Integrated Floating-Gate Programming Environment for System-Level ICs

Sihwan Kim, Jennifer Hasler, *Senior Member, IEEE*, and Suma George

***Abstract*— We present the first integrated system to handle heterogeneously used and programmed floating-gate (FG) elements in a single modular approach. We focus on IC design, integration, characterization, and algorithmic development of an integrated FG programming system for a large-scale field-programmable analog array. We work through tunneling approaches to initialize the FG devices for precision programming, as well as hot-electron injection approaches for precise device programming.**

***Index Terms*— Floating-Gate Programming.**

I. Introduction

WE PRESENT the first IC system to program heterogeneous digital and analog floating-gate (FG) elements in a single modular approach. This paper focuses on the IC design, integration, characterization, and algorithms of an integrated FG programming system for system-level ICs, such as large-scale field-programmable analog arrays (FPAAs) [1], [2].

Fig. 1 shows the high-level view of the current FG programming approach implemented in this IC system. This programming system only requires that the IC user simply download the desired programming file, including code and FG devices to program into the IC. The on-chip components, such as DACs, ADCs, and microprocessor (μP), for programming are drop-in modules for a larger programmable system design.

This paper's approach builds an FG programming algorithm requiring only a few fixed-point computations. This paper requires a fresh look into FG programming techniques and algorithms, particularly, when the FG programming approach must work for highly diverse circuit functions. Precision programming of FG devices uses hot-electron injection due to the nearly ideal selectivity between the devices, while global initialization uses electron tunneling due to the relatively poor device selectivity [3], [4]. Target programming using hot-electron injection requires measuring the channel current (14-bit fixed-point value) and computing the voltages (7-bit fixed-point DAC value) for the next injection pulse.

These approaches are based on the previous component-level hardware structure [5], hot-electron injection [6], [7], and discussions on programming very low target currents [8]. The previous MATLAB-based algorithms required extensive floating-point computations. Specialized FG circuit programming, such as voltage sources [9]–[16], or standard digital memories [17]–[22] are both far too constraining for a heterogeneous array of computations; nondigital memory approaches fall short of user-friendly interfaces (e.g., downloading a device switch list). This paper experimentally shows the measured results unless otherwise mentioned.

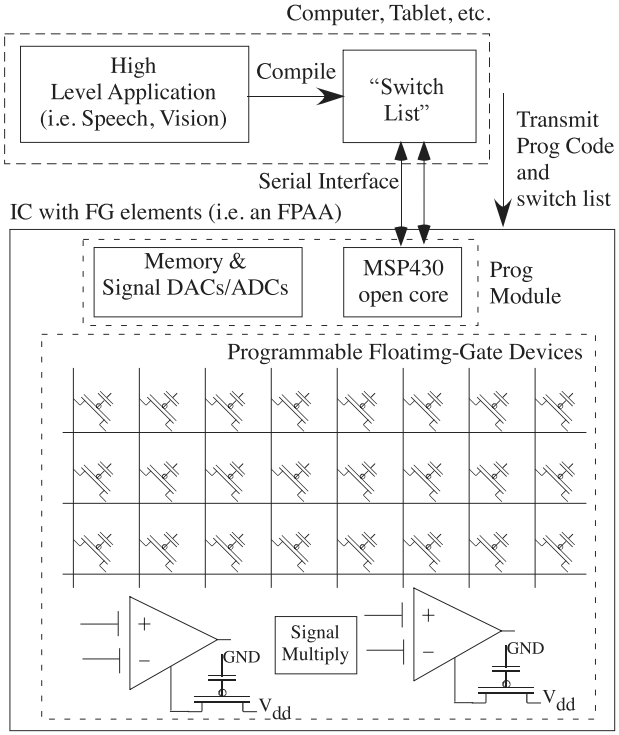


Fig. 1. FG programming approach we describe in this paper, enabling a direct download, as typical of digital programming (i.e., SRAM chains for FPGAs) has programmed switch and parameter information. This process empowers through complete on-chip FG analog and digital targeted programming to a heterogeneous set of FG circuits and computing devices. The programming structure uses on-chip DACs, ADCs, and a 16-bit open-source processor available for postprogramming computation. From the user's perspective, after their design has been compiled, like on a large-scale FPAA, the resulting programming operation looks like a typical data-download operation to an embedded device.

II. FG Programming Infrastructure

We first discuss the infrastructure and programming framework before jumping into the physics and programming discussions. Fig. 2 shows our standard for accessing the FG devices for programming. We show a representative structure, including switches and active devices, typical of a computational analog block in a large-scale FPAA device [25]. For programming, the entire circuitry gets reconfigured into a single crossbar array. We program in a crossbar array because hot-electron injection is a product of the current in the transistor channel and the voltage between the drain and channel potentials ($\approx$near source voltage); by only allowing current for a particular column and by only allowing a

Manuscript received April 6, 2015; revised July 21, 2015 and October 2, 2015; accepted November 5, 2015.

The authors are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332-250 USA (e-mail: jennifer.hasler@ece.gatech.edu).

Digital Object Identifier 10.1109/TVLSI.2015.2504118

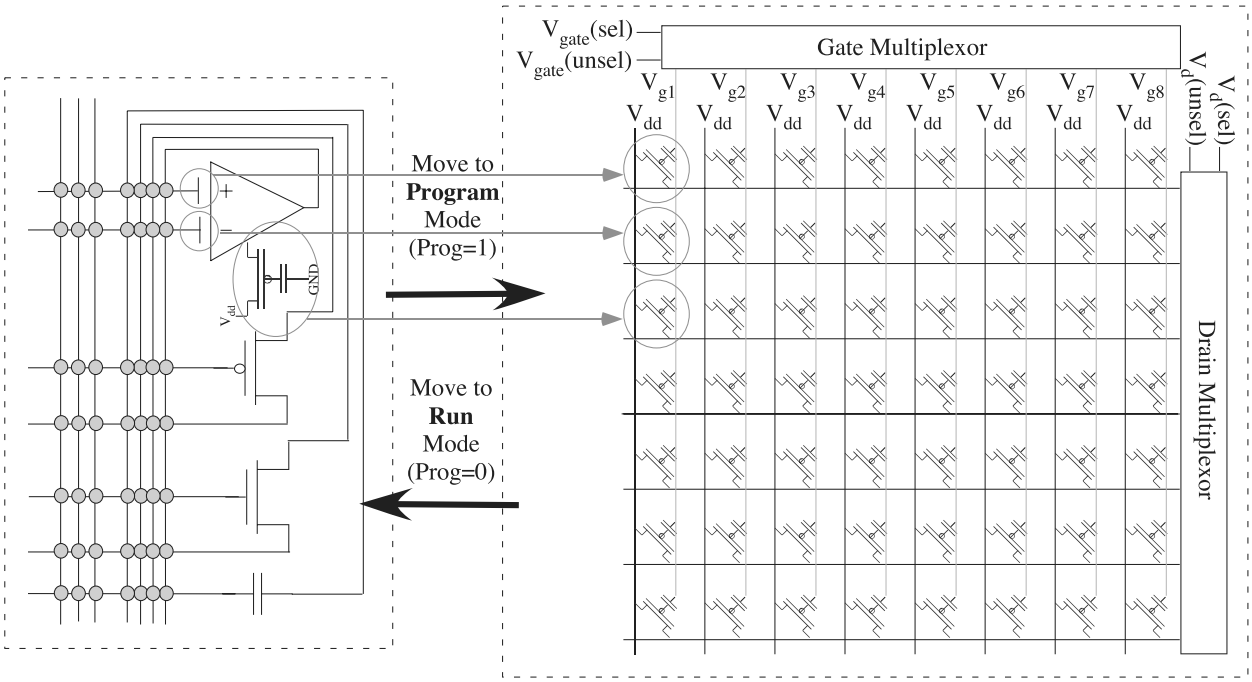


Fig. 2. When using a heterogeneous array of FG devices, we require that all FG devices are reconfigured into a single crossbar array of FG devices. In this configuration, we can program using hot-electron injection with nearly perfect selectivity, because a device requires channel current and high voltage from the drain-to-channel potential. Sometimes, we have additional control structures to guarantee transistor currents are turned OFF when not selecting a column; for example, a switch programmed ON will have some current in the case when its V_g is at V_{dd}. Reconfigurability between the program mode and the run mode is essential to enabling programming of all potential FG elements. The significance of this crossbar array and the gate and drain selecting multiplexers is that we can talk about addressing, measuring, and programming a single device in the array; equivalent to a device separated from the rest of the array.

high voltage between the drain and source terminals on a particular row, we are assured that only the desired device is affected. As a result of the nearly ideal selectivity due to the AND process for hot-electron injection and resulting selectivity for each transistor's current–voltage response, this entire structure effectively collapses as a single FG pFET device with a selected gate and drain terminal. We globally erase and restore devices by electron tunneling effects, partially because of the limited selectivity of these two-terminal devices typical of most physical two-terminal devices. When we program a device, we can measure the device properties in as close a situation as desired to the actual operation, therefore minimizing the effect of parasitic elements degrading the resulting programming accuracy when we move from the program mode to the run mode.

Fig. 3 shows the board and on-chip infrastructure. Fig. 3(a) shows the board-level infrastructure for programming and accessing the FG programming structure, as well as the digital infrastructure for accessing the chip during the normal programming operation. Fig. 3(b) shows the high-level block diagram for the on-chip programming structure, which includes utilizing an open-source μP, embedded 16 k $\times$ 16 SRAM for program and data memory, as well as the memory-mapped registers which control the elements for programming. We see in Fig. 2 that we simply need to control the gate and drain voltage (through two separate DACs) and then measure the resulting device current, which is at the core of the structure. The accuracy of the gate and drain DACs is not directly correlated with the final programmed accuracy; the frequency and noise of the ADC, which is 14 bit, are directly related to the final programming accuracy.

III. On-Chip Integrated FG Programming

Our FG programming relies on a combination of electron tunneling for erasing and resetting FG devices and hot-electron injection for programming FG devices. Fig. 4 shows the FG programming framework, with the resulting tunneling and injection steps. Over Sections III-A–III-D, we will discuss the global erasing and initialization (reverse tunneling) steps, FG recovery by injection, FG rough programming through open-loop injection, and final short step of FG fine programming through predictive FG calculations.

A. Global Array Erasure and Initialization

Erasing a block requires raising an entire block of tunneling junction voltages (V_{tun}) to a sufficiently high voltage to move toward a high FG voltage, which in turn results in a low channel current (i.e., fA in a pFET device) while still allowing programming to the desired target location. We initially tunnel all the FG values to a high voltage that ensures a pFET device has no channel current, and then we perform a reverse-tunneling operation to bring FG voltages to a small, but negligible, pFET current for a gate selection voltage at 0 V. Electron tunneling for erasing blocks is a common approach for erasing FG devices [3]–[7]. The 12 V charge-pump IC is only operational during the tunneling erase operation. One could choose to tunnel the voltage just enough to reach a sufficiently low current and measure a few representative currents, expecting that all the devices are erased. Unfortunately, tunneling current measurements for identical operating conditions show a variability of 2–3 (or more), resulting in exponentially different rates between the devices. Therefore,

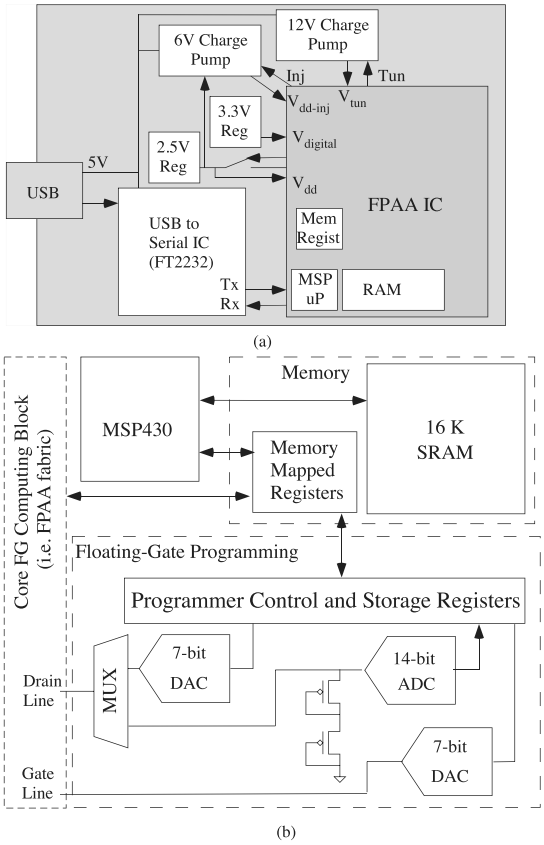


Fig. 3. Integrated infrastructure blocks, including test board, high-level IC schematic for programming, and detailed programming schematic. (a) System interface block diagram used for programming a representative FPAA device with the on-chip open-source MSP430 μP and memory. The primary off-chip infrastructure is μP IC-controlled high-voltage power handling (12 and 6 V charge-pump ICs); these components were left off-chip to minimize the IC design risk. A USB to serial converter IC was chosen to interface to μP. (b) Core on-chip circuit infrastructure used for μP-based programming arrays of FG devices. The μP and integrated 16 k $\times$ 16 SRAM block programming FG devices through a sequence of memory-mapped registers for the DACs supplying the gate and drain voltages, the FG current measurement structure using a ramp ADC, and two pFET transistors to convert from the current to the voltage. We estimate the processor requires approximately 200 pJ per instruction, including the local memory access.

without measuring all the devices, we tunnel enough so that each device's FG voltage is sufficiently high.

Reverse tunneling turns the polarities around on the tunneling junction to bring the resulting currents back toward a small but reasonable current for injection with less device mismatch than tunneling. The reverse tunneling phase requires lower voltages; for the 350-nm IC process, voltages between 0 and 6 V are used, resulting in lower charge across the tunnel oxides. The resulting code for implementing these two phases only requires applying the desired voltages, waiting a particular timeframe (allowing the processor to shut down or download programming instructions), and resetting voltages to the normal operating condition.

B. First Injection Step: FG Recovery

Once we have cleared all of the FG devices (FG voltage sufficiently high), we begin the process of programming FG devices that have nonnegligible current. FG pFET devices that we do not program will stay in accumulation and pull negligible levels (<1 pA) of current, even for scaled-down devices. Furthermore, our FG devices in the run mode are biased with V_g at roughly 0.6 V, enabling the algorithm to observe lower current values by measuring current at V_g at 0 V. For the IC used for measurements, we had a constant leakage current, due to the reverse-bias source–drain junction currents near 1 nA, thereby enabling current measurements in the 10–30-pA range even with this high leakage current. For a switch FG device, we typically measure 30 pA of current for V_g at 0.6 V but 1 nA for V_g at 0 V; for devices with larger FG capacitive coupling, this effect is even stronger.

The initial process simply looks for a significant channel current (i.e., 20–30 nA) when measuring current at $V_g = 0$ V for a switch element corresponding to 1 nA for $V_g = 0.6$ V; levels for a significant channel current differ for different groups of FG devices with different capacitive couplings. Furthermore, we have a programming sequence in parallel to what is shown in Fig. 4 for lower currents (<1 nA for switches) where we just simply inject until we have roughly 1 nA of current for $V_g = 0$ V, enabling targeted currents between 30 pA and 1 nA as needed. One can modify the drain voltage for the pulses to move the current values closer.

C. Approximate FG Programming by Injection

Target programming typically requires measuring channel current, comparing it with the desired current, performing a range of calculations for the conditions (i.e., drain voltage) during the next injection pulse, and repeating until it sufficiently converged. Previously, these calculations were rather complex, particularly, for a fixed-point embedded processing environment, giving an opportunity to reformulate this approach to fit better with fixed-point arithmetic.

We start by describing our measurement of the channel current at a compressed FG voltage through the 14-bit ramp ADC, as shown in Fig. 5. We are measuring an FG pFET device through the drain current switched through the programming crossbar infrastructure. Measuring current typically requires a conversion from current to voltage, and we potentially require four-to-seven orders of magnitude in our measurement. Therefore, we need some form of compression; in this case, a better approach would be translating the current into a representation near the original voltage difference from the well voltage to the FG voltage. We use a pFET device with the drain voltage tied to the gate voltage with the special case of the well voltage tied to the source voltage for our measurement circuitry. Fig. 5 also shows the reduced circuit to look at the resulting relationship between the FG voltage and the resulting V_{out}. A straightforward, large signal analysis of this circuit (assuming matched devices) shows

$$V_{\text{out}} = 2(V_{dd} - V_{\text{fg}}). \tag{1}$$

Threshold voltage variations simply require adding terms to the resulting structure.

We look next at the resulting-type S curves, taken from an FG switch element, looking at this V_{out}, which is directly related to V_{fg}. For a starting (subthreshold) drain current, injection decreases the FG voltage, increasing the drain current, further decreasing the FG voltage. The process slows down as the current moves to above-threshold operation

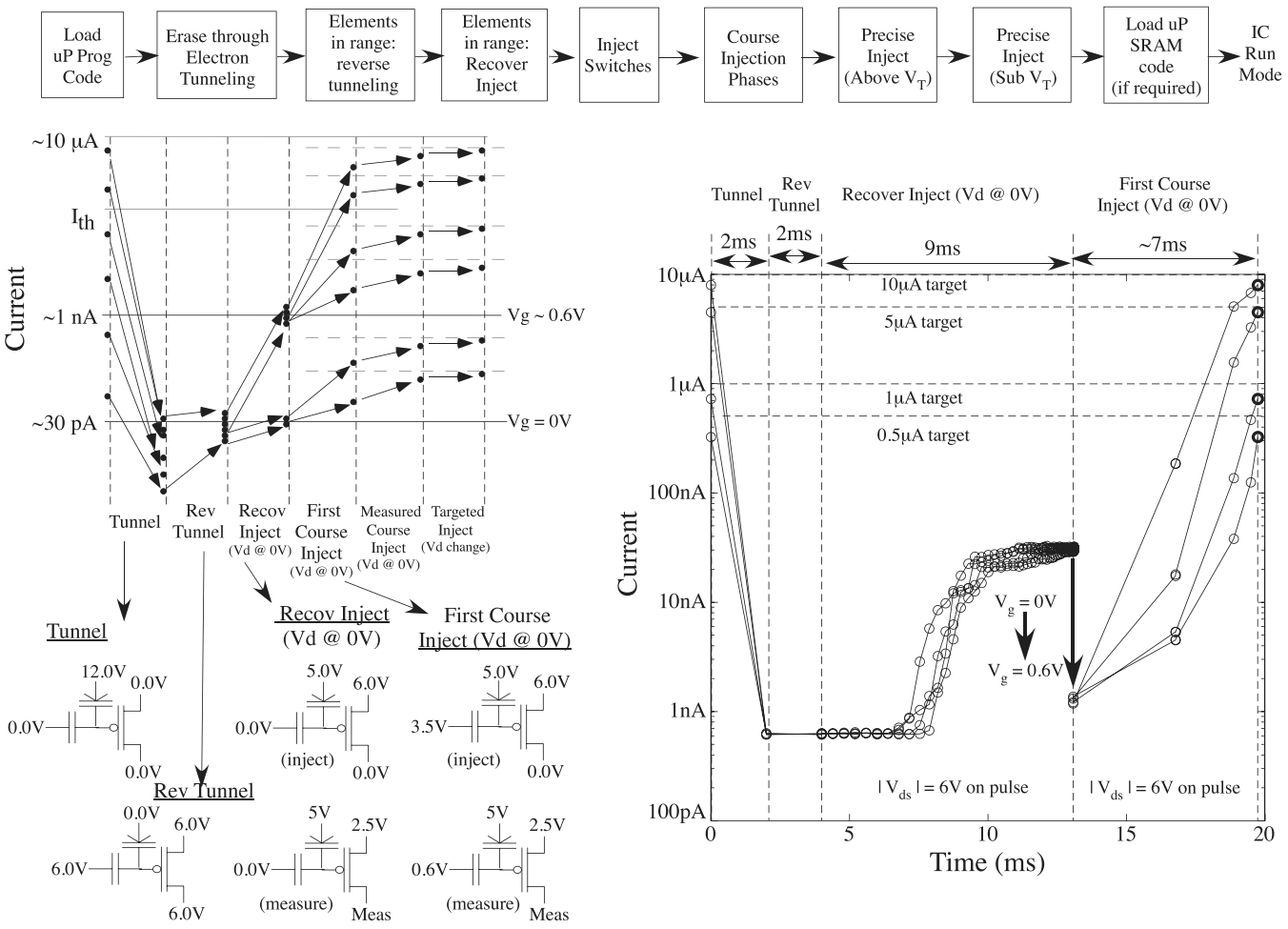


Fig. 4. Algorithm steps for programming an array of FG devices. Top: block diagram of the key required programming steps. The overall programming sequence requires putting the programming code into memory (for each operation), erasing and recovering devices through electron tunneling, reverse tunneling, and an additional recovery injection step, injecting FG switches where used, and then using a sequence of course and fine injection phases to reach the target for the subthreshold and above-threshold currents. In each case, switch data are loaded into the data SRAM block, programmed, and then additional pages of data are input to finish a phase if necessary. The final steps are loading the SRAM memory for the code desired (if necessary) for the IC operation and then switching the IC into the run mode. Left: graphical illustration of the FG programming steps following the procedure for a range of FG devices starting and ending at a desired target value. Right: experimental data showing actual trajectories of four FG devices through the first four programming steps; Fig. 5 will show the remaining precision injection measurements. The goal is to get devices close to target programming. Our devices operate in the run mode, biasing V_g at roughly 0.6 V; our measurements start with V_g at 0 V to measure devices with low currents and enable FG precision targeting for currents below the ≈1-nA measured leakage current from the array.

(defined as significantly greater than the threshold current or I_{th}), because as the FG voltage decreases, the increased drain current decreases the drain-to-channel voltage available for injection due to the additional voltage drop across the channel. This response, designated S curve, given the shape of the response.

Fig. 5 shows a measurement of V_{out} after an injection pulse (always for V_d to 0 V for a 6 V V_{ds}) versus the initial measured V_{out}; this process was repeated until reaching a near steady-state solution. For these measurements, we used a pulsewidth of 10 μs. We measure the output through the ADC, which follows the linear relationship:

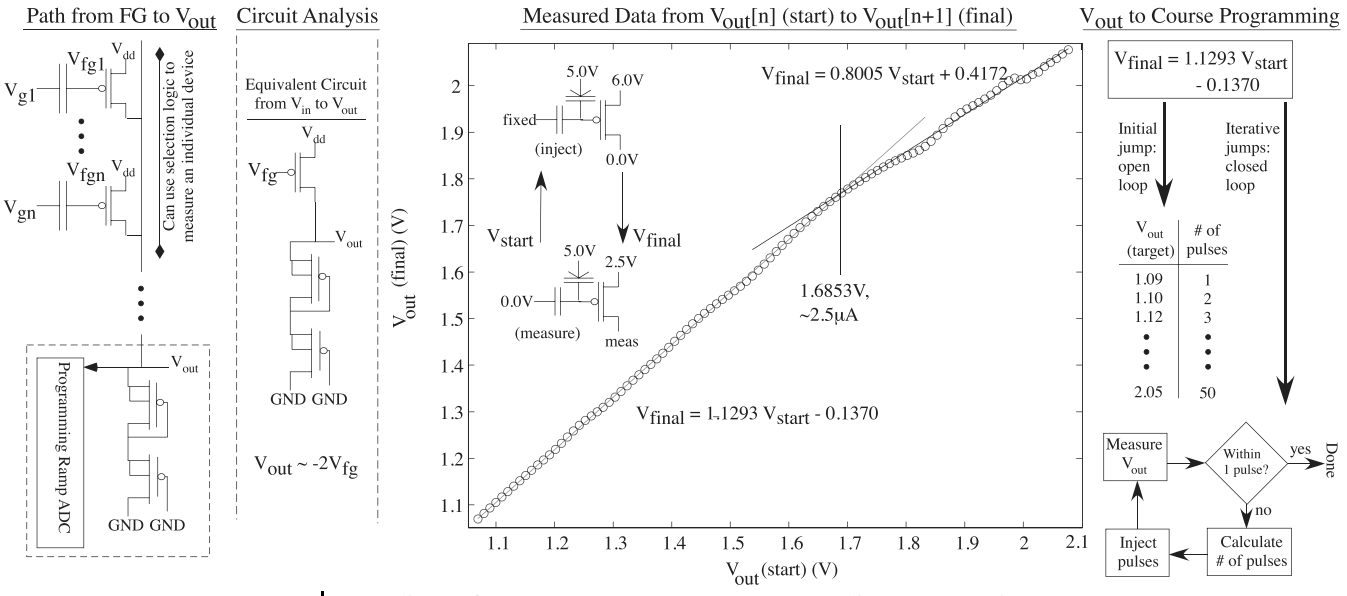
$$V_{\text{out}}[n] = 0.0001602a[n] + 0.3490 \tag{2}$$

where a[n] is the 14-bit integer code (0 through 16 383) measured in the ADC (we can keep all the codes within the 14-bit code on the 16-bit processor). Fig. 5 also shows two straight-line curve fits to the resulting data, as well as the table for the equation both in V_{out} and a, allowing simple, fixed-point computations for targeted programming.

Typically, we use a single pulse time width T_{inj} for every pulse, although the timing could be modified where desired. Therefore, the better the computational model, the fewer the number of pulses, and the shorter the resulting programming time (assuming the computation is fast). The fundamental model for hot-electron injection current (I_{inj}), using transistors operating with the subthreshold or near subthreshold bias currents [23], [24]

$$I_{\text{inj}} = I_s e^{f(V_{\text{fg}}, \Phi_{\text{dc}})} \approx I_{\text{inj0}} \left(\frac{I_s}{I_{\text{so}}}\right)^{\alpha} e^{\Phi_{\text{dc}}/V_{\text{inj}}} \tag{3}$$

where V_{inj} represents the one parameter for the $f(\cdot,\cdot)$ linearization. with all voltages relative to its V_{dd}. At each step, we should be exponentially decreasing the percentage change needed for the target, improving 1 bit of accuracy per iteration. Many combinations of V_g and V_d schemes are possible for the algorithm. The S curves have exponential growth in FG voltage



	V_{out} linear fit	a_{out} linear expression
Lower	V_{out}[n+1] = 1.1293 V_{out}[n] - 0.1370	a[n+1] = 1.1293 a[n] - 573
Upper	V_{out}[n+1] = 0.8005 V_{out}[n] + 0.4172	a[n+1] = 0.8005 a[n] + 2170

Fig. 5. Movement from the basic FG circuit array elements to course programming algorithm. Path from FG to V_{out}: crossbar network of indirect FG devices communicates through the drain current outputing a voltage through two pFET devices directly related to the FG voltage. Each FG device must be measured as well as enabled for hot-electron injection. Circuit analysis: equivalent circuit from the FG voltage (V_{fg}) to the output voltage (V_{out}). Using the same size (or similar size) pFET devices (V_{sw} = 0) implies that transistor k match, resulting in $\Delta V_{out} / \Delta V_{fg} = 2$. Measured data from V_{out}[n] (start) to $V_{\text{out}}[n+1]$ (final): measurement of the final value after injection for V_{out} on the current measurement versus the initial value before injection for V_{out} on the current measurement for maximum injection drain voltage pulse (6 V). Two straight-line curves, modeling the subthreshold part of the regime and the above-threshold part of the regime, enable the fixed-point computation for the updates for the on-chip μP with a simple fixed-point multiply and addition operations. V_{out} to course programming: using this data, the first initial injection jumps require simple operations (that can be stored in a table) due to the linear modeling, making an open-loop jump for a particular number of pulses or pulsewidth, based on the target value, as well as for iterative jumps based on V_{out} measurement to get the FG voltage within a single pulse spacing. Lower equations: resulting extracted (and used) V_{out} equations expressed both in the measured voltage, as well as in the measured 14-bit ADC codeword (a[n]).

(exponentially growing from an unstable equilibrium) for the subthreshold and near-threshold measurements and exponential convergence (exponentially decreasing toward a stable equilibrium) for higher above-threshold measurements.

From this framework, the next stages for the programming algorithm include a first injection phase, then a measured injection phase to get results within one pulse step, ready for the last sequences of fine programming. Fig. 4 shows the results of four representative trajectories where target levels are 0.5, 1, 5, and 10 μA. Since the starting drain current and measured V_{out} are roughly the same ($\approx$1–2 nA), and we have good matching between these parameters on an IC, we use a first injection pulse to approximate but not overshoot the target level. From the extracted linear curve, we can make a table (shown in Fig. 5) of the target value as a function of the number of pulses on a given device. A longer pulsewidth is roughly equivalent to multiple pulses of the same total time.

The table is not large, since in 29 steps, we reach the cross over point, and in 48 pulses, we are at the top of the second curve, resulting in under 500 μs for this open-loop programming step; a full 14-bit measurement, using a typical 25-MHz clock, takes roughly 1 ms to complete, so these injection measurements, even at 6 V, are shorter than a full measurement. These approaches are not limited by the speed of the μP. This first step does not require an additional measurement, reducing the programming time.

Next, μP calculates the number of pulses, based on measurement, to reach within one pulse of the target without overshooting the device. Effectively, if we had zero mismatch in the array, this step would be unnecessary, but we use this step to get devices within one injection step even with potential device mismatches. We will repeat this step as needed.

D. Precise Targeted FG Injection Programming

Our programming approach starts by measuring the desired device current, comparing that result with the desired target result, and computing the desired drain and/or gate voltages used to reach the desired target without overshooting the desired result. After the system applies the programming pulse, it proceeds to measure the new device current and repeats until sufficient accuracy has been achieved.

Targeted programming in one sense is the one aspect in this paper that leans heavily on the previous history of FG programming algorithms, including targeted subthreshold and near-threshold devices [7], adaptive targeting of subthreshold currents [6], and early efforts using on-chip ADC for current measurement [5]. In another sense, our approach for this phase of programming takes a different turn by constraining/considering FG targeted programming using the fixed-point, reduced arithmetic using lower precision DACs than the precision of the targeted value while still obtaining the required high accuracy on a single FG device.

Fig. 6 shows representative measured subthreshold and above-threshold target injection programming. μP computes resulting error between the target value and the measurement value after a measurement. That error estimates the optimal drain voltage, without overshoot, for the next injection pulse. The resulting processing requires first finding the bit where we have the next significant error, then finding the resulting drain DAC code to minimize the error for that bit, using the resulting few bits of the DAC code that are computed related to the next few bits of the error approach. Our approach measures, pulses all devices, and then repeats until error is minimized to minimize the effect of voltage transients after the injection supply ramps up to, and down from, the 6 V supply.

Fundamentally, the task is controlling the injection process through a sequence of measurements and pulses of fixed time (T_{inj}), to hit the desired target in as few pulses as possible without overshooting the target. The previous process moves V_{fg} reasonably near its desired target (i.e., within 100 mV), minimizing the amount of FG dependence when modeling injection current, and ignoring the dependence of I_{inj} on V_{fg} (subthreshold and above V_{T0} currents). Pulses are modeled as

$$\Delta V_{\text{fg}}[n+1] \approx \Delta V_{\text{fg}}[n] - \frac{I_{\text{inj}0} T_{\text{inj}}}{C_T} e^{-\Delta V_d / V_{\text{inj}}} \tag{4}$$

where $V_{\text{fg}}[n]$ is the FG voltage at programming iteration n, T_{inj} for injection pulse, and we now set I_{inj0} to this phase injection current at $\Delta V_d = 0$.

Drain voltage results in an exponential factor for the V_{fg} change per iteration, enabling the system to improve on MSB as well as LSB through a compressed, linear drain voltage. Fig. 6(d) shows a shift of nearly a factor of 1000 in injection current change, resulting in a factor of 1000 change in the FG voltage, over a range of 1 V change in the drain voltage, due to the exponential dependence between V_d and injection current. An increase of 3.5 codes results in roughly a factor of two, corresponding to correcting the next least significant bit for the resulting error. Typically, we will tabulate the resulting error size and V_d required to get sufficiently close (i.e., at least one additional binary bit of resolution) to the next iteration. The exponential function between the drain voltage and the resulting FG charge/current change enables a wide dynamic range of FG changes with a linear voltage range. This approach does not require high-precision DAC components; the drain voltage (injection pulse) and gate voltage are 7-bit DACs.

Although one could choose a drain voltage pulse for optimal convergence, we give some safety margin resulting from mismatches in the V_{inj} parameter between the FG devices; systematic characterization is beyond the scope of this paper but the subject of future discussions. Potentially, adaptive modeling for V_{inj} during programming could be used, as initially proposed in [6], to decrease the total number of resulting pulses for successful programming.

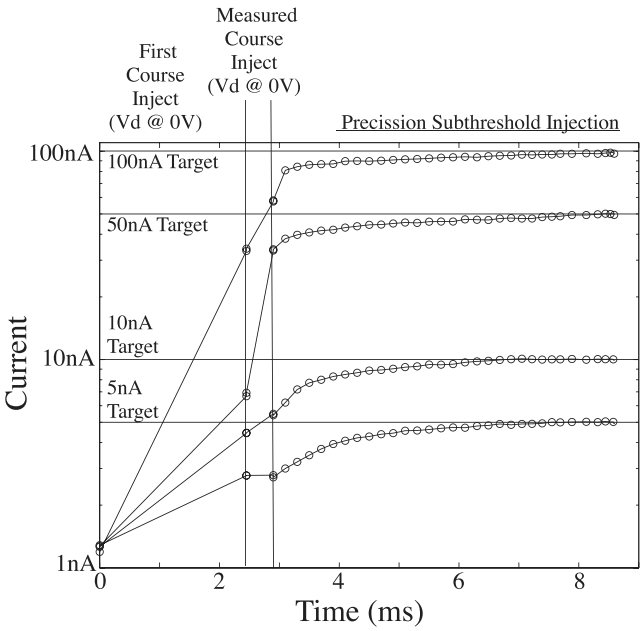
A practical issue with this current DAC implementation is a nonlinear step for low voltages. We have a code for 0 V, but the next code is typically 0.4–0.5 V with nearly linear spacing for higher values. For a typical value V_{inj} of 150 mV, a change of 450 mV reduces the effect by a factor of roughly 20; having a pulsewidth ten times greater (10 μs $\rightarrow$ 100 μs) roughly gets to the next least significant bit to correct. Therefore, the algorithms have two cases, one for a full size drain-to-source injection pulse and another for a changing drain-to-source injection pulse with a wider pulsewidth.

The measuring ADC (14 bit) is the component that requires accuracy to program the FG to a precise value. For targeted programming, one key issue not addressed at this point is the accuracy of the resulting measurement, in particular, what to predict in terms of the noise, the shape of the noise, and if we average to improve the resulting measured accuracy. The theoretical limitation in accuracy comes from using a 14-bit ADC over the (roughly) 2 V output voltage range, resulting from 1 V shift in FG voltage for the measured device. The LSB for the 14-bit ADC results in 61 μV in FG voltage accuracy, resulting in 0.166% error for the subthreshold currents ($\kappa = 0.7$). A single electron changes the FG voltage 10 μV for a total FG capacitance of 16 fF; a larger total FG capacitance results in a proportionally lower voltage per electron.

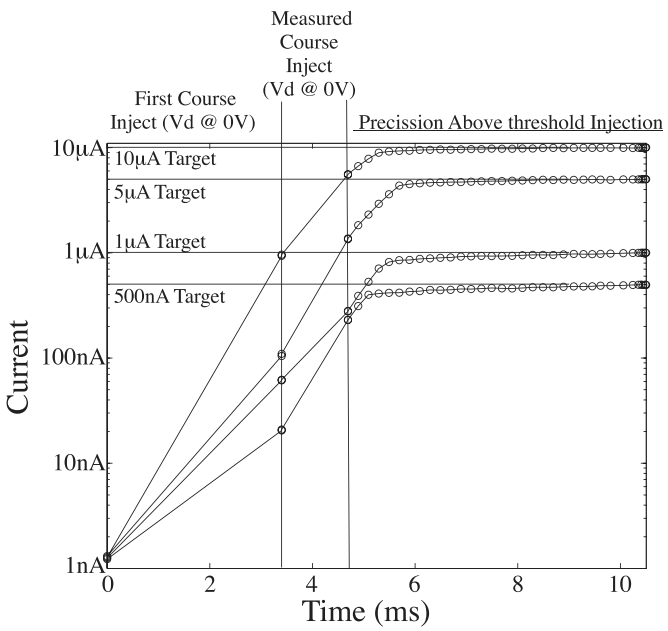
These results show that V_{out} would not be the source of noise, but the noise source tends to be a combination of comparator noise, input ramp noise, and clock jitter into the resulting counter. The measured resulting noise at V_{out} through the 14-bit ADC for bias currents throughout the entire measured current range shows that the noise is roughly five to seven codes for a standard deviation (two codes as a function of USB power supply noise, lower noise at lower current), the noise is a weak function of the resulting current (increases a factor of 2 over 4+ orders of magnitude in current), even though the resulting three-transistor circuit bandwidth is not constant, and the resulting measured noise spectrum is flat, characteristic of thermal noise. Since the noise follows a thermal noise spectrum, we expect that we can average the values to get further accuracy. We find the measured noise occurs at roughly the 10-bit/11-bit measurement level (<1–2% for the subthreshold currents); therefore, we need to employ averaging to get accurate measurements for the last 3 bits of accuracy. Averaging four samples results in one additional bit of accuracy; the final measurements use 16 samples.

IV. Summary of FG Programming: Performance and Historical Perspective

We focused on the IC design, integration, characterization, and algorithmic development for an integrated FG programming system. We used a recent FPAA IC enabled with an on-chip processor to experimentally demonstrate this system [25]. We use hot-electron injection for precision programming of FG devices due to the nearly ideal selectivity between the devices, whereas we use electron tunneling for global initialization because of their relatively poor device selectivity. We discussed the foundations of FG devices and programming, including electron tunneling and hot-electron injection. We presented the methods, approaches, and infrastructure, both on-chip and on-board, for FG programming. We discussed this programming algorithm from erasing, setting up FG charge to be ready for programming, methodology and approach for course programming steps, and methodology and approach for precision targeting



(a)



(b)

I_{target}	10μA	5μA	1μA	500nA	100nA	50nA	10nA	5nA	Average
$I_{measure}$-I_{target}	0.89%	0.43%	1.02%	0.79%	0.29%	0.98%	1.00%	1.00%	0.80%
σ_I / I_{target}	0.0024	0.0025	0.0042	0.0041	0.0046	0.0029	0.0019	0.0033	0.0032

(c)

Hex	FE	FC	FA	F8	F0	EA	E2	DC	D4	C6	B8
Vd(V)	0.48V	0.51V	0.54V	0.57V	0.69V	0.78V	0.90V	0.99V	1.11V	1.32	1.53
% change	40.7	33.4	27.3	22.4	10.1	5.51	2.48	1.36	0.61	0.151	0.032

(d)

Fig. 6. Precision programming measurements for the subthreshold and above-threshold currents, showing representative course injection, measured course injection (single V_d), and precision target injection. (a) Four target current measurements for the subthreshold currents. (b) Four representative current measurements for the above-threshold currents. We programmed several different FG devices to these currents and summarize the average measurement error as well as the standard deviation over the average current from these multiple measurements using this entire programming infrastructure. Since injection current and the change in FG voltage are an exponential function of drain (V_d) voltage, the resulting drain pulse is approximated by pulling apart the difference of target and measured values. (c) Percentage accuracy for target programming for a range of currents, including the standard deviation after performing multiple injection target programming rounds. (d) Values for relevant drain DAC codes, and their change on the injection current through a change in V_d during the programming injection pulse. A constant V_{inj} of 150 mV (typical value) is assumed for these calculations; in practice, there is some weak curvature to these calculations over this range of evaluation. A change of 4 (2 bits) occurs for the transition through seven codes or 210 mV.

steps, all based on the opportunities afforded to us through the current infrastructure.

Table I summarizes the memory requirements, pulse, and computation time for programming steps, and resulting energy estimate required for performing these steps. Table II shows a typical switch list for the coordinates of the FG devices to be programmed, their resulting current to be programmed (where relevant) or switch type, as well as the type of FG device (i.e., particular C_T) used. We programmed many FG devices in a large array (over 200 000 devices) using these definitions that can be compiled from higher level tools. Fig. 7 shows the die photographs of the key components for programming, including the programmer module (DACs, ADCs, and so on), the open-source MSP430 μP, and the resulting 16 k $\times$ 16 SRAM block used for the data and memory. The biggest issue in terms of size and power dissipation during programming is due to SRAM size and communication.

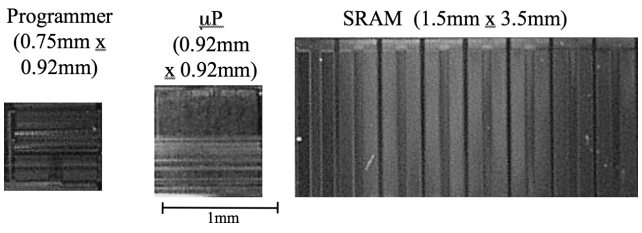


Fig. 7. Die photo of the programmer infrastructure, μP, and SRAM memory (16 k $\times$ 16) in a 350-nm CMOS process. The area of the SRAM memory is much larger than the other two blocks.

The time estimates do not include the required measurement time, roughly 7-ms per measurement requiring 0.5 s to program a single targeted device, which in the current

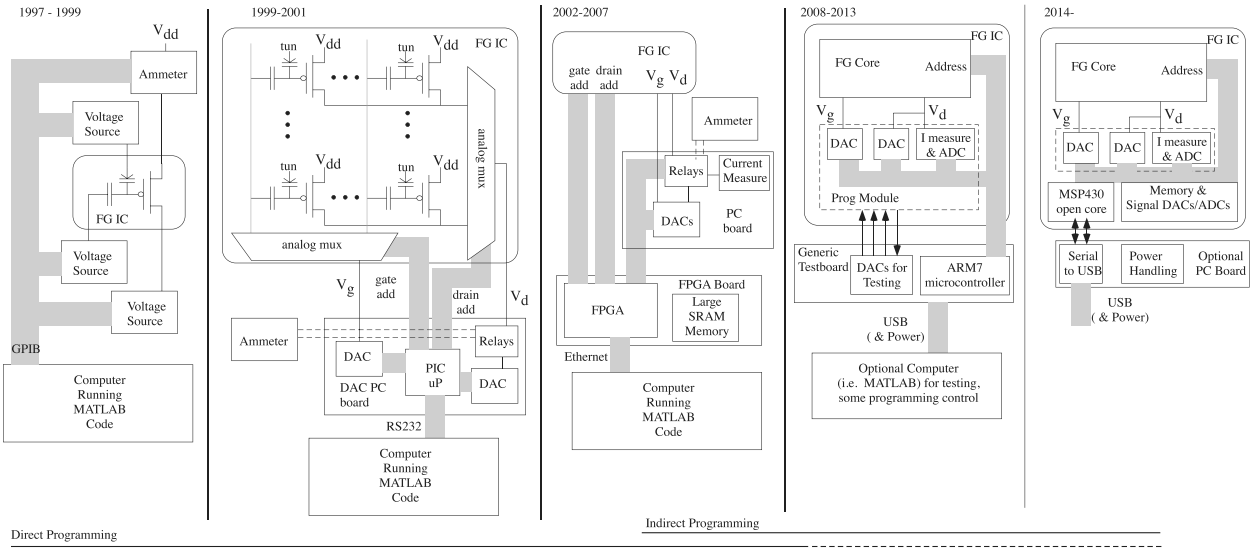


Fig. 8. Pictorial history of FG circuit programming algorithms developed at Georgia Tech, where we show the complexity for on-chip computation, as well as board and overall infrastructure required for a programming step. FG devices started from the original single transistor synapse learning device [4], and developed into a range of FG circuit applications (one summary in [3]). The approaches start with external bench-top instruments programming a few FG devices [3], to an interface PC board with simple interface to allow computer control [6], to a PC board + field-programmable gate array (FPGA) board solution to perform the programming infrastructure along with MATLAB programming control [7], to having some of the circuit infrastructure on board as a programmer module [5], with an on-board microcontroller with MATLAB control of the programming algorithm, and to final our current integrated solution with the entire programming control and infrastructure entirely on the IC requiring no μP control over the process other than writing the proper file format to the IC. At each level, we roughly increased the number of FGs routinely programmed by an order of magnitude, going from 10 to 100, to 1 k to 10 k to our current structure routinely programming arrays of 100 k or larger, such as our current family of FPAA devices, further enabling larger and larger system application solutions. Our approach also enables using both direct and indirect programming, whether we have nFET or pFET devices.

TABLE I

TABLE OF PARAMETERS FOR μP MEMORY SIZE, TYPICAL COMPUTATION TIME, AND RESULTING ENERGY ESTIMATE FOR THAT FULL OPERATION

Functional Block	Memory Size (Bytes)	Time	Energy Estimate
Tunnel Erase + + Reverse Tun	1204	2ms + 2ms (array)	20μJ
Switch Program	2391	10ms	50μJ
Recover Inject	2166	10ms	50μJ
First Course Program	2190	2-7ms	$\approx$ 25μJ
Measured Course Program	2329	$\approx$ 1ms	5μJ
Targeted Programming	2460	6-7ms	30μJ

In all cases, the energy cost is dwarfed by the energy required for the μP, and require a small fraction of the 16k byte program memory, using almost all of the 16k byte data memory for loading switch data.

TABLE II

PART OF TYPICAL SWITCH LIST

Row	Col	I_{prog}	C_T
344	585	0	0
319	858	1	0
263	67	2	0
339	45	1.00e-07	1
338	993	1.00e-06	2
339	961	1.00e-09	3
473	977	0.00001	4
317	945	10e-09	6

The first two values are the list of coordinates, in row and column, in the FG crossbar array for programming. If the third value is an integer, we have a switch to program *ON*, where the integer will indicate a particular switch type (0 being the typical default FG switch). The fourth value selects one of multiple characterized FG C_T values.

implementation consumes most of the resulting programming time. Practically, this limitation means we use injection supply at 6 V. The V_{out} measurement requires less than 1 ms for currents less than 1 nA; therefore, this component is not a limitation, and the measurement can be further accelerated as needed. The resulting issue is measurement through a single 14-bit ramp ADC used in our IC required to measure the full resolution using a 10-MHz down sampled clock (from 20–25-MHz processor clock). We see an opportunity in building more intelligence into the ADC measurement based on variations on the ramp function. For example, we can use the ADC as a threshold when we need a course target voltage. As another example, we can see setting the ramp between the 6- and 8-bit linear DAC values to enable faster precision measurement, increasing the ramp measurement by a factor of 64 or further. We see these issues as being the next level question for programming large-scale FG arrays. Furthermore, as devices scale to millions of devices, we can visualize using parallel measure and program components, organized for blocks of FG devices used to minimize the programming time. The ramp ADC structure could easily become a parallel bank of ADCs, and the small amount of μP assembly code could allow for multiple device operation.

The programming approach shown in Fig. 2 focused on enabling arrays of arbitrary FG devices, creating the minimal number of rules/constraints for this implementation, and thereby separating the design of FG circuits and systems from the FG programming issues. Large FG systems require automatic programming of arbitrary circuits, particularly for systems with high configurability that have to be enabled by higher level tools. These approaches required going past the basic FG transistor concepts, operation, and programming physics, including requiring hot-electron injection for precision programming with electron tunneling for erasure (see [3], [4]) to developing a structure that all the FG devices can be reconfigured into a 2-D crossbar array for programming. This novel perspective enabled a range of developments, from external bench-top instrument programming [3], to a PC board controlled programming [6], to a PC board + FPGA board solution programming [7], to an on-chip module +

PC board programming [5], and finally resulting to the fully integrated solution presented in this paper.

This paper presents the first integrated system to handle heterogeneously used and programmed FG elements in a single modular approach. Fig. 8 shows the progression from the beginning of the programming approach (shown in Fig. 1) of FG arrays, which has been a systematic march toward on-chip integration, while in parallel continuing to build structures enabling on-chip analog and digital signal processing, including configurable architectures. In all the cases, we have the capability to program a general FG array, therefore requiring no predefined constraints except for the basic configuration rules during programming. The on-chip processor enables an embedded programming approach not done outside of MATLAB, unlike other previous approaches. Our technical approach builds on a novel, fixed-point, limited infrastructure, potentially allowing a translation to verilog processing for a dedicated block, reducing the power required (as compared with μP for programming where required). In applications requiring μP, it often makes sense to utilize that resource directly. Low milliwatt programming power consumption is acceptable for USB downloading data into the IC.

Other approaches have built more custom designed programmer modules for a range of applications. One of the first developed was a specialized method for programming a range of voltage sources, called *epots* [9], for a benchtop user-friendly programming experience, requiring a significantly larger overall FG cell to perform this one function. A few additional improvements to this approach have been developed since [10], [11], [13], and are used in multiple applications [12]. Often, when a custom IC using FG devices is built, a custom programmer dedicated to the particular application [14]–[16], requiring FG circuit and programming expertise for every IC being developed (as opposed to the approach described here). Finally, one sees the ultimate custom application, nonvolatile digital memory, as well as the model for the general programming structure used in this discussion. A good overview of EEPROM/flash history was presented at ISSCC 2012 [22]. Current EEPROM devices already store 4 bits (16 levels) in a single transistor of 100 nm $\times$ 100-nm area in a 32-nm process [17], [18]. Recent data on EEPROM devices show commercially announced devices at 15 nm (Hynix, IEDM) and 19 nm (Toshiba/SanDisk [19], [20] and Samsung [21]) as well as the production of 32-nm devices.

ACKNOWLEDGMENT

The authors would like to thank S. Nease who worked on porting programming circuits into this current architecture.

REFERENCES

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

[2] R. B. Wunderlich, F. Adil, and P. Hasler, "Floating gate-based field programmable mixed-signal array," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 8, pp. 1496–1505, Aug. 2013.

[3] P. Hasler, B. A. Minch, and C. Diorio, "Adaptive circuits using pFET floating-gate devices," in *Proc. IEEE 20th Anniversary Conf. Adv. Res. VLSI*, Atlanta, GA, USA, Mar. 1999, pp. 215–229.

[4] P. Hasler, C. Diorio, B. A. Minch, and C. A. Mead, "Single transistor learning synapses," in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D. S. Touretzky, and T. K. Leen, Eds. Cambridge, MA, USA: MIT Press, 1994, pp. 817–824.

[5] A. Basu and P. E. Hasler, "A fully integrated architecture for fast and accurate programming of floating gates over six decades of current," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 6, pp. 953–962, Jun. 2011.

[6] M. Kucic, A. Low, P. Hasler, and J. Neff, "A programmable continuous-time floating-gate Fourier processor," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 1, pp. 90–99, Jan. 2001.

[7] A. Bandyopadhyay, G. J. Serrano, and P. Hasler, "Adaptive algorithm using hot-electron injection for programming analog computational memory elements within 0.2% of accuracy over 3.5 decades," *IEEE J. Solid-State Circuits*, vol. 41, no. 9, pp. 2107–2114, Sep. 2006.

[8] P. D. Smith, D. Graham, and P. Hasler, "A kappa projection algorithm (KPA) for programming to femtoampere currents in standard CMOS floating-gate elements," *Analog Integr. Circuits Signal Process.*, vol. 50, no. 1, pp. 83–91, 2008.

[9] R. R. Harrison, J. A. Bragg, P. Hasler, B. A. Minch, and S. P. Deweerth, "A CMOS programmable analog memory-cell array using floating-gate circuits," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 1, pp. 4–11, Jan. 2001.

[10] J. Lu and J. Holleman, "A floating-gate analog memory with bidirectional sigmoid updates in a standard digital process," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2013, pp. 1600–1603.

[11] C. Huang, P. Sarkar, and S. Chakrabartty, "Rail-to-rail, linear hot-electron injection programming of floating-gate voltage bias generators at 13-bit resolution," *IEEE J. Solid State Circuits*, vol. 46, no. 11, pp. 2685–2692, Nov. 2011.

[12] J. Lu, S. Young, I. Arel, and J. Holleman, "A 1 TOPS/W analog deep machine-learning engine with floating-gate storage in 0.13 μm CMOS," *IEEE J. Solid-State Circuits*, vol. 50, no. 1, pp. 270–281, Jan. 2015.

[13] L. Zhou and S. Chakrabartty, "A 7-transistor-per-cell, high-density analog storage array with 500 μV update accuracy and greater than 60 dB linearity," in *Proc. IEEE ISCAS*, Jun. 2014, pp. 1572–1575.

[14] M. Cohen and G. Cauwenberghs, "Floating-gate adaptation for focal-plane online nonuniformity correction," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 48, no. 1, pp. 83–89, Jan. 2001.

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

[16] M. Gu and S. Chakrabartty, "Subthreshold, varactor-driven CMOS floating-gate current memory array with less than 150-ppm/°K temperature sensitivity," *IEEE J. Solid-State Circuits*, vol. 47, no. 11, pp. 2846–2856, Nov. 2012.

[17] G. G. Marotta *et al.*, "A 3 bit/cell 32 Gb NAND flash memory at 34 nm with 6 MB/s program throughput and with dynamic 2 b/cell blocks configuration mode for a program throughput increase up to 13 MB/s," in *Proc. ISSCC*, Feb. 2010, pp. 444–445.

[18] Y. Li *et al.*, "A 16 Gb 3 b/cell NAND flash memory in 56 nm with 8 MB/s write rate," in *Proc. ISSCC*, Feb. 2008, pp. 506–507.

[19] N. Shibata *et al.*, "A 19 nm 112.8 mm^2 64 Gb multi-level flash memory with 400 Mb/s/pin 1.8 V toggle mode interface," in *Proc. ISSCC*, Feb. 2012, pp. 422–423.

[20] Y. Li *et al.*, "128 Gb 3 b/cell NAND flash memory in 19 nm technology with 18 MB/s write rate and 400 Mb/s toggle mode," in *Proc. ISSCC*, Feb. 2012, pp. 436–437.

[21] D. Lee *et al.*, "A 64 Gb 533 Mb/s DDR interface MLC NAND Flash in sub-20 nm technology," in *Proc. ISSCC*, Feb. 2012, pp. 430–432.

[22] E. Harari, "Flash memory—The great disruptor!" in *Proc. ISSCC*, Feb. 2012, pp. 10–15.

[23] P. Hasler, A. G. Andreou, C. Diorio, B. A. Minch, and C. A. Mead, "Impact ionization and hot-electron injection derived consistently from Boltzmann transport," *VLSI Design*, vol. 8, nos. 1–4, pp. 454–461, 1998.

[24] P. Hasler, A. Basu, and S. Koziol, "Above threshold pFET injection modeling intended for programming floating-gate systems," in *Proc. IEEE ISCAS*, May 2007, pp. 1557–1560.

[25] S. George *et al.*, "A programmable and configurable mixed-mode FPAA SoC," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, to be published.

Sihwan Kim, photograph and biography not available at the time of publication.

Jennifer Hasler (SM'02) photograph and biography not available at the time of publication.

Suma George, photograph and biography not available at the time of publication.