LOW POWER MIXED SIGNAL SYSTEM DESIGN ENVIRONMENT USING FLOATING-GATE FPAAS

A Dissertation
Presented to
The Academic Faculty

by

Sihwan Kim

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

Georgia Institute of Technology
May 2018

LOW POWER MIXED SIGNAL SYSTEM DESIGN ENVIRONMENT USING FLOATING-GATE FPAAS

Approved by:

Professor Jennifer Hasler, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Aaron D Lanterman
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Saibal Mukhopadhyay
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Scott Koziol
School of Engineering and Computer Science
Baylor University

Professor Sung Kyu Lim
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: April 2018

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

FLOATING-GATE DEVICES AND THE SYSTEM

Over recent decades, there have been notable advances in Field-Programmable Analog Arrays (FPAAs) using Floating-Gate (FG) devices. Research on FG devices has shown the capability of low-power analog computation and the possibility of neuromorphic applications, as well as modern FG FPAAs have increased the programming parameter density and the arrays' scale. However, users who design system-level applications do not have a solid design environment yet, compared to digital systems (e.g., FPGAs). Therefore, to empower a wider community for analog-digital mixed system designers, it is essential to establish a user friendly and extensible platform on both hardware and software.

The purpose of this research is to create a low-power mixed-signal system design environment using FG FPAAs. To achieve this, my research focused on implementing a compact hardware, developing algorithms and tools, and establishing a solid calibration flow. The first phase, implementing a compact hardware, involved developing FG SoC FPAAs including an FG programming infrastructure and built-in self test circuits, as well as designing smaller test boards to enable IoT embedded applications. Secondly, FG FPAA systems need to provide interfacing tools for application designers. This required a synthesis toolset generating a switch list from the user's design and an FG programming algorithm compatible with the SoC FPAAs. Lastly, a solid calibration flow solved the variation problem of each IC, which has been an issue in typical analog system designs. By coordinating the development of these three phases, it has been enabled to provide a powerful low-power system design environment, where system engineers consider analog circuits as black boxes, so they

can enjoy the design and implementation of higher level applications, as if they are designing digital systems.

CHAPTER II

SCALING OF FG DEVICES

Floating-Gate (FG) transistors, which have gates entirely surrounded by electrical insulator, have been playing an important role in digital computations as a non-volatile memory storage since it was originally reported in 1967 [1]. Flash memories, a class of Electrically Erasable ROMs (EEPROMs) using FG transistors, have established themselves in digital storage products such as memory cards, USB flash drives, and Solid-State Drives (SSD). To increase the performance and reduce the price of flash memories, recent research has been focusing on scaling down the size of the FG device [2], storing multiple bits information in a single FG transistor [3], and stacking layers to a vertical direction [4].

Current EEPROM devices already store 4 bits (16 levels) in a single transistor of 100nm x 100nm area in 32nm process [3, 5]. A good overview of EEPROM / Flash history was presented at ISSCC2012 [6]. Recent data on EEPROM devices shows commercially announced devices at 15nm (Hynix, IEDM) and 19nm (Toshiba / ScanDisk [7, 8] and Samsung [9]) as well as production of 32nm devices. From the current EEPROM progress, such devices are expected to migrate to 7nm and 11nm technology nodes; therefore the risk that the industry will not commercially produce a 10nm floating-gate device is very low.

The FG transistors have also been essential in analog and mixed mode computation, since the FG method was used for Field-Programmable Analog Array (FPAA) [10] [11] systems. FG devices provide programmability enabling the tuning of parameters/variables and mismatch compensation of transistors as well as reconfigurability enabling changes of topology, program, and data flow. These advantages

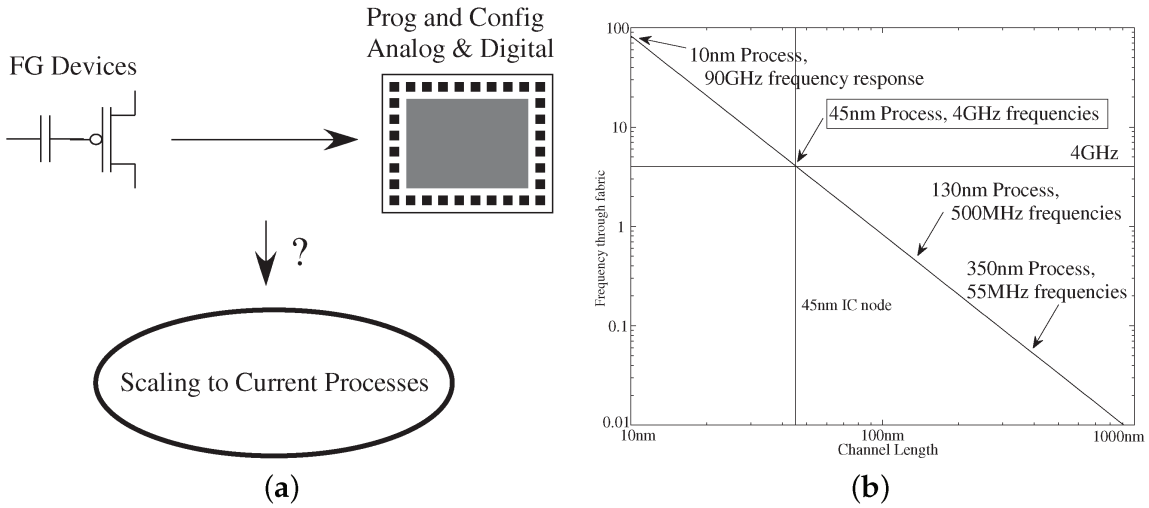


Figure 1: Scaling of Floating-Gate FG devices. (a) The question of scaling these devices to more modern processes (e.g. 130nm, 45nm CMOS). (b) Frequency response of FPAA architectures as a function of minimum channel length. The results come from FPAA architecture modeling, CMOS process modeling, and experimental data where available (350nm, 130nm, 40nm).

help system designers to reduce their time and efforts for the hardware development, where a typical design cycle requires a long time of process including circuit design, fabrication, packaging, and test platform design. It also helps designers to focus on higher level of applications such as always-ON contextaware processors, acoustics, vision, and robotics.

This chapter discusses scaling of Floating-Gate (FG) devices, and the resulting implication to larger systems, such as large-scale Field Programmable Analog Arrays (FPAA). Figure 1 shows our high level figure, connecting the properties of FG circuits and systems in one technology (e.g. 350nm CMOS), and predicting the behavior and advantages in smaller technologies. FG devices have been essential in demonstrating programmable and configurable analog and mixed mode computation, although typically at processes like 350nm CMOS. The question of scaling these devices to more modern processes (e.g. 130nm, 40nm CMOS), typical of other system ICs (e.g. FPGAs) remains, even though EEPROMs have moved to smaller and smaller linewidths($<$ 20nm gate length), and continued growth expected.

Figure 1 shows an important win of scaling FG devices is higher frequency response, say through an FPAA fabric, and often lower power consumption due to lower parasitic capacitances. Scaling of Floating-Gate (FG) devices is a key issue when working to improve the density as well as raising the density of FG based memories, computing in memory systems, and FPAA. For example, how will an FPAA's operating frequency improve as the IC technology process is scaled down. Figure 1b shows a modeling summary of the capability in frequency of a particular FPAA device architecture as a function of process geometry used. Although the initial FPAA devices, built in 350nm process, have achieved frequencies in the 50-100MHz range (i.e. [12]), scaled FPAA devices should enable significantly higher frequencies, enabling RF type signals at 40nm and smaller IC processes. Therefore the potential of scaled down devices, and the resulting computation, from a 350nm process down to a 40nm process requires investigating both experimentally and analytically the effects of a 40nm process.

2.1 350nm FG Devices as Baseline for Scaling Performance

This section addresses what is needed for a functional FG device, typical of a wide range of circuit applications [12–14], as well as how these processes are characterized. Figure 2 shows the fundamental plots to characterize the resulting FG devices, enabling an automated FG algorithm [15]. Figure 2a shows that the current-voltage relationship is programmed through stored FG charge, resulting in a programmable weighting factor (i.e. subthreshold) and/or a programmable threshold voltage (V_{T0}, i.e. above-threshold). Although one has a capacitive divider, we have typical current-voltage relationships for a single curve. Changing the FG charge moves to a different curve, either increasing it by electron tunneling, or decreasing it by hot-electron injection. The resulting charge results in a voltage change on the FG node by the total capacitance at the FG node, or C_T, the sum of all capacitances at the FG node. A

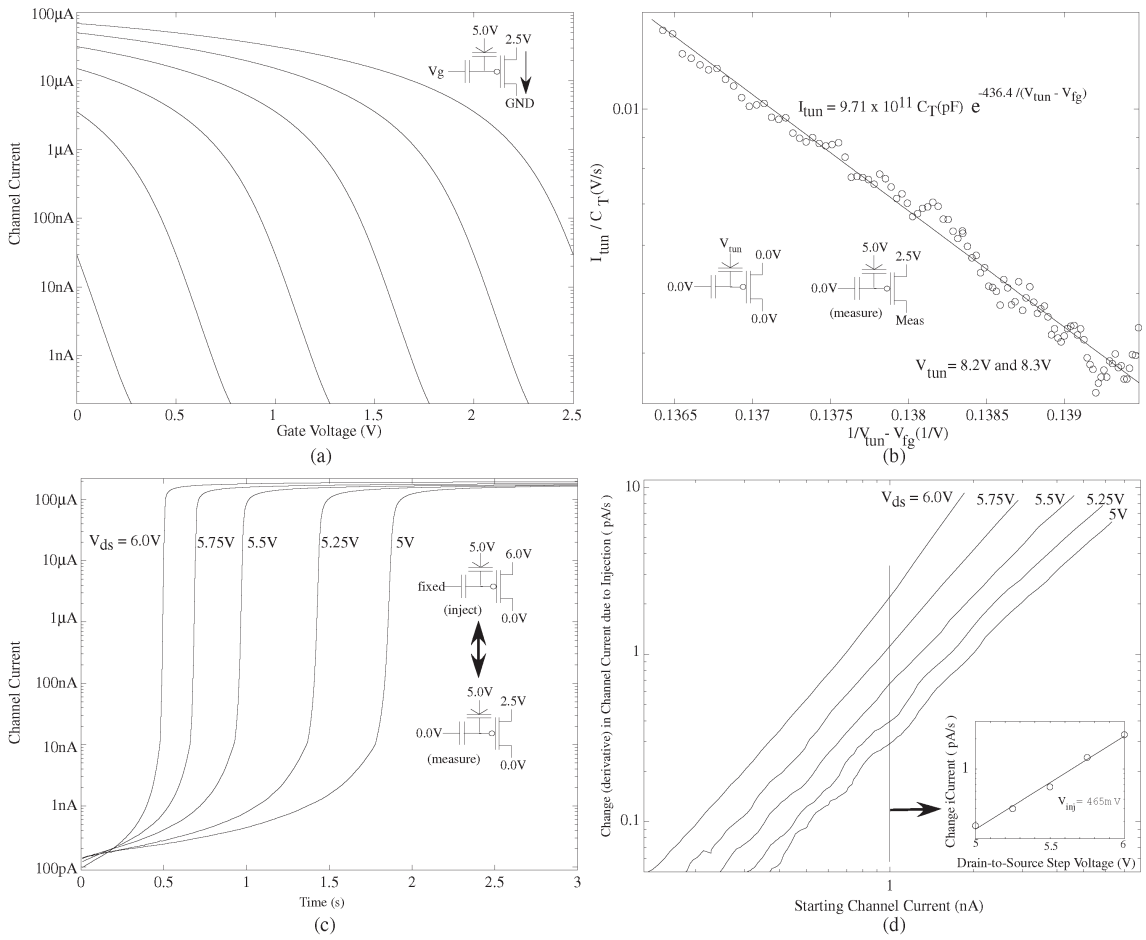


Figure 2: Fundamental measured data on FG devices fabricated in a 350nm commercially available CMOS process. (a) **Channel Current vs. Gate Voltage**: A FG pFET transistor has a similar behavior to a pFET device, but with a different effective value for the subthreshold slope (U_T/κ). (b) **Tunneling Current vs. Gate and Tunneling Voltage**: Electron-tunneling erases our FG devices; therefore, tunneling characterization finds the right applied erasing voltage, V_{tun}, and the corresponding time required for erasing a device or an array. (c) **Injection Current vs. Time**: Channel current measurement sequence (S curve) showing effect of successive fixed-drain pulse injection, for multiple drain voltages. Started at a low current ($\approx$ 100pA), the hot-electron injection FG current increases the channel current to a nearly converged current of $\approx 100\mu$A. (d) **Change in Injection Current vs. Injection Current**: The change in the measured channel current versus channel current from S curve measurements. This data representation enables characterization of the exponential dependance on drain voltage (V_{inj}) on the change in channel current.

FG pFET transistor has a similar behavior to a pFET device, but with a different effective value for the subthreshold slope (U_T/κ for a typical FET device, where κ is the capacitive voltage divider between gate and surface potential, and U_T is the thermal voltage (kT / q)) due to the capacitive divider, the incoming (*gate*) capacitance and (C_T), and a programmable flatband voltage that can move the curve throughout

the voltage range.

Because of the high quality gate insulators, the FG charge, once programmed, will remain roughly unchanged months and years later (at the same temperature) (e.g. [13,14]). We expect a typical FG device to change 1-100μV at room temperature over a 10 year device lifetime as characterized by accelerated temperature measurements for this 350nm CMOS process [14]. Long-term charge loss in floating-gate transistors occurs due to the phenomenon of thermionic emission, classically described by the simple model [16–19]

$$Q(t) = Q(0) \exp\left(-ve^{-q\phi_b/U_T}t\right) \tag{1}$$

where Q(0) is the initial charge on the floating-gate, Q(t) is the floating-gate charge at time t, v is the relaxation frequency of electrons in polysilicon, and $q\phi_b$ is the effective Si-insulator barrier potential (Volts). In [13] it has been shown that this model over-estimates the charge loss and that it does not follow such a simple curve, but a classical starting point.

Figure 2b shows characterization of electron tunneling for a FG pFET in this 350nm CMOS process. Electron tunneling adds charge at the floating gate [20, 21]. Tunneling current increases the resulting FG voltage, decreasing the resulting current measured from a pFET device connected to this FG [20]. The tunneling line sets the tunneling voltage (V_{tun}) controlling the tunneling current; thus we can increase the floating-gate charge by raising the tunneling line voltage. Tunneling arises from the fact that an electron wavefunction has finite spatial extent [22,23]. For a thin enough barrier, this spatial extent is sufficient for an electron to pass through the barrier. Tunneling current depends on the exponential of a term proportional to the thickness and proportional to the square-root of barrier energy ($E_{barrier}$); the classic expression for tunneling through a square barrier [22–24]

$$I_{tun} = I_{tun0} \exp\left(-\frac{2\sqrt{2m^*}}{\hbar}\sqrt{E_{barrier}}t_1\right), \tag{2}$$

where t_1 is the insulator thickness, m^* is the effective mass of an electron, and I_{tun0} is an experimentally determined constant for the particular insulator. An electric field across the insulator, created by the voltage difference, reduces the thickness of the barrier to the electrons on the floating gate, allowing some electrons to move through the oxide. Fowler-Nordheim tunneling, or tunneling through a triangle barrier, models electron tunneling current as [22]

$$I_{tun} = I_{tun0} \exp\left(-\frac{4\sqrt{2m^*}}{3\hbar}\frac{E_{barrier}^{3/2}}{q\mathcal{E}}\right) = I_{tun0} \exp\left(-\frac{V_o}{V_{tun} - V_{fg}}\right) \tag{3}$$

where q is the charge of an electron, and $\mathcal{E}$ is the electric field in the insulator, $\mathcal{E}t_1 = V_{tun} - V_{fg}$, and $V_o = \frac{4\sqrt{2m^*}}{3q\hbar}E_{barrier}^{3/2}t_1$, typicaly an experimentally measured parameter. Note that we can relate the pFET voltages as $V_{tun} - V_{fg} = V_{tun} - V_{dd} + V_{T0} + (V_{dd} - V_{fg} - V_{T0})$.

Figure 2c shows measurement for hot-electron injection sweeping through current for an FG pFET in this 350nm CMOS process. Hot-electron injection enables programming FG devices by decreasing the FG voltage to the particular target location. Our approach for hot-electron injection is based around fundamental physics [25], as well as fundamental FG device and circuit innovations using transistors operating with subthreshold or near subthreshold bias currents [26]. The fundamental model for hot-electron injection current (I_{inj}) is [25]

$$I_{inj} = I_s e^{f(V_{fg}, \Phi_{dc})}, \tag{4}$$

where I_s is the channel current, V_{fg} is the floating-gate voltage, and Φ_{dc} is the drain-to-channel potential for the pFET device; we often use a linearized exponential function for injection current

$$I_{inj} \approx I_{inj0}\left(\frac{I_s}{I_{s0}}\right) e^{\Phi_{dc}/V_{inj}} \tag{5}$$

where V_{inj} represents the one parameter for this linearization. The exponential dependance of drain voltage on injection current will be utilized to enable a wide dynamic

range of programming step sizes with linearly-scaled, lower-precision gate and drain voltages.

Figure 2c shows the characteristic positive feedback process for subthreshold channel currents, and the eventually saturating behavior for above-threshold channel currents, which we designate as an S curve for hot-electron injection given the shape of the response [15,26]. For a starting drain current, injection decreases the FG voltage, increasing the drain current, further decreasing the FG voltage [21]. The process slows down as the current moves to above-threshold operation (defined as significantly greater than 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 additional voltage drop across the channel [26]. Eventually, the resulting injection current slows down, resulting in minimal change in FG voltage.

Figure 2d shows that we can get more information by extracting measured current changes as a result of injection, the values required in FG programming algorithms. From these S curves, we can find the change in current as a function of current, which yields a straight line for subthreshold currents [15]. From these curves, one can, for fixed current levels, curve fit to extract out the value of V_{inj} for that device at that particular bias condition.

2.2 Scaling of FG devices

The following sections illustrate measured data for characterized FG devices fabricated in 130nm and 40nm CMOS processes as carefully chosen representative processes to show the impact of device scaling. The FG device uses a thicker insulator MOSFET, available starting in 350nm CMOS processes. Thicker oxide or effective insulator enable long (i.e. 10 year) charge storage lifetimes.

Figure 3a shows scaled pictures of different transistor sizes; the thicker insulator device for 45nm process, although having a gate similar to a 250nm process enabling

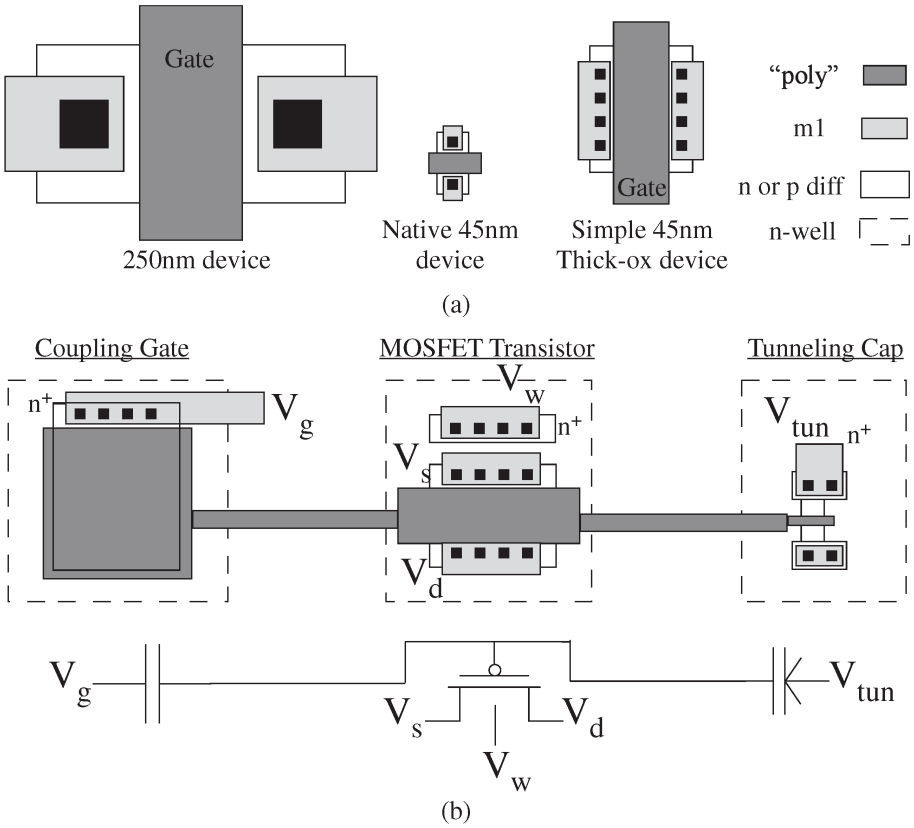


Figure 3: FG MOSFET device structure. (a) Multiple picture of the resulting FG devices and how to look at larger insulators but with smaller parasitics. A typical 250nm device, a typical 45nm device, as well as a thicker insulator 45nm device are shown. (b) Single-poly cross-section typical for FG devices, as used for 130nm and 45nm measurements. Practical devices often have additional process-dependent modifications.

long-term storage, has drain-source parasitic capacitance similar to a 45nm process. Minimizing these parasitics is critical for frequency performance for any implementation, as well as important for keeping routing fabric as small as possible. Decreasing the entire size to a typical 45nm device with a thicker insulator, typical of EEPROM type devices is possible.

Figure 3b shows the top level (e.g. layout) generic view of a single-poly FG device. Practical devices have additional improvements; some are process dependent (and therefore covered by NDA). This core structure, as shown, is used in every FG test structure since it characterizes baseline performance of these devices, starting from its initial introduction [27]. The structure is similar to the double-poly structure

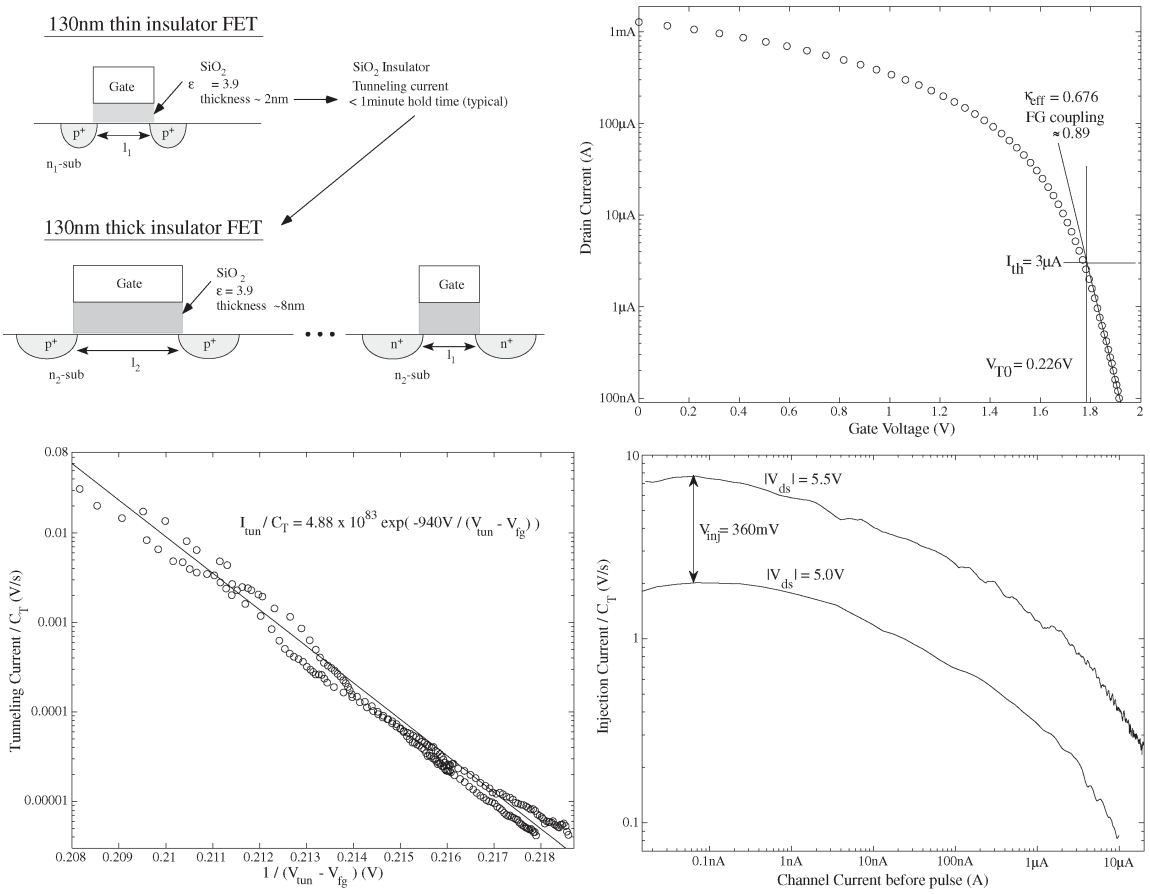


Figure 4: Moving FG devices from 350nm to smaller line width processes with SiO_2 gate insulator; this example shows data from a 130nm CMOS process. FG devices are built from the larger insulator thickness, available in all processes smaller than 350nm CMOS; the smaller insulator thickness allows significant tunneling current even with no voltage across the insulator. *Top Right:* Typical FG channel current versus gate voltage (coupling capacitively to the FG voltage), with the typical sub threshold and above-threshold regions, effective κ from the input capacitor coupling, and resulting threshold voltage ($V_{T0} = 0.226V$) and threshold current (I_{th}). *Bottom Left:* Typical tunneling current measured from two identical FG devices with extracted parameters. *Bottom Right:* Typical injection current measured from a single FG device by continuous pulsing. The drain coupling for these devices creates significant current increases due to pulsing, shifting injection towards above threshold behavior for sub threshold currents.

shown in Fig. 2 of [13]. These devices do not put the gate coupling directly above the gate electrode, as in EEPROM devices, but rather the gate is brought out for *analog* control of the resulting device. One should never have contacts to the gate electrode; contacts can significantly increase the resulting gate leakage.

2.2.1 130nm and 40nm FG devices measurements

Figure 4 illustrates moving FG devices from 350nm to smaller line width processes with SiO_2 gate insulator; this example shows data from a 130nm CMOS process. FG devices are built from the larger insulator thickness, available in all processes smaller than 350nm CMOS; the smaller insulator thickness allows significant tunneling current even with no voltage across the insulator. The insulator thickness is typically the size of a 350nm CMOS insulator thickness ($\approx 7nm$); we expect (and measure) similar (if not better) FG charge storage seen in the 350nm processes [14]. Figure 4 shows measurements of typical channel current versus gate voltage, typical tunneling current versus gate voltage measured through channel current, and typical injection current versus gate voltage (measured through channel current) and drain voltage. These measurements show typical behavior seen in 350nm devices (e.g. Fig. 2).

At 45nm / 40nm one sees a major change in the resulting MOSFET device, in that we have a change in gate insulator from the time-tested SiO_2 to HfO_2 to reduce gate leakage in the thin insulator devices. Figure 5 shows a comparison of 350nm to 40nm FG devices, with the opportunities and changes due to a change in the gate insulator. The first question is whether these new FG devices hold charge, at least sufficiently long for testing our systems, and further do we get sufficiently long hold-times to expect anything close to 10 year lifetime results. Measurements to date have shown FG devices that hold charge over days with negligible change in the stored charge. To understand the effect, one looks at the square barriers between the 350nm and 40nm devices in Fig. 5. The change in insulators do enable a thicker insulator but with a smaller barrier potential (1.4eV [28] versus 3.0eV [24]), therefore for a square barrier we would expect lower leakage than the 350nm device. The leakage for the typical MOSFET at 40nm, due to the larger insulator is lower than the leakage for a 90nm / 130nm device. The FG device uses a thicker insulator to enable long(i.e. 10 year) charge storage lifetimes; this insulator thickness results in leakage levels expected in

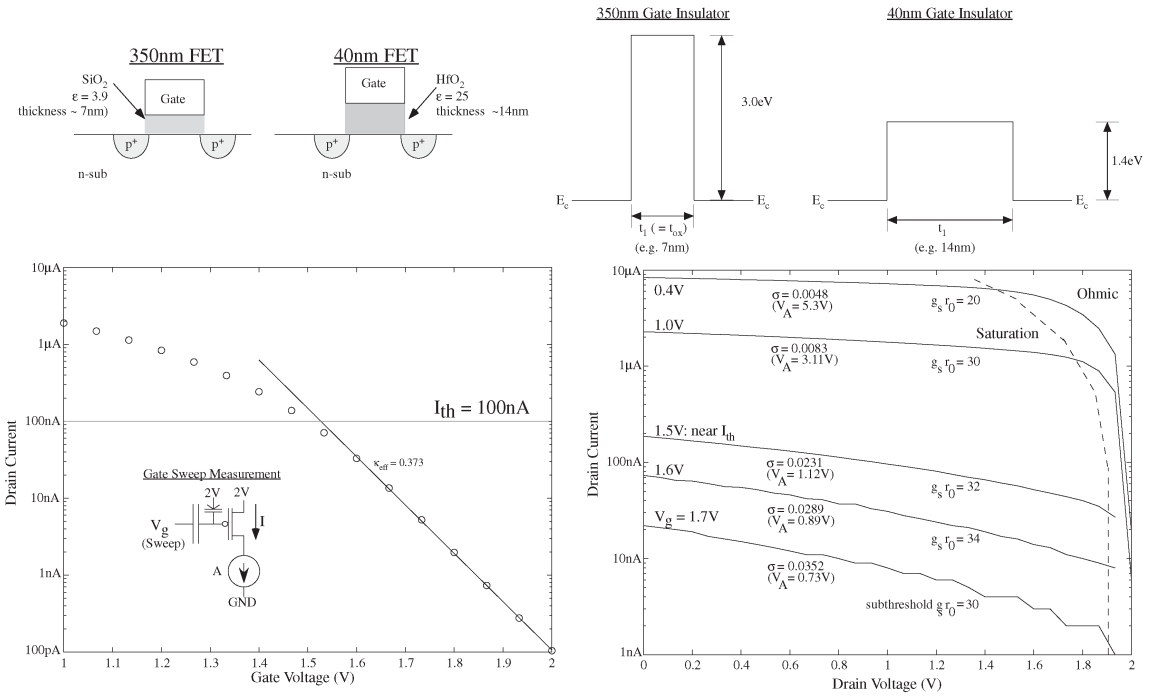


Figure 5: Illustration comparing a 350nm FG FET and a 40nm FG FET. We compare the typical device used for a 350nm FET device versus a thicker insulator available 40nm FET device that could enable long-term lifetimes for FG devices. The key change in MOSFET topology at 40nm/45nm is the use of HfO_2 instead of SiO_2. The higher ϵ of HfO_2 (25) enables a much thicker material while enabling increased coupling capacitance into the MOSFET surface potential (Ψ). The change in insulators do enable a thicker insulator but with a smaller barrier potential (1.4eV versus 3.0eV), therefore for a square barrier we would expect lower leakage than the 350nm device. From experimentally built FG devices in 40nm IC process, we can measure the channel current for gate sweeps and drain sweeps, enabled by having a FG device that holds charge (currently tested to timescales of days with no degradation). From the measured drain current as a result of a FG gate sweep through the pFET subthreshold region and near threshold region, we extrapolate an effective κ of 0.373, and a threshold current of 100nA. From the measured drain current versus swept drain voltage we extract the resulting $g_s r_0$ of these devices that includes the effect of overlap capacitances.

a 350nm device.

Figure 6 shows the concept and measurement of electron tunneling through the HfO_2 gate insulator. In both cases, electron tunneling is described by classic Fowler-Nordheim tunneling. The modified 40nm FG FET insulator results in higher electron tunneling current because of the smaller barrier to Si (1.4eV) versus the classic SiO_2 to Si barrier (3.0eV). In regressing the tunneling data, we can assume for the region used that we are in a typical MOSFET region, since these are designed to handle higher voltages. For our above-threshold current measurement versus time, we can

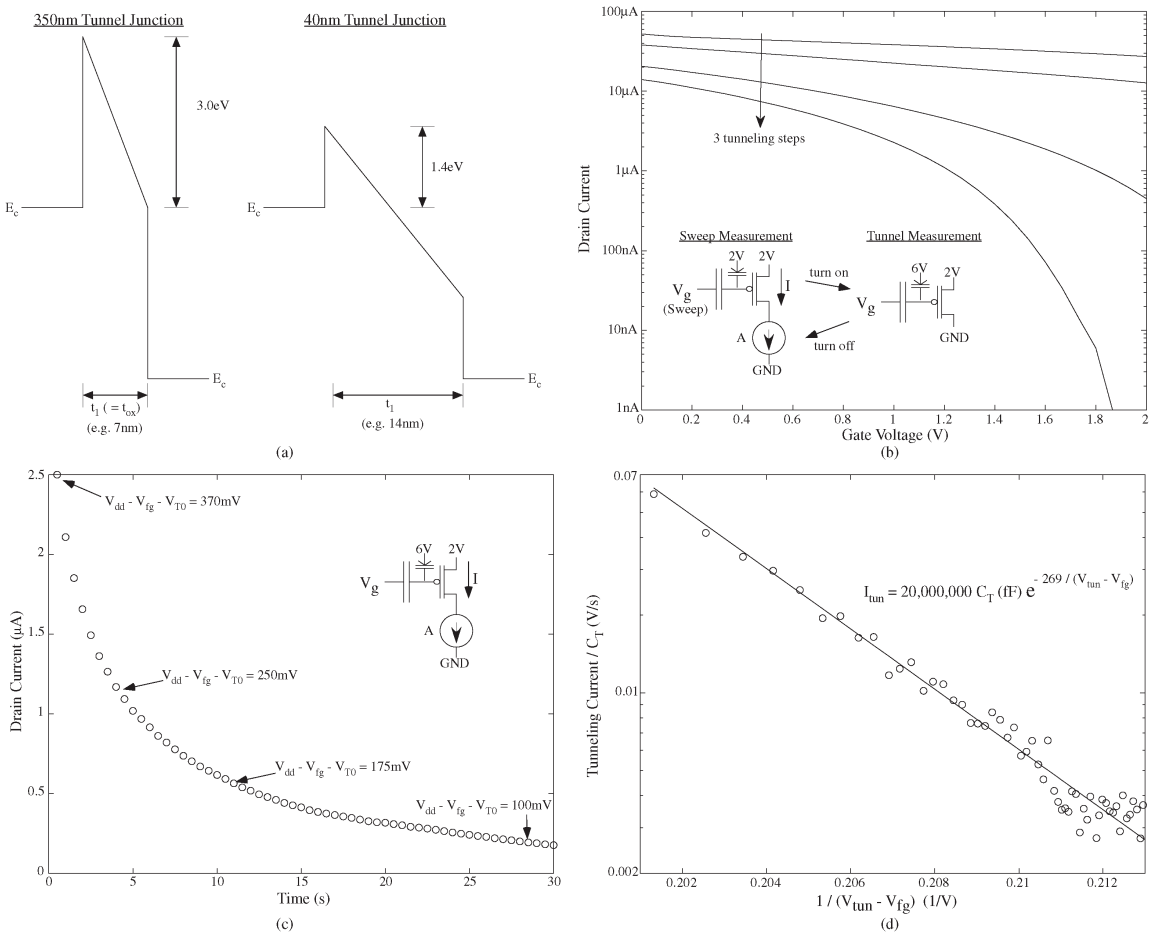


Figure 6: Measured drain current from a single 40nm FG device demonstrating electron tunneling between sweeps. (a) Comparison between 350nm and 40nm processes for electron tunneling are rooted in looking at the resulting band-diagrams. (b) One can take several gate sweep curves with tunneling between the curves. Tunneling occurred at 6V supplied to V_{tun}, with delays on the order of a minute between curve sweeps (further indicating reasonable holding times from the FG devices). Curve sweeps were taken with V_{tun} at 2.0V. (c) We can measure the time course of tunneling. From the resulting current (above-threshold) current measurements, we can extract floating-gate voltage (V_{dd} - V_{fg} - V_{T0}), enabling characterizing tunneling current versus tunneling terminal voltages (V_{tun} - V_{fg}). (d) We regressed tunneling current per unit total floating-gate capacitance (C_T) versus 1 / (V_{tun} - V_{fg}) enabling a direct comparison of the data with the theoretical expression for Fowler-Nordheim tunneling. We also plot a curve fit to that theoretical expression in (3).

take our model of current-voltage relationship (verified by data to be reasonable) as

$$I = \frac{\kappa^2 I_{th}}{4U_T^2}\left(V_{dd} - V_{fg} - V_{T0}\right)^2 \rightarrow V_{dd} - V_{fg} - V_{T0} = \frac{2U_T}{\kappa}\sqrt{\frac{I}{I_{th}}} \tag{6}$$

where threshold current, I_{th}, as $2KU_T^2/\kappa$, $K = \mu C_{ox}(W/L)$, and we extracted I_{th} as 100nA from our data on this particular device. From these measurements of V_{fg}, we

can extract tunneling by writing KCL at the FG as

$$C_T \frac{dV_{fg}}{dt} = I_{tun}(V_{fg}) \tag{7}$$

The resulting formulation allows us to take a numerical derivative to see the resulting tunneling current, enabling the plot in Fig. 6 and resulting curve fit of (3).

Figure 7 shows the discussion for the hot-electron injection process. The lower energy barrier between HfO_2 impacts channel hot-electron injection by reducing the barrier for electrons injecting into the insulator.

Figure 7b shows the band diagram and the three steps required for hot-electron injection. The first step requires movement of holes through the channel region. The second step requires movement of holes through the drain-to-channel region, resulting in high energy carriers impact ionization, creating a source of electrons for the conduction band. The third step requires movement of electrons back through the drain-to-channel region, resulting in high-energy electrons that can surmount the insulator interface. For SiO_2 barriers, a wide range of the effects were limited by hole impact ionization and we expect in these processes that the correlation will be far stronger. We expect that we will need similar voltages for injection across processes.

Qualitatively the results are similar to hot-electron injection in larger MOS devices. Figure 7 shows a typical MOSFET injection characterization to determine parameters for FG programming [15]. Figure 7d shows an incremental increase due to injection. Often in programming algorithms, we make use of an effective linear difference equation(s) for early steps in reaching target value [15]. These approaches allow using simple fixed point functions for calculating FG injection pulses to reach an analog FG target.

Figure 8 shows a progression in efficiency between the three processes for these approaches in terms of required applied voltages. The change in insulator, with its change in barrier height (3.0eV to 1.4eV for HfO_2), makes 40nm significantly more efficient in FG writing capability, while still enabling 10year lifetimes. For tunneling,

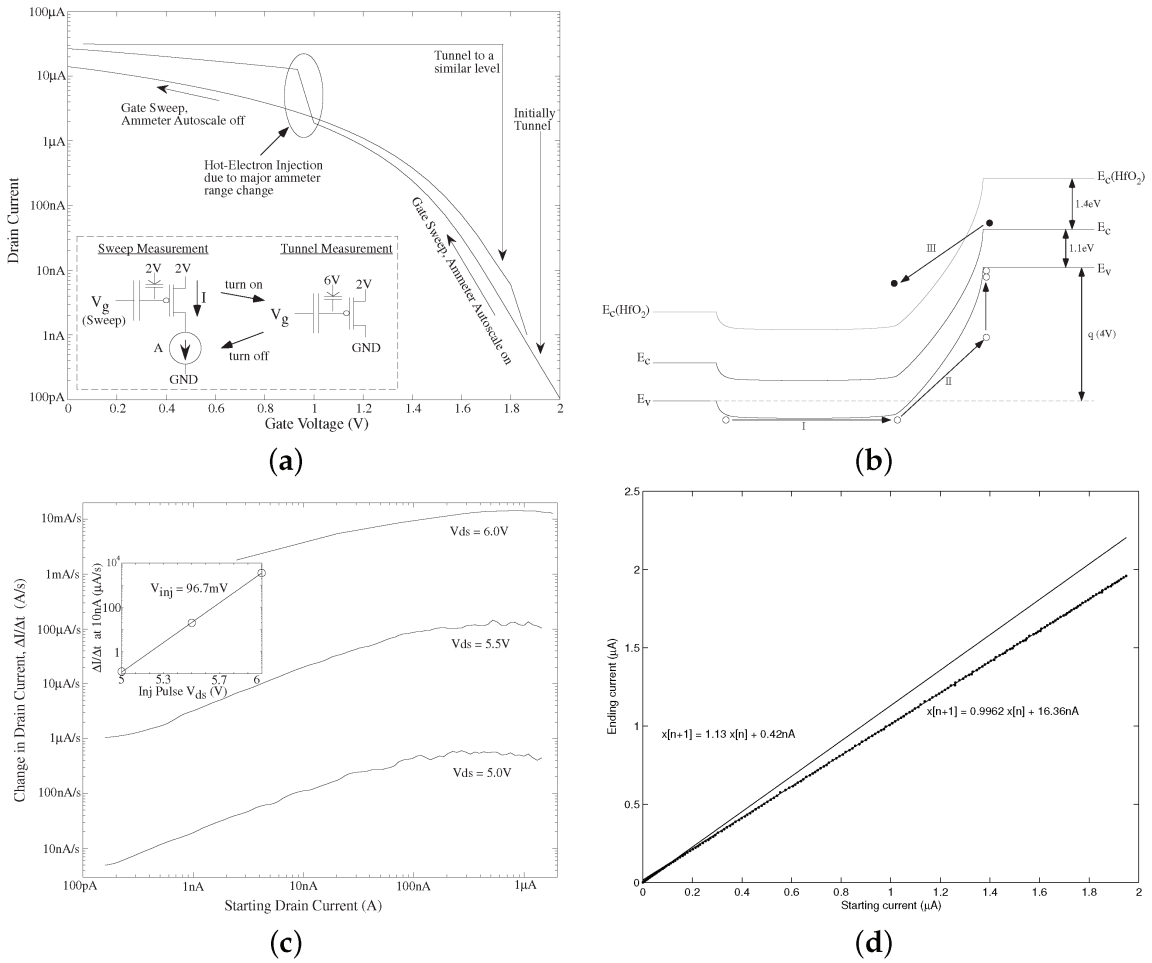


Figure 7: Initial FG hot-electron injection through measured drain current versus gate voltage sweep from a single 40nm FG device. (a) Initial tunneling to bring the initial curve into range. We took a gate voltage sweep that included a sharp jump in current, performed a tunneling step to return to a similar condition, and then took another gate sweep without auto ranging, eliminating the step in current during measurement. During the major ammeter range change at 2μA, the drain voltage dropped for a short time to a voltage below GND, enabling enough field on this FG device to inject, as seen by the immediate step in current resulting from an decreased level of FG charge. (b) MOSFET band diagram for channel hot-electron injection for sub threshold currents in a 40nm CMOS technology. (c) Measured change in measured drain current for a fixed pulse width, T, versus starting drain current, measured before the pulse at low injection voltages. We show these measurements for three values of — V_{ds} —. We extract the resulting slope at a fixed current (i.e. 10nA). This slope is $1/V_{inj}$; $V_{inj} = 96.7$mV. (d) Measured resulting current after an injection event versus initial measured current.

we get a smaller V_o due to the lower starting barrier; the quantity $V_o \propto t_1(E_{barrier})^{3/2}$ remains nearly constant (less than 6% change) between 350nm and 40nm devices. For injection, we get a significantly higher injection current and sharper slope (as seen by V_{inj}), as a combination of efficiency and higher substrate doping. We expect similar

	350nm	130nm	40nm
$E_{barrier}$	3.0eV	3.0eV	1.4 eV
t_1	7nm	8nm	14nm
V_o	436.4V	940V (2 stage)	269V
V_{inj}	465mV	360mV	96.7mV
($\|V_{ds}\|$ center)	5.5V	5.25V	5.5V
ΔI(I = 1nA) /s	1pA/s	4pA/s	1μA/s

Figure 8: Comparison of measured device parameters. All three devices showed FG retention equivalent to less than 1mV room temperature drop over 10year lifetime.

behavior scaling down to 14nm devices give the similar insulator structure.

Although scaling of tunneling physics is straight-forward, scaling of hot-electron injection physics for these pFET devices should receive additional discussion. Hot-electron injection in pFET devices, operating at sub threshold and near threshold currents, are influenced mostly in the increased substrate doping allowed by the stronger insulator capacitor coupling, as well as the different Si-insulator barrier height. The substrate doping does not increase as fast because of the doping profile used for thicker insulator devices; when this layer is removed, one expects higher electric fields and lower impact-ionization and hot-electron injection voltages. Impact ionization always occurs significantly before any further device breakdown effects. We have two regions to consider, the hot-hole transport and resulting impact ionization that creates the resulting conduction band electrons, and the hot-electron transport and resulting electron injection efficiency.

The restoring force for the electron and hole high-field transport is primarily optical phonons, where first the carrier needs to gain more energy per unit distance than the optical phonon restoring force (E_R /λ) typically requiring a starting distance (z_{crit}) for an increasing potential, and then the average carrier trajectory includes field gained energy minus this required starting energy. The resulting distribution function around this average trajectory is a local convolution of Gaussian functions,

the eigenfunction of a linear diffusion equation (e.g. heat equation). We have discussed these fundamental transport details elsewhere [25].

Electrons in an electric field will gain energy faster than holes in an electric field, as characterized by their typical mean free length for an optical phonon collision of energy E_R ($\approx$ 63meV), where electrons (λ_e) are approximately 6.5nm [25], and holes (λ_h) are approximately 4.2nm [29]. Although impact ionization can occur for an electron or hole with energy of 1.1eV, requiring carriers to exceed this barrier, electron impact ionization is known to be an efficient process for electron energies above 2.3eV [25,30], and hole impact ionization is known to be an efficient process for hole energies above 3.0eV [29]. The resulting created electrons, sometimes starting with additional energy because of the impact ionization process, begin around the highest field region, gaining energy as they accelerate to get over the 3.0eV (Si-SiO_2) or 1.4eV (Si-HfO_2) barrier.

The 3.0eV barrier level for significant hole impact ionization would be the primary limit the hot-election injection process when using a Si-SiO_2 barrier (3.0eV) given the significant difference between λ_h and λ_e, although some functional dependance is still possible. The 2.3eV level for hot-electron impact ionization means we will get some significant loss of electrons before reaching the Si-SiO_2 barrier (3.0eV), while we have negligible loss of electrons before reaching the Si-HfO_2 barrier (1.4eV), resulting in significantly higher injection current. For the Si-HfO_2 barrier, the electron dynamics can almost be approximated by a constant factor, and approximation often desired (but not physically correct) between injection and impact ionization currents.

2.2.2 FG Switch Behavior

Because we have proven that FG devices are functional throughout a wide range of CMOS IC processes, we now transition to looking at the scaling properties of an FG MOSFET acting as one of multiple switches say in a small crossbar array

(e.g. [12, 21]). The operating speeds of an FPAA array is, to first order, limited by the FG MOSFET switch. We will analyze the high frequency behavior of a switch in an array of switches (e.g. routing fabric). For this analysis, a programmed switch is at one of two cases, when the switch is "off" and when the switch is "on". A programmed FG voltage can be set at significantly higher or lower voltages than the power supply range. Our discussions focus on nFET and pFET devices; these dynamics are effectively interchangeable with slight changes in parameters.

For the "off" switch case, we start with the channel in accumulation. Because the switch value can be programmed outside the power supply, a slightly positive value (above V_{dd}) will guarantee the device stays in accumulation for all applied voltages including the GHz frequencies we are considering in this discussion. In accumulation, we have no appreciable depletion capacitance, and therefore the capacitance between the floating-gate terminal and the substrate is the oxide (or insulator0 capacitance (C_{ox} W L). The effective conductance between source and drain is effectively zero, being the conductance of two reverse-biased diodes. Gate length has little to do with the off case (other than total gate capacitance) in accumulation.

With the zero conductance between source and drain, any potential communication between switches must happen through capacitive coupling. The gate to source-drain junction capacitance, C_{ov}, is the biggest issue in terms of signal feedthrough, which scales proportionally to other device properties. Minimizing C_{ov} decreases the amount of capacitive signal feedthrough feedthrough, which could be further decreased by opening up drain-source regions (avoiding some of the self-aligned device). C_{sb} and C_{db} p-n capacitors connected to signal GND (actually V_{dd}). Therefore, the frequency-indpendant coupling gain from source to drain voltage would be the resulting capacitive divider network as C_{ov}^2 / (C_T C_{db}), where C_T is the total capacitance of the floating-gate node. Both terms result in a coupling less than 10^{-3}; with multiple switches in series, this value is nearly negligible. In a switch matrix, the resulting

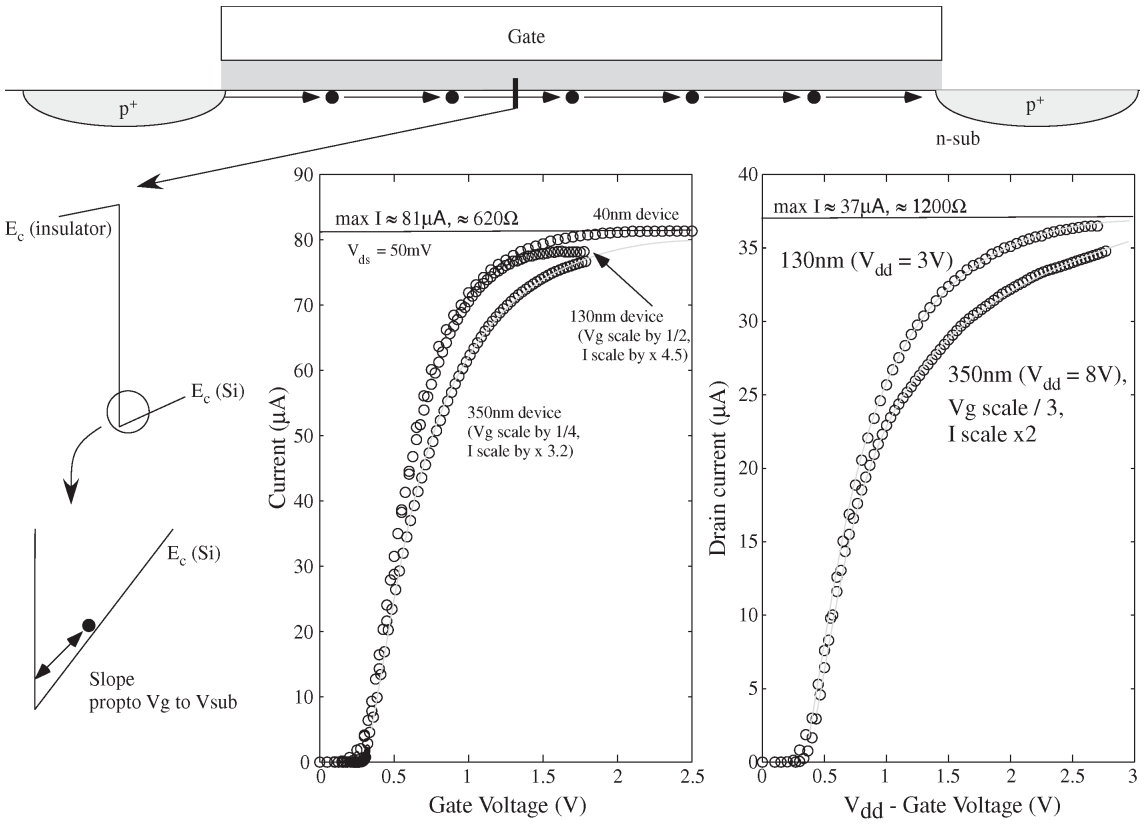


Figure 9: Maximum conductance for a MOSFET device is determined by the two-dimensional carrier behavior. The *channel* electron oscillates along the triangle barrier created by the insulator interface and MOS capacitor depletion region as it moves from source to drain regions. This oscillation increases the number of elastic collisions, decreasing the carrier mobility, as seen by the conductance saturating for higher gate voltages.

load capacitance is the sum of all of the capacitors on the line, further decreasing this effect. Further, this effect is negligible in 350nm FPAA devices at low frequencies, with no significant coupling has been measured whether in characterization or in applications.

For the "on" switch case, we are primarily concerned with the frequency response through the particular device. The MOSFET channel is biased far above threshold behavior, operating in the ohmic regime. The FG voltage is not constrained between the power supply rails, allowing the transistor to its maximum conductance point, the point for high gate voltage where the source-to-drain conductance of channel is approximately independent of gate voltage. This maximum conductance, or R_c, is

roughly independent on process minimum channel length, where the conductance is set by velocity saturation of electrons / holes for the MOSFET channel. For a device with equal width and length, the nFET saturates around 3kΩ and pFET near $6k\Omega$.

Figure 9 presents the discussion for this maximum conductance, where an increase in gate voltage increases number of collisions with Si–insulator barrier while carriers effectively move from source to drain voltage. An *on* MOSFET switch typically would have a small voltage between the source (V_s) and drain (V_d) voltages, resulting in the MOSFET modeling

$$I = \mu C_1 \frac{W}{l}(V_g - V_{T0} - V_s/\kappa)(V_d - V_s), Q_d \approx Q_s = C_{ins}(V_g - V_{T0} - V_s/\kappa) \qquad (8)$$

where μ is the carrier mobility in the channel, C_1 is the insulator capacitance per unit area (ϵ_1/t_1), κ is the capacitive divider between C_{ins} and the total capacitance in the channel, V_{T0} is the threshold voltage, W and l are the width and length of the MOSFET device, Q_s and Q_d are the channel charge at the source and drain edges of the channel region, respectively. The measurement in Fig. 9 uses V_s = 0V, V_d = 50mV for continuously ohmic operation, while sweeping V_g over a wide voltage range. In this setup, V_s is set to the substrate (so $V_s = 0$); pFET are measured down from their substrate held at V_{dd} (different for each technology). One would expect a conductance (G) that linearly increases, after V_{T0}, with V_g, as

$$G = \frac{I}{V_d - V_s = 50mV} = \mu C_1 \frac{W}{l}(V_g - V_{T0}) \qquad (9)$$

Figure 9 shows measured data illustrating this initial linear behavior, as well as deviations from it leading towards conductance saturation.

The modeling for conductance saturation investigates the change of μ with gate voltage; other terms remain nearly constant. Figure 9 shows that although we draw carriers (e.g. electrons) moving in a straight line path through the channel from V_s to V_d, we have a field in the orthogonal direction (gate direction) that pulls these carriers towards the gate region. These carriers get pulled into the Si–insulator

barrier, elastically colliding and reversing direction towards the substrate until the electric field of the channel brings the carriers back towards equilibrium. From (8), the electric field at the Si-insulator barrier

$$\text{insulator} : \frac{V_g - V_{T0}}{t_1},$$

$$\text{Si edge} : \mathcal{E}_{Si} = \frac{\epsilon_{ins}}{\epsilon_{Si}} \frac{V_g - V_{T0}}{t_1}. \tag{10}$$

The electric field in Si will decrease moving into the depletion region. A constant electric field (linear change in potential) is expected at the boundary layer right at the Si-insulator boundary. The additional elastic collisions will decrease the resulting carrier mobility, μ,

$$\mu = \frac{q\tau}{m^*}, \frac{1}{\tau} = \frac{1}{\tau_0} + \frac{1}{\tau_{gateE}} \tag{11}$$

where τ is the mean free time due to collisions, m^* is the carrier effective mass, τ_0 is the mean free time due to typical restoration forces in the channel, such as acoustic phonons, elastic scattering mechanisms, as well as any effects due to some optical phonon behavior, and τ_{gateE} is the collision component due to average elastic collisions with the Si–insulator barrier. Transit time would be the ratio of average distance traveled over the average velocity of carriers. The energy of the carriers through the channel is never high, roughly at the typical kT =q U_T average energy for a fermi distribution (Energy less than Fermi level), for an average distance traveled (due to electric field) as U_T $/\mathcal{E}_{Si}$. Velocity of carriers at lower $\mathcal{E}_{Si}$ is proportional to $\mu\mathcal{E}_{Si}$. Velocity of carriers at higher $\mathcal{E}_{Si}$ approaches velocity saturation, v_{sat}; in this region we get

$$\tau_{gateE} = \frac{U_T}{v_{sat}} \frac{\epsilon_{Si}}{\epsilon_1} \frac{t_1}{V_g - V_{T0}} \tag{12}$$

For increasing V_g, the transport progresses starting in the region with a constant τ_0, resulting in classical linear increase in conductance, to the region where τ is decreasing due to elastic collisions, resulting in a sub linear increase in conductance,

to the region where τ is dominated by τ_{gateE} with large $\mathcal{E}_{Si}$. In this final region, where the conductance saturates at G_{max}, the current is expressed as

$$I \rightarrow \frac{q}{m^*}\tau_{gateE}\frac{\epsilon_1}{t_1}\frac{W}{l}(V_g - V_{T0})(V_d - V_s)$$

$$= \frac{qU_T}{m^*}\frac{\epsilon_1}{t_1}\frac{\epsilon_{Si}}{\epsilon_1}\frac{t_1}{V_g - V_{T0}}\frac{1}{v_{sat}}\frac{W}{l}(V_g - V_{T0})(V_d - V_s)$$

$$= \frac{qU_T}{m^*}\frac{\epsilon_{Si}}{v_{sat}}\frac{W}{l}(V_d - V_s) \tag{13}$$

$$G_{max} = \frac{I}{V_d - V_s} = \frac{qU_T\epsilon_{Si}}{m^* v_{sat}}\frac{W}{l} \tag{14}$$

where G_{max} is not a function of typical device parameters, except for drawn transistor width and length. G_{max} is not a function of the insulator thickness. Figure 9 shows 350nm, 130nm and 40nm nFET or pFET device measurements illustrating the conductance saturation in each case.

The other side of the question is the resulting capacitance to set the resulting time constant. For example, we can make wider MOSFET switches, decreasing the resulting channel resistance, but increasing the resulting capacitance found in a dense array of FG devices. Switch capacitance is primarily due to source-drain junctions, C_{sb} and C_{db}, as well as a small capacitance through the FET gate oxide, through the resulting capacitive network to other potentials. We identify the resulting capacitance as C_s. The relative size of these capacitances typically scales quadratically with scaling of process line width.

The design of such an array must discuss whether to have the well connected to signal GND, which would be a solid GND even for RF frequencies, or have a high-impedance connection. A useful high-impedance to each switch requires that each switch be placed in a separate isolated well device, as well as having a high resistance connection to the resulting well terminal.

Therefore, for a single switch, we see minimal additional coupling effects through the switch. We expect there will be some effect of the RC effect in the channel

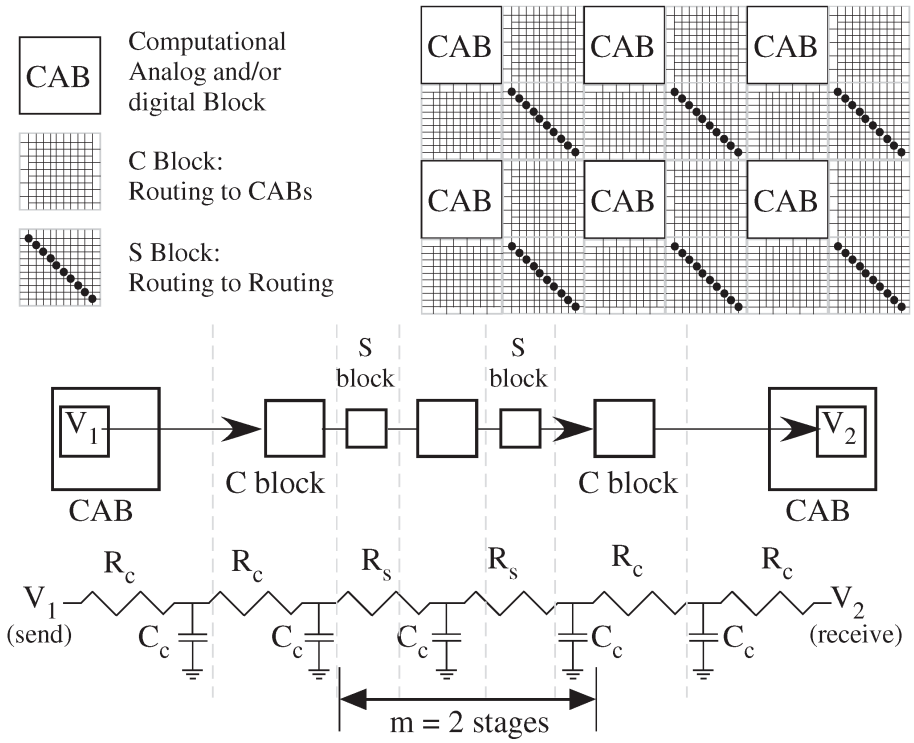


Figure 10: Manhattan FPAA architecture, including the array of computation blocks and routing, composed of Connection (C) and Switch (S) blocks. The routing infrastructure can effectively be modeled as a distributed RC line. The lowest figure shows a typical routing fabric assuming a single routing of C and S block switches, where C_c is the connection capacitance including the C block lines, R_c is the C block switch resistance, and R_s is the unbuffered S block switch resistance. m = typical number of switches needed for a connection.

(usually modeled by a resistor and inductor) of course, around f_T (max) due to maximum conductance. The low frequency modeling directly applies to our modeling approaches.

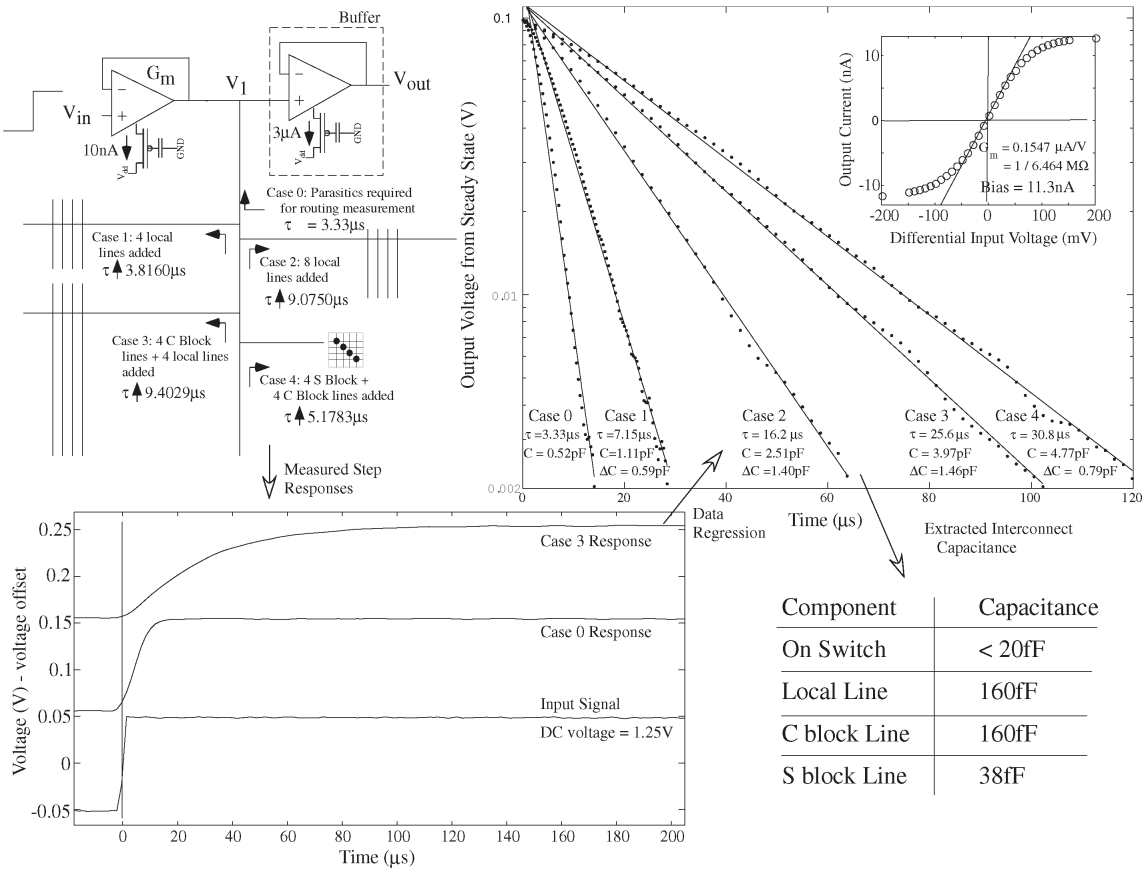
2.2.3 Routing Capacitance

Having working FG devices as well as modeling of individual switches, we move towards modeling the frequency response of an FPAA fabric, justifying Fig. 1b. Figure 10 shows the basic Manhattan based routing structure used for our SoC FPAA device (i.e. [12]). The approach includes a compartment for a Computational Analog Block (CAB) or a Computational Logic Block (CLB), includes two Connection ("C") blocks to connect these devices, and includes one routing Switch ("S") block. Using the

Manhattan style routing enables direct interactions with existing tool flows (i.e. [31]). We still hold that the routing fabric in this architecture is both useful for computation as well as switching, particularly for local CAB / CLB routing as well as in the "C" block routing. Other FPAA architectures show similar approaches and tradeoffs, although this architecture mades these issues more explicit.

From a classical FPGA approach, one considers the capability of the device to be solely in its components (CLBs, specialized blocks), and the routing fabric is simply a mechanism to interconnect these components. Minimizing the effect of the routing fabric reduces, from a circuit perspective, dead weight that can only degrade the circuit. That approach requires minimizing the number of switches, each of which adds resistance, as well as minimizing the resulting capacitance of the routing. The routing infrastructure can effectively be modeled as a distributed RC line. The architecture looks at the relationship of the resulting switch resistances, as well as other circuit uses of the FG switch devices as a function of the number of CAB inputs and number of tracks, as well as the typical number of switches needed for a connection.

The SoC FPAA [12] enables programming experiments that characterize the fundamental properties of the configurable fabric by experimental measurements on the configurable routing fabric. Figure 11 illustrates compiling (and measuring) two circuits to characterize precisely the behavior of these circuits, including load capacitance of the fabric itself. This FPAA structure facilitates the direct characterization of the resulting capacitance; coupled with the resistance of an on-switch, R_c. One can directly predict delays along each of these lines. Every experiment uses the same voltage biasing, fixing the capacitance of p-n junction devices throughout this experiment. The resulting measurements give a measurement of the resulting routing capacitance, as well as enables, through the routing fabric, a range of tunable capacitor blocks. Precise measurement of routing capacitances enables tuning, through programming



Component	Capacitance
On Switch	< 20fF
Local Line	160fF
C block Line	160fF
S block Line	38fF

Figure 11: FPAA characterization of routing capacitances. Initially, one first measures the current-voltage relationship for a specific OTA device, shown in the inset, to exactly find the resulting G_m (0.1547μA/V) of the device. That exact OTA with the same programmed current is used to measure the time-constant of the step response (on a 1.2V dc for the 2.5V supply) for different (additive) routing combinations. From the step responses measurement shown, a linear curve (in log amplitude) fits to the time constant after removing the effect of the steady state voltage. The extracted routing capacitance values for multiple measurement configurations are summarized.

switches, for precise capacitances where needed for matching. Matching of capacitances and programmability of current sources by FG techniques dramatically reduces the effect of mismatch in small cell sizes.

2.3 Conclusion

This chapter presented scaling of FG devices, and the resulting implication to larger (FPAA) systems. The properties of FG circuits and systems in one technology (e.g. 350nm CMOS) are experimentally shown to roughly translate to FG circuits in scaled

down processes in a way predictable through MOSFET physics concepts. This discussion addressed the question of scaling these devices to more modern processes, in particular using the example processes of 130nm, 40nm CMOS, empowering moving such approaches to smaller linewidths CMOS processes. Scaling FG devices results in higher frequency response, (e.g FPAA fabric) as well as lower parasitic capacitance and lower power consumption. An FPAA's operating frequency improves as the IC technology process is scaled down. FPAA architectures, limited to 50-100MHz frequency ranges could be envisioned to operate at 500MHz-1GHz for 130nm line widths, and operate around 4GHz for 40nm line widths.

CHAPTER III

PROGRAMMING OF FG DEVICES

Fowler-Nordheim tunneling [22] enables erasing FG devices by increasing the FG voltages, which is a global operation requiring a sufficiently high voltage (12 V) on the tunneling junction of all the FG devices. Hot-electron injection enables programming FG devices by decreasing the FG voltage to the particular target location, which is an individual operation. The approach for hot-electron injection is based around fundamental physics [25], as well as fundamental FG device and circuit innovations using transistors operating with subthreshold or near subthreshold bias currents [26].

3.1 FG Programming Infrastructure

This section first discusses the infrastructure and programming framework, Figure 12 illustrates the standard for accessing the FG devices for programming. We show a representative structure, including switches and active devices, typical of a Computational Analog Block (CAB) in an FPAA device. 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 only allowing a high voltage between 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

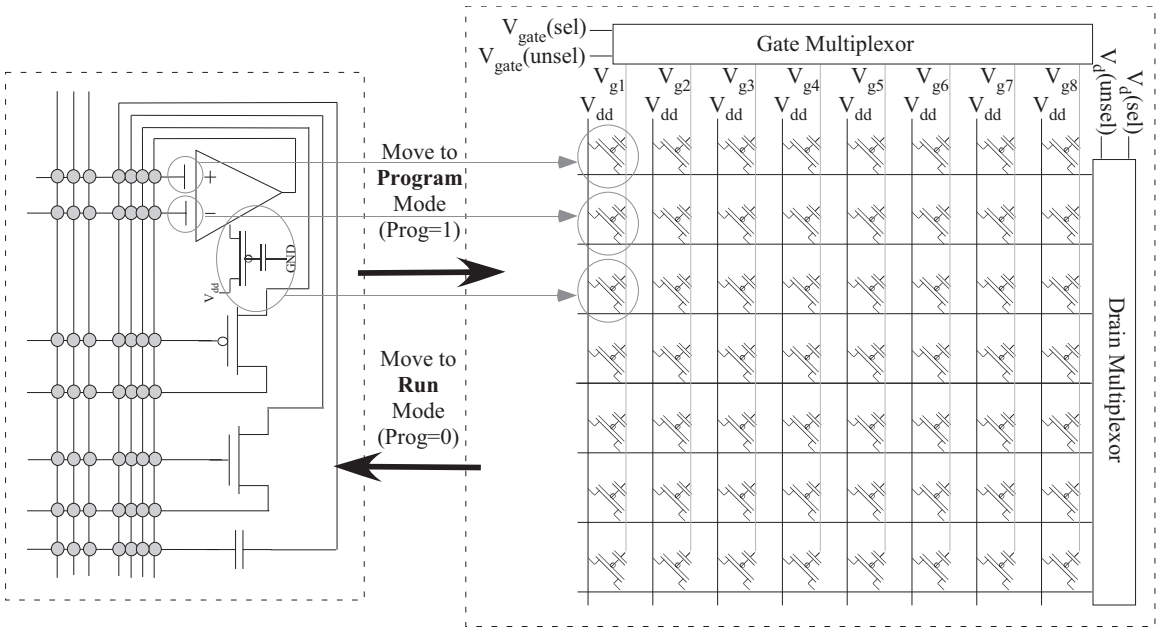


Figure 12: When using a heterogeneous array of FG devices, we require that all FG devices are reconfigured into a single crossbar array of FG devices. In this configuration, we can program using hot-electron injection with nearly perfect selectivity because a device requires channel current *and* high voltage from 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 *program* mode and *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.

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 *program* to *run* mode.

Figure 13 shows the board and on-chip infrastructure. Figure 13a shows the board level infrastructure for programming and accessing the FG programming structure, as well as the digital infrastructure for accessing the chip during normal programming operation. Figure 13b shows the high level block diagram for the on-chip programming structure, which includes utilizing an open-source μP, embedded 16k $\times$ 16 SRAM for program and data memory, as well as the memory mapped registers which control the elements for programming. We see in Fig. 12 that we simply need to control the gate

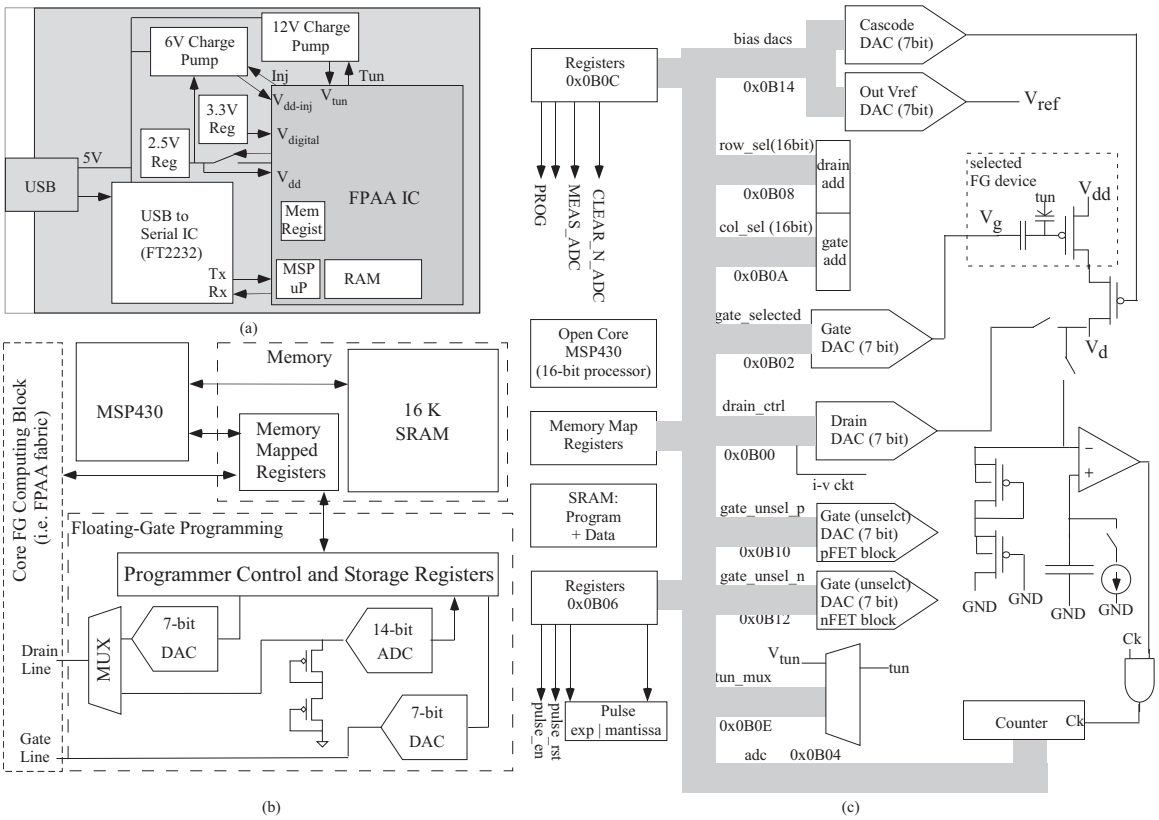


Figure 13: Integrated infrastructure blocks including test board, high level IC schematic for programming, and detailed programming schematic. (a) System interface block diagram used for programming a representative FPAA device with the on-chip open-source MSP430 μP microprocessor (μP) and memory. The primary off-chip infrastructure is μP IC controlled high-voltage power handling (12V and 6V 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 the μP. (b) Core on-chip circuit infrastructure used for a μP based programming arrays of FG devices. The μP and integrated 16k x 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 current to voltage. We estimate the processor requires approximately 200pJ per simple instruction, including the local memory access. (c) Detailed programming architecture showing the entire register map, including DACs, measurement ADC, and row-column selection registers for the configurable device. As seen in Figure 12, we supply multiple input voltages (DACs) and a cascade voltage to isolate the pFET devices from the measurement circuitry. Further, we use additional DAC voltages for unselected pFET programming rows and/or columns.

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 DAC are not directly correlated to the final programmed accuracy; the frequency and noise of the ADC, which is 14 bit, is directly related to the final programming

accuracy. Figure 13c gives a better sense of the actual programmer complexity required for programming an entire crossbar array of FG devices, including selection registers, control bits, DACs and ADCs, all implemented on-chip, to handle the resulting integrated programming for a heterogeneous array of FG devices, including the particular memory-mapped register locations. Once the infrastructure is characterized to be functional, then this entire block becomes an IP block for any further design; our recent designs all utilize this infrastructure, and as a result, the code is compatible and reusable throughout these designs.

3.2 On-Chip Integrated FG Programming Algorithm

Our FG programming relies on a combination of electron-tunneling for erasing and resetting FG devices and hot-electron injection for programming FG devices. Figure 14 shows the framework for our FG programming approach, with the resulting tunneling and injection steps. Over the following subsections, we will discuss the global erasing and initialization (reverse tunneling) steps, FG recovery by injection, FG rough programming through open-loop injection, and the final short step of FG fine programming through predictive FG calculations.

3.2.1 Global Array Erasure and Initialization

Erasing a block requires raising an entire block of tunneling junction voltages to a sufficiently high voltage to result in 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 floating-gate values to a high voltage that ensures a pFET device has no channel current, and then we perform a *reverse tunneling* operation to bring floating-gate voltages to a small, but negligible, pFET current for a gate selection voltage at 0V. Electron tunneling for erasing blocks is a common approach for erasing FG devices [20, 21, 32–34]. The 12V charge-pump IC is only operational during tunneling erase operation. One could choose to tunnel

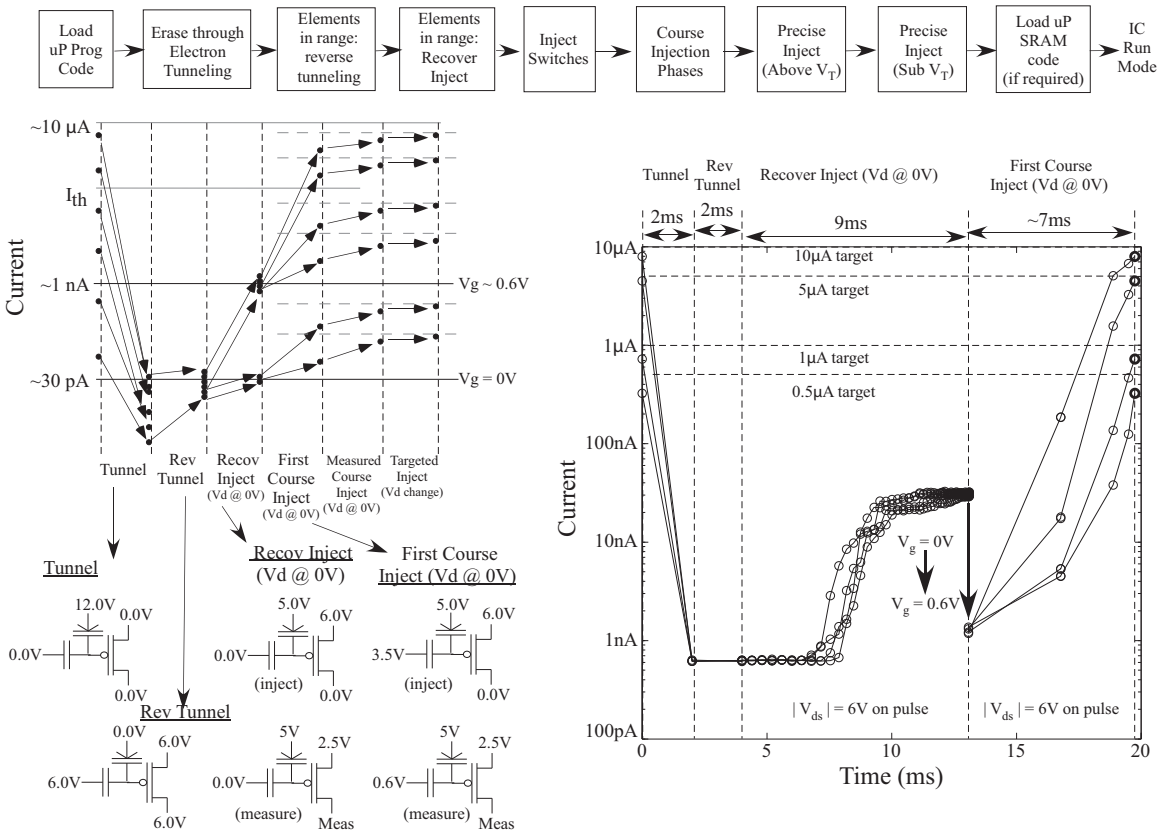


Figure 14: Algorithm steps for programming an array of FG devices. **Top**: Block diagram of the key required programming steps. The overall programming sequence requires putting the programming code into memory (for each operation), erasing and recovering devices through electron tunneling, reverse tunneling, and an additional recovery injection step, injecting FG switches where used, then using a sequence of course and fine injection phases to reach the target for subthreshold and above-threshold currents. In each case, switch data is 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 *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 4 FG devices through the first four programming steps; Fig. 15 will show the remaining precision injection measurements. The goal is to get devices close to target programming. Our devices operate in *run* mode, biasing V_g at roughly 0.6V; our measurements start with V_g at 0V to measure devices with low currents and enable FG precision targeting for currents below the $\approx$ 1nA measured leakage current from the array.

the voltage *just enough* to reach a sufficiently low current and measure a few representative currents, expecting that all 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 devices. Therefore, without measuring all devices, we find it useful to simply tunnel enough so that each device's

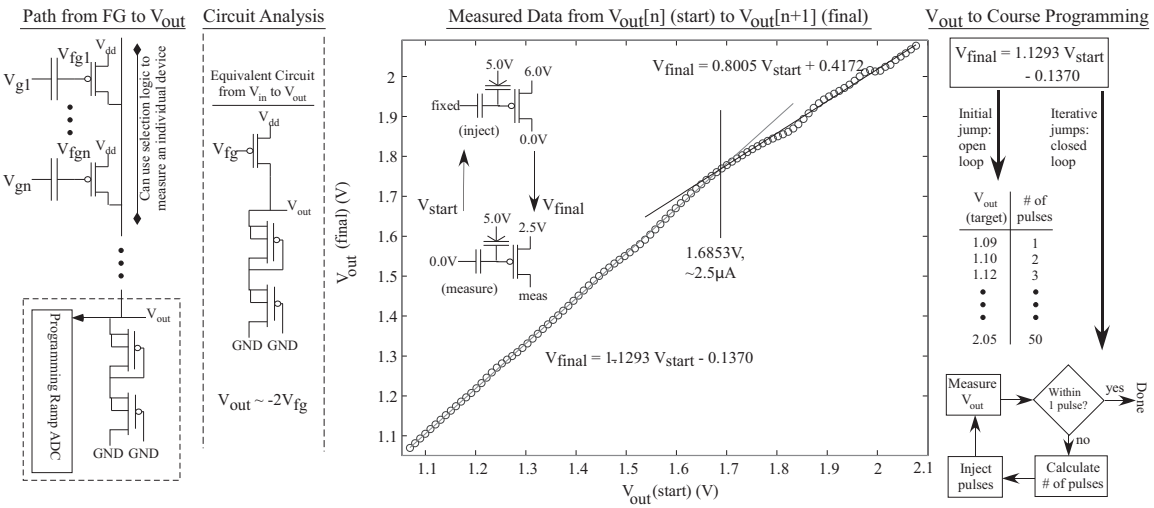
FG voltage is sufficiently high.

Reverse tunneling turns the polarities around on the tunneling junction to bring the resulting currents back towards a small but reasonable current for injection with less device mismatch than tunneling. The reverse tunneling phase requires lower voltages; for the 350nm IC process, voltages between 0V and 6V 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 then resetting voltages to normal operating condition.

3.2.2 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 non-negligible current. FG pFET devices that we do not program will stay in accumulation and pull negligible levels ($<$ 1pA) of current, even for scaled down devices. Further, our FG devices in run mode are biased with V_g at roughly 0.6V, enabling the algorithm to observe lower current values by measuring current at V_g at 0V. For the IC used for measurements, we had a constant leakage current, due to the reverse-bias source–drain junction currents near 1nA, thereby enabling current measurements in 10-30pA range even with this high leakage current. For a *switch* FG device, we typically measure 30pA of current for V_g at 0.6V but 1nA for V_g at 0.0V; for other devices with larger capacitive coupling into the FG, this effect is even stronger.

The initial process simply looks for a significant channel current (i.e. 20-30nA) when measuring current at $V_g = 0.0$V for a switch element corresponding to 1nA for $V_g = 0.6$V; levels for a significant channel current differ for different groups of FG devices with different capacitive couplings. Further, we have a programming sequence in parallel to what is shown in Fig. 14 for lower currents ($<$ 1nA for switches) where



	Vout linear fit	aout linear expression
Lower	Vout [n+1] = 1.1293 Vout [n] - 0.1370	a [n+1] = 1.1293 a[n] - 573
Upper	Vout [n+1] = 0.8005 Vout [n] + 0.4172	a [n+1] = 0.8005 a[n] + 2170

Figure 15: Movement from basic FG circuit array elements to course programming algorithm. **Path from FG to $\mathbf{V}_{out}$**: A crossbar network of indirect FG devices communicates through the drain current outputting a voltage through 2 pFET devices directly related to the FG voltage. Each FG device must be measured as well as enabled for hot-electron injection. **Circuit Analysis**: The equivalent circuit from the FG voltage (V_{fg}) to the output voltage (V_{out}). Using the same size (or similar size) pFET devices, each with the source and well voltage tied together that further implies that κ values should match, the gain from the FG voltage to V_{out} of 2; the gain from V_g to V_{out} depends on the capacitive coupling. **Measured Data from $\mathbf{V}_{out}$[n] (start) to $\mathbf{V}_{out}$[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 (6V). We approximately get two straight-line curves, one modeling the subthreshold part of the regime and one modeling the above-threshold part of the regime. The resulting model enables direct fixed point computation for the updates for the on-chip μP with a simple fixed point multiply and addition operations.**$\mathbf{V}_{out}$ to Course Programming**: Using this data, the first initial injection jumps requires 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 pulse width, 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**: The resulting extracted (and used) V_{out} equations expressed both in measured voltage, as well as in measured 14-bit ADC codeword (a[n]).

we just simply inject until we have roughly 1nA of current for $\mathrm{V}_g = 0.0\mathrm{V}$, enabling targeted currents between 30pA and 1nA as needed. One can modify the drain voltage for the pulses to move the current values as close as desired.

3.2.3 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. These approaches build a path to having an integrated custom programmer module.

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. 15. We are measuring a floating-gate pFET device through the drain current switched through the programming crossbar infrastructure. Measuring current typically requires a conversion from current to voltage, and we require potentially 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 well voltage to floating-gate 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. Figure 15 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

$$V_{out} = 2(V_{dd} - V_{fg}) \tag{15}$$

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

We look next at the resulting type S curves, taken from a FG switch element, looking at this V_{out}, which is directly related to V_{fg}. Figure 15 shows a measurement of V_{out} after an injection pulse (always for V_d to 0V for a 6V 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 pulse width of 10μs. We measure the output through the ADC, which follows the linear relationship

$$V_{out}[n] = 0.0001602a[n] + 0.3490 \tag{16}$$

where a[n] is the 14 bit integer code (0 through 16383) measured in the ADC (We can keep all codes within the 14 bit code on the 16 bit processor). Figure 15 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). At each step, we should be exponentially decreasing the percentage change needed for the target, improving one bit of accuracy per iteration. Many combinations of V_g and V_d schemes are possible for the algorithm.

Looking at the formulation of the experimental data in Fig. 15, we want to understand why we get this useful formulation. We see for the S curves that we have exponential growth in FG voltage (exponentially growing from an unstable equilibrium) for subthreshold and near threshold measurements, and exponential convergence (exponentially decreasing towards a stable equilibrium) for higher above-threshold measurements, so we are not surprised that we have two potential exponential functions that are expressed by linear difference functions. In general, the difference equation is of the form

$$x[n+1] = Ax[n] + B, \tag{17}$$

which results in an exponential solution either away from (if A $>$ 1) or towards (if A $<$ 1) a steady state solution ($-B/(A-1)$).

The dynamics are caused by hot-electron injection at the FG, which can be modeled as [26]

$$C_T \frac{dV_{fg}}{dt} = -I_{inj0} e^{-\kappa\alpha\Delta V_{fg}/U_T} e^{-\Delta V_d/V_{inj}} \tag{18}$$

where C_T is the total capacitance at the floating-gate node, I_{inj0} is the injection current at the V_{fg} and V_d bias point, α is $1-U_T/V_{inj}$, and ΔV_{fg}, ΔV_d are the changes in V_{fg}, V_d from that bias point. In this case, we use the same V_d for all pulses, so ΔV_d is zero, although we will use this formulation in the next subsection for precission FG targeting. Moving the transition from $\Delta V_{fg}[n]$ to $\Delta V_{fg}[n+1]$ at iteration n requires integrating the above equation. The FG voltage changes a small to moderate amount per pulse in our experiments, defined as $|\Delta V_{fg}[n+1] - \Delta V_{fg}[n]| < U_T/\kappa\alpha$. We get the resulting expression in this case

$$\Delta V_{fg}[n+1] = \Delta V_{fg}[n] - \frac{I_{inj0} T_{inj}}{C_T} \left(1 - \frac{\kappa\alpha\Delta V_{fg}[n]}{U_T}\right) \tag{19}$$

resulting in a linear difference equation where $A - 1 = \frac{\kappa\alpha I_{inj0} T_{inj}}{C_T U_T} > 0$.

These dynamics stretch into the above-threshold region with the inflection around 2-3μA to finish the characteristic S curve; for larger currents the resulting drop in the drain-to-channel potential dominates the overall behavior, resulting in an approximate slope (empirically from theoretical above-threshold modeling [26]) for the next order of magnitude of current as

$$C_T \frac{dV_{fg}}{dt} \approx -I_{inj0} \left(1 - 0.16 \frac{\kappa(V_{dd} - V_{fg})}{V_{inj}}\right) \tag{20}$$

$$\Delta V_{fg}[n+1] = \Delta V_{fg}[n] - \frac{I_{inj0} T_{inj}}{C_T} \left(1 - 0.16 \frac{\kappa V_{dd}}{V_{inj}} + \frac{0.16\kappa}{V_{inj}} \Delta V_{fg}[n]\right) \tag{21}$$

where we set the injection current parameter (I_{inj0}) around a new, higher level, and resulting in a linear difference equation where $A - 1 = -\frac{0.16 T I_{inj0} T_{inj}}{C_T V_{inj}} < 0$.

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

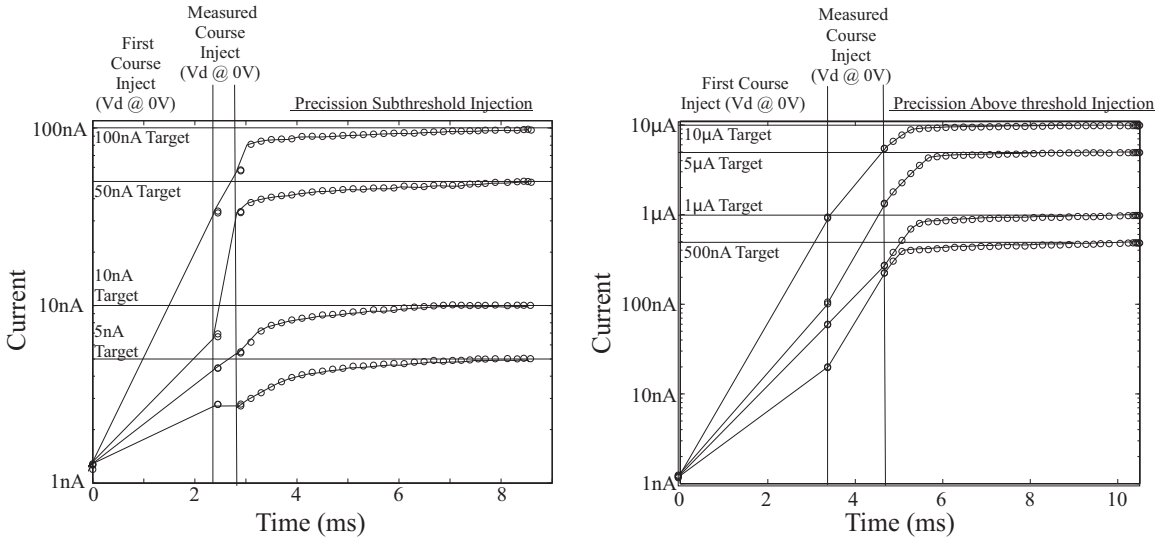
step, ready for the last sequences of fine programming. Figure 14 shows results of 4 representative trajectories where target levels are 0.5μA, 1μA, 5μA, and 10μA. Since the starting drain current and measured V_{out} are roughly the same ($\approx 1 - 2nA$), and we have good matching between these parameters 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. 15) of the target value as a function of number of pulses on a given device. We can equivalently have a longer pulse width as well as multiple pulses being roughly equivalent.

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 500μs for this open-loop programming step; a full 14 bit measurement, using a typical 25MHz clock, takes roughly 1ms to complete, so these injection measurements, even at 6V, are shorter than a full measurement. These approaches are not limited by the speed of the μP. We can take this first step without requiring an additional measurement, reducing some of the programming time.

Next, we measure the resulting device and calculate the number of pulses 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.

3.2.4 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.



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	F2	DC	D4	C6	B8
Vd (V)	0.48V	0.51V	0.54V	0.57V	0.69V	0.78V	0.90V	0.99V	1.11V	1.32V	1.53V
% change	40.7	33.4	27.3	22.4	10.1	5.51	2.48	1.36	0.61	0.151	0.032

(d)

Figure 16: Precision programming measurements for subthreshold and above-threshold currents, showing representative course injection, measured course injection (single V_d), and precision target injection. (a) 4 target current measurements for subthreshold currents and (b) 4 representative current measurements for 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 floating-gate voltage is an exponential function of drain (V_d) voltage, we can approximate the resulting drain pulse by pulling apart the floating-point representation of 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. We assume a constant V_{inj} of 150mV (typical value) 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 7 codes or 210mV.

Targeted programming in one sense is the one aspect in this work that leans heavily on previous history of FG programming algorithms, including targeted subthreshold and near-threshold devices [34], adaptive targeting of subthreshold currents [33], and

early efforts using on-chip ADC for current measurement [32]. In another sense, our approach for this phase of programming takes a different turn by constraining / considering FG targeted programming using 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 floating-gate device.

Figure 16 shows representative measured subthreshold and above-threshold target injection programming. We will compute, after a measurement, the resulting error between the target value and measurement value; we will use that error to estimate 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 DAC code that are computed related to the next few bits of the error approach. Our approach measures, pulses all devices, then repeats until error is minimized to minimize affect of voltage transients after the injection supply ramps up to, and down from, the 6V 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 100mV), minimizing the amount of FG dependance when modeling injection current as in (18). We can roughly ignore the first and higher order V_{fg} terms in (18) by centering the baseline V_{fg} for our analysis to the starting (or target) current location, both for subthreshold and above VT current biasing. The pulses are modeled as

$$\Delta V_{fg}[n+1] \approx \Delta V_{fg}[n] - \frac{I_{inj0} T_{inj}}{C_T} e^{-\Delta V_d / V_{inj}} \tag{22}$$

where $V_{fg}[n]$ is the floating-gate voltage at programming iteration n. Larger values for T_{inj} linearly increase this voltage, where we now set I_{inj0} to this phase injection current at $\Delta V_d = 0$, again assuming injection does not make huge changes through

the integration.

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. Figure 16d 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 1V change in drain voltage, due to the exponential dependance 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 drain voltage and 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) control and gate voltage control 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 FG devices. Potentially, adaptive modeling for V_{inj} during programming could be used, as initially proposed in [33], 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 0V, but the next code is typically 0.4V to 0.5V with nearly linear spacing for higher values. For a typical value V_{inj} of 150mV, a change of 450mV reduces the effect by a factor of roughly 20; having a pulse width 10 times greater ($10\mu s \rightarrow 100\mu s$) 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 one for a changing drain-to-source injection pulse with a wider pulse width.

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) 2V output voltage range, resulting from 1V shift in FG voltage for the measured device. The LSB for the 14 bit ADC results in 61μV in FG voltage accuracy, resulting in 0.166% error for subthreshold currents ($\kappa = 0.7$). For a FG device, with the total FG capacitance of a small 16fF, this results in roughly 10μV for a single electron; 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 5 to 7 codes for a standard deviation (2 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 3-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 subthreshold currents); therefore we need to employ averaging to get accurate measurements for the last 3 bits of accuracy. Averaging 4 samples results in 1 additional bit of accuracy; typically we use a maximum of 16 samples for the final accuracy.

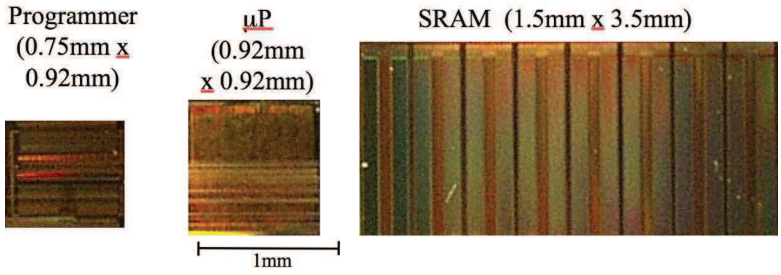
3.3 Conclusion

Figure 17 shows some of the summary particulars for this approach. Figure 17a 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 floating-gate 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. Figure 17b shows the die photographs of the key components for programming, including the programmer module (DACs, ADCs, etc.), the open source MSP430 μP, and the resulting 16k x16 SRAM block used for data and memory. The biggest issue in terms of size and power dissipation during programming is due to SRAM size and communication. Figure 17c summarizes the memory requirements, pulse and computation time for programming steps, and resulting energy estimate required for performing these steps. Figure 17d shows the extrapolation of number of FG elements that can be programmed given these time estimates, and extrapolating the resolving speed if we increase the injection supply to 7V and 8V, utilizing the faster injection efficiency at these speeds. Using a higher Vdd could drop the required energy because the processor cycles dominates the power consumed for programming.

For Fig. 17c,d the time estimates do not include the required measurement time, roughly 7ms per measurement requiring 0.5s to program a single targeted device, which in the current implementation consumes most of the resulting programming time. Practically, this limitation means we use injection supply at 6V. The V_{out} measurement requires less than 1ms for currents less than 1nA; 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 10MHz down sampled clock (from 20-25MHz processor clock). We see an opportunity in building more intelligence into the ADC measurement based on variations on the ramp function. For example, we

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

(a)



(b)

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

(c)

Figure 17: Summary particulars for the FG programming. (a) A switch list example. The first two values are the list of coordinates, in row and column, in the FG crossbar array for programming, as seen in Fig. 12. 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. (b) Die Photo of the programmer infrastructure, μP, and SRAM memory (16k x 16) in 350nm CMOS process. The area of the SRAM memory is much larger than the other two blocks. (c) Table of parameters for μP memory size, typical computation time, and resulting energy estimate for that full operation; in all cases, the energy cost is dwarfed by the energy required for the μP. All cases require a small fraction of the 16k byte program memory, using almost all of the 16k byte data memory for loading switch data. Each case is independently loaded into program memory as part of loading the programming data script.

can use the ADC as a threshold when we need a course target voltage. As another example, we can see setting the ramp between 6-8bit 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. Further, 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 approach described could eventually be compiled into (partial) custom digital hardware to further improve such parallelism with minimal

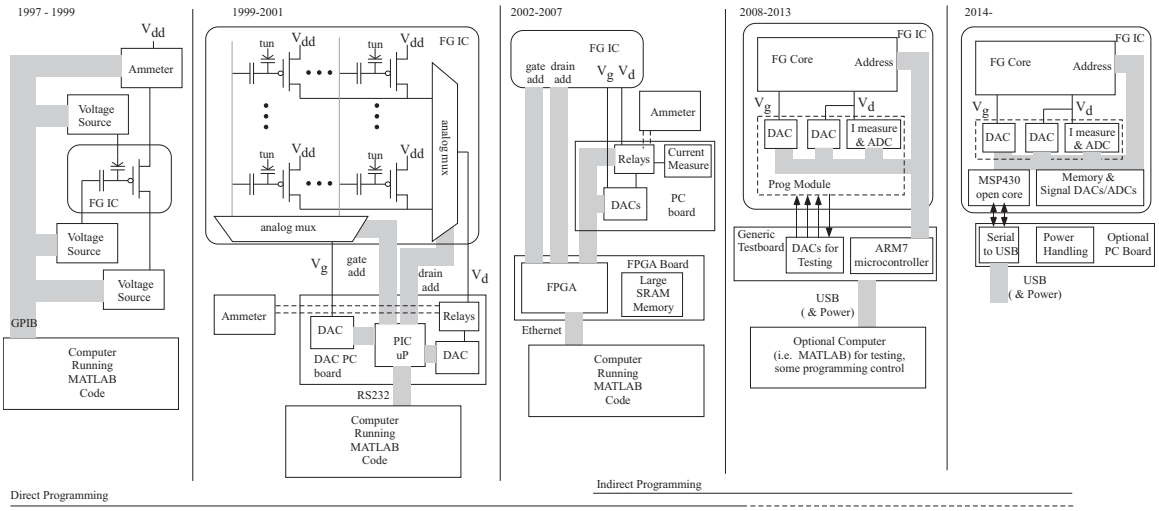


Figure 18: Pictorial History of FG circuit programming algorithms developed at Georgia Tech (GT), where we show the complexity for on-chip computation, as well as board and overall infrastructure required for a programming step. FG devices started from the original single transistor synapse learning device [21], and developed into a range of FG circuit applications (one summary in [20]). The approaches start with external bench top instruments programming a few FG devices [20], to an interface PC board with simple interface to allow computer control [33], to a PC board + FPGA board solution to perform the programming infrastructure along with MATLAB programming control [34], to having some of the circuit infrastructure on board as a programmer module [32], with an on-board microcontroller with MATLAB control of the programming algorithm, to finally 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 floating-gates routinely programmed by an order of magnitude, going from 10, to 100, to 1k to 10k to our current structure routinely programming arrays of 100k or larger, such as our current RASP 3.0 family of FPAA devices, further enabling larger and larger system application solutions. Our approach also enables using both direct and indirect programing, whether we have nFET or pFET devices.

Si die area.

This work presents the first integrated system to handle heterogeneously used and programmed FG elements in a single modular approach. Figure 18 shows the progression from the beginning of the programming approach of FG arrays, which has been a systematic march towards on-chip integration, while in parallel continuing to build structures enabling on-chip analog and digital signal processing, including configurable architectures. In all cases, we have the capability to program a general FG array, therefore requiring no predefined constraints except for 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 to the μP for programming where required). In applications requiring a μP, it often makes sense to utilize that resource directly. Typically when downloading data into an IC from a USB device, low mW power requirements for programming is rarely a concern.

This chapter focused on the IC design, integration, characterization, and algorithmic development for an Integrated Floating-Gate (FG) Programming system. We used a recent FPAA IC enabled with an on-chip processor to experimentally demonstrate this system [12]. We use hot-electron injection for precision programming of FG devices due to the nearly ideal selectivity between devices, whereas we use electron tunneling for global initialization because of their relatively poor device selectivity. 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 steps, all based on the opportunities afforded to us through the current infrastructure.

CHAPTER IV

RASP 3.0: A MIXED-MODE FG FPAA SOC

This chapter presents an integrated Ultra-Low Power System-On-Chip (SoC) Field-Programmable Analog Array (FPAA) IC enabling configurable and programmable analog and digital computation and interfacing. Figure 19 shows this IC fully integrates rapid reconfigurable analog–digital computation with configurable fabric of interdigitated analog and digital computing blocks and with a microprocessor (μP, open-source MSP430 [35]) enabling both computing and control, to address a wide range of ultra-low power embedded system computational needs. This work builds on early concepts of rapid reconfigurable analog blocks [36], as well as early concepts of integrated digital blocks in analog fabric [11]. The integration of these different concepts results in a jointly optimized FPAA performance, both in terms of high parameter density (number of programmable elements / area / normalized to process), as well as high accessibility of each of the resulting computations due to its advanced data flow handling. This IC was fabricated in a 350nm CMOS process; such approaches have recently been shown to be possible in scaled down IC processes [37].

This large-scale FPAA enables analog computational energy efficiency (e.g. MMAC (/s)/W) x1000 lower than and a die area x100 smaller than digital solutions. This capability enables low-power system computation, in the μW levels, enabling a whole range of applications, particularly always-on context-aware processors. Computational efficiency, not including the energy required for communication, is measured in equivalent Multiply and Accumulate (MAC) operations per unit time (sometimes implicit in the units) per unit power, fundamental computing operations found

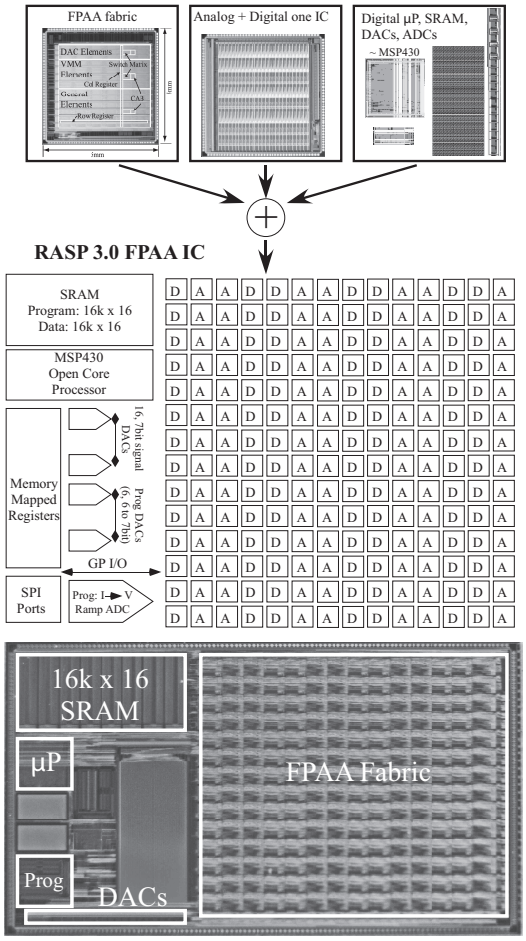


Figure 19: The RASP 3.0 integrating divergent concepts from previous multiple FPAA designs [10, 11, 36] along with low-power digital computation, including a 16bit microprocessor (μP), interface circuitry, and DACs + ADCs. The FPAA SoC die photo measures 12mm x 7mm, fabricated in a 350nm standard CMOS process. The die photo identifies μP, SRAM memory, DACs, and programming (DACs + ADC) infrastructure; the mixed array of the FPAA fabric is composed of interdigitated Analog (A) and Digital (D) configurable blocks on a single routing grid. DACs and programming infrastructure are accessed through memory-mapped registers.

in analog and digital computation. For example, both custom [38] and configurable implementations [39] of Vector-Matrix Multiplication (VMM) demonstrate 1-10 MMAC(/s)/μW power ranges, while the digital MAC energy wall remains roughly at 10MMAC(/s)/mW [40]. The saturation of computational digital computation energy efficiency [40] influenced this SoC FPAA, a representative of physical computing, reducing energy requirements for embedded system applications (acoustics, vision, communication, robotics) through x1000 energy efficiency improvement [41, 42].

The chapter focuses on the description of the SoC FPAA IC and the resulting

measurements of compiled circuits to show the resulting functionality. In each case, the focus is not necessarily the most optimized circuit design, which would be complete papers unto themselves, but showing good performance for a compiled IP block that can be routinely used. The following sections describe this FPAA IC architecture, basic analog and digital computational approaches, capacitance, timing, rapid reconfigurability of the routing fabric, implementation of data converters in the mixed mode fabric, and utilizing the routing fabric as part of the computation.

This chapter demonstrates (Section 4.2.3) the first embedded classifier structure (command word recognition) compiled onto a single FPAA device, going from sensor input (audio) to classified word, experimentally demonstrated in analog hardware; this demonstration is a small fraction of the overall IC. The system power for this compiled system on the SoC FPAA (23μW) is consistent with the x1000 improvement factor (comparison of MACs) for physical computation over digital approaches, with future opportunities for improved performance in the same IC.

Section 4.3 summarizes the SoC FPAA design, as well as presents the comparison showing the SoC FPAA as the most sophisticated FPAA device built to date. The presented SoC FPAA device maximizes both parameter area normalized to the process node, nearly a factor of 500 improvement in area efficiency as typical of other analog FPAA devices, as well as utilization and accessibility of the resulting computational resources for the data flow. The closest high utilization structure (i.e. PSoC5) has nearly a *600,000* factor less in parameter density than this SoC FPAA device.

4.1 The FPAA SoC IC Architecture

Figure 20 shows the block diagram for the RASP 3.0 FPAA IC based on a Manhattan FPAA architecture, including the array of computation blocks and routing, composed of Connection (C) and Switch (S) blocks. This configurable fabric effectively integrates analog (A) and digital (D) components in a hardware platform easily mapped

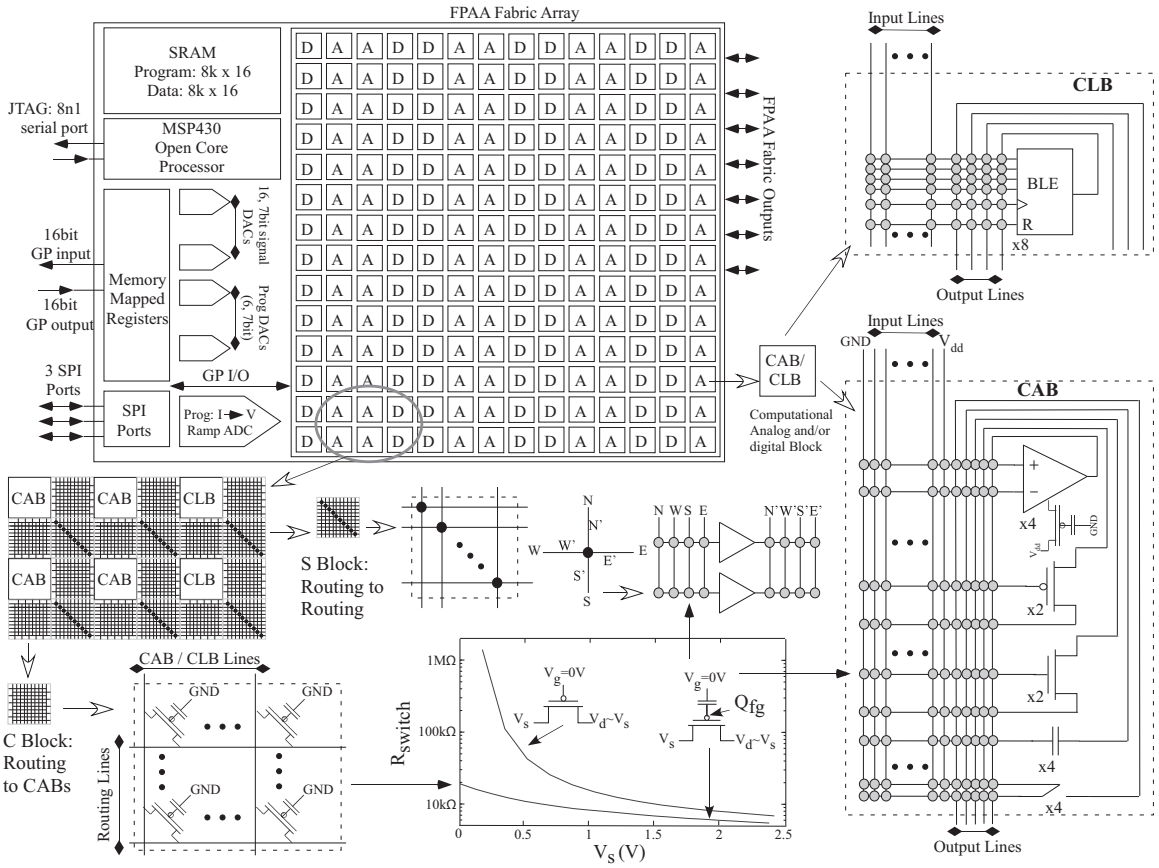


Figure 20: RASP 3.0 functional block diagram illustrating the resulting computational blocks and resulting routing architecture. The infrastructure control includes a μP developed from an open-source MSP 430 processor [35], as well as on chip structures include the on-chip DACs, current-to-voltage conversion, and voltage measurement, to program each Floating-Gate (FG) device. The FG switches in the Connection (C) Blocks, the Switch (S) Blocks, and the local routing are a single pFET FG transistor programmed to be a closed switch over the entire fabric signal swing of 0 to 2.5V [43]. The Computational Analog Blocks (CAB) and Computational Logic Blocks (CLB) are similar to previous approaches [11]. Eight, 4 input Boolean Logic Element (BLE) lookup tables with a latch comprise the CLB blocks. Transconductance amplifiers, transistors, capacitors, switches, as well as other elements comprise the CAB blocks.

towards compiler tools. The switchable analog and digital devices are a combination of the components in the Computational Analog Blocks (CAB), in the Computational Logic Blocks (CLB), and in the devices in the routing architectures that are programmed to non-binary levels. The architecture is based on Floating-Gate (FG) device, circuit, and system techniques; we present the particular FG programming approach elsewhere [15].

The interaction of analog computation, digital FPGA-like components, and a μP

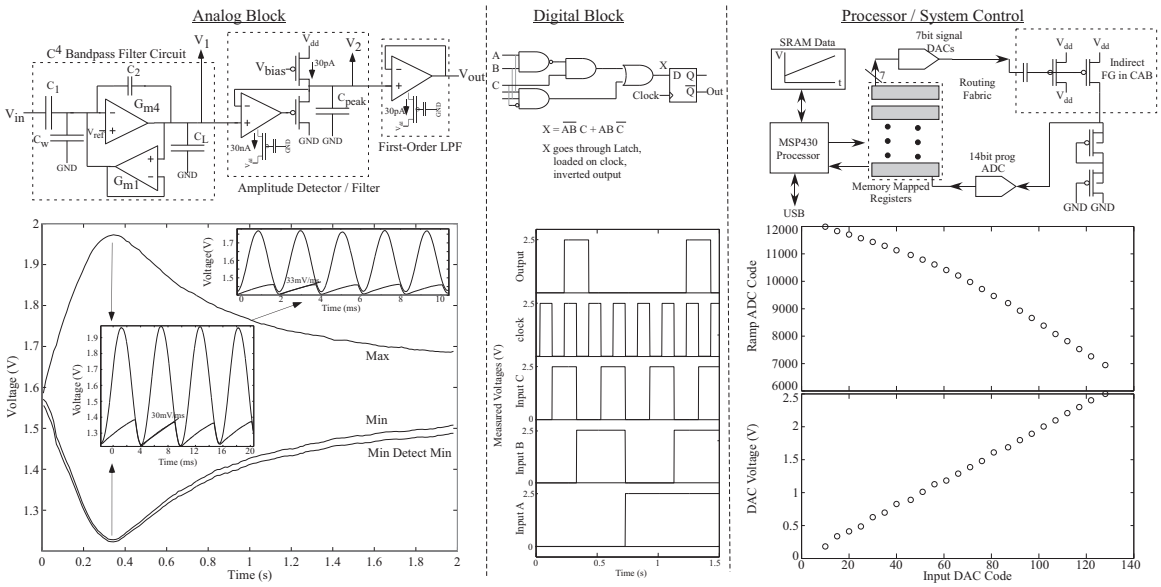


Figure 21: The SoC FPAA IC enables integration of Analog and Digital Blocks in the routing fabric, as well as standard digital computation (i.e. μP) and infrastructure. This figure illustrates demonstrates experimental measurements of heterogeneous programmable components from this FPAA IC. **Analog Block:** Circuit diagram, compilation, and experimental measurement of a representative single signal processing chain: second-order bandpass filter, amplitude detector, and smoothing filter. **Digital Block:** Circuit diagram, compilation, and experimental measurement of a representative digital function using the look-up tables in a CLB; illustrates the basic capability in a single BLE element and register. **Digital Computation / Infrastructure:** Block diagram, compilation, and experimental measurement demonstrating a complete loop using a CAB device, the μP, a signal using (7-bit) DACs, the ramp ADC used in programming (14-bit), and a memory-mapped General Purpose (GP) IO. Instrumenting and measuring analog and digital blocks requires similar loops, employing all these capabilities as part of the FPAA computation.

infrastructure coming together creates a significant co-design space between these three domains (analog, digital, μP). The analog computation combines significant innovations, enabling integration of previous heterogeneous concepts [10,11,36], from our earlier FPAA designs in ways not allowed or envisioned by the previous architectures. What is unique is the addition of digital low-power programmable and configurable FPGA fabric, first attempted in [11] (and fully integrated in this work), to fully streamline the routing of analog and digital signals through a continuous fabric. With the routing fabric and characterization, integrating these capabilities with an on-chip μP component and a range of digital communication ports completes the picture that this FPAA is a SoC computing device, not just an analog signal-conditioning device.

This FPAA further employs an open-source MSP 430 microprocessor (μP) with on-chip structures for 7bit signal DACs, a ramp ADC, memory-mapped General Purpose (GP) IO, and related components. The processor is able to send information to and from the array through memory-mapped I/O special purpose peripherals. These peripherals include 16 memory-mapped 7bit signal DACs for the architecture, allowing measurements to be performed on chip, with the data taken by and stored in the processor, as well as additional DACs (and one 14bit ramp ADC) for the FG programming. The processor supplements the processing power of the digital portion of the system and increases overall implementation flexibility; portions of a problem can be mapped to reconfigurable analog, reconfigurable digital, or a general purpose digital processor.

Figure 21 shows that our SoC FPAA approach enables integrated analog interfacing and computation with digital blocks, both FPGA and μP blocks. The analog components show the compilation of an auditory processing chain for subband signal detection. Where possible, one wants to compile key blocks into a single CAB to minimize parasitic capacitances, as well as minimize global routing requirements.

4.2 Circuit measurements on RASP 3.0

Our approach further moves away from the classical FPGA approach, in a radical perspective, because the FG devices are programmed to analog levels; our routing fabric is no longer dead weight, as we hypothesized previously [44], and fully implemented in our SoC FPAA.

4.2.1 Routing Fabric Computation

Our routing fabric is capable of partial rapid reconfigurability, while using mostly FG devices, by adding an additional set of switch configuration into the fabric. This rapid reconfigurability comes by adding a row of T-gate switches set by a shift register into the switch fabric; the I/O lines for the added T-gate row and the shift register signals

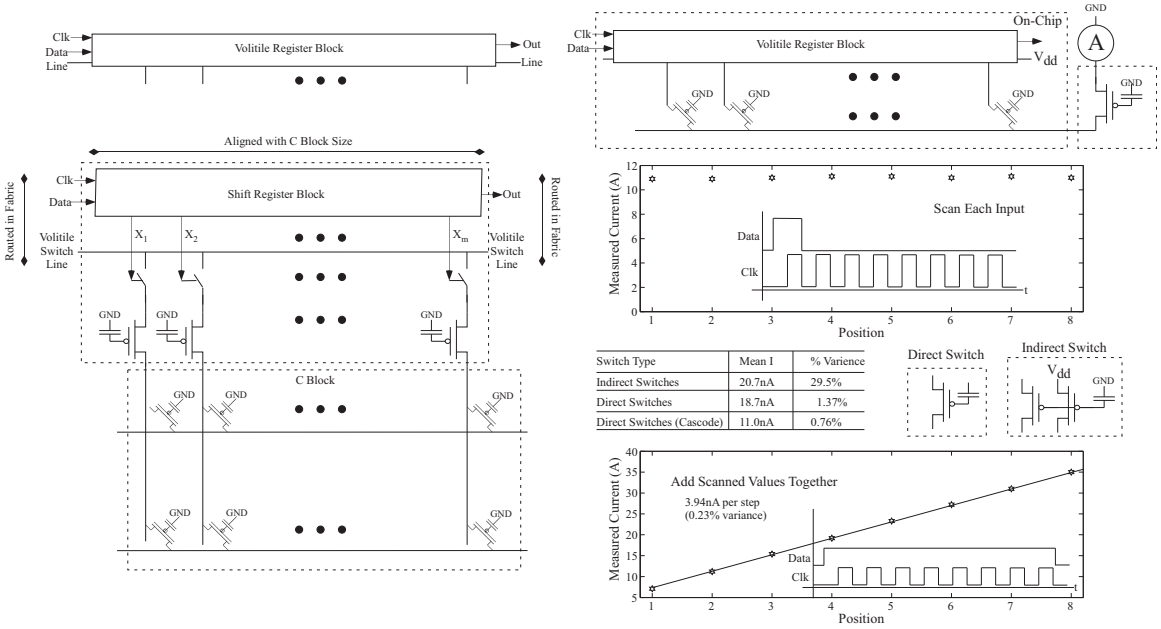


Figure 22: The FPAA SoC includes a set of T-gate based switch elements in the routing fabric to empower rapid reconfigurability. These switches are accessed through a shift register that enables rapid change of configuration on a single clock cycle; different lines of the resulting C block and/or local routing store the different configurations. The resulting switches, resulting shift register, and switches connecting the block to the routing fabric are represented as a single volatile routing block. Utilizing routing elements programmed as precise current source elements illustrates both using them as an input to the shift register to scan through the individual signals, and using them as an input to the shift register to accumulate the resulting outputs through the individual signals. It is straight-forward to imagine a range of arbitrary waveform generation based on patterns stored in routing fabric. This measurement gives a metric of programming accuracy in operational mode. The accuracy for these switches were within 0.2 to 0.76 percent for programmed subthreshold currents for uncorrected FG values; the resulting accuracy can be improved after such an initial measurement. Further, some switches in the routing fabric use only a single pFET transistor (Direct Switches), while some use two pFET transistors (Indirect Switches), where one device is used for computation and one device is used for programming. The indirect switches show characteristically higher mismatch for uncorrected FG programming due to the threshold voltage mismatch of the two pFET devices. GND is signal GND; we bias the gate terminal for the FG devices at 0.6V.

are available through the routing fabric. These volatile switches are found directly at the interface between the *C* block and the local interconnect; depending on desired higher level of abstraction, these switches may be considered as part of either block. One simple application of this technique is enabling a scan-chain for either digital or analog circuit debugging.

Figure 22 shows an added routing structure component that enables rapid reconfigurability in the FPAA fabric. These techniques minimize the amount of intermediate

data storage required for many computations, enabling data flow techniques for analog processing. Intermediate data storage often requires the largest power and complexity system cost. The rapid fabric reconfigurability can change between programmed aspects in a single clock cycle or asynchronous request–acknowledge loop. SoC FPAA shift register control signals are directed by locally routed signals in the fabric, thus determining the controlling clock (CK) and data signals. Data stored in the FG fabric would be as optimal as data stored in an off-chip nonvolatile memory without the complexity of loading the resulting computation. Figure 22 shows using the routing fabric elements, this time as a bank of parallel current sources, as well as a cascading transistor. One easily sees an Arbitrary Waveform Generator that could be compiled into the fabric; the circuit also becomes the non-volatile memory for the function, eliminating outside memory and resulting complexity and energy requirements. The measurements show the accuracy of the FG transistor programming, either in the FG voltage or resulting channel current. The measured accuracy is tighter than 1 percent for subthreshold currents, relating to less than 250μV variation.

Figure 22 discusses the programming accuracy for the direct and indirect switches, where both are available within our routing fabric, sometimes in the same C block or local interconnect block. The difference between directly programmed and indirectly programmed FG devices is whether or not current measurements are made on the circuit transistor or the injection transistor during the programming algorithm. In the direct case, both the circuit and injection transistor are the same transistor. In the indirect case, they are two separate transistors. The indirect FG device leads to a more efficient switch (fewer parasitics), but one must account for the V_{T0} mismatch between the two pFET devices. The direct FG device uses the same pFET to program, measure, and compute, eliminating any V_{T0} mismatch, but requiring additional transmission gates in the signal path for programming.

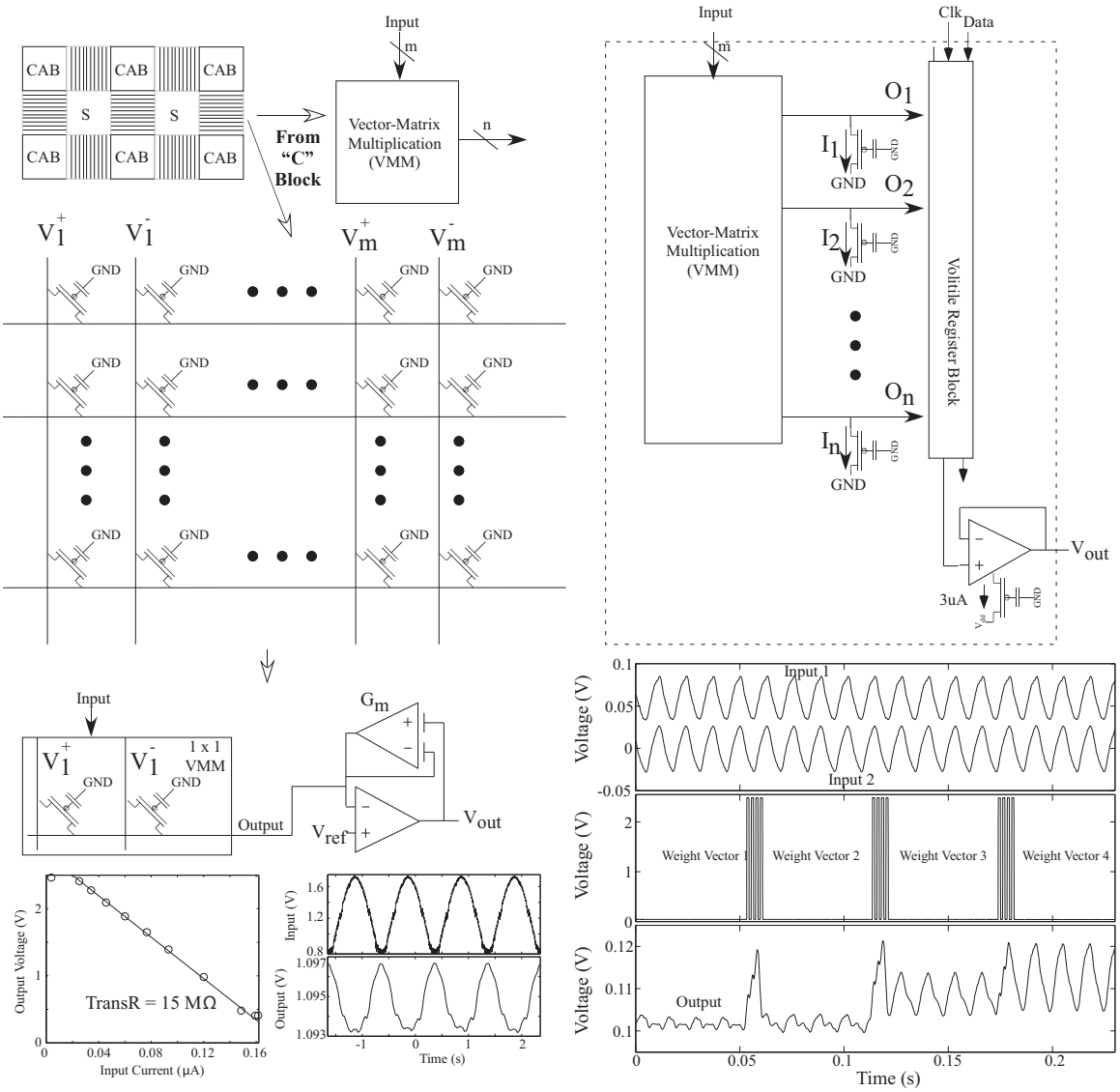


Figure 23: Vector-Matrix Multiplication (VMM) as a computational block instantiated in C Block routing fabric. The C Block forms a natural crossbar network typical for a VMM computation. The basic behavior is illustrated by the data for a single VMM element in routing fabric; two pFET transistors are required for source-input 4-quadrant multiplication. We independently measure the resulting transresistance as 15MΩ. Further, we show an application of VMM integrated with the volatile switch register block to enable rapid (single-clock) switching between weight vectors.

Computing Vector-Matrix Multiplication (VMM) solidifies the radical use of routing fabric as a computational element. Figure 23 shows implementation of a VMM in the routing fabric of our FPAA structure. We implement this functionality either in the C block or in the local CAB / CLB routing fabric, being that both structures are naturally crossbar arrays. Longer discussion on VMMs in early FPAA routing fabric

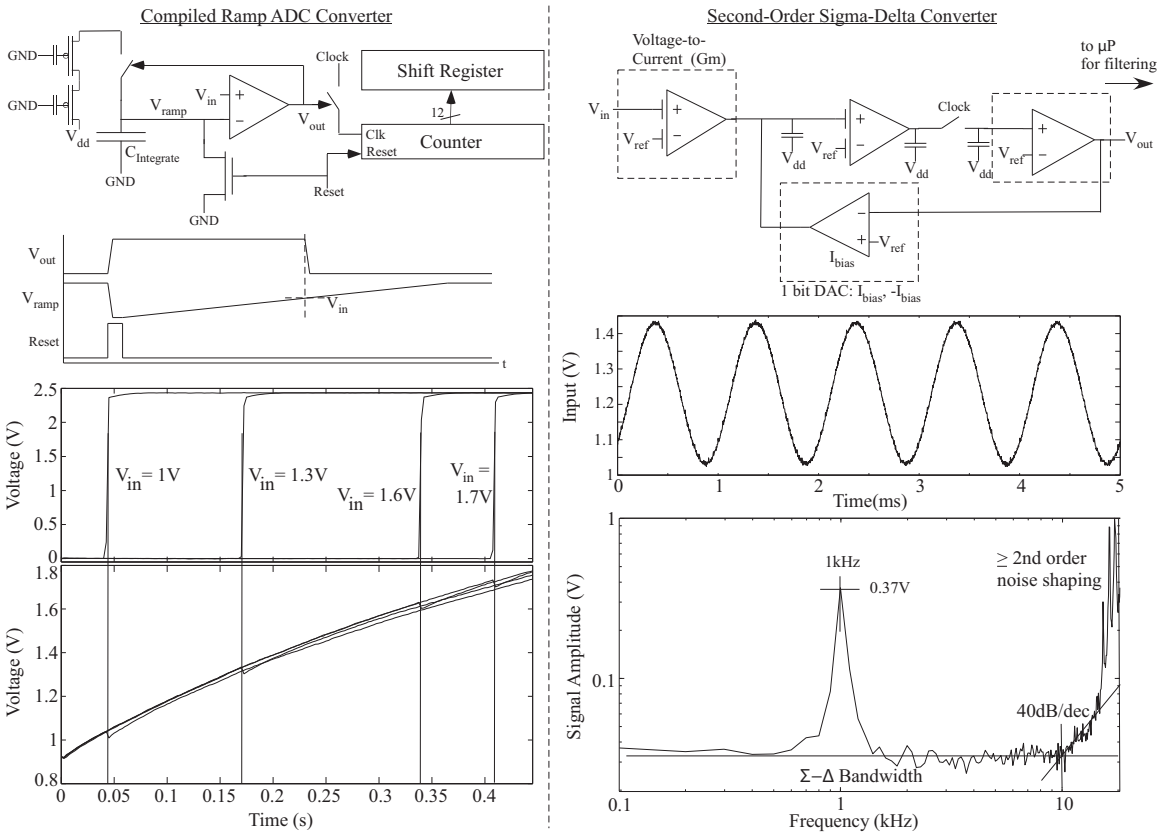


Figure 24: A compiled and measured Ramp Analog-to-Digital Converters (ADC) on the SoC FPAA in the analog and digital enabled routing fabric. **Ramp ADC**: Circuit diagram, timing diagram, and experimentally measured data for a compiled Ramp ADC. The circuit requires one CAB, 3 BLEs, and 4 CLBs (24 registers for 12-bit Counter and Shift Register). When the ramp crosses the input value, the open-loop OTA output switches. The first plot shows the OTA output voltage as a function of time for multiple different input voltages (1V, 1.3V, 1.6V, 1.7V), and the second plot shows the measured V_{ramp} voltage (through a buffer). The voltage switches nearly when the ramp equals the input voltage but with a 45mV offset. We present experimental data showing the comparison points for an approximately linear ramp input voltage. The largest systematic error is the curvature in the input ramp, generated by two FG routing switches charging up a single capacitor. The overlap capacitance to the FG reduces the effective Early Voltage of a FG device.

is described elsewhere [39]. The VMM computation occurs through the memory device, using non-volatile voltage storage, directly in routing fabric; other approaches, including traditional FPGA approaches, typically utilize memory separated from required computations. Further, we show integrated VMM and rapid reconfigurability enabling switching between metrics in the FPAA architecture. This feature further permits data flow architectures to do a particular computation right when data arrives, reducing the need for short-term storage.

4.2.2 Signal Processing Components

This section considers the behavior for some basic mixed-signal processing circuits compiled and experimentally measured in this system, illustrating basic analog-digital co-design approaches. Our first example is compiling two basic ADC devices in the routing fabric; remember, compilation in our case refers to the user starting with a high-level description of the IC and ending with an experimentally measured IC utilized in the same places as a commercial IC.

Figure 24 shows circuit compilation at the analog–digital boundary through compilation of multiple forms of ADCs as an example of integration of the capabilities. Being able to both compile an ADC and the particular needed ADC allows for optimal power computation and heavy IP block reuse, blurring system lines between analog and digital for more effective approaches of classifying raw analog data. The design of the routing fabric was not a block of analog components and a block of digital components with hard-build data converters in between, but rather a mixed fabric to explicitly allow the lines to be blurred as the application requires.

Our second example is a basic FPAA classifier using a single Layer VMM + Winner-Take-All (WTA) as a non-ADC conversion between analog and digital signals. Figure 25 shows a one-layer classifier approach based on the combination of a VMM and a k-winner Winner-Take-All (WTA) circuit [45], that elegantly compiles into routing fabric [46]. The one layer architecture can perform standard one-layer hyperplane classifiers, while also performing tasks considered impossible for typical one-layer Neural Network architectures (i.e. XOR). Figure 25 shows experimental measurements for both of these cases: an XOR function and a linear approximator function. The result experimentally verifies the universal approximator behavior of this one-layer VMM+WTA architecture compiled on this RASP 3.0 FPAA structure.

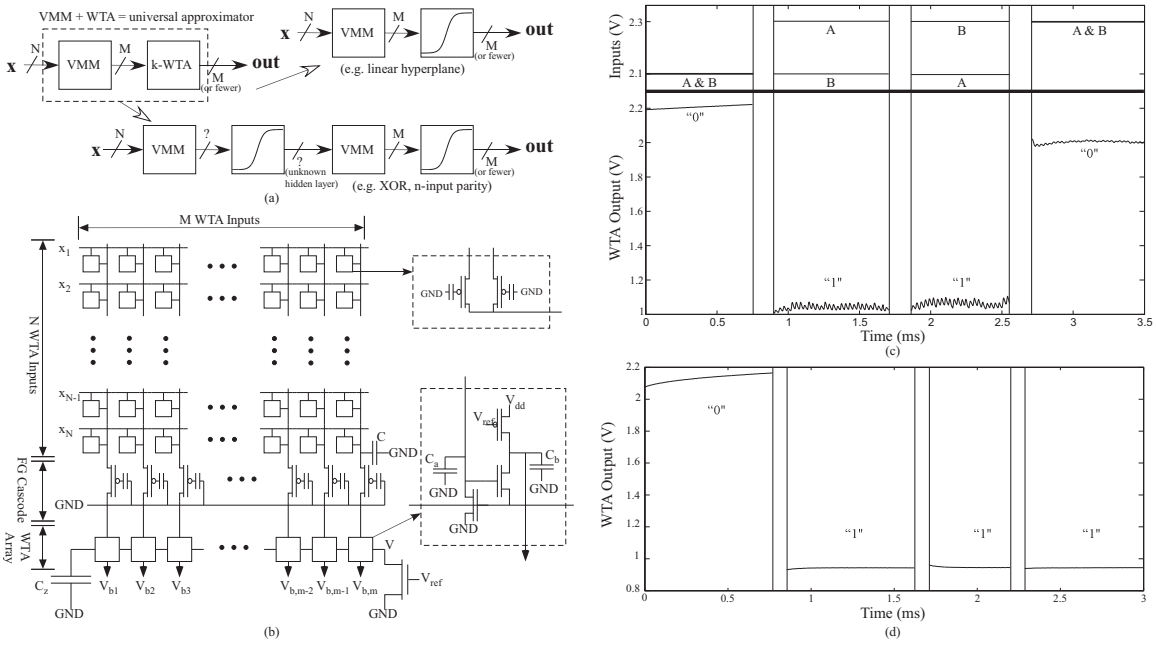


Figure 25: Instantiated FPAA classifier block based on a combination of a VMM with a Winner-Take-All (WTA) block. (a) Circuit diagram for an N-input, M-output VMM+WTA classifier block, including typical circuits compiled for the individual VMM and individual WTA blocks. The WTA circuit is operated as a single winner circuit or as a k-WTA circuit, where up to k winners are possible if their metric is above a basic threshold as originally described in [45]. (b) Experimental measurements for a compiled 3-input, 3-output 1-layer VMM+WTA classifier verifying the XOR functionality programmed into this classifier.

4.2.3 Analog Auditory Word Classification

We next show a complete analog signal processing application compiled in a SoC FPAA; this system is the first compiled sequence of signal processing algorithms shown on any FPAA / configurable device. The signal processing circuits compiled and measured on this SoC FPAA show measured data for performing command word recognition. We expect the SoC FPAA could be used for a variety of potential applications for sound / acoustics / speech, image processing and vision sensors, robotics, and wireless communication.

We show an example application of auditory / speech classification looking at detecting a command word in a sentence. Figure 26 shows the first application example of an auditory classifier structure for a limited phrase, like a command word, that can

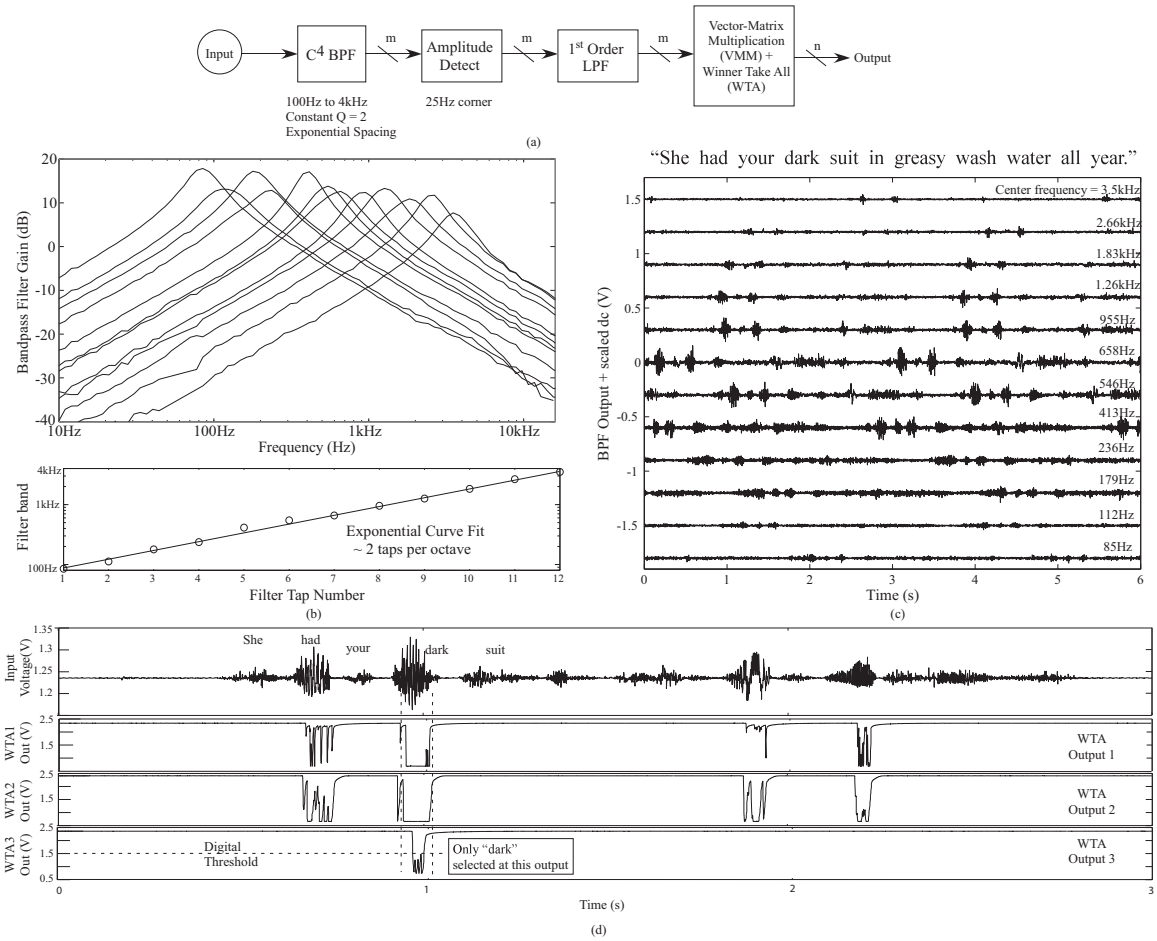


Figure 26: Analog auditory word classification application compiled into the RASP 3.0, showing the experimental waveforms from the IC. (a) Block diagram for the classifier algorithm, in a similar representation used for our tool framework. (b) BPF center frequencies that are scaled evenly on a log-frequency scale between 100Hz and 4kHz with a nearly constant Q (Q≈2). The programming approach only accounted for part of the threshold voltage mismatch variations (indirect FG devices) and did not consider effects of capacitor mismatch; additional techniques can be used for the existing IC for tighter measurements. The peak gain was the largest variation between the filters; the Q was roughly 2 with some variation. The center frequency for each of the filters, as shown, was monotonic and fairly close to an ideal exponential spacing. (c) BandPass Filter (BPF) outputs and Amplitude Detection for a single phrase from the TIMIT database. (d) Classification of word and components for a TIMIT waveform, using a k-WTA with three outputs to detect the word "dark" in the resulting phrase.

be classified through features in the averaged signal spectrum. Continuous-time spectrum decomposition used a bank of constant Q filters at the first processing stage, using a bank of amplitude detection and filtering operations, and then a VMM + WTA classifier block to classify each of the resulting spectrum into simple symbols.

Table 1: Measured Power Numbers for Compiled Command-Word Classifier Function

Computational Block	Average Power
C^4 block power (12 blocks, exp spaced from 100Hz to 4kHz)	4.86μW
Amplitude detection	3 μW
LPF (12 blocks)	3μW
VMM + WTA block	12.3μW
Total computation Power	23μW
Output signal buffers to outside IC measurement	30μW

In a more complex speech recognition system, one might have the spectrum correspond to phonemes or part of phonemes and build up the temporal representations using temporal classification (i.e. HMM classification) to word spot the resulting phonemes, syllables, and words. A simple command word application, required only to distinguish between a few simple symbols, can be directly computed as a state machine on the MSP430 processor; a next level of computation, say a simple Viterbi decoder, could be directly implemented on the MSP430 processor as well.

Table 1 shows the power required for the compiled command-word classifier computation by functional block, as well as the entire system, which requires 23μW for the current implementation. The implementation is not optimal, particularly in the LPF and classifier blocks, but shows the functional capability of the system at a very low (23μW) power level; the LPF did not filter at a low enough rate, resulting in 5kHz bandwidth signals into the VMM, while the VMM could have been operated at lower frequencies. Optimal biasing for these blocks could have resulted in a factor of 100 in the overall power budget, reducing the power by a factor of 3. We used buffers to characterize the resulting output signals off-chip; routing the signals into the digital μP would nearly eliminate the need for the buffer power. When programming the resulting system, we only partially accounted for the threshold voltage mismatch due to indirect programming mechanisms and did not program around capacitance

Table 2: Table of important SoC FPAA IC Parameters

Parameter	Value	Parameter	Value
Number of CABs	98	Number of CLBs	98
On-Chip μP	Open Source MSP430	μP clock frequency	0 - 50MHz
C block Line Cap.	160fF	S Block Line Cap.	38fF
V_{dd} (analog)	2.5V	V_{dd} (digital)	2.5V, 3.3V
V_{dd} Injection	6.0V	V_{dd} Tunneling	12V
Program Memory	16k x 16	Data Memory	16k x 16
IC Process	350nm CMOS	Die Size	12mm x 7mm
General Digital I/O	16 (in), 16(out)	SPI ports	5
General Analog I/O	125	Analog Parameters	359,014

variations. A closer look at Fig 26b shows the center frequency for each of the filters is monotonic and fairly close to an ideal exponential spacing, the Q is roughly 2 with some variations. The peak gain shows the largest variations due to capacitance (C_2, in Fig. 21) mismatches, where this capacitance is the small parasitic capacitance between the OTA terminals resulting in more device to device variation. The gain variation can be eliminated by modifying the weights of the VMM block. This level of programming is shown sufficient for this application, while further programming accuracy possible in the system might be required in further applications.

This classifier system, compiled on this FPAA, is consistent with the x1000 improvement factor in computation (measured in equivalent multiplication and accumulate, or MAC, operations), is similar to systems developed for VMMs (custom and compiled) [39], as well as other custom classifier networks [46–50]. The VMM+WTA classifier, being a universal approximator [46], can generate the same classification functions as other Radial Basis Function networks or Guassian Mixture Model networks [47, 48], as well as related algorithms [50]. This case shows a full system for an embedded classifier structure, going from sensor input (audio) to classified word, and further, is experimentally demonstrated in configurable analog hardware utilizing high-level design tools.

Table 3: Summary of Application Complexity of Analog and Digital Elements. The chip has 98 CLBs and 98 CABs

Measured System	CAB (devices)	CLB (devices)
C^4 + LPF + Amplitude Detect	1 CAB (4 OTAs, 1 cap)	
Digital block	0	1 CLB (1 BLE)
Ramp ADC	1 CAB (1 OTA, 1nFET, 1 switch, 2 fabric pFETs)	3 CLBs (24 registers, 1 BLE)
Sigma-Delta Modulator	1 CAB (4 OTAs, 1 switch)	
VMM + WTA (XOR)	3 CABs (1 per WTA input, VMM in local routing)	
Command Word Classifier	12 CABs (filterbank) + 8 CAB (WTA, 3transitors, routing)	

4.3 Conclusion

This chapter presented an IC that integrates divergent concepts from multiple previous FPAA designs along with low-power digital computation and interface circuitry (i.e. DACs, ADCs). We showed through discussion and measured data that this unified structure enables a wide range of SoC computing options that can be optimized for multiple parameters, showing the most sophisticated FPAA capability built to date; we hope that the success of this IC inspires additional devices built in the near future. Table 3 illustrates most of the circuits presented, compiled, and experimentally measured in this section, as well as a summary of the resources used in each case. Table 2 shows the table of parameters for the resulting SoC FPAA. Largest signal processing functions shown to date [11, 36, 51] take a small percentage of the available IC.

Figure 27 plots various FPAA devices showing the Percentage of Control Path implemented versus Analog Parameter Density. Figure 27 shows two key metrics for FPAA approaches based on a collection of published FPAA devices [10, 11, 36, 43, 51, 54, 56–63]. We define analog parameter density as the number of programmable parameters per mm^2, normalized to a 1μm CMOS node. Analog parameter density

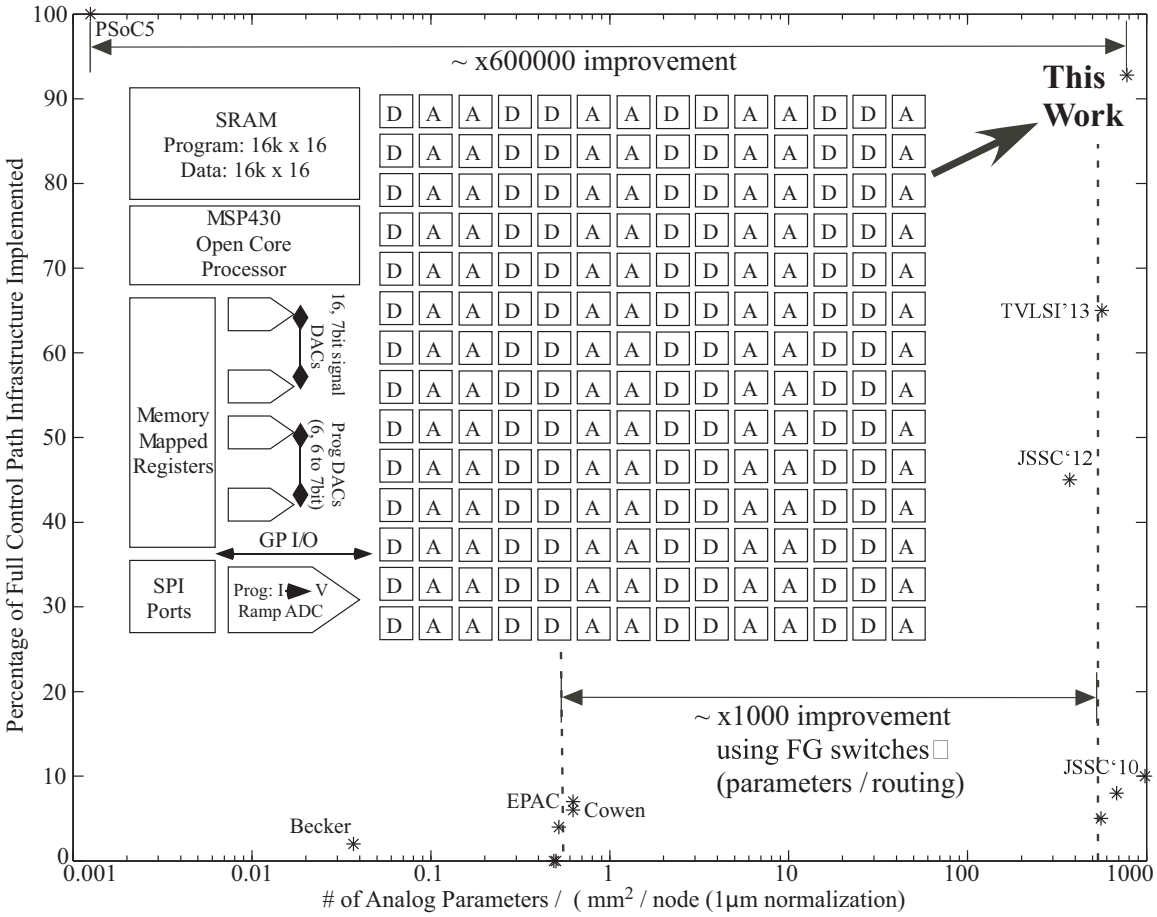


Figure 27: Using data from generations of FPAA devices, built at GT and elsewhere, various FPAA devices are plotted as the Percentage of Control Path implemented versus Analog Parameter Density. Recent FPAA ICs, like the dynamically reconfigurable FPAA or FPAADD device, begin to effectively maximize both parameters. Analog Parameter Density is the number of analog parameters per mm^2, normalized to a $1\mu m$ process (or analog parameter density). Analog parameters directly sets the complexity possible by the particular FPAA device.

determines critically the IC computation complexity, particularly when using routing as computation. Figure 27 shows FG based FPAAs enable $\approx$ 1000 parameter density improvement, providing increased computation on a single device; FG device alternatives require a DAC at every node or dynamic techniques.

Table 4 shows another comparison among FPAA devices in table form, as an updated table following along the comparison made for one of Georgia Tech's first general purpose FPAA device [10]. Previous papers have more detailed discussion on early FPAA work, and we invite the interested reader to read these comparisons [10, 64].

Table 4: FPAA Comparison Table

Ref.	Process Area	Num of CAB/CLB parameters	CAB/CLB elements	CAB lines	Architecture	Capability
This Work	350nm 84mm^2	98/98 359,014	OTAs, FETs, FG, T-gates, multiply, 8, 4-input BLEs	55	FG, 16-bit μP, rapid reconfig, Manhattan	Ana./Dig. Circuits, Full SP
[11]	350nm 25mm^2	108/108 80,000	OTAs, FETs, T-gates, 4, 3-input BLEs	20	FG, Manhattan	Ana./Dig. Circuits
[36]	350nm 25mm^2	78/ 0 76,000	OTAs, FETs, FG	30	FG crossbar	Ana. circuits 1 stage SP
[10]	350nm 9mm^2	32 / 0 50,000	OTAs, FETs, FG	40	FG crossbar	Ana. circuits
[51]	250nm 100mm^2	16 / 0 416			log-domain SRAM crossbar	ODE simulation
[52], [53], [54], [55]	130nm 1mm^2	7 / 0 58	1 Digitally tuned OTAs	4	Minimal OTA routing	Basic OTA circuits

The table shows the impact of the FG-based FPAA devices, compared against other approaches, the most advanced of the non-FG devices being presented by Cowan, et. al. [51]. The highest frequency response devices by Becker, et. al, operate at expected frequencies given the IC process, but otherwise, are extremely simple in structure and capability [52–55]. Although one possible second metric is maximum measured frequency, normalized to $1\mu m$ process, the maximum analog frequency response being directly related to process technology for devices from 1μm CMOS to 40nm CMOS.

Designing an SoC FPAA devices requires maximizing both metrics, so that we have a large number of programmable parameters and resulting computation, as well as having the infrastructure to get data communicated to these processing devices. Our second metric is to describe the amount of control flow (mostly digital) relative to the amount of analog and digital data flow capability. Practically, the ability to get data to all of the processors can be a primary limitation for a series of application spaces, such as image processing, where data does not always arrive in the desired

order for the computation. Recent RASP based FPAA designs [11,36] have started to focus on improving this second metric while not losing the analog parameter density efficiency. The presented SoC FPAA device maximizes both metrics, being nearly a factor of 500 improvement in area efficiency as typical of other analog FPAA devices, but with high utilization of the resulting computational resources; the closest high utilization structure (i.e. like PSoC5) is nearly a *600,000* factor improvement.

CHAPTER V

SYSTEM CALIBRATION

This chapter illustrates a calibration flow for an integrated FG programming system for a large-scale Field Programmable Analog Array (FPAA), including characterizing the FG programming infrastructure and hot-electron injection parameters in the integrated SoC FPAA, calculating the EKV model parameters for the golden FETs, calibrating the compiled DAC and ADC blocks that interfaces between the on-chip μP and compiled analog circuits in the array.

5.1 Calibration on digital / analog systems

Digital system design is enhanced when an algorithm can be directly ported to any number of equivalently designed systems, with effectively the same performance for all devices. Although digital SoC systems require a calibration (e.g., a clock speed, bad memory blocks, internal voltage regulators) and precision components (e.g., a clock crystal, oscillator), this process is independent of the algorithm, performed away from system programmers.

One rarely expects this property in analog systems, even when some form of programmability is possible. Every system is handled in a special way; a mismatch is the primary limiting factor for analog systems (e.g., [65]) resulting from the fact that "not all transistors are created equal."[1] Typically an ADC and filters (e.g., Gm-C topologies) utilize programmable elements to deal with mismatches; larger analog systems significantly effect larger levels of algorithm modification. One can reduce calibration via an increased device area to reduce mismatches, resulting in a larger die

[1] [66], Chapter 5, p. 72

area and cost, implying higher power consumption as well as lower levels of system integration.

This chapter describes bringing analog computation towards the expected (digital) system techniques, where a one-time calibration of a batch of devices enables the same algorithm at similar performance levels to be downloaded to all devices. Figure 28 illustrates the concept of enabling algorithms to be directly downloaded to a large number of FG analog programmable and configurable ICs using a single calibration flow. Our primary need for calibration is to account for the threshold voltage mismatch (V_{T0}) between two pFETs for indirect FG programming [67], where previous characterization initially shows V_{T0} mismatches between these devices [68].

5.2 FG FPAA Architecture, Programming, Compilation

The infrastructure for FPAA systems has been integrated onto a chip to increase area efficiency, as well as analog parameter density [70], [52] to enable more complicated applications [12], [71]. Figure 29 shows the PCB and IC level architecture of the latest version of the FG FPAA family [12]. The IC comprises an FPAA fabric array, an FG programming infrastructure, a μP (open-source MSP430 [35]), and 16 k $\times$ 16 SRAM. The FG programming infrastructure includes a 7-bit gate DAC, a 7-bit drain DAC, a pFET diode I-V converter, and a 14-bit ramp ADC, interfacing with the μP through memory mapped registers.

The PCB consists of power components regulating 2.5 V / 3.3 V, charge pump units handling high voltages (6 V / 12 V), and Input/Output pins for external connection (to be used with voltage generators, voltmeters, ammeters, etc.). Some of the external pins are connected to the array to provide direct input or enable measurements, and some are connected to the FG programming infrastructure in calibration mode.

The FPAA array includes Computational Analog Blocks (CAB), Computational

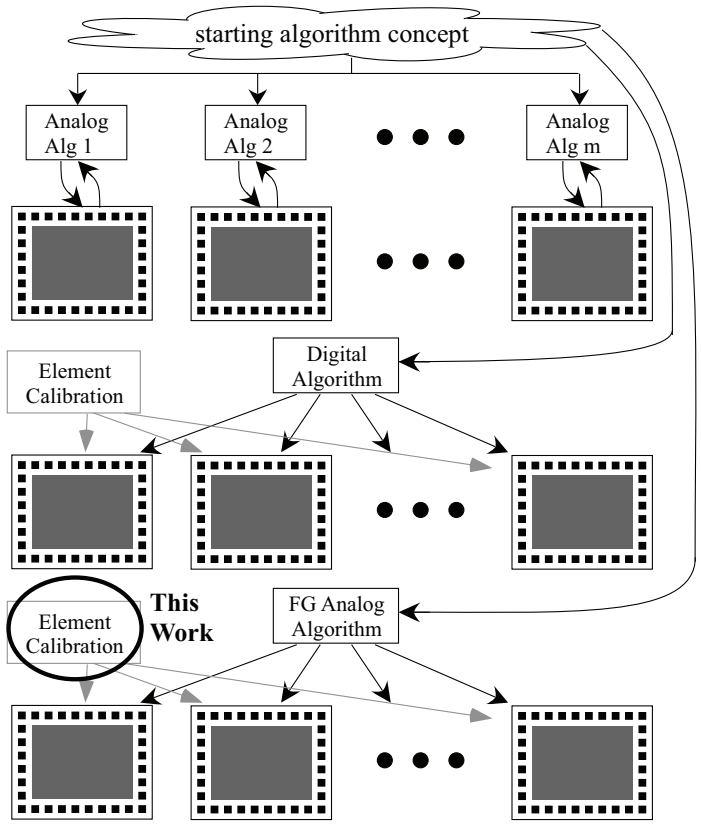


Figure 28: Separation of calibration and algorithm enables the same algorithm implementations at similar performance levels in both digital and analog systems. Digital systems enable a single algorithm directly downloaded to a large number of ICs (m), however classical analog systems need each algorithm to be *tuned* for each particular application. The digital approach, especially a digital SoC system including μP, SRAM and analog components (e.g., a clock crystal, oscillator [69]) and providing several V_{dd} for low-power consumption, requires a calibration on a clock speed, bad memory blocks, and internal voltage regulators, as well as precision components due to the mismatches [42], whereas this process is independent of the algorithm. This work focuses on developing a single calibration flow to bridge the gap, enabling algorithms directly to be downloaded to (m) FG analog programmable and configurable ICs.

Logic Blocks (CLB) and routing switches composed of Connection (C), Switch (S) and Input/Output (I/O) blocks. Each CAB includes local routing switches for connecting the inputs/outputs of a CAB to its elements such as Operational Transconductance Amplifiers (OTA) with and without FG inputs, nFETs, pFETs, capacitors, and T-gates. Each CLB includes local routing switches with BLE lookup table circuits. FG switches can be be used for computation (e.g., VMM) as well as for connections between CAB/CLB/IO blocks.

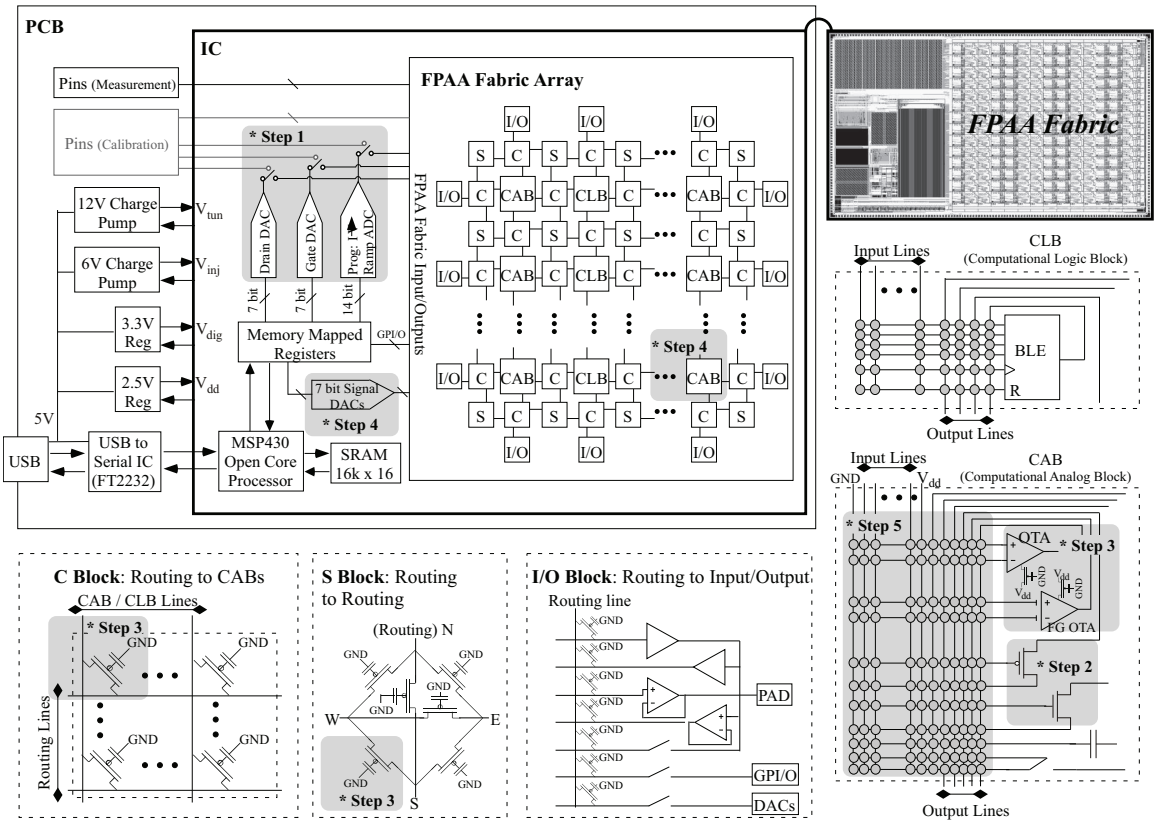


Figure 29: The FG FPAA system interface between the on-chip μP and external devices (e.g., computer / tablet) is a USB, which provides the system power (5 V) as well. The PCB includes voltage regulators for the power supply (2.5 V / 3.3 V) to the IC, charge pumps to generate 6 V and 12 V for the injection and electron tunneling, and pins for a measurement or calibration. The IC consists of a μP, 16 k $\times$ 16 SRAM, an FPAA fabric array, and an FG program infrastructure comprised of a 7-bit gate DAC, a 7-bit drain DAC, an I-V converter, and a 14-bit ramp ADC. The FPAA fabric array is composed of Computational Analog Blocks (CAB), Computational Logic Blocks (CLB), Connection (C) blocks, Switch (S) blocks, and Input/Output (I/O) blocks. "$*$" indicates each calibration step in Fig. 31

Figures 30a and 30b show the compilation flow from designing a high-level application in Scilab/Xcos (open-source programs similar to MATLAB/Simulink) by a user to measuring the output. When the user compiles the design, each chip's calibration information is integrated with it. As shown in Fig. 30, a switch list refers to an FG V_{T0} mismatch table, an input vector refers to a calibrated DAC table, program assembly codes (prog codes) and lookup tables for programming refer to FG device parameters and program infrastructure characterization tables. These generated files

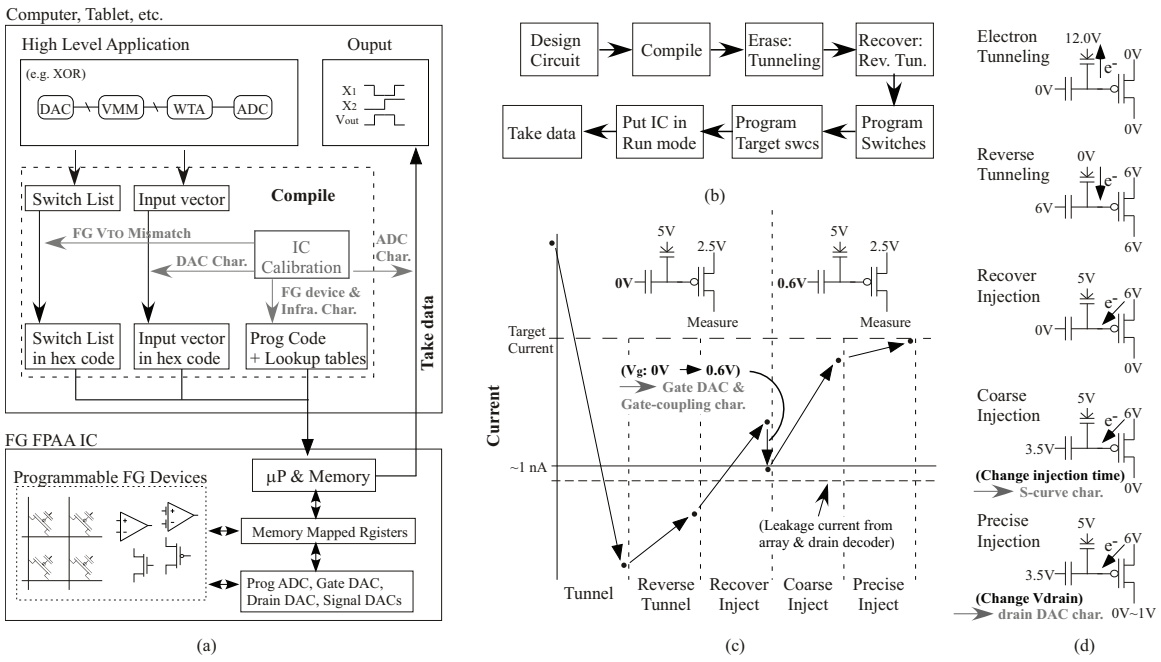


Figure 30: The design and test flow includes the compilation and programming of FG devices. (a) The design compilation interfaces between high level application designs and the FG FPAA IC. A circuit designed by a user in XCOS is compiled to a switch list, input vectors, and program codes, which are transmitted to and executed by the IC. The calibrated IC information is integrated into the compilation process, including converting the measured data sent by the IC to real values (e.g., voltage). (b) The system employs electron tunneling to erase and hot electron injection to program FG devices. (c) The measured current at the end of the recover injection is set to 1 nA by using the FG's gate capacitive coupling, which is characterized in the calibration flow. (d) It shows the tunneling and injection conditions. Coarse injection, which modulates the pulse width at a fixed drain voltage (0 V), requires S-curve characterization for the pulse width table. Similarly, precise injection, which modulates the drain voltage at a fixed pulse width (10 μs), requires a 7-bit drain DAC characterization.

are sent to the FG FPAA IC, which programs the switches and measures data. When the output is sent back, the characterized ADC table is used to map the hex codes to their analog values (e.g. voltages). The programming of FG devices relies on a combination of electron tunneling and hot-electron injection. Figure 30c shows a program sequence from tunneling to precise injection, and Fig. 30d shows the terminal voltage condition of the FG device for each step.

Erasing FG devices is a global operation requiring a sufficiently high voltage (12 V) on the tunneling junction of all the FG devices, which results in a low channel current (∼fA). Reverse tunneling, also a global operation, requires a lower voltage

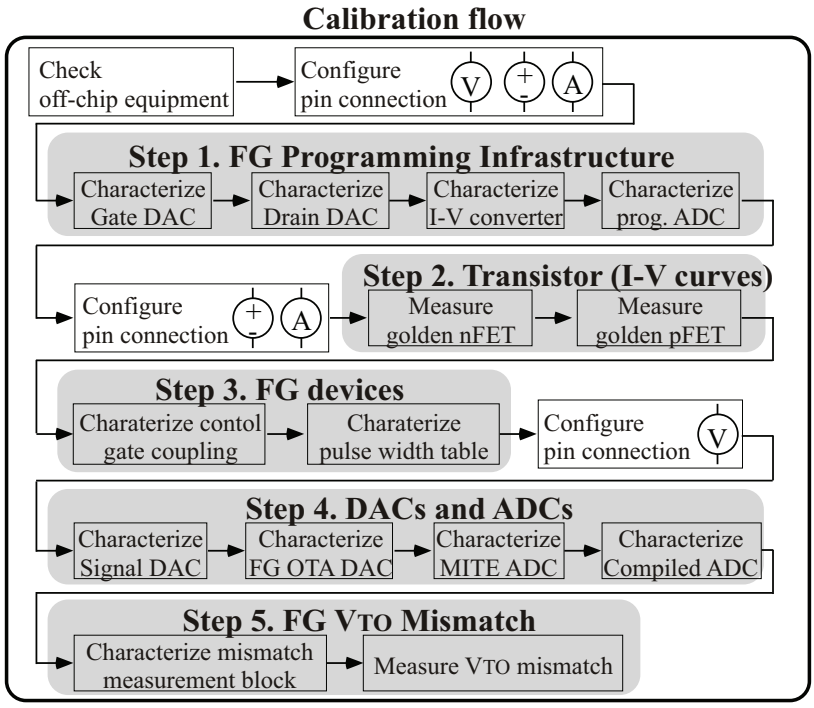


Figure 31: Off-chip equipment (voltage generator, voltmeter, ammeter) is required for steps 1, 2, and 4, but the external measurement device is no longer necessary after the calibration. In particular, the ammeter, which is large and heavy compared to the FG SoC FPAA system, is not in use after step 2. Each step has been automated to enable a mass chip calibration and then integrated into the compilation flow.

(6 V) on all the terminals of the FG device except the tunneling junction, resulting in a current of a few pA which is at a proper range for injection. During recover injection, each FG is programmed to a current of 1 nA. Since the leakage from the array and drain decoder is several hundreds pA, the current in the recover injection is measured by using the gate capacitive coupling effect of the FG device. 20 - 30 nA of current, measured in the recover injection, with V_g at 0 V, corresponds to 1 nA when measured with V_g at 0.6 V in the coarse injection, which is the next step. The effective FG capacitive coupling with a different V_g is characterized in the calibration flow and integrated into the programming algorithm during the compilation.

Hot-electron injection current (I_{inj}) in subthreshold or near subthreshold operation [25], [26] is $I_{inj} \propto I_s e^{f(\Phi_{dc})}$, where I_s is the channel current and Φ_{dc} is the drain-to-channel potential. Q_{fg} (charge on the floating-gate) ($Q_{fg} = \int I_{inj} dt$) is a

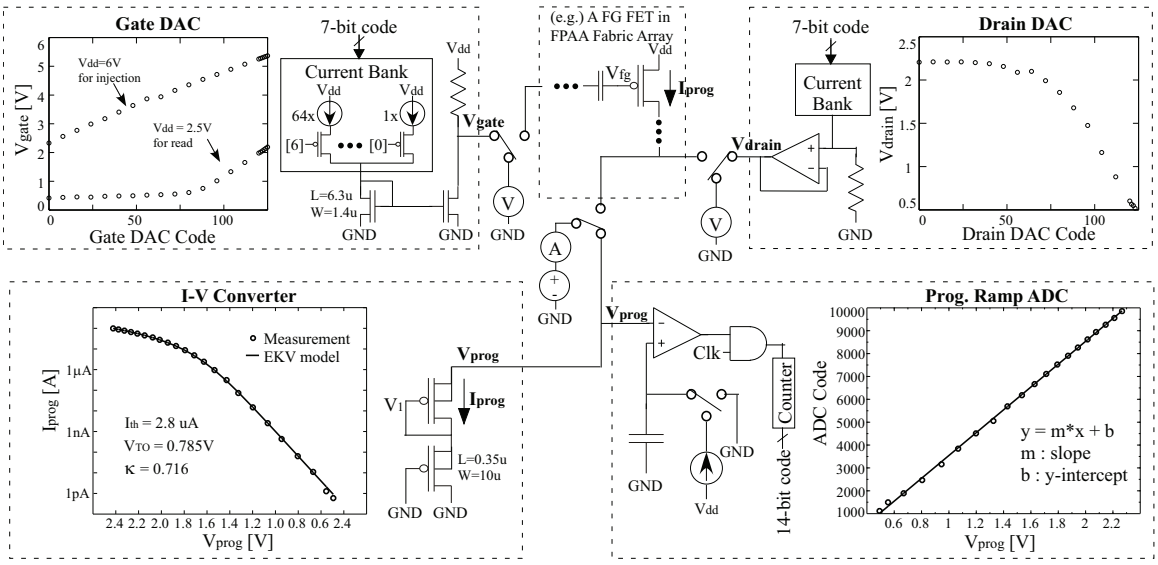


Figure 32: A characterization of the on-chip FG programming infrastructure circuits is shown. The gate DAC which converts a 7-bit code to an output voltage through a current bank is measured by an external voltmeter. With two different supply voltages (V_{dd}) for the FG injection and current measurement, the gate DAC has the output voltage in roughly 2 V to 5 V with V_{dd} at 6 V and 0.6 V to 2 V with V_{dd} at 2.5 V. A 7-bit drain DAC consisting of a current bank, a resistor, and a buffer is characterized by an external voltmeter. Body-source connected two pFET diodes convert the FG current (I_{prog}) to V_{prog} and a ramp ADC converts V_{prog} to a 14-bit code. Based on the characterization by an external voltage generator and ammeter, EKV parameters (κ, V_{T0}, I_{th}) and the slope (m) and y-intercept (b) on the ramp ADC of each chip are calculated.

function of time and voltage between source and drain. Coarse injection fixes V_d at 0 V for fast electron injection and controls the time of drain pulse, requiring characterization of the pulse width table to calculate the number of unit pulses (10 μs) to program an FG at a close range from the target current. Precise injection fixes the drain pulse width and controls the drain voltage for precise electron injection, requiring characterization of a 7-bit drain DAC.

5.3 Calibration of FG SoC FPAA

This section illustrates five steps of the calibration flow shown in Fig. 31 and shows non-linear classifier results working in multiple calibrated chips. Off-chip equipment used for the calibration step 1, 2 and 4 includes Analog Discovery for generating or measuring voltage and Keithley 6485 Picoammeter for measuring currents through

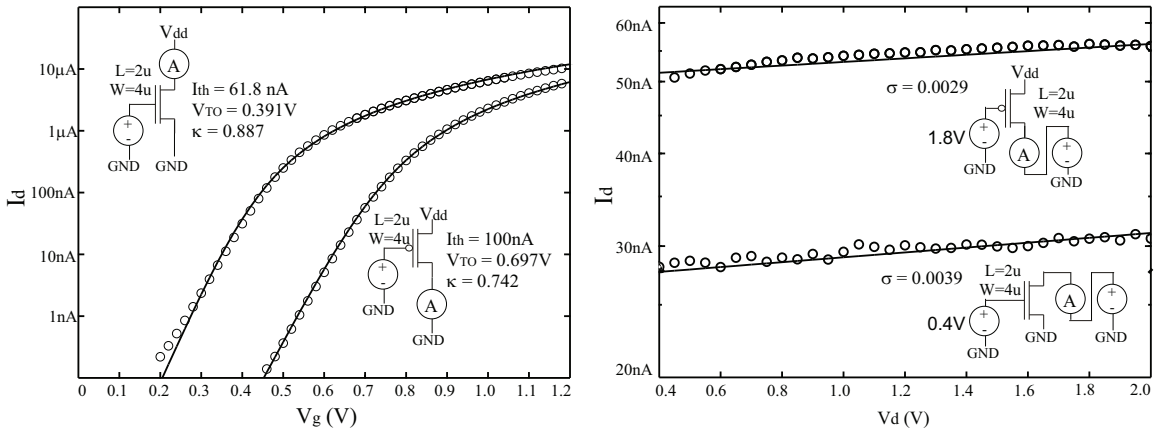


Figure 33: A golden set of nFET and pFET, compiled at a specified location in the FPAA fabric array of each chip, is modeled with EKV parameters (I_{th}, V_{T0}, κ, and σ). An ammeter is no longer required for the rest of the calibration steps or for data measurement in a user's design. V_{T0}, κ, and σ are calculated from the measured I_d–V_g and I_d–V_d data.

the external pins. The automated calibration script communicates with those external devices through a USB interface.

5.3.1 Step1: Gate & Drain DACs, I-V Converter, and Ramp ADC

The characterization of the on-chip programming infrastructure in Fig. 32 is the first step of the FG SoC FPAA IC calibration. The gate of an FG device is controlled by a 7-bit gate DAC consisting of a current bank and a resistor with a current mirror, where the 7-bit code steers currents, and the mirrored current and resistor set the DAC output voltage. A current bank includes seven kinds of current sources and seven pFETs controlling the amount of the current based on the code. The gate DAC is calibrated through external voltmeter with two different supply voltages (V_{dd}), 6 V for injection and 2.5 V for current measurements. The output voltage is in a range from 2 V to 5 V with a V_{dd} of 6V and in a range from 0.6 V to 2 V with a V_{dd} of 2.5 V. The 7-bit drain DAC has a structure similar to the gate DAC, but the resistor is connected to ground without a current mirror and it has a buffer to drive the drain line. The drain DAC is also calibrated through an external voltmeter, which has an output voltage in the range of 0.5 V to 2.2 V.

Table 5: Transistor equations

	Ohmic	Saturation
sub V_{th}	$I_{th}e^{\kappa(V_g-V_{T0})/U_T}\left(e^{-V_s/U_T}-e^{-V_d/U_T}\right)$	$I_{th}e^{(\kappa(V_g-V_{T0})-V_s+\sigma V_d)/U_T}$
above V_{th}	$\frac{I_{th}}{4U_T^2}\left((\kappa(V_g-V_{T0})-V_s)^2-(\kappa(V_g-V_{T0})-V_d)^2\right)$	$\frac{I_{th}}{4U_T^2}(\kappa(V_g-V_{T0})-V_s+\sigma V_d)^2$

Table 6: Programming infrastructure Parameters

		Chip 1	Chip 2	Chip 3
I-V converter	κ	0.716	0.707	0.699
	I_{th}	2.8μA	3.1μA	3.2μA
	V_{T0}	0.785V	0.847V	0.828V
Ramp ADC	m	4490	5709	5474
	b	-1445	-1991	-1679

The drain of FG device is connected to the I-V converter when measuring current (I_{prog}). The I-V converter consists of two pFETs that have their body connected to the source. The two pFET diode connected transistors are characterized through an external voltage generator and ammeter, which results in the I_{prog}-V_{prog} curve. When we assume that the FG transistor is matched with two pFET diode connected transistors in the I-V converter, the relationship between V_{fg} and V_{prog} [15] is given by $V_{prog} = 2(V_{dd} - V_{fg})$. The source current of the FG pFET is given in

$$I_{prog} = I_{th} \ln^2 \left(1 + e^{\kappa(V_{dd}-V_{fg}-V_{T0})/2U_T}\right), \tag{23}$$

where κ ("kappa") is the fractional change in the surface potential due to a fractional change in the applied gate voltage, U_T is the thermal voltage, V_{T0} is the threshold voltage, I_{th} is the threshold current. κ, V_{T0}, and I_{th} are calculated from the measured I_{prog}-V_{prog} curve.

A Ramp ADC, which interfaces with the μP, converts V_{prog} to a 14-bit code. The slope and y-intercept is calculated based on the 14-bit code - V_{prog} measurement. Table 6 shows programming infrastructure parameters in multiple chips.

5.3.2 Step2: EKV modeling of golden FETs

Modeling of MOSFET devices' transconductance characteristics is essential for a high level analog system simulation before the measurement. It also provides an environment to the user that does not need an ammeter. The EKV model [72], [73] is well-known as a MOS transistor model to illustrate a FET's behavior. The equation of nFET I_d in the EKV model is

$$\begin{aligned} I_d =& I_{th} \ln^2 \left(1 + e^{(\kappa(V_g - V_{T0}) - V_s + \sigma(V_d - V_s))/2U_T}\right) \\ & - I_{th} \ln^2 \left(1 + e^{(\kappa(V_g - V_{T0}) - V_d - \sigma(V_d - V_s))/2U_T}\right) \end{aligned} \tag{24}$$

σ is U_T/V_A, where V_A it the Early voltage. (24) includes all equations of the ohmic/saturation current in the sub/above threshold region shown in Fig. 33.

Figure 33 shows EKV parameters (κ, I_{th}, V_{T0}, and σ) which are extracted from the measured I-V curves taken from a golden set, compiled at a fixed location in each chip, of the nFET and the pFET. Characterizing the golden nFET and pFET means one can always figure out the relationship between current and voltage, as well as calibrate between different devices. κ, I_{th}, and V_{T0} for nFET and pFET are calculated based on I_d-V_g curves sweeping V_g with a fixed V_d and V_s. First, each starting value for V_{T0} and I_{th} is set to the x-axis intercept in a linear line extracted from $\sqrt{I_d}$ - V_g curve and twice the value of I_d when V_g is V_{T0} via a cubic-spline interpolation, respectively. Then, the optimal I_{th} to minimize the curvature of the EKV model inverse expression is found in the interval between one tenth and ten times the initial value of I_{th}, which results in κ and the final V_{T0}. σ for nFET and pFET is calculated from $\sqrt{I_d}$-V_d curves sweeping V_d with a fixed V_g and V_s. In each characterization, V_g and V_d are set by external voltage generators, and I_d is measured through an external ammeter. Table 5 shows the transistor equations of the ohmic/saturation current in the sub/above threshold region. Table 7 shows measured nFET and pFET EKV parameters in multiple chips.

Table 7: nFET, pFET EKV Parameters

		Chip 1	Chip 2	Chip 3
nFET	κ	0.887	0.781	0.856
	I_{th}	61.8nA	64.1nA	86.9nA
	V_{T0}	0.391V	0.390V	0.418V
	σ	0.0039	0.00049	0.0023
pFET	κ	0.742	0.772	0.723
	I_{th}	100nA	107nA	118.41nA
	V_{T0}	0.697V	0.714V	0.705V
	σ	0.0029	0.0022	0.0029

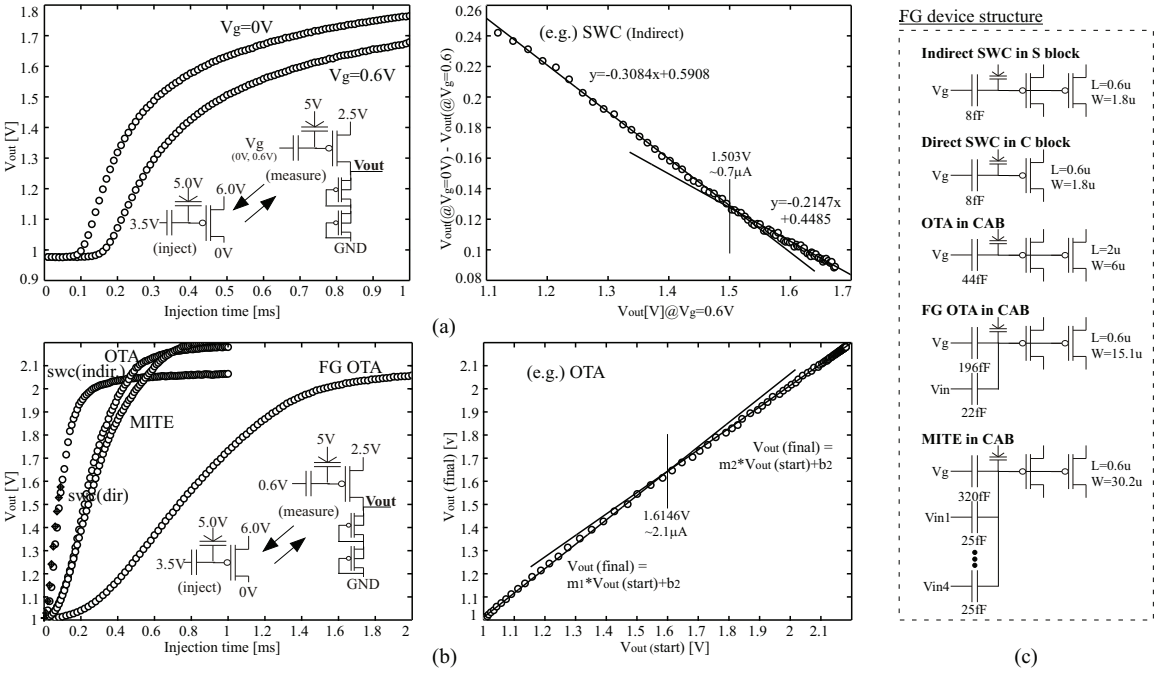


Figure 34: FG devices require a characterization of the FG programming parameters. (a) Gate capacitive coupling offsets between V_{out} measured with V_g at 0 V and 0.6 V in the injection and current-measurement loop are calculated and set for each chip's recover injection. As V_{out} increases, the offset decreases due to the increase of the MOSFET depletion capacitor. (b) S-curves are measured for the pulse width table in the coarse program. The injection-measurement loop starts from the V_{out} corresponding to 1 nA in current. The pulse width table is calculated based on the linear relation on V_{out}(final)-V_{out}(start). (c) We have five kinds of FG devices in the FG SoC FPAA. Each gate capacitive coupling offset and pulse width table for each FG device is measured in an automated calibration script.

5.3.3 Step3: Gate coupling offset and Injection characterization

FG programming parameters are calibrated without any external equipment. Figure 34a shows the calibration of the gate capacitive coupling offset required for the recover

Table 8: Gate Coupling parameters

	FG	Chip 1	Chip 2	Chip 3
ΔV_{out} @1nA	SWC (Ind.)	0.190V	0.224V	0.243V
	SWC (Dir.)	0.205V	0.268V	0.256V
	OTA	0.226V	0.270V	0.282V
	FG OTA	0.317V	0.383V	0.388V
	MITE	0.358V	0.429V	0.426V

injection in the target program. V_{out}, the output voltage of the two pFET diodes, is measured with V_g at 0 V and 0.6 V, while applying a 10 μs injection pulse with V_d at 0 V. $\kappa_{eff}(= \kappa C/C_T)$, which is proportional to ΔV_{out} measured at different V_g, decreases as V_{out} increases since the MOSFET depletion capacitance increases. The slope changes around the boundary of the sub- and above-threshold currents ($\sim$ 0.7 μA) since the current with V_g = 0 V is in the above threshold region although the current with V_g = 0.6 V is still in the subthreshold region.

Figure 34b illustrates the calibration of the coarse injection characteristic, i.e. S-curve, which is measured in the loop of injection with V_d at 0 V and current measurement with V_g at 0.6 V. The injection current in the S-curve, which exponentially grows from an unstable equilibrium for the sub / near threshold and exponentially converges towards a stable equilibrium, forms two linear lines crossing at the current of 2.1 μA on the V_{out}(final)-V_{out}(start) plot [15]. The pulse width table, which shows the number of injection pulses to reach V_{out}(final) from V_{out}(start), is calculated based on the S-curve measurement. Figure 34c shows the FG device structure in an FG FPAA array. Five kinds of FG devices exist; Indirect and direct switches for connection or computation (e.g., VMM), an FG device for OTA bias, an FG device at the input of the FG OTA, and an input bias FG for Multiple-Input Translinear Element (MITE). The gate coupling offset and the pulse width table for each FG device are calibrated respectively in each chip, shown in Table 8 and 9.

Table 9: Pulse Width parameters

	FG	Chip 1	Chip 2	Chip 3
m_1/b_1	SWC (Ind.)	0.953/0.114	0.945/0.121	0.894/0.228
	SWC (Dir.)	0.880/0.200	0.873/0.199	0.805/0.318
	OTA	1.060/-0.050	1.045/-0.036	1.026/-0.015
	FG OTA	1.081/-0.077	1.029/-0.009	1.001/0.032
	MITE	1.049/-0.038	1.021/-0.003	1.007/0.0184
m_2/b_2	SWC (Ind.)	0.930/0.145	0.938/0.121	0.947/0.111
	OTA	0.941/0.130	0.978/0.047	0.964/0.076
	FG OTA	0.973/0.059	0.944/0.117	0.924/0.166
	MITE	0.959/0.093	0.965/0.077	0.957/0.095

5.3.4 Step4: Signal DACs and Compiled DAC/ADC blocks

Figure 35 shows the calibration of DACs and ADCs, which provides a mixed-signal design environment for users and eliminates the need for external equipment for measurement. Signal DACs, consisting of a current bank and a resistor, interface with the μP through memory mapped registers. Signal DACs could be used as arbitrary waveform generators by the user. The input is compiled as a vector on the SRAM. The run-mode assembly code sends the input vector uploaded on SRAM to a memory mapped register at a given frequency. A signal DAC is calibrated by connecting V_{out} to an external voltmeter through an I/O block in the array.

An FG OTA DAC, a compiled block in a CAB to set a DC voltage, comprises an FG OTA in a unity-gain follower configuration. $V_{in}(+)$ is connected to V_{dd}, $V_{in}(-)$ is connected to V_{out}. $V_{fg}(-)$ is

$$V_{fg}(-) = V_{fg}(+) + Q_{inj}/C_T + V_{out} \cdot C/C_T \tag{25}$$

where Q_{inj} is the injected charge to the FG node, C_T is the total capacitance of the FG. V_{out} is

$$V_{out} = -\frac{Q_{inj}}{C_T}/(\frac{1}{A_v} + \frac{C}{C_T}) \tag{26}$$

where A_v is the gain of an FG OTA. A digital input DC voltage set by the user in the Xcos design is converted to a corresponding value of Q_{inj}/C_T based on the Q_{inj}/C_T - V_{out} curve, calibrated through an external voltmeter for calibration.

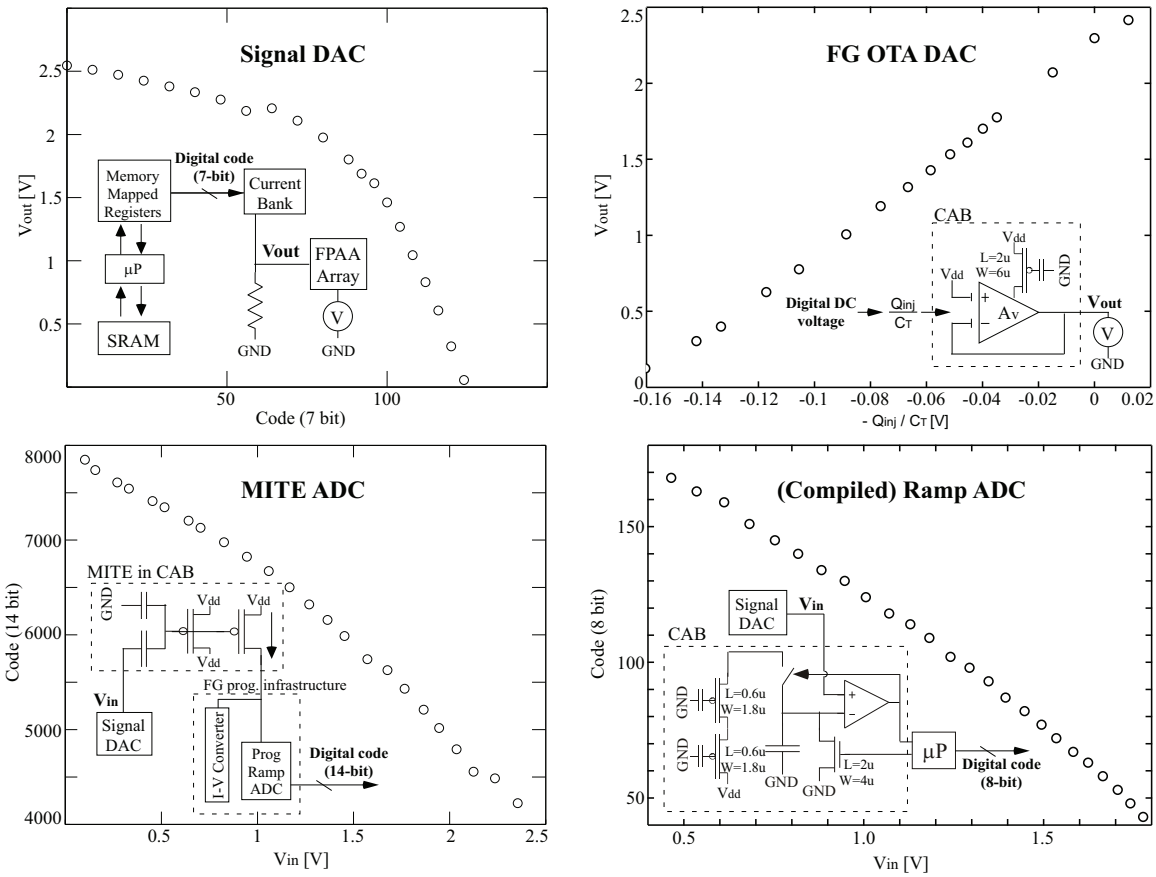


Figure 35: A signal DAC is a dedicated circuit in the IC, but other DAC and ADCs are compiled blocks in CABs. **Signal DAC:** V_{out} of 16 7-bit on-chip signal DACs are calibrated by an external voltmeter through I/O blocks in the array. **FG OTA DAC:** A feedback FG OTA in a CAB operates as a DC DAC. V_{out} is set by Q_{inj}, the offset of injected charge on two input FG nodes. The FG OTA DAC block is calibrated through an external voltmeter. **MITE ADC:** V_{in} of a Multiple-Input Translinear Element (MITE) device in a CAB couples V_{fg}, which is measured by a pFET diode I-V converter and a 14-bit ramp ADC in the program infrastructure. A calibrated signal DAC is used to apply V_{in}. **(Compiled) Ramp ADC:** A compiled ramp ADC block including two FG pFETs, an nFET, a capacitor, an OTA in a CAB converts V_{in} to 8-bit codes, interacting with the μP through GPIO.

A MITE ADC is implemented with a Multiple-Input Translinear Element (MITE) [74] block in a CAB and the programming infrastructure. The surface potential of the MITE FG pFET is capacitively coupled by V_{in}. By measuring the increase/decrease in the current through the I-V converter and the program ramp ADC, V_{in} with analog voltage is converted to a 14-bit digital code. A previously calibrated signal DAC is applied to V_{in} to minimize the use of external equipment.

A compiled Ramp ADC includes two FG pFETs, a capacitor, an nFET, and an

OTA in a CAB. The μP resets the ramp ADC by turning the nFET on and counts clock cycles until the output of the OTA is flipped from V_{dd} to gnd. The slope of the ADC depends on the capacitor's size and the bias current of the FG pFETs. The compiled Ramp ADC has an 8-bit code.

5.3.5 Step5: V_{T0} Mismatch map

A threshold voltage (V_{T0}) mismatch due to the indirect FG structure [67] and small device sizes causes errors in the analog computation. Especially since FG switches are used for computation (e.g., VMM), as well as connections between analog/digital elements, it is essential to measure and compensate for V_{T0} mismatches. Figure 36 shows a V_{T0} mismatch characterization of FG devices. The indirect pFET's drain is connected to the mismatch measurement block in Fig. 36a. A compiled mismatch measurement block includes a reference FG device, a pFET, an FG OTA DAC and an open-loop FG OTA in a CAB. The FG OTA's gain, A_V ($\sim$10), is measured by a MITE ADC ahead of the mismatch characterization. The FG OTA DAC and the FG OTA's input offset between (+) and (-) are set to have V_{out} at 1.25 V. Then, the V_{T0} mismatch, causing the difference between I_{meas} and I_{meas}(ref), is calculated from ΔV_{out}. ΔV_{T0} is $\Delta V_{out}/(A_v \cdot \kappa)$.

Figure 36b shows an example of a mismatch table. The first and second elements are the row and column address of an FG device, respectively. Each V_{T0} mismatch value in the third column is directly added to V_{fg} of each FG device, which was calculated from the target current in the switch list and will be converted to a hex code. This allows the algorithm to compensate for δV_{T0} between the two transistors.

Figure 36c shows a mismatch distribution and grayscale map before and after mismatch compensation. Due to the small size (W / L = 1.8 u / 0.6 u) of the FG device, FG devices have a wide range of V_{T0} mismatches from -35 mV to 36 mV. The mismatch table compensates those V_{T0} mismatches, as a result, the standard

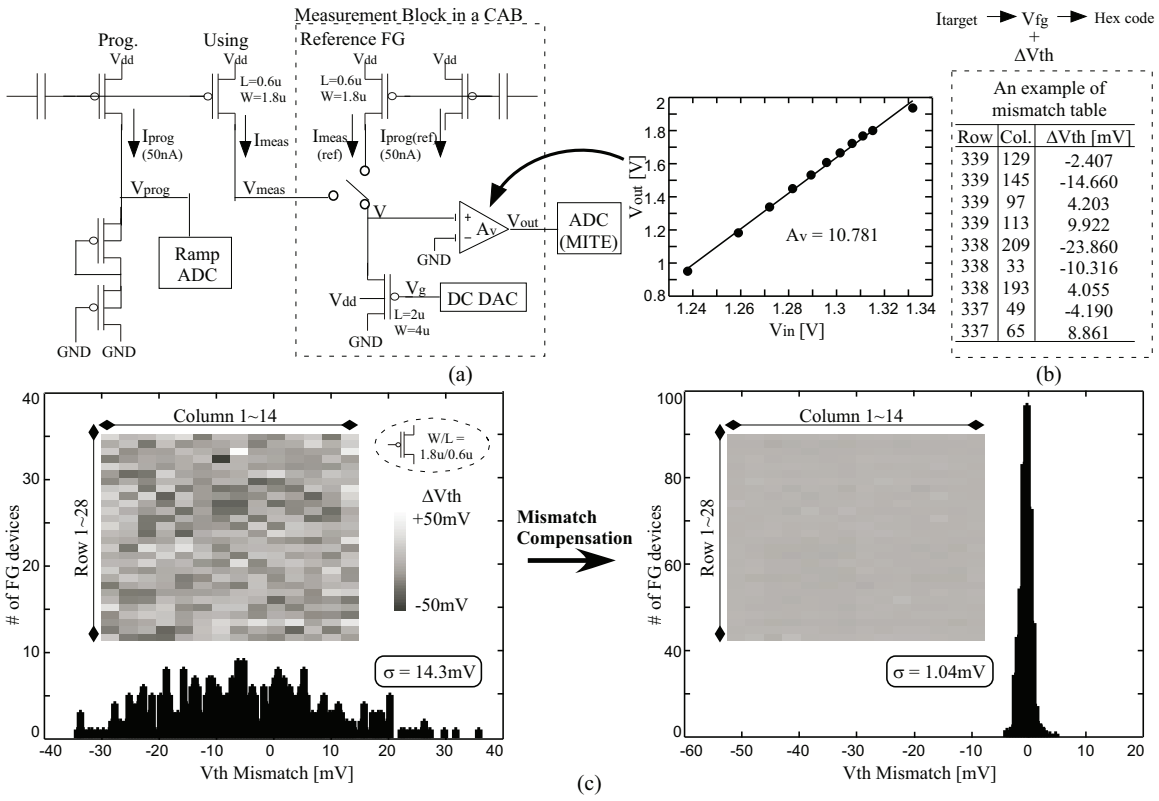


Figure 36: The characterized mismatch table compensates V_{T0} mismatches effectively. (a) A compiled block in a CAB measures V_{T0} mismatch. After FG devices are programmed at a fixed current (e.g., 50 nA), the current difference between I_{meas} and $I_{meas}(ref)$ is converted to a voltage by pFET, then amplified by FG OTA having a gain of ∼10. A V_{T0} mismatch value is calculated from the measured V_{out}. (b) In an example of a mismatch table, the first two elements represent the row and column address of FG devices. The third element indicates each V_{T0} mismatch value. (c) It compares the results of the V_{T0} mismatch compensation on 392 FG devices (14 rows X 28 columns) in a CAB. In the grayscale map and mismatch distribution graph, a wide range of V_{T0} mismatches ($\sigma = 14.3$ mV) due to the small size of FG pFETs are compensated by the mismatch map, resulting in $\sigma = 1.04$ mV.

deviation (σ) decreases from 14.3 mV to 1.04 mV. Table 10 shows that the V_{T0} mismatch compensation effectively decreases σ values in multiple chips.

A boolean function XOR using a VMM and WTA, showing a non-linear classification, is tested with the calibrated FG SoC FPAA system. Figure 37a shows the circuit, weight information, input, and expected output logic. The XOR, the third WTA's output, functions as a combination of the input voltage (X_1, X_2) and weights. The WTA drives the output low when it has a higher current compared to the other WTAs. The input voltage by signal DACs to represent "1" and "0" is set to 2.5V and

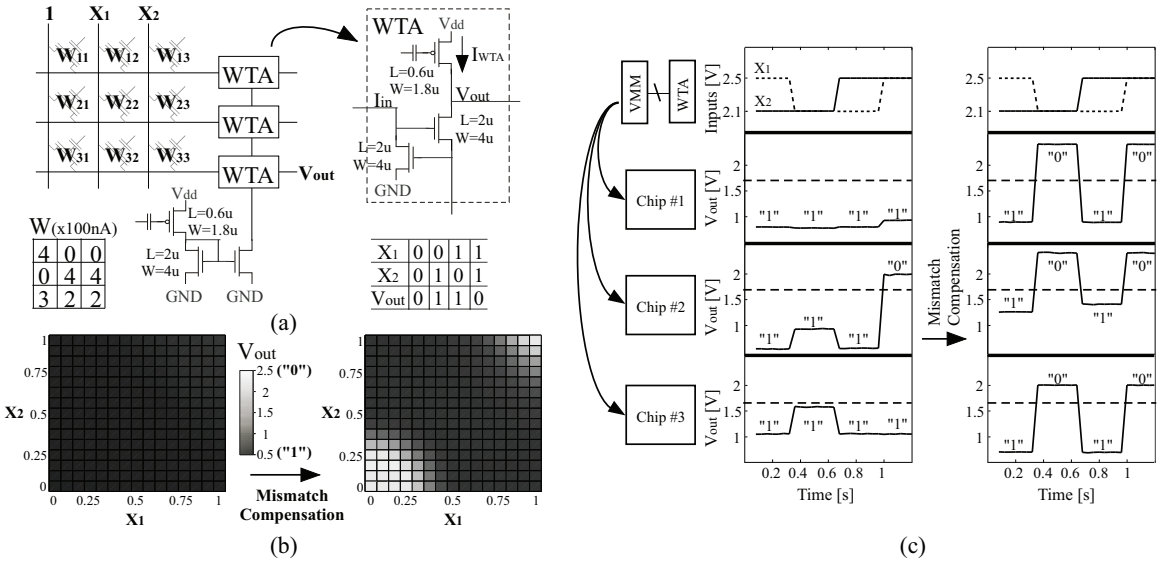


Figure 37: A nonlinear classifier is tested on multiple chips. (a) A boolean function XOR, as an example of a nonlinear classifier, is implemented with a VMM+WTA structure [46]. A combination of inputs and Weights (W) determines the WTAs' output voltage, in which the winner has a low voltage ("1"). V_{T0} mismatches on the VMM weights and FG pFETs for WTA bias currents (I_{WTA}) cause a malfunction. (b) The V_{T0} mismatch compensation integrated into the compilation of the FG FPAA system brings the decision boundary to the right operation range in the measured hyperplane. (c) V_{out} with the V_{T0} mismatch compensation shows the same results with the XOR truth table in all three chips.

Table 10: Mismatch map

	Chip 1	Chip 2	Chip 3
σ_{start}	14.3mV	15.2mV	13.1mV
σ_{final}	1.04mV	1.77mV	1.21mV

2.3V respectively. The experiment includes the calibrated on-chip DACs and ADC as an input and output, as well as utilizes the characterized programming infrastructure, FG parameters, and the V_{T0} mismatch table.

Due to the V_{T0} mismatches on the weights and pFET biases, the XOR without a mismatch compensation results in an incorrect classification. Figure 37b shows a measured hyperplane, where V_{out} corresponding to X_1 and X_2 is presented with grayscaled values. It is clear that the V_{T0} mismatch compensation enables decision boundaries for XOR function resulting in "1" when X_1 and X_2 is "1", "0" or "0", "1." Figure 37c shows results of three different ICs for the XOR classification. Results

without a mismatch compensation shows failures due to the V_{T0} mismatches, where the expected output is "1010". The V_{out} with a mismatch compensation shows the expected XOR results in multiple chips.

5.4 Conclusion

A calibration flow for an integrated FG programming system for a large-scale Field Programmable Analog Array (FPAA) has been presented. We focused on characterizing the FG programming infrastructure and hot-electron injection parameters in the integrated SoC FPAA, calculating the EKV model parameters for the golden FETs, calibrating the compiled DAC and ADC blocks that interfaces between the on-chip μP and compiled analog circuits in the array. V_{T0} mismatches due to the indirect FG structure are characterized through a compiled mismatch measurement block. A compiled classifier implementing XOR function using a VMM and WTA on different chips shows the effectiveness of the V_{T0} mismatch-map compensation integrated into the compilation flow.

In our recent work, we have been focusing on an implementation of FG SoC FPAA ICs including an on-chip FG programming infrastructure and providing a high analog parameter density [12], developing an FG programming algorithm to achieve precise target currents [15], and providing a high-level design tool supporting a graphical design environment and compiling it to necessary files (e.g., assembly program codes) [75]. The standardized and automated calibration method in the system, remained as the last piece of this puzzle, is required to enable users to design analog circuits without considering the device variation; even users with little exposure to an analog circuit and system design (e.g., users from the signal processing community) can design function blocks with abstracted blocks for a top-level design [76].

An iterative approach for measuring the input and output voltages of a VMM to find the V_{T0} mismatch based on calculated output currents was implemented in [77].

However, the iterative approach requires new calibration routine for each specific application. A calibration flow to characterize hot-electron injection parameters in a mechanical usage monitoring the system employing FG devices was shown in [78]. A previous work [68] modeled FG devices' mismatch and characterized some of the analog devices in a CAB, providing an inspiration for the fully implemented system-level automated calibration presented here. The proposed calibration method includes all necessary parts for the FG SoC FPAA system from characterization of the programming infrastructure, MOSFETs, threshold voltage mismatch, and FG devices to the compiled DAC and ADC blocks. The μP and SRAM integrated into the SoC IC simplified the calibration scripts by allowing the use of compact and efficient assembly codes, which enabled calibration at a more complicated system level. Since the calibrated information is integrated into the compilation in the analog design flow, users can focus on more complicated applications (e.g., large neuromorphic systems [42]) as if they are designing digital circuits.

CHAPTER VI

APPLICATION DESIGN TOOLS

Automated or semi-automated Computer-Aided Design (CAD) tools have played a decisive role in the development and design of digital and mixed-signal systems. Automated synthesis tools in digital systems convert a Hardware Description Language (HDL) design written by a system designer into a layout in custom digital IC or routing and logic information in Field-Programmable Gate Array (FPGA) system. In analog systems, semi-automated design tools (e.g. Cadence Virtuoso) provide numerous functions to help users to design, simulate, and verify the circuits. Although CAD tools in digital and analog systems have achieved remarkable progress for decades in each area, the application designer still does not have a unified system for digital-analog design flow; it is required to define digital and analog parts at the first stage, test each custom IC / FPGA, and integrate two parts at the system level in Fig. 38.

A large-scale Floating-Gate (FG) Field Programmable Analog Arrays (FPAA) is a solution to provide a unified mixed-signal system. FG System-on-Chip (SoC) FPAAs, including analog-digital mixed arrays and FG programming infrastructure with micro-processor and SRAM [12], have been developed. Tools for graphical high-level design environment [75] and FG programming algorithms to precisely target FGs [15] have been integrated for seamless connection to the FPAA.

The focus of this chapter is on CAD tools to bridge the gap between the high-level user design and the hardware, which are essential for FPAA productive use and development since individuals could only go as far as the tools are capable. The first requirement for the tools is to use existing, working FPAA devices. Devices have

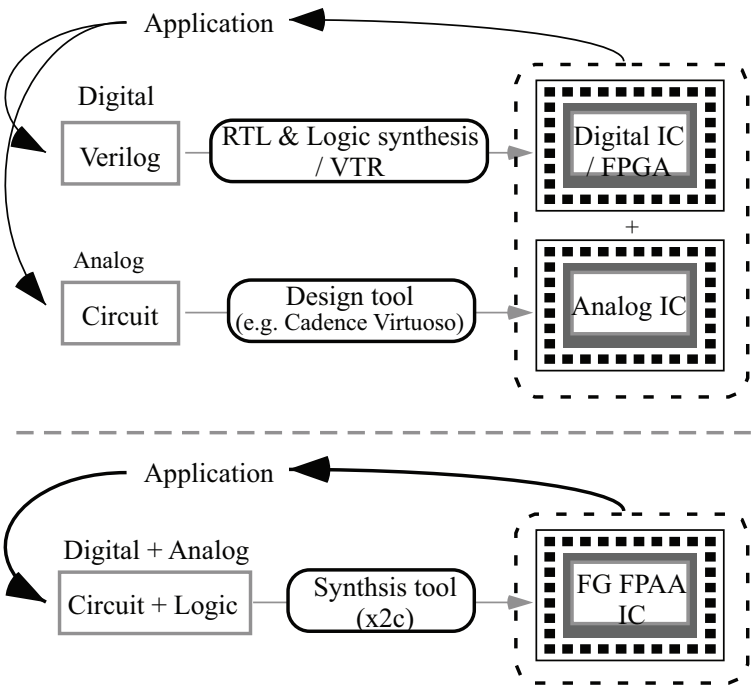


Figure 38: Comparison of two different hardware implementation flows. To implement an application (e.g. speech recognition), system designers have been taking a traditional approach to separate the algorithm into digital and analog parts, which results in using different tools and requiring efforts on digital / analog interface after testing each custom IC. Floating-Gate (FG) FPAA system provides a mixed-signal design and test flow on FPAA ICs, which is enabled by a synthesis tool, "x2c" meaning Xcos to Chip, compiling the design into necessary hardware files (e.g. switch list).

been characterized [79] and used in classes by multiple students [80] and numerous collaborators [81]. The compilation tools generating a switch list for the FG programming are based on Versatile Place and Route (VPR) due to its capability as well as a possibility of improvements and contribution from a wider CAD community. A second reason is to explore different FPAA architectures. This chapter will talk about the CAD tool for compiling on an already built FPAA [12] architecture and a graphical design environment, as it is the critical first step, where things are based around a working FPAA device.

6.1 CAD tools for Mixed Mode Design

This section introduces different CAD tools used for FPGAs and FPAAs and tool requirements for FG FPAAs.

FPGAs have look-up tables as a basic unit which has enabled reconfigurable / reprogrammable digital system design post-fabrication of ICs, for over three decades based on CAD tools' support. Verilog-to-Routing (VTR) [82], [31] is one of the open-source tools for FPGAs, where ODIN II transforms a given digital circuit described in a Verilog code to a Berkeley Logic Interchange Format (BLIF) [83] netlist, ABC optimizes BLIF netlist by synthesizing logic and performing technology mapping, and Versatile Place and Route (*VPR*) maps the BLIF to the placement of CLBs and routing track configuration on the FPGA architecture.

Similarly, reconfigurable analog CAD tools for CAB-based FPAAs have been proposed [84], [85]. A tool called Generic Reconfigurable Array Specification and Programming Environment (GRASPER) [85] is a solution for an automated place and route in FPAA systems. This tool takes a SPICE netlist as an input, places analog circuits based on Modified Hyper-edge Coarsening (MHEC) order of cells [86], and generates optimized switches.

In case of FG SoC FPAA one has to handle both CAB and CLB with an integrated processor, it has been essential to develop a new automated design flow enabling mixed-signal applications, as well as for it to be an open-source software for wider adoption / development of such tools. Although one might consider a modified version of GRASPER as an option, the tool has limitations in scalability on different hardware architectures and capability to cover digital circuits. Also, the high-level graphical interface of GRASPER is based on MATLAB Simulink, which constrains the outreach of the tool set.One might consider utilizing Verilog and extending ODIN II but the compilation tool needs to fit with the high level design tool, *sci2blif* [75], provide a graphical design environment, and convert the design to a BLIF file directly.

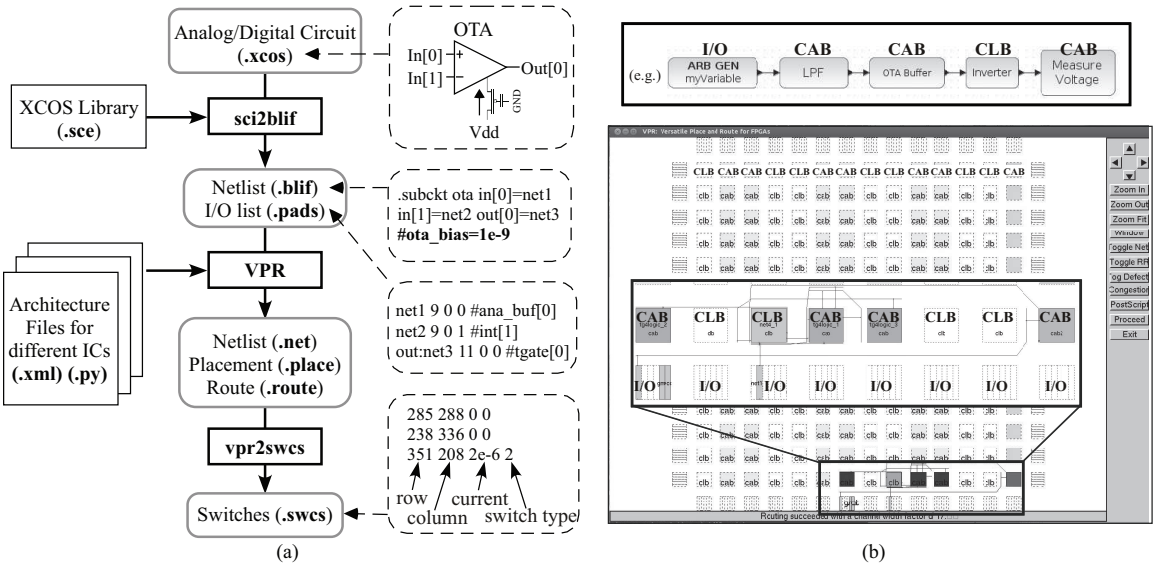


Figure 39: *x2c* compilation flow. (a) Compilation tools and the resulting files. *sci2blif* converts graphical design file (.xcos) into blif netlist file (.blif) and I/O list (.pads). *VPR* places the blocks and calculates necessary tracks for global routing. *vpr2swc* maps the FG addresses based on the place and route information. (b) Snapshot of *VPR* usage in an example. *VPR* provides a graphical interface, where a user can check the place and route results visually.

6.2 Compilation tools to generate a switch list

We provide a mixed-signal system design environment using Xcos in Scilab [87] and a Graphical User Interface (GUI) for compiling and testing the system, which are based on open-source codes. On the GUI, "New Design" starts a new application design, "Compile Design" creates necessary files for programming FG devices and built-in self test of the system. "Program Design" sends programming files to the USB-connected FG FPAA IC and programs FG devices. "Take Data" tests the system and shows measured output data. For the users who do not have access to FG FPAA ICs, "Send Email" enables testing of the design by compiling it to a remote system via a simple PoP protocol [81].

We propose a tool set to provide a high-level graphical design environment in Xcos / Scilab and also compiles it to a switch list which is transferred to the FG SoC FPAA and programmed for the application, while handling heterogeneous elements for mixed-signal circuits in different hardware architectures, as well as relying on

a completely open-source codes. Figure 39a shows the proposed compilation flow integrating three different open-source tools, *sci2blif* [75], *VPR* [31], and *vpr2swc*. In the next subsections, we summarize *sci2blif* and *VPR* and then focus on *vpr2swc* developed in this work and solutions to challenges caused by using VPR for analog design.

6.2.1 sci2blif: Xcos → blif

Xcos is a graphical system design environment in Scilab, which is an open-source software with similar functions to Matlab. A user designs a system in Xcos by dragging blocks from a palette browser, connecting blocks' inputs and outputs, and setting parameters such as bias current or DC voltage conditions. Analog / digital elements and basic functional blocks (e.g. I/O, ADC, DAC, LPF, etc.) are predefined in the Xcos library, also user-defined functional blocks can be added to the library through a macro block generation tool.

sci2blif converts block level information in Xcos into a blif netlist and a pad file, which are inputs to *VPR*. The blif netlist has a description of the block, connections, and parameters. Each FG parameter in a block is described with "♯" and "&." A pad file includes necessary input / output information and connected I/O blocks.

sci2blif also builds a file set to be transferred to SRAM and generates assembly codes to be executed by μP. For example, the tool converts the user-defined input vector to a HEX file when it compiles "ARB GEN" block, which is a DAC applying voltage values stored in SRAM at a given frequency. "Measure Voltage" block, an ADC using FG device and programming infrastructure in the IC, requires the tool creating a specific assembly code interfacing with μP for data acquisition.

Figure 40 illustrates converting from Xcos visual representation to blif files for the analog components; the digital procedures are similar, although typically simpler. Scilab saves the graph as a data structure, shown in Figure 6a, that describes the

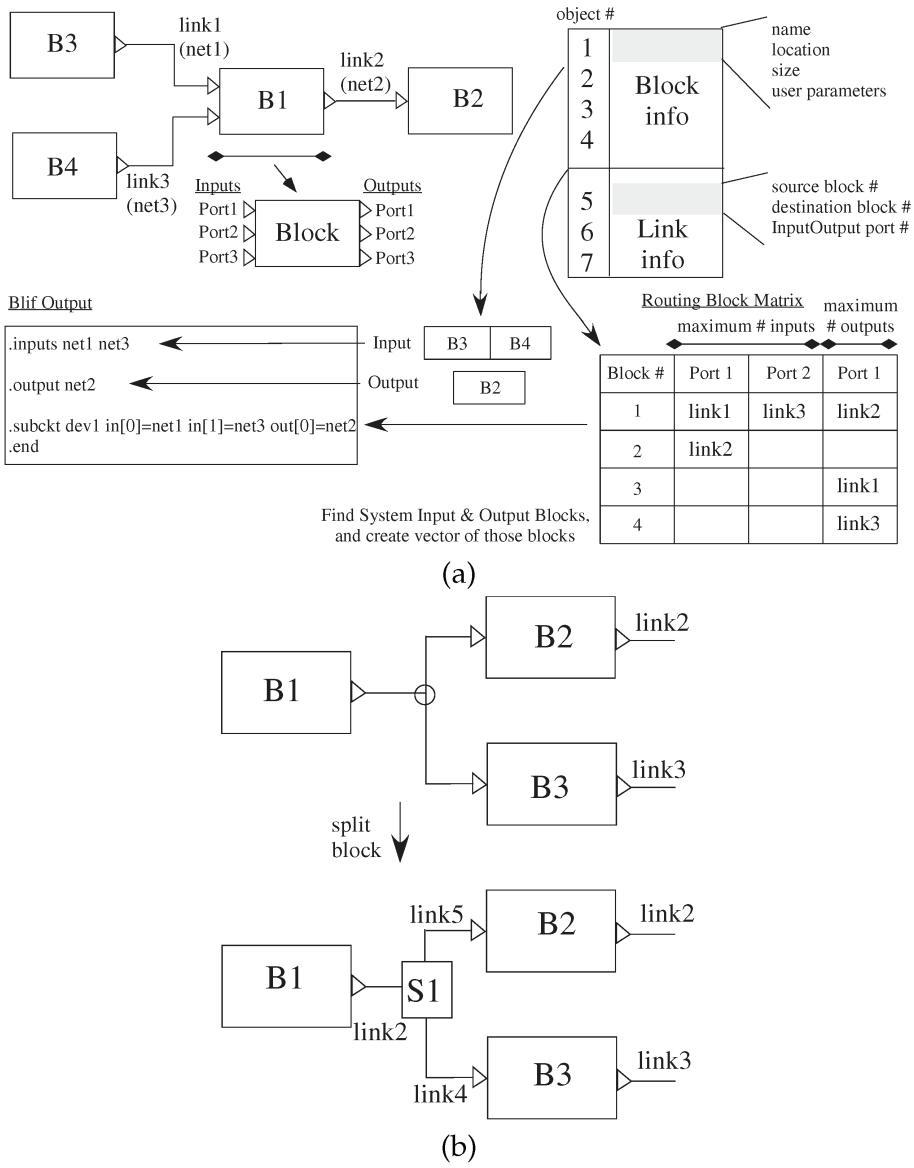


Figure 40: *sci2blif* Xcos model to blif/Verilog net list to put into VPR. (a) The data structure for a single set of blocks is an array with the block information, as well as link information. Blocks are enumerated by when they are created in Xcos; links are enumerated by where they are located on the block. This data structure transforms to blif representation for VPR. (b) The resulting data structure of the Xcos network only allows for a single input and output for a particular link; therefore requiring additional blocks included to handle when converting a single output going to multiple inputs.

Xcos file contents. The block objects are listed first, followed by the link objects, and then, they are listed by link numbers.

The high-level Xcos file is converted into three passes over the data structure. The

first pass parses data over the blocks portion to determine the number of blocks that are compiled to CAB/CLB, input blocks and output blocks. The input and output block object numbers are saved in two separate vectors (a and b, respectively). B is the number of blocks; I is the number of inputs; and O as the number of outputs. Finally, the data object is represented as a matrix, G, of size $[(B + I + O) \times B]$, that contains the net numbers corresponding to each of the blocks to be compiled.

The second pass parses data to determine which blocks input or output port is connected to another blocks input or output port. Each link is represented by two values: the source and destination in data. The information provided is the block number, port number (ports on blocks are numbered top-down for inputs and outputs) and if the port is an input or output. The net number is placed in the matrix mentioned above. The third pass parses data to generate resulting blif statements for compilation. The input and output vectors and the matrix are used to put the nets of inputs and outputs at the beginning of the blif file. Then, for each case, the command for each block is identified, where the net numbers are retrieved from the matrix using the block number. Figure 6c shows when users connect an output of a block to at least two inputs, an extra small block is inserted into the Xcos internal representation, increasing the number of blocks and links that is removed before generating blif file.

6.2.2 VPR: blif $\rightarrow$ route

We utilize *VPR* tool to calculate optimized place and route of blocks. Figure 39b shows a graphical result of *VPR* for a mixed-signal system. The *VPR* architecture file is customized to include CABs, as well as CLBs, in the fabric array. The global routing structure is directly applied to *VPR*, where S-blocks switch the direction of routing (North, South, East, West).

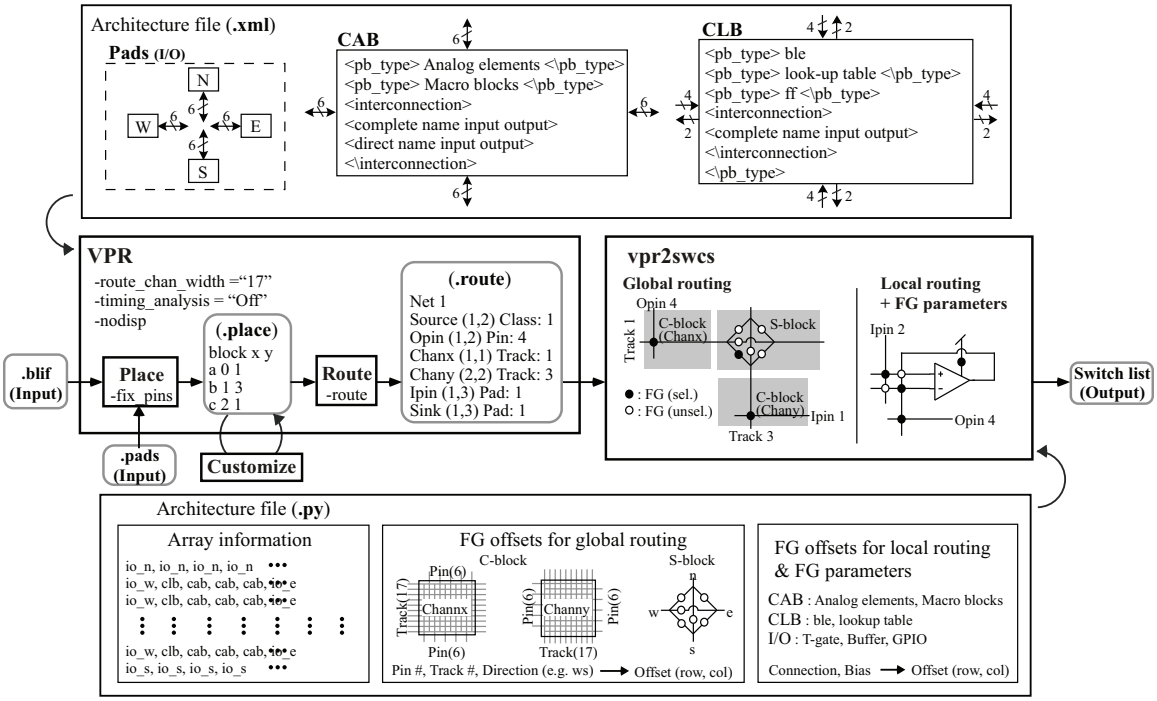


Figure 41: Detailed tool flow and configuration of *VPR* and *vpr2swc*. Architecture file (.xml) for *VPR* requires new definitions for analog elements and macro blocks in CAB and modification of CLB and I/O blocks corresponding to the hardware. *vpr2swc* creates a switch list of global routing, local routing, and FG parameters for analog / digital devices based on the architecture file (.py) including array information, offsets, and FG parameters.

Figure 41 shows how to configure the *VPR* architecture file and specify the tool options in our compilation flow. Architecture file with ".xml" extension requires definition of array blocks, which are I/O pad blocks, CABs, and CLBs. North, South, East, and West I/O pad blocks have 6 pins connecting the array. CAB uses <pb_type> tag, to specify the properties of a complex block, for defining analog elements and macro-blocks consisting of analog elements. CAB includes the definition of 24 pins (6 pins in each direction), which can be assigned to input or output. <Interconnection> tag maps inputs and outputs of each block to CAB's pins. <complete> tag connecting input/output to any pin at the output/input is used for general blocks, while <direct> tag is used for specified blocks such as Vector-Matrix Multiply (VMM). Similarly in CLB, Look-Up-Tables (LUT) and Flip-Flops (FF) in BLE are defined as a complex block with the <pb_type> tag, the <complete> tag connects inputs /

outputs of blocks to CLB pins. CLB has 16 input / 8 output pins (4 input and 2 output pins to each direction).

Based on the architecture file, *VPR* places and routes the elements from blif netlist and pads information. The use of "-fix_pins" option locks each I/O pad to a desired location listed in the pads file and results in the output file (*.place*) which includes details such as block name and locations. In certain applications one can customize the *.place* file where the system designer can assign a specific location for a block in the Xcos design. The option "-route" invokes this functionality to create a route file (*.route*) including switch / source block locations and track numbers.

VPR runs with a default options of "-nodisp" which hides the graphical interface, "-route_chan_width = 17" indicating the number of tracks, "timing_analysis = Off" turning off the timing analysis while performing global routing. Although the global routing optimization here is based on congestion information, a timing driven optimization can be integrated into this system by measuring and modeling line resistance and capacitance [37].

6.2.3 vpr2swc: route → a switch list

As the final step of the compilation process, a *vpr2swc* code developed in python calculates FG addresses and creates a switch list. Figure 41 shows the configuration of *vpr2swc* architecture file and how to map global / local routing and FG devices to FG addresses.

The architecture file (*.py*) includes information on physical arrangement of the array and offsets for FG devices, which varies according to different FG FPAA IC architecture. Row and column addresses of CAB / CLB / I/O blocks are described in the array information. The *.py* file defines FG offsets for global routing, where Chanx in C-block connects horizontal tracts to vertical pins, Chany in C-block connects vertical tracks to horizontal pins, and the tracks intersect in S-block, as well as local

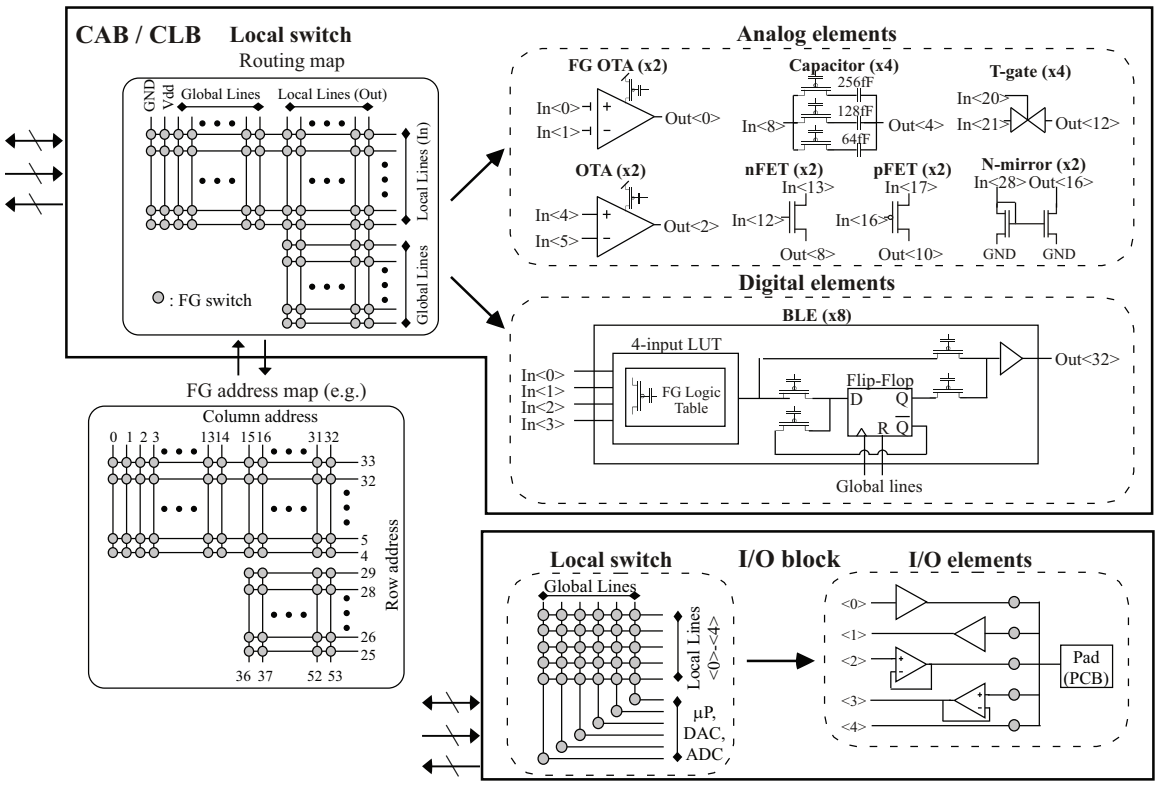


Figure 42: FG mapping for local routing and analog / digital / I/O elements. *vpr2swc* calculates row and column address from *FG address map* sharing FG node with each corresponding FG device in the *routing map*. Local switch matrix in the *routing map* connects global routing lines, gnd, and V_{dd} to input / output of analog / digital / I/O elements. Local switches in I/O block connect global routing lines to I/O elements as well as DACs, ADCs, μP. *vpr2swc* adds FG device addresses in the elements (e.g. OTA, FG LUT, I/O buffers) into the switch list.

routing and analog / digital / I/O devices in blocks.

vpr2swc reads and parses routing file to get necessary information for global routing. The routing of each net begins on a Opin (a certain output pin), goes through Chanx and Chany, and ends on a Ipin (a certain input pin). (x,y) location and pin / track numbers of each channel are described at each line. Based on pin, track, and FG offsets defined in the architecture file, the tool creates C-block FG switch addresses from pin number in Opin / Ipin and track number in Chanx / Chany and S-block FG switch addresses from track numbers in two adjacent Chanx or Chany.

A list of local switches and FG parameters in CAB / CLB / I/O blocks, handled by *VPR* as black boxes, is generated based on the FG offsets defined in the architecture

file and block location information assigned by the placement. *vpr2swc* parses the placement file and integrates circuit parameter information described in blif netlist.

Figure 42 shows detailed configuration of blocks, including local routing and elements in the recent FG FPAA IC [12]. Local switches in *routing map* enables connection of local lines, which are inputs / outputs of block elements, to global lines from C-block, to the power rails gnd / V_{dd}, as well as interconnection between local lines. Each FG switch in the *routing map* corresponds to an *FG address* which shares the same FG node and has a row and column address.

Analog elements in CAB include two Operational Transconductance Amplifiers (OTA) using a FG device for bias current, two OTAs using FG devices for the input transistors and a bias current, four capacitors using FG devices for selecting capacitor sizes, two nFETs, two pFETs, four Transmission gates (T-gate), and two N-mirrors. Digital elements in CLB are comprised of eight Basic Logic Elements (BLEs) made of 4-input Look-Up Tables (LUT), a Flip-Flop (FF), and FG switches for the logic configuration. A FG switch is set to program the user-defined logic and is embedded in the LUT. The FG switch also enables a sequential or combinational logic based on whether the output is either routed through a FF or not respectively. Local switches of I/O block enable connection to pad on PCB as well as DAC, ADC, General-Purpose Input/Output (GPIO) that interface with the μP. A user could also route the output via a digital/analog buffer, or choose an unbuffered pad.

Since *VPR* and blif netlist have been developed for the description of logic gates in FPGAs, extending it to heterogeneous systems is accompanied by two big challenges, listed in Fig. 43. One challenge is to handle the input / output directionality of analog circuits. Logic circuits, e.g. AND gate, function with explicitly defined input and output ports. On the other hand, the ports of analog circuits are close to a concept of bus, which requires bidirectional definition depending on the application. As an example a Shift Register (SR) block could be configured in multiple ways. A

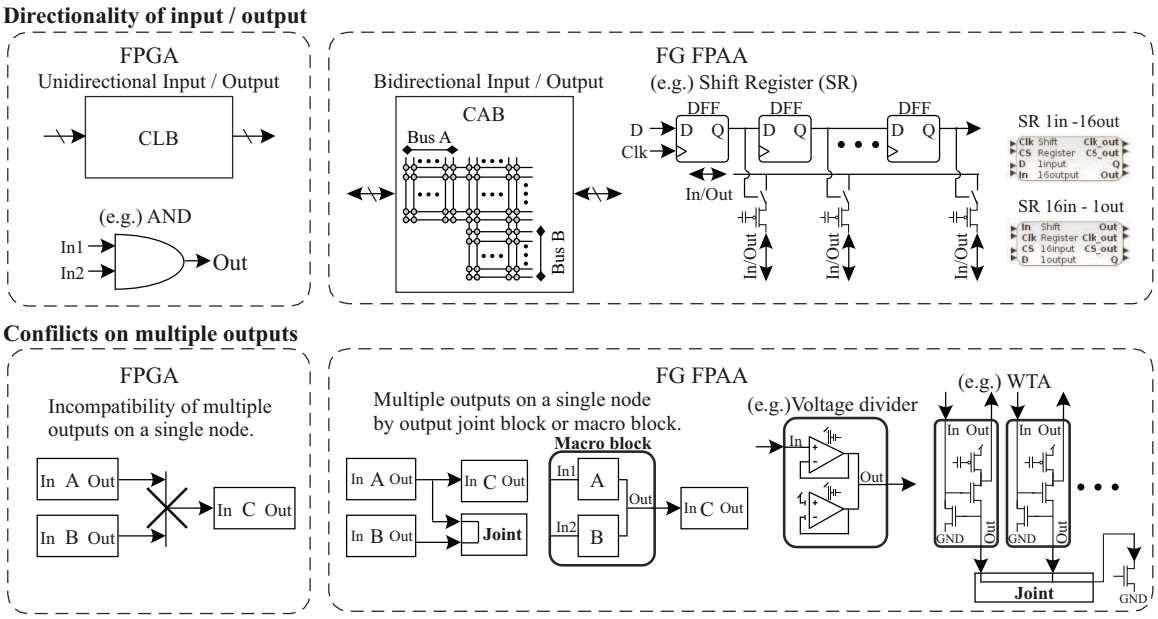


Figure 43: Challenges on applying *VPR* to analog system. **Directionality of input / output:** *VPR* originally designed for FPGA handles element blocks with explicitly defined inputs and outputs, which functions on logic circuits (e.g. AND gate). On the other hand, input or output of analog circuits are required to be defined bidirectionally. The architecture file of FG FPAA defines global lines in CAB as bus A / B, which allows analog blocks to have different definition on the same global line depending on the application. The example of Shift Register (SR) shows two blocks defined differently with same bus. **Conflicts on multiple outputs:** Although *VPR* does not provide multiple drivers for a single net, it is a required function for analog circuits, shown in the examples of voltage divider or WTA. We enable this functionality by using output joint blocks in global routing or generating macro block with local routing.

user could compile a shift register having either 16 inputs / 1 output or 1 inputs / 16 inputs for a variety of usages. We modified CAB definition in *VPR* architecture file to map each physical port to have either input or output, which enables bus concept of input / output for analog blocks.

Another challenge is conflict arising from connection of multiple outputs. *VPR* does not allow multiple drivers for a single net since it is regarded as a logical error in digital circuits. In analog circuits, however, multiple outputs on a single node is important for its functionality. For example, a voltage divider using two OTAs requires combining two outputs of OTAs to a single node, a Winner-Take-All (WTA) circuit has an architecture which requires a common current bias and hence multiple outputs of the WTA have to be connected to an input. A macro block approach provides a

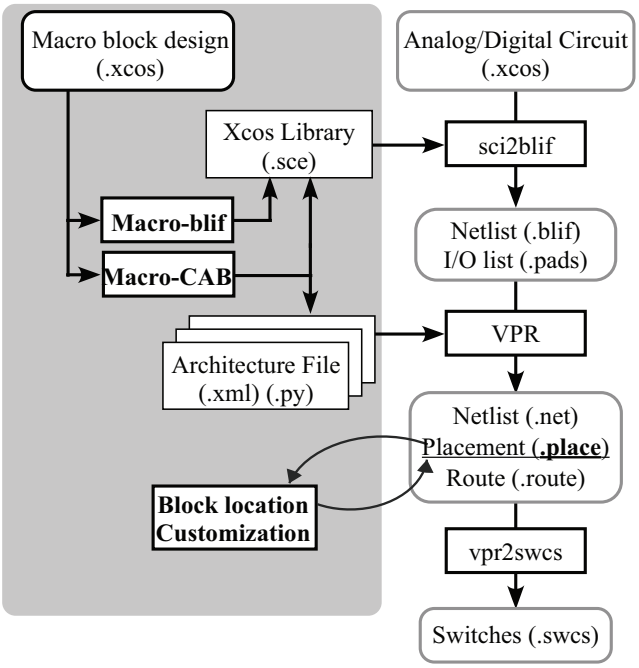


Figure 44: Compilation flow with advanced tools. A tool generating a user-defined macro block encapsulates a complex analog circuits into a high-level block in Xcos library and architecture files. The tool also customizes the location of the block which could be specified by a user in the Xcos design and used as a parameter by the placer.

solution by routing locally inside the CAB, encapsulating complex circuits with interconnections on a node for multiple outputs. As a global routing level solution, we provide a joint block allowing multiple inputs to be driven by an output, which is compatible with *VPR*.

6.3 Advanced Design Tools

The tools introduced so far enabled compilation of the user's design by creating a switch list. In this section, we introduce advanced tools for generating macro block and customizing block's location to support complicated and specific system design, as well as designing Vector-Matrix Multiply (VMM) blocks to enable power and area efficient computation by using routing FG devices. Figure 44 shows the whole compilation flow integrating two advanced tools.

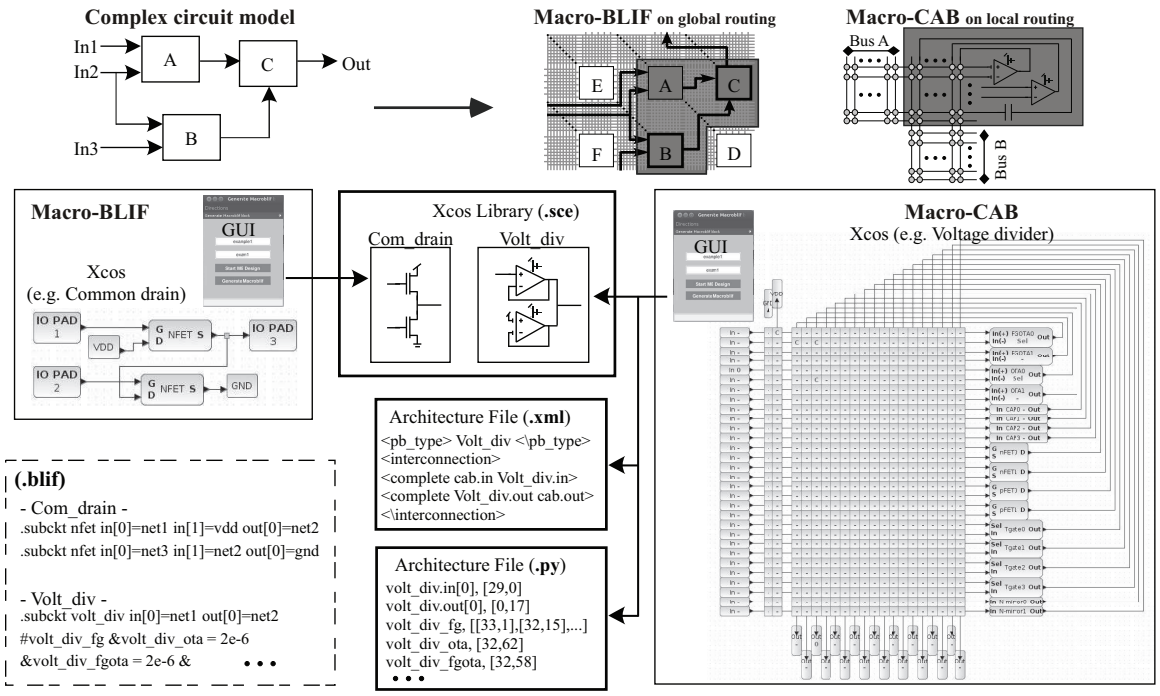


Figure 45: Automated generation tool for macro blocks integrating a complex circuit model into a single block. A tool for macro-blif block encapsulating the circuit on global routing adds the compilation description to Xcos library. Another tool for macro-CAB block encapsulating the circuit on local routing adds the information to Xcos library and architecture files for VPR and *vpr2swc*.

6.3.1 Macro Block: Encapsulating complex circuits

The tool abstracts complex mixed signal circuits to a high-level block by two different methods of encapsulation as illustrated in Fig. 45. The first is macro-blif block integrating circuits into a blif netlist. The tool extracts the inputs, outputs, interconnections, and FG parameters from the original circuit design and adds a compilation description of the macro block to the Xcos library. For internal nets, unique net names using the user-defined macro block name are assigned to avoid errors which may arise from overlapped net names in a blif file. The second is macro-CAB block integrating circuits into FG switches in a CAB. The macro-CAB block generation tool provides a Xcos file that analog elements and interconnection FG switches in CAB are predefined, which includes mapping information of each FG address corresponding to each FG device. A new user-defined macro-CAB block is designed by

modifying the provided Xcos file. FG devices are used for interconnecting analog element by connecting there inputs to outputs or gnd / V_{dd}, as well as for setting the bias value of a circuit, for example a bias of an OTA, by specifying a targeted current. After the inputs, outputs, and names of parameters with default values are set by the user in the Xcos design, the tool generates a macro-CAB block, which encapsulates the users design, and its necessary files to add Xcos library and architecture files for *VPR* and *vpr2swc*.

Macro-CAB block enables compact design, which means efficient area and lower parasitic capacitors, using the limited number of analog elements in a CAB. On the other hand, macro-blif block is not limited by the number of elements in the CAB and hence is suitable for system-level design.

Customizing the placement of blocks in a specific CAB in the FPAA fabric is required for certain applications. For example, a ramp ADC calibrated with a CAB needs to be compiled at a fixed location, since the slope of the ramp changes depending on the capacitor mismatch and biasing current. A Gm-C filter which is sensitive to parasitic node capacitance is also an example, where we can calculate routing capacitance based on [12] when the block location is fixed, and set the filter parameters.

For Customizing the placement of blocks, a user sets the location of the block in the Xcos design by changing a block parameter, "Fix_location". The tool searches the block name in the placement (having a suffix .place) file and swaps the location for the defined value.

6.3.2 VMM: computation with routing

Vector-Matrix Multiply (VMM) block is a core component for variety of signal processing and machine learning algorithms, performing a multiply operation between a vector inputs and a matrix of weights trained. FG devices on routing nodes, storing

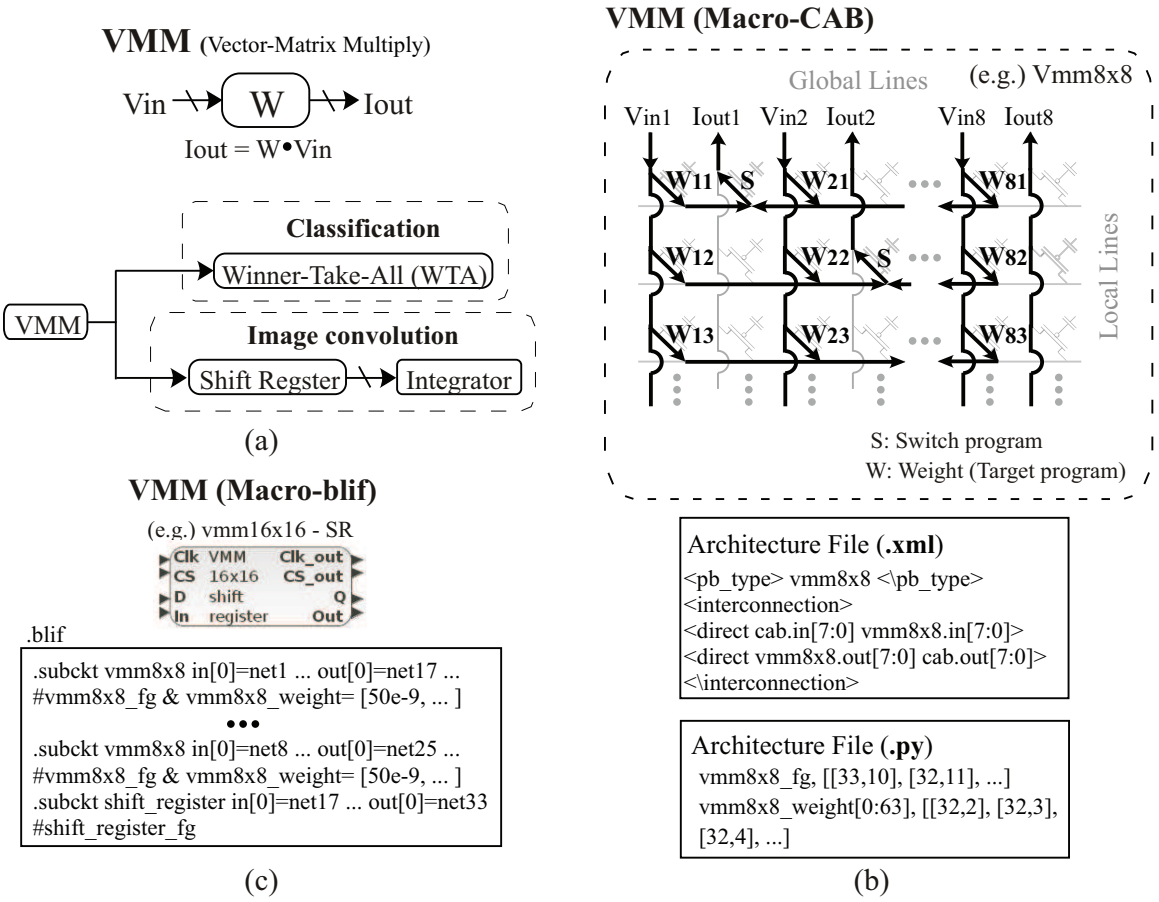


Figure 46: Vector-Matrix Multiply (VMM). (a) Applications using VMM. (b) VMM macro-CAB block conducting a multiply operation between a input voltage vector and a weight matrix stored in the FG by using local routing FG devices. (c) A large number of input / output VMM using a macro-blif block.

the weight and converting a voltage input linearly into a current, enable a power-efficient and compact implementation of VMMs. Figure 46a shows examples of the application, an analog classifier combining VMM with a WTA [46] and an image convolution combining VMM with shift register and integrator [36].

Figure 46b illustrates an example of 8×8 VMM implementation with a macro-CAB block in local routing. In local routing fabric, FG devices connected to inputs (V_{in}) are programmed to a target current level corresponding to each weight values. The converted currents in a row are summed to each output (I_{out}) through a programmed switch. To pair the weight matrix with dedicated FG switches in the local routing,

Table 11: System compilation examples

	No. of blocks			No. of FGs	Ref.
	CAB	CLB	I/O		
DAC+Com_drain+ADC	2	0	2	40	
DAC+Volt_div+ADC	2	0	2	44	
$\overline{AB}$C+AB$\overline{C}$	1	1	4	63	[12]
DAC+LPF+ADC	2	0	1	39	[81]
Universal Approximator	7	0	4	222	[75]
Speech processing	5	0	3	131	[88]
Speech Classifier	19	0	5	995	[12]

the inputs / outputs of VMM block are assigned to fixed global lines by using "direct" option in the architecture file.

Based on the VMM blocks, which are special type of macro-CAB blocks, it is easy to extend a VMM block with large number of input / output by using macro-blif block. An example of 16×16 VMM with shift register is shown in Fig. 46c. The description in the architecture file for blif file includes four of 8×8 VMMs and a shift register.

6.4 System Examples

Table 11 shows several system design examples based on the compilation using the proposed tools in this work. The table includes number of blocks in the placement file and number of FG devices created in the switch list. In this section, we illustrate three different systems built using a low pass filter, universal approximator, and a speech classifier as a complex system design example using macro blocks, where each functionality of the system has been proved with experimental data [12], [75], [81], [88].

Figure 47a shows a Xcos design and the *VPR* result of a first-order low-pass filter (LPF) system with a DAC and an ADC. An OTA connecting the output to (-) input and a FG device ADC using FG infrastructure are integrated into macro-CAB blocks, respectively. A dedicated circuit converting user-defined input vector into a voltage on FG SoC FPAA is utilized. The tool creates switch list based on the *VPR* placement

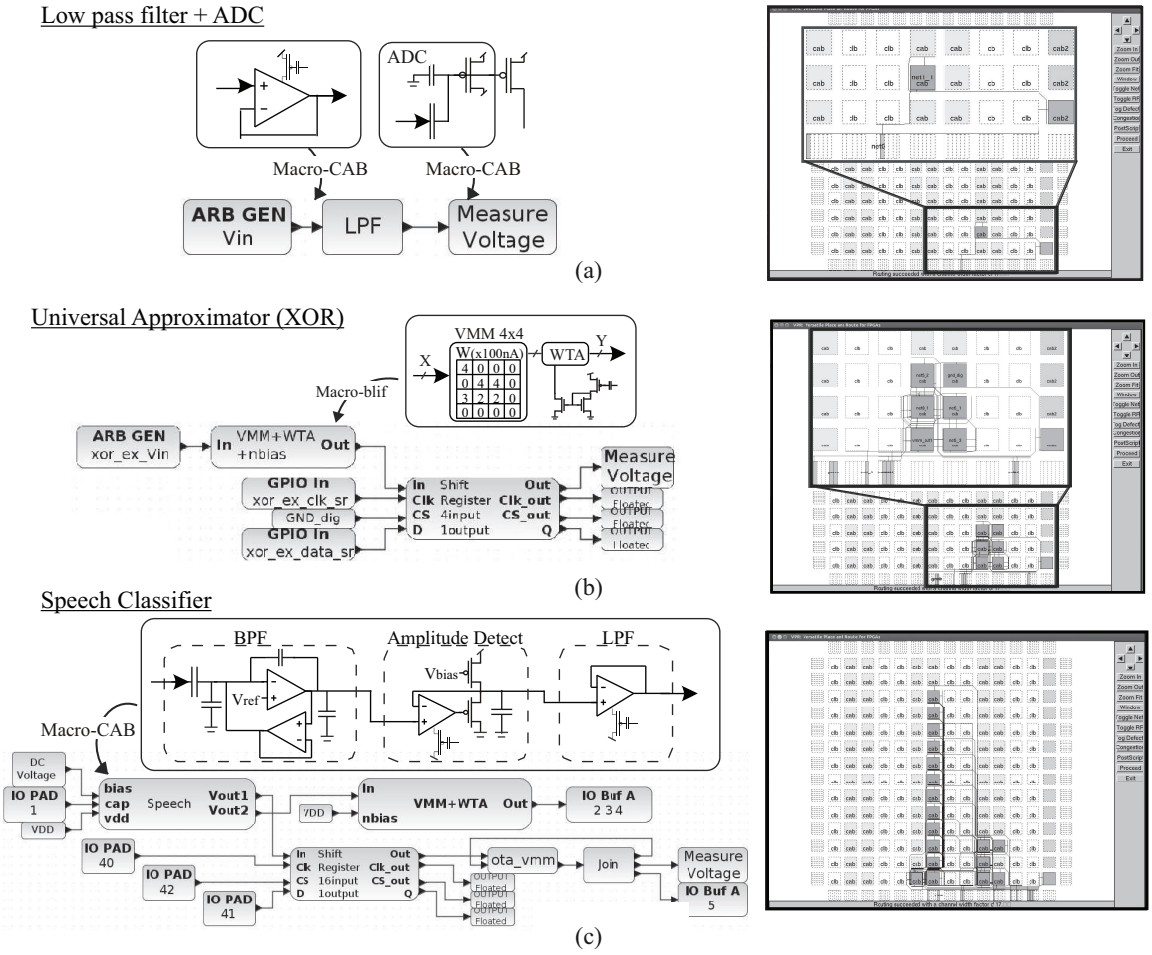


Figure 47: System examples using FG SoC FPAA. (a) DAC + Low-Pass Filter (LPF) + ADC. (b) Universal approximator. XOR is implemented by using VMM + WTA. (c) Speech classifier.

and routing, and the experimental results have been proved in [81].

A boolean function XOR using a VMM and Winner-Take-All (WTA) is an example of universal approximator [46]. Figure 47b shows the Xcos design, weight information, the expected input / output logic, and the resulted *VPR* placement and routing. The second and third input of the circuit and the weights forms the XOR output on the third output of the WTA. A macro-blif block including VMM, WTA, and a bias current FG nFET mirror circuit is implemented. A shift register block and GPIO logic signals are employed to measure the third output of WTA. The experimental results based on the compiled switch list have been introduced in [75].

Figure 47c shows an example of speech classification detecting a word in a sentence. A macro-CAB block, "Speech," in Xcos design integrates band-pass filter (BPF), amplitude detection, and low-pass filter (LPF). Twelve speech blocks perform continuous-time decomposition with different Q values on BPF. A VMM + WTA block classify each of the resulting spectrum into simple symbols. A shift register block and ADC are employed to measure intermediate nodes. Location of speech blocks are customized for the performance, the experimental results have been shown in [12].

6.5 Conclusion

This chapter presented a mixed-signal co-design environment using FG SoC FPAAs. The tools developed in this work take an essential role in the compilation flow, converting a user's Xcos design into a switch list and enabling experimental measurements in the same integrated design tool framework. The tool set is an open source setup provided in a Virtualbox package with Linux Ubuntu OS [1].

We expect our tools to empower a wider community for analog and digital system designers, as well as share the opportunities with VTR community. This work opens up interesting questions in the optimization capabilities of the *VPR*. For example, we employed a "Joint" block to solve the incompatibility problem of multiple outputs on a single node, which results in using an extra block. Also, a constraint on the VPR array structure, in which CAB / CLB should be arranged in a column direction, limits the flexibility of the array structures. We believe that an extended version of VTR / *VPR* covering both FPGA and FPAA can cope with the problems in easier and more efficient way.

This work starts the discussion on formulating the benchmarks for analog / mixed

[1] http://users.ece.gatech.edu/phasler/FPAAtool/index.html

computation. Benchmarks imply understanding computation, which required the effort of this work and parallel efforts to reach a point where developing a reasonable benchmark is possible. A benchmark likely would be composed of components seen in Fig. 47, although they are not at the necessary complexity for a well formulated benchmark. The initial benchmarks include small number of CLBs and CABs, however, we believe more complicated system-level benchmarks (e.g., Image convolution / classification or speech recognition) will fully utilize most of digital/analog blocks.

One can infer more about the computation by choosing the right benchmark. Digital computation benchmarks are about matrix equation solutions (e.g.LINPACK [89]), including LU decomposition. Analog / mixed computation benchmarks would look at different optimization metrics such as Ordinary Differential Equations (ODE) and Partial Differential Equations (PDE) [90]. The system examples illustrated here show, what the benchmarks can be, on the path and are beginning to be clear for such systems. These would be the critical next steps as we move forward from our efforts.

CHAPTER VII

A REMOTE FG FPAA SYSTEM

This chapter discusses a novel remote test system, enabled by configurable analog–digital ICs to create a simple interface for a wide range of experiments; a wide range of previous remote test systems have to spend considerable time developing their hand-tailored configurable system [91–96]. Figure 48 remote test system requires no additional setup, other than the experimental system, other than simple email handling, which is available over almost all network systems without affecting the network. Independent of distance, the system enables users anywhere with an internet connection sufficient to send and receive email, opportunities both in academic as well as research and industrial applications. This approach minimizes computer support setup and maintenance, relieving the pressure overworked computer support staff, particularly in cost-conscious academic environments, trying to keep pace to maintain a larger number of computing systems.

The remote test infrastructure is enabled through a single digital external digital interface to an analog–digital programmable and configurable IC system, empowered using large-scale Field Programmable Analog Array (FPAA) device(s) [12]. This approach gives a simple digital peripheral using a standard interface (i.e. USB), enabling a small Internet of Things (IoT) block interfaced through an email system to an open-source design / control tool. Our open-source tool platform empowers the user to do seamless low-power analog-digital co-design in a single environment [75]. The resulting controlling device, whether it be directly connected through this digital port (i.e. phone or tablet) or through a network, can be a potentially simple OS enabling all features on the resulting system, including sensory / actuation devices

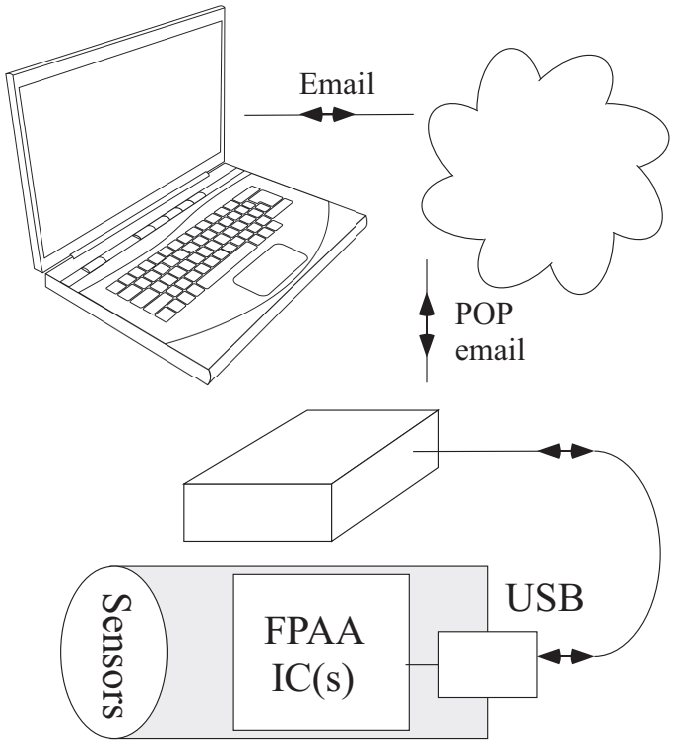


Figure 48: Remote test system based on FPAA devices that can be used within our current framework of high-level, open-source Xcos/Scilab tools. With a single button click in the graphical tool, the system will email the resulting targeting code for the FPAA device to a server location, to be picked up by the remote system, that compiles, runs, and then emails back the target results.

connected to this remote platform.

POP email communication fits with the limited operations required, easy to code, and keeps the resulting computation requirements for operation as minimal as possible, particularly for academic environments. Email protocol simplifies individual access while requiring effectively zero administrative support. Other educational remote test systems require the user to take control of the system, usually through a log-in. This remote system approach differs from the area of one-way updating FPGA software, or remote FPGA reconfiguration for a device in the field [97, 98].

The following sections present the resulting FPAA based remote test setup including, overviewing the remote-tool structure, presenting multiple system examples, while presenting the range of user interfacing, expanding the remote user application as well as measurement capability.

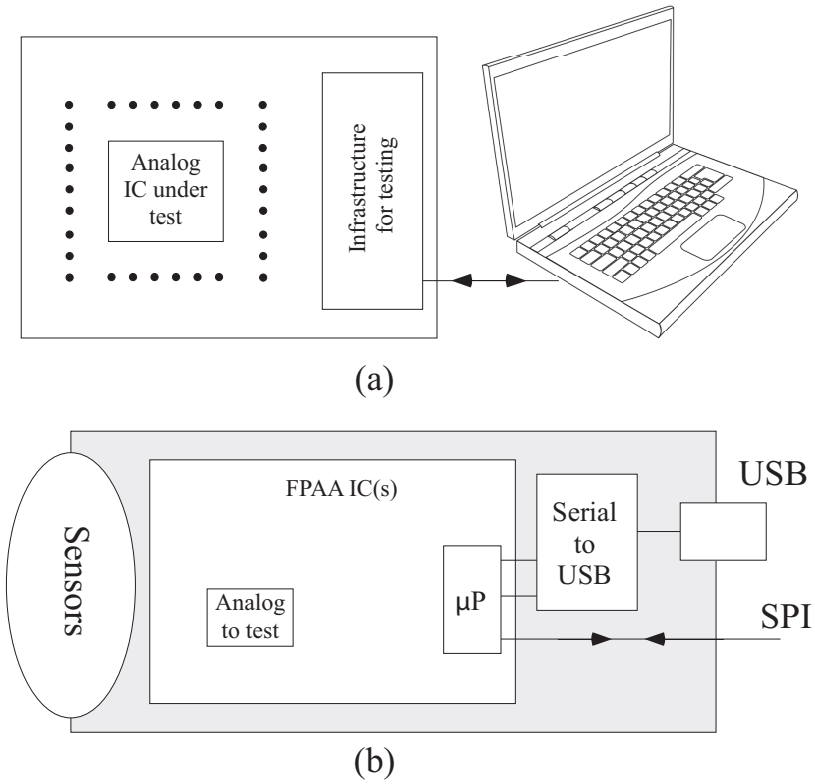


Figure 49: System perspective using a remote test system to utilize mixed-signal configurable systems. (a) Classical approach when considering analog IC design and testing that have an analog component under test, along with all of the required board (or bench) infrastructure required. (b) For these mixed signal ICs, the IC is an entire system, acting as a peripheral through a USB port and/or optional SPI port. Also, the resulting analog to test, if making a complete parallel to the system in (a), is a small part of the overall available computing infrastructure.

7.1 Remote Tool Framework

Figure 50 shows the perspective of our remote testing approach. When one thinks of testing an analog IC, or even a mixed mode IC, one tends to visualize, in the best case, a test setup like Fig. 49a. Such systems mostly have a single or a group of analog ICs, some controlled switches, and an infrastructure to connect to a computer to control the entire setup as well as run the testing interface. When looking at a remote testing system, the testing interface layer requires further complication depending on the particular system.

In comparison, Fig. 49b shows the FPAA system used, the FPAA IC is a full system with a processor, requiring only simple interfacing to the outside world through

USB or say SPI ports, appearing to be a standard peripheral to a typical device. Such an approach enables a family of configurable hardware, utilizing a single configurable framework and tool infrastructure to control the resulting device. This difference in configuration provides the opportunity for empowering our remote test system or classroom use, research groups, as well as interested users. Integrating this approach into an existing tool framework keeps compatibility for a range of applications, and not limited to just an academic or research application.

Figure 50 shows the framework for the remote system approach. The resulting structure should be easy to use on both the user and remote server side, requiring minimal user maintenance, using an integrated user tool platform, and having as few location constrains as possible. The tool platform [75], implemented in Xcos / Scilab (an open source clone for MATLAB / Simulink), simulates designs as well as enables experimental measurements after compiling to SoCs in the same integrated design tool framework. The open-source toolset is setup as an Ubuntu 12.04 Virtual Machine (VM)[1] enabling use in classrooms as well as research and development groups.

Our modifications to the high level Xcos tools on the user side required extending the GUI interface, as well as updating the (python) code to enable emailing the resulting compiled FPAA targeting file out from the VM. The user will receive their resulting data to their email location as an attached representation of the measured results; the user can move this data into Scilab or another analysis / plotting tool of their choice. By using an email based system, one would not expect real-time control occurring through the resulting email network.

The small remote server code framework minimizes resulting system overhead, utilizing standard email servers to enable a relatively stable remote platform capable with nearly zero administrative overhead. The remote server will periodically check for email on the server, POP the resulting message from the server, check its control

[1]available at http://users.ece.gatech.edu/phasler/FPAAtool/index.html

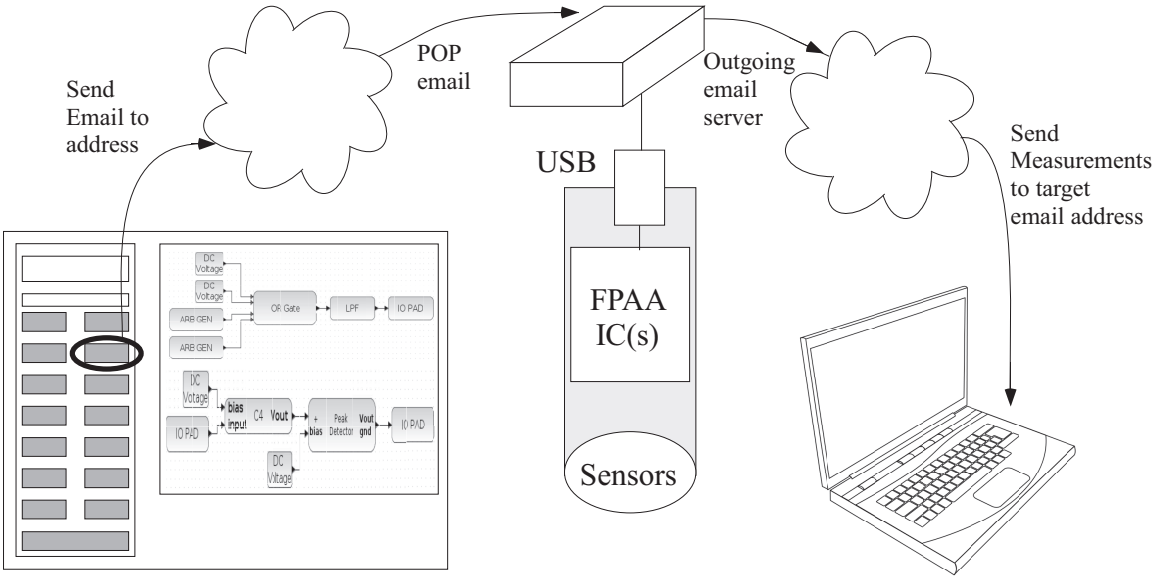


Figure 50: Detailed flow for the remote test system implementation. The design toolset in scilab / Xcos allows the user to "send email" in addition to "program FPAA." When that option is chosen, the resulting file is sent by email into the cloud. The resulting email is POP-ed off the server, the resulting programming files are extracted and executed, the resulting data measurement is performed on the device, and the results are sent back by email to the original sender. The user can directly use the results in Scilab or any other data analysis program to observe their data as well as complete their analysis. The resulting flow is enabled by having a highly configurable analog / mixed-mode system with a simple digital interface through USB, really enabling the connection as a typical digital peripheral.

syntax, and have the object code ready for programming. Recent FPAA devices now enable Floating-Gate (FG) device programming entirely on the device as an input data stream; therefore the entire data stream, including μP code to execute programming, simply looks like a single stream of data to the system. Encapsulating this entire structure in a single file requires small, unix-based code to communicate the file to be programmed. After the device is programmed and the input data is loaded into the processor, the IC proceeds to compute the resulting function, also storing data in local or in the remote server memory. The resulting output data results are pulled together and sent out by email to the host's chosen email address.

This approach enables programming by any small embedded devices (after design), particularly those with direct USB connections (tablet, phone) allowing minimal (linux) code for programming and operation. Whether remote or not, these

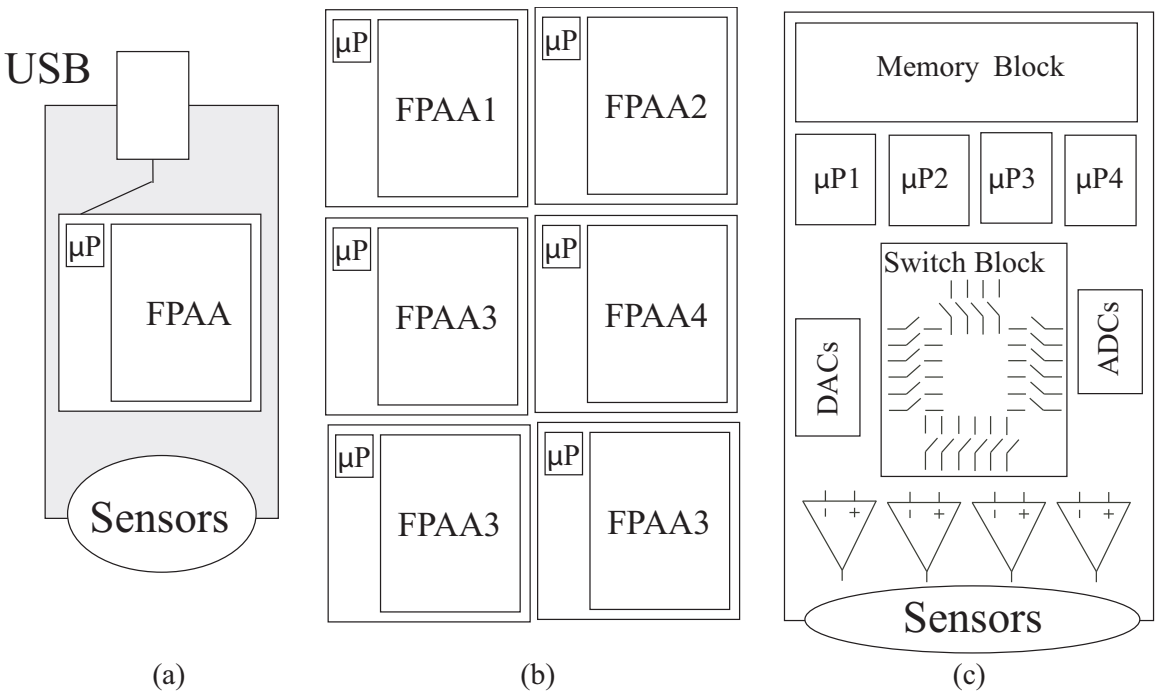


Figure 51: Possible approaches for mixed-mode computing systems. Implementation could be (a) a single FPAA device, (b) a board of FPAA devices, or even (c) a board with no FPAA devices but with programmable parameters and topology for a resulting board encoded in the resulting technology file.

concepts allow an easy approach for networks of remote sensor nodes; for example an Internet of Things (IoT) approach enabling real-time sensor processing that can send context aware results while requiring minimal support overhead for the approach.

Figure 51 shows FPAA IC board design, as well as systems using additional programmable components. The high-level graphical tool enables a user to be able to try different approaches to optimize the system performance, allowing consideration of tradeoffs of power, system utilization, time to market, etc. Digital Hardware-Software CoDesign is an established, although unsolved, discipline (e.g. [99]). Including of programmable and configurable analog computation to current digital approaches requires revisiting existing tradeoffs. The approach is focused to enable system designers to integrate useful systems, while still enabling circuit experts to continue to develop creative and reusable designs within the same tool flow.

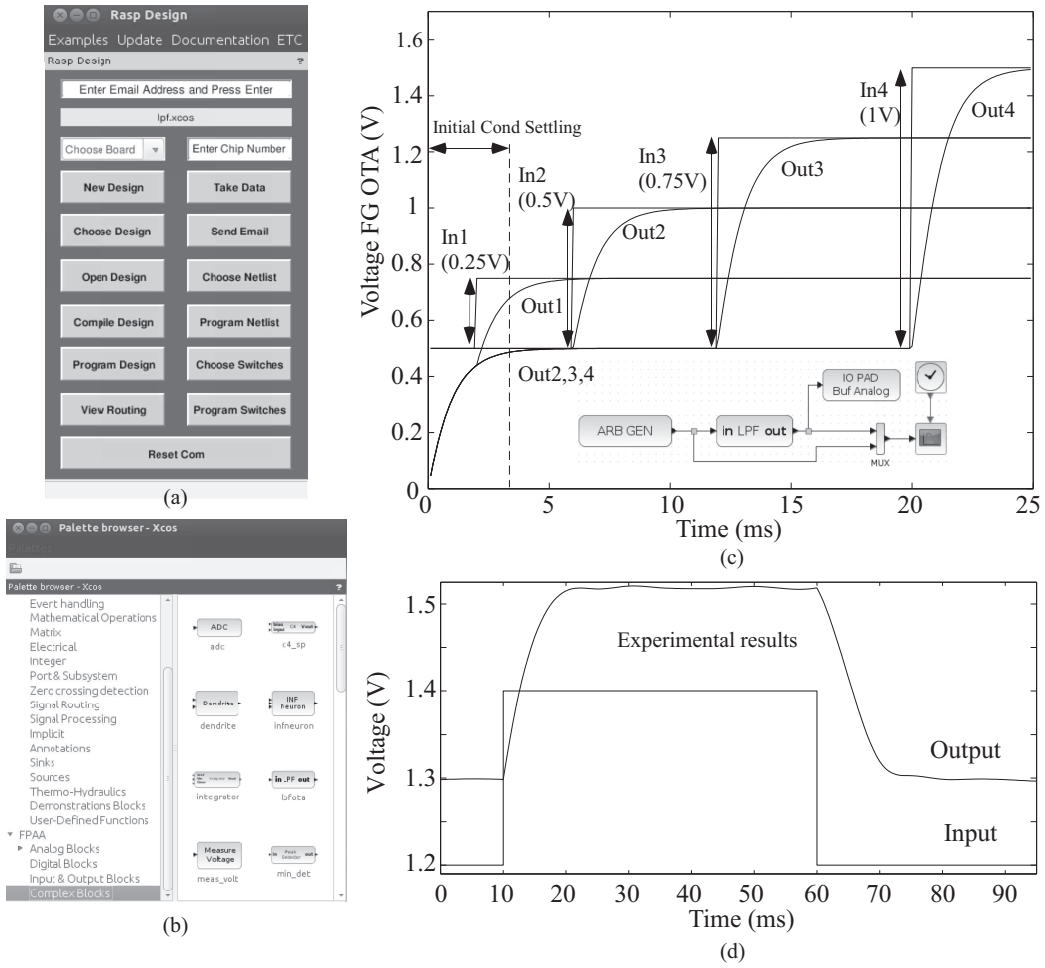


Figure 52: An example of the entire tool flow for a Low-Pass Filter (LPF) computation. (a) The user chooses basic design options through the FPAA Tools GUI, which starts running when the Scilab tools are started in the distributed Ubuntu Virtual Machine (VM). (b) Snapshot of the Xcos palette for FPAA blocks. There are four sections, namely the Analog, Digital, Input/Output and Complex Blocks; the Analog, Digital and I/O blocks consists of basic elements in different tiles of a chip. Complex blocks are pre-defined circuit blocks made of more than one basic element. (c) Simulation results for 4 input and output computation. Lines, and resulting blocks, allow for vectorized as well as scalar inputs. Inset shows the Xcos diagram; the user sets parameters for simulation or for compiling into IC. (d) Experimental results for a 1 input and output computation.

7.2 Remote System Overview Examples

Figure 52 shows an initial full tool example of the graphical interface and results for a first-order Low-Pass Filter (LPF) system, showing both simulation results and experimental results from the remote test system. Figures 53, 55, 56, 57 show more complex tool examples using this remote system. Xcos gives the user the ability to create, model, and simulate analog and digital designs [75]. The Xcos editor is

Table 12: Components of Single Programming File

Function	Data Type	Core Files
erasing and initialization	Compiled (assembly)	tunnel_revtun_CAB.elf switch_program.elf
measuring outputs	Compiled	voltage_meas.elf
input (e.g. DAC) data	data	input_vector
FG block	data	output_info
Switch list info (num, address)	data	switch_info target_info
Course Prog T_{inj}	data	pulse_width_table_offset_d1o2
Fine prog V_d table	data	Vd_table_30mV

standard blocks that are compartmentalized into classes or palettes that range from mathematical operations to digital signal processing. The editor allows the internal simulator to utilize the functionality of each block to compute the final answer. Our tool structure took advantage of user-defined blocks and palettes that can interact with Scilab inherent blocks.

Figure 52a shows the FPAA Tool GUI and grey buttons with the labels: New Design, Choose Design, Open Design, Compile Design, Program Design, and Take Data; these buttons open a new Xcos editor window, select a previously saved design, view the design in Xcos editor, convert the information in Xcos file into a programmable format, program hardware for the current Xcos file selected, and cause the hardware to be in the mode to take data, respectively. It includes a button which is labeled **email**, and a top box that one can put in an email address. These two small buttons on the tool framework show the minimal additional user overhead for using the remote system.

Our Xcos [87] tool uses user-defined blocks and libraries. When the user opens the Xcos editor, a palette browser is displayed, as shown in Fig. 52b. The browser lists Scilab's collection of palettes as well as user defined palettes. One selects from a palette of available blocks to build the resulting system, which can be composed

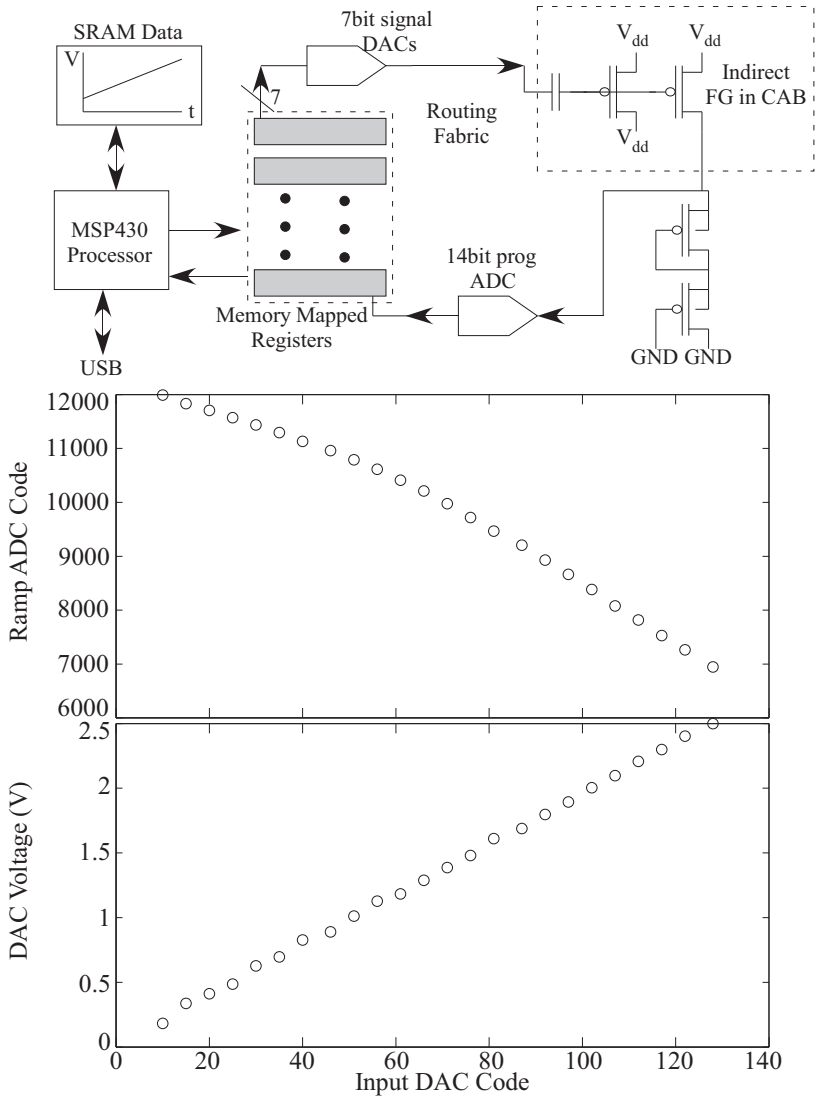


Figure 53: One basic voltage measurement scheme utilized in the SoC FPAA [12] for low-speed measurements utilizing the entire IC infrastructure. The μP design is an open-core MSP 430 processor with on-chip structures for 7-bit signal DACs, a ramp ADC, memory mapped General Purpose (GP) IO and related components. The measurement through a FG transistor in a CAB utilizes the processor, signal DACs and memory mapped register, a typical loops for instrumenting and measuring analog and digital blocks; Often, the FPAA computation utilizes all of these capabilities.

of a mixture of analog (blif), digital (verilog, blif), and software (assembly language) components.

The single programming file, attached to the email for programming the remote system, is a compressed structure of multiple files. The tools output a single programming file, a combination of multiple files, that is used for FG programming and

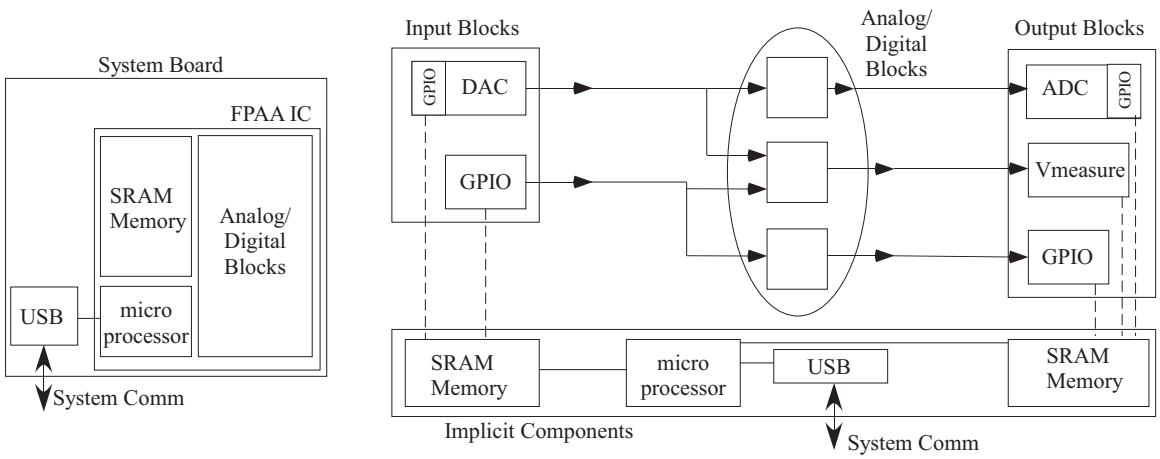


Figure 54: Our approach is enabled through a single port of communication, through a USB port to a self contained programmable and configurable mixed-signal IC through its programmable interface. This simplicity results in understanding both the analog and digital capability of the FPAA IC, and the resulting input and output blocks for the system as well as the resulting system control, at the level allowed for the high-level tool framework (Xcos / Scilab). The entire IC is the computing system, and all components are part of the computation. A typical user will often use digital (GPIO) or analog (DAC through GPIO) input blocks, interfaced through the SRAM memory and μP control, and will often use digital as well as analog compiled ADC through GPIO or Vmeasure through slower and more accurate 14bit ADC. The tool framework compiles down the resulting analog, digital, and μP components, as well as the vector input into the system.

SRAM memory setup for the SoC FPAA [15]. The design tools can process the downloading of this file, as well as other devices (e.g. remote computer, tablet). The SoC FPAA devices now enable Floating-Gate (FG) device programming entirely on the device as an input data stream, therefore the entire data stream, including μP code to execute programming, simply looks like a single stream of data to the system. The FPAA utilizes an open-source μP, embedded 16k $\times$ 16 SRAM for program and data memory, as well as the memory mapped registers for FG programming. We give the file definition in Table I. We expect this structure will remain stable across future generations; fortunately, each programming file is self contained for programming since it includes its own code and parameters for programming.

7.3 The Range of User Interfacing Expands Remote User Capability

Figure 54 shows the remote server was partially enabled by a simple digital (USB) interface, with simple interfacing between the FPAA IC (or multiple ICs) to the resulting USB infrastructure to the host device. The input data, output data, FG programming, and other control functions all move through a single standard digital USB interface; the device to the remote system looks like any other embedded, USB peripheral, where the tools handle a similar case whether the board is local to the tools or emailed to the remote server. The setup uses no external pins; one could connect multiple devices, heterogeneous devices, and additional sensors, but to the external world, the device is still a simple USB connected device.

The practical issue is understanding the particular interfacing options available on the FPAA, as well as the computation possible on the FPAA device. The USB interface is connected through serial interfaces, a simple serial (8n1) interface through an FTDI 2232D IC on the board.

Figure 54 shows the high-level blocks representation, similar to the corresponding Xcos diagram, both including computational blocks, as well as input and output blocks available and their interfacing into the μP / SRAM memory block. The arbitrary waveform generator connects data to a vector input representation versus time in the Scilab workspace that gets converted, where needed (i.e. DAC devices), as well as packed with the programming file; the interface for the data is directly set by the high-level tools. A short list of potential on-chip interfacing options for moving data in and out of the μP / SRAM memory are:

- General Purpose Input / Output (GPIO) memory mapped digital registers
- Hardware interrupts based on routed fabric signals
- Signal DAC through memory mapped registers

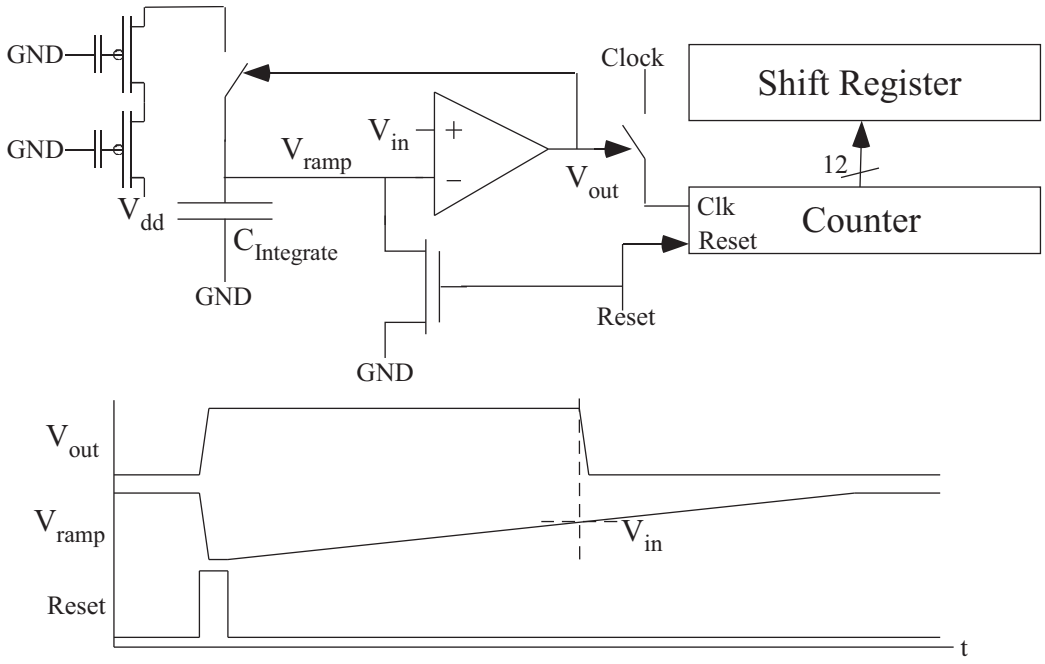


Figure 55: For the remote system, one key aspect is the range of potential methods for voltage measurement in the fabric to connect to the μP device. Typically, these devices include direct digital inputs, with resulting level comparisons, Analog-to-Digital Converters (ADC), as well as a range of other classifier components. One can compile a Ramp ADC in the routing fabric to operate at a range of sample frequencies and resolution.

- Compiled ADC coupled with FG elements in the fabric through GPIO registers

These approaches are all voltage-mode inputs and outputs, consistent with level=1 block definition for system building [76]. Further, blocks can have some assembly code (or completely assembly code) as part of the functionality, therefore would interact with memory, potentially, more directly. Where necessary, one can measure currents through compiled transimpedance amplifiers, switched capacitor network, and related circuit techniques. One key constraint on any algorithm is efficiently using the 16k x 8 SRAM data space either to hold data, or buffer data coming through the serial communication from the host system running the remote interface.

One representative question would be the types of Analog-to-Digital Converter (ADC) blocks, and their applicability for a configurable architecture. Figure 55 shows a programmed ADC block, a ramp ADC. An important question is what type of ADCs to compile in such a configurable system, particularly in a compiled system with a

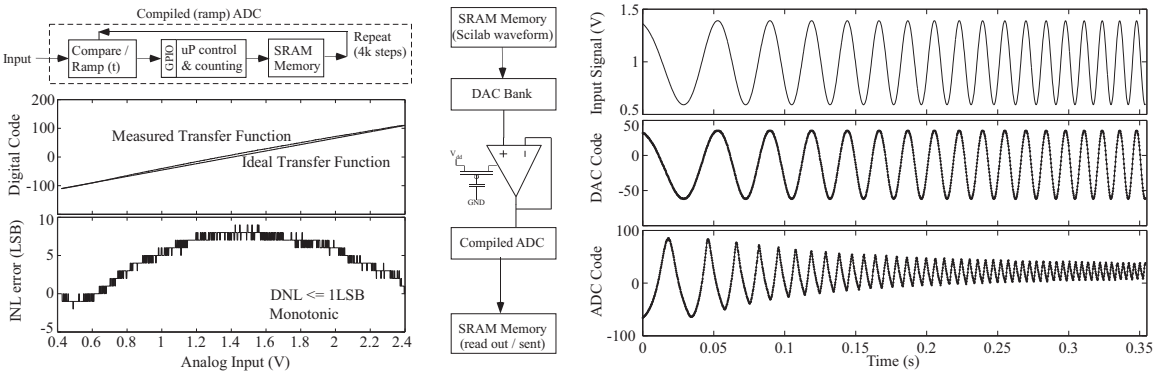


Figure 56: Measurement of the compiled 8-bit ramp ADC used in the FPAA device. 4kSPS ADC for this 8-bit ramp conversion integrates the compare + ramp functions compile into a single CAB, routing the digital output bit to GPIO. This simple ADC polls for the digital bit to change while counting in the processor; more advanced blocks utilize interrupts and compiled counter blocks in the CLBs for the conversion. The ADC is not meant to be the highest precission component, but a very small, monotonic ADC calibrated for the resulting response as necessary. A compiled system for a first-order low-pass filter using these components, where the output load capacitance is implied (output going into the ADC), because of the node capacitance from the routing infrastructure. The programmed corner frequency of the LPF is 150Hz. The resulting device is measured by applying a linear chirp input signal going from 25Hz to 250Hz, showing the resulting signal attenuation, as expected, as well as the resulting signals from the input 7-bit DAC as well as the 8-bit ramp-ADC (4kSPS). The output of the ADC inverts the resulting response from the original signal.

high density of FG devices available. One would not expect an architecture with lower latency than a pipelined 1-bit ADC architecture, primarily because a requirement for a smaller latency delay would likely result in analog computation. These pipelined devices are typically used mostly for acquisition of data, often for circuit debugging opportunities. Algorithmic converters, being related to the pipelined ADCs, would most likely win over successive-approximation ADCs because of not requiring the design of a separate DAC for the system; we estimate achieving an algorithmic converter in 1-2 CABs in a typical architecture. The approaches get closure on the most promising ADC IP blocks to compile down for these architectures.

Figure 56 shows a simple compiled ramp ADC converter to illustrate the flow from input SRAM memory data to the computed output SRAM memory data. The resulting simple 8bit ramp ADC illustrates using the digital infrastructure and μP to move analog signals into stored SRAM memory. The simplicity only requires part of a single CAB element while still giving reasonable monotonic performance

with some curvature due to the nonideality of the current source element creating the ramp; Measurement examples use 4kSPS ADC sampling. Figure 56 shows a complete computing loop for data through a board used as a remote test, where the starts data loaded into SRAM as part of the programming structure, processes through one of multiple memory mapped 7bit DACs, through a first-order LPF block programmed with a corner frequency of 150Hz, through the above ADC block, to arrive back as stored solution vector in SRAM that is transmitted back to the user.

Figure 57 shows a more complex example using the remote FPAA system, a parallel bank of bandpass filters and amplitude detection typically used for low-power sub band analysis, as another example of co-design between assembly, analog, and digital components and interfacing. The corner frequencies are programmed between 100Hz and 750Hz, seen by the different peaks in the chirp response in Fig. 57. This example uses the same SRAM memory, 7bit input DAC, and compiled ADC. A similar chirp signal linearly varies between 25 and 1kHz.

7.4 Conclusion

This chapter presented a novel remote test system, enabled by configurable analog–digital ICs to create a simple interface for a wide range of experiments. Our remote test system requires no additional setup, resulting both from using highly configurable FPAA devices, as well as from the advancement of straight-forward digital interfaces for the resulting experimental FPAA system. An analog–digital programmable and configurable IC system, enabled by FPAA device(s), requiring a single digital external digital interface opens opportunities for a simple remote test infrastructure. The system overhead requirements are straightforward, requiring simple email handling, available over almost all network systems with no additional requirements.

Opportunities appear both in academic, as well as research and industrial applications. This technical platform enables collaborators in different areas to investigate

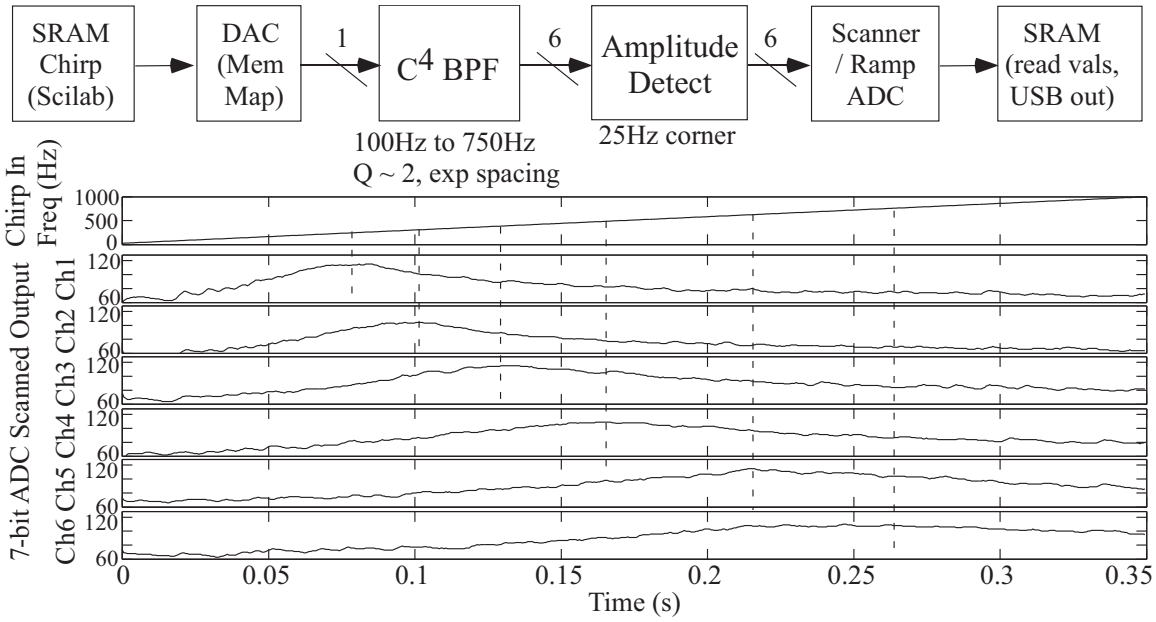


Figure 57: Block diagram (similar to Scilab / Xcos definition) and output results for a bank of 6 BandPass Filter (BPF) and Amplitude detect elements compiled and measured through a remote FPAA board. The input chirp signal, a linear sweep between 25Hz and 1kHz, is stored on-chip SRAM to be played through a memory-mapped DAC. The output signals went through a demultiplexing module, compiled using the shift-registers located in routing fabric, and then through the same 8-bit ramp ADC module; this measurement only used the upper half of the positive values (effectively 6bit). The linear frequency sweep with time is illustrated, as well as the resulting outputs of all 6 frequency channels, each programmed to a different frequency location (exponentially spaced).

items on a single platform; setting up a system is straight-forward and capable for multiple systems. These approaches have been utilized as part of a graduate level class at GT [100]. Our novel open-source tool platform empowers the user to do seamless low-power analog-digital CoDesign in a single environment. Our FPAA SoCs consist of an integrated processor, I/O peripherals, and an FPAA comprised of analog and digital components [12], although the approaches can be extended to other platforms. Our approach integrates multiple open-source tools, using Scilab and VPR, to develop a coherent user-friendly design flow, with a custom software toolkit that generates and implements high-level simulation and experimental measurement of the resulting hardware system [75]. The ability to seamlessly move between tools designed for a board in hand as well as a remote test system has greatly improved the student's interest in using either hardware platform, as well as kept development complexity

for such a complex laboratory course under control.

This approach gives a simple digital peripheral using a standard interface (i.e. USB), enabling a small Internet of Things (IoT) block interfaced through an email system to an open-source design / control tool. The resulting controlling device, whether it be directly connected through this digital port (i.e. phone or tablet) or through a network, can be a potentially simple OS enabling all features on the resulting system, including sensory / actuation devices connected to this remote platform.

We chose a POP email system among the many alternatives for system communication. First, the simple structure of a POP email system (see for example [101]) fits with the limited operations required, easy to code, and keeps the resulting computation requirements for operation as minimal as possible. POP supports simple download-and-delete, where this system only requires connection, message retrieval, and temporary storage (local). Anything more complex is extra overhead with little function. Email is ubiquitous, particularly for students. The wide variety of email services (e.g. Gmail) results in many painless options, without the need for server infrastructure. A simple POP server keeps the control code small allowing systems extending to small, remote platforms (e.g. devices worn on the body, etc.), particularly where low power battery operation is possible.

Second, starting from the inspiration of academic classroom experiences, the use of email protocol simplifies individual access while requiring effectively zero administrative support. For example, this remote system is being used in a graduate course this Spring 2016 [100]. This remote system approach allows a host system to even be operated anywhere, such as a coffee shop, without significantly affecting the system users. Most can get some form of a wireless network allowing basic email protocols on that network. Any administrative barrier for faculty to set up technology for their classes tends to result in the approach not being ever implemented. This remote system provides an alternate path through the constraints of the layers of resources

and administration of those resources. We expect such a system can have significant research impact ease of collaborators having shared systems opens up options for remote sensing and computing nodes with nearly zero required resource support.

Other remote test systems used in education typically require the user to remotely log-in to the system, and effectively take control of the system. From one of the classic circuit measurement systems [102], one sees attempts for remote test structure to replace a large number of dedicated lab stations with expensive equipment, similar to the setups used at Caltech (CNS 182) and later at GT utilizing even the earliest FPAA devices [100]. Additional remote systems have attempted other concepts having users log-in and control a remote computer [103–107], as well as debates over the right use of remote hardware for classes [103]. These approaches involve using a hardware board (e.g. Spartan 3 or Arduino board) and using graphical languages like Labview (e.g. [108]). Much of the time on the system relates to the user *thinking* about their problem, rather than performing operations on their experiment; the system gets around these issues, while using a tool approach identical when a physical board is used.

This remote system approach is different from the area of one-way updating FPGA software for a device in the field. Remote FPGA reconfiguration already utilizes internet connections on allowed networks (with any resulting permission overhead), and can update FPGAs using protocol like User Datagram Protocol (UDP) [109]. Xlinix provides direct developer support when using external devices (e.g. CPLD) [97], or even cases using the FPGA as part of the updating infrastructure [98]. Hardware systems are designed for in-field FPGA updating [110], including space applications [111]. This remote system looks towards an expanded purpose, while also not being subjected to any network infrastructure constraints.

We expect the building of such a system raises a host of additional issues as users begin to consider these approaches. For example, the current system has limited

security features in its initial development. Going forward, one would want a form of authorization (i.e. a user access code) as part of the email message. Using a reduced email based system does reduce a significant amount of potential security issues compared to a web-based system, as well as we don't need to support a wide range of internet browsers. The linux based system and email retrieval system is used to download files, extract resulting targeting files, and use those files for targeting; we do not use an entire email browser for the resulting system, eliminating many potential virus issues. The biggest security issue seems to be uncompressing the resulting files, where that is the only place we need to execute anything on the files, and should be developed as a trusted source. We also expect a host of issues will arise as people work with such a system where outside groups, say running the remote system, might have exposure to the user's designs, and resulting Intellectual Property; we expect current approaches towards individual security will be used where applicable.

CHAPTER VIII

CONCLUSIONS

The subject of this research is to create a programmable and reconfigurable system design environment enabling low-power mixed-signal processing and built-in self test from the user application design to the measurement. To accomplish it, I have focused on implementing a compact hardware, developing algorithms and tools, and establishing a solid calibration flow. By combining these three components, signal-processing engineers are able to implement low-power embedded system without a long time of tape-out IC development cycle as well as without considering the hardware variation. This powerful platform also enables exploring and implementing nontraditional solutions such as analog-digital mixed, or neuromorphic computational techniques.

Recently the need for IoT devices has been increasing. Although a lot of sensors have been developed, the processing is still poor in power. Also, low power computation hardware and the design tools are essential in AI assistant or embedded vision devices based on machine learning. The timing for the transition of the FG technology to real products is perfect for this opportunity. I look forward to making the transition happen and making a huge impact on the industry.

8.1 Research Summary

Chapter 2 provided an overview of floating-gate technology and experimental results of floating-gate devices at technology nodes smaller than 350nm, which shows the possibility of reconfigurable systems with scaled-down FG devices. Experimental data from a 130nm and a 40nm CMOS process of floating-gate transistors exhibit the ability to retain charge and modify charge via electron tunneling and hot-electron injection. Also, FG switch behavior and routing capacitance on floating-gate transistors

in very deep sub-micron nodes have been discussed.

Chapter 3 described the basics of programming the charge on a floating-gate transistor and introduced an FG programming algorithm using on-chip integrated programming infrastructure. We use hot-electron injection for precision programming of FG devices due to the nearly ideal selectivity between devices, whereas we use electron tunneling for global initialization because of their relatively poor device selectivity. On-chip integrated programming infrastructure, including μP, SRAM, DACs, ADCs, and current-voltage converter, enables an FG programming algorithm requiring only a few fixed-point computations compared previous MATLAB-based algorithms requiring extensive floating-point computations.

Chapter 4 introduced the RASP 3.0 chip that integrates divergent concepts from multiple previous FPAA designs along with low-power digital computation and interface circuitry (i.e. DACs, ADCs). This IC fully integrates rapid reconfigurable analog-digital computation with configurable fabric of interdigitated analog and digital computing blocks and with a microprocessor (μP, open-source MSP430) enabling both computing and control. We showed measured data that this unified structure enables a wide a wide range of SoC computing options that can be optimized for multiple parameters, showing the most sophisticated FPAA capability built to date.

Chapter 5 introduced a calibration flow for an integrated FG programming system for a large-scale Field Programmable Analog Array (FPAA), including characterizing the FG programming infrastructure and hot-electron injection parameters in the integrated SoC FPAA, calculating the EKV model parameters for the golden FETs, calibrating the compiled DAC and ADC blocks that interfaces between the on-chip μP and compiled analog circuits in the array. V_{T0} mismatches due to the indirect FG structure are characterized through a compiled mismatch measurement block. A compiled classifier implementing XOR function using a VMM and WTA on different chips shows the effectiveness of the V_{T0} mismatch-map compensation integrated into

the compilation flow.

Chapter 6 introduced application design tools for a mixed-signal co-design environment using FG SoC FPAAs, which are based on open-source codes. *sci2blif* converts block level information in Xcos, which is a graphical system design environment in Scilab, into a blif netlist. *vpr2swc* calculates FG addresses and creates a switch list from a route information generated by using VPR tool. Advanced design tools for macro blocks and Vector-Matrix Multiply (VMM) help users to design system level of applications.

Chapter 7 introduced a remote test system, enabled by configurable analog-digital ICs to create a simple interface for a wide range of experiments. The remote test system requires no additional setup, resulting both from using highly configurable FPAA devices, as well as from the advancement of straight-forward digital interfaces for the resulting experimental FPAA system. The system overhead requirements are straightforward, requiring simple email handling, available over almost all network systems with no additional requirements.

8.2 List of Contributions

- Analysis and test of RASP3.0 IC. The design of IC and test board was done by previous members in our group. I tested the functionality of element blocks (e.g., DACs, ADCs, a shift register) and the VMM operation using routing devices. I designed and measured a nonlinear classifier with a VMM + WTA structure. This work has been published in a TVLSI [12].

- Development of an FG programming algorithm for FG SoC FPAAs. Based on the analysis of the FG device's operation, I developed assembly code modules and Scilab scripts, enabling the erase, switch programming, and precise programming, while interfacing with the on-chip μP. The work has resulted in TVLSI [15].

- Measurement and analysis of scaled devices. This work has been done in collaboration with Farhan Adil. I measured I_d-V_g curves on 130nm and 40nm devices, and routing resistance and capacitance in 350nm, which resulted in a JLPEA publication [37].

- Tool development for the design compilation. The initial framework was made by Richard Wunderlich. I put a lot of efforts on debugging the tools / technology files to get a correct switch list, and improved it by adding functions (e.g., fix location, macro blocks) enabling mixed signal processing. This has been submitted to DAEM.

- Development of a remote system. I collaborated with Ishan Kumal Lal to develop the initial python codes handling emails and Sahil Shah to manage and improve the codes when it is used for GT classes. This work has been published in JLPEA [81].

- Development of calibration flow. I designed a calibration flow for FG FPAA ICs and developed codes including GUI. This work has been published to TVLSI [79].

- Design of RASP3.1 using IBM 130nm process. This work has been done in corporation with Sahil Shah.

- Design of a small version of RASP3.0 PCB board for commercialization.

REFERENCES

[1] Dawon Kahng and Simon M Sze. A floating gate and its application to memory devices. *Bell Labs Technical Journal*, 46(6):1288–1295, 1967.

[2] S. Lee, J. y. Lee, and et al. 7.5 a 128gb 2b/cell nand flash memory in 14nm technology with tprog=640??s and 800mb/s i/o rate. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 138–139, Jan 2016.

[3] G. G. Marotta, A. Macerola, and et al. A 3bit/cell 32gb nand flash memory at 34nm with 6mb/s program throughput and with dynamic 2b/cell blocks configuration mode for a program throughput increase up to 13mb/s. In *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 444–445, Feb 2010.

[4] D. Kang, W. Jeong, and et al. 256 gb 3 b/cell v-nand flash memory with 48 stacked wl layers. *IEEE Journal of Solid-State Circuits*, 52(1):210–217, Jan 2016.

[5] Y. Li, S. Lee, Y. Fong, F. Pan, T. C. Kuo, J. Park, T. Samaddar, H. Nguyen, M. Mui, K. Htoo, T. Kamei, M. Higashitani, E. Yero, G. Kwon, P. Kliza, J. Wan, T. Kaneko, H. Maejima, H. Shiga, M. Hamada, N. Fujita, K. Kanebako, E. Tam, A. Koh, I. Lu, C. Kuo, T. Pham, J. Huynh, Q. Nguyen, H. Chibvongodze, M. Watanabe, K. Oowada, G. Shah, B. Woo, R. Gao, J. Chan, J. Lan, P. Hong, L. Peng, D. Das, D. Ghosh, V. Kalluru, S. Kulkarni, R. Cernea, S. Huynh, D. Pantelakis, C. M. Wang, and K. Quader. A 16gb 3b/ cell nand flash memory in 56nm with 8mb/s write rate. In *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 506–632, Feb 2008.

[6] E. Harari. Flash memory - the great disruptor! In *2012 IEEE International Solid-State Circuits Conference*, pages 10–15, Feb 2012.

[7] N. Shibata, K. Kanda, T. Hisada, K. Isobe, M. Sato, Y. Shimizu, T. Shimizu, T. Sugimoto, T. Kobayashi, K. Inuzuka, N. Kanagawa, Y. Kajitani, T. Ogawa, J. Nakai, K. Iwasa, M. Kojima, T. Suzuki, Y. Suzuki, S. Sakai, T. Fujimura, Y. Utsunomiya, T. Hashimoto, M. Miakashi, N. Kobayashi, M. Inagaki, Y. Matsumoto, S. Inoue, Y. Suzuki, D. He, Y. Honda, J. Musha, M. Nakagawa, M. Honma, N. Abiko, M. Koyanagi, M. Yoshihara, K. Ino, M. Noguchi, T. Kamei, Y. Kato, S. Zaitsu, H. Nasu, T. Ariki, H. Chibvongodze, M. Watanabe, H. Ding, N. Ookuma, R. Yamashita, G. Liang, G. Hemink, F. Moogat, C. Trinh, M. Higashitani, T. Pham, and K. Kanazawa. A 19nm 112.8mm2 64gb multi-level flash memory with 400mb/s/pin 1.8v toggle mode interface. In

2012 IEEE International Solid-State Circuits Conference, pages 422–424, Feb 2012.

[8] Y. Li, S. Lee, K. Oowada, H. Nguyen, Q. Nguyen, N. Mokhlesi, C. Hsu, J. Li, V. Ramachandra, T. Kamei, M. Higashitani, T. Pham, M. Honma, Y. Watanabe, K. Ino, B. Le, B. Woo, K. Htoo, T. Y. Tseng, L. Pham, F. Tsai, K. h. Kim, Y. C. Chen, M. She, J. Yuh, A. Chu, C. Chen, R. Puri, H. S. Lin, Y. F. Chen, W. Mak, J. Huynh, J. Chan, M. Watanabe, D. Yang, G. Shah, P. Souriraj, D. Tadepalli, S. Tenugu, R. Gao, V. Popuri, B. Azarbayjani, R. Madpur, J. Lan, E. Yero, F. Pan, P. Hong, J. Y. Kang, F. Moogat, Y. Fong, R. Cernea, S. Huynh, C. Trinh, M. Mofidi, R. Shrivastava, and K. Quader. 128gb 3b/cell nand flash memory in 19nm technology with 18mb/s write rate and 400mb/s toggle mode. In *2012 IEEE International Solid-State Circuits Conference*, pages 436–437, Feb 2012.

[9] D. Lee, I. J. Chang, S. Y. Yoon, J. Jang, D. S. Jang, W. G. Hahn, J. Y. Park, D. G. Kim, C. Yoon, B. S. Lim, B. J. Min, S. W. Yun, J. S. Lee, I. H. Park, K. R. Kim, J. Y. Yun, Y. Kim, Y. S. Cho, K. M. Kang, S. H. Joo, J. Y. Chun, J. N. Im, S. Kwon, S. Ham, A. Park, J. D. Yu, N. H. Lee, T. S. Lee, M. Kim, H. Kim, K. W. Song, B. G. Jeon, K. Choi, J. M. Han, K. H. Kyung, Y. H. Lim, and Y. H. Jun. A 64gb 533mb/s ddr interface mlc nand flash in sub-20nm technology. In *2012 IEEE International Solid-State Circuits Conference*, pages 430–432, Feb 2012.

[10] A. Basu, S. Brink, C. Schlottmann, S. Ramakrishnan, C. Petre, S. Koziol, F. Baskaya, C. M. Twigg, and P. Hasler. A floating-gate-based field-programmable analog array. *IEEE Journal of Solid-State Circuits*, 45(9):1781–1794, Sept 2010.

[11] R. B. Wunderlich, F. Adil, and P. Hasler. Floating gate-based field programmable mixed-signal array. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(8):1496–1505, Aug 2013.

[12] S. George, S. Kim, S. Shah, J. Hasler, M. Collins, F. Adil, R. Wunderlich, S. Nease, and S. Ramakrishnan. A programmable and configurable mixed-mode FPAA SoC. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(6):2253–2261, June 2016.

[13] V. Srinivasan, G. J. Serrano, J. Gray, and P. Hasler. A precision cmos amplifier using floating-gate transistors for offset cancellation. *IEEE Journal of Solid-State Circuits*, 42(2):280–291, Feb 2007.

[14] V. Srinivasan, G. Serrano, C. M. Twigg, and P. Hasler. A floating-gate-based programmable cmos reference. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(11):3448–3456, Dec 2008.

[15] S. Kim, J. Hasler, and S. George. Integrated floating-gate programming environment for system-level ICs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(6):2244–2252, June 2016.

[16] L. R. Carley. Trimming analog circuits using floating-gate analog mos memory. *IEEE Journal of Solid-State Circuits*, 24(6):1569–1575, Dec 1989.

[17] E. Sackinger and W. Guggenbuhl. An analog trimming circuit based on a floating-gate device. *IEEE Journal of Solid-State Circuits*, 23(6):1437–1440, Dec 1988.

[18] C. Bleiker and H. Melchior. A four-state eeprom using floating-gate memory cells. *IEEE Journal of Solid-State Circuits*, 22(3):460–463, Jun 1987.

[19] Hiroshi Nozawa and Susumu Kohyama. A thermionic electron emission model for charge retention in samos structure. *Japanese Journal of Applied Physics*, 21(2A):L111, 1982.

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

[21] Paul Hasler, Chris Diorio, Bradley A Minch, and Carver Mead. Single transistor learning synapses. *Advances in neural information processing systems*, pages 817–826, 1995.

[22] M Lenzlinger and EH Snow. Fowler-nordheim tunneling into thermally grown sio2. *Journal of Applied physics*, 40(1):278–283, 1969.

[23] Carver A Mead. Scaling of mos technology to submicrometer feature sizes. *The Journal of VLSI Signal Processing*, 8(1):9–25, 1994.

[24] EH Nicollian and JR Brews. Mos physicsand technology. *New York: John Wiley&Sons*, 1982.

[25] Paul Hasler, Andreas G Andreou, Chris Diorio, Bradley A Minch, and Carver A Mead. Impact ionization and hot-electron injection derived consistently from Boltzmann transport. *VLSI Design*, 8(1-4):454–461, 1998.

[26] P. Hasler, A. Basu, and S. Kozil. Above threshold pFET injection modeling intended for programming floating-gate systems. In *2007 IEEE International Symposium on Circuits and Systems*, pages 1557–1560, May 2007.

[27] B. A. Minch and P. Hasler. A floating-gate technology for digital cmos processes. In *Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on*, volume 2, pages 400–403 vol.2, Jul 1999.

[28] Ragesh Puthenkovilakam and Jane P Chang. An accurate determination of barrier heights at the hfo 2/ si interfaces. *Journal of applied physics*, 96(5):2701–2707, 2004.

[29] C. Duffy and P. Hasler. Scaling pfet hot-electron injection. In *2004 Abstracts 10th International Workshop on Computational Electronics*, pages 149–150, Oct 2004.

[30] William Shockley. Problems related to pn junctions in silicon. *Solid-State Electronics*, 2(1):35IN961–60IN1067, 1961.

[31] Jason Luu, Jeffrey Goeders, Michael Wainberg, Andrew Somerville, Thien Yu, Konstantin Nasartschuk, Miad Nasr, Sen Wang, Tim Liu, Nooruddin Ahmed, Kenneth B. Kent, Jason Anderson, Jonathan Rose, and Vaughn Betz. VTR 7.0: Next generation architecture and CAD system for FPGAs. *ACM Trans. Reconfigurable Technol. Syst.*, 7(2):6:1–6:30, July 2014.

[32] A. Basu and P. E. Hasler. A fully integrated architecture for fast and accurate programming of floating gates over six decades of current. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(6):953–962, June 2011.

[33] M. Kucic, A. Low, P. Hasler, and J. Neff. A programmable continuous-time floating-gate fourier processor. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 48(1):90–99, Jan 2001.

[34] A. Bandyopadhyay, G. J. Serrano, and P. Hasler. Adaptive algorithm using hot-electron injection for programming analog computational memory elements within 0.2 percent of accuracy over 3.5 decades. *IEEE Journal of Solid-State Circuits*, 41(9):2107–2114, Sept 2006.

[35] O Girard. Openmsp430 project. *available at opencores. org*, 2013.

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

[37] Jennifer Hasler, Sihwan Kim, and Farhan Adil. Scaling floating-gate devices predicting behavior for programmable and configurable circuits and systems. *Journal of Low Power Electronics and Applications*, 6(3):13, 2016.

[38] R. Chawla, A. Bandyopadhyay, V. Srinivasan, and P. Hasler. A 531 nw/mhz, 128 times;32 current-mode programmable analog vector-matrix multiplier with over two decades of linearity. In *Proceedings of the IEEE 2004 Custom Integrated Circuits Conference (IEEE Cat. No.04CH37571)*, pages 651–654, Oct 2004.

[39] C. R. Schlottmann and P. E. Hasler. A highly dense, low power, programmable analog vector-matrix multiplier: The fpaa implementation. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 1(3):403–411, Sept 2011.

[40] B. Marr, B. Degnan, P. Hasler, and D. Anderson. Scaling energy per operation via an asynchronous pipeline. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(1):147–151, Jan 2013.

[41] C. Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, Oct 1990.

[42] Jennifer Hasler and Harry Marr. Finding a roadmap to achieve large neuromorphic hardware systems. *Frontiers in Neuroscience*, 7:118, 2013.

[43] S. Brink, J. Hasler, and R. Wunderlich. Adaptive floating-gate circuit enabled large-scale fpaa. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(11):2307–2315, Nov 2014.

[44] C. M. Twigg, J. D. Gray, and P. E. Hasler. Programmable floating gate fpaa switches are not dead weight. In *2007 IEEE International Symposium on Circuits and Systems*, pages 169–172, May 2007.

[45] John Lazzaro, Sylvie Ryckebusch, Misha Anne Mahowald, and Caver A Mead. Winner-take-all networks of o (n) complexity. In *Advances in neural information processing systems*, pages 703–711, 1989.

[46] S. Ramakrishnan and J. Hasler. Vector-matrix multiply and winner-take-all as an analog classifier. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(2):353–361, Feb 2014.

[47] P. Hasler, P. Smith, C. Duffy, C. Gordon, J. Dugger, and D. Anderson. A floating-gate vector-quantizer. In *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.*, volume 1, pages I–196–9 vol.1, Aug 2002.

[48] S. Y. Peng, P. E. Hasler, and D. V. Anderson. An analog programmable multidimensional radial basis function based classifier. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(10):2148–2158, Oct 2007.

[49] J. Oh, G. Kim, B. G. Nam, and H. J. Yoo. A 57 mw 12.5 uj/epoch embedded mixed-mode neuro-fuzzy processor for mobile real-time object recognition. *IEEE Journal of Solid-State Circuits*, 48(11):2894–2907, Nov 2013.

[50] 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 Journal of Solid-State Circuits*, 50(1):270–281, Jan 2015.

[51] G. E. R. Cowan, R. C. Melville, and Y. P. Tsividis. A vlsi analog computer/math co-processor for a digital computer. In *ISSCC. 2005 IEEE International Digest of Technical Papers. Solid-State Circuits Conference, 2005.*, pages 82–586 Vol. 1, Feb 2005.

[52] J. Becker, F. Henrici, S. Trendelenburg, M. Ortmanns, and Y. Manoli. A field-programmable analog array of 55 digitally tunable OTAs in a hexagonal lattice. *IEEE Journal of Solid-State Circuits*, 43(12):2759–2768, Dec 2008.

[53] F. Henrici, J. Becker, S. Trendelenburg, D. DeDorigo, M. Ortmanns, and Y. Manoli. A field programmable analog array using floating gates for high resolution tuning. In *2009 IEEE International Symposium on Circuits and Systems*, pages 265–268, May 2009.

[54] J. Becker and Y. Manoli. A continuous-time field programmable analog array (fpaa) consisting of digitally reconfigurable gm-cells. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, volume 1, pages I–1092–5 Vol.1, May 2004.

[55] J. Becker, F. Henrici, S. Trendelenburg, M. Ortmanns, and Y. Manoli. A continuous-time hexagonal field-programmable analog array in 0.13 um cmos with 186mhz gbw. In *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 70–596, Feb 2008.

[56] Massimo Antonio Sivilotti. Wiring considerations in analog vlsi systems, with application to field-programmable networks., 1991.

[57] B. Pankiewicz, M. Wojcikowski, S. Szczepanski, and Yichuang Sun. A field programmable analog array for cmos continuous-time ota-c filter applications. *IEEE Journal of Solid-State Circuits*, 37(2):125–136, Feb 2002.

[58] E. K. F. Lee and P. G. Gulak. Field programmable analogue array based on mosfet transconductors. *Electronics Letters*, 28(1):28–29, Jan 1992.

[59] C. A. Looby and C. Lyden. A cmos continuous-time field programmable analog array. In *Proceedings of the 1997 ACM Fifth International Symposium on Field-programmable Gate Arrays*, FPGA '97, pages 137–141, New York, NY, USA, 1997. ACM.

[60] Vincent Gaudet and Glenn Gulak. 10 mhz field programmable analog array prototype based on cmos current conveyors. In *Micronet Annual Workshop, Ottawa, Ontario*, 1999.

[61] T. S. Hall, C. M. Twigg, P. Hasler, and D. V. Anderson. Application performance of elements in a floating-gate fpaa. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, volume 2, pages II–589–92 Vol.2, May 2004.

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

[63] C. M. Twigg and P. Hasler. A large-scale reconfigurable analog signal processor (rasp) ic. In *IEEE Custom Integrated Circuits Conference 2006*, pages 5–8, Sept 2006.

[64] T. S. Hall, C. M. Twigg, J. D. Gray, P. Hasler, and D. V. Anderson. Large-scale field-programmable analog arrays for analog signal processing. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 52(11):2298–2307, Nov 2005.

[65] V. Srinivasan, D. W. Graham, and P. Hasler. Floating-gates transistors for precision analog circuit design: an overview. In *48th Midwest Symposium on Circuits and Systems, 2005.*, pages 71–74 Vol. 1, Aug 2005.

[66] Carver Mead. *Analog VLSI and Neural Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[67] D. W. Graham, E. Farquhar, B. Degnan, C. Gordon, and P. Hasler. Indirect programming of floating-gate transistors. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(5):951–963, May 2007.

[68] S. Shapero and P. Hasler. Mismatch characterization and calibration for accurate and automated analog design. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(3):548–556, March 2013.

[69] T. Wu, K. Mayaram, and U. K. Moon. An on-chip calibration technique for reducing supply voltage sensitivity in ring oscillators. *IEEE Journal of Solid-State Circuits*, 42(4):775–783, April 2007.

[70] G. E. R. Cowan, R. C. Melville, and Y. P. Tsividis. A VLSI analog computer/digital computer accelerator. *IEEE Journal of Solid-State Circuits*, 41(1):42–53, Jan 2006.

[71] S. Shah, C. N. Teague, O. T. Inan, and J. Hasler. A proof-of-concept classifier for acoustic signals from the knee joint on a fpaa. In *2016 IEEE SENSORS*, pages 1–3, Oct 2016.

[72] Christian C Enz, François Krummenacher, and Eric A Vittoz. An analytical MOS transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications. *Analog integrated circuits and signal processing*, 8(1):83–114, 1995.

[73] A. Low and P. Hasler. Cadence-based simulation of floating-gate circuits using the EKV model. In *Circuits and Systems, 1999. 42nd Midwest Symposium on*, volume 1, pages 141–144 vol. 1, 1999.

[74] B. A. Minch, P. Hasler, and C. Diorio. The multiple-input translinear element: a versatile circuit element. In *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, volume 1, pages 527–530 vol.1, May 1998.

[75] Michelle Collins, Jennifer Hasler, and Suma George. An open-source tool set enabling analog-digital-software co-design. *Journal of Low Power Electronics and Applications*, 6(1):3, 2016.

[76] C. R. Schlottmann and J. Hasler. High-level modeling of analog computational elements for signal processing applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(9):1945–1953, Sept 2014.

[77] S. Suh, A. Basu, C. Schlottmann, P. E. Hasler, and J. R. Barry. Low-power discrete fourier transform for OFDM: A programmable analog approach. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(2):290–298, Feb 2011.

[78] C. Huang, N. Lajnef, and S. Chakrabartty. Calibration and characterization of self-powered floating-gate usage monitor with single electron per second operational limit. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(3):556–567, March 2010.

[79] S. Kim, S. Shah, and J. Hasler. Calibration of floating-gate soc fpaa system. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, PP(99):1–9, 2017.

[80] J. Hasler, A. Natarajan, S. Shah, and S. Kim. Soc fpaa immersed junior level circuits course. In *2017 IEEE International Conference on Microelectronic Systems Education (MSE)*, pages 7–10, May 2017.

[81] Jennifer Hasler, Sahil Shah, Sihwan Kim, Ishan Kumal Lal, and Michelle Collins. Remote system setup using large-scale field programmable analog arrays (fpaa) to enabling wide accessibility of configurable devices. *Journal of Low Power Electronics and Applications*, 6(3):14, 2016.

[82] Jonathan Rose, Jason Luu, Chi Wai Yu, Opal Densmore, Jeffrey Goeders, Andrew Somerville, Kenneth B. Kent, Peter Jamieson, and Jason Anderson. The VTR project: Architecture and CAD for FPGAs from verilog to routing. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '12, pages 77–86, New York, NY, USA, 2012. ACM.

[83] UC Berkeley. Berkeley logic interchange format (blif). *Oct Tools Distribution*, 2:197–247, 1992.

[84] A. Doboli and R. Vemuri. Exploration-based high-level synthesis of linear analog systems operating at low/medium frequencies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(11):1556–1568, Nov 2003.

[85] F. Baskaya, D. V. Anderson, P. Hasler, and S. K. Lim. A generic reconfigurable array specification and programming environment (grasper). In *2009 European Conference on Circuit Theory and Design*, pages 619–622, Aug 2009.

[86] F. Baskaya, S. Reddy, Sung Kyu Lim, and D. V. Anderson. Placement for large-scale floating-gate field-programable analog arrays. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(8):906–910, Aug 2006.

[87] Scilab Enterprises et al. Scilab: Free and open source software for numerical computation. *Scilab Enterprises, Orsay, France*, page 3, 2012.

[88] Aishwarya Natarajan and Jennifer Hasler. Modeling, simulation and implementation of circuit elements in an open-source tool set on the fpaa. *Analog Integrated Circuits and Signal Processing*, pages 1–12, 2017.

[89] Jack J Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: past, present and future. *Concurrency and Computation: practice and experience*, 15(9):803–820, 2003.

[90] J. Hasler. Opportunities in physical computing driven by analog realization. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, Oct 2016.

[91] Ananda Maiti, Andrew D Maxwell, Alexander A Kist, and Lindy Orwin. Merging remote laboratories and enquiry-based learning for stem education. *International Journal of Online Engineering*, 10(6):50–57, 2014.

[92] V. J. Harward, J. A. del Alamo, S. R. Lerman, P. H. Bailey, J. Carpenter, K. DeLong, C. Felknor, J. Hardison, B. Harrison, I. Jabbour, P. D. Long, T. Mao, L. Naamani, J. Northridge, M. Schulz, D. Talavera, C. Varadharajan, S. Wang, K. Yehia, R. Zbib, and D. Zych. The ilab shared architecture: A web services infrastructure to build communities of internet accessible laboratories. *Proceedings of the IEEE*, 96(6):931–950, June 2008.

[93] D. Lowe, S. Murray, E. Lindsay, and D. Liu. Evolving remote laboratory architectures to leverage emerging internet technologies. *IEEE Transactions on Learning Technologies*, 2(4):289–294, Oct 2009.

[94] N. Sousa, G. R. Alves, and M. G. Gericota. An integrated reusable remote laboratory to complement electronics teaching. *IEEE Transactions on Learning Technologies*, 3(3):265–271, July 2010.

[95] M. A. Bochicchio and A. Longo. Hands-on remote labs: Collaborative web laboratories as a case study for it engineering classes. *IEEE Transactions on Learning Technologies*, 2(4):320–330, Oct 2009.

[96] M. Cooper and J. M. M. Ferreira. Remote laboratories extending access to science and engineering curricular. *IEEE Transactions on Learning Technologies*, 2(4):342–353, Oct 2009.

[97] K Park and Hyuk Kim. Remote fpga reconfiguration using microblaze or powerpc processors. *Application Note: XAPP441 (v1. 1) ed., Xilinx*, 2005.

[98] Randal Kuramoto. Quickboot method for fpga design remote update. *XAPP1081 (v1. 3)*, 18, 2014.

[99] W. H. Wolf. Hardware-software co-design of embedded systems [and prolog]. *Proceedings of the IEEE*, 82(7):967–989, Jul 1994.

[100] J. Hasler, S. Kim, S. Shah, F. Adil, M. Collins, S. Koziol, and S. Nease. Transforming mixed-signal circuits class through soc fpaa ic, pcb, and toolset. In *2016 11th European Workshop on Microelectronics Education (EWME)*, pages 1–6, May 2016.

[101] Tamara Dean. *Network+ guide to networks*. Cengage Learning, 2012.

[102] C. D. Knight and S. P. DeWeerth. A shared remote testing environment for engineering education. In *Frontiers in Education Conference, 1996. FIE '96. 26th Annual Conference., Proceedings of*, volume 3, pages 1003–1006 vol.3, Nov 1996.

[103] S. E. Poindexter and B. S. Heck. Using the web in your courses: what can you do? what should you do? *IEEE Control Systems*, 19(1):83–92, Feb 1999.

[104] Sven K Esche, Marehalli G Prasad, and Constantin Chassapis. Remotely accessible laboratory approach for undergraduate education. *age*, 5(5.525):1–5, 2000.

[105] Sven K Esche, Marehalli G Prasad, and Constantin Chassapis. Remotely accessible laboratory approach for undergraduate education. *age*, 5(5.525):1–5, 2000.

[106] Mr Firdous Saleheen. Design and evaluation of a web-based virtual open laboratory teaching assistant (volta) for circuits laboratory. *age*, 26:1, 2015.

[107] Marco Casini, Domenico Prattichizzo, and Antonio Vicino. E-learning by remote laboratories: A new tool for control education. *IFAC Proceedings Volumes*, 36(10):73–78, 2003.

[108] S Karthik, P Shreya, P Srihari, and NM Viswanath. Remote field programmable gate array (fpga) lab. *Int. J. Res. Eng. Technol*, 3:842–845, 2014.

[109] James F Kurose and Keith W Ross. *Computer networking: a top-down approach*. Addison-Wesley Reading, 2010.

[110] J. Vliegen, N. Mentcns, and I. Verbauwhede. A single-chip solution for the secure remote configuration of fpgas using bitstream compression. In *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1–6, Dec 2013.

[111] M. Surratt, H. H. Loomis, A. A. Ross, and R. Duren. Challenges of remote fpga configuration for space applications. In *2005 IEEE Aerospace Conference*, pages 1–9, March 2005.