FIELD–PROGRAMMABLE ANALOG ARRAYS: A FLOATING–GATE APPROACH

A Dissertation
Presented to
The Academic Faculty

by

Tyson S. Hall

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Electrical and Computer Engineering
Georgia Institute of Technology
July 12, 2004

FIELD–PROGRAMMABLE ANALOG ARRAYS: A FLOATING–GATE APPROACH

Approved by:

Professor David V. Anderson, Advisor	Professor Aaron Lanterman
Professor David Citrin	Professor Milos Prvulovic
Professor Paul Hasler	Professor Sudhakar Yalamanchili

Date Approved: July 9, 2004

ACKNOWLEDGEMENT

Research is rarely pursued in a vacuum. I am grateful to each and every friend for their support, encouragement, assistance, and influence. My advisor, Dr. David Anderson, has been with me every step of the way. I am most thankful for his thoughtful guidance, understanding, and friendship. My co–advisor, Dr. Paul Hasler, has been a continual source of knowledge and encouragement. If it were not for Paul, I would still be ignorant of floating–gate transistors and even FPAAs.

I count it a privilege to know each one of the many graduate students in the Cooperative Analog/Digital Signal Processing (CADSP) group. In particular, I recognize David Graham and Paul Smith for their assistance during my early days of VLSI layout. I am also grateful for the talent and skills that Chris Twigg, Jordan Gray, and Dave Abramson have brought to the FPAA research team. We have spent many long hours together working on layout, testing FPAA chips, and discussing all things FPAA. I also appreciate the time and effort that Sourabh Ravindran has spent working on analog implementations of his feature extraction algorithms. Sourabh is one of my longest and best friends in the CADSP group.

If I did not have such a great team of editors, reading this document would have been much more tedious. In particular, Julie Hall (cousin), Vicki Hall (mother), Burton Hall (father), Barry Hall (brother), and Dr. Roger Hall (uncle) all graciously read rough drafts of this thesis and/or gave editorial assistance despite their varying levels of interest in the topic.

I thank my friends and family who have kept me in their thoughts and prayers during my tenure at Georgia Tech. To the Creator of all that is and was and is to come, I am most grateful. As my knowledge increases, my awe of His handiwork abounds.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Field–programmable analog arrays (FPAAs) provide a method for rapidly prototyping analog systems. Currently available commercial and academic FPAAs are typically based on operational amplifiers (or other similar analog primitives) with only a few computational elements per chip. While their specific architectures vary, their small sizes and often restrictive interconnect designs leave current FPAAs limited in functionality, flexibility, and usefulness. Recent advances in the area of floating–gate transistors have led to an analog technology that is very small, accurately programmable, and extremely low in power consumption. By leveraging the advantages of floating–gate devices, a large–scale FPAA is designed that dramatically advances the current state of the art in terms of size, functionality, and flexibility. A large–scale FPAA is used as part of a mixed–signal prototyping platform to demonstrate the viability and benefits of cooperative analog/digital signal processing. This work serves as a roadmap for future FPAA research. While current FPAAs can be compared with the small, relatively limited, digital, programmable logic devices (PLDs) of the 1970s and 1980s, the floating–gate FPAAs introduced here are the first step in enabling FPAAs to support large–scale, full–system prototyping of analog designs similar to modern FPGAs.

CHAPTER 1

RAPID PROTOTYPING OF ANALOG SYSTEMS

The process of designing, fabricating, and testing an analog chip requires certain expertise and is often long and expensive. As shown in Fig. 1, the process is not unlike designing digital ASICs (application–specific integrated circuits), except that there are fewer tools and libraries available to the designer. The traditional analog design cycle often requires several iterations of the fabrication process, which with the simulation, VLSI layout, and testing phases can easily consume a year or more for typical IC designs. However, the use of a reconfigurable analog chip, dubbed a field–programmable analog array (FPAA), would dramatically reduce the design cycle by removing the fabrication stage from the iterative process. Thus, many designs may be tested and modified within a single day.

Like field–programmable gate arrays (FPGAs), FPAAs are not optimal for all solutions. They are, however, very useful for many situations, and a solution can be found for many problems not requiring full functionality. Relative to custom–designed analog circuits, a design implemented on an FPAA results in higher parasitics as well as increased die area for a given design; therefore, the design always possesses some inefficiencies (i.e., lower bandwidth and higher consumed power). On the other hand, since analog circuit design is often time–consuming, these adverse characteristics are well balanced by markedly decreased time to market.

FPAAs have been of interest for some time, but historically, these devices have had very few programmable elements and limited interconnect capabilities, making them limited in their usefulness and versatility. The next–generation FPAA needs to correct these problems in order to extend the usefulness and acceptance of FPAAs. As shown in Fig. 2, traditional FPAAs resemble the early PLDs in that they are focused on small systems such as low–order filtering, amplification, and signal conditioning. However,

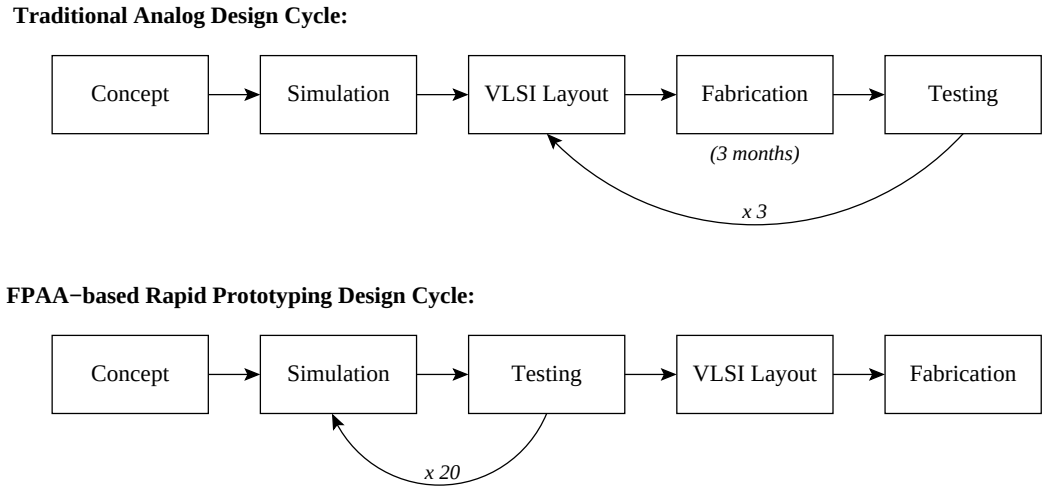
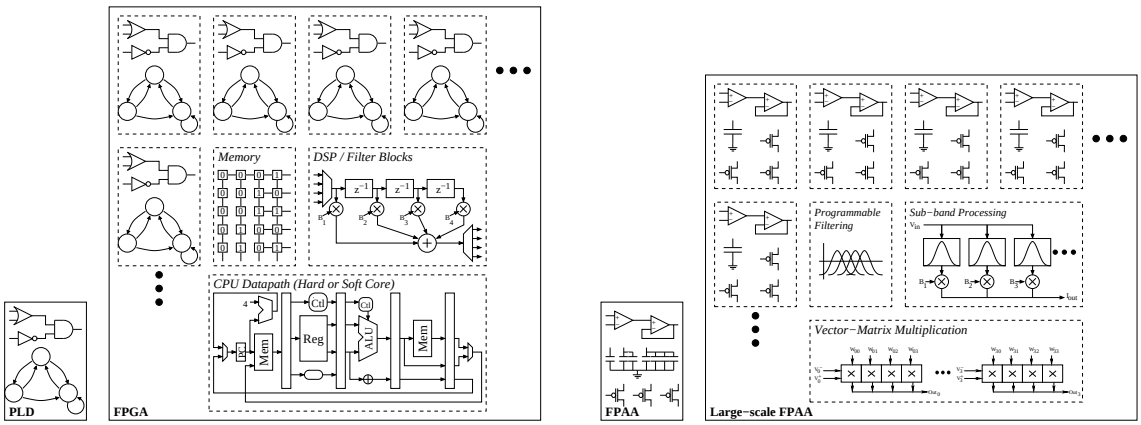


Figure 1. This figure illustrates the advantages of designing analog ICs using an FPAA–based rapid prototyping technology as opposed to the traditional design cycle of VLSI layout and fabrication. The traditional analog design cycle often requires 3 or more iterations of the fabrication process which extends the development process to over a year. With an FPAA–based system, designs can be synthesized, tested, and modified 20 or more times within a matter of days instead of years.

the class of large–scale FPAAs that we are exploring here are more analogous to modern FPGAs. These FPAAs are much larger devices with the functionality needed to implement high–level system blocks such as programmable high–order filtering, Fourier processing, and signal analysis in addition to having a large number of fine– and medium–grain, programmable analog blocks (e.g., operational transconductance amplifiers (OTAs), transistor elements, capacitors, etc.).

1.1 Low–Power Signal Processing

The future of FPAAs lies in their ability to speed the implementation of advanced, low–power signal processing systems. In this thesis, an FPAA architecture is presented for achieving flexible, large–scale FPAAs targeted at mainstream signal processing systems. These FPAAs are intended to impact analog signal processing in two ways: first, they perform the function of all rapid prototyping devices in reducing development time. Second, they are a platform for implementing advanced signal processing functions—usually realized only in digital systems—with analog circuits.



(a) (b)

Figure 2. (a) Digital PLDs can be used to implement small, carefully defined pieces of a complex system, while FPGAs can be used to implement entire systems including processor datapaths, complex DSP functions, and more. Modern FPGAs can be 100 - 10,000 times larger and more complex than the PLDs of the 1970s and 1980s. (b) Analagously, traditional FPAAs resemble the early PLDs in that they are focused on small systems such as low–order filtering, amplification, and signal conditioning. However, the FPAAs based on floating–gate devices presented here are much larger devices with the functionality needed to implement high–level system blocks such as programmable high–order filtering and Fourier processing in addition to having a large number of programmable op–amp and transistor elements.

The primary benefit of implementing analog signal processing systems is the potential for large savings in power consumption. For digital signal processing (DSP) microprocessors, Gene's law postulates that the power consumption, as measured in milliwatts per million multiply–accumulate (mW/MMAC) operations, is halved about every 18 months, as shown in Fig. 3 [28]. These advances largely follow Moore's law, and they are achieved by using decreased feature size, intelligent clock gating, and other refinements. Unfortunately, a problem looms on the horizon; the power consumption of the analog–to–digital (A/D) converter does not follow Gene's law and will soon dominate the total power budget of digital systems. While A/D–converter resolution has been increasing at roughly 1.5 bits every five years, the power performance has remained the same, and soon, physical limits will further slow progress.

For analog systems to be desirable to the largely DSP–oriented commmunity, they not only need to have a significant advantage in terms of size and power, but

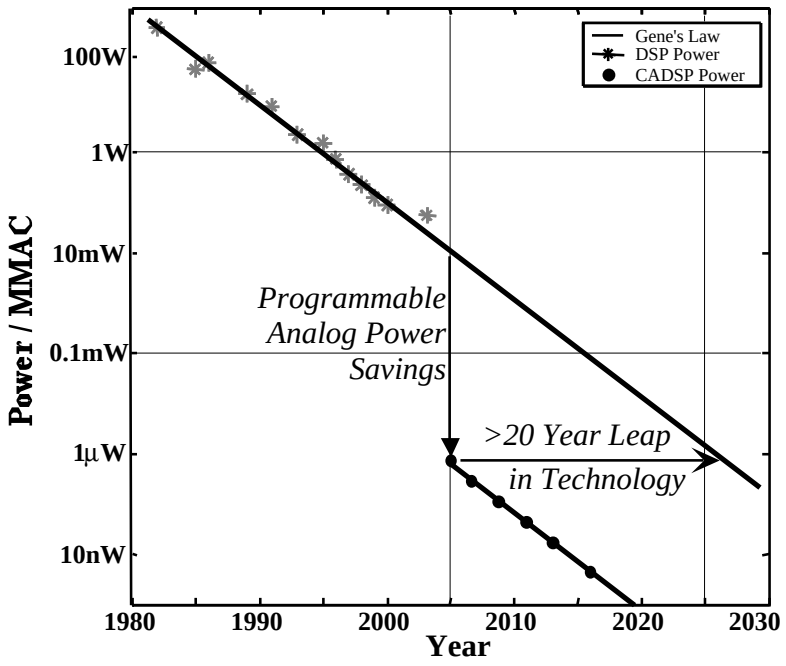


Figure 3. Data from [28] showing the power consumption trends in DSP microprocessors along with data taken from a recent analog, floating–gate integrated chips developed by the CADSP team [13, 22, 47, 51, 77].

they must also be relatively easy to use and easily integrated into a larger digital system. In addition, they must be shown to be accurately programmable and effective at implementing many of the key systems found within DSP. As shown in Table 1, the functionality desired for any technology focused on signal processing includes monolithic filters, linear and nonlinear scalar functions, vector–matrix operations (i.e., transforms, distance metrics, winner-take-all, principle component analysis, etc.), linear–phase filters, adaptation, and tap delay lines for FIR systems.

1.2 Cooperative Analog/Digital Signal Processing

This thesis is part of a larger effort within the Cooperative Analog/Digital Signal Processing (CADSP) group at the Georgia Institute of Technology. The goal of the CADSP group is to investigate the partitioning of signal processing systems between

Functionality	DSP μP	Trad. Analog	Large–scale FPAA
Programmable	●	–	●
Monolithic Filters	○	●	●
Linear Scalar Operations	●	○	●
Nonlinear Scalar Operations	○	●	●
Vector–Matrix Operations	○	○	●
Linear–phase Filters	●	–	○
Adaptivity	○	–	○
Tap Delay Lines	●	○	○

Key:
– = No or very limited support
○ = Possible
● = Efficient, well–suited to technology

Table 1. Summary of Signal Processing Functionality

the analog and digital domains. Most current signal processing systems that generate digital output place the ADC as close to the analog input signal as possible to take advantage of the computational flexibility available in digital processors (see Fig. 4). However, the development of large–scale FPAAs—and the CAD tools needed for their ease of use—would allow engineers the option of performing some of the computations in reconfigurable analog hardware prior to the analog–to–digital (A/D) converter, resulting in both a simpler A/D converter and a substantially reduced computational load on the digital processors that follow. By leveraging the power efficiencies mentioned in the previous section, some analog signal processing systems have been shown to achieve as much as five orders of magnitude over typical DSP microprocessor implementations [5, 22, 77]. As illustrated in Fig. 3, this corresponds to a 20 year leap forward on the power curve predicted by Gene's law [37]. Additionally, the output of such an analog system can be higher–level information, such as Fourier coefficients or phonemes of speech. This information can potentially be converted into the digital domain with a much lower resolution and/or conversion speed than would be needed in the traditional system where a literal sampling of the incoming

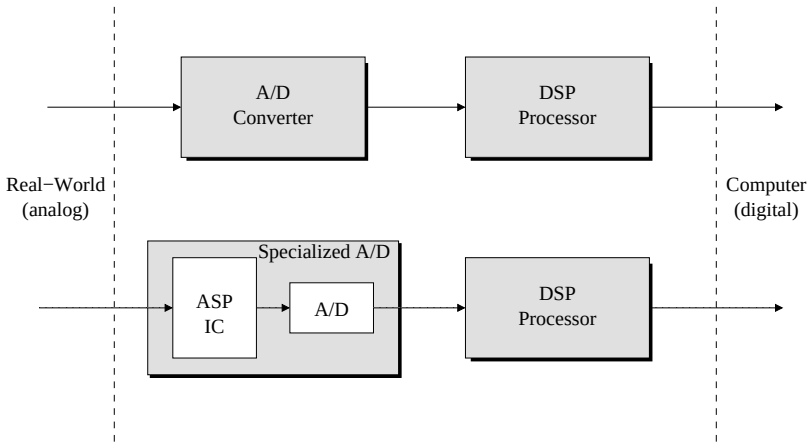


Figure 4. In most traditional digital signal processing systems, the analog–to–digital converter (ADC) is placed as close to the real–world, analog inputs as possible. However, significant power savings can be achieved by moving some of the signal processing functionality into the analog domain (in front of the ADC). Conceptually, this analog signal processing can be combined with the ADC to form a "specialized ADC" as shown here.

signal is needed [45]. Thus, a simpler and much smaller A/D converter can be used.

As an example of the advantages of moving the A/D boundary, one can consider a software radio system. In the traditional implementation, the analog signals coming from multiple (e.g., four to eight) antennas must each be converted to the digital domain. The signals coming from the antennas are still modulated by the carrier frequency, and thus, multiple high–speed A/D converters must be used. In a CADSP system, however, the demodulation and beam–forming functions can be implemented in floating–gate analog circuitry [41, 45, 51, 63]. The output from the analog domain is then a single, baseband signal that can be converted using a much slower A/D converter.

To realize the power savings of analog systems in advanced signal processing systems, the CADSP group focuses on the use of floating–gate devices as the enabling technology. Recent advances in floating–gate circuit technology have yielded promising results in the implementation of complex analog signal processing systems. Floating–gate circuits have already been demonstrated in the areas of noise suppression, speech processing, image processing, and adaptive systems [6, 20, 23, 39, 47, 77,

81]. Work has also progressed on creating fast, accurate methods of programming large arrays of floating–gate transistors [76]. This is an important factor in determining the viability of using floating–gate devices in large, complex systems, particularly in mass–production environments.

The CADSP group brings together researchers from three areas of electrical engineering: analog circuit design, digital signal processing, and computer/systems engineering. Within the group, the analog circuit designers focus on the development, characterization, and testing of low–level floating–gate circuits as the core technology of CADSP systems. The digital signal processing researchers focus on the algorithmic translation of digital signal processing systems into the analog domain. They are particularly interested in capitalizing on the large–scale system parallelization that is possible within the analog domain, as well as the real–time, continuous–flow nature of analog systems and its affect on traditional algorithms. Finally, the computer engineering researchers focus their attention on the system–level development of the hardware systems and computer–aided design (CAD) tool flows necessary to realize practical CADSP systems.

This thesis falls within the computer engineering sub-area of the CADSP group. As a practical matter, the investigation of a reconfigurable system for rapidly prototyping analog and digital systems is extremely important in moving the CADSP effort forward. To date, there are no large–scale FPAAs available that will allow designers to test complex analog and mixed–signal systems. Furthermore, given the programmable nature of floating–gate devices, the development of FPAAs based on this core technology is a natural and important step toward the realization of a widely accessible method of creating analog and mixed–signal designs. FPAAs will provide a platform for exploring cooperative analog/digital signal processing systems that optimally balance the low–power, real–time computational nature of analog circuits with the flexibility and robustness of digital systems.

CHAPTER 2
HISTORY AND MOTIVATION

Reconfigurable hardware has long been of interest to circuit designers and engineers. In the digital domain, programmable logic devices (PLDs) have made a large impact on the development of custom digital chips by enabling a designer to try custom designs on easily reconfigurable hardware. Since their conception in the late 1960s and early 1970s, PLDs have evolved into high–density field–programmable gate arrays (FPGAs) [9, 14, 80]. Modern FPGAs are widely used in the laboratory for rapidly prototyping digital hardware, as well as in production goods to decrease time-to-market and to allow products to be easily upgraded after being deployed.

In the analog domain, however, progress has been much slower. While early analog integrated circuits (ICs) were often tunable with adjustable biases, truly reconfigurable analog circuitry in the form of field–programmable analog arrays (FPAAs) did not emerge until the late 1980s [35, 75], and commercial offerings did not reach the market until 1996 [61].

2.1 Background on FPAAs

FPAAs can be broadly classified into two categories: continuous–time devices and discrete–time devices [33]. There are academic and commercial examples of both categories, as well as advantages unique to each design methodology. This section will focus more on continuous–time FPAAs, because the large–scale FPAA architecture developed herein is a continuous–time FPAA; however, previous work on discrete–time FPAAs will be summarized for comparison sake. Previous FPAAs have varied greatly in terms of computational granularity and capability, interconnect structure, performance, and application focus.

Fundamentally, FPAAs include two functions: routing and computation. (The

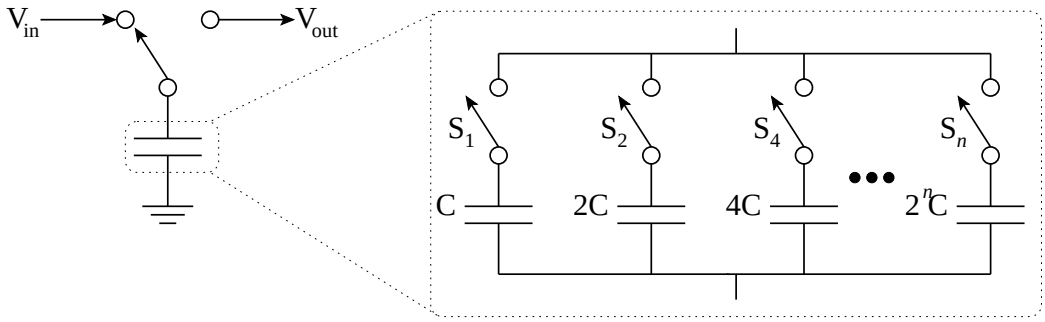


Figure 5. Most discrete–time FPAAs use switched–capacitor designs. As the switch momentarily rests on the V_{in} terminal, the input voltage charges the capacitor. Then the switch moves to the V_{out} terminal where it is discharged through the next phase of circuitry. The programmability within switched–capacitor designs is usually achieved using an array of appropriately sized capacitors. Thus, the effective capacitance at each switch can be varied by setting the n digital memory cells controlling switches S_1 to S_n. In essense, this capacitor bank is equivalent to having a Digital–to–Analog Converter (DAC) being included in each computational block.

next section will show that some designs combine these two functions into a single unit; however, both elements are still present whether explicitly or implicitly.) The routing elements are typically networks of switches connected together by signal lines with the network architecture and switch types varying dramatically across different FPAAs. The switch networks then connect to the computational elements of the system. If there is more than one type of computational element, the computational elements are usually grouped together to form a computational analog block (CAB) that is analagous to the computational logic blocks found on FPGAs. If there is only one type of computational element, the element and the CAB are one and the same; however, for convenience, the computational elements will still be referred to as CABs here.

2.2 Discrete–time FPAAs

Discrete–time FPAAs are typically switched–capacitor designs. For these circuits, the incoming voltage is sampled by opening and closing a switch that connects the input to an initial capacitor (Fig. 5). The switch and capacitor form a type of analog register, and the system's signal path is partitioned by these registers. The basic computational

	Typical Computational Elements	Advantages	Disadvantages
Fine	Transistors	Small Simple CAB design Generic building blocks	Large # of switches Large parasitics
Medium	Op-amp OTA Current conveyor	Semi-generic building blocks Moderate CAB design Large variety of CAB / interconnect designs	Limited size Severe functionality / performance trade–off
Coarse	Fourier Processor "Expert cell"	Higher performance Easier user interface	Limited flexibility Limited functionality

Table 2. Summary of FPAA Granularity: The granularity of the computational logic used in an FPAA impacts the size, performance, flexibility, and functionality of the device.

elements are usually operational amplifiers and analog registers, which synthesize a linear resistor whose value is determined by the switching rate and capacitor value. The synthesis of linear variable resistors gives switched–capacitor FPAAs greater flexibility than traditional continuous–time FPAAs; however, they can also be harder to design proficiently, because the switches and capacitors can introduce noise and nonlinearities into the system that must be overcome [61]. In addition, these designs have a limited bandwidth based on the sampling rate, are complicated by the need for continuous–time anti–aliasing and reconstruction filters at the input and output, and can be large if programmable capacitor arrays are included [35, 61].

In the late 1990s, several switched–capacitor FPAAs were introduced by both academic and commercial entities. In the academic arena, basic computational elements vary from the simple operational amplifier [21, 50] to the more complex blocks, such as a lossless integrator and lossy integrator connected in a loop [53]. These devices also can have programmable capacitor and/or programmable resistor arrays, which add programmability [21, 35]. In the commercial arena, Motorola was one of the first companies to bring a general–purpose FPAA to market with their MPAA020 and MPA1000 series [4, 10, 61]. Since then, a spin–off company named Anadigm has

marketed these switched–capacitor FPAAs [2]. The newest Anadigm devices have CABs with two differential operational amplifiers, programmable capacitor banks, a successive approximation register (for implementing an A/D converter), and a high–speed comparator [3]. However, even the latest devices are relatively limited with only four CABs per chip and are targeted at basic signal conditioning and filtering applications.

Switched–capacitor designs are not the only discrete–time FPAAs. Switched–current circuits can also be used to build an FPAA. The advantages of this technique include not requiring operational amplifiers, capability of fabrication on standard digital CMOS processes, and elimination of distortion on the signals due to parasitic resistances. To their detriment, these designs can produce less accuracy than switched–capacitor circuits, and since the signals are all currents, a given output stage can drive only one input stage [12].

2.3 Continuous–time FPAAs

Continuous–time FPAAs typically use an array of fixed components (often operational amplifiers and/or transistors) that are interconnected by a switching matrix. The switches are usually controlled by digital registers, which can be loaded by an external controller, thus allowing the FPAA to be configured to implement a number of different designs. This type of FPAA is advantageous because potential sampling artifacts are avoided; anti–aliasing filters are not needed; common, relatively easy design processes can be used (e.g., standard CMOS processes); and large signal bandwidths can be supported with predictable performance [61]. However, the switching networks introduce parasitic impedances into the signal path that limit the bandwidth and add noise to the system. Some of the literature has focused on minimizing the number of switches in the signal path, but this can severely limit the flexibility of the FPAA [24, 55, 58, 67].

2.3.1 Computational Granularity and Capability

The granularity of the computational logic that forms the basis of the FPAA's design is an important design characteristic. As summarized in Table 2, the finest–grain architectures typically use transistors as the core computational cell. While these designs offer the most flexibility and generality, synthesizing a sufficiently complex system requires a large number of transistors to be wired together. Thus, a large number of switches are introduced into the signal path. The switch parasitics and finite resistance increases the noise within the system and limits the performance/bandwidth [24, 49, 67]. Fine–grain FPAAs have been primarily relegated to research in evolvable hardware [48, 72, 78], where the lowest–level building blocks are desirable for generating unique designs using non-traditional design methodologies. Systems that are designed using genetic algorithms are not as negatively affected by the parasitics and non-ideal resistances of switches, since these parameters are taken into account and even exploited throughout the evolutionary design process.

On the coarse–grain extreme, one finds FPAAs such as IMP's EPAC devices, which contain an "expert cell" as the core computational block [49]. In the IMP50E10 device, this cell is a very high–level block with limited interconnects that is aimed directly at signal conditioning applications. The logic within the cell can be configured to function as an amplifier with an optional low–pass filter or as a comparator with optional hysteresis. There is also a dedicated D/A converter for defining the reference point for the comparator. These coarse–grain designs sacrifice flexibility and generality in favor of increased, more predictable performance [49].

The majority of FPAAs fall in–between these two extremes. A number of FPAAs use an operational transconductance amplifier (OTA) as the basic computational element [26, 64, 65, 67, 70, 71]. OTAs work well as the core computational cell, because their transconductance can be programmed either by varying the analog bias voltage or by changing the gain of the output current mirrors [1, 65]. In addition, it has

been shown that OTAs can implement a wide range of linear and nonlinear circuits. Several FPAA designs have focused on synthesizing linear circuits and use OTAs to implement amplification, integration, and filtering functions [64, 65, 67]. Ray et al. have proposed a more generalized scheme in which linear circuits are synthesized using an OTA–based lossless integrator and an OTA–based lossy integrator as the basic functional blocks. They also use an OTA–based multiplier and OTA–based integrator as the basic functional block for synthesizing nonlinear circuits such as amplitude and frequency modulation [70]. Sanchez–Sinencio et al. have used OTAs to implement nonlinear functions such as multiplication, division, square root, exponentiation, and piece–wise linear operations [71].

Similiarly, several FPAA designs have been proposed using a current–conveyor structure as the basic building block. Current conveyors are similar to OTAs; however, when used as an amplifier they exhibit a constant bandwidth that is independent of gain. They also do not require compensation circuitry to insure stability, and thus, they can operate at higher frequencies [34]. Gaudet et al. have proposed a current–conveyor based FPAA in which each CAB contains a second–generation current–conveyor, two programmable capacitors, and two programmable resistors (transconductors) [34]. This CAB is shown to implement amplification and first–order filtering functions, as well as log and anti–log functions with the addition of switchable diodes. Premont et al. also describe an FPAA based on current–conveyors [68]. The core cell includes tuneable resistors and a current–conveyor. It has been demonstrated that this cell can be configured to implement a tuneable capacitor, and thus is suitable for amplification and filtering functions.

Other medium–grain computational blocks have been used in FPAAs as well. Pierzchala et al. used an OTA–based amplifier/integrator cell that does not require switches in the signal path [67]. Quan et al. proposed a current–mode FPAA that uses a cascode current–mode integrator as the basic building block [69]. This core cell

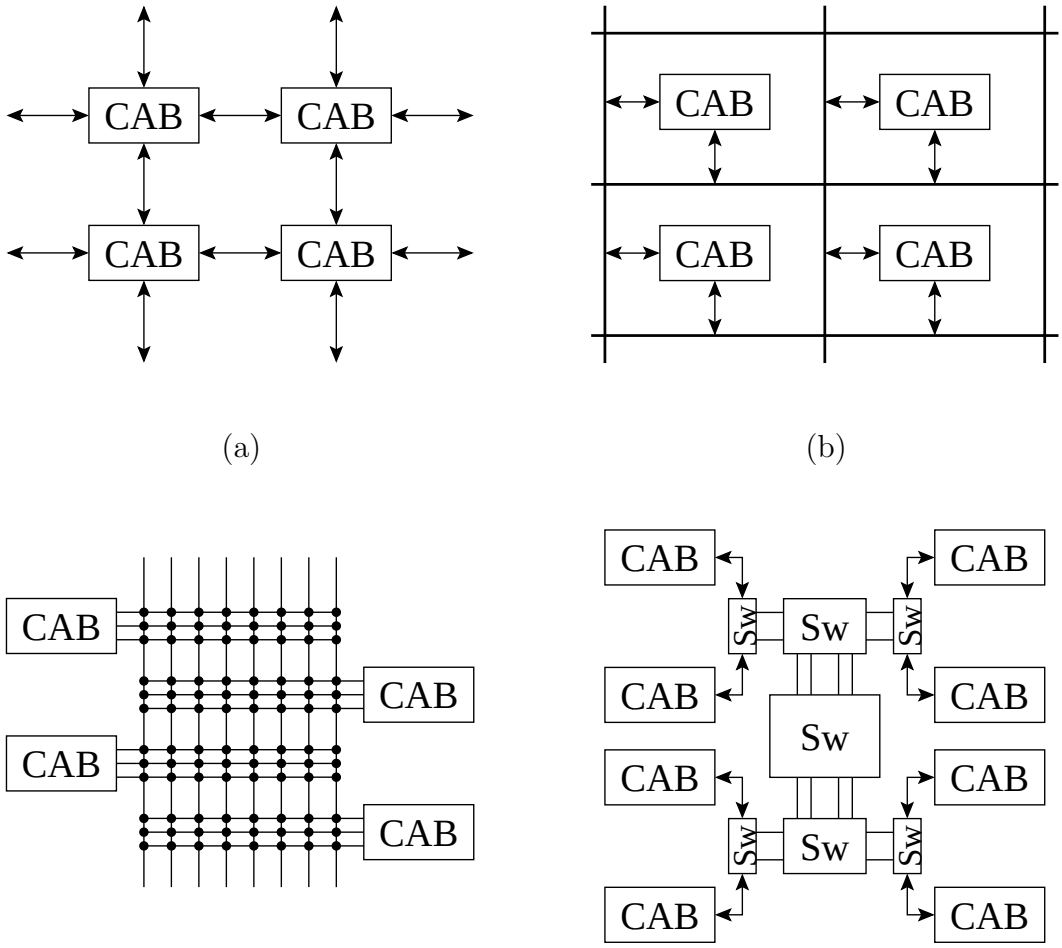


Figure 6. A number of different interconnect schemes have been used in FPAAs including (a) local connections, (b) global connections, (c) cross–bar networks, and (d) fat–tree interconnects.

can implement amplification, integration, and attenuation with a minimum number of switches in the signal path [24]. In a more specialized chip design for prototyping machine learning algorithms, Bridges et al. used a mix of components including transistors, a differential pair, a current mirror, a bias current source, and configurable capacitors [11]. They used these blocks to synthesize single learning primitives such as correlational learning, regression learning, and clustering.

2.3.2 Interconnect Structure

Aside from CAB components, a number of different interconnect structures have also been proposed for FPAAs (see Fig. 6). The most common interconnect switch is a MOS transistor controlled by a digital memory [4, 12, 34, 55, 65, 69]. Unfortunately, switch resistance in these devices can range from 1000 to 5000 Ω making them a limiting factor in designing large, complex FPAAs [21]. Lee and Gulak used this type of pass–transistor switch in [55, 58]; however, the parasitic effects of these switches greatly limited the performance and capability of their FPAA [59]. As a result, they replaced the pass transistors with four–transistor transconductors, which increased the performance and functionality of their FPAA [56]. The new design reduced the parasitic effects, increased the linearity, and increased the noise immunity. In addition, a transconductor switch exhibits a linear resistance, thus each switch can also be used as a variable resistor by driving the gate voltage with a multi-valued memory (or another internal or external signal). However, the large transistors needed for low–frequency operation and the addition of a multi-valued memory for each switch greatly increases the area required for the interconnects [59].

Other switch designs have been proposed as well. Premont et al. used a current conveyor as the switching element [68]. This was particularly novel, because they used the current conveyor for both the switching element and the active computational element. In an effort to provide a radiation–tolerant FPAA for space applications, Edwards et al. proposed the use of metal–to–metal antifuses for the switches [21]. The antifuse design they used also has the benefit of a relatively low resistance (in the 15 to 25 Ω range).

In addition to using different switches, interconnect schemes have also varied in overall architecture. In the Premont et al. FPAA previously mentioned [68], the use of a current conveyor for both the switching element and active computational element results in CABs and switching interconnects that are indistinguishable from one

another. The overall architecture becomes a homogeneous mesh of logic with a minimum number of switches introduced into the signal path. Various other approaches have also been tried in an effort to minimize the number of switches. Quan et al. proposed the use of local interconnects. In their architecture, each CAB can be connected to its eight neighbors and itself [24]. This would seem to be a severe limitation on the flexibility of this FPAA; however, they focus on the large number of analog circuits with mostly local interconnections [69]. A similar idea was proposed by Becker et al. [7]. They used digitally tunable transconductors as the computational element. In this design, CABs were arranged in a hexagonal pattern; each CAB consists of seven programmable transconductors—six for connecting to each neighboring CAB and one for local feedback. This FPAA is designed for implementing Gm–C filters (with the required capacitance being provided by the parasitic impedance of the transconductors). Again, the lack of dedicated routing hardware is a limiting factor in this design, but if the target applications are constrained to be Gm–C filters, it is satisfactory. Pierzchala et al. tried an even more limiting architecture in which no electronic switches were included in the signal paths [67]. While these designs may provide benefits in bandwidth and signal–to–noise ratio (SNR), they lack the flexibility and generality needed in a truly general–purpose FPAA.

In another design, Pierzchala et al. introduced an interconnect scheme with both local and global signal paths [66]. This configuration provided local routing paths for a cell's four neighbors (north, south, east and west), as well as connections to global busses that run horizontally, vertically, and diagonally. This two–tiered hierarchy increases the routing flexibility within the FPAA. An even more flexible interconnection network is the crossbar switch [70]. The crossbar switch offers a nonblocking, fully connectable architecture; however, for a large number of inputs and outputs its size can be too big ($O(N^2)$ growth rate) [55]. Lee and Gulak tried to solve this problem by using an area–universal fat–tree network [58]. They used a heirarchical fat–tree

network of small crossbar switches where the CABs were connected as the leaves of the tree. In an additional effort to minimize the size required by the switch networks, the number of connections was constrained [55]. Unfortunately, their prototype was too small to fully test this interconnect concept.

2.4 Complimentary Research

There are several areas of research progressing in fields complimentary to FPAAs. On the hardware side, there are different attempts at combining programmable digital and analog hardware. Chow et al. have proposed a single–chip field–programmable mixed–analog–digital array (FPMA) for prototyping mixed–signal systems [15]. This effort combines an array of standard digital FPGA cells with a programmable array of operational amplifiers, capacitors, resistors, and diodes [16]. The reconfigurable analog hardware is limited in nature and is not the primary emphasis in this case. Instead, the focus of this work is on configurable A/D and D/A converters that reside between the blocks. Lee has also done some work in this area; however, his focus is also on the design of reconfigurable A/D and D/A converters and not the design of complex analog arrays [57].

Faura took a slightly different approach to the FPMA concept by combining digital FPGA logic, programmable analog logic, and a digital microcontroller on the same integrated circuit (IC) [27]. Once again, the reconfigurable analog hardware was not very complex. In this case, the analog circuitry was used to perform limited signal conditioning on the incoming signals, and as such, it was designed using a very large–grained architecture that limited the degree of programmability.

Dudek and Hicks took yet another approach to reprogrammable hardware with their analog microprocessor [17]. Their system was modeled after a standard digital microcontroller with digital memory and digital fetch and decode control logic; however, the actual datapath was analog. They used a sampled–data approach similar

to switched–capacitor systems to temporarily store values in analog registers between instructions [18]. While this research may not be directly relevant to FPAAs now, it does hold promise for the development of future FPAAs that contain advanced intellectual property (IP) modules, such as analog microprocessors similar to modern digital FPGAs.

On the software side, several different efforts are focused on various aspects of the CAD tool flow necessary for the rapid prototyping of analog and mixed–signal systems. Within the simulation and synthesis processes, it is important to have an accurate and efficient method of modeling the behavior of the generated system. Various models have been suggested including those by Enright and Mack in [25] and Long in [60]. Additionally, work has been done in the synthesis of analog circuits from user input in the form of an analog high–level description language (AHDL). Several AHDLs exist including Cadence's SpectreHDL. In one effort, Binns et al. introduced a high–level, top–down methodology for designing analog systems that are described in SpectreHDL [8]. However, with the creation of an IEEE standard AHDL that has been dubbed VHDL–AMS (VHSIC hardware description language – analog and mixed–signal), proprietary AHDLs will probably fade in popularity. Thus, the efforts of Ganesan and Vemuri in building a VHDL–AMS synthesis tool [30, 31] seem more relevant to the tool flow of future FPAAs.

Ganesan and Vemuri are developing an automated CAD tool flow for analog circuits that they call the VHDL–AMS Synthesis Environment (VASE) [30]. VASE is built on their previous work on a performance–oriented router for FPAAs [29]. The router was designed for and tested using Motorola's MPAA020 FPAAs (now sold by Anadigm [2]). As was previously noted, these devices were based on a small number of operational amplifiers and switched–capacitor technology. The design of a large–scale, functionally rich FPAA will provide researchers in the CAD tool flow arena with a more complex and more interesting platform to test and develop their

automated software applications. In a similar effort, Ganesan and Vemuri have published work focusing on the partitioning of systems between the analog and digital domains [32, 33]. Again, a large–scale, mixed–signal prototyping platform will benefit this area of research by providing a viable platform to investigate the advantages and difficulties involved in the analog–digital partitioning of systems.

2.5 Performance

The performance characteristics of FPAAs in regards to bandwidth, noise immunity, and signal–to–noise ratio (SNR) are important in order for FPAAs to be widely accepted by circuit designers. As with many new technologies, the initial effort on FPAAs has focused more on their functional development and proof–of–concept rather than the rigid performance parameters needed for marketable products. Thus, the single biggest unknown in designing large–scale FPAAs is the resulting performance. Often, the initial reaction to FPAAs is that the noise introduced by device mismatch and the parasitic effects of large numbers of switches will cripple the performance of any large, highly flexible FPAA.

Floating–gate devices alleviate some of these problems. Specifically, device mismatch can be compensated for by individually programming the floating–gate transistors to match in output behavior. In addition, the programmable nature of the switches will allow them to be used as variable resistors within the system, and thus, use the non-ideal nature of the switch as an active circuit element [36]. However, this does not completely solve the problem of switch parasitics, because certainly, a number of "unwanted" switches will ultimately be required in a given circuit. The effect of these floating–gate switches has been studied here and results are shown in Chapters 4 and 5. The user and/or automated compilers can use this information to optimize a given system in terms of switch routing so as to minimize their effect and meet the required performance constraints.

Previous FPAAs have also been limited in bandwidth based on their underlying technology. One of the first commerical FPAAs, the EPAC device, was limited to 125 kHz at the input [49]. In general, switched–capacitor designs are typically found to have maximum frequencies in the kilohertz to low–megahertz range [35]. This is similar performance to that shown by continuous–time FPAAs with local routing [24, 69]. One of the fastest designs on record is a small FPAA with a simple CAB based on the second–generation current–conveyor. An amplifier implemented on this FPAA is shown to have a 3-dB frequency of 11 MHz, which is on the order of the performance needed for video applications [34]. While scaling up this FPAA to a meaningful size would inevitably diminish it's bandwidth, it does offer hope that future FPAAs will be able to balance flexibility, functionality, and performance.

2.6 Application Focus

A general–purpose, commercially viable FPAA similar to commercial FPGAs remains elusive. Many FPAA designs have sacrificed size and generality in favor of better performance for a constrained set of circuit designs. FPAAs have been proposed for evolvable hardware [48, 72, 78], neural networks [55, 58], signal conditioning [49], programmable filters [7, 24, 69], fuzzy logic [66], machine learning algorithms [11], and high–frequency applications [34]. Other FPAA designs have attempted to focus on a broader class of systems including both linear and nonlinear elements [10, 70]. However, these efforts have failed to produce a suitably generic, user–friendly FPAA. In addition, all of the FPAAs to date have been very small. The number of CABs on a given device remains under 50 with many of the devices having less than 10 CABs. While several companies currently sell FPAA devices, the market remains relatively immature, and no single device or technology has received wide–spread acceptance.

CHAPTER 3

BUILDING A LARGE–SCALE FPAA

As shown in Chapter 2, traditional FPAAs resemble the early PLDs in that they are focused on small systems often having only a handful of computational logic blocks and limited programmability. Likewise, an analogy can be made between modern FPGAs and the large–scale FPAAs introduced in this thesis. However, while their overall architectures may be similar, reconfigurable analog systems are not completely analogous to their digital counterparts. Developing robust, programmable analog circuits presents a number of challenges not found in the digital world. In particular, the noise sensitivity (and effects of the switch network on the results of the computation) and the design space to which programmable devices are applicable are more critical factors in designing FPAAs.

3.1 Noise Sensitivity and the Effects of Switches

Analog circuits tend to be more sensitive to noise than digital designs. Because of the quantization and resulting representation of ones and zeros as discrete voltages, digital designs can tolerate a relatively large amount of noise in the system without changing the precision of the result. Problems arise only when noise levels are high enough to move a signal from a logical one to a logical zero or vice versa. In the analog domain, however, values are represented as continuous voltages or currents. Any noise in the system will directly affect the precision of the result. For reconfigurable analog systems that must rely on networks of switches to set the internal signal paths, this means that the parasitics of the switches in a signal's path can affect the result and are a critical factor in the performance of the FPAA.

Adding switches in the signal path can have several effects including the addition

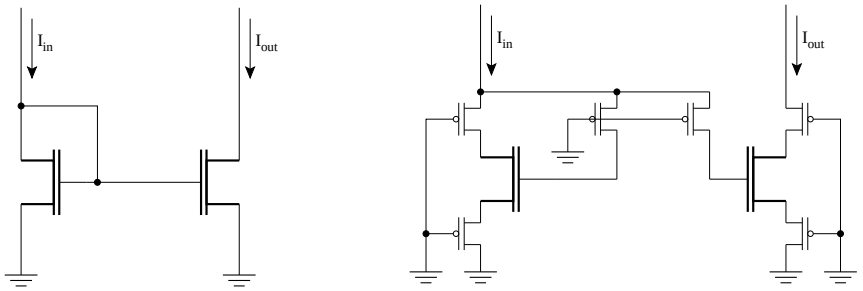


Figure 7. (a) A simple, two–transistor current mirror illustrates the challenges of design reconfigurable ICs with fine–grain building blocks. (b) The parasitic capacitance that results from using switches to form the current mirror will reduce the bandwidth of the system.

of parasitic capacitance, resistance, and transistor leakage currents to the path. Increased capacitance and resistance on a signal line will lower the bandwidth of the system. For example, a simple two–transistor current mirror is shown in Fig. 7a. The same circuit is shown in Fig. 7b with switches being added to the signal paths as they would need to be if the current mirror were synthesized on the FPAA using the MOSFET transistors in the CAB. In this case, there should not be any current flow between the gate nodes, so the voltage should remain equal on the two gate nodes even with the switches in the signal path. However, the parasitic capacitance will decrease the bandwidth of the current mirror.

As shown in Fig. 8, other circuits that have switches in a signal path with current flowing through them will have a voltage drop across the transistor. In the case of a digital inverter (also considered a high–gain amplifier), adding switches to the sources and drains of the nFET and pFET transistors will result in an output with maximum and minimum voltages that are slightly inside the power rails (see Fig. 9). Since the switches in the FPAAs designed as a part of this thesis are pFET devices, one can expect this effect to be much worse for low voltages (i.e., ground) than voltages near the 3.3 V power rail. In this case, the parasitics also increase the rise and fall times, and thus, descrease the maximum frequency for the system.

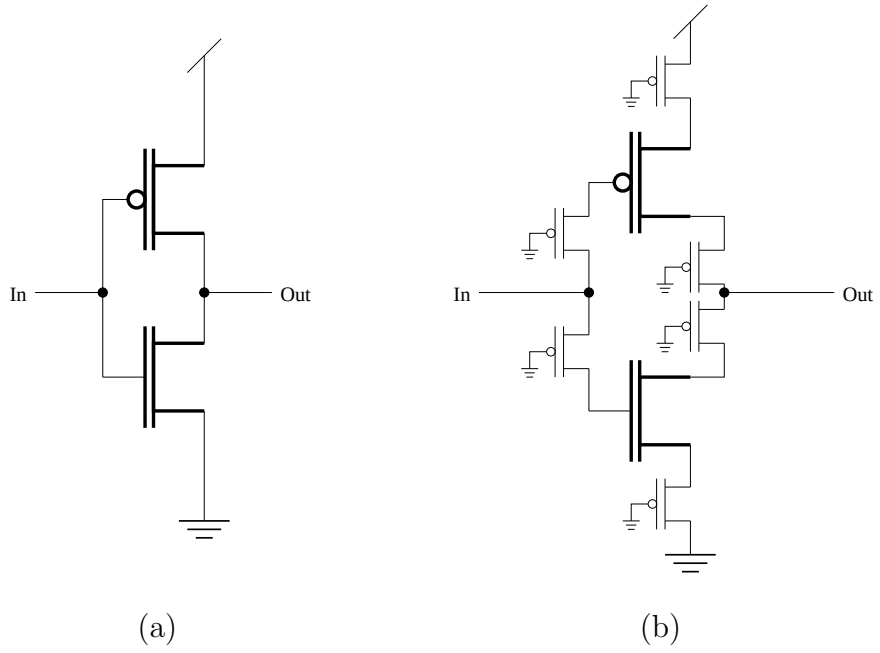


Figure 8. (a) A MOSFET inverter (or high–gain amplifier) circuit diagram is shown here without the switches as it would be designed in a custom CMOS chip. (b) Synthesizing the inverter on an FPAA using switches to connect transistors from the computational logic blocks results in this circuit diagram. The switches will add parasitic capacitance as well as create a voltage drop across the transistor for those switches that have current flowing through them.

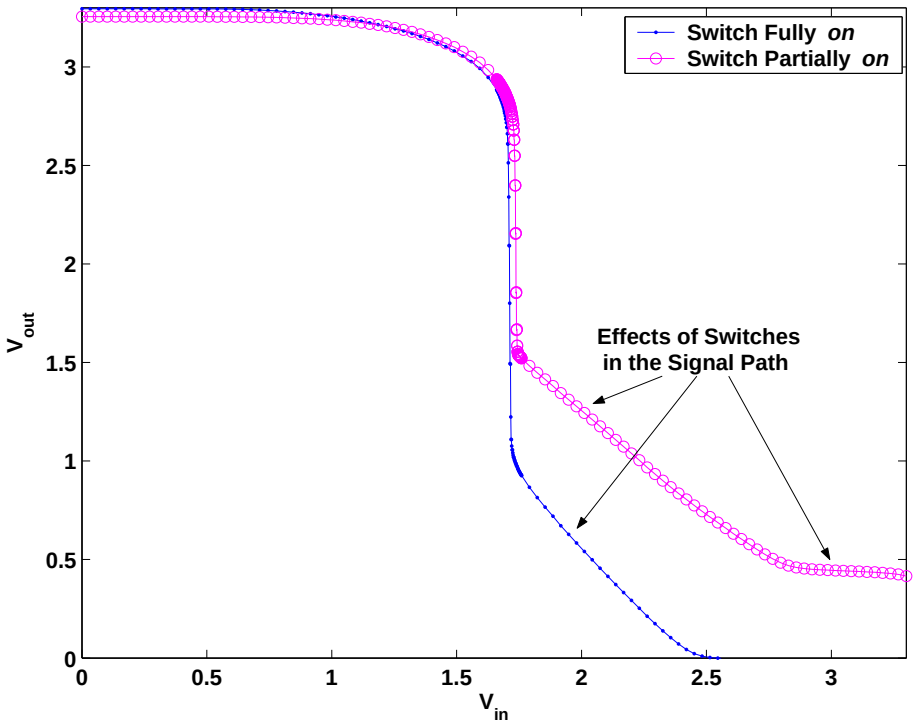


Figure 9. Here, the input to output voltage relationship is shown for an inverter that is synthesized in a testbed FPAA. The inverter is synthesized using an nFET and pFET in one of the CABs on the chip. The switches used to make the appropriate connections are shown in Fig. 8b. The effects of the switches in the signal path are evident by the uncharacteristic linear decay as the output approaches ground. Also, it is clear that when the input is High, the output is not equal to 0 V as one would expect. The switches in the signal path connecting ground to the inverter cause the output to settle to a voltage greater than 0 V due to the non–zero resistance of the switches.

3.2 Switch Networks and Interconnect Design

The resistive/voltage drop effects of the switches will certainly increase as more switches are added to the signal path. However, there is another subtlety involved. The parasitic capacitance and resistance added by the switches at any given switch terminal is actually the sum of all the parasitics of all the switches in a row or column. The summation of parasitics is due to the fact that all the sources for a given row of switches are tied together and that all the drains for a given column of switches are tied together. Therefore, the parasitic contribution of the switches will be present regardless of the state (on or off) of each switch. Since the primary effect of parasitics is lowering the bandwidth, an important architectural issue is present, especially if

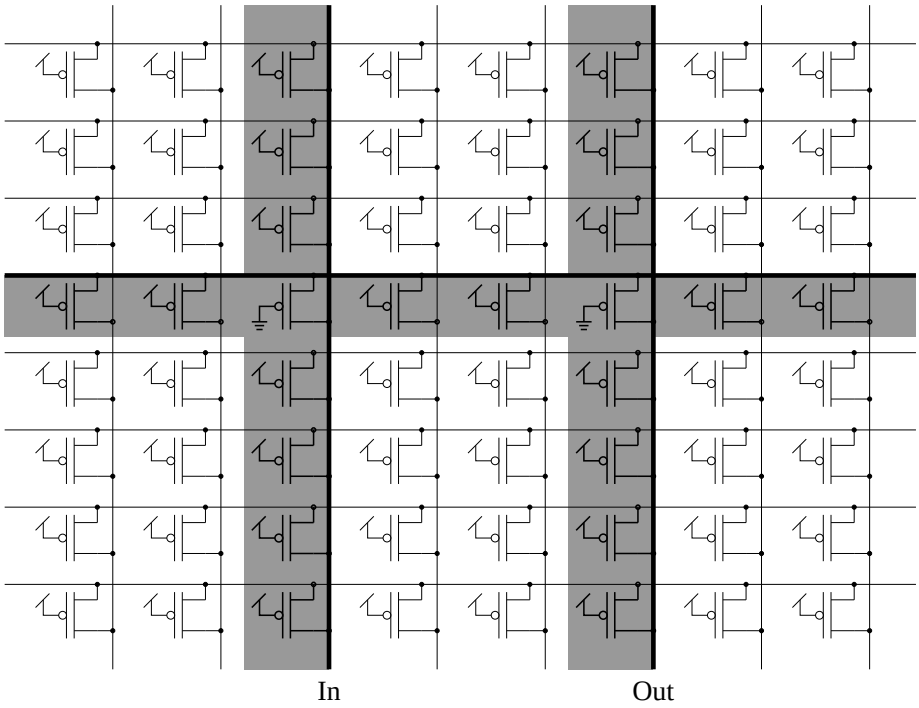


Figure 10. A full crossbar switch provides the most interconnectivity between the components and the input/output (I/O) pins. This architecture also leads to the fewest number of switches in series in a given signal path. However, it results in the worst parasitic effects because of the large number of switches tied to each row and column.

the goal is high–speed design. As FPAAs scale up in size, designers will not be able to use large crossbar switches to make global connections without seriously limiting the bandwidth of large–scale FPAAs.

Instead, hierarchical interconnects will be necessary. By limiting the number of switch connections at each tier of routing, the maximum bandwidth can be set. However, as more tiers are added to the heirarchy, the number of switches that a signal must travel through will increase, thus increasing the resistive drop in voltage due to the routing network. Extreme examples of these two routing paradigms (crossbar vs. hierarchical) are shown in Fig. 10 and Fig. 11. In Fig. 10, a full crossbar switch is shown to illustrate the large parasitic contributions of the switches that result from this type of routing design. The transistors that contribute parasitics to the path from *In* to *Out* are highlighted. Figure 11 shows the other extreme, a binary tree. In

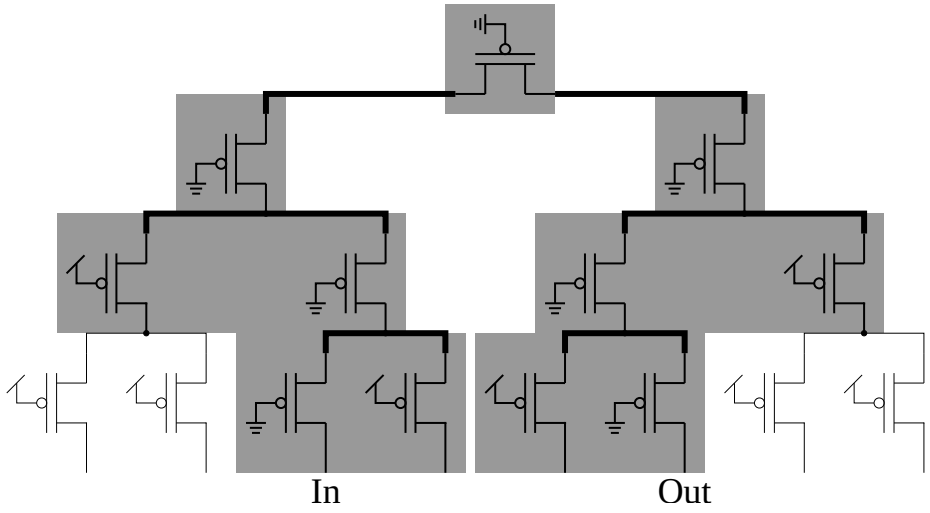


Figure 11. A full binary tree provides the least parasitic effects; however, it requires the most switches to be present in series in a given signal path, and has limited interconnectivity between the components and I/O pins.

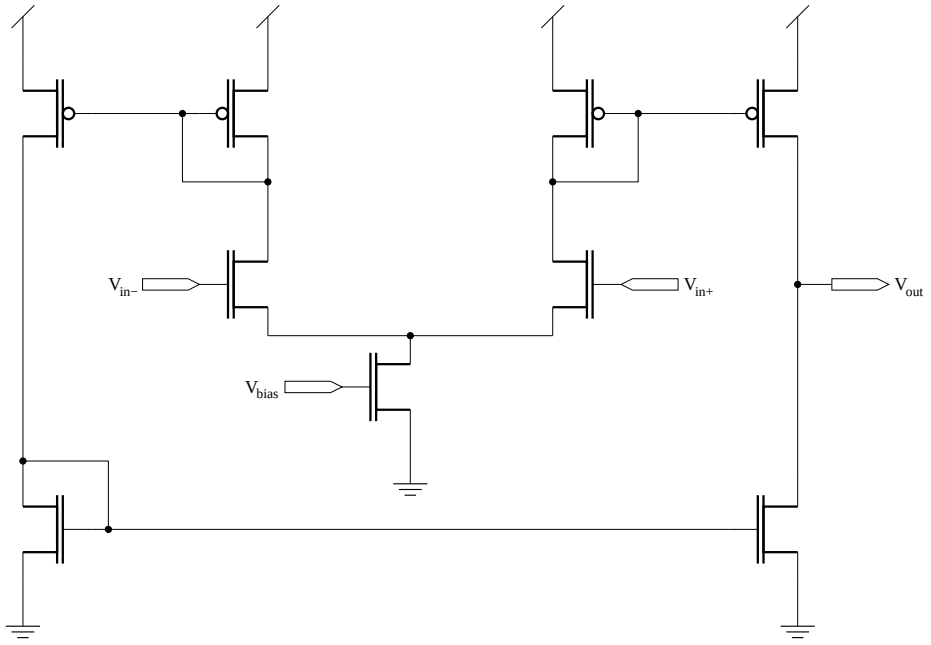
this case, the parasitics at each tier of the hierarchy are minimized, but the number of switches through which a signal must pass increases rapidly with the size of the FPAA. Connectivity is also more limited in a binary tree style architecture. Large–scale FPAA designs must balance the need for bandwidth, signal degradation (number of switches in the signal path), and interconnectivity. The optimal point for this trade–off will be a function of the signal mode of the computational logic (current–mode, voltage–mode, mixed–mode), the size of the overall FPAA, the process technology used (and thus the level of parasitics present), and the bandwidth requirements of the end–user.

A compromise between the extreme routing designs is necessary. One such solution uses crossbar switch networks in the local CAB routing to provide the most connectivity and flexibility in connecting the components together. Global routing will then occur through a moderate number of tiers. For example, CABs can be clustered into megaCABs (groups of four to eight CABs), where limited routing connects CABs within each megaCAB together. These megaCABs can then be tiled across the FPAA in a mesh–style architecture with horizontal and vertical routing between

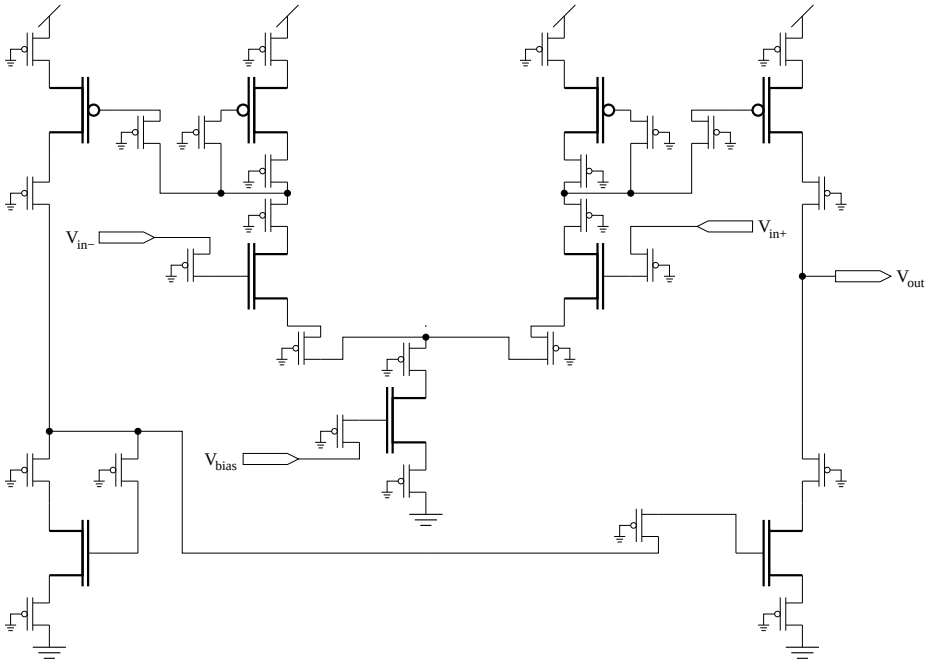
the megaCAB blocks. By tiering the global routing interconnects, the routing area growth is slowed from $O(N^2)$ to $O(logN)$.

3.3 Design Space and Computational Analog Block Design

Another difference between reconfigurable analog and digital devices is the design space that each must encompass. Functionality in the digital domain can be reduced to a basic set of combinational and sequential primitives. For example, a NAND gate can be configured to implement any of the other Boolean logic gates. Thus, with a sufficiently large number of NAND gates, any combinational logic function can be achieved. Similiarly, an asynchronous read–only memory (ROM) primitive can be used to implement any combinational function. For the sequential functions, any basic storage element (e.g., flip-flop or latch) can be used to provide the necesary memory. Most modern FPGAs use asynchronous ROMs to synthesize the combinational logic and D–type flip–flops for implementing the memory/sequential logic. Thus, by replicating these two basic primitives thousands of times across a chip (and a sufficient routing network), an FPGA can be created that synthesizes a very large number of different digital systems. It is tempting to think that one might be able to do the same thing in the analog domain. However, there has not been a sufficiently generic set of medium–grained building blocks (on the same order of complexity as flip–flops or asynchronous ROMs) proposed for synthesizing a wide–range of analog circuits. To get the desired generality, one must use fine–grain building blocks, such as transistors, resistors, diodes, and capacitors. Indeed, a large number of analog systems can be built with these basic blocks; even digital systems could be synthesized with such a device. However, these primitives are so fine–grained that it would require such a large number of components—and thus a large number of switches—to implement a design that the switch parasitics would degrade the performance. For example, the circuit diagram for a basic 9–transistor operational transconductance



(a)



(b)

Figure 12. (a) The circuit diagram of a basic 9–transistor operational transconductance amplifier (OTA). (b) The circuit diagram of the same OTA with the switches needed if it is implemented on a fine–grain FPAA with only transistors. The addition of the 27 switches will dramatically reduce the performance and functionality of this circuit.

amplifier (OTA) is shown in Fig. 12a. In part (b) of this figure, the same OTA is shown with the switches necessary to synthesize this circuit on a fine–grained FPAA with transistors only. The FPAA design requires at least 27 switches, in addition to the nine transistors, to implement the OTA. The switches will drastically affect the performance and functionality of the OTA and may cause the circuit to break. To mitigate these effects, coarser–grained blocks must be used. The task then is to do so while still maintaining sufficient flexibility, functionality, and generality.

Using coarse–grain blocks can be appealing given their increase in performance and robustness over fine–grain blocks. However, if the basic building blocks in an FPAA are of too high a level, then the flexibility is greatly diminished. To be as flexible as possible, an FPAA needs to have a wide range of fine–grained, medium–grained, and coarse–grained components. This means that there will often be more than one way of synthesizing the same system on the FPAA. This provides the most flexibility to end–users, because they can vary the levels of performance, utilization, flexibility, and complexity. For example, a CAB could contain a high–level cell for bandpass filtering, several OTAs, and several pFET and nFET transistors. If there are a sufficient number of CABs, then a bandpass filter could be implemented in at least three ways:

1. *Specialized bandpass filter block:* The specialized cell in one of the CABs could be used directly. This is the simplest implementation method, and it will provide the highest performing design. However, since the full circuit is set in silicon, this will provide the least flexibility in terms of specifying the filter parameters and circuit topology.

2. *OTA–level Design:* The OTAs located in CABs could be connected via the switch network to implement a number of different circuit topologies. This design method will result in lower performance than the first method, but it gives the end–user a lot more flexibility and input into the filter specifications.

Here, the only fixed parameter is the type of OTA that is used in the CABs.

3. *Transistor–level Design:* The transistors located in the CABs could be used to synthesize different operational amplifiers or other medium–sized blocks, which could then be wired together as in method two to form a bandpass filter module. Obviously, this would result in a large number of switches in the signal paths, and thus, it would use alot of the routing infrastructure and would be plagued with large parasitics. Because of the large number of switches used in this design, it would result in the lowest performance (assuming basic functionality can be achieved at all), but it would result in the highest degree of freedom in specifying the circuit topology and filter parameters.

The perfect CAB is elusive indeed. However, careful analysis of common circuit topologies can lead one to a group of components that strike a reasonable balance between flexibility, performance, and generality. Also, one should not limit designs to homogeneous meshes of CABs. In reality, the overall architecture presented in this thesis is designed to be a framework that can be used with many different CABs, while not requiring changes to the overall infrastructure. It is forseen that a number of different CABs will be tried and used in various circuit genres. For example, one can imagine FPAAs targeted at problems in audio processing, image processing, neuromorphic signal processing, telecommunications, etc. These different FPAAs need not largely differ. Instead, a reasonably general–purpose CAB can be tiled across the FPAA with special–purpose CABs targeted at each genre interspersed among them. This type of architecture is very similar to modern FPGAs, where a mesh of general–purpose logic blocks (i.e., asynchronous ROMs and D flip–flops) is interspersed with specialized blocks, such as dense memory blocks, hardware multipliers, digital filters, and even entire processor cores.

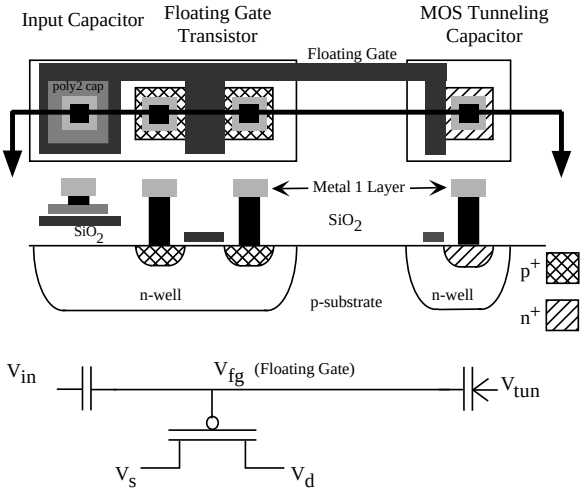


Figure 13. Layout, cross section, and circuit diagram of the floating–gate pFET in a standard double–poly, n-well MOSIS process: The cross section corresponds to the horizontal line slicing through the layout view. The pFET transistor is the standard pFET transistor in the n-well process. The gate input capacitively couples to the floating–gate by either a poly–poly capacitor, a diffused linear capacitor, or a MOS capacitor, as seen in the circuit diagram (not explicitly shown in the other two figures). Between V_{tun} and the floating–gate is our symbol for a tunneling junction—a capacitor with an added arrow designating the charge flow.

3.4 Floating–gate Technology in Programmable Analog Circuits

Previous FPAAs have suffered from their small size and lack of functionality/generality. Next–generation FPAAs need to correct these problems in order to extend the usefulness and acceptance of FPAAs. Ideally, one would like a small, easily programmable element that can be configured to act as an ideal switch, variable resistor, and configurable computational element. While such a device is indeed ideal, floating–gate transistors do offer some of these qualities. Previously, we have shown that the floating–gate transistor can be used as a (nonideal) switch, variable resistor, and programmable element within larger computational blocks (e.g., analog multiplier, programmable filter, programmable OTAs, etc.) [36].

In addition, the small size of the floating–gate structure will allow larger, more

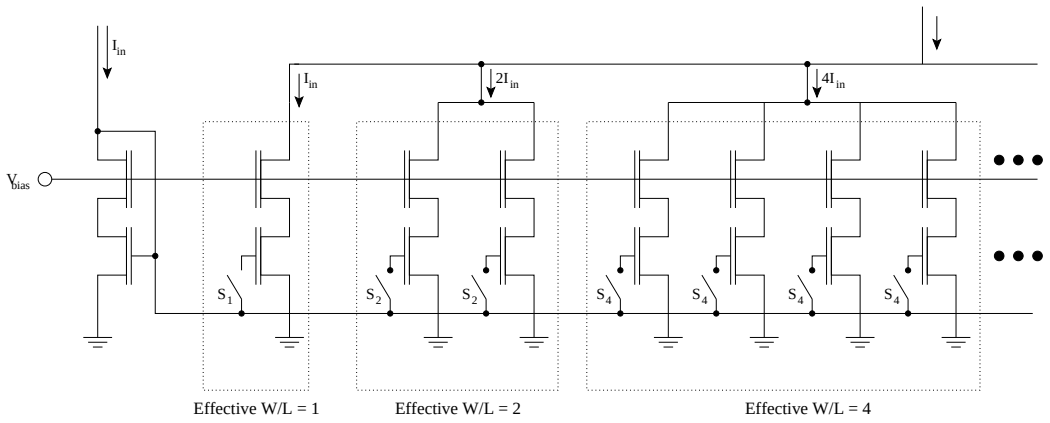


Figure 14. Standard designs often achieve circuit programmability by embedding switchable arrays of elements (such as transistors or capacitors) within the logic cells. Here, a conceptual version of [65]'s programmable current mirror is shown. In their case, 5 bits were used to set the switches. This required 64 MOS transistors, 31 digitally controlled switches, and five memory cells to hold the configuration of the switches. Using floating–gate technology, this entire structure can be replaced with two programmable floating–gate transistors.

functional FPAAs to be built using this technology. One example of the capability/area improvement that can be achieved with floating–gate transistors is the programmable current mirror. Pankiewicz et al. have presented one of the most recent FPAA designs. Their FPAA is based on OTAs in which the current mirrors on the differential outputs can be programmed. They use a bank of current mirrors similar to the simplified form shown in Fig. 14. Each current mirror requires 64 MOS transistors, 31 digitally controlled switches, and five memory cells to hold the configuration of the switches. The entire structure can be replaced with two MOS transistors and a programmable floating–gate transistor. The area savings in this case are considerable. Furthermore, the resolution of the bank of current mirrors is set at five bits; whereas, the floating–gate current mirror's resolution can be varied based on the need of a given application with a maximum resolution of approximately 10 bits [73].

The floating–gate transistors used in these FPAAs are standard pFET devices whose gate terminals are not connected to signals except through capacitors (e.g., no DC path to a fixed potential) [43]. Because the gate terminal is well insulated from external signals, it can maintain a permanent charge, and thus, it is an analog

memory cell similar to an EEPROM cell. With a floating gate, the current through the pFET channel is dependent on the charge of the floating–gate node. By using hot–electron injection to decrease the charge on the floating–gate node and electron tunneling to increase the charge on the floating–gate node, the current flow through the pFET channel can be accurately controlled [43, 52].

3.4.1 Floating–gate Switches

Using a floating–gate transistor as a switch requires that the device be turned *on* or turned *off*. Ideally, the *on* state corresponds to the free flow of current through the device or equivalently, zero impedance between the source and the drain. Likewise, the *off* state is ideally characterized by zero current flowing through the device – an infinite impedance between the source and the drain nodes. A floating–gate transistor, however, will not act as a perfect switch. The *on* state will be characterized by an impedance greater than zero, and the *off* state will have an impedance less than infinity. Therefore, the quality of a floating–gate transistor as a switch will be determined by measuring the *on* and *off* impedances.

The quality of the switches is an important factor in determining the final architecture. The main concern is that routing a signal through multiple switches could degrade data as the cummulative impedance of the switches becomes prohibitive.

The impedance of the floating–gate transistor is a function of the charge on the floating–gate node allowing it to be set using hot–electron injection and electron tunneling. Figure 15 shows the relative I–V curves for a floating–gate transistor as it is programmed from *off* to *on*. Ideally, each transistor would be programmed to the extreme ends of the graph, but programming floating–gate transistors is a time–comsuming task. The desired quality of the switch will have to be chosen with regard to the time it will take to program the device.

Choosing a reasonable time scale for programming leads to a compromise in the quality of the switch. Also, note that the operating voltage of the gate node is not

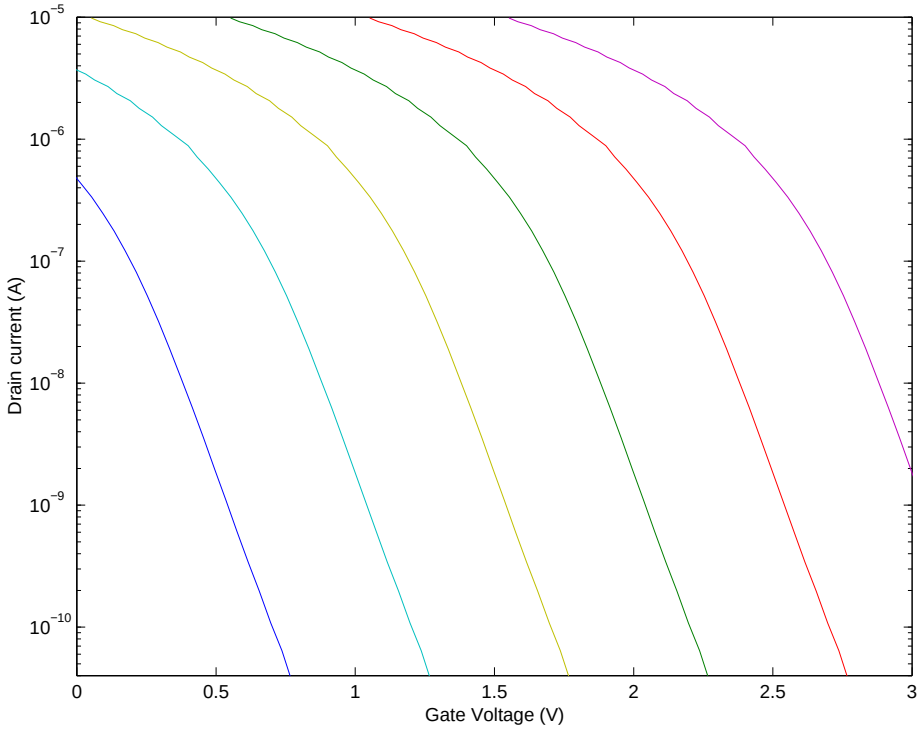


Figure 15. This is a current–voltage plot of a single floating–gate transistor programmed to different levels of floating–gate charge. The floating–gate transistors can be programmed over a wide range of currents.

fixed. Due to the parasitic capacitances between the gate and drain/source nodes, the gate voltage—and thus the switch's impedance—will vary as a function of the signal current. This variation must be minimized.

3.4.2 Switch as Computational Element

When used as a switch, the floating–gate should be as transparent a part of the circuit as possible. However, Fig. 15 shows that the floating–gate transistor can be used as an in–circuit element [42, 51]. By adjusting the charge on the floating–gate node between the extremes used for *on* and *off*, the impedance of the switch can be varied over several orders of magnitude. Thus, a variable, non-linear resistance can be synthesized by the floating–gate switch.

Using the floating–gate switches as in–circuit elements allows for a very compact architecture. The physical area needed for the CABs is greatly reduced, because

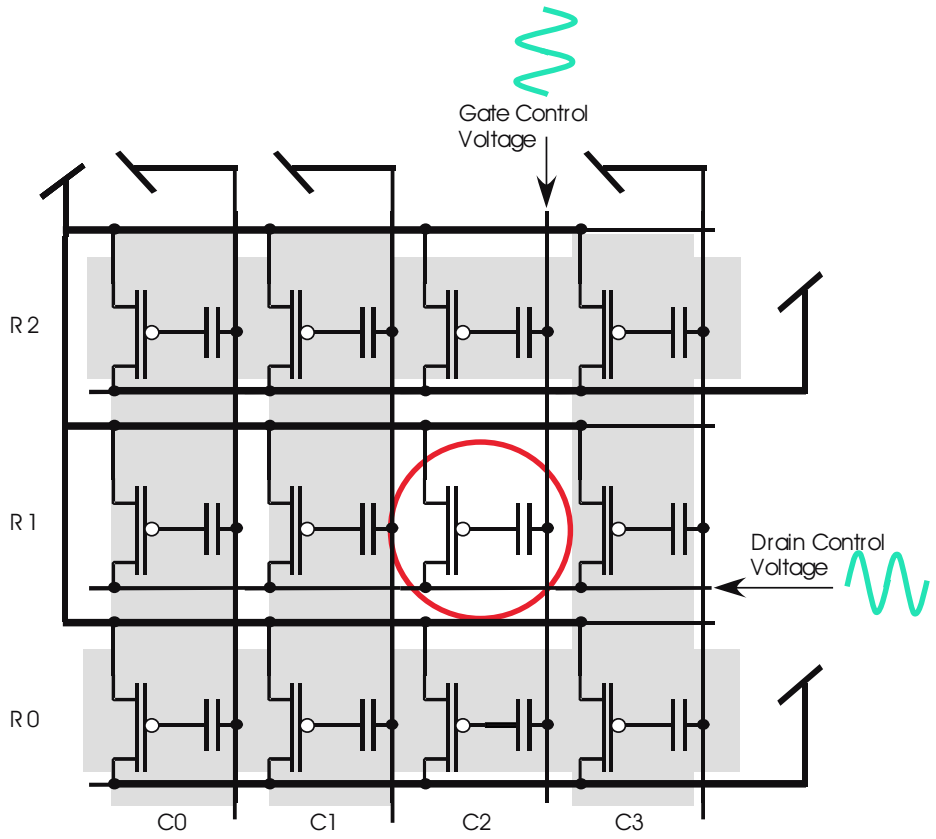


Figure 16. By selectively setting the gate and drain voltages of the columns and rows, a single floating–gate transistor can be programmed using hot–electron injection without affecting the neighboring devices.

resistors, which consume relatively large amounts of space on CMOS processes, are not needed as separate components. Also, by reducing the number of individual circuit elements, signal routing is simplified without losing functionality.

3.4.3 Programmability

The use of floating–gate devices as the only programmable element on the chip, greatly simplifies chip configuration. Additionally, all of the floating–gate transistors are clustered together to aid in the programming logic and signal routing. Decoders on the periphery of the circuit are connected to the drain terminals, source terminals, and gate capacitors of the floating–gate matrix. In programming mode, these decoders allow each floating–gate transistor to be individually programmed using hot–electron

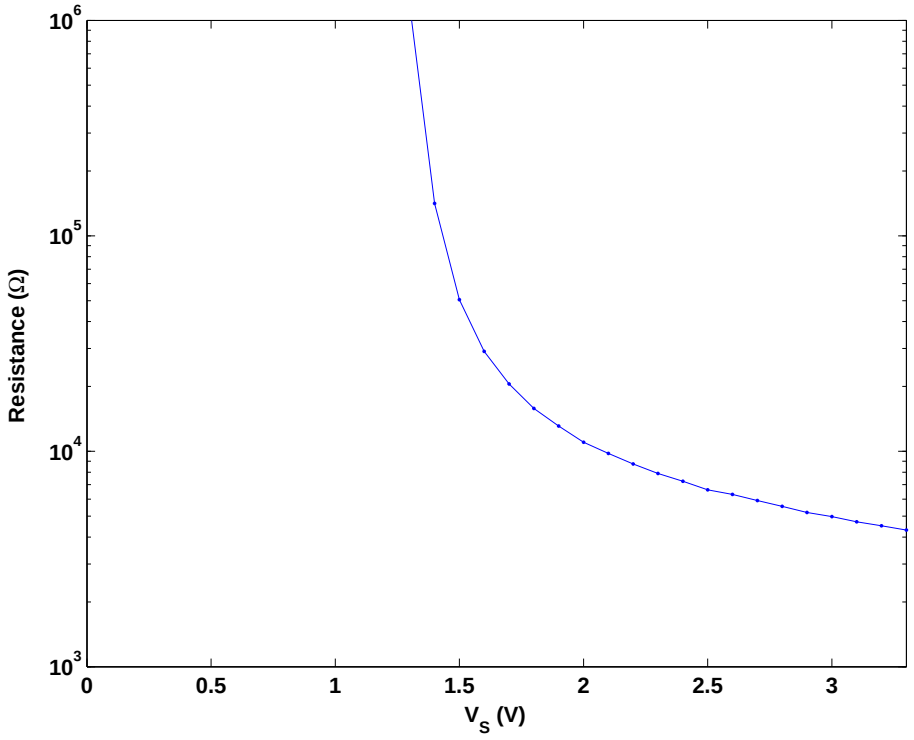


Figure 17. This is a typical resistance plot for a pFET switch in the *on* configuration. With an exponentially increasing resistance as the DC bias approaches 0 V, the pFET switch clearly does not pass low voltage signals very well.

injection (see Fig. 16) [52].

Part of the previous work has been the development of a systematic method for programming arrays of floating–gate transistors [51, 52, 76]. A microprocessor–based board has been built to interface a PC to these analog floating–gate arrays for the purposes of programming and testing. With a PC controlling the programming of these devices, the details of using hot–electron injection and tunneling to program individual floating–gate switches have been abstracted away from the end–user. The programming algorithms have been optimized for accuracy and speed, while giving the end–user an easy–to–use interface for configuring arrays of floating–gate devices.

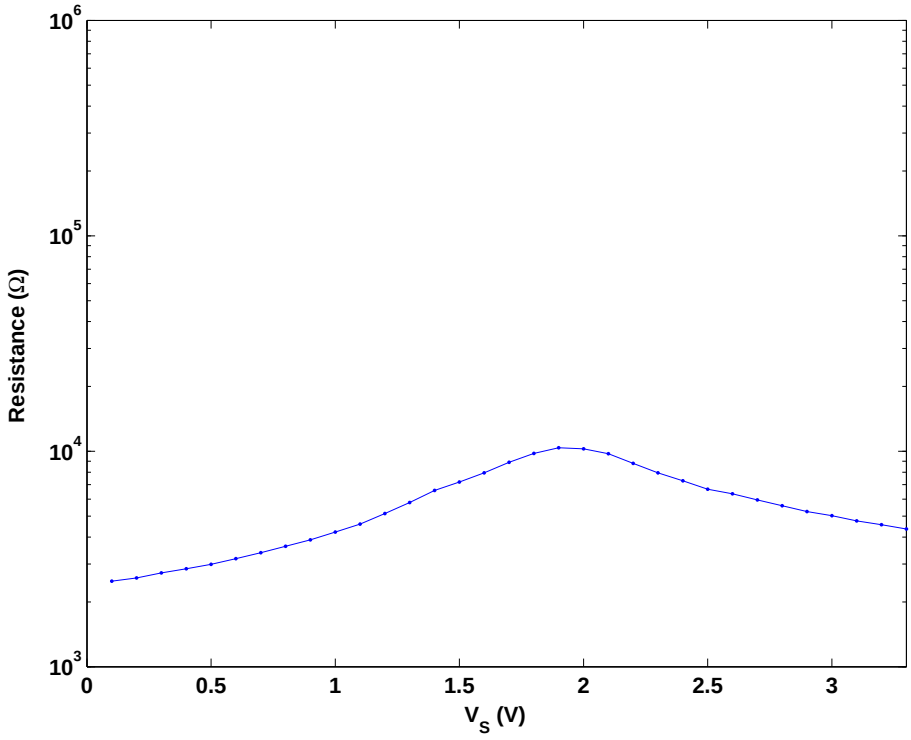


Figure 18. This is a typical resistance plot for a T–gate switch in the *on* configuration. With an additional nFET transistor, the T–gate switch exhibits a much lower resistance for low input voltages than the lone pFET transistor switch. This benefit, however, comes at the cost of more area and more parasitic capacitance added to the signal line.

3.5 Selecting a Switch

As discussed in Section 2.3.2, a number of different switches that have been tried in FPAAs. A switch must be selected based on bandwidth, area, and switch quality (*on* and *off* impedances). Typically, the switch is turned *on* or *off* with a digital memory element, such as an SRAM cell. This scheme is popular in modern FPGAs, as it is well understood and relatively quick to configure.

The simplest switch is a single pFET transistor; however, pFET transistors have an *on* resistance that exponentially increases as DC bias voltage drops. A typical resistance plot for a relatively small pFET device is shown in Fig. 17. As this figure illustrates, the *on* pFET device will not pass low voltage signals.

A larger switch can be formed by adding an nFET transistor to the pFET and

forming a T–gate structure. This allows low–voltage signals to be passed equally as well as those closer to the positive power rail. As shown in Fig. 18, the resistance through a T–gate switch is more constant than the lone pFET transistor. The peak in this resistance plot near 2 V can be changed by carefully sizing the pFET and nFET that are used in the T–gate switch. While the switch resistance is much better than the pFET transistor switch, the disadvantages of the T–gate structure include increased area and lower bandwidth due to the increased parasitic capacitance. The T–gate switch requires an additional nFET transistor and an inverter for the select lines (two more transistors). Thus, it is at least three times larger then the simple pFET switch. Also, the extra nFET transistor in the signal path means at least doubling the parasitic capacitance and resistance. Increasing the parasitics on one or two switches may not seem significant; however, as mentioned previously, the parasitics are summed across the rows and columns of the switch networks. Even a moderately sized FPAA could have 50–100 switches along a single dimension. Doubling the parasitics on each switch in the network would result in a significant decrease in bandwidth.

A perfect switch would have the small size, minimal parasitic effects, and simplicity of the pFET switch and the lower, more constant resistance of the T–gate switch. The floating–gate switch comes close to this ideal. As shown in Fig. 19, the resistance of the floating–gate switch is much more constant across the full voltage swing than the pFET transistor. By injecting the floating–gate node to an extremely *on* point, the effective voltage on the gate of the transistor is allowed to drop below 0 V. This results in a shifting of the resistance curve until the exponential region is pushed below 0 V as well. The primary trade–off for this increased performance is configuration speed. Programming floating–gate switches to this extreme position currently takes nearly one to two minutes. The current programming scheme, however, is externally controlled. By moving the programming scheme onto the chip and optimizing the logic

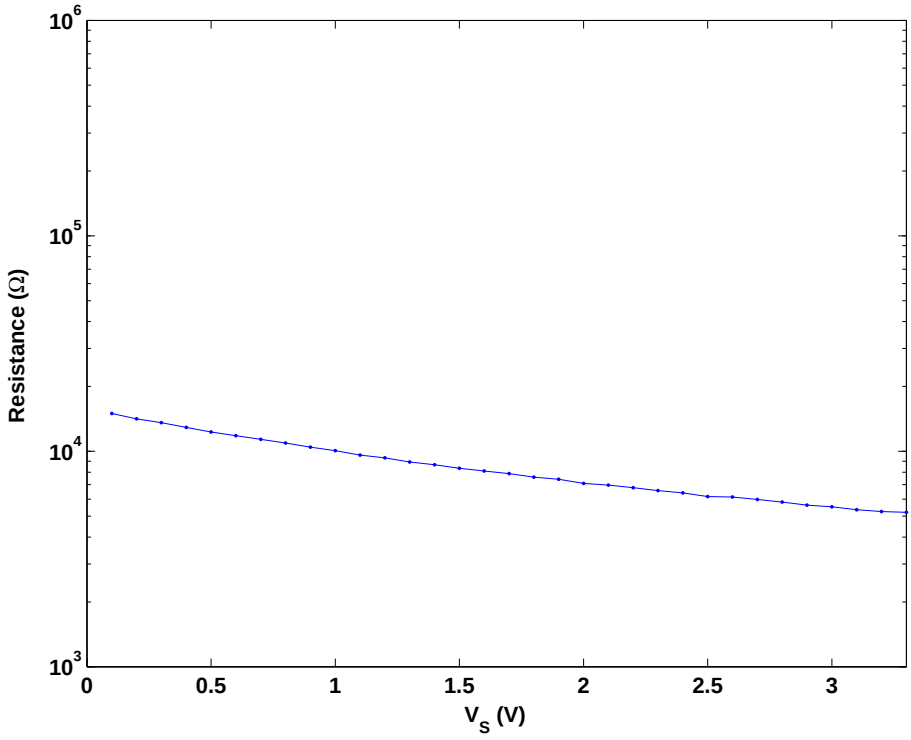


Figure 19. This is a typical resistance plot for a floating–gate switch that has been programmed into the extreme *on* position. By injecting the floating–gate node, the effective voltage on the gate of the switch is allowed to drop below 0 V. This results in a shifting of the resistance curve until the exponential region seen in Fig. 17 is pushed below 0 V as well.

for programming switches, a significant increase in speed can be achieved. In addition, work is progressing within the CADSP group on parallelizing the programming control logic so that an entire row or column of switches can be programmed at once. The potential decrease in programming time with this scheme is quite dramatic.

CHAPTER 4

RASP 1.0

Large signal processing systems will require a chip with many (100 or more) computational analog blocks (CABs) on it. However, before such a chip could be successfully designed, a number of implementation details were tested on a smaller scale. Specifically, two testbed FPAAs have been designed, fabricated, and tested as a part of this thesis. Reconfigurable Analog Signal Processor (RASP) 1.0 was primarily used to characterize the switch networks and test small systems, and it will be discussed in detail in this chapter. RASP 1.5 was similar; however, a number of layout and architectural improvements were made that increased the quality of switches and allowed larger testbed systems to be synthesized. Chapter 5 contains a detailed discussion of RASP 1.5.

RASP 1.0 was fabricated in a 0.5-micron, standard CMOS process. A die photo of this chip is shown in Fig. 20. This FPAA contains two CABs connected by a floating–gate crossbar switch network [36]. The CAB design included a bandpass filter module, 4 x 4 vector–matrix multiplier, and three wide–range OTAs. RASP 1.0 is designed as a testbed integrated circuit (IC) to study the floating–gate switches and the interaction between the CAB components and the switch matrix [38]. A two CAB system should be of a sufficient size to test the concept of floating–gate FPAAs.

RASP 1.0 uses floating–gate transistors as the sole programmable element within the FPAA. Floating–gate analog circuits are used to implement advanced signal processing functions and are very useful for processing analog signals prior to A/D conversion. The architecture introduced here is extensible to larger systems and will allow systems that go beyond simple programmable amplifiers and filters to include programmable and adaptive filters, multipliers, winner–take–all circuits, matrix–array signal operations, frequency decomposition, and subband processing.

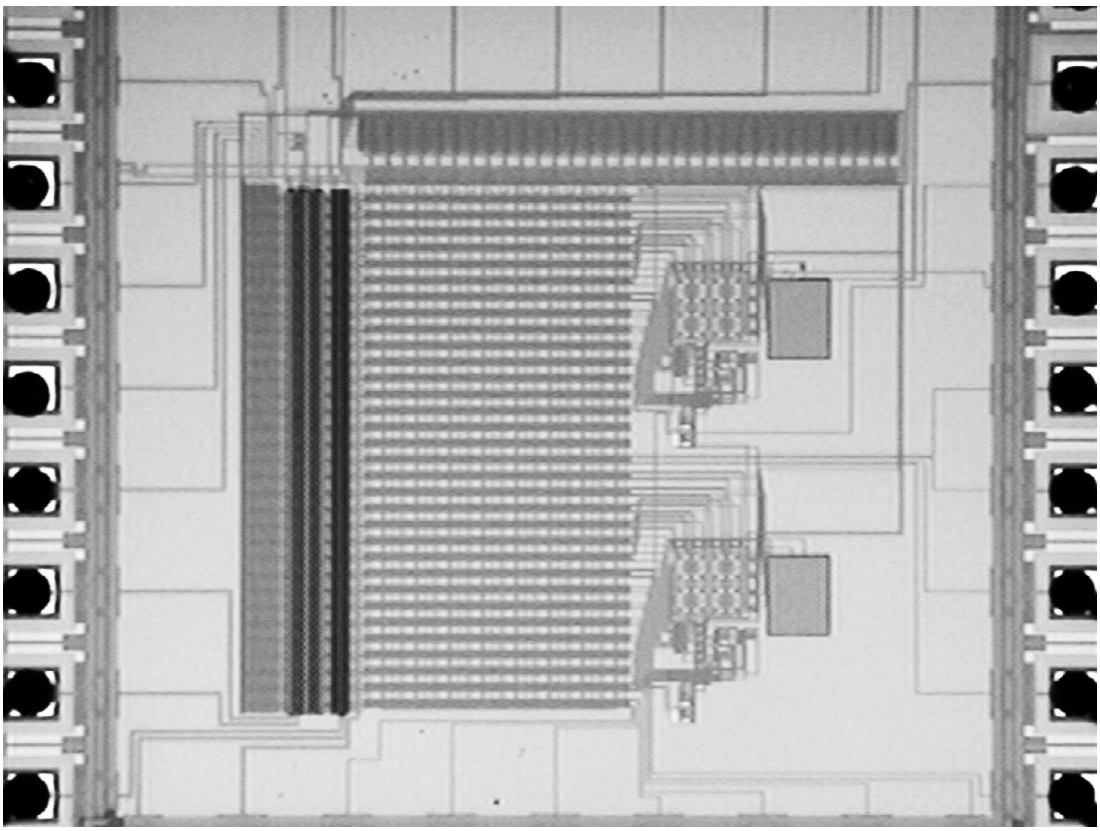


Figure 20. This is a die photo of the RASP 1.0 chip. It contains two CABs with a single switch network to connect the CABs together.

4.1 Computational Analog Blocks

The computational logic is organized into a compact CAB providing a naturally scalable architecture. CABs can be tiled across the chip with busses and local interconnects in–between as shown in Fig. 21.

Each CAB is comprised of components critical to signal processing applications, including a 4 x 4 matrix–vector multiplier, three wide–range OTAs, and a transistor–only version of the autozeroing floating–gate amplifier (AFGA) or capacitively coupled current conveyor (C^4) [40]. The CAB architecture is shown in Fig. 22.

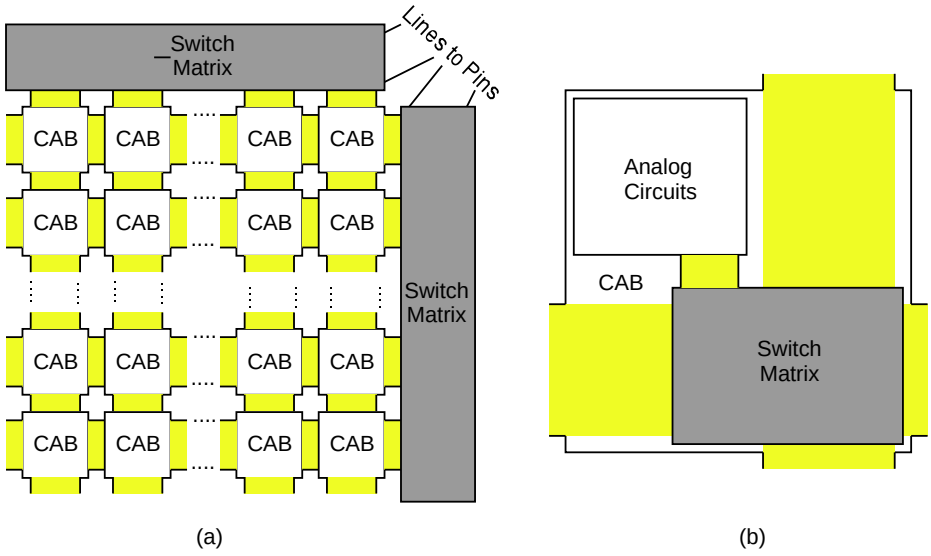


Figure 21. Block diagram of our generic FPAA chip. (a) Architecture diagram of analog programmable computing blocks. Future FPAAs will have 100 or more Computational Analog Blocks (CAB) each with 23 ports to the mesh switching matrix. (b) Signal flow in a single CAB.

4.2 Analog Circuit Components

Selecting the types of analog components to include in a general–purpose FPAA is a difficult task. To be as universal as possible, one must consider adding a number of basic linear and nonlinear functions, including integration, summation, gain amplification, logarithmic and exponential operations, and more [31]. Because these elements are so basic, constructing larger systems can become very complex due to the required routing resources. Also, as discussed earlier, as the number of switches involved in a circuit increases, the cummulative effects of the switches on the circuit may seriously degrade the performance and/or results. To mitigate these challenges, RASP 1.0 was constrained to be a signal–processing FPAA with specific functions such as adaptive filtering and Fourier (frequency domain) processing in mind. A limited number of basic elements were also included for completeness; however, the focus was placed on selecting an appropriate mix of higher–level components that could facilitate the prototyping of a wide range of problems.

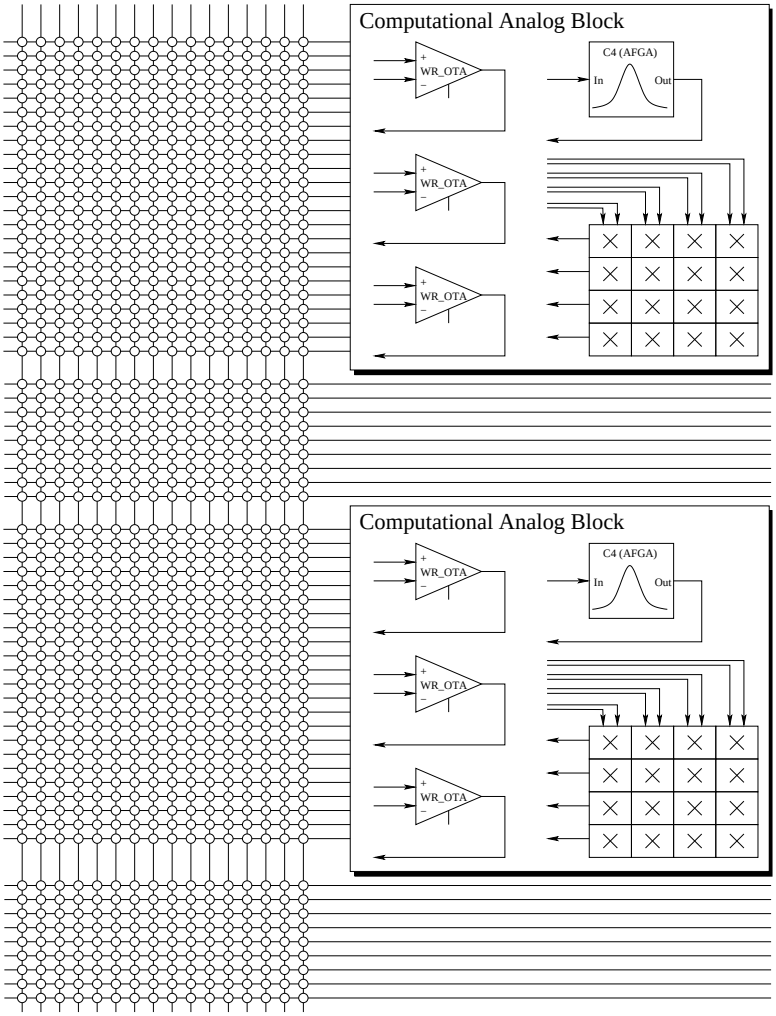
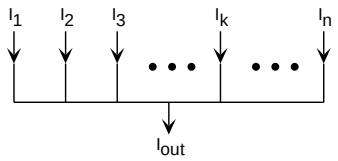


Figure 22. Each Computational Analog Block (CAB) has a four–by–four matrix multiplier, three wide–range operational transconductance amplifiers (OTAs), and a capcatively coupled current conveyor (C^4). The input and output signals shown in this figure are routed to the rows of the switch matrix.

4.2.1 Basic Analog Elements

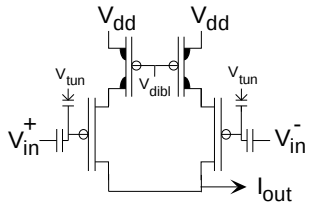
The basic analog functions such as summation, integration, and gain amplification can be included in the FPAA with only a few analog components. In the case of summation, only the switch matrix is needed. Figure 23 shows that if the input signals are currents, summation is achieved by simply connecting the input signals together (Kirchoff's current law).

By adding several configurable OTAs to each CAB, one can configure the FPAA to



$$I_{out} = \sum_{k=1}^{N} I_k$$

Figure 23. The output current of a node is equal to the sum of the input currents (Kirchoff's current law).



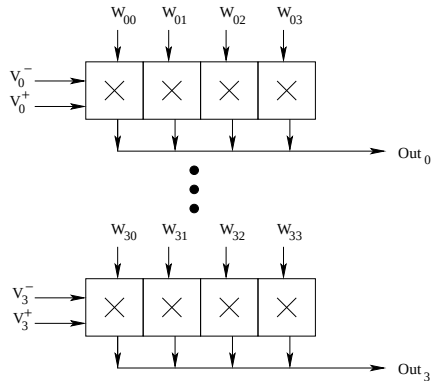
$$I_{out} = I_{bias} + W\left(V_{in}^{+} - V_{in}^{+}\right)$$

Figure 24. Multiplication of a signed (differential) input by a stored weight can be achieved using two floating–gate transistors. Since the weight is stored as the charge of the floating–gate node it is both programmble and adaptable.

perform integration, differentiation, gain amplification and more. Also, by including op–amps in the computational logic, RASP 1.0 can compete with current commercial offerings that are based solely on programmable op–amps.

4.2.2 Matrix–Vector Multiplication

Multiplication is an important element in many signal processing applications. Figure 24 shows a basic multiplier using two floating–gate transistors. The differential voltage input allows signed numbers to be represented. The multiplier's functionality can be extended by cascading them together to form a matrix–vector multiplier. Each CAB on RASP 1.0 has a 4 x 4 matrix multiplier in which four signed (differential) inputs are mutliplied by a 4 x 4 matrix of programmable weights. The functionality of this matrix–vector multiplier is shown in Fig. 25. Of course, by setting the appropriate weights to zero, matrices smaller than 4 x 4 can be multiplied by the input vector.



$$\left[\begin{array}{cccc} Out_0 & Out_1 & Out_2 & Out_3 \end{array}\right] =$$

$$\left[\begin{array}{cccc} (V_0^+ - V_0^-) & (V_1^+ - V_1^-) & (V_2^+ - V_2^-) & (V_3^+ - V_3^-) \end{array}\right] \left[\begin{array}{cccc} W_{00} & W_{10} & W_{20} & W_{30} \\ W_{01} & W_{11} & W_{21} & W_{31} \\ W_{02} & W_{12} & W_{22} & W_{32} \\ W_{03} & W_{13} & W_{23} & W_{33} \end{array}\right]$$

Figure 25. Matrix–vector multiplication is achieved by cascading the floating–gate multiplier shown in Fig. 24 and summing the outputs along each row. The multiplier outputs are current values, so they can be summed by tying the signal lines together. The weight matrix is stored as the charges on the floating–gate nodes of the transistors that form the multipliers.

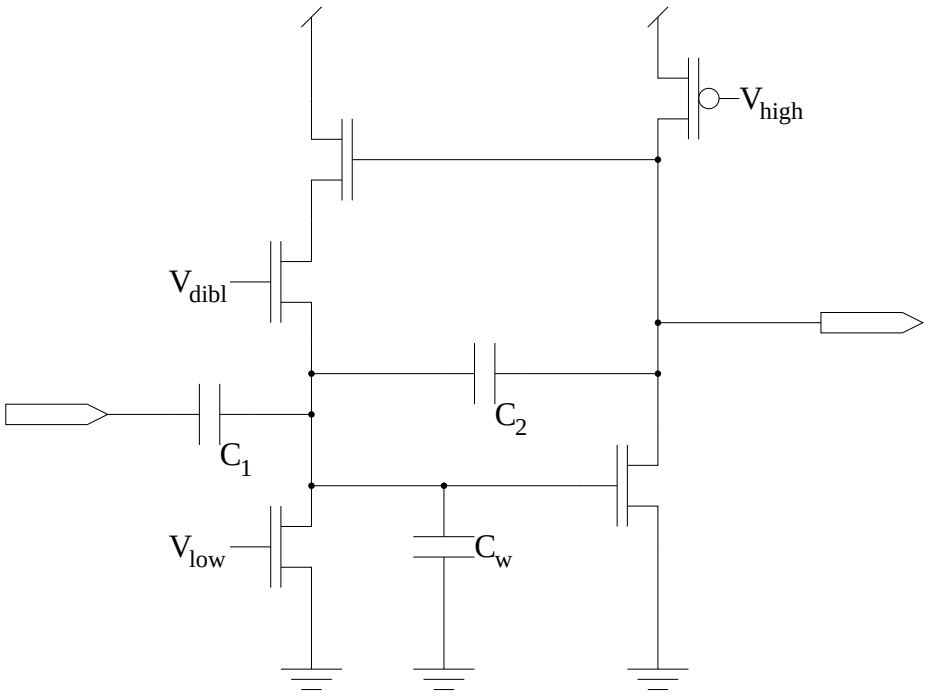


Figure 26. This circuit is a capacitively coupled current conveyor (C^4). It is an element in the CAB and performs a bandpass filter operation. The corner frequencies of the filter can be programmed using the floating–gate current sources that connect through a current mirror to the biases V_{low} and V_{high}.

4.2.3 Filtering and Fourier Processing

FPAA devices based on RASP 1.0's floating–gate architecture can have as many as 100 or more CABs on a single chip. At this level of complexity, a number of interesting signal processing systems can be implemented using a Fourier processor. An analog Fourier processor decomposes an incoming signal into its frequency components (subbands). Each subband is then operated on before reconstructing the output signal by summing the subbands together. This is analagous to taking a Fourier transform, operating on each frequency component, and then taking an inverse Fourier transform to generate the time–domain output signal.

In RASP 1.0, the frequency decomposition is done by using C^4 circuits to bandpass filter the incoming signal [44, 52]. A C^4 circuit is built using two floating–gate transistors and several capacitors as shown in Fig. 26. This method allows the bandpass

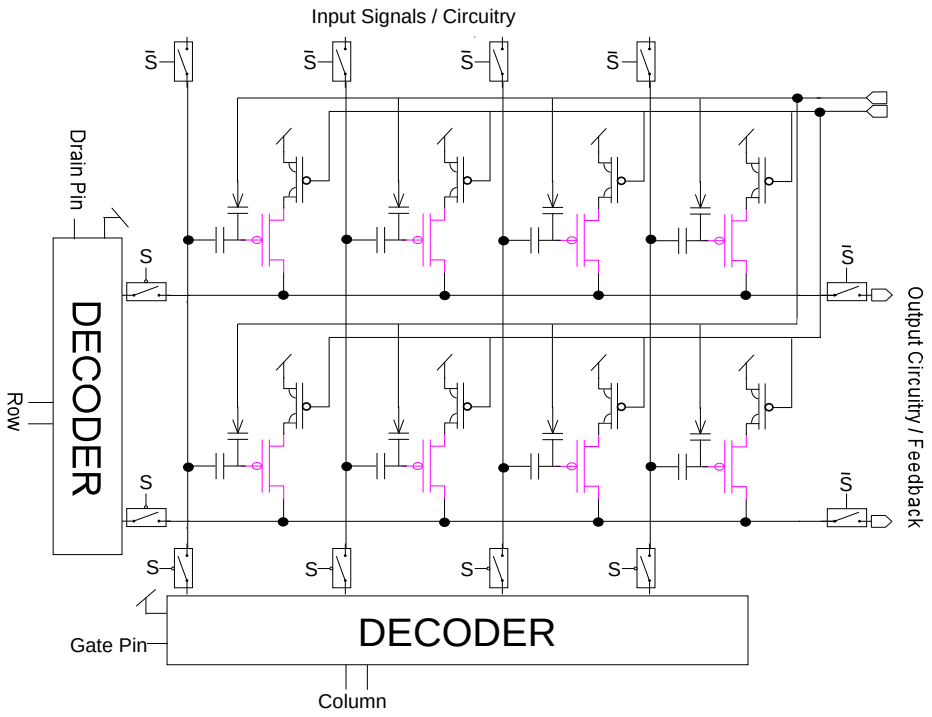


Figure 27. The switching matrix is built with floating–gate transistors. The charge of the floating–gate node can be adjusted to allow current to flow through the channel (*on*) or to restrict current flow (*off*). When in programming mode, T–gates connect the floating–gate transistors to the decoders, and when in operational mode, the decoders are unconnected and the output signals are switched on.

filters to be programmably placed at the desired frequencies (some applications prefer linear spacing while others prefer logarithmic spacing of the subbands). When used in combination with the floating–gate multiplier, a wide range of filters, including adaptive filters, can be achieved [43].

4.3 Switch Matrix

RASP 1.0 has one 16 x 64 crossbar switching matrix that provides local interconnects between the two CABs and connections to the external input/output signal lines. The switching matrix uses floating–gate transistors as the switches (Fig. 27).

The digital decoders on the outside of the switch matrix provide access to the individual floating–gate transistors for programming. After programming is complete, the decoders are disconnected (using T–gate switches), and external bus lines are

connected to start the operation of the chip.

The source lines of the 16 floating–gate switches in each row are connected together; likewise, the drain lines of the 64 floating–gate switches in each column are tied together. Programming a switch to *on* allows current to flow in the source–drain channel and connects a row with a column. Programming a switch to *off* restricts the current flow in the channel creating a very high impedance between the row and column.

The input/output signals from each CAB are connected to the source nodes of the floating–gate switches. The drain nodes (columns) are either connected to external busses or are used for internal connections only. By doing this, any input/output signal from the CABs can be connected to an external bus by turning on one floating–gate switch. Similarly, it can be connected to another signal in the same CAB or to a signal in an adjacent CAB by turning on two switches in the same column.

4.3.1 Floating–gate Switches

As mentioned in Chapter 3, using a floating–gate transistor as a switch requires that the device be turned *on* or turned *off*. Ideally, the *on* state corresponds to the free flow of current through the device or equivalently, zero impedance between the source and the drain. Likewise, the *off* state is characterized by zero current flowing through the device – an infinite impedance between the source and the drain nodes. A floating–gate transistor, however, does not act as a perfect switch. The *on* state is characterized by an impedance greater than zero, and the *off* state has an impedance less than infinity. Therefore, the quality of a floating–gate transistor as a switch is determined by measuring the *on* and *off* impedances.

The floating–gate switch network has been characterized in [37]. The switches were found to exhibit similar characteristics to standard pFET switches with an *on* resistance as low as 11 kΩ and an *off* resistance in the low gigaohm range. They have also been shown to be accurately programmable and capable of implementing a

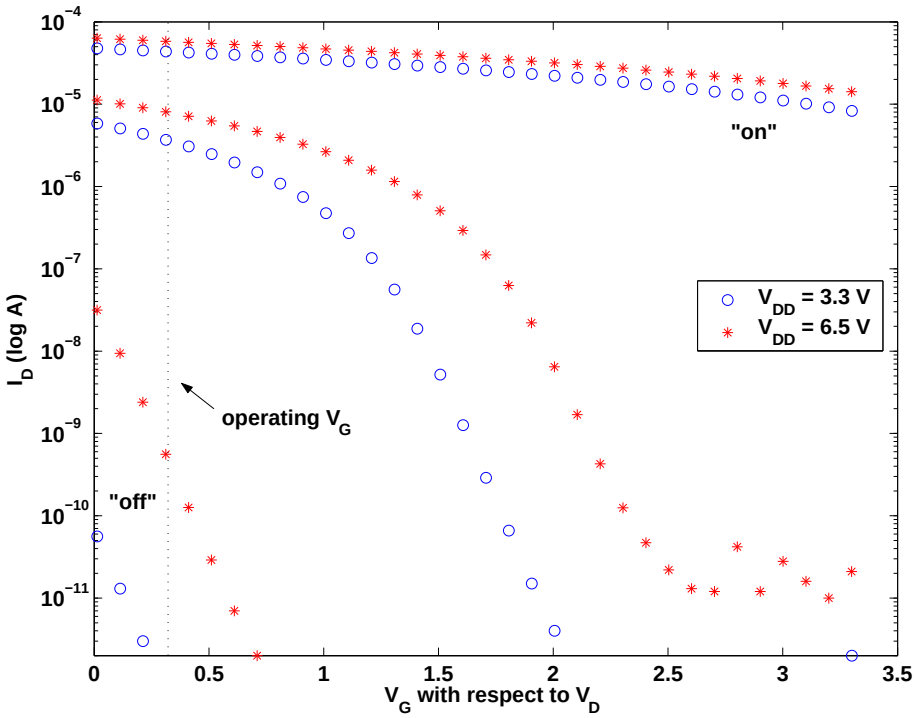


Figure 28. Floating–gate switches can be programmed within a wide range. Here, examples of an *on*, *off*, and mid–position device are shown. During programming, currents are measured with $V_{DD} = 3.3\ V$ for large currents and $V_{DD} = 6.5\ V$ for small currents. This effectively extends the programming range of the device.

variable resistance. As shown in Fig. 28, the floating–gate switch can be programmed between the *on* and *off* extremes.

To increase the quality of a switch, the floating–gate transistors are programmed to the far extremes of their range. When switches are being programmed *off*, currents in the low picoampere range must be measured. These measurements are near the limits of standard laboratory equipment; therefore, to extend the viable programming range, current measurements are taken at a larger drain–to–source voltage. Typically, V_{DS} is set to the supply voltage, V_{DD}, and an increase in V_{DS} is achieved by increasing V_{DD}. As shown in Fig. 28, measuring the currents with $V_{DD} = 6.5\ V$, allows the I–V curves to be visible to the programming infrastructure 1 V below the point visible when $V_{DD} = 3.3\ V$ [38].

For simplicity, the voltages on the gate capacitors of all the switches are set to a

constant potential. This means that the voltage driving the gate capacitors will be the same for both *on* and *off* switches. To determine the appropriate gate voltage for run mode, the relative quality of *on* and *off* switches must be balanced. From Fig. 28, it is clear that the *off* switches do not pose a problem, since any gate voltage selected at or above 0.3 V should provide a sufficiently high impedance. However, the *on* switch exhibits a decrease in quality as the gate voltage is increased to V_{DD}. Thus, an operating gate voltage of 0.3 V is deemed optimal for the current programming scheme.

4.4 Circuit Characterization

The resistance of the floating–gate switch in the *on* position proved similar to standard pFET switches with the measured on–resistance starting at 11 kΩ. As shown in Fig. 29a, the architecture of our FPAA required that the resistance be measured through two devices in series. During these experiments, both of the transistors were programmed to the same position; the measured resistances shown here have been divided by two to report the resistance for a single transistor. The resistance of the floating–gate switch in the *off* position was not directly measured due to the extremely small currents. Instead, the saturation current of the device was determined (Fig. 29b). This data was acquired with $V_{DD} = 7.5\ V$ since this was the first reliable measurement voltage. The resistance can then be calculated to be $R = \frac{U_T}{I_{SAT}} = \frac{25.8\ mV}{145.4\ pA} = 177.5\ M\Omega$. At an operating $V_{DD} = 3.3\ V$, the off–resistance is higher. At this level, the worst case bias current is 70 pA, which equates to an *off* resistance in the 1 GΩ range.

Figures 29c and 29d show the current and resistance of a switch as it is programmed mid–way between the *on* and *off* positions. As can be seen by the current measurements, these devices have been injected close to the *off* position. While this resistance is non-linear over the full operational range, linearity can be achieved given

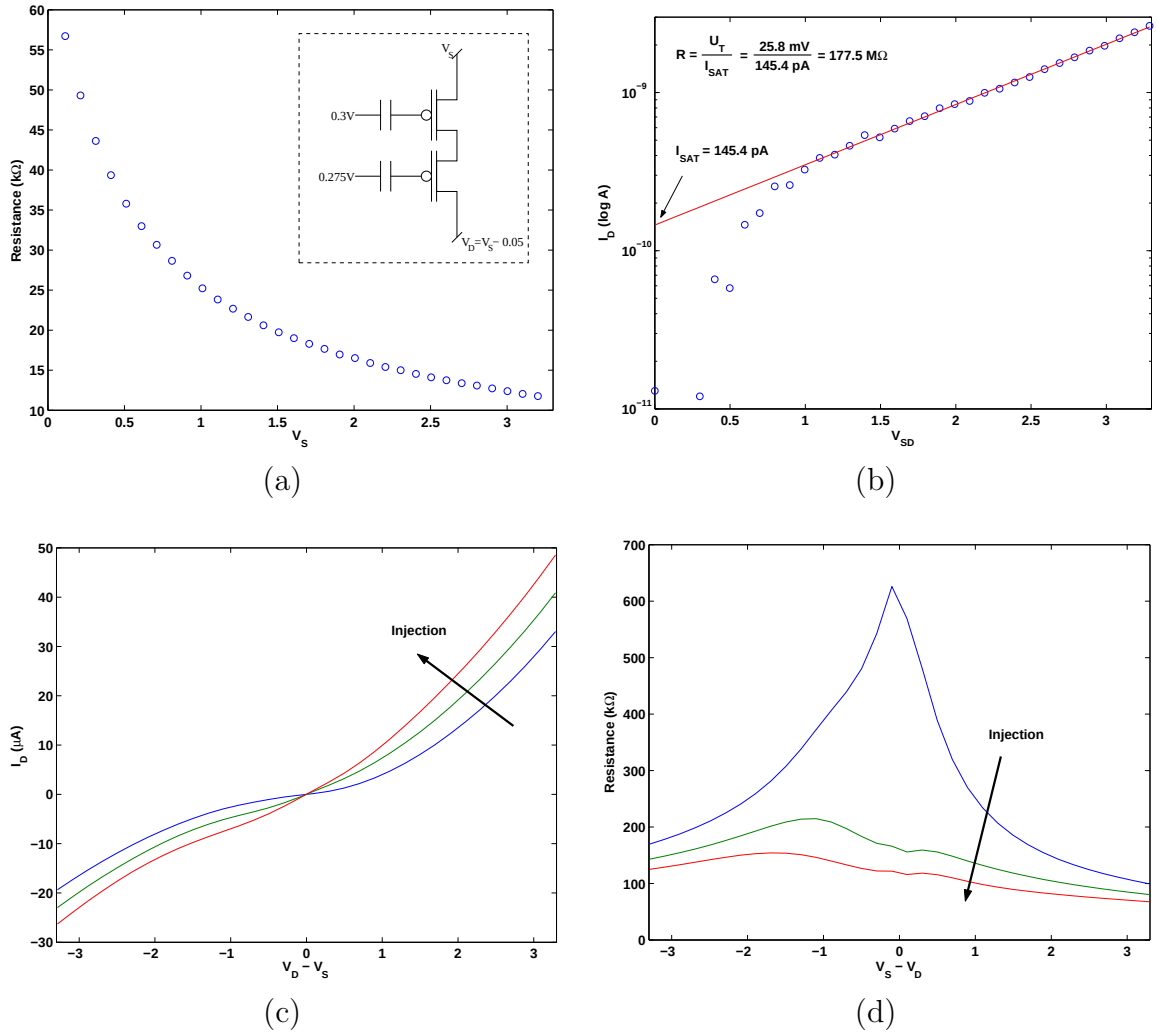


Figure 29. Switch Characteristics of Floating–gate Transistors: (a) Resistance for an *on* switch was measured using two floating–gate devices in series, both programmed to the *on* position. Here, the experimental resistance has already been divided by two to represent a single floating–gate switch. (b) Currents for an *off* switch are too low for reliable picoammeter measurements. So, a conservative *off* resistance was determined by measuring the saturation current of a floating–gate transistor programmed to the *off* position while $V_{DD} = 7.5\ V$. (c) The floating–gate switch can also be programmed mid–position (between *on* and *off*) to synthesize a variable resistance. Here, a sampling of the differential currents achievable with this programming scheme are shown. (d) The resistance plots correspond to the same injection levels as the differential currents shown in (c).

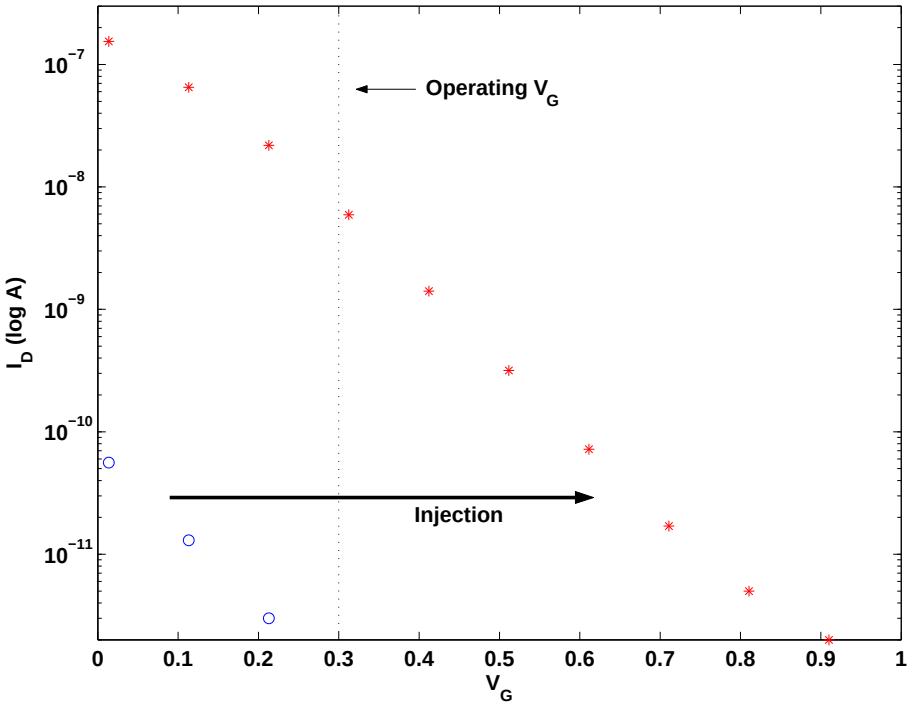


Figure 30. This is a floating–gate switch programmed to 8 nA at an operating gate voltage of 0.3 V. Thus, current sources can be accurately defined within the switching network itself.

a sufficient constraint on the input.

Using the programming method described in [74], floating–gate switches in the FPAA can be accurately programmed. Figure 30 shows a switch that was programmed to an arbitrary current of 8 nA at an operating gate voltage of 0.3 V. Thus, floating–gate switches can be used to accurately set a current level within the system (i.e., these devices can be used to implement a current source).

4.5 System Results

As an initial example of the testbed system, a first–order filter is implemented using an OTA in one of the CABs. Figure 31 shows how the circuit is mapped onto the FPAA using five floating–gate switches. Once the switch network is configured, the biasing floating–gate transistor is programmed to vary the corner frequency of this first–order filter. The frequency response is shown for several programmed corner

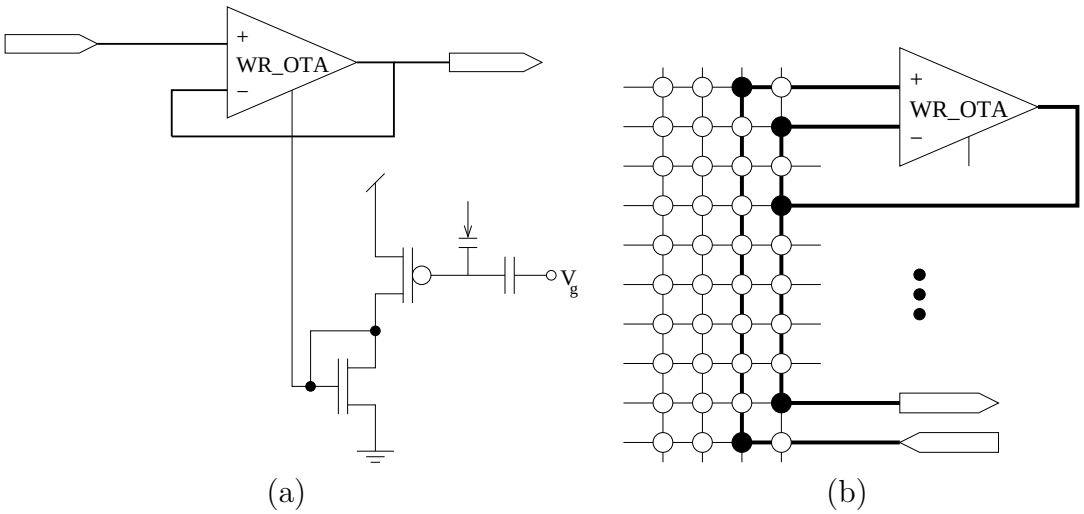


Figure 31. (a) The source–follower configured using a floating–gate current source. By programming the floating gate charge, the current is set in the current mirror (the other half of the current mirror is internal to the wide–range OTA) is fixed. Thus, the effective conductance can be modified for each of the OTAs on the chip. (b) Using the switch matrix, an OTA located in one of the Computational Analog Blocks (CABs) is connected in a source–follower configuration, and two external pins are routed to the OTA as the input and output signals. The programmable biases illustrated in (a) are not shown here for simplicity, but each OTA has a current mirror and floating–gate current source that sets its bias.

frequencies in Fig. 32. The moderate gain in the lower frequencies is due to the switches in the feedback loop of the OTA. Ideally, the output node and the negative input node would be directly connected. However, in the FPAA, this path must be routed via the switch network, which means that a minimum of two floating–gate switches will be in the feedback loop. The gain can be minimized by injecting the floating–gates of these switches to a lower charge, or if gain is desired for a given application, then it can be set by programming these switches to a higher charge.

In Fig. 33, a second–order section filter is shown along side the FPAA implementation. Once again, explicit capacitors are eliminated since the switch parasitics provide the necessary capacitance. Using the floating–gate programmble biases, the two OTAs in a source–follower configuration were biased to the same level and the the third OTA's bias current was increased to adjust the Q–peak of the system. The simulated frequency response for this circuit is shown in Fig. 34. As expected, the

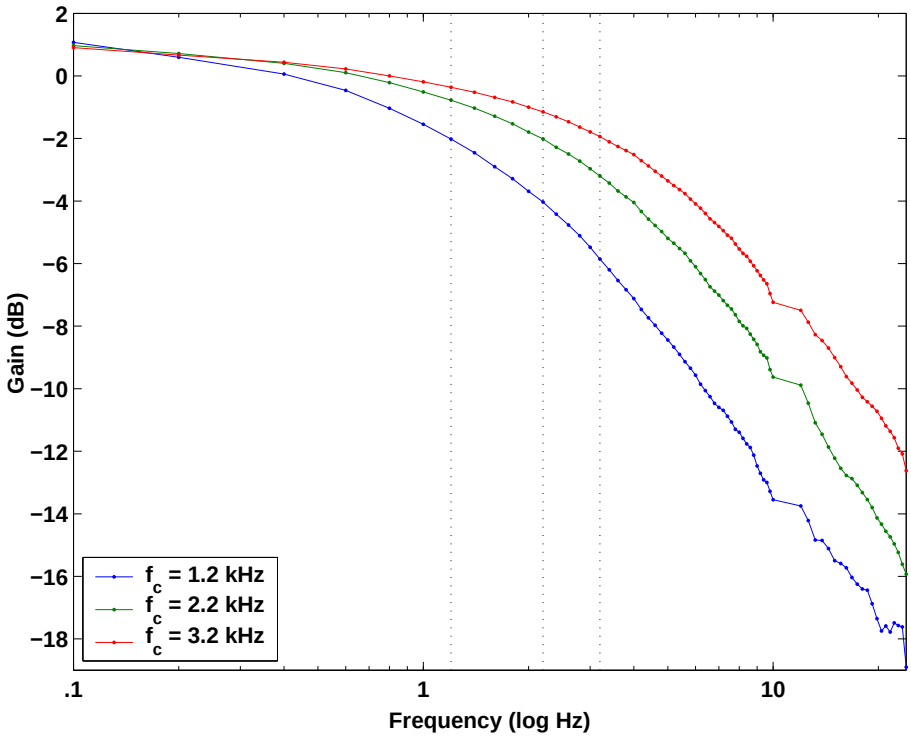


Figure 32. Here, the frequency response of the source–follower circuit is shown for several bias currents. An internal floating–gate transistor is used as a current source to set the OTA's bias. Injecting the floating–gate device, increases the current and thus the bandwidth of this first order filter.

Q–peak increases as the bias current (e.g., conductance) increases. The experimental frequency response shown in Fig. 35, also shows a programmable Q–peak; however, the peak is not very high. This is most likely an effect of the switches in the signal paths. Improvements were made to the OTA and switch components in RASP 1.5, and a much higher Q–peak resulted (see Chapter 5).

For second–order functions such as the second–order section and the *diff2* circuit introduced by Mead in [62], reasonable Q–peaks and filter bandwidths require small bias currents (in the picoamp to femtoamp range). While the floating–gate transistors can set bias currents this low, the constraint becomes the ability to accurately measure these currents while programming the floating–gate transistors. Experimental results from Fig. 28 show a measurement threshold of 1 pA using present measurement techniques. An important consideration here is the relative size of the transistors

that set the bias currents. The floating–gate transistor shown in Fig. 31a sets the current through the nMOS current mirror (the other half of the current mirror is internal to the OTA module). To set small bias currents, it is preferable to have the nFET and floating–gate transistor sized larger than the current mirror nFET, which is internal to the OTA. In this configuration, the current mirror functions as a current divider, and thus, very low bias currents can be set by programming the floating–gate transistor to generate currents in the picoamp range.

Based on these testbed systems, one can start to imagine a wide class of systems that can be implemented and configured on FPAAs with many CABs on them. In particular, differentiators, cascaded second–order sections, bandpass filters, matrix transforms (including DCTs and wavelet transforms), and frequency decomposition are all well–suited for this architecture. In the audio arena alone, designs could be prototyped to implement forms of noise suppression, audio enhancement, feature extraction, auditory modeling, and simple audio array processing. Other potential interest areas include communications signal conditioning (modulation, mixing, etc.), transform coding, and neural networks (with external training). Most of these systems rely on efficient subband processing; therefore, each CAB has been designed with a C^4 bandpass filter to optimize this operation. A realistic simulation of this C^4 block is shown in Fig. 36 with the center frequency of the C^4 filter being moved over a large range of frequencies.

4.6 Summary

FPAAs based on floating–gate technologies are an emerging design concept that will increase the current state of the art in the analog and mixed–signal prototyping. In particular, these FPAAs are well–suited to facilitate the design of low–power signal processing systems based on analog floating–gate devices. In this chapter, a

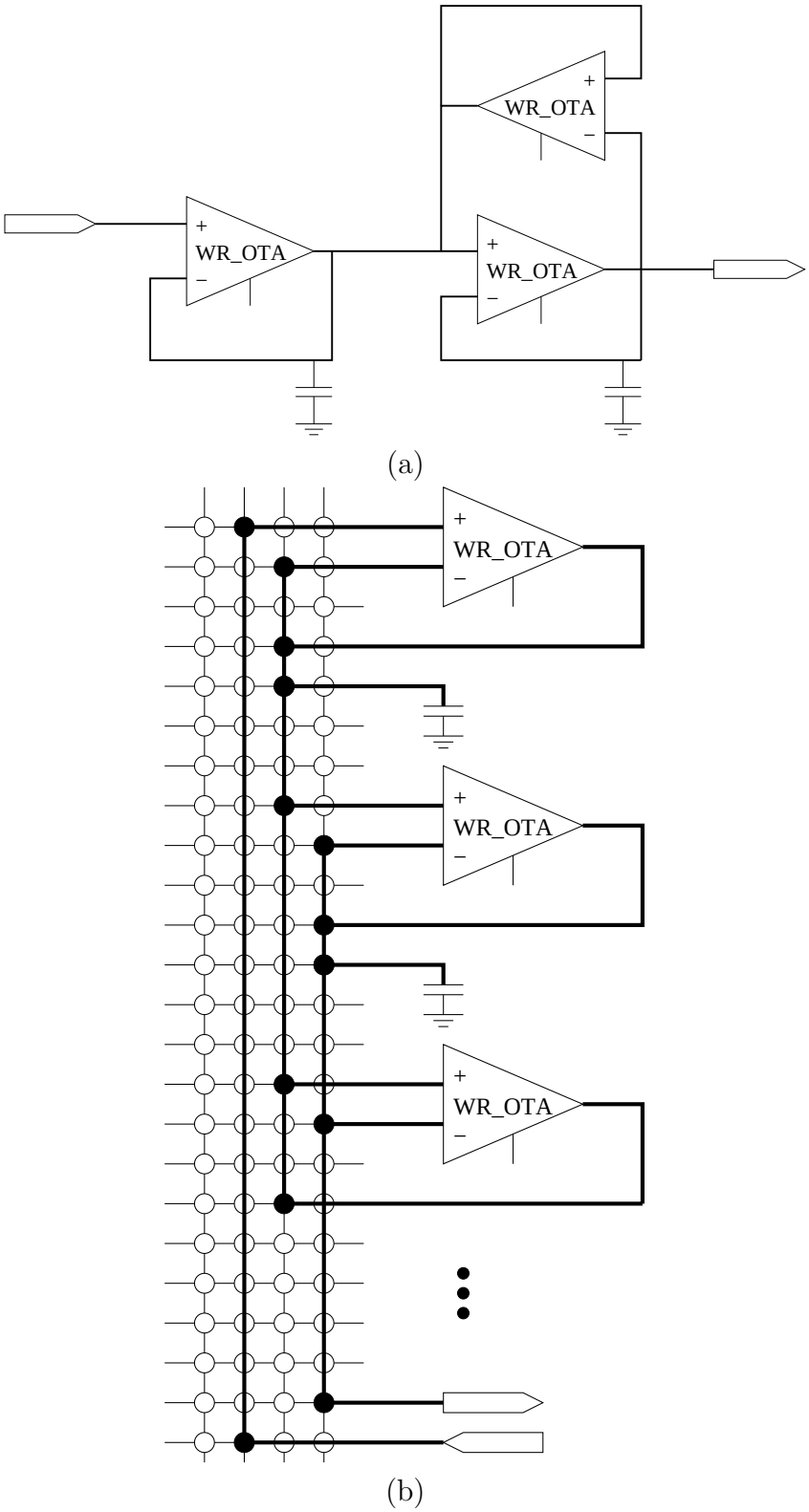


Figure 33. (a) A second–order section filter can be implemented with two OTAs in a source–follower configuration and a third OTA that creates postive feedback. (b) Using the switch matrix, two OTAs within the CABs are connected in a second–order section configuration. The programmable biases shown in Fig. 31(a) are not included here for simplicity, but each OTA has a current mirror and floating–gate current source that sets its bias.

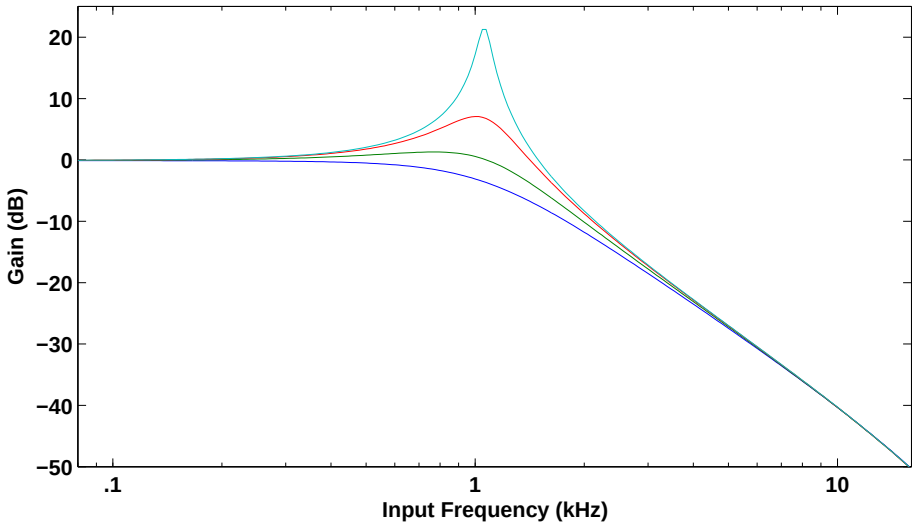


Figure 34. The simulated frequency response of a second–order section filter is shown here. The Q parameter is adjusted by increasing the bias current of the positive feedback amplifier via a floating–gate current source.

novel FPAA architecture was presented that utilizes floating–gate transistors as programmable switches, in–circuit active elements, and the configurable device within the computational analog blocks. Floating–gate switches were characterized and several systems were implemented on this FPAA. In the next chapter, an updated version of the RASP 1.0 FPAA is discussed, and additional system–level data is shown.

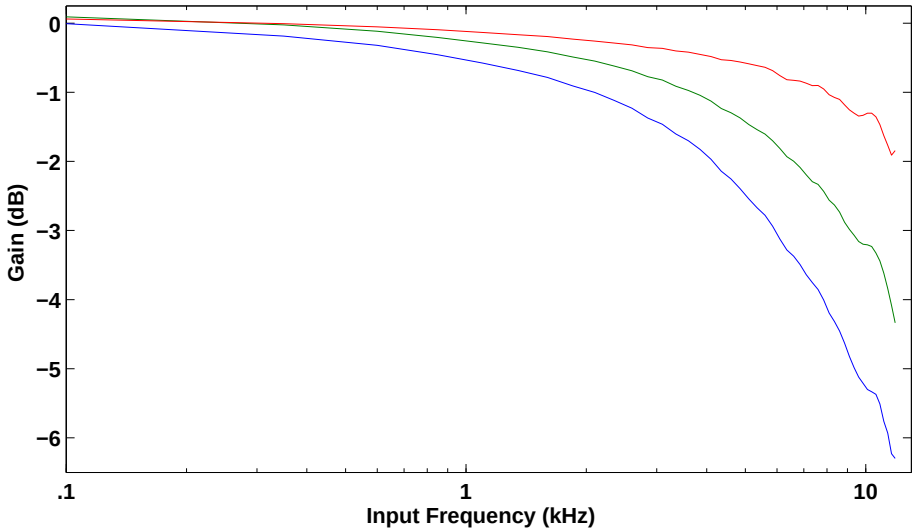


Figure 35. The experimental frequency response of a second–order section filter is shown here. The Q parameter is adjusted by increasing the bias current of the positive feedback amplifier via a floating–gate current source.

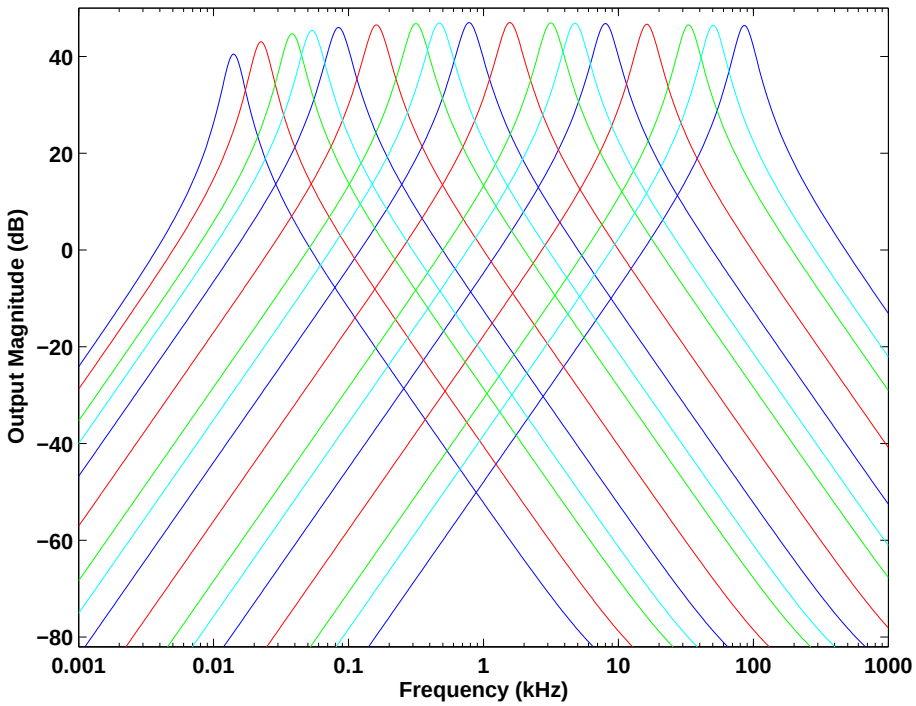


Figure 36. Frequency decomposition (subband processing) can be achieved on the RASP 1.0 FPAA by using the C^4 bandpass filter block in each CAB. In this simulation, the center frequency of the C^4 is shown to be programmable over a wide range of frequencies.

CHAPTER 5

RASP 1.5

The RASP 1.0 FPAA allowed the switches in the switch network to be characterized and some small systems to be implemented on it. However, a number of circuit–level changes were needed to realize the full functionality of the RASP architecture. RASP 1.5 is an FPAA that is similar in size to RASP 1.0, but it has a number of small circuit and architectural improvements that allow the system–level functionality of the computational logic to be further tested.

5.1 RASP 1.5 Architecture

The RASP 1.5 FPAA is fabricated in AMI's 0.5-micron CMOS process. A picture of the top–level layout is shown in Fig. 37. A die photo of RASP 1.5 is shown in Fig. 38.

5.1.1 RASP 1.5 vs. RASP 1.0

The improvements to RASP 1.5 can be categorized as programming infrastructure, circuit–level, and architectural changes. In the programming infrastructure, advancements were made to bring more of the programming logic on the chip. During programming mode, the control logic on RASP 1.5 automatically sets the gate and source voltages on the non–selected rows to turn all switches off except the one currently selected.

At the circuit level, a number of the transistors were resized. The switches in the switch network were made bigger to reduce the resistance through the devices. The T–gate structures in the programming control logic were also made much bigger to allow more current to flow through them. This is important when turning on multiple switches in a column, since the currents through the multiple switches will sum on the column lines. If the T–gates are not large enough, then the current through the switches will overload the T–gate causing a voltage drop across the T–gate. This

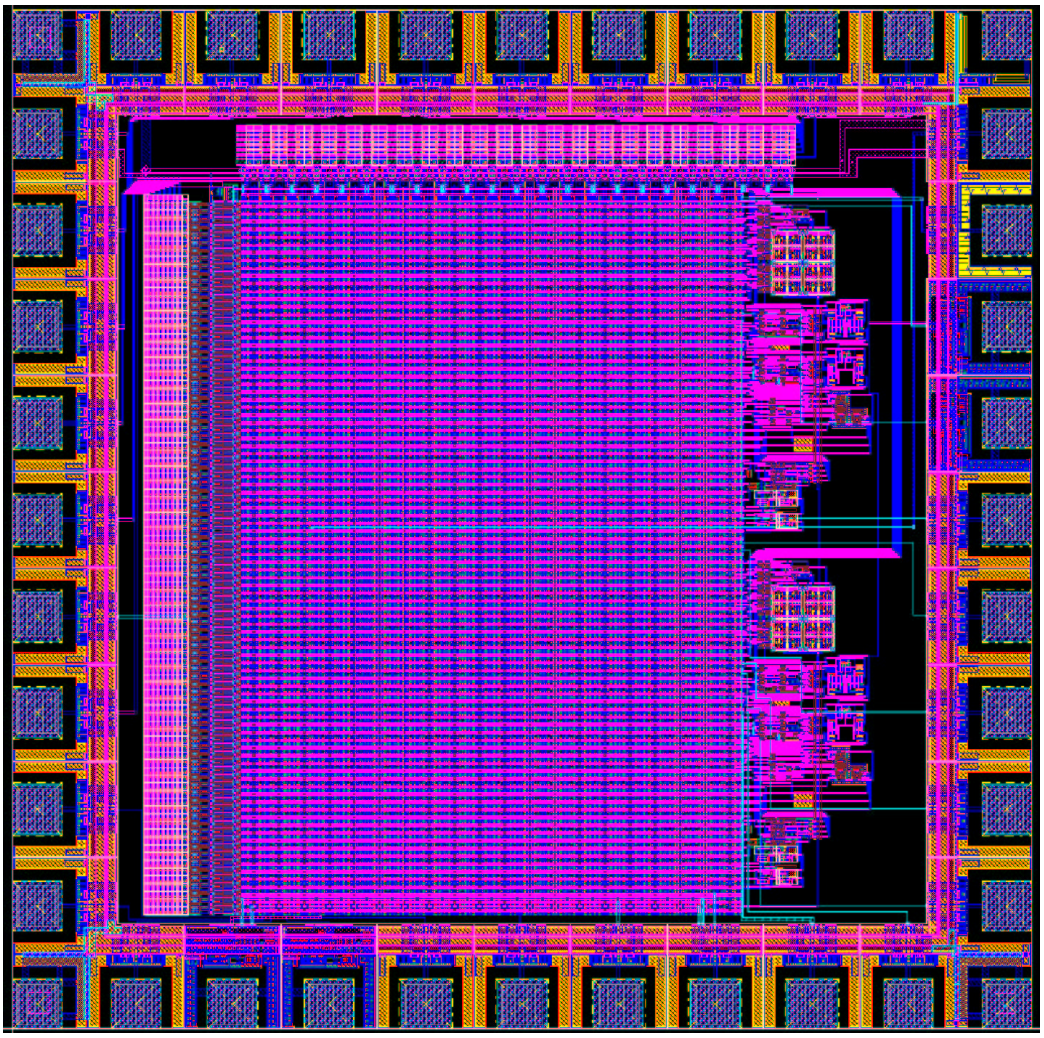


Figure 37. This is a picture of the top–level layout for the RASP 1.5 FPAA. It has two CABs and a single switch network connecting the CABs.

raises the drain voltage on the switches and slows or even stops injection. Additional changes included modifying transistor sizing in the CAB components to improve the performance of the computational logic elements. In particular, current dividers were added on all of the biases so that the currents needing to be sourced by floating–gate current sources would be in an optimal programming range for the floating–gate transistors (100 pA – 10 uA).

As a part of the architectural changes, two levels of I/O hierarchy are introduced. On RASP 1.0, signals coming into or going off of the FPAA were only allowed to do

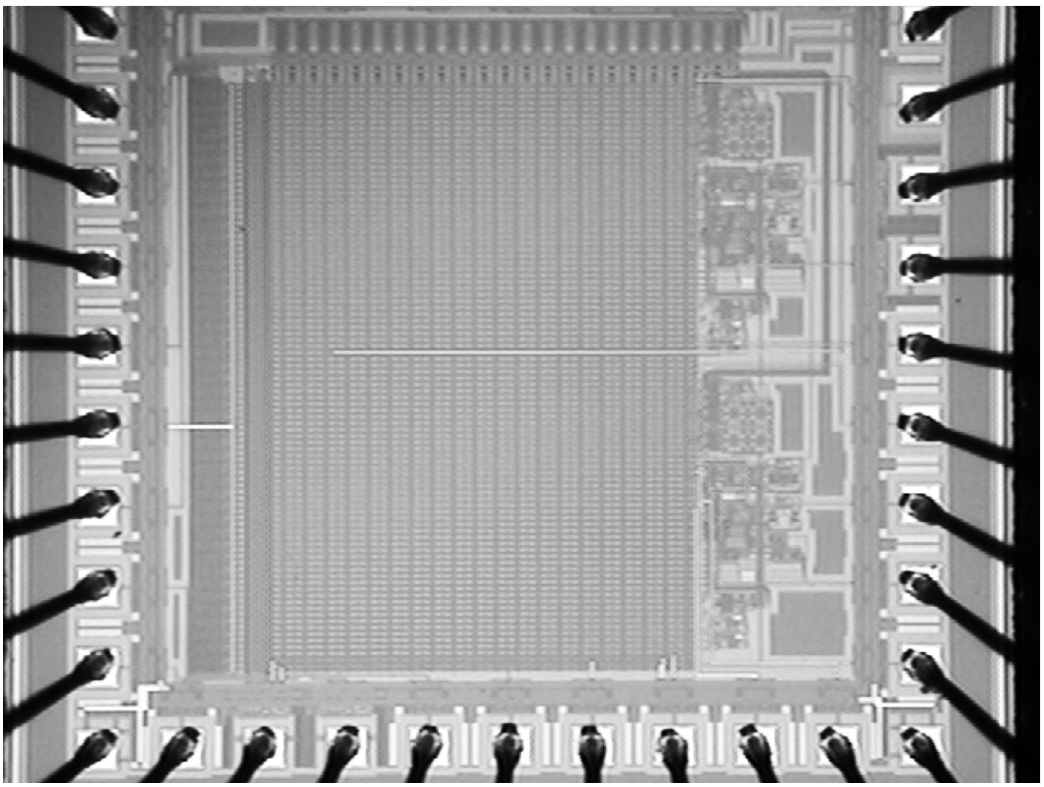


Figure 38. This is a die photo for the RASP 1.5 FPAA. It has two CABs and a single switch network connecting the CABs together.

so only through source lines. This scheme requires a minimum of two switches to get an input or output from the I/O pad to a component input or output. On RASP 1.5, however, two levels of I/O signals are included: Level 1 and Level 2. Level 2 I/O is routed through the source lines just like RASP 1.0. Level 1 I/O, however, is routed into the columns (drain lines). Signals entering RASP 1.5 on Level 1 I/O can be routed to the input of any component with a single switch. Adding I/O pads to the columns dramatically increases the capacitance on those drain lines, which results in lower bandwidths on those nodes. Therefore, only a limited number of drain lines are used as Level 1 I/O lines. The majority of the columns are reserved for internal routing.

The second part of the architectural changes involved increasing the size and functionality of the CABs. First, three fixed–value capacitors, two transistors (one

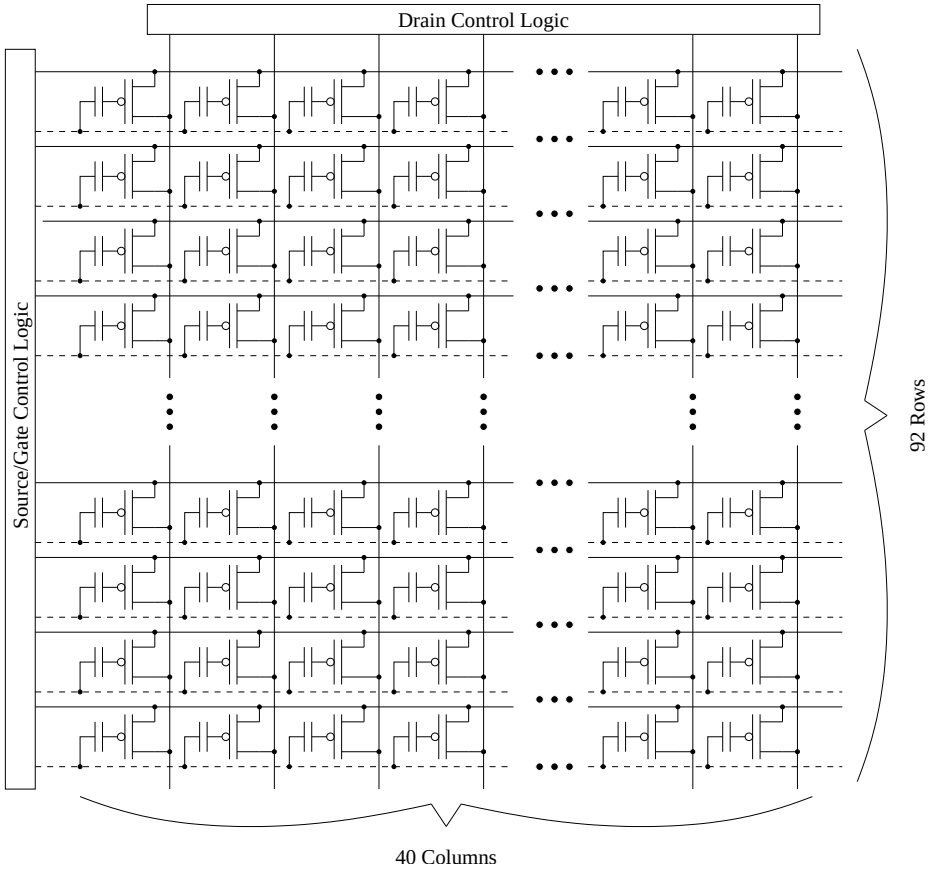


Figure 39. The RASP 1.5 has a single fully connectable crossbar switch that has 40 x 92 switches in it. The gate and source nodes from the transistors run horizontally into the programming control logic and row decoders on the side of the chip. The drain lines for the columns and run vertically into the drain control lines and column decoders.

nFET and one pFET), a peak detector, and a min detector were added. Additionally, the C^4 bandpass filter block was replaced with a C^4–based second–order section (SOS) bandpass filter module (see Section 5.2).

5.1.2 Switch Networks

As shown in Fig. 39, the routing infrastructure on RASP 1.5 is achieved with a single, fully connectable crossbar switch. The switch network has 40 columns (drains) and 92 rows (sources). The inputs and outputs from all of the components in the two CABs connect to the rows of the switch network. I/O lines that go off the FPAA chip are divided into two levels: Level 1 and Level 2 I/O. Level 1 I/O pins are connected

directly to the columns allowing them to be connected to any input or output on the computational logic through a single switch. There are six Level 1 I/O pins on RASP 1.5 and two of them are dedicated outputs since they are routed through an output buffer in their I/O pads. There are 12 Level 2 I/O pins. They connect to the switch network via extra rows that are not used for the computational logic. Level 2 I/O pins require at least two switches to connect them to any input or output of the computational logic.

5.2 Computational Analog Blocks

The computational logic is organized in a compact computational analog block (CAB) providing a naturally scalable architecture. CABs are tiled across the chip in a regular mesh–type architecture with busses and local interconnects in between. In RASP 1.5, there are two CABs with a single switch network connecting them together.

Many potential CABs can be imagined using this technology. Figure 40 shows one example CAB, whose functionality is enhanced by a mixture of fine–grained, medium–grained, and coarse–grained computational blocks similar to many modern FPGA designs. The computational blocks were carefully selected to provide a sufficiently flexible, generic architecture while optimizing certain frequently used signal processing blocks. For generality, three OTAs are included in each CAB. OTAs have already been shown to be effective at implementing a large class of systems, including amplification, integration, filtering, multiplication, exponentiation, modulation, and other linear and non-linear functions [26, 65, 70, 71]. In addition, the two FET devices provide the ability to perform logarithmic and exponential functions, as well as convert from current to voltage and vice versa. The three capacitors are fixed in value in order to minimize the size of the CAB and are primarily used on the outputs of the OTAs; however, they will be available for any purpose. The use of floating–gate transistors in the OTAs will give the user sufficient control in programming the transconductance of

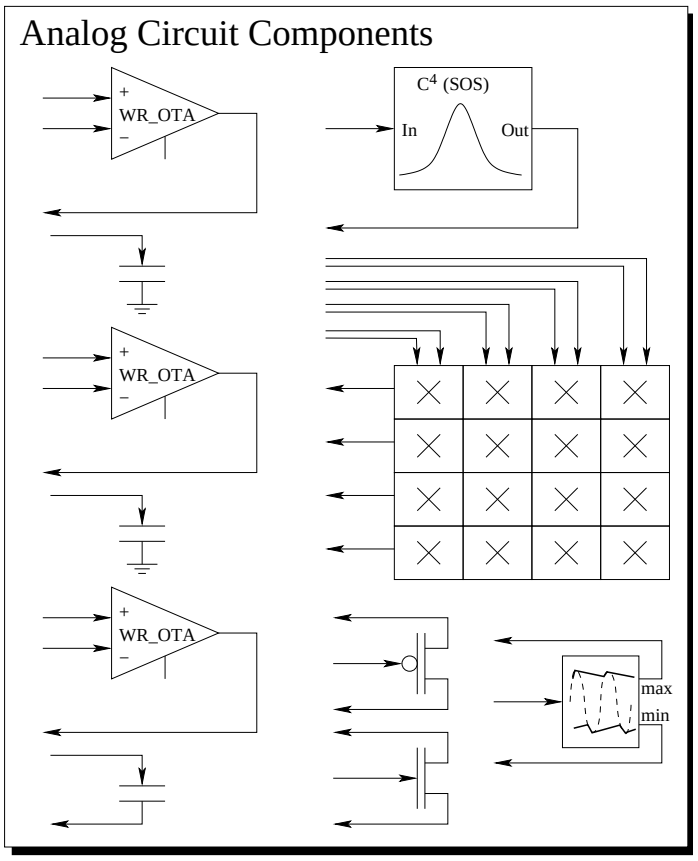


Figure 40. The CAB on RASP 1.5 contains a four-by-four matrix multiplier, three wide–range operational transconductance amplifiers (OTAs), three fixed–value capacitors, a capacitively coupled current conveyor (C^4) second–order section (SOS), a peak detector, a min detector, and two FET transistors.

the amplifiers, thereby eliminating the need for the variable capacitor and/or current mirror banks found in some designs [37, 65]. Removing the capacitor banks creates a large savings in the area required for each CAB.

The high–level computational blocks used in this design are an SOS bandpass filter module comprised of two C^4 and the 4 x 4 vector–matrix multiplier block. In general, the C^4 SOS module provides a straightforward method of subbanding an incoming signal. This allows Fourier analysis analagous to performing a Fourier transform. The vector–matrix multiplier block allows the user to perform a matrix transformation on the incoming signals. Together these blocks can be used like a Fourier processor [44, 52]. In addition, a peak detector and min detector is added to

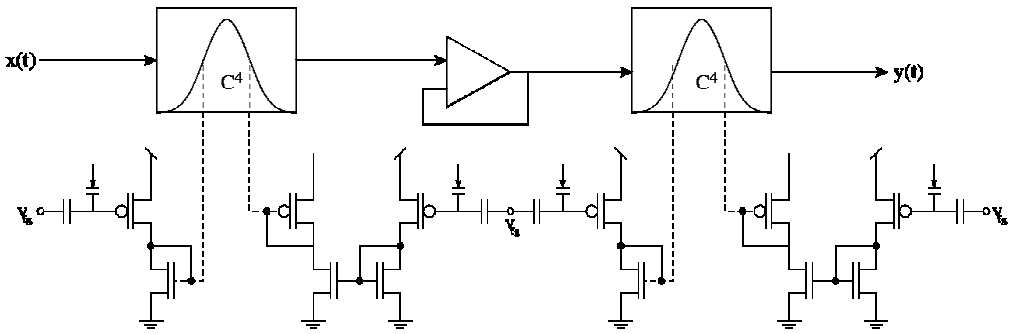


Figure 41. This is a block diagram of the second–order section (SOS) bandpass filter module used in RASP 1.5's CAB. The C^4 SOS is comprised of two C^4 circuits (as shown in Fig. 42) with a buffer in between them. The corner frequencies for both the C^4's are independently controlled with the floating–gate current sources located in the CAB.

each CAB.

The bandpass filter module is different from that of RASP 1.0. As shown in Fig. 41, the bandpass filter is an SOS that is comprised of two C^4 circuits with a buffer in between them. Both C^4 blocks have the circuit topology shown in Fig. 42. As shown in the figure, the C^4 circuit automatically sets the V_{dibl} gate voltage needed in RASP 1.0 (see Fig. 26). This increases the linear range of the module while maintaining relative ease of control. The corner frequencies for both of the C^4 blocks are controlled with the floating–gate current sources located in the CABs. This allows the maximum flexbility and control.

5.3 Testbed FPAA

RASP 1.5 contains two CABs with a floating–gate crossbar switch network connecting them. Both CABs are identical to the large CAB illustrated in Fig. 40.

As discussed earlier, the resistance and capacitance of the floating–gate switch are important characteristics. The *on* switch resistance is plotted in Fig. 43. For reference, this figure also shows the resistance of a standard pFET (with an SRAM memory bit setting the gate) and T-gate (both an nFET and a pFET passing the signal). When programmed to a point that is not extremely *on*, the floating–gate

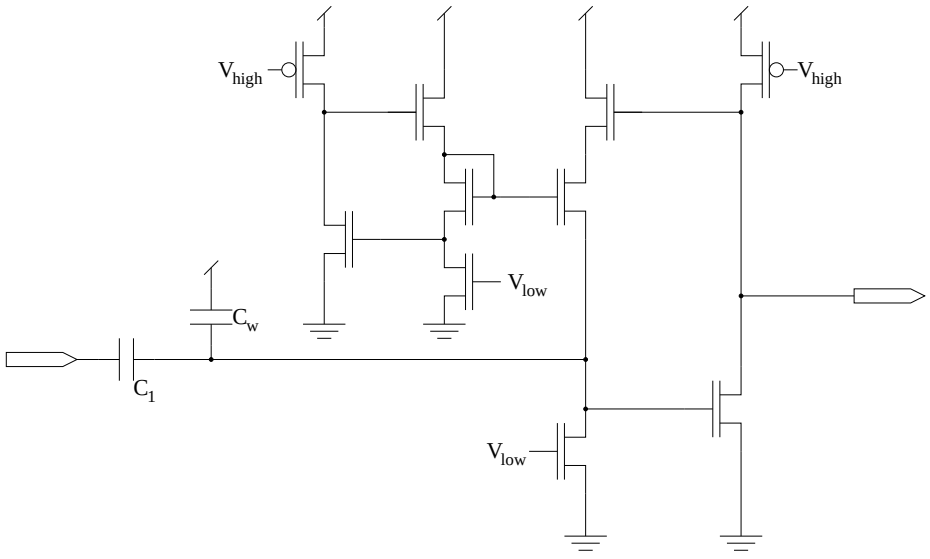


Figure 42. This is the version of the capacitively coupled current conveyor (C^4) used in the second–order section in RASP 1.5's CAB. It does not use an explicit C2 capacitor in the middle node. Also, this diagram includes an "autodibl" circuit that automatically sets the V_{dibl} gate voltage based on the selected corner frequencies. This increases the linear range of the module while maintaining relative ease of control.

switch exhibits a resistance that is very similar to the standard pFET shown here (as seen in [37]. However, by injecting the floating–gate switch further, the voltage on the isolated gate node is pushed lower and thus the resistance curve shifts to the left. This figure shows that by programming the switch far enough, the resistance through the switch can maintain a more consistent level through the operating range (power rails) of the switch. This allows a single floating–gate pFET to exhibit a resistive characteristic that is similar to the resistance of a standard T-gate with two transistors. As shown, the resistance of the floating–gate switches is approximately 10 KΩ, which is about what is expected for relatively small (W/L = 3) pFETs.

The *off* resistance is harder to measure given the limitations of standard test equipment. Even at $V_{DS} = 3.3V$, current through the *off* switches is below the measureable range of standard picoammeters. Given this, the *off* resistance should be in the gigaohm range, and in the worst case, hundreds of megaohms. Likewise, the parasitic capacitance of the switches is difficult to measure when they are embedded

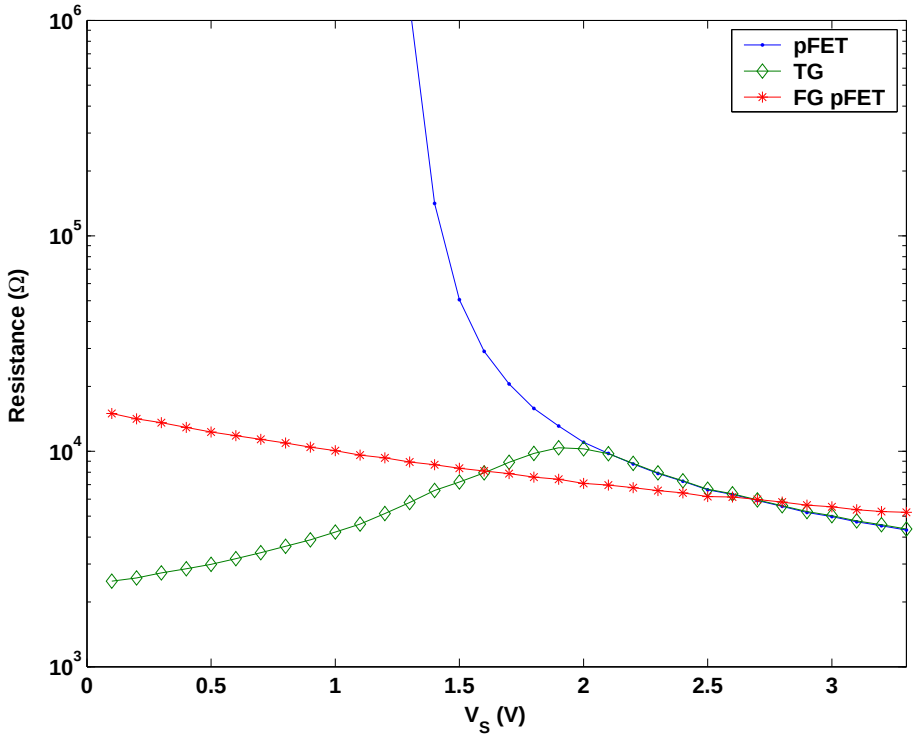


Figure 43. Here the switch resistances for a floating–gate pFET, standard pFET controlled by an SRAM memory cell, and a standard two–transistor T–gate are shown. The floating–gate switch has been programmed to an extremely *on* position such that the high–impedance region (at 1.5 V for the standard pFET shown) has effectively been shifted below the power rail. This results in a relatively flat resistance similar to the larger T–gate.

in the switch network and accessible only through the programming infrastructure. A theoretical estimate based on the layout and fabrication parameters yields a value of 1 fF for each switch on each column and row. Thus, for RASP 1.5, each column is estimated to contribute 96 fF of parasitic capacitance and 46 fF for each row.

5.4 System Results

A number of different analog testbed systems have been synthesized on RASP 1.5. These systems vary from simple one– and two–element systems to more complex systems with as many as seven on–chip components. These systems also use a range of different CAB components, including fine–grained (transistors and capacitors), medium–grained (OTAs), and coarse–grained (C^4 SOS and peak detector). In each

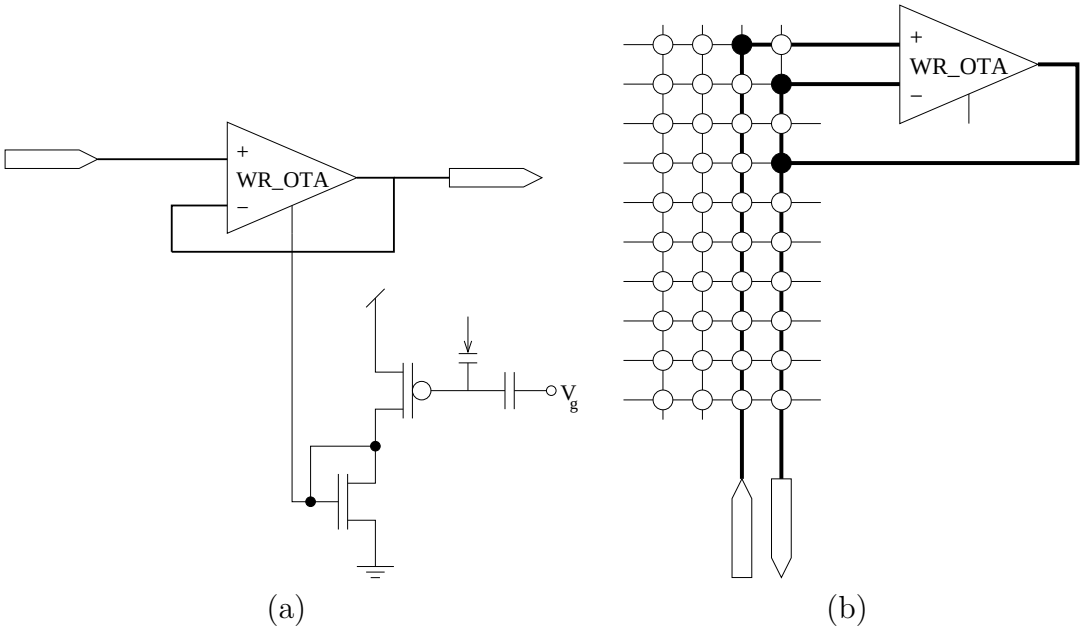


Figure 44. (a) The source–follower is configured using a floating–gate current source. By programming the floating gate charge, the current is set in the current mirror (the other half of the current mirror is internal to the wide–range OTA). Thus, the effective conductance can be modified for each of the OTAs on the chip. (b) Using the switch matrix, an OTA located in one of the Computational Analog Blocks (CABs) is connected in a source–follower configuration, and two external pins are routed to the OTA as the input and output signals. The programmable biases illustrated in (a) are not shown here for simplicity, but each OTA has a current mirror and floating–gate current source that sets its bias.

of these examples, floating–gate transistors are used as current sources to set biases. Depending on the circuit, these programmable biases are shown to control filter corner frequencies, Q–peaks, and time constants.

5.4.1 Low–Order Filtering with OTAs

As an initial example of the testbed system, a first–order filter is implemented using an OTA in one of the CABs. Figure 44 shows how the circuit is mapped onto the FPAA using five floating–gate switches. Once the switch network is configured, the biasing floating–gate transistor is programmed to vary the corner frequency of this first–order filter. The frequency response is shown for several programmed corner frequencies in Fig. 45. The plot in Fig. 46 shows the correlation between programmed bias current and measured corner frequency. By fitting a curve to this data, it is possible to predict

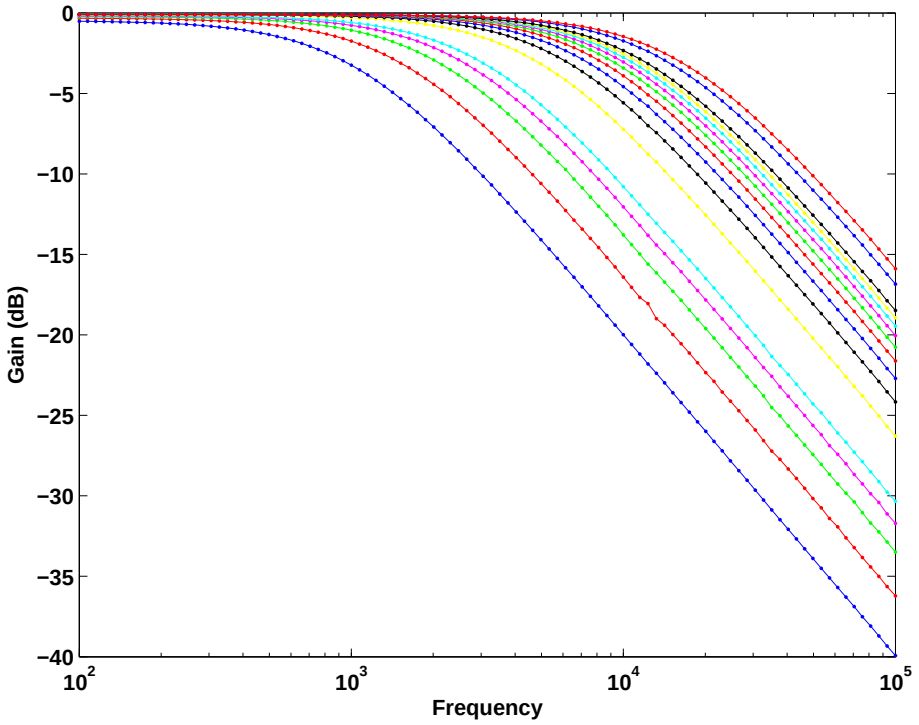


Figure 45. Here, the frequency response of the source–follower circuit is shown for several bias currents. An internal floating–gate transistor is used as a current source to set the OTA's bias. Injecting the floating–gate device, increases the current and thus the bandwidth of this first order filter.

the necessary bias current for a desired corner frequency. This is very important for the usability of FPAAs in general. The end–user will typically want to specify the system parameters in terms of corner frequency, Q-peak, time constants, offsets, etc., and then let the programming interface make the translation to the appropriate bias currents necessary to generate these parameters.

In Fig. 47, an SOS filter is shown alongside the FPAA implementation. Once again, explicit capacitors are eliminated since the switch parasitics provide the necessary capacitance. Using the floating–gate programmble biases, the two OTAs in a source–follower configuration were biased to the same level, and the third OTA's bias current was increased to adjust the Q–peak of the system. The frequency response for this circuit is shown in Fig. 48. As expected, the Q–peak increases as the bias current (i.e., conductance) increases.

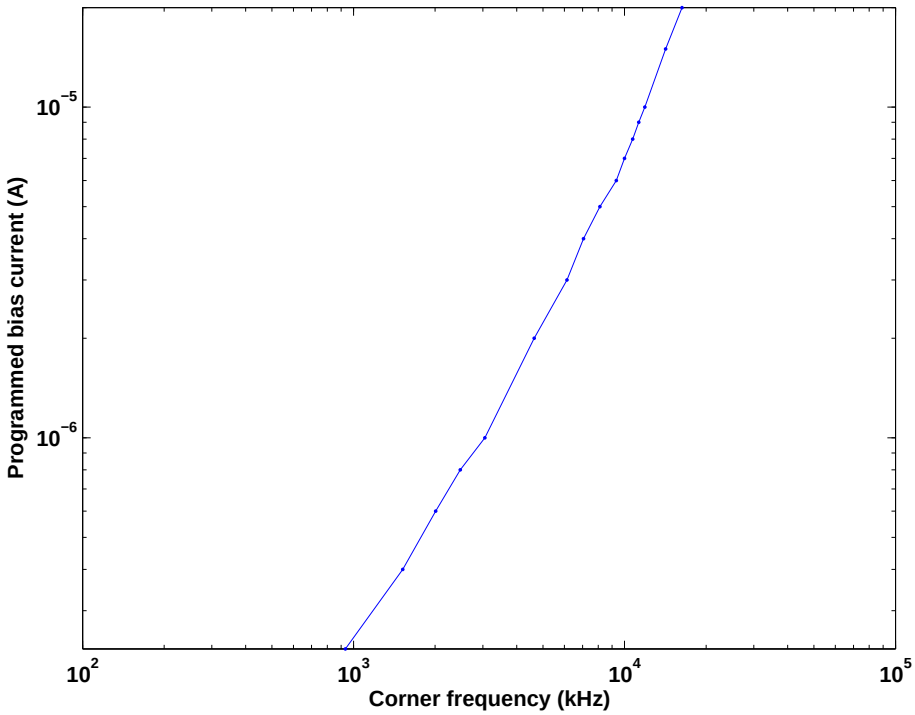


Figure 46. This plot shows the correlation between bias current and programmed corner frequency. This data can be used in future experiments to select the appropriate bias current for the desired corner frequency.

For second–order functions, such as the second–order section and *diff2* circuit, reasonable Q–peaks and filter bandwidths require small bias currents (in the picoampere to femtoampere range). While the floating–gate transistors can set bias currents this low, the constraint becomes the ability to accurately measure these currents while programming the floating–gate transistors. Experimental results from Fig. 28 show a measurement threshold of 1 pA using present measurement techniques. An important consideration here is the relative sizing of the transistors that set the bias currents. The floating–gate transistor shown in Fig. 44a sets the current through the nMOS current mirror (the other half of the current mirror is internal to the OTA module). To set small bias currents, it is preferrable to have the nFET and floating–gate transistor sized larger than the current mirror nFET, which is internal to the OTA. In this configuration, the current mirror functions as a current divider, and thus, very

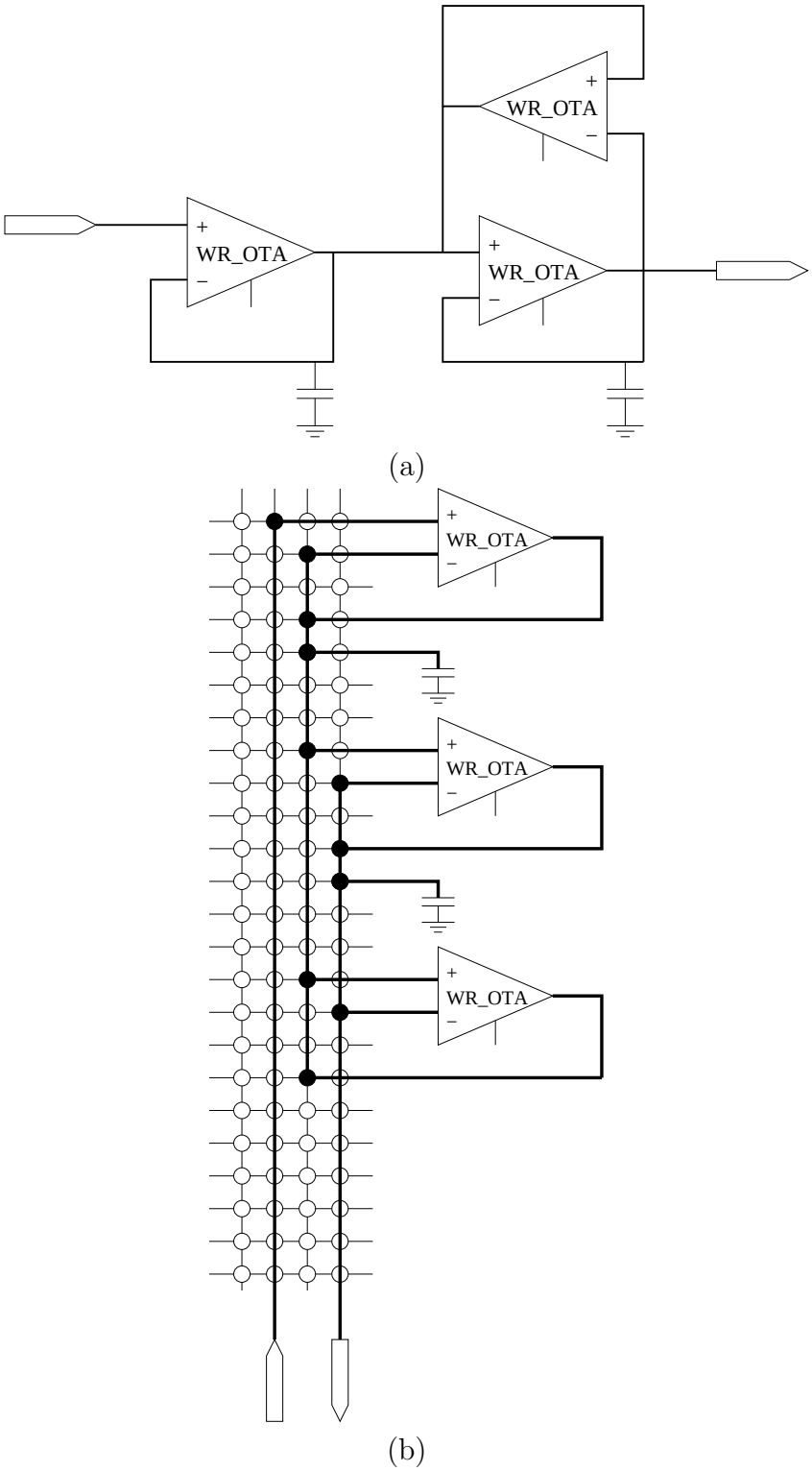


Figure 47. (a) A second–order section filter can be implemented with two OTAs in a source–follower configuration and a third OTA that creates postive feedback. (b) Using the switch matrix, two OTAs within the CABs are connected in a second–order section configuration. The programmable biases shown in Fig. 44a are not included here for simplicity, but each OTA has a current mirror and floating–gate current source that sets its bias.

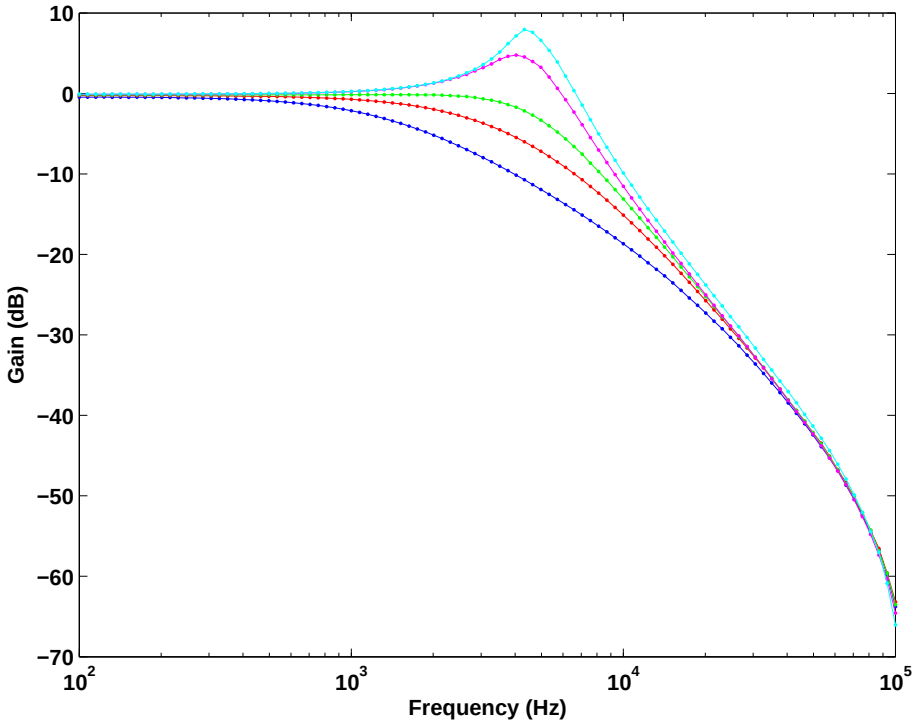


Figure 48. The experimental frequency response of a second–order section filter is shown here. The Q parameter is adjusted by increasing the bias current of the positive feedback amplifier via a floating–gate current source.

low bias currents can be set by programming the floating–gate transistor to generate currents in the picoampere range.

5.4.2 Third–Order Gm-C Ladder Filter

The availability of OTAs and grounded capacitors makes the RASP FPAA ideal for implementing Gm-C filters. One way to realize a particular filter is by modeling it with resistors, inductors, and capacitors, and then synthesizing the design using Gm-C filters. In this example, a third–order Butterworth filter is implemented. The canonical prototype of the filter, a double–resistance terminated LC filter, is shown in Fig. 49a. By using the signal simulation method outlined in [79], the Gm-C filter shown in Fig. 49b is generated. In order to maintain a maximally flat response, the following must hold: $2 * g_{m1} = g_{m2}$. Accordingly, the bias current of OTA-3 was set to half of the other OTA bias currents. A range of bias currents was used to create the

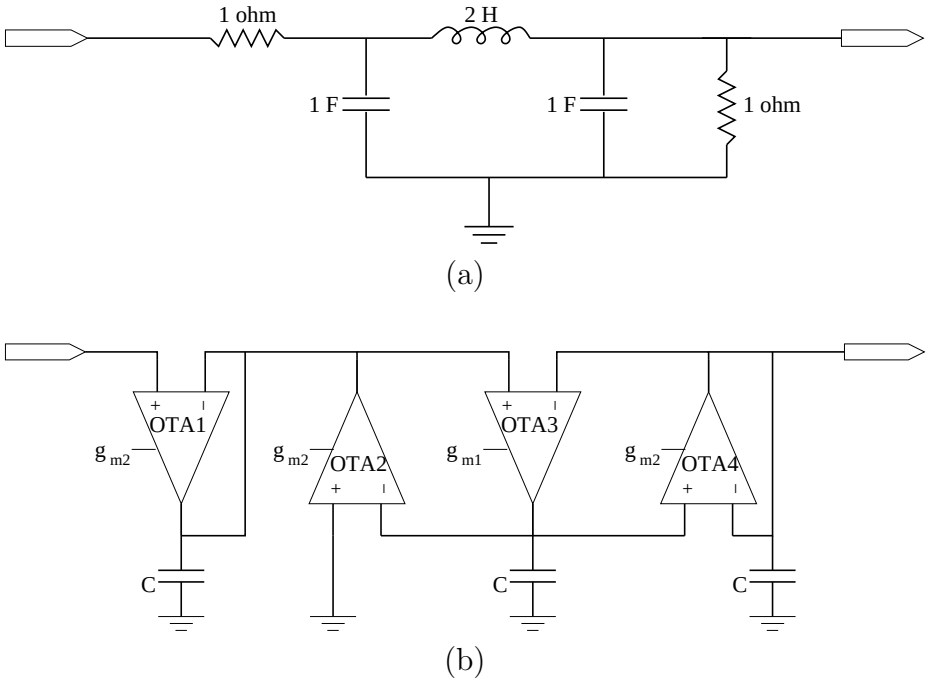


Figure 49. (a) This is the canonical prototype of a third–order Butterworth double–resistance terminated LC filter. (b) This is the Gm-C implementation of the same filter. This form of the filter can be realized on the RASP 1.5 FPAA.

frequency response shown in Fig. 50. As expected, the corner frequency of the filter is proportional to the bias currents of the OTAs. The lower corners were obtained by using a bias current in the range of hundreds of picoamperes, while the highest corners required currents of up to one microampere.

5.4.3 Coarse–Grain CAB Components

As mentioned earlier, the CABs on this FPAA have several special–purpose components that have been designed to optimize specific functions. In particular, these CABs include programmable peak detectors and programmable bandpass filter modules (C^4 SOS circuits).

The peak detector's operation is shown in Fig. 51. A sine wave is input with a sudden increase magnitude occurring in the middle of the data window. The output of the peak detector is shown to follow the peak of the sine wave through this transition. In addition, the time constant of the decay for this peak detector can be varied using

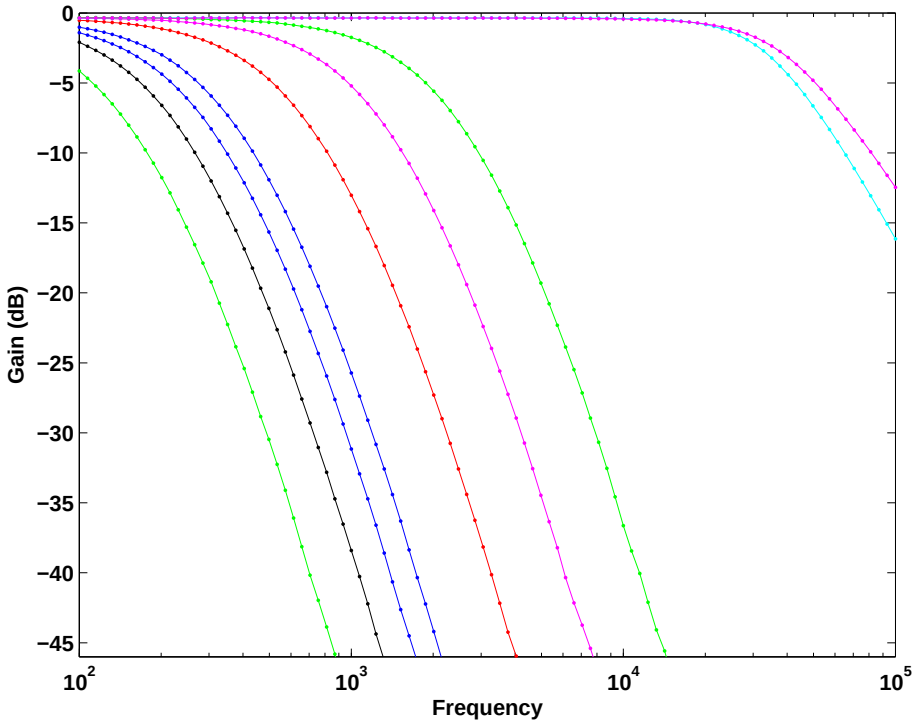


Figure 50. The experimental frequency response of a third–order Gm-C filter is shown here. The corner frequency is adjusted by programming the bias currents of the four OTAs.

a floating–gate controlled bias. The output of the peak detector is shown for two different time constants.

There is a wide range of systems that can be implemented and configured on FPAAs with many of these CABs on them. In particular, differentiators, cascaded second–order sections, bandpass filters, matrix transforms (including DCTs and wavelet transforms), and frequency decomposition are all well suited for this architecture. In the audio arena alone, designs could be prototyped to implement forms of noise suppression, audio enhancement, feature extraction, auditory modeling, and simple audio array processing. Other potential interest areas include communications signal conditioning (modulation, mixing, etc.), transform coding, and neural networks (with external training). Many of these systems rely on efficient subband processing; therefore, each CAB has been designed with a C^4 SOS bandpass filter module to optimize this operation.

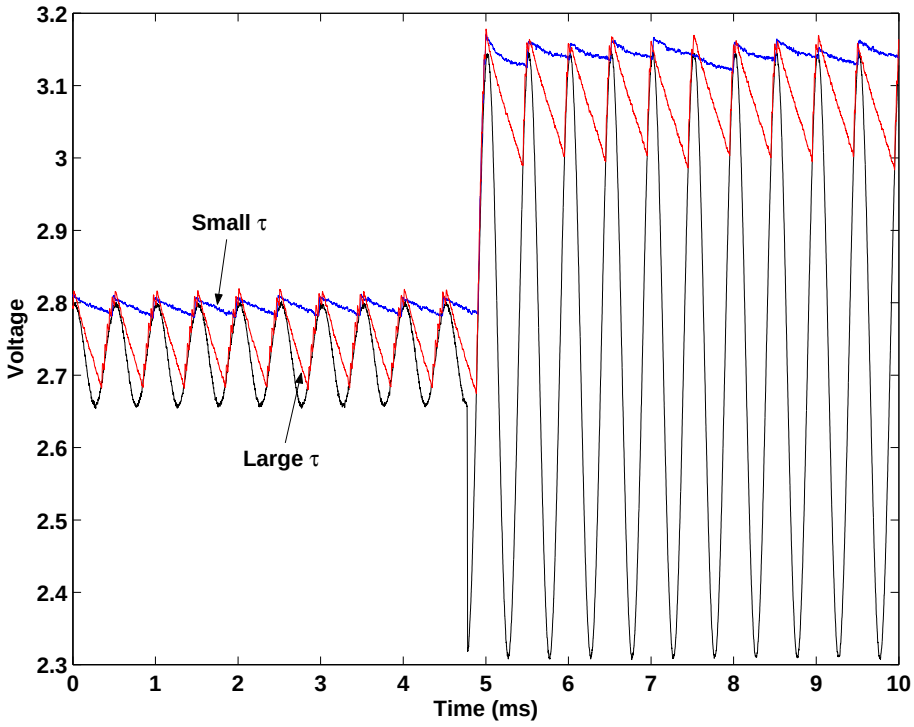
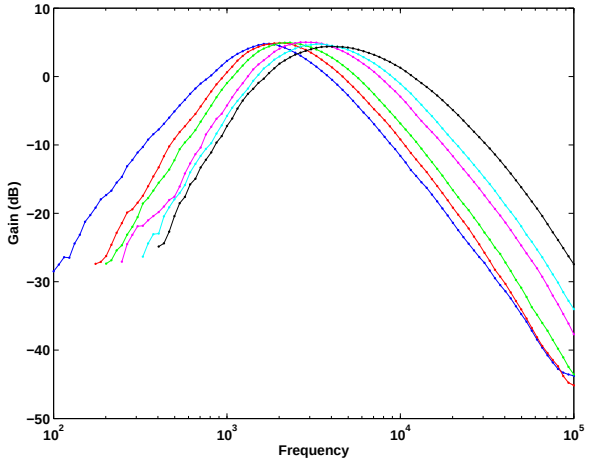


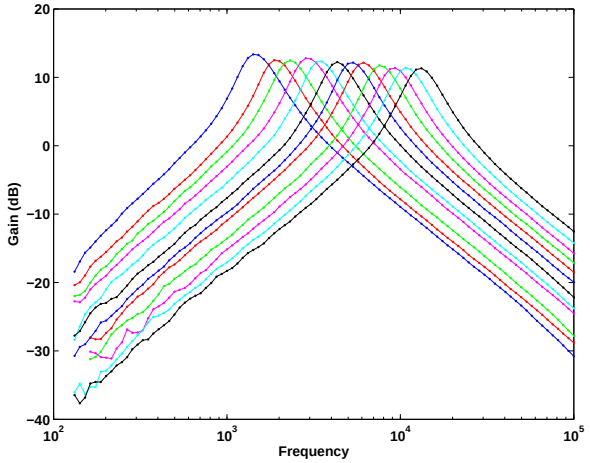
Figure 51. The output of the peak detector module is shown to track the peaks of the sine wave input. The time constant is varied by programming a floating–gate transistor connected as a current source to the bias of the peak detector module. The outputs of the peak detector shown here are for two different time constants.

The C^4 SOS module is comprised of two C^4 modules cascaded with a buffer in–between them. Either C^4 module can be used alone by spreading apart the corner frequencies of the other module. To characterize this module, frequency response plots of each of the individual C^4 modules are shown in Fig. 52a and 52b. The bandwidth and Q-peak of the C^4 modules are quite different. This is due to the difference in output capacitance of each module. The output of the first C^4 is tied to the input of a buffer, which results in a relatively small capacitance. The output of the second C^4, however, is tied into the switch network. Therefore, the output load capacitance for this device will be much higher due to the parasitics of the switches and the capacitance of the next circuit in the path. In these experiments, the next stage was a relatively large buffer in an output pad.

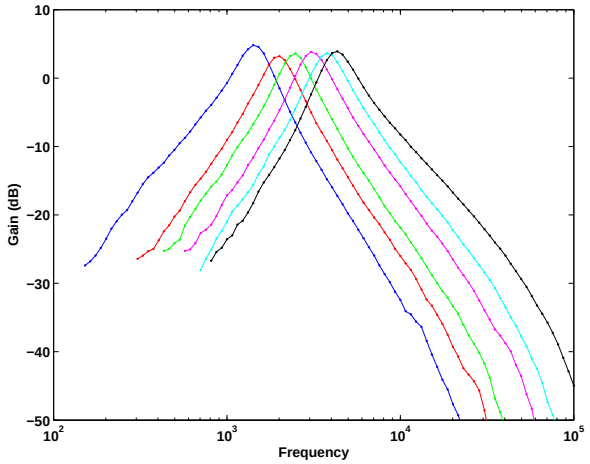
When both of the cascaded C^4s are set to the same corner frequencies, the output



(a)



(b)



(c)

Figure 52. The C^4 second–order section (SOS) block is comprised of two C^4 circuits with a buffer in between. By spreading the corner frequencies of one of the C^4 circuits to be far apart, the frequency response of the other C^4 can be measured. This method is used here to generate the frequency response plots for the first (a) and the second (b) C^4 circuits. Then the frequency response for the SOS is generated by programming both C^4 circuits to the same corner frequencies as shown in (c).

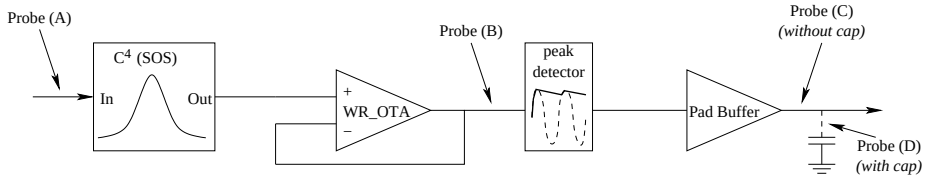


Figure 53. This is the circuit diagram for a typical subband system. The incoming signal is bandpass filtered and then the magnitude of the subband is output from the peak detector. This is analagous to taking a discrete Fourier transform.

of the module shows the desired second–order roll–off as shown in Fig. 52c. In all of these plots, the corner frequencies are shown to be programmable over a wide range of frequencies. The bias current to corner frequency correlation is different for each of the cascaded devices. However, all of the bias currents for these plots were within the range of 25 pA to 200 nA.

The coarse–grain components are most useful when they can be combined to form a larger system. In Fig. 53, a circuit is shown that uses a C^4 SOS block, an OTA, and a peak detector in series. This configuration is very powerful when it is replicated 64, 128, or more times on the FPAA with the center frequencies of the bandpass filters varying over the desired frequency range. The outputs of the different subbands are analagous to the magnitudes of the discrete Fourier transform. As a test of this system, data was taken from RASP 1.5 for a single subband. As shown in Fig. 54, the input is an amplitude–modulated signal with 1.8 KHz and 10.0 KHz frequency components. The C^4 SOS module is biased to have a center frequency near 1.8 KHz, and the OTA is configured to be a noninverting buffer. The output of the system is shown in Fig. 54. Also, the output of the system is shown after an external 2.2 μF capacitor has been added at the output of the FPAA. This change has the effect of smoothing (i.e., low–pass filtering) the output, thus creating a longer effective time constant for the system.

To illustrate the use of multiple subbands, the system shown in Fig. 55a was synthesized on RASP 1.5. Each subband is implemented in a single CAB, and the

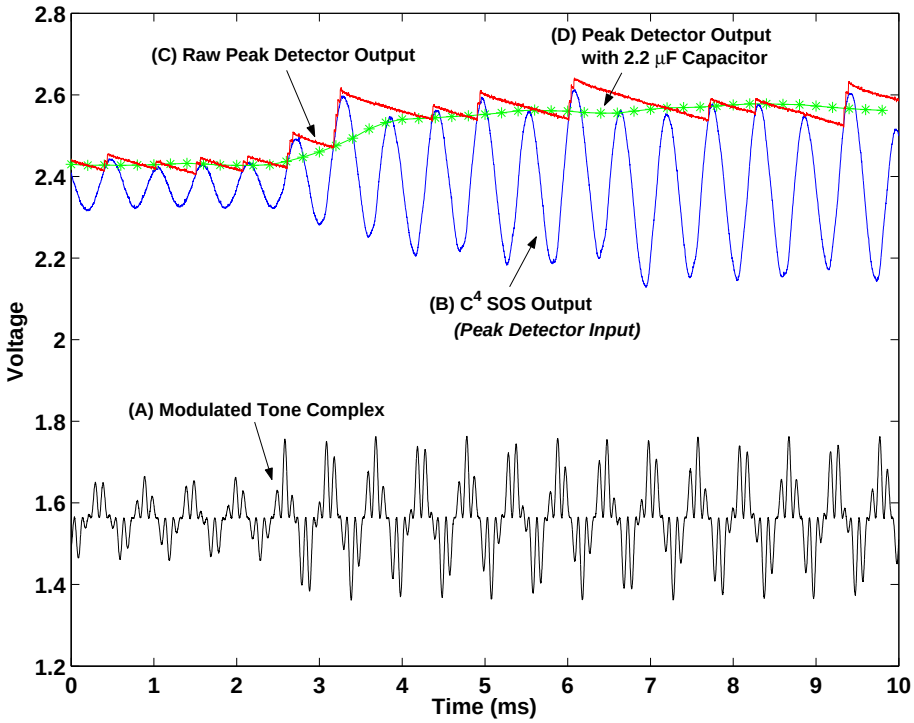
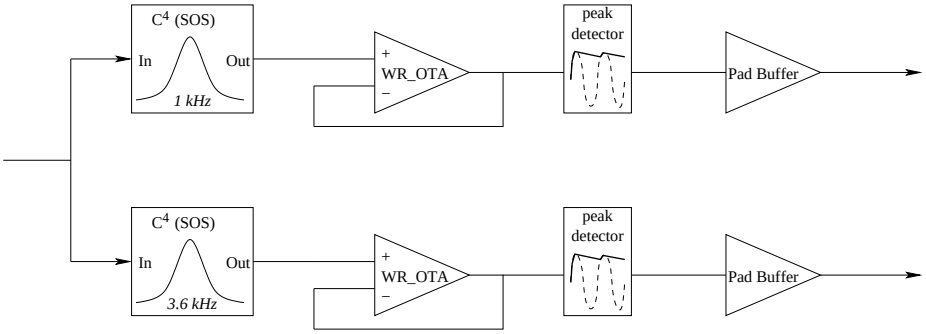
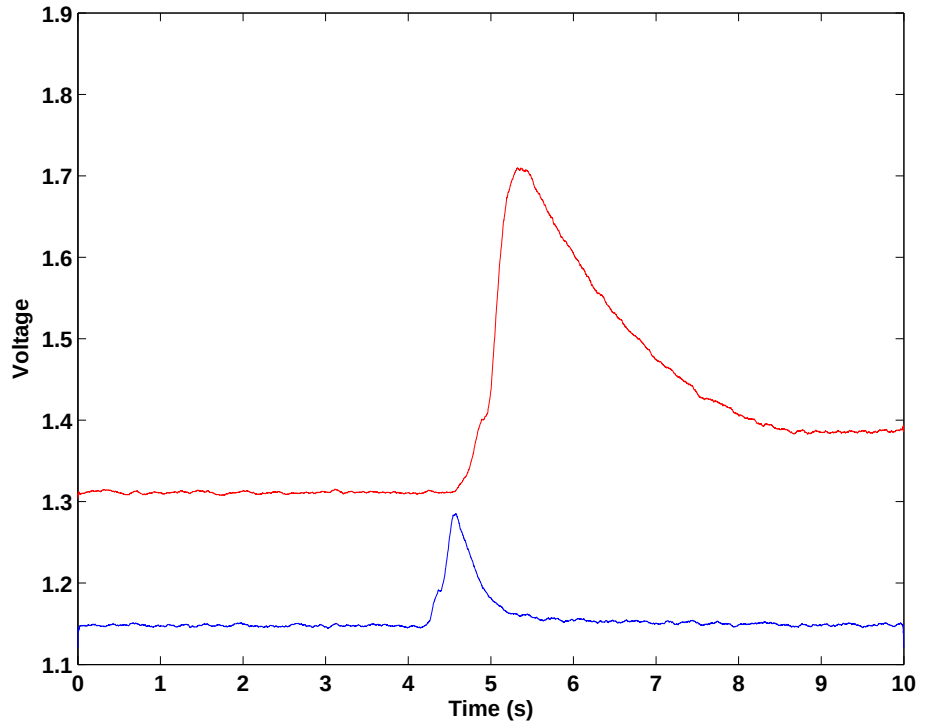


Figure 54. This is the experiment data from the subband system shown in Fig. 53. The input waveform is an amplitude modulated signal with 1.8 KHz and 10.0 KHz components. The output of the peak detector is shown with and without an integrating capacitor added to the output stage.

center frequencies of the C^4 SOS blocks are programmed to different frequencies. The results of this system are shown in Fig. 55b. As the input frequency increases with time, the output from the subband with a center frequency of 1 kHz responds by increasing to a peak at 1 kHz, and then it decreases as the input frequency increases past its center frequency. The output of the second subband responds similarly as the input frequency increases past its center frequency of 3.6 kHz.



(a)



(b)

Figure 55. (a) This block diagram shows a two–subband version of the system in Fig. 53. (b) Experimental data from this system is shown here. As the input frequency increases with time, the outputs from the two subbands are shown to peak as the input frequency reaches their center frequency.

CHAPTER 6

MIXED–SIGNAL PROTOTYPING PLATFORM

The FPAAs introduced here are designed for use within mixed–signal systems. When coupled with FPGAs, they can provide a complete mixed–signal rapid prototyping system. In addition, the same infrastructure that is developed to connect FPGAs and FPAAs during run–time mode can be used during programming mode to configure the FPAA.

6.1 Development System

The current development system was originally built for the RASP 1.0 FPAA. However, it has been easily extended to work with more recent FPAAs (and other custom floating–gate chips as well). The core of the platform consists of an FPGA development board and a daughter card for interfacing to the FPAA (Fig. 56). For simplicity, an off–the–shelf FPGA board is being used here along with a custom daughter card developed for the FPAA; however, future versions of this system could easily have the functionality of both boards integrated into a single mixed–signal reconfigurable prototyping board. Work is also progressing on moving much of the programming functionality shown here onto the FPAA chip [74]. This effort will further simplify the board–level design of mixed–signal prototyping platforms.

A block diagram of the overall development system is shown in Fig. 57. A program running on the PC controls the programming at the highest level. The FPGA receives the high–level commands from the PC and implements the specific protocols to carry out the comamand. The FPAA daughter card consists of level shifters and D/A converters to interface from the FPGA to the FPAA, A/D converters for interfacing from the FPAA to the FPGA, and the appropriate power supplies for running and programming the FPAA.

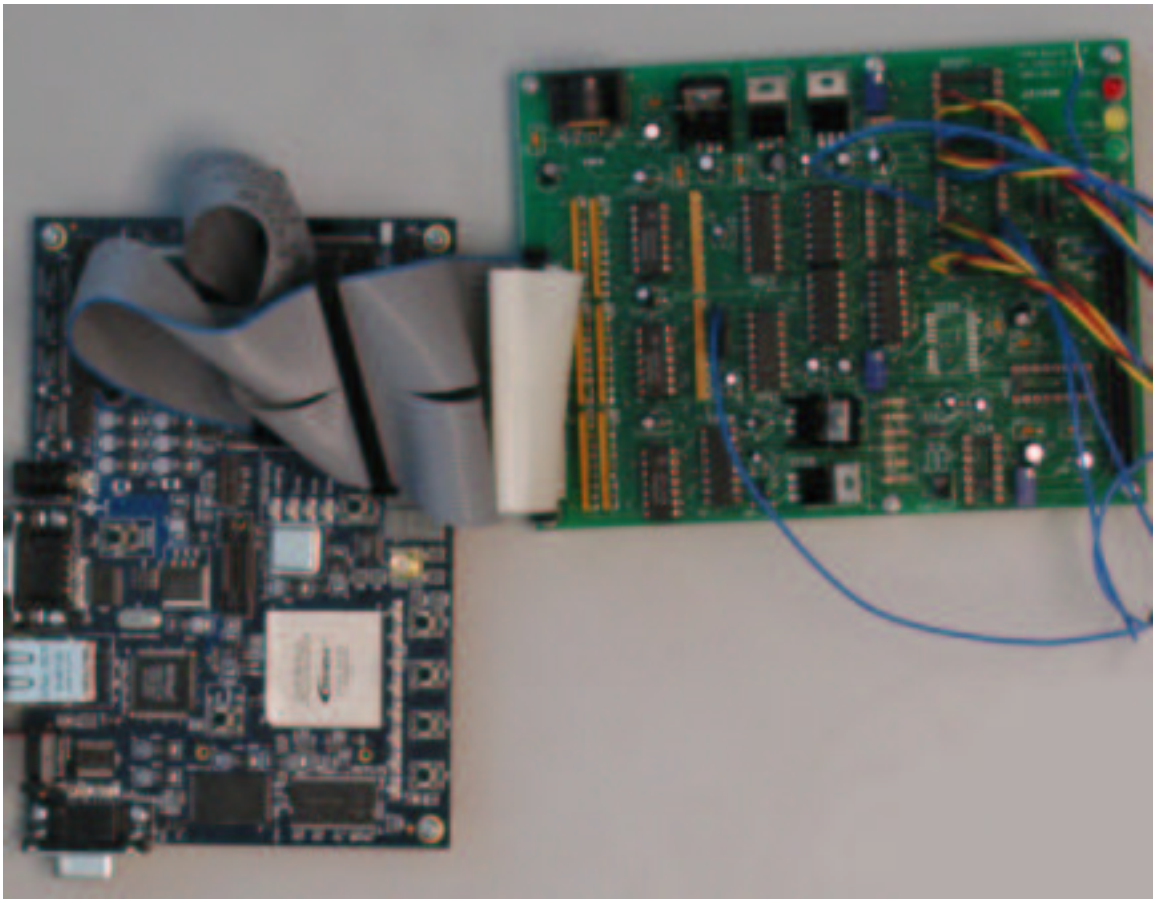


Figure 56. This is a picture of the mixed–signal prototyping platform. It is currently comprised of an off–the–shelf FPGA development board and a custom daughter card developed for the FPAA.

6.1.1 FPGA Development Board

The FPGA board used is the Nios Development Kit–Stratix Edition (Nios–Stratix) provided by the Altera Corporation. In addition to the FPGA IC, the Nios–Stratix board contains a rich array of prototyping devices, including 1 MB of SRAM, 8 MB of Flash memory, 16 MB of SDRAM, a CompactFlash connector, an Ethernet connector, two serial ports, and two 41–pin general I/O headers for attaching daughter boards. The FPGA on this board is a Stratix EP1S10, which contains approximately 300,000 general–purpose logic gates, over 920 Kbits of on–chip RAM, and a number of specialized digital signal processing blocks. This FPGA is large enough to synthesize a customized soft–core processor (Altera's 32–bit Nios processor), and when needed,

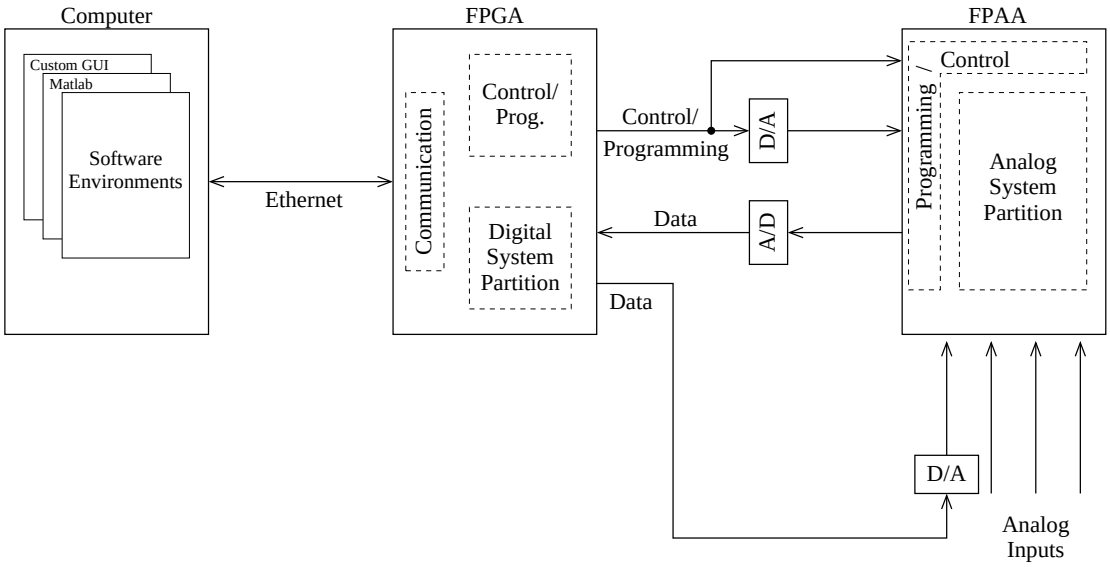


Figure 57. This is a block diagram of the overall mixed–signal development system. The FPGA receives the high–level commands from the PC and implements the specific programming protocols to perform the specified operation to the FPAA.

specialized hardware modules that handle timing–critical communication between the FPAA daughter card and the soft–core processor. The Nios processor controls the overall programming system and handles the Ethernet communications with the PC.

6.1.2 FPAA Daughter Card

The FPAA daughter card is a custom printed circuit board that has been designed to interface directly with the I/O headers on the Nios–Stratix FPGA board. The daughter card has three primary functions:

1. Provide the physical interface between the FPAA and the FPGA
2. Provide supply voltages for the FPAA in both programming and operational modes
3. Provide current and voltage feedback data to the FPGA for controlling the programming algorithms and analyzing systems implemented on the FPAA

The interface to the FPGA is complicated by the requirement that the FPAA's supply voltage must be able to vary during programming. Thus, the digital control

signals coming from the FPGA must be level shifted to be compatible with the supply voltage of the FPAA at any given time during the programming process. Open–collector inverting buffers are used for this purpose; the outputs of the inverters are pulled up to the FPAA's power supply with resistor networks.

The FPAAs discussed in this thesis are designed to operate at very low currents. Bringing these currents off of the FPAA while maintaining a reasonable SNR is very difficult. In addition to using standard board design techniques, great care must be taken when dealing with small currents. For example, current outputs can not be routed through an I/O header block, because the pin–to–pin resistance for standard headers is in the tens of megaohms range. This can lead to nanoampere or even low microampere leakage currents between the pins, which will drastically affect measurements when the desired currents are at or below this range. Ideally, current–to–voltage converters will be integrated onto future FPAA ICs and currents will not be dealt with at the board level. However, on legacy chips, pins must be raised off of the board and connected directly to picoammeters to achieve reliable measurements.

6.2 Programming Software

There are three different levels of software currently used to program the RASP FPAAs. At the most abstracted level, a graphical schematic–entry environment is used to specify the desired circuit. Next, a Matlab script is used to initiate programming of the individual switches and biasing transistors needed to implement the circuit. Finally, firmware running on the the soft–core processor (synthesized on the FPGA) receives the commands from the PC and executes the programming protocols to set the individual floating–gate transistors at the desired current levels.

6.2.1 Graphical Circuit Specification

A custom Visual Basic program has been written to enable graphical specification of the desired circuit. Design entry is completed by manually selecting the rows and

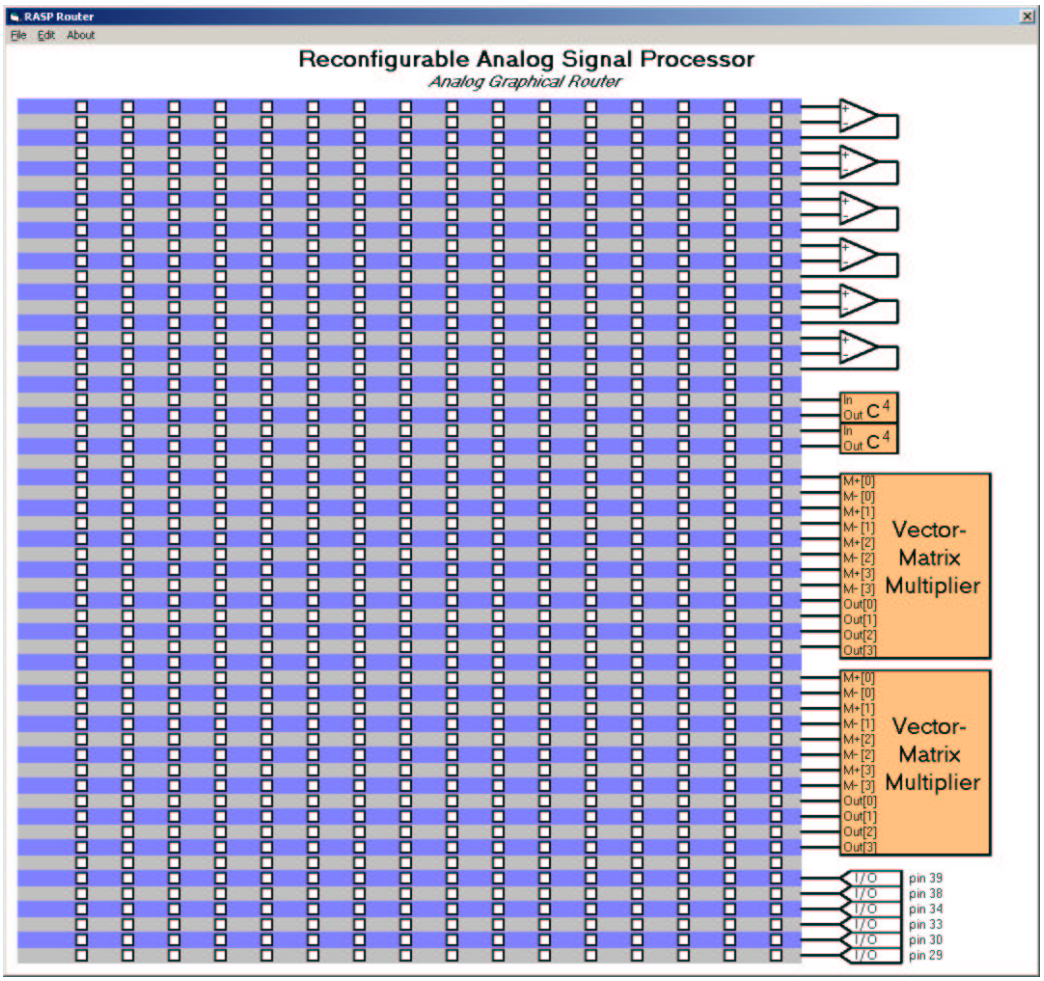


Figure 58. This is a screen capture of the graphical FPAA configuration software. This program allows the user to enter a circuit diagram graphically and export a text file with netlist for the entered circuit. This version of the software is customized for the RASP 1.0 FPAA. A circuit is entered by clicking on the switch symbols (squares) to specify each *on* switch.

columns for each signal path in the desired circuit. As shown in Fig. 58, RASP 1.0 is small enough to allow the entire switch network and CAB components to be displayed on a single screen. This greatly simplifies the program code and design entry. Larger FPAA designs will require that more complexity be added to this software.

To enter a design, the user clicks on the individual switches (illustrated by square checkboxes) to turn them *on* or *off*. When a switch is turned *on*, an implicit connection is made between the row and column of the switch. Multiple switches can be turned *on* for any given row or column. Selecting *Preview* from the *File* menu will hide the *off* switches and explicitly show the column and row connections. As an example, the second–order section (SOS) circuit introduced in Fig. 47 was entered into this program. The Preview–mode view of this circuit is shown in Fig. 59.

Bias currents can be specified for each of the CAB components by clicking on the respective component in the program window. A unique dialog box will appear for each CAB component that allows the user to specify the appropriate biases for that component. These biases are not used or displayed elsewhere within this program, but they are exported in the output netlist file.

Once a circuit design is entered into the graphical configuration program, a circuit netlist is exported. This netlist contains the row and column indices for each switch that needs to be turned *on* and pertinent information for each of the CAB component bias transistors being used. The netlist file for the SOS circuit illustrated in the previous figure is shown in Fig. 60. As can be seen in this figure, the switches are listed without voltage and current information, since switch programming is uniformly implemented in the programming code. The bias transistors, however, must have voltage and current values specified. In this case, there are three biases (for the three OTAs in the circuit) that must be set. The current values for these biases are all specified at a gate voltage of 0.3 V, since that is the operating gate voltage of the floating–gate transistors during run mode.

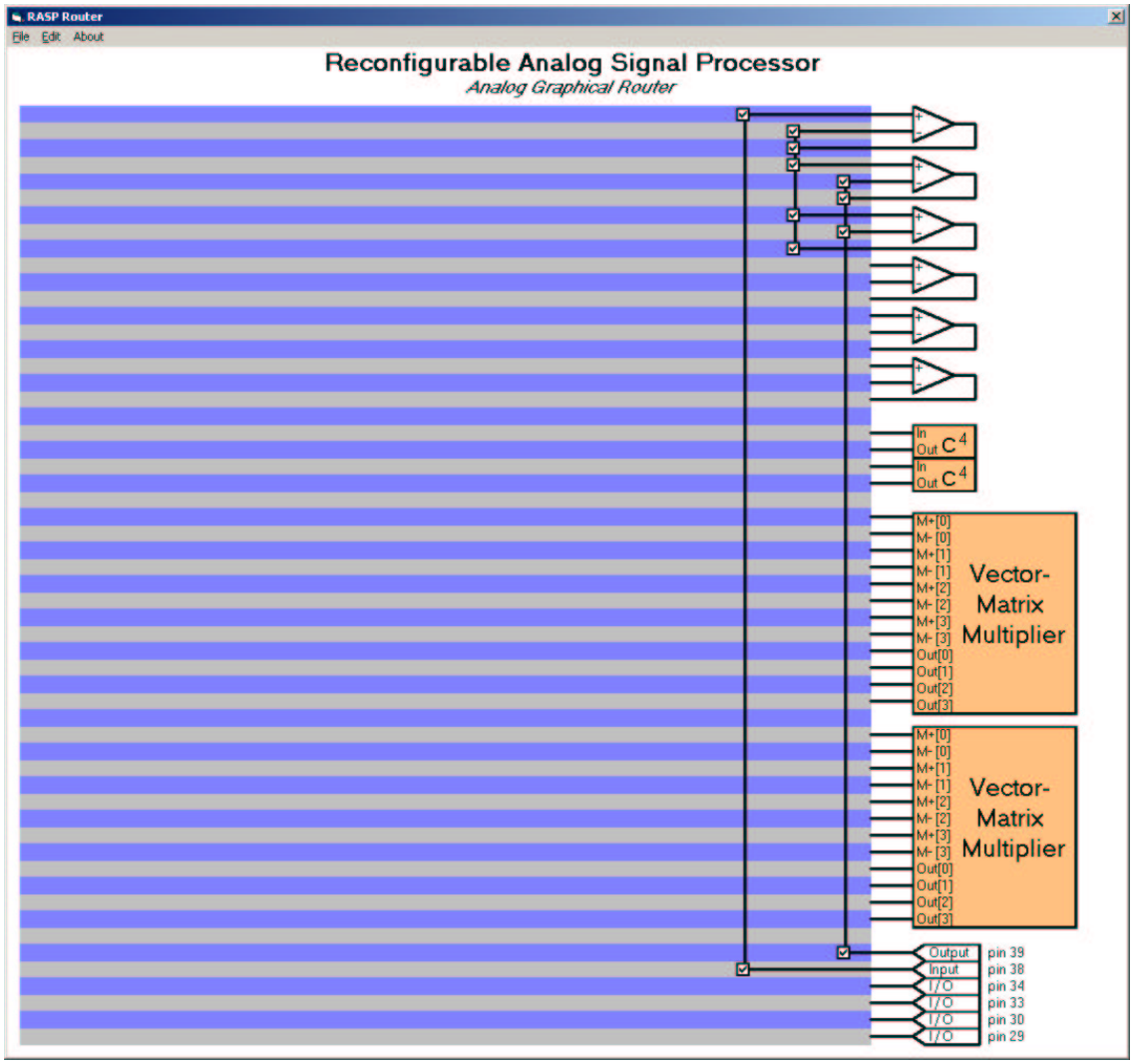


Figure 59. In Preview mode, the software connects the switches that are turned *on* and turns the rest of the switches *off*. This view allows the user to more easily verify the entered circuit design. Here, the second–order section (SOS) shown in Fig. 47 has been entered.

```
Switch?, Row, Column, Voltage, Current
1,  12,  13, 0, 0
1,  15,  14, 0, 0
1,  18,  14, 0, 0
1,  13,  14, 0, 0
1,  16,  15, 0, 0
1,  19,  15, 0, 0
1,  14,  14, 0, 0
1,  17,  15, 0, 0
1,  20,  14, 0, 0
1,  32,  15, 0, 0
1,  33,  13, 0, 0
0,  8,  16, 0.3, 300e-9
0,  9,  16, 0.3, 300e-9
0,  10,  16, 0.3, 340e-9
```

Figure 60. The graphical FPAA configuration software outputs a netlist that includes the complete circuit specification. This sample netlist file is the output for the second–order section circuit as entered in Fig. 59. The row and column numbers used here are for the RASP 1.0 FPAA. This netlist can be directly converted into a Matlab script that will configure the FPAA to synthesize the specified circuit.

6.2.2 High–level Matlab Code

Matlab is correctly used to implement the high–level command and control of the programming circuitry. During development, the details of the programming algorithms were also implemented in Matlab. However, as settings were solidified, most of the low–level details of programming were moved to the firmware executing on the FPGA. At this point, Matlab is used primarily for its ability to easily script the programming commands for a given circuit. In addition, Matlab is used during run–time to retrieve data from the FPGA and computer–controlled measurement equipment. It is then used to view and analyze the data.

6.2.3 Low–level C Code

The FPGA has a customized Nios soft–core processor synthesized on it. The firmware running on this processor is written in C and controls the low–level details of programming. The Nios processor has SPI peripherals implemented on the FPGA that communicate with the D/A converters on the FPAA daughter card to control the

voltages needed during programming. The Nios processor also controls the external picoammeter via a serial port peripheral. Implementing the programming algorithms directly in firmware speeds up the programming process by avoiding the overhead incurred in computer–to–FPGA communications.

CHAPTER 7

FPAAS: A ROADMAP TO THE FUTURE

Large–scale FPAAs are feasible. While previous FPAAs have suffered from their small size and lack of functionality/generality [4, 49, 54, 59], next–generation FPAAs based on the architecture introduced in this thesis can overcome these challenges, thereby extending the usefulness and acceptance of FPAAs. In addition, large–scale FPAA designs with computational logic included at multiple levels of granularity will address the complex design space that analog designs entail (including a wide–range of linear and non–linear functions) while keeping switch parasitics minimized.

The work outlined in this thesis is only the beginning of modern FPAA research. As a direct result of the research explored in this thesis, a number of different research avenues have emerged. Many of these directions are already being pursued by doctoral students within the CADSP group, and others will be explored as time and resources become available. In addition to future research, a start–up company coming out of the CADSP group is working to commercialize this FPAA technology. Impetus on the development side of FPAAs in cooperation with the research activities in the CADSP group will provide an ideal environment for advancing FPAAs from a laboratory testbed to a robust, off–the–shelf solution.

7.1 Next–generation RASP

The next–generation RASP FPAA architecture has been developed as a part of this thesis. Chips based on this architecture have been designed with as many as 72 CABs on them. Several designs are currently being fabricated. One of those FPAAs, dubbed RASP 2.5, is described in more detail in this section. A top–level block diagram for RASP 2.5 is shown in Fig. 61.

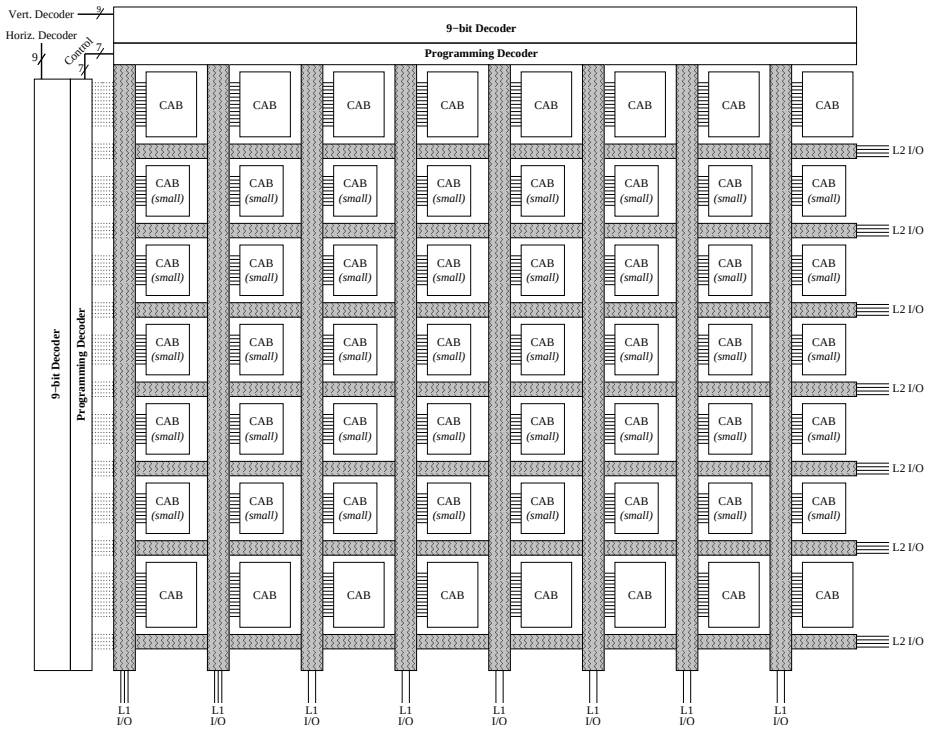


Figure 61. This is the top–level block diagram for the RASP 2.5 FPAA. It has a non-homogeneous mesh of CABs with 16 full CABs and 40 small CABs for a total of 56 CABs.

7.1.1 Switch Networks

As shown in Fig. 62a, the routing architecture of our large–scale FPAAs is a combination of global and local switch networks. Each CAB has an associated local switch network for making connections within a single CAB. The switches' source lines are routed along the rows and connect the inputs and outputs of each CAB to the switch network. The drain lines of the switches are connected along the columns. By turning a switch *on*, a single row (source) can be connected through the switch to a single column (drain).

The size of the switch network is dependent upon the number of I/O lines in each CAB. For the design shown in Fig. 62, there are two types of CABs. For the large CABs, the local switch networks are comprised of a 10 x 42 matrix of switches; in the small CABs, the local switch network is a 10 x 32 matrix. Each local switch network is integrated into a matching global routing switch network. The global routing switch

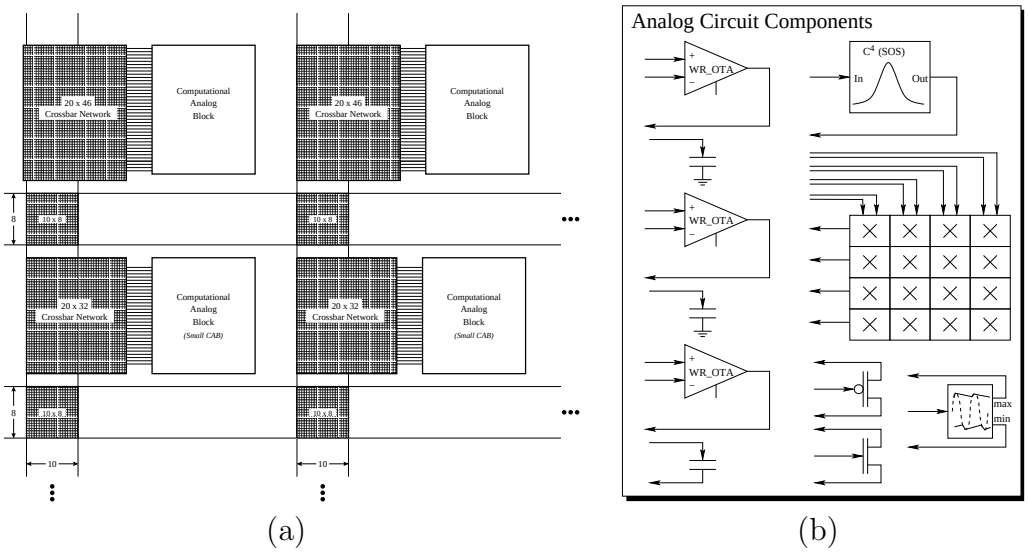


Figure 62. (a) This block diagram shows the routing architecture for RASP 2.5. The switching interconnects are fully connectable crossbar networks built using floating–gate transistors. (b) This FPAA has two slightly different CABs. The large or regular CAB contains a 4 x 4 matrix multiplier, three wide–range OTAs, three fixed–value capacitors, a C^4 SOS, a peak detector, and two FET transistors. The small CAB is the same except it does not include the 4 x 4 matrix multiplier. This design includes large CABs at the top and bottom of each column and small CABs in between.

network allows local signals from a CAB to be connected to the global routing busses and be routed off the chip or to another CAB. There are also 10 x 8 switch networks at each junction of the horizontal and vertical global routing busses.

7.1.2 Computational Analog Blocks

The computational logic in RASP 2.5 is organized in a compact CAB providing a naturally scalable architecture. CABs are tiled across the chip in a regular mesh–type architecture with busses and local interconnects in between as shown in Fig. 62a.

Many potential CABs can be imagined using this technology. Figure 62b shows one example CAB, whose functionality is enhanced by a mixture of fine–grained, medium–grained, and coarse–grained computational blocks similar to many modern FPGA designs. The computational blocks were carefully selected to provide a sufficiently flexible, generic architecture while optimizing certain frequently used signal processing blocks. For generality, three OTAs are included in each CAB. OTAs have

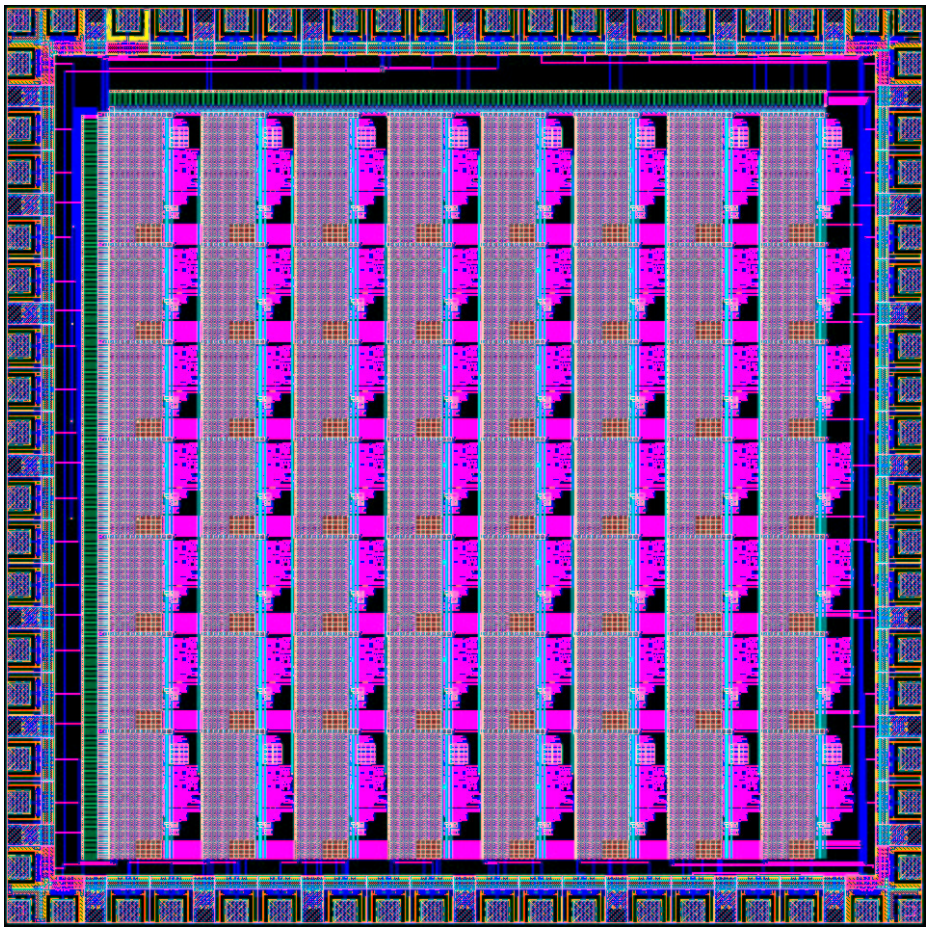


Figure 63. This is the top–level layout for the RASP 2.5. It is comprised of 16 large CABs (on the top and bottom rows) and 40 small CABs for a total of 56 CABs. In TSMC 0.35-micron process, it covers an area of approximately 9 mm^2.

already been shown to be effective at implementing a large class of systems including amplification, integration, filtering, multiplication, exponentiation, modulation, and other linear and non-linear functions [26, 65, 70, 71]. In addition, the two FET devices provide the ability to perform logarithmic and exponential functions, as well as convert from current to voltage and vice versa. The three capacitors are fixed in value to minimize the size of the CAB and are primarily used on the outputs of the OTAs; however, they will be available for any purpose. The variable capacitor and/or current mirror banks found in some designs are not needed here, because the use of

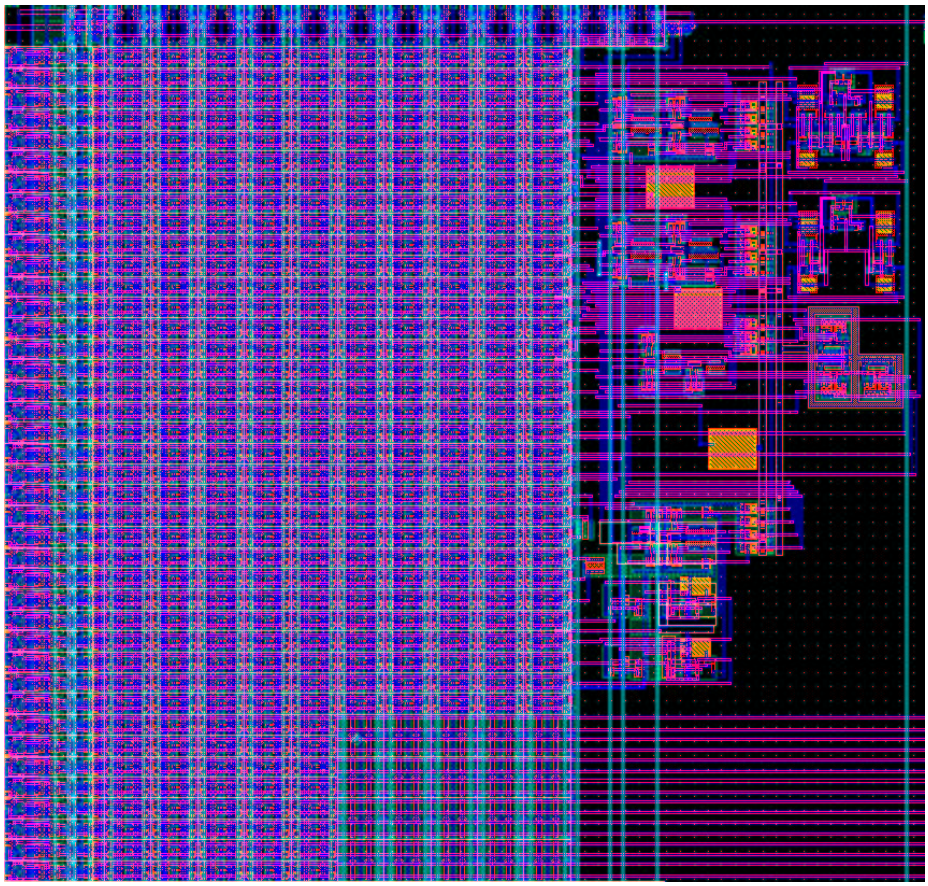


Figure 64. This picture is a more detailed look at the small CAB and associated local switch network.

floating–gate transistors in the OTAs will give the user sufficient control in programming the transconductance of the amplifiers [37, 65]. Eliminating the capacitor banks creates a large savings in the area required for each CAB.

The high–level computational blocks used in this design are an SOS bandpass filter module comprised of two C^4 and the 4 x 4 vector–matrix multiplier block. In general, the C^4 SOS module provides a straightforward method of subbanding an incoming signal. This is analagous to performing a Fourier transform. The vector–matrix multiplier block allows the user to perform a matrix transformation on the incoming signals. Together these blocks can be used like a Fourier processor [44, 52]. In addition, a peak detector is added to each CAB.

The architecture illustrated in Fig. 62a is non-homogeneous in that there are two

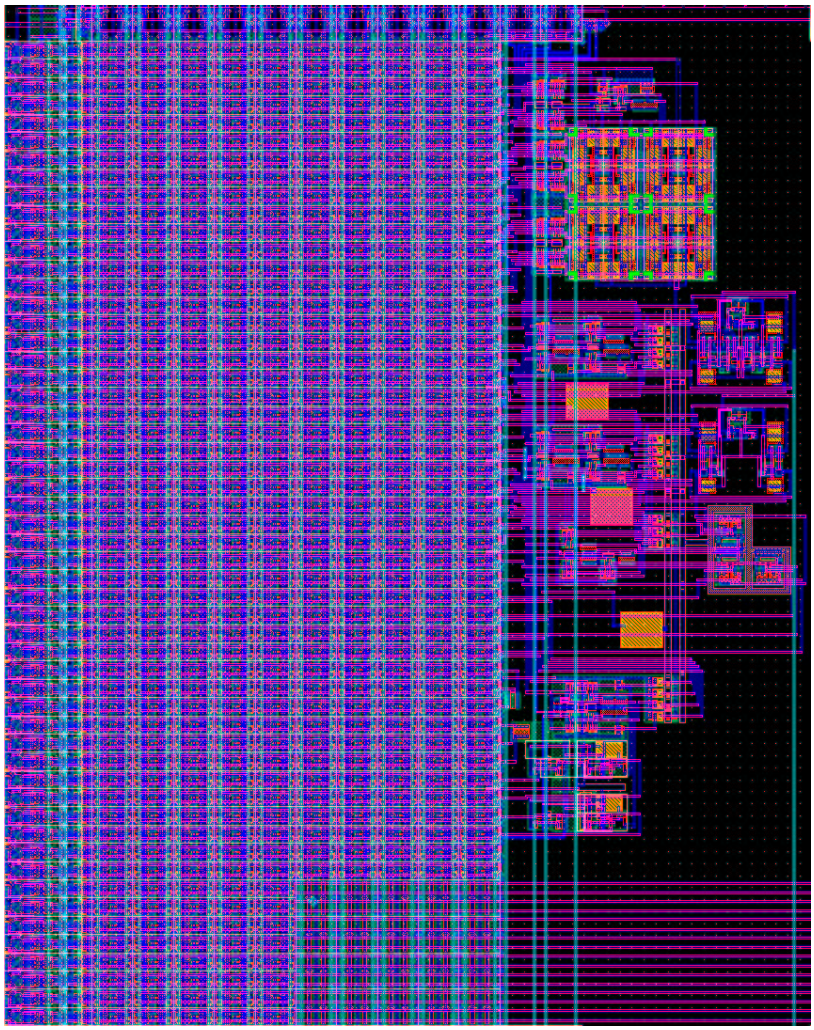


Figure 65. This picture is a more detailed look at the full CAB and associated local switch network.

different CABs tiled across the chip. The small CAB is identical to the large CAB except it does not include the vector–matrix multiplier module. Since the vector–matrix multiplier takes four inputs and each input will often be derived in a separate CAB (from a separate subband created by the C^4 SOS module), designs will typically utilize only one vector–matrix multiplier for every four CABs. Thus, FPAAs that have 50-100 CABs can be made more compact by removing the vector–matrix multiplier from all the CABs except those on the top and bottom rows (assuming the FPAA is more or less square in design).

A sample FPAA based on this architecture with 56 CABs (16 large CABs and 40

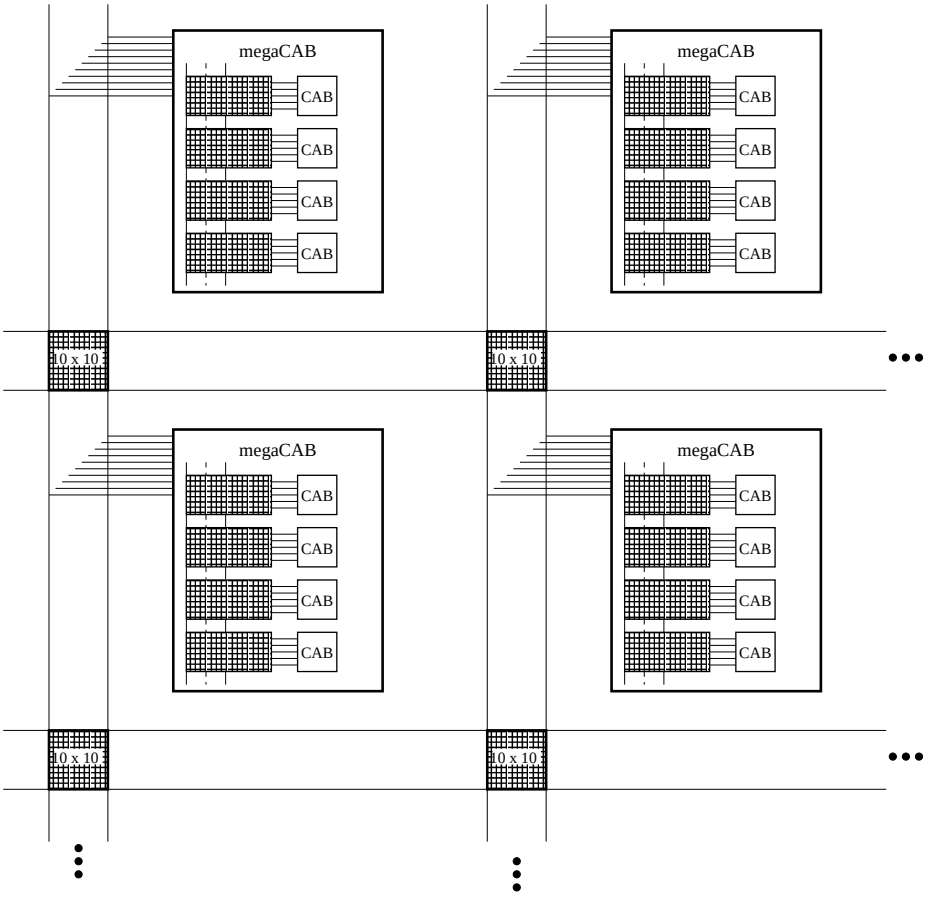


Figure 66. This shows an example of future FPAAs with hundreds of CABs. As FPAAs scale up in size, more tiers of routing must be added to the heirarchy. Here, global vertical and horizontal busses (tier 1) connect clusters of CABs, dubbed megaCABs, together.

small CABs) on a single chip is being fabricated in TSMC 0.35-micron; it covers an area of approximately 9 mm^2. A picture of the top–level layout is shown in Fig. 63, and more detailed pictures of the small and large CABs are shown in Fig. 64.

7.2 Future FPAAs

Commercially viable FPAAs are forseen that have hundreds, if not thousands, of CABs based on this same architecture. These designs use a heirarchical routing structure with more levels than has been shown. An example of a 512 CAB FPAA is shown in Fig. 66. The CABs are clustered in groups of four with a routing scheme

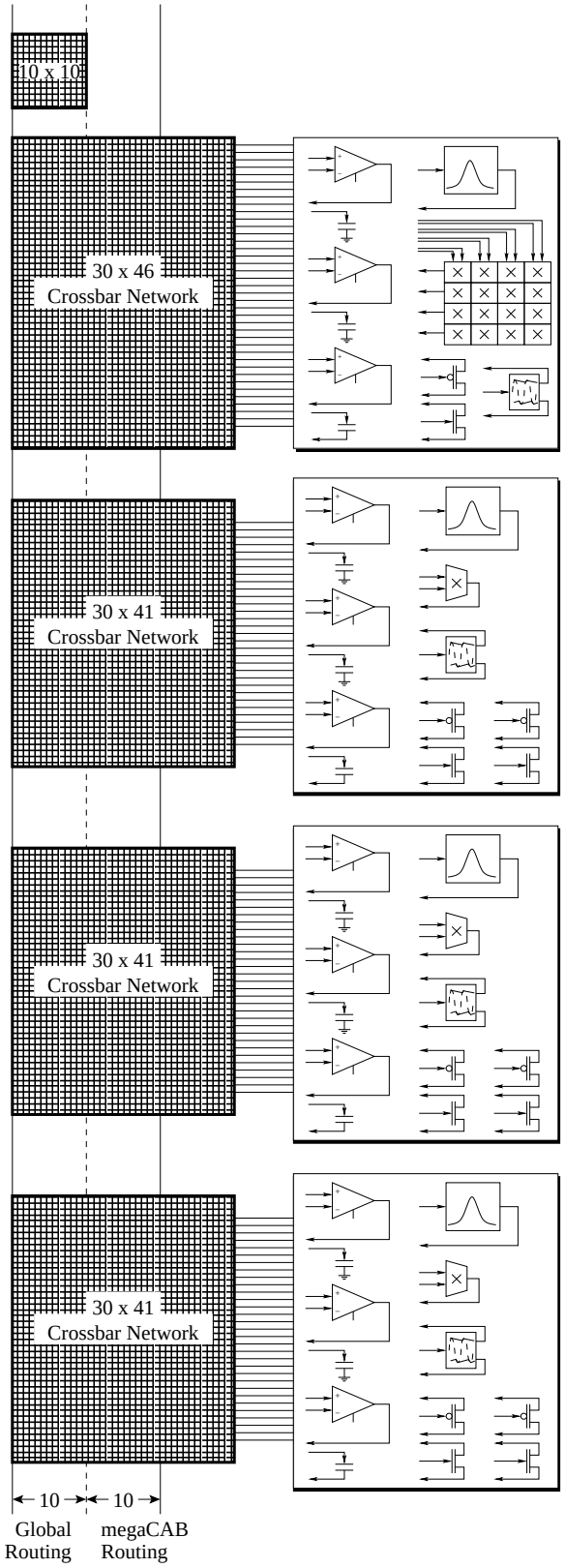


Figure 67. In future FPAAs with hundreds of CABs, CABs are clustered together in megaCABs that contain four to eight CABs. megaCAB–level routing (tier 2 routing) connects the local switch networks (tier 3 routing) together.

similar to RASP 2.5. These CAB clusters have been dubbed megaCABs and an example is shown in Fig. 67. Within each megaCAB, CABs can be varied slightly to provide more flexibility. In this example, only one of every four CABs contains a vector–matrix multiplier. The other CABs then have additional transistors and a signal–by–signal multiplier.

An FPAA with this architecture can synthesize 512–channel subbanding, a 44–point discrete cosine transform, 256 third–order ladder filters, or many other circuit combinations. Larger systems can be built as well. As is shown in Section 7.3, this FPAA can implement a 32–channel auditory feature extraction system, a 128–channel cochlear model, or a low–power hearing aid algorithm. These examples are just a sample of the full systems that can be achieved on large–scale FPAAs. In addition, FPAAs based on this architecture can be effectively used to prototype analog front–ends for digital systems, broadband processing for high–speed communication systems (allowing low–speed, baseband A/D converters), real–time image processing, low–power distributed sensor networks, and data acquisition systems.

7.3 Testbed Applications for Large–Scale FPAAs

To illustrate the viability of large–scale FPAAs, several distinct applications will be discussed. Their functionality will be shown along with their implementation on large–scale FPAAs.

7.3.1 Auditory Feature Extraction

Extraction of features from an incoming audio stream is a key component in audio classification and auditory scene analysis systems. The algorithm illustrated in Fig. 68 has been developed previously and digital implementations exist. However, for embedded processing environments, this algorithm is computationally intensive. By implementing most or all of this algorithm in analog hardware, a big savings in time and power can be achieved.

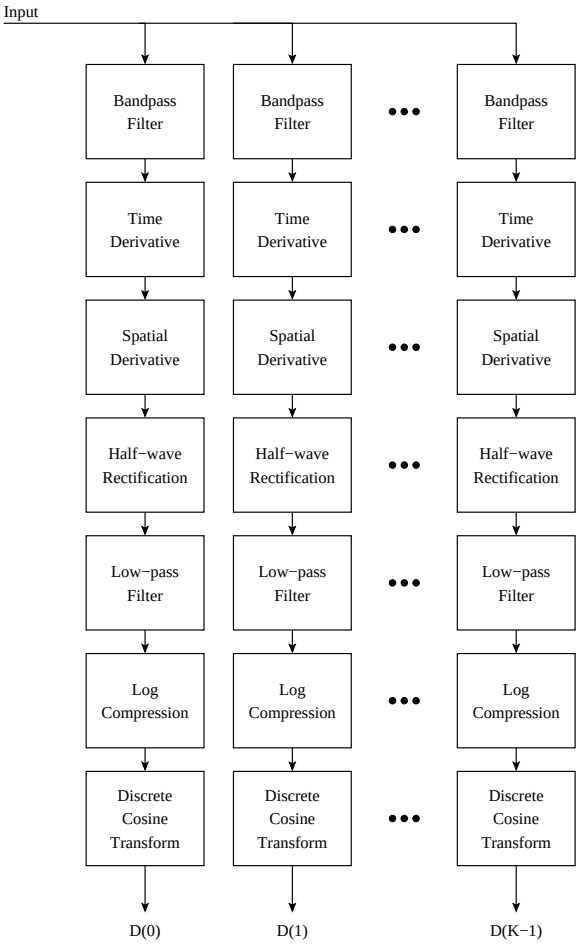


Figure 68. This is the block diagram for a digital feature extraction algorithm. Typically, 32–128 channels are needed for this algorithm.

$$\left[\begin{array}{cccc} I_0 & I_1 & I_2 & I_3 \end{array}\right] = \left[\begin{array}{cccc} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{cccc} (V_0) & (V_1) & (V_2) & (V_3) \end{array}\right]$$

Figure 69. The vector–matrix multiplication module is used to compute the first–order difference equation for three of the channels.

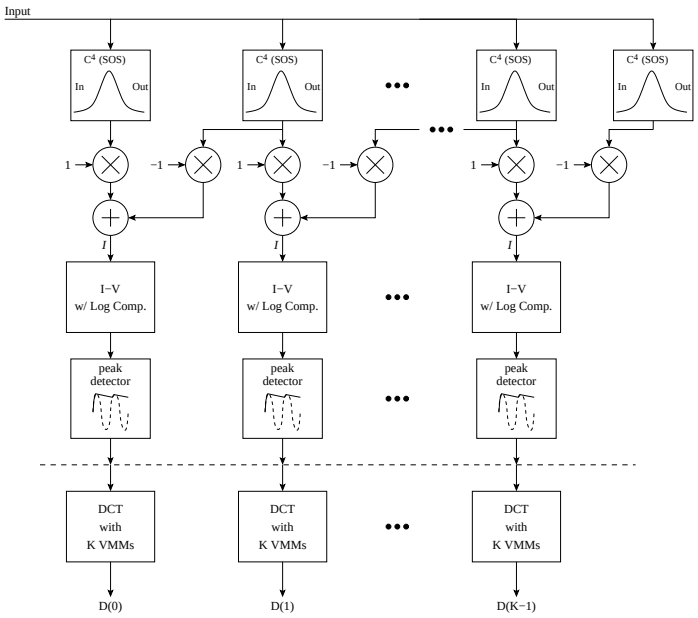


Figure 70. This is the block diagram for the analog feature extraction algorithm as implemented on an FPAA. Without the DCT block, a 32–channel system can be synthesized on RASP 2.5. For a 32–channel system, the DCT blocks will require 64 4 x 4 vector–matrix multiplier blocks. The future FPAA outlined in Section 7.2 has 128 of these blocks and could implement the entire feature extraction algorithm shown here.

The analog blocks required to implement this function are shown in Fig. 70. As shown in this figure, the bandpass filter module and time derivative can be combined into a C^4 SOS bandpass filter module. The spatial derivative can be computed with a first–order difference of the neighboring subband using the vector–matrix multiplier to compute three subbands (see Fig. 69). Finally, the half–wave rectification and temporal averaging are achieved with a peak detector programmed to an appropriate time constant.

The RASP 2.5 FPAA is large enough to implement a 32–channel version of this feature extraction algorithm with the exception of the discrete cosine transform (DCT). In this case, the analog/digital boundary would fall between the peak detector and the DCT as shown by the dashed line in Fig. 70.

To implement the entire algorithm in analog, the future FPAA outlined in Section 7.2 would need to be used. With 128 4 x 4 vector–matrix multiplier blocks,

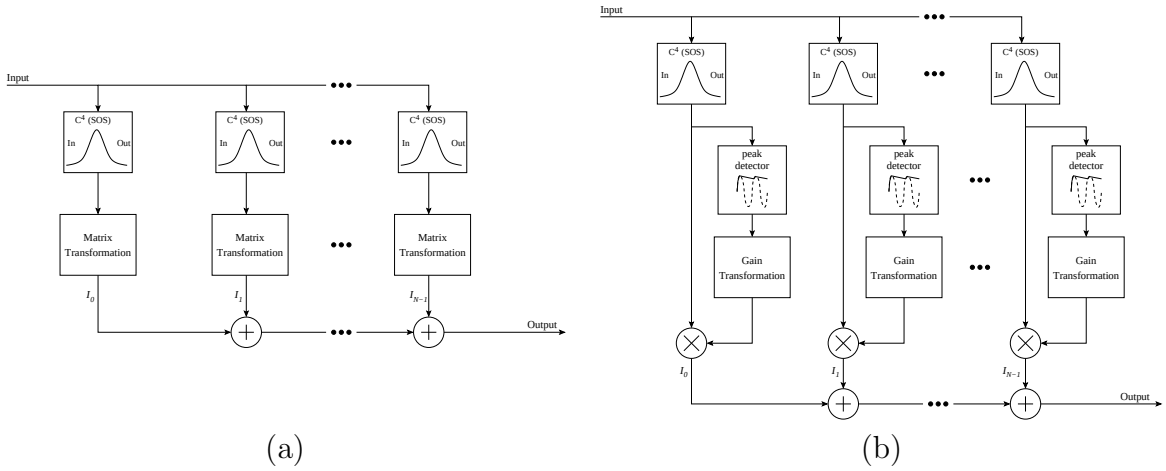


Figure 71. These generalized block diagrams typify many audio processing functions, including noise suppression, audio classification, and hearing aids. The various RASP architectures have been optimized to implement one or both of these system flows.

a 32 x 32 matrix transformation implementing the DCT can be implemented along with the first–order difference equations needed for each channel.

In addition to the feature extraction algorithm outlined above, a number of different audio and speech processing functions can be realized on large–scale FPAAs with architectures similar to the RASP FPAAs. The generalized block diagrams shown in Fig. 71 are typical of many audio processing functions, including noise suppression, audio classification, and hearing aids. The various RASP architectures have been optimized to implement one or both of these flows.

7.3.2 Neuromorphic Modeling

It is natural to use analog circuits to model neural systems [62]. Models of the basic neural processes—such as axons, synapses, dendrites, action potentials, and whole neurons—are often not very large when done in isolation. However, modeling full neuromorphic systems requires hundreds or thousands of these basic blocks. The size and functionality of the large–scale FPAAs introduced here will enable the rapid prototyping of neuromorphic systems, such as networks of neurons, the cochlea, and more.

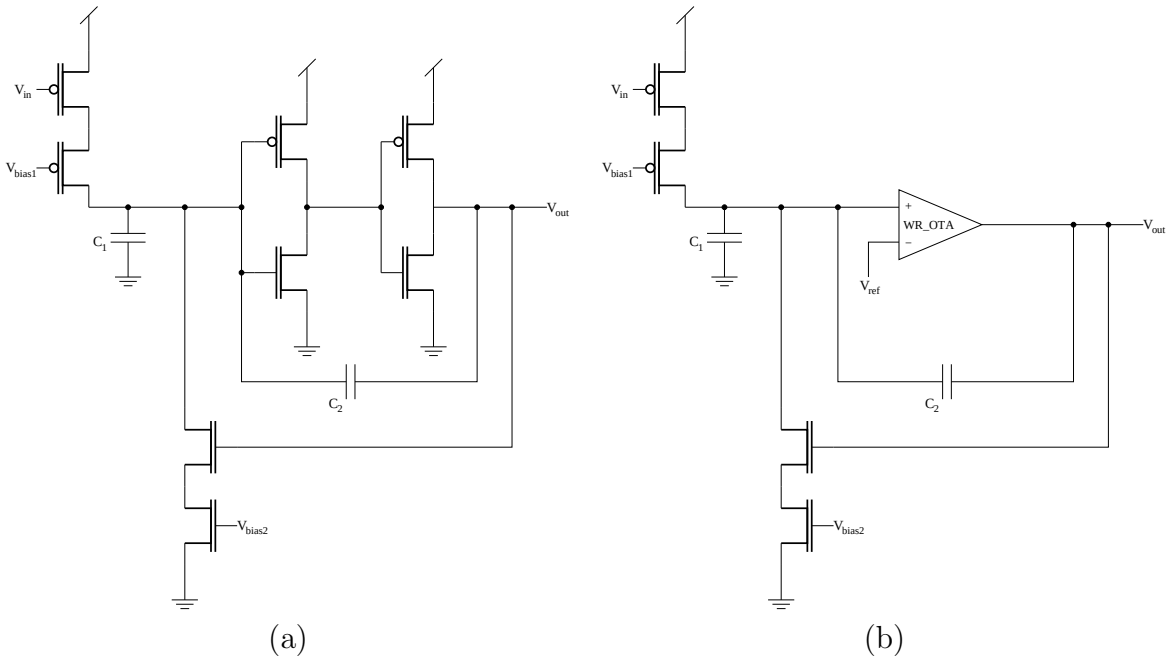


Figure 72. These circuit diagrams illustrate two slight variations of the self–resetting (or integrated–and–fire) neuron from [62]. Both circuits can be implemented in RASP 2.5; however, the circuit in part (a) requires one more CAB than the circuit in part (b) because of the six transistors needed.

One basic model of the neuron is Mead's self–resetting neuron [62]. Two slight variations of this design are shown in Fig. 72. Both of these circuits can be implemented in RASP 2.5; however, the transistor–only version requires an additional CAB because of the six transistors needed.

Like the variety of neuron models, a number of different models exist for building larger neuromorphic systems. As an example, two models of the cochlea are shown in Fig. 73. Both of these systems can be synthesized with RASP FPAAs. Typically, these models will be used with 32 channels. The center frequencies of the C^4 SOS modules are spaced logarithmetically through the human hearing range.

Modeling more complex neuromorphic systems often requires adaptability. It has been shown previously that floating–gate transistors are well–suited to implementing adaptive systems [20, 46, 51]. An adaptive CAB based on this technology could be built that would readily fit within the RASP architecture. This ability to combine

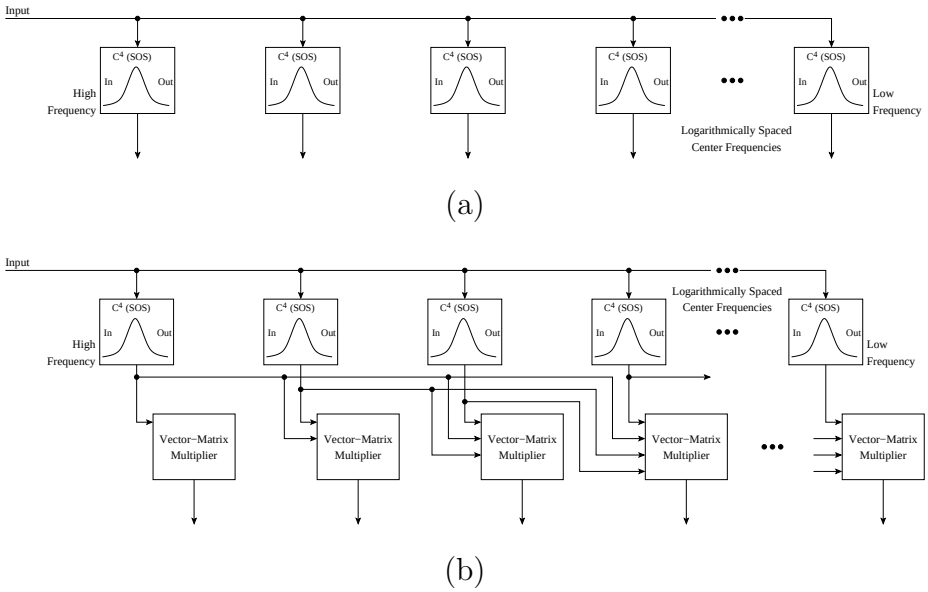


Figure 73. These circuit diagrams illustrate two models of the cochlea. Both of these systems can be synthesized on the RASP FPAAs. Typically, 32 channels would be used with the center frequencies of the variations spaced logarithmetically throughout the auditory range.

general–purpose computational elements with blocks that can enable the synthesize of specific signal processing functions is an important feature of the FPAA architecture and framework presented in this thesis.

7.4 Future Research

More than just a single device, the FPAA enables a whole new design paradigm. The FPAA architecture and infrastructure developed here form a design methodology that can be applied to an ever–widening scope of problems. The FPAA framework introduces programmability, reconfigurability, and reuseability to analog circuit design. While FPAAs have existed before this thesis, they have not had the size or functionality to enable true system–level design, and thus, they have failed to be the enabling technology that takes rapid prototyping of analog systems into the mainstream.

Approaching FPAA design from the perspective of design methodology—instead of merely circuit design—has led to a number of different avenues of research. These research directions are a direct result of this FPAA research and can be broadly

classified into three areas: enabling technologies, computational logic, and computer–aided design (CAD) of analog systems.

7.4.1 Enabling Technologies

Developing large–scale FPAAs has created a need for new and improved methods of programming. The increased size alone mandates that switches must be programmed quickly or the large device is handicapped by the amount of time required to program it. The vast majority of the floating–gate transistors on large–scale FPAAs are switches, and programming switches does not typically require a high degree of accuracy. Thus, a separate programming scheme for switches can be optimized for speed (to the detriment of programming accuracy).

An additional increase in programming speed can be achieved by moving the programming control logic on–chip. When this is done, the programming logic can be replicated for each row of the device, and a row–parallel programming scheme can be implemented. The speed–up that can be realized in this scheme is significant and is an ongoing research project in the CADSP group.

The architecture of these large–scale FPAAs requires large switch networks. Even in a routing–hierarchy approach, the local switch interconnects will need to be a crossbar switch to enable maximum connectivity. As can be seen in the layout picture in Fig. 63, the switch networks consume a large area. Making these networks denser would be a large benefit in terms of cost and scalability. Research on the core floating–gate technology is ongoing. Researchers in the CADSP group are currently characterizing alternative methods of programming (particularly tunnelling) floating–gate transistors.

7.4.2 Computational Elements

A number of different CAB designs have been proposed within the CADSP group. Researchers are currently working on CAB designs that include minor variations to

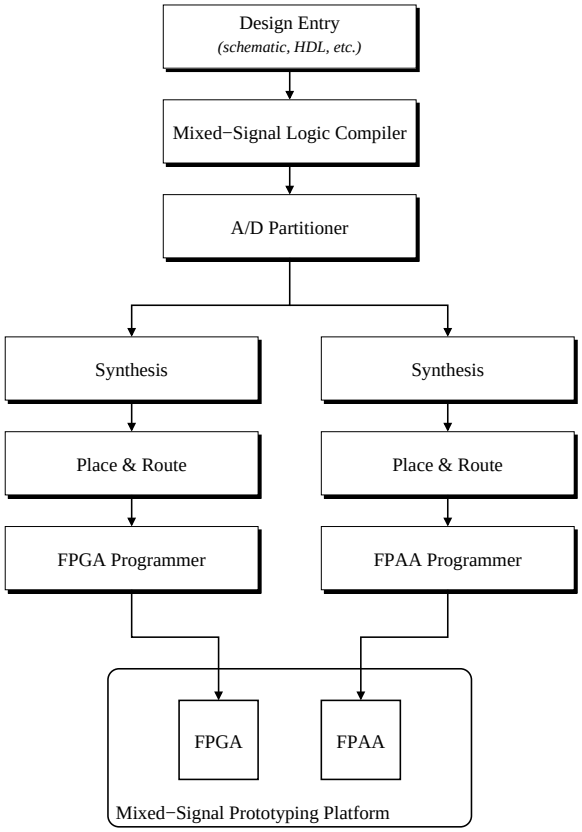


Figure 74. This is a block diagram of the ideal tool flow to enable mixed–signal rapid prototyping. Here, the CAD toolflow accepts a high–level, mixed–signal system description (i.e., a VHDL-AMS description or graphical schematic) and then performs an automated partitioning of the system into an optimal balance of digital and analog circuitry. The resulting analog and digital netlists are then passed through separate synthesis, place and route, and programming tool flows that are targeted for the specific devices on the development system.

the RASP CABs, designs optimized for high–speed operation, and designs based on emerging logic circuits, such as multiple–input translinear elements (MITEs). Some of these designs will fit directly into the RASP architecture with no significant change to the routing interconnects or control infrastructure. Other designs, such as the MITE FPAAs, are new designs that have been developed from the framework introduced here.

7.4.3 CAD Tool Flow for Analog Systems

When FPAAs are small, manually specifying placement and routing of circuits onto the FPAAs architecture is not too burdensome. However, as FPAAs increase in size to

include complex CABs, non–homogeneous meshes of computational logic, and hierarchical routing interconnects, manual design becomes impractical or even impossible. Thus, a CAD tool flow needs to be developed for FPAAs. As shown in Fig. 74, the ideal CAD tool flow would support a high–level entry methodology such as VHDL and VHDL-AMS for mixed–signal hardware design specification. The CAD tools would then automatically partition the system into the optimal balance of digital and analog hardware and issue separate digital and analog netlists. Each netlist would then proceed through the synthesis, placement, routing, and programming of the respective reconfigurable devices (FPGA and FPAA). As outlined in Section 2.4, some work has already progressed on the analog side. Most of this work is targeted at smaller FPAAs such as Anadigm's FPAAs. It has yet to be shown if these algorithms will scale to the larger FPAAs discussed here. Researchers in the CADSP group have begun to look at the challenges of placement and routing for large–scale FPAAs and are using the RASP 2.5 architecture as a testbed target for their software.

7.5 Commercialization

As FPAA research has progressed, academia and industry have shown increased interest in this work. Because of this interest, the FPAA research discussed here is covered in a pending patent [19]. In particular, GTronix, a start–up company coming from the Georgia Institute of Technology's VentureLabs business incubator, has received funding to pursue commercialization of an FPAA on the scale of the RASP 2.5. This effort will result in further development in order to transform the RASP FPAA architecture from a laboratory testbed into a viable commercial product. It is also anticipated that by moving the development associated with large–scale VLSI design into the corporate environment, students and faculty in the CADSP group will be able to focus on advancing the research.

7.6 Original Contributions

This thesis has detailed the investigation of a novel, large–scale FPAA that is analogous to current commercial FPGAs in its size, usefulness, and flexibility. The engineering contribution of this work can be summarized as follows:

1. **System–level architecture of a floating–gate FPAA:** The system–level architecture of an FPAA based on floating–gate technology has been presented. A novel, large–scale FPAA architecture has been specified and two testbed FPAAs based on this design have been fabricated and tested. Using analysis, simulation, and experimental data, these FPAAs have been shown to be highly flexible, functional, and generic in nature. The feasibility of large–scale, useful FPAAs based on this design has been shown.

2. **Mixed–signal prototyping platform for signal processing applications:** A mixed–signal prototyping platform has been created. By combining large–scale FPAAs and FPGAs together into a single system, the first realistically viable mixed–signal prototyping platform is demonstrated.

3. **Roadmap for future FPAAs:** This research has laid the foundation of a new genre of research in reconfigurable analog systems. By showing the viability of large–scale FPAAs, this effort has renewed interest in the research and development of reconfigurable analog systems in both academia and industry. Within the CADSP group alone, FPAA research has transitioned from a single project to being a defining thrust of the group. Not only is a core team of researchers now dedicated to FPAA research, but the design methodologies introduced by reconfigurable logic have permeated many of the research projects within the group (see Section 7.4). On the industry side, a number of corporations have shown an interest in the research presented here, and a start–up company has been created and funded to commericialize an FPAA based on this work.

7.7 Conclusion

Large–scale FPAAs based on floating–gate technologies provide the necessary levels of programmability and functionality to implement complex signal processing systems. With orders–of–magnitude savings in power consumption over traditional digital approaches, this reconfigurable analog technology offers an attractive alternative for implementing advanced signal processing systems in low–power embedded systems. The floating–gate transistors are shown to provide a compact switch that exhibits relatively flat resistance characteristics across the full operating voltage and can be programmed to be an active circuit element (e.g., variable resistor).

The technologies and architectures explored in this thesis have been demonstrated in fabricated, testbed FPAAs. Characterization and system–level data have been experimentally acquired. Systems implemented on these FPAAs have shown that FPAAs are highly flexible, and individual computational elements have proved to be programmable across a wide range of frequencies, Q-peaks, bandwidths, and time constants.

Large–scale FPAAs based on the testbed architectures have been designed and are currently being fabricated. However, more than just a single design, large–scale FPAAs enable a whole new design paradigm. The FPAA architecture and infrastructure developed here form a design methodology that can be applied to an ever–widening scope of problems. This FPAA framework brings programmability, reconfigurability, and reuseability to analog circuit design.

REFERENCES

[1] Adams, W. J., Nedungadi, A., and Geiger, R. L., "Design of a programmable OTA with multi–decade transconductance adjustment," in *Proceedings of the International Symposium on Circuits and Systems*, vol. 1, pp. 663–666, May 1989.

[2] Anadigm, http://www.anadigm.com/Prs_15.asp/, *Anadigm Company Fact Sheet*, May 2003.

[3] Anadigm, http://www.anadigm.com/Supp_05.asp/, *Anadigm FPAA Family Overview*, Mar. 2003.

[4] Anderson, D., Marcjan, C., Bersch, D., Anderson, H., Hu, P., Palusinski, O., Gettman, D., Macbeth, I., and Bratt, A., "A field programmable analog array and its application," in *Proc. IEEE Custom Integrated Circuits Conference*, May 1997.

[5] Anderson, D. V., Hasler, P., Ellis, R., Yoo, H., Graham, D. W., and Hans, M., "A low–power system for audio noise suppression: A cooperative analog–digital signal processing approach," in *Proc. 2002 DSP Workshop*, (Pine Mountain, GA), Oct. 2002.

[6] Bandyopadhyay, A. and Hasler, P., "A fully programmable CMOS block matrix transform imager architecture," in *Proceedings of the 2003 IEEE Custom Integrated Circuits Conference*, pp. 189–192, Sept. 2003.

[7] Becker, J. and Manoli, Y., "A continuous–time field programmable analog array (FPAA) consisting of digitally reconfigurable Gm–cells," in *Proceedings of the 2004 IEEE International Symposium on Circuits and Systems*, pp. I.1092–I.1095, May 2004.

[8] Binns, R. J., Hallam, P., Mack, B., and Massara, R., "High–level design of analogue circuitry using an analogue hardware description language," in *IEE Colloquium on Mixed–Signal AHDL/VHDL Modelling and Synthesis*, pp. 3/1–3/8, Nov. 1997.

[9] Birkner, J. M. and Chua, H.-T., "Programmable array logic circuit." U.S. Patent No. 4,124,899.

[10] Bratt, A., "Motorola field programmable analogue arrays, present hardware and future trends," in *IEE Half–day Colloquium on Evolvable Hardware Systems*, pp. 1/1–1/5, Mar. 1998.

[11] Bridges, S., Figueroa, M., Hsu, D., and Diorio, C., "Field–programmable learning arrays," in *Advances in Neural Information Processing Systems 15* (Becker, S., Thrun, S., and Obermayer, K., eds.), pp. 1155–1162, Cambridge, MA: MIT Press, 2002.

[12] Chang, S., Hayes-Gill, B., and Paul, C., "Multi-function block for a switched current field programmable analog array," in *1996 Midwest Symposium on Circuits and Systems*, Aug. 1996.

[13] Chawla, R., Bandyopadhyay, A., Srinivasan, V., and Hasler, P., "A 531nW/MHz, 128 x 32 current–mode programmable analog vector–matrix multiplier with over two decades of linearity," in *Proceedings of the 2004 IEEE Custom Integrated Circuits Conference*, (Orlando, FL), 2004.

[14] Chow, P., Seo, S. O., Rose, J., Chung, K., Paez-Monzon, G., and Rahardja, I., "The design of an SRAM–based field–programmable gate array—part i: architecture," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 7, pp. 191–197, June 1999.

[15] Chow, P., "A field–programmable mixed–analog–digital array," Master's thesis, University of Toronto, 1994.

[16] Chow, P. and Gulak, P. G., "A field–programmable mixed–analog–digital array," in *Proc. 3rd International ACM Symposium on Field–Programmable Gate Arrays*, pp. 104–109, ACM Press, 1995.

[17] Dudek, P. and Hicks, P. J., "A CMOS general–purpose sampled–data analog processing element," *IEEE Transactions on Circuits and Systems II*, vol. 47, pp. 467–473, May 2000.

[18] Dudek, P. and Hicks, P. J., "A CMOS general–purpose sampled–data analogue microprocessor," in *Proceedings of the International Symposium on Circuits and Systems*, vol. 2, pp. 417–420, May 2000.

[19] Duggar, J. D., Hall, T. S., Hasler, P., Anderson, D. V., Smith, P. D., Kucic, M. R., and Bandyopadhyay, A., "Floating–gate analog circuit." U.S. Patent Application No. 20030183871, 2003. Patent Pending.

[20] Dugger, J. and Hasler, P., "A continuously-adapting analog node using floating-gate synapses," in *IEEE Midwest Conference on Circuits and Systems*, (East Lansing, MI), Aug. 2000.

[21] Edwards, R. T., Strohbehn, K., and Jaskulek, S. E., "A field–programmable mixed–signal array architecture using antifuse interconnects," in *Proceedings of the International Symposium on Circuits and Systems*, vol. 3, pp. 319–322, May 2000.

[22] Ellis, R., Yoo, H., Graham, D., Hasler, P., and Anderson, D., "A continuous–time speech enhancement front–end for microphone inputs," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 2, (Phoenix, AZ), pp. II.728–II.731, May 2002.

[23] Ellis, R., Yoo, H., and Anderson, D. V., "An analog floating-gate IC for audio noise suppression," in *Proceedings of the International Symposium on Circuits and Systems*, (Phoenix, AZ), May 2002. Invited Paper.

[24] Embabi, S., Quan, X., Oki, N., Manjrekar, A., and Sanchez-Sinencio, E., "A field programmable analog signal processing array," in *IEEE 39th Midwest Symposium on Circuits and Systems*, vol. 1, pp. 151–154, Aug. 1996.

[25] Enright, D. and Mack, R. J., "A hierarchical technique to model parametric device variation in non–linear analogue CMOS architectures," in *IEE Colloquium on Mixed–Signal AHDL/VHDL Modelling and Synthesis*, pp. 1/1–1/7, Nov. 1997.

[26] Fast Analog Solutions Ltd., http://www.zetex.com, *Totally re-configurable analog circuit – TRAC*, Mar. 1999.

[27] Faura, J., Lacadena, I., Torralba, A., and Insenser, J. M., "Programmable analog hardware: a case study," in *IEEE International Conference on Electronics, Circuits and Systems*, vol. 1, pp. 297–300, Sept. 1998.

[28] Franz, G., "Digital signal processor trends," *IEEE Micro*, vol. 20, pp. 52–59, Nov–Dec 2000.

[29] Ganesan, S. and Vemuri, R., "FAAR: A router for field–programmable analog arrays," in *Proc. 12th International Conference on VLSI Design*, pp. 556–563, Jan. 1999.

[30] Ganesan, S. and Vemuri, R., "A methodology for rapid prototyping of analog systems," in *International Conference on Computer Design*, pp. 482–488, Oct. 1999.

[31] Ganesan, S. and Vemuri, R., "Technology mapping and retargeting for field–programmable analog arrays," in *Proc. Design, Automation and Test in Europe Conference and Exhibition*, pp. 58–64, Mar. 2000.

[32] Ganesan, S. and Vemuri, R., "Analog–digital partitioning for field–programmable mixed signal systems," in *Proc. Conference on Advanced Research in VLSI*, pp. 172–185, Mar. 2001.

[33] Ganesan, S. and Vemuri, R., "Behavioral partitioning in the synthesis of mixed analog–digital systems," in *Proc. Design Automation Conference*, pp. 133–138, June 2001.

[34] Gaudet, V. C. and Gulak, P. G., "CMOS implementation of a current conveyor–based field–programmable analog array," in *Conference Record of the 31st Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1156–1159, Nov. 1997.

[35] Gulak, P. G., "Field–programmable analog arrays: past, present and future perspectives," in *IEEE Region 10 International Conference on Microelectronics and VLSI*, pp. 123–126, Nov. 1995.

[36] Hall, T. S., Hasler, P., and Anderson, D. V., "Field–programmable analog arrays: A floating–gate approach," in *Proc. 12th International Conference on Field Programmable Logic and Applications*, (Montpellier, France), pp. 424–433, Sept. 2002.

[37] Hall, T. S., Twigg, C. M., Hasler, P., and Anderson, D. V., "Application performance of elements in a floating–gate FPAA," in *Proceedings of the 2004 IEEE International Symposium on Circuits and Systems*, pp. II.589–II.592, May 2004.

[38] Hall, T. S., Twigg, C. M., Hasler, P., and Anderson, D. V., "Developing large–scale field–programmable analog arrays," in *Proc. 18th International Parallel and Distributed Processing Symposium*, (Santa Fe, New Mexico), April 2004.

[39] Hasler, P., Bandyopadhyay, A., and Smith, P., "A fully programmable CMOS block matrix transform imager architecture," in *Proceedings of the 2002 IEEE International Symposium on Circuits and Systems*, pp. III.337–III.340, May 2002.

[40] Hasler, P., Kucic, M., and Minch, B. A., "A transistor-only circuit model of the autozeroing floating-gate amplifier," in *Midwest Conference on Circuits and Systems*, (Las Cruces, NM), 1999.

[41] Hasler, P. and Lande, T., "Special issue on floating-gate devices, circuits, and systems," *IEEE Journal of Circuits and Systems*, vol. 48, Jan. 2001.

[42] Hasler, P. and Minch, B. A., *Floating-gate Devices, Circuits, and Systems.* in press, 2002.

[43] Hasler, P., Minch, B. A., and Diorio, C., "Adaptive circuits using pFET floating-gate devices," in *Proceedings of the 20th Anniversary Conference on Advnaced Research in VLSI*, (Atlanta, GA), pp. 215–229, March 1999.

[44] Hasler, P., Minch, B. A., and Diorio, C., "An autozeroing floating-gate amplifier," *IEEE Transactions on Circuits and Systems II*, vol. 48, pp. 74–82, Jan. 2001.

[45] Hasler, P. and Anderson, D. V., "Cooperative analog-digital signal processing," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Orlando, FL), May 2002. Invited Paper.

[46] Hasler, P. and Dugger, J., "Correlation learning rule in floating-gate pFET synapses," *IEEE Transactions on Circuits and Systems II*, vol. 48, pp. 65–73, Jan. 2001.

[47] Hasler, P., Smith, P., Ellis, R., Graham, D., and Anderson, D. V., "Biologically inspired auditory sensing system interfaces on a chip," in *2002 IEEE Sensors Conference*, (Orlando, FL), June 2002. Invited Paper.

[48] Keymeulen, D., Zebulum, R. S., Jin, Y., and Stoica, A., "Fault–tolerant evolvable hardware using field–programmable transistor arrays," *IEEE Transactions on Reliability*, vol. 49, pp. 305–316, Sept. 2000.

[49] Klein, H. W., "The EPAC architecture: an expert cell approach to field programmable analog circuits," in *IEEE 39th Midwest Symposium on Circuits and Systems*, vol. 1, pp. 169–172, Aug. 1996.

[50] Koneru, S., Lee, E. K. F., and Chu, C., "A flexible 2–d switched–capacitor FPAA architecture and its mapping algorithm," in *42nd Midwest Symposium on Circuits and Systems*, vol. 1, pp. 296–299, Aug. 1999.

[51] Kucic, M., Hasler, P., Dugger, J., and Anderson, D. V., "Programmable and adaptive analog filters using arrays of floating-gate circuits," in *2001 Conference on Advanced Research in VLSI* (Brunvand, E. and Myers, C., eds.), pp. 148–162, IEEE Computer Society, March 2001.

[52] Kucic, M., Low, A., Hasler, P., and Neff, J., "A programmable continuous-time floating-gate Fourier processor," *IEEE Transactions on Circuits and Systems II*, vol. 48, pp. 90–99, Jan. 2001.

[53] Kutuk, H. and Kang, S.-M., "A field–programmable analog array (FPAA) using switched–capacitor techniques," in *Proceedings of the International Symposium on Circuits and Systems*, vol. 4, pp. 41–44, May 1996.

[54] Lattice Semiconductor Corporation, Hillsboro, OR, *ispPAC Overview*, Sept. 1999.

[55] Lee, E. K. F. and Gulak, P. G., "A CMOS field–programmable analog array," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 1860–1867, Feb. 1991.

[56] Lee, E. K. F., *Field–programmable analog arrays on MOS transconductors*. PhD thesis, University of Toronto, 1995.

[57] Lee, E. K. F., "Reconfigurable pipelined data converter architecture," in *IEEE 39th Midwest Symposium on Circuits and Systems*, vol. 1, pp. 162–165, Aug. 1996.

[58] Lee, K. and Gulak, P., "A CMOS field-programmable analog array," in *IEEE International Solid–State Conference Digest of Technical Papers*, pp. 186–188, Feb. 1991.

[59] Lee, K. and Gulak, P., "A transconductor-based field-programmable analog array," in *IEEE International Solid–State Conference Digest of Technical Papers*, pp. 198–199, Feb. 1995.

[60] Long, D. I., "Behavioural modelling of mixed–signal circuits using PWL waveforms," in *IEE Colloquium on Mixed–Signal AHDL/VHDL Modelling and Synthesis*, pp. 2/1–2/6, Nov. 1997.

[61] Marsh, D., "Programmable analogue ICs challenge spice–and–breadboard designs," in *EDN Europe*, pp. 30–36, http://www.ednmag.com: Reed Business Information, Oct. 2001.

[62] Mead, C., *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1989.

[63] Minch, B. A., Diorio, C., Hasler, P., and Mead, C. A., "Translinear circuits using subthreshold floating-gate MOS transistors," *Analog Integrated Circuits and Signal Processing*, vol. 9, no. 2, pp. 167–179, 1996.

[64] Pankiewicz, B., Wojcikowski, M., Szczepanski, S., and Sun, Y., "A CMOS field programmable analog array and its application in continuous–time OTA-C filter design," in *Proceedings of the International Symposium on Circuits and Systems*, vol. 1, pp. 5–8, May 2001.

[65] Pankiewicz, B., Wojcikowski, M., Szczepanski, S., and Sun, Y., "A field programmable analog array for CMOS continuous–time OTA-C filter applications," *IEEE Journal of Solid-State Circuits*, vol. 37, pp. 125–136, Feb. 2002.

[66] Pierzchala, E., Perkowski, M. A., and Grygiel, S., "A field programmable analog array for continuous, fuzzy, and multi–valued logic applications," in *24th International Symposium on Multiple–Valued Logic*, pp. 148–155, May 1994.

[67] Pierzchala, E., Perkowski, M. A., Halen, P. V., and Schaumann, R., "Current–mode amplifier/integrator for a field–programmable analog array," in *IEEE International Solid–State Conference Digest of Technical Papers*, pp. 196–197, Feb. 1995.

[68] Premont, C., Grisel, R., Abouchi, N., and Chante, J.-P., "Current–conveyor based field programmable analog array," in *IEEE 39th Midwest Symposium on Circuits and Systems*, vol. 1, pp. 155–157, Aug. 1996.

[69] Quan, X., Embabi, S., and Sanchez-Sinencio, E., "A current-mode based field programmable analog array architecture for signal processing applications," in *IEEE 1998 Custom Integrated Circuits Conference*, (Santa Clara, CA), pp. 277–280, May 1998.

[70] Ray, B., Chaudhuri, P. P., and Nandi, P. K., "Design of OTA based field programmable analog array," in *Proc. 13th International Conference on VLSI Design*, pp. 494–498, Jan. 2000.

[71] Sanchez-Sinencio, E., Ramirez-Angulo, J., Linares-Barranco, B., and Rodriguez-Vazquez, A., "OTA-based non-linear function approximations," in *ISCAS Proc.*, vol. 1, pp. 96–99, May 1989.

[72] Santini, C. C., Zebulum, R., Pacheco, M. A. C., Vellasco, M. M. R., and Szwarcman, M. H., "Evolution of analog circuits on a programmable analog multiplexer array," in *Proc. IEEE Aerospace Conference*, vol. 5, pp. 2301–2308, Mar. 2001.

[73] Sarpeshkar, R., *Efficient precise computation with noisy components: extrapolating from an electronic cochlea to the brain.* Pasadena, CA: PhD thesis, California Institute of Technology, 1997.

[74] Serrano, G., Smith, P., Lo, H. J., Chawla, R., Hall, T., Twigg, C., and Hasler, P., "Automatic rapid programming of large arrays of floating–gate elements," in *Proceedings of the 2004 IEEE International Symposium on Circuits and Systems*, pp. I.373–I.376, May 2004.

[75] Sivilotti, M. A., *Wiring Considerations in Analog VLSI Systems, with Application to Field-Programmable Networks (VLSI).* PhD thesis, California Institute of Technology, Pasadena, CA, 1991.

[76] Smith, P., Kucic, M., and Hasler, P., "Accurate programming of analog floating–gate arrays," in *Proceedings of the 2002 International Symposium on Circuits and Systems*, (Phoenix, AZ), pp. V.489–V.492, May 2002.

[77] Smith, P. D., Kucic, M., Ellis, R., Hasler, P., and Anderson, D. V., "Mel–frequency cepstrum encoding in analog floating–gate circuitry," in *Proceedings of the International Symposium on Circuits and Systems*, vol. 4, pp. 671–674, 2002.

[78] Stoica, A., Zebulum, R., Keymeulen, D., Tawel, R., Daud, T., and Thakoor, A., "Reconfigurable VLSI architectures for evolvable hardware: from experimental field programmable transistor arrays to evolution-oriented chips," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 9, pp. 227–232, Feb. 2001.

[79] Sun, Y., ed., *Design of high frequency integrated analogue filters.* London, UK: The Institution of Electrical Engineers, 2002.

[80] Wakerly, J. F., *Digital Design: Principles and Practices*, ch. 5: Combinational logic design practices, pp. 311–455. New Jersey: Prentice Hall, 3rd ed., 1999.

[81] Yoo, H., Anderson, D. V., and Hasler, P., "Continuous–time audio noise suppression and a custom low-power IC implementation," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Orlando, FL), May 2002. Invited Paper.