LEARNING IN SILICON: A FLOATING-GATE BASED, BIOPHYSICALLY INSPIRED, NEUROMORPHIC HARDWARE SYSTEM WITH SYNAPTIC PLASTICITY

A Thesis
Presented to
The Academic Faculty

by

Stephen Brink

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Biomedical Engineering

Georgia Institute of Technology
December 2012

LEARNING IN SILICON: A FLOATING-GATE BASED, BIOPHYSICALLY INSPIRED, NEUROMORPHIC HARDWARE SYSTEM WITH SYNAPTIC PLASTICITY

Approved by:

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

Professor David Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology

Asst. Professor Pamela Bhatti
School of Electrical and Computer Engineering
Georgia Institute of Technology

Assoc. Professor Robert Liu
Department of Biology
Emory University

Professor Steve DeWeerth
School of Biomedical Engineering
Georgia Institute of Technology

Date Approved: 23 August 2012

To my amazing family and friends.

Your love, support, and companionship

mean the world to me.

ACKNOWLEDGEMENTS

One of the most gratifying moments of a major accomplishment is giving credit to all of the people who have helped along the way, and I certainly have a lot of people to thank, so here goes. To my parents, thank you for inspiring my love of learning, and for guiding and supporting my 24-year-long education every step of the way. More importantly, thank you for teaching me to value honesty, to not fear hard work, and to treat others with respect and love. You are the most important good influence in my life.

To Lauren, thank you for brightening my life for the past 5 years. You have helped me keep things in perspective through the ups and downs of graduate school. I am thrilled and excited for us to enter the next stage of our lives together.

To David and Jared, Earl and Cindy, Dylan, Christian and Sarah, Nick and Amber, Ryan and Laveeta, Aubrey, Travis, Andrew, Wes, Ali, and Quill, thank you for the countless hours that we have spent together. You have enriched my life in so many ways. I have learned so much from you all, and you have challenged and inspired me to be the best that I can be. I cherish the happy memories of times we have had, and look forward to many more to come.

To Mark van Schilfgaarde, thank you for giving me my first opportunity to participate in a scientific research effort. Thank you for introducing me to the extremely rewarding study of modern physics. You were my first role model of a scientist who expects and demands that the theory agrees with experiment, and your example remains influential to me to this day.

I have had the privilege of learning from some excellent educators throughout my life, but Denis Kirkley, Dan Ayub, Ian Gould, and Jerry Coursen have been especially

influential to me. Thank you all for your dedication to maintaining the highest standard of education, and thank you for the incredibly positive impact you have had on my life.

Arindam Basu, thank you for being a fantastic mentor to me. I have learned so much from you about IC design, mathematics, and dynamics. Working closely with you was one of the highlights of graduate school for me. I can't enumerate the contributions you made to my work because you influenced all of it.

Richie Wunderlich, thank you for the many enlightening discussions of quantum physics and semiconductor devices. Thank you for teaching me most of what I know about digital design, for your instruction and technical support regarding CAD tools and IC layout, and for your significant contributions to the adaptive FPAA, the tunneling studies, and the neuron IC. Thank you so much for providing moral support at the moments when I needed it most. Also thanks for including me in memorable events such as the hot tub camping.

Steve Nease, thank you for your huge contributions to the neuron IC, including the mismatch characterization, STDP measurements, help with layout, and countless coding and debugging details. A significant portion of the credit for the successes of this effort belongs to you, and I'm excited to see what you do with this research effort in the future. Thank you also for the many discussions about philosophy of science and life in general, and for your incredibly positive and cooperative attitude. Collaborating with you has been a delight.

Scott Koziol, thank you for your early work on the PCB designs, for teaching me about the PCB CAD tools, and for the rewarding collaboration on the path planning project. Thanks for the countless administrative tasks you have performed through the years to support the lab. Thanks for lending a sympathetic ear in times of frustration, and for your bright personality that has always made our lab a more enjoyable place to work.

Shubha Ramakrishnan, thank you for your help with the design of the neuron IC and your significant contributions to the STDP implementation and measurements. Farhan Adil, thank you for your help with the neuron IC design, and for providing me with your insights about RF design and IC design in general. Brian "Degs" Degnan, thanks for your help with the neuron IC layout, the USB coding for the microcontroller, for helping out with creating artwork at various times, and for the test chip that you made that proved so valuable for my research. Thanks also for making the lab a more interesting place by always saying things that nobody else would dream of saying, regaling us with your (tall?) tales, sharing your amazingly unique life experiences, and keeping me fixed up with new music. Jordan Gray, thanks for so many interesting technical discussions about circuits and systems and computing, and thanks for willingly helping me get set up with useful software and hardware tools. Bo Marr, thanks for including me in some of your interesting research early in my graduate career, and for being such a fun companion at the Telluride Neuromorphic Workshop. Hakan Toreyin and Jon Bergdoll, thank you for your willing help with the neuron IC layout. Craig Schlottmann, thanks for your early work on the PCB design. Sam Shapero, thanks for engaging me in some very interesting discussions about neural computing, neural modeling, and circuit design. You continue to impress me with your work, and a few of your ideas have been very influential to me. Gokce Gurun and Jaime Zahorian, thank you for providing me with die photos of my ICs. Csaba Petre, thanks for sharing my enthusiasm for neural computation, and for the interesting discussions that resulted.

To the "older generation"– Chris Twigg, Ryan Robucci, Dave Abramson, Kofi Odame, Thomas Peng, and Christal Gordon– thank you for the legacy of hardware, software, and general know-how that you passed down, and for your belief in me, for the inspiration to accomplish great things, and for the many nuggets of sage advice.

To Suma George, presently the lone representative of the "younger generation",

thanks for the fun discussions that we've had. I look forward to seeing some exciting results of your research. Thank you to my many colleagues in BME and ECE for providing companionship, camaraderie, and inspiration. In particular, thank you to Jon Newman, Keith Van Antwerp, and Prashanth Challa for some engaging and influential scientific discussions.

Thanks to my committee for thoughtful critiques of my research, and for your positive influence on my graduate career through the various roles that you have played.

To Jennifer Hasler, my advisor, thank you for facilitating my involvement in neuromorphic engineering research. You have helped me to realize goals and ambitions that I have had for many years prior to coming to Georgia Tech. I have learned a tremendous amount about semiconductor devices and circuit design from you. You introduced me to fascinating field of IC design and insisted that I become an expert in it. This has created many exciting possibilities for a rewarding career for me in the future. Thank you for always pushing me to excel and for helping me out at crucial times when I was struggling.

Thanks to the wonderful faculty and staff of the BME and ECE schools and the bioengineering program.

Thanks to the NSF for providing the majority of my funding by way of the IGERT program and the graduate research fellowship. Special thanks to Steve DeWeerth and Rob Butera for doing such a great job with the IGERT program. Thanks to DARPA for providing some funding for my work, through the CT2WS, SyNAPSE, and IRIS programs.

Contents

DEDICATION . . . . . **iii**

ACKNOWLEDGEMENTS . . . . . **iv**

LIST OF TABLES . . . . . **xi**

LIST OF FIGURES . . . . . **xii**

SUMMARY . . . . . **xxviii**

I THE BIG PICTURE **1**

1.1 Goals: neural and analog computing 1

1.1.1 Why study neural computing? 1

1.1.2 The importance of computational modeling 3

1.1.3 Computational modeling by numerical integration 5

1.1.4 The neuromorphic approach 7

1.1.5 Analog computing 9

1.2 Methods: VLSI and floating-gate transistor technology 10

1.3 Achievements: from devices to systems 12

II BUILDING BLOCKS FOR FLOATING-GATE SYSTEMS . . . **14**

2.1 Hardware platform 14

2.2 Fast and flexible software interface 17

2.2.1 Communication speed and floating gate programming times . 17

2.2.2 Flexible framework to support multiple hardware systems . . 17

2.2.3 Specialized functions for data acquisition 18

2.3 Approach for floating-gate transistor programming 18

2.3.1 Modeling hot electron injection 19

2.3.2 Reasoning for injection algorithm 22

2.4 FPAA circuits measured in this framework 25

III NEW DEVICE CHARACTERISTICS: BI-DIRECTIONAL TUNNELING 36

3.1 Floating-gate array programming and bidirectional tunneling 36

3.2 Modeling floating-gate voltage and electron tunneling 39

3.3 Measured tunneling characteristics 45

3.4 Choosing bidirectional tunneling parameters 50

3.5 Characterization method 56

3.6 Process variation of tunneling properties 59

IV ANALOG COMPUTATION USING FLOATING-GATE ADAPTATION 62

4.1 The core device: floating-gate nFET 63

4.2 An FPAA with floating-gate adaptation 66

4.3 Software support- routing tool 72

4.4 Measured results- adaptive circuits 73

V A LEARNING-ENABLED NEURON ARRAY IC 81

5.1 Design paradigm and resulting architecture 81

5.2 Circuits for spiking network implementation 85

5.3 Characteristics of neuron behavior 93

5.4 Circuits for synaptic plasticity 99

5.5 System tools enabling network experiments 106

5.6 Power efficiency analysis 108

VI COMPUTING WITH SPIKING NETWORKS ON NEURON ARRAY IC 111

6.1 Introduction 111

6.2 Basic networks: “dot matrix” and synfire chain 112

6.2.1 “Dot matrix” network 112

6.2.2 100-neuron synfire chain 113

6.3 Computational networks 115

6.3.1 Spatiotemporal pattern generation and detection 115

6.3.2 Ring Winner-Take-All (WTA) network 117
6.3.3 Bistable oscillator 122
6.4 Path planning 124
6.5 Conclusion 131
VII LEARNING IN SPIKING NETWORKS 135
7.1 Two-synapse model 135
7.2 Multi-synapse interactions 136
7.2.1 Methods: weight initialization and rate-coded input pattern generation 137
7.2.2 Potentiation of synchronous inputs 138
7.2.3 Competition among rate encoded inputs 139
7.2.4 Learning from correlations in the presence of noise 142
7.2.5 Receptive field formation 142
VIII CONCLUSIONS AND FUTURE DIRECTIONS 146
REFERENCES 151
VITA 161

List of Tables

1 Device parameters for two example devices characterized in this study. The capacitive coupling ratios for terminals of devices that share global terminal voltages for tunneling operations results in different floating gate voltages and therefore different tunneling rates for the different devices. Accounting for this effect is essential for achieving desired behavior from tunneling operations. The abbreviation "HC" (for high coupling) is used for the device whose gate capacitor is large, whereas "LC" is used for the other device. 41

2 Ranges of terminal voltages used for tunneling 43

3 Optimal choice of tunneling parameters for 3 different realistic sets of constraints 54

4 Comparison of synapse density and function. The FACETS IC, Stanford STDP, INI IC 1, and INI IC 2 chips are described in detail in ([57, 56, 5, 38, 14]) . The normalized synapse area is computed by dividing the synapse area by the square of the process feature size. . . 91

5 Estimated power consumption for a neuromorphic simulation of 100 neurons and 30,000 synapses, wherein each neuron spikes at an average rate of 10 Hz. The total power of 204 uW is about 6000 times smaller than the estimate of the power required for running the simulation using numerical integration. 110

6 Estimated energy to perform the network simulations presented in this work on this neuromorphic platform versus a numerical integration approach. 133

List of Figures

1 Printed circuit board for programming and testing floating-gate array ICs. The block diagram is shown in a), and a photograph of the board is shown in b), with a pencil to show the scale. 15

2 Comparison of voltage supplies and DAC output on new PCB versus the previous version. All of the signals except for the 5V supply are dramatically improved. 16

3 Comparison of output of audio circuit on new PCB versus previous version. In both measurements, the input to the amplifier is a 30mV sine wave. The substantial improvement in the performance for the new board is due to the greatly reduced noise on the 12V supply, which powers the audio amplifiers. 16

4 Schematic of floating gate cell. Capacitors drawn with dashed lines represent parasitic capacitances. 20

5 Measured characteristics of hot electron injection for floating-gate pFET. In this measurement, the drain voltage is always 0 so V_{dd}=V_{sd}. The current is normalized by the total capacitance at the floating gate, which was not known, and is generally a difficult parameter to measure experimentally. 21

6 Measurement of frequency response of on-chip op amp at various bias currents. The circuit diagram is shown in a) and the measured results are in b). As the bias current is increased, the amplifier's bandwidth increases while the phase margin decreases. The bias current that corresponds to critical damping (thus avoiding a resonant peak in the transfer function) is between 100nA and 1uA, and yields a corner frequency between 1 and 10MHz. 26

7 First order low pass filters measurements. The performance of the floating-gate OTA based low pass filter is shown in a), while that of the conventional OTA based low pass filter is shown in b). Schematics of both circuits are shown in c). The capacitive attenuator in the floating-gate OTA yields an attenuation of about 10x, which reduces the transconductance, and therefore the low pass corner frequency. The fact that the corner frequency of a single circuit can be programmed over a range of 6 orders of magnitude illustrates the flexibility derived from using subthreshold circuit design techniques and floating gate transistor technology. 29

8 Family of curves resulting from DC sweeps of the floating gate OTA depicted in Figure 7 with a range of differential offset voltages programmed on the inputs. With a signal on the input, this circuit functions as a programmable level shifter, and its range of programmability is such that with the input grounded it functions as a rail-to-rail programmable supply-independent bias voltage generator. 30

9 High pass filter circuit and measurement results. Programming the floating gate biases for the OTAs over many decades of current yields a large range of tunability for the filter characteristics. Measured performance is shown in a) and the circuit topology is shown in b). The capacitive divider provides an attenuation factor of approximately 10x. 30

10 Resonator topology for high-Q bandpass filter that can be used for spectral decomposition. Measured performance is shown in a) and circuit topology is shown in b). The capacitive dividers provide an attenuation factor of approximately 10x to each OTA. 31

11 Capacitively coupled current conveyor (C4) topology for high-Q bandpass filter that can be used for spectral decomposition. Measured performance is shown in a) and circuit topology is shown in b). The capacitive dividers provide an attenuation factor of approximately 10x to each OTA. 32

12 Unity-gain inverting amplifier implemented with a floating gate and capacitive feedback. Measured performance for several bias voltages is shown in a), and the circuit topology is shown in b). 33

13 Voltage-mode summing circuits that take advantage of capacitive summation at a floating node. In a) and b), the response of the circuit to consecutive triangle waves on the two inputs is shown. The measured output is plotted with open circles and a linear combination of the two inputs is plotted as a solid line. The coefficients in the linear combination are nearly equal, but not exactly (due to mismatch between the two input capacitors). In c), the circuit diagrams are shown for two different version of the summing circuit (one with a 6x capacitive attenuation and one without any attenuation). 34

14 Full-wave rectifier circuit. The response of the circuit to a sinusoidal input is shown in a), and the circuit topology is shown in b). 35

15 Context and motivation for the study of tunneling with floating-gate transistor arrays. a) Structure of field-programmable analog arrays (FPAAs) wherein this work is applied. Such chips contain hundreds of thousands of floating-gate transistors, the charge of which is manipulated by global tunneling operations. b) Description of a single cycle of programming for the floating gates in such arrays. c) Cross-section view of floating-gate transistor with tunneling junction and gate coupling capacitor. Some example terminal conditions for forward and reverse tunneling are shown. d) Measured I-V relationship of the tunneling junction for both directions of tunneling. Reverse tunneling clearly requires much lower applied voltages to obtain similar currents. 38

16 Depiction of floating-gate voltages throughout the various phases of a programming cycle. The forward tunnel operation turns off all devices, but raises the voltage on some devices too far, making them impossible to inject reliably. The reverse tunnel operation then lowers the voltages back into the range wherein targeted injection can be done successfully. The trajectories of tunneling for infinite time illustrate what happens to un-injected devices in a floating-gate array that does not use bidirectional tunneling. 39

17 Details of the core floating-gate element. a) Schematic of floating-gate element. Parasitic capacitances are denoted with dashed lines. b) . . 40

18 Band diagrams depicting the tunneling processes. a) Reverse and b) forward Fowler-Nordheim tunneling band diagrams for a an n-well varactor with a degenerately p-doped gate. 43

19 Measured Fowler-Nordheim plots of forward and reverse tunneling characteristics. a) Setup for characterizing tunneling. Iteration with the control loop allows the tunneled charge to be calculated as $C_{gate}\Delta V_{DAC}$. b) Forward tunneling characteristics for high coupling (HC) and low coupling (LC) devices, characterized with two different fixed tunneling voltages V_{tun}. b)Reverse tunneling characteristics high coupling (HC) and low coupling (LC) devices. 46

20 Tunneling characterization with direct control of V_{fg}. a) Test circuit used for measurements. b) Measured Fowler-Nordheim plots of the characteristic for 3 different voltages V_{tun}. c) Example waveform measured at V_{out} in the circuit shown in (a). 47

21 Measured and simulated trajectories of floating-gate voltages during a 5-second tunneling pulse followed by a 5-second reverse tunneling pulse. a) Floating-gate voltage trajectory a) for low coupling (LC) device and b) for high coupling (HC) device. 49

22 Measured and simulated V_{fg} over many programming cycles comparing bidirectional tunneling to forward tunneling only. The floating gate voltage just prior to the targeted injection phase of programming is measured in each cycle. Modeled trajectories are shown as solid lines and measured trajectories are shown as crosses and open circles. a) Multi-programming cycle behavior when only forward tunneling is used in the programming cycle. b) Multi-programming cycle behavior when bidirectional tunneling is used. 51

23 Floating gate voltage trajectories of three large populations of different device types during tunneling. The "switch" devices, shown in black, are the transistors that make up the switch matrix for routing in the FPAA. The "diff pair input" devices, shown in blue, are the input floating gates for the floating-gate based OTAs described in Chapter 2 (they are the same as the "HC" devices in this chapter). The "bias" devices, shown in red, are the programmable bias transistors for OTAs (they are the same as the "LC" devices in this chapter). The floating gate voltages spread significantly, and they also are seen to cluster by device type. 60

24 Analysis of the distribution of the final floating gate voltages depicted in Figure 23, according to physical location on the array. In a), the values are plotted versus the column coordinate of the device. Because of the chip architecture, the switch devices are found at most column addresses, while the devices in the CABs (the OTA bias and differential pair devices) only exist on two column addresses in each of the 4 columns of CABs on the IC. The the OTA bias and differential pair devices found in the rightmost column of CABs tunneled significantly faster than the same types of devices in the rest of the array. In b), the OTA bias devices alone are plotted versus their location within a CAB (the local row address). Two of the locations in the cab (rows 38 and 43) are observed to tunnel faster than the other two locations (at rows 17 and 33). 61

25 Characterization of hot carrier injection for the floating gate nFET. a) Circuit used for characterization. The amplifier is used to fix the floating gate voltage at V_{ref}, and the slew rate of V_{out} provides a measure of the current being injected to the floating gate. An nFET and pFET share the same floating gate so that the charge may be easily manipulated by tunneling or hot electron injection in the pFET. b) Measured injection currents in the nFET for a range of gate voltages from 0.3 to 2.5V, and drain voltages from 4.4 to 6V. As expected from the theory, hot hole injection is the dominant source of gate current when the channel current is low, and hot electron injection dominates when the channel current is sufficiently high. The maximum current over this range of terminal conditions is approximately 1000x smaller than the corresponding maximum for a floating gate pFET. 65

26 Results from stress testing of floating gate nFET by 350 cycles of 7.5 pC hot hole injection. a) The rate of hole injection declines significantly with the amount of charge injected. Similar sweeps to those shown in Figure 25 were taken after the test in order to verify that the peak of injection had reduced in amplitude, not simply shifted. b) Gate sweeps taken every 30 cycles show a significant change in the I-V characteristic of the device. c,d) The curves in b) were shifted horizontally so as to coincide at a drain current of 1 uA. The required shifts are plotted versus cycle number in c) (This roughly corresponds to threshold voltage shift), and the shifted curves are plotted together in d), showing the reduction in the slope of the characteristic with increasing cycle number. 67

27 Overview of adaptive FPAA. a) Schematic of one tile (CAB and routing) of the array, highlighting the different floating gate FETs used for low voltage and high voltage routing, and for the disconnect switch between rows of high and low voltage routing. b) Schematic of all of the components found in a single CAB. The most important component in the CAB is the adaptive floating gate, which has a self-contained amplifier for tunneling, and multiple FETs that can be compiled into circuits through the routing. The tunneling amplifier allows a voltage in the routable range (0-$V_{dd,inject}$) to control a higher tunneling voltage that is applied to the floating gate. c) Schematic of the tunneling amplifier. All of the transistors in the amplifier are 13.5V devices. . . 69

28 Maximizing dynamic range of nFET switches by preventing unwanted reverse tunneling during array programming. a) The schematic of the nFET switch cell is shown before (left) and after (right) the metal mask change to the design. Keeping the terminals of the nFET switches at V_{dd} during programming minimizes the unwanted reverse tunneling that takes place. The result of a voltage sweep of a single switch is shown b) before the change and c) after the change. The operating range of the high voltage nFETs is seen to be extended by about 1V, a substantial improvement that allows orders of magnitude higher hot electron injection currents in the adaptive pFET CAB element. In both b) and c), the sweeps are shown for a range of gate coupling voltages and supply voltages (V_{gate}'s and V_{dd}'s), both of which can be used to globally increase the floating gate voltages, and thereby increase the maximum range of the floating gate nFETs. 71

29 Low voltage performance of a) nFET and b) pFET switches. The currents in the nFET switch sweeps at high floating gate voltages are indicative of parasitic currents coming from "off" nFET devices in the array, and the result in b) illustrates the degradation of the pFET switch conductance at low voltage as the floating gate voltage is raised. Both of these effects can be mitigated by choosing $V_{gate} = V_{dd} = 2.7V$, which still results in acceptable high voltage nFET switch performance. 72

30 System of software tools that constitute the design environment for typical FPAAs. For the adaptive FPAA, this framework was replaced with a more low level MATLAB tool that provides a compromise between software development time and ease of use for the system. . . 73

31 Measured results from continuous-time a) tunneling and b) injection using the adaptive pFET element in the CAB of the adaptive FPAA. The qualitatively different dynamics of injection and tunneling are apparent. The rate of tunneling monotonically decreases over the course of the tunneling exposure, while the injection process is characterized by a positive feedback process that saturates at a large current level. 74

32 Autozeroing floating gate amplifier circuit (AFGA) originally described in [33]. The circuit diagram is shown in a) and the measured step responses, demonstrating time constants from under a minute to about an hour, are shown in b). 75

33 Measured output of AFGA circuit when the input is a square wave with successively increasing amplitude. The nonlinearity of the amplifier is clearly seen to result in an encoding of the signal amplitude in the output DC level. 75

34 Floating gate differential pair circuit. The circuit diagram is shown in a), and the continuous-time injection curve is shown in b). The unstable dynamics eventually results in a condition in which all of the current is being conducted through branch 1, and essentially no current is flowing through branch 2. 76

35 Fully differential OTA circuit with adaptive differential pair and common-mode feedback. The circuit, shown in a), exhibits floating gate adaptation that is characterized by a stable equilibrium, which is independent of the differential mode of the input voltages. The settling back to this equilibrium after perturbations is illustrated by the step responses on b) V_{in1} and c) V_{in2}. The asymmetry of the step responses is due to mismatch between the injection, tunneling, and I-V characteristics of the two branches of the amplifier. The adaptation could be enabled to allow the circuit to compensate an input offset, then disabled to allow the differential amplifier to operate around that differential offset. This capability is verified by differential sweeps of the amplifier before and after the supply voltage is ramped down to a level that prevents adaptation. The result of these sweeps, shown in d), shows that the offset that results from the adaptation is preserved when the supply voltage is reduced. 78

36 Adaptive "bump" circuit. The schematic is shown in a) and differential sweeps are shown before and after two stages of adaptation in b). This circuit, under this bias condition, is another example of an unstable, diverging form of adaptation, much like the adaptive differential pair. 79

37 Adaptive floating gate correlation detector. The circuit, shown in a), is based upon a pFET source follower, and a second pFET on the floating gate drives a current into two series diode-connected nFETs. The voltage across the nFETs provides a measure of the floating gate voltage, which is determined by a balance of injection and tunneling processes. If the input signals V_1,V_2, and V_3 are highly correlated, the rate of injection increases, lowering the floating gate voltage. In b), the equilibrium output voltage is plotted versus the phase shift from V_1 to V_2 and V_3. The output voltage is seen to track the cross-correlation of the inputs. 80

38 Illustrations of the design paradigm and its parallels to biology. In a), the model for a single neuron is depicted. In b), the model for two neurons is shown, and cartoons of some of the signal waveforms give a qualitative appreciation for the signal flow. 82

39 Signal flow for a single neuron in the neuron array IC. The synaptic input currents are summed on a wire. When the synaptic input currents are sufficiently strong, the model neuron emits a spike, which is picked up by the spike detector block and transmitted to the AER system and a gate waveform shaping circuit. The gate waveform shaping circuit conditions the pulse from the spike detector to create the desired time course for the post-synaptic currents on all of the neuron's output synapses. 83

40 Top level architecture of the IC. In a) the connectivity of the synapse array is shown, including the partition of synapses into recurrent connections, AER inputs, learning-enabled, fixed weight, and configurable excitatory/inhibitory synapses. All of the STDP synapses are excitatory. In b) a die micrograph shows the amount of die area dedicated to the neuron models (Soma Array), the synapse array, and the gate waveform shaping circuits (Triangle Circuits). 84

41 PyNN-based Matlab code for setting up a network simulation. . . . 85

42 Models for voltage gated channels that constitute a model neuron. The voltage supplies and biases V_{amp}, $V_{ss,amp}$, E_{Na}, E_K, V_{gK} are shared by all neuron models in the array. The use of floating gate transistors in the models allow for their conductances and frequency responses to be programmed individually. This allows for networks of neurons with heterogeneous properties to be modeled. 86

43 Schematic of transimpedance amplifier for making voltage clamp measurements. The voltage V_{clamp} is used to control the membrane voltage, and the channel model is represented by the current source I_{in}. The voltage $V_{out}-V_{in}$ is proportional to the input current, so V_{out} and V_{in} are buffered out to pads (buffers not shown in this figure). There is a 5-bit programmable variable compensation capacitor and an OTA that can provide a range of known currents in order to calibrate the amplifier's characteristic precisely. Small squares are used to denote connections to pads, and the capacitive attenuators shown on two OTAs have about a 10x attenuation factor. 87

44 Step responses in a voltage clamp experiment of a) the potassium channel model and b) the sodium channel model. Note the instantaneous step in the potassium channel's response. This is a result of the fact that the n-well of the potassium conductance is not connected to its source, as well as the fact that the capacitor C_K does not dominate the total capacitance at the net labeled V_K in Figure 42. The response in the sodium amplifier has a very short transient that is an artifact of the voltage clamp circuit. 88

45 Schematic for STDP-enabled synapses. The synapse consists only of the two transistors M_1 and M_2 and the capacitors connected to its gate. The gate voltage V_{gate} is provided by the circuit shown in Figure 47. The current mirrors that level shift the synaptic current, the cascode pFET, and the tunnel and drain control blocks are all shared by all of the synapses that connect to a particular neuron. The cascode device prevents hot electron injection in M_2, while the current mirrors allow the current to come from the voltage supply E_{exc}. The tunnel control and drain control blocks, described later in this chapter in detail, mediate the hot electron injection and tunneling processes that implement the STDP learning rule. 89

46 Circuits for the fixed weight synapses. a) Diagram of the synapse and multiplexer for selecting the polarity of the synapse group (excitatory versus inhibitory). Changing the logic controlling this multiplexer changes the reversal potential of the synapse (E_{exc} for excitatory and E_{inh} for inhibitory). The fact that this choice affects all of the synapses in a particular group was an architectural necessity, but it is exactly analogous to the fact that biological neurons produce only a single type of neurotransmitter at all of their synapses. b) Diagram of the circuit that selects the input for a group of fixed weight synapses. The polarity and input type are independently selectable for each of the 100 groups of fixed weight synapses. 90

47 Gate waveform shaping circuit and measured synaptic current. a) Circuit diagram for spike detector and gate waveform shaping circuit. The fall time of the gate waveform is set by the width of the pulse output from the edge detector (which is programmable by a floating gate voltage), and the rising and falling slopes are set by the voltages V_1 and V_2, which are also set by floating-gate based bias voltage generators. b) Measured output from the gate waveform shaping circuit. c) Measured excitatory post synaptic current. The waveforms in b) and c) were not taken in the same experiment, so the time courses of the two waveforms are not the same. 91

48 Functional block diagram of the AER module on the IC. a) Top-level signal flow depicting the various channels of communication with the microcontroller. b) AER receiver diagram. c) AER transmitter diagram 92

49 Measurements of membrane voltage response to synaptic inputs, illustrating the excitatory threshold for spiking in the neuron model. All waveforms in this figure are averaged over about 20 captures in order to reduce the effect of power supply noise that is present in the measurement system. a) Three input synapses with approximately equal conductance are connected, and the response to various subsets of them are shown. b) The response to synaptic inputs of varying amplitudes is shown (the synaptic weight, normalized to the minimum weight that produced a spike, is shown). 94

50 Measured relationship between synaptic input amplitude and likelihood of the synaptic input stimulating an action potential in the neuron. If the input is sufficiently weak, there is no chance that it can cause an action potential. If it is sufficiently strong, it is certain to cause an action potential. There is a relatively narrow region of input amplitudes where the probability is seen to vary continuously between these two extremes. The family of curves shown here is this "p-I" characteristic that was measured for 6 different neurons. 95

51 Illustration of the paired pulse facilitation phenomenon that is observed in this system. The second pulse arrives before the gate waveform has recovered to its steady-state value, which results in a greater amplitude gate waveform. This makes the second input stronger than the first, and thus it is consistently able to produce a spike. While the second gate waveform pulse appears to clip at about 300mV, this is actually due to the nFET input buffer amplifier that drives this signal off chip. The signal that goes to the synapses can slew clear to ground. These data were averaged for 10 trials to reduce high frequency noise. 96

52 Data illustrating the effects of temporal summation of inputs. In a), the membrane voltage is compared for a single synaptic input (shown in blue) and two successive inputs (shown in green). In b), the probability of eliciting a spike by providing a pair of sequential excitatory inputs is plotted as a function of the timing difference between the two inputs. The two curves shown are from two different trials of the experiment. There is a clear effect of enhanced spike probability for inputs that arrive within about 2 ms, and a suppression of spikes if the inputs arrive between 2 and 6 ms apart. This suppression is the result of the underdamped characteristic of the sodium amplifier. 97

53 Properties of the stable oscillations of the membrane voltage. a) Typical waveform for the stable oscillations. b) Variation of oscillation period with the pFET bias voltage that sets the bias current in the sodium bandpass amplifier. As the bias voltage goes up, the current goes down, which raises the DC operating point of the amplifier. This, in turn raises the high pass corner frequency, which shortens the period of oscillations. At some point, the high pass corner frequency becomes sufficiently high that the rebound from the falling edge of the spike is not sufficient to trigger another spike. At this point, the stable oscillations cease. 98

54 Responses of neuron model to inhibitory input. a) A very small amplitude inhibitory synaptic current can result in a spike if the inhibitory reversal potential is too low. This unwanted behavior can be eliminated by setting the inhibitory reversal potential sufficiently high (to a value of 30-50mV below E_K). b) With E_{inh} set to an appropriate level, the effect of inhibition to prevent a spike can be observed. In this experiment, an excitatory input was delivered a short time after an inhibitory one. If the excitatory input arrives too soon after the inhibitory input (in this case, 0.5 ms), it fails to elicit a spike. If it arrives a little later, after the inhibitory conductance has begun to fall, then it causes a spike (both traces averaged over 10 trials to reduce high frequency noise). 100

55 Waveforms that orchestrate the STDP learning rule. Arrows indicate the timing of the pre and post-synaptic spikes, as well as the fact that the drain pulse and V_{tun} are time-locked to the post-synaptic spike while V_{gate} is time-locked to the pre-synaptic spike. 101

56 Signal flow for pulse timing circuits, including the gate waveform circuits and the circuits that generate the signals for STDP. Note that the voltage V_{gate} is driving the output synapses of the neuron that has just spiked, while the STDP signals (V_{tun} and the drain pulse) are connected to the input synapses of this neuron. The gate waveforms that influence the STDP rule on the input synapses of this neuron are the ones driving those synapses. 102

57 Circuits for timing and shaping a) the drain pulse and b) the tunneling voltage V_{tun}. The voltages V_b and V_{b1} are set by floating-gate transistors, and are used to set the delay between pre-synaptic spike and drain pulse or V_{tun} waveform, respectively. The voltage V_{b2} is also set by a floating-gate transistor, and it sets the fall time of the V_{tun} waveform. The global bias voltage V_{clamp} is used to set the baseline value for V_{tun}. The input V_{in} to these circuits is shared, and is the output of the edge detect block shown in Figure 56. 103

58 Circuit diagrams of a) latch-based one-shot circuit and b) falling edge one-shot circuit. The latch-based one-shot produces a pulse with programmable width in response to a rising edge on its input. The latch-based circuit filters out glitches that accompany its input pulses. The falling edge one-shot produces a pulse with programmable width in response to a falling edge on its input, and it does not provide glitch filtering. 104

59 Measured STDP characteristic, showing dependence of the change in synaptic weight on the timing separating the pre-synaptic and post-synaptic spikes. 104

60 Long-term evolution of synaptic weights when the synapses undergo a) repeated post-pre spike pairings, and b) repeated pre-post spike pairings. in both cases, the synaptic weights are seen to vary over about 4 orders of magnitude. 105

61 Results from mismatch compensation of gate waveform circuits. In the left two panes, the gate waveform outputs from 100 different circuits are shown before and after mismatch compensation. The panes on the right compare the histograms of slopes of these waveforms before and after the mismatch compensation. 107

62 Closing the loop around neuromorphic systems. Clearly, our understanding of neuroscience and the theory of computation influences the designs of neuromorphic systems. Additionally, by using neuromorphic hardware to perform nontrivial neural modeling, we can gain new insights into those foundational sciences. 112

63 Results from "dot matrix" experiment. a) Diagram of the connectivity. The synaptic weights are set sufficiently high that a single input reliably elicits a spike from the target neuron. b) Measured outputs from this circuit. Only the spiking output activity of the neurons is depicted, but the pattern of AER inputs is essentially the same. 113

64 Simulation results for 100-neuron synfire chain. a) Depiction of synfire chain network. For the data shown here there are 100 neurons so N is 98. b) Raster plot of spiking output resulting from two sequential inputs initiated at the beginning of the chain. This results in two simultaneous waves propagating along the chain. c) Raster plot of periodic behavior in the synfire chain when the neuron at the end of the chain is connected back to the one at the beginning. d) measured propagation delays for each neuron in the chain. The mean and +/- 1σ levels are shown by horizontal lines. In b) and c), input spikes are denoted by vertical tick marks, while outputs are denoted by open circles. 114

65 Simulation results for spatiotemporal pattern detector network. a) Diagram showing network connectivity. Brackets indicate repeated units connected sequentially to form a chain. b) Raster plot of activity in the network when the tuned pattern of "ABBA" is given with the appropriate timings of 0.5, 7.5, and 4.5 ms separating successive inputs. Input spikes are denoted by vertical tick marks, while outputs are denoted by open circles. The synfire chain "A" corresponds to neuron numbers 2-42, chain "B" is numbers 43-82, and the output neuron is 89. 116

66 Simulation results for spatiotemporal pattern generator network. a) Diagram showing network connectivity. Brackets indicate repeated units connected sequentially to form a chain, and a "signal bus" notation is used to denote the various connections from the synfire chain to the output neurons. b) Raster plot of activity in the network in response to a single "pattern trigger" spike. Input spikes are denoted by vertical tick marks, while outputs are denoted by open circles. . . 117

67 Results for 6-input WTA network. a) Diagram showing network connectivity for ring winner-take-all. Arrows that terminate in small open circles are used to denote inhibitory connections. b) Raster plot of this network for the "first-to-spike" encoding scheme. In this scheme, the first neuron to spike in each burst of inputs prevents the others from spiking. c) Raster plot of 5-input WTA for the rate encoding scheme. The input rate to all neurons is 20Hz for the first 1.2s, after each of the 5 inputs sequentially receives a burst of stimulation at 320 Hz while the others maintain 20Hz stimulation. In b) and c), input spikes are denoted by vertical tick marks, while outputs are denoted by open circles. 118

68 Simulation results for 30-input ring winner-take-all (WTA) network. a) Raster plot depicting a typical example of activity in the network when the input rate for neuron 15 is 1Hz for 10s, and is then sustained at 240 Hz for 10s. The inputs on the other neurons remain at 1Hz for the duration of the experiment. Input spikes are denoted by vertical tick marks, while outputs are denoted by open circles. b) Plot of average firing rates (over 100 seconds of measured data) of each of the 30 neurons, shown for 6 different input rates on neuron 15 (again, with 1Hz input rate on all other neurons). Note that the vertical axis is set up to approximately align with the axis in a), and that a log scale is used for the horizontal axis in order to simultaneously show the changes in rates of the "winner" neuron and the others. c) Firing rate of the neuron 15 versus the input rate on neuron 15. d) Firing rates of the neurons other than 15 versus the input rate on neuron 15. The population average is plotted with a line. 120

69 Histogram of time since preceding inhibitory input for each excitatory action potential for 30-input WTA. The average input rate on the "winner" neuron is annotated on each strip. The effect of the "winner" neuron's activity to reduce the time between consecutive inhibitory spikes is apparent. Also, the suppression of action potentials during the 20ms after the inhibitory action potential is clearly visible. . . . 122

70 Simulation results for bistable oscillator network. a) Schematic depiction of network connectivity. b) Zoomed in depiction of the end of the data shown in c), to illustrate the timing of the cyclically propagating wave of activity. c) Raster plot showing activity of the network as the inputs drive it back and forth between the stable oscillations and the stable fixed point. In b) and c), input spikes are denoted by vertical tick marks, while outputs are denoted by open circles. 123

71 Context for robotic path planning problem. a) Picture of a maze (or obstacle-filled landscape) that the robot must navigate. The robot starts at location A, and is attempting to get to location C. b) Graphical representation of the immediate neighborhood of the starting point (A) shown in a). Hash marks through the edges connecting B to D and C to D indicate that these transitions are not allowed (the graph will not contain these edges). c) Depiction of a set of neurons with reciprocal connections to represent traversability. 125

72 Illustration of example problem, a) in terms of the geometries of the obstacles, and b) in terms of the synaptic connections among the neurons in the array. 126

73 Illustration of solution to example path planning problem. a) Depiction of the optimal path in the spatial representation. b) Raster plot of the activity in the neurons that make up the path, ordered vertically according to their locations along the path. This way of plotting the activity highlights the wave of activity that ultimately results in the first spike at the goal neurons. c) Analysis of the synaptic delays along the path. The delays are found to be fairly uniform at a little less than 1 ms per synapse. 127

74 Flow chart depicting the two alternative approaches to path planning with the neuromorphic IC. The AER infrastructure can be used together with a sequence of simple searches in the recorded spike data to plan the entire path to the goal. Then, the movement to the goal can be executed, and the process can repeat if there is another goal location. Alternatively, the spike data need not be recorded, and thus no post-processing of the spike data is required. This approach only provides a plan for the next move, so the robot executes this move and then re-plans. 128

75 Use of variable synaptic delays to represent a cost function associated with different locations. In this maze, every location that borders an obstacle is assigned an increased cost (representing the desire to leave margin around obstacles). In a), the additional cost is low, and the planner charts a course that passes near all of the obstacles, but minimizes the path length. In b), the additional cost is increased sufficiently that the planner chooses the longer path that cuts a wide swath around the obstacles. 130

76 Two-synapse model for investigating the effect of STDP. The setup of the experiment and the resulting change in synaptic weights are shown in a). The horizontal axis in a) is equivalent to post-synaptic minus pre-synaptic spike time. In b), the evolution of the synaptic weights with repeated potentiations is shown for a variety of different injection voltages. In both plots, the synaptic strength is measured by the voltage produced by forcing the current through a series of two diode-connected nFETs. The subthreshold model $\frac{I}{I_0} = exp(\frac{0.7}{2}\frac{\triangle V}{0.025})$ can be used to convert changes in diode voltage into relative changes in synaptic current. 137

77 Potentiation of synchronous inputs. a) shows the network under consideration, in which multiple synapses all drive a single neuron. b) Raster plots from the three phases of the experiment. In the top pane, the response of the neuron in the initial state is shown, in which each input is activated individually and they all fail to elicit a post-synaptic spike. In the center pane, the training process is depicted, in which all three synapses are repeatedly activated simultaneously, each time eliciting a spike in the target neuron. In the bottom pane, the result of strengthening the synapses is demonstrated by presenting the same stimulus that was used in the initial state, but this time each individual synapse is able to elicit a post-synaptic spike. 139

78 Competitive learning among 10 synapses on the same neuron. Nine of the synapses receive rate-coded inputs at a low rate, while one receives a high rate of inputs. The inputs and resulting spiking behavior are shown in a). As the input with the elevated firing rate gets potentiated, the firing rate of the target neuron increases. This synapse attains a strength that is sufficient to always cause a post-synaptic spike, which results in steady potentiation. Eventually, this leads to the synapse becoming so strong that it clamps the membrane of the postsynaptic neuron to E_{exc} (the excitatory reversal potential), preventing any further spikes. b) Histogram of changes in synaptic weights pooled across 10 experiments of the type shown in a). The distribution is clearly separated into neurons that experienced a positive feedback process of potentiation and neurons that experienced a moderate depression. The inset shows the evolution of the synaptic weights during the experiment. 141

79 Results of experiment where synchronized inputs among channels 0, 5, and 9 were sparsely mixed with random, uncorrelated inputs. All other synapses received only random uncorrelated inputs such that the average rates on all inputs was the same. The synapses that had correlated activity were potentiated while the others were depressed. 143

80 Simplistic model for initial studies of receptive field formation early in the visual pathway. A single neuron receives 49 input synapses, each associated with a pixel in a 7x7 monochromatic image. The image shown in a) is repeatedly presented to the network, using a "time to spike" encoding of the pixel brightness on each synapse. The evolution of the synaptic weights over repeated presentations of the input is shown in b). The blue traces, which represent the synapses on the bright half of the image, tend to get potentiated, while the others tend to be depressed. 144

SUMMARY

The goal of neuromorphic engineering is to create electronic systems that model the behavior of biological neural systems. Neuromorphic systems can leverage a combination of analog and digital circuit design techniques to enable computational modeling, with orders of magnitude of reduction in size, weight, and power consumption compared to the traditional modeling approach based upon numerical integration. These benefits of neuromorphic modeling have the potential to facilitate neural modeling in resource-constrained research environments. Moreover, they will make it practical to use neural computation in the design of intelligent machines, including portable, battery-powered, and energy harvesting applications. Floating-gate transistor technology is a powerful tool for neuromorphic engineering because it allows dense implementation of synapses with nonvolatile storage of synaptic weights, cancellation of process mismatch, and reconfigurable system design.

A novel neuromorphic hardware system, featuring compact and efficient channel-based model neurons and floating-gate transistor synapses, was developed. This system was used to model a variety of network topologies with up to 100 neurons. The networks were shown to possess computational capabilities such as spatio-temporal pattern generation and recognition, winner-take-all competition, bistable activity implementing a "volatile memory", and wavefront-based robotic path planning. Some canonical features of synaptic plasticity, such as potentiation of high frequency inputs and potentiation of correlated inputs in the presence of uncorrelated noise, were demonstrated. Preliminary results regarding formation of receptive fields were obtained. Several advances in enabling technologies, including methods for floating-gate transistor array programming, and the creation of a reconfigurable system for

studying adaptation in floating-gate transistor circuits, were made.

Chapter I

THE BIG PICTURE

1.1 Goals: neural and analog computing

1.1.1 Why study neural computing?

When I received a basic introduction to neuroscience as a junior in high school, I enthusiastically came to the conclusion that all human behavior is a result of brain activity, which we can understand mechanistically at the neuronal level. Thus, all of the mysteries of the human mind - our ability to reason, to communicate using language, to experience emotions, the variety of different personality types, our capacity for artistic expression and intellectual discovery, our seemingly boundless capability to learn new skills and information, and even consciousness, that elusive and intangible phenomenon that lies at the center of our very identities - seemed to become accessible through the study of neuroscience. In these speculations, I was following in the footsteps of many scientists and philosophers throughout history. The first group of such thinkers with the tools, knowledge, and ambition to attempt to really test hypotheses about these principles by creating engineered systems with the capabilities of the brain was the artificial intelligence (AI) community, which kicked off these efforts in the mid 20th century. Many of the founders of this field expected the lofty goal of designing a human-like intelligence to be solved relatively quickly. For example, Marvin Minsky famously published in 1967 that "within a generation..., the problem of creating artificial intelligence will substantially be solved". While the early efforts in AI led to innovations that are still influential today, they fell far short of these high expectations.

The failure of AI to fulfill these ambitious expectations was partly due to a failure

to fully appreciate the difficulty of basic problems of perception and motor control, which seem so easy because we generally perform them without conscious effort. Tasks such as recognizing spoken words and picking faces out of an image have proven to be difficult, and only recently (after several decades of development of computing hardware and algorithms) has the engineering community begun to produce somewhat reliable solutions for them. In the motor control domain, the robust control of bipedal locomotion (or to the layman, "walking"), continues to be a topic of cutting-edge research. While decades of diligent research has produced some impressive and promising results for perception and motor control tasks, our solutions to these "low-level" problems still have significant drawbacks when compared to their neural counterparts. Engineered solutions for perception and motor control often perform more poorly (compared to neural systems) in noisy environments or across wide variations in signal dynamic range. Moreover, the important system metrics of size, weight, and power are generally orders of magnitude higher in engineered systems (consider, for example, the real-time processing for avionics, image processing, and path planning that is performed by the brain of a fly). So what about our progress on the more "high-level" cognitive functions of the brains such as language processing, reasoning and planning, social interactions, and learning? Research in all of these topics is ongoing, but our understanding of the neural mechanisms that support these features of intelligence is poor.

Clearly then, the goal of mechanistically understanding brain function is far from completed, and will require a coordinated and sustained effort from the scientific and engineering communities. So is it worth the effort? What is there to gain by reaching this goal? As with most questions about the motivation for scientific inquiry, there are two distinct philosophies. The first is a practical one: if the mechanisms underlying brain function can be understood well enough to mimic in engineered systems, then one can envision a variety of immediate applications for intelligent

machines in the consumer, medical, manufacturing, and military markets. Also this understanding of brain function could inform more effective treatment and prevention of pathologies such as epilepsy, Parkinson's disease, and Alzheimer's disease, and could allow for prosthetic devices that interface effectively with the nervous system. The second answer is purely aesthetic (in my experience, for the scientists who passionately commit their lives to research, this motivation is often at least as strong as the practical one): convincing ourselves that we understand the mysteries of the human mind/brain would be an immensely satisfying achievement from a philosophical/intellectual standpoint. This intense drive to understand the workings of our world seems to be fundamental to the human psyche, and history has shown that individuals following this drive often unwittingly contribute mightily to the march of technological progress.

1.1.2 The importance of computational modeling

In the pursuit of understanding brain function, the "ground truth" for evaluating hypotheses must be measurements made on brains. The modern methods of experimental neuroscience provide an impressive array of ways to make such measurements, including electrophysiology on tissue cultures or animal models, anatomical studies, manipulations using molecular genetics, and both structural and functional imaging techniques. As important as these experimental studies are, computational modeling is equally important for several reasons. Firstly, networks of neurons have sufficiently complicated dynamics that it is generally difficult for the unaided intellect to fully appreciate the consequences of a set of assumptions. For example, a scientist may propose a guess as to the network topology that is present in a given neural circuit, but may not be able to tell whether that network topology will actually produce the patterns of activity that are observed in experiments on that circuit. It is simply too much information to picture in one's head without the aid of a computational

tool. A second advantage offered by computational modeling is that it can be used to expand the "field of view" of experimental studies. Specifically, experiments are severely limited with regard to how many of the system's degrees of freedom are observable, as well as what types of manipulations can be made. In contrast, it is feasible in computational studies to track any dynamical variable that is of interest, and to simulate an identical experiment multiple times, systematically varying any arbitrary aspect of the system (e.g. synaptic conductance, dendrite shape, distribution of ion channels, network topology, etc). Both of these advantages ultimately lead to making better hypotheses for experimental measurements. This means that the costly, time-consuming, and difficult experimental work can be done as efficiently as possible.

One final advantage to performing computational modeling studies is that a model that successfully demonstrates an aspect of brain function such as word recognition or face recognition also serves as a "proof of concept" for an implementation of this functionality in an engineered system. Thus, one good approach to computational modeling is to hypothesize principles of neural computation that subserve a particular function (such as face recognition), and then construct networks from those principles and evaluate their performance on the function of interest. In the best case, the principles that result in good performance on the applications will provide successful hypotheses for predicting outcomes in neuroscience experiments, in which case it is reasonable to claim to have gained insight into the scientific question of how brains work. In the worst case, those principles will generate hypotheses that will prove inconsistent with neuroscience experiments. Even in this case, a viable guess as to how the brain functions has been ruled out, which also constitutes scientific progress. Moreover, in both cases, useful knowledge for designing intelligent machines has been generated.

Given the roles of computational modeling of neural systems outlined above, it is

clear that speed, ease of use, and ease of technology transfer to applications are key metrics of any approach for such modeling efforts. Speed and ease of use are important so that scientists can rapidly perform many modeling studies without being hampered by run time or long learning curves of the computational tools. Ease of technology transfer becomes important as soon as a computational principle is discovered that could be useful when applied to a "real-world" problem. If the nature of the modeling approach is such that it easily translates into a product, then technology benefits from the discovery in addition to pure science. In language used earlier in this section, such translation results in a practical payoff from the research effort in addition to the aesthetic payoff.

1.1.3 Computational modeling by numerical integration

The most popular method for performing computational modeling in neuroscience is numerical integration of model equations using digital computers. This method offers the modeler an unfavorable trade-off between speed, ease of use, and ease of technology transfer. The most easy-to-use approach is offered by the personal computer. On a personal computer, a real-time simulation is limited to somewhere between 1 and a few hundred neurons, and between 100 and a few thousand synapses, depending on the level of detail in the models, the level of activity in the network, and the software implementation details. The high end of this range is achievable only by using the most simplistic models and by extensively optimizing the simulator. The personal computer approach is fairly easy to use. Its size, weight, and cost are not prohibitive for translation to high-end applications (such as medical or military), but may be prohibitive for consumer applications. It requires a few hundred watts of power, which rules out many battery powered or energy harvesting systems as possible application domains. If the personal computer implementation was done without much optimization of the software or hardware, a clever engineer may cut

as much as a factor of 10 from the size, weight, and power metrics in the process of translation.

The speed of the computation may be increased by using a cluster instead of a single personal computer. In this case, the ease of use generally decreases substantially due to the difficulty of writing code that efficiently takes advantage of parallel processors (unless the problem can be partitioned into independent tasks for each processor, as in a parameter sweep). The speed is increased by a factor that is generally somewhat less than the number of processors in the cluster (how much less depends on the software and hardware implementations). The cost becomes prohibitive for translation to anything other than low volume, high end applications. The power increases by at least the number of processors, and the size and weight increase as well. This type of implementation is not suitable for most robotics applications due to size, weight, and power constraints.

The fastest performance for numerical integration-based neural simulation is achieved by the large-scale supercomputer. An example of such an approach was recently published by IBM, who used their Blue Gene supercomputer, featuring 147,456 parallel processors, to perform a simulation of a network of 1 billion single-compartment integrate and fire neurons and 10 trillion learning synapses at 1/100th real time [3]. This simulation required 1.4MW of power, and the hardware that it requires fills a room. Clearly such a system is the opposite of easy to use and is virtually useless for translation to applications. The cost of such a system is such that only the wealthiest entities such as major corporations and governments can afford to own one.

In the recent past, some neuroscientists expected the trade-off presented by the method of numerical integration to rapidly become more favorable as a result of continued improvements in digital computing technology. For instance, in 2005 Eugene Izhikevich performed a simulation of 10 billion point neurons and 100 trillion (non-learning) synapses [39]. He performed the simulation on a cluster of 27 processors,

which simulated 1 second of model time in 50 days. At that time, he published a projection of how this performance would improve over time. The premise of the projection was simple; he assumed that processor clock speed will double every 18 months for at least the next 40 years (which was as far as the projection went), and that the simulation time is inversely proportional to the product of clock speed and number of processors. Comparing the clock speeds of typical processors on the market today (3-4 GHz) to the projected clock speed for this year (48 GHz) shows that even just a few years after the projection, it is obvious that this projection was far too optimistic. In fact, the trend for clock speed seems to be much more accurately modeled as a flat line than as a rapidly growing exponential. While the clock speed is more or less holding steady, it is true that the number of transistors on a chip is generally continuing to increase, especially as processor manufacturers increase the number of cores that are integrated into a single chip. It is reasonable to expect that this will increase the speed and ease of use for the user of the desktop system, however a fundamental limitation remains. The increases in computing power are coming at the cost of increases in power consumption, as observed in [45]. This constitutes a major obstacle for translation of any large-scale neural computing algorithm to any portable, battery-powered, or energy harvesting application. This key insight about the fundamental limitations of the numerical integration approach was foreseen over 20 years ago by Carver Mead [46].

1.1.4 The neuromorphic approach

Having acknowledged the limitations of computational modeling that is based on numerical integration, the obvious question is whether a viable alternative exists. The answer to this involves a change in thinking about what it means do computing. Consider the example of aircraft design. The equations governing fluid flow around an aircraft in flight are difficult to solve accurately, leading to high costs in terms of

time, money, and infrastructure for solutions based on digital computing. However, aerospace engineers can place scale models of aircraft in a wind tunnel, and can then experimentally ascertain the behavior of a given aircraft under a variety of conditions. One interesting way to think about this approach is that the fluids in the wind tunnel are being used as a computer to calculate solutions to the relevant equations of fluid mechanics, which they do naturally by obeying those equations. Once the physical apparatus for the experiment is constructed, this computation is performed quickly and efficiently. Of course, this computer is highly specialized for computing fluid flows. It is useless for solving most of the problems that we typically perform on digital computers.

The goal of neuromorphic engineering in general, and this work in particular, is to create just such a specialized tool for computing the dynamical behavior of neurons. Using the powerful technology of integrated circuit (IC) fabrication, vast numbers of circuits that are governed by equations similar to those that describe neurons and synapses can be constructed on a tiny chip. On the one hand, this approach can be thought of as an experimental platform for network electrophysiology that affords a level of observability and control well beyond what is achievable with experiments on tissues. On the other hand, it can be regarded as a new type of computer, one that is specialized for the task of simulating networks of neurons, and is therefore highly efficient at this task. From the latter perspective, if the model circuits can be implemented using low-power design techniques, and can be densely integrated into single IC, then the elegance of this approach will translate to tremendous improvements in the metrics that determine ease (or even feasibility) of translation to applications. In short, I believe that this approach can yield tools that will profoundly push forward the study of neural computation by making fast, easy-to-use computational models of neural networks widely available, and will also provide a practical technology for implementing principles of neural computation in

intelligent machines.

The idea of making circuits that perform as "scale models" of networks of neurons has been around since the early days of AI, with Frank Rosenblatt's "Mark 1" and "Tobermory" being perhaps the earliest notable examples (1960 and 1962)[47]. Early approaches such as this did not model the spiking behavior of neurons, so they constitute a coarser approximation than approaches that employ realistic models of the excitable dynamics of the individual cells. These two machines in particular were built from thousands of discrete devices, so they were physically large and their construction required an extraordinary effort of meticulous manual labor. Beginning in the early 1980s, Carver Mead resumed these efforts, this time leveraging the rapidly-developing "very large scale integration" (VLSI) technology by which ICs are designed and fabricated, as well as employing a modeling approach that sought to more faithfully capture the dynamics of biological neurons. His efforts created the modern field of neuromorphic engineering. Research in neuromorphic engineering has continued from that time, and it is continuing to grow (in part due to funding from large projects such as SyNAPSE and FACETS), but because the community is so much smaller than that of the multi-billion dollar industry for making digital computers, the neuromorphic technology is at an earlier stage of development than digital computing technology. Still, because the elegance of the neuromorphic approach will lead to large advantages in the important metrics of size, weight, and power consumption, the long-term outlook appears to be great for neuromorphic engineering.

1.1.5 Analog computing

As demonstrated by some of the successes of Rosenblatt's models and their descendents (which includes multi-layer perceptrons), effective approaches to neural-inspired computing can share some high level features with neural systems without accurately

modeling the membrane dynamics of the individual neurons. Specifically, where analog circuits can be designed to implement a particular function in an elegant way, significant savings of power are possible. One example is the 1000x efficiency improvement for vector-matrix multiplication shown in [16]. An analysis based on just the fundamental tradeoffs encountered in circuit design concludes that systems with relatively low signal to noise ratio (SNR) requirements will be more efficiently implemented using analog representations than digital ones [55].

One class of tools that has been created for exploring the possibilities of analog computation is the field programmable analog array (FPAA). As the name suggests, the FPAA is essentially the analog equivalent of the field programmable gate array (FPGA). FPAAs have a variety of analog circuits and devices, interconnected by a configurable switch matrix, which allows the synthesis of many circuits and systems using only a single chip. This allows for exploration of novel circuits and applications for analog computing (or synonymously, analog signal processing). A large number of applications have been demonstrated on FPAAs [49, 50, 59, 58, 66].

1.2 Methods: VLSI and floating-gate transistor technology

For both neuromorphic engineering and analog computing, the most important enabling technology is VLSI IC fabrication, a technology that allows billions of electronic circuit elements to be wired together on a single silicon chip. Employing this technology allow designers to leverage computer aided design tools and sophisticated manufacturing technology to make systems of a complexity that would be unimaginable if the manual design and assembly techniques of the past were relied upon. In particular, the IC technology of choice for this work is the "complementary metal oxide semiconductor" (CMOS) process, wherein most of the circuit elements on the chip are "metal oxide semiconductor field effect transistors" (MOSFETs). CMOS is an excellent technology for this application for many reasons. Firstly, the charge

transport in MOSFETs is similar to the transport across the cell membrane through ion channels, which determines the electrical behavior of neurons. Secondly, CMOS is a very mature technology from a manufacturing and a circuit design standpoint. As a result, there is a large "toolbox" of canonical circuit design techniques available in CMOS, and the manufacture of newly-designed chips can be performed very reliably. Furthermore, if the volume of production of the chips increases, it is possible to obtain significant economies of scale.

Another advantage of using CMOS technology is that it enables the use of a type of MOSFET called a floating-gate transistor. This is a standard MOSFET except that its gate terminal is electrically isolated (or floating), and the effective gate terminal of the floating-gate device is a capacitor that couples electrostatically to the gate of the MOSFET. The voltage on the floating gate is thus determined by the voltage on the control gate terminal as well as the charge on the isolated gate node. This charge stays fixed unless the device is subjected to conditions that cause tunneling or hot carrier injection, which are the processes used to manipulate the charge on these devices. As a result, the device is simultaneously a standard MOSFET and a device for stable storage of an analog nonvolatile memory. The programmable parameters of the devices are useful for combating the pesky problem of device mismatch, storing synaptic weights of model synapses, and efficiently creating circuits that can be configured in a variety of different ways. Additionally there is a class of useful building blocks for analog circuit design that rely on this degree of freedom.

When a negative overdrive voltage is applied to the gate of a MOSFET, it operates in a regime called subthreshold, in which the transport is dominated by diffusion of minority carriers, much like in a bipolar junction transistor (BJT). A subthreshold MOSFET, like a BJT, has an exponential I-V relationship. Designing circuits for

operation in this regime gives access to many orders of magnitude of current, including the very low currents that enable ultra-low power performance. Furthermore, the allowable voltage swing is maximized by subthreshold MOSFETs, which allows them to operate well with very low voltage supply rails. This is another significant advantage for low power design. A final advantage of subthreshold operation is that the exponential form is a useful building block for implementing many computations. For instance, a shift in gate voltage represents a multiplication in current, which is a principle that can be applied to build a highly efficient analog vector-matrix multiplier using a matrix of floating-gate transistors operating in subthreshold.

The final element that plays a central role in this work is the analog floating-gate array. This is simply a large group (tens or hundreds of thousands) of floating gate transistors on a single chip, in addition to addressing infrastructure and other circuitry that allows for selective programming of each floating gate transistor on the array. Floating-gate arrays allow an elegant implementation of the FPAA systems described in Section 1.1.5. They reduce the parasitics of the switch devices in the switch matrices, and they allow the switch matrix to function as a useful part of the circuit (a bias generator or a vector-matrix multiplier, for instance). The infrastructure for the array also makes it easy to leverage the benefits of floating gates in the design of the circuits outside of the switch matrix. In the context of a neuromorphic IC, a floating-gate transistor array can be used as a very dense implementation of synapses, which have permanent local storage of synaptic weights, and are capable of implementing synaptic plasticity.

1.3 Achievements: from devices to systems

A universal truth of modern science and engineering is that we can only see farther by standing on the shoulders of giants. The work that I present here is no exception. I did not come up with the idea of neuromorphic engineering or analog computing. Nor

did I make more than a few minor contributions in the form of basic circuit blocks in the systems presented here. I did, however, make substantial contributions to the field, including improving the understanding of the behavior of the floating-gate devices, creating tools (software, hardware, and algorithms) that allow effective use of ICs that feature floating-gate arrays, designing complex and sophisticated systems for neuromorphic engineering and analog computation, and exploring the computation that can be achieved with the use of these novel tools.

In the remainder of this document, I will describe this work in further detail. Chapter 2 covers the software, hardware, and algorithms that I developed in order to support the use of systems with floating-gate transistor arrays. Chapter 3 describes new findings about the characteristics of the floating-gate transistors with regard to the tunneling processes that are used for programming. It also includes analysis of optimal techniques for the use of tunneling in the setting of arrays of floating gates. The work described in Chapters 2 and 3 forms a technical foundation, without which the system results in the later chapters would not have been possible. Chapter 4 describes my design of a novel type of FPAA, one that enables floating-gate transistors to undergo injection and tunneling while processing signals, thus supporting a class of adaptive circuits with a rich and interesting extra dimension of dynamics. Chapter 5 presents the full details of a neuromorphic IC that I designed. It also shows measured results from the system illustrating the operation of the circuits. In Chapter 6, this neuromorphic IC is put to work as a simulation engine for a variety of networks. Networks are demonstrated to perform arbitrary spatio-temporal pattern generation and recognition, "winner-take-all" competition, bistable oscillations capable of acting as a "volatile memory", and optimal path planning. Chapter 7 describes the results from simulations of networks in which the synapses learn by spike-timing dependent plasticity (STDP). Finally, Chapter 8 offers some concluding remarks, as well as my thoughts about the future of this line of research.

Chapter II

BUILDING BLOCKS FOR FLOATING-GATE SYSTEMS

Experimental measurements of ICs can require sophisticated infrastructure in the form of hardware and software for testing. This is especially true when the IC of interest is a very complex chip with many inputs and outputs. If the IC contains analog floating-gate transistors, this adds an extra element of complexity, since testing generally requires programming of the floating gates. This chapter discusses the design of hardware, software, and algorithms to facilitate computation on neuromorphic and analog computing ICs that contain floating-gate arrays. The results are demonstrated by showing some measured data from the FPAA described in detail in [7].

2.1 Hardware platform

The hardware for programming and testing floating-gate array based ICs consists of a printed circuit board (PCB), depicted in Figure 1. The board has a pin grid array (PGA) socket, which is used to interface to the floating-gate array IC. The user can connect the board to the USB port on a desktop computer, and then perform all programming and testing operations from a convenient MATLAB interface. The board can be receive power over the USB connection, so it does not require an additional power supply. From the 5V supplied by the USB, 12V and 3.3V supplies are derived.

The programming and testing operations involve setting and reading voltages on the pins of the floating-gate array. These operations are performed by a 40-channel digital-to-analog converter (DAC) and an 8-channel analog-to-digital converter (ADC), the latter of which is embedded in the AT91SAM7S ARM core microcontroller on the board. The microcontroller controls the setting of the DAC channels, the reading of the ADC, and setting and reading of digital input/output (I/O) lines

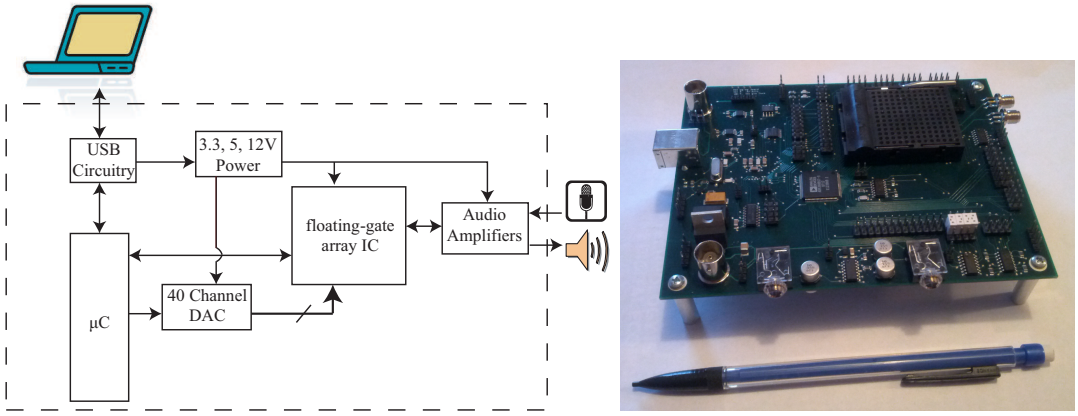


Figure 1: Printed circuit board for programming and testing floating-gate array ICs. The block diagram is shown in a), and a photograph of the board is shown in b), with a pencil to show the scale.

that interface to the IC. The DAC is powered from the 3.3V supply, so op-amp circuits on the 12V supply are used to amplify the DAC outputs where high voltages are needed (for instance, for floating-gate injection and tunneling). In addition, there are op-amp circuits available for buffering weak signals from the IC. There are also audio amplifiers and audio jacks for easy interfacing to audio equipment. All of the functions that the microcontroller performs can be controlled by the user from the MATLAB environment.

The board is an improvement on a previous design by Scott Koziol, Craig Schlottmann, and Csaba Petre [41]. In comparison with the previous design, it has significantly reduced power supply noise and cross-coupling among signal traces, increased available voltage ranges, and improved ease of use for applications requiring external voltage supplies. The improved noise performance of this board, as compared to the previous design, is shown in Figures 2 and 3.

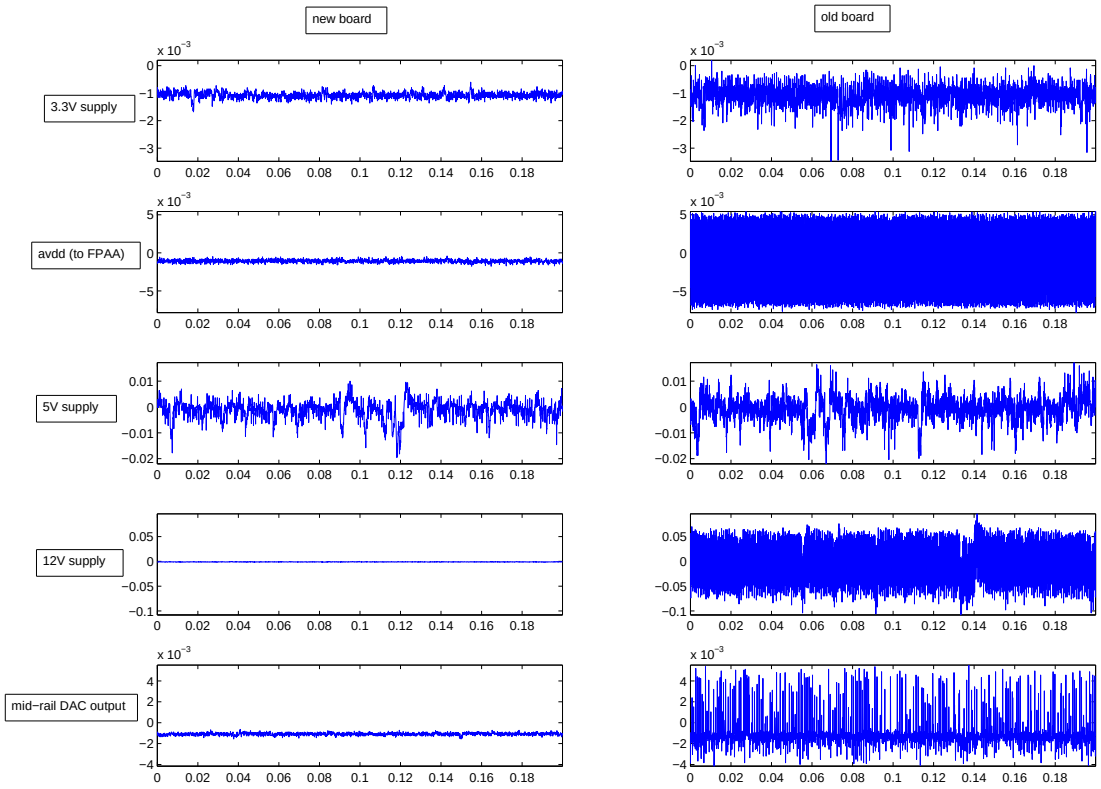


Figure 2: Comparison of voltage supplies and DAC output on new PCB versus the previous version. All of the signals except for the 5V supply are dramatically improved.

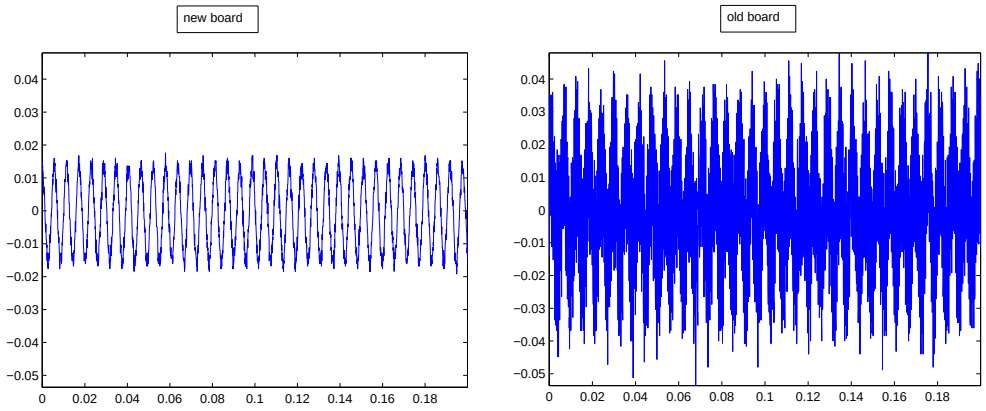


Figure 3: Comparison of output of audio circuit on new PCB versus previous version. In both measurements, the input to the amplifier is a 30mV sine wave. The substantial improvement in the performance for the new board is due to the greatly reduced noise on the 12V supply, which powers the audio amplifiers.

2.2 Fast and flexible software interface

2.2.1 Communication speed and floating gate programming times

The speed of communication between the personal computer the microcontroller is important for determining the speed of floating gate programming operations. This makes communication speed a very important metric for the user of this system. A significant boost in communication was obtained by switching the communication protocol from RS-232 to USB. In the first system that was implemented, commands to set DAC channels and read ADC voltages were sent to the microcontroller and executed one at a time. Since each command requires only a few bytes of data, and the typical latency for sending data is 10-70ms, the communication speed in this framework was severely limited by the latency (at a baud rate of 12 Mbaud, clocking a few bytes requires only a few microseconds).

The speed of the interface was greatly increased by the implementation of a queue, whereby the MATLAB interface accumulates many commands before sending them, thus reducing the penalty incurred from the communication latency. This framework yielded one other advantage. It provided the user the ability to execute sequences of commands with timing accuracy on the order of a microsecond, which is orders of magnitude better than in the non-queuing system. These improvements had a huge impact on the user's experience with this system. They reduced the time required to program an analog floating gate transistor from 5-10 minutes to a few seconds.

2.2.2 Flexible framework to support multiple hardware systems

The endeavor of researching floating-gate transistor array technology and its applications produced a large variety of hardware systems. My collaborators and I have found occasion to work with 4 different custom PCB systems and about a dozen different ICs. Many of the ICs have multiple modes of functionality, which can affect the way that floating gate programming is done on the chips. It is a non-trivial software

design problem to write code that is compatible for all different combinations of these sources of hardware system variation. In the absence of a systematic approach to this problem, there is a tendency for software to proliferate, as each person writes code that is specific to his/her application. This causes significant duplication of effort, and perhaps more importantly it reduces interoperability of different researchers' systems, raising the barrier for collaboration.

A software framework to address and avoid this situation was created. The concept of the framework is simple. Code is written for programming floating gates in a way that does not refer to specific signals on the IC or PCB. Separate files are written to describe the various ICs and PCBs. The code that programs the floating gates refers to data structures defined in these files in order to determine which signals need to be set and read in the course of programming. This approach provides a logical organization for the various degrees of freedom encountered in the hardware systems, and allows users to change aspects of their hardware systems with minimal changes to the software.

2.2.3 Specialized functions for data acquisition

The microcontroller on the PCB has architectural features and embedded peripherals that allow the creation of arbitrary waveforms and the capture of traces of data, thus providing a multi-channel arbitrary waveform generator and a rudimentary oscilloscope. Software support for these functions, including easy-to-use MATLAB interfaces, were created. These functions effectively integrate the instruments that make up a simple bench-top testing setup into a single small, lightweight, and portable PCB (described in Section 2.1).

2.3 Approach for floating-gate transistor programming

In order to reap all of the benefits of using analog floating gate transistors described earlier in Section 1.2, one must have an effective method for programming the charge

on the floating gates. This is normally done by means of two physical processes for manipulating the floating-gate charge: channel hot electron injection and Fowler-Nordheim tunneling. Getting the desired results from these two processes is a non-trivial task. The use of the hot electron injection process for floating gate programming has been the topic of many studies [6, 29, 61, 62]. However, this previous work has treated the matter of the floating-gate voltage during injection in an incomplete way. This observation yields some insights about the characteristics of the hot electron injection process and the reasoning for choosing parameters to control it.

2.3.1 Modeling hot electron injection

As a quick review, the circuit diagram of the typical floating gate pFET cell is depicted in Figure 4 (floating gate pFETs are much more commonly used than floating gate nFETs, for reasons that will be discussed in Chapter 4). If current flows through the channel of the device and the voltage drop from the channel to the drain is sufficiently high, impact ionization occurs in the drain region, creating hot electrons that can have sufficient energy to cross the gate oxide, thus making the net charge on the floating gate more negative. It is important to note here that this process may only decrease the floating gate charge, and that in most analog floating gate array architectures, the only operation for increasing the floating gate charge is a tunneling operation which is global, *i.e.* it affects many floating gate devices simultaneously. The consequence of this is that when many devices are to be targeted for hot electron injection, "overshooting" on a single one (*i.e.* injecting too many hot electrons) would have the undesirable result of needing to perform a tunneling operation and then redo the injection of all of the devices that were tunneled.

With that review out of the way, the study of hot electron injection begins with a characterization of the injection current as a function of the terminal voltages of the

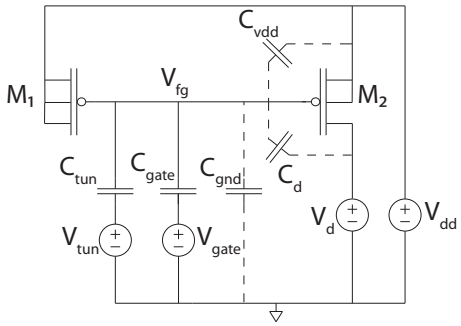


Figure 4: Schematic of floating gate cell. Capacitors drawn with dashed lines represent parasitic capacitances.

transistor. Physically, the injection current depends on both the channel current and the voltage drop from the channel to the drain. In a device with the well connected to the source as in Figure 4, these two quantities are uniquely determined by two voltage differences: V_{sd} (the source voltage minus the drain voltage) and V_{sg} (the source voltage minus the floating gate voltage). Thus it seems natural to measure the device characteristics for a range of these two voltage differences. The result is the family of curves shown in Figure 5. This characteristic completely specifies the dependence of injection on the terminal voltages.

Note that if the terminal voltages of the device are held fixed and hot electron injection occurs, the floating gate voltage will decrease, which constitutes rightward motion along the curve in Figure 5. To the left of the maximum, the slope of the curve is positive, so as injection proceeds, this rightward motion accelerates the injection process in a positive feedback loop. To the right of the maximum, the rightward motion decreases the rate of injection so the floating gate voltage gradually approaches an asymptote. Interestingly enough, while the previous studies of hot electron injection have described various aspects of this behavior, none of them have displayed the injection characteristic as a function of these two quantities (V_{sg} and V_{sd}).

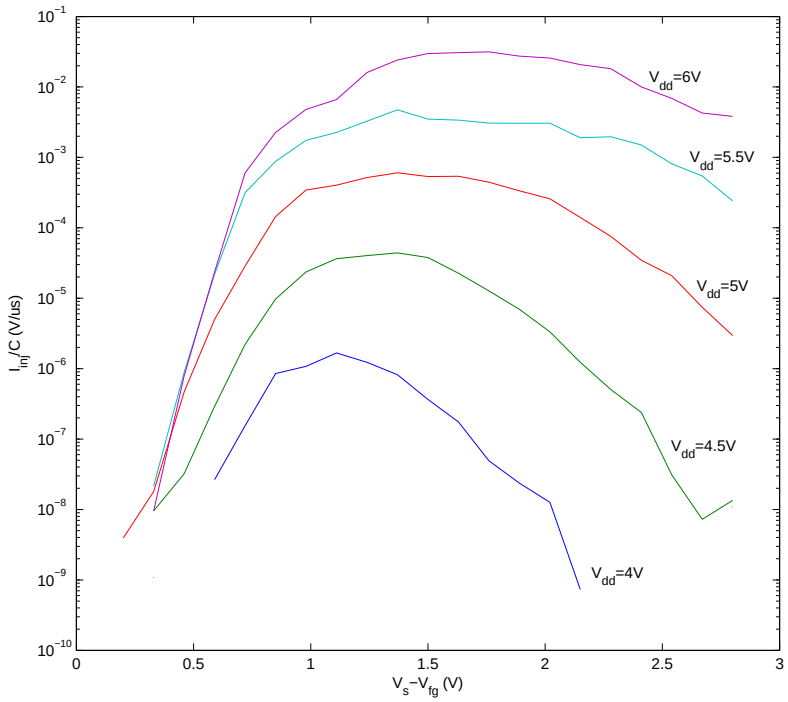


Figure 5: Measured characteristics of hot electron injection for floating-gate pFET. In this measurement, the drain voltage is always 0 so V_{dd}=V_{sd}. The current is normalized by the total capacitance at the floating gate, which was not known, and is generally a difficult parameter to measure experimentally.

2.3.2 Reasoning for injection algorithm

Having a clear picture of the device characteristics is a useful start for developing an approach for using injection to achieve a particular state of floating gate charge. Several observations create the context for this problem. Firstly, the floating gate voltage is not directly observable. It is generally inferred from a measurement of the channel current of the transistor (in conjunction with a model for the I-V characteristic- the EKV model [23] was used in this work). Secondly, the supply voltages required for injection are typically higher than the voltages that the transistors are operated at, in which case the voltage V_{dd} is increased in order to perform injection. When programming accuracy is important, the mismatch in device parameters (parasitic capacitances and Early voltages) among the floating gate transistors in an array is large enough that a current measurement made in the elevated V_{dd} condition provides an inaccurate estimate of the current that would be measured in the regular "run mode" condition. For this reason, the injection is performed using an iterative algorithm, which consists of alternating steps of raising V_{dd} to perform injection, then lowering V_{dd} to measure the channel current and infer V_{fg}. This iteration continues until the target value of V_{fg} is reached, but as mentioned earlier, it must be designed to minimize the occurrence of "overshooting".

With this context in mind, the logic in the algorithm boils down to three questions, which must be answered on each iteration: 1) how much change in V_{fg} should the pulse attempt to achieve?, 2) how much time should be used to achieve this change in V_{fg}?, and 3) what terminal voltages should be used to achieve the injection current that corresponds to the desired change in V_{fg} and pulse time?

The factors to be considered in order to answer these questions are as follows. A naive answer to question 1 would be that the change in V_{fg} should be the difference between the current V_{fg} and the target V_{fg}. There are two possible problems with this. Firstly, this answer leaves zero margin for error. It is highly advantageous to allow

some room for error, because error tolerance in the algorithm can significantly reduce the burden of device characterization (for instance, it could allow one to characterize only a single device on the chip and use its characteristic as an approximation for the other devices' characteristics, knowing full well that this introduces some error due to process variation). Thus, a better choice for the change in V_{fg} might be some fraction (e.g. 1/2 or 2/3) of the difference between the current V_{fg} and the target V_{fg}. The second potential problem is that if the desired change in V_{fg} is large enough that the injection current will change significantly over the course of the pulse, it will be difficult to predict the relationship between injection time and the change in V_{fg}. For this reason, it is a good idea to limit the target change in V_{fg} some 200-500mV.

The question about how much time should be used for the pulse seems easy enough to answer. Speed is a desirable characteristic for the algorithm, so the time should be as small as possible. This is actually a good answer, with a couple caveats. "As small as possible" should be such that the system timing the injection pulse has acceptably low jitter (how much jitter is acceptable depends on how aggressively the target change in V_{fg} was chosen and how large the other sources of error are). Also, there are typically constraints on the allowable terminal voltages for the devices, and within these constraints there is some global maximum of injection current. Clearly the time needs to be at least enough to bring about the desired change in V_{fg} at the maximum achievable injection current. In summary, the time should be minimized subject to the constraints on timing accuracy and injection current. Once a time is selected, this specifies a corresponding injection current.

The question about choosing terminal voltages is where the insight from the device characterization in Figure 5 pays off. There is a clear advantage to starting the injection pulse with the floating gate voltage at or near the peak of the injection characteristic. Since the first order variation in the injection current with floating gate voltage is zero, this is the point on the curve at which the charge injected during

the pulse is best approximated as $I_{inj} * T_{pulse}$, the product of the pulse time and the injection current at the peak of the curve. Furthermore, since the second order term is negative, this formula will only overestimate the injected charge, so it is guaranteed to not contribute to a dreaded "overshoot" error. Having established this principle, it remains only to find the V_{sd} corresponding to the curve whose maximum is the rate specified by the second question, and to find the V_{sg} at which that curve attains its maximum. This requires a model of peak injection rate versus V_{sd} and a model of V_{sg} at peak injection rate versus V_{sd}. Both of these models can be extracted from the characterization data using regression fits to first or second order polynomials (best results are obtained by fitting to $ln(I_{inj})$ rather than I_{inj}). Once V_{sd} and V_{sg} are specified, there is one extra degree of freedom since there are three terminal voltages (source, drain and gate) and only two constraints (V_{sd} and V_{sg}). This degree of freedom can be chosen arbitrarily, or in some systems, other considerations may provide an additional constraint.

As mentioned earlier, the reasonable approach of characterizing only one device and then applying this characterization to all of the device introduces some error. There is a nice way to correct for this error in the algorithm. When the floating gate voltage gets near to the target voltage (say, within 100mV), the rate of injection is generally not changing very much with repeated iterations (especially if the algorithm has been at all successful in reaching the peak of the injection curve). Thus, at this point, the injection current may be measured for one pulse, and then the measured injection current can be used in place of an inaccurate model of the injection characteristic.

One of the natural questions that arises when one thinks about this system for a while is "what is the limit on programming resolution/accuracy?". Clearly the injection current varies in a predictable way with V_{sd} such that we can easily achieve extremely low injection currents. It would seem that an injection pulse that injects

only one or two electrons could be performed easily. A short investigation of this idea revealed that the accuracy is limited by noise in the floating-gate transistor and the measurement device. If this noise were additive white noise, its effects could be eliminated by application of averaging (or equivalently, low-pass filtering). However, the noise has a "1/f" component due to flicker noise in the transistor, and this noise process adds enough inaccuracy to a comparison of the channel current before and after an injection pulse that changes in floating gate charge on the order of an elementary charge are not possible to observe clearly. The 1/f noise in the device means that the achievable accuracy of floating-gate programming decreases with the time elapsed since the device was measured (the error increases logarithmically with time since programming). Thus, the achievable accuracy of a programmed floating-gate current is fundamentally limited by the intrinsic device noise, which can be calculated from standard noise models based upon device dimensions, process parameters, target current, time available for averaging the measurement, and a specification of the duration for which the current needs to be valid after programming.

2.4 FPAA circuits measured in this framework

Developing a fast, flexible software framework and an effective algorithm for floating-gate programming allowed for a demonstration of several interesting circuits on an FPAA. The following circuits are included here because they illustrate some of the unique capabilities of analog floating-gate technology, they demonstrate the power of a reconfigurable platform like this FPAA, and because these measurements are an example of the type of work that is enabled by all of the "nuts and bolts" presented in Sections 2.1 through 2.3.

Before characterizing the frequency response of a variety of circuits, the achievable frequency range was briefly investigated. In general, the parasitic resistance and

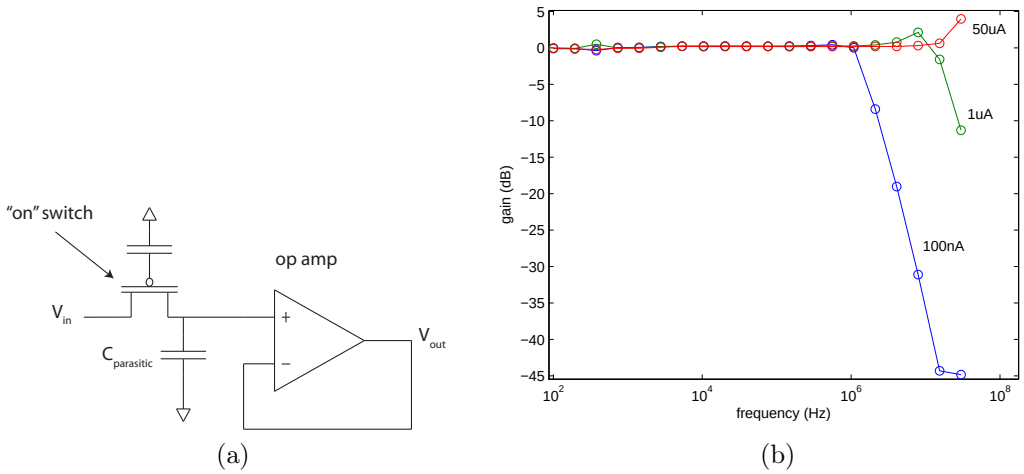


Figure 6: Measurement of frequency response of on-chip op amp at various bias currents. The circuit diagram is shown in a) and the measured results are in b). As the bias current is increased, the amplifier's bandwidth increases while the phase margin decreases. The bias current that corresponds to critical damping (thus avoiding a resonant peak in the transfer function) is between 100nA and 1uA, and yields a corner frequency between 1 and 10MHz.

capacitance of the interconnect in an FPAA imposes some constraints on the operating frequencies of the circuits that are routed using this interconnect. The values of parasitic resistance and capacitance depend on the details of the routing of the particular circuit, but the lowest achievable parasitics correspond to a case in which a single local routing line is driven by an ideal voltage source through a single switch, shown in Figure 6 with an op amp to buffer the signal off-chip. In this case, characterization data from the FPAA in [9] suggests that the resistance and capacitance will be approximately 10kOhm and 200fF, respectively. This creates a simple first order system with a pole at about 80MHz.

Measurements made on this circuit in the 100Hz-30MHz range, shown in Figure 6, indicate that the on-chip op amp used to buffer the signal is limited to a lower range than 80MHz. The response of the circuit is shown for several values of the

unit bias current that is used to bias the op amp. As the bias current increases, the bandwidth increases, but the phase margin decreases, eventually resulting in an underdamped frequency response. The bias current that corresponds to critical damping (thus avoiding a resonant peak in the transfer function) is between 100nA and 1uA, and yields a corner frequency between 1 and 10MHz. Of course, the stability of this amplifier is limited by its large load capacitance (PCB traces and oscilloscope probe/input). This load capacitance could perhaps be decreased by optimized PCB design and a buffer amplifier to isolate the oscilloscope capacitance from the op amp on the IC. In any case, this measured result gives no reason to doubt that the parasitic pole introduced by the routing is at the theoretically predicted frequency of 80MHz.

In addition to constraints imposed by the on-chip op amp, the testing frequency in this experiment was limited by the range of the available function generator (30MHz). Furthermore, the wavelength of electromagnetic waves at 20MHz in a coaxial cable is approximately 10m, so the available cables, which are on the order of 1m in length, are long enough that transmission line effects start to become important at this frequency and above. This means that some additional techniques for matching impedances must be used for effective testing in this frequency range.

One attractive feature of subthreshold CMOS circuits is the availability of many orders of magnitude of current with a fixed device size. Using analog floating gate transistors allows for programming of currents in this wide range in a simple and compact fashion. This principle is illustrated by using a floating-gate pFET as the bias current source for a standard wide-swing operational transconductance amplifier (OTA). When the OTA is connected in a follower topology, shown in Figure 7, the frequency response is a first order low pass filter, with the corner frequency being proportional to the transconductance (which in subthreshold operation is proportional to the bias current). Measured results from this circuit are shown over 7 orders of magnitude of bias current in Figure 7. This illustrates how a filter with corner

frequency that varies over 6 orders of magnitude can be implemented with a simple circuit. Figure 7 also shows a variation of this circuit, wherein the OTA has floating gate inputs. The input is coupled through a capacitive divider, the attenuation of which is approximately 10x. This increases the linearity of the OTA, while decreasing the transconductance.

The floating gate inputs on the OTA in Figure 8 provide another very useful feature. By changing the floating gate charge on the inputs, a common-mode and differential offset may be added to the inputs. A common-mode offset is often useful for keeping input signals within the input common mode range of the OTA, while a differential offset allows the implementation of a programmable level shifter. DC sweeps of this level shifter, programmed to a variety of differential offsets, result in the family of curves shown in Figure 8. One interesting application of this circuit can be observed by considering the behavior when V_{in} is 0. In that case (with V_{in} grounded), this circuit is a rail to rail programmable, supply-independent bias voltage generator.

Like the low-pass filter shown earlier, the high-pass filter topology shown in Figure 9 is tunable over many orders of magnitude in frequency, a feature that results from subthreshold operation and floating-gate programmability.

Many signal processing algorithms require spectral decomposition of the inputs. One classic example of this from the field of neurobiology is the spectral decomposition performed by the cochlea at the first stage of auditory processing. A similar

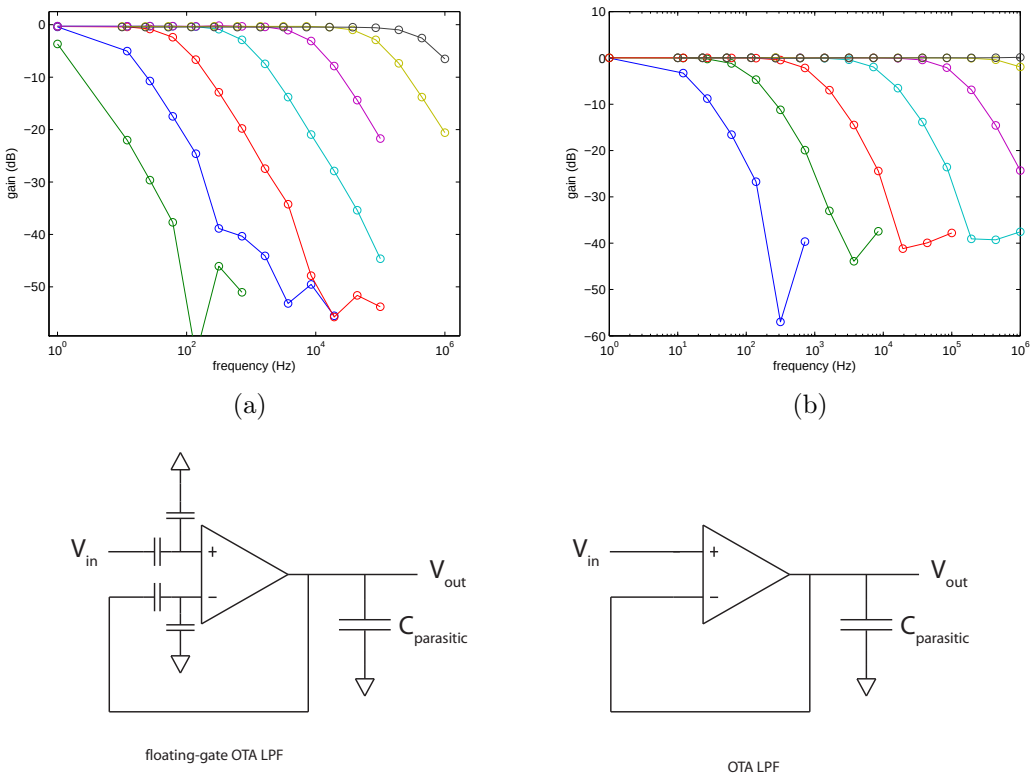


Figure 7: First order low pass filters measurements. The performance of the floating-gate OTA based low pass filter is shown in a), while that of the conventional OTA based low pass filter is shown in b). Schematics of both circuits are shown in c). The capacitive attenuator in the floating-gate OTA yields an attenuation of about 10x, which reduces the transconductance, and therefore the low pass corner frequency. The fact that the corner frequency of a single circuit can be programmed over a range of 6 orders of magnitude illustrates the flexibility derived from using subthreshold circuit design techniques and floating gate transistor technology.

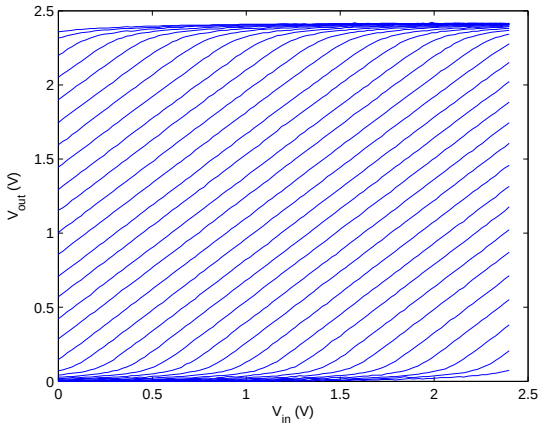
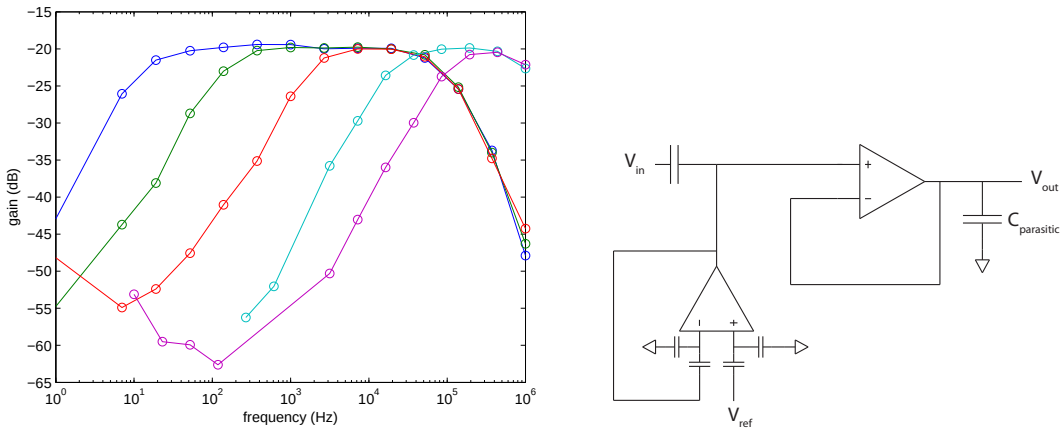
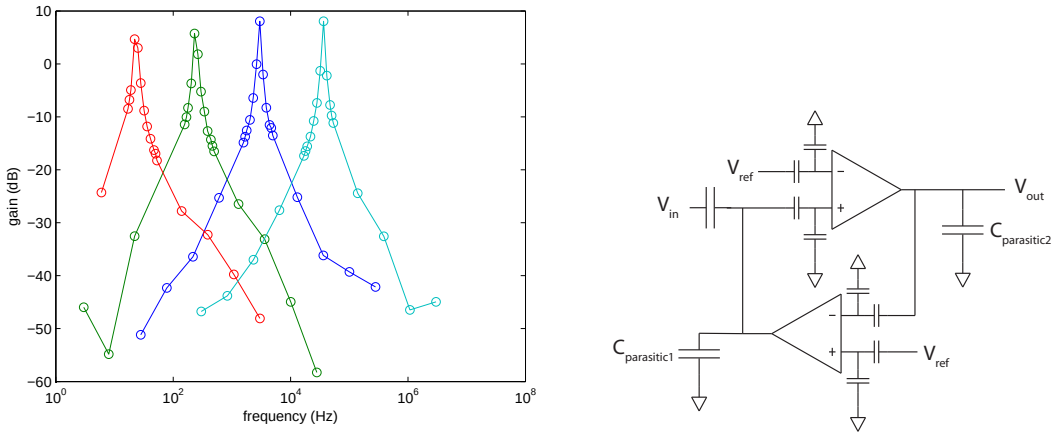


Figure 8: Family of curves resulting from DC sweeps of the floating gate OTA depicted in Figure 7 with a range of differential offset voltages programmed on the inputs. With a signal on the input, this circuit functions as a programmable level shifter, and its range of programmability is such that with the input grounded it functions as a rail-to-rail programmable supply-independent bias voltage generator.



(a) (b)

Figure 9: High pass filter circuit and measurement results. Programming the floating gate biases for the OTAs over many decades of current yields a large range of tunability for the filter characteristics. Measured performance is shown in a) and the circuit topology is shown in b). The capacitive divider provides an attenuation factor of approximately 10x.

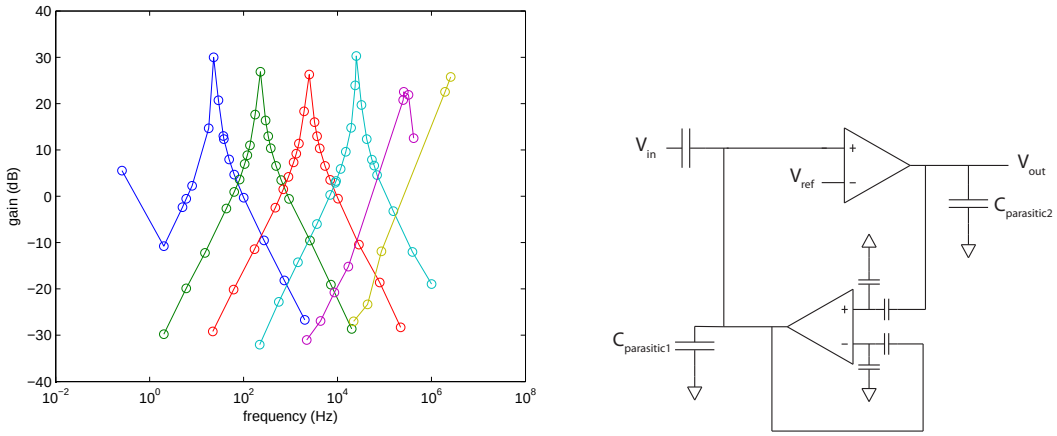


(a) (b)

Figure 10: Resonator topology for high-Q bandpass filter that can be used for spectral decomposition. Measured performance is shown in a) and circuit topology is shown in b). The capacitive dividers provide an attenuation factor of approximately 10x to each OTA.

functionality can be achieved by a bank of sharply tuned filters. High-Q resonant filters such as those depicted in Figures 10 and 11 can be used to create such a filter bank. In both of these circuits, the quality factor Q of the circuit is limited by the open-loop gain of the amplifiers.

These use of floating-gate transistors allows for circuit techniques that employ capacitive summing, yet do not have high-pass characteristics (*i.e.* the capacitive summation works even at low frequencies). One interesting example of a circuit that nicely takes advantage of this feature is shown in Figure 12. It is a common source amplifier with capacitive feedback, resulting in a unity gain inverting amplifier. The conventional circuit design approach for obtaining a unity gain inverting amplifier an op-amp with resistive feedback. The relatively large current that flows through the resistive network places constraints on the amplifier design, resulting in op-amp



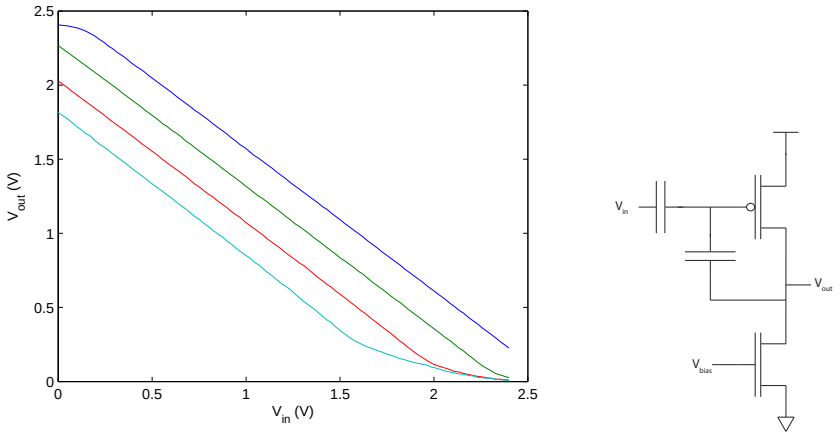
(a) (b)

Figure 11: Capacitively coupled current conveyor (C4) topology for high-Q bandpass filter that can be used for spectral decomposition. Measured performance is shown in a) and circuit topology is shown in b). The capacitive dividers provide an attenuation factor of approximately 10x to each OTA.

circuits that use power and silicon area inefficiently. By contrast, the circuit in Figure 12 accomplishes this purpose using only two transistors and a low bias current.

Another illustration of the power of capacitive summation is shown in Figure 13, which depicts a voltage mode summing circuit. Here the same considerations apply when comparing the efficiency of this circuit (with regard to silicon area and power) to that of the conventional "op-amp and resistive network" approach. Figure 13 also depicts a version of this voltage mode summing circuit with capacitive attenuation to impart a scaling factor to the summed result.

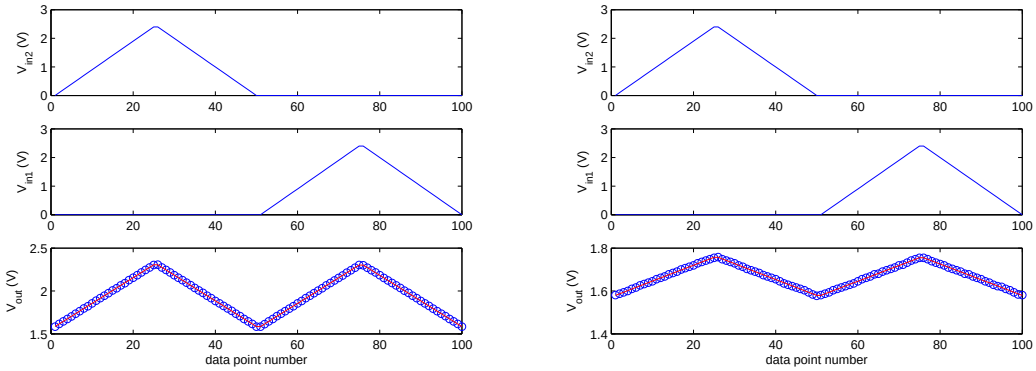
The inverting amplifier shown in Figure 12 can be combined with a differential pair to create a full-wave rectifier circuit, shown in Figure 14. The intuition behind this circuit is based on the fact that the common node in the differential pair is determined by the maximum of the two inputs. Thus, a differential pair that receives



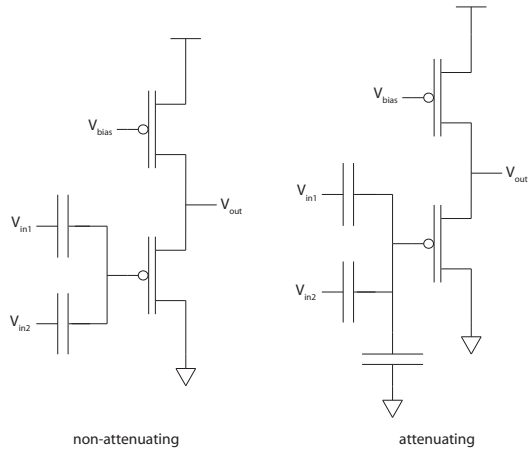
(a) (b)

Figure 12: Unity-gain inverting amplifier implemented with a floating gate and capacitive feedback. Measured performance for several bias voltages is shown in a), and the circuit topology is shown in b).

a signal and the inverted signal can be used for full wave rectification. This kind of operation can be useful for envelope detection and audio feature extraction.

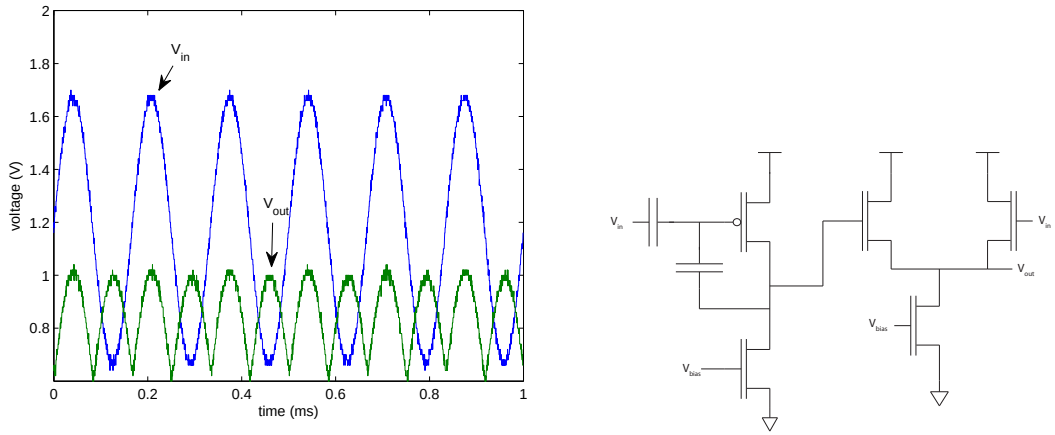


(a) (b)



(c)

Figure 13: Voltage-mode summing circuits that take advantage of capacitive summation at a floating node. In a) and b), the response of the circuit to consecutive triangle waves on the two inputs is shown. The measured output is plotted with open circles and a linear combination of the two inputs is plotted as a solid line. The coefficients in the linear combination are nearly equal, but not exactly (due to mismatch between the two input capacitors). In c), the circuit diagrams are shown for two different version of the summing circuit (one with a 6x capacitive attenuation and one without any attenuation).



(a) (b)

Figure 14: Full-wave rectifier circuit. The response of the circuit to a sinusoidal input is shown in a), and the circuit topology is shown in b).

Chapter III

NEW DEVICE CHARACTERISTICS: BI-DIRECTIONAL TUNNELING

3.1 Floating-gate array programming and bidirectional tunneling

As described in Chapter 1, floating-gate transistor arrays enable a variety of useful circuits. A few examples include a radial basis function classifier [49], field programmable analog arrays (FPAAs) for signal processing applications [30, 7], and more specialized FPAAs for sensor interfacing [50] and computation using multi-input translinear elements [59].

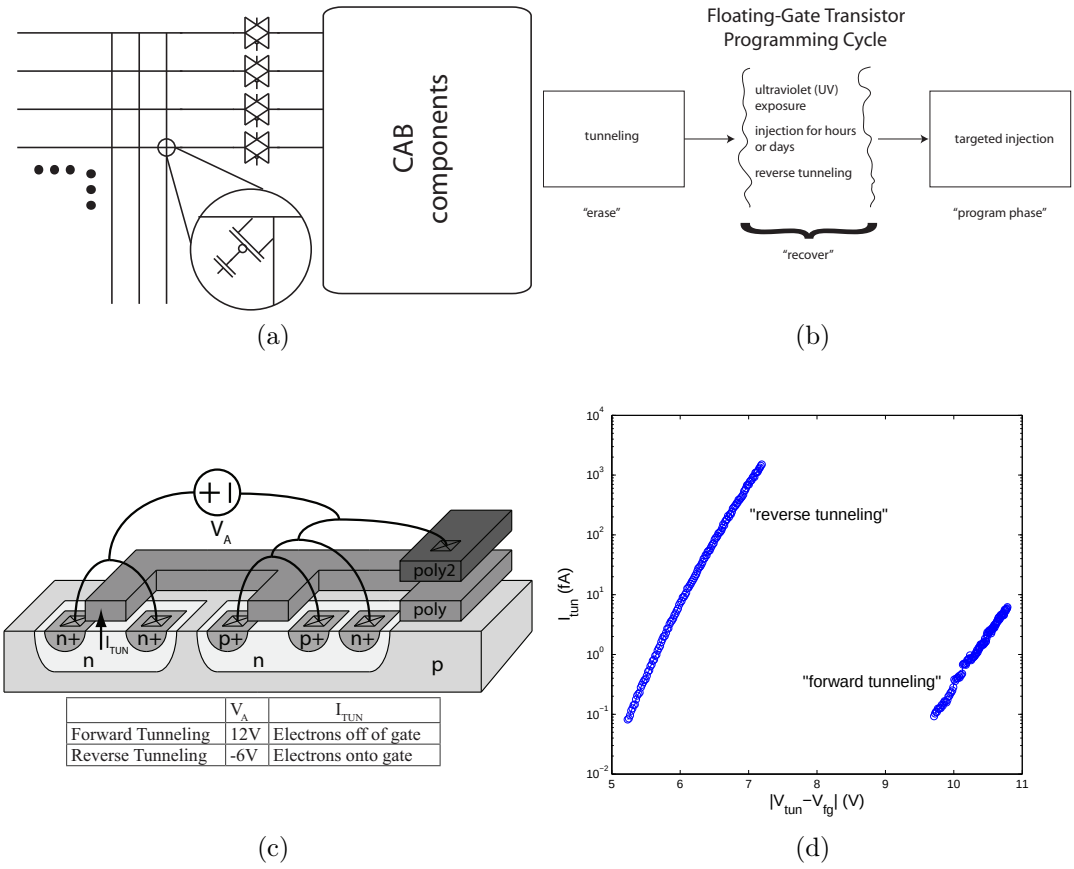
In all of the above-mentioned systems, many floating-gate transistors in an array are tunneled simultaneously as a global erase operation (Fowler-Nordheim tunneling), then individual floating-gate transistors are programmed via hot electron injection. In Chapter 2, the characteristics of injection for hot electron injection were presented and analyzed. One important feature of the characteristics is that for subthreshold currents, the rate of injection is roughly proportional to the channel current. A consequence of this fact is that if a device is tunneled too much, the rate of injection drops so low that the injection phase of programming either fails or requires unacceptably long times to complete. In the past, such a situation has been remedied by the application of ultraviolet light or very long hot electron injection exposure times.

This chapter describes a better option for addressing this issue, namely the use of bidirectional tunneling. In bidirectional tunneling, electrons are first removed from the floating gates in an erase operation that we will call "forward tunneling". Then, the polarity of the field applied to the tunneling junction is reversed, and electrons

are added to the floating gates in a global recover operation that we call "reverse tunneling". With proper selection of fields and exposure times, this sequence of operations can ensure that the injection phase of the programming cycle is successful. Figure 15 illustrates how bidirectional tunneling fits into the scheme of floating-gate programming . One interesting feature of reverse tunneling is that the terminal voltages required are much lower than for forward tunneling. This can be clearly seen by comparing the measured I-V characteristics of a single tunneling junction for forward and reverse tunneling, shown in Figure 15.

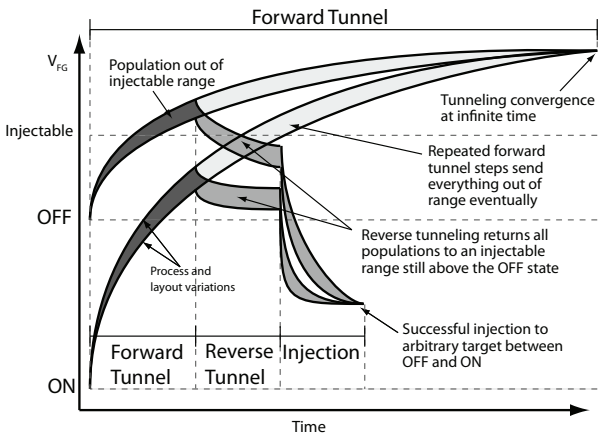
An extremely beneficial result of bidirectional tunneling is that it introduces a steady-state equilibrium floating gate voltage that can be controlled by the balance of tunneling and reverse tunneling parameters. By contrast, in an array that is programmed using only forward tunneling and hot electron injection, any devices that are not injected will simply be repeatedly tunneled, which drives them toward an equilibrium floating gate voltage that is far above the limit for reliable injection. As a result, it is simply a matter of time before such a system will encounter problems with injecting floating-gate devices that have not been injected for many programming cycles. This crucial difference between unidirectional and bidirectional tunneling approaches is illustrated in Figure 16.

In order to understand the bidirectional tunneling processes and gain insight into how to choose parameters for forward and reverse tunneling, it is helpful to make a quantitative model based on the basic theory of the tunneling processes, fitted to characterization data from a real floating-gate transistor array. The remainder of this chapter describes the results from implementing such a model, based on measured data from an FPAA designed for analog signal processing applications (described in detail in [9]), and also illustrates the use of the model to address the question of choosing optimal bidirectional tunneling parameters.



	V_A	I_{TUN}
Forward Tunneling	12V	Electrons off of gate
Reverse Tunneling	-6V	Electrons onto gate

Figure 15: Context and motivation for the study of tunneling with floating-gate transistor arrays. a) Structure of field-programmable analog arrays (FPAAs) wherein this work is applied. Such chips contain hundreds of thousands of floating-gate transistors, the charge of which is manipulated by global tunneling operations. b) Description of a single cycle of programming for the floating gates in such arrays. c) Cross-section view of floating-gate transistor with tunneling junction and gate coupling capacitor. Some example terminal conditions for forward and reverse tunneling are shown. d) Measured I-V relationship of the tunneling junction for both directions of tunneling. Reverse tunneling clearly requires much lower applied voltages to obtain similar currents.



(a)

Figure 16: Depiction of floating-gate voltages throughout the various phases of a programming cycle. The forward tunnel operation turns off all devices, but raises the voltage on some devices too far, making them impossible to inject reliably. The reverse tunnel operation then lowers the voltages back into the range wherein targeted injection can be done successfully. The trajectories of tunneling for infinite time illustrate what happens to un-injected devices in a floating-gate array that does not use bidirectional tunneling.

3.2 Modeling floating-gate voltage and electron tunneling

The schematic of a floating-gate array element, which was shown in Chapter 2, is included again here in Figure 17 for convenience. The associated capacitors, including the parasitic capacitances, are shown. The charge stored on the floating gate, in concert with the terminal voltages, determines the floating-gate voltage V_{fg}. Using a linear model for all of the capacitors, this relationship takes the form:

$$V_{fg} = (Q_{fg} + \sum V_i C_i)/C_{total} \tag{1}$$

Where the summation index i runs over all of the capacitors that couple to the floating gate, Q_{fg} denotes the net charge on the floating gate, and C_{total} is given by $\sum C_i$. In many circuits, the floating-gate charge Q_{fg} is left unchanged while the circuit is active (the circuit is somehow disabled to allow for programming of Q_{fg}), so the capacitor C_{gate} provides a means of manipulating V_{fg} while Q_{fg} remains fixed.

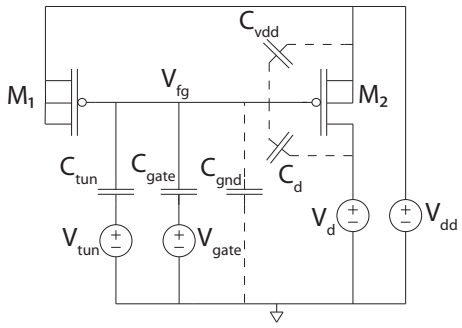


Figure 17: Details of the core floating-gate element. a) Schematic of floating-gate element. Parasitic capacitances are denoted with dashed lines. b)

Q_{fg} may be changed by way of tunneling through one of the oxides (in principle this could be any oxide, but tunneling happens much more readily through the gate oxides, which rules out tunneling through C_{gate} for most realistic terminal voltages).

The tunneling capacitor is designed to minimize the coupling C_{tun}, which in turn maximizes the voltage across this device subject to the constraint $V_{tun}<V_{tun,max}$, where $V_{tun,max}$ is the maximum available voltage for tunneling. The transistors M_1 and M_2 are matched. One of them is only used for manipulating Q_{fg} by way of hot electron injection, and the other is used as a circuit element. The motivation for this two-transistor design of the floating-gate element is described in [28]. The floating-gate element can also be designed with a single transistor approach, and the principles described in this article apply equally well to such devices.

If one cannot assume that the capacitors are linear, V_{fg} is determined by:

$$dV_{fg} = (dQ_{fg} + \sum dV_i C_i)/C_{total} \tag{2}$$

together with the boundary condition that V_{fg}=0 when Q_{fg} and all V_i are 0. In this case, C_i and C_{total} are not constants. Often C_i is described as a function of V_i-V_{fg}. Thus, C_i and C_{total} are implicitly functions of Q_{fg} and all the voltages V_i.

	LC device	HC device
W (um)	6	15.1
L (um)	2	0.6
C_{gate}, nominal (fF)	44	119
C_{ox}, nominal (fF)	118	89
C_{ox}, measured (fF)	130	98

Table 1: Device parameters for two example devices characterized in this study. The capacitive coupling ratios for terminals of devices that share global terminal voltages for tunneling operations results in different floating gate voltages and therefore different tunneling rates for the different devices. Accounting for this effect is essential for achieving desired behavior from tunneling operations. The abbreviation "HC" (for high coupling) is used for the device whose gate capacitor is large, whereas "LC" is used for the other device.

This capacitor nonlinearity complicates the model significantly, so linear capacitors is a very useful approximation for cases in which the resulting error is tolerable.

For the purpose of the present study, it was found that in order to predict the outcome of tunneling operations with a reasonable accuracy (V_{fg} errors of tens of mV), only the largest capacitor nonlinearity, namely that of C_{Vdd}, must be modeled. While the small capacitances C_d, C_{tun}, and C_{gnd} are all nonlinear, it was found that a linear model of them yielded satisfactory results. Not surprisingly, a linear model is a good description of the poly-poly capacitor C_{gate}.

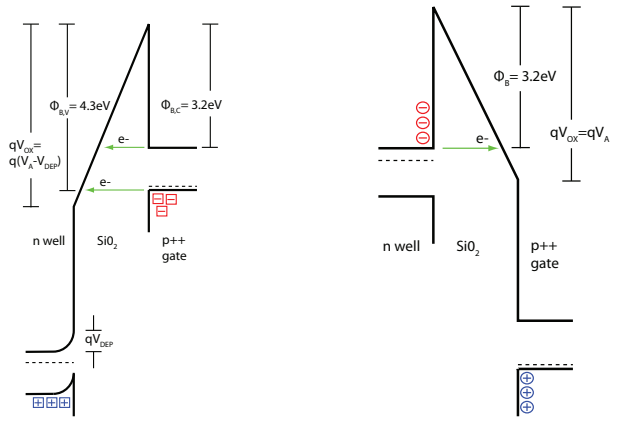
Several aspects of the system for floating-gate transistor programming are important for understanding the metrics of performance for tunneling. When floating-gate transistors are used as analog programmable parameters, a large range of achievable floating-gate voltages is usually desirable. The lower end of the programmable range is determined by limitations on the hot electron injection process, while the upper end is governed by the tunneling process. Thus, the achievable range is maximized by tunneling as much charge as possible onto the floating gate. However, too high a V_{fg} can cause injection to take arbitrarily long or to fail, depending on the control algorithm for injection. This limitation can result from the mechanics of the hot electron injection process itself, or it can be a consequence of constraints imposed by

the method of measuring the floating gate voltage.

Thus, the erase/recover operation must strike a compromise between V_{fg} dynamic range and hot electron injection time/reliability. For architectures in which tunneling is a global operation, a single choice of parameters must simultaneously strike this compromise for a large set of devices. This would not be an added challenge if all of the devices responded to a tunneling exposure in identical fashion. However, in practice, the various devices' responses can significantly differ from one another for two basic reasons. The first reason is that the capacitors coupling the global terminals to the floating gate voltage vary in size (this is generally the result of different devices having different nominal sizes for M_1, M_1, and C_{gate} due to circuit design considerations). This causes the floating gate voltages to change differently in response to changes of the global terminal voltages that occur before and after tunneling exposures. The second reason is that the I-V characteristic of tunneling junctions on nominally identical floating-gate devices differ significantly due to process variation. Both of these sources of mismatch in tunneling behavior are significant.

In order to illustrate the approach for handling the first source of mismatch (*i.e.* systematic differences in capacitors coupling to V_{fg}), the next two sections (Sections 3.3 and 3.4) present results for two floating gate elements that have different nominal parameter values for M_1, M_2 and C_{gate}, yet share a global tunneling voltage. Table 1 shows the nominal parameters for the two devices, which significantly differ in their ratios of C_{gate}/C_{ox}, where C_{ox} is the lumped oxide capacitance of the gates of M_1 and M_2, which can be estimated as the value of C_{Vdd} under conditions of accumulation or inversion. Throughout this chapter, the shorthand HC (for high coupling) is used to refer to the device with the larger ratio C_{gate}/C_{ox}, while LC (low coupling) is used for the other device.

Some results pertaining to the second source of mismatch (process variation in tunneling characteristics of nominally identical devices) are presented in Section 3.6.



(a) (b)

Figure 18: Band diagrams depicting the tunneling processes. a) Reverse and b) forward Fowler-Nordheim tunneling band diagrams for a an n-well varactor with a degenerately p-doped gate.

Table 2: Ranges of terminal voltages used for tunneling

	forward tunneling	reverse tunneling	run mode
V_{gate}	0	V_{rev1}(0-6V)	0
V_d	0	V_{rev2}(4.5V-6V)	0-2.4V
V_{dd}	0	V_{rev2}(4.5V-6V)	2.4V
V_{tun}	9V-14V	0	0

Tunneling is the quantum mechanical phenomenon whereby an electron may pass through an electrostatic potential barrier without having sufficient energy to propagate over the barrier. It is a fundamental result from quantum mechanics that the probability of an electron tunneling through a barrier decreases exponentially with the barrier's width. If the barrier under consideration is an insulator separating two conductors, applying a voltage two the two terminals creates a field in the insulator, which can reduce the barrier width, as shown in Figure 18. This process may be modeled analytically by the use of the WKB approximation [27]. This results in what is called a Fowler-Nordheim type expression for the tunneling current:

$$I_{tun} = C_{FN} E^2 exp(-\frac{4}{3}\frac{(q\phi)^{1.5}\sqrt{2m_{ox}}}{Eq\hbar}) \quad (3)$$

where I_{tun} is the tunneling current, C_{FN} is a scaling factor, E is the field in the insulator, m_{ox} is the electron effective mass in the insulator, q is the elementary charge, $\hbar$ is the reduced Planck constant, and ϕ is the barrier height. Thus, in practice, tunneling can be done on floating-gate transistors by establishing a high voltage across a gate oxide. Figure 18 qualitatively depicts the band diagrams through the tunneling junction for forward and reverse tunneling, under the assumption that the polysilicon gate is degenerately doped p-type. As discussed further in Section 3.3, the differences in these pictures could partially account for the dramatic difference in I-V characteristics for forward and reverse tunneling.

The terminal voltages applied to the floating-gate transistors in order to achieve forward tunneling and reverse tunneling are shown in Table 2. This table also shows the terminal voltages for the devices in "run mode", *i.e.* after the floating-gate programming operations are completed, which is significant for this discussion because this constitutes a reference set of terminal conditions that is used when calculating numbers for V_{fg}. For forward tunneling, all of the terminals are set to ground except for V_{tun}, which sets up a maximal voltage across the capacitor C_{tun}. For reverse tunneling, all of the terminals are set to a high potential, while V_{tun} is set to ground, which again maximizes the voltage across C_{tun} (this time in the opposite polarity). The user-specifiable parameters of these tunneling operations are V_{tun} during forward tunneling, V_{dd} and V_{gate} during reverse tunneling, and the exposure times for forward and reverse tunneling. Section 3.4 presents an approach for choosing these parameters in a way that allows one to achieve the desired balance between floating-gate voltage dynamic range and hot electron injection reliability/time while minimizing tunneling time. The motivation for allowing V_{gate} to differ from V_{dd} during reverse tunneling is also described in that section.

3.3 Measured tunneling characteristics

The dependence of the tunneling current on the voltage across the tunneling capacitor was measured for forward and reverse tunneling, for both the high and low coupling devices. The measurements were taken by leaving V_{tun} fixed and varying V_{fg}, measuring the tunneling current for each different V_{fg}. In order to get the measurement right, one must have a means of accurately inferring V_{fg} across a wide dynamic range (including voltages for which no measurable current will flow in the transistor), and one must also ensure that V_{fg} does not change by an appreciable amount during a single tunneling pulse. The inference of V_{fg} requires characterization of all of the capacitances that couple to the floating gate, including the nonlinear C_{Vdd}. An elegant experiment that simultaneously characterizes C_{Vdd} and the tunneling current was devised, and it is described in detail in Section 3.5.

The results of the measurement are shown in Figure 19, which depicts the data in a type of plot known as a Fowler-Nordheim plot. In a Fowler-Nordheim plot, the quantity $\ln(\frac{I}{E^2})$ is plotted versus $\frac{1}{E}$. Examining (3) reveals that such a plot should yield a straight line. Note that for forward tunneling, the experiment was repeated at two different fixed levels of V_{tun} (12.5V and 14V). Several aspects of these results require further discussion. Note in (3) that the slope of the Fowler-Nordheim plot is a function of the barrier height and fundamental physical constants, allowing the barrier height to be extracted from the measured data. Performing this calculation yields 5.54eV for the barrier height for forward tunneling and 3.19eV for the barrier height for reverse tunneling. The latter is consistent with the numbers typically reported for experimentally determined silicon-SiO2 junction barrier height[18].

As shown in Figure 18, assuming that the gate polysilicon is degenerately p-doped could result in an increase in barrier height for forward tunneling that is approximately

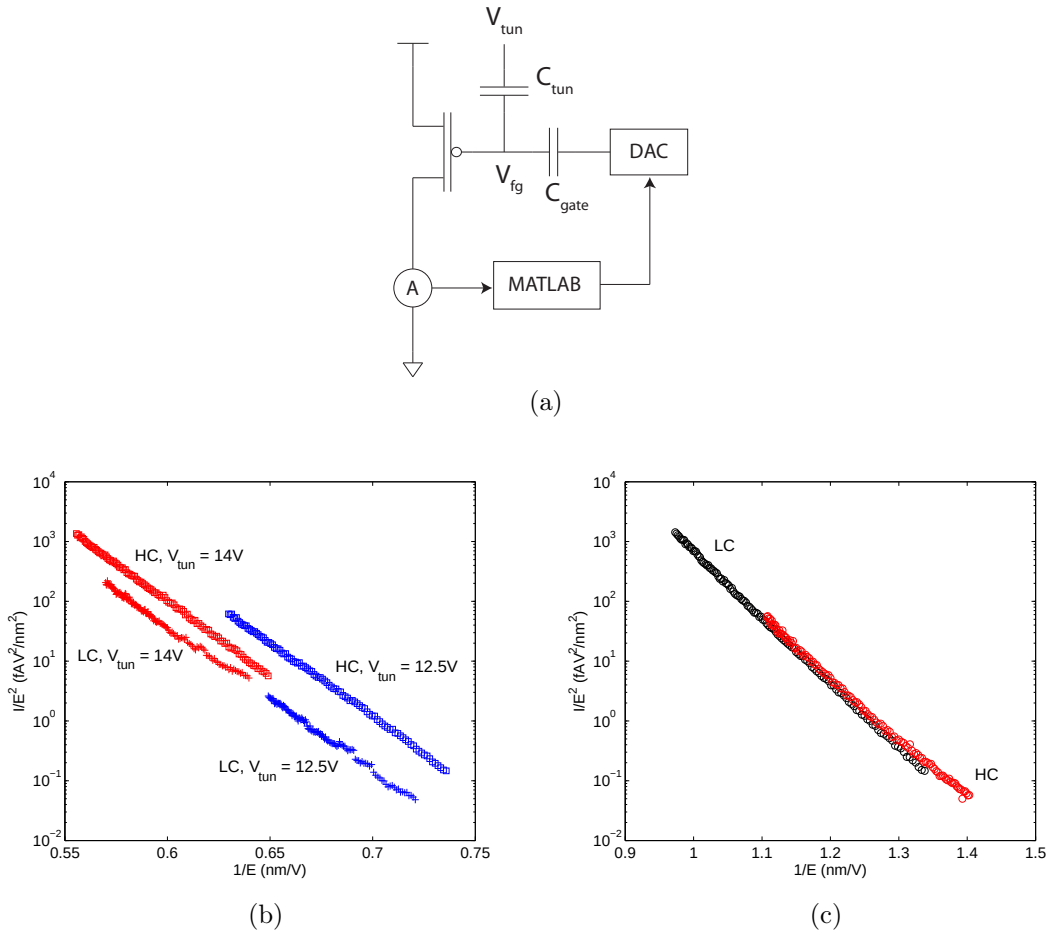


Figure 19: Measured Fowler-Nordheim plots of forward and reverse tunneling characteristics. a) Setup for characterizing tunneling. Iteration with the control loop allows the tunneled charge to be calculated as $C_{gate}\Delta V_{DAC}$. b) Forward tunneling characteristics for high coupling (HC) and low coupling (LC) devices, characterized with two different fixed tunneling voltages V_{tun}. b)Reverse tunneling characteristics high coupling (HC) and low coupling (LC) devices.

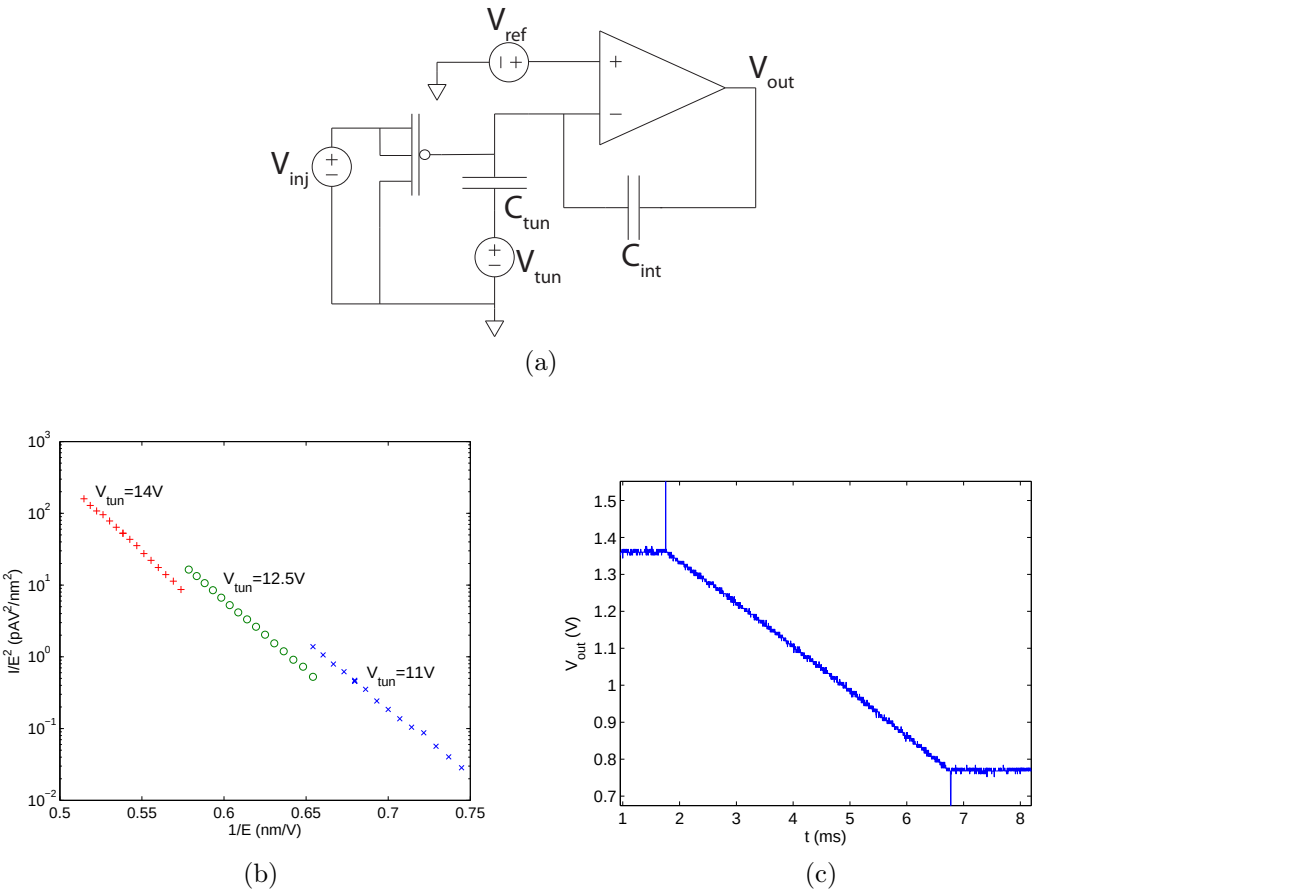


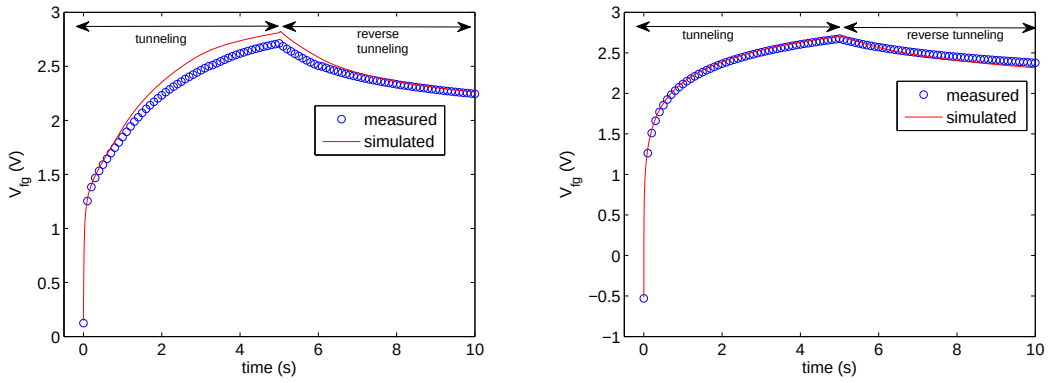
Figure 20: Tunneling characterization with direct control of V_{fg}. a) Test circuit used for measurements. b) Measured Fowler-Nordheim plots of the characteristic for 3 different voltages V_{tun}. c) Example waveform measured at V_{out} in the circuit shown in (a).

equal to the band gap of Si. Even taking this into account, the resulting barrier height of 4.3eV is significantly lower than the value extracted for forward tunneling in these devices. The cause of this discrepancy is not known at this time. Of course, the fits of the measured data are in terms of the field in the insulator, which was calculated using the nominal gate oxide thickness for the process, 7nm. So the disparity between forward and reverse tunneling characteristics could also be interpreted as a change of barrier thickness with the same barrier height. For instance, if the effective insulator thickness for forward tunneling is 16nm, this is consistent with a barrier height of 3.2eV for the measured data.

A second interesting feature of the measured characteristics is the dependence on the voltage V_{tun}. Since V_{tun} is taken into account to compute the value on the x-axis, if the device obeys the characteristic in (3), the curves for different voltages V_{tun} should lie on the same line. However, the curves corresponding to different V_{tun} levels for the HC device are clearly different. This difference is unexpected, and at this time I have no physical explanation for this effect. To my knowledge, this effect has never been reported elsewhere.

Because this result is fairly surprising, and the experimental method involved a somewhat complicated inference of the floating-gate voltage, a separate means of verification of this phenomenon was investigated. Measurements were made on a floating-gate transistor that is specifically instrumented for such characterization. This circuit, which is on a different chip in the same process as the FPAA device that was characterized, is shown in Figure 20a. The circuit allows one to fix the floating-gate voltage using an amplifier in feedback, and thereby eliminates the need for inference of V_{fg}. When V_{tun} is set to a high enough voltage to cause tunneling through C_{tun}, the feedback loop regulates V_{out} such that the displacement current through C_{int} exactly cancels the tunneling current, so V_{fg} remains fixed. Thus, we have $I_{tun} = C_{int}\frac{dV_{out}}{dt}$. If V_{tun} is fixed, then I_{tun} is fixed, so the amplifier output slews linearly and its slope gives a measurement of I_{tun}. Using this more direct form of measurement, the characterization of V_{tun} versus the voltage across C_{tun} was performed for several fixed values of V_{tun}. Figure 20 shows the resulting characteristic, as well as an example of the measured waveform V_{out}. The data confirms the dependence of the tunneling current on V_{tun}.

Even without having a physical explanation for the dependence of the forward tunneling current on V_{tun}, it is relatively straightforward to take this effect into account when modeling the devices' behaviors during tunneling. The simple model that



(a) (b)

Figure 21: Measured and simulated trajectories of floating-gate voltages during a 5-second tunneling pulse followed by a 5-second reverse tunneling pulse. a) Floating-gate voltage trajectory a) for low coupling (LC) device and b) for high coupling (HC) device.

is used to simulate tunneling is as follows. Linear fits are made to the measured data for forward and reverse tunneling (the data shown in Figure 19). A separate fit is used for each of the two values of V_{tun} that were used in the characterization measurement. For forward tunneling at a given voltage V_{tun}', the two coefficients in the linear fit are linearly interpolated or extrapolated from the fits at the measured values of V_{tun} (namely, 12.5V and 14V). Then, having a simple model for I_{tun} as a function of the voltage across C_{tun}, this tunneling current I_{tun} can be related to $\frac{dV_{fg}}{dt}$ using (2). When all of the terminal voltages V_i are fixed, (2) reduces to $C_{total}dV_{fg} = dQ_{fg}$, from which one can conclude that $C_{total}\frac{dV_{fg}}{dt} = I_{tun}$.

This equation was integrated in time using Euler integration in MATLAB®. In order to take into account the effect of the terminal voltages used during tunneling, (2) is used, together with measured estimates of all of the capacitances that couple to the floating gate (see Section 3.5 for a detailed description of the measurement of the capacitances).

Using the numerical model described above, the effect of tunneling pulses with a variety of terminal voltages, initial floating-gate voltages, and durations can be simulated. The simulated results can be compared with measured data. In Figure 21, the trajectories of the floating-gate voltages for both device types are shown as the devices undergo a 5-second forward tunneling pulse followed by a 5-second reverse tunneling pulse. The experimental data are measured by alternately performing short tunneling pulses and then making measurements to infer V_{fg}. The accuracy of the model predictions is representative of the accuracy across the full range of initial conditions and terminal voltages that are used in practice. The error in the model is the combined effect of the approximate description of the dependence of tunneling current on V_{tun}, the approximation that only C_{Vdd} is nonlinear, and errors in the measurement for characterization of the capacitive couplings and the tunneling currents.

3.4 Choosing bidirectional tunneling parameters

In Section 3.1, the advantage of using tunneling and reverse tunneling in concert was described qualitatively. This point can be illustrated in a concrete way by measuring and modeling the behavior of floating gate devices over many programming cycles. Figure 22 compares the floating-gate voltages of two HC devices over 200 programming cycles. One of the HC devices is injected to a low V_{fg} (approximately -0.5V) during each programming cycle, and the other starts at the same low V_{fg}, but is never injected during this period of interest. This illustrates a realistic situation for floating-gate arrays in which some floating-gate devices are not targeted by injection for many consecutive programming cycles.

If the reverse tunneling step is not used in each programming cycle (as in Figure 22a), the un-injected device's V_{fg} is seen to steadily increase with each programming cycle. Thus is it only a matter of time before the device becomes un-injectable. In contrast, if both forward and reverse tunneling are performed in each programming

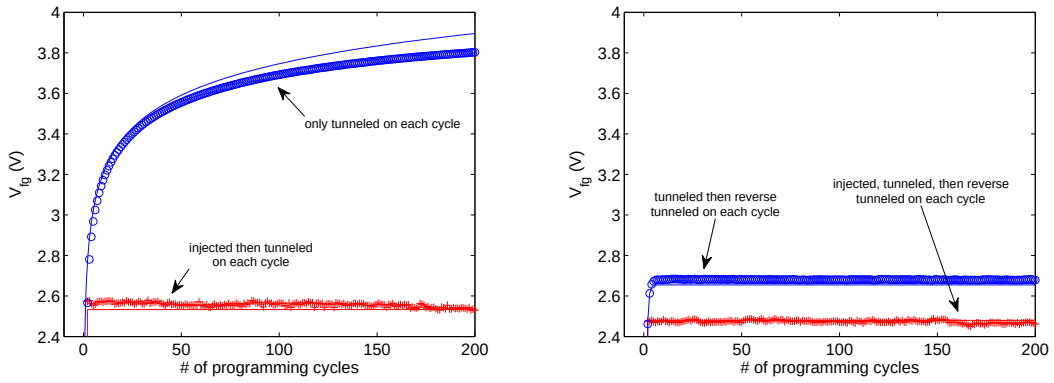


Figure 22: Measured and simulated V_{fg} over many programming cycles comparing bidirectional tunneling to forward tunneling only. The floating gate voltage just prior to the targeted injection phase of programming is measured in each cycle. Modeled trajectories are shown as solid lines and measured trajectories are shown as crosses and open circles. a) Multi-programming cycle behavior when only forward tunneling is used in the programming cycle. b) Multi-programming cycle behavior when bidirectional tunneling is used.

cycle (as in Figure 22b), the un-injected device's V_{fg} settles to a steady-state value that is not much higher than that of the injected device. This behavior is greatly preferable, since one can be confident in targeted injection for any device in the array, regardless of how many programming cycles it has been since it was last injected.

The natural question that arises at this point is how to determine tunneling parameters, *i.e.* terminal voltages and exposure times for forward and reverse tunneling, given system constraints. In order to frame the question properly, one may first observe that each set of tunneling parameters specifies an equilibrium V_{fg}. In order to understand this, first consider that the change in V_{fg} that results from a forward tunneling pulse monotonically decreases as the initial V_{fg} increases. Similarly the change that results from a reverse tunneling pulse monotonically increases as the initial V_{fg} increases. Thus, for any set of tunneling parameters, there is some initial V_{fg} such that the change due to the forward tunneling pulse is equal and opposite the

change due to the reverse tunneling pulse. This is the equilibrium voltage specified by the particular choice of tunneling parameters, and it also is the highest V_{fg} that will be present in the array at the beginning of any injection operation. Similarly, the lowest V_{fg} in the array with be that resulting from a device that is injected to the lowest attainable V_{fg}, then erased and recovered once. Note that this maximum and minimum V_{fg} correspond to the limiting behavior observed in the two different devices in Figure 22b.

Thus, for a given type of device, the problem of choosing parameters becomes a constrained optimization. There is an acceptable minimum and maximum V_{fg}, which will be referred to as $V_{fg,min\_allowable}$ and $V_{fg,max\_allowable}$, respectively. $V_{fg,min\_allowable}$ is dictated by desired dynamic range of floating-gate programmable parameters, while $V_{fg,max\_allowable}$ is determined by the requirements for hot electron injection reliability and time. There are maximum and minimum voltages on all terminals (dictated by device ratings and/or by available voltage supplies). The tunneling parameters, together with the device characteristics (capacitive couplings and Fowler-Nordheim plots for forward and reverse tunneling), determine a minimum and maximum V_{fg}, which will be referred to as $V_{fg,min}$ and $V_{fg,max}$, respectively. The problem is to find a set of tunneling parameters such that all terminal voltages stay in allowable ranges, $V_{fg,min} \geq V_{fg,min\_allowable}$, and $V_{fg,max} \leq V_{fg,max\_allowable}$, and the time required for the combined erase/recover operation is minimized. When multiple devices with different characteristics must be tunneled simultaneously, then each different device type will impose two constraints, $V_{fg,min,i} \geq V_{fg,min\_allowable,i}$ and $V_{fg,max,i} \leq V_{fg,max\_allowable,i}$.

The models presented in this chapter can be used to find $V_{fg,min}$ and $V_{fg,max}$ as a function of tunneling parameters and device characteristics, and thus can be combined with any numerical algorithm for constrained optimization in order to find optimal tunneling parameters. However, some intuition can be brought to bear in order to obtain good parameter choices without going through the trouble of performing the

numerical optimization. Because tunneling is exponentially faster at higher fields, and the goal is to minimize tunneling time, a reasonable starting guess would be to pick the voltages that maximize the field (*i.e.* the highest V_{tun} for forward tunneling and the highest V_{dd} and V_{gate} for reverse tunneling). This reduces the dimensionality of the parameter space to 2 (exposure times for forward and reverse tunneling), which is a fairly tractable problem for a simple "guess and check" approach.

This method was tested to find tunneling parameters for the HC and LC devices characterized in this study, for 3 different sets of system constraints. For all 3 cases, the minimum terminal voltage was ground, the maximum for V_{dd}, V_{gate}, and V_d was 6V (sufficiently low to reliably avoid any junction breakdowns from occurring), $V_{fg,min\_allowable}$ was 2.4V (this corresponds to rail-to-rail programmability of the V_{fg}'s). In case 1, the maximum V_{tun} was 14V, and the $V_{fg,max\_allowable}$ voltages were set as the maximum voltage such that the FET will produce a clearly measurable current (20nA) when V_{dd}=6V and V_{gate}=0V. For case 2, the voltages $V_{fg,max\_allowable}$ were set assuming V_{dd}=4V instead of V_{dd}=6V. This is a real constraint encountered for the FPAA in [9] when a particular circuit is used to measure the drain current of the floating-gate transistors. For case 3, the maximum V_{tun} was 12V, which is also a real constraint for some of the hardware environments in which these FPAAs are used.

For each of the above 3 cases, some locally optimal tunneling parameters were found. The constraints, resulting parameters, and predicted V_{fg}'s are shown in Table 3. Not surprisingly, the least constrained case (case 1) resulted in the fastest combined erase/recover time, 43ms. There is an important difference between case 1 and case 2, which provides a useful intuition for both floating-gate array design and selection of tunneling parameters. In case 1, V_{gate}=V_{dd}=6V during reverse tunneling, which maximized the voltage across C_{tun}, thus maximizing the reverse tunneling current and minimizing the time spent in reverse tunneling. After the reverse tunneling pulse

Table 3: Optimal choice of tunneling parameters for 3 different realistic sets of constraints

(a) For this case, minimum and maximum allowable post-tunnel V_{fg} for the LC device are 2.4V and 3.24V. Minimum and maximum allowable post-tunnel V_{fg}'s for the HC device are 2.4V and 4.06V.

	model prediction	experimental result
V_{tun}	14V	14V
V_{rev1}	6V	6V
V_{rev2}	6V	6V
$t_{tun,forward}$	30ms	37ms
$t_{tun,reverse}$	8ms	6ms
LC min V_{fg}	2.85V	2.88V
LC max V_{fg}	3.21V	3.16V
HC min V_{fg}	2.4V	2.42V
HC max V_{fg}	2.67V	2.62V

(b) For this case, minimum and maximum allowable post-tunnel V_{fg}'s for the LC device are 2.4V and 2.7V. Minimum and maximum allowable post-tunnel V_{fg}'s for the HC device are 2.4V and 2.95V.

	model prediction	experimental result
V_{tun}	14V	14V
V_{rev1}	3.6V	3.6V
V_{rev2}	6V	6V
$t_{tun,forward}$	30ms	32ms
$t_{tun,reverse}$	1.1s	1.1s
LC min V_{fg}	2.59V	2.57V
LC max V_{fg}	2.68V	2.63V
HC min V_{fg}	2.43V	2.41V
HC max V_{fg}	2.63V	2.64V

(c) For this case, minimum and maximum allowable post-tunnel V_{fg}'s are the same as in (b), but the maximum available V_{tun} is reduced to 12V.

	model prediction	experimental result
V_{tun}	12V	12V
V_{rev1}	3.6V	3.6V
V_{rev2}	6V	6V
$t_{tun,forward}$	2.5s	3.2s
$t_{tun,reverse}$	0.8s	0.8s
LC min V_{fg}	2.44V	2.42V
LC max V_{fg}	2.67V	2.62V
HC min V_{fg}	2.43V	2.5V
HC max V_{fg}	2.64V	2.6V

is finished, the floating-gate voltages of the HC and LC devices are approximately equal, but after the terminal voltages are returned to their "run mode" condition, V_{fg} for the HC device is much lower than that of the LC device (due to their different coupling strengths). This leads to a significant difference in the pre-injection V_{fg} for the two devices, which is not a problem because the constraints in case 1 are relatively loose.

In case 2, the constraints are much tighter, and the disparity after erase and recover between V_{fg}'s of the two devices that would result from using V_{gate}=V_{dd} is unacceptably high. By setting V_{gate} to 3.6V, the rate of reverse tunneling is decreased significantly and so the reverse tunneling time is considerably longer, bringing the total erase/recover time to about 1.1s. However, the change from the reverse tunneling terminal voltages to the "run mode" terminal voltages does not introduce any extra difference between the two devices' V_{fg}'s. This allows the much tighter constraint on the V_{fg}'s to be met. In general, when two devices with significantly different capacitive couplings their respective V_{fg}'s share a tunneling line, it will be necessary to use this technique to meet very tight constraints on maximum and minimum V_{fg}, which will result in an increase in the time required for erase/recover operations.

In case 3, the bounds on allowable V_{fg} are kept the same as in case 2, but the maximum tunneling voltage is reduced. This does not introduce a qualitatively different effect. It simply results in a longer time required for the erase/recover operations.

For all 3 cases, the parameters predicted by the model were verified by experimental measurements. In order to satisfy the constraints of V_{fg}'s, the tunneling exposure times needed to be adjusted slightly from the predictions of the model. Comparison of the two columns in Table 3 shows that the agreement is quite good, and that the model is indeed a good basis for determining optimal tunneling parameters.

3.5 Characterization method

Measuring the tunneling characteristics of a floating-gate transistor requires knowledge of the floating-gate voltage V_{fg}. This information can be inferred from a drain current measurement on the basis of a model of the I-V characteristics of the FET. However, this approach has limitations in dynamic range. Specifically, when V_{fg} is sufficiently high, it is often impossible to make a meaningful measurement of the drain current. Furthermore, for an array such as the FPAA used in this study, the drain current that is measured when V_{fg} is very low is limited by parasitic series resistances that are poorly characterized. The range over which V_{fg} may be inferred from a current measurement is extended substantially by making use of models of the capacitive couplings to V_{fg}, as in (2). Of the capacitors shown in Figure 17, only C_{Vdd} was modeled as a nonlinear capacitor. Its capacitance was modeled as a function of V_{fg}-V_{dd}, and this function was measured experimentally in the same experiment wherein the tunneling current was measured (as described below).

The linear capacitances were characterized in terms of the gate coupling capacitor C_{gate} by using (2) in the case of fixed Q_{fg}:

$$dV_{fg} = \sum dV_i C_i / C_{total} \tag{4}$$

By making a small change dV_i to a terminal voltage V_i, then sweeping V_{gate} until drain current is unchanged, we can conclude that dV_{fg}=0, and thus $\frac{C_i}{C_{gate}} = -\frac{dV_{gate}}{dV_i}$.

For this work, inference of the floating-gate voltage was made by manipulating the source and drain terminals until the drain current was measured to be a fixed current I_{ref} that is about an order of magnitude lower than threshold current of the the device. The value of I_{ref} that was used is 20nA. This allows inference of V_{fg} using the simple subthreshold I-V relationship $V_g = V_s - V_{th} - \frac{U_T}{\kappa} \ln(\frac{I_{ref}}{I_{th}})$, where V_g is the gate voltage (for a floating-gate transistor V_g=V_{fg}), V_{th} is the threshold voltage, U_T is the thermal voltage, κ is the subthreshold parameter describing the capacitive coupling

of the gate to the surface potential, and I_{th} is the threshold current. The process parameters V_{th}, κ, and I_{th} were extracted from measurements on non-floating-gate transistors on the same chip. Having found a set of terminal voltages that results in a known V_{fg}, this V_{fg} may be referred to any other set of terminal voltages using the model of capacitive couplings to the V_{fg}, (2).

A set of measurements was devised that simultaneously characterizes the nonlinear capacitor C_{Vdd} and the tunneling current in the device. It consists of three basic steps: 1) Measure I_{tun} under a set of terminal conditions that yields a drain current of I_{ref}. Under these terminal conditions, V_{fg} can be inferred without having knowledge of the C-V characteristic of C_{Vdd}. 2) Measure C_{Vdd}/C_{gate} under these same terminal conditions. 3) Repeat measurements 1 and 2 over a range of floating-gate voltages, using previously measured C_{Vdd} values to infer the V_{fg} that results from changing the terminal voltage V_{gate}. Details of each of these steps are as follows.

1) Measure $\mathbf{I_{tun}}$ at condition in which drain current is $\mathbf{I_{ref}}$. V_{dd} and V_d terminals are fixed to voltages sufficient to put the transistor in saturation for subthreshold current levels without allowing injection (2.9V and 1.9V were used in this work). Set set V_{tun} to any fixed voltage that is sufficiently low to not cause tunneling (8V was used in this experiment). Sweep V_{gate} and measure drain current until a current of I_{ref} is obtained. Pulse the tunneling line to a fixed tunneling voltage (12.5V and 14V were used here in 2 different experiments) for a period of time, T_{pulse}. After the tunneling pulse, once again sweep V_{gate} until the drain current is I_{ref}. The change in V_{gate} (before the tunneling pulse versus after) is proportional to the charge that was transferred during the tunneling pulse: $\triangle V_{gate} C_{gate} = \triangle Q_{fg}$. The time-averaged tunneling current during the pulse is given by $I_{tun,avg} = \frac{\triangle Q_{fg}}{T_{pulse}}$. If $\triangle V_{fg}$ is small, then I_{tun} is approximately constant during the tunneling pulse and $I_{tun,avg}$ may be interpreted as the instantaneous tunneling current. Thus this step is iterated, with the pulse time T_{pulse} being varied until $\triangle V_{fg}$ is small but well above

the measurement noise (10-20mV was used).

2) Measure $\mathbf{C_{Vdd}/C_{gate}}$ under these same terminal conditions. Repeat step 1, but during the tunneling pulse, set V_{dd} 100mV higher than in step 1. Again, repeat, but leave V_{dd} the same as in step 1 and set V_{gate} 100mV higher than in step 1. Collecting the results from steps 1 and 2 yields finite difference approximations for the derivatives $\frac{dI_{tun}}{dV_{gate}}$ and $\frac{dI_{tun}}{dV_{dd}}$. This yields an estimate of C_{Vdd}/C_{gate}, since $\frac{C_{vdd}}{C_{gate}} = (\frac{dI_{tun}}{dV_{dd}})/(\frac{dI_{tun}}{dV_{gate}})$, which follows from the fact that V_{dd} and V_{gate} only affect I_{tun} via coupling to V_{fg}.

3) Repeat measurements 1 and 2 over a range of floating-gate voltage. The basic operation described in steps 1 and 2 can be repeated, but with a shift of V_{fg} from the reference condition (in which drain current is I_{ref}). This shift can be achieved by added some change $\triangle V$ to the V_{gate} after finding the V_{gate} where the drain current is I_{ref}. This $\triangle V$ can be varied over a wide range, and a value of V_{fg} can be associated with each $\triangle V$ using the relationship $V_{fg} = V_{fg,ref} + \int_0^{\triangle V} \frac{C_{gate}}{C_{total}} dV_{gate}$, where $V_{fg,ref}$ is the floating-gate voltage at which the drain current is measured to be I_{ref}. C_{total}/C_{gate} can be found by summing each of the linear capacitances C_i/C_{gate} with the estimate of C_{Vdd}/C_{gate} from step 2. Note that while the ratios C_i/Cg are sufficient for modeling the changes in V_{fg} that result from changes in terminal voltages, C_{gate} still appears as a proportionality constant in the inferred values of I_{tun}. For this work, the nominal values of Cg (shown in Figure 17) were used to obtain estimates of I_{tun} in amperes.

This same approach can be used to characterize reverse tunneling, with the only difference being the terminal voltages used. Those were 5.9V for V_{dd}, 5V for V_d, and 3.4V for V_{tun} (0 for V_{tun} during tunneling pulse).

3.6 Process variation of tunneling properties

The balance of global tunneling and reverse tunneling described in this chapter enables extremely accurate control of the floating gate voltage for an array of identical devices. However, in practice the devices are found to vary substantially in their tunneling properties. Figures 23 and 24, which depict data taken from the same FPAA using a slightly less accurate characterization method than the one presented in this chapter, illustrate some of the interesting features of the behavior of a large array of floating gate devices in response to tunneling.

Figure 23 depicts the result of tunneling on large populations of three different floating gate device layouts present in the FPAA. One striking feature of this figure is the very substantial spread (up to nearly 2V) in the floating gate voltages of nominally identical devices. This is in stark contrast to the convergence of floating gate voltages that would occur if all of the devices had identical tunneling characteristics. A second striking feature is the fact that the floating gate voltages roughly cluster by device layout. While one might naively expect the characteristics of devices with different layouts to be different, this result is still surprising because the layout of the tunneling capacitors themselves are identical in all of these devices. In theory, this should eliminate differences in the populations' means of the device characteristics.

A little more insight into this issue is obtained by analyzing the final values of the floating-gate voltage trajectories shown in Figure 23 in terms of the physical location on the array of the devices. Figure 24 shows the results of this analysis, which yields two clear conclusions. Firstly, the devices in the right-most column of the array have significantly different characteristics than those in the rest of the array. Secondly, the tunneling characteristics of a device depends on the specifics of the layout that surrounds it. Both of these observations suggest that the systematic differences in

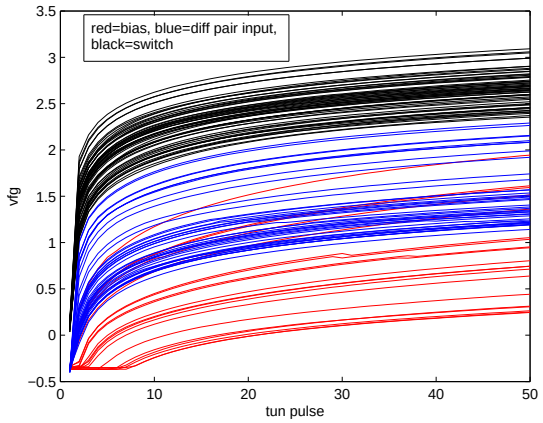
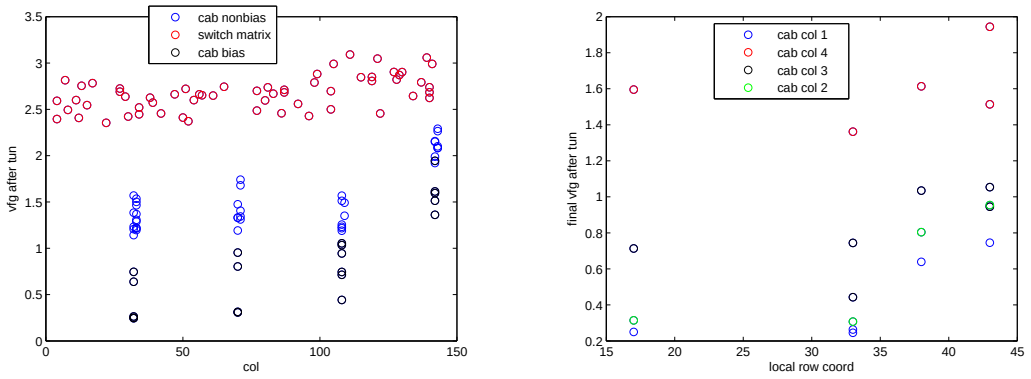


Figure 23: Floating gate voltage trajectories of three large populations of different device types during tunneling. The "switch" devices, shown in black, are the transistors that make up the switch matrix for routing in the FPAA. The "diff pair input" devices, shown in blue, are the input floating gates for the floating-gate based OTAs described in Chapter 2 (they are the same as the "HC" devices in this chapter). The "bias" devices, shown in red, are the programmable bias transistors for OTAs (they are the same as the "LC" devices in this chapter). The floating gate voltages spread significantly, and they also are seen to cluster by device type.

tunneling characteristics could be eliminated by appropriate placement of dummy layout near the tunneling junctions. The nature and extent of the dummy layout that would accomplish this is unknown at this time.



(a) (b)

Figure 24: Analysis of the distribution of the final floating gate voltages depicted in Figure 23, according to physical location on the array. In a), the values are plotted versus the column coordinate of the device. Because of the chip architecture, the switch devices are found at most column addresses, while the devices in the CABs (the OTA bias and differential pair devices) only exist on two column addresses in each of the 4 columns of CABs on the IC. The the OTA bias and differential pair devices found in the rightmost column of CABs tunneled significantly faster than the same types of devices in the rest of the array. In b), the OTA bias devices alone are plotted versus their location within a CAB (the local row address). Two of the locations in the cab (rows 38 and 43) are observed to tunnel faster than the other two locations (at rows 17 and 33).

Chapter IV

ANALOG COMPUTATION USING FLOATING-GATE ADAPTATION

As mentioned in Chapter 1, one desirable feature unique to floating-gate transistor technology is the use of this embedded long-term analog memory to perform learning/adaptation. As an example, consider the radial basis function circuit described in [49]. Its output approximates a diagonal-covariance vector Gaussian function of its 2 inputs, where the mean and variance parameters are set by programming floating-gate transistors. As such, it could be used in conjunction with a "winner-take-all" block to implement a classifier. One exciting possibility allowed by floating-gate transistors is to train such a classifier by enabling processes that program the floating gate using continuous-time feedback, and simply presenting training data. This model for classifier training eliminates the explicit computation of parameters from statistics of training data, which is a small step in the direction of the massively parallel self-organization that brains are able to perform during development.

In order to explore the possibilities of such an approach, an FPAA that allows continuous-time floating-gate programming is a useful tool. Since the circuit topologies that lead to the desired dynamics will be the subject of research, this FPAA should ideally allow the user to leverage reconfigurability of the circuits used for floating-gate injection. This inherently requires that the routing in the FPAA be capable of handling voltages sufficiently high to cause hot electron injection. This presents an interesting problem: if injection-level voltages are routed through a switch matrix of floating-gate transistors, how can the system be designed to prevent unwanted injection on the switch devices? The answer to this question hinges upon getting the

maximum achievable dynamic range of floating gate programming, an endeavor for which the findings presented in Chapters 2 and 3 provide an excellent foundation. In addition to careful choice of algorithms and parameters for injection and tunneling to maximize the range of floating gate programming, the different properties of floating gate nFETs and pFETs can be used to advantage in this endeavor.

This chapter covers the relevant properties of floating-gate nFETs, describes an "adaptive FPAA" that takes advantage of these devices to create circuits that do floating-gate programming in real time, and presents results from circuit measurements on this FPAA.

4.1 The core device: floating-gate nFET

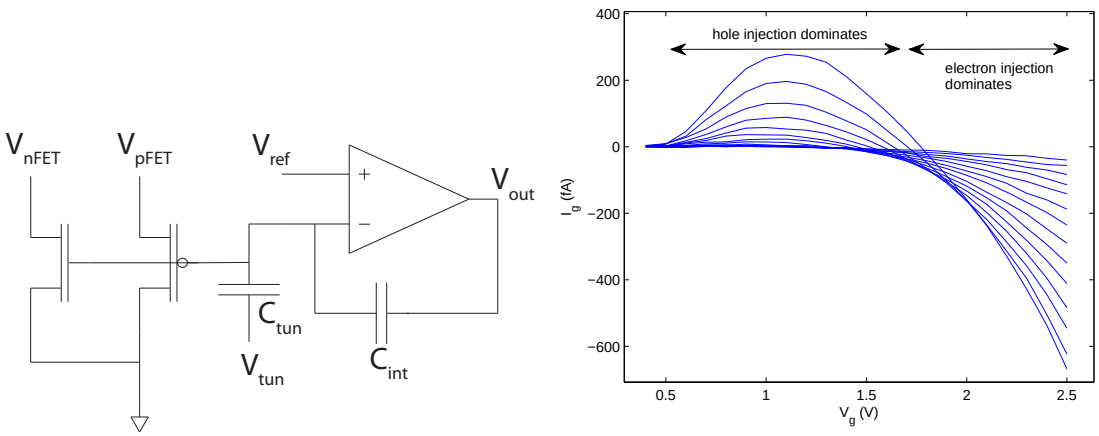
In a floating gate pFET, when impact ionization in the drain region creates hot electrons, the electrostatic fields in the device pull the hot electrons up through the gate oxide and onto the floating gate. In contrast, the fields in the drain region of an nFET tend to pull hot electrons into the drain terminal. Thus, nFET hot electron injection requires the generation of hot electrons in the channel of the device, before the drain depletion region is reached. This process is expected to require a high channel current and a high V_{ds}. At low current levels, hot hole injection can occur by exactly the same mechanism as hot electron injection in the pFET, but the holes need significantly more (1-2eV) energy to cross the oxide than electrons do. As a result, under low current conditions, hot hole injection is expected to occur in the floating gate nFET, but at a rate that is significantly lower than the rate of hot electron injection for floating gate pFETs with the same terminal voltages. This difference means that the "off" switches in a switch matrix of floating gate nFETs will be able to withstand higher voltages without appreciable injection occurring.

In order to quantify the effects of charge injection on the floating-gate nFETs under the desired ranges of terminal voltages, characterization was performed using

the circuit shown in Figure 25. The amplifier is used to fix the floating gate voltage at V_{ref}, and the slew rate of V_{out} provides a measure of the current being injected to the floating gate. An nFET and pFET share the same floating gate so that the charge may be easily manipulated by tunneling or hot electron injection in the pFET. As expected from the theory, the results show that hot hole injection is the dominant source of gate current when the channel current is low, and hot electron injection dominates when the channel current is sufficiently high. The results also agree with the qualitative predictions made by the theory regarding the rate of injection in the nFET compared to the pFET; the maximum current over this range of terminal conditions is approximately 1000x smaller than the corresponding maximum for a floating gate pFET.

To my knowledge, this thorough characterization of the dependence of the nFET gate current on terminal conditions, including the transition from hole injection to electron injection as the gate voltage is increased, has not been shown elsewhere. This characteristic itself is suggestive of some interesting possibilities for floating gate adaptation using nFET injection, since the process has a stable equilibrium (unlike pFET injection), and the same basic process can be used to add or remove charge from the floating gate (as compared to the use of two different processes in floating gate pFETs).

Because of the higher energies required for holes to cross the oxide barrier, hot hole injection has been shown to cause more rapid changes to the structure of the nFET device, and thereby to alter its properties by creating surface states in the drain region [63]. In order to quantify the effects of hot hole injection on the device's properties, stress testing was performed, in which 350 cycles of injection of 7.5 pC of charge were performed. The results are shown in Figure 26. The rate of hole injection



(a) (b)

Figure 25: Characterization of hot carrier injection for the floating gate nFET. a) Circuit used for characterization. The amplifier is used to fix the floating gate voltage at V_{ref}, and the slew rate of V_{out} provides a measure of the current being injected to the floating gate. An nFET and pFET share the same floating gate so that the charge may be easily manipulated by tunneling or hot electron injection in the pFET. b) Measured injection currents in the nFET for a range of gate voltages from 0.3 to 2.5V, and drain voltages from 4.4 to 6V. As expected from the theory, hot hole injection is the dominant source of gate current when the channel current is low, and hot electron injection dominates when the channel current is sufficiently high. The maximum current over this range of terminal conditions is approximately 1000x smaller than the corresponding maximum for a floating gate pFET.

declined significantly over the course of the experiment, and the I-V characteristic of the device under a gate sweep changed. Analysis of the I-V characteristic shows that the threshold voltage shifts by about 180mV, and the subthreshold parameter "kappa", describing the slope of the I-V characteristic, decreases as well. The conclusions from the study of the floating gate nFET are that the device will provide a good switch for routing injection-level voltages, that there is some potential for using hot electron and hot hole injection to enable "push-pull" control of the charge on an nFET floating gate, but that the characteristics of the device are significantly altered by extensive hot hole injection.

4.2 An FPAA with floating-gate adaptation

An FPAA that provides a reconfigurable framework for testing floating gate adaptive circuits was designed and fabricated. The properties of floating gate nFETs described in the previous section were leveraged to enable routing of injection level voltages through a switch matrix of floating gate transistors. The floating gate nFETs used for high voltage routing share their floating gate with a pFET, which is used to perform hot electron injection for targeted programming. Bi-directional tunneling, as described in Chapter 3, is used as a global operation to reset the floating gate voltages to a high level. This "turns on" all of the nFET switches, so in the targeted injection phase of programming, all of the switches except the ones that are to be left "on" are injected. In order to maximize density and minimize the number of nFET switches that must be programmed in this way, the high voltage-enabled nFET switch matrix is restricted to a local switch matrix in each computational analog block (CAB). The interconnect between CABs is implemented using floating-gate pFET switch matrix, and the high voltage and low voltage switch matrices are separated by a programmable nFET disconnect switch on each row of routing, as shown in Figure 27. Figure 27

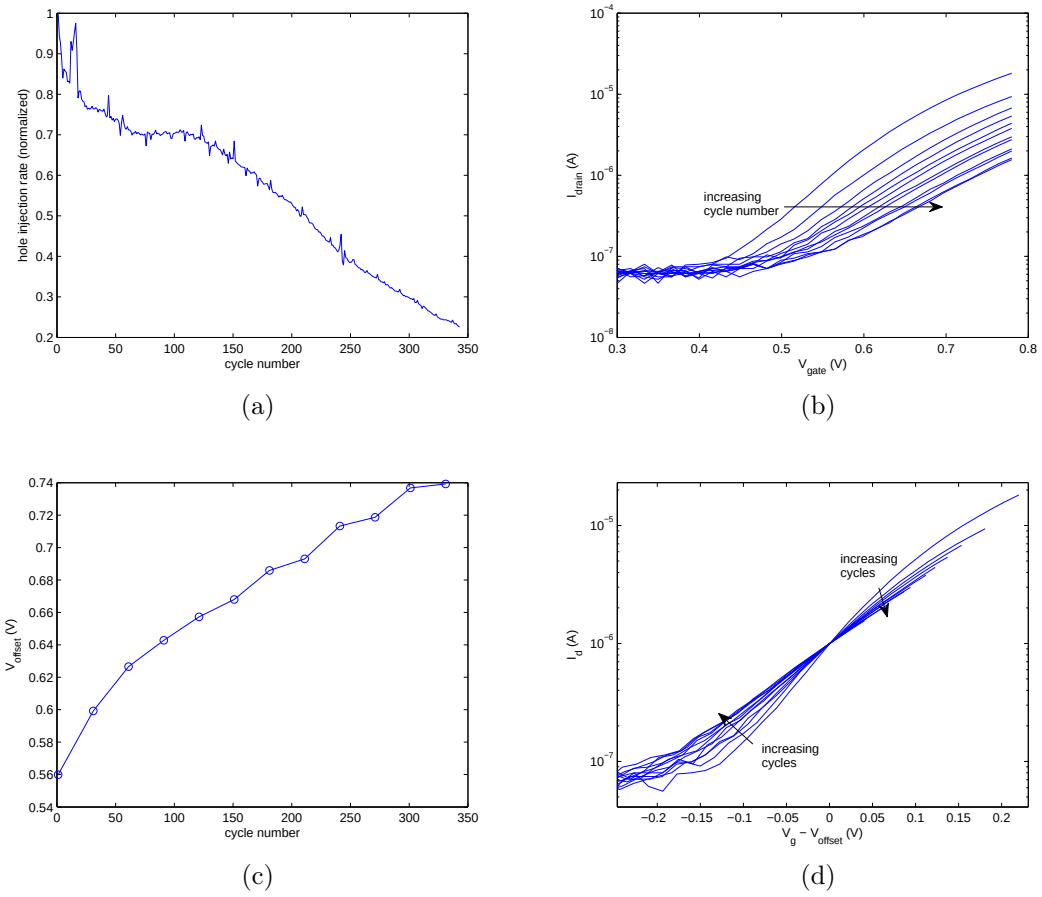


Figure 26: Results from stress testing of floating gate nFET by 350 cycles of 7.5 pC hot hole injection. a) The rate of hole injection declines significantly with the amount of charge injected. Similar sweeps to those shown in Figure 25 were taken after the test in order to verify that the peak of injection had reduced in amplitude, not simply shifted. b) Gate sweeps taken every 30 cycles show a significant change in the I-V characteristic of the device. c,d) The curves in b) were shifted horizontally so as to coincide at a drain current of 1 uA. The required shifts are plotted versus cycle number in c) (This roughly corresponds to threshold voltage shift), and the shifted curves are plotted together in d), showing the reduction in the slope of the characteristic with increasing cycle number.

also shows the a die photo of the IC, with a designation to indicate a single CAB and its associated switch matrix, as well as the schematics of the components that are found in the CABs. The most important component in the CAB is the adaptive floating gate, which has a self-contained amplifier for tunneling, and multiple FETs that can be compiled into circuits through the routing. The tunneling amplifier allows a voltage in the routable range (0-$V_{dd,inject}$) to control a higher tunneling voltage that is applied to the floating gate.

The description of the chip architecture is completed by noting that in addition to the global low voltage routing that connects the CABs, there are also six global vertical lines in the high voltage routing that span all of the rows in the array. One of these lines is in each column of tiles. These lines allow for high voltage nets to be shared across CAB devices in the same column.

As discussed earlier, the dynamic range of the voltages that can be passed by the nFET switches in the high voltage switch matrix is of paramount importance. In a first version of the IC, the source/drain terminals of the nFET switches in the high voltage switch matrix were grounded during programming of the array, as shown in Figure 28. This resulted in the devices with high floating-gate charge (e.g., all of the devices after global tunneling) to have a large voltage drop across the gate oxide in a direction that promotes reverse tunneling (as described in detail in Chapter 3). The resulting reverse tunneling current significantly reduced the maximum achievable floating gate voltage for the nFET switches, thus limiting their dynamic range to about 2.8V. A design change was implemented in the metal layers for the same wafer set in order to set the source/drain terminals of the nFET switches to V_{dd} during programming. This significantly reduced the effect of reverse tunneling, and thereby improved the dynamic range to about 3.8V, which is sufficient for hot electron

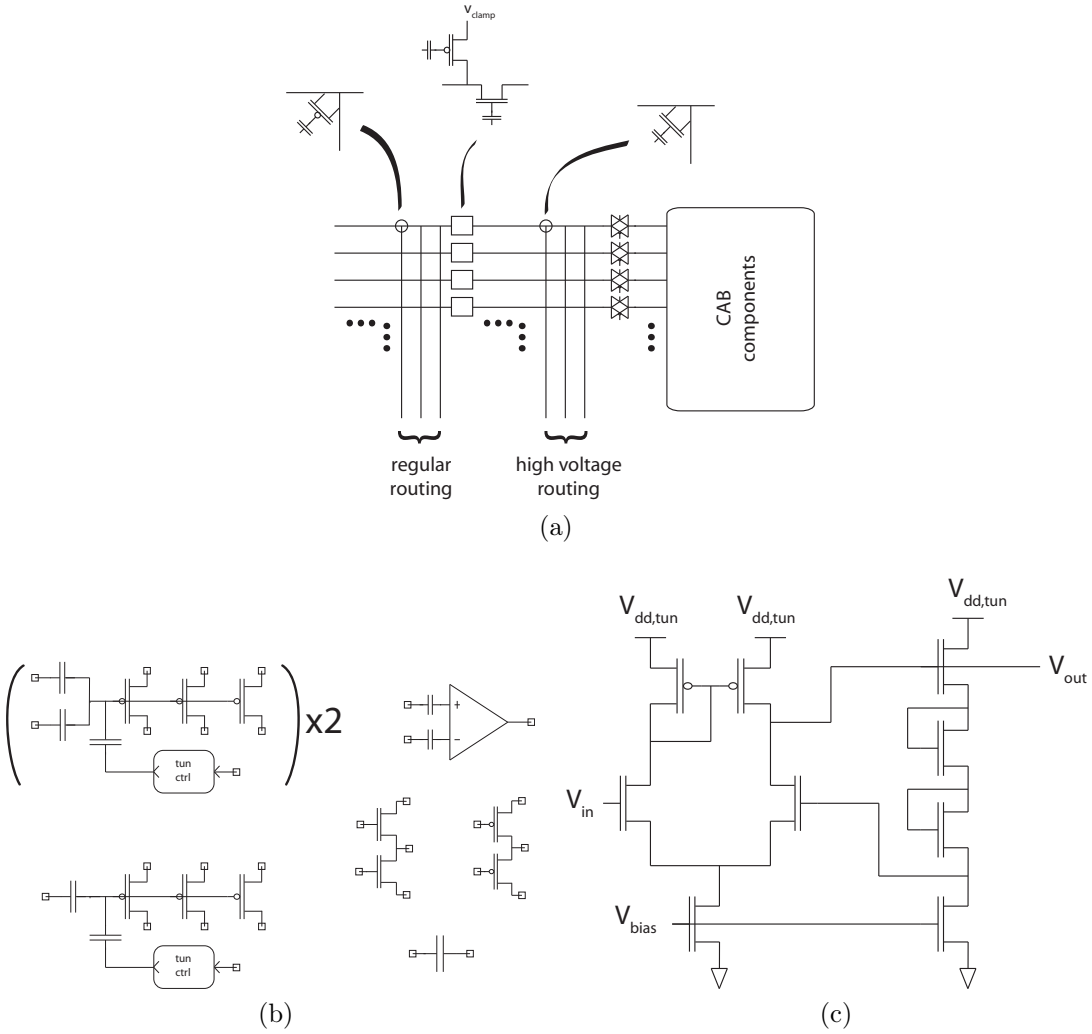
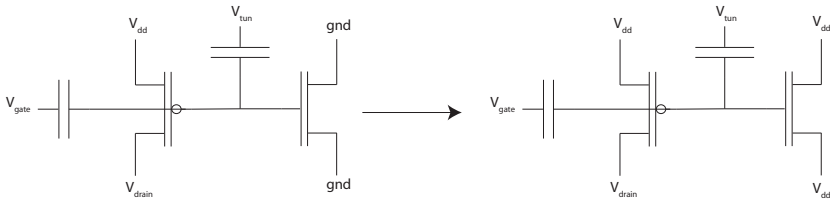


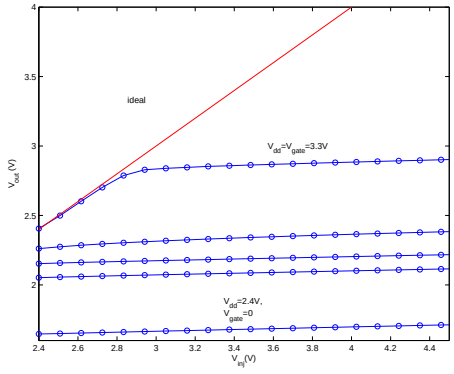
Figure 27: Overview of adaptive FPAA. a) Schematic of one tile (CAB and routing) of the array, highlighting the different floating gate FETs used for low voltage and high voltage routing, and for the disconnect switch between rows of high and low voltage routing. b) Schematic of all of the components found in a single CAB. The most important component in the CAB is the adaptive floating gate, which has a self-contained amplifier for tunneling, and multiple FETs that can be compiled into circuits through the routing. The tunneling amplifier allows a voltage in the routable range (0-$V_{dd,inject}$) to control a higher tunneling voltage that is applied to the floating gate. c) Schematic of the tunneling amplifier. All of the transistors in the amplifier are 13.5V devices.

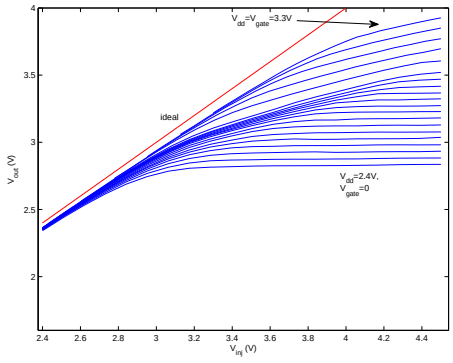
injection at reasonable time scales (adaptation on the order of seconds to minutes) in floating gate pFET devices like the adaptive elements in the CABs. The change to the nFET devices, as well as the comparison of the resulting switch performance, is shown in Figure 28.

While Figure 28 illustrated the performance of the high voltage switches for passing high voltages, the performance of the pFET and nFET switches at low voltages must also be taken into consideration. As the global gate coupling voltage V_{gate} and supply V_{dd} are increased in order to globally raise floating gate voltages and improve the high voltage range of the nFET switches, this global change to the floating gate voltage constitutes a trade-off between switch performance at high and low voltages. For the pFET switches, the increase in their floating gate voltages reduces their conductance at low voltages. This issue could be improved significantly by using different global V_{gate}'s for the nFET and pFET switch matrices, a solution which will be implemented in future designs. For the nFET switches, as the floating gate voltage is raised, eventually the "off" nFET switches begin to conduct appreciably when one of their terminals is at a low voltage. This tradeoff between "on" conductance at high voltages and "off" conductance at low voltages of the nFET switches constitutes the fundamental limitation on the dynamic range that can be routing using a single floating gate switch element, and the range is ultimately limited by the amount of charge that can be selectively injected on the "off" floating gates. Sweeps of the low voltage switch behavior for both the nFET and pFET switches are shown for a range of V_{gate}'s and V_{dd}'s in Figure 29. These measurements confirmed that with V_{gate} and V_{dd} at 2.7V, good switch behavior is achieved in a range from about ground to 3.6V.



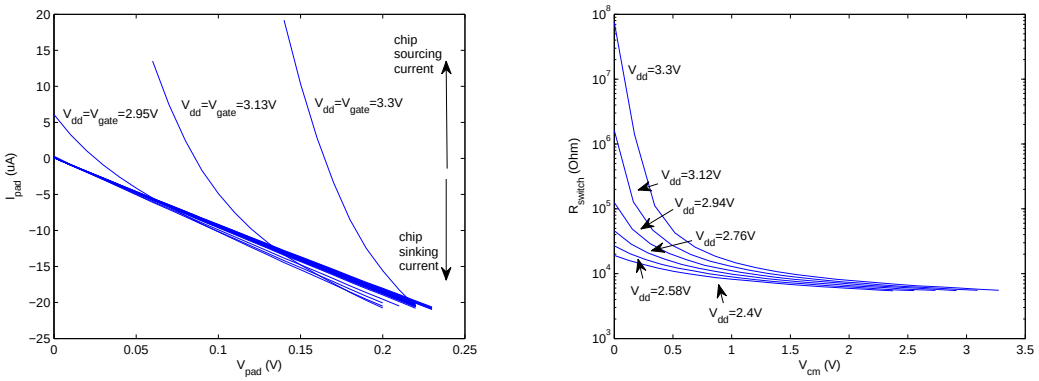
(a)





(b) (c)

Figure 28: Maximizing dynamic range of nFET switches by preventing unwanted reverse tunneling during array programming. a) The schematic of the nFET switch cell is shown before (left) and after (right) the metal mask change to the design. Keeping the terminals of the nFET switches at V_{dd} during programming minimizes the unwanted reverse tunneling that takes place. The result of a voltage sweep of a single switch is shown b) before the change and c) after the change. The operating range of the high voltage nFETs is seen to be extended by about 1V, a substantial improvement that allows orders of magnitude higher hot electron injection currents in the adaptive pFET CAB element. In both b) and c), the sweeps are shown for a range of gate coupling voltages and supply voltages (V_{gate}'s and V_{dd}'s), both of which can be used to globally increase the floating gate voltages, and thereby increase the maximum range of the floating gate nFETs.



(a) (b)

Figure 29: Low voltage performance of a) nFET and b) pFET switches. The currents in the nFET switch sweeps at high floating gate voltages are indicative of parasitic currents coming from "off" nFET devices in the array, and the result in b) illustrates the degradation of the pFET switch conductance at low voltage as the floating gate voltage is raised. Both of these effects can be mitigated by choosing $V_{gate} = V_{dd} = 2.7V$, which still results in acceptable high voltage nFET switch performance.

4.3 Software support- routing tool

Advanced software tools have been designed to create an interface to FPAAs that is easy to use for circuit experts and people with signal processing expertise who lack circuit knowledge. The tools are described in detail in [60], and depicted in Figure 30. Because the routing in this circuit requires special treatment of high voltage signals, and has different switch matrix topology due to the separate high and low voltage routing matrices, the adaptive FPAA described in this chapter does not fit easily into this framework. Instead, an intermediate tool was created using MATLAB. This tool requires the user to know the chip architecture and specify component placement in CABs, but it takes care of the tedious details of determining and accumulating lists of switch addresses for programming.

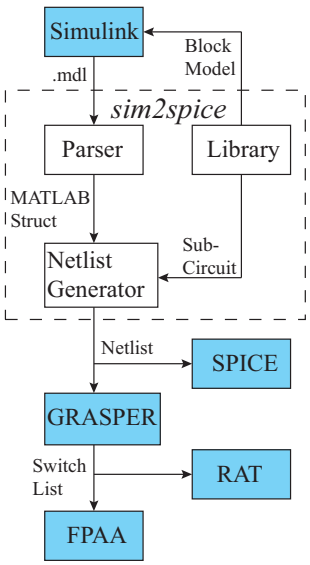
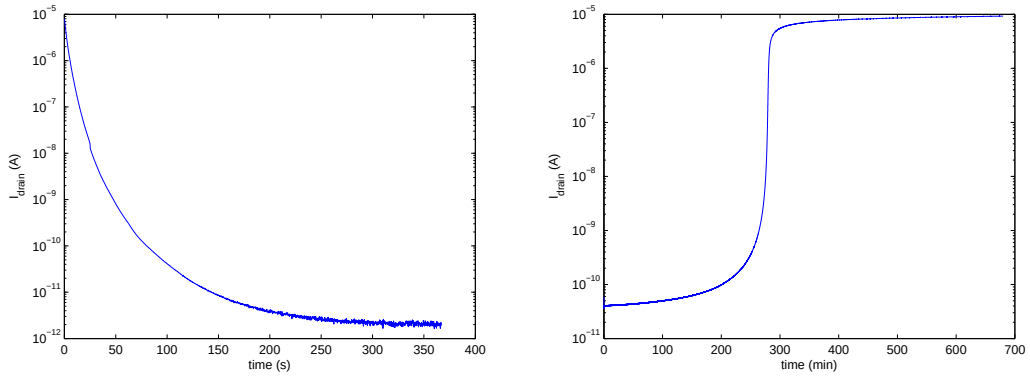


Figure 30: System of software tools that constitute the design environment for typical FPAAs. For the adaptive FPAA, this framework was replaced with a more low level MATLAB tool that provides a compromise between software development time and ease of use for the system.

4.4 Measured results- adaptive circuits

A good starting point for demonstrating the operation of circuits on the adaptive FPAA is demonstration of continuous time injection and tunneling in an adaptive pFET element from the CAB. The outcome of this experiment is shown in Figure 31. These results show how a device with fixed terminal voltages can conduct a current that varies of many orders of magnitude as its floating gate charge is changed. Also it serves as a good illustration of the characteristics of injection and tunneling (which were described in detail in Chapters 2 and 3).

One of the earliest examples of an adaptive floating gate circuit is the "autozeroing floating-gate amplifier" (AFGA), originally described in [33]. The AFGA, depicted in Figure 32, is a very compact implementation of a nonlinear bandpass amplifier, in which the low frequency time constant is determined by the simultaneous injection and tunneling processes. Because the parameters of tunneling and injection can

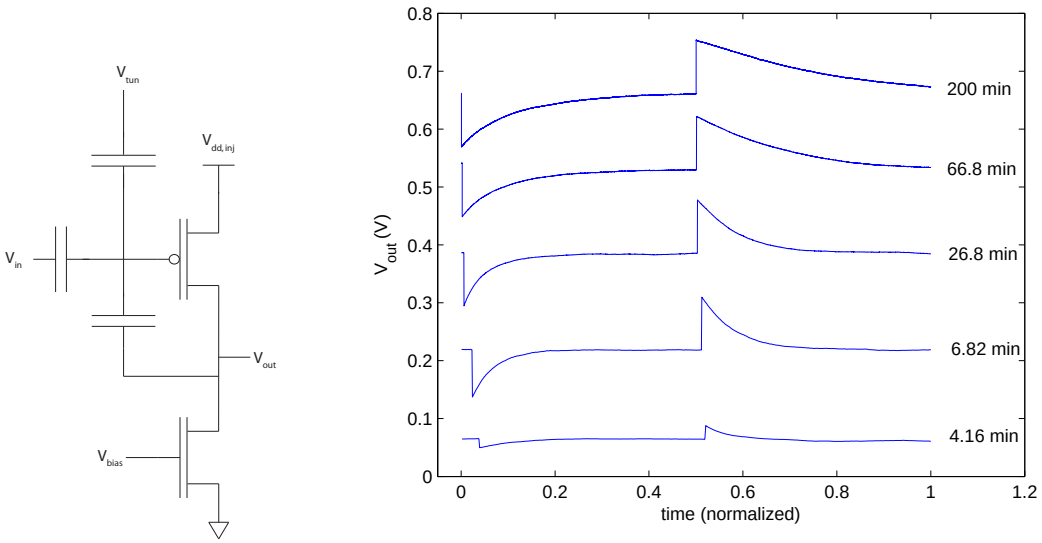


(a) (b)

Figure 31: Measured results from continuous-time a) tunneling and b) injection using the adaptive pFET element in the CAB of the adaptive FPAA. The qualitatively different dynamics of injection and tunneling are apparent. The rate of tunneling monotonically decreases over the course of the tunneling exposure, while the injection process is characterized by a positive feedback process that saturates at a large current level.

easily be adjusted to produce extremely low gate currents, extraordinarily long time constants can be observed in this circuit. Figure 32 shows measured step responses of the amplifier for a range of different time scales (achieved by changing the tunneling voltage V_{tun}, demonstrating time constants that range from under one minute to about an hour.

One interesting feature of the AFGA circuit is its second order nonlinearity. Because of the exponential dependence of the injection current on the source-drain voltage drop V_{sd}, when the output deviates downward, the resulting "restoring force" is stronger than that which results from an equal upward deviation. As a consequence, the DC component of the output voltage rises with increasing input amplitude. This effect is demonstrated in Figure 33, which shows the measured output of the AFGA circuit with a sequence of increasing amplitude square wave inputs.



(a) (b)

Figure 32: Autozeroing floating gate amplifier circuit (AFGA) originally described in [33]. The circuit diagram is shown in a) and the measured step responses, demonstrating time constants from under a minute to about an hour, are shown in b).

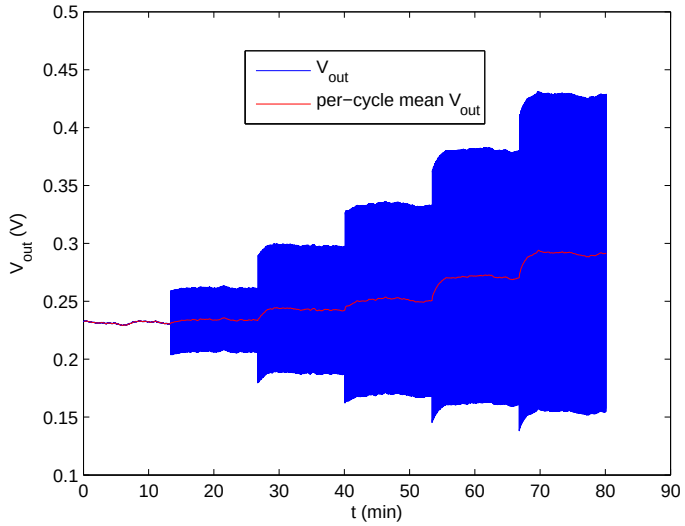
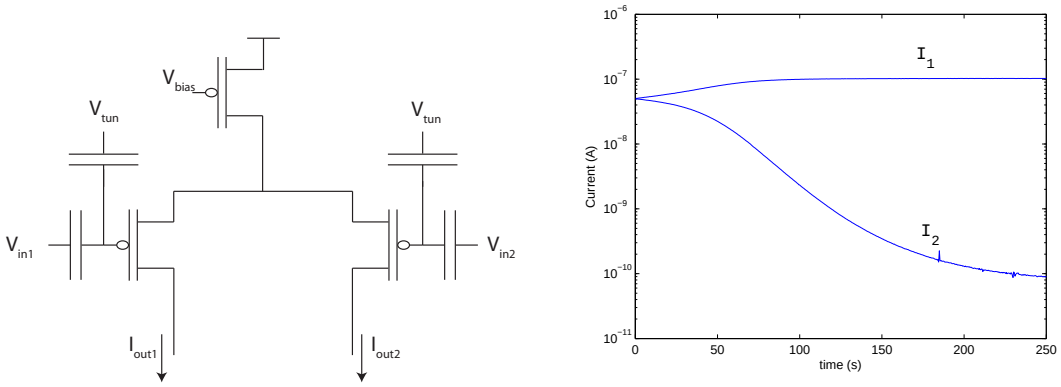


Figure 33: Measured output of AFGA circuit when the input is a square wave with successively increasing amplitude. The nonlinearity of the amplifier is clearly seen to result in an encoding of the signal amplitude in the output DC level.



(a) (b)

Figure 34: Floating gate differential pair circuit. The circuit diagram is shown in a), and the continuous-time injection curve is shown in b). The unstable dynamics eventually results in a condition in which all of the current is being conducted through branch 1, and essentially no current is flowing through branch 2.

The dynamics of adaptive circuits become even more interesting when multiple adaptive floating gates are coupled together in a single circuit. Perhaps the simplest example is the floating gate differential pair, shown in Figure 34. The floating gate differential pair has inherently unstable dynamics, since the branch of the differential pair that draws more current will also inject faster, which will result in it receiving even more current. This positive feedback process eventually results in one of the two devices conducting virtually all of the current from the tail current source. This process, starting from an initial condition of near perfect balance, is shown in Figure 34.

While the open loop differential pair's dynamics are unstable (divergent) and characterized by competition, incorporation of a load that reduces the field available for injection when the current increases can result in stable (convergent) dynamics. Specifically, implementing a fully differential OTA with common-mode feedback, as shown in Figure 35, results in a circuit that tends to settle back to the same equilibrium operating point, regardless of the input voltages. The step responses of this

circuit, for single-ended steps on each input, are shown in Figure 35. The step responses clearly show the circuit settling back to the stable equilibrium after transient perturbations caused by the input steps. The mismatch in the two branches of the circuit (transistor I-V characteristics as well as injection and tunneling characteristics) results in the equilibrium occurring at a point where the currents in the two branches are not equal. Consequently, the amplitudes of the step responses are not symmetrical.

The stable adaptation demonstrated in this fully differential amplifier provides the interesting possibility of compensating for an unknown offset in an unsupervised fashion during a "training phase", and then ramping down the supplies to allow the amplifier to operate without further changes to the floating gate charges. In order to demonstrate that the offset that is learned in the training phase will have the same impact after ramping down, differential sweeps of the amplifier before and after ramping the supply voltage down are shown in Figure 35.

The floating gate differential pair also finds a useful application in the "bump" circuit, invented by Tobi Delbruck [19], shown with floating gate inputs and active loads in Figure 36. This circuit produces an output current that is approximately a Gaussian function of the differential input voltage. This circuit is the core element of the radial basis function circuit described in [49]. The difference in charge on the two input floating gates provides an offset that sets the location of the peak of the Gaussian. Figure 36 shows differential sweeps of the input before and after two periods of adaptation. If the nFET bias current is just right, this adaptation would stably converge to an equilibrium rather than diverging, but in the case of this experiment, the nFET bias current was too strong, so the adaptation proved to be unstable. A load with common-mode feedback such as the one shown in Figure 36 could be used to achieve the stable adaptation.

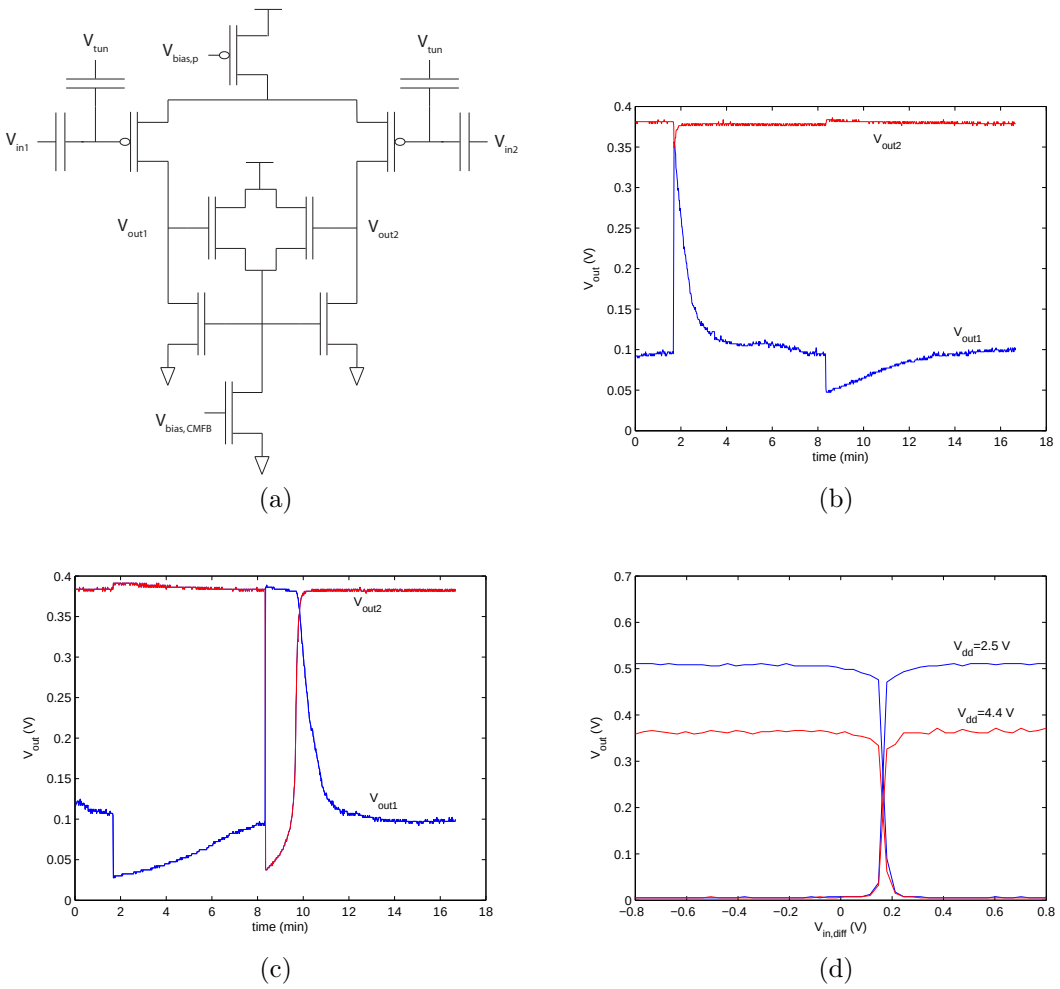
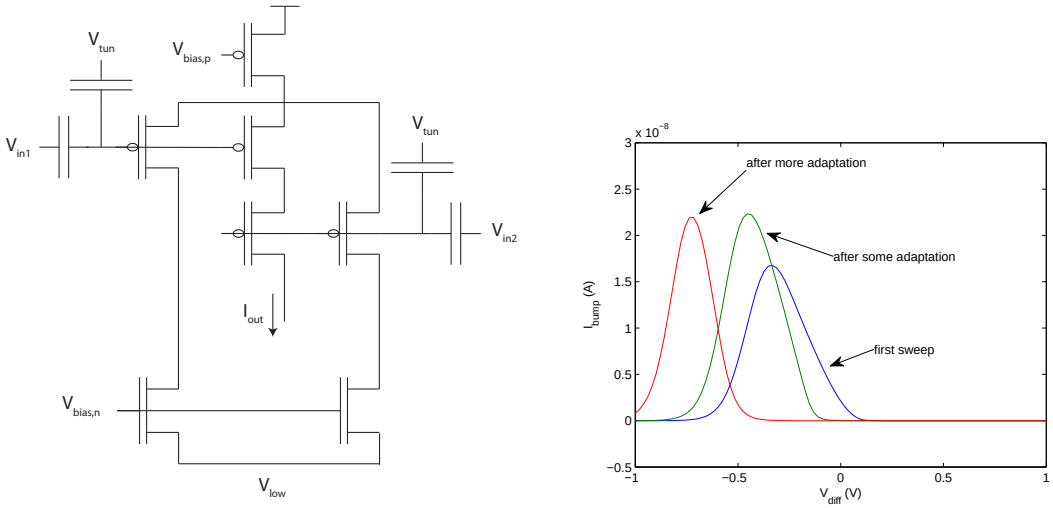


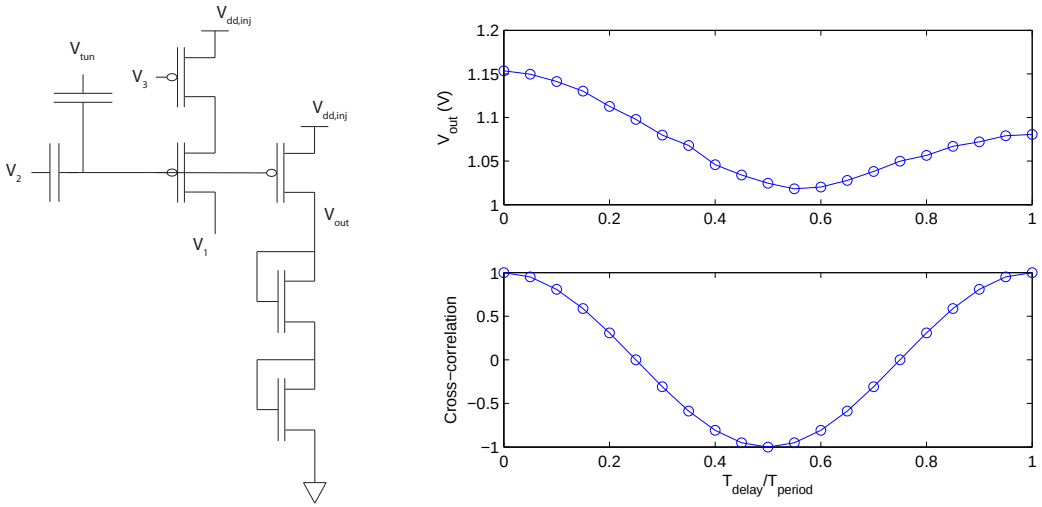
Figure 35: Fully differential OTA circuit with adaptive differential pair and common-mode feedback. The circuit, shown in a), exhibits floating gate adaptation that is characterized by a stable equilibrium, which is independent of the differential mode of the input voltages. The settling back to this equilibrium after perturbations is illustrated by the step responses on b) V_{in1} and c) V_{in2}. The asymmetry of the step responses is due to mismatch between the injection, tunneling, and I-V characteristics of the two branches of the amplifier. The adaptation could be enabled to allow the circuit to compensate an input offset, then disabled to allow the differential amplifier to operate around that differential offset. This capability is verified by differential sweeps of the amplifier before and after the supply voltage is ramped down to a level that prevents adaptation. The result of these sweeps, shown in d), shows that the offset that results from the adaptation is preserved when the supply voltage is reduced.



(a) (b)

Figure 36: Adaptive "bump" circuit. The schematic is shown in a) and differential sweeps are shown before and after two stages of adaptation in b). This circuit, under this bias condition, is another example of an unstable, diverging form of adaptation, much like the adaptive differential pair.

As described in detail in [42], the injection and tunneling in an adaptive floating gate transistor can provide a measure of the correlation between the signals on the drain and gate of the device. The basic intuition for understanding this is that if more time is spent with the drain and gate both at low voltages, then there will be more injection, resulting in a lower floating gate voltage. A circuit for illustrating this effect, shown in Figure 37, was tested on the adaptive FPAA. Sinusoidal inputs at 100 Hz were applied to the inputs of the circuit, and the relative phase of V_1 was shifted with respect to V_2 and V_3. Smoothly varying the phase shift from 0 to 2π results in a cross correlation between these signals that varies from 1 to -1, then back to 1. As shown in Figure 37, the equilibrium voltage V_{out}, to which the circuit settles after several minutes, approximately tracks the cross-correlation between the inputs. Some hysteresis was observed, as noted by the fact that the initial and final values of V_{out} are not the same (the origin of this hysteresis is not known).



(a) (b)

Figure 37: Adaptive floating gate correlation detector. The circuit, shown in a), is based upon a pFET source follower, and a second pFET on the floating gate drives a current into two series diode-connected nFETs. The voltage across the nFETs provides a measure of the floating gate voltage, which is determined by a balance of injection and tunneling processes. If the input signals V_1,V_2, and V_3 are highly correlated, the rate of injection increases, lowering the floating gate voltage. In b), the equilibrium output voltage is plotted versus the phase shift from V_1 to V_2 and V_3. The output voltage is seen to track the cross-correlation of the inputs.

Chapter V

A LEARNING-ENABLED NEURON ARRAY IC

The tools of CMOS circuit design and floating-gate transistor arrays can be brought to bear to create an elegant solution for implementing spiking networks. This chapter describes the full details of a neuromorphic IC beginning from the high-level design philosophy and architecture, then "drilling down" to the circuit implementations. The descriptions of the circuits are augmented by measured data demonstrating their functionality.

5.1 Design paradigm and resulting architecture

The primary goal for design of this neuron array IC was to implement as many neurons as possible while using biologically realistic transistor models for neurobiological computation, as well as enabling synaptic plasticity approximating the spike timing dependent plasticity (STDP) phenomenon observed in biological neurons. The model neurons are single-compartment models, i.e. the dendrites are modeled as ideal wires. Models for dendrites exist that would be compatible with this approach, but they were excluded here in order to maximize the number of neurons that can be implemented. Ease of use of the IC for simulating networks of neurons is greatly enhanced by an address-event representation (AER) module that communicates precisely timed spikes to and from the chip in real time. Figure 38 illustrates this approach, highlighting the parallels between this system and neurobiological systems.

The resulting neuron array IC consists of 100 model neurons and a configurable array of 30,000 synapses capable of creating arbitrary connectivity among the neurons.

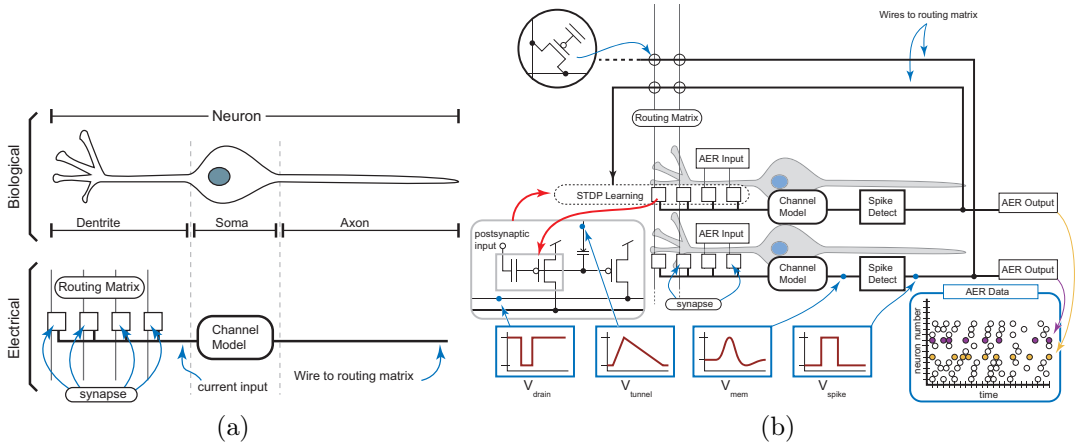


Figure 38: Illustrations of the design paradigm and its parallels to biology. In a), the model for a single neuron is depicted. In b), the model for two neurons is shown, and cartoons of some of the signal waveforms give a qualitative appreciation for the signal flow.

The IC was fabricated in 350-nm double-poly CMOS, and the die consumes a 5x5mm square. The model neurons are biophysically inspired channel-based models, which were originally introduced in [24], and which exhibit Class 2 excitability ([8]). These models constitute a favorable point in the tradeoff between biological realism and density (which translates to size of networks that can be modeled). The synapses are implementations of the "single transistor learning synapse" (STLS) described in [32]. They produce post-synaptic current (PSC) waveforms that approximate those measured in biological neurons, with programmable rise and fall times.

The STLS structure provides compact local storage of continuously variable synaptic weights (i.e., PSC amplitudes) by using floating-gate transistors, the core technology in electrically erasable programmable read-only memory (EEPROM). Two thirds of the synapses in the array are able to implement STDP according to the scheme described in [52]. In addition to the synapses between neurons, the system includes synapses from the AER interface onto each neuron. This allows for a wide range of complex input stimulation patterns to the networks being simulated.

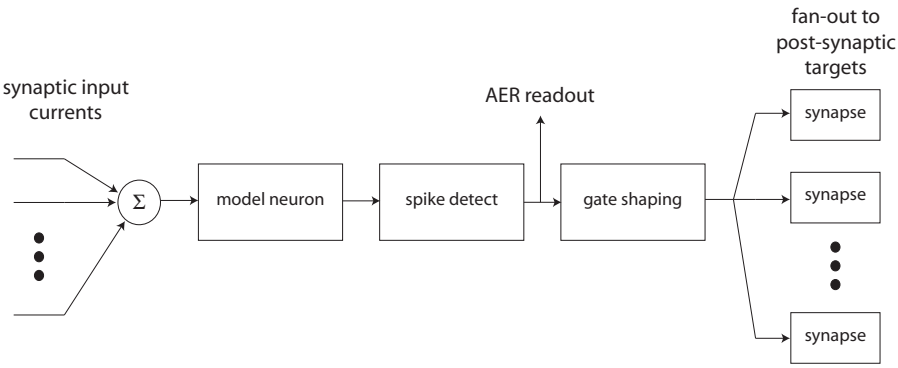
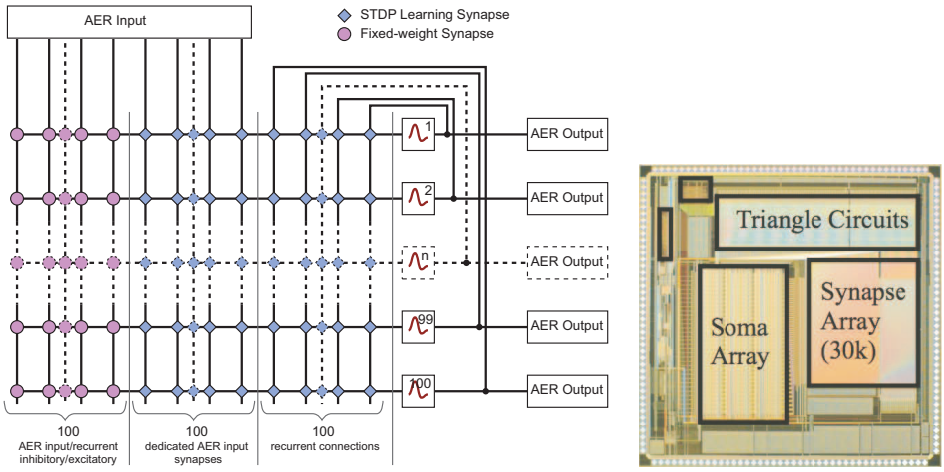


Figure 39: Signal flow for a single neuron in the neuron array IC. The synaptic input currents are summed on a wire. When the synaptic input currents are sufficiently strong, the model neuron emits a spike, which is picked up by the spike detector block and transmitted to the AER system and a gate waveform shaping circuit. The gate waveform shaping circuit conditions the pulse from the spike detector to create the desired time course for the post-synaptic currents on all of the neuron's output synapses.

The basic signal flow in the system is described in Figure 39, which depicts the signals processed by a single neuron in the system. The inputs to the neuron are an array of synaptic currents, which arrive asynchronously when the presynaptic neurons spike (just as is the case in a biological neuron). The currents all connect in parallel to the same wire, which automatically results in summation of the synaptic inputs. The model neuron is displays electrical excitability, just as biological neurons do. Specifically, if the perturbation provided by the total synaptic current is sufficiently high, a positive feedback process in the channel model leads to a stereotyped excursion of the membrane voltage, or an "action potential" in biological terms (more colloquially, a "spike"). When the model neuron emits a spike, this event is sensed by a "spike detector" circuit, the output of which is an asynchronous digital pulse. This pulse is reported by means of the the AER module, and it also simultaneously activates the gate waveform shaping circuit. This circuit transforms the digital input pulse into a waveform that can be applied to a synapse in order to produce a post-synaptic current with a time course that matches the ones produced by biological synapses. The output of a single gate pulse shaping pulse connects to all of the output synapses



(a) (b)

Figure 40: Top level architecture of the IC. In a) the connectivity of the synapse array is shown, including the partition of synapses into recurrent connections, AER inputs, learning-enabled, fixed weight, and configurable excitatory/inhibitory synapses. All of the STDP synapses are excitatory. In b) a die micrograph shows the amount of die area dedicated to the neuron models (Soma Array), the synapse array, and the gate waveform shaping circuits (Triangle Circuits).

for this neuron, just as a spike in a biological neuron triggers post-synaptic currents in the synapses at the ends of each branch of its axon.

The configurable synapse array features learning and fixed-weight synapses, synapses that are selectable as inhibitory or excitatory, and all-to-all connectivity between neurons and input channels from the AER module. Setting synaptic weights in this matrix allows networks to be created by interconnecting multiple instances of the block shown in Figure 39. The synapse array and a die micrograph of the IC are shown in Figure 40.

The system uses Matlab-based tools for defining network topologies and parameters. The syntax of this system closely follows PyNN, a network description language developed with the goal of facilitating interoperability of neuromorphic hardware ([17]). Some example code for specifying a network is depicted along with a graphical interpretation in Figure 41.

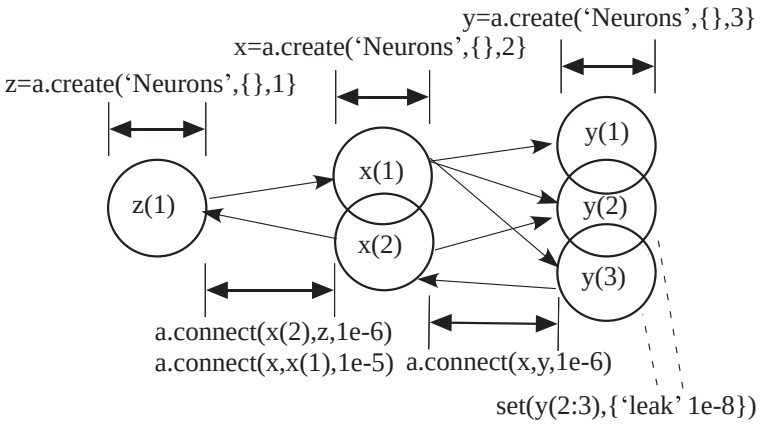


Figure 41: PyNN-based Matlab code for setting up a network simulation.

5.2 Circuits for spiking network implementation

The neuron consists of a circuit that models a population of voltage gated sodium channels and another circuit that models a population of voltage gated potassium channels. The interaction of the two produces a dynamical system that models the membrane dynamics at the axon hillock (the site of action potential initiation) in a biological neuron. The neuron models, which were originally described in [24], were designed to match the behavior of voltage gated channel populations in voltage clamp experiments, which are a foundational method of characterizing the dynamics of excitable tissues pioneered by Alan Hodgkin and Andrew Huxley [36, 35]. The channel models are shown in Figure 42, which also shows the manner in which they are connected in order to form a model neuron.

The voltage gated sodium channel in biology has two different gating mechanisms, one that closes the channel in the resting state, and one that closes the channel in the depolarized state after it has opened. This results in a step response (in a voltage clamp experiment) that is similar to that of a bandpass filter. That is, in response

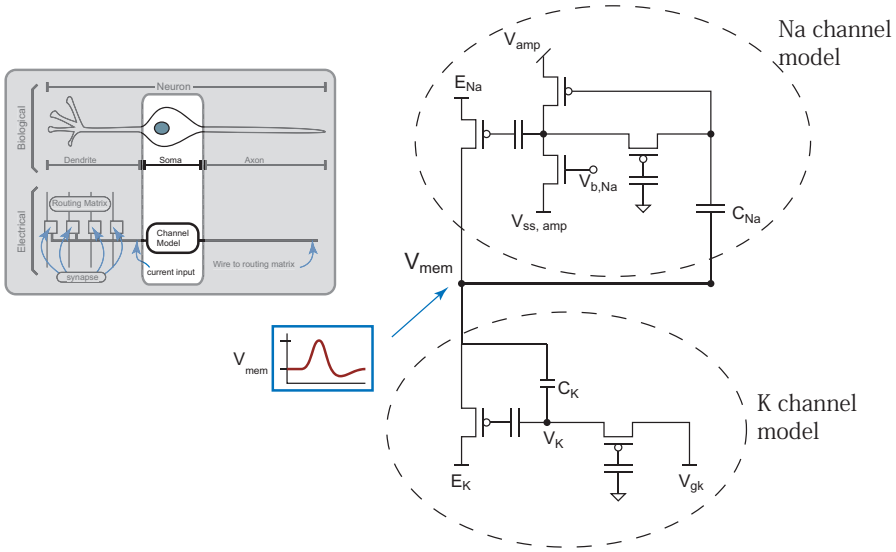


Figure 42: Models for voltage gated channels that constitute a model neuron. The voltage supplies and biases V_{amp}, $V_{ss,amp}$, E_{Na}, E_K, V_{gK} are shared by all neuron models in the array. The use of floating gate transistors in the models allow for their conductances and frequency responses to be programmed individually. This allows for networks of neurons with heterogeneous properties to be modeled.

to a step in membrane voltage, the sodium current initially rises, then (more slowly) falls back down. In contrast, the response of a voltage gated potassium channel has a single gating mechanism, and its steady-state conductance increases as the membrane is depolarized. In other words, its response to a step input in a voltage clamp experiment is similar to that of a low pass filter. With this guiding intuition, configurations of transistors which result in band pass and low pass step responses were found for modeling the sodium and potassium channels, respectively. The gains and poles of these filters can be tuned by various bias voltages in the circuit, which are implemented using floating-gate transistors on the neuron array IC, enabling theme to be set independently. This mitigates the effects of device mismatch, and also allows for simulations of networks of neurons with heterogeneous characteristics.

The neuron array IC includes a circuit for performing voltage clamp measurements on the channels, shown in Figure 43. Measurements of the responses to a depolarizing pulse in the two channels are shown in Figure 44. The rise and fall times of these

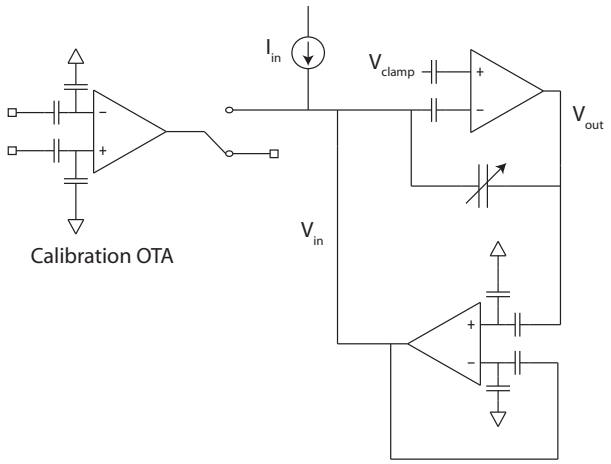
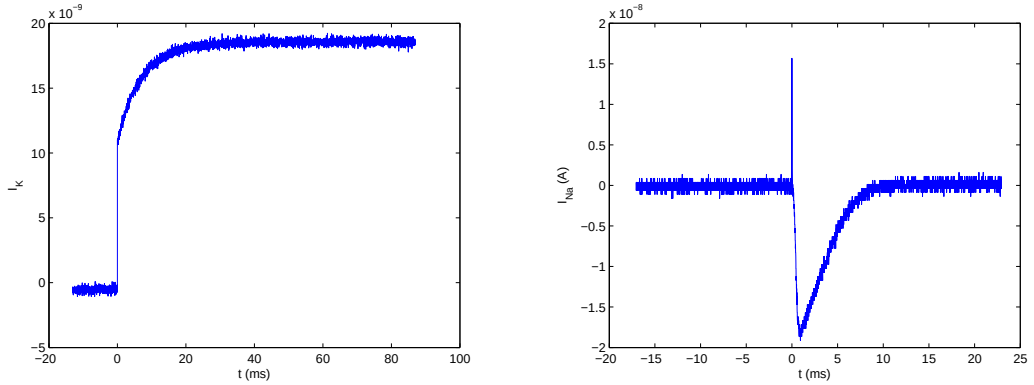


Figure 43: Schematic of transimpedance amplifier for making voltage clamp measurements. The voltage V_{clamp} is used to control the membrane voltage, and the channel model is represented by the current source I_{in}. The voltage $V_{out} - V_{in}$ is proportional to the input current, so V_{out} and V_{in} are buffered out to pads (buffers not shown in this figure). There is a 5-bit programmable variable compensation capacitor and an OTA that can provide a range of known currents in order to calibrate the amplifier's characteristic precisely. Small squares are used to denote connections to pads, and the capacitive attenuators shown on two OTAs have about a 10x attenuation factor.

responses were tuned to match realistic values for biological neurons. One flaw in the design manifests itself in the potassium channel response. The response to the step starts with an instantaneous step up, which then settles to a still higher value according to the programmed time constant, whereas a true low pass filter would not exhibit the instantaneous step. This nonideality is caused by two factors. Firstly, the transistor that forms the conductance to E_K is pFET with its n-well connected to a fixed potential rather than to the source. Secondly, the capacitance C_K does not dominate the total capacitance at the net labeled V_K. This nonideality of the potassium channel causes the potassium current to turn on faster than it does in a biological neuron, reducing the excitability of the model neuron. This makes it a little bit tricky to tune the properties of the model neuron.



(a) (b)

Figure 44: Step responses in a voltage clamp experiment of a) the potassium channel model and b) the sodium channel model. Note the instantaneous step in the potassium channel's response. This is a result of the fact that the n-well of the potassium conductance is not connected to its source, as well as the fact that the capacitor C_K does not dominate the total capacitance at the net labeled V_K in Figure 42. The response in the sodium amplifier has a very short transient that is an artifact of the voltage clamp circuit.

The synapses are arranged in groups of 100. Each group has one synapse onto each neuron, and all of the synapses in a group are activated by the same input (either an AER input or a spike coming from one of the neurons). There are two types of these synapse groups. One of the types implements an STDP-type learning rule, while the other has fixed synaptic weights. Schematics of these two types of synapse groups are shown in Figures 45 and 46. The fixed weight synapses can be configured as excitatory or inhibitory by means of a multiplexer that selects the reversal potential of the synapse (E_{exc} for excitatory and E_{inh} for inhibitory). This selection affects all of the synapses in a particular group. This constraint was an architectural necessity, but it is exactly analogous to the fact that biological neurons produce only a single type of neurotransmitter at all of their synapses. However, unlike biological neurons, the neurons on this IC are not necessarily constrained to form only excitatory or only inhibitory synapses. This extra freedom is a result of the fact that each neuron in the IC can connect to two synapse groups, one which is STDP enabled (and is therefore

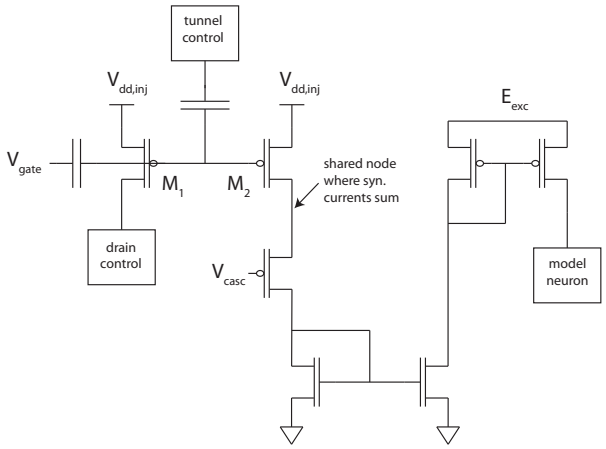


Figure 45: Schematic for STDP-enabled synapses. The synapse consists only of the two transistors M_1 and M_2 and the capacitors connected to its gate. The gate voltage V_{gate} is provided by the circuit shown in Figure 47. The current mirrors that level shift the synaptic current, the cascode pFET, and the tunnel and drain control blocks are all shared by all of the synapses that connect to a particular neuron. The cascode device prevents hot electron injection in M_2, while the current mirrors allow the current to come from the voltage supply E_{exc}. The tunnel control and drain control blocks, described later in this chapter in detail, mediate the hot electron injection and tunneling processes that implement the STDP learning rule.

excitatory), and one which has fixed weights (and can be configured as excitatory or inhibitory).

In both types of synapses, the gate waveform determines the time course of the synaptic current, and its amplitude dictates a baseline synaptic current amplitude. The floating gate charge on all of the synapse defines a multiplier that scales this baseline amplitude. The gate waveforms are generated by the gate waveform shaping circuits, shown with the spike detector circuit in Figure 47. The measured output from a gate waveform shaping circuit and a measured excitatory post-synaptic current (EPSC) are also shown in Figure 47.

The use of floating gate transistors for the synapses leads to a highly flexible simulation platform with a very dense storage of synaptic weights. A comparison

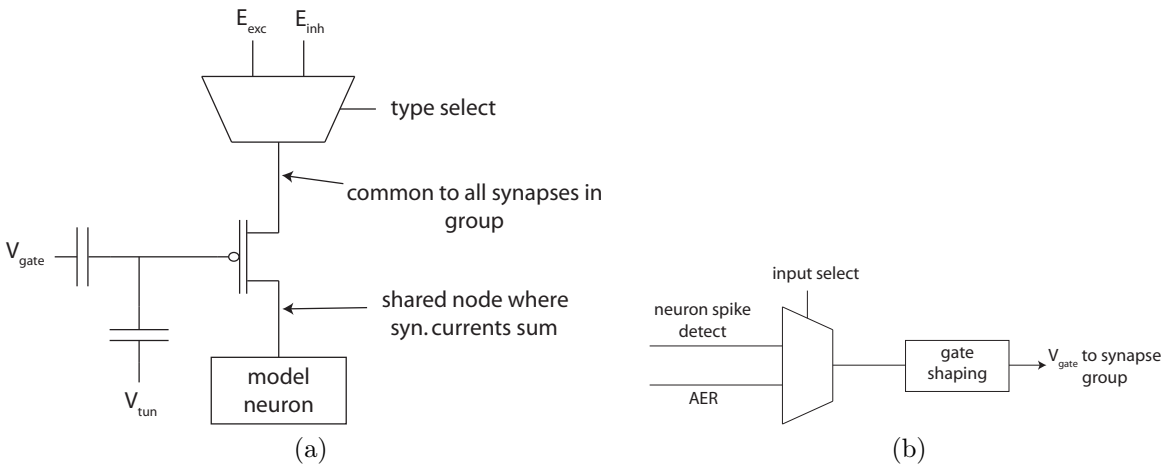
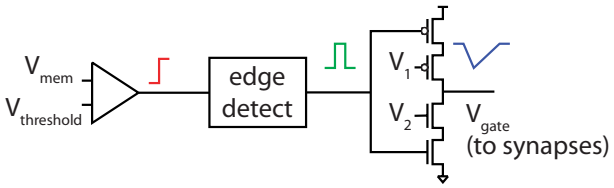


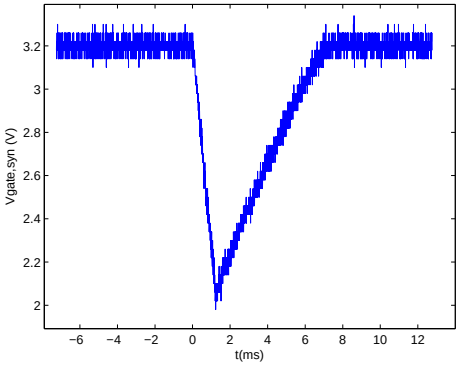
Figure 46: Circuits for the fixed weight synapses. a) Diagram of the synapse and multiplexer for selecting the polarity of the synapse group (excitatory versus inhibitory). Changing the logic controlling this multiplexer changes the reversal potential of the synapse (E_{exc} for excitatory and E_{inh} for inhibitory). The fact that this choice affects all of the synapses in a particular group was an architectural necessity, but it is exactly analogous to the fact that biological neurons produce only a single type of neurotransmitter at all of their synapses. b) Diagram of the circuit that selects the input for a group of fixed weight synapses. The polarity and input type are independently selectable for each of the 100 groups of fixed weight synapses.

of synapse density for other neuromorphic network simulator ICs is given in Table 4. The use of floating-gate transistors also allows for a fairly simple implementation of synaptic plasticity via STDP, and significantly mitigates the effects of process variation that increasingly plague IC technologies as we scale to processes with smaller feature sizes.

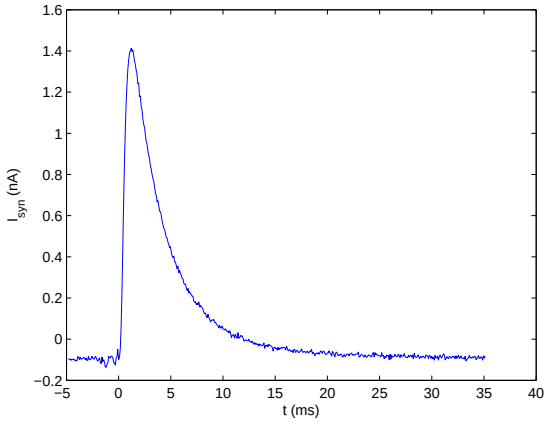
The AER module consists of a receiver and a transmitter. Functional block diagrams for these components are shown in Figure 48. The transmitter continually loops through the 100 neurons in the array, checking if its spike detector has emitted a pulse. When it encounters a neuron that has spiked, it immediately sends the address of that neuron to the microcontroller on the system PCB. The receiver delivers



(a)



(b)



(c)

Figure 47: Gate waveform shaping circuit and measured synaptic current. a) Circuit diagram for spike detector and gate waveform shaping circuit. The fall time of the gate waveform is set by the width of the pulse output from the edge detector (which is programmable by a floating gate voltage), and the rising and falling slopes are set by the voltages V_1 and V_2, which are also set by floating-gate based bias voltage generators. b) Measured output from the gate waveform shaping circuit. c) Measured excitatory post synaptic current. The waveforms in b) and c) were not taken in the same experiment, so the time courses of the two waveforms are not the same.

Table 4: Comparison of synapse density and function. The FACETS IC, Stanford STDP, INI IC 1, and INI IC 2 chips are described in detail in ([57, 56, 5, 38, 14]) . The normalized synapse area is computed by dividing the synapse area by the square of the process feature size.

Chip Description	Process Node (nm)	No. of Synapses	Synapse Area (μm^2)	Normalized Syn. Area
GT Neuron IC	350	30000	133	**1088**
FACETS IC	180	98304	108	3338
Stanford STDP	250	21504	238	3810
INI IC 1	800	256	4495	7023
INI IC 2	350	16384	3200	26122

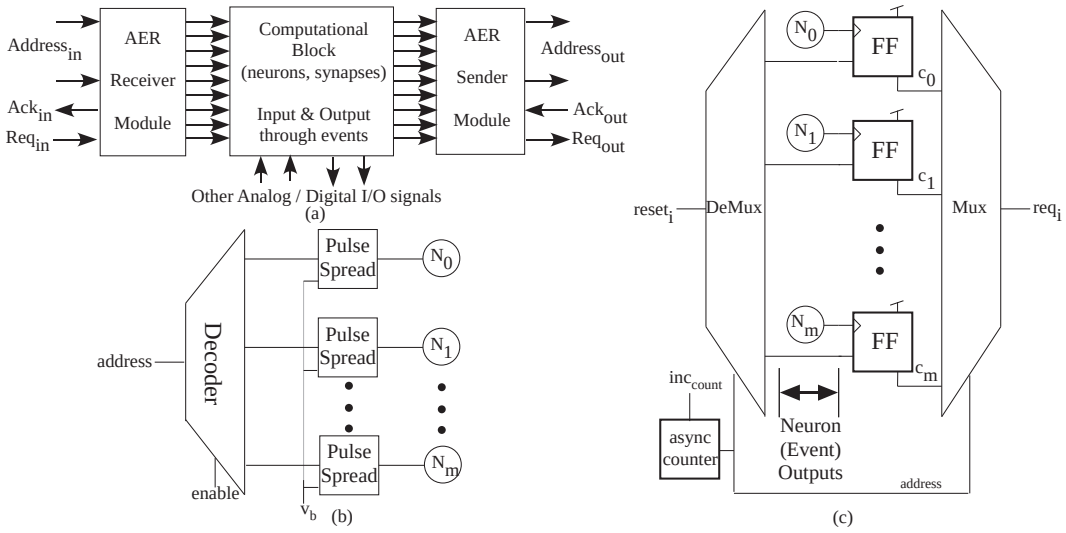


Figure 48: Functional block diagram of the AER module on the IC. a) Top-level signal flow depicting the various channels of communication with the microcontroller. b) AER receiver diagram. c) AER transmitter diagram

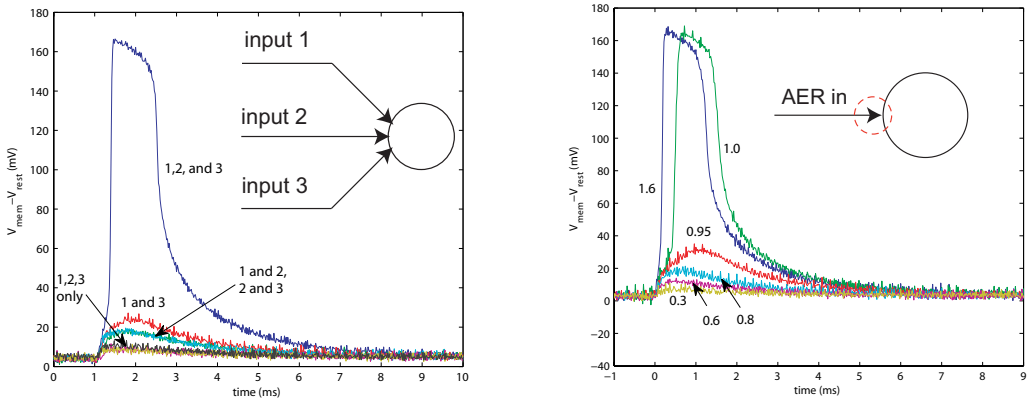
pulses to the groups of synapses that are allotted as AER input channels. When the microcontroller sends in an address, it immediately delivers a pulse on the line specified by that address. The communication with the microcontroller uses 4-phase handshaking, and the address bits can be configured to be sent serially or in parallel (this selection can be made independently for the receiver and transmitter). The logic for the AER module was synthesized from a couple dozen lines of Verilog code, and it is synchronous logic. The IC has a 25MHz ring oscillator to generate the clock, and the clock can also be supplied externally. At a clock speed of 25 MHz, the transmitter is able to scan all of the neurons in the array in 4us. This sets the maximum latency for the off-chip communication of any given spiking event, and is therefore an upper bound on the timing errors in the system, provided the total spike rate in the network is lower than about 200k spikes/second. The handshaking and communication of an event takes between 1 and 2 us, which limits this interface to 500k spikes/second. This rate corresponds to 5kHz spiking on all 100 neurons, which is well above the firing rates that are typical of biological neurons.

Since this neuron array IC was not an incremental change from a previous IC, but was a synthesis of many newly designed circuit blocks in untested configurations, special care was taken to design the IC with testing and debugging in mind. The outputs of most of the circuit blocks are multiplexed out to debugging pins, and many of the blocks inputs can be directly switched to demultiplexers coming from the pads. These design features proved invaluable throughout the course of testing the IC and understanding the results observed in network simulations performed on the IC.

5.3 Characteristics of neuron behavior

One of the key features of the biological neurons is the thresholding of inputs. This behavior can be illustrated by measuring the neuron's responses to an excitatory PSC of varying amplitudes, or similarly by measuring the response to a number of synaptic inputs that all add current to the neuron simultaneously. The results of these simulations using a single neuron are depicted in Figure 49. In both of these simulations, subthreshold depolarizations can be observed in response to weaker stimuli, while stronger stimuli cause an action potential on the target neuron, as expected.

Integrate-and-fire type neurons (as well as biological neurons) are often described in terms of their f-I characteristics, that is, the relationship between a DC input current and the firing rate. Although an analysis of the model neuron used in this IC concluded that it can undergo a Hopf bifurcation with respect to the DC input current [8], oscillations in response to a DC current were not observed, rendering the f-I characteristic a useless concept for this neuron model. Perhaps the disparity between the analytical prediction and the measured behavior is the result of the faster response of the potassium channel discussed in Section 5.2. In any case, the lack of



(a) (b)

Figure 49: Measurements of membrane voltage response to synaptic inputs, illustrating the excitatory threshold for spiking in the neuron model. All waveforms in this figure are averaged over about 20 captures in order to reduce the effect of power supply noise that is present in the measurement system. a) Three input synapses with approximately equal conductance are connected, and the response to various subsets of them are shown. b) The response to synaptic inputs of varying amplitudes is shown (the synaptic weight, normalized to the minimum weight that produced a spike, is shown).

a response to DC input currents is not necessarily a major concern in light of the fact that the inputs the neurons actually come in the form of time-varying synaptic currents. With this in mind, an analogous concept to the f-I characteristic could be termed a "p-I characteristic". The likelihood of a synaptic input current producing a spike is found to increase with increasing synaptic current amplitude. The result of characterizing this dependence for several different model neurons is shown in Figure 50.

An unintended, but interesting consequence the circuits used to create the synaptic gate waveforms is the fact that if a neuron fires two spikes in rapid succession, the amplitude of the post-synaptic current resulting from the second spike is larger than that of the first. This is similar to the phenomenon of paired pulse facilitation

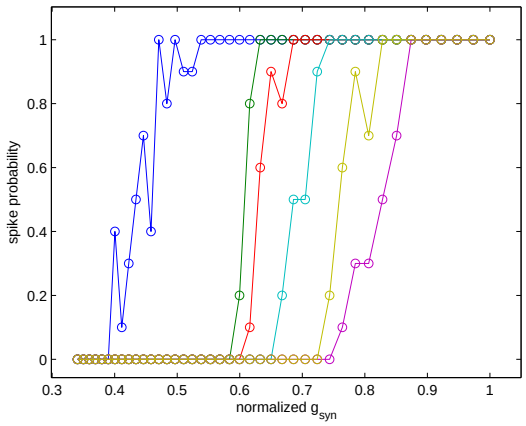


Figure 50: Measured relationship between synaptic input amplitude and likelihood of the synaptic input stimulating an action potential in the neuron. If the input is sufficiently weak, there is no chance that it can cause an action potential. If it is sufficiently strong, it is certain to cause an action potential. There is a relatively narrow region of input amplitudes where the probability is seen to vary continuously between these two extremes. The family of curves shown here is this "p-I" characteristic that was measured for 6 different neurons.

observed in biological neurons. In biological neurons, paired pulse facilitation results from the fact that the concentration of calcium ions in the presynaptic terminal (which controls neurotransmitter release) is still elevated from the first spike when the second spike arrives [65]. If the voltage output of the gate waveform shaping circuit is considered to be a representation of the presynaptic calcium concentration (which is at least a qualitatively valid analogy), then the mechanism of the paired pulse facilitation phenomenon in the neuron IC is the same. Figure 51 depicts measured outputs from the gate waveform circuit and the membrane voltage in a scenario that illustrates this paired pulse facilitation phenomenon.

Temporal summation is another important characteristic of neurons. Synaptic inputs currents add charge to the membrane voltage, and if the excitation is insufficient to trigger an action potential, the perturbation decays in time. When multiple input current pulses are applied rapidly enough, the perturbations to the membrane voltage

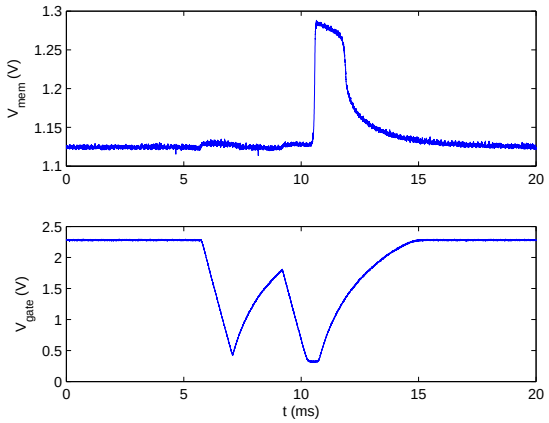
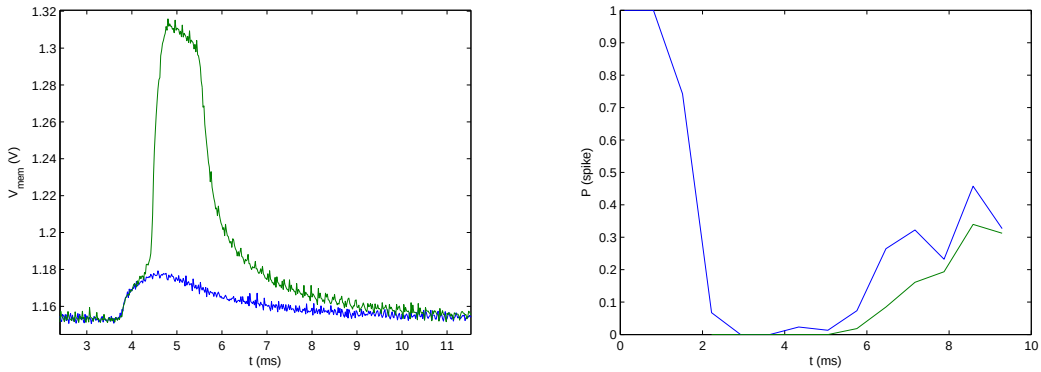


Figure 51: Illustration of the paired pulse facilitation phenomenon that is observed in this system. The second pulse arrives before the gate waveform has recovered to its steady-state value, which results in a greater amplitude gate waveform. This makes the second input stronger than the first, and thus it is consistently able to produce a spike. While the second gate waveform pulse appears to clip at about 300mV, this is actually due to the nFET input buffer amplifier that drives this signal off chip. The signal that goes to the synapses can slew clear to ground. These data were averaged for 10 trials to reduce high frequency noise.

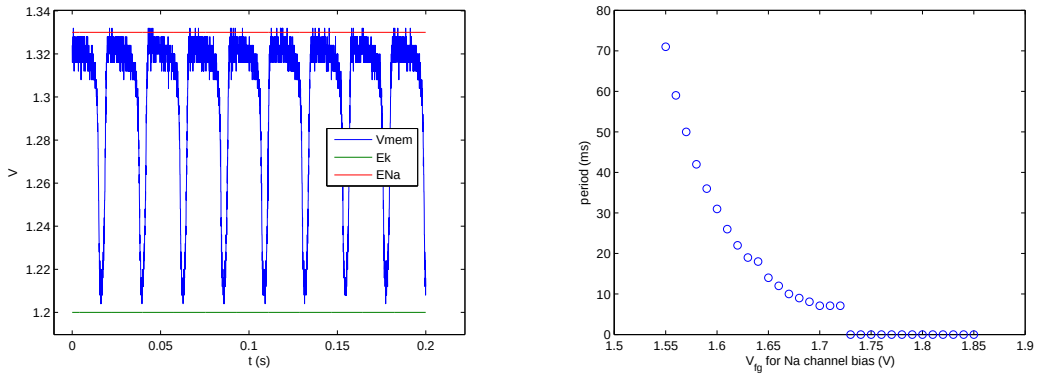
can accumulate to a level that does trigger an action potential. This phenomenon is illustrated in Figure 52, which tests the simplest scenario for temporal summation, i.e. the summation of two synaptic current pulses at different times to cause a spike. If the two inputs coincide they are certain to cause a spike, then as the time interval separating their arrival grows, the likelihood of producing a spike decreases. This plot also shows one other interesting effect. There is a range of time intervals at which the interaction of the two inputs actually reduces the likelihood of the second input causing a spike. This is due to the underdamped nature of the bandpass amplifier for the sodium channel model. The amplifier is ringing a little bit in response to the first input, and this leads to a period of time during which the neuron is less excitable than normal. This is somewhat similar to the phenomenon of subthreshold resonance that is observed in biological neurons [37].



(a) (b)

Figure 52: Data illustrating the effects of temporal summation of inputs. In a), the membrane voltage is compared for a single synaptic input (shown in blue) and two successive inputs (shown in green). In b), the probability of eliciting a spike by providing a pair of sequential excitatory inputs is plotted as a function of the timing difference between the two inputs. The two curves shown are from two different trials of the experiment. There is a clear effect of enhanced spike probability for inputs that arrive within about 2 ms, and a suppression of spikes if the inputs arrive between 2 and 6 ms apart. This suppression is the result of the underdamped characteristic of the sodium amplifier.

If the low corner frequency of the bandpass amplifier in the neuron model is sufficiently low, the system exhibits a stable limit cycle. This results in stable oscillations of the membrane voltage. A typical waveform for the membrane voltage is shown in Figure 53. In this dynamical state, the falling edge of the membrane voltage turns off the potassium channel very strongly, and the resulting rebound of the membrane voltage immediately reactivates the positive feedback in the sodium channel, causing another spike. The width of the spike is determined by the time constant of the high-pass corner of the bandpass amplifier (which corresponds to the time constant for inactivation of a sodium channel). Thus, as this time constant is lengthened, the spike width (and thus the spiking period) is lengthened as well. Figure 53 also shows the dependence of the period of spontaneous oscillations upon the bias voltage of a pFET that sets the DC current that flows in the bandpass amplifier. As the bias voltage goes up, the current goes down, which raises the DC operating point



(a) (b)

Figure 53: Properties of the stable oscillations of the membrane voltage. a) Typical waveform for the stable oscillations. b) Variation of oscillation period with the pFET bias voltage that sets the bias current in the sodium bandpass amplifier. As the bias voltage goes up, the current goes down, which raises the DC operating point of the amplifier. This, in turn raises the high pass corner frequency, which shortens the period of oscillations. At some point, the high pass corner frequency becomes sufficiently high that the rebound from the falling edge of the spike is not sufficient to trigger another spike. At this point, the stable oscillations cease.

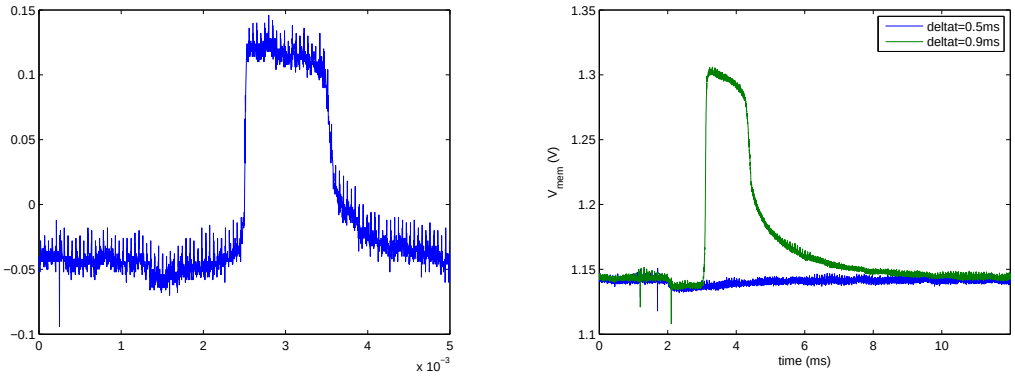
of the amplifier. This, in turn raises the high pass corner frequency, which shortens the period of oscillations. At some point, the high pass corner frequency becomes sufficiently high that the rebound from the falling edge of the spike is not sufficient to trigger another spike. At this point, the stable oscillations cease. In this state, the circuit is sufficiently excitable that pulses of input current can stimulate spikes, yet insensitive enough to maintain a stable resting membrane potential in the absence of any input stimuli. This behavior is the best qualitative match to the dynamics of the typical cortical neuron, and therefore it is the bias point that is chosen as the default for the neurons in network simulations.

One more important characteristic of neuron dynamics is the response to inhibitory inputs. In this model neuron, if the reversal potential for inhibitory synapses

(E_{inh} in Figure 46) is sufficiently low, then inhibitory synaptic currents cause a hyperpolarization of the membrane voltage, and the recovery from this hyperpolarization is a sufficiently strong perturbation to cause a spike in the membrane (as shown in Figure 54). This phenomenon is somewhat similar to that of post-inhibitory rebound that is observed in biological neurons [51]. However, it is a stronger effect than post-inhibitory rebound, and can in general lead to all inhibitory synaptic inputs triggering spikes, which is an unwanted behavior. This situation can be avoided by setting the inhibitory potential E_{inh} to a value very close to the resting membrane potential (as it is in the biological neurons). When this approach is taken, the membrane voltage is not significantly perturbed by inhibitory synaptic inputs that arrive in the absence of excitatory inputs. However, if the conductance of an inhibitory synapse is elevated at the same time that excitatory currents are adding to the membrane voltage, the inhibitory conductance will prevent the excitatory currents from depolarizing the membrane as much as they would have done in the absence of the inhibitory conductance. In neurobiology, this behavior of inhibitory synapses is referred to as "silent inhibition" or "shunting inhibition" [26]. An illustration of this mechanism of inhibition preventing an excitatory input from causing a spike is shown in Figure 54.

5.4 Circuits for synaptic plasticity

It was shown by Shubha Ramakrishnan that applying waveforms on the drain and tunneling voltage of a synapse following each post-synaptic spike, in concert with the gate waveform that is applied to the gate of a synapse following a pre-synaptic spike, results in a combination of tunneling and injection to the floating gate of the synapse in a way that approximates the STDP learning rule [52]. This relies on the fact that the drain pulse causes faster injection if it coincides with the gate waveform, and similarly the tunneling pulse will cause faster tunneling if it coincides



(a) (b)

Figure 54: Responses of neuron model to inhibitory input. a) A very small amplitude inhibitory synaptic current can result in a spike if the inhibitory reversal potential is too low. This unwanted behavior can be eliminated by setting the inhibitory reversal potential sufficiently high (to a value of 30-50mV below E_K). b) With E_{inh} set to an appropriate level, the effect of inhibition to prevent a spike can be observed. In this experiment, an excitatory input was delivered a short time after an inhibitory one. If the excitatory input arrives too soon after the inhibitory input (in this case, 0.5 ms), it fails to elicit a spike. If it arrives a little later, after the inhibitory conductance has begun to fall, then it causes a spike (both traces averaged over 10 trials to reduce high frequency noise).

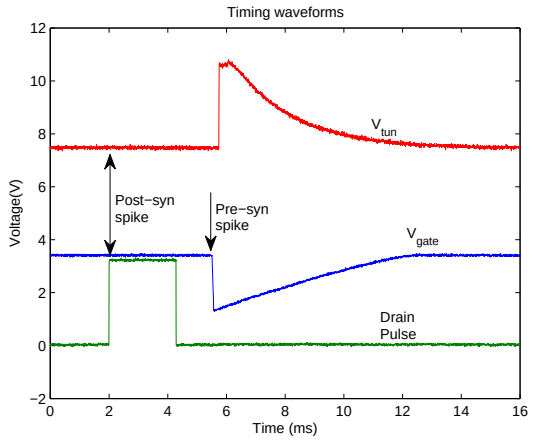


Figure 55: Waveforms that orchestrate the STDP learning rule. Arrows indicate the timing of the pre and post-synaptic spikes, as well as the fact that the drain pulse and V_{tun} are time-locked to the post-synaptic spike while V_{gate} is time-locked to the pre-synaptic spike.

with the gate waveform. Thus if a drain pulse shortly follows the post-synaptic action potential (with a timing that is about equal to the fall time of the pre-synaptic gate waveform), and a long tunneling pulse occurs some time after the drain pulse, the desired dependence of the change in synaptic weight upon the spike timing can be achieved. These waveforms are illustrated in Figure 55.

The basic signal flow for producing the tunneling, gate, and drain pulse waveforms is shown in Figure 56. There are two main branches in the spike detector, one that sends its inputs to the AER transmitter and the gate waveform shaping circuit (as discussed earlier and depicted in Figure 47), and a second that initiates tunneling and drain pulse waveforms. The circuits for the drain control and tunnel control are shown in Figure 57. Both circuits use a sequence of slew-rate limited inverters, "one-shot" circuits, and simple logic gates to produce a pulse of a programmable width that follows the post-synaptic spike by a programmable delay. In addition, the tunneling control circuit has a high-voltage slew-rate limited inverter that transforms the low voltage digital pulse the desired tunneling waveform at a sufficiently high voltage

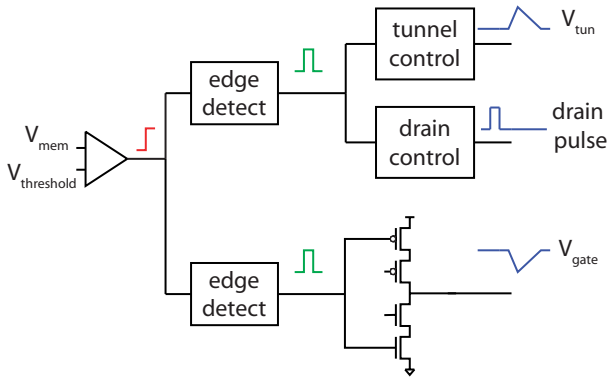


Figure 56: Signal flow for pulse timing circuits, including the gate waveform circuits and the circuits that generate the signals for STDP. Note that the voltage V_{gate} is driving the output synapses of the neuron that has just spiked, while the STDP signals (V_{tun} and the drain pulse) are connected to the input synapses of this neuron. The gate waveforms that influence the STDP rule on the input synapses of this neuron are the ones driving those synapses.

level to cause tunneling. Both circuits feature a logic bit that controls whether or not STDP is enabled (this bit can be set independently for each neuron in the array).

For completeness, the circuit diagrams for the one-shot circuits employed in pulse timing blocks are shown in Figure 58. The latch-based one-shot produces a pulse with programmable width in response to a rising edge on its input. Because of the hysteresis inherent in the latch structure, this circuit filters out glitches that accompany its input pulses, which is a desirable feature for a circuit whose input comes from a thresholded analog voltage (this is the circuit that implements the "edge detect" blocks in Figures 47 and 56). The falling edge one-shot produces a pulse with programmable width in response to a falling edge on its input, and it does not provide glitch filtering.

Figure 55, which was referenced earlier to illustrate the timing of the STDP waveforms, shows measured data from the outputs of the STDP timing circuits. The

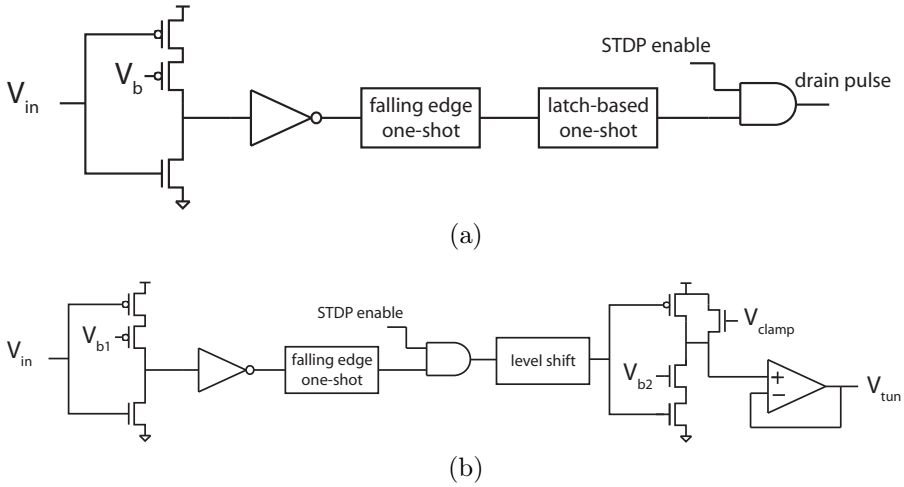
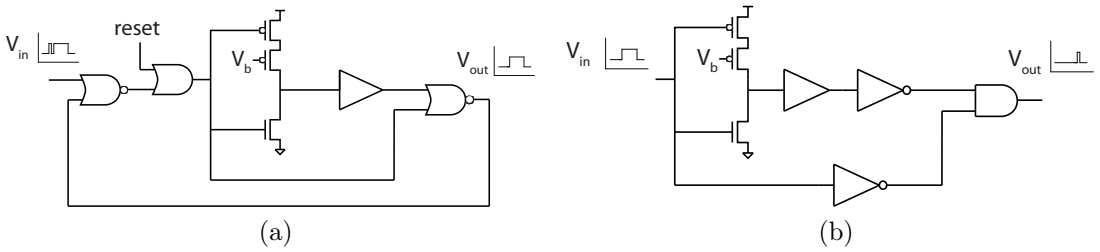


Figure 57: Circuits for timing and shaping a) the drain pulse and b) the tunneling voltage V_{tun}. The voltages V_b and V_{b1} are set by floating-gate transistors, and are used to set the delay between pre-synaptic spike and drain pulse or V_{tun} waveform, respectively. The voltage V_{b2} is also set by a floating-gate transistor, and it sets the fall time of the V_{tun} waveform. The global bias voltage V_{clamp} is used to set the baseline value for V_{tun}. The input V_{in} to these circuits is shared, and is the output of the edge detect block shown in Figure 56.

resulting STDP rule can be characterized by application of repeated spike pairings at a synapse, with controlled timing between the pre-synaptic and post-synaptic spikes. Varying this timing produces a characterization of the change in synaptic weight as a function of the "pre-post" spike timing, shown in Figure 59. This characteristic is the classic illustration of the STDP phenomenon, originally characterized in detail in electrophysiological data by Bi and Poo [10]. The characteristic shown in Figure 59 is qualitatively similar to the one described by Bi and Poo, with a few differences. The width of the peaks in the characteristic are only a few milliseconds wide, as opposed to widths of tens of milliseconds in the biological measurements. The width of the potentiation peak is controlled by the shape of the gate waveform, so extending the duration of the gate waveform would widen this peak. However, it would also increase the duration of the post-synaptic currents. The width of the depression peak could be extended by increasing the duration of the tunneling waveform, as was done in [52]. However, for reasons that are not understood at this time, decreasing the bias current



(a) (b)

Figure 58: Circuit diagrams of a) latch-based one-shot circuit and b) falling edge one-shot circuit. The latch-based one-shot produces a pulse with programmable width in response to a rising edge on its input. The latch-based circuit filters out glitches that accompany its input pulses. The falling edge one-shot produces a pulse with programmable width in response to a falling edge on its input, and it does not provide glitch filtering.

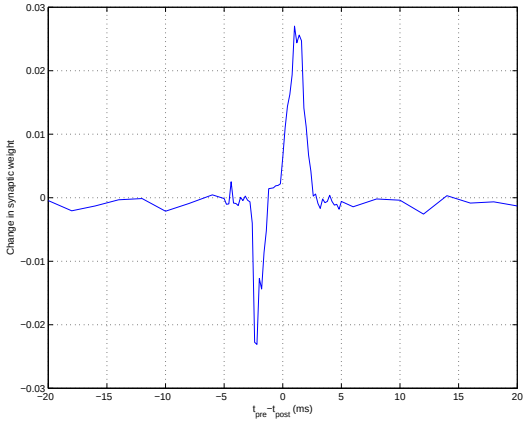
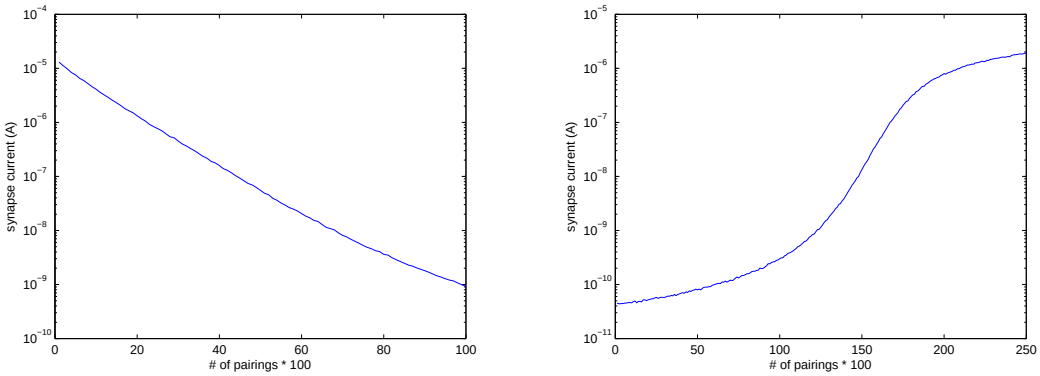


Figure 59: Measured STDP characteristic, showing dependence of the change in synaptic weight on the timing separating the pre-synaptic and post-synaptic spikes.

in the high-voltage slew-rate limited inverter that creates the V_{tun} waveform does not work to increase the duration of the tunneling waveform. Simulations and "pencil and paper" analysis have been unable to provide an explanation for this observed behavior. Investigations are underway to learn more about this problem, as well as to modify the scheme for STDP so as to decouple the width of the potentiation peak from the time course of the synaptic current.



(a) (b)

Figure 60: Long-term evolution of synaptic weights when the synapses undergo a) repeated post-pre spike pairings, and b) repeated pre-post spike pairings. in both cases, the synaptic weights are seen to vary over about 4 orders of magnitude.

One interesting question about the STDP rule is what happens over the long run to synapses that are repeatedly potentiated or repeatedly depressed? Do the synaptic weights get arbitrarily high or low, or is there a saturation that occurs? This question is of interest in the neurobiology community, and the answer to this question has consequences for how the synaptic plasticity process affects the signal processing properties of the neural network [11]. For the implementation of STDP described here, the answer is that many repeated depression events do indeed cause the synaptic weight to fall arbitrarily low (although the rate of decrease slows as the synaptic weight gets smaller). In contrast, with many repeated potentiation events, the synaptic weight eventually reaches a stable "steady-state". The synaptic strength at this stable equilibrium is enormously high, and is well above the minimum weight required for a single post-synaptic current to reliably elicit a spike. The evolution of the synaptic weights with repeated pulse pairings is shown in Figure 60.

5.5 System tools enabling network experiments

The hardware and software platform described in Chapter 2 is used for testing, debugging, and collecting data on this neuron array IC. Several more specialized tools were developed specifically for this system. These include automated routines for mismatch correction, a compiler for PyNN descriptions of networks, and microcontroller code for handling the AER communication. Each of these tools is described briefly here.

The bias currents controlling fall time and upward and downward slopes for the gate waveforms are set by floating-gate transistor programming. While floating-gate programming itself is fairly accurate, device mismatch between the programming transistor and the in-circuit transistor (M_1 and M_2 in Figure 4) results in significant variability in these currents. The relationship between currents in the programming transistor and in-circuit transistor can be inferred by measuring the gate waveforms. Knowing this relationship allows for a correction to be introduced when the floating-gate transistor is programmed. The result is a striking improvement in the matching among the gate waveform circuits, shown in Figure 61. This illustrates extremely valuable ability to cancel device mismatch that is allowed by using floating-gate transistors (without this ability, the designer would have to consider device mismatches as a fundamental limitation of the fabrication technology). A MATLAB software tool was created to automate the inference of the device mismatch characteristics and the application of the correction during floating gate programming. Because of the significant sensitivity of the subthreshold transistor to its gate voltage, the result shown in Figure 61 before mismatch compensation is nearly un-usable for network experiments. Consequently, the importance of this tool for the system cannot be overstated.

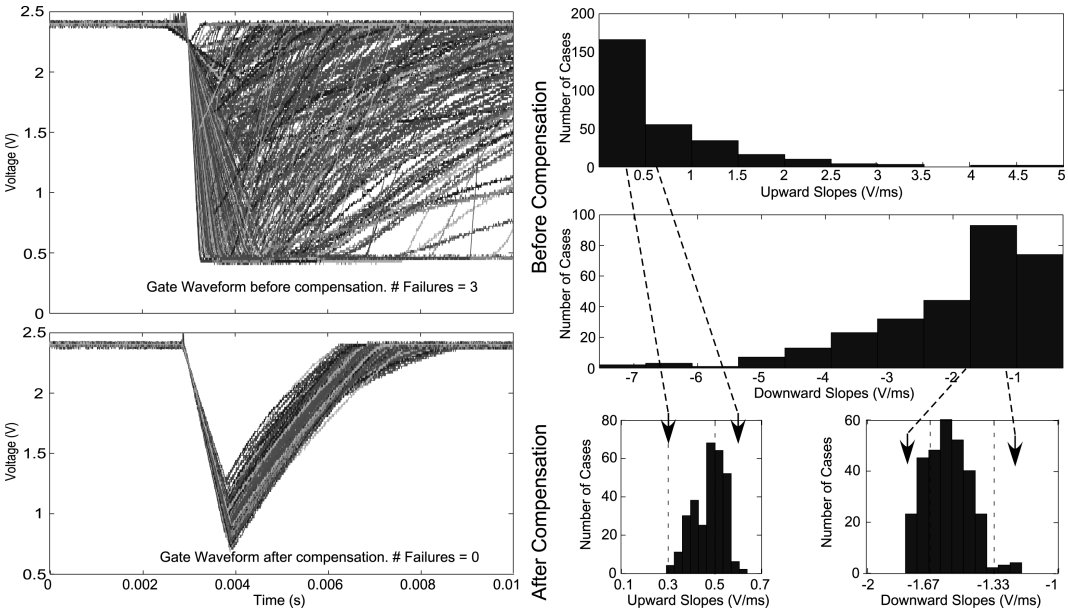


Figure 61: Results from mismatch compensation of gate waveform circuits. In the left two panes, the gate waveform outputs from 100 different circuits are shown before and after mismatch compensation. The panes on the right compare the histograms of slopes of these waveforms before and after the mismatch compensation.

A MATLAB software tool was created to support the definition of network topologies, synaptic weights, and neuron parameters in a syntax that is compatible with PyNN, as shown in Figure 41. This tool handles the mapping from these user-level descriptors of the network configuration to a list of floating-gate transistor addresses on the array and the targets to which these transistors should be programmed. It also handles the setting of the various digital logic signals to select synapse input types and polarity and to enable STDP on the desired set of model neurons.

Custom software for the microcontroller on the PCB was required in order to allow network experiments to simultaneously send and receive AER events with the low latency, high throughput, and minimal timing errors. An input stimulus can be defined by the PyNN code by specifying a set of input spike times and channels. This data is communicated to the microcontroller and stored. The microcontroller has embedded digital counters that allow it to time intervals without using the CPU. One of these counters is used for timing the inter-spike intervals for the AER inputs,

and one is used to time inter-spike intervals for the AER outputs from the neuron IC. When the counter for the input timing reaches the desired interval, the CPU receives an interrupt that is handled by sending the address of the AER input that is scheduled to receive a spike at that time. When the neuron IC initiates the handshaking process (indicating that the AER module has sensed a spike on the IC), the CPU receives an interrupt that is handled by reading and resetting the counter for AER output inter-spike intervals, reading the address of the neuron that spiked from the neuron IC, then storing the address and counter value that were read. This process can be scheduled to run for a fixed duration, or until a maximum number of spike events have been read. The two 14-bit counters have selectable clock division factors, allowing some flexibility in the tradeoff between temporal resolution and maximum interval that can be timed. When this AER experiment is complete, the saved spike data is sent to MATLAB, where it can be plotted in the form of a raster plot.

5.6 Power efficiency analysis

Power consumption has been discussed as one of the fundamental advantages of the neuromorphic approach to modeling neural computation. Now that the details of the neuromorphic system have been presented, it is possible to discuss this aspect in a concrete way in terms of the power efficiency of this system. The calculation of the power required by this neuromorphic approach can be calculated as follows:

Each neuron requires a 1nA bias current for the steady-state current through the channels, 2 1nA bias currents for the Na amplifier circuit, 3nA static current for the spike detector amplifier, and 3 1nA bias currents for the tunneling amplifier. These bias currents are drawn from voltage supplies of 0.2V, 1.2V, 2.4V, 3.3V, and 14V. This consumes about 33 nW of static power per neuron. Each gate waveform circuit requires 3 1nA bias currents from a power supply of 3.3V, resulting in about 10nW static power consumption. The AER module has 9 gates that are clocked at 25MHz,

which consumes about 60uW of static power (estimating the capacitance per gate at 25fF).

Each spike is estimated to consume approximately 170pJ in the model neuron circuit, and the resulting gate waveform lines have a swing of about 2V on a capacitance that is about 1pF. Accounting for the fact that there are two recurrent gate waveform circuits that are activated by each neuron, this consumes 8pJ of energy per spike. In the pulse timing circuitry for the gate waveform and the STDP circuits, each spike causes a 3.3V swing on 56 logic gates, which have approximately 25fF of capacitance, which consumes approximately 15pJ per spike. Each AER input and output event requires 12 transitions of 3.3V digital lines (8 address and 4 handshaking), which can be estimated at 15pF capacitances, which consumes in 2nJ per AER input or output. The current that flows in each synapse in response to a presynaptic event can be estimated as a 2nA current that is sustained for 2ms from a 6V supply. The same current can be used as an upper bound on the current in the indirect FET that is used for injection. Accounting for both of these currents in the synapse gives an estimate of 48pJ per synapse event.

Given all of the above sources of power dissipation in the system, it is clear that system power consumption in a simulation will depend on the network topology and activity (see Chapter 6 for analysis of power in the specific networks described there), but as an example, consider a fully connected network with 100 neurons and 200 AER inputs (for a total of 300 input synapses per neuron), with each neuron and the AER inputs spiking at an average rate of 10 Hz. The power consumed by this network is estimated at 200uW. By comparison, consider a method based on integrating differential equations, using 4th order Runge-Kutta method and time step of 50us. The neurons require about 200 32-bit multiply-accumulate (MAC) operations per time step, while the synapses require about 20. Using the highest efficiency implementation that has been reported, 100pJ/MAC [45], neglecting power used in

Table 5: Estimated power consumption for a neuromorphic simulation of 100 neurons and 30,000 synapses, wherein each neuron spikes at an average rate of 10 Hz. The total power of 204 uW is about 6000 times smaller than the estimate of the power required for running the simulation using numerical integration.

Load	Power consumption
AER	183 uW
Synaptic currents	14 uW
Analog circuits (static power)	6 uW
Neuron spikes	0.2 uW
Total	204 uW

communication between processor(s) and memory (which is likely to consume even more power than the actual computation, but is more difficult to estimate because it depends on the implementation), this simulation would require approximately 1.25W, almost 6,000 times more power than the neuromorphic implementation.

The contributions of the total power dissipation by the AER communication, static power of the analog circuits, synapse currents, and neuron currents are given in Table 5. Examining the relative contributions highlights the fact that the AER communication consumes almost 90% of the system power. In a research roadmap wherein the role of AER is restricted to facilitating an easy user interface for spiking inputs and outputs of the whole network model, one expects that the number of neurons and synapses could be increased by a much greater factor than the AER inputs and outputs. This is because the number of primary sensory afferents or motor efferents from the human brain is a negligibly small fraction of the total neurons. Thus, as the system scales, the power consumption in the neuromorphic system should be dominated by the neuron and synapse power consumption, which, based on this analysis, are about a factor of 60,000 more efficient than the implementation based in numerical integration.

Chapter VI

COMPUTING WITH SPIKING NETWORKS ON NEURON ARRAY IC

6.1 Introduction

It has been said that in order to fully understand something, one must figure out how to build it. While this blanket statement may not be true in all cases, neuromorphic engineers believe that developing neuromorphic hardware will ultimately lead to a better understanding of biological neural systems and the principles upon which their computation is based. Neuromorphic systems are designed with the expectation that they will be useful for improving our knowledge of neuroscience and computational science. These sciences in turn inform the design of improved neuromorphic systems. This feedback path is illustrated in Figure 62, which emphasizes the role of computational experiments performed on neuromorphic hardware. Experiments on neuromorphic hardware not only inform future neuromorphic designs, but also have significantly different constraints than numerical simulations have. Thus, such experiments are expected to be a stimulant of creative approaches to thinking about neural systems and computation. This chapter describes results from real-time simulations of networks with up to 100 channel-based model neurons, which were run on the neuromorphic IC described in Chapter 5. These networks perform computationally relevant functions such as arbitrary spatio-temporal pattern generation and recognition, "winner-take-all" competition, stable generation of rhythmic outputs, "volatile memory", and a graph search algorithm that has applications for path planning in robotics.

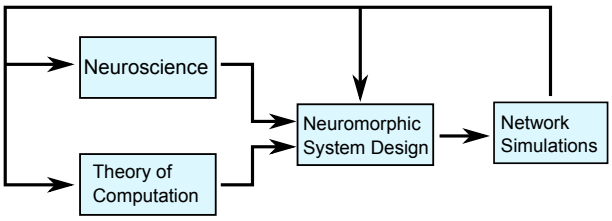


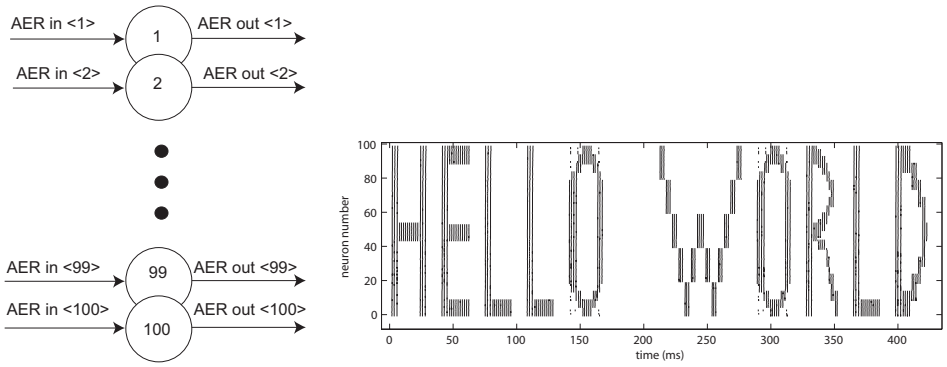
Figure 62: Closing the loop around neuromorphic systems. Clearly, our understanding of neuroscience and the theory of computation influences the designs of neuromorphic systems. Additionally, by using neuromorphic hardware to perform nontrivial neural modeling, we can gain new insights into those foundational sciences.

Section 6.2 describes a "Dot matrix" network and a synfire chain, which is important in several of the other networks studied here. Section 6.3 describes spatio-temporal pattern detection and generation, a ring winner-take-all network, and a bistable oscillator. The biological relevance of these networks is discussed, and measured data from network simulations on the neuromorphic IC are presented. Section 6.4 describes a graph search algorithm, its application to robotic path planning, and presents measured results from this network. Section 6.5 analyzes the computation performed by these networks and by the neuromorphic platform, with a comparison to alternative approaches.

6.2 Basic networks: "dot matrix" and synfire chain

6.2.1 "Dot matrix" network

A dramatic demonstration of the functionality of all of the components of the signal flow can be made by simulating a "dot matrix" network, wherein each neuron can be caused to spike by applying an AER input to it. All 100 neurons in the chip were given a stream of input events designed to result in a desired pattern when viewed in a raster plot format. Figure 63 shows the network topology and the measured result.



(a) (b)

Figure 63: Results from "dot matrix" experiment. a) Diagram of the connectivity. The synaptic weights are set sufficiently high that a single input reliably elicits a spike from the target neuron. b) Measured outputs from this circuit. Only the spiking output activity of the neurons is depicted, but the pattern of AER inputs is essentially the same.

6.2.2 100-neuron synfire chain

Pools of neurons that are connected serially into a chain, called "synfire chains", can produce sequences of spikes with consistent timings each time the chain is stimulated. This has been proposed as an explanation for some electrophysiological findings [1], and several contexts for computational significance of this type of network have been suggested [2, 4].

In order to make the most of the limited neurons available, the simplified network topology depicted in Figure 64a (wherein each pool in the chain is modeled by a single neuron) was simulated. A raster plot of the result of the simulation is also shown in Figure 64b. The plot depicts the response of two successive spikes at the input of the chain. Each spike starts a wave of activity down the chain. Note that the second wave is initiated while the first wave is still propagating. The speed of the wave propagation, which is determined by the delay in synaptic transmission and the strength of the synaptic connections, averages about 1ms per synapse.

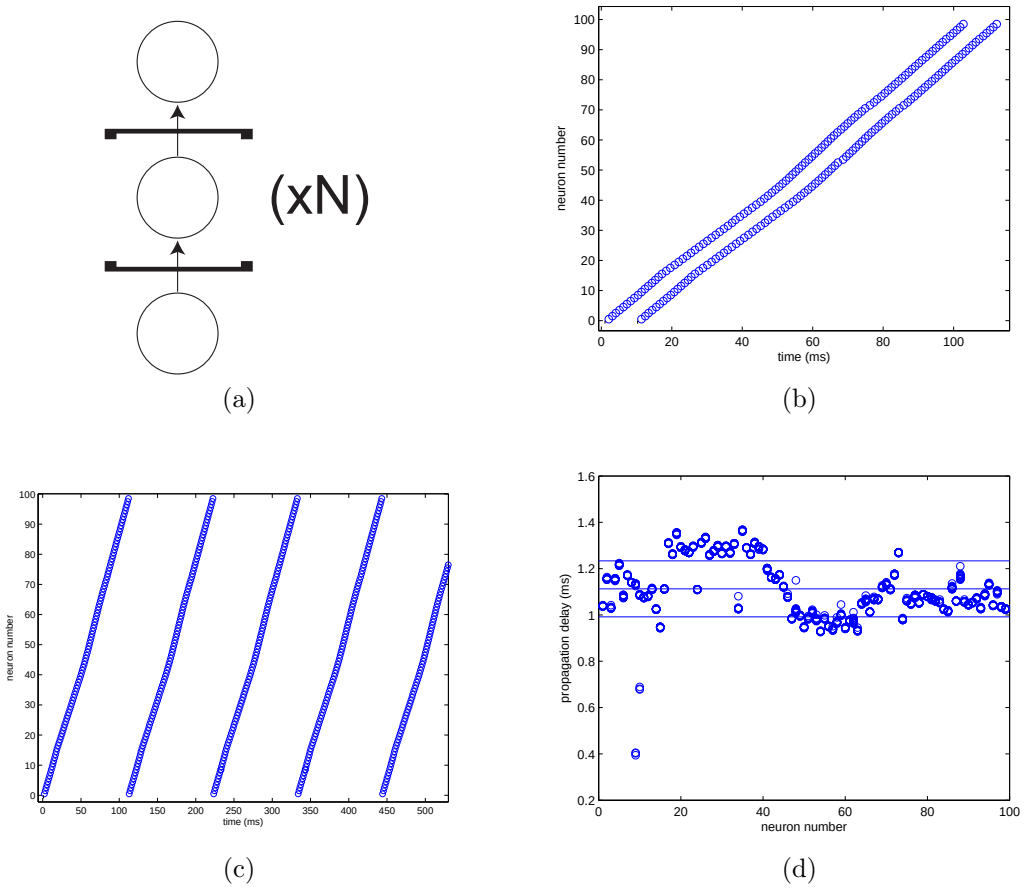


Figure 64: Simulation results for 100-neuron synfire chain. a) Depiction of synfire chain network. For the data shown here there are 100 neurons so N is 98. b) Raster plot of spiking output resulting from two sequential inputs initiated at the beginning of the chain. This results in two simultaneous waves propagating along the chain. c) Raster plot of periodic behavior in the synfire chain when the neuron at the end of the chain is connected back to the one at the beginning. d) measured propagation delays for each neuron in the chain. The mean and +/- 1σ levels are shown by horizontal lines. In b) and c), input spikes are denoted by vertical tick marks, while outputs are denoted by open circles.

This propagation delay is fairly consistent across all of the neurons in the chain. The measured propagation delays are shown in Figure 64d. This kind of uniformity would not be possible without the facility for trimming the synaptic waveform shaping circuitry by programming floating-gate transistors. Some preliminary results about this topic were shown in [12], and we have since fully integrated the approach into the system for using the IC.

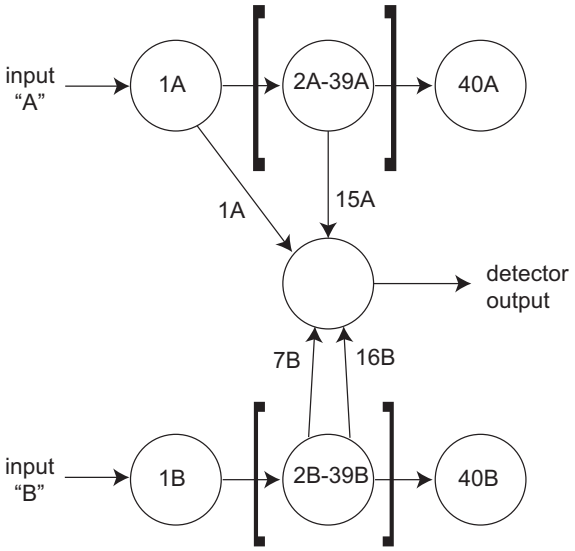
6.3 Computational networks

6.3.1 Spatiotemporal pattern generation and detection

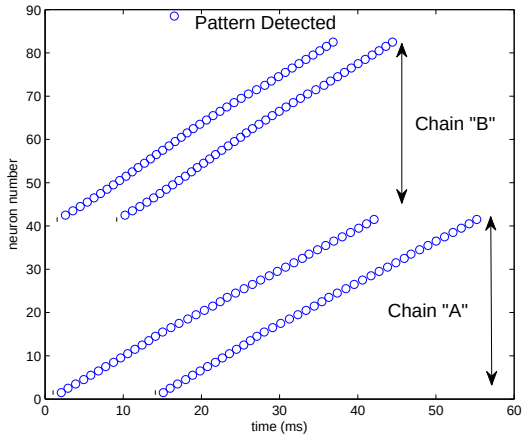
The reliable timings demonstrated in the synfire chain make it useful for creating some networks wherein spike timing is important in the information encoding scheme. For instance, arbitrary spatio-temporal patterns can be detected by a network that has a distinct synfire chain for each distinct input channel. The approach is illustrated for a sequence of 4 spikes total on 2 inputs A and B, with the sequence ABBA and intervals of 0.5 ms, 7.5 ms, and 4.5 ms separating sequential spikes.

The network that performs this detection is depicted with the raster plot of its response to the input pattern of interest in Figure 65. If any of the four spikes in this pattern are omitted or shifted by more than about 1 ms, the output neuron does not spike. Getting the synaptic weights to a value such that the coincidence of all 4 inputs is necessary and sufficient to produce a spike in the output neuron required an iterative process of programming weights and testing the response to inputs.

It is worth noting that sound localization in animals requires the detection of delays in two input lines (the inputs coming from the two ears), and that a neural structure that operates on a similar principle to the one presented here has been characterized in the barn owl [40].

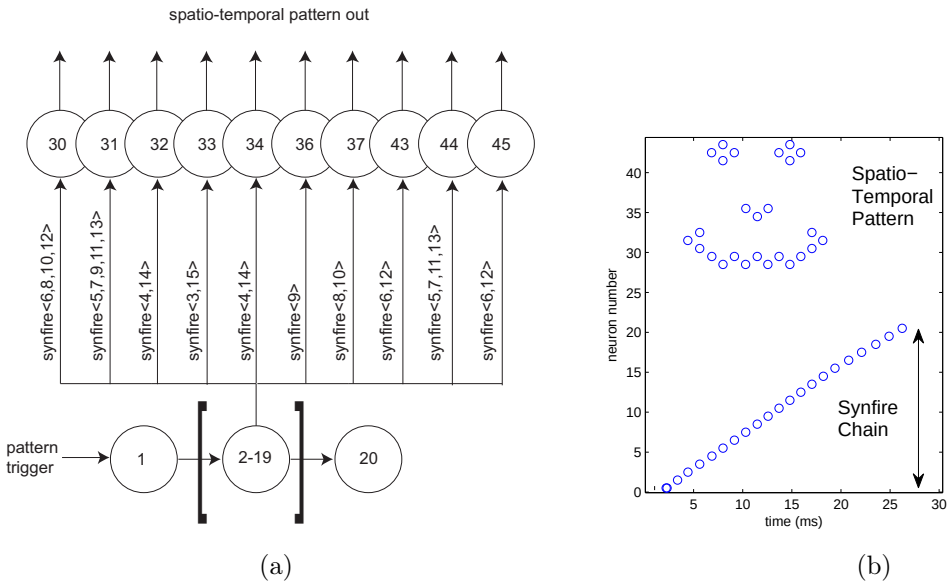


(a)



(b)

Figure 65: Simulation results for spatiotemporal pattern detector network. a) Diagram showing network connectivity. Brackets indicate repeated units connected sequentially to form a chain. b) Raster plot of activity in the network when the tuned pattern of “ABBA” is given with the appropriate timings of 0.5, 7.5, and 4.5 ms separating successive inputs. Input spikes are denoted by vertical tick marks, while outputs are denoted by open circles. The synfire chain “A” corresponds to neuron numbers 2-42, chain “B” is numbers 43-82, and the output neuron is 89.



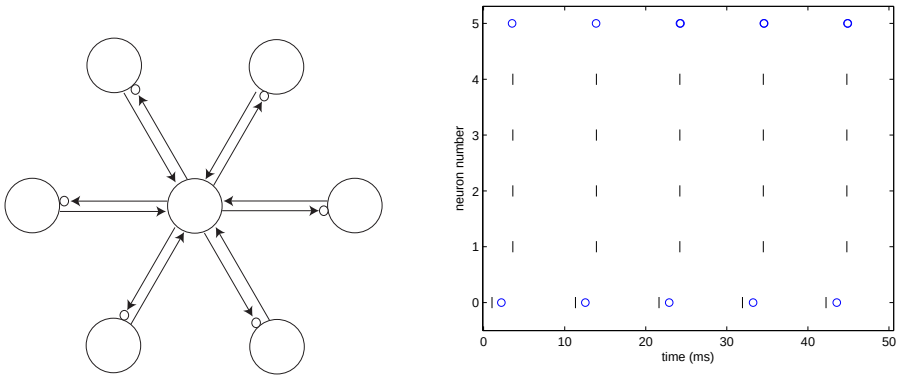
(a) (b)

Figure 66: Simulation results for spatiotemporal pattern generator network. a) Diagram showing network connectivity. Brackets indicate repeated units connected sequentially to form a chain, and a "signal bus" notation is used to denote the various connections from the synfire chain to the output neurons. b) Raster plot of activity in the network in response to a single "pattern trigger" spike. Input spikes are denoted by vertical tick marks, while outputs are denoted by open circles.

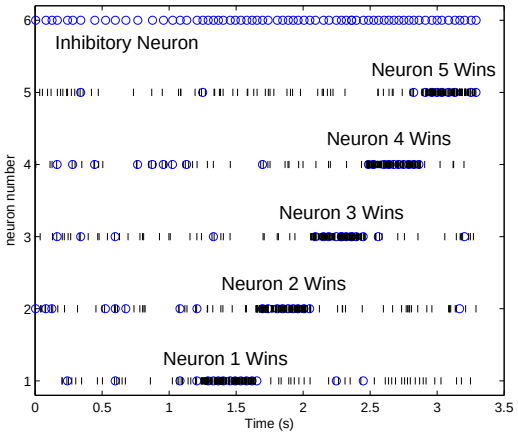
Arbitrary spatio-temporal patterns can also be generated using a similar approach. The network shown in Figure 66a generates the sequence of spikes depicted the raster plot in Figure 66b when its input is stimulated with a single spike. This kind of a functionality could be useful in generating a precisely timed stereotyped motor output from a neural system.

6.3.2 Ring Winner-Take-All (WTA) network

The WTA network models competition through lateral inhibition. It is a functional block that is useful for performing classification tasks (for example, [53]), and it is an integral part of many models of neural phenomena, especially regarding attention [48]. An example of such a network that has six input neurons is shown in Figure 67.



(a) (b)



(c)

Figure 67: Results for 6-input WTA network. a) Diagram showing network connectivity for ring winner-take-all. Arrows that terminate in small open circles are used to denote inhibitory connections. b) Raster plot of this network for the "first-to-spike" encoding scheme. In this scheme, the first neuron to spike in each burst of inputs prevents the others from spiking. c) Raster plot of 5-input WTA for the rate encoding scheme. The input rate to all neurons is 20Hz for the first 1.2s, after each of the 5 inputs sequentially receives a burst of stimulation at 320 Hz while the others maintain 20Hz stimulation. In b) and c), input spikes are denoted by vertical tick marks, while outputs are denoted by open circles.

A simulation with a 30-input winner-take all was run, and the results are depicted in Figure 68. The duration of the post-synaptic currents of the inhibitory synapses was set to 20ms in order to increase the effect of inhibition, and thus maximize the competitive dynamics. The inputs were Poisson spike trains at a fixed average spike rate of 1Hz, except for one input channel (the "winner"), whose input rates were varied from 1Hz to 240Hz.

In Figure 68b, the resulting firing rates of all the neurons (averaged over 100 seconds of measurement) are shown for each input rate on the "winner" channel. The average firing rate of the "winner" neuron strongly increases when its input rate is increased, while the average firing rates of the others are suppressed. This illustrates the competitive behavior that is the essential feature of this network. Figure 68a depicts a sample raster plot of network activity for a case in which all neurons receive inputs at an average rate of 1Hz for 10 seconds, and then a single input rate goes to 240 Hz for 10 seconds. The increase in activity of the "winner" neuron and the decrease of the others is apparent. The symmetry of the structure was verified, i.e. all of the features pointed out here do not depend upon which neuron is selected to receive the faster input stimulation.

It is interesting to note that since the dynamics of the neurons are different from the simple integrate-and-fire models that are often used in spiking network simulations (in particular, temporal integration does not play an important role over time scales much greater than the duration of an action potential), using rate coding for a network like this is not particularly natural. This is reflected by the fact that in this example the duration of the inhibitory PSC needs to be extended in order to observe significant competitive effects, and also by the rather gradual suppression of the "non-winning" inputs as in firing rate of the "winner" is increased.

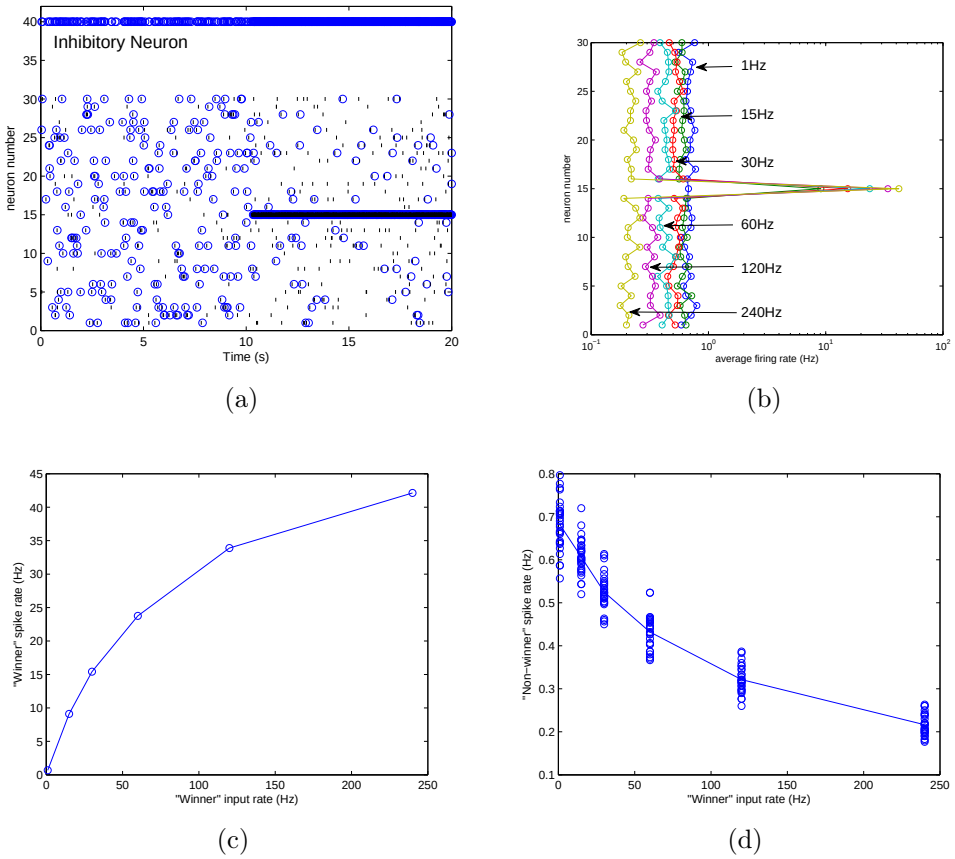


Figure 68: Simulation results for 30-input ring winner-take-all (WTA) network. a) Raster plot depicting a typical example of activity in the network when the input rate for neuron 15 is 1Hz for 10s, and is then sustained at 240 Hz for 10s. The inputs on the other neurons remain at 1Hz for the duration of the experiment. Input spikes are denoted by vertical tick marks, while outputs are denoted by open circles. b) Plot of average firing rates (over 100 seconds of measured data) of each of the 30 neurons, shown for 6 different input rates on neuron 15 (again, with 1Hz input rate on all other neurons). Note that the vertical axis is set up to approximately align with the axis in a), and that a log scale is used for the horizontal axis in order to simultaneously show the changes in rates of the "winner" neuron and the others. c) Firing rate of the neuron 15 versus the input rate on neuron 15. d) Firing rates of the neurons other than 15 versus the input rate on neuron 15. The population average is plotted with a line.

An encoding scheme that is more compatible with this system would be one in which the inputs arrive in burst that could, for example, correspond to a particular phase of a global oscillation in a neural system. The "winner" in each burst is the neuron whose inputs arrive first. This type of encoding scheme has been proposed as a mechanism by which real biological networks process data efficiently and with low latency [64]. An illustration of a simulation of such a regime is shown for a 6-input WTA in Figure 67b. In this experiment, the inhibitory PSC is only 1-2ms, and the suppression of the later-arriving inputs is complete.

The "first to arrive" encoding actually has more in common with the rate encoded system than one might expect. In the rate encoded system in Figures 67b and 68, the first neuron to receive an input after the end of a given inhibitory PSC usually elicits a spike from the inhibitory interneuron, thus preventing the other neurons in the network from firing for a period of time. If the inputs to the neurons are Poisson spike trains, the higher the average frequency on a given neuron's inputs, the more likely it is to be the first to spike. As the frequency of spikes on the input of the "winner" neuron increases, the "winner" ends up getting this first spike more often, so its spike frequency increases, and the interval between successive inhibitory spikes decreases, which suppresses firing of the other neurons. This is the mechanism of competition in the rate-encoded version of the ring WTA.

The mechanics of this network are nicely illustrated by examining a histogram of the time since the preceding inhibitory action potential for each excitatory action potential that is recorded. This is shown for the 30-input WTA data in Figure 69 (this plot is constructed from the same data depicted in Figure 68b. Note that regardless of input rates, there are very few excitatory spikes that occur during the inhibitory PSC, which spans a period of about 20ms following the inhibitory spikes. The distribution

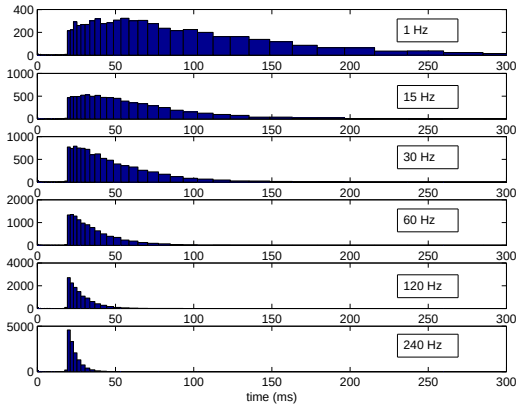


Figure 69: Histogram of time since preceding inhibitory input for each excitatory action potential for 30-input WTA. The average input rate on the "winner" neuron is annotated on each strip. The effect of the "winner" neuron's activity to reduce the time between consecutive inhibitory spikes is apparent. Also, the suppression of action potentials during the 20ms after the inhibitory action potential is clearly visible.

of spikes after the end of the PSC is broad if the input rate of all of the neurons is low, but becomes more sharply peaked as the input rate to the "winner" neuron increases.

6.3.3 Bistable oscillator

If the final neuron in the synfire chain depicted in Figure 64a is connected back to the input of the chain, the propagating wave of action potentials will be restarted each time it reaches the end of the chain, forming a stable oscillator. Given the refractory period and synaptic delays in the system, this topology was found to produce a stable propagating wave of action potentials using any synfire chain that is at least 4 neurons long. If the circuit also has a global inhibitory input, as depicted in Figure 70a, then the oscillations may be turned on or off by a single excitatory or inhibitory input. This circuit is somewhat reminiscent of a central pattern generator (CPG), with the global inhibition perhaps playing the role of neuromodulators that promote or suppress rhythmic activity.

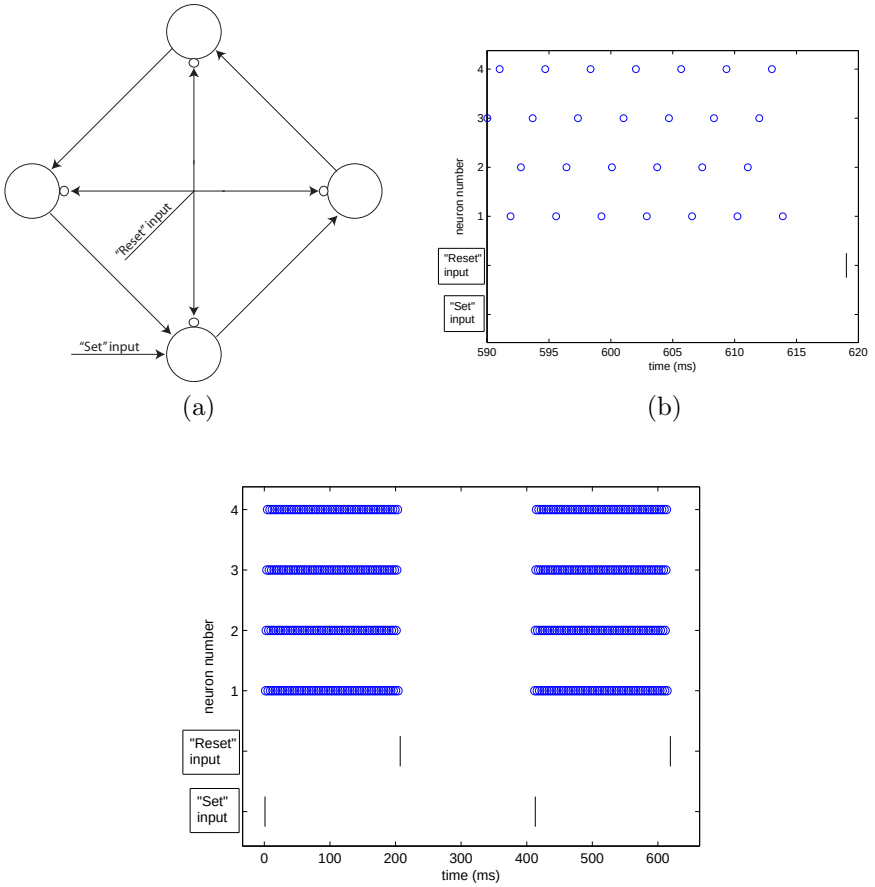


Figure 70: Simulation results for bistable oscillator network. a) Schematic depiction of network connectivity. b) Zoomed in depiction of the end of the data shown in c), to illustrate the timing of the cyclically propagating wave of activity. c) Raster plot showing activity of the network as the inputs drive it back and forth between the stable oscillations and the stable fixed point. In b) and c), input spikes are denoted by vertical tick marks, while outputs are denoted by open circles.

This network has two stable dynamical states: the propagating wave, and the quiescent state of no spiking (for a longer chain, multiple waves can propagate stably at different phases). This bistability that can be controlled by two different inputs is directly analogous to the digital latch circuit. Like the latch, this circuit's dynamical state stably stores a bit of information about the past inputs of the system. This system is an example of recurrent excitation giving rise to multiple stable attractors in a network's dynamics, which is the essence of the well-known Hopfield model for associative memory [22]. In fact, a Hopfield network with the same connectivity captures the behavior demonstrated by this circuit.

Spiking activity recorded from this circuit is depicted in Figure 70.

6.4 Path planning

A configurable array of spiking neurons turns out to be a useful tool for solving a class of problems in robotics known as path planning. One example of the problem, depicted in Figure 71, involves an autonomous robot that is attempting to traverse a maze. Or equivalently (and somewhat more realistically), the robot is attempting to cross terrain that is filled with obstacles. This problem can be represented as a graph by dividing the space into discrete regions, assigning a vertex in the graph for each region, and using a directed edge between any pair of neighboring vertices that are not separated by a barrier. This type of representation is depicted in Figure 71, which also shows how this graph can be translated into a network of neurons. Each vertex in the graph is represented by a neuron, and directed edges between vertices are represented by synapses. The synaptic weights of all synapses are made sufficiently high that a presynaptic spike reliably triggers a post-synaptic spike. Thus, a spike initiated at any location in the network propagates outward at a rate that is determined by the synaptic delays.

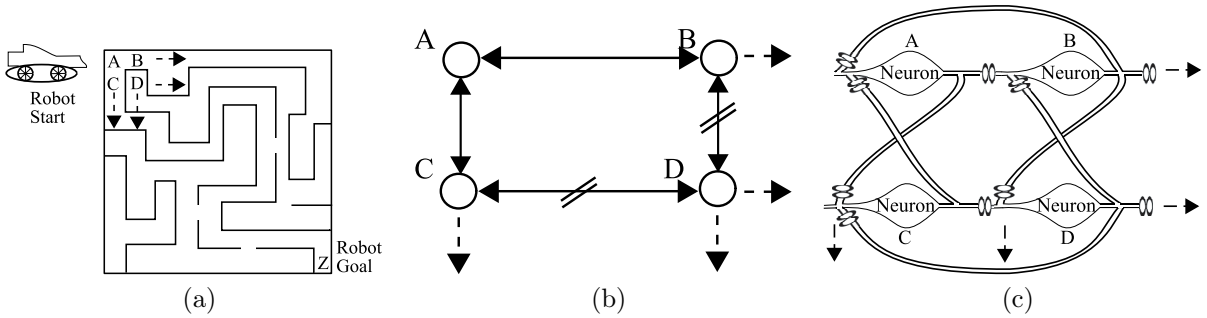


Figure 71: Context for robotic path planning problem. a) Picture of a maze (or obstacle-filled landscape) that the robot must navigate. The robot starts at location A, and is attempting to get to location C. b) Graphical representation of the immediate neighborhood of the starting point (A) shown in a). Hash marks through the edges connecting B to D and C to D indicate that these transitions are not allowed (the graph will not contain these edges). c) Depiction of a set of neurons with reciprocal connections to represent traversability.

If a spike is initiated at the start of the maze (or the robot's initial location in space), the shortest path to the goal can be found by observing the sequence of spikes that eventually causes the first spike at the goal. If the neuron representing the goal location never spikes, the robot can correctly conclude that there is no viable path that will take it to the goal. The concept of path planning by emulating a propagating wave in this manner is referred to as "wavefront planning", an early example of which is discussed in [21]. The idea that a network of neurons is capable of implementing a wavefront planner has also been suggested previously [54]. However, this study is the first demonstration of this method using neuromorphic hardware.

The following example illustrates the approach that was taken for this study in more detail. The 100 neurons were used to represent a 10x10 grid in two dimensional space. Obstacles were randomly generated in the grid, and the starting point and goal were also chosen randomly. An example map, and its representation by neural connections, is shown in Figure 72. A single input spike was provided to the neuron representing the starting point via the AER receiver. The resulting activity in the

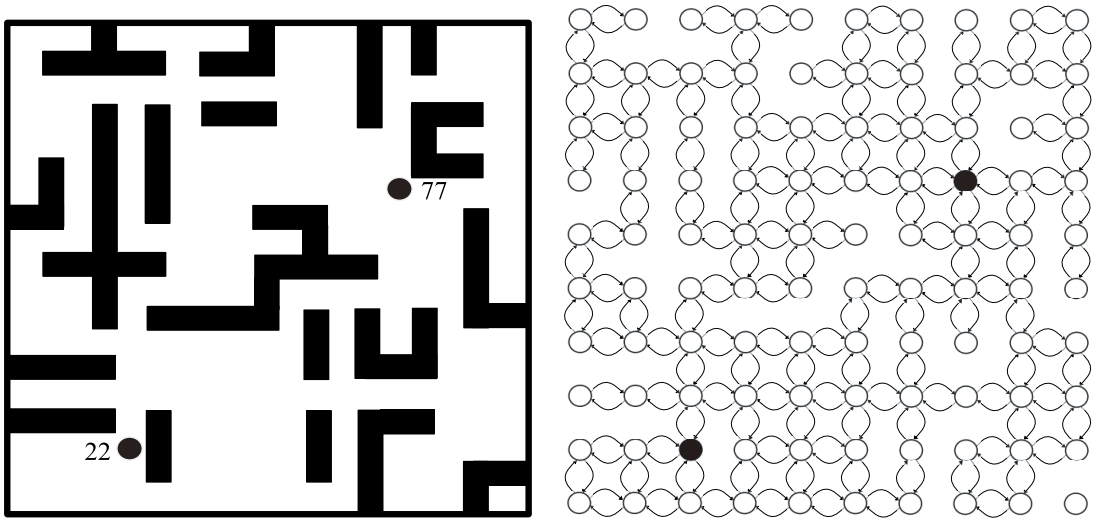


Figure 72: Illustration of example problem, a) in terms of the geometries of the obstacles, and b) in terms of the synaptic connections among the neurons in the array.

network was recorded by the AER setup described in Chapter 5. The sequence of spikes that led to the first spike in the goal neuron was extracted from the recorded AER data. This was done by a simple process: the first spike in the goal neuron was identified, followed by the first spike among its neighbors. Then the first spike among the neighbor of that neuron was identified, and so on, until the neuron at the starting location was reached. The path that was found using this method is shown in Figure 73, along with a reduced raster plot and an analysis of the timing that separated pre and post synaptic spikes along the path. This analysis shows that the synaptic delays were very well matched.

An obvious question that arises is how much computation is saved by performing the path planning with the aid of this network of neurons. A detailed analysis is currently being performed by Scott Koziol, a colleague who is an expert in robotic path planning. Intuition would suggest that replacing the search in path space (which

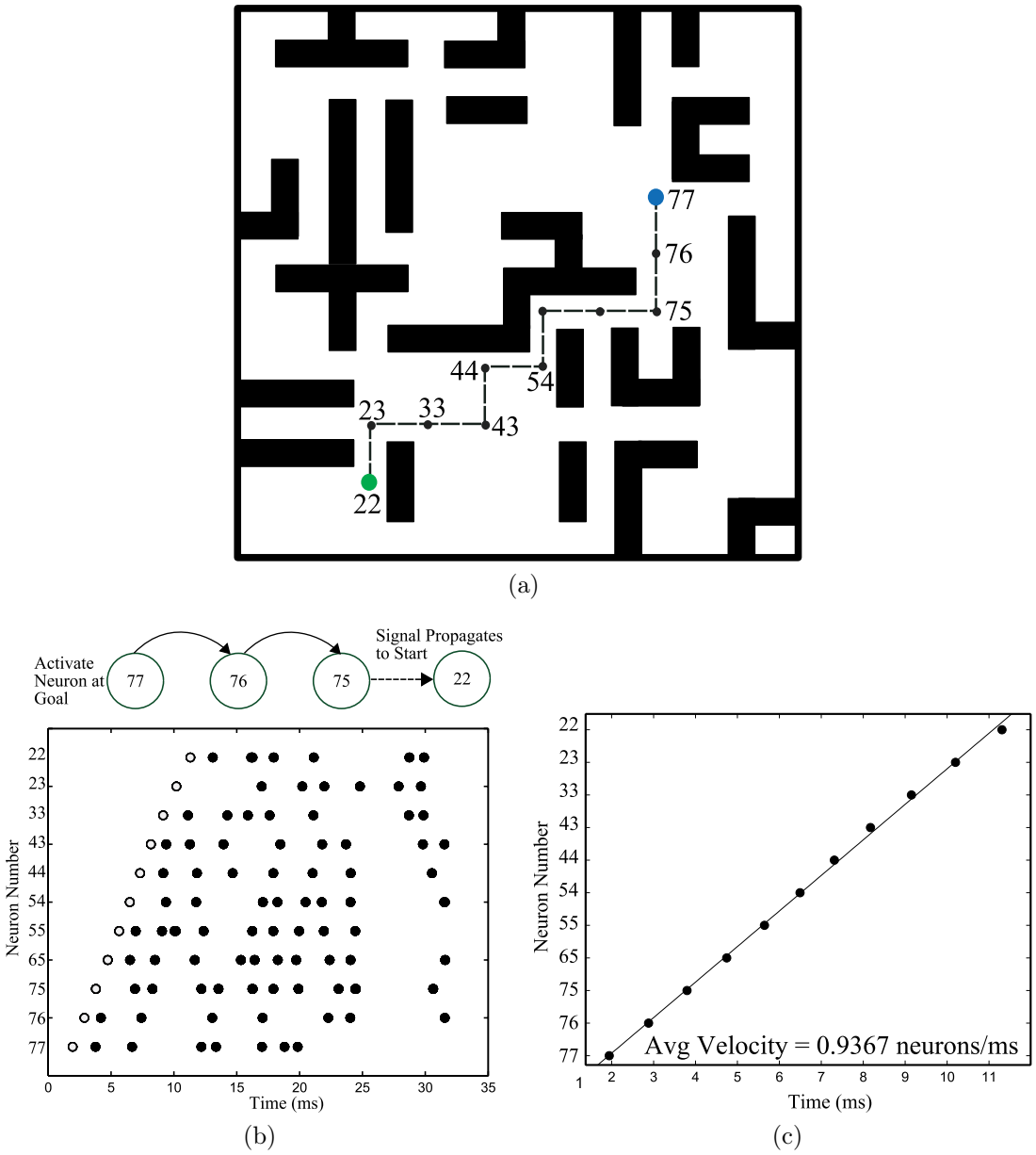


Figure 73: Illustration of solution to example path planning problem. a) Depiction of the optimal path in the spatial representation. b) Raster plot of the activity in the neurons that make up the path, ordered vertically according to their locations along the path. This way of plotting the activity highlights the wave of activity that ultimately results in the first spike at the goal neurons. c) Analysis of the synaptic delays along the path. The delays are found to be fairly uniform at a little less than 1 ms per synapse.

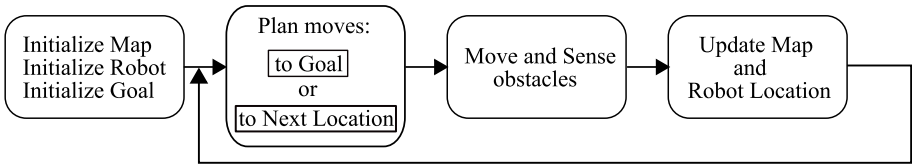


Figure 74: Flow chart depicting the two alternative approaches to path planning with the neuromorphic IC. The AER infrastructure can be used together with a sequence of simple searches in the recorded spike data to plan the entire path to the goal. Then, the movement to the goal can be executed, and the process can repeat if there is another goal location. Alternatively, the spike data need not be recorded, and thus no post-processing of the spike data is required. This approach only provides a plan for the next move, so the robot executes this move and then re-plans.

scales exponentially with the map size) with a sequence of highly constrained searches among neighbors (a sequence that is only as long as the minimum path length) will provide a large savings in computational effort, especially as the size of the graph grows. Of course, performing this computation with neurons for a larger graph would require hardware with more model neurons. This is a reasonable thing to plan for, since the development of larger arrays of neurons is already a major focus of the roadmap for this line of research, as discussed further in Chapter 8.

For robotic systems with limited computational resources (for instance, systems without the capability of recording the stream of spike data from the AER output of the IC), this approach is still viable, but with a small modification. If instead of providing the stimulus spike at the starting location in the graph, it is provided at the goal location, then the system need only wait for the first spike among the neighbors of the robot's current location. This spike will give the optimal choice of the next move that the robot should make. This experiment could be performed repeatedly as the robot traverses the region, once at each decision point. A comparison of these two alternative schemes is depicted in Figure 74, which shows the original scheme for comparison.

In order to make sure that the intuition behind this algorithm is sound, a proof of the completeness and optimality of the method was written (casting the algorithm in graph theoretic terms). The proof, which is based on the approximation that all of the synaptic delays are uniform, shows that this approach will identify the shortest path between any two vertices of any directed graph (and will conclude that no path exists if and only if there is actually no such path). It is interesting to note that applications of path planning need not be restricted to 2-dimensional state spaces or to uniform discretizations of the state space, two simplifying features of the example that was presented here. In fact, any problem that can be mapped into a set of discrete states, with a rule that describes what state transitions are allowed and what transitions are forbidden can be solved equally well using this technique.

One further generalization is possible in a straightforward way. The synaptic delays can be programmed to variable values in order to represent a continuously variable cost for making each state transition. In order to avoid complicating the inference of the path of wave propagation, all of the inputs to any given neuron must have the same synaptic delay. A quick analysis concludes that this constraint can be satisfied while still allowing a continuously variable cost at each grid location by using multiple neurons to represent a single grid location (one neuron per distinct cost among the grid location's neighbors). I conjecture that a network constructed in this manner, using the same method that was described for finding the shortest path, would find the path of lowest total cost.

This generalization could have a nice application for the problem of maze navigation. It is a known flaw with wavefront planners that, in problems with many optimal paths, they tend to choose paths that move along obstacles. Moreover, if a path can be shortened very slightly by traveling right next to an obstacle, the wavefront planner will indeed choose this shorter path. This behavior is undesirable in some robotic applications because it increases the risk of collisions with obstacles. A

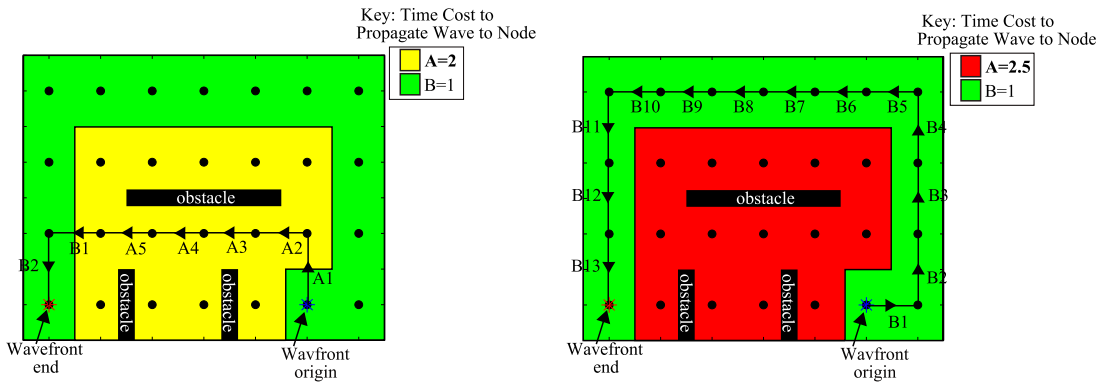


Figure 75: Use of variable synaptic delays to represent a cost function associated with different locations. In this maze, every location that borders an obstacle is assigned an increased cost (representing the desire to leave margin around obstacles). In a), the additional cost is low, and the planner charts a course that passes near all of the obstacles, but minimizes the path length. In b), the additional cost is increased sufficiently that the planner chooses the longer path that cuts a wide swath around the obstacles.

desire to cut a wider swath around obstacles could be represented in an elegant way by increasing the cost of edges in the graph that represent motion near obstacles. The extent of this increase could be tuned to effect a choice in the tradeoff between path length and margin for obstacle avoidance. In order to illustrate this concept, this approach was applied to the problem of navigating the maze shown in Figure 75. In this maze, every location that borders an obstacle is assigned an increased cost. When the additional cost is low, the planner charts a course that passes near all of the obstacles, but minimizes the path length. If the additional cost is increased, a threshold is reached at which the planner chooses the longer path that cuts a wide swath around the obstacles.

In order to gain some confidence in the robustness of the implementation of path planning on this neuromorphic IC, the system was tested for a set of 55 randomly generated (unweighted) maps. The result was completely successful. The algorithm

found the shortest path for each map that had a path, and it also correctly identified each map that did not have a path.

6.5 Conclusion

Having demonstrated modeling of multiple neural structures on this neuromorphic IC, we turn our attention to the topic of what computation the networks are performing, and how efficiently they are doing it. It is difficult to compare the computations done by a network of neurons to those performed by a traditional computing structure such as a microprocessor. The network of neurons is unable to find all of the eigenvalues or the inverse of a matrix, while this task is straightforward for the microprocessor. However, this does not mean that the network of neurons is not computing. In fact, the neural network can effectively solve problems such as generating intricately timed sequences of control commands, detecting specific patterns in input data, latching and storing input data, implementing competitive comparisons among many input channels simultaneously, and parallel graph search for path planning. These are all useful functional blocks in sensory processing and motor control, and thus could form important building blocks for sophisticated autonomous robotic systems.

Given that the networks studied in this work can perform useful computations, it is interesting to ask what alternative approaches might yield the same computations. The synfire chain could be implemented simply using an oscillator, a binary counter, and a decoder. The pattern generation and pattern detection could then be implemented by adding some combinational logic gates to the synfire chain outputs. The volatile memory of the bistable oscillator could be performed simply with a latch. Making a similar statement for the WTA requires specifying exactly what aspects of the WTA behavior must be achieved. In the case that the WTA output is considered to be a "one-hot" encoding of the maximum of its inputs (replacing rate coding with static inputs representing rates), the WTA could be implemented using a tree

of comparators. If one requires outputs with finer granularity than 0's and 1's, then matrix algebra would probably be required for any digital implementation. On the other hand, an analog circuit such as the one described in [43] is able to perform this function with few transistors and low power.

Of the computational tasks demonstrated on the neuromorphic IC, the path planning is the most difficult problem to implement on a digital computer. For the simpler case of an unweighted graph, the problem is often solved in practice using the A* ("A-star") search algorithm [31]. This algorithm takes a significant amount of computing resources, and in general the time complexity (roughly, the number of processor operations) scales exponentially with the length of the optimal path. If certain conditions are met, the time complexity scales as a polynomial of the optimal path length. In comparison, the run time of the neuromorphic algorithm scales linearly with the length of the optimal path, and the number of spikes required (thus the energy used) scales as a polynomial with order between 2 and 3.

If one is interested not only in the most expedient solution to a computational problem, but instead has a specific desire to see how the problem might be solved by a neural structure, then the computational solutions described in the previous paragraphs are not acceptable. In this case, perhaps the most common approach is to use a digital computer to numerically integrate model equations describing a network and its constituent neurons and synapses. From a computational efficiency standpoint, it is instructive to make a comparison between the power consumption of running such a model simulation on a neuromorphic IC versus using numerical integration.

The energy usage of the various blocks in the neuron IC was described in detail in Section 5.6. Using the numbers described there, all of the networks presented in this chapter were analyzed in terms of power consumption, comparing the neuromorphic approach to the numerical simulation approach. The results are shown in 6.

Table 6: Estimated energy to perform the network simulations presented in this work on this neuromorphic platform versus a numerical integration approach.

Model Description	Neuron Energy (nJ)	Synapse Energy (nJ)	AER Energy (uJ)	Neuromorph. Energy (uJ)	Num. Soln. Energy (mJ)
Synfire chain	508	10	20	20	4.8
Pattern detect	221	0.4	10	10	2.0
Pattern gen.	41	2	4.5	4.6	0.35
WTA	1343	15	178	179	16
Bistable osc.	77	11	36	36	0.37
Path plan	85	27	2.5	2.4	0.57

Clearly a significant advantage in power efficiency is obtained, with the improvement factor ranging from about 10x for the bistable oscillator to about 240x for the synfire chain. While these numbers are not as impressive as the 6000x improvement for the case discussed in Chapter 5, one important fact is immediately apparent from examination of Table 5. The AER power completely dominates the power consumption for these network simulations. If the AER power is neglected, the neuromorphic approach becomes more efficient than the numerical integration approach by factors ranging from 4,200 to 11,700. This is a relevant consideration because as the size of the network simulation scales, the AER interface should not scale proportionally, and eventually the power dissipation will be dominated by the power required by the neuron and synapse models, as discussed in Chapter 5.

It seems relevant to reiterate here a fact that was mentioned in Chapter 1. Fundamental limitations about power consumption notwithstanding, a research group with limited access to supercomputing resources can model relatively large (up to a hundred neurons and thousands of synapses) networks on this neuromorphic system in real time, whereas the same network simulations would be very time consuming on a personal computer.

In summary, several interesting and useful computations were demonstrated on a model neural structure simulated on neural hardware. These results are obtained with a computational efficiency that is orders of magnitude better than that offered by

the approach of numerically integrating model equations. However, the computations performed by these neural models is, at best, only marginally competitive with the best custom (non-neural) solutions for performing the same computations (with the possible exception of the path planning computation). To the my knowledge, the same could be said of any applied computation that has been demonstrated on any other neuromorphic platforms that are intended for network simulations. In fact, work demonstrating computation in networks of neurons on a neuromorphic simulation engine are very rare. One other notable example of such work was performed by the Heidelberg group [13]. That study provided significant inspiration for the present work, and the fact that both of these systems use PyNN should make it easier for results to be compared across these two platforms.

The scarcity of implementations of neural computation in neuromorphic hardware should not be seen as an indication that models of neural systems on configurable neuromorphic platforms cannot outperform other approaches. To the contrary, I believe that such systems have enormous potential to compete in applications, a potential that will increasingly be realized as neuromorphic platforms continue to improve. The current status of the field merely reflects the fact that the mainstream computational approaches have benefited from significantly more time and money invested in development. Because the neuromorphic approach is much less constrained by fundamental limitations, further development in this field has enormous potential to outpace the traditional approach.

Chapter VII

LEARNING IN SPIKING NETWORKS

In 1945 Donald Hebb published a book entitled "The Organization of Behavior", which was very influential in the field of neural computation [34]. In the book, Hebb hypothesized that when a pre-synaptic neuron repeatedly takes part in causing a post-synaptic neuron to spike, the strength of the synapse is increased. This process, which came to be known as Hebbian learning, receive sufficient support from electrophysiological data that it came to be accepted as an important part of how animals learn behaviors. In the 1980s and '90s, electrophysiological studies were carried out to characterize how a form of Hebbian learning in which changes in synaptic strengths result from repeated pairings of pre and post-synaptic spikes at a synapse. In 1998, Guoqiang Bi and Mu-Ming Poo used cultured hippocampal neurons to fully characterize the dependence of the weight change upon the timing of these pre and post-synaptic spikes [10]. The showed that a pre-post pairing results in strengthening of the synapse and a post-pre pairing results in weakening and that in both cases, the extent of the change to the synapse decreases strongly with increasing time, becoming negligible within tens of milliseconds. The phenomenon is referred to as spike timing dependent plasticity (STDP). It has been observed in many electrophysiological experiments in different preparations, and plays an important role in many models for learning [15].

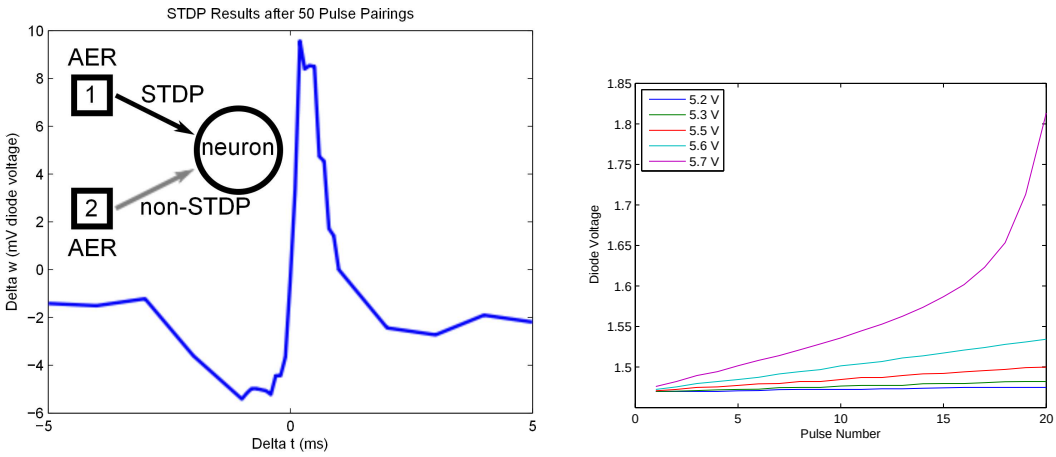
7.1 Two-synapse model

The simplest network of neurons that can be simulated with STDP is the associative learning model shown in Figure 76. In this model, two different synapses connect to the same neuron, and the activity on the two synapses is correlated. Specifically,

the time separating activation of the two synapses is the same. Moreover, one of the synapses is sufficiently strong that it reliably produces a post-synaptic spike, and STDP at this synapse is not enabled. This setup is the simplest way to shown the learning properties of a single synapse within a network context. Figure 76 shows the change in synaptic current that results from 50 pulse pairings as a function of the time difference between the onset of the STDP-enabled synapse's activation and the post-synaptic spike. It is worth noting that in this figure (as well as several others in this chapter), synaptic currents are measured by the voltage produced by forcing the current through a series of two diode-connected nFETs. The subthreshold model $\frac{I}{I_0} = exp(\frac{0.7}{2}\frac{\triangle V}{0.025})$ can be used to convert changes in diode voltage into relative changes in synaptic current. For instance, a 10-mV change in diode voltage corresponds to a change in synapse current of about 15%. Figure 76 also shows how the strength of potentiation increases as the voltage used for injection is increased. This dependence, which is expected on the basis of the characteristics of pFET channel hot electron injection, can be used to speed up or slow down the rate of learning (i.e., to scale the amplitude of the characteristic shown in Figure 76a).

7.2 Multi-synapse interactions

Several interesting features of synaptic plasticity can be observed by testing various patterns of inputs among a set of synapses that all target the same neuron, as shown in Figure 77. This is a simple framework for studying how the dynamics of plasticity at multiple synapses are coupled together through the post-synaptic neuron dynamics, and how input patterns can drive learning that changes the response properties of the post-synaptic neuron.



(a) (b)

Figure 76: Two-synapse model for investigating the effect of STDP. The setup of the experiment and the resulting change in synaptic weights are shown in a). The horizontal axis in a) is equivalent to post-synaptic minus pre-synaptic spike time. In b), the evolution of the synaptic weights with repeated potentiations is shown for a variety of different injection voltages. In both plots, the synaptic strength is measured by the voltage produced by forcing the current through a series of two diode-connected nFETs. The subthreshold model $\frac{I}{I_0} = exp(\frac{0.7}{2}\frac{\triangle V}{0.025})$ can be used to convert changes in diode voltage into relative changes in synaptic current.

7.2.1 Methods: weight initialization and rate-coded input pattern generation

In all of the experiments presented in this section, the synaptic weights of the neurons are initialized at a level that is sufficiently weak that a single synaptic input is unable to produce a post-synaptic spike, but strong enough that two or more synaptic inputs arriving simultaneously will produce a post-synaptic spike. Because the synaptic weight depends on both the floating-gate charge stored on the synapse transistor and the amplitude of the gate waveform signal that drives the synapse, the highest floating-gate charge that will still reliably produce a post-synaptic spike varies from synapse to synapse (since there is still some small mismatch in gate waveform amplitudes). A software routine was developed to experimentally determine this threshold value of floating-gate charge for each synapse, and then in the network experiments that follow, each synapse was initially programmed with a weight that was reduced from

that threshold by about 10-15%.

Because biological neurons are often observed to spike at roughly constant average frequencies in response to stimuli, but the patterns are not periodic, it is common practice to model neural spiking patterns as Poisson processes. For the studies in this section that involve rate coding, this tradition was observed. One interesting feature of the Poisson process is that the distribution of the timings that separate subsequent events (the inter-spike intervals in this case) exponentially decreases with time, so the maximum of this distribution occurs at 0 time difference. This distribution results in a tendency for spikes to "cluster" in time. As discussed in Chapter 5, inputs that arrive within short times of each other cause a "paired pulse facilitation" effect. While this effect could have some interesting consequences for the network's behavior, it is desirable to have a baseline case in which this paired-pulse facilitation effect does not affect the outcome. For that reason, for the experiments described in this section that require rate-coded inputs, the timing of these inputs was generated as follows. An exponentially distributed sequence of inter-spike intervals was randomly generated. Then all of the inter-spike intervals that were shorter than the duration of the gate waveform (3-5ms) were removed, and the resulting sequence of interspike intervals was then used to define the spike times.

7.2.2 Potentiation of synchronous inputs

One important intuitive element of Hebbian learning is the fact that multiple synapses onto a single neuron should cause that neuron to fire if they synchronize, and that this process should result in a strengthening of these synapses. This phenomenon is illustrated in dramatic fashion by the experiment displayed in Figure 77. In this experiment there are three excitatory synapses onto a single neuron. Initially, none of the synapses are strong enough to single-handedly cause a post-synaptic spike. However, when all three of the synapses are activated simultaneously, they are able

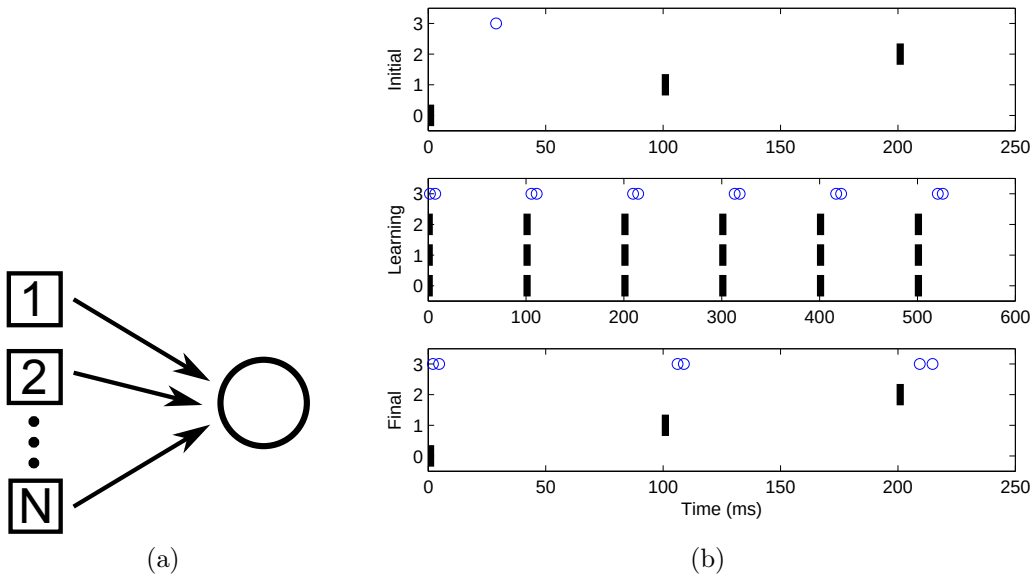


Figure 77: Potentiation of synchronous inputs. a) shows the network under consideration, in which multiple synapses all drive a single neuron. b) Raster plots from the three phases of the experiment. In the top pane, the response of the neuron in the initial state is shown, in which each input is activated individually and they all fail to elicit a post-synaptic spike. In the center pane, the training process is depicted, in which all three synapses are repeatedly activated simultaneously, each time eliciting a spike in the target neuron. In the bottom pane, the result of strengthening the synapses is demonstrated by presenting the same stimulus that was used in the initial state, but this time each individual synapse is able to elicit a post-synaptic spike.

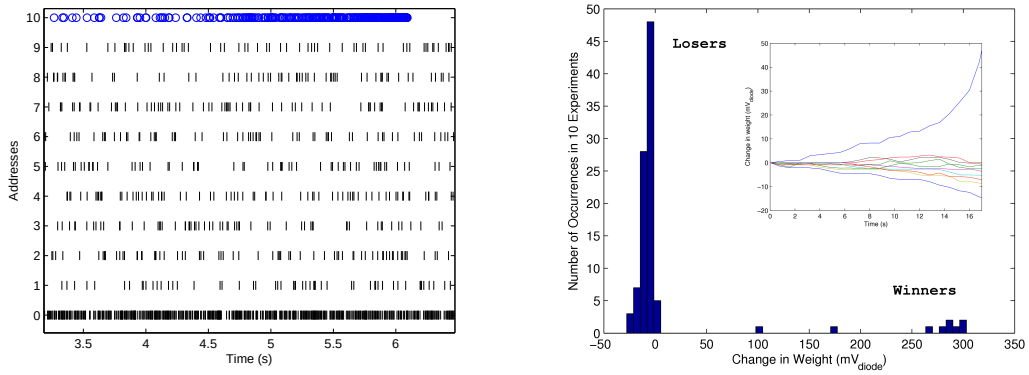
to cause a spike. When this synchronous pattern is presented repeatedly, the synaptic weights get strengthened to the extent that any single one of the synapses is capable of stimulating a spike in the post-synaptic neuron.

7.2.3 Competition among rate encoded inputs

Another important characteristic of Hebbian learning is that it provides a mechanism of competition among the synapses, wherein the input synapse that has the pattern of activity that is most likely to produce spikes in the post-synaptic neuron gets strengthened more than the other synapses onto the same neuron. A simple demonstration of this process can be made by using the network in Figure 78 with 10 inputs,

driven by a set of rate-coded Poisson inputs, one of which has a higher frequency than the others. This input is more likely than the other inputs to contribute to the firing of the target neuron, which results in preferential potentiation of this synapse. As the synapse gets strengthened, the firing rate of the target neuron increases. There is a sharp increase in the firing rate of the target neuron when the "winning" input becomes sufficiently strong to cause a spike without any contribution from the other synapses. Eventually, the "winning" synapse gets potentiated so strongly that it clamps the membrane potential of the neuron to the excitatory synaptic reversal potential E_{exc}. From this point on, the neuron is unable to spike.

In order to verify the symmetry of this effect with respect to the various inputs, the experiment was run 10 different times, each time with a different input receiving an elevated input rate. The results, depicted in the form of a histogram of the final weights (pooled across the 10 trials) in Figure, show that in general this phenomenon does hold true. In most of the cases, the inputs that received elevated input rates increased their synaptic weights while the weights of the others were moderately decreased. However, in a few of the trials, one of the synapses with the lower input rate actually ended up getting strongly potentiated while the others where depressed. The cause of this unexpected result was that because of mismatch and the random chance in the input timings, that channel got potentiated early in the course of the experiment. Because of the positive feedback inherent in this kind of competitive learning, the initial strengthening of the synapse led to further strengthening until ultimately the synapse dominated the spiking of the target neuron in spite of its lower input rate.



(a) (b)

Figure 78: Competitive learning among 10 synapses on the same neuron. Nine of the synapses receive rate-coded inputs at a low rate, while one receives a high rate of inputs. The inputs and resulting spiking behavior are shown in a). As the input with the elevated firing rate gets potentiated, the firing rate of the target neuron increases. This synapse attains a strength that is sufficient to always cause a post-synaptic spike, which results in steady potentiation. Eventually, this leads to the synapse becoming so strong that it clamps the membrane of the postsynaptic neuron to E_{exc} (the excitatory reversal potential), preventing any further spikes. b) Histogram of changes in synaptic weights pooled across 10 experiments of the type shown in a). The distribution is clearly separated into neurons that experienced a positive feedback process of potentiation and neurons that experienced a moderate depression. The inset shows the evolution of the synaptic weights during the experiment.

7.2.4 Learning from correlations in the presence of noise

Having established that synchronization of the inputs can result in strengthening of the inputs in the tightly controlled experiment described in Section 7.2.2, an obvious follow-up question is how robust this process is to random perturbations In order to address this question, the same experiment was run, only with 10 inputs. The timing of the inputs was determined as follows. All inputs had an average rate of 20 Hz. Seven of them were generated according to a Poisson process as described in Section 7.2.1. The other three were generated as the superposition of two processes: 1) a 15-Hz Poisson process, and 2) a 5-Hz input that was synchronized across all three channels. The interesting result of the simulation, depicted in Figure 79, was that even in the presence of noise in the form of uncorrelated spiking and random inputs on seven other channels, the three correlated inputs were potentiated while all of the other inputs were depressed. This result illustrates the power of this simple learning rule to pick out the structure that is present in a high-dimensional and seemingly unstructured data set (seemingly unstructured in the sense that any person examining the input spike patterns would probably overlook the correlations among the three input channels and conclude that all of the inputs are random).

7.2.5 Receptive field formation

One of the interesting phenomena that is exhibited by neural systems is a self organization into receptive fields based upon the statistics of the input data. There have been numerous modeling studies that have helped elucidate the role of synaptic plasticity in this process [44, 25, 20]. There are many questions that remain open in this field, including the nature of the encoding of inputs, the interaction between the input encoding and the details of the neuron dynamics, and the implications of the details of the rule governing synaptic plasticity (especially questions of long-term

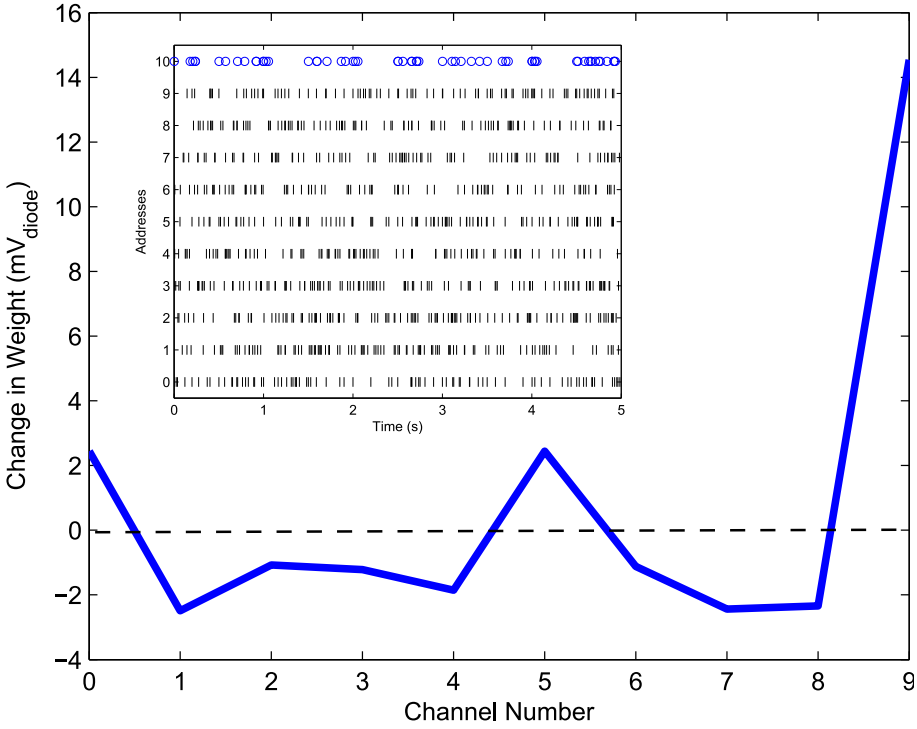
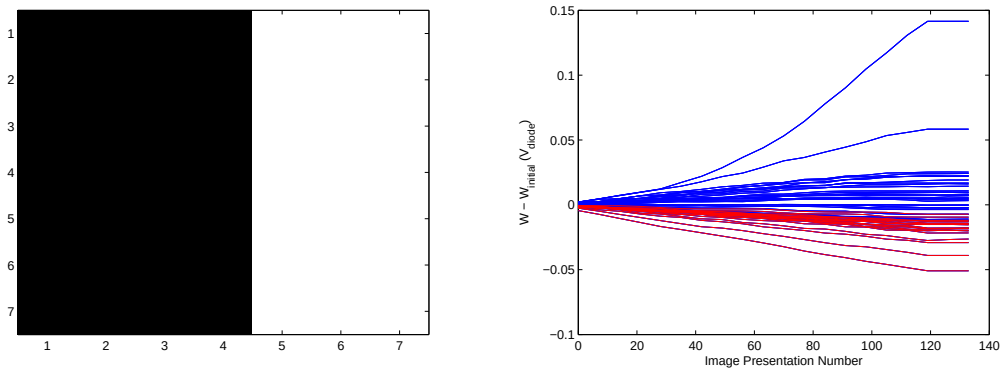


Figure 79: Results of experiment where synchronized inputs among channels 0, 5, and 9 were sparsely mixed with random, uncorrelated inputs. All other synapses received only random uncorrelated inputs such that the average rates on all inputs was the same. The synapses that had correlated activity were potentiated while the others were depressed.



(a) (b)

Figure 80: Simplistic model for initial studies of receptive field formation early in the visual pathway. A single neuron receives 49 input synapses, each associated with a pixel in a 7x7 monochromatic image. The image shown in a) is repeatedly presented to the network, using a "time to spike" encoding of the pixel brightness on each synapse. The evolution of the synaptic weights over repeated presentations of the input is shown in b). The blue traces, which represent the synapses on the bright half of the image, tend to get potentiated, while the others tend to be depressed.

stability with STDP).

While the study of these questions using this neuromorphic platform is in its infancy, some interesting observations can still be made by considering an extremely simplified model of receptive field formation in the early visual pathway. The model consists of a single input neuron with 49 input synapses, each of which is associated with a pixel in a 7x7 monochromatic image. In the experiment, the initial synaptic weights are all uniform (with random, mismatch-related variations). The neuron is repeatedly presented with the high-contrast image shown in Figure 80, and the evolution of the synaptic weights is recorded after each image presentation. In this experiment, a "first-to-spike" encoding is used, in which all of the inputs spike once in response to each image presentation, and the brighter the pixel, the earlier the spike occurs. Not surprisingly, the group of synapses representing the bright pixels in the image get potentiated with respect to those representing the dark pixels.

Perhaps more interesting is the fact that there is significant spread in the learning rates for the different synapses, resulting in an increasingly varied population of synaptic weights, as shown in Figure 80. A few synapses outpace the rest, and in experiments with less synchronized input patterns, these "fast-potentiating" sysnapses end up dominating the spiking behavior of the target neuron, resulting in depression of all of the other synapses. This positive feedback is a common feature of Hebbian learning, and modelers often deal with it by setting maximum and minimum limits on the synaptic weights that can be achieved through learning. The possibility of taking such an approach by choosing tunneling and injection parameters such that the synaptic weights naturally saturate upon reaching some high level. Preliminary characterization of the injection and tunneling rates' dependence on the floating gate voltage confirms the intuition that such a saturation will happen (as the floating gate voltage decreases, the rate of tunneling increases while the rate of injection eventually declines). However, it appears that the weight at which this saturation occurs will always be extremely large (regardless of the voltages used for injection and tunneling), which is undesirable because ideally the weight should be confined to an arbitrarily specifiable range.

Chapter VIII

CONCLUSIONS AND FUTURE DIRECTIONS

Chapter 1 described the motivation behind neuromorphic engineering and analog computation and introduced the foundation of this research: analog VLSI, floating-gate transistors, and subthreshold circuit design. Chapter 2 described the hardware, software, and algorithms that were developed in order to enable the design and application of large-scale systems based on floating gate transistors. As an example, these tools were applied for the demonstration of a variety of circuits for analog signal processing on a field-programmable analog array. Chapter 3 presented new observations about the characteristics of electron tunneling in the individual floating gate transistor, and applied novel characterization techniques to the examination the behavior of populations of these devices in an array. Chapter 4 presented a novel FPAA that facilitates the implementation of a special class of circuits with adaptive capabilities that are enabled by continuous-time hot electron-injection and tunneling. The properties of the floating-gate nFET, including the newly-observed transition between hot electron injection and hot hole injection. Several canonical circuits demonstrating floating-gate based adaptation were implemented. Chapter 5 explained the architecture and circuit-level design of a configurable neuromorphic IC that features 100 channel model-based neurons capable of implementing STDP, 30,000 synapses for interconnect, and an AER interface. Measured data demonstrating the functionality of all of the blocks was shown. Chapter 6 showed how this neuromorphic IC can be applied to simulate networks of neurons with interesting computational features such as spatio-temporal pattern generation and detection, winner-take-all competition, stable nonvolatile memory, and wavefront-based path planning. Finally, Chapter 7 described

studies of STDP-based synaptic plasticity in networks on neurons simulated using the neuromorphic IC.

The results presented in this thesis constitute significant advances along this particular line of research in neuromorphic engineering. This is the first time that a system has allowed simulations of more than a few neurons of this type to be performed. In fact, the neuron array IC presented here is the largest array ever built using any type of neurons models other than the very simplistic integrate and fire model. It is one of only a few neuromorphic IC designs in the world that has the capability of implementing networks with STDP. It offers a significant advantage over many other designs in that the pervasive use of floating-gate technology in the IC provides the ability to cancel device mismatch due to process variation and also the flexibility to tune a wide variety of parameters in a very compact manner. While the development of this neuromorphic IC was a significant accomplishment, equally important was the development of the necessary tools to get the most out of this system.

After taking a moment to appreciate all of this great progress, I will indulge in echoing the eternal refrain of the researcher: "The current results are just the beginning. Listen to what we can do in the future...". This is the fun part, wherein we can zoom out our perspective to the highest level (much like the prince of all cosmos after rolling up everything in the world into a Katamari), and look forward at the possibilities in the future, setting aside the technical details that consume the days and sleepless nights of engineers. One of the exciting possibilities for the future of this research is the scaling up to larger networks. We have estimated that an IC that fills an entire reticle (2.5 cm on a side) in 350nm CMOS could implement 3 to 5 thousand neurons and about a million synapses. Moreover, if the density scales proportionally to the square of the process feature size, a die of that same size in a modern process such as 22nm or 45nm could contain hundreds of thousands of model neurons and

hundreds of millions of synapses. The scale could be increased another order of magnitude or more by making use of 3-dimensional IC fabrication technologies. With all of this scaling, these millions of neurons and billions of synapses all would fit within about a cubic inch of matter. If a larger size like a desktop computer (or even a supercomputer) were allowable, techniques such as wafer-scale integration and systems employing networks of multiple neuromorphic chips could be used to scale up another order of magnitude or two. For perspective, it's interesting to note that at this size (say, for example, 100 million neurons and 100 billion synapses), the system is still a few orders of magnitude short of the scale of the human brain. This is a testament to the marvel of nanotechnology that is the cellular machinery by which biological neurons are able to self-assemble into a fully developed brain.

All of this talk about scaling brings to mind some important questions, the first of which is "what will we do with these enormous piles of model neurons and synapses?". This is the important challenge that faces systems neuroscientists, regardless of whether they use neuromorphic hardware or more conventional methods of computational modeling. It is easy to answer the question in a vague way, to say "We will use these computational models to test hypotheses about how biological neural systems perform computation". This is a valid answer, and in fact it is the best answer that we have, but putting this plan into practice is going to be challenging. The problem is analogous to putting together a vast jigsaw puzzle without having an image of the completed puzzle to use as a reference. Each researcher in the community has taken up some sets of pieces, and proposed models that fit several pieces together. Some of these models are bound to correct, and some will prove to be incorrect. Other researchers are surveying the existing fragments and speculating as to what the whole picture may look like. In this analogy, to test a hypothesis about neural computation is to ask whether a proposed fragment of several puzzle pieces is put together correctly. The obvious difficulty is that the answer depends

on the context. It depends upon what the rest of the puzzle turns out to look like. It is difficult to really know for sure whether a model of a particular functional or anatomical element in the nervous system has correctly captured the essence of the computation of that element unless one also has a good model for computation in the whole nervous system. While this problem is very difficult, the outlook is not hopeless. In the problem of the jigsaw puzzle, any time a group of pieces fit together to create an image of a recognizable object, it would be reasonable to conclude that a correct configuration of those pieces has been found, even without having the rest of the puzzle completed. In the same way, we can gain some confidence in neural models that are demonstrated to perform successfully in a concrete application (recognizing a spoken word, a face, or the scent of food, to cite some possibilities from the field of perception). The goal, then, is to continue to strive for understanding of how multiple functional blocks fit together to support the intelligence of an animal, and simultaneously to explore and attempt to validate models of the functional blocks themselves. This will be a challenging process that will require innovation and creative thought on the part of many scientists.

The second question with regards to the scaling is how the architecture of the neuron IC presented in this thesis will handle orders of magnitude of scaling. There several of promising aspects of the approach. Firstly, the use of floating-gate transistors should limit the detrimental effects of device mismatch that plague the most modern (smallest feature-size) processes. Secondly, the low-power design techniques employed in this approach should make the crucial problem of thermal management (which is a major limitation in modern digital processors) much more tractable. Thirdly, the computation in the system is performed according to the same parallel, distributed model that is used by biological neural networks, which makes it easy to scale the system to model more neurons. One aspect of the design will not hold up well to scaling over many orders of magnitude. On this neuron array IC with 100 neurons,

it was reasonable to support a fully configurable architecture that allows all-to-all connectivity among the neurons. This approach requires a number of synapses that increases quadratically with the number of neurons, which would be hard to sustain and would result in many orders of magnitude more synapses per neuron than what is observed in real nervous systems. How will this problem be addressed? One possibility would be to arrange the neuromorphic IC into a hierarchy of smaller networks, where the available synaptic connections are dense in each neuron's immediate locality, but increasingly sparse at the more global levels of the hierarchy. Since this could potentially constrain the set of models that could be implemented on the system, these design choices would have to be informed by surveys of the topologies of network models to be simulated. A second concern is regarding the AER interface to the IC. On the one hand, when analyzing simulations it would be ideal to have a record of all of the spike times of all of the neurons. On the other hand, for networks with a large number of neurons, such a data stream becomes difficult to communicate, store, and even analyze. This issue will likely be handled by reducing the number of neurons that are can be directly read or stimulated by way of the AER interface during a single experiment. It is conceivable that the system could maintain access to all neurons in a configurable way, such that the user specifies which neurons will receive AER inputs, and which ones will have their spiking activity reported over the AER interface. Alternatively a set of neurons may explicitly be designated as "hidden layer neurons" (the multi-layer perceptron terminology seems appropriate here), with others being specified as "input neurons" or "output neurons", each set having the corresponding AER connectivity.

In summary, the jigsaw puzzle that is neural computation will be pieced together by efforts to find neural models that perform useful computational tasks, and by continually striving to understand how the various models can combine together to form an intelligent system. This process will require countless model network simulations

as a means of testing hypotheses and validating the performance of pieces of the puzzle. The tools and techniques presented in this thesis have tremendous potential to aid in this process, and I await the future progress in this field with great anticipation.

REFERENCES

[1] M. Abeles, H. Bergman, E. Margalit, and E. Vaadia. Spatiotemporal firing patterns in the frontal cortex of behaving monkeys. *Journal of Neurophysiology*, 70(4):1629–1638, 1993.

[2] M. Abeles, G. Hayon, and D. Lehmann. Modeling compositionality by dynamic binding of synfire chains. *Journal of Computational Neuroscience*, 17(2):179–201, 2004.

[3] R. Ananthanarayanan, S.K. Esser, H.D. Simon, and D.S. Modha. The cat is out of the bag: cortical simulations with 10 9 neurons, 10 13 synapses. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 63. ACM, 2009.

[4] H.M.R. Arnoldi, K.H. Englmeier, and W. Brauer. Translation-invariant pattern recognition based on synfire chains. *Biological cybernetics*, 80(6):433–447, 1999.

[5] J. Arthur and K. Boahen. Learning in silicon: timing is everything. *Advances in neural information processing systems*, 18:75, 2006.

[6] A. Bandyopadhyay, G.J. Serrano, and P. Hasler. Programming analog computational memory elements to 0.2% accuracy over 3.5 decades using a predictive method. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pages 2148–2151. IEEE, 2005.

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

[8] A. Basu, C. Petre, and P. Hasler. Bifurcations in a silicon neuron. In *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pages 428–431. IEEE, 2008.

[9] A. Basu, C.M. Twigg, S. Brink, P. Hasler, C. Petre, S. Ramakrishnan, S. Koziol, and C. Schlottmann. Rasp 2.8: A new generation of floating-gate based field programmable analog array. In *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*, pages 213–216. IEEE, 2008.

[10] G. Bi and M. Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *The Journal of Neuroscience*, 18(24):10464–10472, 1998.

[11] G. Billings and M.C.W. Van Rossum. Memory retention and spike-timing-dependent plasticity. *Journal of neurophysiology*, 101(6):2775–2788, 2009.

[12] S. Brink, S. Ramakrishnan, P. Hasler, R. Wunderlich, A. Basu, and B. Degnan. A learning-enabled neuron array ic based upon transistor models of biological phenomena. *Biomedical Circuits and Systems, IEEE Transactions on (in press)*, 2012.

[13] Capo Caccia. Exploring network architectures with the facets hardware and pynn. `http://capocaccia.ethz.ch/capo/wiki/2010/facetshw10`, 2010. Accessed: 5/31/2012.

[14] P. Camilleri, M. Giulioni, V. Dante, D. Badoni, G. Indiveri, B. Michaelis, J. Braun, and P. Del Giudice. A neuromorphic avlsi network chip with configurable plastic synapses. In *Hybrid Intelligent Systems, 2007. HIS 2007. 7th International Conference on*, pages 296–301. IEEE, 2007.

[15] N. Caporale and Y. Dan. Spike timing-dependent plasticity: a hebbian learning rule. *Annu. Rev. Neurosci.*, 31:25–46, 2008.

[16] R. Chawla, A. Bandyopadhyay, V. Srinivasan, and P. Hasler. A 531 nw/mhz, 128× 32 current-mode programmable analog vector-matrix multiplier with over two decades of linearity. In *Custom Integrated Circuits Conference, 2004. Proceedings of the IEEE 2004*, pages 651–654. IEEE, 2004.

[17] A.P. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger. Pynn: a common interface for neuronal network simulators. *Frontiers in neuroinformatics*, 2, 2008.

[18] BE Deal, EH Snow, and CA Mead. Barrier energies in metal-silicon dioxide-silicon structures. *Journal of Physics and Chemistry of Solids*, 27(11-12):1873–1879, 1966.

[19] T. Delbruck. Bump circuits for computing similarity and dissimilarity of analog voltages. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 1, pages 475–479. IEEE, 1991.

[20] A. Delorme, L. Perrinet, and S.J. Thorpe. Networks of integrate-and-fire neurons using rank order coding b: Spike timing dependent plasticity and emergence of orientation selectivity. *Neurocomputing*, 38:539–545, 2001.

[21] L. Dorst and K.I. Trovato. Optimal path planning by cost wave propagation in metric configuration space. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 1007, page 186, 1988.

[22] D. Durstewitz, J.K. Seamans, and T.J. Sejnowski. Neurocomputational models of working memory. *nature neuroscience*, 3:1184–1191, 2000.

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

[24] E. Farquhar and P. Hasler. A bio-physically inspired silicon neuron. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 52(3):477–488, 2005.

[25] P. Földiák. Learning invariance from transformation sequences. *Neural Computation*, 3(2):194–200, 1991.

[26] W. Gerstner and W.M. Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge Univ Pr, 2002.

[27] RH Good and E.W. Müller. Handbuch der physik. page 188.

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

[29] J.D. Gray, C.M. Twigg, D.N. Abramson, and P. Hasler. Characteristics and programming of floating-gate pfet switches in an fpaa crossbar network. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pages 468–471. IEEE, 2005.

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

[31] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[32] P. Hasler, C. Diorio, B.A. Minch, and C. Mead. Single transistor learning synapses. In *Advances in Neural Information Processing Systems 7*. Citeseer, 1995.

[33] P. Hasler, B.A. Minch, and C. Diorio. An autozeroing floating-gate amplifier. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 48(1):74–82, 2001.

[34] Donald O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York, new edition edition, June 1949.

[35] A.L. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Bulletin of mathematical biology*, 52(1):25–71, 1990.

[36] A.L. Hodgkin and B. Katz. The effect of sodium ions on the electrical activity of the giant axon of the squid. *The Journal of physiology*, 108(1):37, 1949.

[37] B. Hutcheon, R.M. Miura, and E. Puil. Subthreshold membrane resonance in neocortical neurons. *Journal of Neurophysiology*, 76(2):683–697, 1996.

[38] G. Indiveri, E. Chicca, and R. Douglas. A vlsi array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *Neural Networks, IEEE Transactions on*, 17(1):211–221, 2006.

[39] E. Izhikevich. Simulation of large-scale brain models. `http://www.izhikevich.org/human_brain_simulation/Blue_Brain.htm#Simulation%20of%20Large-Scale%20Brain%20Models`, 2005. Accessed: 8/6/2012.

[40] E.I. Knudsen. The hearing of the barn owl. *Scientific American*, 1981.

[41] S. Koziol, C. Schlottmann, A. Basu, S. Brink, C. Petre, B. Degnan, S. Ramakrishnan, P. Hasler, and A. Balavoine. Live demonstration: Hardware and software infrastructure for a family of floating-gate based fpaas. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 2793–2793. IEEE, 2010.

[42] M. Kucic, P. Hasler, J. Dugger, and D. Anderson. Programmable and adaptive analog filters using arrays of floating-gate circuits. In *Advanced Research in VLSI, 2001. ARVLSI 2001. Proceedings. 2001 Conference on*, pages 148–162. IEEE, 2001.

[43] J. Lazzaro, S. Ryckebusch, M.A. Mahowald, and C.A. Mead. Winner-take-all networks of o (n) complexity. Technical report, DTIC Document, 1988.

[44] R. Linsker. Self-organization in a perceptual network. *Computer*, 21(3):105–117, 1988.

[45] B. Marr, B. Degnan, P. Hasler, and D. Anderson. Scaling energy per operation via an asynchronous pipeline. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on (in press)*, (99):1–5, 2012.

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

[47] G. Nagy. Neural networks-then and now. *Neural Networks, IEEE Transactions on*, 2(2):316–318, 1991.

[48] E. Niebur and C. Koch. Computational architectures for attention. *The attentive brain*, pages 163–186, 1998.

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

[50] S.Y. Peng, M.S. Qureshi, P.E. Hasler, A. Basu, and F.L. Degertekin. A charge-based low-power high-snr capacitive sensing interface circuit. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 55(7):1863–1872, 2008.

[51] D.H. Perkel and B. Mulloney. Motor pattern production in reciprocally inhibitory neurons exhibiting postinhibitory rebound. *Science*, 185(4146):181, 1974.

[52] S. Ramakrishnan, P.E. Hasler, and C. Gordon. Floating gate synapses with spike-time-dependent plasticity. *Biomedical Circuits and Systems, IEEE Transactions on*, 5(3):244–252, 2011.

[53] M.E. Robinson, H. Yoneda, and E. Sanchez-Sinencio. A modular cmos design of a hamming network. *Neural Networks, IEEE Transactions on*, 3(3):444–456, 1992.

[54] U. Roth, M. Walker, A. Hilmann, and H. Klar. Dynamic path planning with spiking neural networks. *Biological and Artificial Computation: From Neuroscience to Technology*, pages 1355–1363, 1997.

[55] R. Sarpeshkar. *Efficient precise computation with noisy components: extrapolating from an electronic cochlea to the brain.* PhD thesis, California Institute of Technology, 1997.

[56] J. Schemmel, J. Fieres, and K. Meier. Wafer-scale integration of analog neural networks. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 431–438. IEEE, 2008.

[57] J. Schemmel, A. Grubl, K. Meier, and E. Mueller. Implementing synaptic plasticity in a vlsi spiking neural network model. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 1–6. IEEE, 2006.

[58] C. Schlottmann, C. Petre, and P. Hasler. Vector matrix multiplier on field programmable analog array. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 1522–1525. IEEE, 2010.

[59] C.R. Schlottmann, D. Abramson, and P.E. Hasler. A mite-based translinear fpaa. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, (99):1–1.

[60] C.R. Schlottmann, C. Petre, and P.E. Hasler. A high-level simulink-based tool for fpaa configuration. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, (99):1–1, 2010.

[61] G. Serrano, PD Smith, HJ Lo, R. Chawla, TS Hall, CM Twigg, and P. Hasler. Automatic rapid programming of large arrays of floating-gate elements. In *Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on*, volume 1, pages I–373. IEEE, 2004.

[62] P.D. Smith, M. Kucic, and P. Hasler. Accurate programming of analog floating-gate arrays. In *Circuits and Systems, 2002 IEEE International Symposium on*, volume 5, pages V–489. IEEE, 2002.

[63] E. Takeda, A. Shimizu, and T. Hagiwara. Role of hot-hole injection in hot-carrier effects and the small degraded channel region in mosfet's. *Electron Device Letters, IEEE*, 4(9):329–331, 1983.

[64] S.J. Thorpe. Spike arrival times: A highly efficient coding scheme for neural networks. *Parallel processing in neural systems*, pages 91–94, 1990.

[65] LG Wu and P. Saggau. Presynaptic calcium is increased during normal synaptic transmission and paired-pulse facilitation, but not in long-term potentiation in area ca1 of hippocampus. *The Journal of neuroscience*, 14(2):645–654, 1994.

[66] R.B. Wunderlich, F. Adil, and P. Hasler. A floating gate based field programmable mixed-signal array. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on.*

VITA

Stephen Brink received a B.S.E. in Bioengineering from Arizona State University in 2005. While at Arizona State, he conducted research on first-principles electronic structure calculations of ferromagnetic materials. As an undergraduate he received the Barry M. Goldwater Scholarship. He received a Ph.D. in Bioengineering from Georgia Institute of Technology in 2012. At Georgia Institute of Technology, Stephen researched VLSI circuits for modeling neural systems. He was awarded the NSF IGERT fellowship for hybrid neural microsystems, as well as the NSF graduate research fellowship. His most strongly pursued research interests are in neural computation, large-scale analog signal processing, low-power integrated circuit design, analog floating-gate transistor technology, and semiconductor device physics.