gate transistors has been developed previously [36], functions have been added to make interfacing with an FPAA much easier. Most importantly, a GUI has been created to show the output of the synthesis and place-and-route functions. This GUI shows the FPAA and draws the switches that will be turned on and the connections between them. It also includes diagrams of the CABs so the user can easily understand what is being connected. A sample of the GUI is in Figure 23. In addition to allowing the user to easily understand how the equation is being implemented on the FPAA, the GUI also allows the user to modify the implementation if they desire.

Once the equation has been synthesized and routed, the list of switches and programmable elements are programmed on the chip. The setup that allows for this to happen includes a printed circuit board (PCB), a microcontroller, and a computer for communication [37]. Routines for selecting devices, programming switches, and programming computational elements are stored on the microprocessor and initiated by communication from the computer. The computer communicates, over either serial or USB, directly from MATLAB allowing easy interfacing between the synthesis, place-and-route, and programming code.

3.4 Results

To the test the MFPAA, a wide range of circuits were compiled onto it. First, some static functions were tested including circuits for multiplying, squaring, and cube root. Next, dynamic functions were tested. These included a low-pass filter, a high-pass filter, and a

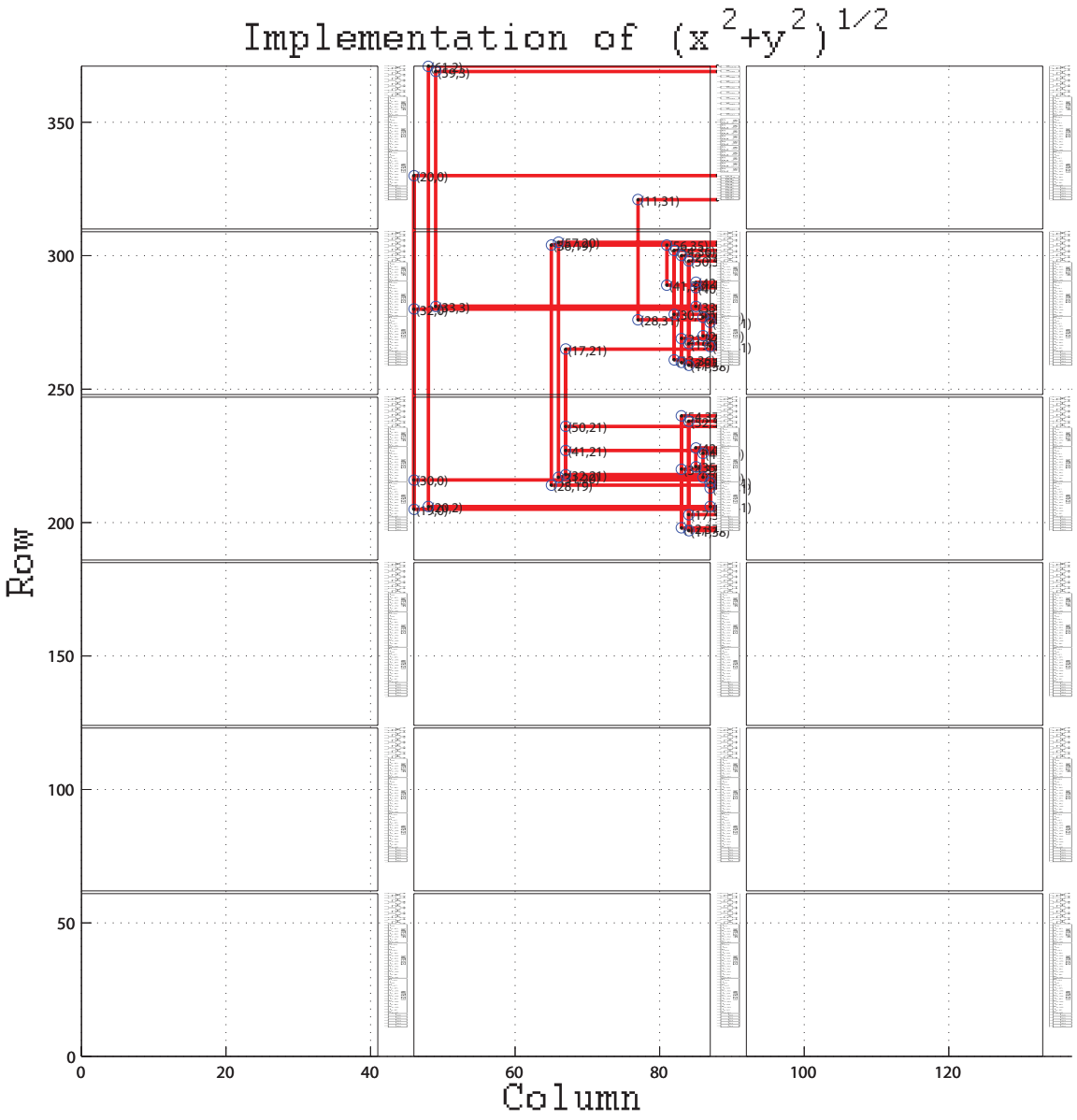


Figure 23: Visualization GUI for interfacing with the MITE FPAA. The GUI output is shown for a vector magnitude circuit.

RMS-to-DC converter. The circuits were compiled using the synthesis procedures previously discussed.

3.4.1 Static Examples

The first static example compiled onto the MFPAA implements the equation

$$I_{out} = \frac{I_a I_b}{I_c}. \tag{21}$$

To test this circuit, I_a was swept while I_b and I_c were held constant. In addition, I_b/I_c, was set to produce a variety of coefficients. The results are shown in Figure 24.

Next, a squaring circuit was compiled onto the MFPAA. The circuit uses a scaling current, I_s, that determines the value of unity in the system. This idea is illustrated in the equation

$$I_{out} = \frac{I_{in}^2}{I_s}, \tag{22}$$

which describes the system's input-output relationship. The results of the squaring circuit are shown in Figure 25. The most important feature of the output characteristic is its inaccuracy for large input to scaling current ratios. This causes currents larger than the subthreshold range to flow through the output MITE.

A cube root circuit was also compiled on the MFPAA. The circuit is shown in Figure 26. The output MITE of another MCE is used to gain access to the output current. Again, a scaling current is used set the value of unity in the system. The equation that describes the system is

$$I_{out} = I_{in}^{1/3} I_s^{2/3}. \tag{23}$$

The results of the cube root are shown in Figure 27. In contrast to the squaring circuit, the cube root results are more accurate because of its compressive nature.

3.4.2 Dynamic Examples

The first dynamic circuit compiled onto the MFPAA was a first-order low-pass filter. The filter is included as one of the CAB components on the MFPAA, shown in Figure 28. The

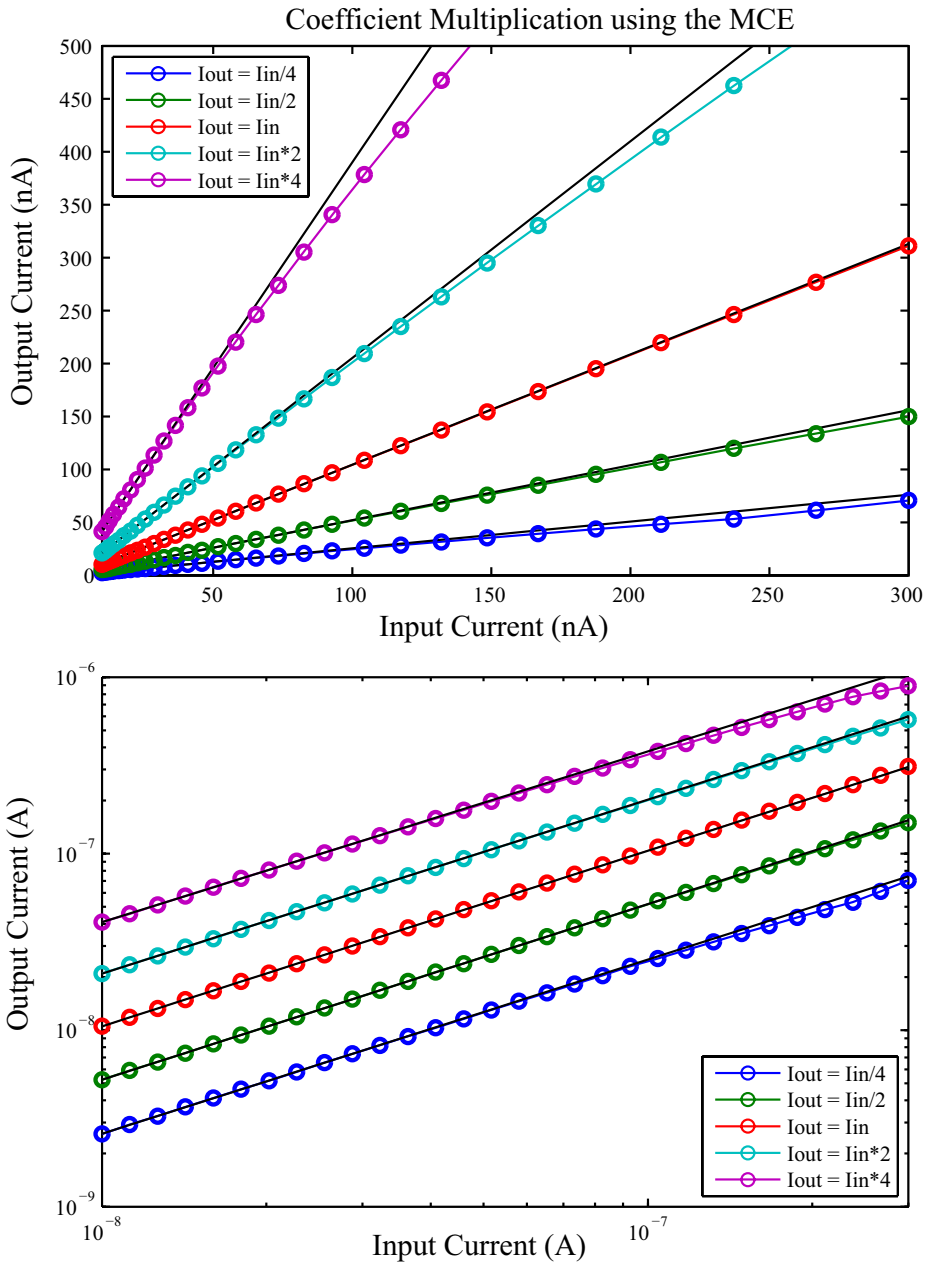


Figure 24: Results of a coefficient multiplication circuit implemented with the MITE FPAA. The results are shown in a linear plot (top) and a log plot (bottom) to show both the accuracy and the dynamic range of the computation.

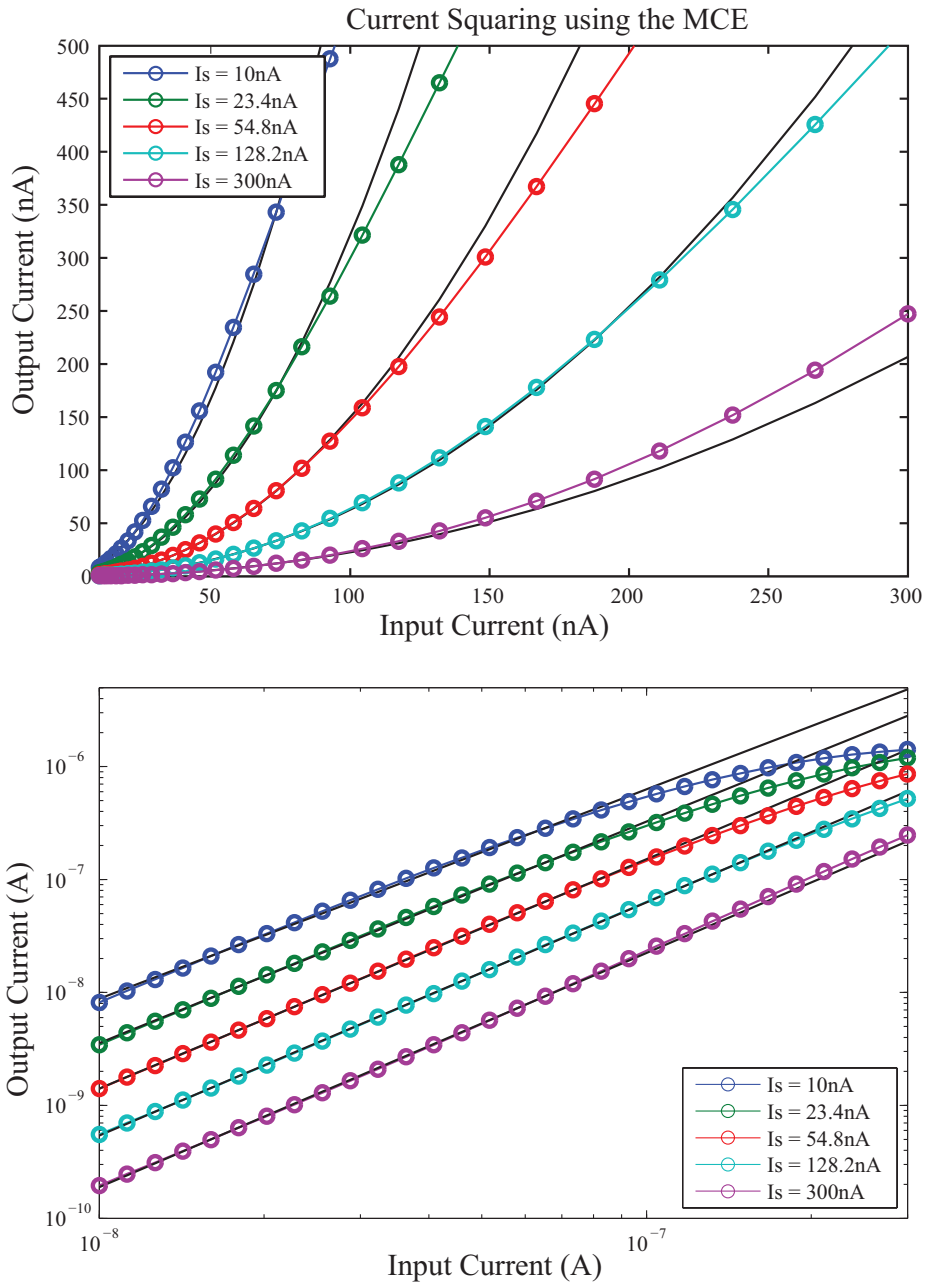


Figure 25: Results of a squaring circuit implemented with the MITE FPAA. The results are shown in a linear plot (top) and a log plot (bottom) to show both the accuracy and the dynamic range of the computation. Note that the inaccuracy at high output currents is due to devices leaving subthreshold operation.

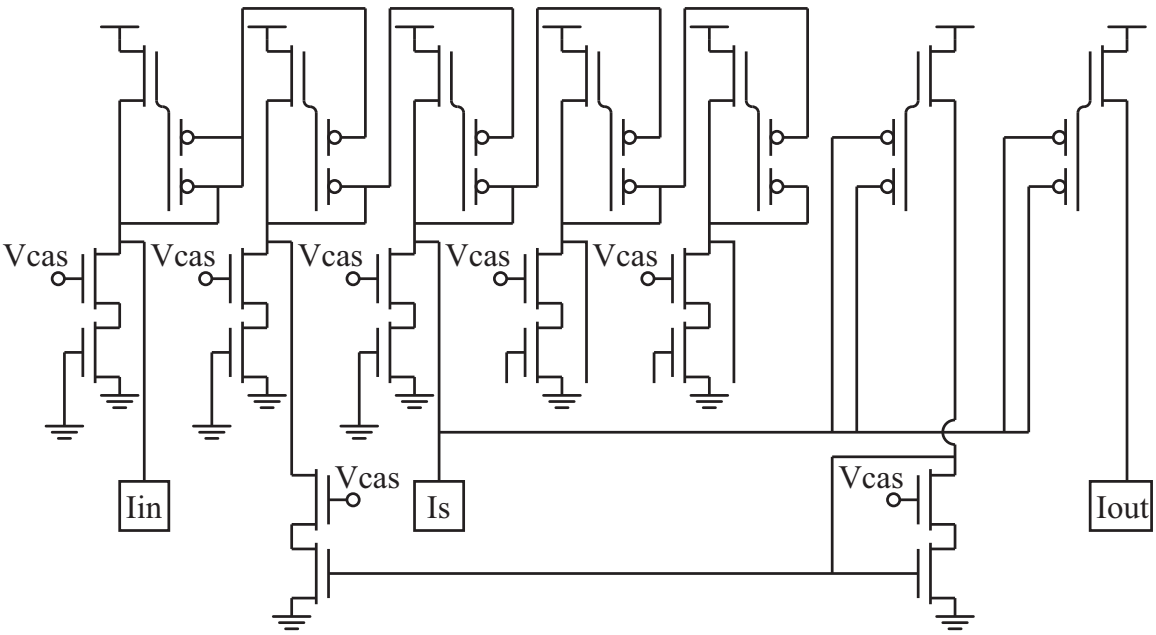


Figure 26: Circuit that implements a cube root on the MITE FPAA. A second output MITE, from the other MCE in the CAB, is used to gain access to the output current. In addition, a current mirror is used to feed back the output current to create the cube root.

filter was tested by adjusting the bias currents that set the corner frequency of the filter and measuring the transfer function. The results are shown in Figure 28b.

Next, a first-order high-pass filter was compiled onto the MFPAA. The filter was built by subtracting a low-pass filtered version of the input from the original signal. The MFPAA implementation of this design is shown in Figure 29. Again, the filter was tested by measuring the transfer function for multiple bias currents. The frequency response of the entire system is more apparent here than in the low-pass filter case. Here, the pass-band shows the effects of the mismatch due to the current mirror. The results are shown in Figure 29b.

An RMS-to-DC converter was also compiled onto the MFPAA. A combination of three static and dynamic circuits, in addition to the V/I converter, are needed in order to realize the converter. First, the input, which has been rectified by the input V/I structure, is squared. Second, it is passed through a low-pass filter to find the mean. Third, the square root of the mean is found. The MFPAA implementation of this design is shown in Figure 30. The converter was tested by varying the input amplitude of a sine wave and measuring the

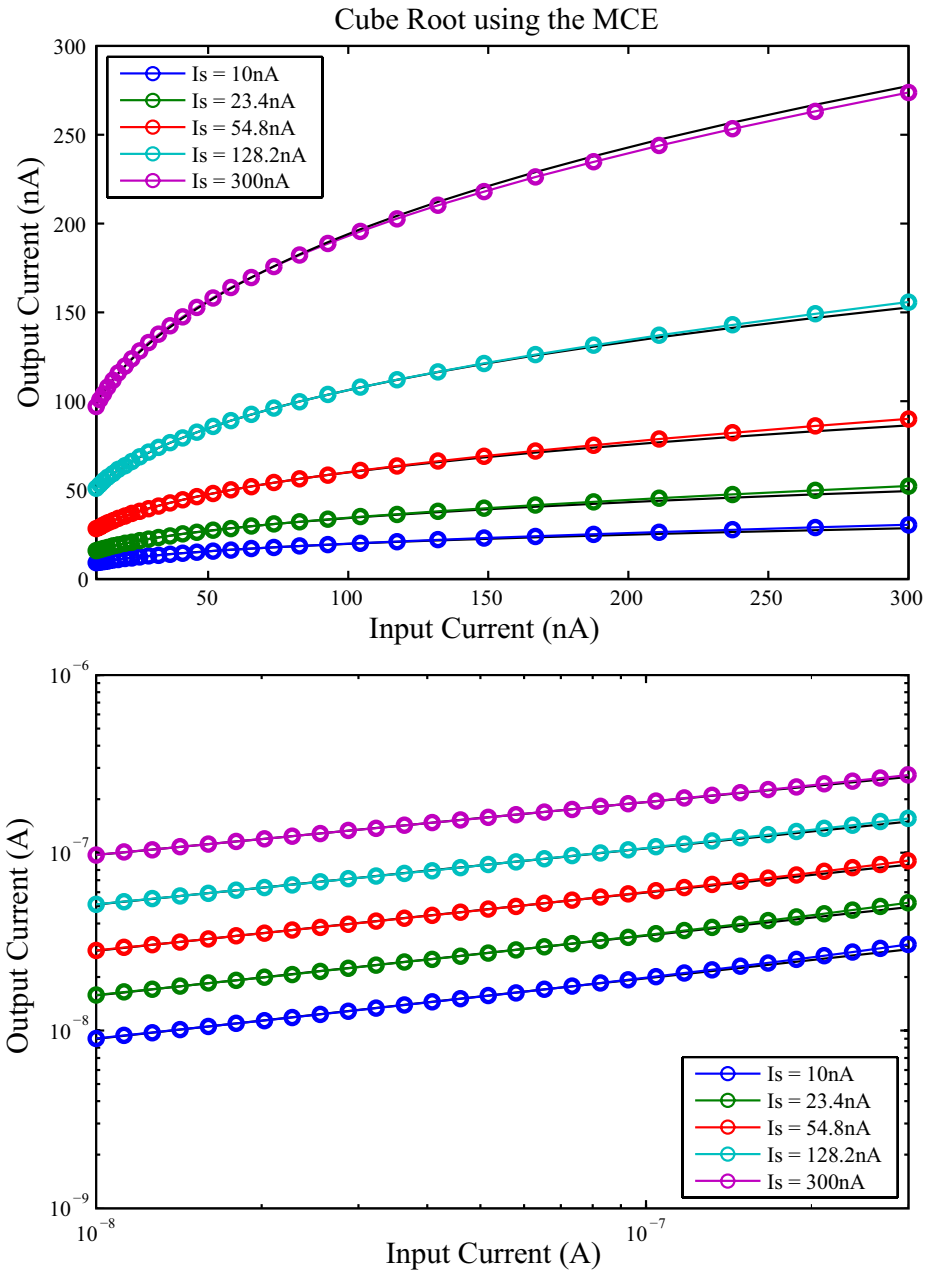
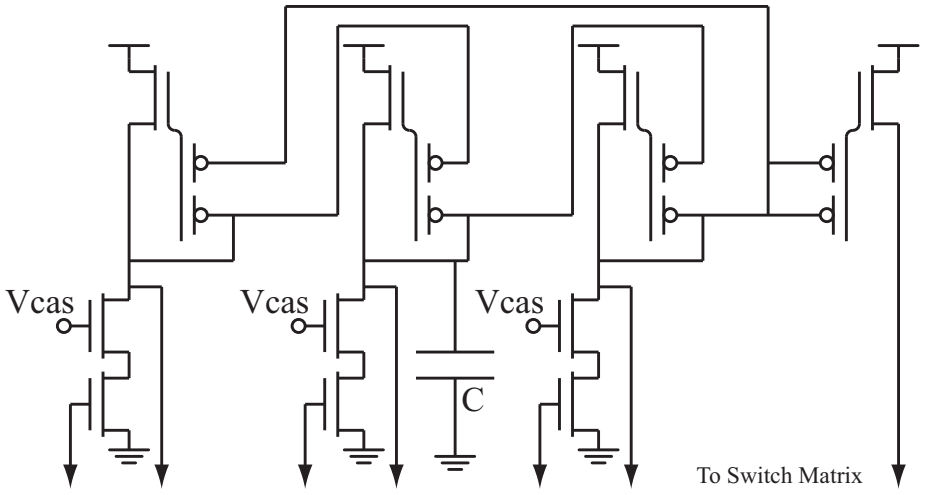
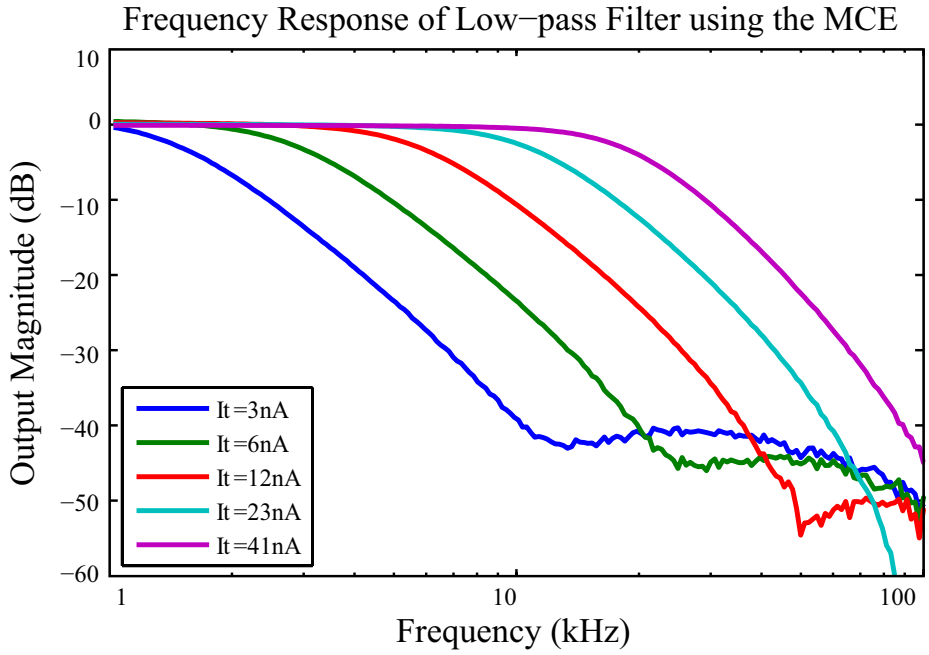


Figure 27: Results of a cube root circuit on the MITE FPAA. The results are shown in a linear plot (top) and a log plot (bottom) to show both the accuracy and the dynamic range of the computation.

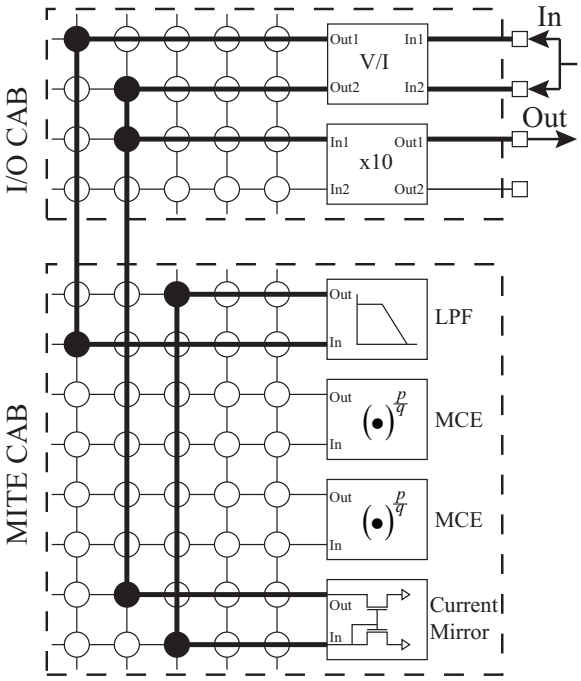


(a)

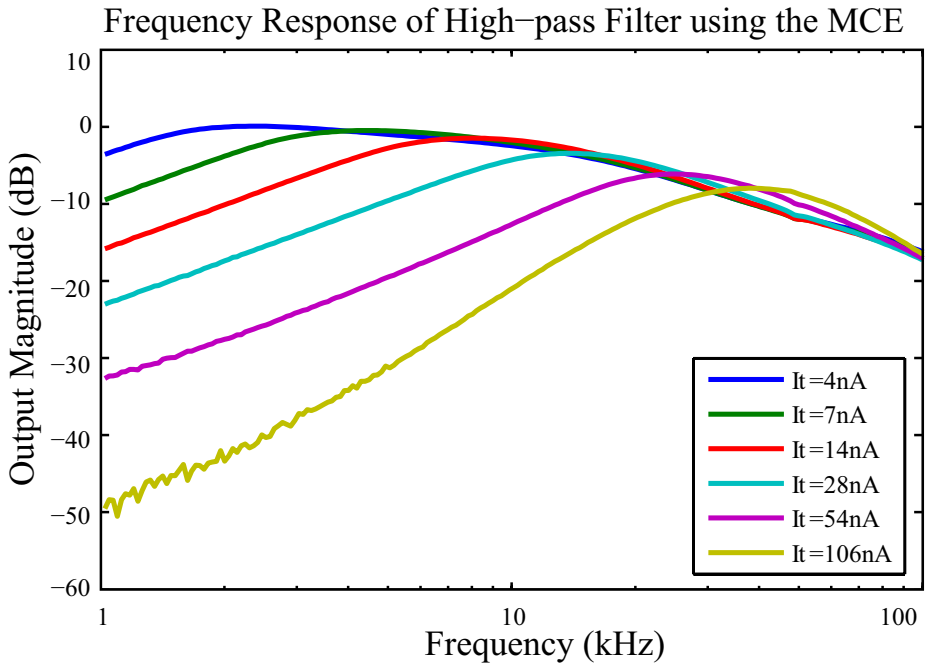


(b)

Figure 28: Log-domain filter of the MITE FPAA. (a) The MITE FPAA uses a standard first-order MITE log-domain filter in order to implement dynamic functions. (b) The transfer function of a first-order low-pass filter for various bias currents is shown. The bias currents used were logarithmically spaced between 3 nA and 41 nA. Note that the highest achievable corner frequency is 200 kHz.



(a)



(b)

Figure 29: Log-domain high-pass filter. (a) The log-domain high-pass filter can easily be complied into a single MITE CAB. To implement the high-pass filter a low-pass version of the signal is subtracted from the original signal using the current mirror. (b) The transfer function of the filter for various bias currents is shown. The bias currents used were logarithmically spaced between 4 nA and 106 nA.

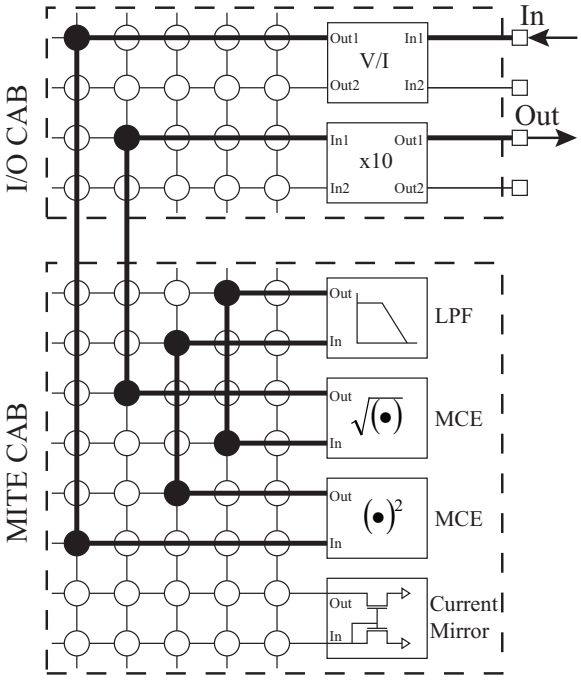
Table 4: MITE FPAA device parameters.

	This work	[38]
Process	350 nm CMOS	350 nm CMOS
Die Size	9 mm^2	.92 mm^2
Power Supply	2.4 V	not given
Number of CABs	18	25
No. of T-L elements	272	25
Largest order filter	17^{th}	4^{th}
Bandwidth	200 kHz (measured)	7 MHz (simulated)
Current Range	1 nA - 1 μA	1 nA - 10 μA
Synthesis tools	Complete	none reported

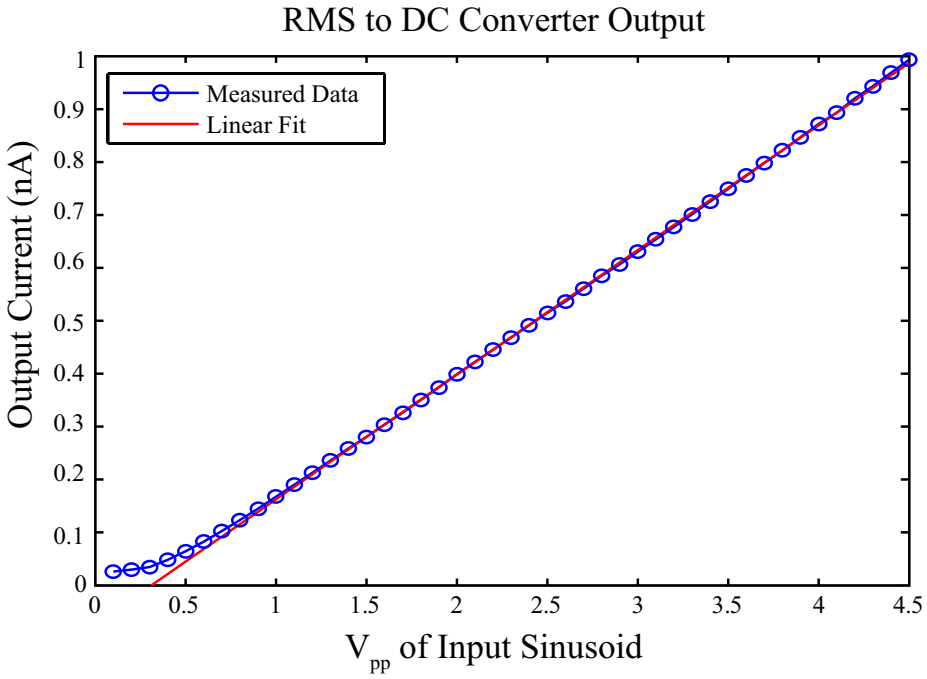
output current. The results are shown in Figure 30b.

3.5 Conclusion

In this chapter, we have discussed the design of a reconfigurable MITE system, the MFPAA. This MITE-based FPAA was designed, fabricated in 350 nm CMOS, and tested. A summary of this technology and comparison to another translinear FPAA is given in Table 4. It was designed using the floating-gate switch matrix framework of the RASP 2.8 line of FPAAs. Floating-gate switches are a natural choice for MITE systems because they can share the programming overhead that is already required to program the MITEs. Along with the MFPAA IC, we also presented an entire chain of design tools: a synthesis tool, a place-and-route tool, a routing visualization GUI, an evaluation board, and the programming system. This complete system allows the user to go from a system of equations all the way to a working hardware MITE implementation. In addition to presenting the hardware and design tools, we demonstrated several working circuits. Static systems such as multipliers and squaring circuits, as well as dynamic systems such as filters and an RMS-to-DC converter were successfully tested on the hardware system.



(a)



(b)

Figure 30: RMS-to-DC converter. (a) The RMS-to-DC converter as it is complied into a single MITE CAB. The three computational stages are: square, filter, and square-root. These three functions can each be performed by a single MCE. (b) The output characteristic of the RMS-to-DC converter. The amplitude of the input sinusoid was swept from 0.1 - 4.5 V. The frequency of the input was held at 500 Hz.

CHAPTER 4

A DIGITALLY ENHANCED FPAA: THE RASP 2.9V

After discussing the concept of core FPAA primitives, the next step of the coordinated approach to analog signal processing is to design efficient hardware *architectures*. As monolithic integration of analog and digital circuitry pervades the market, integrated circuit designers are faced with the increasingly difficult task of verifying complex mixed-signal systems. The most common approach for this task is to simulate the analog sub-system, fabricate, test the mixed-signal system, and then repeat [39]. We propose that a faster and more efficient approach is to prototype mixed-signal systems using reconfigurable hardware. In this scenario, the digital portion is compiled to a reconfigurable digital platform (such as an FPGA) and the analog portion is compiled to reconfigurable analog hardware (an FPAA). For this mixed system to work harmoniously, the analog portion needs to have the capability of being digitally controlled. In addition to simple prototyping, digitally enhanced reconfigurable analog systems are extremely powerful for embedded computing applications, providing enhanced controllability to digital elements.

This chapter presents the RASP 2.9v, the next generation of FPAA architecture [40, 41].

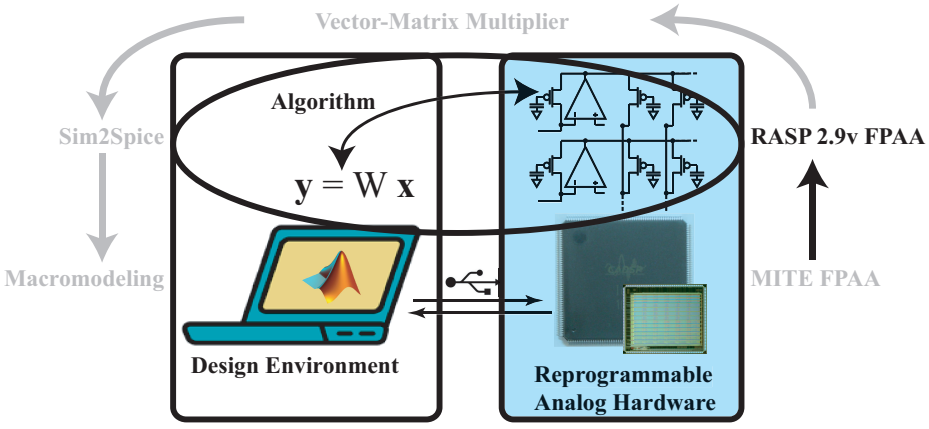


Figure 31: The coordinated approach to FPAA design: RASP 2.9v.

The RASP 2.9v includes over 76,000 programmable analog parameters and a varied toolbox of components (OTAs, FETs, caps, multipliers, T-gates) to synthesize almost any analog system (complete parameters are in Table 5). The RASP 2.9v, unlike previous FPAAs [19], includes a novel volatile switching architecture. This switching architecture allows the digital control and dynamic reconfigurability that are important in embedded systems, especially if the device is working in conjunction with a digital system. Debugging prototyped systems is quite easy with this architecture, because the volatile switches can be used to multiplex internal circuit nodes out to measurement equipment. Figure 31 shows how the RASP 2.9v FPAA fits into the coordinated-design framework.

While other FPAAs, such as the analog math co-processor in [15], have substantial digital interfacing capabilities, they tend to have far more limited application space. The co-processor is designed for ODE computation, while the hexagonal FPAA in [16] is designed to operate as a single high-dimension Gm-C filter. The higher density and greater variety of components on the RASP 2.9v permit it to reach a much wider application space.

The embedded digital control structures combine with the high-density analog arrays to produce the first dynamically reconfigurable FPAA. The digital enhancements were carefully designed to maximize their usefulness as control and storage devices, while minimizing their footprint on the overall chip. This architecture extends the high computational density of previous RASP architectures, while the digital enhancements enable higher chip utilization, effectively increasing the size of realizable systems. The digital control also provides the ability to compile banks of on-chip data converters, which increases the device's usefulness in embedded systems.

Another novel advancement of this chip is the hybrid switch matrix, which is comprised of both directly and indirectly programmed switches. The previous generation of FPAAs utilized an indirect programming scheme, which achieves very low switch resistances. However, the precision of these indirect switches has been plagued by mismatch issues inherent to the indirect scheme. To remedy this problem, direct switch elements

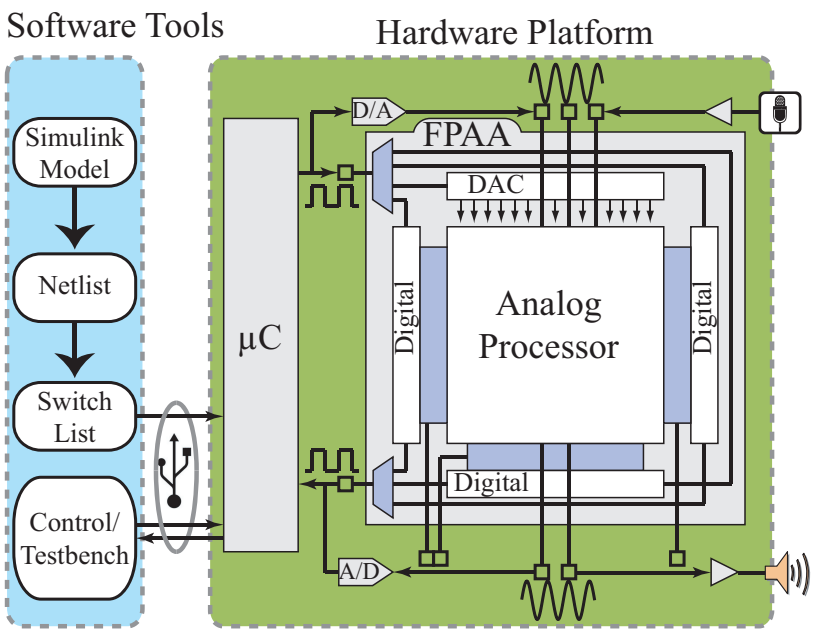
Table 5: RASP 2.9v device parameters.

Process	350 nm CMOS
V_{DD}	2.4 V
Die Size	5 mm × 5 mm
Number of CABs	18 DAC, 36 Regular, 24 VMM
Programmable parameters	> 76,000
Number of Volatile Switches	4728: 6 × 400-bit (vertical), 14 × 156-bit (horizontal), 6 × 24-bit (DAC)
Chip I/O	79 Analog, 20 Dynamic output lines 18 compilable DAC
Regular CAB Elements	132 OTA, 168 FG-OTA, 36 T-gate, 72 nFET, 72 pFET, 36 OTA buffer, 144 500 fF Cap
Programming Speed	Volatile Switch: 719 ns FG Switch: 31 ±2 ms Analog Indirect FG: 38 ±10 ms Analog Direct FG: 36 ±8 ms

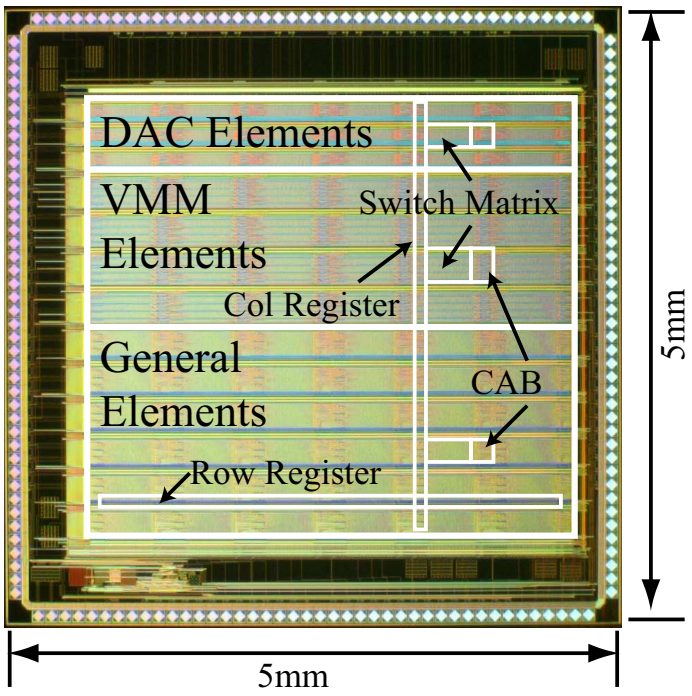
have been added to the switch matrix, which do not have the same mismatch issues as indirect devices. The introduction of the hybrid switch matrix eliminates the burden of characterizing and storing the offset of each switch for every chip. This option allows for high programming accuracy in a single programming pass, which is extremely valuable in high-volume production. These new features—dynamic control, on-chip DACs, and high precision switches—uniquely position this chip as the only platform reported that is capable of large-scale embedded digitally enhanced analog processing.

4.1 Processing Elements

Figure 32 illustrates the FPAA system-level architecture, with the analog processor at its core. This processor contains thousands of analog components that can be configured and routed to implement many analog signal processing systems. The RASP 2.9v features a novel volatile switching scheme that allows the user to scan thousands of outputs, assert a signal onto any internal node via 20 dedicated I/O pins, and store and retrieve digital



(a)



(b)

Figure 32: Architecture and layout of the RASP 2.9v FPAA. (a) The system-level diagram shows the analog core and surrounding digital control and interfacing. The analog processor communicates directly with the microcontroller via an SPI interface. A complete software tool chain is available for analog synthesis in Simulink and connects to the hardware platform with USB. (b) The RASP 2.9v IC was fabricated in 350 nm CMOS and consumes 25 mm^2 of area.

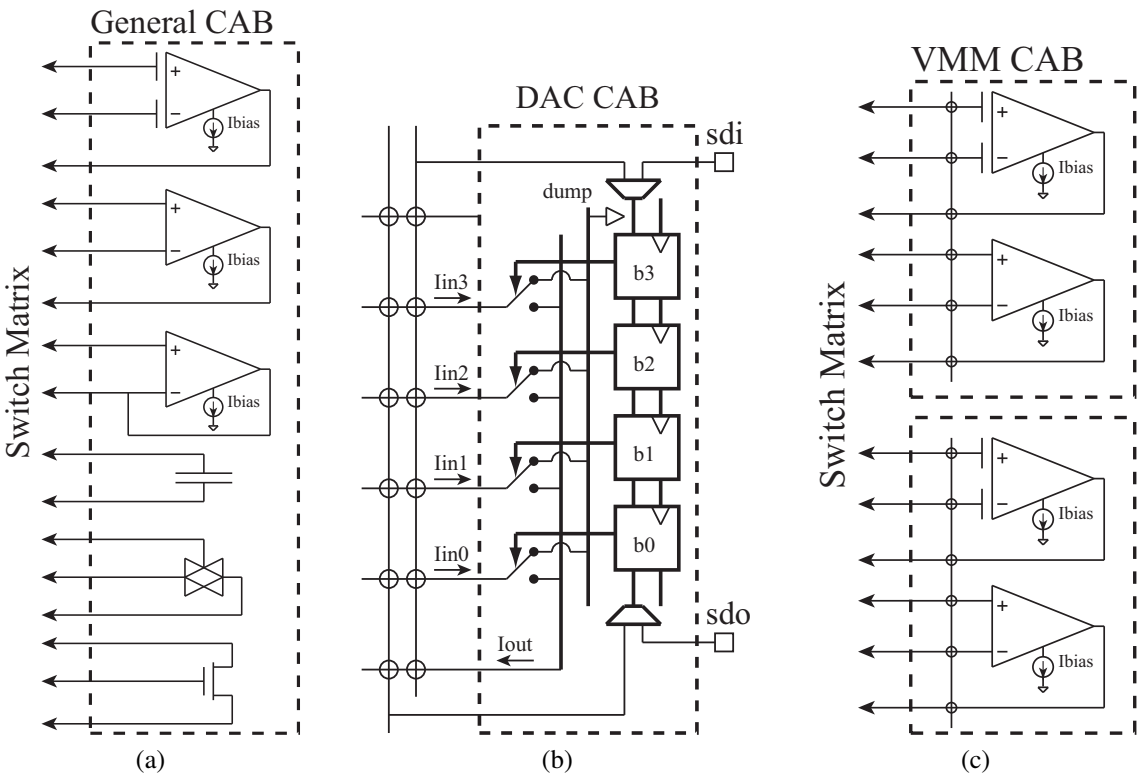


Figure 33: Structure of the RASP 2.9v CABs. (a) The General CAB consists of common analog elements: OTAs, MOSFETs, capacitors, and transmission-gates. (b) The DAC CAB contains an 8-bit register that toggles a bank of switches, allowing for multiple DAC topologies to be compiled. (c) The VMM CAB contains both regular and floating-gate-input OTAs, which are commonly used as the front end to the VMM: the FG-OTA for V/I conversion, and the OTA for the active feedback of the sense transistor.

values.

The analog processing core is composed of analog primitives that are arranged in computational analog blocks (CABs). The various CABs are shown in Figure 33. The IC contains 78 CABs: 36 for general purpose analog computation, 18 designed for compiling current-mode digital-to-analog converters (DACs), and 24 optimized for performing vector-matrix multiplication (VMM) operations.

4.1.1 The General Analog CAB

To accommodate the widest possible application space, the largest chip real estate was given to the general processing CAB. Each general CAB contains 4 operational transconductance amplifiers (OTAs), 4 FETs (50/50 split of n/p-type), 1 transmission-gate switch

(T-gate), and 4 500 fF capacitors.

Of the four OTAs, three are fully port accessible and one is connected in unity-gain feedback. Two of the port-accessible OTAs have floating-gate input stages with a capacitive divider attenuation of 1:9, which increases the linear input range by a factor of 9 (see Figure 34). The floating-gate inputs can be programmed to compensate for—or introduce—a fixed offset. The floating gates are programmed by modifying the charge on the gate with hot electron injection and Fowler-Nordheim tunneling. On-chip circuitry can measure the floating gate's state and apply the necessary terminal voltages to modify the charge to the desired level.

The OTAs all utilize a wide-linear-range 9-transistor topology with pFET inputs. A floating-gate pFET transistor sets the tail current of the OTAs, which is programmable from 100 pA to 10 μA. The transconductance of each OTA can be accurately by the bias current. Control over the transconductance of the device is useful for programming systems such as analog filters or voltage-controlled current sources. The power consumed by each OTA is calculated as $2I_{bias}V_{DD}$, where the factor of two is a result of the I_{bias} flowing in both branches.

The OTA can also be used as an amplifier with voltage gain. The voltage gain is independent of bias current, so we will choose current values only large enough to drive the amplifier's load. For power analysis of systems with voltage-gain OTAs and high impedance loads, we will choose I_{bias} of 100 nA, which results in 480 nW of static power.

4.1.2 The DAC CAB

One improvement of the RASP 2.9v over previous FPAAs is the incorporation of dedicated current DAC sections. The DAC CAB is composed of digitally controlled switches that are connected to the switch matrix, allowing users to compile binary-weighted current-mode DACs.

Each DAC CAB contains one 8-bit register, with three CABs down a column connected in series. Thus, the DAC section could also be configured as six 24-bit registers (there are

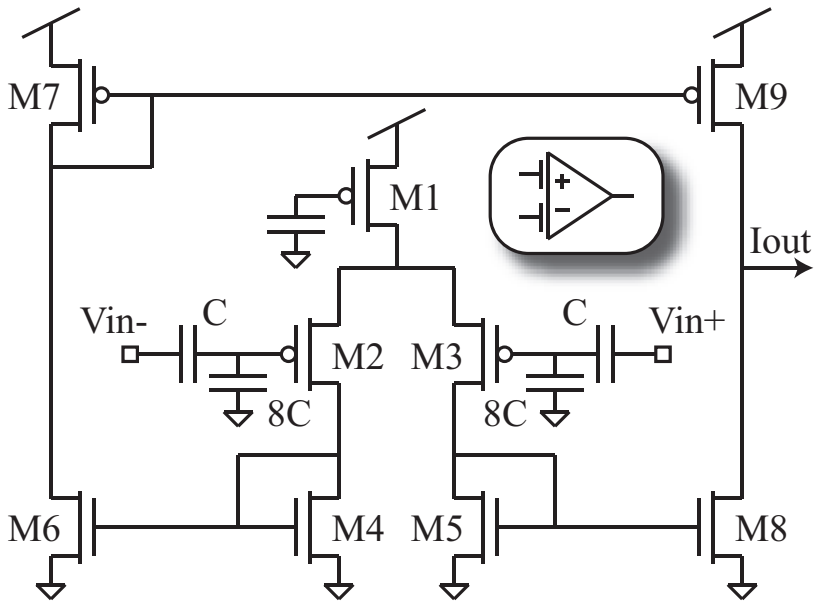


Figure 34: Schematic of the operational transconductance amplifier (OTA). Both the regular and FG-input OTAs use a 9-transistor structure. The bias current is set with an FG pFET and can be programmed from 100 nA to 30 μA. The FG-input has an attenuation factor of 1:9 on the input stage for wider input linear range (with lower gain). The FG input elements can also be programmed to remove any input offset.

six CAB columns). These registers can be configured either to take the serial-data input from the off-chip source or from the switch matrix. The same configuration setting will determine if the output goes off chip or into the switch matrix. This capability allows the digital registers to double as storage for system data.

4.1.3 The VMM CAB

Vector-matrix multiplication is an extremely efficient operation when performed in the analog domain [42, 43]. Because, this computation is such a "killer app" for modern analog computation, special consideration was taken to facilitate their large-scale design in the RASP 2.9v.

Each of the 24 VMM CABs contains four pairs of OTAs. In each pair, one OTA is for the current-scaling active current mirror, and the other is a floating-gate input OTA for V/I or I/V conversion. Between each pair of devices is a short vertical line to allow repeated connections to the same column address, drastically increasing routing efficiency. These

particular floor plan choices increase the dimension of synthesizable matrix multipliers, while all of the OTAs can still be used for any purpose, resulting in no loss of flexibility.

4.2 Routing and Analog Switches

The RASP 2.9v FPAA is arranged in 13 rows and 6 columns of CABs. A full cross-bar switch matrix (SM) is incorporated to interconnect the CAB elements to each other. Nonvolatile switches are located at the intersection of each row and column line.

4.2.1 Routing

Figure 35 shows the routing architecture. The cross-bar matrix contains a mixture of global, local, and power routing. Each section of SM is composed of 3 global power lines, 11 vertical global lines, and 14 vertical local lines. The global lines span all of the CAB rows, where the local lines can be connected to each top or bottom neighbor with a bridge switch. The locals can be reconfigured into global lines, at the cost of higher parasitic resistance and capacitance. This combination of two line types allows for greater versatility.

Analysis from a similar (but smaller) FPAA structure extracted a parasitic capacitance of 1.6 pF for global vertical lines, 1.5 pF for global horizontal lines, and 220 fF for vertical local lines [19]. While the local lines have approximately the same length in the RASP 2.9v, the global vertical and global horizontal lines are 63% and 50% longer, respectively. We can use these values to extrapolate respective capacitances of 2.6 pF and 2.3 pF. Good estimates of these capacitances allow designers to take routing into account when designing circuits.

The power lines support a chip-wide global voltage supply (V_{DD}), ground, and reference voltage $\left(V_{ref}\right)$. The inclusion of a global V_{ref} is novel to this chip and was the direct result of previous FPAA design experience. Many analog signal processing systems use a common mid-rail voltage that feeds multiple elements. Including a global V_{ref} drastically reduces routing complexity. The V_{ref} line is pinned out where is can be driven off-chip to any voltage, or it can be left open at the pin and driven by an on-chip source.

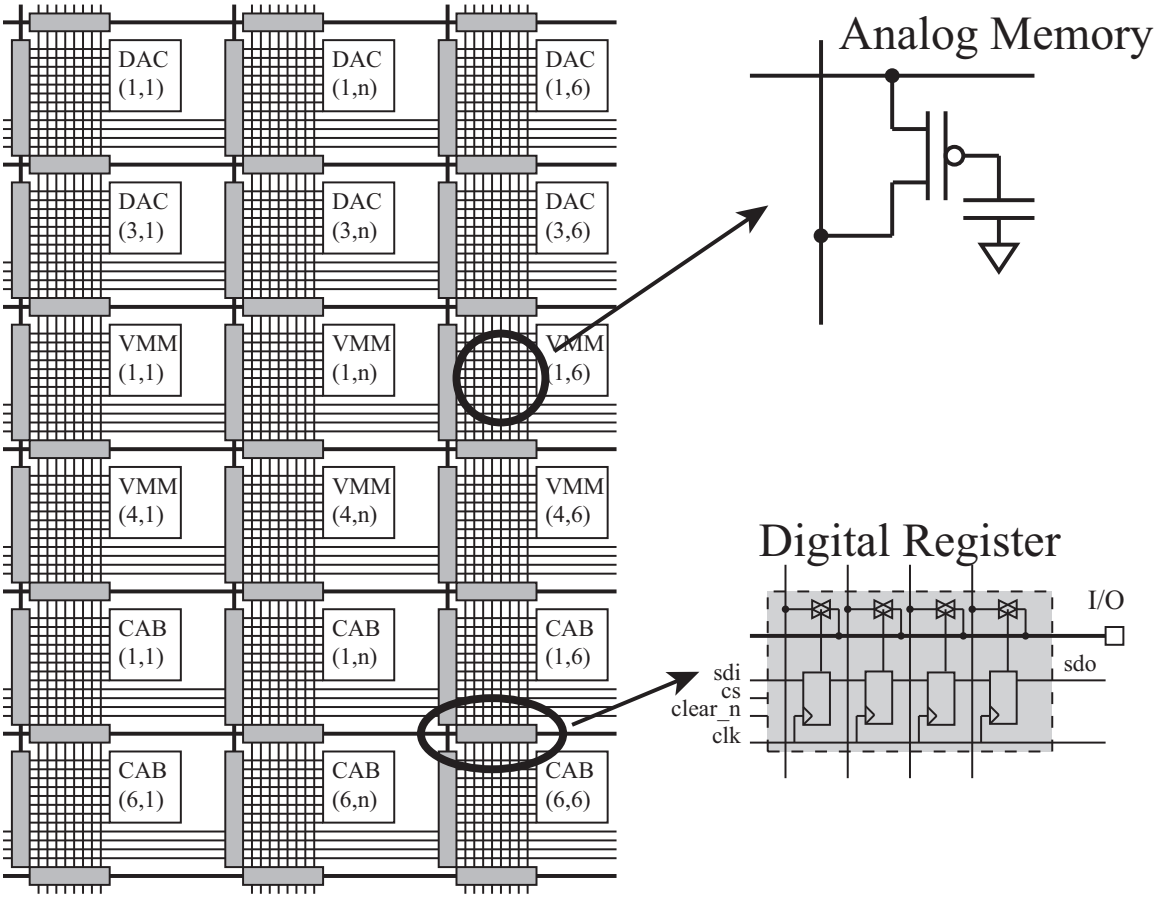


Figure 35: The CABs are arranged in 13 rows and 6 columns. There are 36 Regular CABs, 24 VMM CABs, and 18 DAC CABs. The routing is a full cross-bar switch matrix with floating-gate switches intersecting each row and column. This topology allows for great functional density, as each floating gate stores its own memory and acts as either a switch or an analog computation device. The volatile switches are controlled by digital shift registers that span all of the columns (156-bit each) and rows (400-bit each).

4.2.2 Non-Volatile Switches

The cross-bar SM is composed of programmable floating-gate (FG) transistor switches, of which there are a total of 76,000. Each element can be programmed using hot electron injection or Fowler-Nordheim tunneling. The FGs double as reconfigurable switches and nonvolatile memory that store their own conductance. Since the FGs are analog elements, they can be programmed to intermediate states, allowing their use for dense analog computation.

The indirect switch programming scheme, shown in Figure 36a, is used for general routing situations. This structure allows us to measure the programmed current in the indirect device (M2)—which shares a gate with the in-circuit device (M1)—while removing selection circuitry (M3) from the signal path, minimizing parasitic resistances. However, the cost of this indirect system is the inherent mismatch between the device that is measured (M2) and the device that is used in the circuit (M1). This effect is not a problem for fully programmed switches, but can cause a loss in precision when the FGs are used for computation. This issue can be compensated for by characterizing and storing the offset coefficients of each device.

Although the mismatch of the indirect switches can be characterized and compensated for, that is not always a practical solution. Characterization involves testing each element, a process that involves time (and thus money) for large-volume production. Compensation would also involve maintaining a memory of the offset for each element. With this in mind, direct switches, shown in Figure 36b, were added to create a novel hybrid switch matrix. The direct programming method uses one FG as both the programmed device and the in-circuit device (M4), so the mismatch between the program-time and run-time devices for a particular SM address has been eliminated. Figure 36c illustrates the increased precision in a single programming pass. This method is an important improvement to the analog processor, which relies heavily on the use of floating-gate switch elements for precise computation. The direct device frees us from the cumbersome task of having to

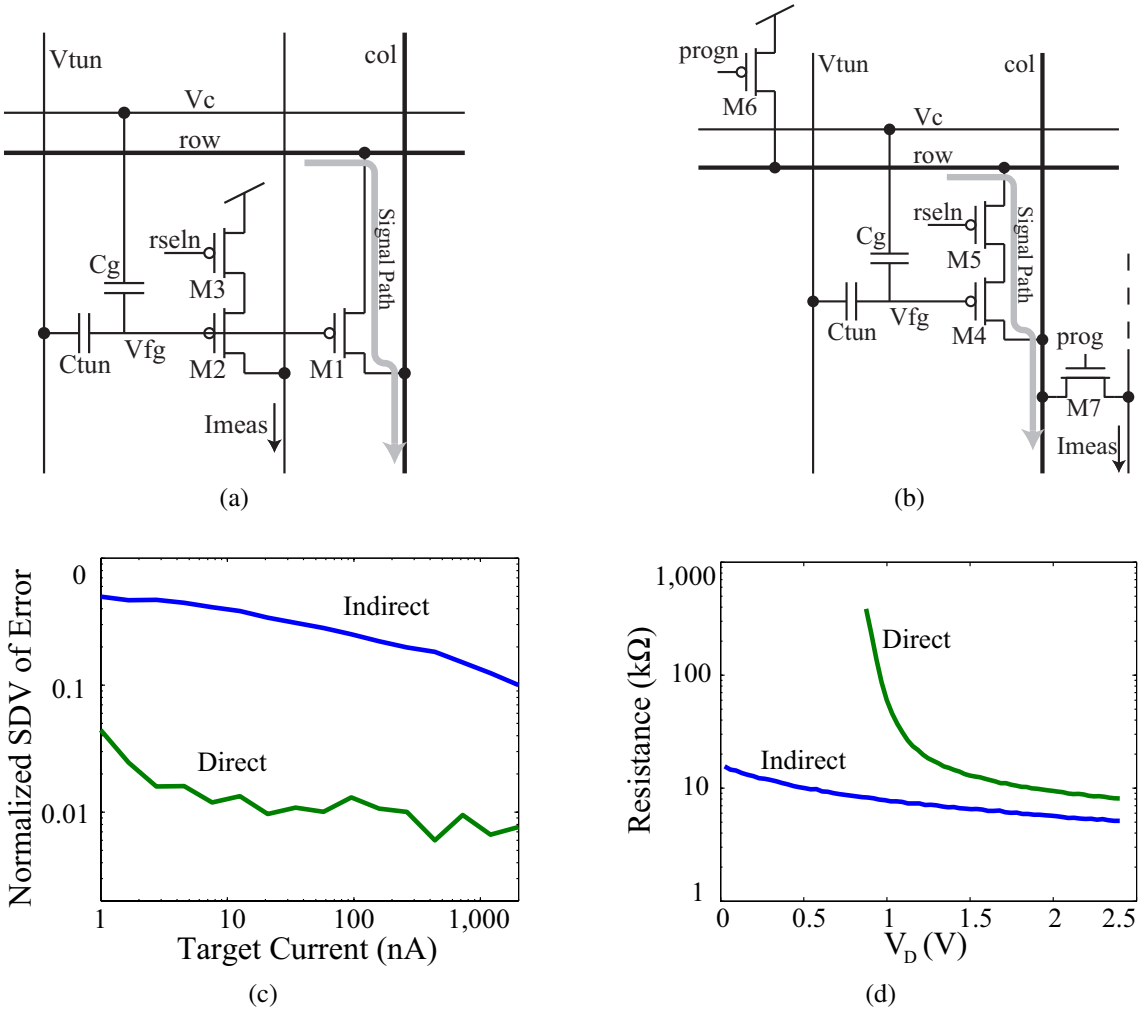


Figure 36: The routing structure contains two variations of floating-gate switches: indirect and direct. (a) The indirectly programmed floating-gate switch provides a very good pass element because there is no selection circuitry in the signal path. (b) The directly programmed floating-gate switch was included for improved precision. However, it is not an optimal all-purpose switch because selection circuitry is added to the signal path for programming isolation. (c) Comparison of the two types of floating-gate switches shows that the direct switch has a much lower first-pass programming error. (d) Each switch shows an on resistance of about 10 kΩ; the direct switch's resistance, however, rises sharply at low voltages because of the pFET in the signal path.

map all of the coefficients of the chip. To keep the same form factor of the switch cell, a single pFET (M5) is inserted above the FG-FET (M4) for programming isolation. Because the pFET has low conductance at low voltages, the direct scheme makes a poor all-purpose switch. A comparison of the two switches' ON resistance is shown in Figure 36d.

An on-chip programmer, based on the design in [22], is used to program all of the non-volatile switches and other programmable elements. For the direct-programmed switch, the routing column measures drain current in program mode. This approach means that all of the switches down a column must be of the same type: direct or indirect. The global verticals are therefore subdivided into 3 indirect lines and 8 direct lines, and the local verticals are subdivided into 6 indirect and 8 direct. The switches are skewed towards the direct configuration because it is very valuable for precise current sources and multiplier weights.

4.2.3 Volatile Switches

The incorporation of volatile switches on the RASP 2.9v marks a vast improvement in digital interfacing compared to earlier FPAAs. The volatile switches are composed of shift registers that control the selection of T-gate switches, referred to here as registered switches. The T-gates can connect routing lines to a common I/O bus. Registered switches were inserted across every CAB row and down every CAB column, for a total of 20 registers. This new tool allows us to probe any given circuit node in run mode. Figure 37 shows the volatile switches used to dynamically test and measure the FG switch resistance.

The registers are loaded serially with the SDI line (serial data in), can be read on a common SDO line (serial data out), and clocked with a dedicated SCLK (serial clock). This SPI protocol lets the FPAA interface with most modern microcontrollers. The shift registers are buffered with a data latch that loads on a global chip select (CS). This data buffer allows us to shift configurations while maintaining the previous switch control. Communication with each register is multiplexed using a 5-bit address. All of the registers are on a global clear.

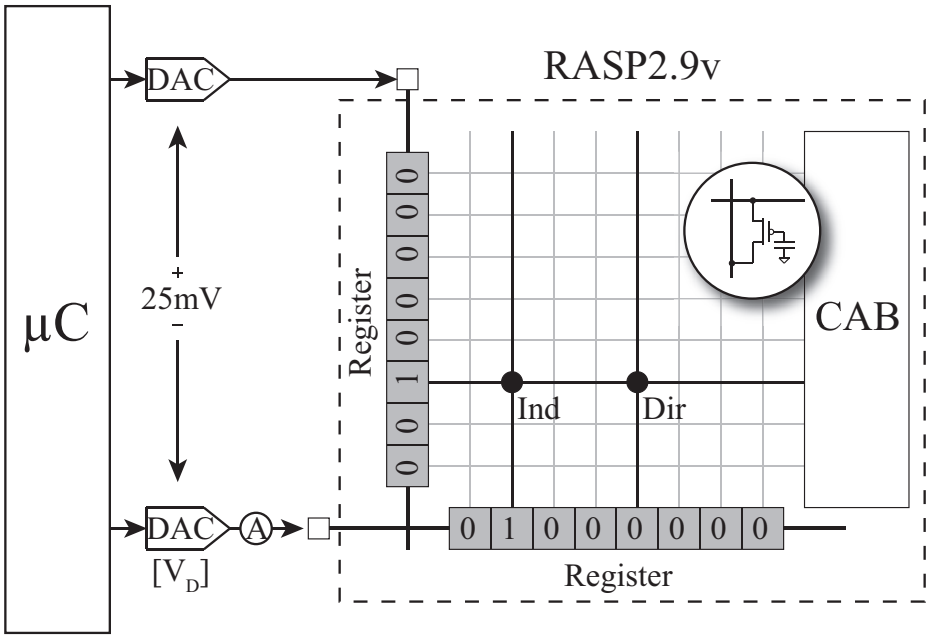


Figure 37: The volatile switches can be leveraged to dynamically select which FG switch to read from in a measurement test.

Some of the registers (the ones in the DAC CAB) can be configured to take SDI signals from on-chip sources, as illustrated in Figure 38. The timing diagram in the figure shows that when the select line is disabled, the register is filled from the default external source (typically a microcontroller). When the select line is enabled, the register is connected to a line in the SM so that it can be loaded with on-chip data. This option is useful when storing a digital pulse train that is generated on chip (for instance when synthesizing a sigma-delta converter).

The registered switches come at the cost of pin count and nonvolatile switch density. The whole structure requires 29 dedicated pins (4 SPI, 5 address, 20 I/O), reducing the general analog I/O to 79 pins (based on a 200-pin QFP). This cost is acceptable because the registers' 20 I/O greatly expands the chip's effective I/O because they can serially reach every circuit net when operated as a scanner multiplexer. The other cost is in density; each bit of the register consumes the area of 8 FG switches. This approach reduces the available analog routing lines, as well as 8 local horizontal outputs per CAB. The great improvement in overall routing versatility by the run-mode volatile switches makes this an acceptable

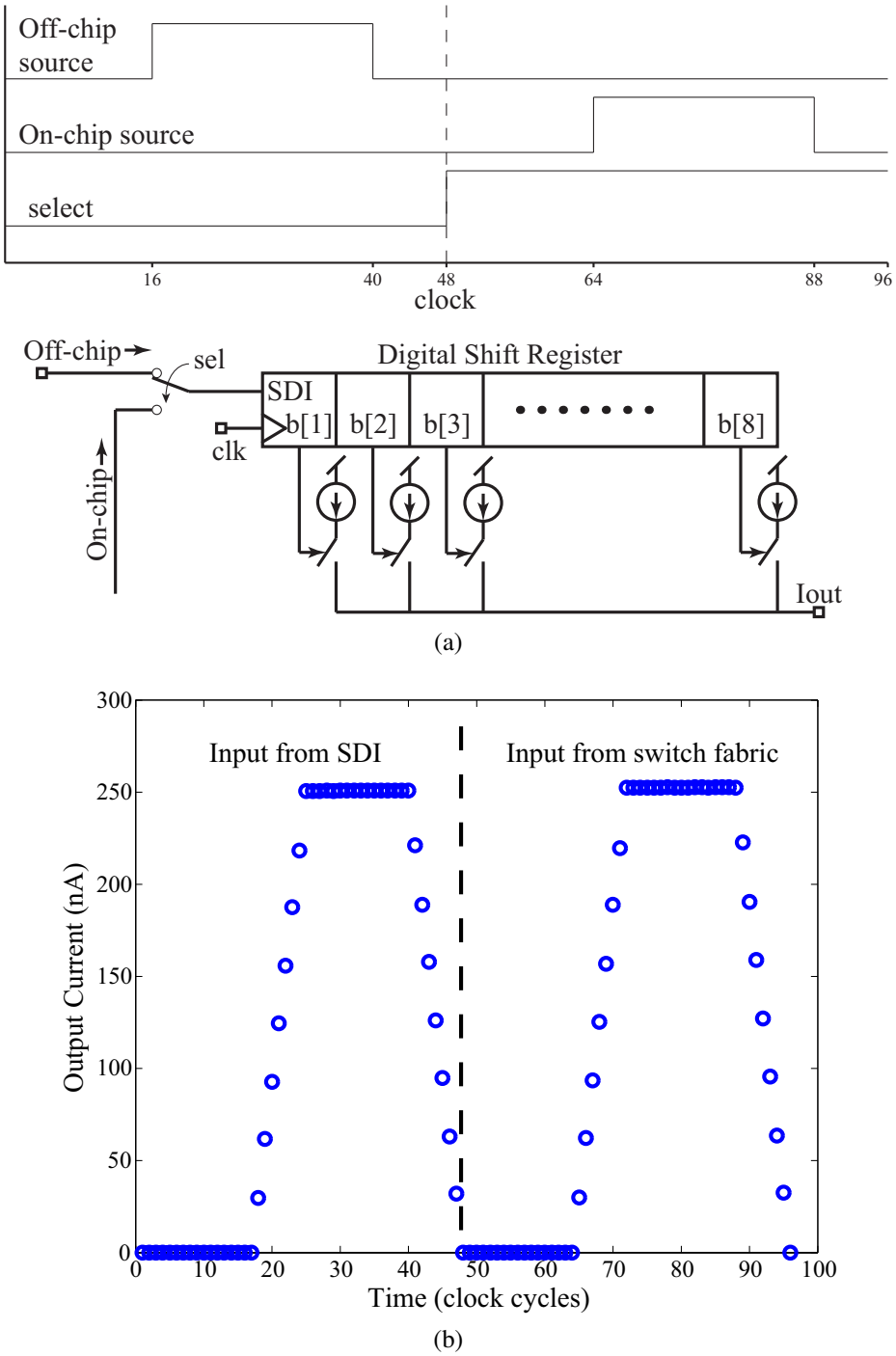


Figure 38: Serial data for the registers can be loaded from either an off-chip source or the on-chip switch matrix. (a) The test setup for loading the register highlights the switch that selects between on- and off-chip sources. The top graph shows a timing diagram with trains of zeros and ones coming from each the input sources. The schematic diagram below it shows each register bit controlling an equally weighted current source for easy read out. (b) The output measurement shows identical current readings from both the on- and off-chip register data.

cost as well.

The unit capacitance of the register is simulated to be 50 fF. There are many gates in the flip-flop, so a lumped unit capacitance is extracted from the dynamic power of 3 uW simulated for one bit at 10 MHz. The lumped unit capacitance value is used to calculate the dynamic power of the registers in systems with the equation $P = N_{bits}C_{unit}V_{DD}^2 f$.

4.3 Programming Methods

The RASP 2.9 analog signal processor can be fully programmed and tested with the Mathworks MATLAB environment. Furthermore, the general CABs are supported by a Mathworks Simulink design framework. The user begins by creating a block level diagram of the desired circuit in Simulink, using a library of linear and nonlinear elements. Each block has both a signal processing function in Simulink and a corresponding SPICE subcircuit definition. After testing the block diagram in Simulink, the MATLAB program 'Sim2Spice' compiles the block diagram into a SPICE netlist of RASP 2.9 components [44]. The SPICE netlist can be viewed independently for debugging or immediately compiled into a switch list via GRASPER, a C-based place-and-route tool [35]. The switch list represents bias currents for included CAB elements, analog switch elements, and the digital routing between the programmed elements and the I/Os. Compiling the netlist and the switch list is typically accomplished with a single MATLAB operation.

To aid in debugging, there also exists the FPAA Routing & Analysis Tool (RAT) [37], a GUI that illustrates the topology of the programmed switches and CAB elements. Once a switch list has been generated, MATLAB can program the circuit on the RASP via an Atmel ARM7 TDMI microcontroller. Individual switches can be programmed to within 9.5 bits of accuracy in less than 50 ms, using methods similar to [22].

Temperature and noise effects are important to any analog designer; however, we are presenting a platform for countless system implementations. Each system will have its

own non-idealities; therefore, a global specification cannot be provided. We have, however, provided an analysis of the line capacitance of the routing fabric, which will be useful when calculating the SNR of a given system. Future work involves using the routing information to accurately model individual signal processing blocks. By incorporating these second-order effects into the Simulink-level blocks, the design engineer will have a better understanding of the overall system behavior.

The registered switches on the RASP 2.9v are designed to rapidly expedite testing. The current DACs, shift registers, and off-chip DACs allow the user to insert a desired current or voltage to any circuit node, and to probe every other node. The shift registers allow us to test each block individually and to quickly debug and calibrate any programmed device on the chip. MATLAB scripts can automatically perform calibration by comparing outputs to desired results, modifying the switch list, and reprogramming the circuit for another test cycle.

4.4 Results and Applications

The RASP 2.9v adds capabilities for on-chip data conversion and digital enhancement, while maintaining the functionality of the earlier RASP 2.8 chips, which have been used to implement Gm-C filters, AM receiver, and speech processors [19].

This section reports on the performance of four systems that highlight key improvements of the chip: the current-mode DACs, a large-scale image processor, an arbitrary waveform generator, and an analog architecture for bitwise arithmetic. Each of these example systems would have been difficult or impossible to compile on previous FPAA platforms. The chip is validated by demonstrating a wide versatility of systems that can be compiled, while achieving comparable performance to systems that were fabricated in custom silicon.

4.4.1 Programmable DAC Core

One important use of the chip's digital infrastructure is to compile current-mode DACs onto the chip. This new capability allows users to easily apply inputs to current-mode circuits, using the chip's SPI protocol. In each column of CABs, SPI controls three 8-bit DAC CABs connected serially. Taking advantage of the FPAA's reconfigurable nature, the chip provides the resources to compile DACs rather than include fixed DACs. This flexibility allows us to try various topologies, alter the least significant bit, or use that area for something else if DACs are not needed.

The RASP 2.9v architecture makes it easy to implement binary-weighted current DACs. Figure 39 shows the schematic and FPAA implementation of a DAC based on individual current sources. The current source implementation has the benefit of ease and flexibility of design; even a non-standard mapping can be programmed. Another potential topology is an FG-based diffuser tree. The diffuser tree implementation has a more constrained design, but the use of small conductance ratios dramatically reduces temperature dependence.

Figure 39c shows the response of an 8-bit floating-gate current source DAC with LSB of 0.98 nA. Currently, the setting time of the DAC is limited by the SPI clock speed of the microcontroller. The system is clocked at 1.39 Mbit/s. This architecture is most efficient when all three DACs in a column are being utilized. Three DACs in a column can be clocked in 17.3 μs, yielding an effective SPI setting time of 5.77 μs/sample. For the eight-bit DAC at 1.39 Mbit/s, the dynamic power consumption is calculated to be 3.2 μW from $P_{dynamic} = N_{bits} C_{unit} V_{DD}^2 f$, where C_{unit} is the unit capacitance of the register. The static power is calculated as 614 nW, which is $P_{static} = \left(2^{N_{bits}} LSB\right) V_{DD}$. Our total power at 173 kS/s is thus 3.8 μW. The max INL and DNL are measured to be 2.13 and 1.16 LSB, respectively (shown in Figures 39d and 39e).

The closest DAC architecture comparison from the literature is the floating-gate current-mode DAC in [45]. This DAC is based on binary weighted FG current sources where the FGs are programmed to a LSB of 50 nA. The DAC reported 7-bit accuracy with 0.5 LSB

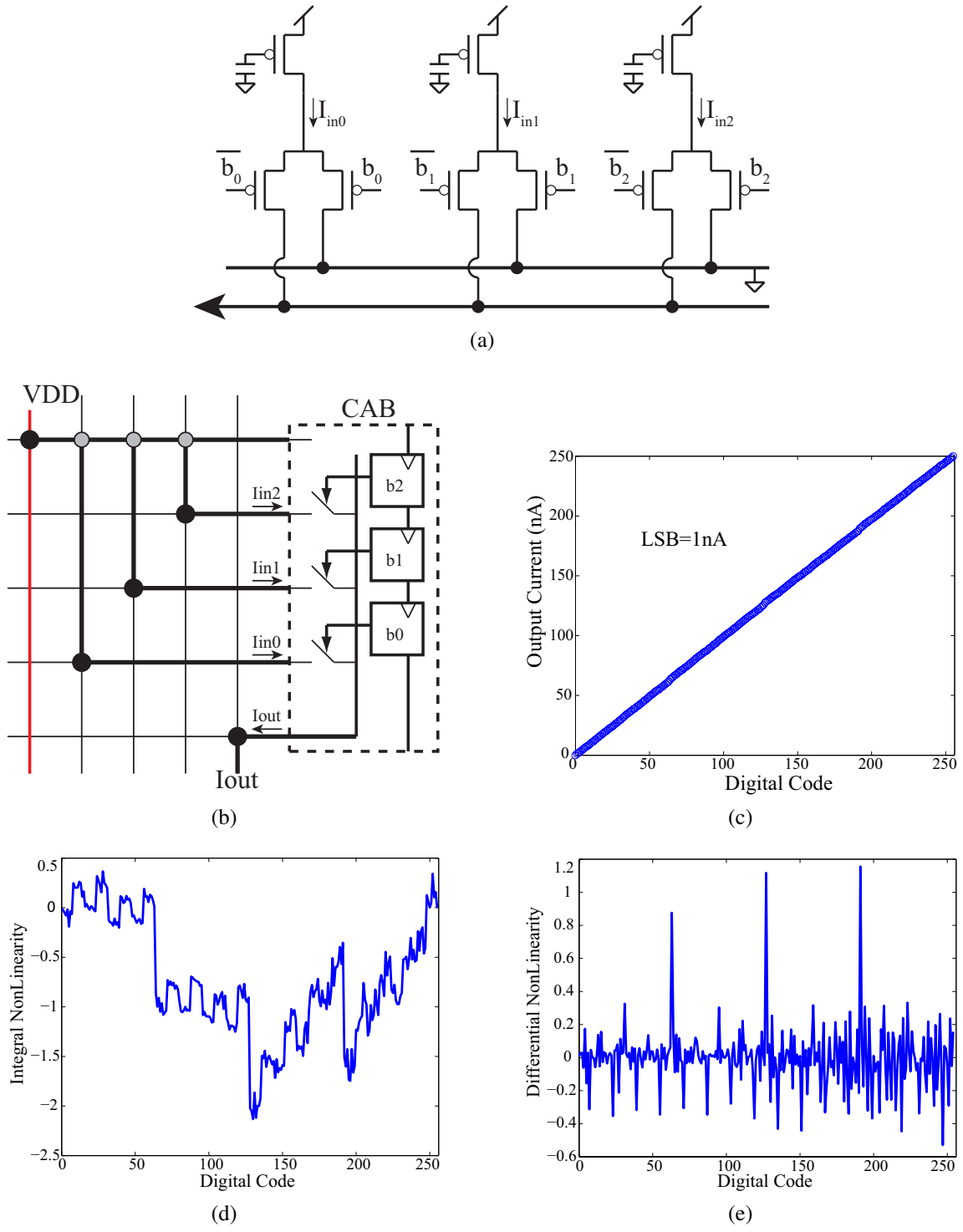


Figure 39: The on-chip compilable DAC. (a) The schematic and (b) FPAA implementation of the floating-gate current-source DAC. (c) The measured results from a compiled 8-bit current DAC shows an LSB of 0.98 nA. (d) The INL and (e) DNL plots from the 8-bit current DAC.

linearity error, but no power or speed numbers. The next closest topology is the floating-gate DAC presented in [46]. This DAC is cited in Table 6 for a more thorough comparison because it is very similar to our own, and the DAC in [45] did not include many specifications. Their architecture also uses floating-gate current sources, but it is slightly different in that it uses multiple gates to couple onto the floating node rather than programming it with precise charge. The 8-bit DAC in [46] was fabricated in 1.2 μm custom silicon and reports INL and DNL of 1.09 and 0.8 LSB respectively. They used an LSB of 3.75 μA and achieved 5 MS/s at 850 μW.

4.4.2 VMM Applications

Figure 40 illustrates the implementation of a current-mode VMM based on the design in [47]. The VMM is a modified current mirror that uses the weights of directly programmed switch elements to multiply the input currents and sum the output currents via Kirchhoff's current law (KCL). Negative multiplications can be implemented with a differential configuration. Using a constant bias current for inputs allows for consistent speed and power. The VMM CABs in the RASP 2.9v were created to efficiently place and route this architecture, utilizing a large proportion of the routing fabric for computational purposes, as shown in Figure 40b. A more detailed VMM discussion is the topic of Chapter 5.

Figure 40c illustrates the performance of scalar multiplication using the directly programmed devices compared to using the indirect devices. The advantage of the direct FG is clearly shown with one programming pass. The direct FG VMM shows accurate 4-quadrant multiplication of 4.5 bits, whereas the indirect FG system shows significant gain error as well as large offset error. These errors are traditionally compensated for with multiple programming passes using an adaptive process. However, such an adaptive programming step is not always practical or even possible. By restricting the range around a bias current and calibrating each multiplication, the accuracy of the direct VMM can be increased to 6 bits.

One application of the VMM circuit in the RASP 2.9v is as a low power front-end image processor. Since CMOS imagers produce currents as outputs, current mode VMMs are a

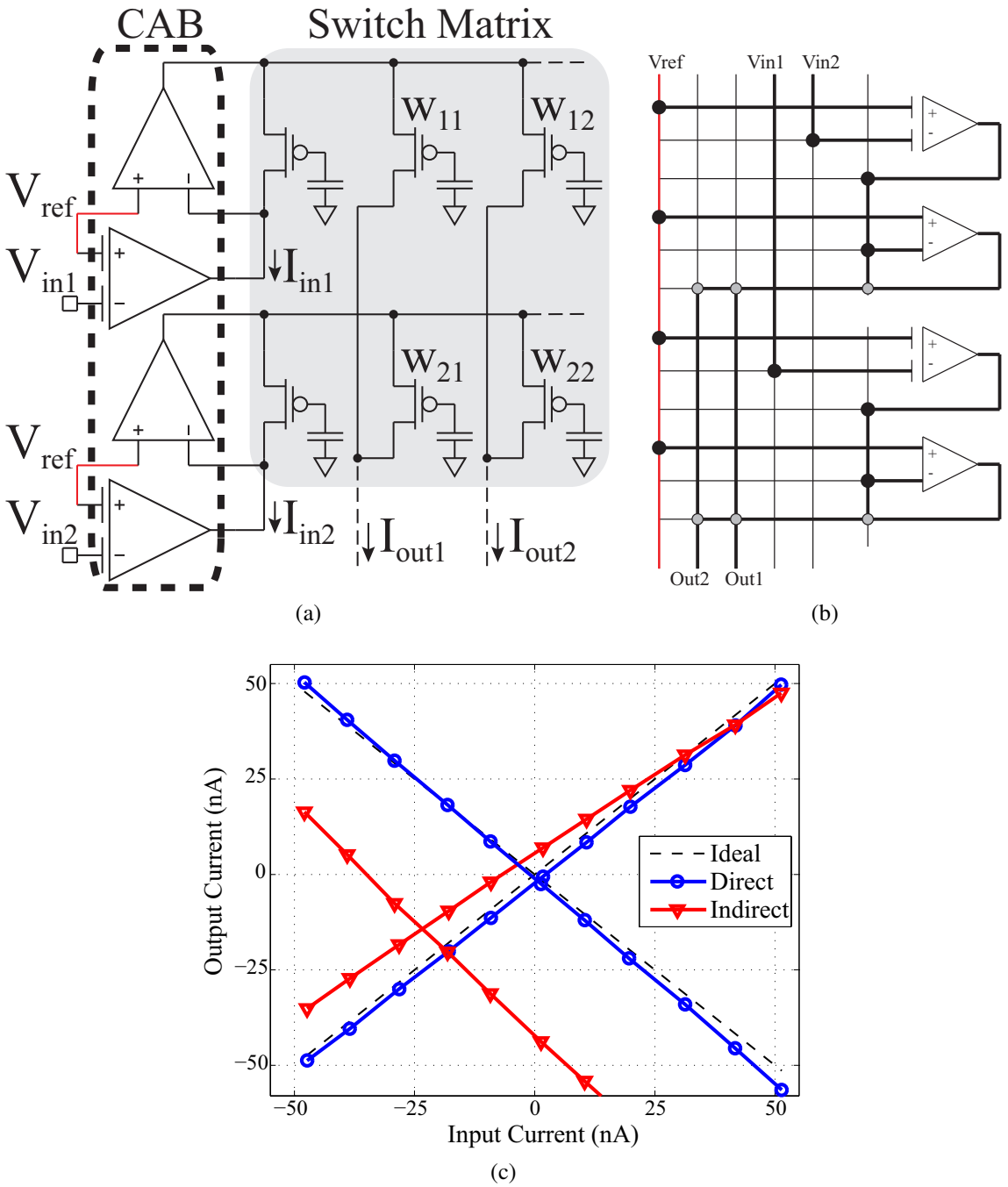


Figure 40: (a) The schematic and (b) FPAA implementation of a 2×2 VMM. (c) Data from a differential 1×1 VMM. The directly programmed devices in the VMM cab allow accurate multiplication (to 4.5 bits) on the first programming pass, eliminating the calibration needed in earlier RASP designs for linear processing. The one-pass programmed indirect-switch VMM shows significant gain and offset error. Calibration can be used, however, to increase the accuracy to 6 bits.

natural way of performing linear operations such as edge detection, smoothing, whitening, and discrete cosine transforms.

Figure 41 illustrates an image transform system implemented on the RASP 2.9v. The image is scanned in and a separable transform is performed with two passes: the first is a convolution with the S1 vector, and the second is a convolution with S2. Data from the two transforms are shown on the right side of the figure: a 3×3 Sobel edge detector and a 9×9 smoothing filter. This system makes use of the on-chip DAC because the test image is being generated by a PC rather than a current-mode imager. This topology highlights a design feature of the compilable DACs in that they can be configured in serial; we only need to shift in one new sample and let the other samples shift to the next DAC.

4.4.3 Arbitrary Waveform Generator

The RASP 2.9v is particularly well suited for arbitrary waveform generation (AWG). Figure 42 illustrates the architecture of an AWG programmed on the FPAA, as well as several waveforms that were generated. The AWG makes optimal use of the switch fabric, as every transistor acts as a memory element, holding the value of the current it will pass to the output channel. As the shift register scans the rows sequentially, the stored currents of the elements in that row flow down to the appropriate channels. The scan bits are calibrated to the number of elements in the stored signal, so that the first row of devices switches on just as the last row switched off. The waveform frequency is controlled by changing the scan speed. This structure allows multiple columns to be selected by the row register at the bottom, to select among many stored waveforms. Additionally, since the waveforms are in current mode, the register can be set to select more then one column at a time, resulting in a waveform that is the sum of the two source waveforms.

We used a wide range amplifier in transimpedance configuration as a I-to-V converter in order to generate easily readable voltage outputs, shown in the lower portion of Figure 42a. The amplitude and offset of the output waveform was controlled by the amplifier bias and reference voltage respectively. This topology is also beneficial in that it fixes the output

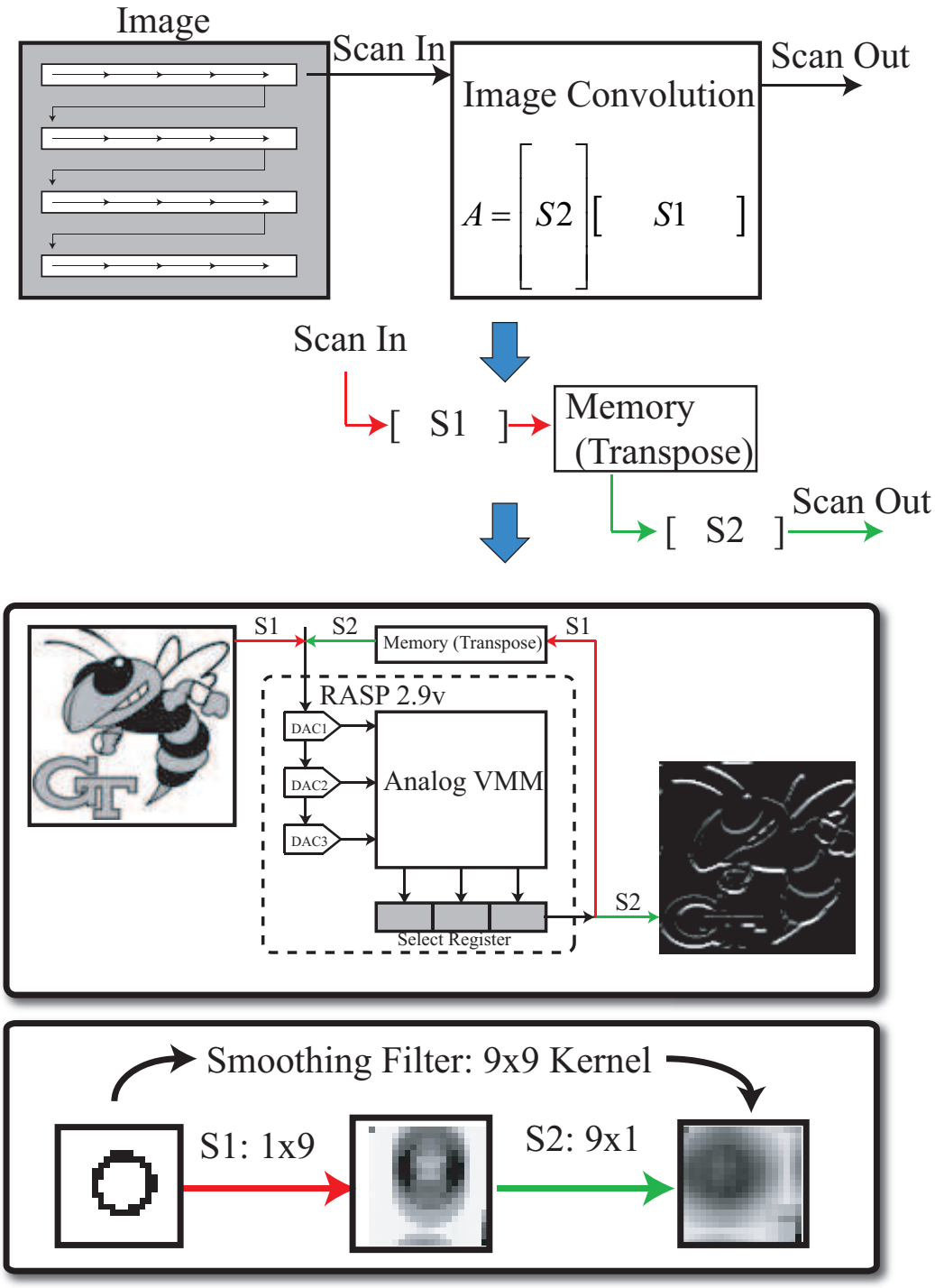


Figure 41: Application of the VMM in an image processing system. The image processor performs separable transforms scanning in the image and convolving by S1 and S2 in two passes. The kernels chosen for this test are a 3×3 Sobel edge detector and a 9×9 smoothing filter. The system schematic of the image processor front end shows the on-chip DAC components providing the signals to the VMM.

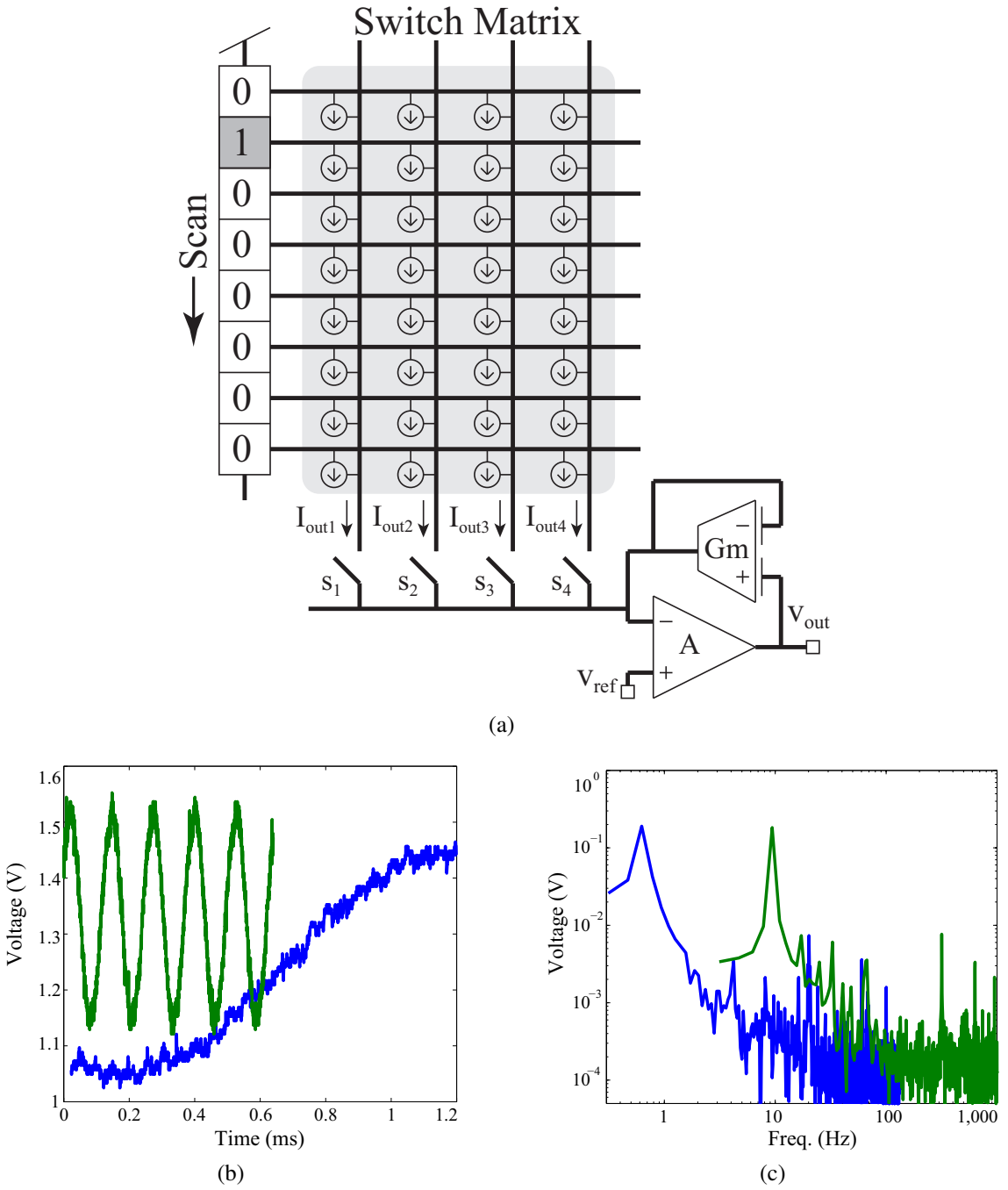


Figure 42: (a) Architecture for a 4-Channel Arbitrary Waveform Generator. The volatile switches short each row to V_{DD} serially, so each column passes the current supplied by the floating gate element at the intersection with the active row (shown here as empty circles). The volatile switch lines can be used to choose one channel—or combine multiple channels—to be converted to an output voltage. (b) Two sine waves generated by the AWG, using 40 devices scanned at 17.5 kHz and 310 kHz. Note that the FG-OTA has been programmed to allow different gain and offsets. (c) FFT of the two sine waves. Total harmonic distortion is -29.5 dB and -25.5 dB respectively. Note the small spike at the scan frequencies.

voltage of the current sources to the reference voltage. Although direct switches are used on the input-current side of the I-to-V, the output voltage signal must be routed on the indirect-switch lines, so that the output swing is not limited by the high resistance of the direct switches.

The time response (Figure 42b) and frequency response (Figure 42c) are shown for two waveform speeds. As with the on-chip DAC, the clock speed is limited by the SPI line from the microcontroller. Each wave is programmed with 40 elements, with one clocked at 17.5 kHz and the other at 310kHz. These clock speeds result in waveforms that are 437 Hz (17.5 kHz/40) and 7.7 kHz (310 kHz/40). The number of elements in a waveform can be expanded up to the full length of the vertical register—400 bits. This AWG structure is similar to that reported in [48], with our system being compiled in the reconfigurable hardware and the other being fabricated in custom 0.5 μm silicon. Ours achieves similar speeds, with the previous design reporting maximum clock of 250 kHz, where the SPI clock was run up to 1.92 MHz. At this maximum speed, the dynamic power dissipation is calculated to be 1.11 μW. The system operates with a single bit shifting down the registers, so we only need to add up the two bits that are changing. The static power is from a combination of the signal DC current and the I/V stage. The DC current is 100 nA, and the I/V includes two OTAs: the voltage amplifier operating with 100 nA and the feedback Gm stage has a bias of 200 nA to sink the maximum signal current. The static power is 1.6 μW, which results in a total power of 2.7 μW.

4.4.4 Mixed-Signal FIR Filter

The ability of the RASP 2.9v to shift through control bits opens opportunities for bit-wise arithmetic. A distributed arithmetic FIR filter is a common and powerful bit-wise operation. An FIR filter has certain advantages over traditional analog filter—such as linear phase—which motivates its inclusion in our analog toolbox.

The RASP 2.9v is particularly well suited for a current-mode implementation of this operation. The multiplication of bits by weights is implemented by current sources that can

be left open or connected to the output, where the addition is achieved simply by KCL.

With the shift register controlling the bitwise activation of the current sources, the full system can be implemented in multiple ways. A classical architecture for a mixed-signal distributed arithmetic FIR filter is presented in [49] and involves a straight-forward mapping of the digital blocks to the analog domain.

Figure 43 shows a novel architecture for the mixed-signal FIR filter. The filter will maintain its linear phase, while having an analog input and output signal. The input stage is a integrate-and-fire sigma-delta converter [50]. The output of this stage is a digital pulse train, shown in Figure 43b. The spike rate is linear with input voltage and can easily be modified by the size of the capacitor when compiled, as shown in Figure 43c.

The spike train is sampled by the register and filtered by the weighted current sources. The pulse width of the sigma-delta converter can be tuned with V_{bias} to ensure that it matches the sampling rate of the register. Any filter coefficients can be programmed into the current sources, where a differential convention can be used to implement four-quadrant coefficients. Initial results from the low-pass filter are shown in Figure 43d. The output current is accurately reconstructing a filtered version of the input signal. The dynamic power consumption for the 24-bit register is calculated to be 7 μW at 1 MHz. The static power is a combination of the current sources and the four OTAs. The current sources are each biased at 10 nA, of which we average that half (12) are ON. The two OTAs in the I/V stage are biased at 100 nA for the voltage amplifier and 300 nA for the feedback Gm stage to handle the maximum signal current. The V/I OTA and comparator are biased at 500 nA and 100 nA, respectively. The static power is 5 μW, which results in a total power of 12 μW. This architecture was invented to take advantage of the RASP 2.9v elements; therefore, the comparisons to other systems must be made based on function rather than exact architecture. Our system falls under the category of floating-gate based mixed-signal FIR filters. The system in [49] is cited in Table 6 as the closest functional comparison, whereas [51] would also fall into this category of filter.

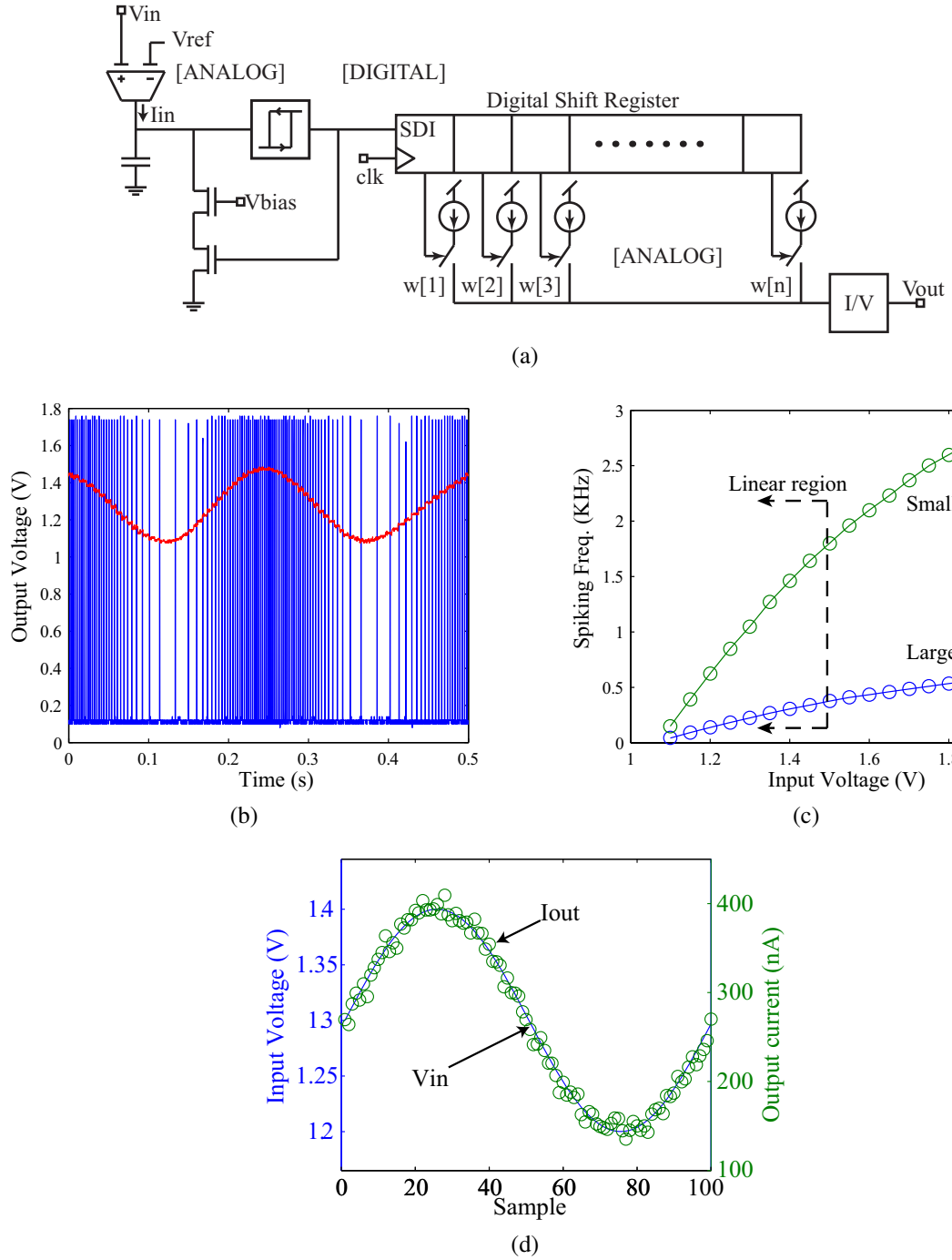


Figure 43: (a) Architecture for the mixed-signal FIR filter. (b) The integrate-and-fire spike generator produces digital pulses with a frequency based of the input current. (c) The integrating capacitor size can easily be reconfigured to tune the spiking frequency range. (d) The output of the mixed-signal system. The initial results show the output current correctly reconstructing a slow-moving input analog signal.

Table 6: RASP 2.9v system performance.

	This DAC	DAC in [46]
Resolution	8 bits	8 bits
INL	2.13 LSB	1.09 LSB
DNL	1.16 LSB	0.80 LSB
LSB	0.980 nA	3.75 μA
Conversion Rate	173 kS/s (SPI speed)	5 MS/s
Power	3.8 μW	850 μW
Number of channels	18	1
	This AWG	AWG in [48]
Wave	100 nA DC 100 nA_{pp}	300 nA DC 100 nA_{pp}
Clock	1.92 MHz	250 kHz
Elements	40	64
Power	2.7 μW	not reported
THD	-25.5 dB @ 310 kHz -29.5 dB @ 17.5 kHz	not reported
	This FIR Filter	FIR Filter in [49]
Size	24 bit	16 tap
Sample speed	40 kHz	50 kHz
Power	12 μW	16 mW
Filters	LPF	LPF, BPF, comb

4.5 Conclusion

The RASP 2.9v is designed for mixed-signal computation, with compilable DACs for signal conversion, VMMs for efficient linear operations, generic analog CABs for many non-linear operations, and digital registers for digital storage and dynamic reconfigurability. We have demonstrated a current-mode DAC, a VMM, an embedded image processor, an AWGs, and a bit-wise FIR filter. These are key building blocks allowing implementation of high impact systems like analog/software-defined radio [52] and low-power FFT processors [53, 54]. A summary of key parameters is provided in Table 5, with a summary of system performance in Table 6.

CHAPTER 5

SYSTEM DESIGN: THE VECTOR-MATRIX MULTIPLIER

The *algorithm* phase of the coordinated approach to analog signal processing is the bridge between the hardware architecture and the software tools. The primary function of this phase is to create efficient circuits that implement signal processing functions. The goal of these efficient circuits is to minimize size, weight, and power (SWaP), which are recognized as key criteria modern embedded systems. Mobile and tactical systems are often on a fixed power supply, so creating compact and ultra low power circuits will have a direct effect on the SWaP of the entire system. Minimizing these costs can result in an increased lifetime, a lighter load for the carrier, or more space for other processing. Modern analog signal processing hardware is poised to make huge leaps in power efficiency over the traditional digital signal processor, but we need to provide the algorithms and circuits that can properly leverage the hardware.

This chapter presents the implementation of an analog vector-matrix multiplier (VMM) on the FPAA [47, 55]. The VMM is a core component in many signal processing applications. Vector-matrix multiplication is commonly performed in FIR filters, 2-D block image transforms, convolution, correlation, and classification [42]. Recently, custom analog

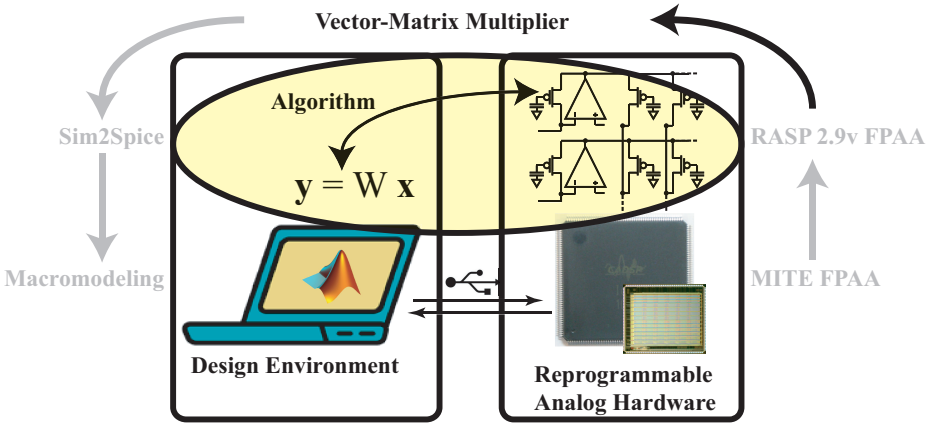


Figure 44: The coordinated approach to FPAA design: VMM.

VMM cores have provided a low-power, high-throughput tool for signal processing [56]. Several orders of magnitude in efficiency can be gained by allowing the natural physics of the transistors to perform the multiply and accumulate (MAC) operations. Analog VMMs have recently demonstrated low-power solutions in such embedded systems as a transform imager and an OFDM receiver [43, 54]. Although the benefits of analog VMM have been clearly demonstrated, the systematic design of such a system has only been loosely defined in the existing literature. The field-programmable analog array (FPAA) is the ideal tool to incorporate analog signal processing into low-power embedded systems. The FPAA gives us a workspace to create a systematic discussion on analog VMM systems. We can use the FPAA to test multiple VMM design choices and put the overall design on solid ground. Figure 44 shows how the VMM algorithm fits into the coordinated-design framework.

5.1 Building a VMM on FPAA Hardware

This section provides a thorough discussion on the design of analog VMMs. Although the RASP FPAA is versatile enough to implement any topology discussed here, we will guide our design choices towards the most efficient use of the FPAA architecture.

5.1.1 Analog Vector-Matrix Multiplication

Vector-matrix multiplication is mathematically defined as $\vec{y} = W\vec{x}$ where $W \in \mathbb{R}^{M\times N}$, $\vec{x} \in \mathbb{R}^N$, and $\vec{y} \in \mathbb{R}^M$. We restrict our discussion to real values, since we will be dealing with physical quantities. To get a sense of the analog elements needed to perform this task, we look at the component-wise output signal:

$$y_i = \sum_{j=1}^{N} w_{ij}x_j, \ i = 1, ..., M. \tag{24}$$

Each element of the output vector is made up of the inner product of the input vector with a row of the matrix, an operation requiring scalar multiplication and addition.

Fortunately, by utilizing current-mode signals, scaling and adding are two very easy and efficient operations. Addition can be performed by Kirchhoff's current law (KCL) simply

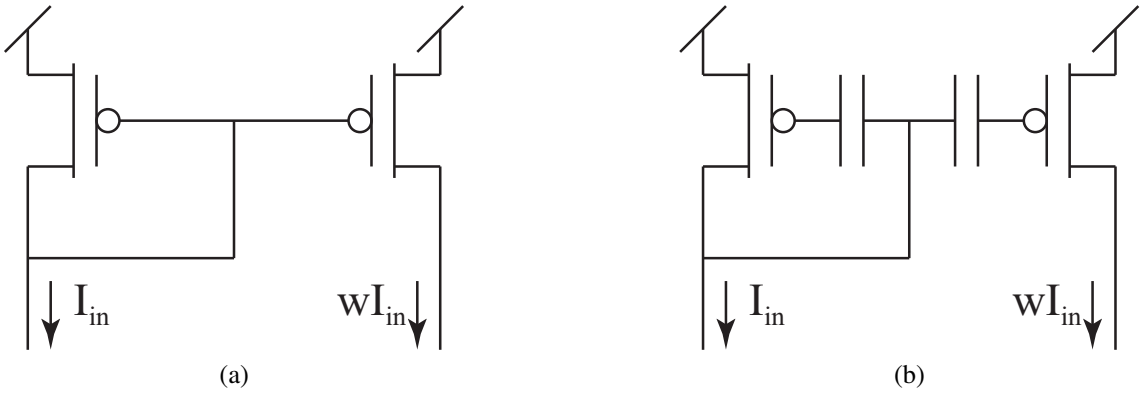


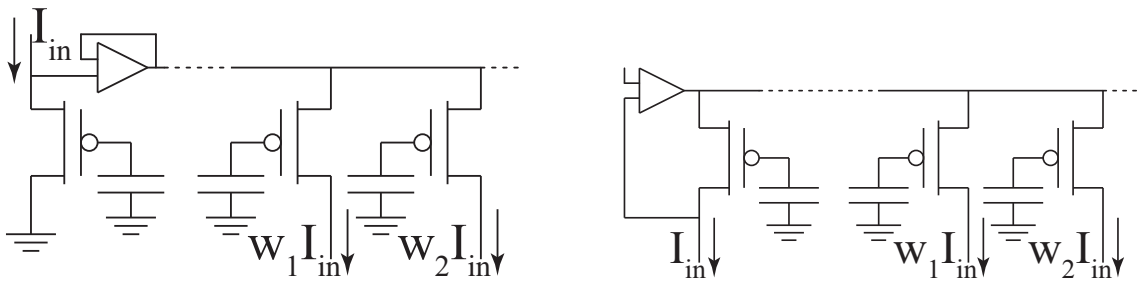
Figure 45: Two implementations of a current-scaling mirror. (a) Traditional current mirrors scale current based on the fabricated aspect ratio. Although this can perform a fixed coefficient multiplication, it is not reconfigurable post fabrication. (b) By introducing floating gates into the current mirror, we can set the scaling weight based on the amount of charge on the floating node.

by mixing two currents. This function requires no power to perform. Scaling currents is simple to perform as well. One transistor is used to sense the input current and broadcast the log-compressed voltage, while another transistor receives the voltage and exponentiates it back into a current. This operation is recognizable as a current mirror, illustrated in Figure 45. A common CMOS current mirror, shown in Figure 45a, scales current based on its geometry: $I_{out} = I_{in}(W/L)_2/(W/L)_1$. This scaling is commonly used to create bias currents as multiples of a reference current.

Although the common-current-mirror approach performs our desired scaling, the application demands a system that is capable of being rescaled after fabrication. Figure 45b illustrates the use of an FG mirror to create a programmable scaling value [57]. To achieve ultra low power, we operate each transistor in the subthreshold regime, where the drain current has the following exponential dependence on gate voltage:

$$I_d = I_0 e^{\frac{V_s - \kappa V_g}{U_T}}. \tag{25}$$

Here, I_0 is a pre-exponential constant term, κ is the capacitive division between the oxide capacitance and the depletion capacitance, and U_T is the thermal voltage. For this pFET, all potentials are referenced to the bulk. Analog subthreshold operation is performed by



(a) (b)

Figure 46: Two implementations of a source-coupled floating-gate current mirror. (a) Buffered input stage. The source voltage is buffered with an OTA follower. (b) Log-amp input stage. A logarithmic amplifier is used to compress the input current.

using gate biases below the threshold value. This facilitates integration with other systems, because we can leverage the subthreshold current levels without the low V_{DD} values that typify subthreshold digital design.

The mirror's scaling factor is found by taking the ratio of the output to the input current,

$$\frac{I_{d,out}}{I_{d,in}} = e^{\frac{\kappa\left(V_{fg,out}-V_{fg,in}\right)}{U_T}} \equiv w. \tag{26}$$

Here, V_{fg} is the voltage on the floating node, which is a function of the stored charge and any capacitively coupled voltages.

Although the gate coupling of Figure 45 serves the purpose of a weighted current mirror, we can also use the source-coupled topology of Figure 46. Here too, the input current is sensed, log compressed, and broadcast. This topology allows us to take advantage of the many thousands of FG switch elements in modern FPAAs, which are two-terminal devices. Source coupling is also beneficial because it eliminates the effect of kappa variation with input signal. However, whereas the gate is a high-impedance node and has no current draw, the source will sink current, so it must be broadcast. Figure 46 shows two ways of broadcasting the source voltage, referred to here as buffered and log-amp stages, respectively. Source coupling involves drawing the input current out of the drain of the sensing FET, then buffering the resultant source voltage to the output stages. The bandwidth of the buffered structure is $\omega = g_m/C$, where $g_m = I\kappa/U_T$. The log-amp structure uses the

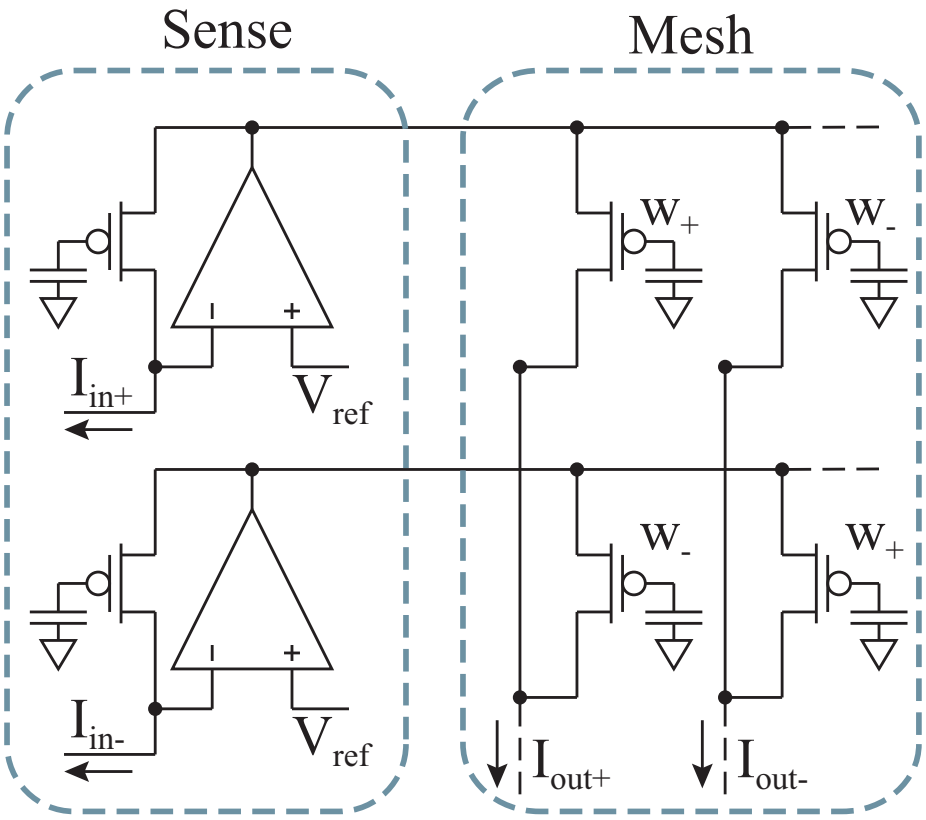


Figure 47: Schematic of the analog VMM circuit. The input floating-gate stage produces a log-compressed voltage representation of the input current. This voltage is broadcast to the source of each FG element in that row. These FGs produce an output current that is a scalar multiple of the input current, which are then summed along a column by KCL.

amplifier of [58] as the sensing stage. This incorporates active feedback that increases the bandwidth to $\omega = Ag_m/C$, where A is the voltage gain of the OTA, which is typically $100 - 200$ V/V on the RASP FPAA.

As mentioned in the previous discussion, the architecture of the FPAA guides the design of the output array. As defined in Equation 26, each output weight depends on the ratio of the output FG charge with the input state's charge. We achieve this ratio by connecting an FG element's source to the broadcast line. For the summing operation we get a current-mode addition by tying the drains of multiple output elements together. This fits exactly with what is available in the RASP FPAA—an array of FG switches that are source coupled along a row and drain coupled along a column.

Pulling all of this together, we have the structure in Figure 47. We use the log-amp

structure to log compress each of the N elements in the input vector, broadcasting the resultant voltage across a row. Each row has M output-stage FGs, creating the multiplier weights. The output current of each scaling element is then summed along the M columns. With FG transistors, the output impedance is mainly degraded by the drain voltage coupling back onto the floating node. Adding a cascode transistor at the output of each column helps to reduce the effect of the drain voltage on the computation. Thus, we have successfully implemented Equation 24 in an extremely compact, highly dense fashion on the RASP FPAA.

5.1.2 Signal Conditioning

From the structure in Figure 47, it is clear that the input and output currents must be unidirectional. This requirement results in weights that need to be strictly positive. In this scenario, we are left with a single-quadrant multiplier—positive signals and weights.

Ideally, we would prefer to have full four-quadrant multiplication—both positive and negative signals and weights. To achieve four-quadrant multiplication, we incorporate a differential syntax. We define the signed signal to be the difference between two positive currents:

$$I_{in} = I_{in+} - I_{in-}. \tag{27}$$

We will constrain the differential signals to small changes around a bias (I_B) current:

$$\frac{I_{in+} + I_{in-}}{2} = I_B. \tag{28}$$

Now, we will utilize a similar syntax for the weights:

$$w_+ = w_B + \frac{\Delta w}{2},\ w_- = w_B - \frac{\Delta w}{2}. \tag{29}$$

For one four-quadrant differential multiplier element, we have

$$\begin{bmatrix} w_+ & w_- \\ w_- & w_+ \end{bmatrix} \begin{bmatrix} I_{in+} \\ I_{in-} \end{bmatrix} = \begin{bmatrix} I_{out+} \\ I_{out-} \end{bmatrix}, \tag{30}$$

which is shown in Figure 47. This core multiplier cell has a mesh four times as large as the single mirror. The final gain of the four-quadrant cell is:

$$I_{out+} - I_{out-} = (I_{in+} - I_{in-}) \cdot (w_+ - w_-)\,. \tag{31}$$

The differential-signal operation is illustrated in Figure 48 [55]. An additional benefit of differential signals is that it will remove DC offsets and even-order harmonics.

5.2 Power, Speed, Noise, and Temperature Performance

In this section, we discuss the relevant performance parameters. The power-delay product illustrates an important trade-off in subthreshold design: speed at the expense of power. Also, two of the most often criticized shortcomings of analog computation are discussed—noise and temperature dependence. Although these effects are unavoidable, by characterizing them we can understand their effects and design the rest of the system to compensate for them.

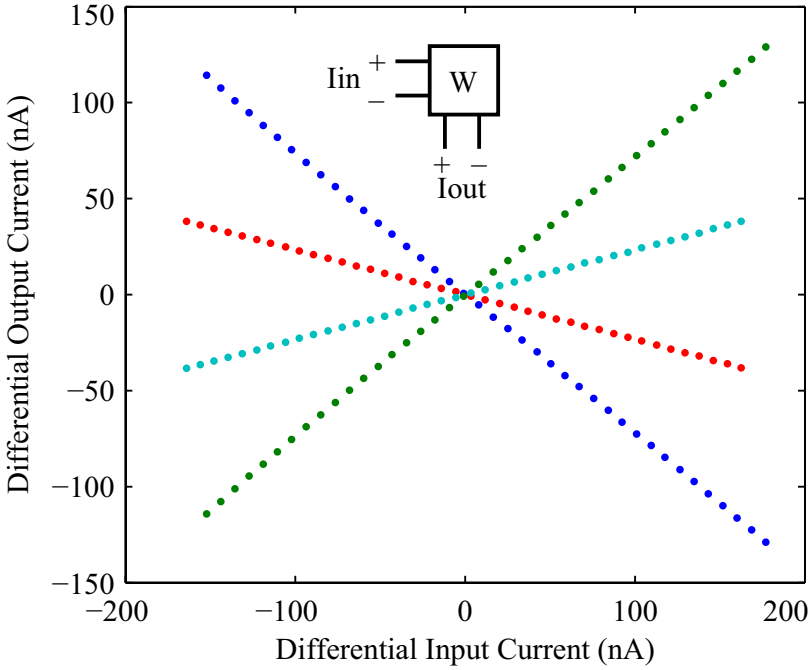
5.2.1 The Power-Speed Tradeoff

One of the most attractive features of subthreshold processing is its extreme power efficiency. This efficiency is very important for VMM applications in mobile devices or any system on a limited power budget. However, this low-power operation comes at the cost of operating speed, manifest in the *power-delay product*. This discussion is in similar fashion to that given in [59].

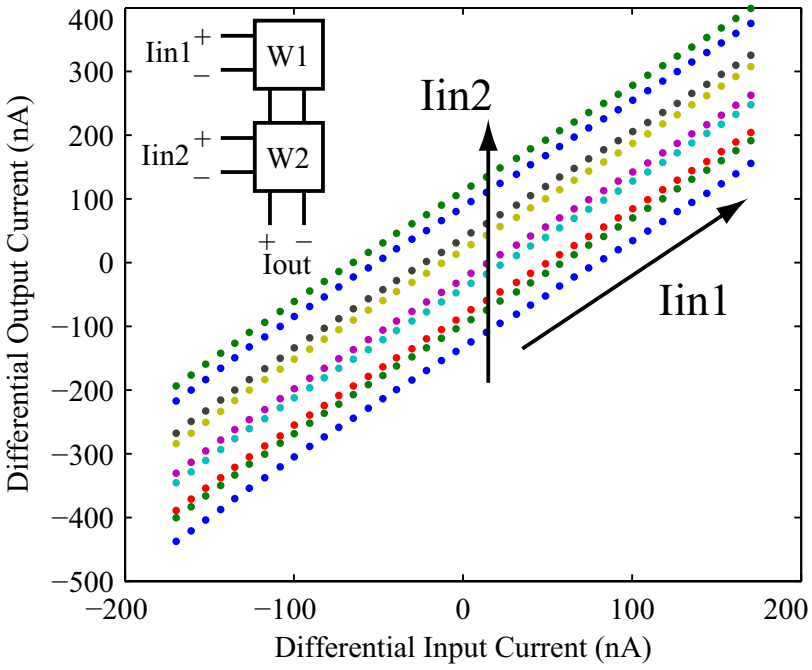
The input stage is shown to dominate the frequency response of the VMM system. The small-signal analysis of the input stage follows the analysis given for the log-amp [58]. For the case where the input capacitance is much larger then the parasitic feedback capacitance, the dominant pole is given as:

$$p = -\frac{AG_{s1}}{C_{in}}, \tag{32}$$

where G_{s1} is the source conductance of the feedback FET, A is the voltage gain of the OTA,



(a)



(b)

Figure 48: VMM current mode sweeps. (a) A 1×1 VMM; the inputs are swept differentially, to create differential outputs. The multiplier coefficients for this sweep are ± 1 and ± 0.5. (b) A 2×1 VMM; this sweep demonstrates the summation of the two inputs. One input is swept differentially for each constant value of the second input, resulting in vertical offsets.

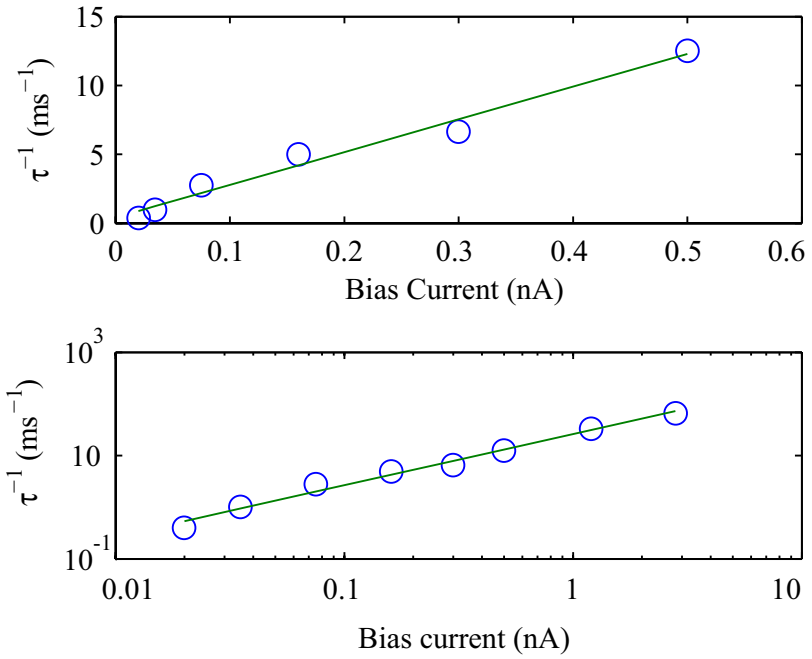


Figure 49: Linear and log plot of the inverse VMM time constant. The inverse of the time constant increases linearly with an increase in bias current. The slope of the linear plot corresponds to an input capacitance of 1.62 pF. The slope of the log plot is 0.99.

and C_{in} is the total capacitance at the input. The overall system will have also an output pole due to the I/V conversion stage, which must be accounted for.

Here, the benefit of the OTA can be seen for the log-amp input stage; the input impedance is decreased from buffered input stage by the factor A, increasing the bandwidth by that same amount. The typical value of A on an OTA of the RASP FPAA is $100 - 200$ V/V, so a sizable increase in bandwidth can be achieved.

By substituting the subthreshold equation for G_s we see a -3 dB frequency of:

$$f_{-3dB} = \frac{1}{2\pi}\frac{AI}{C_{in}U_T}. \tag{33}$$

This equation shows us that the operating frequency scales linearly with signal bias, and thus power. Figure 49 shows the speed of the response of the VMM for given bias currents, shown both on linear and log plots. The slope of the linear plot empirically gives us an input capacitance value of 1.62 pF, a reasonable result given the reconfigurable nature of the routing. The slope of the log plot is 0.99, which is expected from the linear dependence.

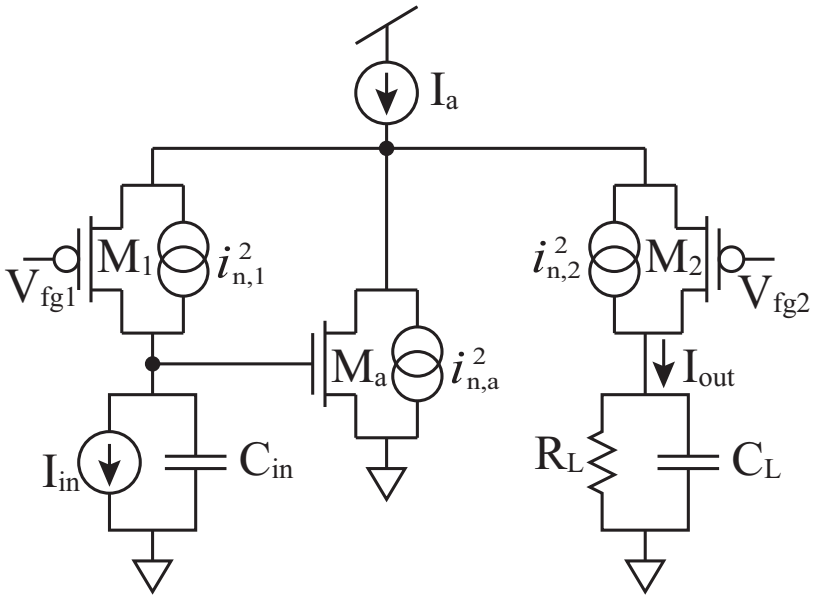


Figure 50: Schematic of the 1×1 VMM noise model.

This results in a power-delay product for our MAC cell of:

$$P\tau = 4V_{DD}U_T\frac{C_{in}}{A}. \tag{34}$$

The power is approximated as the product of the total current and the supply voltage. The factor of 4 is due to the power consumed in the OTA of a 1×1 cell; it provides current to supply both the input and output stages, and it has a copy of the total current in each of 2 internal branches. The full power will scale with the size of the matrix: $I = 2 \cdot row \cdot (1 + col)$. This power-delay equation is shown to be a linear function of capacitance and independent of the signal bias. In the log-amp input stage, the input capacitor has effectively been reduced by the factor A. The inverse of this product is the computation per unit power.

5.2.2 Noise Performance

On the topic of analog computation, the issue of noise performance is very important. We will analyze the core 1×1 cell of the log-amp source-coupled VMM following the discussion in [58]. To create an equivalent noise model, shown in Figure 50, we consider channel noise current sources for each transistor of the FG mirror and the OTA. Since we are restricting our operation to subthreshold currents, we will neglect flicker noise and

focus on thermal noise.

The thermal noise contributions are:

$$\hat{i}_{out}^2 = \hat{i}_1^2 \frac{g_{s2}^2}{g_{s1}^2} + \hat{i}_a^2 \frac{g_{s2}^2}{g_{ma}^2} + \hat{i}_2^2. \tag{35}$$

Referring the noise to the input, using the noise model $\hat{i}^2 = 2qI\Delta f$ [60] and substituting the subthreshold transconductance, we get:

$$\hat{i}_{in}^2 = 2qI_1 \left(1 + \frac{I_1}{\kappa I_a} + \frac{I_1}{I_2}\right) \Delta f. \tag{36}$$

We use the bandwidth found in Equation 33 and utilize the relation that the ratio $I_2/I_1 = w$. We note that the amplifier bias needs to source current for each FG of the current mirror, initiating the constraint $I_a \geq I_1 (1 + w)$. Given these definitions, Equation 36 becomes:

$$\hat{i}_{in}^2 = 2qI_1^2 \left(1 + \frac{1}{\kappa (1 + w)} + \frac{1}{w}\right) \frac{A}{4C_{in}U_T}. \tag{37}$$

What this highlights is the increase in noise power with A, effectively decreasing C_{in}. This goes back to the trade-off in topologies chosen for higher bandwidth. For lower noise, we can use a topology without the amplifier or increase the input capacitance. Figure 51 shows a plot of the current spectral density taken from a VMM compiled on the RASP 2.8a FPAA.

For the current mirror, the input signal is also the bias of the input stage. We will constrain the VMM operation to weights of a small range around 1, in which case we will use the average $w = 1$ for the final signal-to-noise (SNR) relation. With $w = 1$ and $\kappa \approx 0.5$, the three terms summed in the parenthesis can each be approximated as unity. The RMS SNR is now in the relation:

$$SNR = \sqrt{\frac{2C_{in}U_T}{3Aq}}. \tag{38}$$

We see from this equation that this multiplier's SNR is not dependent on input bias, but is increased by the input capacitor. However, the performance trade-off with increasing the input capacitance is an increase in power-delay. This trade-off is illustrated in the following constant:

$$\frac{P\tau}{SNR^2} = 6qV_{DD}. \tag{39}$$

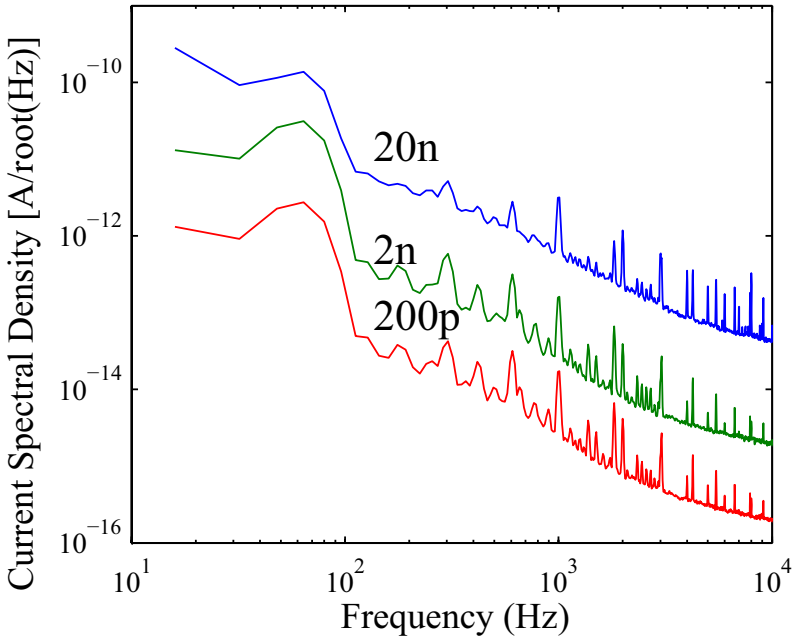


Figure 51: Current spectral density of the VMM noise. The lower bias current produces less current noise.

5.2.3 Temperature Dependence

A weight produced by subthreshold-current ratios will see a temperature dependence. This is unavoidable, but we can characterize it and thus compensate for it. To determine the temperature dependence, we will rewrite the weight from Equation 26 in terms of a constant thermal voltage, $U_T = U_{To}T/T_o$:

$$w = w_o^{\frac{T_o}{T}}. \tag{40}$$

The actual temperature is a change around a constant temperature: $T = T_o + \Delta T$. Now, lets say $\Delta T/T_o << 1$, so $T_o/\left(T_o + \Delta T\right) = 1 - \Delta T/T_o$. Using the relation from Equation 31, we have the VMM differential weight in temperature form:

$$\frac{I_{out+} - I_{out-}}{I_{in+} - I_{in-}} = w_{1o}^{\frac{T_o}{T}} - w_{2o}^{\frac{T_o}{T}}. \tag{41}$$

Imposing the constraints from Equation 29, we get:

$$\Delta w = \left(w_{Bo} + \frac{\Delta w_o}{2}\right)^{1-\frac{\Delta T}{T_o}} - \left(w_{Bo} - \frac{\Delta w_o}{2}\right)^{1-\frac{\Delta T}{T_o}}. \tag{42}$$

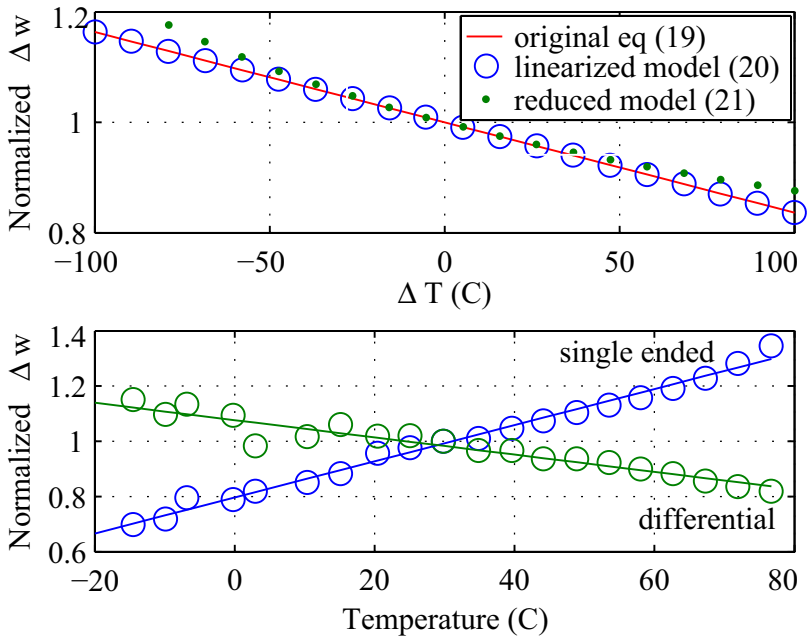


Figure 52: Temperature dependence of the VMM weight. (a) Plotting the derived equations shows that we were founded in our assumptions; the linearized model very closely follows the original equation and the reduced model fits for ΔT within ±50 C. The weights are normalized to the programming condition (30 C). (b) Temperature sweep of the weights from the FPAA VMM FPAA. The differential weights are shown to have a much smaller slope than the single-ended multiplier. The weights are normalized to the programming condition (30 C).

Using the binomial theorem, we can expand the terms to get:

$$\Delta w = \left(1 - \frac{\Delta T}{T_o}\right)(\Delta w_o) + \frac{1}{24}\left[\frac{\Delta T}{T_o} - \left(\frac{\Delta T}{T_o}\right)^3\right]\Delta w_o^3. \tag{43}$$

We benefit from the differential structure by seeing the even-order terms drop out. From this, we can drop the higher-order terms based on our original constraint that $\Delta w < 1$. We are left with a weight that is linear in the inverse of temperature change.

$$\Delta w = \left(\frac{T_o}{T}\right)(\Delta w_o) + O\left(\Delta w_o^3\right). \tag{44}$$

To verify our assumptions when deriving this simplified model, we have plotted Equations 42–44 in Figure 52a. A comparison is shown in Figure 52b of the weight versus temperature for the differential- and single-ended cases. The differential case is much less drastically affected by the change in temperature.

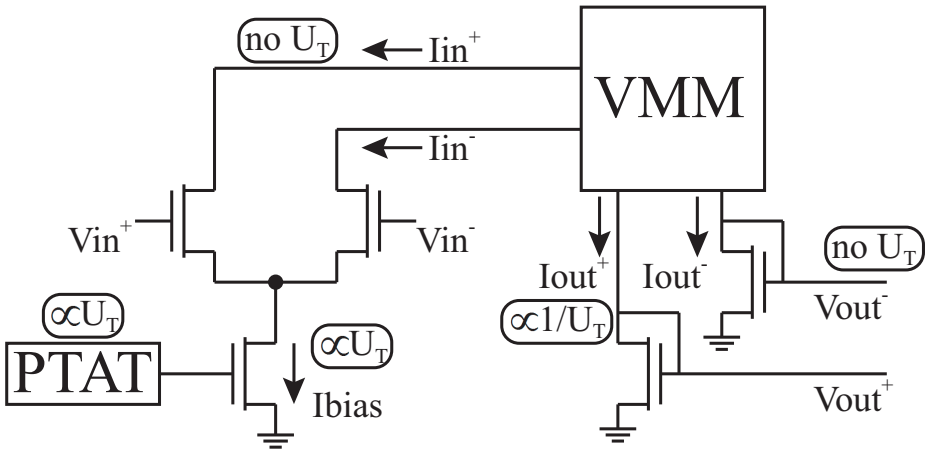


Figure 53: Schematic of the VMM temperature compensation circuitry. The diode-connected nFETs at the output cancel the temperature dependence of the multiplier weight. The input differential pair with PTAT current source will have no temperature dependence.

With the temperature dependence now defined, we can design the surrounding structures to compensate for this dependence. One common way to introduce a temperature term in the numerator is to pass the current-mode signal through a diode-connected nFET. The output voltage signal is now:

$$V_{out} = \frac{U_T}{\kappa} ln\left(1 + \frac{\Delta I_{out}}{I_{bias}}\right). \tag{45}$$

By taking the signals differentially and expanding the log terms, Equation 45 is:

$$V_{out}^{+} - V_{out}^{-} = \frac{U_T}{\kappa}\left[\frac{U_{To}}{U_T}\Delta w_o \left(I_{in}^{+} - I_{in}^{-}\right)\right], \tag{46}$$

canceling the temperature dependent term.

On the input side, we can introduce a temperature-neutral V-to-I stage with a differential pair. For the differential pair to be temperature neutral, it should have a proportional-to-absolute-temperature (PTAT) current source. This overall system is shown in Figure 53. The VMM is now in voltage mode, which makes it much easier to interface with other elements in an embedded system.

Table 7 shows a compilation of the performance metrics discussed in this section. The parameters are calculated for three common values of signal bias, based on a single differential cell (2×2). In the calculations, we used the input capacitance found in the delay

Table 7: Summary of theoretical performance parameters for a 1×1 differential VMM cell. We use: rows (r) and columns (c) = 2, C_{in} =1.6 pF, V_{DD} =2.4 V, A=165, and w=1.

Property	Expression	100 pA	1 nA	10 nA
Bandwidth (f)	$\frac{AI}{2\pi C_{in} U_T}$	63 kHz	630 kHz	6.3 MHz
Power (P)	$2r\,(1+c)\,IV_{DD}$	2.9 nW	29 nW	290 nW
Noise $\left(\hat{i}_{out}\right)$	$\sqrt{\frac{3qI^2A}{2U_T C_{in}}}$	3.1 pA	31 pA	310 pA
$MMAC/\mu W$	$\frac{A}{36\pi V_{DD} U_T C_{in}}$	22	22	22
SNR	$10log_{10}\left(\frac{2U_T C_{in}}{3qA}\right)$	30.2 dB	30.2 dB	30.2 dB

measurements and w of 1. The computation per power ($MMAC/\mu W$) is the inverse of the power-delay product in Equation 39. The computation per power and SNR are shown to be constant and independent of bias. The 45 nW for 1 million MAC operations used by this source-coupled architecture is a marked improvement in power efficiency over the 270 μW in the custom analog VMM in [56], which was itself a 1,000 times improvement over commercially available DSPs.

5.3 Methods and Tools for FPAA VMMs

In this section, we discuss the practical matter of implementing a VMM into an embedded system. We provide a full discussion on the density issues when utilizing an FPAA. To make it easy for engineers to utilize analog VMMs, we have incorporated it into the software compiler tools. In addition, we discuss a few supporting blocks, also supported by the software tools, that make designing full systems very easy.

5.3.1 FPAA Density

The architecture of the RASP FPAA is particularly well suited to implement the VMM structure discussed in Section 5.1. By utilizing the FG switches for computation, we can effectively use the switch matrix as the VMM mesh. Figure 54 shows an illustration of how the core differential 1×1 MAC block can be compiled into the switch matrix.

When discussing VMM density, the question arises of how large of a VMM can be built. In analog processing, the most computational gains are made with highly parallel

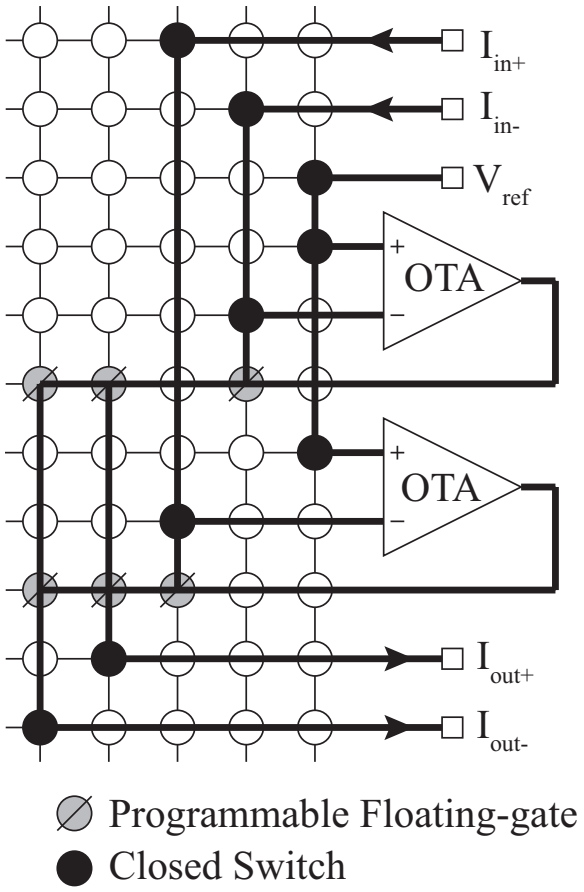


Figure 54: Map of the 2×2 VMM implemented with FPAA switches. The VMM utilizes floating gates programmed as both switches and computational elements. This allows very efficient utilization of all chip area and results in highly dense multipliers.

computation. The RASP 2.8a is one of the most recent and advanced FPAAs, with 55 general I/O and a 16-bit scanner I/O block; however, newer and larger RASP FPAAs are continuously being developed for which the same analysis will hold.

When using an entire FPAA solely for a VMM, the first constraint is the analog I/O limit. With 55 I/O, the largest square computation would be 27×27. Of course, the matrix by no means has to be square. In addition, the on-chip scanner can be used to multiplex multiple results to a single I/O pin. With the scanner, a 55×16 VMM is possible, with the 16 outputs being available in series.

The discussion gets more interesting if dealing with a VMM internal to a system on the

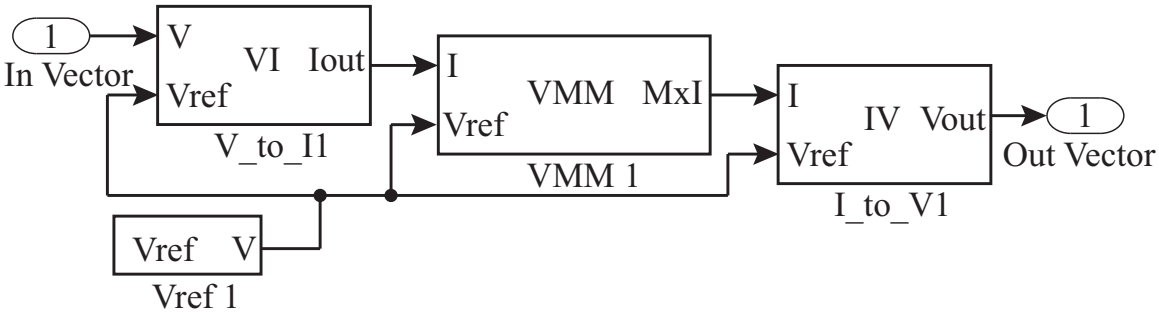
FPAA, where the signals do not need to be pinned out. In this case, the density is routing-limited. The RASP 2.8a is symmetric with 4 identical columns of CABs (cabstacks) that can be treated independently in the calculation and summed down the rows. Each cabstack contains 21 OTAs and 32 vertical lines. By referencing the diagram in Figure 54, we see that each input needs a vertical line and an OTA, each output needs a vertical line, and the reference voltage needs one vertical line. Each cabstack VMM has the constraint that the inputs and outputs add to no more than 31, with no more than 28 inputs. The total VMM is then four times this number by accounting for the four cabstacks. To put in numbers, one possible VMM is calculated to be 82×10.

5.3.2 Compiler Tools

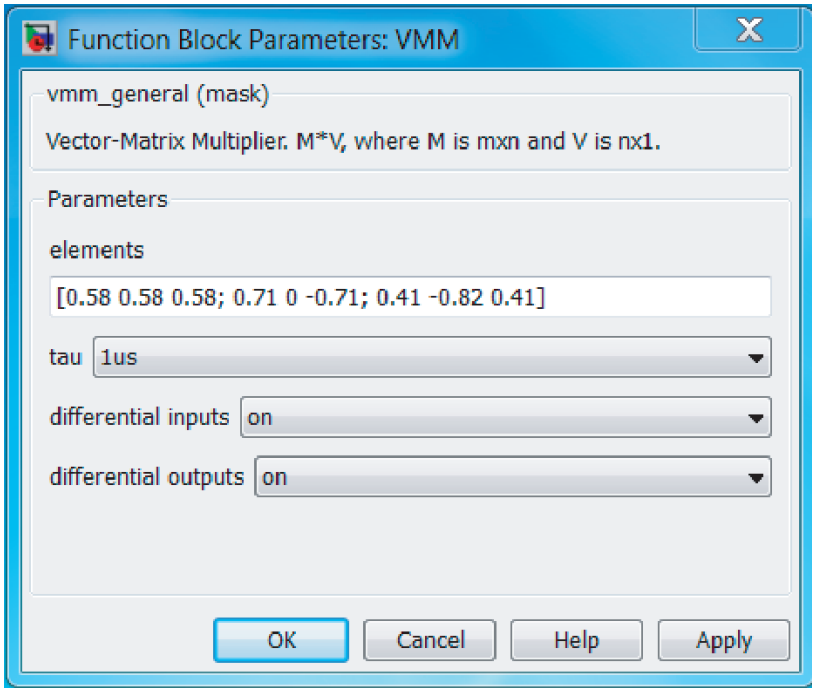
To make it as easy as possible to design FPAA systems with the VMM block, we have incorporated it into the Simulink compiling tool, Sim2Spice. This tool allows engineers to design analog signal processing systems at the block level in Simulink, then compile that design onto the FPAA [44]. A more thorough discussion of the Sim2Spice tool is the topic of Chapter 6.

By adding a block to the Simulink library, the main objective is to abstract the design to as simple as allowable. The Simulink tool compiles down to a netlist, so full transistor-level simulation can be done in SPICE. The Simulink model will capture the important signal attributes without bogging down the simulation time. Certain design parameters are abstracted and presented to the user in a fashion that is intuitive. Figure 55 shows the block design of a system in Simulink using the VMM and the GUI dialog box that corresponds to the VMM. A graphical interface (the RAT) is provided to visualize the compiled chip utilization, shown in Figure 56.

Once the VMM design is compiled from Simulink, it can be programmed and tested using the RASP Program & Evaluation (RPE) board [37]. This platform allows the user to fully test the system before embedding it into a larger system. The RPE board communicates with MATLAB on a PC via a USB connection. The board provides 40 DAC channels



(a)



(b)

Figure 55: (a) Simulink block level design for a VMM system, including supporting blocks. The Sim2Spice tool is used to compile this block diagram down to object code ready to be programmed on the FPAA. (b) The Sim2Spice parameters for the VMM block. Rather than overwhelm the user with circuit specifics, the functionality has been abstracted for high-level system design.

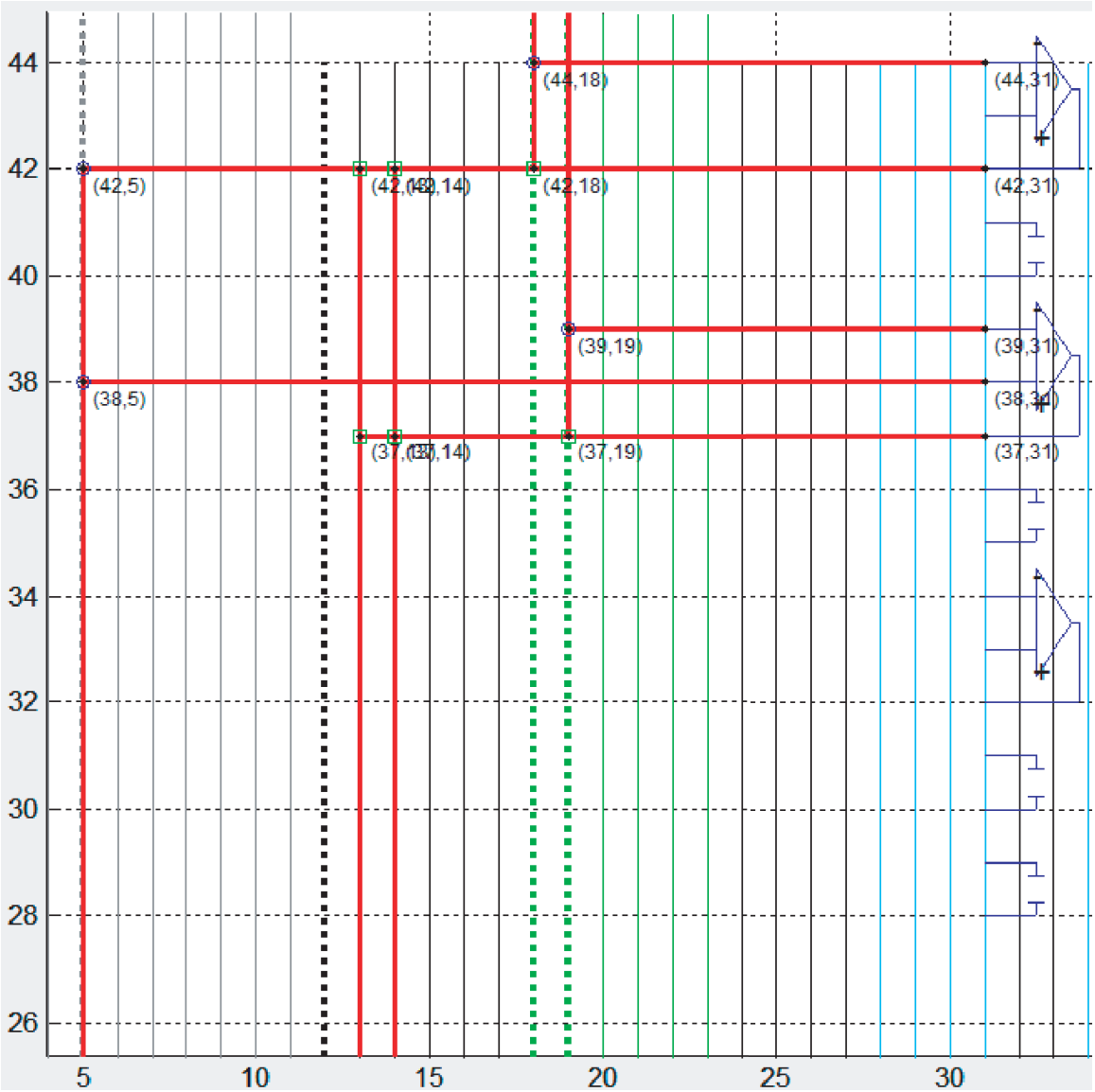


Figure 56: VMM visualization with the RAT tool. This tool allows the user to view and modify the switches that will be programmed on the FPAA.

and 12 ADC channels for testbenching systems.

5.3.3 Supporting Blocks

In addition to the core VMM block, it is helpful to have several supporting blocks available in the design library. Since the VMM is a current-in/current-out system, we have created several signal conversion blocks to transform the input/output into the voltage domain.

The V/I block is shown in Figure 57a. The design for this block was highly motivated by the CAB elements in modern FPAAs; this particular design only needs 1 wide-linear-range OTA. The expanded linear range is a result of the capacitive attenuation on the input. Other V/I converters are certainly possible, such as the differential pair discussed in Section 5.2.3, which is good for designing for temperature.

The I/V block is shown in Figure 57b. Again, this design was highly motivated by the CAB elements, relying on OTAs. Here we display the output characteristics of a transimpedance amplifier (TIA). The TIA has the benefit of being able to convert bi-directional currents, although this means that the two differential output currents of the VMM must first be combined with a current mirror.

As a final example of Simulink supporting blocks, the voltage reference is shown in Figure 57c. This is implemented with a single FG-input OTA with negative feedback. By programming a different ratio of currents on the inputs, we can effectively create a fixed output voltage on the OTA. This block is useful for setting all of the references, without burning any I/O pins, which are valuable for large dimension VMM computation.

5.4 Conclusion

We have presented a thorough design methodology for implementing a vector-matrix multiplier on a field-programmable analog array. We first highlighted the power of analog signal processing and FPAAs in particular. FPAAs provide an ideal framework for ultra low power in embedded system design. We next described the structure of the analog VMM and discussed some of the topology trade-offs. The trade-offs became clear in the

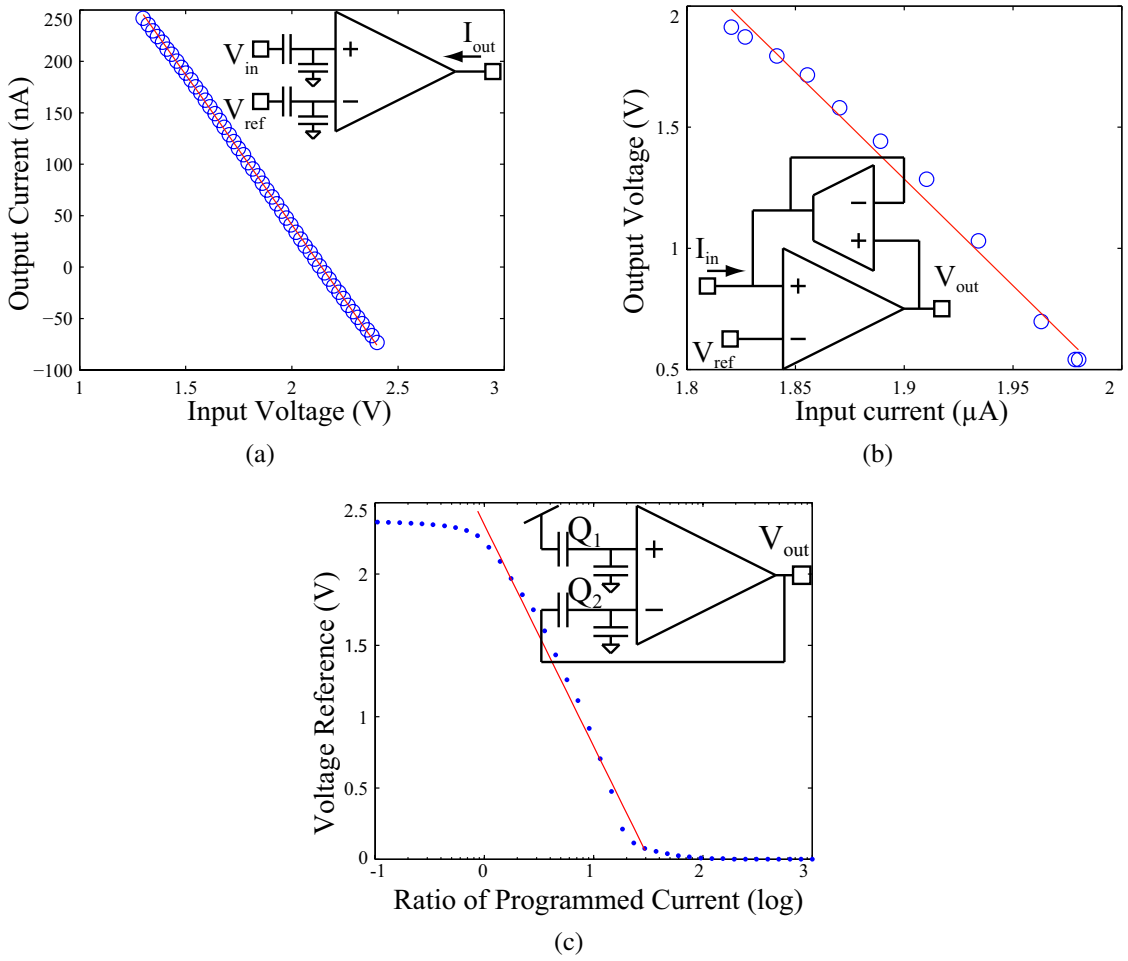


Figure 57: Supporting blocks for the VMM. (a) Input stage V-to-I converter, (b) output stage I-to-V converter, and (c) programmable voltage reference. The design for these blocks was highly motivated by the available components in the FPAA, in this case OTAs.

analysis section where we had to balance the speed, power, noise, and temperature performance. The topology that we focused on proved to have extremely high computational power efficiency. Lastly, we elaborated on the practical implementation. There is an entire tool-set infrastructure available for designing analog systems in Simulink and compiling to the FPAA. We also mentioned several supporting blocks that enable more of a complete system approach.

CHAPTER 6

HIGH-LEVEL DESIGN TOOLS

With the concept of analog computing algorithms in place, the next step of the coordinated approach to analog signal processing is to design high-level *software tools*. Traditionally, engineers implementing signal processing algorithms in digital hardware rely on a large body of work and software tools to simplify the compilation. These software tools use well-developed and intuitive interfaces to allow engineers who may have little-to-no familiarity with circuit design to benefit from the advantages of dedicated hardware. The current effort to advance the FPAA hardware and algorithms has resulted in drastically larger and more complex systems that are possible. Thus, it is a necessity to have software tools comparable to the digital flow to manage such large systems.

This chapter presents *Sim2Spice*, the top-level design space for analog signal processing with the FPAA [44, 61]. This tool allows users to create systems in the Simulink environment and compile the designs down to the FPAA hardware. This capability allows the design flow for analog signal processing systems to be more in line with what a user would expect from a digital flow. Figure 58 shows how the Simulink design tool fits into the coordinated-design framework.

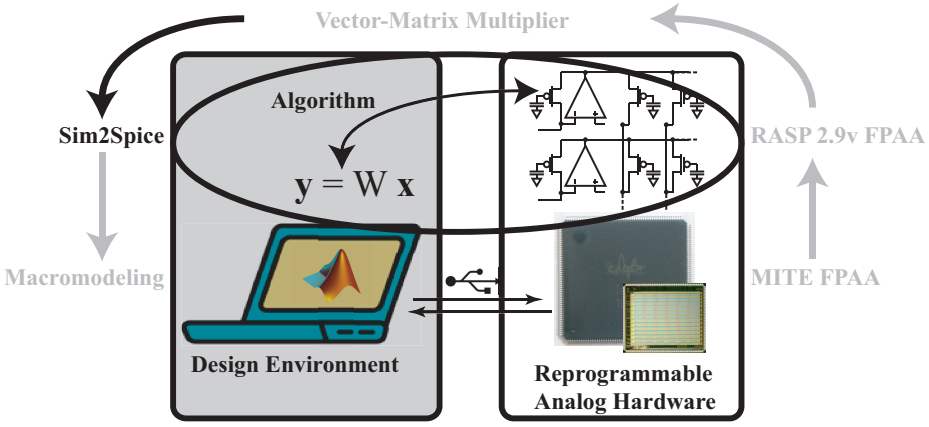


Figure 58: The coordinated approach to FPAA design: Sim2Spice.

6.1 Analog Synthesis

While early versions of FPAAs were modest enough to route circuits by hand using fuse charts, this is quickly becoming impossible. The more recent FPAAs, such as the reconfigurable analog signal processor (RASP) 2.8 [19], have fifty-thousand switches and 32 CABs, and they are only getting larger. In addition, newer FPAAs are far more robust than their predecessors and can support larger and more complicated signal processing systems. With this increase in size and complexity of FPAA systems, higher-level synthesis tools are a necessity—hand routing is no longer a reasonable option.

We have chosen Mathworks Simulink as our high-level design space because it allows for the implementation of signal processing systems in software with an intuitive graphical interface, not to mention many DSP engineers are already familiar with the program. The user connects together functional blocks and has the ability to simulate the system using a variety of simulation tools. Digital designers have already realized the power of Simulink and have developed software tools to compile Simulink block diagrams to reconfigurable digital hardware on an FPGA, as in [62] and [63]. On the analog front, there have also been tools investigated to allow support for certain FPAA designed to be done in VHDL-AMS [64]. However, we chose Simulink over AMS as our high-level design space because the graphical block-based user interface is an important complement to the text-based SPICE netlist.

Our tool, Sim2Spice, is the top-level compiler of an entire tool chain for configuring FPAAs, a diagram of which is shown in Figure 59. This tool allows users to utilize our custom library of analog signal processing blocks to create designs in Simulink, then compile that design down to a SPICE netlist. From there, the GRASPER tool is used to place-and-route the netlist to the FPAA, and the RAT tool is used to visualize and modify the routing. This complete set of tools now provides a useful interface for engineers outside the analog circuit design field to implement their signal processing systems and ideas directly in analog hardware.

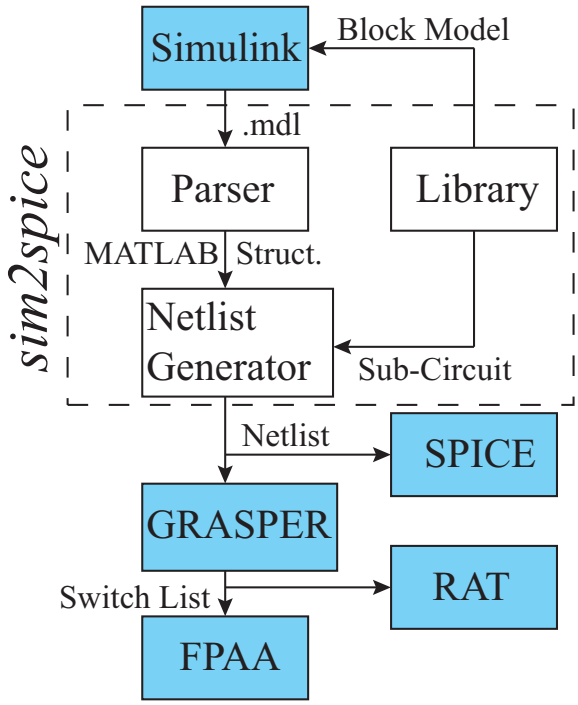


Figure 59: The complete tool set is comprised of Sim2Spice, which converts a Simulink design to a SPICE netlist, and GRASPER, which converts a SPICE netlist to a set of switches for programming on the FPAA.

6.2 From Simulink to SPICE: Sim2Spice

The Sim2Spice tool is the front end of the tool set; it takes in a model (.mdl) file from Simulink and generates a SPICE netlist, ready for further simulation or place-and-route [44, 61]. The program is composed of two main parts, the Simulink model parser and the SPICE netlist generator, which are shown along with the library in the dashed box of Figure 59.

6.2.1 Simulink Model Parser

The parsing of a Simulink model file is done in MATLAB and relies on a custom PYTHON script. The script is packaged as an executable file, allowing it to be called directly from MATLAB without the installation of PYTHON. The input to the program is a Simulink model file. An example section of a model file is shown in Figure 60, which displays the vector-matrix multiplier (VMM) and winner-take-all (WTA) blocks. The output of the parser is a MATLAB structure containing the blocks and connections (lines) that comprise the system as

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

Figure 60: An example section of a Simulink model (.mdl) file. This example shows a representation of the vector-matrix multiplier (VMM) and winner-take-all (WTA) system, which is described in Section 6.4.3.

well as all associated block parameters and values. PYTHON was chosen as the language for the parser due to the ease of parsing text with the PyParsing package [65].

6.2.2 SPICE Netlist Generator

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

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

Next, the netlist generator makes several passes over the parser's output arrays, finding and naming common circuit nets between blocks. At this point, the lines between blocks can be of a variety of forms: vectorized, single-ended, or differential. During compilation, no exhaustive DRC is performed; however, the netlist generator will check if vectorized signals are of the same dimension between blocks. We also use a color coding scheme in the blocks to match signal type, such as blue for current-mode and red for voltage-mode signals.

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

6.2.3 Component Library

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

The analog Simulink blocks are defined by Level-2 M-File S-Functions and their corresponding circuit netlist elements. We have designed the system to make it as easy as possible to add new blocks to the component library. When adding a component to the

library, there are 4 main files that must be written:

1. **S-function Simulink block.** The first step is to create the physical block as a Simulink S-function block. The number of ports and their names, as well as the input parameters, are defined here. There is also a field for the designer to write a description about the functionality of the block, which is useful for future users.

2. **Simulink (.m) behavior file.** This file defines the behavior of the block in Simulink simulations. Here, the designer describes what mathematical function the block will perform on the incoming signal. The behavior can be as detailed or as high level as the user wants. For instance, for the WTA block, the user can simply allow the block to output a high or low corresponding to the "winner" or "losers," or try to accurately describe all of the transistor subtleties. Since a true transistor level simulation can be done in SPICE, in the essence of design and simulation time, it is recommended that this block be made as ideal as allowable. A further discussion on macromodeling is given in Section 6.2.4.

3. **MATLAB (.m) build script.** This file tells MATLAB how to build the netlist that corresponds to the block. In general, this file consists of loops that print text-based SPICE subcircuits according to the block parameters. The name of this file must be the same as the definition file, with "*build_*" appended at the beginning.

4. **Description (.desc) file.** The description file defines the list of parameters that the parser will look for when it reads the model file.

The library is comprised of over 60 different parts and is growing constantly as users continue to add new blocks. As of this writing, there are several libraries of blocks in 5 different sub-categories of functionality and abstraction, including analog signal processing blocks, basic CAB circuit elements, $G_m - C$ filters, bio elements, and neuron channel models.

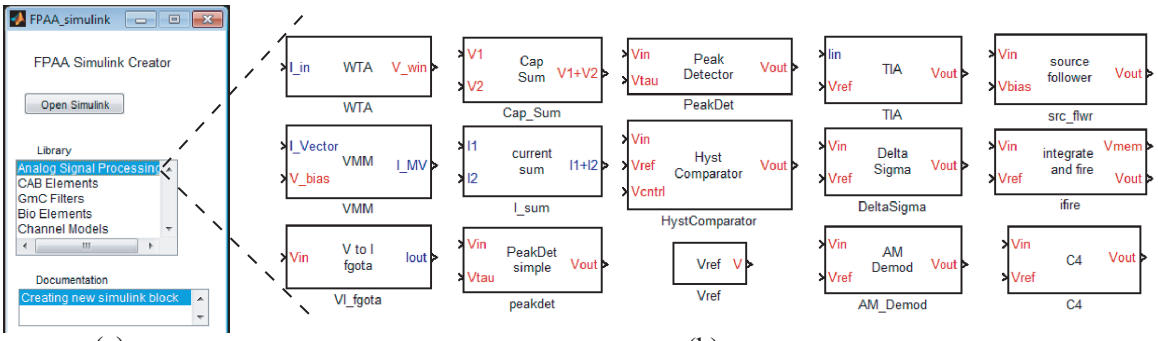


Figure 61: The Simulink component library. (a) The MATLAB GUI presents the user with all of the available component libraries. (b) Expansion of the Analog Signal Processing library. Blocks are continuously being added to the library as other users share their functional blocks.

Figure 61 shows the GUI that organizes the library into sub-libraries. In this figure, the analog signal processing sub-library is broken out to show the available components. Figure 62 shows the dialog box that appears for a specific block, the VMM, allowing the user to edit the parameters of the block. In this case, the user is allowed to edit the matrix elements, the time constant, and the signal representation. During the build cycle, the resulting VMM will consist of floating-gate transistors tiled and programmed according to the *elements* parameter and the bias current set according to the *tau* parameter. An example of a Simulink system using this VMM is demonstrated in Section 6.4.3.

One interesting aspect of building an analog component library is that it allows us to define what we consider analog signal processing blocks. This area, surprisingly, is not as defined as it could be. Although there have been books written on the area, notably [66], most analog signal processing is still done at the custom level, without the use of pre-made blocks. This gives us the freedom to invent the blocks that fit our needs with the hope that once they are built, the community will be inclined to use them as standard analog blocks.

6.2.4 Macromodeling

The need for a Simulink behavior file brings up the issue of macromodeling. Macromodeling is the design of a functional block that captures the desired performance, without being

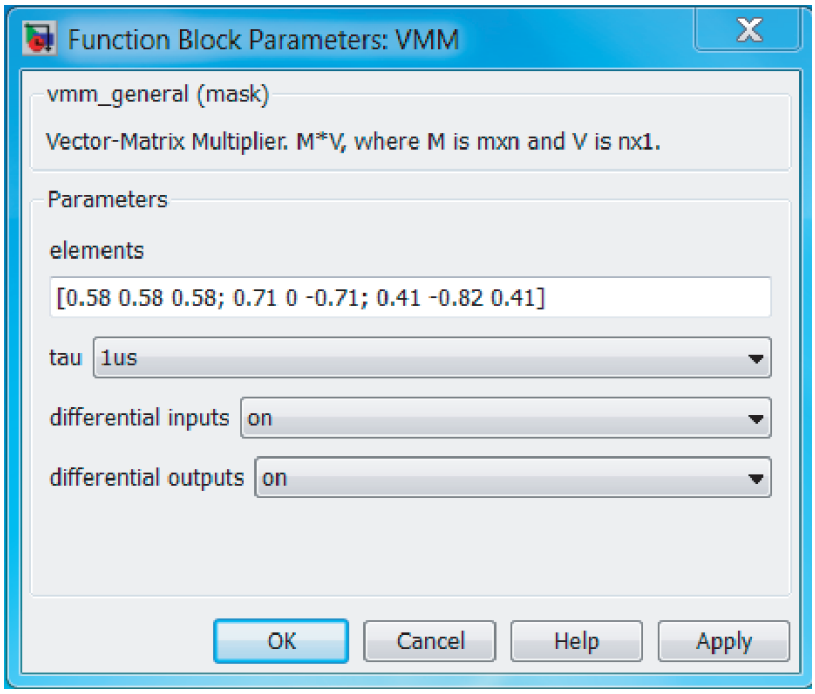


Figure 62: The block parameter window for the VMM provides a brief statement about the usage of the block, and asks the user to specify several needed parameters: the matrix elements, the desired time constant, and whether the inputs and outputs are to be single-ended or differential.

overly detailed. While there have been previous discussions on the varieties of macromodels for Op Amps [67], the same design process needs to be extended to the other analog signal processing blocks. To design large systems with the Simulink tool, we need to have blocks that are accurate enough to demonstrate that the system works, without being overly complicated that it makes the simulation time too long or muddles the results.

One example of macromodel design is for systems utilizing many operational transconductance amplifiers (OTAs). As a first pass over the system, the user would want to test the simple functionality. To test first-order functionality, as quickly as possible, the user would use the basic linear transconductance to define the OTA. This model is the quickest for computation, but will not include the important non-linearities of the OTA. To simulate the non-linearities the tanh function should be used to characterize the OTA. This model is sufficient for testing the DC characteristics containing many OTAs. Figure 63 shows the match of the tanh to a FPAA OTA sweep.

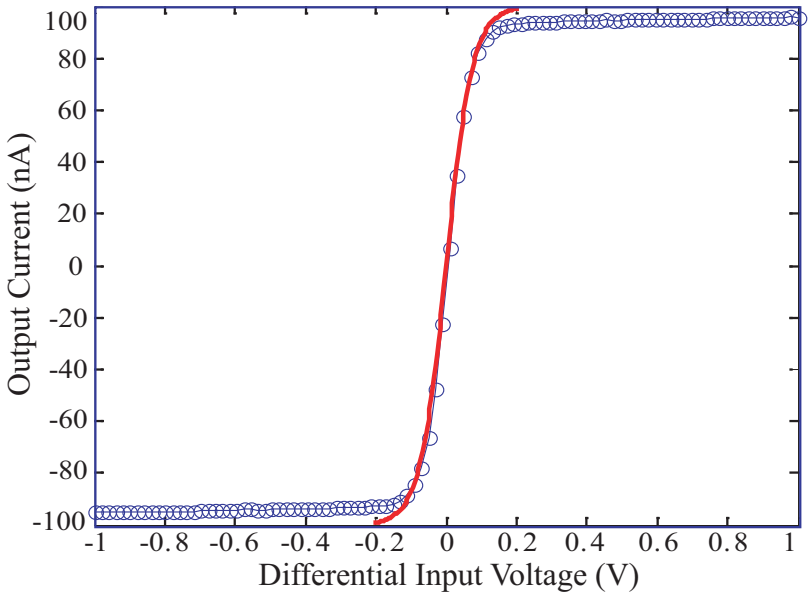


Figure 63: OTA tanh model and FPAA data. The smooth curve is a tanh function, which can be used to model the OTA. The bubbles are data taken from the FPAA hardware.

Once the general functionality of a system has been established, the models can be made more detailed to show certain design metrics. One metric important to most signal-processing systems is signal-to-noise ratio (SNR). To test the SNR, a noise component needs to be added to the blocks. For the OTA, we can easily incorporate a noise source by adding MATLAB's random number generator to the model. This noise source can be made to have the same power spectral density as the amplifier we expect to use. By using these simple models with only the features we are directly looking to test, we can get a better feel for the way the system as a whole is working, with a much shorter simulation time. A more detailed discussion on macromodeling is the topic of Chapter 7.

6.3 From SPICE to Analog Hardware

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

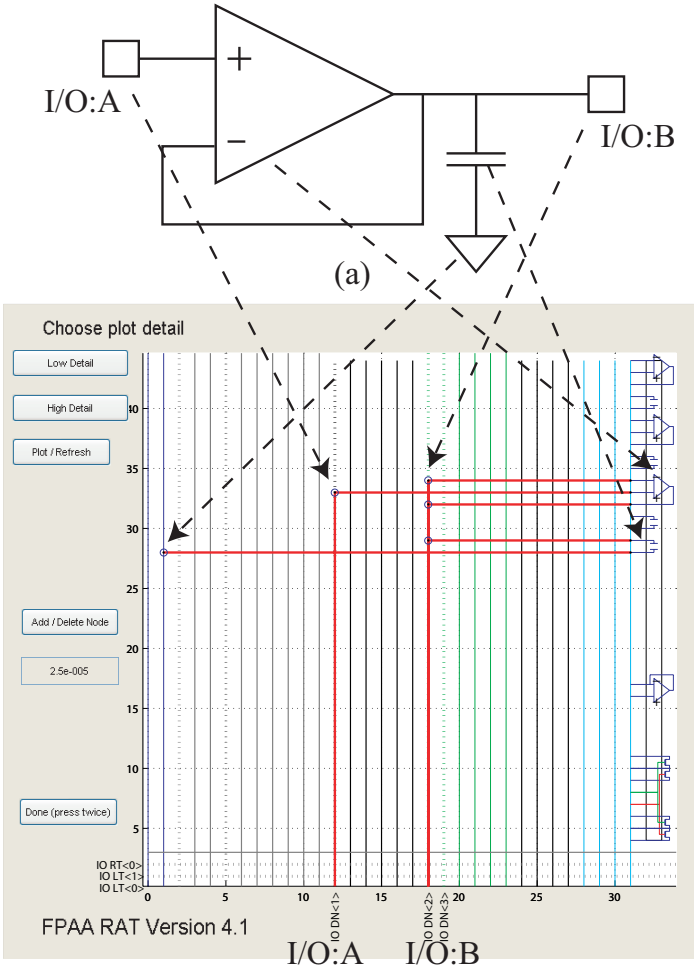
6.3.1 Place-and-Route

GRASPER, developed by Faik Baskaya, is the place-and-route tool used for targeting SPICE netlists to the FPAA [35]. The output is a list of switch addresses and the values to which they should be programmed, given in the format: (row, column, prog value). The (row, col) address refers to the desired floating gate's location in the crossbar matrix [68]. The programmed value indicates if this floating gate is intended as a switch or a computational element. A programmed value above approximately 30 μA will result in a switch that is all the way closed (we often simply use the value 1) and any value below this will describe the amount of current that the FG-pFET is programmed to pass with its source at V_{DD}. This list of switches can then be targeted directly onto the RASP family of FPAAs. In the GRASPER netlist input file, the particular FPAA is specified by the device (.dev) file. The device file describes all of the important attributes of a given FPAA, such as: number and type of horizontal and vertical lines, CAB elements, and I/O lines. By including this file, GRASPER will be compatible with future generations of FPAAs and routing structures. Figure 64 shows the flow of a circuit being mapped to the FPAA and the corresponding object code.

6.3.2 SPICE Library

Although the FPAA's CABs contain pre-defined circuits, in order for SPICE to accurately simulate a design, these CAB elements need to be implemented as subcircuits. Most of these subcircuits are straightforward one-to-one mappings, such as the MOS elements, 500 pF capacitors, and T-gates. However, several CAB elements use floating gates as programmable current sources, in particular the OTAs. For these, an ideal current source is used in the subcircuit for simulation purposes and that value is then passed to the FPAA as the floating gate target value.

One problem that arises is how to model the floating-gate elements when they are explicitly used in the circuit. Examples of this occurrence are the floating-gate input OTAs, the MITEs, and the switch elements used for computation. The problem is a result of the




```
lpf_prog =
   [28, 1, 1;
   33, 12, 1;
   29, 18, 1;
   32, 18, 1;
   34, 18, 1;
   32, 32, 1e-9];
```

(c)

Figure 64: RAT visualization of a low-pass filter. (a) The schematic of a first-order $G_m - C$ filter. This system is complied into an object code that can be programmed onto the FPAA. (b) The RAT tool displaying the connectivity of the filter. The CAB elements are illustrated on the right, with the switch matrix on the left. The nets are highlighted in red to indicate what rows and columns the switches have connected. (c) The object code for the filter consists of a list of the switches that were displayed in the RAT GUI. The first five entries in the list are fully programmed switches, and the last entry is the OTA bias current.

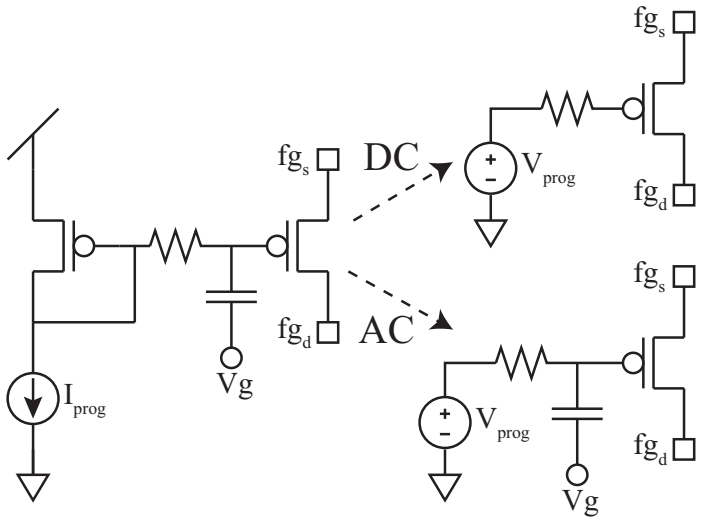


Figure 65: Floating gate SPICE model. This model is needed because SPICE cannot implement a true floating-gate transistor. The model closely resembles the indirect programming structure used on the FPAA. Under DC circumstances, the floating gate model resembles the upper equivalent circuit: a normal pFET with a fixed potential on the gate. Under AC conditions, the circuit will act as the lower circuit; the gate programming voltage will be coupled onto the floating node.

requirement in SPICE that each node needs a DC path to ground to set the operating point. Therefore, gates cannot be left floating.

One popular model is to simply place a DC voltage source on the gate through a large resistor (for instance, 10^{26} Ω). Although this achieves reasonable results, the DC voltage does not intuitively translate to the programming current. Therefore, we chose the model in Figure 65, where the target current is driven through the indirect transistor and mirrored to the in-circuit device. This model was chosen because it allows for the programming current value to be directly used in SPICE and the circuit closely resembles the actual schematic of the indirect programming system [12].

6.3.3 RAT Visualization Tool

The Routing and Analysis Tool (RAT), developed by Scott Koziol and David Abramson, provides a graphical way to view and edit the compiled circuits [37]. This visualization tool has proved to be invaluable when designing and debugging on the FPAA because it has eliminated the need to draw fuse charts. The input to the RAT is a programming (.prg)

file that includes the switch list in the form output by GRASPER. By running the command $FPAA\_RAT\_main(filename.prg)$, the GUI of Figure 64b is launched. The window shows a zoomable image of the FPAA routing structure and CAB elements. The routing lines are color coded by type and the I/O ports are clearly labeled. The switches from the input list appear as large black dots connecting the corresponding horizontal and vertical lines, and if a particular switch is used as a computational element, it is shown with a green circle around it. The lines connecting elements are highlighted in red to easily follow the connectivity of a particular net.

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

6.3.4 Program & Evaluation Board

The custom four-layer PCB in Figure 66 was built to program, communicate with, and test the RASP family of FPAAs. This evaluation board communicates over and is fully powered by USB. Additionally, the board has the capability to be powered by a 5V DC supply and communicate over a serial connection. The board is controlled by an ATMEL ARM7 microcontroller for handling instructions from the computer using MATLAB commands. It also includes a 40-channel 14-bit DAC, a 4-channel 8-bit ADC, audio input/output amplifiers and jacks, and all of the programming circuitry not already on chip. For maximum control and flexibility, almost every signal is pinned out to a header: all 52 FPAA I/O (4 to SMA connectors), the 40 DAC channels, 4 ADC channels, and many of the microcontroller and programming lines. The analog V_{DD} plane is jumpered so power measurements can be taken.

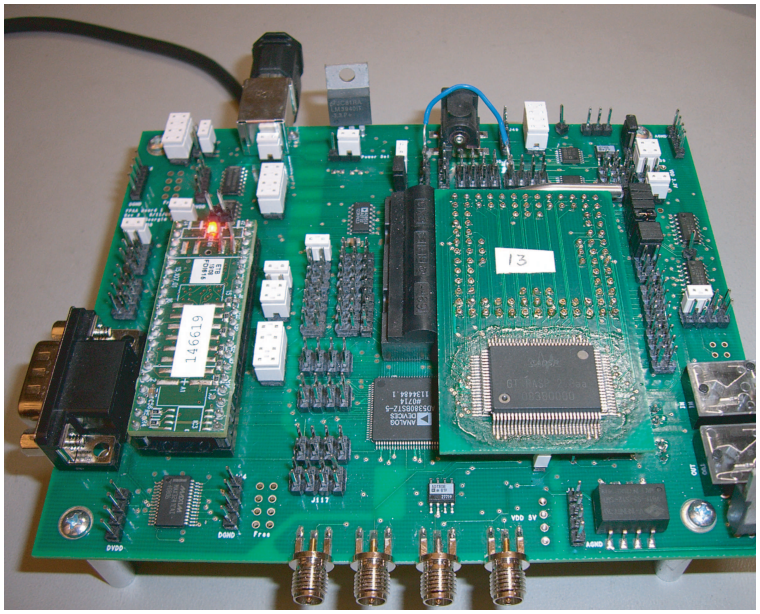


Figure 66: The Program & Evaluation board. This board communicates with MATLAB over a USB connection. The board contains a microcontroller for controlling the programming algorithms, a 40-channel DAC and a 4-channel ADC for creating and reading test signals, and all necessary power management circuitry.

6.3.5 Current FPAA Chips

The bottom level of the tool chain, and really the heart of the system, is the FPAA itself. The most recent and advanced line of FPAAs is the RASP 2.8, which was designed in a 350 nm double-poly CMOS process. This FPAA offers several drastic improvements over its predecessors [69].

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

Another advancement is the movement of most of the programming infrastructure on-chip [22]. By moving the control DACs, current measurement systems, and the ADCs on chip, we have seen a drastic speed up in programming time and an increase in accuracy.

The programming time is down to 50 ms for full-accuracy analog switches, which is a considerable speed up from 500 ms reported in [36]. The increase in accuracy and dynamic range comes from including a log-amplifier in the current-to-voltage conversion. This amplifier expands the lower range of the measurements to sub-picoamps, whereas the lower limit was around 100 pA for off-chip measurements. The on-chip migration has also allowed the form factor of the entire system to shrink. Whereas the previous system was the size of a shoe box and contained three separate boards [70], with the on-chip programming we only need the single 4.6 in $\times$ 5.6 in board discussed in the previous section.

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

1. The general purpose FPAA: RASP 2.8a [20]. This is the most commonly used FPAA and contains common analog building blocks in its CABs: OTAs, n/pFETs, capacitors, T-gates, floating-gate elements, and Gilbert multipliers.

2. The bio-FPAA [71]: RASP 2.8b. The bio FPAA contains neuron and synapse models in its CABs. By using the reconfigurable nature of the FPAA, the neurons and synapses can be arranged into computational neural networks.

3. The sensor-FPAA [72]: RASP 2.8c. This FPAA contains a specialized sensor interface and capacitive sensor CAB elements.

4. The MITE FPAA [23]: RASP 2.8d. This FPAA contains multiple-input translinear elements (MITEs) as its computational primitive. The CABs are made up of translinear loops of MITEs, which can perform various mathematical operations. A more detailed discussion on the MITE FPAA is the topic of Chapter 3.

Each chip in the RASP 2.8 line is compatible with the compiler tools because they contain the same routing architecture. However, each chip will require specific library

blocks because the CAB elements are different. Some blocks can be universal; for instance the $G_m - C$ filter block can be compiled to any chip with OTAs and capacitors. But library blocks designed for a primitive that is unique to a specific FPAA will need to be targeted to that chip. For instance, a block designed specifically for synthesis to translinear loops will need to be targeted to the MITE FPAA, or modified for the discrete MITEs in the general purpose FPAA.

6.4 Example Systems

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

6.4.1 Low-Pass Filter

One simple example system to demonstrate the interaction between Simulink, SPICE, and the FPAA is a low-pass filter block. This particular filter is a first-order $G_m - C$ block and the schematic was previously shown in Figure 64a. The transfer function for the filter is given as

$$\frac{V_{out}}{V_{in}} = \frac{1}{\tau s + 1}, \tag{47}$$

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

$$G_m = \frac{\kappa I_{bias}}{2U_T} \tag{48}$$

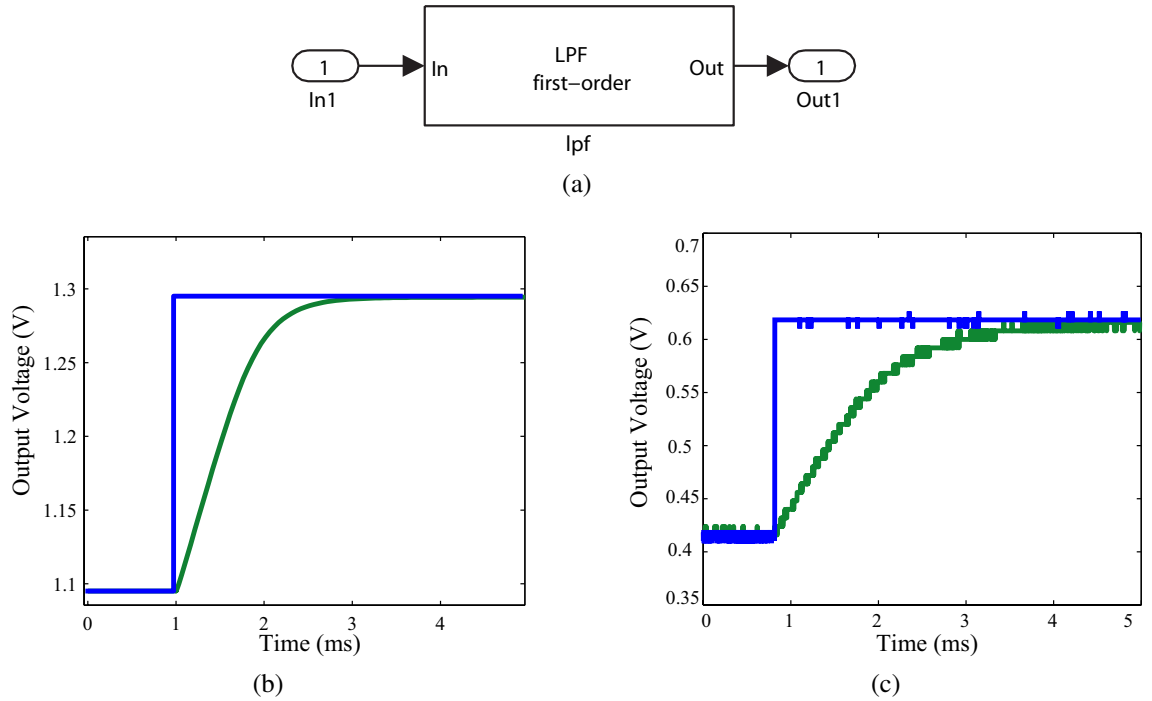


Figure 67: (a) Simulink model of the low-pass filter. (b) Step response in SPICE simulation. (c) Step response on FPAA.

is the linear transconductance gain of the amplifier. To abstract the design, the user is only asked to specify the time constant in the parameter box. By using a fixed-capacitor design, the block will translate this time constant into the gain of the amplifier. In the resulting netlist, the gain is set by the bias current of the OTA, as described in Equation 48. This process illustrates the way non-analog circuit designers can take advantage of the tool: they specify a filter of a certain order and its poles, and the compiler will create amplifiers with the appropriate bias currents. The process of parameter abstraction detailed here can be extended to any block. Figure 67 shows the Simulink block diagram, the SPICE-level step response simulation, and the real FPAA step response.

6.4.2 Computational Neuron Systems

The spiking Hodgkin-Huxley type neuron block is an example system that demonstrates the Simulink design, compilation to SPICE netlist, and the targeting of a special purpose

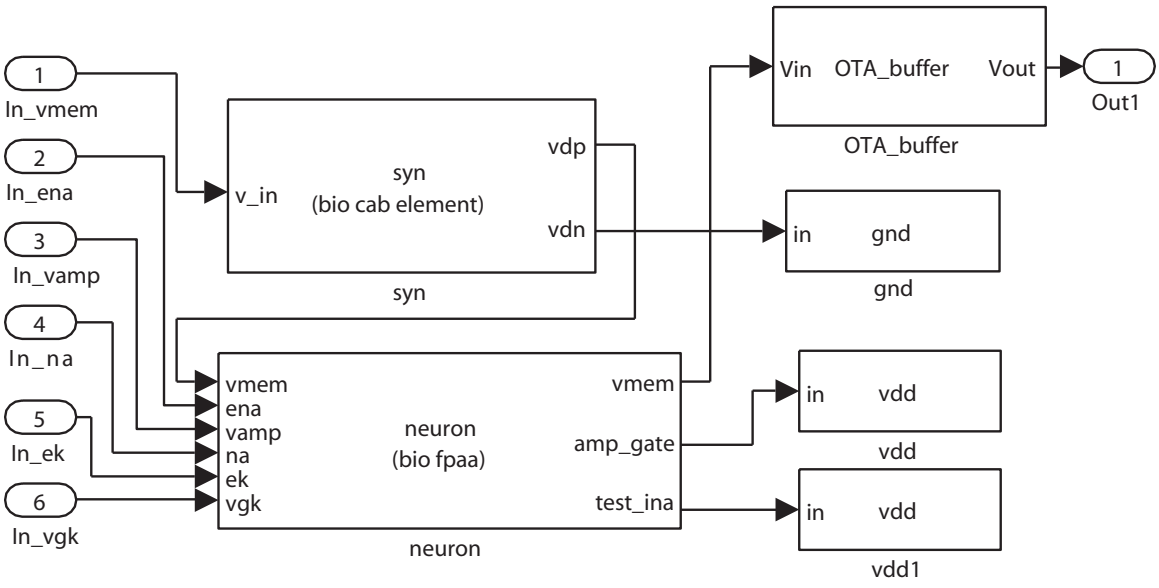
neuromorphic FPAA (the RASP 2.8b). The circuit for this spiking neuron follows the Farquhar model in [73]. The Simulink block diagram of the neuron block and necessary input and output blocks, such as an output buffer, is shown in Figure 68. Spiking data from the Simulink model can be seen in Figure 68b, and from the FPAA in Figure 68c. The neuron block worked as expected in both Simulink-level simulations and when implemented on the analog FPAA hardware.

We also compiled a system of two coupled neurons to demonstrate synchronized firing. The Simulink diagram for this system is shown in Figure 69. Two neurons are shown with one feeding into the other trough a programmable synapse. The synapse is a floating-gate transistor that can be programmed strongly or weakly depending on the desired weight. The output of this coupled system is shown in Figure 69b. The two neurons are shown to be in sync, with the first one coupling onto the second.

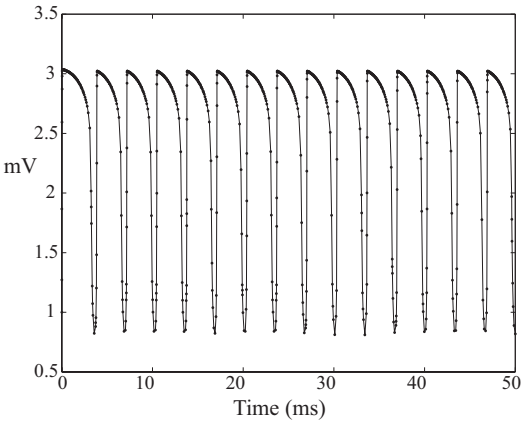
6.4.3 VMM-WTA

The third example system is shown in Figure 70. This system consists of a vector-matrix multiplier (VMM) feeding into a winner-take-all (WTA). Figure 70b shows a sample 2×2 circuit that will result when the VMM block is compiled to SPICE [55]. The circuit implementation contains design fields such as the number of transistors in the array, floating-gate program values, and amplifier biases. To aid the inexperienced analog designer, these fields are all presented as functional parameters to the user. Referring back to Figure 62, the block level design choices include matrix coefficients and time constant. A more detailed discussion on the VMM system is the topic of Chapter 5.

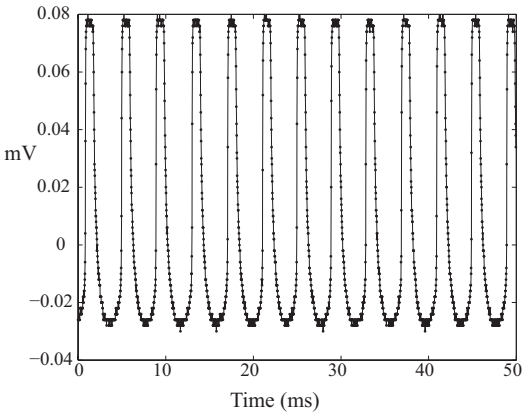
Figure 70c shows what the sample 2×2 VMM will look like after compiled by GRASPER. The diagram shows the same two OTAs that were in the previous schematic, along with the connection switches and floating-gate elements. It should be noted that one benefit of the RASP architecture is that any floating-gate elements can be synthesized right into the switch fabric [21]. This allows for much denser routing and the ability to create larger structures.



(a)

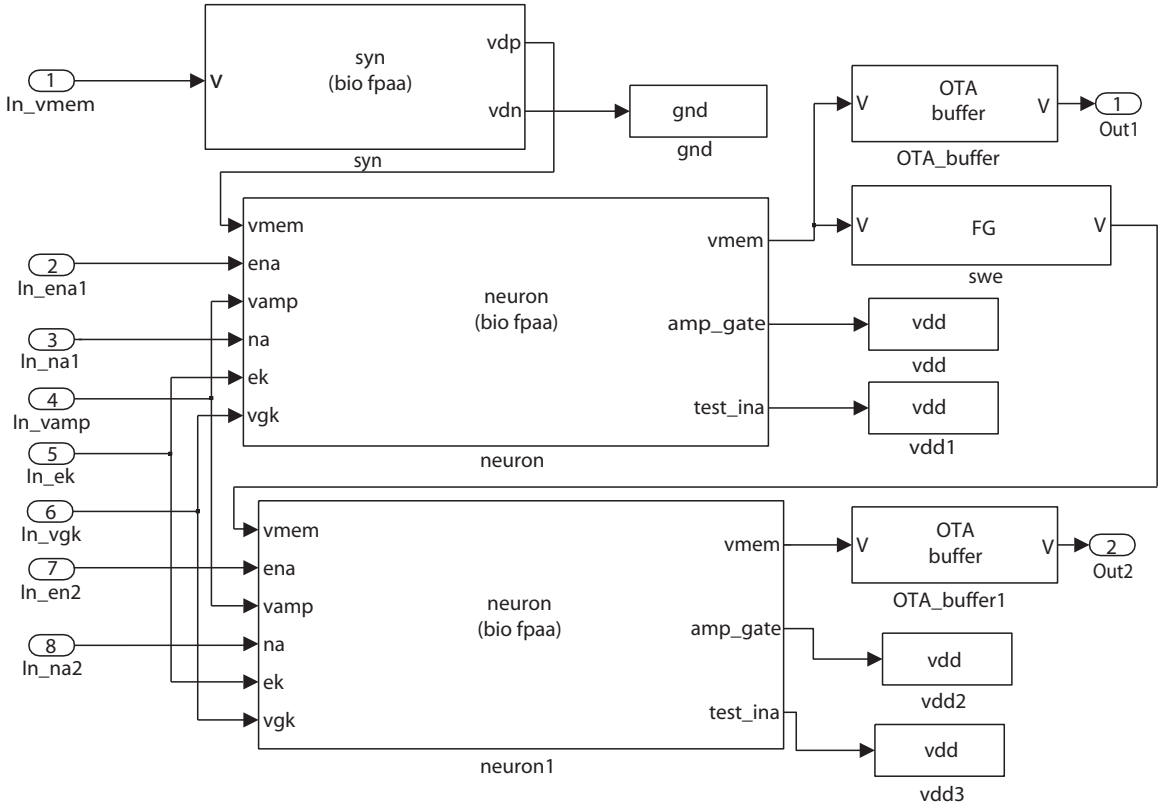


(b)

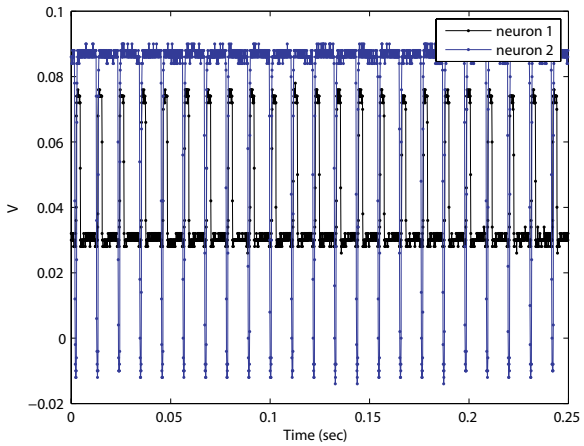


(c)

Figure 68: (a) Simulink model of the spiking neuron system. (b) Simulink simulation of the spiking. (c) Spiking as compiled on FPAA. The complete example is a spiking Hodgkin-Huxley neuron, input block, and output buffer.



(a)



(b)

Figure 69: (a) Simulink model of two neurons and a synapse. (b) Spiking activity of the two neurons. The pair of neurons demonstrates synchronized spiking due to the coupling through the synapse circuit element.

The results from the VMM-WTA system are shown in Figure 71, where (a)–(c) are from the Simulink simulation and (d)–(f) are real data from the RASP 2.8a FPAA. The three plots from each platform correspond to the output at the scopes in Figure 70a.

The two rows of output plots in Figure 71 match almost identically. The input vector (first column of plots) is designed in such a way that each element input has a time being the largest. After the matrix multiplication with an identity matrix (the middle column), the magnitudes are preserved. The output after the winner-take-all (last column) correctly determines which channel was the largest.

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

6.5 Conclusion

We have developed Sim2Spice, a configuration a tool that allows for the conversion of signal processing systems defined as interconnected blocks in MATLAB Simulink to a SPICE circuit netlist. This netlist can then be compiled to a targeting code for reconfigurable switches in an FPAA with an existing tool called GRASPER. A user can quickly and effectively compile relatively complex systems directly into analog hardware, even without a thorough understanding of analog circuit design. This tool flow opens a new world of design and rapid prototyping in analog to the wider signal processing community.

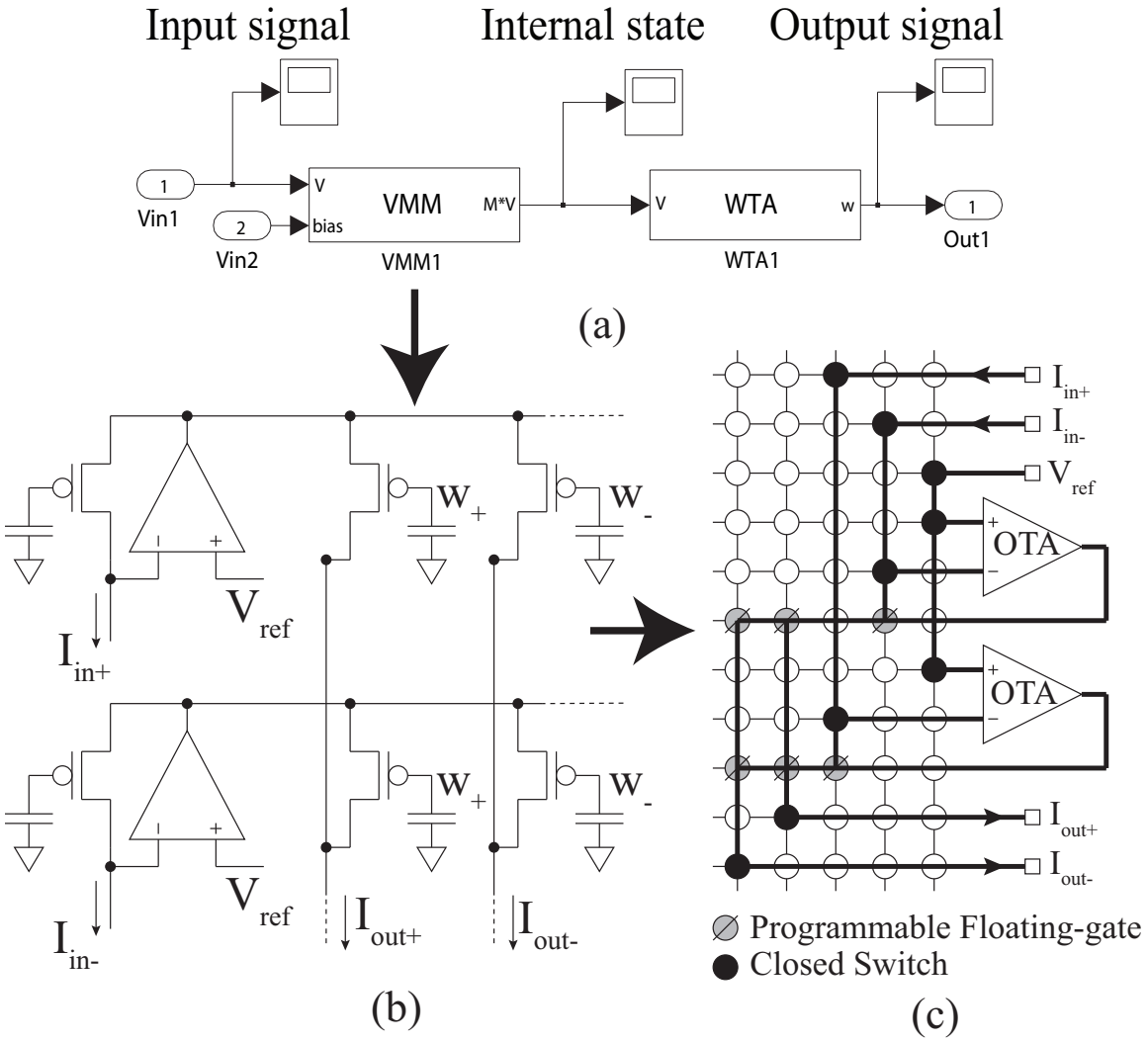


Figure 70: The three phases of implementation for the VMM-WTA system. (a) The Simulink design. Each of the two blocks, the VMM and WTA, will be compiled based on the custom analog library. The in and out ports will be compiled to I/O pins on the FPAA. The scopes in the figure are for simulation purposes and are not compiled to hardware; they represent the locations of the outputs in Figure 71. (b) A sub-block circuit to which Sim2Spice will compile the VMM block. The netlist for this circuit will be tiled in the horizontal and vertical directions according to the parameters specified by the user. (c) The corresponding switch representation to the 2×2 VMM sub-block that GRASPER will compile. This VMM uses the floating-gate routing fabric for both switches and weight storage.

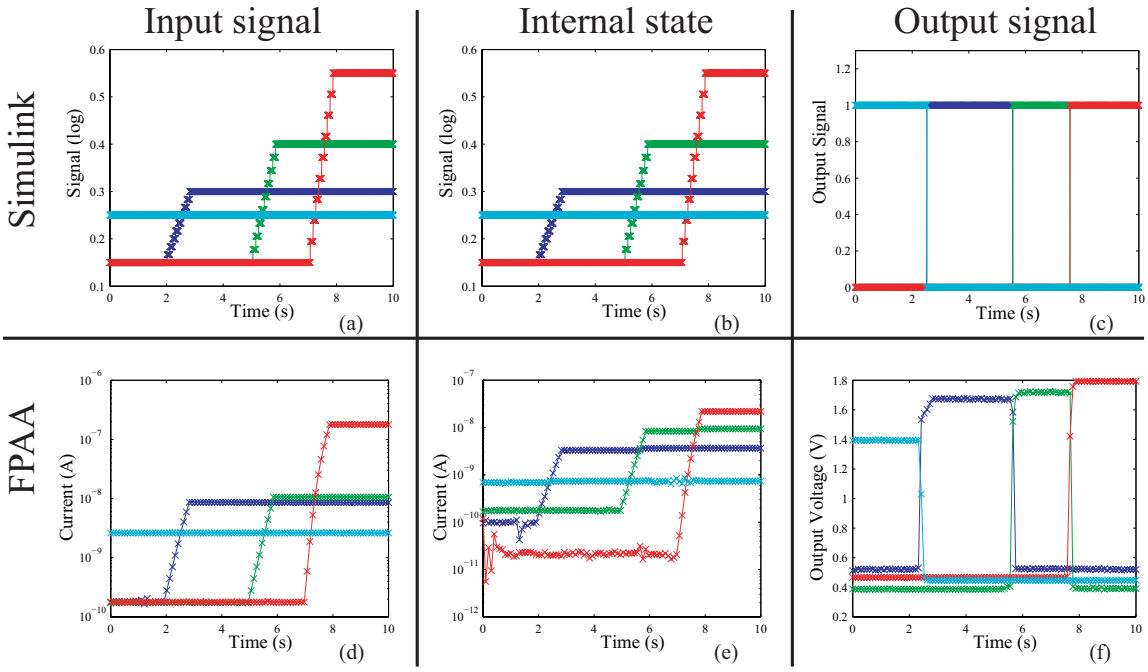


Figure 71: Output of the VMM-WTA system is compared between the Simulink response and the real hardware data from the RASP 2.8a FPAA. The first row of plots correspond to the output of the Simulink simulation, and the second row is from the FPAA. The columns represent a common state of the system between the Simulink simulation and the FPAA. The columns correspond to the scopes in Figure 70a. (a) The input vector evolves in such a way that each component has a time having the highest value. Here, the inputs are dimensionless signals. (b) The Simulink VMM matrix in this example is the identity, so the output of the VMM is equal to its input. (c) The output of the WTA shows the Boolean output corresponding to the input that is highest at a given time. (d) The input vector for the hardware VMM-WTA system is the same as the Simulink example. The input to a hardware system must have physical dimensions; in this case, we used currents. (e) As in the Simulink case, the output of the identity VMM is equal to the input. (f) The output of the hardware WTA demonstrates the same Boolean trend as shown in the simulation.

CHAPTER 7

ANALOG MACROMODELING

With the Sim2Spice Simulink tool providing the top-level FPAA design environment, the final step of the coordinated approach to analog signal processing is to design reliable *analog models* to populate the component library. The development of a high-level framework for abstracting analog design and creating behavioral analog blocks is necessary to bridge the analog and digital design gap for the system engineer. There is currently a body of work that is concerned with the automated synthesis and modeling of analog circuits for mixed-signal systems [74, 75, 76]. These tools tend to use mathematical techniques to decompose each element in an analog system to create a linear model for simulating the system block. Often, the main motivation is simply modeling the entire netlist in a way that is practical for a general CPU to process. This automated modeling gets very complex when the nonlinear dynamics are considered [77, 78, 79]. What is severely lacking is an intuitive architecture for modeling analog systems in a way that makes system design easier for the non-expert.

This chapter presents a method for analog abstraction and macromodeling [80, 81]. We take a holistic look at the signal-processing function being performed and add the analog non-idealities only as needed. This creates a design environment that has much higher

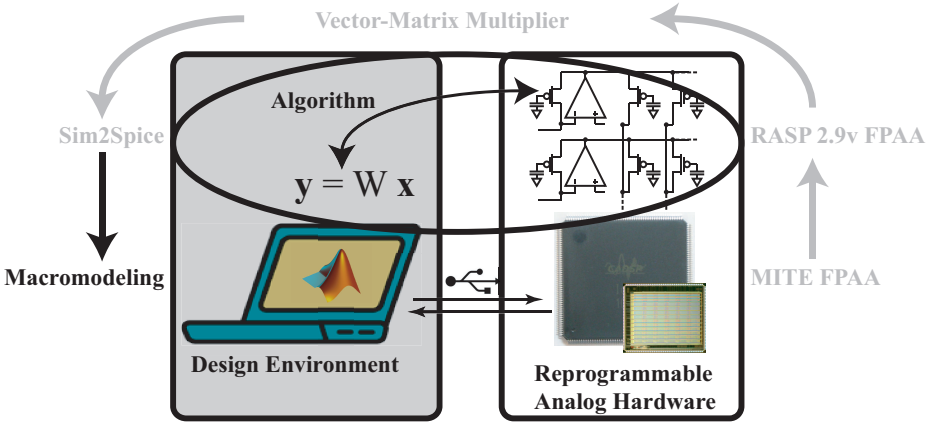


Figure 72: The coordinated approach to FPAA design: Macromodeling.

clarity for the design engineer, as well as greatly reducing the resources required for system simulation. Figure 72 shows how the analog macromodeling fits into the coordinated-design framework.

7.1 Basic Analog Signal Processing Blocks

When defining a library of signal-processing blocks, we find that certain functions are highly efficient to perform with analog elements. This section will describe the key set of analog processing blocks. We save thorough modeling for a later section.

7.1.1 Vector-Matrix Multiplier

The vector-matrix multiplier (VMM) is a core component in many signal processing applications [47]. Vector-matrix multiplication is commonly performed in FIR filters, 2-D block image transforms, convolution, correlation, and classification [42]. Recently, custom analog VMM cores have provided a low-power, high-throughput tool for signal processing [56]. Several orders of magnitude in efficiency can be gained by allowing the natural physics of the transistors to perform the multiply and accumulate (MAC) operations.

The analog VMM is composed of a nested set of programmable current mirrors. The weighted mirror performs the multiply, whereas the we get perfect summation by combining each mirror's output current. Analog VMMs have recently demonstrated low-power solutions in such embedded systems as a transform imager and an OFDM receiver [43, 54]. A more detailed discussion on the VMM is the topic of Chapter 5.

7.1.2 Band-Pass Filter

Filtering is an important application for analog signal processing. For instance, spectral decomposition is the front-end step for many low-power sensor networks [82].

The capacitively coupled current conveyor (C^4) filter is a programmable, continuous-time band-pass filter that is power efficient and can cover a wide range of frequencies [83]. This filter element can be used as a basic second-order filter, or can be cascaded to create

higher-order filters.

The C^4 filter is designed such that both time constants can be set using transistor currents. This is especially useful for the FPAA since we have an abundance of programmable current sources—potentially every element in the switch matrix. The C^4 filter addresses the common problems with Gm-C filters such as limited linearity, large overhead of tuning circuitry, and offsets due to mismatch.

7.1.3 Winner-Take-All

The winner-take-all (WTA) is an important element in classification systems. The WTA works by taking an input vector and finding the channel with the largest magnitude. The block produces an output vector of the same dimension that is filled will all low values, with the exception of a single high value. This single high value corresponds to the element of the largest input.

The WTA is a computational block that is highly power efficient when implemented with analog elements. As described in [84], an N-input WTA can be constructed with a modified N-input differential structure. Each branch consists of two transistors (and an output buffer, if desired) and picks the largest current out of an input vector. This structure is based on the nonlinear inhibition of neural systems.

7.2 Analog Abstraction Concepts

In this section, we describe several of the high-level design choices that were made in creating the CADSP framework. From choosing a top-level design space, to constraining the interface between blocks, a well-planned framework facilitates the design of large-scale systems.

7.2.1 High-Level Analog Design With Simulink

Simulink is used as the top-level design space for analog signal processing in the RASP FPAA [44]. The use of Simulink was important to us because it is already a familiar tool

to many DSP and control-system engineers. The intuitive nature of high-level blocks with wires in between makes it easy to design at the system-level. The Simulink tool has proven to be an intuitive interface for graphical analog design and has been used extensively in a graduate-level analog system design course [85]. A more detailed discussion on the Simulink compiler is the topic of Chapter 6.

Simulink comes prepackaged with many libraries of components, yet lacks high-level analog blocks. Therefore, we needed to create our own libraries for custom ASP blocks. The tool framework allows the analog engineer to easily add new blocks to the analog libraries. The key with block design is that the system should be modeled at the behavioral level, so that it is easy for the system engineer to place a block into a larger design. The ASP libraries promote the reuse of well-tested circuits as well as the propagation of expertise.

The creation of high-level blocks introduces the question of how much abstraction is required. If large mixed-mode systems are to be simulated, we need to provide macromodels for each analog block. Macromodels serve to reduce the simulation time and may include options as to how many second-order effects to include (such as noise and distortion). Circuit abstraction also means that we should cover up the detailed circuit parameters by fixing all of the static parameters and presenting the user-defined parameters only as they relate to the system specifications.

Our approach to the level of abstraction has been to create multi-level libraries. The Level-1 library contains high-level system blocks and is the topic of most of this chapter. These blocks conform to all of the interface specifications and analog abstraction techniques discussed here. They are presented in terms of the *function* performed, rather than by circuit-level description. On the other end of the spectrum is the Level-2 library. This is the library for analog experts and is explored at the end of this chapter. In this library, we provide blocks that treat all aspects of traditional analog design, without abstraction.

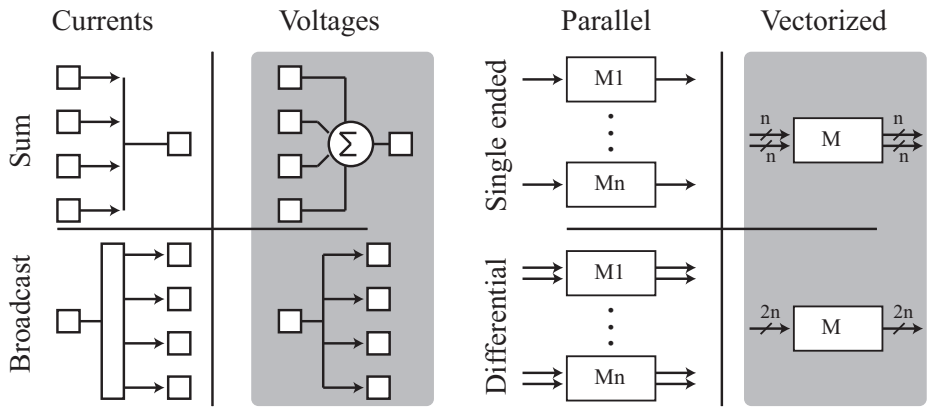


Figure 73: The system abstraction first involves defining our signal protocol. Custom analog design is free to take advantage of current-mode signals, which are easily summed. However, we constrain the analog processing tool to use only voltage-mode between blocks because it is more similar to digital design and fits into the Simulink framework. Vectorized signals are also important because they take advantage of the analog processor's parallel processing capabilities.

7.2.2 Voltage Mode Systems

The first step in making analog design feel like digital design is to define a standard protocol for the interface between blocks. Digital design benefits from a very simple convention of high and low voltages. Conversely, analog systems can propagate information by means of small, large, voltage, or current signals. In general, these operating domains create advantages for analog systems. As illustrated in Figure 73, current-mode systems can easily sum signals, while voltage-mode systems can broadcast signals to many destinations. Although each domain has its advantages, these choices are exactly what we want to abstract away so that things are easy and familiar to the digital designer.

At the expense of the current-mode system's efficient summing, we constrain the interface of our Simulink blocks to voltage-mode operation. This constraint is more like traditional digital design where a single block can fan out to many, but signals must be summed through a device, not simply shorted. We can still take full advantage of the current-mode analog processing inside the block, but the interface is exclusively voltage.

The voltage-mode design methodology has implications on the up-front design of each analog block. Many analog systems have a native current-mode interface, in which case

we will embed conversion stages. The voltage-to-current (V/I) or current-to-voltage (I/V) stages can take many forms, and the best choice will depend on the particular application or specification. Within each block, we generally characterize multiple conversion choices so that the user can select the one they want based on the performance.

7.2.3 Vectorized Signals

Frequently in DSP, and in particular when using MATLAB, the lines between blocks are vectorized. This is common in matrix operations where the inputs are all in parallel. We have incorporated this vectorized net aspect into the analog tool structure. Although a size of unity is often sufficient, each net can have any size vector dimension. Rather than forcing the user to define every size, the signals are automatically scaled based on the blocks that are used. For example, if an $M \times N$ VMM is instantiated, the input vector will automatically have a size N, and the output will have size M.

Figure 73 illustrates the use of differential mode along with single-ended vectorized lines. Often in analog design, differential signals are used to increase SNR or cancel even-order harmonics. To keep the design simple, single-ended or differential mode can be selected inside a block as a parameter without changing the complexity of the blocks in the design window.

7.2.4 Biasing

A major design element of analog systems is the proper biasing of the blocks. This is a concept that is not manifest in digital design, and therefore must be dealt with behind the scenes.

The RASP line of FPAAs is built in a network of floating-gate switch elements. This element is a very useful, as it can also store bias values for computation (one of the reasons such high computational density is achieved). The analog designer can store the FG bias values inside the block without necessitating input for the end user. Often though, the bias value is derived from a parameter in the system's function. For instance, in an OTA-C filter,

the time constant is given by a C/G_m relation. These hardware mappings can be written into the block, so that the user only needs to specify the time constant, and the correct bias will be programmed.

One of the benefits of this approach is that we can abstract away "traditional" analog design choices. For instance, by specifying a programmable bias in the OTA stage, we do not have to explicitly design for the element's output impedance. This abstraction is possible because it is the *gain* itself that we can target and program. This approach is important for thinking about analog design at a functional level and makes the system specification feel more like digital design.

7.3 Analog Modeling Techniques

To have reliable Simulink simulations, we must accurately model each analog block. Analog system modeling can be approached at different levels, depending on the desired characteristic. Here we derive the expressions for three common analog characteristics: nonlinearities, noise, and the conversion stage transfer function. This section will define the methods we will use to model analog blocks, whereas other specific blocks will be modeled in Section 7.4.

7.3.1 Nonlinearities

Electronic devices are inherently nonlinear. We will frame our discussion around the MOS device operating in the subthreshold regime, since that is where ultra low currents are achieved. The analysis could be expanded to above threshold, but most ASP systems operate in subthreshold for the power efficiency. The nonlinearity is seen clearly in the drain current [86]:

$$I_d = I_0 e^{\left[\kappa\left(V_g - V_{T0}\right) - V_s + \sigma V_d\right]/U_T}. \tag{49}$$

Here $U_T = kT/q$ is the thermal voltage, V_{T0} is the threshold voltage, κ is the inverse subthreshold slope, I_0 is a device-dependent pre-exponential term, and σ is the DIBL parameter that models subthreshold current vs. drain voltage changes. To model this nonlinearity, we will generally use the expansion:

$$e^x - 1 \rightarrow x + \frac{x^2}{2} + \frac{x^3}{6} + O\left(x^4\right). \tag{50}$$

Other common analog nonlinear functions are the hyperbolic tangent and hyperbolic sine [87]. These odd nonlinear functions appear when the current is measured as a function of the input voltage and output voltage of common transconductors, respectively. We will use the following expansions to approximate the hyperbolic functions:

$$\tanh(x) \rightarrow x - \frac{x^3}{3} + O\left(x^5\right), \tag{51}$$

$$\sinh(x) \rightarrow x + \frac{x^3}{6} + O\left(x^5\right). \tag{52}$$

These linearization techniques are illustrated in the analysis of the dynamics of the OTA-C first order filter, shown in Figure 74a. Here, the current summed on the output capacitor is

$$C\frac{dV_{out}}{dt} = I_{bias}\tanh\left[\frac{\alpha\kappa}{2U_T}\left(V_{in}(t) - V_{out}(t)\right)\right]. \tag{53}$$

This equation is easily non-dimensionalized to the common form

$$\tau dy/dt = \tanh(x - y), \tag{54}$$

where $x = \alpha\kappa V_{in}(t) / (2U_T)$, $y = \alpha\kappa V_{out}(t) / (2U_T)$, and $\tau = 2CU_T / (\alpha\kappa I_{bias})$.

We can use the expansion in Equation 51 to obtain

$$\tau\dot{y} = (x - y) - \frac{1}{3}(x - y)^3. \tag{55}$$

The expansion is useful not only to help us see the harmonic pattern, but it reduces the computation when the nonlinearity is small. If we assume that $(x - y)$ is small, then we can

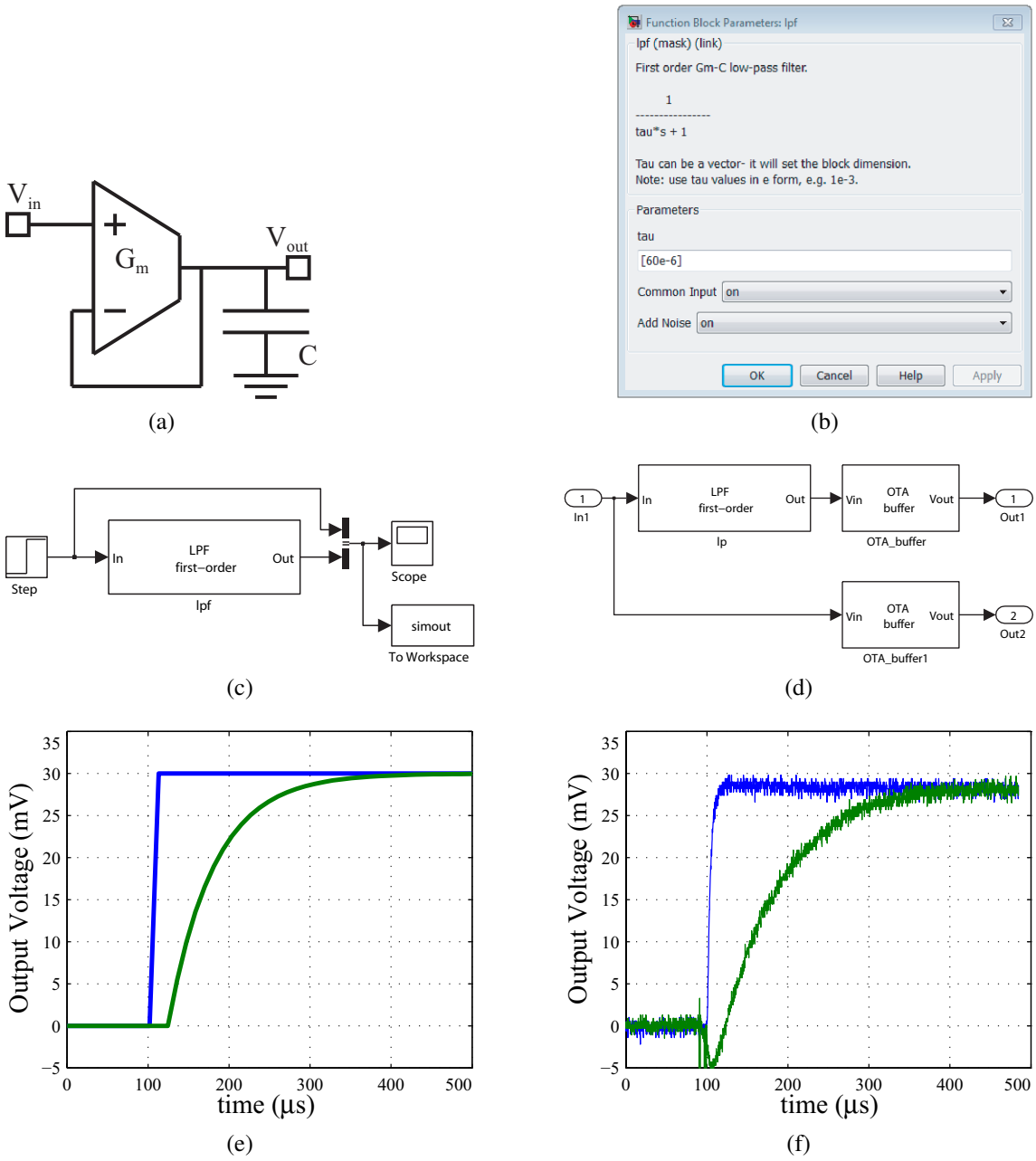


Figure 74: Design and simulation of the LPF Simulink block. (a) The basic OTA-C implementation of the LPF block. (d) The parameter box asks for a time constant for the first-order response. (c) The testbench required for Simulink simulation. (d) The adaptation of the testbench for programming the FPAA. The input signal is also buffered to an output pin to trigger the oscilloscope. The buffers are included so we have a predictable amount of capacitance at the filter. (e) The Simulink simulation uses the time constant in an ideal exponential function. (f) The actual step response from the FPAA closely resembles the simulation. The time constant was evaluated as a bias current for the OTA and amount of capacitance at the output.

drop the cubic term. To meet this small requirement, a common rule is that the expression inside the tanh should be less than 0.1. When the attenuation factor (α) is unity, the resultant differential input voltage should be less than 10 mV. Here, we see one of the trade-offs of the wide-input-range OTAs: with an attenuation factor of 0.1, the input is linearized up to 100 mV, but the time constant is increased. This decrease in speed can be compensated for by an increase in I_{bias}, at the expense of power. With the constrained step sizes, we can rewrite Equation 55 as $\tau\dot{y} = (x - y)$. In voltage mode, we are left with the transfer function

$$\frac{V_{out}}{V_{in}} = \frac{1}{s\tau + 1}, \tag{56}$$

where the linearized model is used. In other filter applications, we typically use the OTA approximation

$$I_{out} = G_m \left(V_1 - V_2\right), \tag{57}$$

where $G_m = I_b\alpha\kappa/\left(2U_T\right)$, and therefore time constants are in the form C/G_m. For large steps, the tanh function will saturate and can be approximated as a signum. This slewing condition can be written as $\tau\dot{y} = \mathrm{sgn}\,(x - y)$, where the output's rate of change is no longer controlled by the input.

Another common circuit nonlinearity is in the form of a sinh, exemplified by the dynamics of the class A/B output stage in Figure 75. For this system, KCL at the output gives

$$C\frac{dV_o}{dt} = I_p(t) - I_n(t), \tag{58}$$

where, with subthreshold currents, it is written as

$$C\frac{dV_o}{dt} = I_{op}\exp\left[\frac{\kappa_p}{U_T}\left(V_{DD} - V_{in}\right) + \frac{\sigma}{U_T}\left(V_{DD} - V_{out}\right)\right] - I_{on}\exp\left(\frac{\kappa_n}{U_T}V_{in} + \frac{\sigma}{U_T}V_{out}\right). \tag{59}$$

To find the dynamics of this system, we note at the DC operating point $V_{in} = V_{out}$ and we define the quiescent current as I_{bias}. With a dynamic input signal, we decompose the

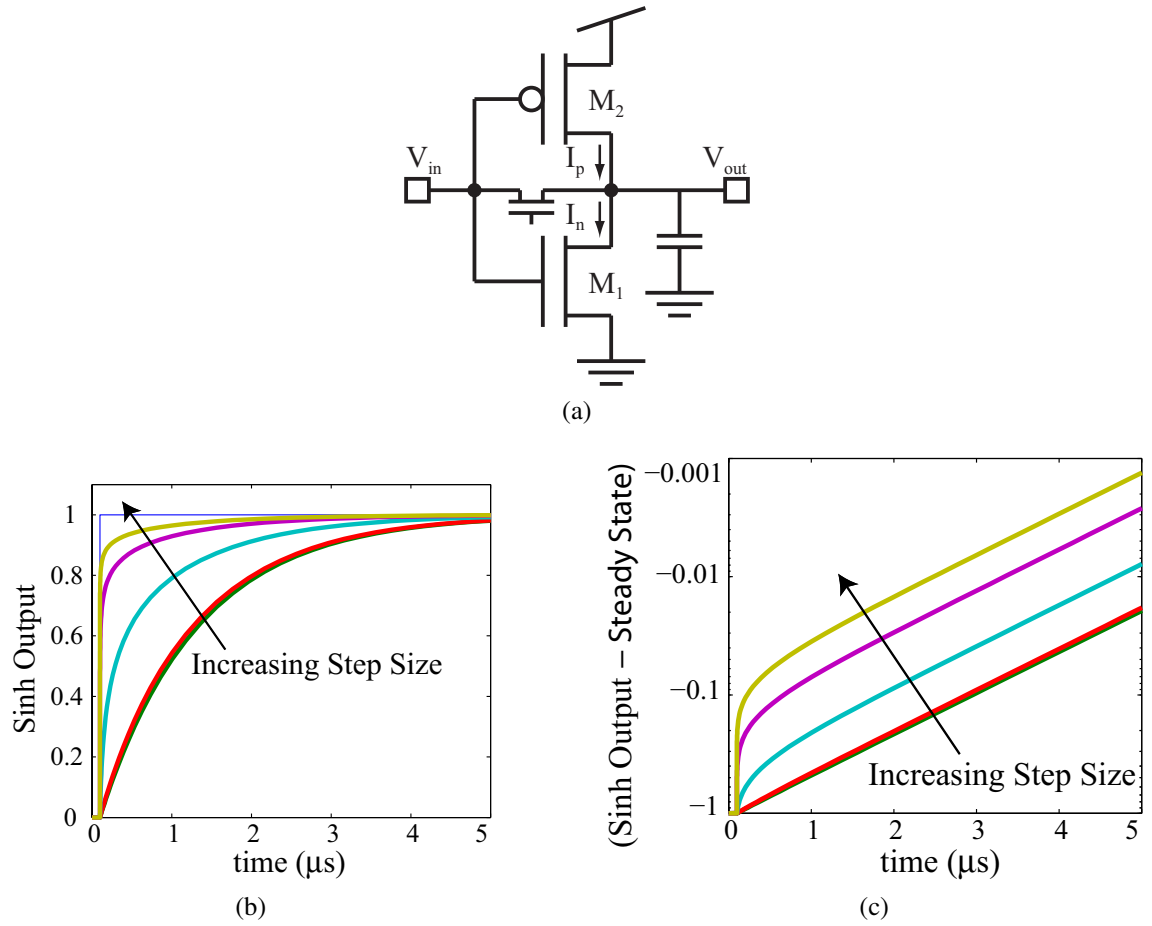


Figure 75: The dynamics of the sinh function. (a) The class A/B driving stage circuit implementation. (b) The dynamics of the sinh function are demonstrated by a step response. The simulation normalizes each output to a value of 1 (where the true response has a very large negative gain), with input steps of increasing sizes. For input steps greater than 1, the nonlinearities can be observed by the increasing response speed. (c) In log scale, the nonlinearity is apparent by the bowing off of the straight time constant line.

input and output into DC and time-varying components:

$$V_{in}(t) = V_{DC} + v_i(t), \tag{60}$$

$$V_{out}(t) = V_{DC} + v_o(t). \tag{61}$$

With the dynamic function now in the form

$$C\frac{dv_o}{dt} = I_{bias}\exp\left[-\frac{\kappa}{U_T}v_i(t) - \frac{\sigma}{U_T}v_o(t)\right] - I_{bias}\exp\left[\frac{\kappa}{U_T}v_i(t) + \frac{\sigma}{U_T}v_o(t)\right], \tag{62}$$

we can non-dimensionalize the equation with

$$x = \frac{\kappa}{U_T}v_i(t), \tag{63}$$

$$y = -\frac{\sigma}{U_T}v_o(t), \tag{64}$$

$$\frac{dy}{dt} = -\frac{\sigma}{U_T}\frac{dv_o}{dt}, \tag{65}$$

$$\tau = \frac{CU_T}{2\sigma I_{bias}}. \tag{66}$$

After plugging in these values, we are left with

$$2\tau\dot{y} = \exp(x-y) - \exp[-(x-y)], \tag{67}$$

$$\tau\dot{y} = \sinh(x-y). \tag{68}$$

Using our expansion from Equation 52, we can write this nonlinear dynamic equation as

$$\tau\dot{y} = (x-y) + (x-y)^3/6. \tag{69}$$

This form makes it easier to intuitively see how the system is acting. Again, for small inputs, we can neglect the cubic term and model the function as a simple difference of output and input. Figure 75b shows the step response for small and large inputs. For larger inputs, the output is shown to speed up and deviate from the linear time constant.

When modeling the computational elements for the Simulink library, we will utilize these linearization techniques wherever possible. However, it is prudent to be mindful of where the higher-order nonlinearities come from, so we can add options for more advanced simulation.

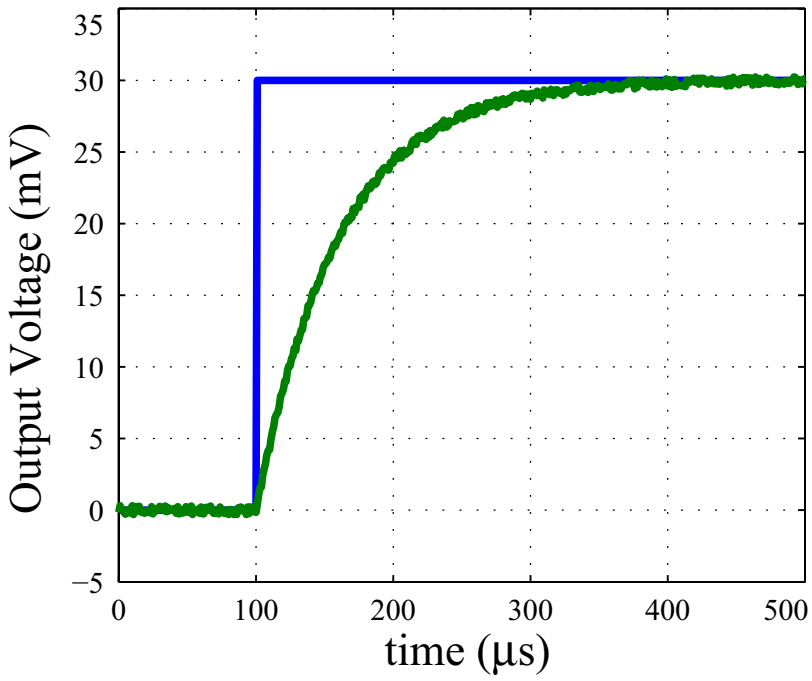


Figure 76: Simulink simulation of the first-order filter with noise enabled.

7.3.2 Noise

The performance of analog circuits is highly susceptible to noise. To include a noise component in our model, we will find the noise contribution of each element of the system and refer it to a single noise source at the input. Modern FPAAs include n/pFETs and capacitors, so we will be most interested in characterizing the channel and kTC noise.

In our first-order filter example, a noise source can be added in series with the output. The noise is modeled as kT/C, which we rewrite as qU_T/C to use the global parameters. Figure 76 shows the low-pass filter with noise enabled.

The current power of the thermal noise in a subthreshold transistor is given as [60]:

$$\hat{I}^2 = 2qI\Delta f, \tag{70}$$

where I is the DC current and Δf is the bandwidth. At small current levels, the flicker-noise current power ($KI^2\Delta f/f$) is negligible even at low frequencies due to the square term.

At this point we have enough know-how to create a model for the first-order low-pass filter. Table 8 shows the basic MATLAB code for the filter. It demonstrates the vectorized

Table 8: Macromodel of the first-order linear filter.

```
(1)   % Read in Dialog parameters
(2)   tau = block.DialogPrm(1).Data;
(3)   commonInput = block.DialogPrm(2).Data;
(4)   noiseOn = block.DialogPrm(3).Data;
(5)   signalDim = length(tau);
(6)   % Set port Dimensions
(7)   if commomInput == 1
(8)     block.InputPort(1).Dimensions = 1;
(9)   else
(10)    block.InputPort(1).Dimensions = signalDim;
(11)  end
(12)  block.OutputPort(1).Dimensions = signalDim;
(13)  % Initial states
(14)  block.ContStates.Data = block.InputPort(1).Data;
(15)  % Constants
(16)  Ut=25e-3; cap=10e-15; q=1.602e-19;
(19)  r=rand(signalDim,1)-.5;
(20)  noise=sqrt(q*Ut/cap)*r;
(21)  % Output
(22)  if noiseOn == 1
(23)    block.OutputPort(1).Data = block.ContStates.Data + noise;
(24)  else
(25)    block.OutputPort(1).Data = block.ContStates.Data;
(26)  end
(27)  % Derivatives
(28)  u = block.InputPort(1).Data;
(29)  y = block.ContStates.Data;
(30)  dy=(u-y)./tau;
(31)  block.Derivatives.Data = dy;
```

ports, the system dynamics, and the addition of noise sources.

7.3.3 Voltage-In to Voltage-Out

Many powerful analog blocks are inherently current-mode systems. To conform with the voltage-mode system protocol, we need to create interface blocks: voltage-to-current (V/I), and current-to-voltage (I/V). These interface blocks will be embedded into the system block, as illustrated in Figure 77. Here, the block can default to various converter implementations depending on the overall design. A clear example of a current-mode analog

block is the VMM.

The simplest V/I source is a single FET, which will produce a current according to Equation 49. The complement I/V is the diode-connected FET, shown in Figure 77b, which has the relation:

$$V_{out} = \frac{U_T}{\kappa} \ln\left(\frac{I_{in}}{I_0}\right). \tag{71}$$

This pair of converters is advantageous in its simplicity and works well for single-ended designs. There are three major considerations when using these blocks: (1) they are non-linear, so they are most useful when used together around a fully current-mode block; (2) the input converter is exponentially expansive and the output converter logarithmically compressive, therefore the analog block should have a large dynamic range; and (3) the currents are unidirectional.

For differential systems, we can use a differential pair in place of the single FET for the V/I stage, shown in Figure 77c. The differential current is in the form well-known from OTAs:

$$I_1 - I_2 = I_b \tanh\left[\frac{\kappa}{2U_T}(V_1 - V_2)\right]. \tag{72}$$

Assuming small differential voltage, we can linearize the tanh

$$I_1 - I_2 = I_b \frac{\kappa}{2U_T}(V_1 - V_2). \tag{73}$$

This topology has the useful feature that its bias (I_b) can be programmed independently of the system operation. This is useful because the bias current often sets the time constant of the current-mode circuit. For differential diode-connected I/V stages, we will use the convention that the output currents are small swings around the bias current: $I_{out} = I_b(1 + \Delta I_{out}/I_b)$. Therefore, the differential diodes produce

$$V_1 - V_2 = \frac{U_T}{\kappa}\left[\ln(1 + \Delta I_1/I_b) - \ln(1 + \Delta I_2/I_b)\right], \tag{74}$$

which can be shown for small $\Delta I_{out}/I_b$ to reduce to

$$V_{out1} - V_{out2} = \frac{U_T}{\kappa}\frac{1}{I_B}(I_{out1} - I_{out2}). \tag{75}$$

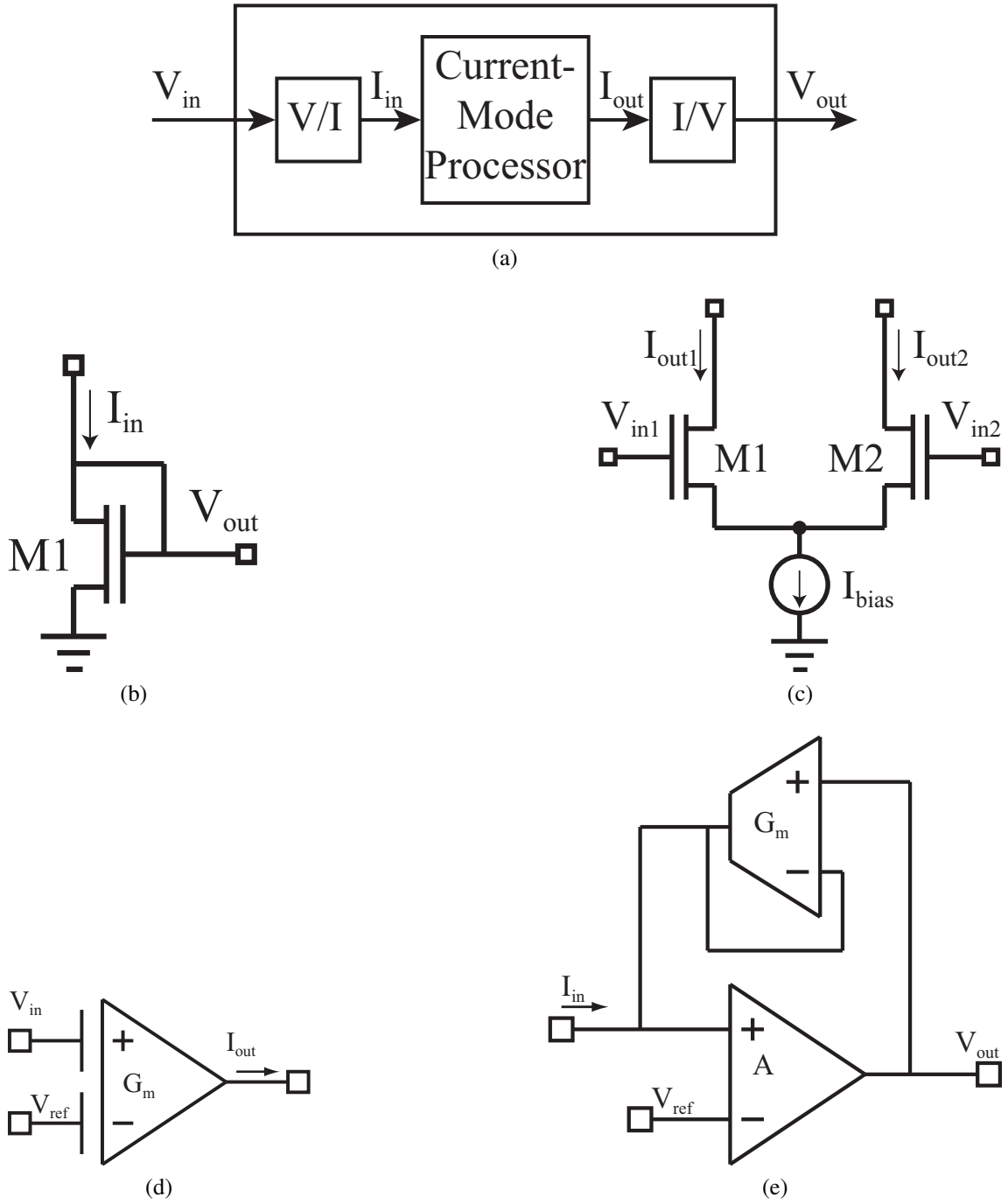


Figure 77: (a) Embedding V/I and I/V stages into the analog blocks allows the system interconnects to be voltage-mode. There are multiple implementations of the V/I and I/V stages, such as the (b) diode-connected FET, (c) differential pair, (d) wide-range OTA, and (e) transimpedance amplifier.

Lastly, if bi-directional currents are desired, we can use a wide-range OTA V/I, shown in Figure 77d, and a transimpedance amplifier (TIA) I/V, shown in Figure 77e. The wide-range OTA has a single output current that is the difference of the two differential pair currents, but with an additional attenuation factor (α) that helps to linearize the tanh:

$$I_{out} = I_b \frac{\alpha\kappa}{2U_T} (V_1 - V_2) . \tag{76}$$

The output stage TIA has the transfer function: $V_{out} = V_{ref} - I_{in}/G_m$.

This pair of converters is linear and provides a bias current to the system to set the time constant. The linearity makes them useful individually for blocks that only need conversion on one port. They are also the choice for single-ended bi-directional systems.

7.4 The Process of Functional-Level Modeling

Now that we have defined the basic modeling parameters, let's apply them to our basic analog processing blocks. We provide the characteristic equations in a way that encapsulates the desired performance, without unnecessary computation. While we cover the modeling of three important blocks in this section, it is by no means an exhaustive list. The VMM, C^4 bandpass filter, and peak detector are merely representative examples. These techniques can easily extend to fill out the signal processing space to such blocks as DACs and analog arbitrary waveform generators (AWGs), which have recently been implemented in FPAA technology [40].

7.4.1 Vector-Matrix Multiplier

The VMM is one of the most powerful analog processing blocks. As described in [47], the analog VMM can perform one- two- or four-quadrant multiplication. Figure 78 illustrates the circuit-level implementation of one of the four-quadrant cells. The three important things to recognize in the schematic are: (1) the inputs and outputs are both current-mode signals; (2) because the currents are uni-directional, we perform the operation with differential signals and weights; and (3) the multiplier weights are programmed as floating-gate

values.

The matrix operation of the four-quadrant cell performs

$$\begin{bmatrix} w_+ & w_- \\ w_- & w_+ \end{bmatrix} \begin{bmatrix} I_{in+} \\ I_{in-} \end{bmatrix} = \begin{bmatrix} I_{out+} \\ I_{out-} \end{bmatrix}, \tag{77}$$

where the differential weights are deviations around a base weight (w_B):

$$w_+ = w_B + \frac{\Delta w}{2},\ w_- = w_B - \frac{\Delta w}{2}. \tag{78}$$

The overall transfer function for the current-mode VMM cell is thus:

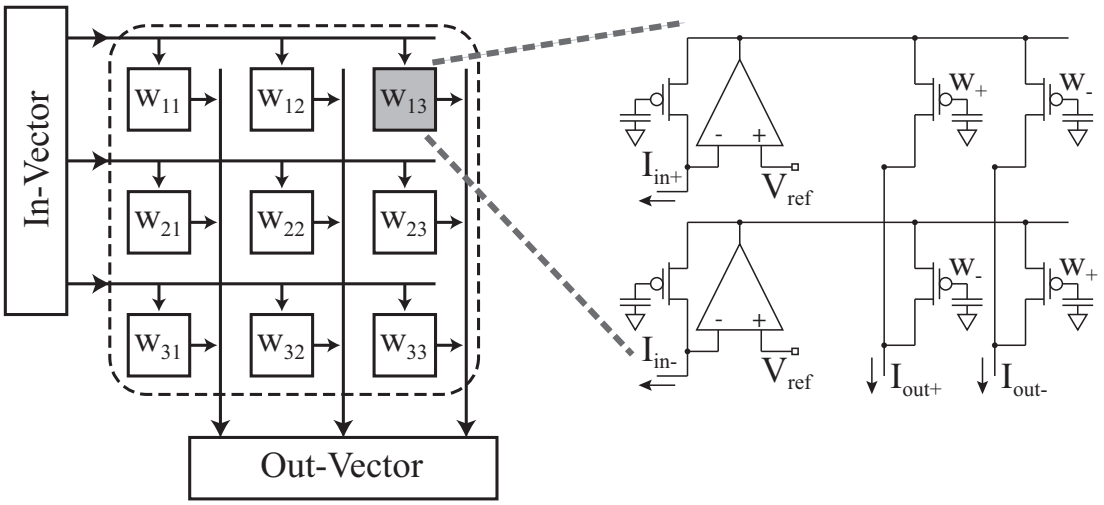
$$I_{out+} - I_{out-} = (w_+ - w_-) \cdot (I_{in+} - I_{in-}). \tag{79}$$

To complete the voltage-mode transfer function, we need to add the embedded conversion to the equation. The differential nature of the VMM sets up nicely for the use of a differential-pair input stage. The overall function is formed by cascading the VMM with the differential pair from Equation 73, and the differential diodes from Equation 75, resulting in:

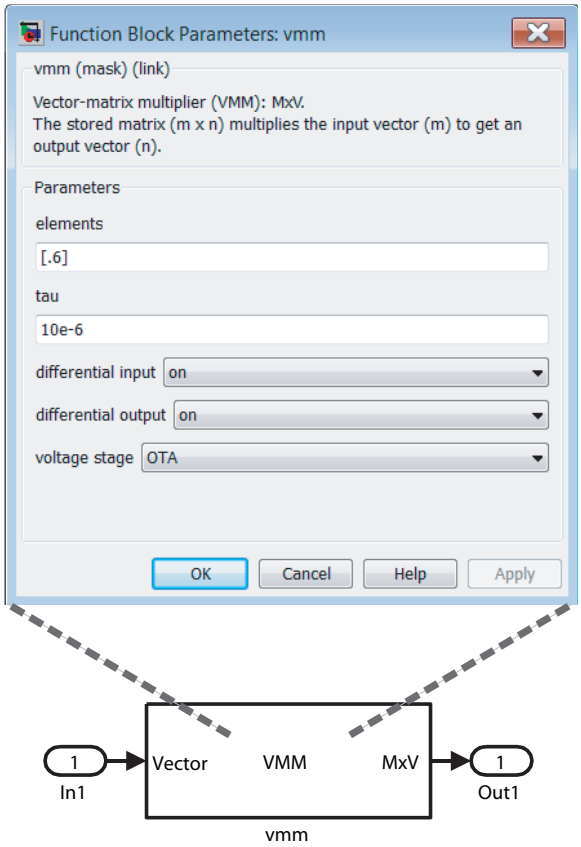
$$V_{out1} - V_{out2} = \frac{\Delta w}{2} (V_{in1} - V_{in2}). \tag{80}$$

The overall dynamics of the system is also composed of the three stages: input, VMM-processor, output. We can treat each stage as a single-pole low-pass filter, resulting in a three-pole system. To make the modeling as simple as possible, we will approximate the whole function as a single-pole system, using the lowest frequency pole of the three stages. The dynamics of the VMM stage are primarily set by the bandwidth of the log-amp. Dynamic analysis of the log-amp is described in [88] and the dominant pole is at $AI_b/(C_{in}U_T)$. The factor A is due to the active feedback in the log-amp and increases the effective transconductance (and thus speed) by about a factor of 100.

The pole at the output stage does not have this amplifier and is set by the transconductance of a single subthreshold FET, $I_b\kappa/(CU_T)$. Since the pole at the output is at the lowest frequency, we will use it as our single-pole approximation for the system. It is clear that the



(a)



(b)

Figure 78: Design of the VMM Simulink block. (a) The conceptual-level diagram of the VMM shows that the output channels are sums of products of the input channels. The circuit implementation is very compact with FG elements performing the weights. (b) The input parameters for the VMM block, showing options for matrix elements, time constant, differential signals, and voltage structure. The final design is packaged into a single Simulink block.

Table 9: Macromodel of the VMM.

```
(1)   % Read in Dialog parameters
(2)   M = block.DialogPrm(1).Data;
(3)   tau = block.DialogPrm(2).Data;
(4)   % Set port Dimensions
(5)   block.InputPort(1).Dimensions = size(M,2);
(6)   block.OutputPort(1).Dimensions = size(M,1);
(7)   % Initial state
(8)   block.ContStates.Data = M * block.InputPort(1).Data;
(9)   % Output
(10)  block.OutputPort(1).Data = M * block.ContStates.Data;
(11)  % Derivatives
(12)  y = block.ContStates.Data;
(13)  u = M * block.InputPort(1).Data;
(14)  dy=(u-y)./tau;
(15)  block.Derivatives.Data = dy;
```

bias current of the system will set the speed of each stage and thus should be parameterized in the modeling.

When creating the Simulink model for high-level design and simulation, we will abstract much of the previous circuit considerations. Table 9 provides the basic MATLAB code for the VMM model. The first element that is abstracted is the differential signals; we will simply allow for positive and negative signals and weights. Next, we allow for dynamic setting of the port dimensions. Lines 5–6 show the port sizes based on the size of the multiplier matrix. Lastly, we calculate the output in Lines 8–15, based on the input vector multiplied by the stored matrix, then passed through the single-pole filter.

The VMM dialog box in Figure 78b contains 5 parameters, whereas only the first 2 (elements and tau) were used in the Simulink model. All 5 parameters are used in the circuit compilation. The first one (elements) sets the multiplier weights and is programmed into the floating-gate mesh. The time constant is implemented as the DC bias current of the input stage. This DC current is the same in each stage and will set the overall speed. The last 3 parameters indicate to the circuit compiler if differential signals are needed and which conversion stage is desired by the user.

7.4.2 C^4 Band-Pass Filter

The C^4 filter is commonly used in analog signal processing because it is programmable, extremely compact, power efficient, and can cover a wide range of frequencies [83].

This block is most commonly used within a bank of filters with a narrow passband (Figure 79), such as in a Fourier processor system [89]. The defining parameters for such a block is a vector of center frequencies (f_{center}) and the quality factor (Q).

The dimension of the f_{center} parameter array will set the port dimension of the output and be interpreted as the number of parallel filter channels. If the Q is input as a scalar, that value will be applied to each filter. Alternatively, if the Q is input as a vector matching the dimension of the f_{center}, each filter can have a different value. There is also an option for a common input. This option allows the user to configure the block with vectorized input bus or a single input line that goes to each filter element.

The C^4 is commonly deployed as a differential system, which is in the spirit of the differential nature of our tool. However, to make the analysis simpler, we will be looking at the single-ended version.

The schematics for two implementations of the C^4 filter are shown in Figure 80. The first is based on transistor gain stages and is the most compact for custom integrated circuits. The high and low corners can be tuned independently of each other, and are controlled by V_{bh} and V_{bl}. These values can be programmed with FG transistors to precisely tune the filter. By cascading this second-order section we can achieve higher order filters, or we can place them in parallel to spectrally decompose an incoming signal.

The general transfer function for the C^4 filter in the $Q > 1$ region is provided by [83] to be

$$\frac{V_{out}}{V_{in}} = -\frac{C_1}{C_2}\frac{sC_2/g_{m1}}{1 + s\left(\frac{C_2}{g_{m1}} + \frac{C_0}{g_{m4}}\right) + s^2\frac{C_0 C_T}{g_{m4}g_{m1}}}, \tag{81}$$

where $C_T = C_1 + C_2 + C_W$ and $C_0 = C_2 + C_L$. The center frequency is thus set by:

$$f_{center} = \frac{1}{2\pi\tau} = \frac{\sqrt{g_{m4}g_{m1}}}{2\pi\sqrt{C_0 C_T}}. \tag{82}$$

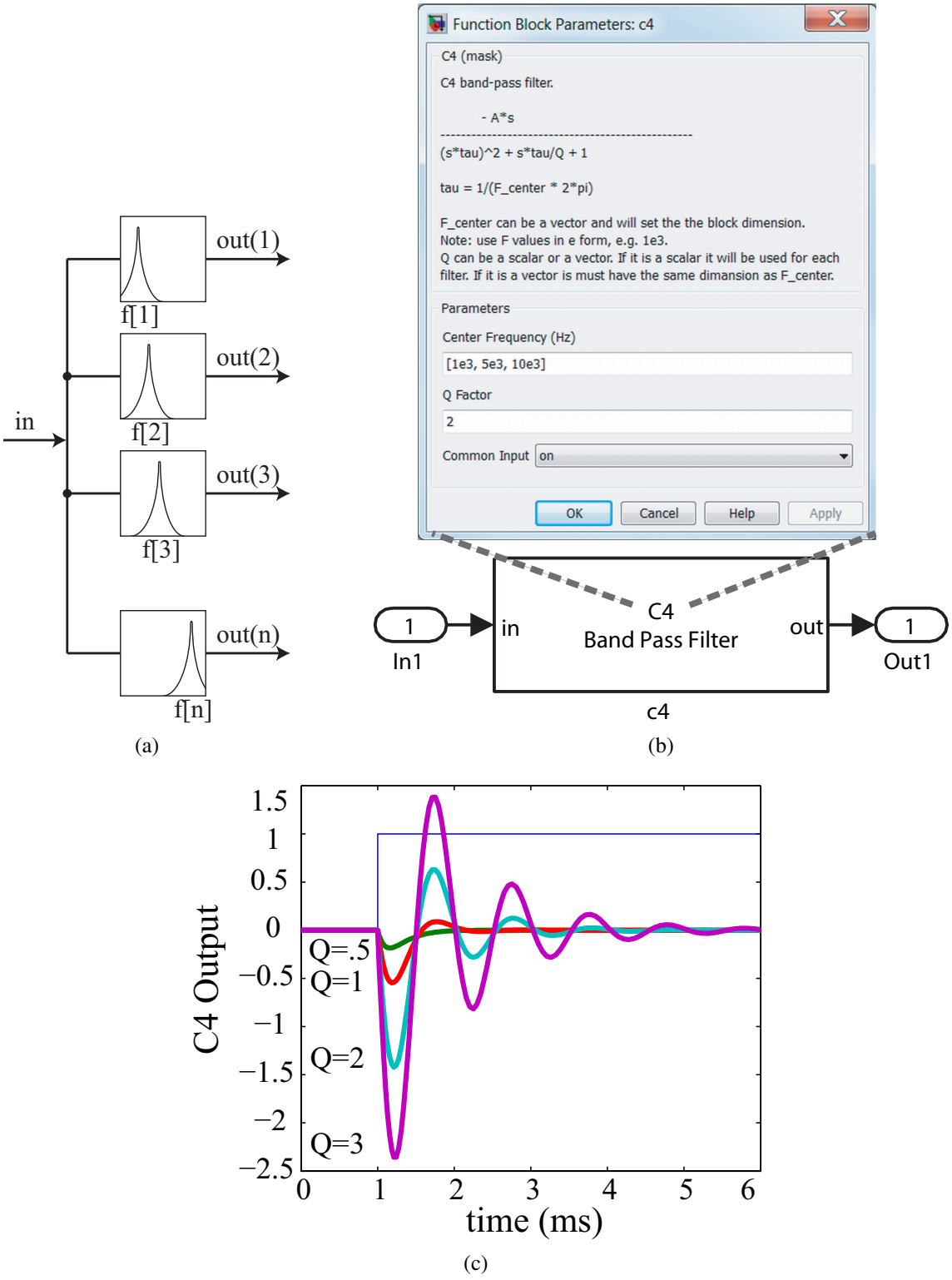


Figure 79: The C^4 band-pass filter system. (a) The system implementation and (b) the Simulink block. (c) The Simulink step response for C^4 filter vectorized with 4 outputs. Each tap is tuned for a center frequency of 1 kHz and a Q vector of [.5, 1, 2, 3].

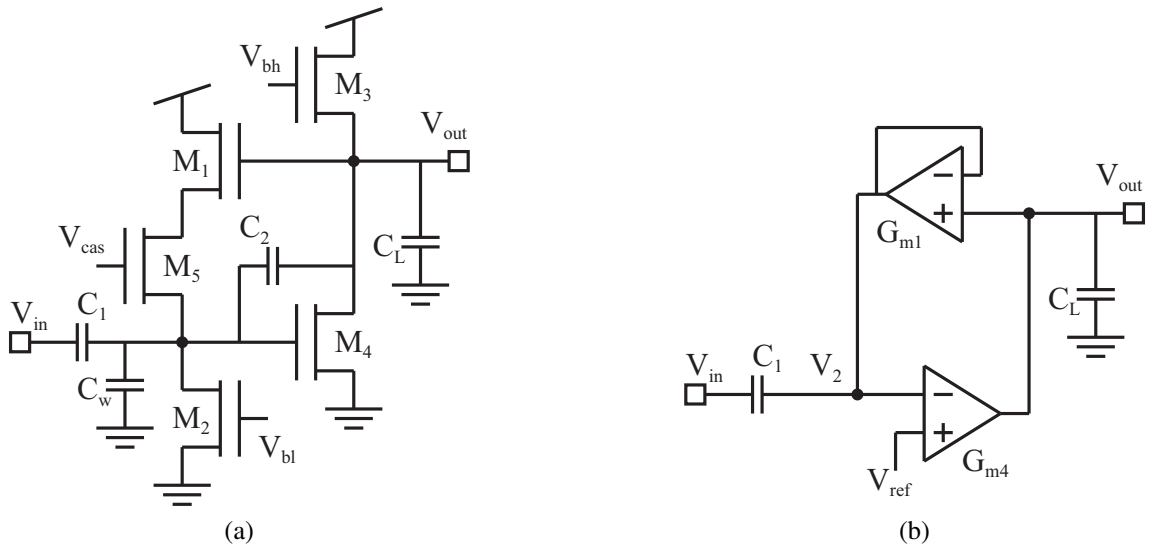


Figure 80: Schematics of the C^4 bandpass filter. (a) The schematic can be implemented either with transistors or (b) OTAs.

This equation provides us with the algorithmic method for generating the circuit netlist from a vector of center frequencies, given that we use Q_{max}.

The structure in Figure 80b makes efficient use of the FPAA components by replacing the single-transistor amplifiers with OTAs. Although this would seem to be less efficient, when working in the FPAA environment each component is an individual CAB elements. Therefore, using a single OTA is more efficient that a transistor with additional biasing circuitry. Also, when synthesized on the FPAA, capacitors C_2 and C_w will not be drawn explicitly, but arise from parasitics.

The OTA structure makes the transfer function easy to visualize as

$$\frac{V_{out}}{V_{in}} = -\frac{\tau_1 s}{1 + \tau_2 s + \tau_1 \tau_2 s^2}, \tag{83}$$

where $\tau_1 = C_1/gm_1$ and $\tau_2 = C_L/gm_4$. Given the canonical form of a second order filter, we can see the frequency ($\tau = 1/2\pi f_{center}$) and Q map to these time constants as $\tau_2 = \tau/Q$ and $\tau_1 = \tau^2/\tau_2$. In Simulink, this system of dynamic equations is scripted as shown in Table 10.

A similar procedure could be used to implement general filters of higher order. The built-in MATLAB tools could be used to generate filter coefficients (e.g., Chebychev) and broken into cascades of second-order sections and a first-order block. Once the coefficients

Table 10: Macromodel of the C^4 linear filter.

```
(1)   % Read in Dialog parameters
(2)   fcenter = block.DialogPrm(1).Data;
(3)   Q = block.DialogPrm(2).Data;
(4)   commonInput = block.DialogPrm(3).Data;
(5)   signalDim = length(fcenter);
(6)   % Set port Dimensions
(7)   if commomInput == 1
(8)     block.InputPort(1).Dimensions = 1;
(9)   else
(10)    block.InputPort(1).Dimensions = signalDim;
(11)  end
(12)  block.OutputPort(1).Dimensions = signalDim;
(13)  % Initial states
(14)  block.ContStates.Data(1) = 0; %x1
(15)  block.ContStates.Data(2) = 0; %x2
(16)  % Output
(17)  block.OutputPort(1).Data = block.ContStates.Data(1); %y=x1
(18)  % Dynamics Parameters
(19)  T=1./(2*pi.*fcenter);
(20)  tau2=T./Q;
(21)  tau1=T.^2./tau2;
(22)  a=-1./tau2;
(23)  b=1./tau1;
(24)  c=1./(tau1.*tau2);
(25)  % Derivatives
(26)  u = block.InputPort(1).Data;
(27)  x1 = block.ContStates.Data(1)
(28)  x2 = block.ContStates.Data(2)
(29)  dx1=x2+a.*u;
(30)  dx2=-c.*x1-b.*x2-a.*b.*u;
(31)  block.Derivatives.Data(1) = dx1;
(32)  block.Derivatives.Data(2) = dx2;
```

are in second-order groupings, they can be applied to the biases and capacitors of the biquad blocks.

Following the discussion in Section 7.3.1, it is useful to understand the nonlinear dynamics of the C^4 system. This level of understanding will allow us to include an additional level of accuracy in our models. In most cases, the model in Equation 83 is the most efficient to simulate and extract understanding of the function; however, having an option to include the nonlinear dynamics will provide a closer match to the dynamic-range of the real analog circuits.

Deriving the system equations from the implementation Figure 80b, we see that the OTA will really provide a tanh rather that a linear transconductance. The resulting model is:

$$\dot{V}_{out} = -\frac{I_{b4}}{C_L} \tanh\left(\frac{\kappa}{2U_T} V_{out}\right), \tag{84}$$

$$\dot{V}_2 = \frac{I_{b1}}{C_1} \tanh\left(\frac{\kappa}{2U_T} (V_{out} - V_2)\right) + \dot{V}_{in}. \tag{85}$$

We also note that the steady-state condition is that all node voltages will be equal and that the V_{ref} would match the zero-input voltage.

7.4.3 Peak Detector

While the C^4 block is often used a parallel filter bank to spectrally decompose an incoming signal, a fundamental complement to such a block is the peak detector. This block will track the envelope of the signal in each band, which is useful for further processing and classification functions. The circuit-level implementation of the peak detector is shown in Figure 81.

Intuitive analysis of this block shows that it acts much like a source follower. When the input rises, the output will track it while charging the capacitor on the output node. However, when the output is decreasing, the bias transistor (M2) will discharge the capacitor as a fixed rate. This behavior will allow the circuit to track the rising peaks, then decay at a fixed rate until it hits the next rising peak. The rate of decay is set by M2.

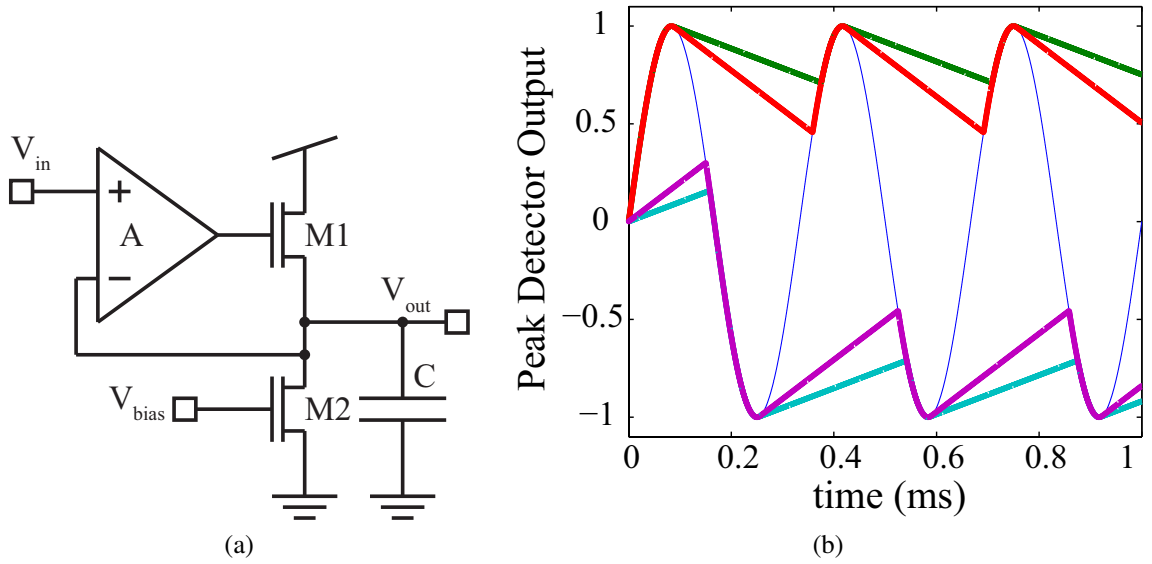


Figure 81: The Peak Detector. (a) Schematic and (b) Simulink simulation. The output shows the block vectorized for four outputs with a common sine input. Of the four signals, two are MAX followers and two are MIN followers, each shows with decay rates of 1e3 and 2e3.

To create the full dynamic model for this block, we start with KCL at the output node:

$$C\dot{V}_{out} = I_0 e^{(A\kappa(V_{in}-V_{out})-V_{out})/U_T} - I_0 e^{\kappa V_b/U_T}. \tag{86}$$

We see that at DC (i.e., at $\dot{V}_{out} = 0$) the current in the top branch is balanced by the bottom branch and is therefore equal to I_b.

Under dynamic conditions, the output changes at a rate of

$$\dot{V}_{out} = \frac{I_b}{C} e^{[A\kappa(V_{in}-V_{out})-V_{out}]/U_T}. \tag{87}$$

This equation is consistent with our intuitive analysis. When V_{in} exceeds V_{out}, the rate of growth on the output will become exponential. As V_{in} falls below V_{out}, the exponentials will become small and the term inside the parenthesis will reduce to -1. This rate of decay will be fixed at I_b/C.

To create a block macromodel of this circuit, we'll simply use Equation 87 as the model. The main parameter will be the rate of decay. At a system level, the rate of decay should be tuned to the expected frequency of the incoming signal so that the peak detector can track

the envelope. If the decay is set too slowly, the block will simply find the largest amplitude and hold it through other cycles. This system can be easily converted into a minimum detector by using a pFET source follower rather than the nFET described.

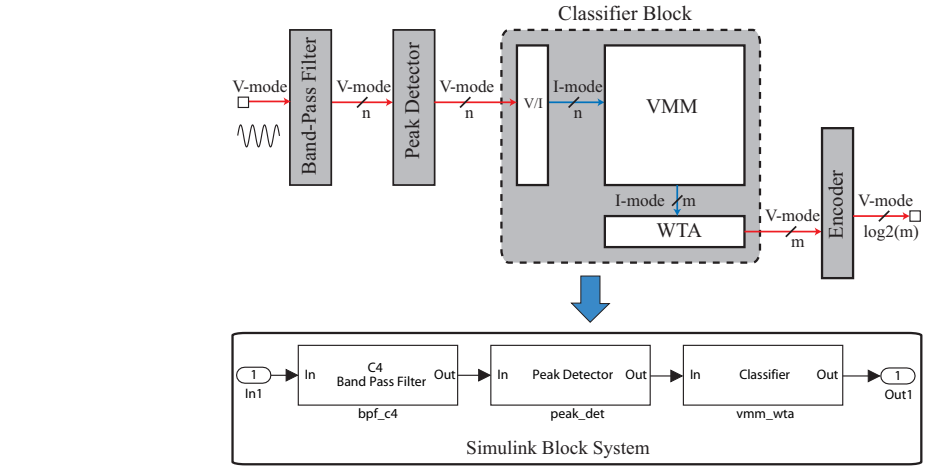
7.5 Case Study: Classifier System

To bring together the process of creating a whole signal-processing system with the design platform, we will use the classifier system in Figure 82 as a circuit example. This example will highlight two important aspects of functional-level analog design: (1) using blocks from the pre-defined library of analog signal processing elements; and (2) utilizing inherent mixed-mode computation to create optimal blocks.

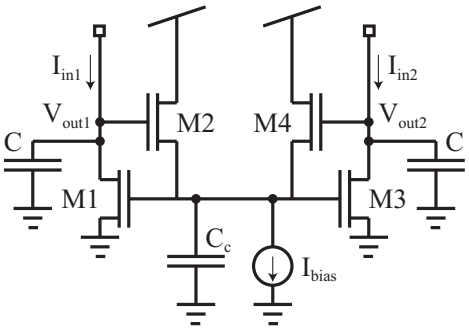
The complete system is a chain of 5 processing blocks: a C^4 filterbank, a peak detector, VMM, winner-take-all (WTA), and an encoder. The overall system will take an input waveform, spectrally decompose it into multiple bands as specified by the C^4 bank, and use the peak detector to track the envelope of each channel. Next, the VMM will project the spectral channels against multiple classification basis, and the WTA will pick the largest output. Finally, the encoder will compress the location of the winning element into a digital value. The first three blocks have already been thoroughly discussed and modeled in this chapter, whereas the encoder is discussed in many digital-design textbooks, and the WTA deserves proper analog modeling.

The WTA circuit implementation was invented in [84]. The function that this block will perform is to take an input vector and produce an output vector of the same dimension that is filled with all low values and a single high value in the element corresponding to largest input element.

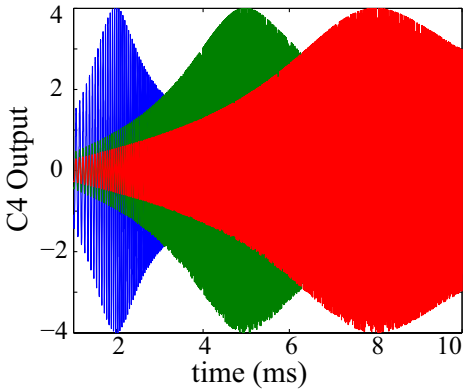
The WTA is so commonly used with a VMM on the front end as a classifier that we combined the two into the larger classifier block shown in the dashed box of Figure 82a. This is an efficient structure for a classifier and is much more compact than a two-layer neural network, which would require two VMMs and sigmoid blocks.



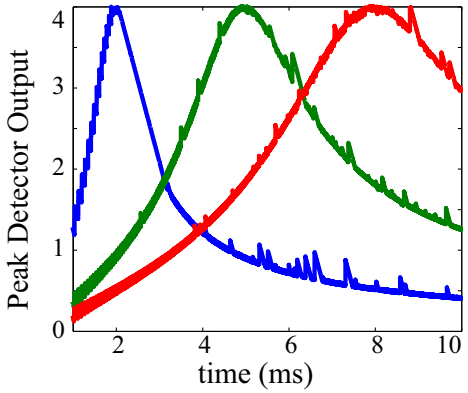
(a)



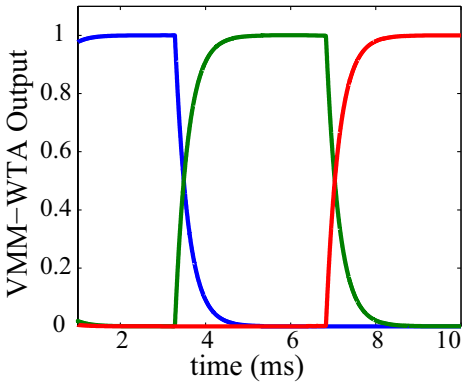
(b)



(c)



(d)



(e)

Figure 82: The classifier system. (a) The VMM and WTA combine to create a single library block, allowing the internal I/V-V/I to cancel. (b) The circuit schematic of the 2-element WTA. (c) Simulink simulation from the classifier system for a linear chirp signal. The C^4 bandpass filer is set with three blocks to pass different sections of the chirp. (d) The peak detector tracks the envelope of the three channels. (e) The VMM-WTA classifier creates an output where only one channel is high at a time. The matrix is [2, 1, .5; .5, 2, 1; 1, .5, 2] simply to demonstrate each channel winning at a time.

Merging the VMM and WTA into a new block has another advantage. The WTA has a current-mode input and voltage-mode output, therefore we would traditionally want to add a V/I stage on the input. However, the VMM has a native current-mode output, so by combining the two we can cancel a I/V-V/I conversion and allow all current-mode processing on the internal nets. This will provide a more efficient and compact realization.

When modeling the WTA for the sake of Simulink simulations, we need to pick among progressively more detailed models. The simplest model of the WTA is the MAX function, which can be programmed easily with the MATLAB toolbox. This model will simulate the quickest, but will miss a lot of the dynamics involved in the analog implementation.

For a more detailed model, we can look at the transistor equations based on the schematic in Figure 82b. From [84], a model for the two input VMM is given for three regions: (1) $V_1 \approx V_2 \approx V_m$; (2) $V_1 \gg V_m$ while $V_2 \ll V_m$; and (3) $V_2 \gg V_m$ while $V_1 \ll V_m$. Here, $V_{1,2}$ refer to the individual elements of the output vector, and V_m refers to the equilibrium voltage. We can use the output voltage because the relation to each other also matches the input current relationship.

For the Region 1, we have a static response of

$$V_1 = (V_e/2)\,(I_1/I_m - 1) + V_m, \tag{88}$$

$$V_2 = (V_e/2)\,(1 - I_1/I_m) + V_m, \tag{89}$$

where $I_{1,2}$ are the input currents, V_e is the Early voltage, and I_m is the I_{bias} of the input signals. For the Region 2, with some approximations, we have

$$V_1 = V_0 \ln(I_1/I_m) + (V_0/2) \ln(V_e/V_0) + V_m, \tag{90}$$

$$V_2 = V_0/2 + V_m - (Ve/I_m)\,\delta_i, \tag{91}$$

where $V_0 = kT/q\kappa$. And finally, for Region 3, we have

$$V_1 = V_m - (Ve/I_m)\,\delta_i, \tag{92}$$

$$V_2 = V_0 \ln(I_m/I_0) + V_0 \ln(I_c/I_0)\,. \tag{93}$$

The system dynamics equations provide a more accurate model of the time-response. Under the condition $I_C > 4I_1\,(C_C/C)$, we do not have ringing in the response and first-order time constants. For the case were the first input starts to win, the first-order time constant for V_1 is CV_0/I and the first-order time constant for V_2 is CV_e/I, where $I_1 \approx I_2 \equiv I$.

It should be obvious, then, that the more detailed model will get much more complex as the number of inputs increase. For this reason, the simple MAX function is the mathematical model of choice when the demand on computer resources is to be minimized or the dynamics of the other blocks are assumed to be slower than that of the WTA.

With the mathematical model for the WTA defined, we simply cascade it with the VMM and V/I models to form the complete classifier block. The input dimension to the classifier will match the column dimension of the VMM, and the output will match the row dimension. The inputs to the block have the ability to be differential, where the branch between the VMM and WTA is single ended. Finally, the output is a single-ended binary vector.

Figure 82 shows the Simulink simulation results for the classifier system. The three plots show the output of each block of the system in the lower half of Figure 82a. For this experiment, the input signal is a chirp, linearly swept to 10 kHz over 10 seconds. The C^4 filter block is set with three center frequencies, resulting in three parallel filters in the bank. Figure 82c shows the output of three channels of the C^4, each passing the signal in a separate frequency region.

Figure 82d shows the output of the peak detector section. The block has 3 parallel tracks that track the envelope of the C^4 output.

Lastly, Figure 82e shows the output of the VMM-WTA classifier section. For this example, the VMM is set with a 3×3 matrix that will give each output a chance to win. Of the three output channels, in general the one corresponding to the largest envelope will have a high value.

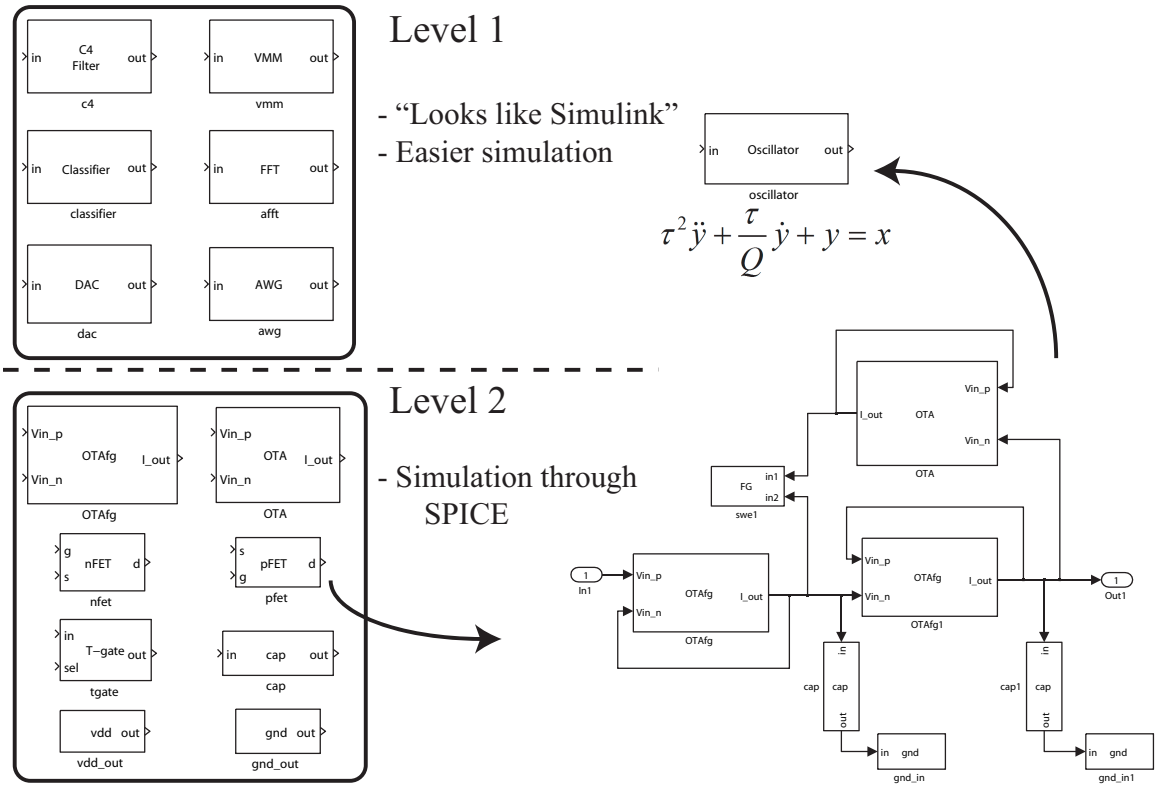


Figure 83: To accommodate users with varying expertise, we have multiple level libraries. The Level-1 library will include functions with which a typical DSP engineer will feel comfortable designing, such as filterbanks, vector-matrix multipliers, classifiers, analog FFT, DACs, and arbitrary waveform generators. The Level-2 library contains the low-level blocks, for instance the CAB elements. The Level-2 library is meant for experienced analog design engineers. The right shows the design cycle of a second-order section oscillator. The analog engineer designs with the Level-2 blocks, simulates with SPICE, and tests on the FPAA. Once the design is stable, a Level-1 block can be packaged. The Level-1 block uses the abstraction techniques and as simple as possible transfer function for simulation.

7.6 Tools for IC Experts

So far, we have discussed the processes of modeling high-level analog signal processing blocks for the use in abstracted top-down design. However, our framework also includes support for the design of analog systems from the bottom up. To separate the two, we have multiple-level libraries in which blocks are posted. The Level-1 and Level-2 libraries are shown in the left half of Figure 83.

The Level-1 library contains the high-level system blocks. These blocks conform to the voltage-mode protocol and contain sufficient abstraction so that they are reasonable

to simulate in Simulink. Examples of Level-1 blocks are the VMM, band-pass filter, and classifier previously discussed.

The Level-2 library contains the low-level blocks, typically mapping directly to FPAA CAB elements. These blocks do not conform to the voltage-mode protocol and might have advanced modeling parameters. These blocks are best used by circuit-design engineers and should be simulated in a SPICE environment. Examples of Level-2 blocks are transistors, OTAs, and capacitors.

Additional digital libraries are not shown in the figure, but are acceptable for use in FPAA mixed-mode design. The RASP FPAA is capable of compiling these digital circuits if an accurate circuit model is attached to each block. Alternatively, if proper FPAA ports are specified, mixed-mode designs can be divided such that the entire system is simulated in Simulink and only the analog portions are compiled to the FPAA.

We use the second-order-section oscillator system in Figure 83 as an example to illustrate the process of designing an analog system with low-level blocks, for inclusion onto a high-level library. This oscillator can be thought of as a second-order filter that can provide controlled instability (oscillations).

The schematic is shown in the bottom right of the figure and built entirely of Level-2 blocks. The circuit contains two FG-input OTAs, one OTA, two capacitors, two ground connections, and one FG element to short the feedback path. This last element, the FG short, demonstrates one difficulty in performing current-mode operations Simulink. The feedback in this circuit mixes two currents and integrates them on the left capacitor. Although mixing currents is a common analog practice, Simulink operates in voltage mode and cannot have two outputs drive a line. Therefore, we have used an FG switch with two "inputs" to short the two nets. This results in a legitimate circuit that will simulate in SPICE and operate in silicon; it will not, however, simulate in Simulink. The simulation of floating gates is difficult in SPICE due to the fact that there is no direct DC path to ground. To accomplish simulation, we use the model described in [90].

To make this block useful to system designers, we abstract it to the high-level block shown in the top right of the figure. Here, we have expressed the eight-element circuit as a simple second-order differential equation [2]. This equation is very easy for Simulink to simulate. In this expression, the user would specify the time constant (τ) and the quality factor (Q). These system parameters will be translated into physical parameters without the user's involvement. If we use equal capacitors and G_m for the forward FG-OTAs, we get $\tau = C/G_m$. With G_m set, we get $Q = 1/(2 - G_{mFB}/G_m)$, where G_{mFB} is the transconductance of the feedback OTA. For oscillations, we want infinite or negative Q, so we should have $G_{mFB} \geq 2G_m$.

With the equation set, we just need to define the signal dimension. There is no need to add any conversion stages because this block is already voltage in and out. This block can be arrayed by allowing the user to input two n-element vectors, one for τ and Q. The block can automatically set its input and output ports based on the size of the parameter array. The resulting system will have N oscillator circuits in parallel, each programmed with the elements of the parameter array.

In our abstracted framework, the process of block design and analysis should be performed by circuit designers. The design and analysis involves rigorous and thorough testing in order to provide accurate and reliable blocks to the Level-1 library.

7.7 Conclusion

In this chapter, we have demonstrated the concept of high-level abstraction and modeling of analog systems. With the drastic increase in size and complexity of modern reconfigurable analog ICs, high-level tools are a necessity. We demonstrated how analog abstraction techniques are a powerful tool for making analog system design easy for non-circuit experts. A key element of this abstraction approach is the creation of high-level analog libraries. We introduced our methodology for analog macromodeling that looks at the function being performed, rather than each low-level element. We validated our approach by macromodeling

several analog signal processing blocks, including a vector-matrix multiplier, a bandpass filter bank, and a peak detector. We then showed the process of combining these blocks into a larger analog classifier system. This body of work introduces and new level of intuition into the field of analog system design.

CHAPTER 8

CONCLUSION

The purpose of this research has been to create a solid framework for embedded system design with FPAAs. Towards this goal, we've discussed a unified approach to the three phases of FPAA design: (1) the hardware architecture; (2) the circuit design and modeling; and (3) the high-level software tools. A major role model in this endeavor has been the Mead-Conway digital revolution of the 1980s. Carver Mead and Lynn Conway helped to spark the VLSI boom by unifying the fields of computer architecture, integrated circuit design, and semiconductor device physics.

The result of this effort is a much more open FPAA design environment, where one need not be a circuit expert to take advantage of analog signal processing technology. The Simulink interface has provided a top-level design space where analog systems can be designed intuitively with blocks and lines. This environment feels much more like traditional digital design and is familiar to many signal-processing engineers. The circuit macromodeling has abstracted the analog blocks and demystified their function for designers. Finally, the advanced FPAA architectures have made it much easier to embed into larger systems and interface with digital electronics.

8.1 Summary of this Dissertation

Chapter 2 provided a review of the fundamental background technology. The two key elements covered were floating-gate transistors and field-programmable analog arrays. The floating gate (FG) was important for analog storage and the FG transistor was used as a switch as well as for computation. We reviewed the device characteristics—including subthreshold operation—as well as the processes for adding and removing charge from the gate. In the review of FPAA technology, we went over a brief history of their evolution and described some of the various architectures that have been tried. We then provided a

detailed description of reconfigurable analog signal processor (RASP) FPAA, which was the hardware platform used throughout this dissertation.

Chapter 3 introduced the multiple-input translinear element (MITE) FPAA. This architecture was based on the MITE as a circuit primitive, which is ideal in computing polynomial equations. There is a robust body of work on the synthesis of high-order static and dynamic equations to systems of MITEs, which made it an ideal platform to get started with analog signal processing.

Chapter 4 introduced the RASP 2.9v, a next-generation FPAA architecture that was optimized for embedded systems. This architecture was motivated by the need for higher-level digital control for dynamic reconfigurability. This controllability allowed the FPAA to be much more easily fielded in embedded electronic systems.

Chapter 5 presented the analog vector-matrix multiplier (VMM) as a bottom-up case study for the analysis of a computational analog element and its mapping to the FPAA architecture. The VMM is one of "killer apps" of analog computation in that it efficiently computes a very common signal processing function. The circuit design process included step-by-step choices motivated by the architecture of the FPAA hardware. The circuit analysis included descriptions of the system's power, speed, noise, and temperature dependence.

Chapter 6 introduced the high-level software tools for FPAA configuration. The top-level design space is Simulink, which provided an intuitive platform for designing signal processing systems with functional blocks. The core elements of this system were the component library and the FPAA compilers. The component library contains abstracted analog blocks that allow non-circuit designers to create large systems with analog blocks. The compilers included two main tools: (1) Sim2Spice for converting Simulink designs to a circuit netlist; and (2) the GRASPER tool for converting the netlist to FPAA targeting code.

Chapter 7 discussed the challenges and opportunities of abstracted analog design. By

defining a standard interface for the analog block library, designers can treat analog blocks more like their digital counterparts. This standardization provides a low barrier-to-entry for engineers trained in the *problem domain* of signal processing, but having limited experience in the *solution domain* of analog hardware. A major focus of this work was the creation of accurate, yet elegant, macromodels for the simulation and overall understanding of the function of the analog signal processing blocks.

8.2 Personal Contributions

Much of the work described in this dissertation is my own. However, almost everything was done in a collaborative atmosphere.

I was involved in the earliest testing of the RASP 2.8a FPAA along with the rest of the ICE Lab: Arindam Basu, Stephen Brink, Scott Koziol, Csaba Petre, and Shubha Ramakrishnan. I designed and tested many circuits, which was much more arduous then it is now, as all of the designs were done with pencil and paper fuse charts. I also had a key role in porting the RASP 2.8a to the larger version, RASP 2.9a. The RASP 2.8a has a journal and a conference paper [19, 20].

I drove the effort to publish and run analysis on the results from the MITE FPAA, where Dave Abramson performed the design and test of the IC. This work has a journal and conference paper [23, 29].

The RASP 2.9v FPAA was a collaboration with Sam Shapero and Steve Nease, in which I led the architecture, layout, and timeline of the project. Sam and Steve helped tremendously with the layout of the DAC and VMM CABs, as well as with the addressing architecture. The two of them were also the main drivers in getting the fabricated chip up and running. All three of us were responsible for the system application testing and data that was presented. This chip has resulted in a journal and conference paper [40, 41].

My work on the VMM analysis came out of the need for a detailed Simulink block for that element. The circuit had been invented long before I joined the lab, but there was no

single paper that included all of the analysis and mapping for FPAA hardware. My analysis and FPAA mapping work has a journal and a conference paper [47, 55].

The Sim2Spice compiler tool was built as a collaboration with Csaba Petre. I wrote the parsing script used to read the model files and Csaba wrote the initial netlist generator. Since the initial design, I have been the sole person maintaining the code—adding more features, fixing bugs, and making it all-around more robust. I have also been creating and maintaining all of the library blocks. Sim2Spice has a journal and a conference paper [44, 61]. I also worked with Scott Koziol and Csaba to build the first three versions of the Programming & Evaluation Board, which has a conference paper where it received the *Best Paper - Live Demonstration* award [37].

The high-level analog modeling and abstraction was individual work and involved defining the interface between analog blocks as voltage-mode vectorized, as well as modeling the transfer function and often nonlinear dynamics of the analog systems. This modeling work has resulted in a conference paper and has been submitted to a journal [81, 80].

Of course, along the way I've been involved in many other efforts. An extremely rewarding aspect of this work was getting it in the hands of willing subjects and seeing their applications realized with the FPAA platform. For instance, I mentored Sangwook Suh in implementing an OFDM receiver with the VMM on the RASP 2.9a [54]. And I've been involved in hosting several FPAA tutorials and workshops. I have even been involved in tools for destructing ICs. It was the culmination of these and other efforts that helped advance the state of FPAA technology

8.3 Future Directions of this Work

As demonstrated in this dissertation, a tremendous amount of work has already been completed in creating a unified framework for FPAA design. However, there are still several areas where work is left to do.

First, true validation of this technology will come with fielding it in larger and more

complex systems. There is already a major effort to apply the FPAA to robotic applications, but the space is much wider open that that. Any system that could benefit from power-efficient processing is a potential applicant. The proof of the technology will come in attacking "far-forward" real-world problems.

In addition, the Simulink analog library development is an open-ended task. We have created many blocks, but our hope is that as others design analog processing circuits, they will add them to the library. Future users are encouraged to look at the existing blocks as examples to follow. The library of elements will only grow in importance as more blocks are added and non-circuit designers look to it as a source of tools. A main aspect of block design is the macromodeling component, which must be done on a block-by-block bases. The accurate models are very important to maintain to that system engineers can easily understand how the block is behaving and can simulate their ever more complex systems.

Also, the trajectory of what future FPAA architectures should look like is open to exploration. The popularity of the RASP 2.9v's new features have demonstrated that FPAAs need more digital support. This can take many forms—from adding digital blocks along with the CABs for mixed-signal systems to embedding full microcontrollers on chip to create a large-scale SoC.

With many of the opportunities still in front of us, it remains an exciting time to be innovating with reconfigurable analog signal processing.

REFERENCES

[1] P. Hasler and D. Anderson, "Cooperative analog-digital signal processing," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, pp. 3972 – 3975, 2002.

[2] C. Mead, *Analog VLSI and Neural Systems*. Addison Wesley, 1989.

[3] G. Frantz, "Digital signal processor trends," *IEEE Micro*, vol. 20, no. 6, pp. 52 – 59, 2000.

[4] P. Hasler, "Low-power programmable signal processing," in *Int. Workshop on System-on-Chip for Real-Time Appl.*, pp. 413 – 418, 2005.

[5] B. Marr, B. Degnan, P. Hasler, and D. Anderson, "Scaling energy per operation via an asynchronous pipeline," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. PP, no. 99, pp. 1–5, 2012.

[6] B. Murmann, C. Vogel, and H. Koeppl, "Digitally enhanced analog circuits: System aspects," in *IEEE Int. Symp. Circuits and Systems*, pp. 560 – 563, 2008.

[7] R. Robucci, K. Leung, J. Gray, J. Romberg, P. Hasler, and D. Anderson, "Compressive sensing on a CMOS separable transform image sensor," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, pp. 5125 – 5128, 2008.

[8] S. Peng, Y. Tsao, P. Hasler, and D. Anderson, "A programmable analog radial-basis-function based classifier," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, pp. 1425 – 1428, 2008.

[9] C. Mead and L. Conway, *Introduction to VLSI Systems*. Addison Wesley, 1979.

[10] D. Kahng and S. Sze, "A floating-gate and its application to memory devices," *The Bell System Technical Journal*, vol. 46, no. 4, pp. 1288–1295, 1967.

[11] P. Smith, M. Kucic, and P. Hasler, "Accurate programming of analog floating-gate arrays," in *IEEE Int. Symp. Circuits and Systems*, vol. 5, pp. 489 – 492, May 2002.

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

[13] E. Lee and P. Gulak, "A CMOS field-programmable analog array," *IEEE J. Solid-State Circuits*, vol. 26, pp. 1860 – 1867, Dec. 1991.

[14] A. Stoica, D. Keymeulen, R. Zebulum, A. Thakoor, T. Daud, Y. Klimeck, R. Tawel, and V. Duong, "Evolution of analog circuits on field programmable transistor arrays," in *NASA/DoD Workshop on Evolvable Hardware,*, pp. 99–108, 2000.

[15] G. Cowan, R. Melville, and Y. Tsividis, "A VLSI analog computer/digital computer accelerator," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 42–53, 2006.

[16] J. Becker, F. Henrici, S. Trendelenburg, M. Ortmanns, and Y. Manoli, "A field-programmable analog array of 55 digitally tunable OTAs in a hexagonal lattice," *IEEE J. Solid-State Circuits*, vol. 43, no. 12, pp. 2759 – 2768, 2008.

[17] Anadigm, "Anadigm fpaa." http://www.anadigm.com/fpaa.asp, May 2012.

[18] Cypress, "Psoc." http://www.cypress.com/?id=1353, May 2012.

[19] 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 J. Solid-State Circuits*, vol. 45, no. 9, pp. 1781 – 1794, 2010.

[20] A. Basu, C. Twigg, S. Brink, P. Hasler, C. Petre, S. Ramakrishnan, S. Koziol, and C. Schlottmann, "RASP 2.8: A new generation of floating-gate based field programmable analog array," in *IEEE Custom Integrated Circuits Conf.*, pp. 213 – 216, Sept. 2008.

[21] C. Twigg, J. Gray, and P. Hasler, "Programmable floating gate FPAA switches are not dead weight," in *IEEE Int. Symp. Circuits and Systems*, pp. 169 – 172, May 2007.

[22] A. Basu and P. Hasler, "A fully integrated architecture for fast and accurate programming of floating gates over six decades of current," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 45, no. 9, pp. 1781 – 1794, 2010.

[23] C. Schlottmann, D. Abramson, and P. Hasler, "A MITE-based translinear fpaa," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 1, pp. 1 – 9, 2012.

[24] D. Abramson, J. Gray, S. Subramanian, and P. Hasler, "A field-programmable analog array using translinear elements," in *Int. Workshop System-on-Chip for Real-Time App.*, pp. 425–428, 2005.

[25] D. Fernandez, L. Martinez-Alvarado, and J. Madrenas, "A translinear, log-domain FPAA on standard CMOS technology," *IEEE J. Solid-State Circuits*, vol. 47, no. 2, pp. 490 – 503, 2012.

[26] B. Minch, "Synthesis of static and dynamic multiple-input translinear element networks," *IEEE Trans. Circuits and Syst. I*, vol. 51, no. 2, pp. 409–421, 2004.

[27] S. Subramanian, *Methods for Synthesis of Multiple-Input Translinear Element Networks*. PhD thesis, Georgia Tech, 2007.

[28] E. McDonald and B. Minch, "Synthesis of a translinear analog adaptive filter," in *IEEE Int. Symp. Circuits and Systems*, pp. 321–324, May 2002.

[29] C. Schlottmann, B. Degnan, D. Abramson, and P. Hasler, "Reducing offset errors in MITE systems by precise floating gate programming," in *IEEE Int. Symp. Circuits and Systems*, pp. 1340 – 1343, May 2010.

[30] J. Mulder, W. Serdijn, A. van der Woerd, and A. van Roermund, *Dynamic Translinear and Log-Domain Circuits: Analysis and Synthesis*. Boston, MA: Kluwer ACademic Publishers, 1999.

[31] P. Hasler, B. Minch, and C. Diorio, "An autozeroing floating-gate amplifier," *IEEE Trans. Circuits Syst. II: Analog and Digital Signal Processing*, vol. 48, no. 1, pp. 74 – 82, 2001.

[32] B. Minch, "A low-voltage MOS cascode bias circuit for all current levels," in *IEEE Int. Symp. Circuits and Systems*, pp. 619–622, May 2002.

[33] V. Srinivasan, R. Chawla, and P. Hasler, "Linear current-to-voltage and voltage-to-current converters," in *IEEE Midwest Symp. Circuits and Systems*, pp. 675–678, 2005.

[34] S. Nag and R. Rutenbar, "Performance-driven simultaneous placement and routing for FPGA's," *IEEE Trans. Computer-Aided Design of Integr. Circuits and Syst.*, vol. 17, no. 6, pp. 499–518, 1998.

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

[36] G. Serrano, P. Smith, H. Lo, R. Chawla, T. Hall, C. Twigg, and P. Hasler, "Automatic rapid programming of large arrays of floating-gate elements," in *IEEE Int. Symp. Circuits and Systems*, pp. 373–376, 2004.

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

[38] L. Martinez-Alvarado, J. Madrenas, and D. Fernandez, "Translinear signal processing circuits in standard CMOS FPAA," in *IEEE Int. Conf. on Electronics, Circuits, and Syst.*, 2009.

[39] G. Ying, A. Kuehlmann, K. Kundert, G. Gielen, E. Grimme, M. O'Leary, S. Tare, and W. Wong, "Guess, solder, measure, repeat - how do I get my mixed-signal chip right?," in *ACM/IEEE Design Automation Conference*, pp. 520 – 521, 2009.

[40] 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*, vol. 47, p. in press, Sept. 2012.

[41] C. Schlottmann, S. Nease, S. Shapero, and P. Hasler, "A mixed-mode FPAA SoC for analog-enhanced signal processing," in *IEEE Custom Integrated Circuits Conf.*, Sept. 2012.

[42] S. Chakrabartty and G. Cauwenberghs, "Sub-microwatt analog VLSI trainable pattern classifier," *IEEE J. Solid-State Circuits*, vol. 42, no. 5, pp. 1169 – 1179, 2007.

[43] A. Bandyopadhyay, J. Lee, R. Robucci, and P. Hasler, "MATIA: A programmable 80 uW/frame CMOS block matrix transform imager architecture," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 663 – 672, 2006.

[44] C. Schlottmann, C. Petre, and P. Hasler, "A high-level simulink-based tool for FPAA configuration," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 1, pp. 10 – 18, 2012.

[45] G. Serrano and P. Hasler, "A floating-gate DAC array," in *IEEE Int. Symp. Circuits and Systems*, pp. 357 – 360, May 2004.

[46] L. Wong, C. Kwok, and G. Rigby, "A 1-V CMOS D/A converter with multi-input floating-gate MOSFET," *IEEE J. Solid-State Circuits*, vol. 34, no. 10, pp. 1386 – 1390, 1999.

[47] C. Schlottmann and P. Hasler, "A highly dense, low power, programmable analog vector-matrix multiplier: The FPAA implementation," *IEEE J. of Emerging and Selected Topics in Circuits and Systems (JETCAS)*, vol. 1, no. 3, pp. 403 – 411, 2011.

[48] R. Chawla, C. Twigg, and P. Hasler, "An analog modulator/demodulator using a programmable arbitrary waveform generator," in *IEEE Int. Symp. Circuits and Systems*, vol. 6, pp. 6106 – 6109, 2005.

[49] E. Ozalevli, W. Huang, P. Hasler, and D. Anderson, "A reconfigurable mixed-signal VLSI implementation of distributed arithmetic used for finite-impulse response filtering," *IEEE Trans. Circuits Syst. I*, vol. 55, no. 2, pp. 510 – 521, 2008.

[50] D. Wei, V. Garg, and J. Harris, "An asynchronous delta-sigma converter implementation," in *IEEE Int. Symp. Circuits and Systems*, pp. 4903 – 4906, May 2006.

[51] W. Figueroa, D. Hsu, and C. Diorio, "A mixed-signal approach to high-performance low-power linear filters," *IEEE J. Solid-State Circuits*, vol. 36, no. 5, pp. 816 – 822, 2001.

[52] M. Kitsunezuka, S. Hori, and T. Maeda, "A widely-tunable, reconfigurable CMOS analog baseband IC for software-defined radio," *IEEE J. Solid-State Circuits*, vol. 44, no. 9, pp. 2496 – 2502, 2009.

[53] A. Wang and A. Chandrakasan, "A 180-mV subthreshold FFT processor using a minimum energy design methodology," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 310 – 319, 2005.

[54] S. Suh, A. Basu, C. Schlottmann, P. Hasler, and J. Barry, "Low-power discrete Fourier transform for OFDM: A programmable analog approach," *IEEE Trans. Circuits Syst. I*, vol. 58, pp. 1 – 1, 2011.

[55] C. Schlottmann, C. Petre, and P. Hasler, "Vector matrix multiplier on field programmable analog array," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, pp. 1522 – 1525, March 2010.

[56] R. Chawla, A. Bandyopadhyay, V. Srinivasan, and P. Hasler, "A 531 nW/MHz, 128x32 current-mode programmable analog vector-matrix multiplier with over two decades of linearity," in *IEEE Custom Integr. Circuits Conf.*, pp. 651 – 654, Oct. 2004.

[57] F. Adil, G. Serrano, and P. Hasler, "Offset removal using floating-gate circuits for mixed-signal systems," in *IEEE Southwest Symp. on Mixed-Signal Design*, pp. 190 – 195, 2003.

[58] A. Basu, R. Robucci, and P. Hasler, "A low-power, compact, adaptive logarithmic transimpedance amplifier operating over seven decades of current," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 10, pp. 2167 – 2177, 2007.

[59] P. Hasler and J. Dugger, "An analog floating-gate node for supervised learning," *IEEE Trans. Circuits Syst. I*, vol. 52, no. 5, pp. 834 – 845, 2005.

[60] R. Sarpeshkar, T. Delbruck, and C. Mead, "White noise in MOS transistors and resistors," *IEEE Circuits Devices Mag.*, vol. 6, no. 9, pp. 23 – 29, 1993.

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

[62] B. Sbarcea and D. Nicula, "Automatic conversion of matlab/simulink models to HDL models," in *Int. Conf. Optimization of Electrical and Electronic Syst.*, 2004.

[63] C. Chang, J. Wawrzynek, and B. Brodersen, "From BEE to BEE2: development of supercomputer-in-a-box," tech. rep., Berkeley Wireless Research Center, University of California, Berkeley, 2004.

[64] G. Asensi, J. Gomez-Diaz, J. Martinez-Alajarin, and R. Merino, "Synthesis on programmable analog devices from VHDL-AMS," in *IEEE Mediter. Electrotechnical Conf.*, pp. 27 – 30, May 2006.

[65] P. McGuire, "Pyparsing." http://pyparsing.wikispaces.com/, June 2012.

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

[67] P. Allen and D. Holberg, *CMOS Analog Circuit Design*. Oxford University Press, 2002.

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

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

[70] C. Twigg and P. Hasler, "Incorporating large-scale FPAAs in analog design courses," in *IEEE Int. Conf. Microelectronic Syst. Education*, pp. 171 – 172, June 2007.

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

[72] S.-Y. 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 Int. Symp. Circuits and Systems*, pp. 2145 – 2148, May 2009.

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

[74] R. Rutenbar, G. Gielen, and J. Roychowdhury, "Hierarchical modeling, optimization, and synthesis for system-level analog and RF designs," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 640 – 669, 2007.

[75] Y. Wei and A. Doboli, "Structural macromodeling of analog circuits through model decoupling and transformation," *IEEE Trans. Comp. Aided Design of Integrated Circuits and Syst.*, vol. 27, no. 4, pp. 712 – 725, 2008.

[76] A. Doboli and R. Vemuri, "Behavioral modeling for high-level synthesis of analog and mixed-signal systems from VHDL-AMS," *IEEE Trans. Comp. Aided Design of Integrated Circuits and Syst.*, vol. 22, no. 11, pp. 1504 – 1520, 2003.

[77] B. Bond, Z. Mahmood, Y. Li, R. Sredojevic, A. Megretski, V. Stojanovi, Y. Avniel, and L. Daniel, "Compact modeling of nonlinear analog circuits using system identification via semidefinite programming and incremental stability certification," *IEEE Trans. Comp. Aided Design of Integrated Circuits and Syst.*, vol. 29, no. 8, pp. 1149 – 1162, 2010.

[78] P. Li and L. Pileggi, "Compact reduced-order modeling of weakly nonlinear analog and RF circuits," *IEEE Trans. Comp. Aided Design of Integrated Circuits and Syst.*, vol. 24, no. 2, pp. 184 – 203, 2005.

[79] X. Huang, C. Gathercole, and H. Mantooth, "Modeling nonlinear dynamics in analog circuits via root localization," *IEEE Trans. Comp. Aided Design of Integrated Circuits and Syst.*, vol. 22, no. 7, pp. 895 – 907, 2003.

[80] C. Schlottmann and P. Hasler, "High-level modeling of analog computational elements for signal processing applications," *IEEE Trans. Comp. Aided Design of Integrated Circuits and Syst.*, p. in review, 2012.

[81] C. Schlottmann and P. Hasler, "FPAA empowering cooperative analog-digital signal processing," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, pp. 5301 – 5304, March 2012.

[82] B. Rumberg, D. Graham, V. Kulathumani, and R. Fernandez, "Hibernets: Energy-efficient sensor networks using analog signal processing," *IEEE J. of Emerging and Selected Topics in Circuits and Systems (JETCAS)*, vol. 1, no. 3, pp. 321 – 334, 2011.

[83] D. Graham, P. Hasler, R. Chawla, and P. Smith, "A low-power programmable band-pass filter section for higher order filter applications," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 6, pp. 1165 – 1176, 2007.

[84] J. Lazzaro, S. Ryckebusch, M. Mahowald, and C. Mead, "Winner-take-all networks of O(n) complexity," *Adv. Neural Inf. Process. Syst*, vol. 1, p. 703711, 1989.

[85] P. Hasler, C. Schlottmann, and S. Koziol, "FPAA chips and tools as the center of an design-based analog systems education," in *IEEE Int. Conf. Microelectronic Systems Education (MSE)*, pp. 47 – 51, 2011.

[86] C. Enz, F. Krummenacher, and E. Vittoz, "An analytical MOS transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications," *Analog Integrated Circuits and Signal Processing*, vol. 8, no. 1, pp. 83–114, 1995.

[87] K. Odame and P. Hasler, "A bandpass filter with inherent gain adaptation for hearing applications," *IEEE Trans. Circuits Syst. I*, vol. 55, no. 3, pp. 786 – 795, 2008.

[88] A. Basu, K. Odame, and P. Hasler, "Dynamics of a logarithmic transimpedance amplifier," in *IEEE Int. Symp. Circuits and Systems*, pp. 1673 – 1676, May 2007.

[89] M. Kucic, A. Low, P. Hasler, and J. Neff, "A programmable continuous-time floating-gate fourier processor," *IEEE Trans. Circuits Syst. I: Analog and Digital Signal Processing*, vol. 48, no. 1, pp. 90 – 99, 2001.

[90] S. Rapp, K. McMillan, and D. Graham, "SPICE-compatible modelling technique for simulating floating-gate transistors," *Electronics Letters*, vol. 47, no. 8, pp. 483 – 485, 2011.

VITA

Craig Schlottmann was born and raised in Orlando, Florida, where he graduated from Bishop Moore High School in 2002. He received the B.S. degree (*summa cum laude*) in electrical engineering from the University of Florida in 2007. Craig's honors thesis was on the noise characteristics of single-walled carbon nanotubes. In the summer of 2007, he interned at the Rockwell Collins Advanced Technology Center in the Intelligence, Surveillance, and Reconnaissance group. Craig then received the M.S. and Ph.D. degrees in electrical engineering from Georgia Tech in 2009 and 2012, respectively. During his graduate studies, he interned at the MIT Lincoln Laboratory in the Embedded and High Performance Computing group.

His research interests include low-power analog signal processing, mixed-signal IC design, and low-power embedded electronics.