Programmable and Adaptive Analog Filters using Arrays of Floating-Gate Circuits

Matt Kucic, Paul Hasler, Jeff Dugger, and David Anderson
Georgia Institute of Technology, Atlanta, GA 30332-0250
E-mail: phasler@ee.gatech.edu

Abstract

In this paper we describe a programmable and adaptive filter based on floating-gate technology. We review the basics of floating-gate techniques and how they enable programmable and adaptive filter circuits. We describe our programmable filter concepts, and show experimental results of programmable filter operation. We also describe programming methods, and extend the programmability to a wide range of functions and circuits using the same approach. Further, we describe our techniques and custom programmer board for floating-gate programming of an IC. We show how to extend our programmable filters as adaptive filters both through weight perturbation methods and continuously adapting correlation rule methods.

1: Analog Computing Arrays

We introduce the use of high density analog computing arrays for signal processing based on non-volatile semiconductor memory cells. We base this model on an efficient computing paradigm in which highly parallel signal processing computations are performed through analog memory elements based on modified EEPROM cells. The high density analog computing arrays (referred to as computing arrays) require a core memory cells similar to a standard EEPROM or SRAM cell with a small amount of additional circuitry. The enabling technology of this approach is a floating-gate circuit technology that allows for simultaneous storage, computation, and programming in each cell. Unlike digital memory, each cell acts as a multiplier that multiplies the analog input signal to that cell by an analog value stored in a floating gate. By performing the computation in the memory cells themselves we avoid the through-put bottlenecks found in most signal processing systems. We can also extend this computational approach to many other signal processing operations and algorithms in a straightforward manner. The range of applications for computable memories reaches from auditory and speech processing, to beam-forming, multidimensional signal processing, and radar computations, communications processing, and image processing and recognition. We believe that the high density analog computing arrays will be an important option for designers who want to implement advanced signal processing algorithms for embedded and very low-power systems.

Our analog computing arrays are based on arrays of dense floating-gate transistors that provide non-volatile storage, compute a product between this stored weight and the inputs, allow for programming that does not affect the computation, and adapt due to correlations of input signals. Figure 1a shows a general block-diagram of our floating-gate computing array. Each *processor* is composed of two floating-gate transistors, and therefore corresponds closely to EEPROM densities. The memory cells may be accessed individually (for readout or programming) or they may be used for full parallel computation within the array (as in matrix-vector multiplication or adaptation). Therefore, we have full parallel computation with the same circuit complexity and power

1522-869X/01 $10.00 © 2001 IEEE

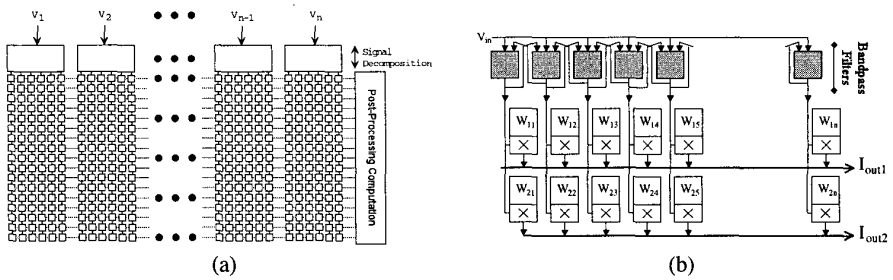

Figure 1: Illustration of our Computing in Floating-Gate Memory Arrays. (a) A typical system is an array of floating-gate computing elements, surrounded by input circuitry to precondition or decompose the incoming sensor signals, and surrounded by output circuitry to post-process the array outputs. We use additional circuitry to individually program each analog floating-gate element. (b) Top level picture of our Programmable Analog Fourier Processor. We separate the signal into frequency bands not by computing a DFT algorithm, but by a series of band-pass filters. We divide our frequency space exponentially, instead of linearly as in typical DFT algorithm. One current implementation for auditory and low MHz filtering. Here we choose a Fourier-transform like basis function; therefore the function is similar to DSP filtering using DFT and inverse-DFT stages.

dissipation as the digital memory needed to store this array of digital coefficients at 4-bit accuracy. This technology can be integrated in a standard digital CMOS process or in standard double-poly CMOS processes. Further, we only need operate this system with one memory access per incoming sample, or in other words, the system only needs to operate at the incoming data speed (maximum input frequency); therefore, reducing requirements on overall system design.

This technology has its roots in three key technologies. The first technology comes from research in Analog VLSI applied toward neural applications. This technology borrows on the history of the early technology development of analog matrix-vector computations from neural network implementations. This technology also borrows several neuromorphic concepts [24], including sigmoid (tanh) functions [24], Winner-Take-All (WTA) circuits, resistive or diffusor networks [2], cochlea models [23, 25], and retina models [24, 2]. This technology, particularly the neuromorphic directions, has potential for low-power computation. The second technology comes from research in data flow architectures and parallel processing [21]. Instead of having data stored and then fetched, etc, the idea is to directly process the data as it comes into the system, and store coefficients locally. Our technology extends this approach by removing the need for the digital processor in the locally stored memory. The third technology comes from floating-gate devices, circuits, and systems. To build this technology, we have already experimentally demonstrated the development of a Floating-gate technology in standard CMOS processes that can store or adapt its memory while simultaneously computing an analog multiplication [9]. This technology is the fundamental element in this approach, and provides the memory density in our large parallel computations. Since this previous work is critical to the success of this proposal, we describe this area in more detail in section 2, and one of the three proposed research directions is to improve this technology to be compatible with industrial practices.

In this paper, we will describe one important subset of these analog computing arrays, that is the space of programmable and adaptive analog filters based upon computing arrays. Figure 1b shows the top level description of our programmable filter chip. In this figure we show four band taps which can be expanded to as many as needed. The architecture is based on a modified DFT implementation in a DSP filter. We present analysis and experimental measurements of these pro-

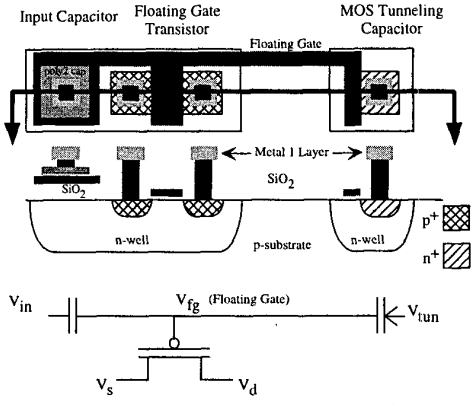

Figure 2: Layout, cross section, and circuit diagram of the floating-gate pFET in a standard double-poly n-well MOSIS process. The cross section corresponds to the horizontal line slicing through the layout view. The pFET transistor is the standard pFET transistor in the n-well process. The gate input 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. We add floating-gate charge by electron tunneling, and we remove floating-gate charge by hot-electron injection. Between V_{tun} and the floating-gate is our symbol for a tunneling junction, a capacitor with an added arrow designating the charge flow.

grammable and adaptive filter chips fabricated in a double-poly, 2.0μm, 1.2μm, and 0.5μm MOSIS processes; we have fabricated these elements on 0.25μm MOSIS single-poly process as well. Section II overviews the relevant floating-gate circuit technology for analog computing arrays. Section III describes our analog programmable filter. Section IV describes our floating-gate array programming scheme and floating-gate array programming board. Section V describes our adaptive filter techniques, including weight perturbation and continuous-time correlation techniques.

2: Overview of Floating-Gate Circuits

We have come to see floating-gate devices as not just for digital memories anymore, but circuit elements with analog memory and important time-domain dynamics [9, 16]. "Floating-gate circuits" is used to refer circuits where floating-gate devices are used as circuit elements and not simply as digital memory elements. Floating-gate devices and circuits typically divide into three major classes:

1. Floating-gate devices used as analog memory elements.
2. Floating-gate devices used as part of capacitive-based circuits.
3. Floating-gate devices used as adaptive circuit elements.

Figure 2 shows the layout, cross-section, and circuit symbol for our floating-gate pFET device. A floating gate is a polysilicon gate surrounded by silicon-dioxide. Charge on the floating gate is stored permanently, providing a long-term memory, because it is completely surrounded by a high-quality insulator. From the layout, we see that the floating-gate is a polysilicon layer that has no contacts to other layers; this floating-gate can be the gate of a MOSFET and can be capacitively connected to other layers. In circuit terms, a floating-gate occurs when we have no DC path to a fixed potential, precisely the effect avoided by many circuit designers and circuit simulators. No DC path implies only capacitive connections to the floating node, as seen in Figure 2.

2.1: Basics of Floating-Gate Circuits

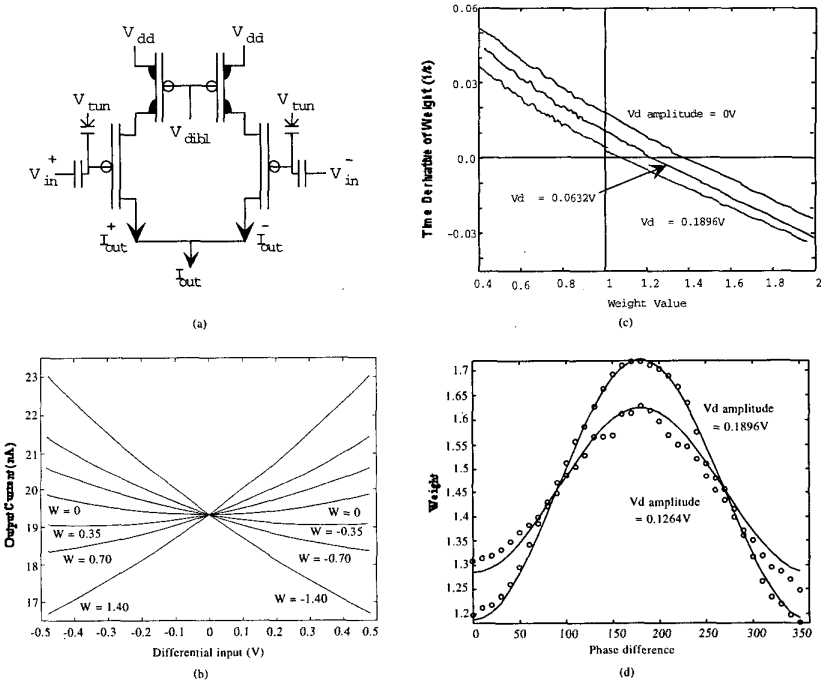

Figure 3: (a) Four-quadrant weighted multiplication using floating-gate devices. Between V_{tun} and V_{fg} is our symbol for a tunneling junction, which is a capacitor between the floating-gate and an n-well. (b) Experimental data showing the multiplication transfer characteristics of these devices. (c) dw/dt vs. w for varying $\Delta\hat{V}_d$ and fixed $\Delta\hat{V}_g$. This data gives strong evidence for our learning rule, $\tau'\Delta\dot{w} = -\Delta w + \eta E[xy]$. (d) Experimental measurements of floating-gate weight adaptation based upon a correlation learning rule. Change in the steady-state weight values due to degrees of correlation (phase difference) between sinusoidal voltage signals at the gate and drain terminals.

The floating-gate voltage, determined by the charge stored on the floating-gate, can modulate a channel between a source and drain, and therefore can be used in computation. Floating-gate circuits provide IC designers with a practical capacitor-based technology; capacitors, rather than resistors, are a natural result of a MOS process. Floating-gate devices can compute a wide range of trans-linear functions by a particular choice of capacitive couplings into floating-gate devices [16, 10, 9]. The charge on this floating-gate can be modified by projecting UV light on the chip, by applying large voltages across a silicon-oxide capacitor to tunnel electrons though the oxide, or by adding electrons using hot-electron injection. Floating-gate technology can eliminate off-chip-biasing voltages by providing these voltages on-chip with arrays of programmable floating-gate bias sources (e-pots) [6].

Adaptive floating-gate circuits utilize continuous programming currents to continuously adapt the floating-gate charge based upon the input signals. The list of currently demonstrated continuously adapting floating-gate circuits are the Auto-zeroing Floating-Gate Amplifier (AFGA) [10, 11], the adaptive floating-gate differential amplifier and bump circuit [12] the Second Order Section AFGA [17], the Gain-Adapting AFGA, and the Adaptive Synapse Element [13, 14]. Adap-

tation in floating-gate circuits is derived from the feedback mechanism caused by electrons moving from the transistor channel to the floating gate by hot-electron injection [12, 14]. Hot-electron injection modifies the channel current, and gate current is proportional to channel current; therefore, floating-gate adaptation is a direct function of the computations being performed.

2.2: Four-Quadrant Adaptive Floating-Gate Synapses

Four-quadrant adaptive floating-gate synapses are a fundamental element in analog computing arrays. We call these elements synapses because of their relationship to synapses in adaptive filters [5] and neural networks [18], and their loose connection with biological synapses. The output current is proportional to the four-quadrant multiplication between the differential input voltage and the stored weight, and the weight is updated by a four-quadrant correlation between the differential input voltage and an error voltage. The weighting is performed by storing charge on floating-gate transistors [10]; a typical circuit used for the multiplication is shown in Figure 3, but many other approaches are possible depending upon the application. The benefit of using a floating gate for the weighting is small size and circuit simplicity. By using the floating-gate transistor for the weighting we use the actually memory element as part of the computation, which allows for the highest obtainable chip density. This density is needed to realize chips with large number of taps or chips with multiple band weighted outputs. This memory element retains its value even when power is not applied to the device, and eliminates the need for separate memory cells with A/D and D/A circuitry to store and reproduce the actual analog weight in the circuit.

Another current research direction involved making the computation cells signal adaptive. Initial attempts used single transistor floating-gate synapses and complicated weight-update equations based upon the programming physics [7, 10]. Later, we showed the spectrum of adaptive floating-gate dynamics [12]. We also showed a weight-update rule based upon correlations of circuit terminals [13, 14, 15]. This weight update rule shows that the equilibrium weight value is proportional to the correlation between changing signals at the gate and drain voltages:

$$\tau' \Delta \dot{w} = -\Delta w + \eta E[xy], \tag{1}$$

where η is a function of device parameters, x is the AC portion of the gate voltage signal, and y is the AC portion of the drain voltage signal. The steady state solution to this equation is $\Delta w = \eta E[xy]$, or the correlation between the gate (input) and drain (error) signals. We can see this correlation effect by applying voltage signals (fast timescale) at the gate and drain and observing the steady-state of the weight (slow timescale). Assume that the input for the drain signal is $V_1 \sin(\omega t)$ and the gate signal input is $V_2 \sin(\omega t + \theta)$. Substituting these two inputs and computing the expected value, yields

$$\Delta w_{eq} \propto V_1 V_2 \cos(\theta). \tag{2}$$

Figure 3 shows the result of an experiment where we sweep θ from 0 to 2π and measure the steady state value of the weight (w_{eq}). We see definite correlations due to phase differences where $\Delta w_{eq} \propto -\cos(\theta)$. This is a Hebbian learning rule, based on the correlations between the signals x and y.

3: Programmable Analog Filters

Figure 1b shows the top-level description of our on-chip programmable analog filter concept In this figure we show three-to-four band taps that can be expanded to as many as desired. Each tap consists of two pieces, one piece that separates the signal into a set of analog basis functions, and a second piece that multiplies each signal by a stored weight and outputs this current onto a

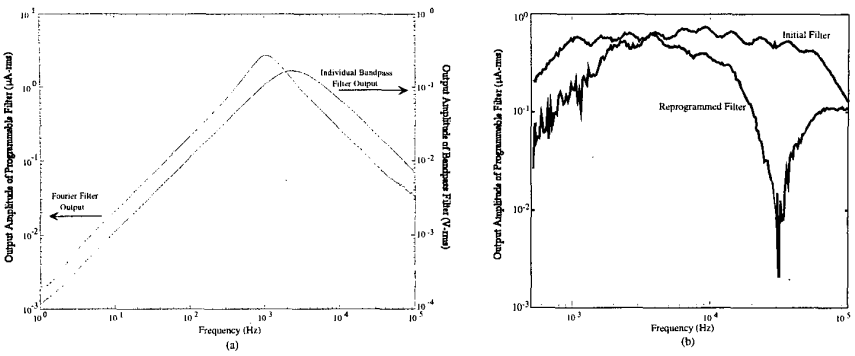

Figure 4: Experimental measurements of frequency response of our programmable band-pass filters. (a) Frequency response for a single band-pass filter, and for the array band-pass filters with programmed weighting function. We set zero tilt on the frequency line; therefore all corner frequencies should be identical aside from mismatch. The result of this programmable filter is a tighter band-pass filter, with a corner frequency roughly half of the original corner frequency. (b) Frequency response for this programmable filter with 10 band-pass elements exponentially spaced in frequency. The ripples on both curves show the location of these band-pass elements. We show an initial programmed frequency response, where the weights are nearly equal, and a second programmed frequency response to program an additional notch in the filter's response. We obtain similar frequency responses generated by our Spectre simulation model used for simulating floating-gate circuits.

common interconnection line. The output of each basis function is a voltage that can be broadcast to several weighted multipliers; therefore, with one processor we can output multiple different filtered versions of the original signal. We use two floating-gate MOSFETs to both store and perform either a two or four quadrant multiplication of signal and weight. By using the floating-gate transistor for the weighting we use the actually memory element as part of the computation, which allows for the highest obtainable chip density. The output of the weighted multiplier from each band is a current to allow simple addition via KCL (sum of the currents entering a node equals the sum of currents leaving that node) to compose the final output signal. With one processor we can output multiple band weighted versions of the original signal. We can post-process this current into a voltage for easy broadcast to further stages. This approach directly extends to multiple inputs and multiple signal-decomposition bases.

To separate the input signal into several time-dependent basis functions we use a voltage that can be broadcast to multiple band-pass filters. These filters are referred to as Capacitively Coupled Current Conveyors or C^4 [8, 20]. C4 filters operate from audio range to MHz range and their corner frequencies can be either arbitrarily spaced to provide , for example, linear or exponential spacing. These band-pass filters give a frequency decomposition of the incoming signal into multiple bands which are chosen based on the biases of each filter. We use an all-transistor version of the auto-zeroing floating-gate amplifier (AFGA), that we described elsewhere [8, 20], to achieve a broadly tuned band-pass response. We can program the filter corner frequencies using floating-gate elements and can program these values within our programming scheme; this ability allows for an arbitrary basis of filter responses. By adding feedback between the stages, we sharpen the filter response roll-off if desired. The frequency limitations of this approach depend upon the frequency limitations of the FETs, which depends upon the biasing point, and other known factors. Arrays of band-pass, low-pass, or high-pass filters are not the only basis functions to be used with these programmable filters, but are often the clearest to implement.

We show the frequency response of individual bands multiplied by their weighted outputs for constant effective weights. Figure 4a shows experimental measurements from our programmable Fourier filter. The result of this programmable filter is a tighter band-pass filter, with a corner frequency roughly half of the original corner frequency. The floating-gate devices used for the multiplication were built with W/L = 10; therefore the devices were biases near threshold with an average bias current of 1μA (total current was 20.58μA). We found that dynamics of the multipliers did not affect the filter transfer function, and that the harmonic distortion is limited by the multipliers, and not the band-pass filters.

Figure 4b shows the frequency response for a programmable filter with 10 band-pass elements that are exponentially spaced in frequency. For the initial frequency response, we used an exponential spacing with nearly identical weight values at each tap. We can see the exponential spacing of these band-pass elements by looking at the ripples on the initial frequency response. We also show a second programmed frequency response, where we programmed an additional notch in the spectrum of this initial filter. In current versions of this programmable filter, the parameters of each band-pass element are programmable, and, therefore, we can build our programmable filters utilizing band-pass filters with an arbitrary spacing.

We have developed simulation models of this circuit to assist in developing future revisions of this circuit. We recently presented a methodology for simulating floating-gate circuits in Cadence's simulator, Spectre, using a modified EKV MOSFET model [22]. The frequency responses that we obtain from this simulation model agree closely with experimental results.

4: Programming Approach for Floating-Gate Arrays

We present a systematic method for programming an array of floating-gate devices, which is a critical part of this single-chip system. These techniques are important when one is building signal-processing algorithms with thousands of floating-gate elements; currently we have fabricated and programmed arrays of 1 to 10 thousand floating-gate elements for signal processing applications. Alternate approaches, such as e-pots [6] require significant circuit complexity, and therefore are inefficient for large floating-gate arrays. In this system we trade-off user-friendly circuits, with all their circuit complexity, for simpler circuits programmed by well–controlled computer algorithms that could be used by industrial testers.

We also desire a programming algorithm that is based on controlling values actually used during operation; therefore there is no need for any compensation circuitry. As a result, we have developed a programming scheme where we perform injection over a fixed time window, and then measure the results after putting the cell in operating mode. Figure 5a shows the control logic we use to automate the programming of an array of floating gate devices. Once programmed, these floating-gate devices retain their channel current in a non-volatile manner.

Before deciding on a particular programming scheme, we first consider how the synapses interact when coupled into an array. We choose the tunneling and drain terminals to be common along a row; therefore, when programming one row, the other rows remain unaffected. We simplified our initial circuit by tying all tunneling rows together to consume only one pin on the package. This approach avoids the need for high-voltage transistor switching to tunnel along an individual row. This simplified approach can only select a column to be modified via tunneling with our implementation. This approach works successfully because we are using tunneling primarily for erasing, but in future revisions we will add row decoding to tunneling, which will add additional control of programming. Developing an efficient algorithm for pFET programming requires discussing the dependencies of the gate currents, and the ability to modify a single device with high selectivity. We program a device by increasing the output current using hot-electron injection, and decrease the output current using electron tunneling. For this described method, devices are programmed

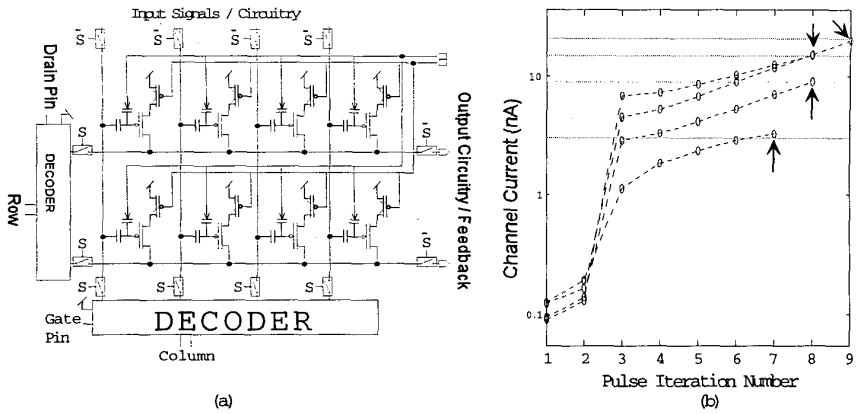

Figure 5: Our programming method for floating-gate pFET arrays. This approach is designed to be transparent to the computation of the pFET synapse array. (a) Circuit diagram of chip design to allow dual programming / operation. When the control signal S is 1, then we close the switches to the decoder circuitry, enabling programming, and open the switches to the normal operating circuitry. Both decoders either set their outputs to V_{dd} if 0 or select an output to an external pin. When the control signal S is 0, then we open the switches to the decoder circuitry, and close the switches to the normal operating circuitry. (b) Plot showing the programming of four current values. All four values converged within 9 steps.

with hot-electron injection and are reset by tunneling the devices below the level to which they are to be programmed. This is chosen due to device selectivity for each method which will be described. Because of the poorer selectivity, we use tunneling primarily for erasing and for rough programming steps.

The programing results where from an 1.2μm MOSIS process. A 2x4 array of floating gates were used for this experiment. The operation voltage for the chip was 3V. For programing, a voltage of 8V was used to allow for significant injection in this process to occur. During programming, the drain voltage was held at 5V to take the current measurements for system operation. The time T used for injection was 2 seconds. This value was chosen only to ensure that no timing problems arose in the test environment. There is no reason fast T values cannot be used, and future revisions of the test setup will include on board timing circuits to ensure constant timing at faster values. These fast values are critical to program mass production or large arrays of floating gates. Figure 5b shows the pulse steps at programming four devices in the array to different values.

We designed a custom programming board to program large floating-gate arrays. The programming board shown diagramatically in Figure 6 allows for flexible floating-gate array programming over a wide range of IC processes, and allows for nearly transparent operation to the user. Using custom circuits to program the floating gates allows for a self-contained programmer at a much lower cost than a rack of testing equipment. This programming board is connected to the chip via a standardize header allowing the option of additional logic when used as part of a larger testing approach. At the heart of the programmer is a PIC 16C77 that provides support for the serial interface, the D/A, the current measurement circuits, and the accurate timing necessary for programing. This board controls the programming sequences as instructed by software control over a standard RS232 port. The circuits found on this programming board besides the PIC control most of the actual programming operations as instructed by the PIC. The DAC provides voltages for the gate and drain, as well as driving a voltage regulator to set the voltage of the chip to program. Level

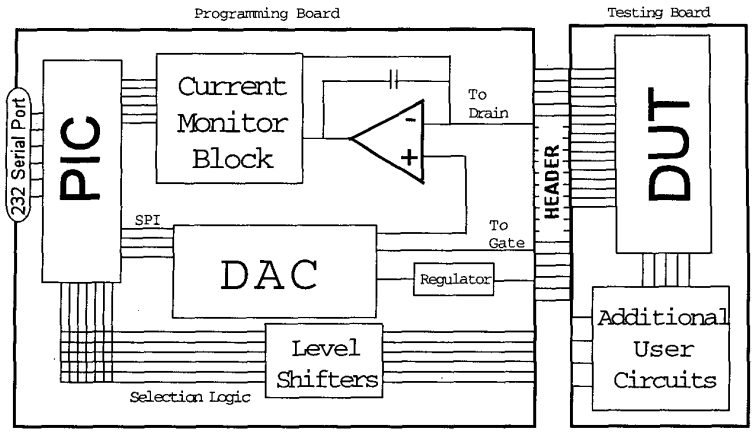

Figure 6: Block diagram of our custom programming board for automatic programming of large floating-gate arrays. This board, controlled by a PIC microcontroller and interfaced with a computer through a serial port, is capable of programming floating-gate arrays fabricated in a wide range of processes. This board allows easy integration with a larger testing platform, where programming and computation are both required.

shifters shift the PIC's logic levels to the chip's logic levels. To measure the current, a circuit has been designed that integrates the charge on the drain line. The integrated value is measured by the PIC as a pulse and then sent back over the serial line to the controlling PC (currently we are using MATLAB). The accuracy of this current measurement, measured as signal-to-noise ratio, has been experimentally been found to be equivalent to 9 bits of accuracy over 2 orders of magnitude in current. Initially a calibration is performed to calibrate the capacitor used to integrate the current; once this capacitor is calibrated, future time measurements from the programmer can easily are directly converted to currents.

5: Programmable Filters: Scalability and Comparison to Digital Filters

With our discussion on programmable analog filters, one might ask how this approach compares with a standard digital approach. A digital processor would require converting the incoming signal to a digital representation, computing an FFT operation, weighting the frequencies, and combining the resulting output. One could compare these techniques based on various functions of area, speed, and power dissipation. For illustration, we will compare the power dissipation of an analog and a digital implementation for the same functionality; power dissipation is important given the increasing demand on portable electronics. The analog approach will be smaller, because we perform the same computation in the space required to store less than 4bits of the required coefficients. We will compare these two approaches towards two applications, one using exponentially spaced output frequencies, and one using linearly spaced output frequencies. In general, analog will be more efficient at some level, if and only if the analog processors take advantage of digital CMOS scaling trends; therefore the analog circuitry does not become obsolete in a few design cycles. Our floating-gate technology has scaled, and will continue to scale, with digital CMOS scaling. Comparison of digital processing with adaptive analog floating-gate techniques—described in Section 6—will lean further towards the analog computation, because the enhanced computation occurs using the same current and same area.

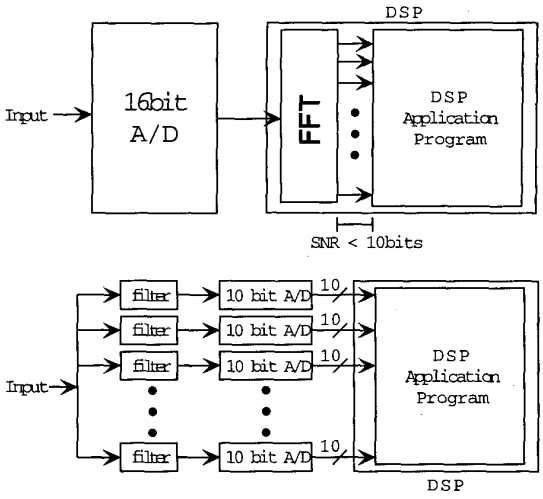

Figure 7: A practical example comparing using analog or digital signal processing for a particular output resolution (Signal–to–noise). One common signal processing step with incoming sensor data is taking an FFT, or equivalent Fourier based algorithm. For DSP computation, we would require a 16bit A/D converter to get some output channels at 10bit resolution. For ASP computation, we would require a bank of bandpass filters with 10bits of Signal-to-noise ratio coupled with a bank (or multiplexed) 10bit A/D converter to get the output channels at 10bit resolution. Both *analog* systems have similar design complexity. These computations are transparent (in resolution) to the engineers developing the remainder of the algorithm, and therefore tradeoffs could be made at these levels.

First we consider a typical speech / audio signal processing problem, Fourier decomposition at audio frequencies (20Hz to 20kHz) with exponentially spaced corner frequencies. Figure 7 illustrates this comparison in the context of a larger signal processing system. For digital computation, we would require a 16bit A/D converter to get some output channels at 10bit resolution. For analog computation, we would require a bank of bandpass filters with 10bits of Signal-to-noise ratio coupled with a bank (or multiplexed) 10bit A/D converter to get the output channels at 10bit resolution. We use exponentially spacing as a measure of comparison because it closely compares to the response of biological sensors (cochlea—Mel spacing) [24] as well as other physical systems. This approach is typically difficult for digital algorithms, for n-exponentially spaced outputs, we either need to take an n^2 point FFT, or an n point DFT with the appropriate coefficients. A modest auditory filter with 64 exponentially spaced outputs requires 4096 multiply operations and 4096 addition operations for a real FFT transform. As a result, we require 600million operations to use the Fourier transform as a filter bank, similar to the analog approach. Current DSP processors can achieve 20 MIPS / mW [1], which consumes 30mW for this computation, assuming no processor overhead; our analog processor would consume approximately 1μW for the same computation.

Second, we consider a typical higher-frequency problem, Fourier decomposition at MHz frequencies (1MHz - 33MHz) with linear spaced corner frequencies, such as a filter bank for IF signals. Because we are using Fourier decomposition with linearly spaced output frequencies, one expects a closer comparison on consumed paper. We would require at least 4 billion operations for this computation, which will consume 200mW [1] using 5 processors; our analog processor would consume approximately 400μW for the same computation.

6: Adaptive Techniques

In this system we describe adaptation mechanisms built from the programming mechanisms; the result is adaptive memory arrays of computing elements. Our approach not only shows the applicability of matched filter algorithms and weight perturbation algorithms, but also provides a common

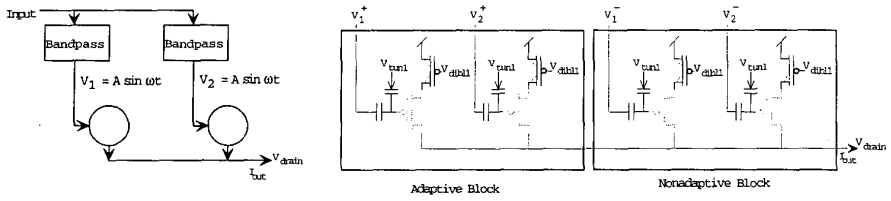

Figure 8: Top level figure of our adaptive filter topology, and a diagram of the adaptive floating-gate circuit. We adapt the synapses with the non-inverting inputs while keeping the synapses with the inverting inputs at a programmed level; in this way, we can account for the large offsets between equilibrium weights due to mismatches in the tunneling and injection processes.

framework to compare these approaches.

6.1: Continuously adapting filters

In this subsection, we will expand the functionality of our arrays of pFET single transistor synapses by making the computation cells signal adaptive. Our building computing elements that adapt as a function of the input and control signals; in general, we will input in the appropriate voltages to the gates, measure the output current coming from the drain, and apply the desired error signal as a drain voltage. Figure 8 shows the block-diagram of this approach as well as shows the circuitry used for our adaptive node. We programmed the negative synapses to eliminate the signal effect of the steady-state current of the positive synapses for similar input sizes (the resulting bias current increases). Also, since we use a differential input voltage, we get our output current by summation on one line, rather than taking the difference of two currents. This approach also compensates for the mismatch between synapse equilibrium points at the point where tunneling current equals injection current at each synapse. The goal is to develop a continuously adapting neural network layer using these floating-gate analog filters that will be used in our multilayer, adaptive control networks.

One straightforward example is that we apply the target signal as a drain voltage, and therefore we mathematically and experimentally connect our adaptive floating-gate devices to a class of adaptive filters. Other choices of circuitry at the drain terminal results in different weight-update rules. For the floating-gate element in Figure 3a, we have that the output, which is proportional to the output current (I_{out}) from the device, is $W_{i,j}\Delta V_j$ ($W_{i,j}x_j$), as well as the weight of an individual device adapting along the following equation:

$$\tau \frac{d\Delta W_{i,j}}{dt} = -\lambda(V_d)E\left[\Delta V_j \Delta V_{d,i}\right] - \Delta W_{i,j} \tag{3}$$

as a result, $\Delta W_{i,j} \rightarrow \lambda(V_d)E\left[\Delta V_j \Delta V_{d,i}\right] = \lambda(V_d)E[x\hat{y}]$. Considering a typical LMS adaptation [5], we get the weight update equation as

$$\frac{d\mathbf{W}}{dt} = -2E\left[\mathbf{x}[y-\hat{y}]\right] = -2\mathbf{Q}\mathbf{W} + 2E[\mathbf{x}\hat{y}],$$

where $\mathbf{W}$ is the weight matrix, $\mathbf{x}$ is the vector inputs, and $\mathbf{Q}$ is the covariance matrix of the inputs. The steady-state (solution) weight is $\mathbf{W} = \mathbf{Q}^{-1}E[\mathbf{x}\hat{y}] \rightarrow E[\mathbf{x}\hat{y}]$, if the inputs are uncorrelated.

We will show the behavior of this algorithm through two separate experimental measurements. Fig 9 shows results on a simple adaptive node using floating-gate devices; we have shown this

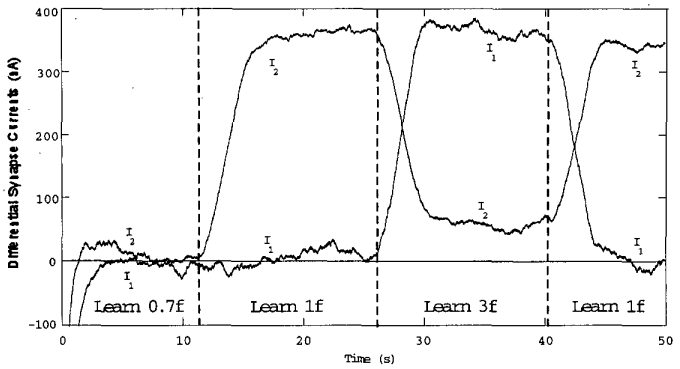

Figure 9: Experimental measurements (Current outputs) from our differential synapses showing frequency correlations for sinusoidal inputs to the two-input node given by $V_{g1} = sin(2\pi 3ft)$ and $V_{g2} = sin(2\pi ft)$. The learning signal, $V_d = sin(2\pi f_d t)$, takes on three different frequencies $f_d = 0.7f, f, 3f$. When $f_d = 0.7$, neither side adapts; when $f_d = f$, only V_{g2} adapts; when $f_d = 3f$, only V_{g1} adapts. We programmed our negative weights to offset the positive synapse current measured with input signals with the same signal variance. We see that the synapse that has an identical synapse input frequency as the drain signal has a non-zero weight.

mathematical formulation over a range of inputs. In the first experiment, we show adaptation to frequency correlations when we apply a different frequency to each of the synapse inputs, and we apply another sinusoid as a learning signal. Figure 9 shows experimental time-course measurements from our adaptive circuit, where the inputs are sinusoids at a fundamental and related third harmonic frequency, and the drain voltage is also another sinusoid. We observe that the circuit identifies a correlation between its input and the drain *learning* signal. As a second experiment, we step our adaptive circuit with inputs sinusoids at a fundamental and related third harmonic frequency, and with the drain voltage learning signal as a square wave of the fundamental frequency. Figure 10 shows the normalized amplitude and frequency of the Fast Fourier Transform of the resulting output (drain current) signal to make it easy to compare relative amplitudes at relative frequencies. From this experimental data, we get the expected square wave Fourier coefficients for the fundamental and third harmonics. This experiment demonstrates this circuit's behavior in extracting Fourier coefficients.

6.2: Weight Perturbation Adaptation in Analog Programmable Arrays

Weight perturbation can be thought of an algorithm between continuous adaptation and direct programming. The algorithm is straightforward: first, make a random change in a weight vector, second, if the vector *improves* the system, we keep this change, otherwise flush this change, and finally, repeat until converged [3, 4, 19]. Convergence is not surprising, because only weight improvements are used, and there is a non-zero probability that we "stumble" on useful directions. Some methods continue making using the *good* weight updates as long as it improves the system. Although these methods have a simple descriptions, these methods can be compared with gradient type methods [3, 4, 19]. Further, weight perturbation allows training using a computer in the adaptation loop. These algorithmic approaches are still the only adaptive algorithms for hybrid analog—digital computer solutions that has consistently converge for simple problems over a wide-range of parameters [3, 4]. Other adaptive approaches perform far worse than expected, primarily

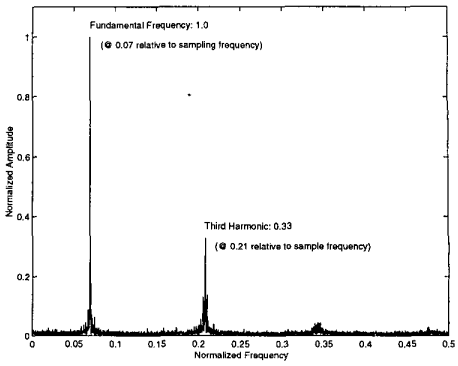

Figure 10: Experimental measurements of a square wave learning signal applied to V_d The output current spectrum shows the amount of each node input frequency matched to frequency components in learning signal. The frequency axis is normalized; the frequency does not affect these results until it approaches the adaptation rate. We obtain 1 and 1/3 for the fundamental and third harmonics as expected. The fifth harmonic appears due to the drain voltage coupling into the floating gate through overlap capacitance.

because other approaches end up using the worst of the analog and digital computing directions.

We will initially consider weight perturbation algorithms coupled with a simple microcontroller for diagnostics, external control, random-number generation. We use a microcontroller, because adding complexity to the synapse elements greatly increases the cost and decreases the overall effectiveness. This approach also has the advantage that it can work with existing ICs or slightly modified ICs with analog computational arrays. Our current approach is use the same IC programming infrastructure and board to perform this algorithm that was described earlier, and slightly adapt the PIC microcode. Generally, the additional cost of adding a microcontroller to a board is fairly inexpensive, and it is somewhat straightforward to integrating a special purpose controller on-chip if desired. Adding additional circuitry into each computational element is very costly for even medium size arrays, and should be avoided in practice.

The algorithm for weight perturbation is a simplification of our basic programming algorithm. We start with the positive and negative weights close (within a factor of 2) to each other as its initial conditions. We either program the weights to these starting values, or we use various continuous-time adaptation mechanisms to adapt the array to these starting values in a single array parallel step [12]. During each programming step we randomly perturb every weight in the matrix either up or down by a fixed ratio by either injecting either the positive weight or the negative weight to increase or decrease this weight. After testing this system with this new weight vector, we combine the decisions of keeping or flushing the particular weight update and providing the next weight update. To erase the effect of a weight update, we only need to inject the opposite weight (positive or negative) that was programmed. Our approach benefits from fast programming methods to update the weights in an array; since we only needing a "course" update for each element, we can envision pulse widths of $1\mu s$ for each array element. We use a global tunneling pulse to proportionally reduce the positive and negative weights (identical drops in the floating-gate voltages); after a few iterations, we tunnel such that the sum of the positive and negative weights returns to a fixed bias value.

7: Conclusions

This paper focused on the the first step of signal processing algorithms by analog computing arrays, namely programmable and adaptive analog filters. The fundamental element of these analog computing arrays is a floating-gate circuit, roughly the size of 3 EEPROM cells in a given CMOS process, where each of these analog processors stores a weight the can be programmed, multiplies

the weight by incoming input signals, and can be signal adaptive based on signal correlations. We see these analog computing arrays to appreciably advance technology that makes possible the integration of very low-power, highly functional analog signal processing circuits with digital signal processing circuits. We presented our analog programmable filter based on this floating-gate technology and from programmable on-chip band-pass filter arrays, as well as experimental results showing that we get widely different filters from a single structure. We presented our programming method for arrays of floating-gate elements, which is practically necessary to program thousands of floating-gate elements. We described our corresponding off-chip programmer board for transparent user programming of these values. We presented techniques to make these analog computing arrays signal adaptive, both through continuous-time adaptation that is directly dependent on the correlations of gate and drain signals, and through weight-perturbation schemes. Our experimental results on continuous-time adaptation gives hope that we can build an adaptive node capable of a wide range of adaptive algorithms.

References

[1] *TMS320C55x Technical Overview*. Texas Instruments Users Manuals, Texas Instruments, December 1999.

[2] K. Boahen and A. Andreou. A contrast-sensitive retina with reciprocal synapses. In J.E. Moody, editor, *Advances in Neural Information Processing Systems 4*. Morgan Kaufman Publishers, San Mateo, CA, 1991.

[3] Gert Cauwenberghs. A fast stochastic error-descent algorithm for supervised learning and optimization. In Gerald Tesauro and David S. Touretzky, editors, *Advances in Neural Information Processing Systems 5*, pages 244–251. Morgan Kaufman, San Mateo, CA, 1993.

[4] Gert Cauwenberghs. Analog vlsi stochastic perturbative learning architectures. *Analog Integrated Circuits and Signal Processing*, 13:195–213, 1997.

[5] Peter M. Clarkson. *Optimal and Adaptive Signal Processing*. CRC Press, 1993.

[6] R.R. Harrison, J.A. Bragg, P. Hasler, B.A. Minch, and S. Deweerth. A CMOS programmable analog memory cell array using floating-gate circuits. *IEEE Transactions on Circuits and Systems*, 2000. In Press.

[7] P. Hasler, C. Diorio, B. A. Minch, and C. A. Mead. Single transistor learning synapses. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 817–824. MIT Press, Cambridge, MA, 1995.

[8] P. Hasler, M. Kucic, and B. A. Minch. A transistor-only circuit model of the autozeroing floating-gate amplifier. In *Midwest Conference on Circuits and Systems*, 1999.

[9] P. Hasler and T.S. Lande. Special issue on floating-gate devices, circuits, and systems. *IEEE Journal of Circuits and Systems*, 2000. in Press.

[10] P. Hasler, B. A. Minch, and C. Diorio. Adaptive circuits using pFET floating-gate devices. In *Proceedings of the 20th Anniversary Conference on Advanced Research in VLSI*, pages 215–229, Atlanta, GA, March 1999.

[11] P. Hasler, B. A. Minch, C. Diorio, and C. A. Mead. An autozeroing floating-gate amplifier. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 2000. in Press.

[12] Paul Hasler. Continuous-time feedback in floating-gate MOS circuits. *IEEE Transactions on Circuits and Systems*, in Press.

[13] Paul Hasler, Chris Diorio, and Bradley A. Minch. A four-quadrant floating-gate synapse. In *IEEE International Symposium on Circuits and Systems*, Monterey, CA, 1998.

[14] Paul Hasler and Jeff Dugger. Correlation learning rule in floating-gate pFET synapses. In *IEEE International Symposium on Circuits and Systems*, volume V, pages 387–400, Orlando, FL, 1999.

[15] Paul Hasler and Jeff Dugger. Correlation learning rule in floating-gate pFET synapses. *IEEE Transactions on Circuits and Systems II*, in Press.

[16] Paul Hasler, Bradley A. Minch, and Chris Diorio. Floating-gate devices: They are not just for digital memories anymore. In *IEEE International Symposium on Circuits and Systems*, volume II, pages 399–391, Orlando, Florida, 1999.

[17] Paul Hasler, Theron Stanford, Bradley A. Minch, and Chris Diorio. An autozeroing floating-gate second-order section. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 351–354, Monterey, CA, 1998.

[18] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991.

[19] M.A. Jabri and B.G. Flower. Weight pertubation: An optimal achitecture and learning technique for analog vlsi feedforward and recurrent multilayer networks. *IEEE Transactions on Neural Networks*, 3:154–157, 1992.

[20] Matt Kucic, AiChen Low, Paul Hasler, and Joe Neff. A programmable continuous-time floating-gate fourier processor. *IEEE Transactions on Circuits and Systems II*, in Press.

[21] S.Y. Kung. *VLSI array processors*. Prentice Hall, Englewood Cliffs, N.J., 1988.

[22] A. Low and P. Hasler. Cadence-based simulation of floating-gate circuits using the EKV model. In *Midwest Symposium on Circuits and Systems*, 1999.

[23] R.F. Lyon and C. Mead. An analog electronic cochlea. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36:1119–1134, 1988.

[24] Carver Mead. *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, MA, 1989.

[25] L. Watts, D.A. Kerns, and R.F. Lyon. Improved implementation of the silicon cochlea. *IEEE Journal of Solid-State Circuits*, 27(5):692–700, 1992.