

# AUTOMATIC RAPID PROGRAMMING OF LARGE ARRAYS OF FLOATING-GATE ELEMENTS

*G. Serrano, P.D. Smith, H.J. Lo, R. Chawla, T. S. Hall, C. M. Twigg and P. Hasler*

Department of Electrical Engineering Georgia Institute of Technology, Atlanta, GA 30332-0250.

## ABSTRACT

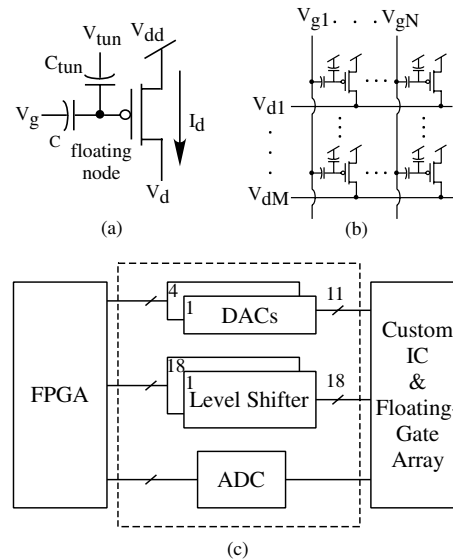
*The use of floating-gate elements on analog circuits has increased over the last few years. Floating-gate transistors are been used for analog multiplication, memory storage, on-chip bias, offset removals, etc. Complex systems, such as imagers and filter arrays, use thousands to millions of programmable floating-gate elements. The programming speed and precision of this elements plays a mayor role on the performance of these systems. In this paper we present a system approach that allows for automatic rapid programming of large arrays of floating-gates. We achieve this by optimizing all the time consuming tasks involved in the programming, such as current measurements and drain pulsing among others.*

## 1. AUTOMATIC ARRAY PROGRAMMING

Solid-State memory arrays have been around for many years in EEPROMs [1]. Although EEPROMs are used for digital storage, similar techniques have been used to implement analog storage [2]. As these arrays have become larger, programming speed has become critical in making these arrays useful. This paper outlines a method for achieving rapid programming speeds of large array of floating-gate elements.

Fig. 1(a) shows the schematic of a floating-gate (FG) transistor. It consists of a pFET transistor with the gate connected to a capacitor, hence this node does not have a dc path to ground. The primary characteristic of a floating-gate transistor is that it can store charge on the floating node. Charge can be removed or added via tunnelling and hot electron injection respectively. The details of floating-gate programming have been previously discussed in [3]. Tunnelling is used as a global erase, while injection is used for accurate programming of individual elements. Selectivity of a single element in an array is achieved using the configuration shown in Fig. 1(b). To select an specific FG, its respective row and column are enabled. Adjacent rows and columns are turned off by placing its drain and gate voltages to  $V_{dd}$ .

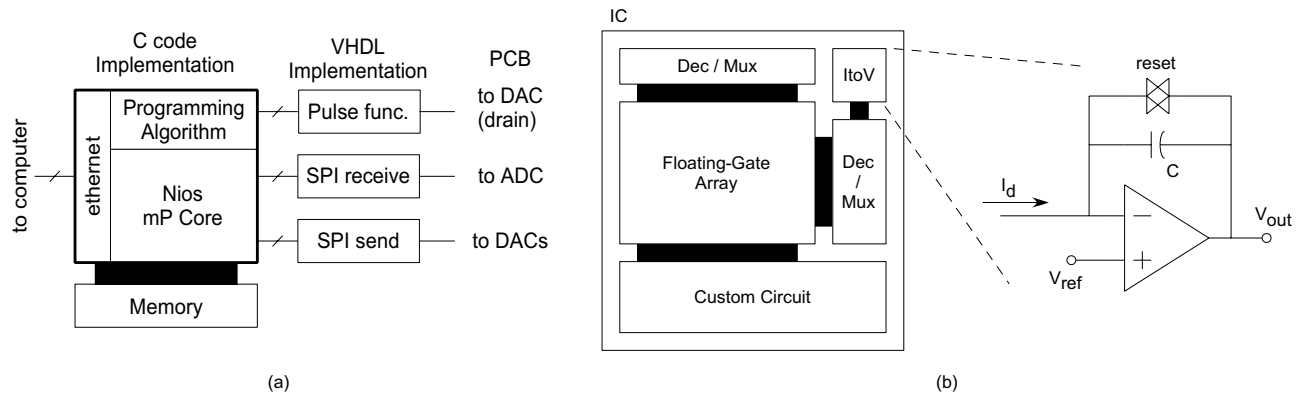
Details of injection have been discussed in [4]. To inject an electron onto a floating gate, the MOSFET must have a high electric-field region to accelerate channel electrons to energies above the silicon-silicon-dioxide barrier. When



**Fig. 1.** (a) Floating-gate transistor. Charge of the floating node can be store and programmed. (b) Floating-gate array. This configuration allows for a selectivity of a single floating-gate transistor when programming. (c) System block diagram. An FPGA provides fast digital control to a custom PC board. The PC board supplies the needed analog and digital voltages to the chip and allows for analog voltage reading. Additional digital I/O can be obtained from the FPGA directly.

this happens, electrons can be injected into the oxide and transported to the floating gate. To physically achieved this,  $V_{dd}$  is increased to a higher voltage (from 3.3V to 6.5V). All the other voltages are also increased the same amount ( $V_{gs}$ ,  $V_{ds}$ , and  $V_{ts}$  voltage values does not change). This process will be call from now on as voltage ramping. To get the high currents for injection, the drain voltage ( $V_d$ ) is then pulsed to a lower voltage for a certain amount of time ( $t_{pulse}$ ). After the injection is completed all voltages are restored to their original values. The electron injection will be a function of  $t_{pulse}$  and the  $V_{ds}$  voltage when pulsing.

Some of the factors that limit the floating-gate programming speed are the injection pulse, the current measurement, the voltage ramping, the amount of pulses needed to hit a target, and the communication with the PC. In order to inject, the bias voltages for the transistor have to be



**Fig. 2.** Block diagram of our software and hardware floating-gate array programming system. (a) Software architecture implemented on the FPGA. This controls the PC interface with matlab and the custom PCB for programming. (b) On-chip programmable analog array architecture. Individual programmable elements can be switched into the program loop using on-chip control circuitry. The current measurement has also been moved on chip using an I-to-V converter. The ItoV converter consists of an integrator structure with a reset switch in parallel to the capacitor. The reset will give us control of the integration. The input current,  $I_d$ , is integrated on the capacitor,  $C$ , thus producing an output voltage  $V_{out}$  that changes linearly with time. The speed of the integration will be proportional to  $C$ .

increased by  $3V$  or more. To avoid unwanted injection, these voltages must be increased or decreased in incremental steps. In legacy systems, it is typical to ramp the voltages up or down using 10 steps. The same procedure applies every time a floating-gate is pulsed, and ramping speed is limited by the DAC speed. After ramping up, a high drain-to-source voltage ( $V_{ds}$ ) is applied to the transistor over a short period of time ( $t_{pulse}$ ). Typical values for these pulses ranged from  $100\mu s$  to  $10ms$ . These pulses are repeated until the target current is achieved.

Using the programming algorithm in [3], the number of steps can be optimized for each target current. For a current within  $1nA$  and  $200nA$ , 30 steps are typically needed, since the instrumentation limits the smallest pulse to  $100\mu s$ . The injection algorithm computations were performed in the PC using Matlab. Serial communication between the PC and their custom board was restricted to  $116kbps$ . Current measurements were done using a commercial picoammeter with the measurement speed limited to the communication between the ammeter and the PC (a  $300kbps$  GPIB link).

## 2. SYSTEM IMPLEMENTATION

To improve the programming speed of large floating-gate arrays, a new programming platform has been developed. This platform consists of a custom printed circuit board to interface to the analog ICs and an FPGA board to control the programming board and communicate with the PC (Fig.1(c)). In addition, on-chip current measurements have been incorporated into the VLSI layout.

A four layer printed circuit board was fabricated with DACs, ADCs, and level shifters that provides 7 bias voltages (0-3.3V), 4 programming voltages (0-10V), 18 level

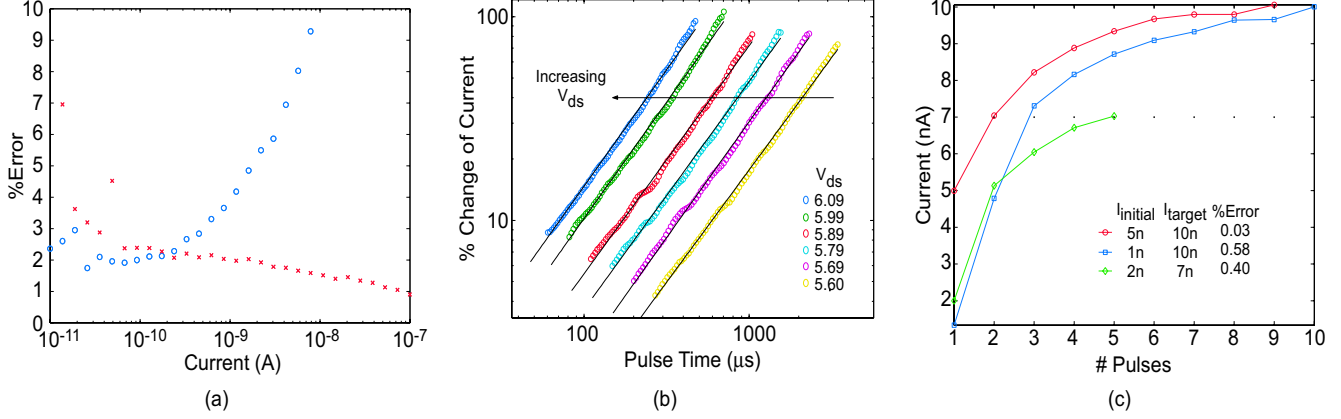
shifted digital signals, and one analog voltage measurement (0-5V). A clock of 10MHz is used for the DACs, while the ADC has a maximum rate of 100M samples per second. Fig. 1(c) shows the block diagram for the board, which is fully controlled by an FPGA.

The FPGA is configured to implement a customized soft-core processor (Altera's 32 bit Nios processor) along with specialized VHDL modules that handle timing critical communication between the programming board and the soft core processor. The Nios processor controls the overall system and coordinates the parallel operation of the different VHDL modules. In addition, software running on the Nios processor implements the TCP/IP protocol and communicates with the PC via a 100 Mbit Ethernet connection (Fig. 2(a)). On the PC side, a Matlab interface has been developed that provides a direct link to the FPGA from the Matlab command line.

## 3. CURRENT MEASUREMENTS

The current measurements are the primary bottleneck on floating-gate programming. A precise and fast reading is required for accurate programming. Measuring the current externally introduces some error, and the noise and parasitic capacitances introduced by the protoboard greatly decrease the accuracy. This will also affect the programming precision.

To get more accurate measurements, currents are measured on-chip using an integrator structure. This circuit, shown in Fig.2(b), consists of an amplifier with a capacitor connected in feedback loop.  $V_{ref}$  sets the drain voltage for the selected transistor. The current flows through the negative terminal of the amplifier and is integrated in the ca-



**Fig. 3.** (a) Programming accuracy of our on-chip I-to-V vs. a commercially available current meter. The increase in error at high currents is due to accuracy of the ADC. During high currents, the output voltage of the I-to-V moves faster than 1LSB change for the ADC. Future versions of the I-to-V will incorporate a sample-and-hold to remove this problem. (b) Injection occurs at pulses down to  $10\mu\text{s}$  and injection rates can be increased or decreased by controlling the injection voltage. The data shows pulses down to  $50\mu\text{s}$ , but this can be decreased because the FPGA can provide accurate pulses down to  $1\mu\text{s}$ . (c) Algorithmic injection for three different floating-gate elements. A target current of  $10\text{nA}$ ,  $10\text{nA}$ , and  $7\text{nA}$  was programmed for initial currents of  $5\text{nA}$ ,  $1\text{nA}$ , and  $2\text{nA}$  respectively. The difference on the target current and the actual programmed current are shown in the table. The programming time for each element was  $0.35\text{s}$  (using an integrator capacitance of  $10\text{pF}$ ).

capacitor, thereby effectively changing the DC current into an AC voltage (I-to-V converter). This voltage will not be susceptible to the output capacitance and will be less affected by noise. The reset signal is used to control the integration. A logic high disables the integration making  $V_{out} = V_{ref}$ , and a logic low enables the integration. The output voltage,  $V_{out}$ , is measured after a certain amount of time ( $dt$ ) has elapsed. The current is then calculated as follows

$$I = C \frac{dV_{out}}{dt}. \quad (1)$$

Fig.3(a) shows the percentage of error found on each current measurement using the ItoV converter, compared to the ammeter. For this experiment  $C$  was  $10\text{pF}$ . The ItoV shows a better performance than the ammeter at very low currents. The performance of this circuit is limited by the sampling rate of our ADC ( $200\text{k}$  samples per second). To increase our accuracy at higher currents, next design will include a sample and hold at the output of the ItoV. The speed of our current measurement will be limited by the magnitude of the current to be measured and the size of  $C$ . If we have  $C=300\text{fF}$ , we could measure a current of  $1.5\text{nA}$  in just  $200\mu\text{s}$ , for a  $1\text{V}$  drop at the output.

Averaging will increase the accuracy of measurement but will also increase the time required for the current measurement. To minimize the time delay an “intelligent” averaging is performed. This consists on increasing the averaging as we get closer to the target. This is implemented as a look up table that is loaded into the FPGA.

#### 4. ARRAY CHARACTERIZATION AND PROGRAMMING ALGORITHM

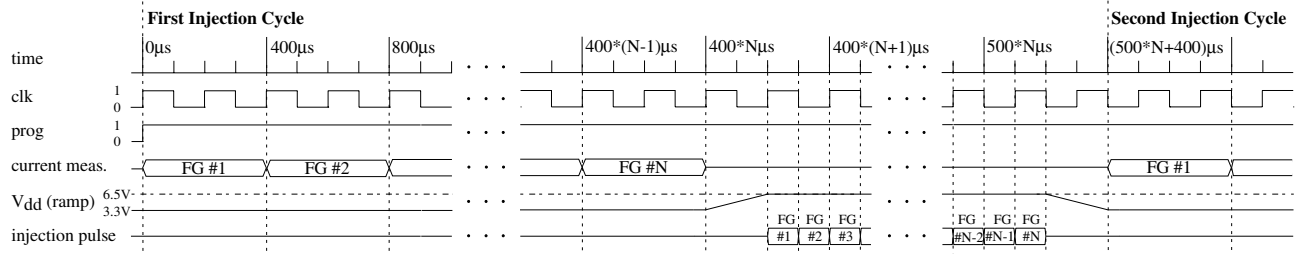
The basics of programming floating gate transistors and the inherent physics have been previously established. Accurate programming of the charge on the floating-gate is achieved using an algorithm based on the physics of hot-electron injection [3]. A change in drain voltage is calculated around a quiescent drain voltage that gives a 10 percent change in current for each injection pulse. The required drain voltage ( $\Delta V_d$ ) to reach a desired current is given by the following equation:

$$\Delta V_d = -\ln \left[ \frac{-C_0}{\alpha \cdot I_{inj_0} \cdot (t)} \left[ \left( \frac{I_{S_0}}{I_{S_{desired}}} \right)^\alpha - 1 \right] \right] \cdot V_{inj}, \quad (2)$$

where  $I_{S_0}$  is the initial current,  $I_{S_{desired}}$  is the target current,  $C_0 = U_T \cdot C_T / \kappa$ , and  $V_{inj}$ ,  $\alpha$ , and  $I_{inj}$  are physical device parameters.

Characterization of the floating-gate array plays a key role on the effectiveness and accuracy of the algorithmic programming. The physical device parameters can be extrapolated with two simple plots,  $\Delta I$  vs  $t_{pulse}$  and  $\Delta I$  vs  $I$ . Fig.3(b) shows experimental data of one of these plots. Here each of the plots for different  $V_{ds}$  were done with different devices. The fitting of the plots resulted in almost identical slopes, suggesting a low mismatch of the device parameters. This plot also shows that injection can occur under very low pulses ( $30\mu\text{s}$ ). Even smaller pulses can be used, with a higher  $V_{ds}$ .

The programming algorithm is computational intensive,



**Fig. 4.** Time diagram of the programming of an array of floating-gate transistors. The programming time is going to be limited by the current measurement, the injection pulse time, and the ramping. Values for these are  $400\mu s$ ,  $100\mu s$ , and  $200\mu s$  respectively. Programming starts by reading all the current in the array. Voltages are then ramped up and individual injection pulses are applied to every element. The voltages are restored to its respective values and the currents read again. This injection cycle will continue until the every element is programmed to its respective target current. Typical number of cycles when programming, ranges from 10 to 30 (these is going to depend on the starting current and the target current).

it requires solving (2) before every injection pulse. The present implementation transforms (2) into a look-up table that is loaded into the FPGA, thus avoiding any kind of computation and greatly increasing the programming speed. Now the flow of data is only between the PC board and the FPGA. Fig. 3(c) shows experimental data of injection using the algorithmic programming. Three different elements were programmed to different target currents using the programming scheme described in this paper. In just 10 pulses, the algorithm converges to the target current.

## 5. PROGRAMMING SPEED

The speed of our programming scheme can be underestimated as follows:

$$T = \left[ \left( t_{meas} + t_{pulse} \right) \cdot E + t_{ramp} \right] \cdot P \quad (3)$$

where  $t_{meas}$  is the time required to perform a current measurement,  $t_{pulse}$  is the time required for a pulse,  $t_{ramp}$  is the time required in ramping,  $E$  is the number of elements to be injected,  $P$  is the number of pulses (cycles), and  $T$  is the total time required to inject  $E$  elements Fig. 4. Typical values for  $t_{pulse}$  and  $t_{ramp}$  are  $10\mu s$  and  $200\mu s$ . Our current ItoV converter has a  $10pF$ , which limits the speed of our measurement to around  $50ms$  (including averaging) for a  $1nA$  current. Using  $t_{meas} = 50ms$ , and  $P = 10$ , the estimated time to inject one element to a target current is  $0.5s$ . Experimental measurements done for Fig. 3(c) showed a speed of  $0.35s$  for one element, which correlates with our estimate.

It can be seen clearly that the dominant factor in our speed is the current measurement. Future implementation will use a lower capacitance size for the integration, thus reducing  $t_{meas}$ . A  $100fF$  capacitance will give us a  $t_{meas}$  of  $400\mu s$  (including average). Assuming  $P=10$  for every element, it will take at most  $2s$  to program a  $500$  element array.

## 6. CONCLUSION

A system that allows a fast and accurate programming of floating gates has been developed. An FPGA controlled PC board was designed and build. This setup allows for optimization our programming speed and accuracy by minimizing all the time consuming steps involved in floating-gate programming.

With the use of the FPGA we can get injection pulses as small a  $1\mu s$ , which is 10 times smaller than before. By providing a  $10MHz$  clock to the DACs we lowered refreshing time needed for the bias voltages when ramping up and down. Still the refreshing time is going to be limited by the settling time of the DACs, which is  $5\mu s$ . On chip current measurements dramatically lowers the reading time. Data flow stays within the chip and the FPGA, avoiding using the PC as intermediary.

## 7. REFERENCES

- [1] D.Kahng and S. Sze, "A floating gate and its application to memory devices," *Bell Syst. Tech. J.*, vol. 46, p. 1288, 1967.
- [2] H. V. Tran, T. Blyth, D. Sowards, L. Engh, B. Nataraj, T. Dunne, H. Wang, V. Sarin, T. Lam, H. Nazarian, and G. Hu, "A 2.5 v 256-level non-volatile analog storage device using eeprom technology," *43rd IEEE International Solid-State Circuits Conference*.
- [3] P. D. Smith, M. Kucic, and P. Hasler, "Accurate programming of analog floating-gate arrays," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 5, (Phoenix, AZ), pp. 489–492, May 2002.
- [4] P. Hasler and J. Dugger, "Correlation learning rule in floating-gate pFET synapses," *IEEE Transactions on Circuits and Systems II*, 2000. in Press.