2015

A Parallel Programmer for Non-Volatile Analog Memory Arrays

Spencer L. Clites

A Parallel Programmer for Non-Volatile Analog Memory Arrays

by

Spencer L. Clites

Thesis submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

David W. Graham, Ph.D., Chair
Vinod Kulathumani, Ph.D.
Dimitris Korakakis, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2015

Keywords: Analog, Integrated Circuits, Floating Gate, Hot Electron Injection, Field-Programmable Analog Arrays

Abstract

A Parallel Programmer for Non-Volatile Analog Memory Arrays

by

Spencer L. Clites
Master of Science in Electrical Engineering

West Virginia University

David W. Graham, Ph.D., Chair

Since their introduction in 1967, floating-gate transistors have enjoyed widespread success as non-volatile digital memory elements in EEPROM and flash memory. In recent decades, however, a renewed interest in floating-gate transistors has focused on their viability as non-volatile *analog* memory, as well as programmable voltage and current sources. They have been used extensively in this capacity to solve traditional problems associated with analog circuit design, such as to correct for fabrication mismatch, to reduce comparator offset, and for amplifier auto-zeroing. They have also been used to implement adaptive circuits, learning systems, and reconfigurable systems. Despite these applications, their proliferation has been limited by complex programming procedures, which typically require high-precision test equipment and intimate knowledge of the programmer circuit to perform.

This work strives to alleviate this limitation by presenting an improved method for fast and accurate programming of floating-gate transistors. This novel programming circuit uses a digital-to-analog converter and an array of sample-and-hold circuits to facilitate fast parallel programming of floating-gate memory arrays and eliminate the need for high accuracy voltage sources. Additionally, this circuit employs a serial peripheral interface which digitizes control of the programmer, simplifying the programming procedure and enabling the implementation of software applications that obscure programming complexity from the end user. The efficient and simple parallel programming system was fabricated in a $0.5\mu m$ standard CMOS process and will be used to demonstrate the effectiveness of this new method.

A Parallel Programmer for Non-Volatile Analog Memory Arrays

by

Spencer L. Clites

Approved by:

Dr. David W. Graham, Chair
Associate Professor, LCSEE
West Virginia University

Dr. Vinod Kulathumani
Associate Professor, LCSEE
West Virginia University

Dr. Dimitris Korakakis
Professor, LCSEE
West Virginia University

Date Approved: April 29, 2015

Dedication

This work is dedicated to Betty M. Clites.

She would be proud.

Acknowledgments

First, I would like to express my gratitude to my adviser, Dr. David Graham, for his knowledge, guidance, and enthusiasm. I would also like to thank my labmates, Brandon Kelly, Alex Dilello, Mir Mohammad Navidi, and Stephen Andryzcik, for their assistance, suggestions, and useful discussion. I would especially like to thank Dr. Brandon Rumberg, whose work formed the foundation for my own and whose help proved invaluable.

Lastly, I would like to thank my friends and family. Thanks to my parents, Jeff and Jeanne, for their love and support. Thanks to my grandfather, Gary, for rewarding me with $1.50 each semester I earned straight A's. Thanks to my brother, Ben, for humoring me when I wanted to talk about electronics. Also, thanks to my sister and brother-in-law, Nicole and Matt, for their encouragement. Thanks to my girlfriend, Keri, for her love and adoration, and for making sure I got home safely after sleepless nights spent in the lab. A final thanks goes to all those that remain unnamed, who provided me with a life fulfilled, outside of my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Floating-gate (FG) transistors—also known as floating-gate metal-oxide semiconductor (FG-MOS) transistors or, simply, floating gates (FGs)—were first introduced by Kahng and Sze in 1967 as a method for non-volatile charge storage [1]. Since their inception, they have achieved great success in the form of non-volatile digital storage in erasable programmable read only memory (EPROM), electrically erasable programmable read only memory (EEPROM), and flash memories [2, 3, 4]. In recent years, floating-gate flash has become so well established that, in 2007, NAND flash bits began to overtake dynamic random access memory (DRAM) bits in sales for the first time in history [3]. They have continued to become even more popular through their expansion from mobile computing to replace disk storage in personal computers as solid-state drives (SSDs) [5].

Although much of the limelight is cast on digital applications, floating gates are not innately digital devices. Rather, it is the method used to perform the read and write operations that determines whether a digital or analog value is stored on the floating gate. It was not until the late 1980s, however, that an interest in floating-gate research emerged which explored their viability as non-volatile *analog* storage.

According to [6], some early innovations that helped establish the field of analog floating-gate research are Mead's adaptive retina circuit presented in [7], Shibata and Ohmi's neuron metal-oxide semiconductor field effect transistor (MOSFET) presented in [8], and Intel's electrically trainable artificial neural network (ETANN) presented in [9]. Many of these early FGMOS applications required ultra-violet (UV) light to modify the charge on the floating gates which proved cumbersome for real-time adaptations. Then, in 1991, a work presented by Thomsen and Brooke showed that electron tunneling could be performed using a standard complementary metal-oxide semiconductor

(CMOS) process, proving that real-time adaptation could be performed without needing access to specialized flash fabrication techniques [10]. This advancement made floating-gate research more easily accessible to a wider range of scientists.

Throughout the rest of the 1990s, much of the floating-gate research was aimed at applications in artificial neural networks and learning systems, due to the fact that the topology of floating-gate transistors naturally lends itself to these types of computations [11, 12, 13, 14, 15]. By the early-to-mid 2000s, floating-gate research started to branch out into more general reconfigurable systems. For instance in [16], the authors implemented a flash analog-to-digital converter (ADC) that autonomously adapted its internal reference voltages to match the characteristics of the input signal. This allowed the ADC to take full advantage of its 10-bits of resolution when converting signals with a multitude of different amplitude distributions such as linear, Gaussian, or exponential. In a similar capacity, floating gates have been used extensively as programmable voltage and current references [17, 18, 19].

Another facet of reconfigurable analog electronics are field-programmable analog arrays (FPAAs), which are the analog counterpart to digital field-programmable gate arrays (FPGAs). FPAAs contain a variety of common analog building blocks which can be configured to form analog systems. They facilitate rapid product prototyping by allowing a design to be tested on integrated hardware without having to adhere to a traditional analog integrated circuit (IC) design flow, which sometimes requires multiple product iterations to arrive at the final design. Floating gates have been used to a great extent in FPAAs to perform biasing, as non-volatile switches, and as programmable references [20, 21, 22]. In fact, our own reconfigurable analog/mixed-signal platform (RAMP) presented in [23] employs floating gates as tunable current references, which can be used for biasing or to modify various circuit parameters, such as transconductance.

Floating gates have also been used to solve more traditional problems associated with analog circuit design such as circuit trimming, offset correction, and auto-zeroing. In [24], [25], and [26], FGs were used to trim current sources. In [27] an FGMOS transistor was used to replace one of the differential input transistors in a two-stage Miller compensated op-amp in order to independently trim the input offset. Likewise, FGs were used for auto-zeroing of amplifiers in [28] and [29].

With such an extensive list of proven applications, why is it that analog implementations of floating gates are still largely confined to academia? We posit that it is due to the high complexity associated with accurately programming floating-gate analog memories. Programming such

memories usually requires high-resolution off-chip voltage or current references, high-resolution ammeters/voltmeters, data acquisition systems with multiple inputs/outputs, and, in most cases, intimate knowledge of the programming circuit. All these requirements make programming costly, complicated to perform, and cause analog FGMOS ICs to be prohibitive to use.

The objective of this work is to alleviate these concerns by presenting a novel floating-gate parallel programming circuit. This new circuit places an emphasis on digitizing the control of programming in order to greatly reduce programming requirements. This technique facilitates the communication between digital systems, such as microcontrollers, and a reconfigurable analog FG array. Furthermore, this work seeks to improve upon current programming techniques by presenting a new method for fast parallel programming.

1.1 Outline

The remainder of this work is organized as follows. Chapter 2 will provide an overview of floating-gate transistors, charge modification techniques, and some typical programming methodologies. Chapter 3 will introduce our novel programming circuit and each of its comprising functional blocks. Chapter 4 will present a programmable filter array employing our parallel programmer that has been fabricated as a proof-of-concept. Finally, Chapter 5 will expound upon broader applications of this programmer, draw conclusions, and propose the direction of future work.

Chapter 2

Introduction to Floating-Gate Transistors

Floating-gate analog memories have been shown to be useful in many applications. However, their proliferation has been hindered by the complex programming procedures required to use them. In the context of analog memory, the term *programming* refers to the process by which an accurate amount of charge is placed on the floating gate. In order to understand the limitations imposed by programming, it is necessary to understand how floating-gate transistors operate and how programming can be performed. Therefore, this chapter presents an introduction to FGMOS transistors, their operation, and typical programming procedures, as well as the respective benefits and drawbacks associated with these various programming techniques.

2.1 Floating-Gate MOSFET Operation

In addition to the bulk connection, a traditional metal-oxide semiconductor field effect transistor (MOSFET) has three terminals: the gate, the source, and the drain (Fig. 2.1 (a)). A Floating-gate (FG) MOSFET differs from this topology in that the gate electrode is electrically isolated by oxide through the addition of a capacitor in series with the gate, as shown in Fig. 2.1 (b). In this configuration, the input signal is applied to the control gate (CG) which couples onto the FG through capacitor, C_g, in order to modulate the current through the channel. This makes the channel current, I_d, a function of both V_{cg} and the amount of charge present on the floating-gate node, Q_{fg}. Given this relationship, and assuming all other nodes are fixed, it is apparent that the channel current can be modulated solely by modifying the amount of charge on the FG.

To quantify this relationship, first recall the expression for channel current of a typical MOS-

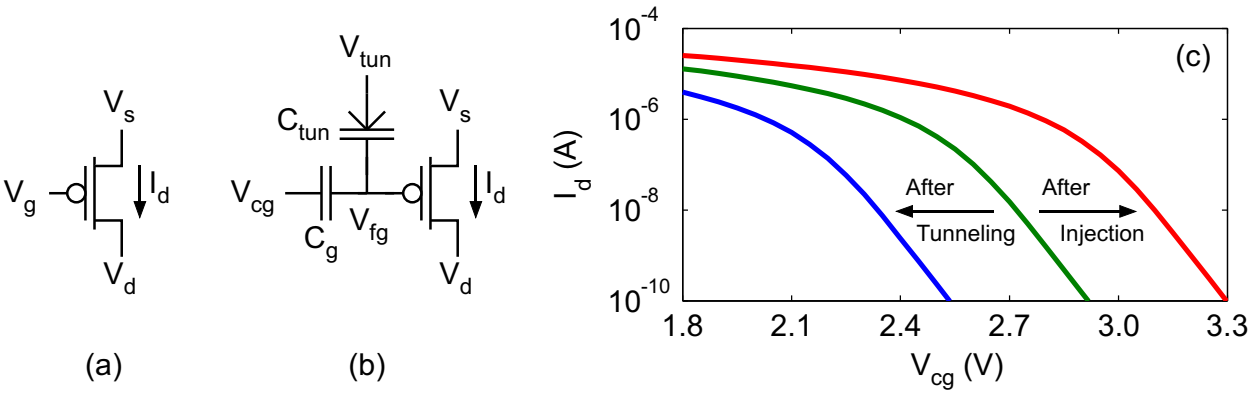

Figure 2.1: Comparison of a typical MOSFET and a floating-gate MOSFET. (a) Schematic symbol for a traditional p-channel MOS transistor. (b) Schematic symbol for a p-channel FGMOS transistor. (c) Gate sweep of FGMOS transistor illustrating the shift in threshold voltage due to programming. Injection adds electrons to the floating gate which decreases the threshold while the opposite effect is achieved by removing electrons through tunneling.

FET, such as in Fig. 2.1 (a). In low-power applications, FGMOS transistors are often biased in the sub-threshold regime ($V_{sg} < V_{TH}$); therefore, the following analysis assumes this operating regime.

$$I_d = I_0 \frac{W}{L} \exp\left(\frac{\kappa V_g}{U_T}\right)\left[\exp\left(-\frac{V_s}{U_T}\right) - \exp\left(-\frac{V_d}{U_T}\right)\right] \tag{2.1}$$

where I_0 is the pre-exponential current scaler, κ is the subthreshold slope, and U_T is the thermal voltage. For an FGMOS transistor, the V_g term in (2.1) is replaced by the effective floating-gate voltage given node voltages V_{cg}, V_d, V_s, V_{tun}, V_w, capacitances C_g, C_d, C_s, C_{tun}, C_w, and the floating-gate charge, Q_{fg}, yielding

$$V_{fg} = \frac{Q_{fg} + C_g V_{cg} + C_d V_d + C_s V_s + C_{tun} V_{tun} + C_w V_w}{C_{total}} \tag{2.2}$$

where C_{total} is the total capacitance present on node V_{fg}. Substituting (2.2) into (2.1) results in

$$I_d = I_0 \frac{W}{L} \exp\left(\frac{\kappa(Q_{fg} + C_g V_{cg} + C_d V_d + C_s V_s + C_{tun} V_{tun})}{C_{total} U_T}\right)\left[\exp\left(-\frac{V_s}{U_T}\right) - \exp\left(-\frac{V_d}{U_T}\right)\right] \tag{2.3}$$

However, C_g is typically drawn such that it dominates C_{total} in order to make V_{cg} dominate the coupling term [30], allowing the expression for V_{fg} to be simplified to

$$V_{fg} \approx \frac{Q_{fg} + C_g V_{cg}}{C_{total}} \tag{2.4}$$

Therefore, a simplified expression for drain current in an FGMOS transistor can be obtained by

substituting (2.4) into (2.1) to get

$$I_d = I_0 \frac{W}{L} \exp\left(\frac{\kappa(Q_{fg} + C_g V_{cg})}{C_{total} U_T}\right)\left[\exp\left(-\frac{V_s}{U_T}\right) - \exp\left(-\frac{V_d}{U_T}\right)\right] \tag{2.5}$$

Since the drain is typically connected to a low voltage, the V_d term becomes negligible and the expression can be simplified even further, resulting in

$$I_d = I_0 \frac{W}{L} \exp\left(\frac{\kappa(Q_{fg} + C_g V_{cg})}{C_{total} U_T}\right) \exp\left(-\frac{V_s}{U_T}\right) \tag{2.6}$$

Comparing (2.1) to (2.6), it can be shown that a change in Q_{fg} results in an effective shift of the threshold voltage of the FGMOS from the perspective of the control gate. Figure 2.1 (c) illustrates this effect by showing a gate sweep performed on an FGMOS transistor fabricated in a $0.5\mu m$ process, programmed to three different values for Q_{fg}.

2.2 Modifying the Floating-Gate Charge

In order to use FGMOS transistors in reconfigurable systems, some means of modifying Q_{fg} is required. Typically, charge modification of an FG is performed using two techniques: Fowler-Nordheim (FN) tunneling and hot-election injection. Hot-electron injection involves placing a high potential across the channel and is used to accurately place charge on the FG. This accurate placement of charge is henceforth referred to as *programming*. Fowler-Nordheim tunneling is typically reserved for global erasure of all programmed FGs across a chip, due to the difficulty of selecting individual devices when using the high voltages required for tunneling.

2.2.1 Fowler-Nordheim Tunneling

Fowler-Nordheim tunneling is the phenomenon in which electrons are placed under the influence of a large electric field, allowing them to tunnel through an oxide [31]. In this process, a high voltage is placed across a capacitor which reduces the effective thickness of the oxide. When the potential across the capacitor is sufficiently high, electrons are able to overcome the barrier and tunnel through the oxide. In an FGMOS, this tunneling capacitor, C_{tun} in Fig. 2.1 (b), is implemented using a simple MOS capacitor due to its thinner oxide when compared to a typical poly-insulated-poly capacitor, which lowers the voltage required to achieve FN tunneling and avoids catastrophic dielectric breakdown of the oxide [30]. Still, these voltages are high enough ($V_{tun} > 14V$ for $0.5\mu m$

CMOS process [17]) that they are difficult to isolate on-chip, so FN tunneling is typically used to globally remove charge from all FGs at a single time[1].

2.2.2 Hot-Electron Injection

Hot-electron injection occurs when electrons entering the drain collide with other electrons, generating impact-ionized hot electron-hole pairs. A portion of these resulting ionized electrons become elevated to sufficiently high energy levels allowing them to travel through the oxide onto the gate electrode. In non-FGMOS applications, hot-electron injection is an undesirable effect, so processing steps are added to mitigate it. In addition to this, the direction of the field lines in nFETs makes injection even more difficult to control, making pFETS a more suitable candidate for non-volatile analog memory in standard CMOS processes.

Injection current in PMOS floating-gate transistors can be expressed as

$$I_{inj} \approx \beta I_d^{\alpha} e^{V_{sd}/V_{inj}} \tag{2.7}$$

where I_d is the drain current, and β, α, and V_{inj} are device-dependent fits [33]. Thus, programming speed is a function of V_{sd} and I_d, as well as V_{gd}, which is lumped into the fit parameters of (2.7). For a $0.5\mu m$ CMOS process, injection requires $V_{sd} \geq 4.2V$ [17], which is significantly lower than V_{tun}. It is much easier to isolate these more moderate voltages to single-transistors on chip, making injection the preferred method for accurate programming of individual FGs.

2.3 Programming Methodologies

There is no standard way to program FGMOS analog memories. The charge modification technique tends to vary from designer-to-designer; however, some trends have emerged and will be discussed here. In general, one of two schemes is employed no matter the topology of the specific programmer circuit: pulsed programming or continuous programming.

2.3.1 Pulsed Programming

Pulsed programming is conceptually the simplest method for programming FGMOS transistors. In this method, short programming pulses, wherein a large V_{sd} is placed across the channel, are

[1]There are exceptions where FN tunneling is used for programming, such as in [32]. However, these are not the majority so this work reserves tunneling for global erasure.

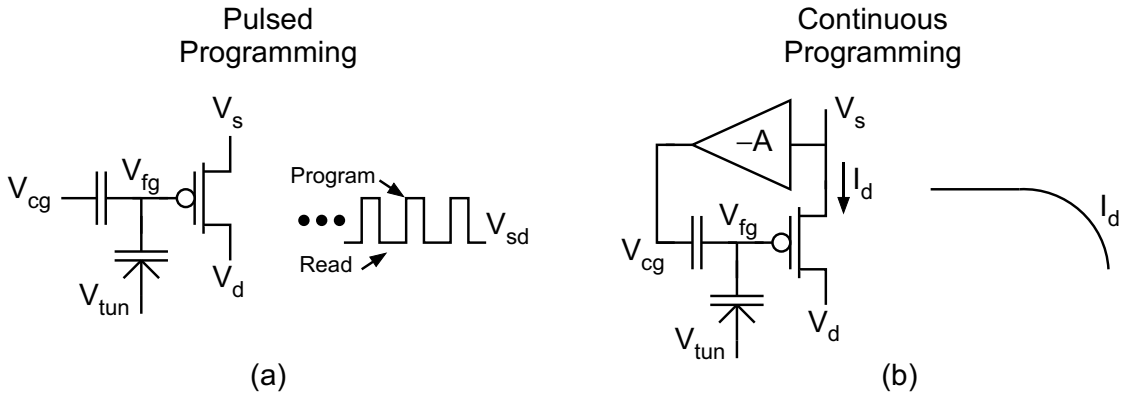

Figure 2.2: Comparison of pulsed and continuous programming techniques. (a) Pulsed programming (b) Continuous-time programming.

punctuated by read intervals during which the output current or voltage is measured (Fig. 2.2 (a)). When the desired current or voltage is observed during one of the reads, programming is ended, and the FG can be placed in its run-mode configuration, where it is connected to its circuit.

To make this method robust, accurate, and repeatable, some type of feedback is typically employed to ensure that the same amount of charge is injected onto the FG during each programming cycle [17]. As one might expect, this method requires large amounts of peripheral circuitry to switch between read and program modes. Moreover, the speed at which a single FG can be programmed is limited by the length of the program and read periods. Thus, larger targets greatly increase the amount of time it takes to finish programming. This is especially true when high accuracy is desired because each pulse must inject a finer amount of charge, requiring the number of programming pulses for a given target to be increased. This speed limitation is further compounded when an entire array of FGs must be programmed.

Some attempts have been made to mitigate the programming speed bottleneck associated with pulsed programming. For instance, in [34] the authors implemented a two-phased approach to programming that separates programming into a preliminary course-programming phase, and a primary fine-programming phase. This method is really a hybrid between pulsed and continuous programming since the course programming phase is one continuous programming period (no read intervals); this coarse phase works much like the self-convergent synapse transistor from [33] in that the drain raises as the FG is injected. A comparator is used to end the coarse programming phase when the drain of the FGMOS raises past a pre-defined point, indicating that I_d is approaching the target current. The fine programming phase employs the traditional pulse-based programming technique to accurately converge the rest of the way onto the target value. Another improved

pulsed programming technique was presented in [35] that uses a low number of fixed-width pulses of varying V_{sd} to quickly program an array of FGs to their targets.

Although these techniques reduce the overall programming time, they still have some significant drawbacks. In order to use the technique presented in [34], a voltage source with $>$ 12-bit precision is required along with some means of accurately measuring current. Complicating this method even further is the fact that it requires *a priori* knowledge of the floating gate transfer characteristic ($I_d = f(V_{cg})$) in order to achieve high accuracy. Similarly, the technique presented in [35] requires a characterization of $I_{inj} = f(V_{sd})$ to extract six parameters for each FG on chip in order to accurately determine the appropriate V_{sd} that will be used for each pulse. Obviously, these methods are extremely prohibitive as they require in-depth knowledge of the injection characteristics as well as high-precision hardware in order to use them.

Pulsed-based programming is not without its benefits, though. One benefit that pulsed programming has over continuous programming is that the FG memory cell is measured in a state similar to its run-time condition. Also, high programming accuracy has been reported, in some cases up to 13-bits of resolution over a $4V$ range of output voltages [17]. Still, the high programming overhead associated with pulsed programming techniques proves too high for dense analog memory arrays. Thus, this work posits that continuous programming provides a more accessible option for ease of programming.

2.3.2 Continuous Programming

The second method generally used to program FGs is continuous programming. Unlike in pulsed programming, continuous programming is performed in a single programming cycle after which the FG is placed back into a low-V_{sd} run-time condition in which its programmed value can be observed. Feedback is typically used to stop programming when a FG reaches its target. A number of continuous programming methods have been developed from simple single-transistor implementations with self-convergent memory writes [33] to more sophisticated implementations that achieve high speed and high accuracy.

In these more sophisticated designs, feedback is employed to maintain a constant V_{fg} throughout programming. For example, in [36], the authors presented a 3-transistor programming cell that employs a current-mirror to maintain a constant I_d through the channel of the FGMOS being programmed. A comparator monitors the drain voltage of the diode-connected pFET in the current

mirror and stops the programming when V_d exceeds the target value. The drawback in this approach is that the programming accuracy is related to programming speed such that lower speeds provide higher accuracy, since error is related to the input offset of the amplifier; in short, a large trade-off exists between programming speed and accuracy.

Therefore, an alternative linearization technique similar to that used in [17] is suggested. This is done by connecting an inverting amplifier between the source and control gate, which raises V_{cg} to compensate for Q_{fg} increasing due to injection (Fig. 2.2 (b)). In this programming scheme, V_{cg} can be monitored, and I_d is gently lowered as V_{cg} approaches the target value. A compact FG memory cell employing this technique has been previously reported in [37] that uses only 4-transistors: the FGMOS, two transistors operating as current sources, and a pFET common source amplifier in place of the inverting amplifier of Fig. 2.2 (b). This memory cell will be discussed in detail in the next chapter since it constitutes the analog storage element of the programmer circuit presented in this work.

2.3.3 Serial vs Parallel Programming

Another distinction in programming methodology that must be made is between serial and parallel programming. Pulsed programmers require much overhead, so they are usually confined to serial programming techniques [38]. Likewise, FGs rarely appear on chip as a single element, so the remainder of this subsection's discussion will focus on the trade-offs associated with continuous programming of FG arrays.

As their names imply, serial programming involves programming one floating gate at a time while parallel programming involves programming a number of floating gates simultaneously. Serial programming requires only one programmer circuit per chip since only one FG is selected at any moment in time. Likewise, only one external pin is required to supply the programming circuit with its target voltage. These characteristics are illustrated in Fig. 2.3 (a) for an example array of N FGs. As shown in the figure, this method is generally slow due to the high number of required programming cycles.

Conversely, parallel programming only requires one programming cycle to program all FGs in an array. However, in order to program N FGs in parallel, there must be N programmer circuits available, one for each FG. In order to program each FG to an independent target, this also requires N pins from the pad frame, supplying each V_{targ} to its programmer. This trade-off is shown in Fig.

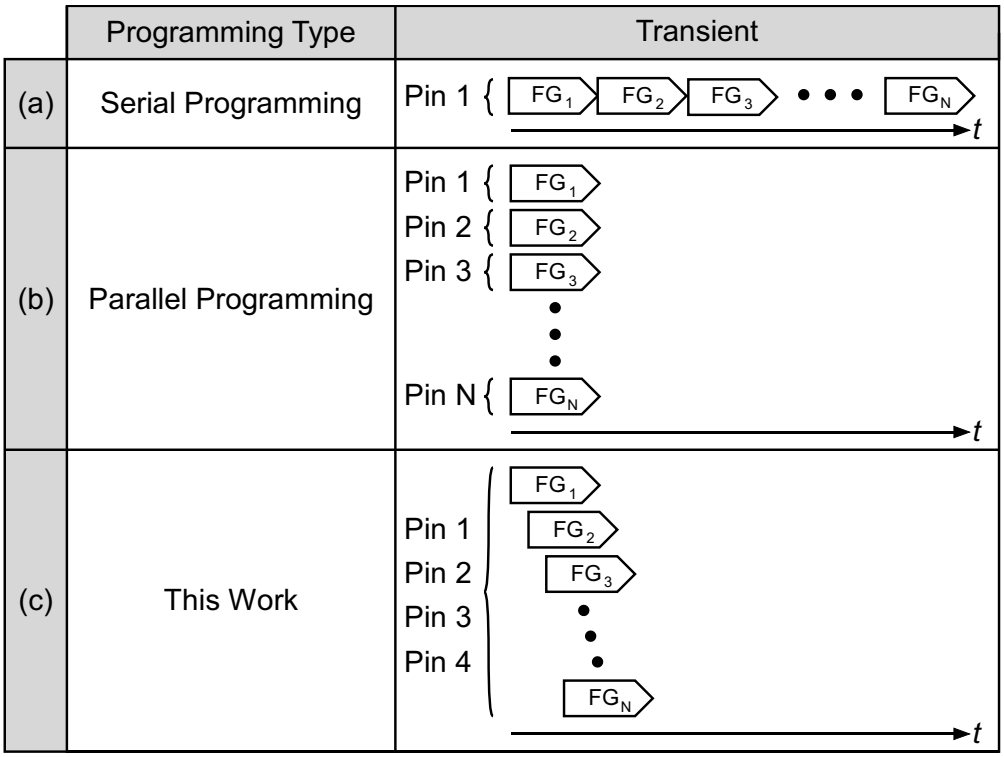

Figure 2.3: Comparison of serial, parallel, and this work's continuous programming techniques, when programming an array of FGMOS transistors. (a) Serial programming requires only one off-chip pin but suffers from long programming times. (b) Parallel programming requires an individual pin for each FGMOS in the array, however programming time is greatly reduced from serial method. (c) The programming method presented in this work requires only one off-chip pin and programming time is reduced through staggered parallel programming methodology.

2.3 (b) for an example array of N FGs.

Thus, serial programming requires less area on chip and less pins from the pad frame; however, it takes $(N-1)$ more programming cycles than a parallel implementation. On the other hand, parallel programming consumes more die area and more pins but only requires one programming cycle for an entire array. The programming method presented in this work, which will be introduced in the next chapter, presents a quazi-parallel programmer that requires only four pins, no matter the size of the array, and reduces the overall programming time by staggering the parallel programming through time as shown in Fig. 2.3 (c). More importantly, these four pins require only digital inputs, as opposed to the high precision analog inputs in figures 2.3 (a) and (b). This method still requires N programmers per N FGs, as with other continuous parallel programming techniques.

2.4 Chapter Summary

Floating-gate MOS transistors are formed by placing a capacitor in series with the gate of a MOSFET to leave the gate floating. The drain current of the resulting device becomes a function of

V_d, V_s, V_{cg}, and Q_{fg}. The charge stored on the FG can be modified using two techniques: Fowler-Nordheim tunneling, which is used as a technique for global charge erasure, and impact-ionized hot electron injection, which is used to accurately place charge on the FG. FN tunneling distorts the energy band of the tunneling capacitor to allow electrons to tunnel through the oxide off of the FG, while hot-electron injection elevates the electrons entering the drain to higher energy levels allowing them to travel onto the floating gate. Two main techniques are used for performing injection: pulsed programming and continuous programming. Pulsed programming entails periodically injecting and reading the FG value until a target is reached, whereas continuous programming requires only one programming period in which negative feedback is used to force Q_{fg} to converge to a target.

Also of importance are serial and parallel programming. Serial programming allows all FGs to share a single pin and programming circuit but requires more overall time to complete programming an entire array (Fig. 2.3 (a)). Parallel programming requires more pins and more die area but allows all FGs to be programmed more quickly since they are all done at one time (Fig. 2.3 (b)). In the next chapter, our parallel programmer will be introduced which achieves a compromise between these two by allowing the FGs to be programmed in parallel and also staggered through time, as shown in Fig. 2.3 (c). This achieves the approximate speed of parallel programming, a reduced pin count similar to serial programming, yet still requires the area of parallel programming. Thus the compromise still exists between speed and area.

Chapter 3

An Improved Parallel Programmer for Floating-Gate Transistor Arrays

In this chapter, a novel floating gate programming circuit is presented which addresses and mitigates the drawbacks of the traditional programming circuits discussed in the previous chapter. A block diagram of the circuit is shown in Fig. 3.1. In this topology, only digital input signals are required to program the full array of FG memory cells through the use of a serial peripheral interface (SPI). This minimizes the number of pins required to interface with the chip, reduces programming overhead, and removes some of the programming details from the end user. The digital outputs from the SPI are applied to a digital-to-analog converter (DAC) to generate analog target voltages which are then sampled by an array of sample-and-hold (S/H) circuits and applied to an array of FG memory cell circuits to perform programming. During programming, a DONE circuit monitors the control-gate voltages of each floating-gate and outputs a digital HIGH when all FGs in the array have finished programming. This allows for an entire array of FGs to be programmed in parallel without the need for a separate pin dedicated to each memory cell.

The circuit operates using two supply voltages: V_{dd} and $V_{dd,fg}$. As discussed in the previous chapter, injection requires source-to-drain voltages greater than $4.2V$ for a $0.5\mu m$ process. In this work, the high V_{sd} value is generated by raising the supply voltage of the programming circuit, in this case from $V_{dd,fg} = V_{dd} = 3.3V$ during run-time operation to $V_{dd,fg} = 6.5V$ during programming. Other circuit blocks are also required to operate from this elevated supply during programming, mainly inter-stage buffers since target voltages are above V_{dd}. The other high-voltage signal required is the tunneling voltage, V_{tun}. This work uses a tunneling pulse of $17V$ applied for $300ms$ in order to erase all FGs on chip.

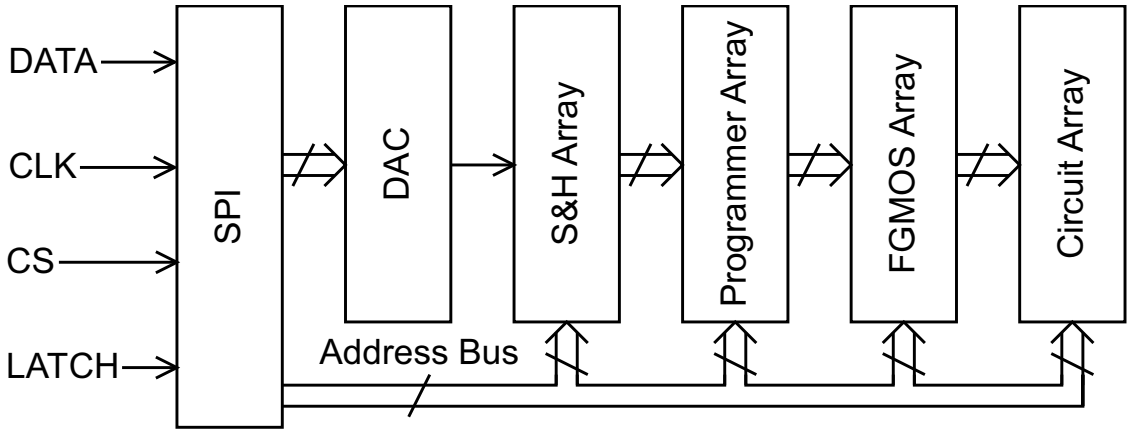

Figure 3.1: Signal flow diagram of the presented programming architecture. From left to right: The DATA, CLK, CS, and LATCH signals are used to load bits into the SPI to choose a DAC output voltage and select a S/H, programmer, FG, and circuit. The DAC output voltage is fed into the S/H whose output is connected to the programmer. The programmer injects the selected FG until it has reached its target. The programmed FG is connected to its circuit in run-time operation.

FGs are arranged in an $N \times M$ array so that each row can be programmed in parallel. In this configuration, selecting a different row only affects which FG is connected to a column's programming circuit. The programming circuit is arranged such that it is addressable by both *row* and *column*. Each column is comprised of a programming transconductor, its corresponding S/H, and an FG from the array. The programming process for this scheme operates as follows:

1. Erase all floating-gates using FN tunneling.
2. Raise programming power supply $V_{dd,fg}$ to its elevated level capable of causing injection.
3. Shift digital bits into the SPI to set the DAC output voltage and to select a specific row/column combination.
4. Sample target voltage from the DAC to set V_{targ}.
5. Start programming the selected FG memory cell.
6. Repeat steps 3-5 for each subsequent FG memory cell in the array.
7. When the DONE circuit outputs HIGH, lower $V_{dd,fg}$ to its run-time level and connect the FGs to their circuits for biasing.

Throughout the rest of this chapter, each block shown in Fig. 3.1 will be discussed individually along with measured data from a chip fabricated in a $0.5\mu m$ CMOS process. In Chapter 4, the blocks will be connected and overall performance will be measured.

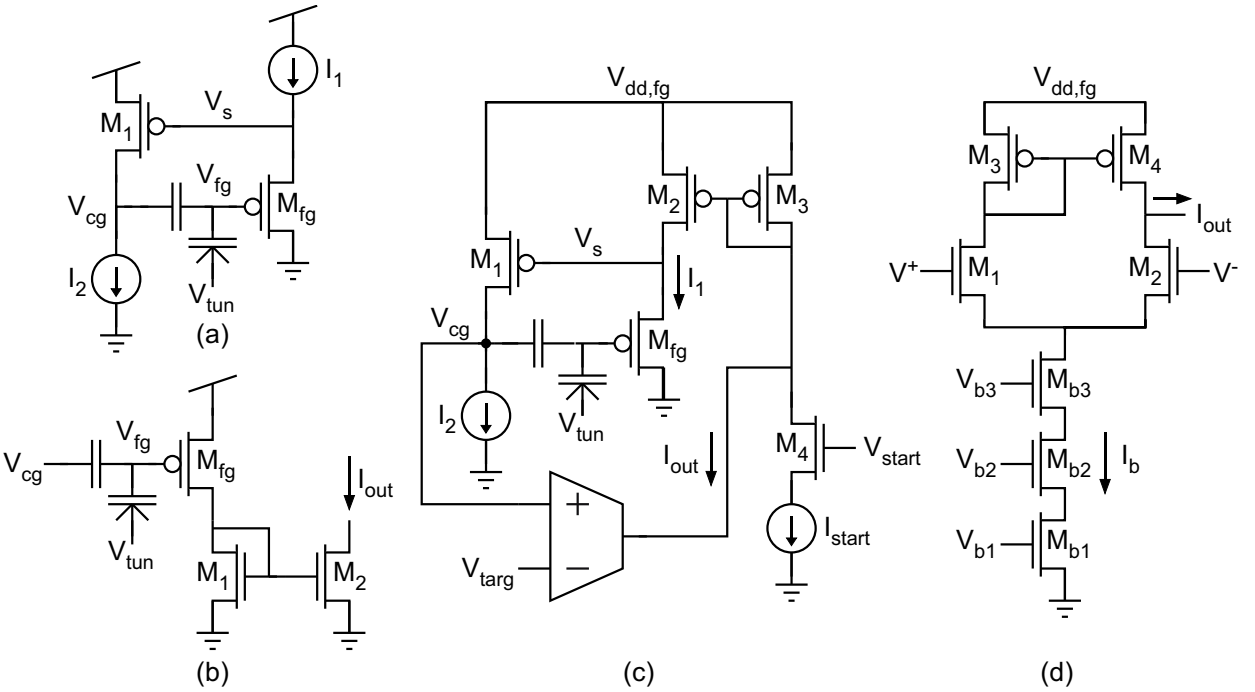

Figure 3.2: Overview of floating-gate memory cell. (a) Current conveyor FG memory cell in voltage output mode where V_{cg} is the output set by I_1, I_2, and V_{fg}. (b) Memory cell in current output mode where I_{out} is the output set by V_{cg} and Q_{fg}. (c) Continuous-time programming mode of the current conveyor. The CS amp formed by M_1 linearly raises V_{cg} as Q_{fg} is decreased through injection. When V_{cg} converges to V_{targ} the OTA shuts off current I_1 through M_{fg} to stop programming. (d) OTA used in programmer circuit. The tail is double-cascoded to maintain a constant I_b as $V_{dd,fg}$ is raised and lowered for programming.

3.1 Floating-Gate Memory Cell Array

The floating gate memory cell forms the basis of the programming circuit; therefore, it is apropos to begin the discussion with this block, as the performance of subsequent blocks will be designed around it. The FG memory cell chosen for this work is based on the one presented in [37] and is shown in Fig. 3.2. This topology is compact, requiring only 4 transistors, which allows for more dense scaling in large arrays, and is low overhead since only a target voltage is required to program V_{fg} to a specific value.

The memory cell has three configurations, the first of which is shown in Fig. 3.2 (a). This is the "voltage output" mode in which the memory cell is operating as a voltage reference, where V_{cg} is taken as the output. In this configuration, transistor M_1 forms a common-source amplifier providing negative feedback from V_s to V_{cg}, which forces V_{fg} so that the drain current $I_d = I_1$. Thus, if I_1 and I_2 are fixed, as the charge on the floating-gate is modified V_{cg} will be adjusted to maintain a constant drain current, $I_d = I_1$. Figure 3.2 (c) shows the memory cell in its programming configuration which operates using this same basic principle. During programming, the supply rail

is elevated to $V_{dd,fg}$ and as electrons are injected onto the floating-gate, the feedback from M_1 raises V_{cg} to keep V_{fg} constant. I_1 is set by the tail current of the programming operational transconductance amplifier (OTA) which cuts off I_1 as V_{cg} converges to V_{targ} to end programming. The third configuration is shown in Fig. 3.2 (b) which is the "current output" mode, in which it is operating as a current reference. In this configuration, V_{cg} is a fixed voltage bias (e.g. midrail) and I_{out} is the output current. Thus, I_{out} is modified by changing the charge stored on the floating-gate. Figure 3.2 shows the transistor-level schematic for the programmer OTA which is based on a 5-transistor OTA with an additional two transistors added to cascode the tail. This cascoding greatly increases the common-mode rejection ratio (CMRR), preventing the OTA from amplifying any changes in the common-mode voltage. This feature was added due to the fact that $V_{dd,fg}$ is raised and lowered throughout the programming process, so the OTA must be resistant to changes in the common-mode.

For the programming configuration in Fig. 3.2 (c), the injection current has been shown to be

$$I_{inj} \approx \beta I_1^{\alpha} \left(\frac{I_2}{I_0}\right)^{-\frac{U_T}{\kappa V_{inj}}} e^{V_{dd,fg}/V_{inj}} \tag{3.1}$$

where I_0 is the pre-exponential current scaler for M_1, κ is the subthreshold slope for M_1, U_T is the thermal voltage, and β and V_{inj} are device-dependent fits for M_1. The floating-gate transistors in this work were fabricated in a $0.5\mu m$ CMOS process and have dimensions $\frac{W}{L} = \frac{3\mu m}{1.2\mu m}$, $C_g = 80fF$, and $C_{tun} = 2fF$ which was implemented as a MOS capacitor with $\frac{W}{L} = \frac{1.5\mu m}{0.6\mu m}$.

The other notable feature of the programming circuit of Fig. 3.2 (c) is the addition of M_4 and I_{start}. In [37], programming is performed by first setting V_{targ} then raising $V_{dd,fg}$ from its run-mode voltage to an elevated level capable of injection. If the floating gate is sufficiently tunneled, then at the moment $V_{dd,fg}$ is raised, $V_{cg} < V_{targ}$. The output current of the OTA will be $I_1 = G_m(V_{cg} - V_{targ})$, where G_m is the transconductance of the OTA. Since $V_{cg} < V_{targ}$, I_{out} will be negative so the OTA will sink current, mirroring I_1 into M_{fg}, and the memory cell will immediately begin programming.

However, in our parallel programmer $V_{dd,fg}$ has to be raised *before* V_{targ} is set, since the DAC buffer and the S/H buffer are operated from the elevated supply during programming. When the circuit is operated in this order, the memory cell enters a zero-current stable state and requires a start-up circuit to force current into the programming loop. The solution implemented in this work was the addition of transistor M_4 which, when V_{start} is pulsed high, pulls the drain/gate of M_3

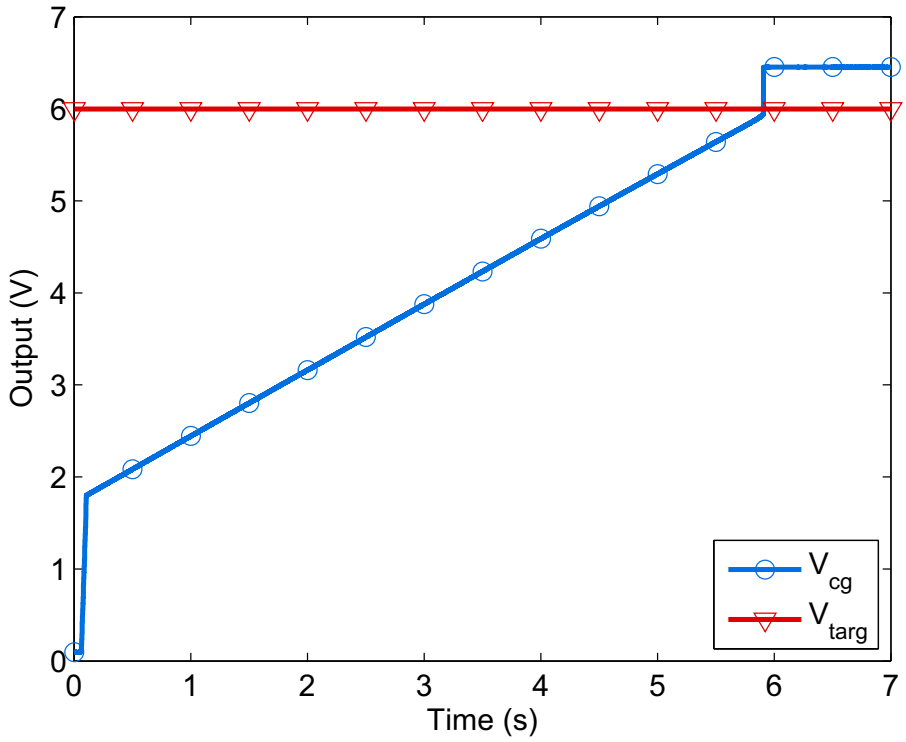

Figure 3.3: Transient response of node V_{cg} of the memory cell during programming, for $V_{targ} = 6V$. V_{start} is pulsed at t = 0s, immediately after which programming starts. During programming, V_{cg} raises as Q_{fg} is raised by injection, until $V_{cg} \approx V_{targ}$ and programming is ended.

down to force a non-zero I_1 into M_{fg} to initiate the loop. As an added precaution, current limiter I_{start} was added to prevent the start-up circuit from causing too much charge to be injected onto the floating gate during the V_{start} pulse. The circuit was designed such that I_{start} in Fig. 3.2 (c) is equal to I_b in Fig. 3.2 (d).

3.1.1 Measured Performance

The programming accuracy of the memory cell was tested using target voltages $4V < V_{targ} < 6V$, $V_{dd,fg} = 6.5V$, $I_b = 250nA$, $I_1 = 100nA$, and $I_2 = 2nA$, which were empirically determined to provide good operation using a previously-fabricated test chip. These same values are used throughout the rest of this work. Figure 3.3 shows V_{cg} during programming for $V_{targ} = 6V$. V_{start} is pulsed at $t = 0s$ to begin programming, then V_{cg} increases as Q_{fg} is increased due to injection, until $V_{cg} \approx V_{targ}$ and programmng is ended.

To measure the programming accuracy, the voltage output configuration of Fig. 3.2 (a) was used since this configuration has a linear V_{targ} to $V_{cg,out}$ relationship. Figure 3.4 (a) shows the average $V_{cg,run}$ vs V_{targ} for $V_{targ} = 4V$ to $6V$ out of 25 trials per target. A line of best fit was determined for this data set and the average deviation from this curve was determined and is presented in Fig.

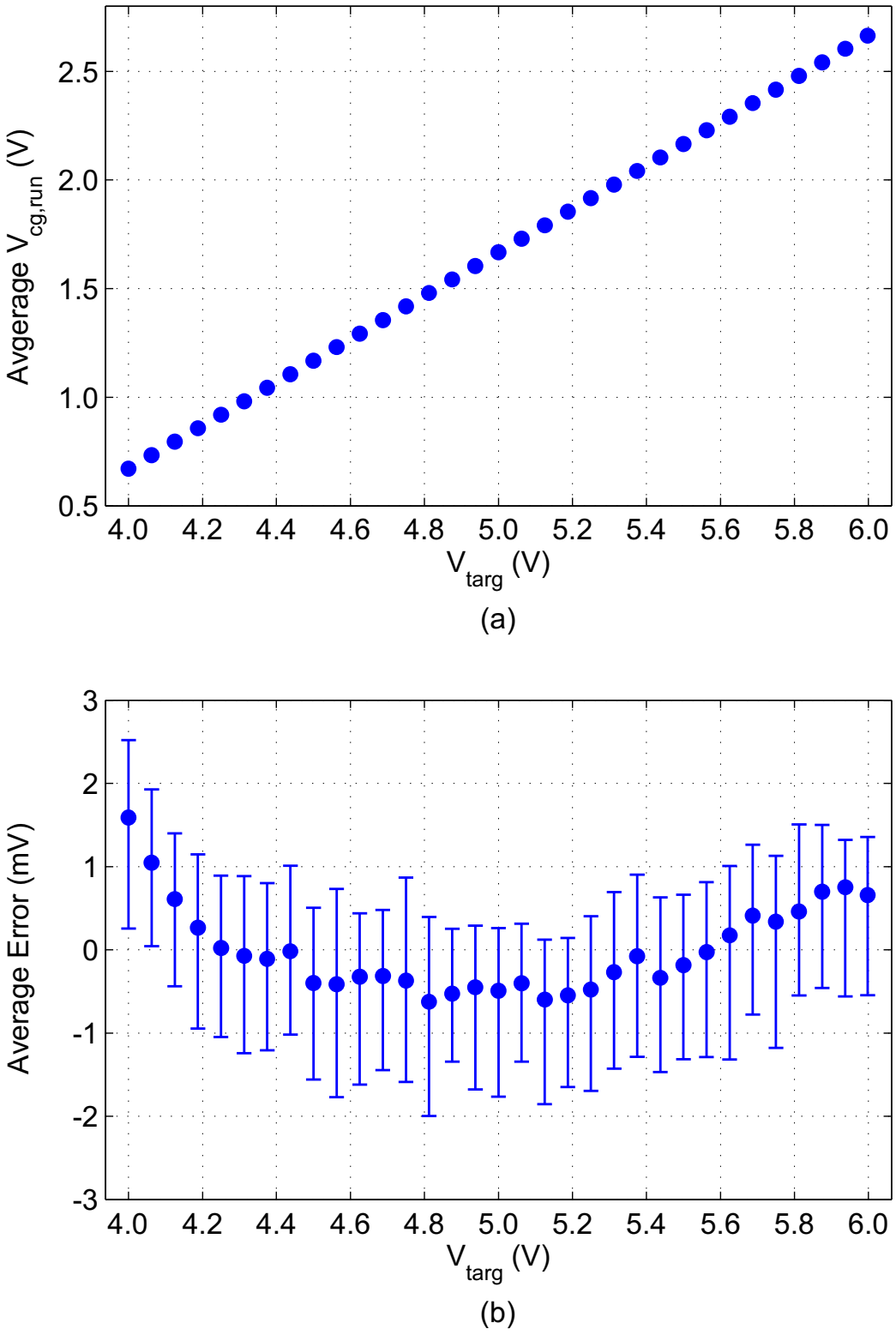

Figure 3.4: Programming accuracy of floating-gate memory cell. (a) Average $V_{cg,run}$ vs V_{targ} out of 25 programming trials per target. (b) Average deviation of $V_{cg,run}$ from linear response vs V_{targ} out of 25 programming trials per target. The error bars represent the range of measured values.

3.4 (b). The maximum overall error was determined to be $2.5mV$ which corresponds to an effective number of bits, or ENOB, equal to 9.63-bits for a $2V$ range of targets according to

$$\text{ENOB} = \log_2\left(\frac{\text{FSR}}{\text{error}}\right) = \log_2\left(\frac{2V}{2.5mV}\right) = 9.63\text{-bits} \tag{3.2}$$

where FSR is the full scale range equal to $2V$. The origin of this term will be discussed in further detail when the DAC is presented. Additionally, a gain error of -0.17% was measured, which is presumed to be due to the finite gain of the common source amplifier, M_1 in Fig. 3.2 (c), which provides feedback from the source to the control gate. The $V_{cg,run}$ vs V_{targ} plot should ideally have unity gain, meaning that a $1mV$ increase in V_{targ} corresponds to a $1mV$ increase in $V_{cg,out}$. In the measured circuit a gain error of -0.17% means that a $1mV$ increase in V_{targ} corresponds to only a $998.3\mu V$ increase in $V_{cg,out}$.

3.2 Serial Peripheral Interface

The serial peripheral interface (SPI) constitutes the front end of the programming circuit. This essential circuit block allows the user to interface with the chip using only digital signals, which greatly simplifies the programming process and also lends itself to automated programming routines through the creation of custom software programs. The SPI consists of 18 D-flip-flops (DFFs) that make up the shift register as well as 18 DFFs that comprise the static random access memory (SRAM) for each bit (Fig. 3.5). Control signals for this SPI are as follows: CLK, CS, LATCH, and DIN. CLK is the clock signal that is used to shift the data, DIN, into the shift register. LATCH is the clock signal that latches the bits held in the shift register into the SRAM. Lastly, CS (Chip Select) is used to either enable or disable the SPI. Additionally, DOUT is an output signal that is used to observe the bits being shifted out of the shift register for debugging purposes.

The purpose of the SRAM is to buffer the rest of the chip from the shift register. This allows the shift register to be operated without the bits being applied to the chip (DAC, address bus, etc.), until all 18 bits have been loaded. Once all the bits have been set, they get applied to the chip upon clocking the LATCH signal to latch the bits into the SRAM. The CS signal is used to ease the process of sending control signals to the chip. CS simplifies the requirements of the programming software routine since it removes the need to maintain a global bitmask variable to keep track of the state of any static digital outputs from the data acquisition system that was used for testing.

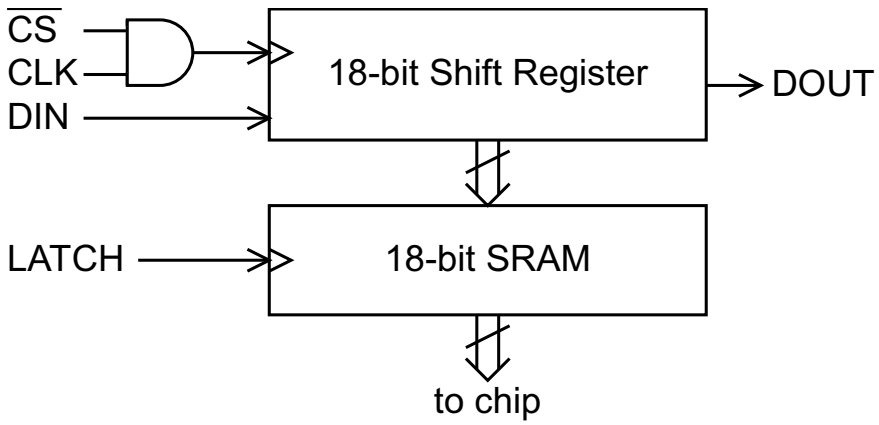

Figure 3.5: Block diagram of the serial peripheral interface. The SPI is comprised of two main blocks: an 18-bit shift register and 18-bits of SRAM. The shift register can be enabled/disabled using the Chip Select (CS) signal which passes/blocks the CLK signal from operating the shift register, respectively. The LATCH signal is used to latch the bits in the shift register into the SRAM which is used to buffer the SPI from the DAC and address lines while new data bits are being shifted into the SPI.

This allows the CS signal to select between using the SPI and sending the digital signals to the chip's circuitry (i.e. sample pulse of the sample-and-hold and V_{start} to the memory cell).

The 18-bits of the SPI are allocated as shown in Table 3.1, where bit 0 is the LSB and bit 17 is the MSB. The bits of data get shifted into the SPI serially from LSB to MSB. Bits 0 and 1 together configure the selected memory cell in one of the three configurations discussed in the previous section, as well as determine whether or not it gets connected to the circuit which it is biasing. Bit 2 is the enable pin for a multiplexer that allows the V_{targ} voltage to the programmer to be applied from an external pin. This is only used for testing and debugging purposes. Bit 3 is the 1-bit row address and bits 4 through 6 are the 3-bit column address. Bit 7 is the enable signal for the pull-down switch connected to the output of the DAC buffer, which will be discussed in the next section. Finally, bits 8 through 17 form the 10-bit input codeword that is sent to the DAC to select an analog target voltage.

Table 3.1: SPI Bit Assignments

Bit(s)	**Function**
0	enable voltage output mode
1	enable circuit connection
2	V_{targ} pin select
3	1-bit row address
4-6	3-bit column address
7	DAC V_{out} pulldown enable
8-17	10-bit DAC input word

3.3 Digital-to-Analog Converter

A digital-to-analog converter (DAC) is a mixed-signal circuit that converts a series of digital bits, called a codeword, into a corresponding analog output voltage. It does this by dividing a reference voltage, V_{ref}, into a number of equidistant voltages and passing one of these voltages to the output, according to the applied codeword.

3.3.1 DAC Metrics

To better understand the design requirements of the DAC, some metrics commonly used to characterize the performance of data converters will first be introduced. The *resolution*, N, of a DAC is equal to the number of bits in the codeword. For an N-bit DAC, there are 2^N unique output voltages. One *least-significant bit* (LSB) of a DAC is equal to the increase in output voltage yielded by increasing the input codeword by one least-significant bit. The expression for 1 LSB is given by

$$1\ LSB = \frac{V_{ref}}{2^N}\ (V) \tag{3.3}$$

Another useful quantity to mention is the *full scale* (FS) voltage, which is equal to the highest possible output voltage, corresponding to the all-ones codeword. In DACs, the lowest output voltage is usually equal to ground, meaning that the highest output voltage is equal to

$$FS = V_{ref} - \frac{V_{ref}}{2^N} = V_{ref} - 1\ LSB\ (V) \tag{3.4}$$

Not to be confused with the FS, the *full-scale range* (FSR) is equal to the maximum output voltage of an ideal infinite-resolution DAC, which is simply equal to V_{ref}. A DAC is considered *monotonic* if an increase in the input codeword always results in an increase in V_{out}.

3.3.2 Design Considerations

The DAC topology was chosen to be a simple voltage-scaling resistive divider in order to minimize design complexity, since this was a proof-of-concept chip. Other DAC topologies (e.g. charge redistribution, pipeline, etc.) can offer advantages such as smaller die area, higher resolution, and higher conversion speed; however, resistive divider DACs have the benefit of guaranteed monotonicity, as well as being simple to design and operate. The DAC was implemented as a series string

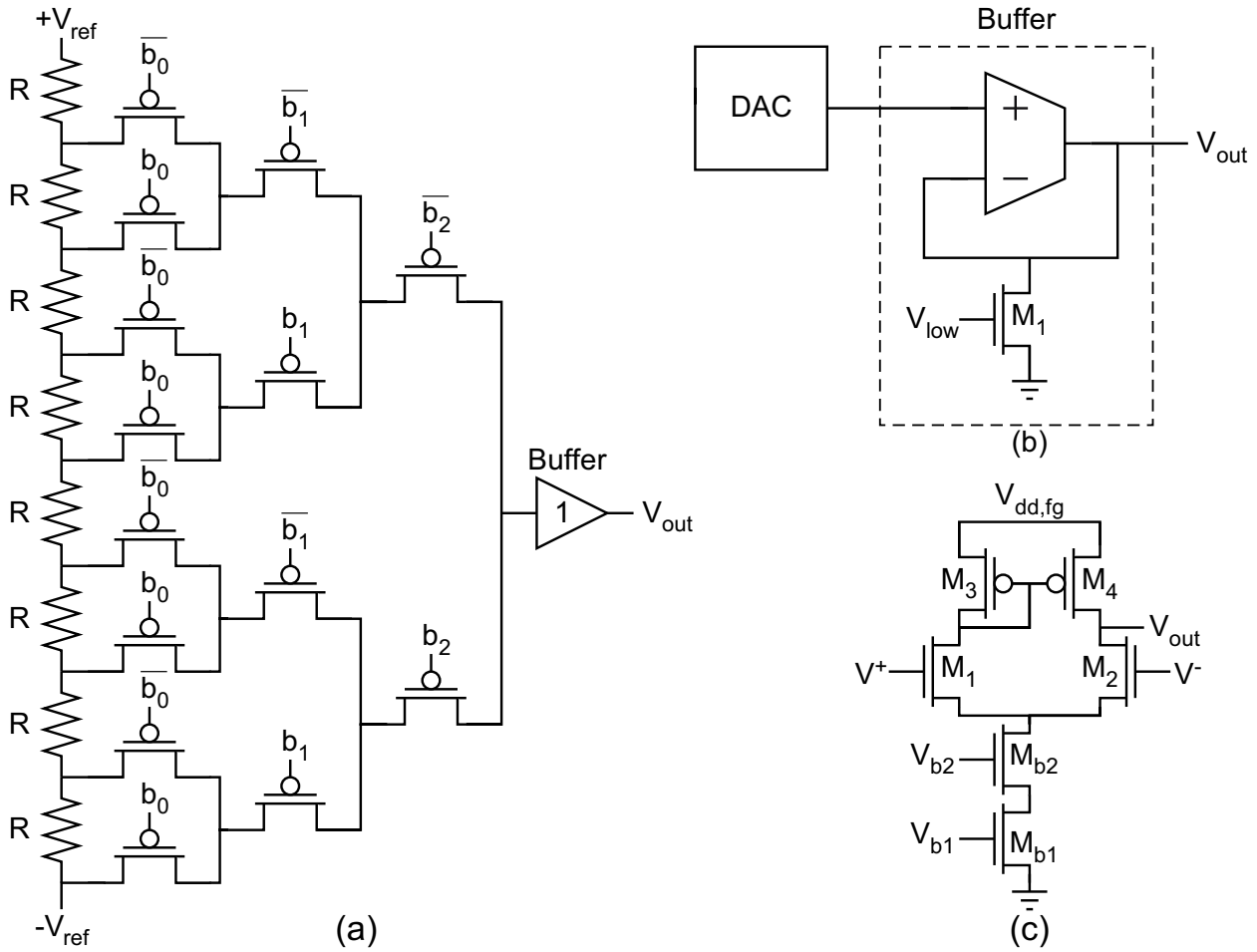

Figure 3.6: Overview of the DAC topology. (a) A 3-bit DAC illustrating the voltage-scaling resistive topology used in this work. The actual fabricated DAC is 10-bits but for brevity an example 3-bit one is shown here. (b) DAC buffer includes a pull-down switch which allows the DAC output to reach near-ground when V_{low} is held high. (c) Transistor schematic of OTA used in DAC buffer.

of equal-valued resistors, where each resistor is referred to as a segment. This string of resistor segments performs voltage division so that the voltage drop across each resistor is the same. A "tap" is made between each segment and a decoder selects the appropriate tap according to the applied input codeword. The DAC was designed to have a resolution of 10-bits, which exceeds the 9.63-bit programming accuracy of our FG memory cell across the required $2V$ FSR, yielding an LSB equal to $\sim 1.953mV$. The decoder was implemented as a simple switch tree which reduces overall area compared to a 10-to-1024 digital decoder that could otherwise be used. The area consumption was further reduced by implementing each switch as a single pFET transistor, as opposed to a transmission-gate switch, which was possible because only high voltages are passed through the switches, so transmission-gates are not required. The fabricated DAC contains 1024 resistors and 2046 transistors so a full schematic cannot be shown, thus an example 3-bit DAC of the same topology is shown in Fig. 3.6 (a). Each segment was implemented as a 5kΩ n-diffusion resistor

and each switch has a $\left(\frac{W}{L}\right) = \frac{0.6\mu m}{0.6\mu m}$. For our programmer, the DAC's output is required to operate across a $2V$ FSR between 4 to 6 volts so V_{ref} is equal to $2V$ and is applied differentially, so that $+V_{ref} = 6V$ and $-V_{ref} = 4V$.

3.3.3 Measured Performance

The output transfer characteristics of the fabricated DAC were measured with $2V$ across $\pm V_{ref}$ and are shown in Fig. 3.7 (a), for every 32nd input word. Since the application requires solely DC output voltages, only static operating characteristics were measured. Thus, only the differential non-linearity (DNL) and integral non-linearity (INL) errors were measured. The DNL is the difference between the non-ideal and ideal voltage steps between input codewords while the INL is the difference between the absolute output voltage for a given codeword and its ideal response [39]. The DNL is calculated using Eq. (3.5) where $V_{out,n}$ is the output voltage corresponding to codeword n and $V_{out,n+1}$ is the voltage output corresponding to one step up in the LSB.

$$DNL = \left(\frac{V_{out,n}}{V_{out,n+1}} - 1\right)(LSBs) \tag{3.5}$$

As shown in (3.5), the DNL is typically expressed in terms of the $LSBs$. The INL, also expressed in $LSBs$, is calculated as the cumulative sum of the DNL.

The measured output voltage of the fabricated DAC is shown in Fig. 3.7 (a) for input codewords in steps of 32, normalized to the FSR. The markers denote the measured values while the solid line denotes the ideal response. The calculated DNL and INL are shown in Fig. 3.7 (b) and (c), respectively. For these measurements, the DAC was placed under the same conditions as when it is used in the programming circuit. These conditions are $V_{dd} = 6.5V$, $+V_{ref} = 6V$, and $-V_{ref} = 4V$.

3.3.4 Output Buffer Pull-Down Transistor

Another important feature of the DAC that should be mentioned is its ability to output voltages close to ground through the addition of the pull-down switch, M_1, at the output of its buffer, as shown in Fig. 3.6 (b). This feature was necessary in order to provide near-ground inputs to the S/H array which could be sampled before raising $V_{dd,fg}$ to prevent the FG array from immediately programming. Recall that with our memory cell in programming mode, when $V_{dd,fg}$ is raised and $V_{cg} < V_{targ}$, programming automatically initiates. Sampling near-ground voltages on the S/H before raising $V_{dd,fg}$ ensures that $V_{cg} > V_{targ}$, preventing the FGs from programming until V_{targ} is

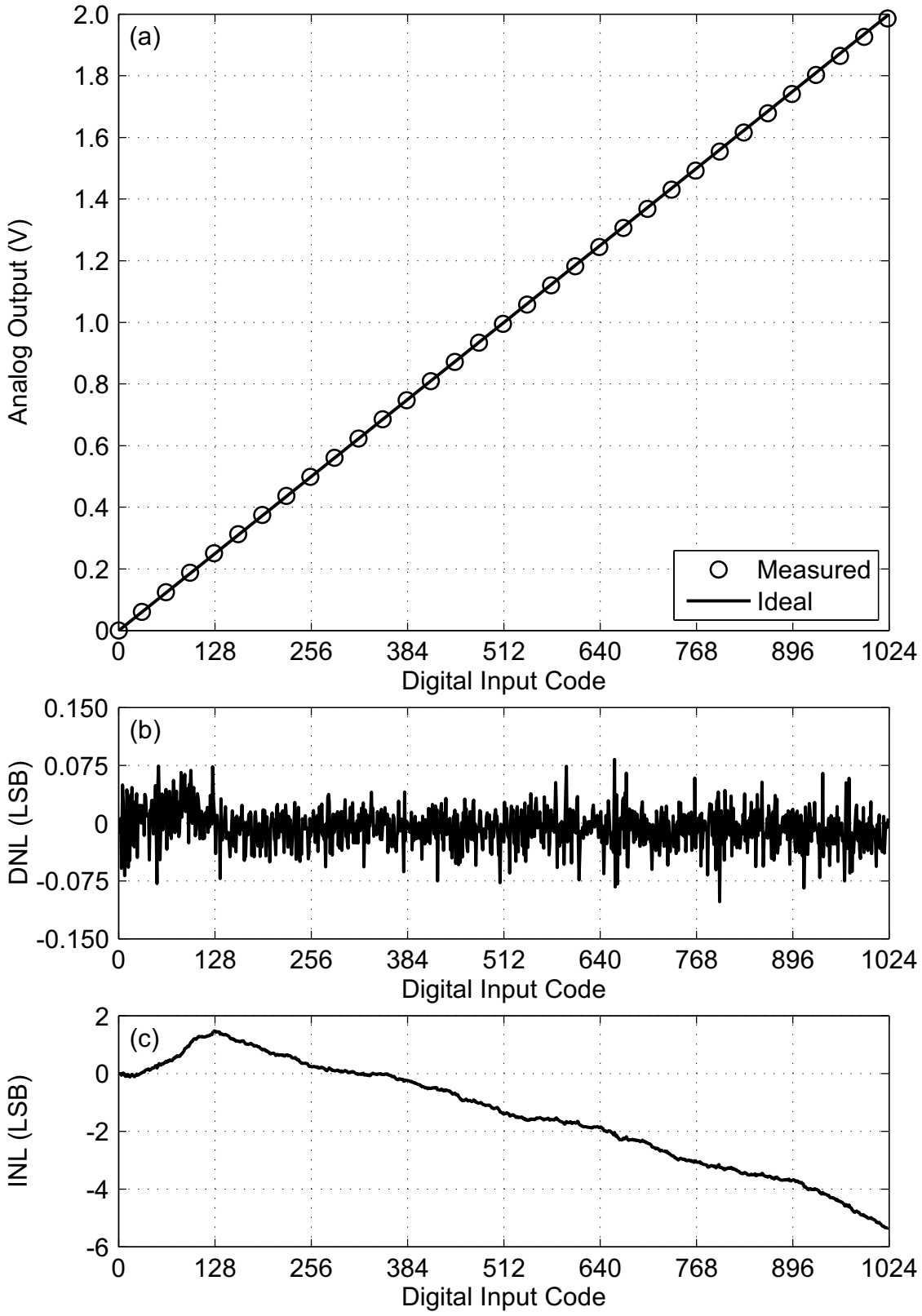

Figure 3.7: Static characteristics of the resistive DAC. (a) V_{out} vs DIN, normalized to the FSR. (b) DNL vs DIN. (c) INL vs DIN.

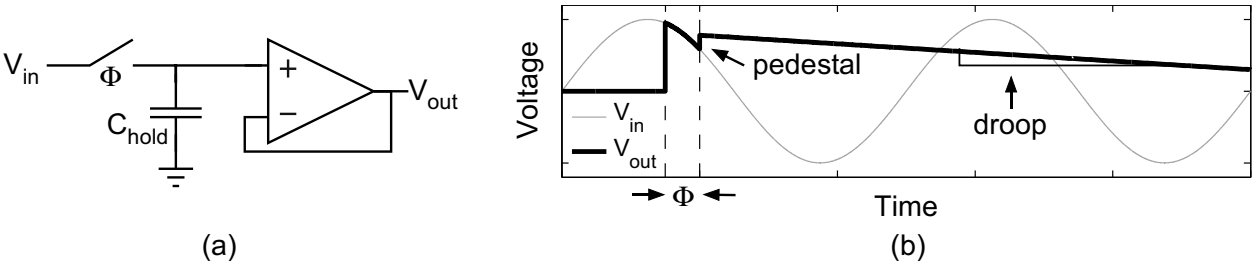

Figure 3.8: Basic S/H operation and associated errors. (a) A basic S/H circuit consisting of a switch, a capacitor, and a unity-gain buffer. (b) During sample-phase, Φ, the switch is closed and the S/H output tracks the input. When the switch is opened the S/H transitions to hold-mode. The pedestal and droop errors introduce uncertainty associated with the sampled value.

sampled on the S/H and V_{start} is pulsed. Figure 3.6 (c) shows the transistor-level schematic of the OTA used in the DAC buffer. It is a simple 5-transistor OTA topology with an added transistor, M_{b2}, to cascode the tail transistor, M_{b1}, to increase the CMRR since it is powered using $V_{dd,fg}$ during programming. This OTA is biased using the same current bias of the memory cell OTA, $I_b = 250nA$.

3.4 Sample-and-Hold Array

A sample-and-hold (S/H) is a circuit which samples its input at an instance in time and holds that value constant at its output until a new sample is gathered. A simple S/H implementation is shown in Fig. 3.8 (a) which consists of a switch, a capacitor, and a buffer. When clock Φ is high, the switch closes and C_{hold} is charged to V_{in}. At the moment the switch opens, $V_{in} = V'_{in}$ and the top plate of C_{hold} is floating so V_{out} will ideally equal V'_{in} until Φ closes the switch again.

S/Hs are ubiquitous in data conversion systems where they are used to hold the input to a system constant for the duration of the conversion. Similarly, our parallel programming procedure also requires a constant input, V_{targ}, for the length of time that an FG is being programmed. Thus, the S/H seemed a natural choice to implement this functionality. In our programmer, the S/H circuits are configured in an array and each one passes its output to the memory cell circuits that follow them. The S/H selection is accomplished through the column selection bits in the SPI, which allows the DAC word, S/H, and memory cell to be selected simultaneously.

3.4.1 Design Considerations

The role of the S/H in the programming system is to apply a constant target voltage to the input of the memory cell OTA. The S/H must accomplish this by sampling a voltage from the DAC

and maintaining that voltage as constantly as possible throughout the duration of programming. The first consideration that affects the ability of the S/H to perform this task is the droop error (Fig. 3.8 (b)). In a S/H, the droop error is defined as the rate at which the voltage on the hold node, V_{hold}, decreases through time. The droop rate is caused by charge leaking off of the hold node and is defined as:

$$r_{droop} = \frac{\Delta Q}{\Delta t} \cdot \frac{1}{C_{hold}} \tag{3.6}$$

where $\Delta Q/\Delta t$ is the leakage current and C_{hold} is the total capacitance on the hold node. Assuming the leakage current has been minimized, this expression implies that in order to minimize the droop rate the capacitance of the hold node must be maximized. However, this increases the footprint on the die and also increases aperture time, the time it takes for the S/H to charge C_{hold} to the same voltage as the V_{in}.

The second consideration that must be taken into account is the pedestal error (Fig. 3.8 (b)). A natural consequence of a S/H transitioning from sample mode to hold mode is the injection of charge on the hold node, causing a finite step in the output voltage at the moment the switch opens. This occurs through a process called charge injection, wherein the charge carriers in a MOSFET switch are expelled from the channel as the switch is turned off during the transition to hold mode.

Figure 3.9 illustrates how charge injection occurs when using an n-type MOSFET as a switch. When the switch is closed, the MOSFET is conducting, and it can be assumed that $V_{in} \approx V_{out}$. In this case, the charge in the channel of an NMOS switch is equal to

$$Q_{ch} = WLC_{ox}(V_{dd} - V_{in} - V_{TH}) \tag{3.7}$$

As the switch is turned off, charge Q_{ch} gets expelled from the source and drain terminals onto nodes V_{in} and V_{out} (Fig. 3.9 (a)). Assuming the charge exits equally on either side, the change in output voltage can be expressed as

$$\Delta V_{out} = \frac{WLC_{ox}(V_{dd} - V_{in} - V_{TH})}{2C_{hold}} \tag{3.8}$$

Rearranging (3.8) yields

$$\Delta V_{out} = V_{in}\left(1 + \frac{WLC_{ox}}{2C_{hold}}\right) - (V_{dd} - V_{TH})\left(\frac{WLC_{ox}}{2C_{hold}}\right) \tag{3.9}$$

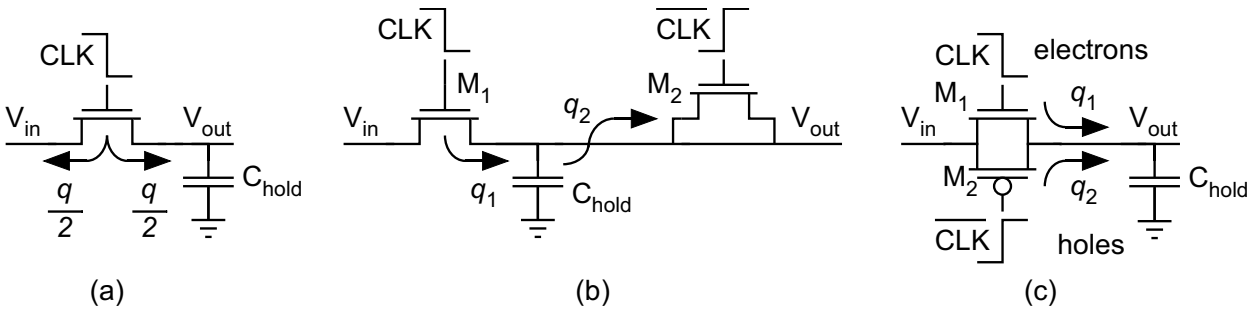

Figure 3.9: Origin and mitigation of charge injection errors in a S/H. (a) Charge injection from an nFET sampling switch. (b) Cancellation of nFET charge injection using a dummy switch. (c) Cancellation of charge injection using complementary MOS transistors.

As shown in (3.9), the charge injection onto the output node results in both a gain error of $(1 + WLC_{ox}/2C_{hold})$ and an offset error of $(WLC_{ox}/2C_{hold})(V_{dd} - V_{TH})$.

The above derivation relies on the assumption that V_{TH} is constant across all input levels, which we know from transistor theory is not the case, since the source and bulk do not remain at the same potential. If the body effect is taken into account, then V_{TH} is expressed as

$$V_{TH} = V_{TH0} + \gamma \left(\sqrt{2\Phi_F + V_{sb}} - \sqrt{2\Phi_F} \right) \tag{3.10}$$

where V_{sb} is the magnitude of the source-to-bulk potential, V_{TH0} is the threshold voltage when $V_{sb} = 0V$, Φ_F is the Fermi level in the substrate, and γ is the bulk-threshold parameter [40]. This expression indicates that there are non-linearities in (3.9) due to the body effect. These non-linearities must be minimized in order to maintain high resolution in the S/H. To minimize these non-linearities, the overall charge injection error must be mitigated.

There are four typical ways to reduce the pedestal error. (1) Minimize the channel size of the MOSFET switch, $(W \cdot L)$. This ensures that less charge, Q_{ch} (Eq. (3.7)), is conducting in the channel so when the switch is turned off less charge is transferred to the hold node. Equation (3.9) illustrates this effect, noting that ΔV_{out} is directly proportional to $(W \cdot L)$; thus, minimizing $(W \cdot L)$ reduces ΔV_{out}.

(2) Use a "dummy switch". Figure 3.9 (b) shows how a dummy switch is connected to the hold node in order to reduce the charge injection error. At the moment switch M_1 closes, M_2 is turned on so that q_2 enters its channel. Charges q_1 and q_2 can be expressed by

$$q_1 = \frac{W_1 L_1 C_{ox}}{2}(V_{CLK} - V_{in} - V_{TH1}), \qquad q_2 = \frac{W_2 L_2 C_{ox}}{2}(V_{CLK} - V_{in} - V_{TH2}) \tag{3.11}$$

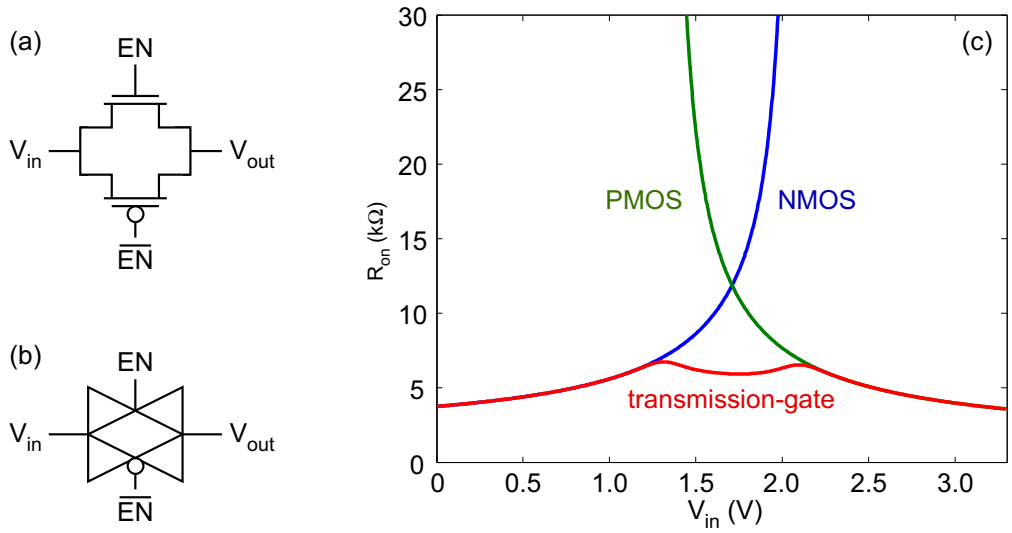

Figure 3.10: Overview of a transmission gate switch. (a) Transistor-level schematic of a transmission gate. (b) Circuit symbol of a transmission gate. (c) Simulated on-resistances of NMOS, PMOS, and transmission-gate switches with transistors sized according to Eq. (3.12). Note that for higher inputs, the on-resistance of an nFET increases as it enters cut-off and stops conducting, and *vice versa* for a pFET. When both operate in parallel they form a switch capable of passing rail-to-rail inputs.

where V_{CLK} is the voltage on the gate of the switch. Again, assuming half the charge of M_1 exits its channel, $q_1 = Q_{ch1}/2$, the transistor sizes can be chosen so that $L_1 = L_2$ and $W_1 = 2W_2$ in order to make $q_1 = q_2$. This cancels charge injection as long as the assumption holds true that half of Q_{ch1} exits onto the hold node. Unfortunately, that is not always an accurate assumption, so other steps must be taken to further reduce charge injection errors.

(3) Use transmission-gate switches (Fig. 3.9 (c)). A transmission gate is a type of switch made up of both an n-type and a p-type MOSFET operating in parallel, as shown in Fig. 3.10 (a). The circuit symbol for a transmission gate is given in Fig. 3.10 (b). This configuration has multiple advantages over a single-FET switch. A MOSFET switch operates in the above-threshold, linear region. In this operating region, an NFET switch is adept at passing low (near-ground) voltages due to the fact that, when enabled, its gate is connected to V_{dd} and its source and drain will ideally be at the same potential ($V_{in} = V_{out}$). The switch will conduct as long as $V_{gs} \geq V_{TH}$; however, for sufficiently high voltages, $V_{in} > V_{dd} - V_{TH}$, the nFET will enter cut-off where $V_{gs} < V_{TH}$, and the switch will no longer conduct. Likewise, a PFET is only able to pass voltages where $|V_{TH}| < V_{in} < V_{dd}$. However, when the two types of MOSFETs are placed in parallel, the resulting transmission gate is able to pass rail-to-rail voltages. Another emergent advantage is that the parallel connection of the two MOSFETs reduces the overall switch resistance as shown in Fig. 3.10 (c).

In regard to reducing charge injection error, the transmission gate is advantageous due to the fact that NFETs and PFETs conduct using opposite charge carriers—electrons and holes, respectively. If the MOSFETs are sized according to (3.12), then the two transistors will contribute equal but opposite charge onto the hold node, cancelling out any effects due to charge injection.

$$\frac{\left(\frac{W}{L}\right)_n}{\left(\frac{W}{L}\right)_p} = \frac{\mu_p}{\mu_n} \tag{3.12}$$

where μ_n and μ_p are the mobility of electrons and holes, respectively. Generally, it is assumed that the mobility of electrons is roughly three-times that of holes, so the PFET width is drawn to be $3\times$ the width of the NFET and both transistors' lengths are minimized in order to satisfy condition (1).

Lastly, method (4) is to increase the size of the capacitance on the hold node which decreases the magnitude of the pedestal error. According to (3.9), ΔV_{out} is inversely proportional to C_{hold}, so increasing C_{hold} reduces the pedestal error.

3.4.2 S/H Topology

To fulfill these design requirements, a S/H topology based on [41] was chosen. This S/H employs Miller feedback in its hold-mode configuration to increase the effective hold capacitance, C_{hold}, without requiring larger drawn capacitors. A simplified version of the S/H schematic is shown in Fig. 3.11 (a). In the fabricated circuit, the two switches, S_1 and S_2, are comprised of transmission gates with half-sized dummy transmission-gate switches on each node except for V_{in}, since this charge injection error gets absorbed by the input source and does not affect V_{out}. Also, note that switch S_1 is clocked using Φ_{1d}, a delayed version of Φ_1. This opens S_2 slightly before S_1 when transitioning to hold mode, further reducing charge injection [42].

Figures 3.11 (b) and (c) show the S/H in its sample- and hold-mode configurations, respectively. In sample mode, the S/H OTA is connected as a unity-gain buffer, forcing $V_A = V_{ref}$ as C_1 and C_2 are charged to V_{in}. In hold mode, S_1 and S_2 are opened, leaving V_{out} floating. In this configuration, Miller feedback from V_B to V_A through C_1 and C_2 forces the capacitance on the hold node $C_{hold} \approx C_2(1 + A)$, where A is the open-loop gain of the S/H OTA, G_{m1}. This effect will be derived in the next section. Figures 3.11 (d) and (e) show the transistor-level schematics for OTAs G_{m1} and G_{m2}, respectively. Note that just as the programmer OTA employed a cascoded tail, so does the buffer OTA in the S/H, since it also operates using the elevated supply rail $V_{dd,fg}$

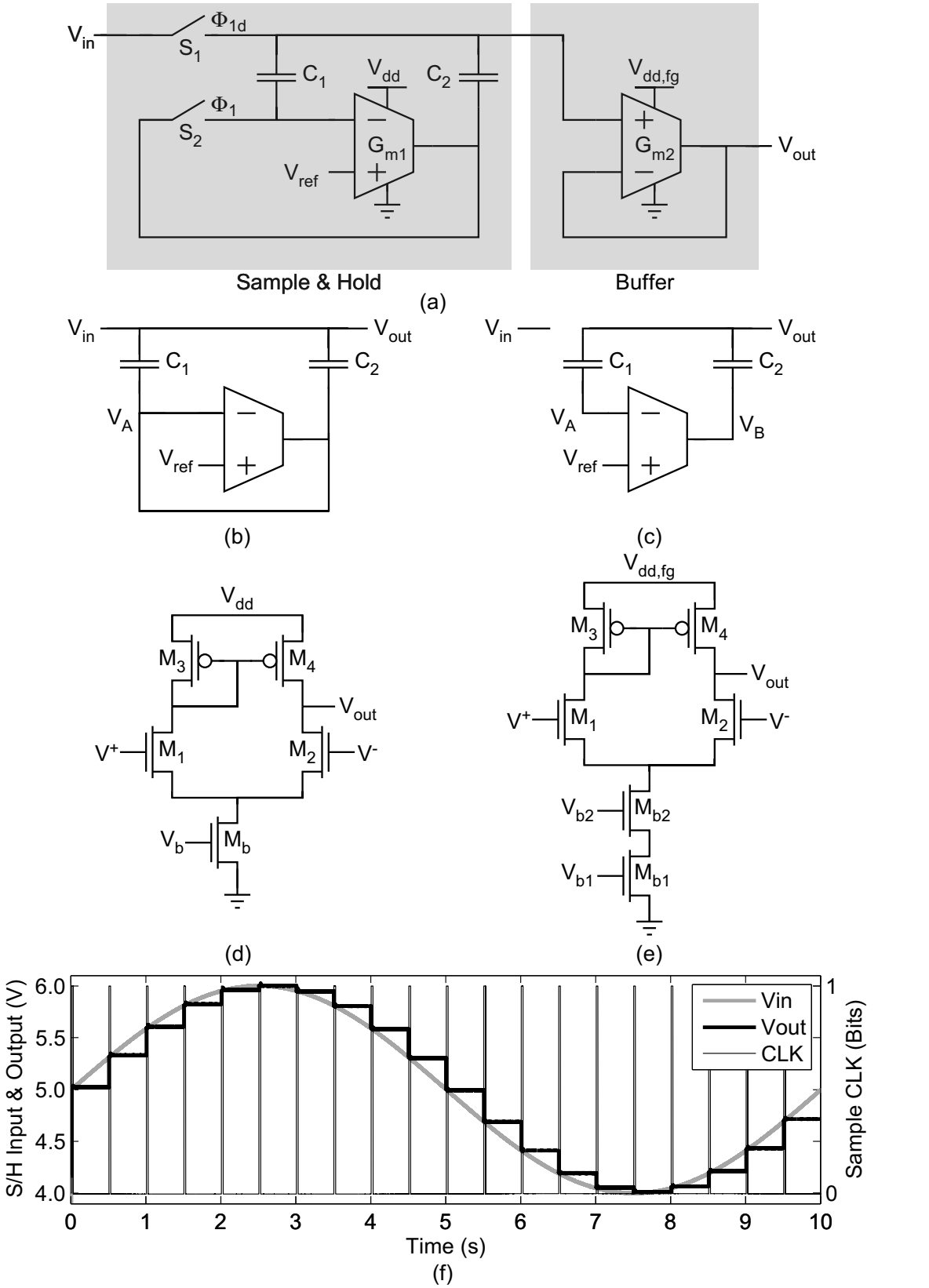

Figure 3.11: Overview of sample-and-hold with Miller hold capacitance. (a) Simplified schematic diagram of S/H. Both switches operate on the same clock phase, Φ_1; switches are closed during the sample period and are opened during the hold period. (b) S/H in sample-mode configuration. (c) S/H in hold-mode configuration. (d) Transistor-level schematic of S/H OTA, G_{m1}. (e) Transistor-level schematic of buffer OTA, G_{m2}. (f) Transient response of S/H sampling a sinusoidal input.

during programming. Interestingly enough, the OTA in the S/H does *not* require being powered from the high voltage rail since the hold node is isolated from the OTA through C_1 and C_2; thus, G_{m1} is left connected to V_{dd}. G_{m1} could have been connected to $V_{dd,fg}$ without issue, but using a lower rail allows for reduced power consumption of the array during programming. Figure 3.11 (f) demonstrates the S/H's operation by showing a transient plot of the S/H sampling a sine wave. The time scale is large because the transconductors are biased in the sub-threshold region in order to yield higher gain and lower power consumption. This sub-threshold operation severely increases the slew rate and, thus, acquisition time of the S/H, requiring the transient input signals to be very low frequency. However, this does not affect the operation of the S/H in our programming scheme due to the fact that it is only used to sample DC voltages from the DAC.

This topology was chosen for a number of reasons. Firstly, since programming takes a finite amount of time, it was important to have a S/H that had very little droop error, ensuring V_{targ} remained constant over the entire programming period. As discussed in the previous section, to reduce the droop rate of a S/H, the hold capacitance must be maximized. Likewise, the pedestal error ΔV_{out} is inversely proportional to C_{hold}. Thus, this topology is useful for reducing both errors that were of importance since it employs Miller feedback to achieve a higher effective C_{hold}, while also reducing die area.

3.4.3 Miller's Theorem

The main reason this S/H was chosen is due to its increased hold-mode capacitance. Therefore, it is of interest to discuss this effect in detail. Recall from electronics theory, Miller's theorem, which presents a means of generating equivalent circuits as an analytical tool to simplify circuit analysis and to gain insight into how a feedback network affects a circuit's operation. Using Miller's theorem, a circuit of the configuration shown in Fig. 3.12 (a) can be rearranged to obtain the equivalent circuit shown in Fig. 3.12 (b). The equivalent impedances have values $Z_1 = \frac{Z}{1+A}$ and $Z_2 = \frac{Z}{1+1/A}$ where $A = \frac{V_{out}}{V_{in}}$ is the gain of the inverting amplifier. The series impedance, Z, of Fig. 3.12 (a) can be separated into two equivalent impedances connected in parallel to the input and output, and whose magnitudes are dependent upon the gain, A, of the inverting amplifier. This simplifies circuit analysis by removing the feedback network and replacing it with impedances to ground, as shown in Fig. 3.12 (b). Thus, it can be noted that, since the complex impedance of capacitance is equal to $1/sC$, capacitances are amplified at the input, Z_1, and attenuated at the output, Z_2.

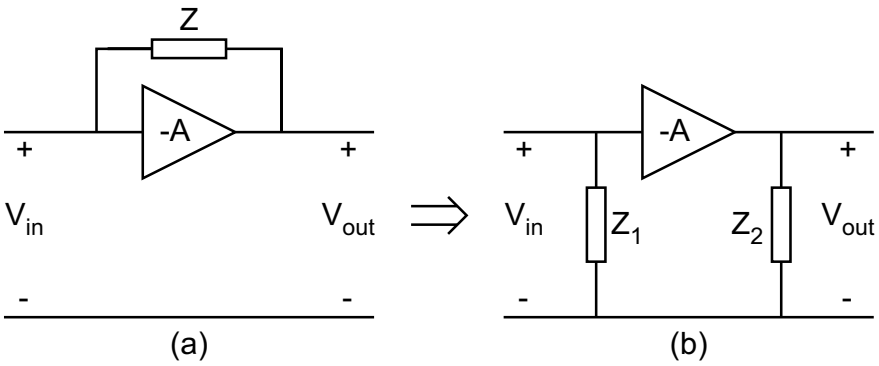

Figure 3.12: Finding an analytical equivalent circuit using Miller's theorem. (a) An inverting amplifier that has an impedance connected between its input and output. (b) The Miller equivalent circuit where $\mathbf{Z_1} = \frac{\mathbf{Z}}{\mathbf{1+A}}$ and $\mathbf{Z_2} = \frac{\mathbf{Z}}{\mathbf{1+1/A}}$, where A is the open loop-gain of the inverting amplifier.

A proof for Miller's theorem as outlined in [43] is as follows: Assuming no current enters the inverting amplifier, all current flowing from V_{in} to V_{out} passes through the impedance Z. For the two circuits to be equivalent, the current passing through Z must be the same as the current through Z_1.

$$I_{in} = \frac{V_{in} - V_{out}}{Z} = \frac{V_{in}}{Z_1} \tag{3.13}$$

$$Z_1 = \frac{Z}{1 - \frac{V_{out}}{V_{in}}} \tag{3.14}$$

Likewise,

$$Z_2 = \frac{Z}{1 - \frac{V_{in}}{V_{out}}} \tag{3.15}$$

This same effect can be applied to the S/H circuit in hold mode (Fig. 3.11 (c)). During the sample mode, the total capacitance that needs to be charged can be easily derived as the parallel combination of C_1 and C_2, which is equal to $C_1 + C_2$ (Fig. 3.11 (b)). However, when the circuit is in its hold mode, Miller feedback from V_B to V_A increases the capacitance of the hold node to $\approx C_2(1 + A)$, where A is the open-loop gain of the OTA. Since capacitance on the node of interest is not the input or output node of the amplifier, the derivation for C_{hold} cannot be performed by directly applying the Miller theorem as shown above. Instead, the derivation of this equivalent capacitance must be obtained by analyzing the small-signal output impedance of the S/H in its hold-mode configuration. Figure 3.13 shows the small-signal model of the S/H in hold mode, where V_{ref} is an AC ground and C_{1B} and C_{2B} are the bottom-plate parasitic capacitances of C_1 and

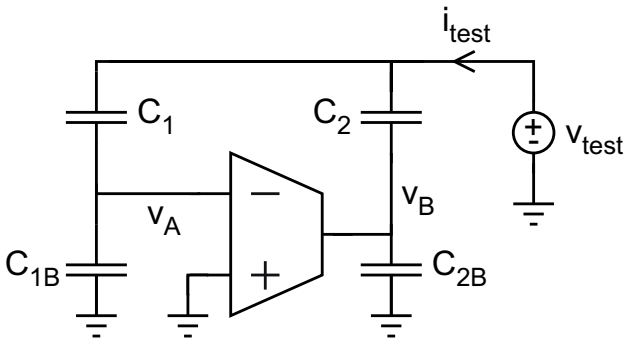

Figure 3.13: Small-signal model of S/H in hold-mode. C_{1B} and C_{2B} represent the bottom-plate parasitic capacitance of C_1 and C_2, respectively. A test voltage and current are applied to the output node in order to find the effective output impedance.

C_2, respectively. If a test voltage is applied to the output node and the resulting output current is measured, then C_{hold} can be found by solving for $Z_{hold} = V_{test}/I_{test}$. First, apply KCL at the output node to get

$$i_{test} = sC_1(v_{test} - v_A) + sC_2(v_{test} - v_B) \tag{3.16}$$

$$= s(C_1 + C_2)v_{test} - sC_1v_A - sC_2v_B \tag{3.17}$$

Next, v_A and v_B can be related using the open-loop relationship of the OTA. At low frequencies $V_{out} = A(V^+ - V^-)$ and V^+ is connected to ground, giving

$$v_B = -Av_A \tag{3.18}$$

Substituting (3.18) into (3.17) yields

$$i_{test} = s(C_1 + C_2)v_{test} - sC_1v_A + sC_2Av_A \tag{3.19}$$

$$= s(C_1 + C_2)v_{test} + s(AC_2 - C_1)v_A \tag{3.20}$$

Note that v_{test} and v_A can be related by the capacitive divider formed by C_1 and its parasitic component C_{1B} as follows

$$v_A = v_{test}\left(\frac{C_1}{C_1 + C_{1B}}\right) \tag{3.21}$$

Equation (3.21) can then be substituted into (3.20) to obtain

$$\begin{aligned} i_{test} =& s(C_1 + C_2)v_{test} + \frac{C_1}{C_1 + C_{1B}}(sAC_2 - sC_1)v_{test} && (3.22)\\ =& \left[s(C_1 + C_2) + \frac{C_1}{C_1 + C_{1B}}(sAC_2 - sC_1)\right] v_{test} && (3.23)\\ \frac{v_{test}}{i_{test}} =& \frac{1}{s(C_1 + C_2) + s(AC_2 - C_1)\frac{C_1}{C_1 + C_{1B}}} && (3.24)\\ =& \frac{1}{s\left[C_1 + C_2 + (AC_2 - C_1)\frac{C_1}{C_1 + C_{1B}}\right]} && (3.25) \end{aligned}$$

Equation (3.25) represents Z_{hold}, thus we can infer that C_{hold}

$$\begin{aligned} C_{hold} =& C_1 + C_2 + (AC_2 - C_1)\frac{C_1}{C_1 + C_{1B}} && (3.26)\\ =& \frac{(C_1 + C_2)(C_1 + C_{1B}) + C_1(AC_2 - C_1)}{C_1 + C_{1B}} && (3.27)\\ =& \frac{C_1^2 + C_1C_2 + C_1C_{1B} + C_2C_{1B} + AC_1C_2 - C_1^2}{C_1 + C_{1B}} && (3.28)\\ =& \frac{C_1C_2(1 + A) + C_{1B}(C_1 + C_2)}{C1 + C_{1B}} && (3.29) \end{aligned}$$

Since $C_{1B} \ll C_1, C_2$ (3.29) further reduces to

$$C_{hold} \approx C_2(1 + A) \tag{3.30}$$

The S/H used in this work was designed using $C_1 = C_2 = 1pF$, corresponding to a sample-mode capacitance of $2pF$. The 5-transistor OTA used in this work is biased using $V_b = 400mV$ which, according to simulation, yields an open-loop gain of ~ 70. Thus, the Miller feedback through capacitors C_1 and C_2 acts upon the circuit to create an effective hold-mode capacitance that is $\sim 35\times$ that of the sample-mode capacitance.

3.4.4 Measured Performance

When the prototype was received from fabrication, an undesirable characteristic of the S/H was discovered that was not indicated in simulations. During initial testing, an unusually high droop rate was measured. Through troubleshooting, it was determined that the droop was likely caused by reverse-biased pn-junction leakage through the pFET switches on the V^- terminal of the OTA, caused by the high well-to-source voltage of the MOSFET switches since the wells were connected to $V_{dd,fg} = 6.5V$. This effect was likely exacerbated by the larger switch area from the

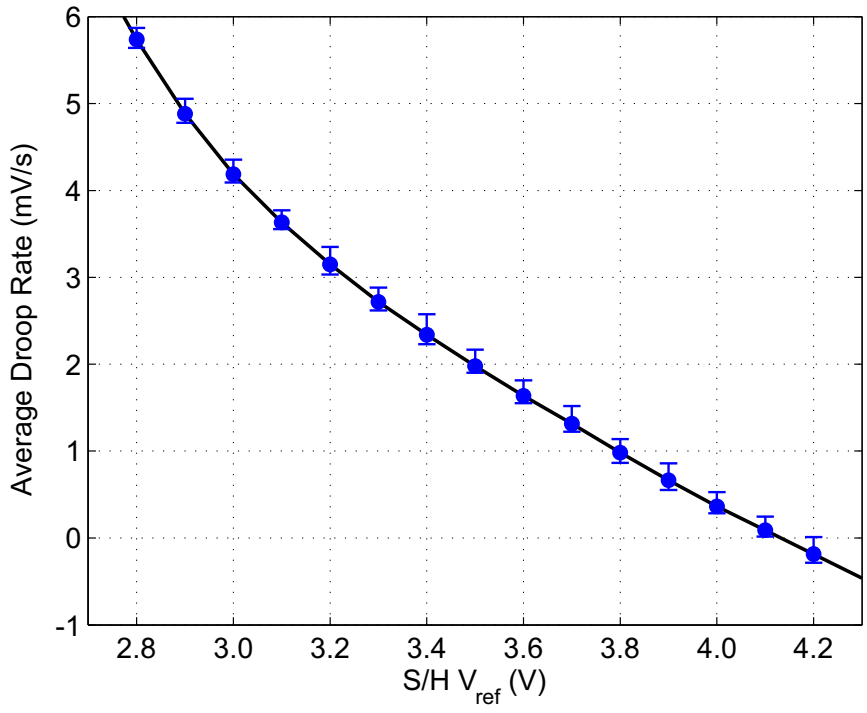

Figure 3.14: Sample-and-hold droop rate dependence on V_{ref}. The plot shows the average droop rate for each value of V_{ref} while sampling voltages between 4V and 6V. The error bars indicate the range of measured values, denoting the dependence on V_{in}. The droop error is nearly eliminated when V_{ref} is equal to 4.1V.

inclusion of dummy switches and a 2× scaling of the main switch size to accommodate the dummy switches. This effect can be mitigated by raising the value of V_{ref} in Fig. 3.11 (a) which reduces the well-to-source potential to lower the leakage. The relationship between V_{ref} and droop rate was measured for $V_{dd} = 4.5V$ and $V_{dd,fg} = 6.5V$ and is shown if Fig. 3.15.

As shown in Fig. 3.15, an inverse relationship exists between the droop rate and V_{ref}. These measurements were taken for values of V_{in} spanning the FSR of the DAC output ($4V$ to $6V$). Also, note that the droop rate is ostensibly eliminated for $V_{ref} = 4.1V$, thus this was chosen to be the value of V_{ref} used throughout the rest of this work. Note that this also requires raising the low power supply, V_{dd}, to $4.5V$ during programming to accommodate the increased value of V_{ref}. The droop rate also depends on the value of V_{in} being sampled which is indicated in Fig. 3.15 by the error bars. Larger values of V_{in} result in more significant droop rates while lower droop rates are had for lower V_{in}. This dependency on V_{in} was measured for $V_{ref} = 4.1V$ and is shown in Fig. 3.15 (a). The pedestal error was also measured for $V_{ref} = 4.1V$ and is shown in Fig. 3.15 (b). The solid line in Fig. 3.15 (b) indicates the overall average pedestal, which was measured to be $3.405mV$. This average pedestal results in a constant offset at the output, which does not contribute to error

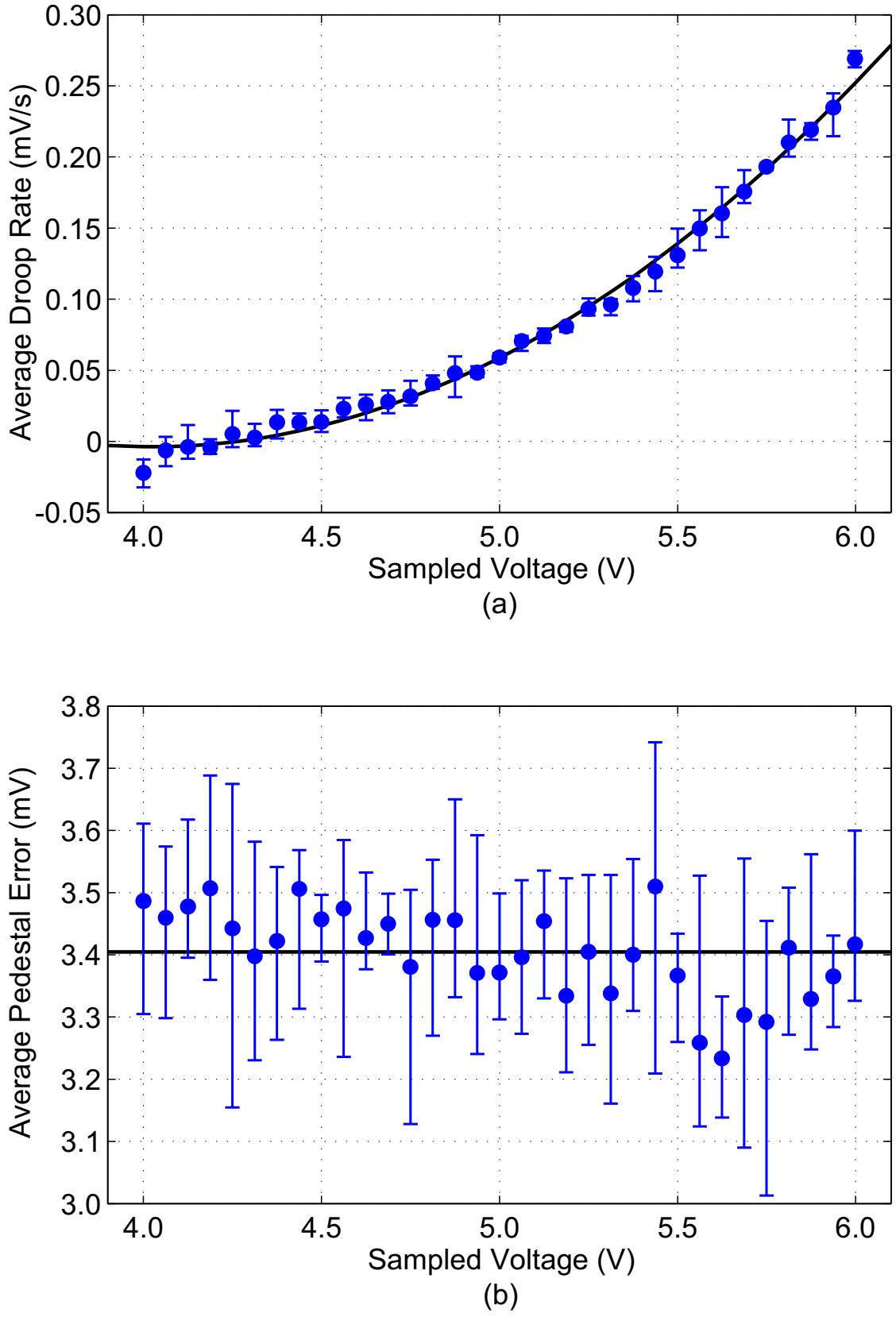

Figure 3.15: Dependence of S/H droop and pedestal errors on V_{in}, while V_{ref} is fixed at 4.1V. (a) S/H droop rate vs V_{in} measured out of five trials per input. (b) S/H pedestal error vs V_{in} measure out of five trials per input. The error bars indicate the range of measured values for each input. The solid line indicates the overall average pedestal, equal to 3.405mV.

since it is independent of the input.

The maximum measured droop rate is $275\mu V/s$; however, we can program the maximum V_{targ} in under 6 seconds. The maximum measured pedestal error is $392\mu V$, taken with respect to the overall average pedestal. Thus, the overall the S/H has a resolution given by

$$\text{ENOB} = \log_2\left(\frac{\text{FSR}}{\text{error}}\right) = \log_2\left(\frac{2V}{6(275\mu V) + 392\mu V}\right) = 9.94\text{-bits} \tag{3.31}$$

which still exceeds the resolution to which we can program our memory cell.

3.5 Programming Methodology

Now that each block has been presented, it is important to discuss how they operate together to perform the programming of an array. Figure 3.16 shows the timing diagram used to program a row of FGs in parallel. For brevity, the SPI bits have been clustered into functional groups comprised of the 10-bit DAC word, 3-bit column address, 1-bit row address, 1-bit DAC pulldown enable, 1-bit voltage output mode enable, and 1-bit circuit connection enable. Bit 3 of the SPI (V_{targ} pin select) has been ignored here since it is reserved for testing purposes and is not used in regular programming cycles.

Before programming, V_{dd} and $V_{dd,fg}$ are raised to $4.5V$, as discussed in the previous section. Then during period A, bit 10 of the SPI (DAC pulldown) is enabled which forces the output of the DAC to equal $\sim 0V$. Each S/H is sequentially selected using the column address bits and $\sim 0V$ is sampled onto each of them. The input codeword to the DAC does not affect this procedure so its value is irrelevant, denoted with an X.

In the next period (B), $V_{dd,fg}$ is raised to its elevated programming level equal to $6.5V$. Voltage output (VO) enable is HIGH and circuit connection (CIRC) enable is LOW, connecting each FG in row r to its corresponding programmer. Each column is then sequentially selected using the COL address bits; simultaneously, the DAC input word for the selected FG, C_n, is applied to the DAC. After these data bits are latched into the SRAM, V_{out} from the DAC is sampled onto the selected S/H, setting V_{targ} for the selected programmer. Finally, the START pin is pulsed HIGH to inject current into the channel of the FG and start the programming circuit. This process is repeated for each column in the row. Once the last programmer has been started, $V_{dd,fg}$ is left high until the DONE circuit (not pictured) indicates that all FGs have reached their targets. Once programming

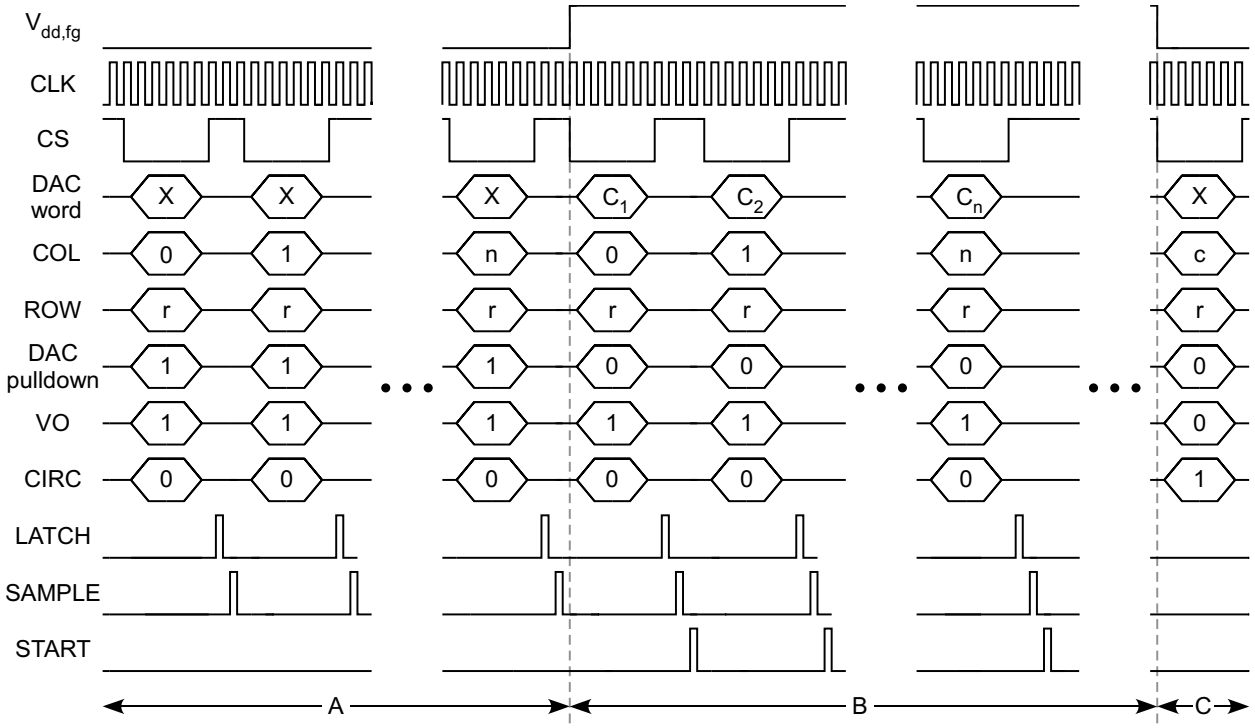

Figure 3.16: Timing diagram of programming an array of n floating-gates in parallel.

is finished, V_{dd} and $V_{dd,fg}$ are lowered to the run-time level ($3.3V$) and the SPI is used to set VO LOW and CIRC HIGH, disconnecting the FGs from their programmers and connecting them to their circuits (time interval C). Note that the selected row, r, remains constant throughout this entire process. This indicates that this procedure must be repeated for each row that is required to be programmed. Also, note that the chip select (CS) signal is low whenever the shift register is in use, otherwise it remains high.

3.6 Chapter Summary

A new method for parallel programming analog floating-gate memory arrays was presented. The circuit uses an SPI to digitally interface with the rest of the chip which is comprised of a DAC, an array of sample-and-holds, and an array of FG memory cells. The DAC is a 10-bit resistive divider with a pFET switch-tree decoder. The S/H employs Miller feedback in its hold-mode configuration to increase its effective hold-mode capacitance by approximately an order of magnitude. This topology was chosen because it was of importance to reduce the droop rate and pedestal errors, both of which are inversely proportional to C_{hold}. The programming circuit is configured in an array where each column is composed of a S/H, FG memory cell, and a circuit which is biased using the FG.

To program an array in parallel, the S/Hs must first be "cleared" of any high values stored on them, to prevent FGs from programming during the next step. Then, the floating-gate supply voltage, $V_{dd,fg}$, is raised to its programming level and the FG memory cells are configured in programming mode. Then, sequentially, each column is selected along with a DAC voltage, the S/H clocked to set V_{targ}, and V_{start} is pulsed to begin programming. A DONE circuit monitors V_{cg} on each memory cell and indicates when all FGs in the row have completed programming. Then, $V_{dd,fg}$ is lowered to its run-time level and the FGs are connected to their circuits for biasing.

Chapter 4

A Parallel-Programmable Bandpass Filter Array

A proof-of-concept programmable filter array employing our parallel programmer was fabricated in a $0.5\mu m$ standard CMOS process available through MOSIS. The chip contains 8 sample-and-holds, 8 programmer OTAs, 16 floating-gate transistors, and 8 bandpass filters as well as the SPI, DAC, and miscellaneous peripheral circuitry. Each bandpass filter requires two FGs for biasing, one for the low corner frequency and one for the high corner frequency. The FGs are distributed in an array of 2 rows and 8 columns. In this configuration, the chip allows for one row of FGs to be programmed in parallel. Thus, two programming sequences, one for each row, are required to program the full chip. This arrangement was chosen to minimize the number of programmers required so that the active area could be reduced.

A die photograph of the chip is shown in Fig. 4.1, and the approximate active area of each block is given in Table 4.1. The area inside the pad frame measures approximately $1.4mm^2$. The DAC is the largest component, consuming roughly 50% of the active area; the C^4s are second with 12%, then S/Hs with 11%, FGs with 9.5%, programmers with 9%, and the SPI with 6%.

Table 4.1: Active Area per Block

Block	**Area (μm^2)**
SPI	57,000
DAC	450,000
S/H Array	100,000
Programmer Array	80,000
FG Array	85,000
C^4 Array	110,000
Misc. Peripheral	10,000

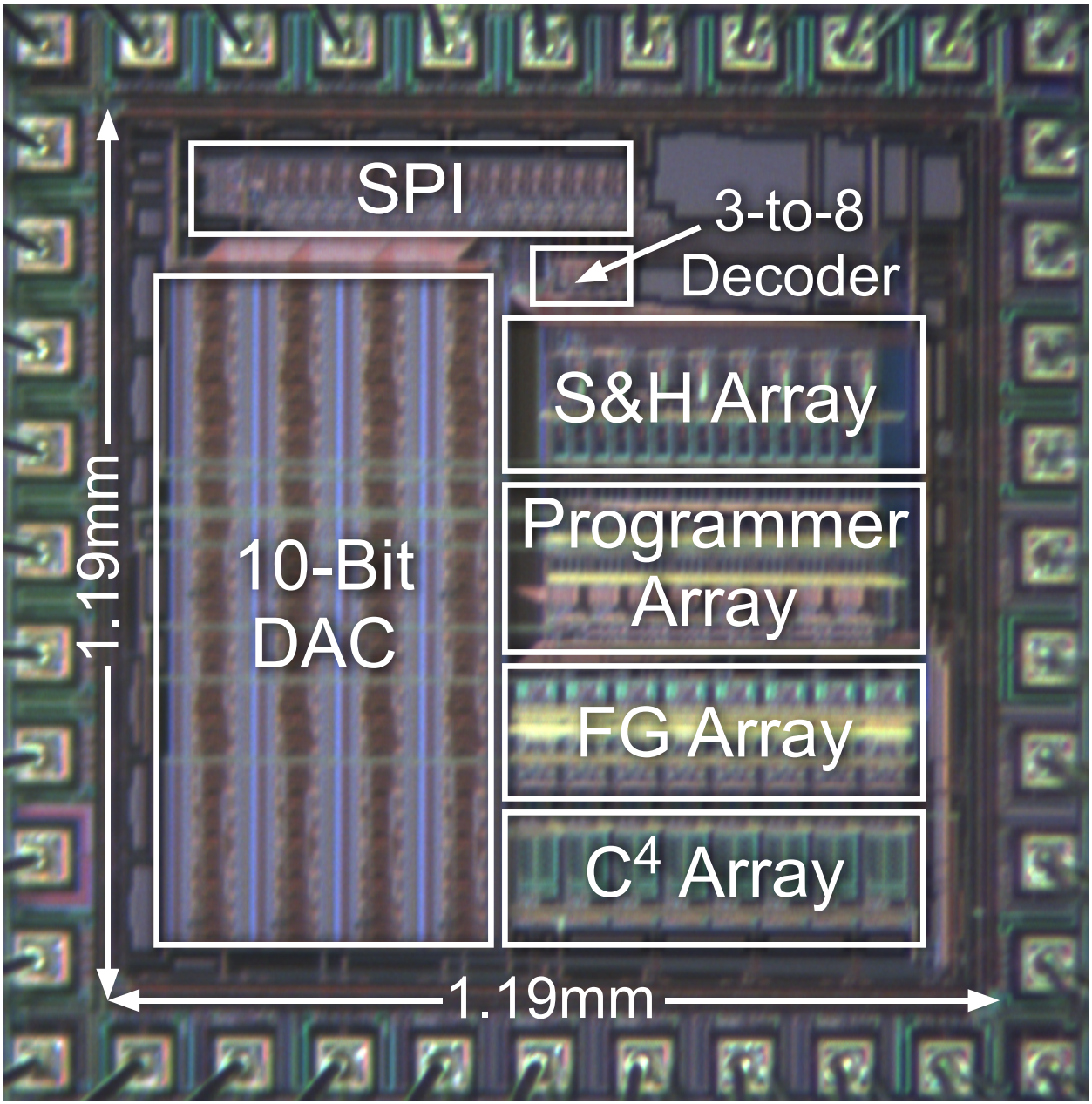

Figure 4.1: Die photograph of the programmable bandpass array chip.

Miscellaneous peripheral circuitry including logic level-shifters, current bias circuits, a 3×8 column address decoder, multiplexers and other switches also consumes 1% of the active area. Not included in active area is the inter-stage routing which consumes roughly $200,000 \mu m^2$ throughout the entire die.

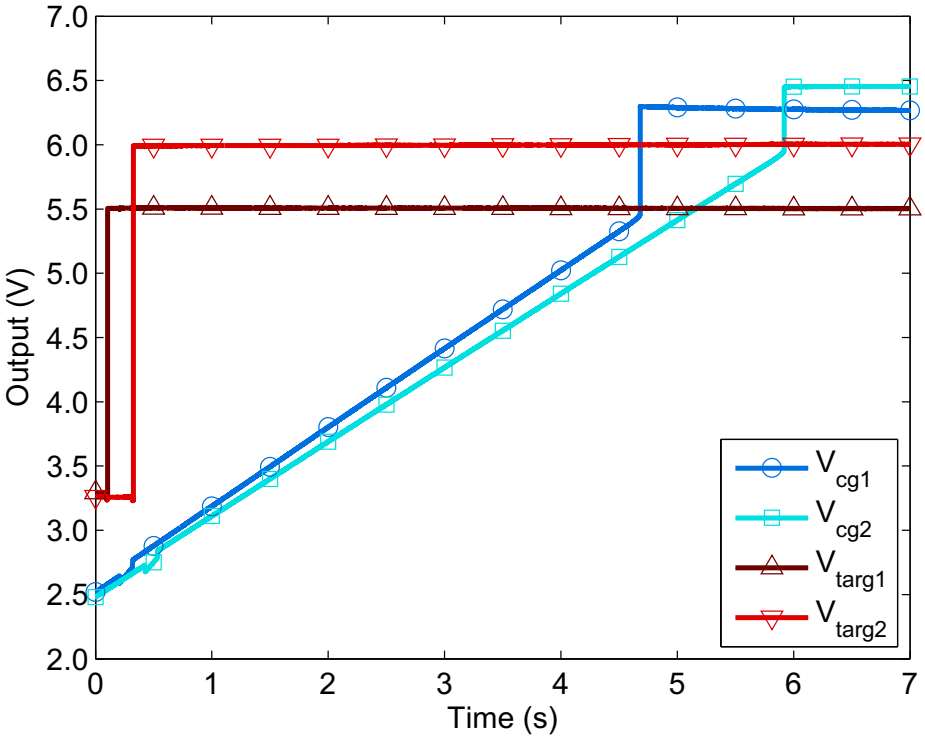

Figure 4.2: Transient response of node V_{cg} on two FG memory cells being programmed in parallel using our parallel programmer.

4.1 Parallel Programmer Accuracy

Before using the FGs in their bandpass circuits, it was of interest to characterize the programming accuracy achieved using our parallel programmer employing the DAC and S/H array. The circuit parameters used for programming the FGs are the same as those used to characterize the standalone memory cell presented in Chapter 3 ($I_b = 250nA$, $I_1 = 100nA$, $I_2 = 2nA$, and $V_{dd,fg} = 6.5V$). In addition to raising $V_{dd,fg}$, the low power rail, V_{dd}, also had to be raised to $4.5V$ in order to allow the S/H OTA enough headroom to operate with $V_{ref} = 4.1V$ to achieve a low droop rate. Again, the programmed value was measured using the voltage-output configuration of Fig. 3.2 (a), and the output was measured from node V_{cg}.

Figure 4.2 shows two FGs being programmed in parallel using our programmer. At $100ms$, the first S/H is clocked, sampling the DAC output to set V_{targ1}. Shortly after, V_{start} is pulsed to start programming. Then, the next column is selected, and the process is repeated. The input codewords used were 768 and 1023 to yield $V_{targ1} = 5.5v$ and $V_{targ2} = 6V$, respectively. Due to the limited number of pins, only two S/H outputs and two control gate outputs were hard-wired to the pad frame, thus only two FGs can be demonstrated in parallel. The rest are multiplexed to an output pin using the row and column address bits; however, the same process applies for the

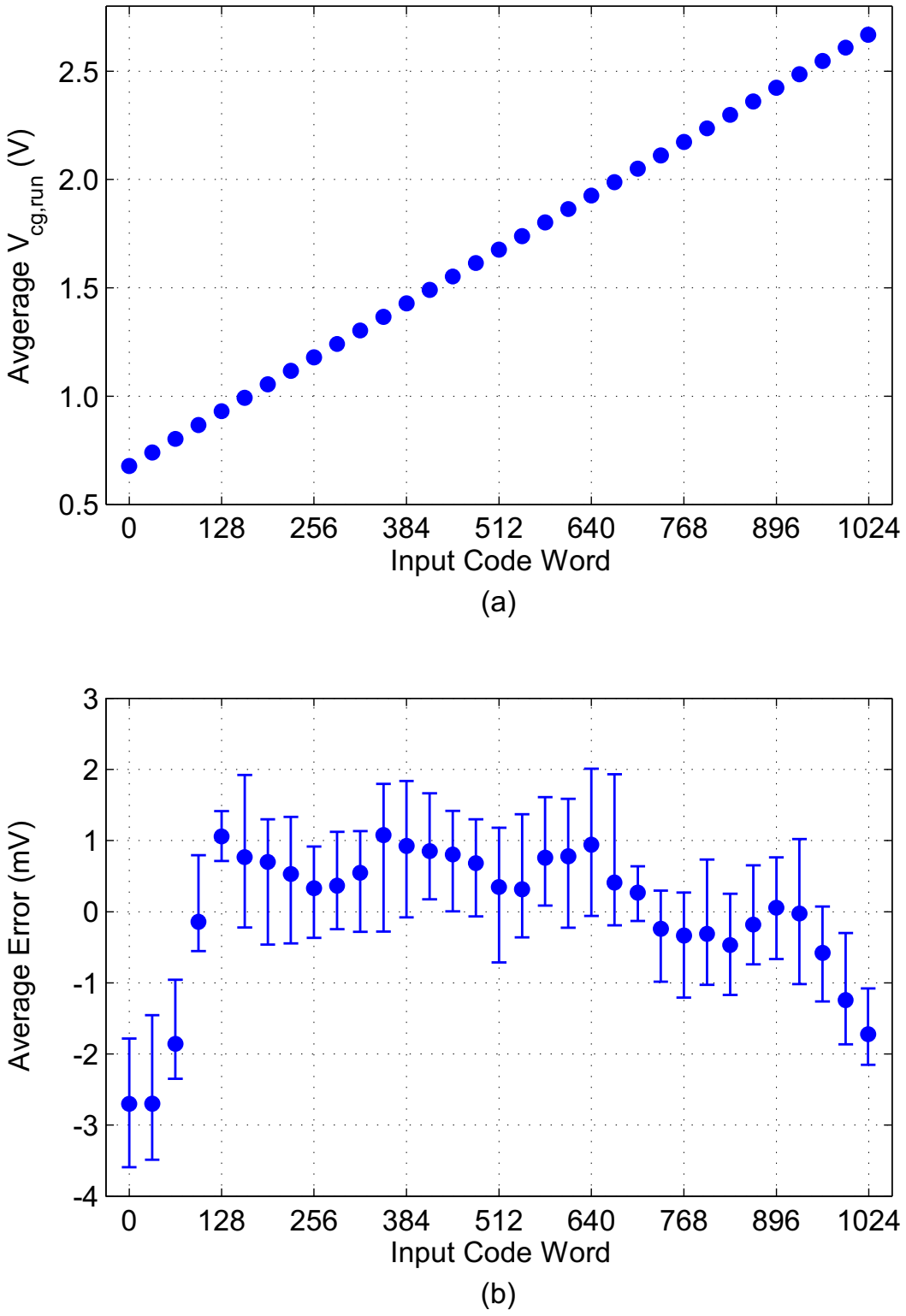

Figure 4.3: Accuracy of programming circuit out of 25 trials. (a) Run-time V_{cg} values vs digital input codeword sent to the DAC. (b) Average deviation from linear, where the error bars indicate the range of measured values.

remaining 6 FGs in parallel.

DAC words from 0 to 1023 in increments of 32 were tested, and each word was programmed across all FGs in the chip 25 times. The average $V_{cg,run}$ of each word was computed, which is shown in Fig. 4.3 (a). Just as with the standalone memory cell, a line of best-fit was drawn through this data set and the average deviation from this line was computed to determine the programming error (Fig. 4.3 (b)). The error bars represent the range of deviations from linear. The difference maximum measured programming error is equal to $3.6mV$, yielding a resolution equal to

$$\text{ENOB} = \log_2\left(\frac{\text{FSR}}{\text{error}}\right) = \log_2\left(\frac{2V}{3.6mV}\right) = 9.12\text{-bits} \tag{4.1}$$

which corresponds to a loss of only 0.51-bits from the standalone programming accuracy. Additionally, our parallel programmer adds -0.2% increase in gain error, resulting in an overall gain error of -0.4% of the FSR.

4.2 The C^4 Bandpass Filter

To demonstrate the programmer's ability to directly tune circuit parameters, we will use the capacitively-coupled current conveyor (C^4) shown in Fig. 4.4 (a). The C^4 is part of a class of transconductance-capacitance (G_m-C) filters whose corner frequencies are proportional to transconductance and capacitance, as apparent from (4.3). The transfer function of the C^4 is given by

$$\frac{V_{out}}{V_{in}} = -\frac{C_1}{C_2}\frac{s\tau_l(1-s\tau_f)}{1+s\left(\tau_l+\tau_f\left(\frac{C_O}{C_2}-1\right)\right)+s^2\tau_h\tau_l} \tag{4.2}$$

where $C_T = C_1 + C_2 + C_W$ and $C_O = C_2 + C_L$ and

$$\tau_l = \frac{C_2}{G_{m,L}} \qquad \tau_h = \frac{C_O C_T - C_2^2}{C_2 G_{m,H}} \qquad \tau_f = \frac{C_2}{G_{m,H}} \tag{4.3}$$

such that τ_l is the time constant of the low corner frequency, and τ_h is the time constant of the high corner frequency. The capacitors are sized so that the zero, τ_f, is designed to be at a sufficiently high frequency that its effect can be ignored.

The C^4's low and high corner frequencies are proportional to transconductances $C_{m,L}$ and $G_{m,H}$ in Fig. 4.4 (a), respectively. Since these transconductances are directly proportional to the bias currents of each OTA, the corner frequencies can be directly tuned using the FG memory cell as

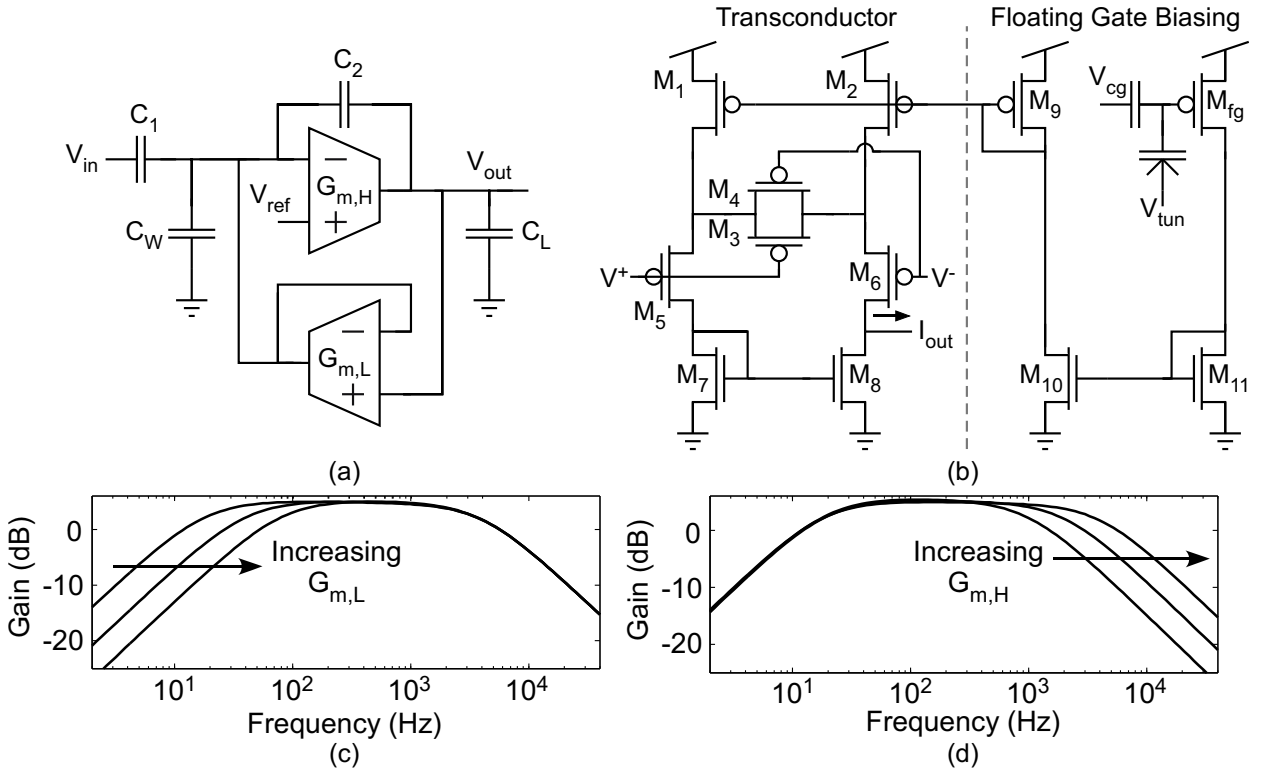

Figure 4.4: Overview of the OTA-based C^4 bandpass filter. (a) Schematic of OTA-based C^4. (b) Schematic of bump-linearized OTA used in C^4. (c) Independent tuning of f_L. (d) Independent tuning of f_H. — Figures (c) and (d) were measured using a 0.35μm CMOS process as presented in [44], but are used here qualitatively.

a current reference to bias them, as shown in Fig. 4.4 (b). The gain and quality-factor are both indirectly controlled by the ratio of these transconductances $G_{m,H}/G_{m,H}$ according to

$$|A_v| = \frac{C_1}{C_2}\frac{1}{1+\frac{C_L}{C_2}\frac{G_{m,L}}{G_{m,H}}} \qquad Q = \frac{\sqrt{C_O C_T - C_2^2}}{C_L\sqrt{\frac{G_{m,L}}{G_{m,H}}} + C_2\sqrt{\frac{G_{m,H}}{G_{m,L}}}} \tag{4.4}$$

The derivation for these relations are outside the scope of this work. An in-depth treatment of the OTA-based C^4 can be found in [45].

The C^4 device sizes are the same as presented in [46], however an algorithmic design procedure for the C^4 presented in [44] allows for the filter to be easily altered to meet other design specifications if needed. In this case, the C^4 was designed for a maximum quality factor of 4.3 and a dynamic range of 50dB. The OTA is also the same as presented in [46], which is designed for an extended linear range in subthreshold operation and was originally presented by Furth, *et al.* in [47]. The schematic diagram for this transconductor is shown in Fig. 4.4 (b). The component parameters for this C^4 implementation are given in Table 4.2.

In (4.3), C_2 is a constant, $\tau_l \propto G_{m,L}^{-1}$, and $\tau_h \propto G_{m,H}^{-1}$, so the corner frequencies have no

Table 4.2: C^4 Device Sizes

Transistor	W (μm)	L (μm)
$M_1 - M_4$	6	1.2
$M_5 - M_6$	12	1.2
$M_7 - M_8$	1.5	9
M_9	12	1
$M_{10} - M_{11}$	3	2.4
M_{fg}	3	1.2

Capacitor	Value (fF)
C_1	50
C_2	25
C_W	3000
C_L	500
C_g	80
C_{tun}	2

dependence on one another, allowing them each to be programmed independently, as shown in Figures 4.4 (c) and (d) — this data was measured on a $0.35\mu m$ CMOS process as presented in [44], but is used here qualitatively. Figure 4.4 (c) shows the effect on frequency response holding $G_{m,H}$ constant and increasing $G_{m,L}$; Fig. 4.4 (d) shows the effect of doing the opposite.

4.2.1 C^4 Programming

The C^4s were operated using $V_{dd} = 3.3V$ and $V_{cg} = 3.0V$. Each corner frequency was set by programming different DAC words into the FG arrays using our parallel programmer. A characterization script extracted the relationships between DAC input word and τ_l and τ_h as well as between τ_l/τ_h and A_v and Q. The results of this script were then used to program the filters by directly specifying f_c, A_v, and Q.

Figure 4.5 demonstrates the results of applying this characterization to tune the C^4s to perform frequency decomposition at various bandwidths and filter spacings. Three filter spacings are demonstrated: full-octave spacing, half-octave spacing, and third-octave spacing. The value of Q for each of these configurations was chosen according to fractional octave spacing rules, such that the filters cross at their -3dB points. Therefore, $Q \sim 1.4$ for octave spacing, $Q \sim 2.9$ for half-octave spacing, and $Q \sim 4.3$ for third-octave spacing. Figure 4.5 (a) shows the results of programming the C^4s to octave spacing starting at $f_c = 88Hz$, (b) shows half-octave spacing beginning at $f_c = 300Hz$, and (c) shows third-octave spacing beginning at $f_c = 445Hz$.

4.3 Chapter Summary

A prototype circuit was fabricated using a $0.5\mu m$ standard CMOS process containing our parallel programmer. The programming accuracy of the parallel programming scheme was measured using the voltage reference configuration shown in Fig. 3.2 (a), and accuracy was computed out of twenty-five trials. A programming accuracy of 9.12-bits was achieved, indicating a loss of only

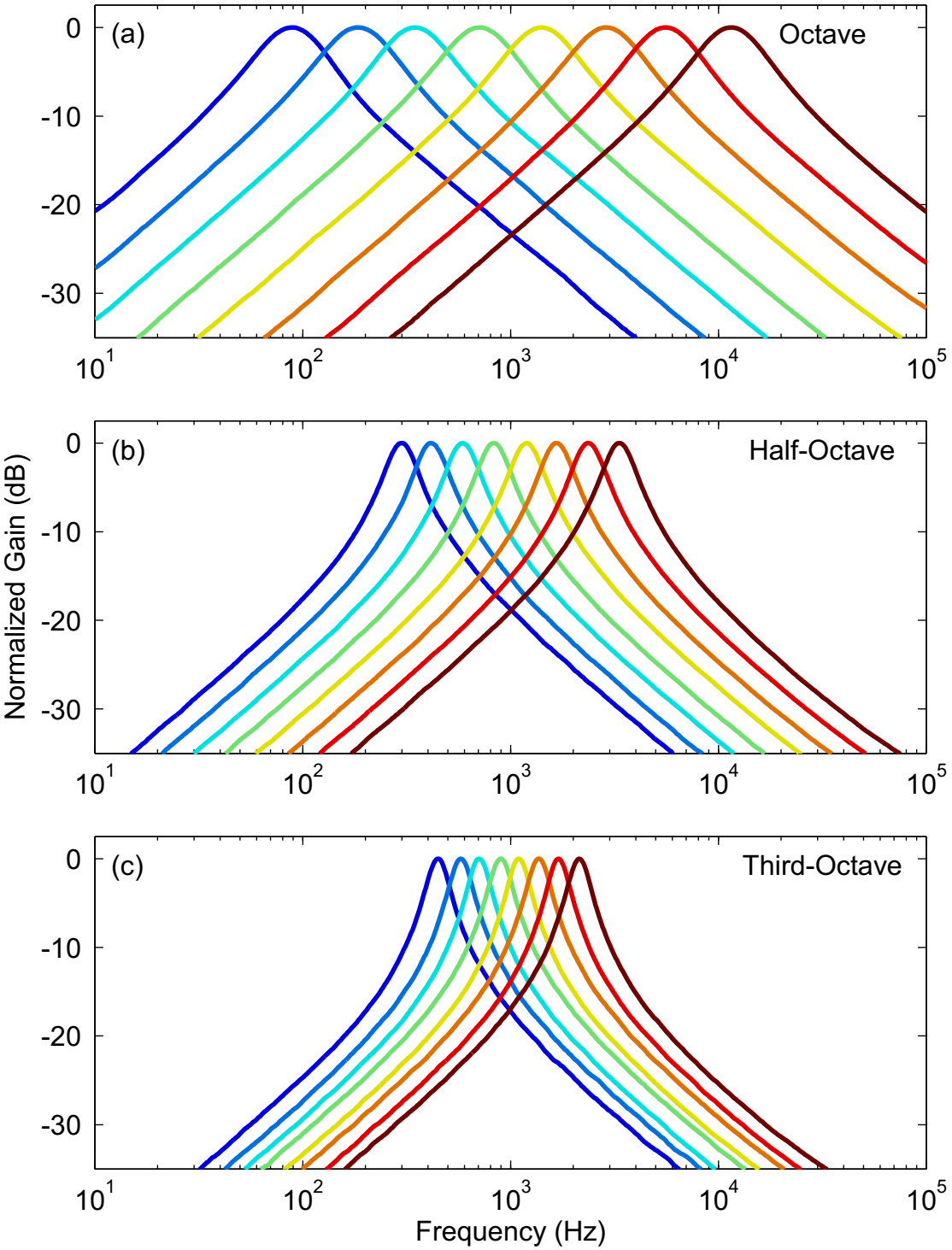

Figure 4.5: Programmed C^4 array frequency responses. (a) octave spacing starting at $f_c = 88$Hz, (b) half-octave spacing starting at $f_c = 300$Hz, and (c) third-octave spacing starting at $f_c = 445$Hz.

0.51-bits of accuracy through the use of our parallel programmer.

An 8-channel bandpass array was used to test the ability of the programmer array to tune operating parameters of a circuit. The filter topology chosen was the C^4, which offers independently tunable corner frequencies set by programming different DAC words into the FG memory cells. A characterization script was run on the C^4 which extracted relationships between DAC words and filter parameters. Using this characterization allows the user to program the C^4s by specifying only f_c, A_v, and Q. The functionality of this programming scheme was demonstrated by programming the filter bank to three different filter spacings: octave, half-octave, and third-octave spacings.

Chapter 5

Broader Applications, Conclusions and Future Work

Many analog floating-gate memory applications require large numbers of FGs to be integrated on a single die, requiring the programmer to be able to write to high volumes of FGs in a short period of time. The programmer circuit presented in this work has the potential to scale up to meet the demands of such dense FG arrays with little complexity. By allocating the FGs into rows that are programmed in parallel, good scalability is achieved since the number of S/Hs and programmers remains fixed while the number of FGs can be increased.

5.1 Field-Programmable Analog Arrays

One large-scale application that would benefit from this programming scheme is the area of field-programmable analog arrays (FPAAs). In fact, an early implementation of the programming DAC presented in this work was previously used for serially programming floating-gate arrays in our reconfigurable analog mixed-signal platform (RAMP) FPAA presented in [23]. FPAAs take inspiration from field-programmable gate arrays (FPGAs) in that they replace custom-designed application-specific integrated circuits (ASICs) with programmable architectures. The result is a reconfigurable platform that facilitates rapid prototyping of fully-integrated analog systems. These reconfigurable analog systems are usually employed in low-power signal processing applications to save energy where digital computations would be more costly. Many of these applications are tasks that are difficult or even impossible to perform using solely digital circuitry.

For instance, one brief example that was synthesized in our RAMP is a reconfigurable Wheatstone bridge used for temperature measurement. The Wheatstone bridge is a classic signal inter-

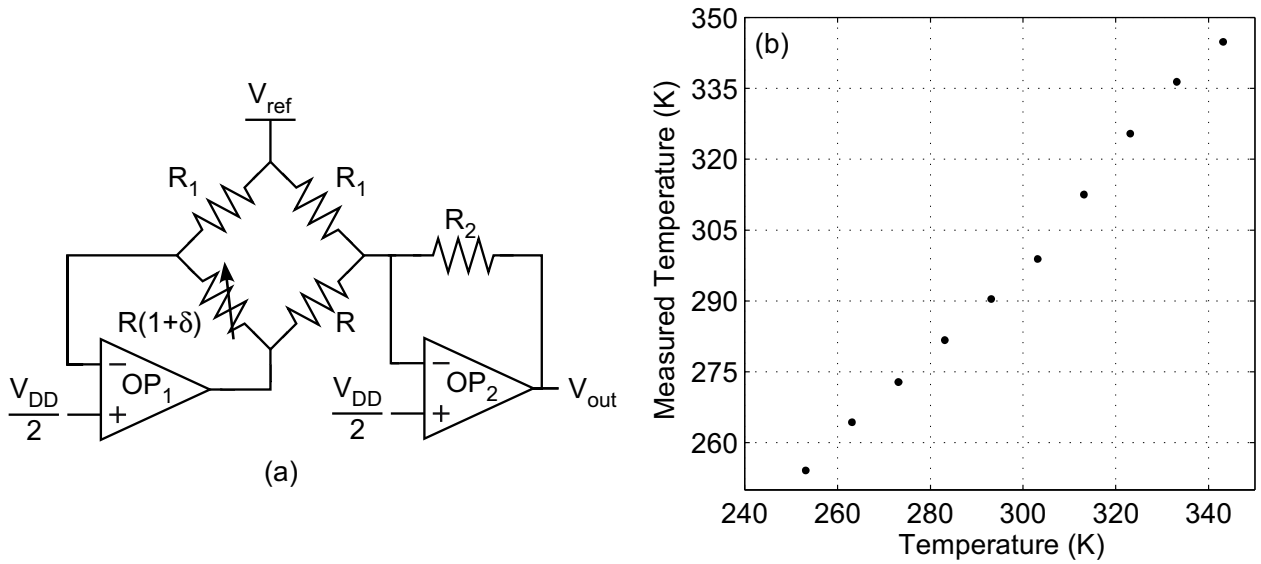

Figure 5.1: Wheatstone bridge for temperature measurement that was synthesized in our FPAA, employing non-volatile analog memory arrays. (a) Schematic diagram of the Wheatstone bridge circuit. (b) Measured temperature using a 1MΩ NTC thermistor.

facing circuit that is used to measure changes in resistance. The circuit synthesized in the FPAA, shown in 5.1 (a), is based on the classic Wheatstone bridge. However, it employs two op-amps to linearize V_{out} with respect to changes in δ [48]. The output voltage is expressed by

$$V_{out} = \frac{V_{dd}}{2} - \frac{R_2}{R_1}\delta\left(V_{ref} - \frac{V_{dd}}{2}\right) \tag{5.1}$$

where $V_{dd} = 2.5V$, $V_{ref} = 1.3V$, $R_1 = 1.1M\Omega$, $R_2 = 2.2M\Omega$, and $R = R_2 = 2.2M\Omega$. A resistive sensor was used in place of $R(1+\delta)$ in Fig. 5.1 (a), in this case a negative temperature coefficient (NTC) thermistor with a nominal resistance of $1M\Omega$ measured at $T = 270°K$ $(0°C)$. If the R vs temperature relationship is known, then V_{out} can be used to solve for temperature based on the change in resistance, as show in Fig. 5.1 (b).

Applications such as these are costly to perform in a fully digital platform, while low power and easy to accomplish using a reconfigurable analog architecture. Such low-power analog processing is especially useful in resource-constrained systems which rely on batteries or energy harvesting for power. Within these systems, FPAAs are capable of reconfiguring their analog circuitry on-the-fly, upon the detection of pre-determined external stimuli. In these cases, the speed at which the FPAA can reconfigure its circuitry determines how much information of this new stimulus is sensed or lost. It follows that faster reconfiguration time corresponds to better performance, thus, it is important for the programmer to be able to reconfigure quickly.

5.2 Conclusions and Future Work

This work has presented a new programming circuit for non-volatile analog memory arrays. By employing an SPI, DAC, and S/H array, the programming is performed using only digital inputs, greatly reducing the amount of overhead required to program an array of FGs. A proof-of-concept chip was fabricated in a $0.5\mu m$ standard CMOS process in order to demonstrate the viability of the proposed method. This chip contains 16 FGMOS transistors used to bias an 8-channel bandpass filter bank.

Programming accuracy using the circuit presented in [37] was measured to be 9.63-bits along with a gain error of -0.17% FSR. The measured accuracy of the programming method presented in this work was 9.12-bits along with a gain error of -0.4% FSR. Thus, our programming method only results in a loss of 0.51-bits of resolution as well as a -2% increase in gain error.

Although high accuracy was maintained, there were several limitations encountered that prevented this methodology from achieving the desired speed increase over serial programming. The first of these limitations was the reverse-bias pn junction leakage on the inverting terminal of the S/H OTA which caused high droop rates for low values of V_{ref}. To mitigate this droop, V_{ref} had to be raised to $4.1V$ during programming phases which required raising V_{dd} to $4.5V$ to allow the S/H OTA sufficient operating headroom. Extra settling time was required when raising this voltage since the programmer current bias circuit is operated from this supply. When V_{dd} is raised, this current bias requires time to stabilize before programming can begin; lower programming accuracy can result if the currents are not allowed sufficient time to settle. This limitation did not affect achievable injection speed but it did increase the overall time required to program the array.

The second limitation was the low-value for I_b/I_{start} of the programmer circuit. In order to save pins, I_{start} was hard-wired to share the same current as I_b of the programmer OTA. In order to achieve high programming rates, I_b must be set to $\sim 1\mu A$, which caused no issue in simulations; however, when the test chip was received from fabrication, it was discovered that high values of I_b created sufficiently large V_{ds} across M_4 in the programming circuit (Fig. 3.2 (c)), preventing the memory cell from beginning to program when V_{start} was pulsed. To mitigate this problem, I_b had to be lowered until V_{ds,M_4} was low enough to allow the start functionality to operate successfully. The value of I_b used to do this was $250nA$, which was low enough to cause a programming speed limitation of $\sim 600mV/s$, even for $V_{dd,fg} = 6.5V$ and $I_2 = 2nA$.

Programming times of $< 100ms$ were reported using this same memory cell in [37], albeit using

a $0.35\mu m$ process. Still, programming speeds of the chip presented in this work were significantly lower than what was expected to be achieved, due to the requirements on I_b.

On future implementations of this programmer, two main changes are suggested: (1) Remove the dependence of I_{start} on I_b. This can be accomplished by omitting I_{start} altogether or simply ratioing the current mirror between I_b and I_{start} such that $I_{start} < 100nA$ for $I_b = 1\mu A$. (2) Fix the droop problem caused by the S/H switch leakage. There are a number of potential approaches to solve this problem. One would be to simply use an entirely different S/H topology; however, the design time associated with this might prove too costly. Another would be to implement S_2 in the S/H using only nFET switches. This is possible due to the fact that if the S/H is operated on the low supply, $V_{dd} = 3.3V$, then ideally V_{ref} can be operated around midrail, $V_{mid} = 1.65V$. Since the nFET switches are capable of passing voltages in this region, only the nFETs are required. If this route is taken, then the charge injection cancellation must be solely performed through the use of other nFET dummy switches which are not always a sufficient solution. Thus, another solution might be to keep the transmission-gate switches but lower the logic level as well as bulk voltage of the pFETs in S_2 on the low supply level ($3.3V$). This latter option is likely the best solution since charge injection cancellation is still maintained to the same degree as the current implementation; also the change in the circuit would be very minimal, requiring only a HIGH-to-LOW level shifter and some re-routing of control signals.

Not explicitly mentioned until now is also the fact that V_{dd}, $V_{dd,fg}$, and V_{tun} as well as various other voltage and current biases were provided using off-chip sources. This chip was designed in the spirit of making programming FGs as easy as possible, so to that end, these voltage and current biases should be integrated on chip in future iterations of the programmer. A high-voltage tunneling charge-pump has already been reported in [23], and the design of an injection charge-pump is currently being finalized. Thus these shall be included in future iterations. These charge pumps were fabricated using a $0.35\mu m$ CMOS process so scaling the design of this charge-pump up to a larger process should be a relatively simple task.

References

[1] D. Kahng and S. M. Sze, "A floating gate and its application to memory devices," *Bell System Technical Journal*, vol. 46, no. 6, pp. 1288–1295, 1967.

[2] S. Lai, "Flash memories: where we were and where we are going," in *IEEE International Electron Devices Meeting*, 1998, pp. 971–974.

[3] ——, "Non-volatile memory technologies: the quest for ever lower cost," in *IEEE International Electron Devices Meeting*, Dec 2008, pp. 1–6.

[4] A. Benvenuti, A. Ghetti, A. Mauri, H. Liu, and C. Mouli, "Current status and future prospects of non-volatile memory modeling," in *International Conference on Simulation of Semiconductor Processes and Devices*, Sep 2014, pp. 5–8.

[5] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim, "A case for flash memory SSD in enterprise database applications," in *ACM SIGMOD International Conference on Management of Data*.

[6] P. Hasler, B. Minch, and C. Diorio, "Floating-gate devices: they are not just for digital memories any more," in *IEEE International Symposium on Circuits and Systems*, vol. 2, Jul 1999, pp. 388–391.

[7] C. Mead, *Analog VLSI and Neural Systems*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[8] T. Shibata and T. Ohmi, "A functional MOS transistor featuring gate-level weighted sum and threshold operations," *IEEE Transactions on Electron Devices*, vol. 39, no. 6, pp. 1444–1455, Jun 1992.

[9] M. Holler, S. Tam, H. Castro, and R. Benson, "An electrically trainable artificial neural network (ETANN) with 10240 'floating gate' synapses," in *International Joint Conference on Neural Networks*, vol. 2, 1989, pp. 191–196.

[10] A. Thomsen and M. Brooke, "A floating-gate MOSFET with tunneling injector fabricated using a standard double-polysilicon CMOS process," *IEEE Electron Device Letters*, vol. 12, no. 3, pp. 111–113, March 1991.

[11] B. Lee, B. Sheu, and H. Yang, "Analog floating-gate synapses for general-purpose VLSI neural computation," *IEEE Transactions on Circuits and Systems*, vol. 38, no. 6, pp. 654–658, Jun 1991.

[12] D. Durfee and F. Shoucair, "Comparison of floating gate neural network memory cells in standard VLSI CMOS technology," *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 347–353, May 1992.

[13] O. Fujita and Y. Amemiya, "A floating-gate analog memory device for neural networks," *IEEE Transactions on Electron Devices*, vol. 40, no. 11, pp. 2029–2035, Nov 1993.

[14] C. Diorio, P. Hasler, B. Minch, and C. Mead, "A floating-gate MOS learning array with locally computed weight updates," *IEEE Transaction on Electron Devices*, vol. 44, no. 12, pp. 2281–2289, Dec 1997.

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

[16] Y. Wong, M. Cohen, and P. Abshire, "A 750-MHz 6-b adaptive floating-gate quantizer in 0.35-μm CMOS," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 7, pp. 1301–1312, July 2009.

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

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

[19] R. Harrison, J. Bragg, P. Hasler, B. Minch, and S. DeWeerth, "A CMOS programmable analog memory-cell array using floating-gate circuits," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, no. 1, pp. 4–11, Jan 2001.

[20] C. Twigg, J. Gray, and P. Hasler, "Programmable floating gate FPAA switches are not dead weight," in *IEEE International Symposium on Circuits and Systems*, May 2007, pp. 169–172.

[21] T. Hall, C. Twigg, J. Gray, P. Hasler, and D. Anderson, "Large-scale field-programmable analog arrays for analog signal processing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 11, pp. 2298–2307, Nov 2005.

[22] C. Twigg and P. Hasler, "A large-scale reconfigurable analog signal processor (RASP) IC," in *IEEE Custom Integrated Circuits Conference*, Sept 2006, pp. 5–8.

[23] B. Rumberg, D. Graham, S. Clites, B. Kelly, M. Navidi, A. Dilello, and V. Kulathumani, "RAMP: Accelerating wireless sensor hardware design with a reconfigurable analog/mixed-signal platform," in *Proceedings of the ACM/IEEE Conference on Information Processing in Sensor Networks*, Apr 2015, pp. 47–58.

[24] S. Shah and S. Collins, "A temperature independent trimmable current source," in *IEEE International Symposium on Circuits and Systems*, vol. 1, 2002, pp. 713–716.

[25] S. Jackson, J. Killens, and B. Blalock, "A programmable current mirror for analog trimming using single poly floating-gate devices in standard CMOS technology," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, no. 1, pp. 100–102, Jan 2001.

[26] A. Negut and A. Manolescu, "Analog floating gate approach for programmable current mirrors and current sources," in *International Semiconductor Conference*, vol. 02, Oct 2010, pp. 525–528.

[27] L. Carley, "Trimming analog circuits using floating-gate analog MOS memory," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 6, pp. 1569–1575, Dec 1989.

[28] P. Hasler, B. Minch, and C. Diorio, "An autozeroing floating-gate amplifier," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, no. 1, pp. 74–82, Jan 2001.

[29] T. Constandinou, J. Georgiou, and C. Toumazou, "An auto-input-offset removing floating gate pseudo-differential transconductor," in *International Symposium on Circuits and Systems*, vol. 1, May 2003, pp. 169–172.

[30] B. Rumberg and D. Graham, "Efficiency and reliability of fowler-nordheim tunnelling in cmos floating-gate transistors," *Electronics Letters*, vol. 49, no. 23, pp. 1484–1486, Nov 2013.

[31] M. Lenzlinger and E. Snow, "Fowler-Nordheim tunneling into thermally grown SiO_2," *IEEE Transactions on Electron Devices*, vol. 15, no. 9, p. 686, Sep 1968.

[32] K. hyoun Kim, K. Lee, T.-S. Jung, and K.-D. Suh, "An 8-bit-resolution, 360-μs write time nonvolatile analog memory based on differentially balanced constant-tunneling-current scheme (DBCS)," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 11, pp. 1758–1762, Nov 1998.

[33] C. Diorio, "A p-channel MOS synapse transistor with self-convergent memory writes," *IEEE Transaction on Electron Devices*, vol. 47, no. 2, pp. 464–472, Feb 2000.

[34] S. Chakrabartty and G. Cauwenberghs, "Fixed-current method for programming large floating-gate arrays," in *IEEE International Symposium on Circuits and Systems*, vol. 4, May 2005, pp. 3934–3937.

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

[36] H. Roman and G. Serrano, "A system architecture for automated charge modification of analog memories," in *53rd IEEE International Midwest Symposium on Circuits and Systems*, Aug 2010, pp. 1069–1072.

[37] B. Rumberg and D. Graham, "A floating-gate memory cell for continuous-time programming," in *IEEE International Midwest Symposium on Circuits and Systems*, Aug 2012, pp. 214–217.

[38] M. R. Kucic, "Analog computing arrays," Ph.D. dissertation, Georgia Institute of Technology, December 2004.

[39] R. Baker, *CMOS: Circuit Design, Layout, and Simulation*, 3rd ed. Hoboken, NJ: John Wiley & Sons, Inc., 2011.

[40] P. Allen and D. Holberg, *CMOS Analog Circuit Design*, 2nd ed. New York, NY: Oxford University Press, Inc., 2002.

[41] P. Lim and B. Wooley, "A high-speed sample-and-hold technique using a Miller hold capacitance," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 4, pp. 643–651, Apr 1991.

[42] T. Carusone, D. Johns, and K. Martin, *Analog Integrated Circuit Design*, 2nd ed. Hoboken, NJ: John Wiley & Sons, Inc., 2012.

[43] B. Razavi, *Design of Analog CMOS Integrated Circuits*. New York, NY: McGraw-Hill Higher Education, 2002.

[44] B. Rumberg and D. Graham, "A low-power and high-precision programmable analog filter bank," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, no. 4, pp. 234–238, April 2012.

[45] B. D. Rumberg, "Low-power and programmable analog circuitry for wireless sensors," Ph.D. dissertation, West Virginia University, December 2014.

[46] B. Rumberg, D. Graham, and V. Kulathumani, "A low-power, programmable analog event detector for resource-constrained sensing systems," in *IEEE International Midwest Symposium on Circuits and Systems*, Aug 2012, pp. 338–341.

[47] P. Furth and A. Andreou, "Linearised differential transconductors in subthreshold CMOS," *Electronics Letters*, vol. 31, no. 7, pp. 545–547, Mar 1995.

[48] S. Franco, *Designing with Operational Amplifiers and Analog Integrated Circuits*, 3rd ed. New York, NY: McGraw-Hill Higher Education, 2002.