ABSTRACT

Neural Circuit Building Blocks For Showing Stochastic Resonance Using Custom Integrated Circuits

Nathaniel Brown

Director: Scott Koziol, Ph.D.

Research has been done on various neural circuits, and the next step in many cases is creating a physical implementation. Programmable technologies such as the Field Programmable Analog Array (FPAA) combined with helper tools can allow circuit construction at high or low levels of detail, partly bridging the gap between simulation and implementation. This thesis combines previous research on silicon neurons with generating stochastic random numbers, targeting how it may be applied to stochastic resonance and implemented on an FPAA. Particular focus is given to converting a noise amplifier into a form with adjustable variance. Documenting the process of designing circuits for a hardware realization on the FPAA will provide guidance for similar work in the future and make it easier to build up to testing the theory of stochastic resonance.

APPROVED BY DIRECTOR OF HONORS THESIS:

_____

Dr. Scott Koziol, Electrical and Computer Engineering

APPROVED BY THE HONORS PROGRAM:

_____

Dr. Andrew Wisely, Interim Director

DATE: _____

NEURAL CIRCUIT BUILDING BLOCKS FOR SHOWING STOCHASTIC RESONANCE USING CUSTOM INTEGRATED CIRCUITS

A Thesis Submitted to the Faculty of

Baylor University

In Partial Fulfillment of the Requirements for the

Honors Program

By

Nathaniel Brown

Waco, Texas

May 2021

TABLE OF CONTENTS

List of Figures . . . . . **iii**
Acknowledgments . . . . . **vi**
1 Introduction **1**
1.1 Motivation 1
1.2 Thesis Overview 1
2 Background **3**
2.1 Neuromorphic Computing 3
2.2 Stochastic Computing 4
2.3 Stochastic Resonance 6
2.4 FPAA Compared to FPGA 7
2.5 FPAA Design Tools 9
2.6 MATLAB Software and Voltage Follower Test Circuit 11
3 Selected Circuits **15**
3.1 Envelope Detector 15
3.2 Stochastic Random Number Generator 20
3.3 Wijekoon Neuron 25
4 Conclusion **30**
4.1 Summary 30
4.2 Discussion 30

LIST OF FIGURES

1 Target output to show stochastic resonance. Figure generated by author of [1]. 1

2 Components required to achieve the circuit response in Figure 1. The CABs are where most of the configurable hardware is located. The noise amplifier is used to add non-trivial amounts of noise to a sinusoidal input which is given to the silicon neuron. 2

3 Mahowald Retina Pixel. Image reproduced from Figure 2 in [2]. © 1990 IEEE 3

4 Images from Temporal Contrast Vision Sensor, reproduced from Figure 11 in [3]. © 2008 IEEE 4

5 Multiplying two bitstreams together using an AND gate. The order of the bits in the $\frac{4}{8}$ stream determines the output result. Image modeled after Figure 1 in [4]. 5

6 Performance using SR increases up to a point with increasing noise. Image reproduces from Figure 2 in [5]. 6

7 FPAA composition. Image reproduced from Figure 2 in [7] 8

8 CLB and CAB examples reproduced from Figure 2 in [8]. © 2016 IEEE. 9

9 FPAA computational blocks inside routing fabric. Image reproduced from Figure 2 in [8]. © 2016 IEEE 10

10 Path from block diagram to FPAA. Image reproduced from Figure 2 in [7] 11

11 MATLAB FPAA Toolset Directory 11

12 The FPAA_simulink dialog and CAB Elements library. The elements used in the voltage follower circuit are boxed. Elements from the Analog Signal Processing library will be used in other circuits. 12

13 The RASP_design dialog, voltage follower .mdl file, .prg file, and command line compile output. The four buttons used in this section are boxed. 13

14 Routing Analysis Tool (RAT) tool prompt, blank RAT tool, and high detail plot zoomed in on the relevant portion of the FPAA. 14

15 Work from [9] showing an envelope detector with an adjustable charge rate and output of minimum envelope for several frequencies of sinusoid. Images reproduced from Figures 7 and 8 in [9]. © 2007 IEEE 16

16 Minimum envelope detector circuit as a Simulink model 16

17 Minimum envelope detector circuit as developed in LTSpice 17

18 Configurable components within the envelope detector. From left to right, the capacitor, floating gate, and OTA. 17

19 Simulated op amp, contrast to responses in Figure 15 19

20 Circuit to run a Bernoulli trial and noise from transistor [10]. Images reproduced from Figures 1 and 3 in [10]. 20

21 Experimental results for probability select mapping, reproduced from Figure 10 in [10]. 21

22 Bernoulli trial circuit. The comparator at the end is boxed. 22

23 Bernoulli trial circuit modified to output noise by the removal of the boxed comparator. 22

24 Noise amplifier circuit for simulation. Both amplifiers have a gain of A. 23

25 Noise output for gains A=10 and A=20. V_3 in the circuit is set to 1.682 and 1.666 respectively. V_1 and V_2 are both set to 1.65V. 24

26 Noise output for A=50. V_3 in the circuit is set to 1.6575. V_1 and V_2 are both set to 1.65V. 25

27 Wijekoon neuron from [12], modeled after Figure 1 in [11]. The boxed voltage input on the left is the input voltage and the boxed voltage input on the right is where noise would be applied to investigate stochastic resonance. 27

28 Baylor Neuromorphic & Robotic Systems Lab diagram showing response to noisy voltage inputs with different variances. Generated by the author of [1] 28

29 Hardware results from an FPAA: Data gathered from applying an external input to the neuron model in Figure 27, to the ”External Test Input” location. 29

ACKNOWLEDGMENTS

This thesis would not have been possible without the help of several people. First and foremost in providing me with tireless assistance is Dr. Scott Koziol. He has guided me through the whole process: learning about Neuromorphic Computing, structuring papers, professionally formatting, and giving credit in a clear manner to all involved parties. From the Baylor Neuromorphic & Robotic Systems Lab, I would like to thank Matthew Carrano for assistance with Latex and Jacob Boline for his work researching stochastic resonance on silicon neurons, which is heavily referenced in my thesis. Last, I would like to thank the many authors and journals I have relied on in conducting research and clearly presenting the material in my thesis.

CHAPTER ONE

Introduction

Motivation

Analog circuit design can be a time-consuming process. Simulation work may be divorced from realization, and although it may not be expected to catch every problem before the first implementation, shortening the feedback loop greatly accelerates the timeline. Abstraction is also invaluable, as the appropriate level allows the circuit designer to focus on circuit design, rather than extraneous details.

In Figure 1, the response of a silicon neuron to a noisy input is shown. It comes from research examining stochastic resonance, in a simulation environment. The goal is to show the components required to reproduce these results on a programmable physical system, as shown in Figure 2. This thesis will show how to use the Field Programmable Analog Array (FPAA) design tools to create circuits that can be used as building blocks for the stochastic resonance system. The primary contribution of this thesis toward the building blocks is presenting simulation results for adjusting the mean and variance of a noise source.

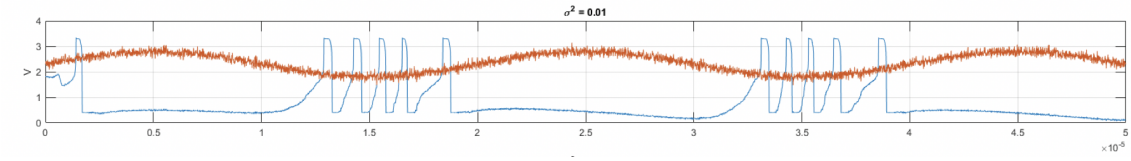
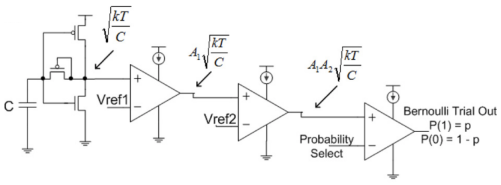


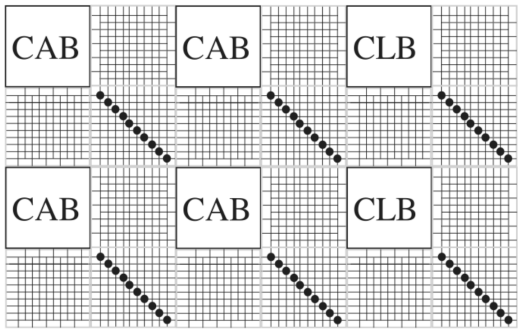
Figure 1: Target output to show stochastic resonance. Figure generated by author of [1].

Thesis Overview

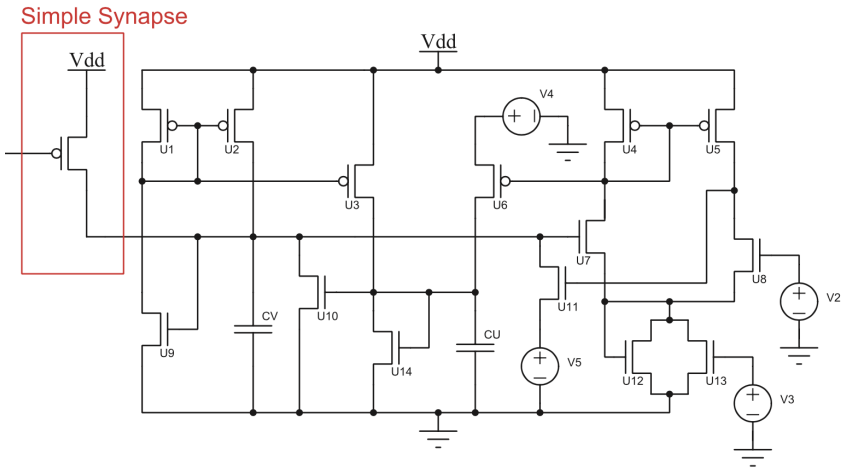
The thesis is broken up into several chapters: background, selected circuits with some simulation and experimental results, and a conclusion. This builds up to



(a) Noise amplifying circuit. Image reproduced from Figure 3 in [10]



(b) Structure of FPAA. Image reproduced from Figure 2 in [8]. © 2016 IEEE



(c) Silicon neuron. Noise would be applied to synapse gate voltage. Image reproduced from Figure 2.12 in [1].

Figure 2: Components required to achieve the circuit response in Figure 1. The CABs are where most of the configurable hardware is located. The noise amplifier is used to add non-trivial amounts of noise to a sinusoidal input which is given to the silicon neuron.

a feasible circuit as suggested in the motivation section, which will connect the different circuits under consideration to the idea of stochastic resonance.

The circuits treated are a basic envelope detector circuit, to verify circuit design approaches used, a stochastic random number generator (modified for noise output), and silicon neuron.

CHAPTER TWO

Background

Neuromorphic Computing

The research in this thesis is in the area of neuromorphic computing. This is a broad area, which can be loosely equated to bio-inspired computing. As a field, it is relatively new, originating at CalTech with Carver Mead. Neuromorphic systems, as defined by Mead in 1990, are based on the principles discernable in the nervous system [2]. His work, applying the subthreshold operating region of CMOS transistors to neural style computation [13, p. 3], is foundational to this thesis.

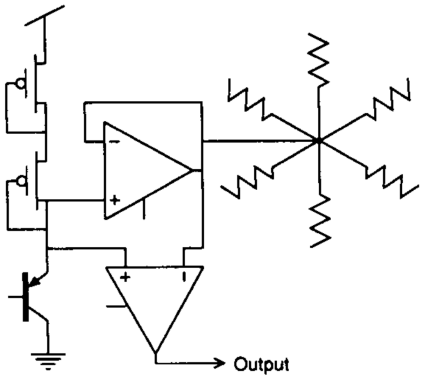


Figure 3: Mahowald Retina Pixel. Image reproduced from Figure 2 in [2]. © 1990 IEEE

Neuromorphic systems should not be expected to match biological ones in every way. The silicon retina created by Mahowald is made up of several pixels, pictured in Figure 3, that each compare illumination to the illumination of their neighbors [2]. A biological retina has several layers with many different components, including rods and cones, horizontal cells, bipolar cells, and ganglion cells [13, pp. 38-39]. This highly interconnected network is modeled by a hexagonal array of pixels connected by resistors [2]. This network is a computing element, which

automatically adjusts the gain to allow visual processing under a wide range of illuminations [2].

Address Event Representation (AER), proposed in 1991, is a protocol where data is encoded as a series of events with information about where the event was generated, the address [13]. In the context of a retina, local memory complements local computation, enabling use of AER. Only when a pixel detects that its value is changing does it send an event to the central processor. This leads to construction of images where most of the data is concentrated in active regions of the field of view, as shown in Figure 4 [3].

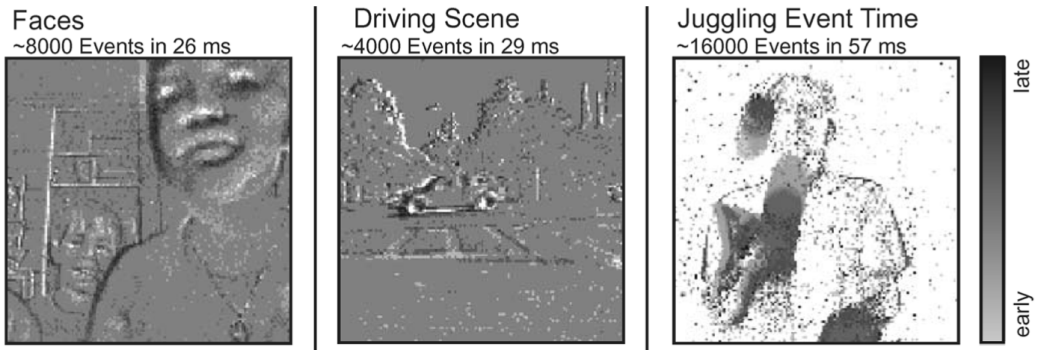


Figure 4: Images from Temporal Contrast Vision Sensor, reproduced from Figure 11 in [3]. © 2008 IEEE

Stochastic Computing

Stochastic Computing (SC), proposed in the 1960s, has been slow to gain traction as conventional digital technology has advanced [4]. It is a paradigm where probabilistic methods are used for computation [14]. While it may not be a mainstream technology, it is valuable because of its concern with power consumption and error tolerance [4]. In fact, recent advances into stochastic computing design methodologies mean that stochastic computing may be making a comeback [15].

Alaghi and Hayes define stochastic numbers as being ones that are represented by a bitstream, where each bit is unweighted and the value is the sum of the bits divided by the length of the bitstream [4]. These stochastic numbers can be viewed

as probabilities, where the number 0.5 represents a one in two chance of an event happening [4].

When a number is represented by a bitstream, certain operations become easier and some harder. Multiplication turns out to be an easy operation, accomplished by using a single AND gate with two inputs [4]. As seen in Figure 5, the ordering of the bits can affect the accuracy of the result. $\frac{4}{8} * \frac{6}{8}$ should be $\frac{3}{8}$, but it could be as small as $\frac{2}{8}$ or as big as $\frac{4}{8}$. If $\frac{4}{8}$ is represented as 11110000, multiplying it by 11111100 yields $\frac{4}{8}$ and multiplying it by 00111111 yields $\frac{2}{8}$

While SC outputs initially have low precision, over time, as the output bitstream increases in length, it has an increasing level of accuracy. A change in the bit ordering of an input to the stochastic multiplier circuit could change the correct answer away from $\frac{3}{8}$ as shown in Figure 5. A bit flip in a longer bitstream would have less effect [4]. Compare this to a bit flip in a traditional number, which could cause 0.011 to change to 0.111, changing the value from $\frac{3}{8}$ to $\frac{7}{8}$ [4].

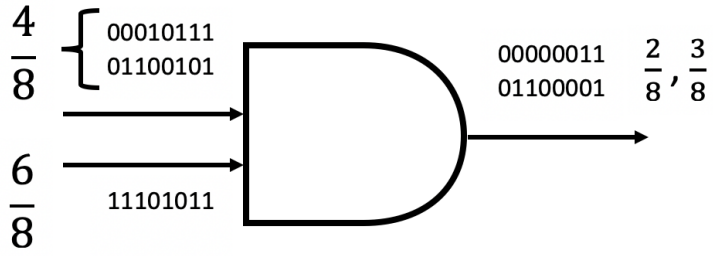


Figure 5: Multiplying two bitstreams together using an AND gate. The order of the bits in the $\frac{4}{8}$ stream determines the output result. Image modeled after Figure 1 in [4].

Generating stochastic numbers is a critical part of being able to use them. As seen in Figure 5, a stochastic circuit can be an AND gate, which would give zero as an output if multiplying two bitstreams together that are inverses of each other. SC works better when the bitstreams are long and randomly ordered, or at least that

any two bitstreams being operated on are uncorrelated [4]. Theoretically, random noise is well-handled by SC paradigms.

Later it will be seen how random noise used to generate stochastic numbers can be modified for other purposes.

Stochastic Resonance

Stochastic resonance (SR) in this thesis refers to "noise-enhanced signal processing" [5]. Typically, noise is thought of as a bad thing, for as the authors of [5] point out, communications engineers spend a lot of time trying to eliminate all effects of noise. Evidently, not all systems can utilize a form of SR without redesign.

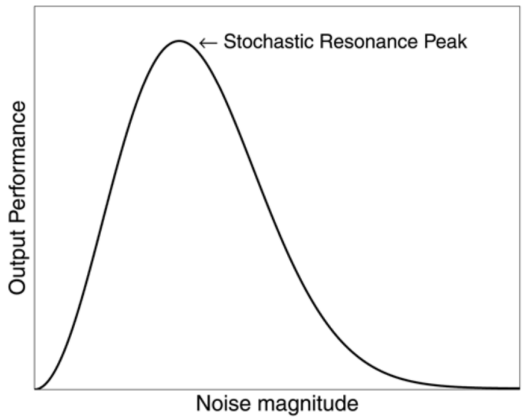


Figure 6: Performance using SR increases up to a point with increasing noise. Image reproduces from Figure 2 in [5].

A simple way to consider implementing SR is to add white noise to a periodic signal [5]. When adding noise to a system, it is expected that there is a point where too much noise would be added, and this is shown in Figure 6.

Could SR lead to more power-efficient systems? It is postulated that biological systems make use of SR [5], and this coupled with the low power usage of biological systems, suggests that this may be feasible.

FPAA Compared to FPGA

The circuits in this paper are intended to be synthesized onto an FPAA, which is similar to a Field Programmable Gate Array (FPGA), as the hardware is programmable through a software interface. FPAAs and FPGAs share a lot common goals, such as prototyping and enabling circuit implementations without fabrication, but differ at the most basic level. In an FPGA, the basic building blocks (i.e. primitives) are digital, such as NAND gates, but in an FPAA, the building blocks include both digital and analog components, such as transistors and amplifiers. Therefore, FPGAs are well-suited to problems that can be reduced to logic problems, while FPAAs excel at tasks such as signal processing, but can also be used as general-purpose logic devices.

A picture of an FPAA is in Figure 7. There are fixed elements on the left side of the diagram, but the main feature is the array of A and D components. In the array of A's and D's, each A stands for a Computational Analog Block (CAB), while each D stands for a Configurable Logic Block (CLB). CLBs and CABs are shown in detail in Figure 8.

An FPGA is predominantly composed of an array of CLBs. At first glance, FPGAs seem to lack features, but in fact the opposite is true, as FPGAs as a device class have had a lot of development and optimization. Programmable logic devices such as FPGAs, along with their hardware programming and run-time mapping, have a chapter in [6], with Figure 11.29 specifically discussing Xilinx FPGAs.

A CLB typically contains basic gates and multiplexers (muxes), which can be used to create logical expressions from inputs, as shown in Figure 8a. An similar treatment of a CLB is given in Figure 11.13 in [6]. Sequential logic elements, such as D flip-flops, may be created with the addition of extra components [6]. The CLB is commonly controlled by on chip RAM, where the memory locations control select lines for the muxes and routing switches [6].

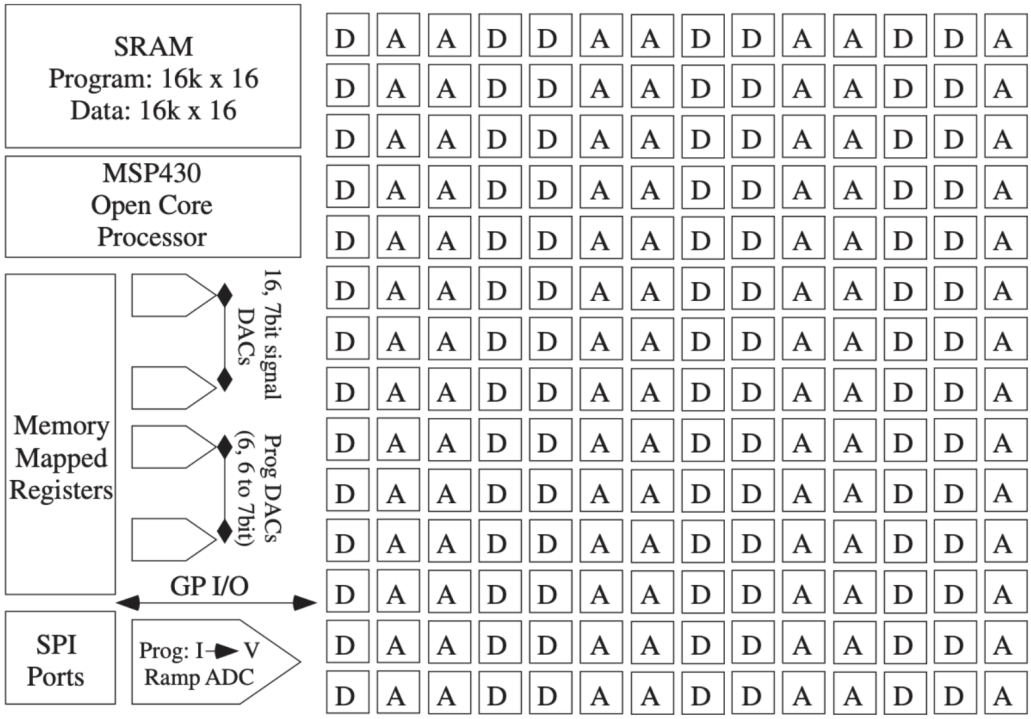
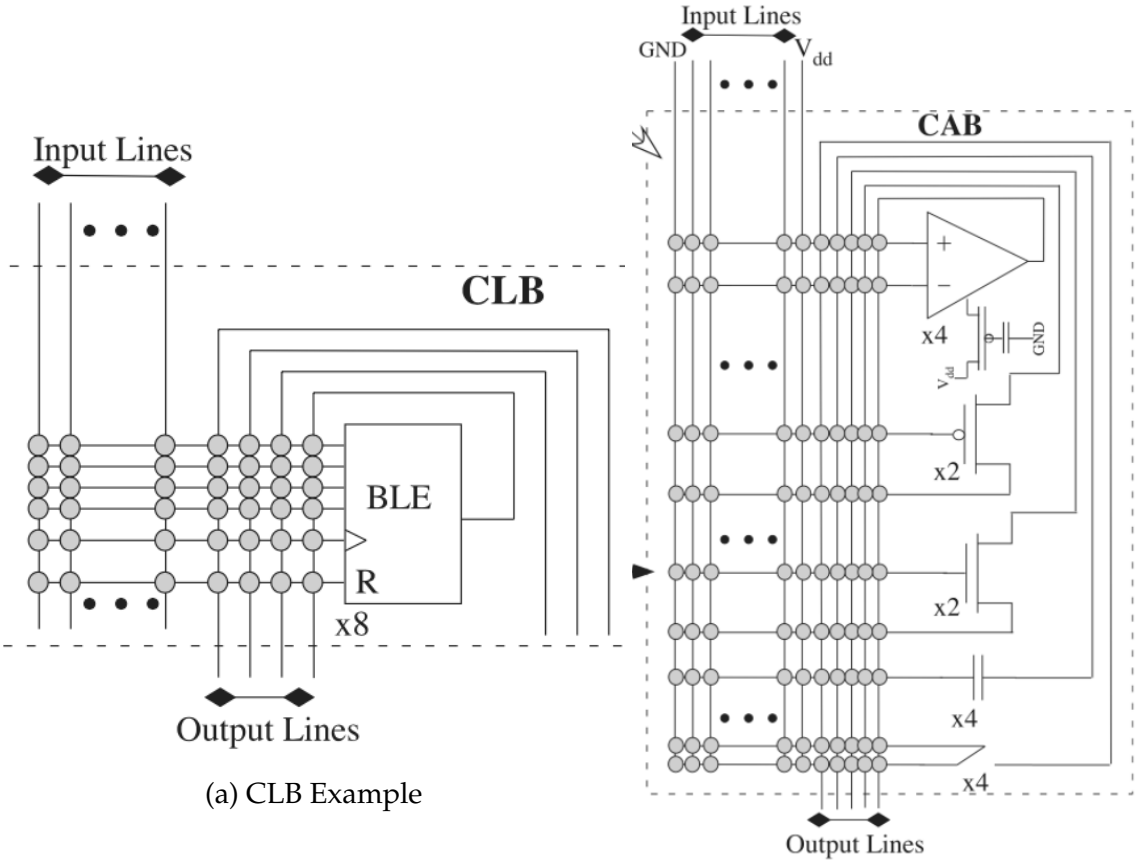


Figure 7: FPAA composition. Image reproduced from Figure 2 in [7]

A CAB has a similar structure, with components including operation transconductance amplifiers (OTAs), transistors, capacitors, and switches [8]. Layout of a CAB is shown in Figure 8b. Routing is accomplished using floating gates (FGs). An FG is a transistor with a capacitor on the gate, which allows a gate voltage to be set and stored without the need to refresh it. FGs can be set all the way on or with a specific bias, in which case electrons are placed on the gate using short bursts of current, until the bias is achieved [16]. FGs are suitable as on switches, conducting approximately $10^{-4}A$, off switches, conducting less than $10^{-10}A$, as well as variable resistors or rough current sources [10].

FPAAs offer large power savings, as illustrated by a word classifier in [8] which offers a 1000x improvement in equivalent multiply-accumulate (MAC) operations. The filter can be implemented with several transistors and capacitors, with the output signal passed on either to another block or off the chip. The classifier identifies words based off of frequency content and requires only 23 μW to run [8].



(a) CLB Example

(b) CAB Example

Figure 8: CLB and CAB examples reproduced from Figure 2 in [8]. © 2016 IEEE.

FPAA Design Tools

How does one program an FPAA? This is a harder question than it might seem at first. While digital hardware-software co-design (i.e. for FPGAs) is a researched topic, the current tools are not adequate for analog co-design [7]. The work done in this area since before 1994 [17] has produced an impressive variety of marketable FPGAs. The tools introduced in [7] are an initial step towards building frameworks and standards that enable design abstraction to make configurable analog circuits more accessible.

The FPAA toolset allows circuit design at a high level, where the circuit designer does not need to know how to implement each individual block [7]. Abstraction allows for block reuse, which leads to greater consistency within and

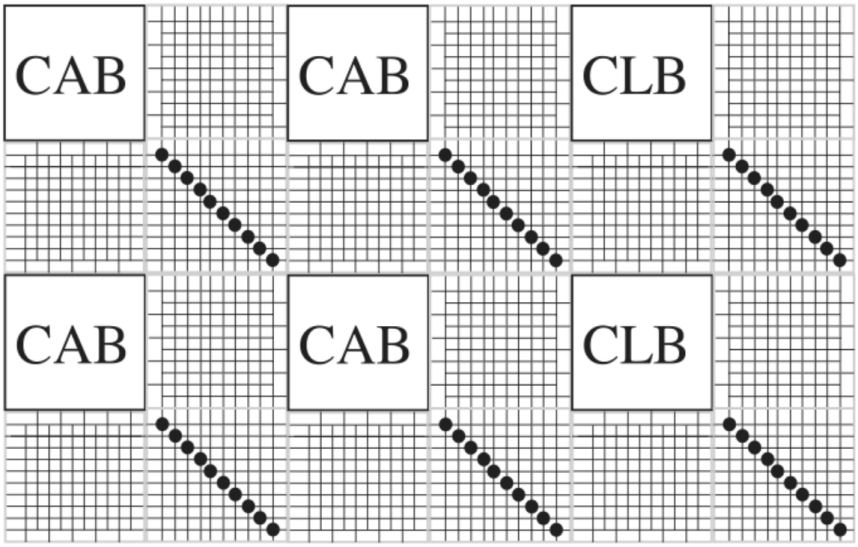


Figure 9: FPAA computational blocks inside routing fabric. Image reproduced from Figure 2 in [8]. © 2016 IEEE

across different circuits. Graphical components belong to one of two broad categories, Level 1 or Level 2 [7]. Level 1 blocks keep a systems designer from being distracted by minor details such as bus size [7]. Level 2 blocks are at the circuit design level, and a network of these may be converted into a Level 1 block [7].

The current version of the FPAA toolset is created with Scilab and Xcos, all open source software [7]. The toolset is packaged in an Ubuntu virtual machine (VM) [7]. Circuits can be created in Xcos' graphical editor, from which the circuit is compiled through x2c [7]. The process, including accounting for different board layouts, is shown in Figure 10.

The version of the toolset used in this work is based on MATLAB Simulink. Specific blocks can be placed in a Simulink model file (.mdl), converted to a SPICE netlist using a "Sim2spice" tool, and converted to a switch list (.prg) using GRASPER. Switch lists in the .prg format are ready for direct programming to the FPAA [18].

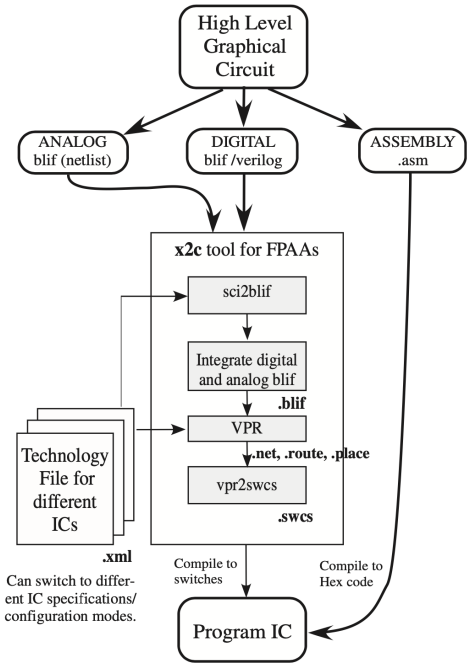


Figure 10: Path from block diagram to FPAA. Image reproduced from Figure 2 in [7]

MATLAB Software and Voltage Follower Test Circuit

This section shows how to implement a very simple circuit using software developed on top of MATLAB. Screen captures are included of the software in use.

The software toolset comes in a directory as shown in Figure 11. Once this is saved on computer, set the working directory of MATLAB to the "Matlab code" subdirectory. (Detailed instructions are found in the file "fpaa setup.pdf" in the usb_drivers subdirectory in Figure 11).

For full functionality, run this on a Windows machine.

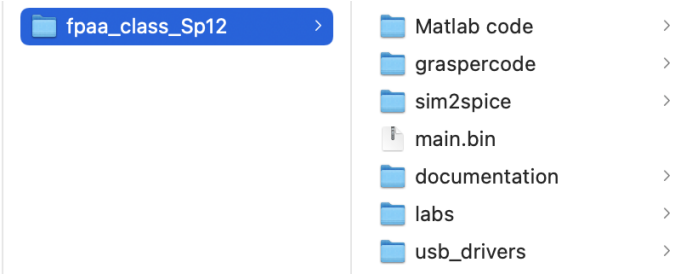


Figure 11: MATLAB FPAA Toolset Directory

When starting up MATLAB, it will run the file startup.m in the Matlab code directory, providing the user with access to the front page of the FPAA toolset.

When starting MATLAB two windows should open, the FPAA Simulink and RASP design dialogs. The FPAA Simulink dialog, as shown in Figure 12, offers easy access to libraries for Simulink. Clicking items will open the corresponding Simulink library. The RASP design dialog, in Figure 13 is where most of the work will take place after a circuit is assembled.

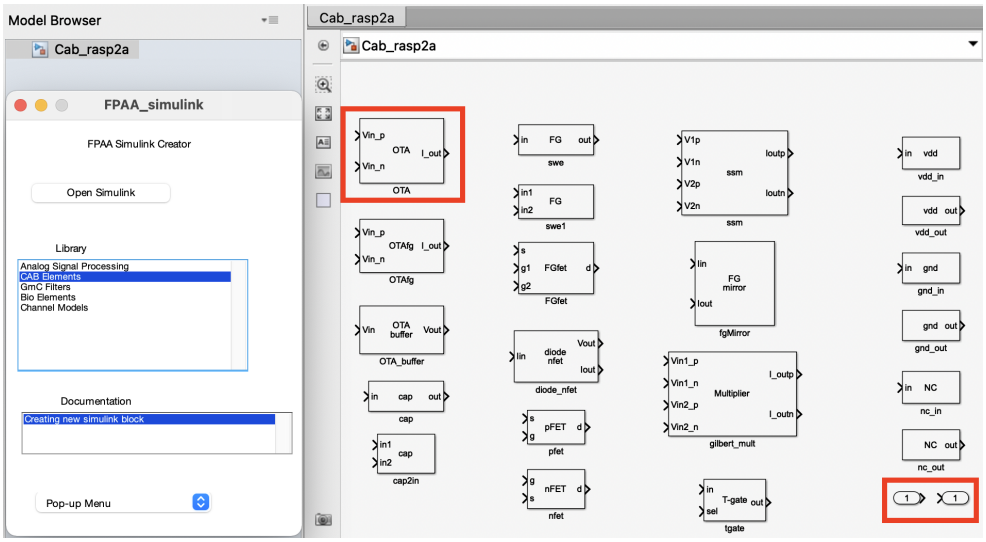


Figure 12: The FPAA_simulink dialog and CAB Elements library. The elements used in the voltage follower circuit are boxed. Elements from the Analog Signal Processing library will be used in other circuits.

In order to open an existing file, first use the "Choose Simulink" button to select a .mdl file, then click the "Open Simulink" button. To compile a circuit, make sure it is saved, then click "Compile Simulink". The file does not need to be open for compilation, but it does need to be selected, as at the top of the RASP design dialog in Figure 13.

A successful compilation results in command line output like that in Figure 13. The file otaFollowerTest.mdl has now been used to generate otaFollowerTest_flattened.prg,

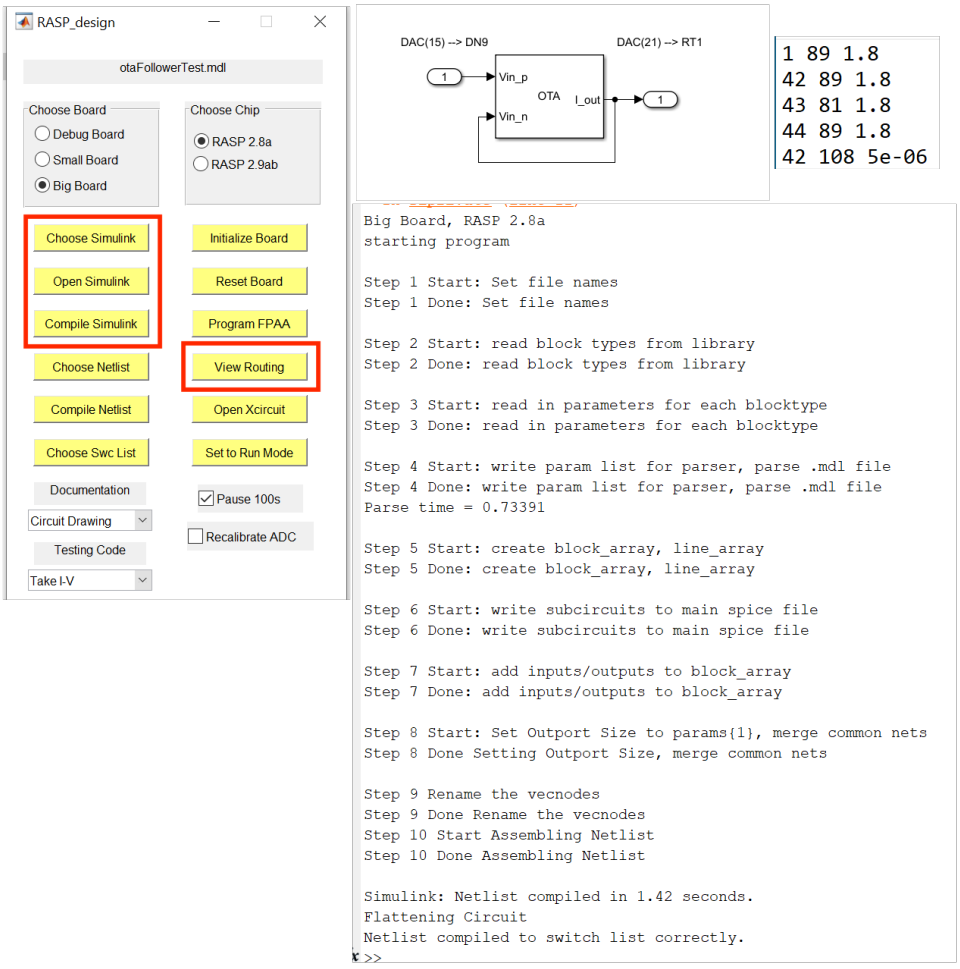


Figure 13: The RASP_design dialog, voltage follower .mdl file, .prg file, and command line compile output. The four buttons used in this section are boxed.

which is shown to the right of the voltage follower block diagram in Figure 13. The five lines shown in the .prg file will be used to program the FPAA.

It is possible to view the .prg file using the Routing Analysis Tool (RAT) [19], which is accessed through the "View Routing" button. Input a RASP version on the command line, and a blank plot will come up as in Figure 14. Select "Low Detail" or "High Detail", then select "Plot / Refresh" to view the routing.

Additional treatment of the Simulink, SPICE, GRASPER, and RAT workflow is given in [19]

Figure 14: Routing Analysis Tool (RAT) tool prompt, blank RAT tool, and high detail plot zoomed in on the relevant portion of the FPAA.

CHAPTER THREE

Selected Circuits

The following circuits are selected as different building blocks for stochastic systems and neural networks, as well as to illustrate use of the different elements in a Computational Analog Block (CAB).

Envelope Detector

This envelope detector is selected to show some of the building blocks available on the FPAA, as well as open a discussion on how to design or simulate circuits taking into account these unique components. This circuit uses a comparator to charge or discharge a capacitor, tracking the lower bound (minimum envelope) of a reference voltage. The key attribute of this circuit is that the charging rate (when voltage is too low) should be slower than the discharge rate (when the voltage is too high), resulting in an output that forms an "envelope". Ideally, if a signal is bounded by an upper and a lower function, the envelope detector will output the lower function.

Both the charging rate and discharging rate are limited to certain frequencies. Observe in Figure 15 how there is a constant rate of change of voltage up to a point based on different circuit and input parameters. There is some amount of error inherent to the process, as the first subfigure shows and discharge rate is more than able to keep up track the signal when it lowers as can be seen in the second subfigure.

The charging rate is configurable based upon an applied voltage, while discharging is fixed by a single transistor with the gate attached to the output of the comparator. However, both are affected by the capacitor, so a capacitance value

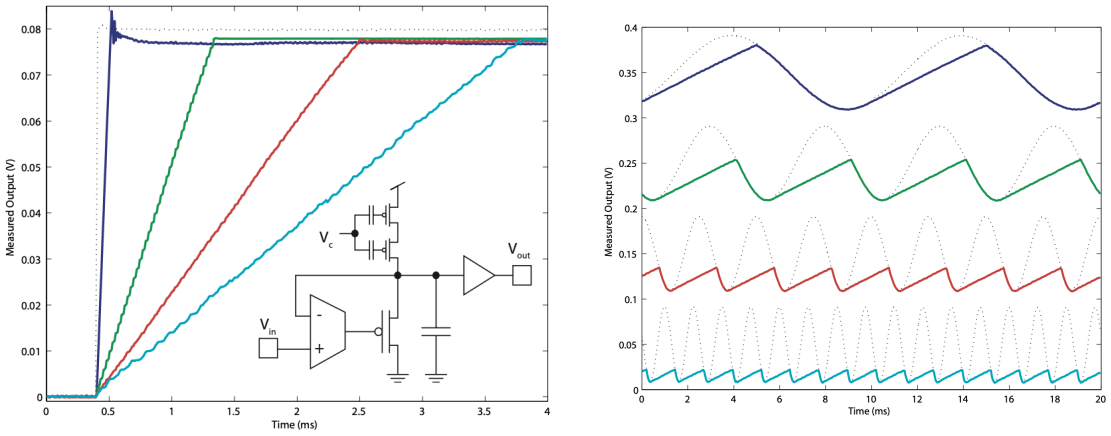


Figure 15: Work from [9] showing an envelope detector with an adjustable charge rate and output of minimum envelope for several frequencies of sinusoid. Images reproduced from Figures 7 and 8 in [9]. © 2007 IEEE

could be selected so that the circuit can track a certain range of frequencies just by adjusting transistor gate voltages. As CABs have multiple capacitors, one possible method of setting a larger capacitor value is to connect them in parallel. In [9], tests are shown for the minimum envelope detector with the input frequency ranging between 100 and 800 Hz, suggesting that it is simple for an FPAA to work with frequencies in that range.

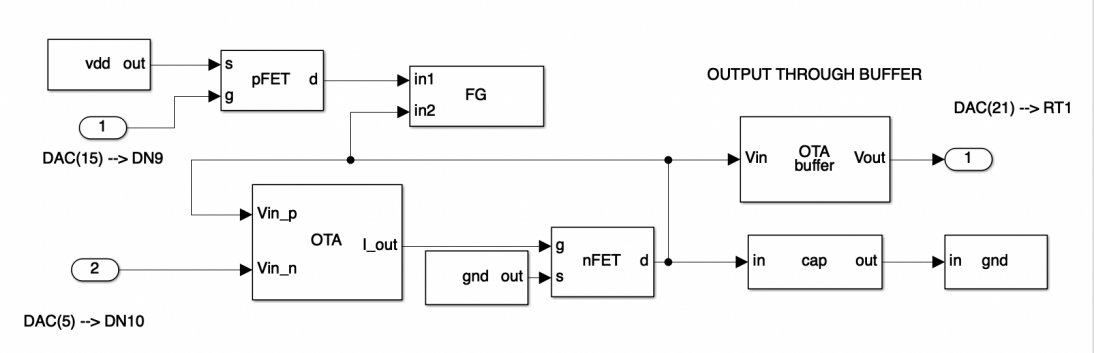


Figure 16: Minimum envelope detector circuit as a Simulink model

The circuit in Figure 16 is intended to be a faithful representation of the circuit in Figure 15. The circuit include several analog components from the CAB: one op-amp as a comparator, another op amp as a voltage follower, a capacitor, and

several transistors. One floating gate transistor (FG) is used to connect the input to the capacitor for charging, and a pFET and nFET are also used for charging and discharging.

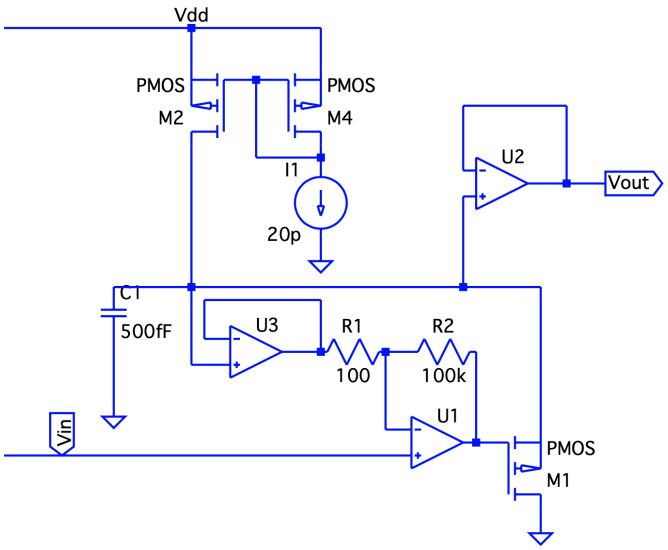


Figure 17: Minimum envelope detector circuit as developed in LTSpice

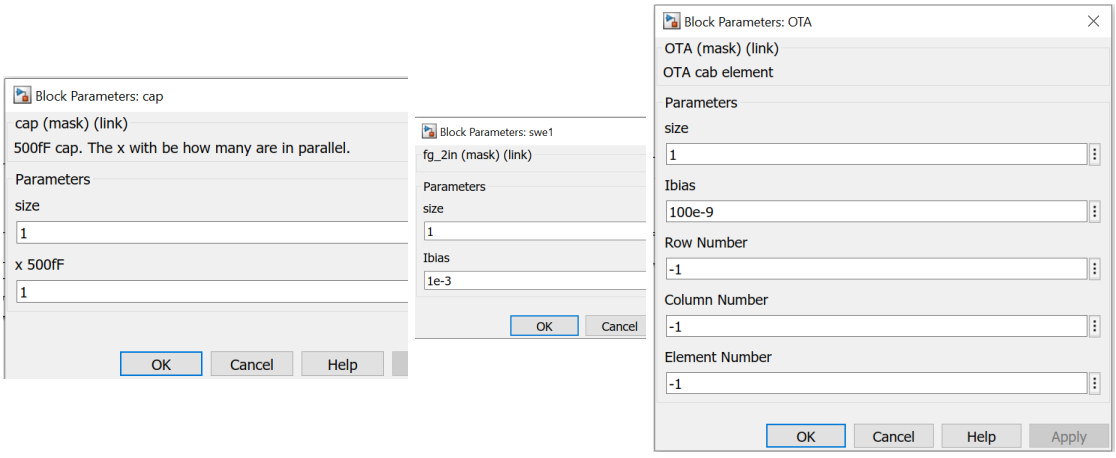


Figure 18: Configurable components within the envelope detector. From left to right, the capacitor, floating gate, and OTA.

This circuit shows a couple possible levels of abstraction offered by the FPAA. Once the structure is determined as in Figure 16, there remains to be set fixed bias current values for the OTA and FG, as well as externally the applied voltage.

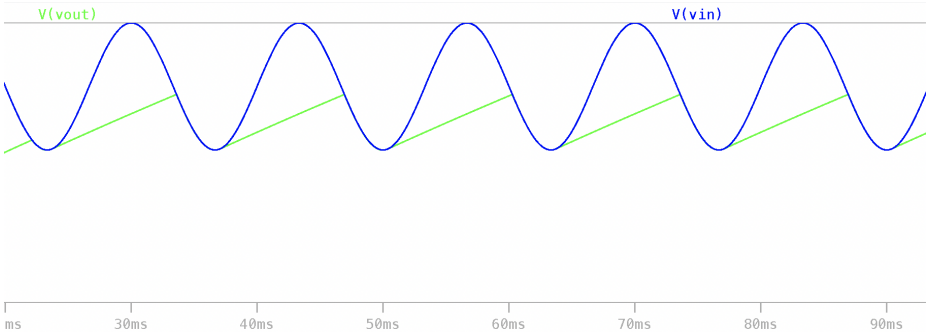
Figure 18 shows how a capacitor's capacitance can be modified and FG and OTA bias currents may be set. Supposing that the design requires a specific bias current (i.e. the charging rate may be linked to it), then this eliminates extra calculations for setting the applied voltage. On the other hand, sometimes a voltage may fit more naturally into calculations. Care should be taken not to complicate circuit design by use of the FPAA.

Figure 17 shows a simulation model of the envelope detector circuit. In order to gain a realistic expectation, specifically of what the output of the Simulink model would look like on the FPAA, simulation needs to be done with a realistic VDD, here selected to be 3.3V, and a small capacitor (on the order of 500fF, which is the base capacitor value on the FPAA), as well as a circuit element to serve a similar function as the FG (on switch with limited current throughput). These are accounted for in the circuit in Figure 17. Open loop op amps do not simulate very well, so closed loop op amps are used with appropriate gains. Note that a buffer amplifier has to be used to prevent feedback from the comparator back into the capacitor.

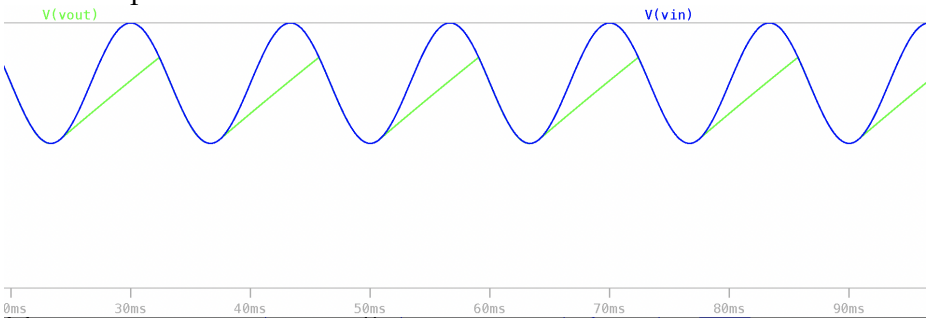
The circuit also has some of its design dictated by the use of the FGs. It is required in the Simulink model to use a FG as a short circuit, although, as this is not assumed to be a zero resistance element, it is placed above the capacitor so that it inhibits charging rather than discharging. The implementation of this circuit in [9] shows FGs in a cascode configuration, which gives a fairly steady current for a wide range of input voltages. To simply approximate both the cascode and FG a single transistor is used in the SPICE simulation, set with a gate voltage for a small (order of 100pA) saturation current.

Figure 17 shows the Simulated circuit in LTSpice where VDD is set to 3.3V, the capacitor is 500fF, both op amps are ideal with positive and negative voltages (note that the FPAA would not have a negative supply voltage), and a pFET (in

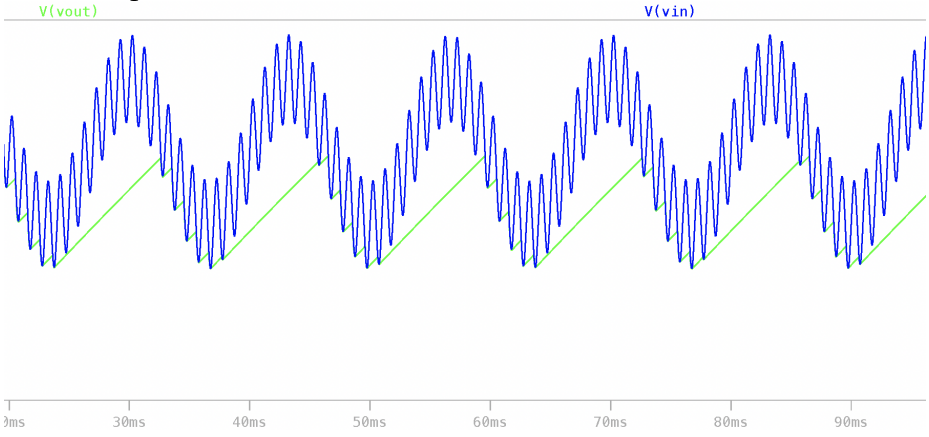
place of the upper pFet/FG combination) biased to allow 20pA or 40pA of current in saturation. The result of this circuit is shown in Figure 19. Higher frequency components would be filtered out.



(a) Minimum envelope detector response to 75 Hz signal with input biased at 20pA.



(b) Minimum envelope detector response to 75 Hz signal with input biased at 40pA.



(c) Minimum envelope detector response to signal with 75 Hz and 1kHz components. Input biased at 40pA.

Figure 19: Simulated op amp, contrast to responses in Figure 15

Stochastic Random Number Generator

One way to generate random bits is to compare random voltages to a set value. For normally distributed noise with a mean of 0, comparing the voltage difference to 0 on a comparator should result in a 50% chance of an output of 1.

This process is well-documented in [10]. The rms noise level for a 500fF capacitor is about 100μV, which must go through multiple amplifiers before being compared to a reference value on the order of a volt. An overview of this is shown in Figure 20, where every time the output is sampled counts as a trial. The quickest the output changes for this circuit is 208ps, which is enough to keep pace with any rate of consumption of random bit values [10].

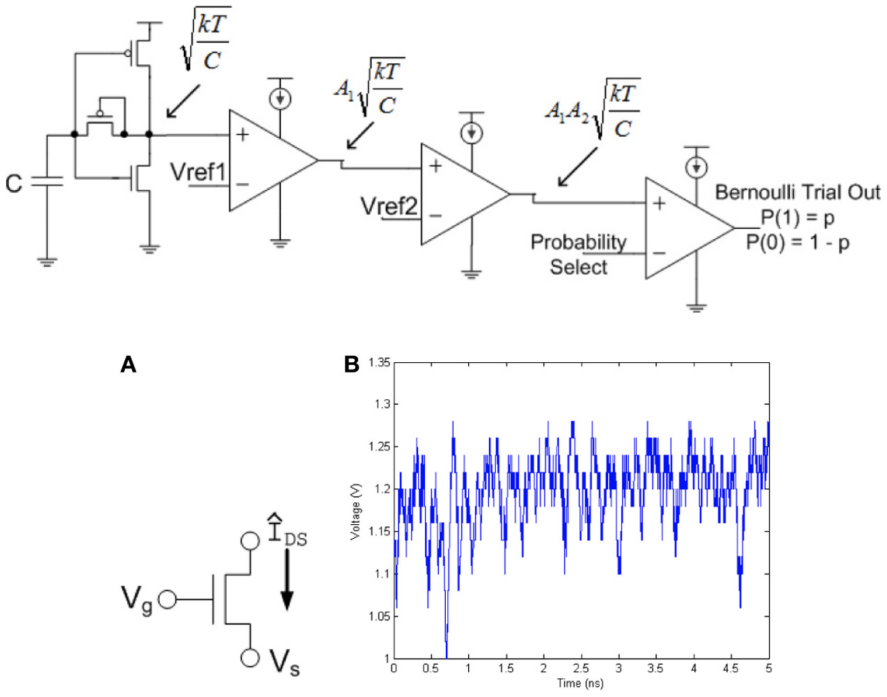


Figure 20: Circuit to run a Bernoulli trial and noise from transistor [10]. Images reproduced from Figures 1 and 3 in [10].

Simulating this requires use of an explicit noise source, while synthesis and testing require characterizing the circuit to determine which reference voltages map to which probabilities. Tests in [10] show that for the implemented Bernoulli proba-

bility circuit, a probability select voltage of 0.1V corresponds to a 100% probability of outputting a 1, with 1.4V corresponding to a 50% chance of a 1, as shown in Figure 21. The probability distribution should be symmetric, so reference voltages greater than 2.8V should have a 0% chance of outputting a 1.

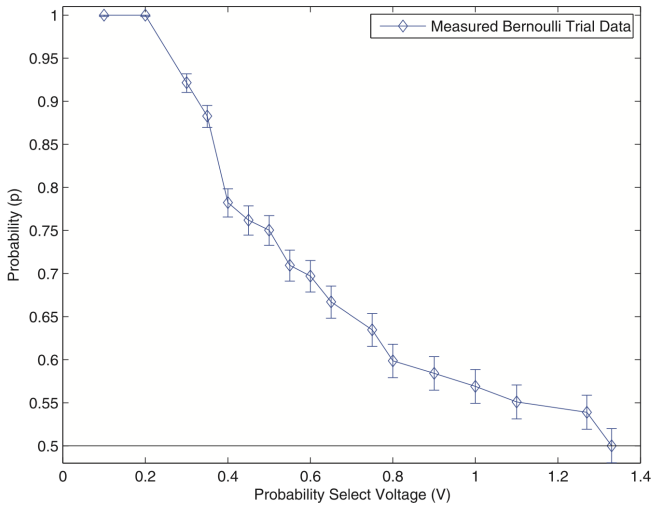


Figure 21: Experimental results for probability select mapping, reproduced from Figure 10 in [10].

Data from a series of Bernoulli trials can be used to create different probability functions, such as exponential distributions [10]. This could be done over multiple clock cycles or, using several circuits as inputs to a priority encoder, just two clock cycles [10].

If a noise source is required, but not a stochastic number, the circuit in Figure 20 could be modified so the output is amplified noise instead of a discrete 1 or 0. This could be useful for the Wijekoon neuron in the following section, if noise needs to be on the order of volts instead of microvolts or millivolts. However, some care must be taken to properly set the noise level, as simply amplifying it will also amplify the variance of the noise (see Figure 25).

The Bernoulli trial circuit in Figure 22 is composed of a noise source and a series of amplifiers, to match the diagram in Figure 20. The third amplifier needs to have

a high gain, likely different from that of the other two amplifiers, in order for its output to be only high or low. Note that here OTAs are used as amplifiers, and these have a bias current which sets the open-loop gain.

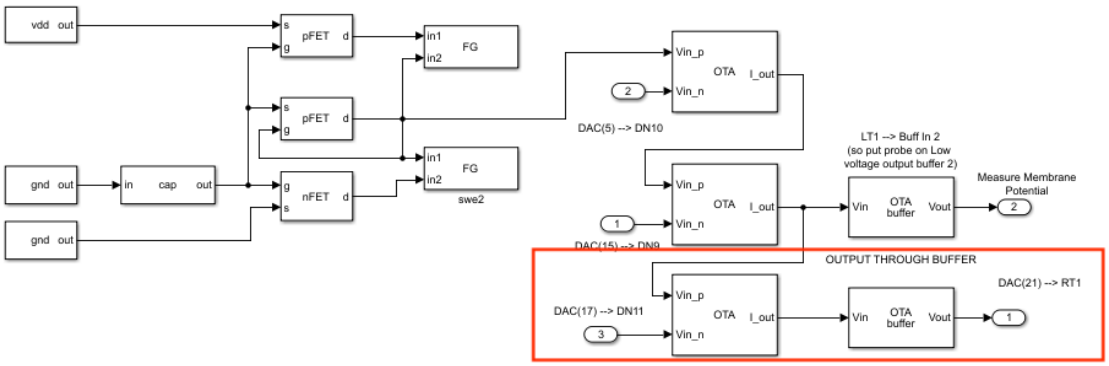


Figure 22: Bernoulli trial circuit. The comparator at the end is boxed.

Figure 23 shows the portion of the Bernoulli trial circuit in which noise is amplified. A more useful version will use a summing op-amp to modify the mean of the noise output to a useful value such as 2 volts. For conducting a range of tests with different noise variances, the noise amplitude needs to be adjustable. It is assumed that the OTAs can be used as the op amp in a closed-loop amplifier. If this is not the case, a different approach would need to be used to vary the noise.

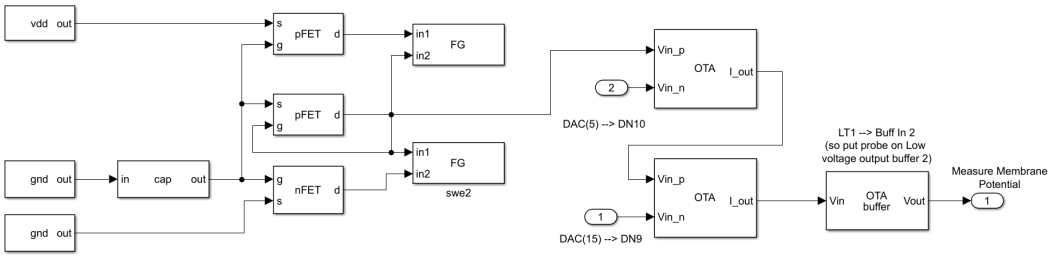


Figure 23: Bernoulli trial circuit modified to output noise by the removal of the boxed comparator.

As simulating this circuit is not straightforward, the OTAs are replaced with closed loop ideal amplifiers and the different parameters are adjusted. It is as-

sumed that the initial noise is biased at half of the supply voltage, so the task is to scale the noise and offset it as desired.

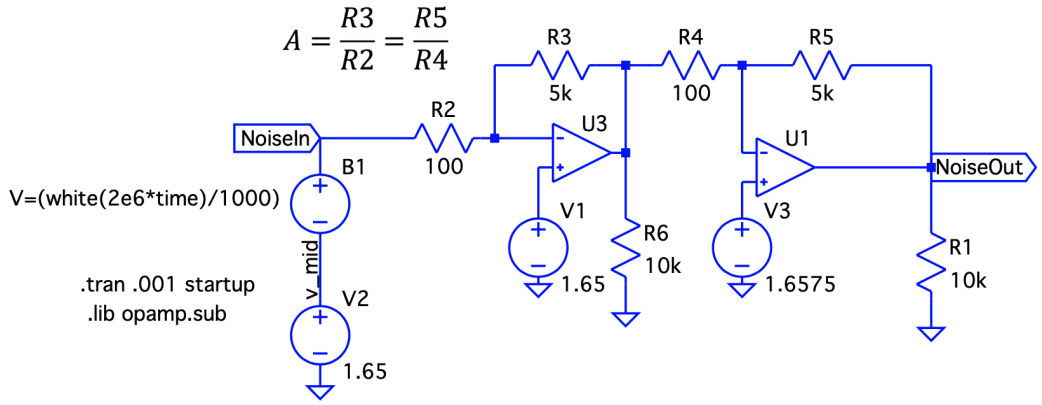


Figure 24: Noise amplifier circuit for simulation. Both amplifiers have a gain of A.

Simulation results from the circuit in Figure 24 are shown in Figures 25 and 26. When applying this noise to the silicon neuron in the following section, it needs to have a mean of 2V (or put another way, noise with a mean of zero is added to two volts). In order to change the variance of the noise, the gains of the op amps may be changed. Changing the offset comes from a minute change on the second op amp's non-inverting input.

Modifying voltage inputs is realistic on the FPAA. Modifying resistors is not directly done on the FPAA, but similar results can be achieved by biasing the FGs with different currents. Therefore, although the circuits appear different, they should have the same adjustment capabilities.

It is posited that the Bernoulli trial circuit from [10] as shown has a high variance similar to that in Figure 26. If the circuit in Figure 23 is unable to accommodate generating the range of variances shown, then a closed-loop form using FGs as resistors may be a viable solution.

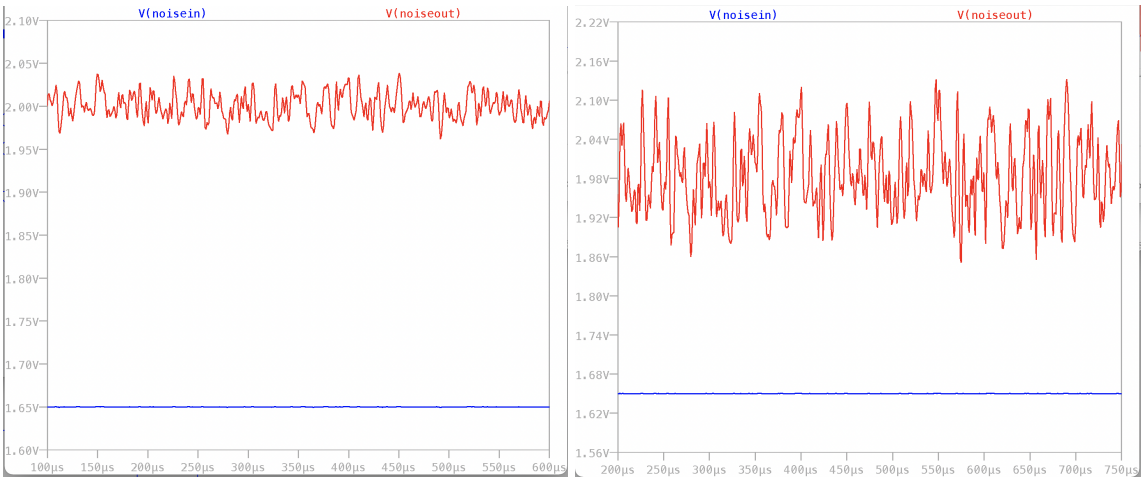


Figure 25: Noise output for gains A=10 and A=20. V_3 in the circuit is set to 1.682 and 1.666 respectively. V_1 and V_2 are both set to 1.65V.

Wijekoon Neuron

In [11], Wijekoon and Dudek present a silicon neuron with configurable behavior on a variety of levels. By tweaking just four voltage parameters on the neuron, an input signal on the order of 0.1μA produces different kinds of spike behavior: fast, regular, chattering, and intrinsic are some of the patterns identified. Spiking patterns are shown in their Figure 15 in [11].

An implementation of the silicon neuron for the FPAA is shown in Figure 27.

When a more sophisticated, biologically accurate, model of neurons is required for a prototyped project, this model is a good choice. The basic circuit contains 14 MOSFETs [11], so a single neuron would likely take up two CABs on an FPAA based on the components in Figure 8b, unless the op-amps can be utilized effectively or the routing fabric used as in [9]. Despite the possibility of taking up more space, this model is highly suited to the FPAA because of the FPAA's high configurability,

This neuron circuit, once combined with extra circuitry to set the four voltages and potentially accept a voltage input, requires the most elements out of any of

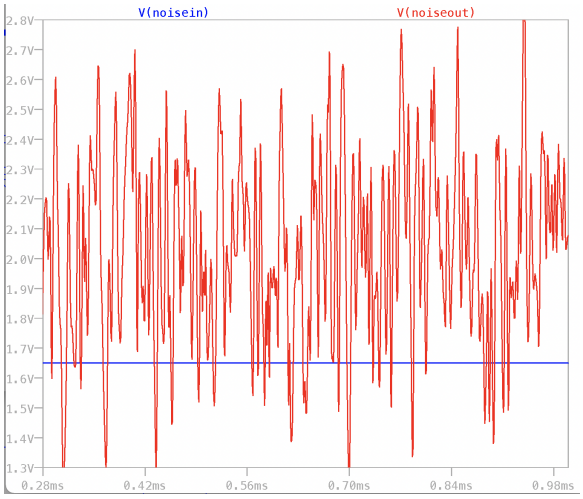


Figure 26: Noise output for A=50. V_3 in the circuit is set to 1.6575. V_1 and V_2 are both set to 1.65V.

the circuits outlined in this chapter. Experiments on increasing the sensitivity to inputs by adding noise to certain voltages would require additional overhead. This overhead opens the door to determining if enough power is saved to make up for the power required to harness noise.

Once able to load the Wijekoon neuron onto an FPAA, it should be relatively easy to design a system of neurons without worrying about how the individual parameters will affect system performance. The parameters used in a working neuron circuit are $V_c = 0$, $V_{th} = 0.2$, $V_{bias} = 2$, and $V_d = 2$. All of these are shown in Figure 27.

Any one of the noisy inputs from Figures 25 and 26 can be applied to V_d in the neuron. To tie this in to the discussion of SR, noise may enhance processing or not to different degrees in different locations. Figure 28 shows noise on the input, although in [1] a different location, V_d, was settled on for application of noise.

Corresponding to Figure 28, tests on an FPAA implementation produced the responses in Figure 29. Input comes in the form of a voltage drop at the external test input, which is shown in Figure 27. The input is attached to a pFET, which increases current when the gate voltage is lowered.

Supposing that V_d is required to be a constant value while the circuit is in operation, this means that a noise amplifier circuit like that in Figure 24 would not need to be modified once configured, whereas adding noise to the input requires some accommodations for another varying signal.

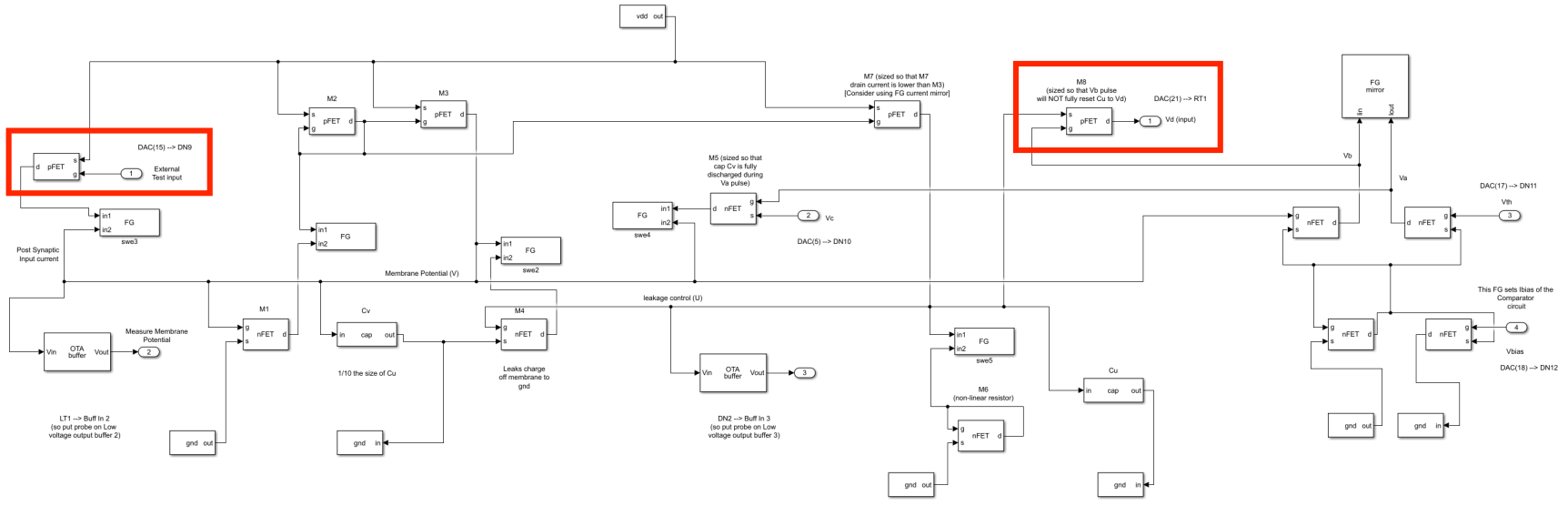


Figure 27: Wijekoon neuron from [12], modeled after Figure 1 in [11]. The boxed voltage input on the left is the input voltage and the boxed voltage input on the right is where noise would be applied to investigate stochastic resonance.

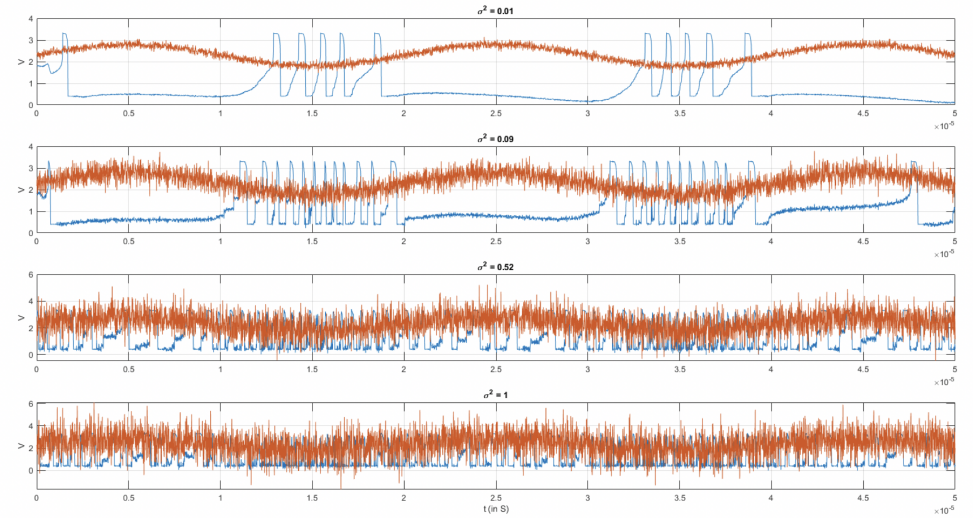
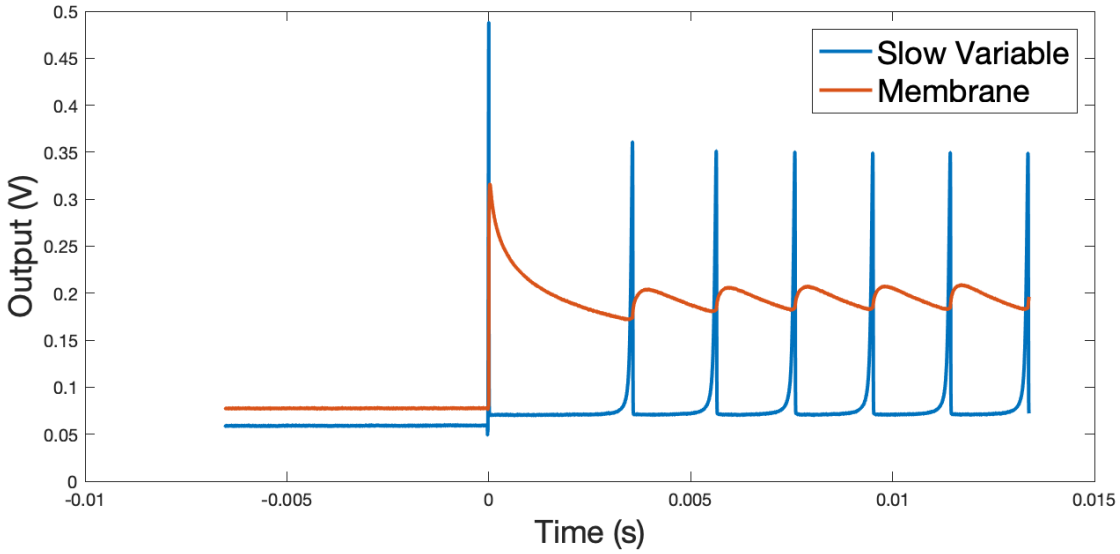
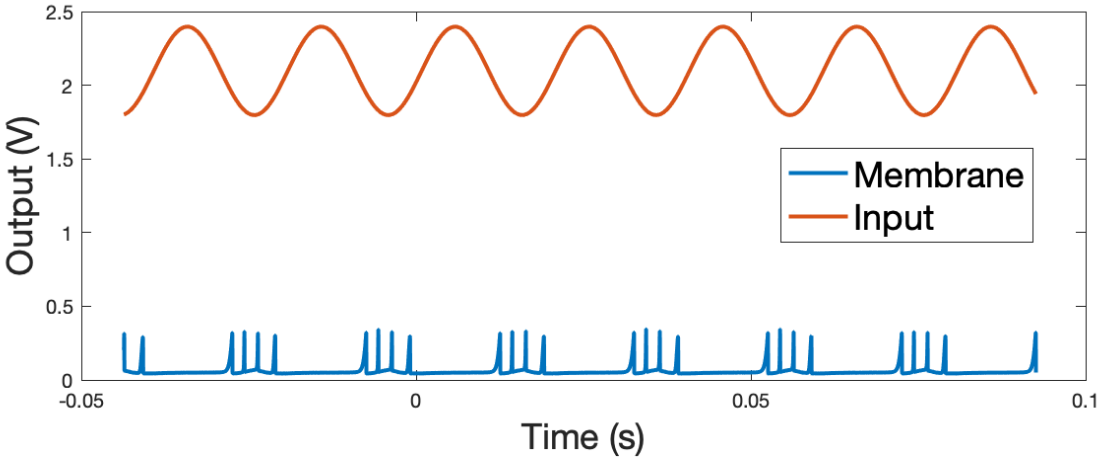


Figure 28: Baylor Neuromorphic & Robotic Systems Lab diagram showing response to noisy voltage inputs with different variances. Generated by the author of [1]



(a) Experimental results for a constant input generated by a step input dropping from 2.4V to 1.8V.



(b) Experimental results for a sinusoidal input between 2.4V and 1.8V. Similar to Figure 28, with no noise.

Figure 29: Hardware results from an FPAA: Data gathered from applying an external input to the neuron model in Figure 27, to the ”External Test Input” location.

CHAPTER FOUR

Conclusion

Summary

The intention of this thesis was to examine how the FPAA and associated design tools have application to the theory of stochastic resonance, which intersects with both neuromorphic and stochastic computing. Recent research suggests that some circuits, such as the Wijekoon neuron, have the capacity to perform better with varying voltage inputs. Building blocks for tying this research into that of harnessing noise inherent to transistors and capacitors was shown [10].

Several circuits were examined to evaluate the complexity of the task, and it turned out that small capacitors, open loop OTAs, and FGs are hard to simulate (see [20] for FG models), but that, as discussed in the section on the stochastic random number generator, there are simple circuits that with some characterization of components should meet the requirement of adjustable noise.

Discussion

Creating analog circuits requires some artistic talent and design frameworks like that for the FPAA should reduce overall user time required as well as the time between prototypes. It does however still require significant care to understand what the goal of the circuit is and how the FPAA architecture can actually be used to properly achieve that goal. The best components for simulation may not be the best for synthesizing the circuit. This is not a barrier to using the FPAA for research, and in fact forces the researcher to have a thorough understanding of the task at hand.

In this test of the theory of stochastic resonance, it turns out that useful amplifying of noise requires multiple OTAs, which adds transistors to analog circuits that utilize noise. This extra space is a small cost when compared to noise-generation approaches such as manually attaching a waveform generator to the circuit. It also has the benefit of being programmable on the FPAA, so in this context there is no need to decide ahead of time if a dedicated IC component for noise generation is required. In many neuromorphic systems, power constraints are vital, so there may be room to evaluate the possibility of using a single noise amplifier to serve multiple locations in a circuit.

BIBLIOGRAPHY

[1] J. Boline, "Stochastic computing & stochastic resonance in digital & analog neuromorphic systems." Master's thesis, Baylor University, 2020.

[2] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, Oct. 1990.

[3] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128x128 120 dB 15 us Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb. 2008.

[4] A. Alaghi and J. P. Hayes, "Survey of Stochastic Computing," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, pp. 1–19, 2013.

[5] M. D. McDonnell and D. Abbott, "What Is Stochastic Resonance? Definitions, Misconceptions, Debates, and Its Relevance to Biology," *PLOS Computational Biology*, vol. 5, no. 5, p. e1000348, May 2009.

[6] B. Holdsworth and C. Woods, *Digital Logic Design*. Oxford, UNITED KINGDOM: Elsevier Science & Technology, 2002. [Online]. Available: http://ebookcentral.proquest.com/lib/bayloru/detail.action?docID=294089

[7] M. Collins, J. Hasler, and S. George, "An Open-Source Tool Set Enabling Analog-Digital-Software Co-Design," *Journal of Low Power Electronics and Applications*, vol. 6, no. 1, p. 3, Feb. 2016. [Online]. Available: http://www.mdpi.com/2079-9268/6/1/3

[8] S. George, S. Kim, S. Shah, J. Hasler, M. Collins, F. Adil, R. Wunderlich, S. Nease, and S. Ramakrishnan, "A Programmable and Configurable Mixed-Mode FPAA SoC," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–9, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7374749/

[9] C. M. Twigg, J. D. Gray, and P. E. Hasler, "Programmable Floating Gate FPAA Switches Are Not Dead Weight," in *2007 IEEE International Symposium on Circuits and Systems*. New Orleans, LA, USA: IEEE, May 2007, pp. 169–172. [Online]. Available: http://ieeexplore.ieee.org/document/4252598/

[10] B. Marr and J. Hasler, "Compiling probabilistic, bio-inspired circuits on a field programmable analog array," *Frontiers in Neuroscience*, vol. 8, May 2014. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4019887/

[11] J. H. Wijekoon and P. Dudek, "Compact silicon neuron circuit with spiking and bursting behaviour," *Neural Networks*, vol. 21, no. 2-3, pp. 524–534, Mar. 2008. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0893608007002705

[12] S. Koziol, "Research notes," Sep. 2015.

[13] S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, R. Douglas, and R. Douglas, *Event-Based Neuromorphic Systems*. New York, UNITED KINGDOM: John Wiley & Sons, Incorporated, 2015. [Online]. Available: http://ebookcentral.proquest.com/lib/bayloru/detail.action?docID=1895762

[14] T. J. Hamilton, S. Afshar, A. van Schaik, and J. Tapson, "Stochastic Electronics: A Neuro-Inspired Design Paradigm for Integrated Circuits," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 843–859, May 2014.

[15] A. Alaghi and J. P. Hayes, "Computing with randomness," *IEEE Spectrum*, vol. 55, no. 3, pp. 46–51, Mar. 2018.

[16] A. Basu and P. E. Hasler, "A Fully Integrated Architecture for Fast and Accurate Programming of Floating Gates Over Six Decades of Current," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 6, pp. 953–962, Jun. 2011.

[17] W. H. Wolf and S. Member, "Hardware-Software Co-design of Embedded Systems," *IEEE Proceedings*, pp. 965–989, 1994.

[18] C. R. Schlottmann, C. Petre, and P. E. Hasler, "A High-Level Simulink-Based Tool for FPAA Configuration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 10–18, Jan. 2012. [Online]. Available: http://ieeexplore.ieee.org/document/5671528/

[19] S. Koziol, C. Schlottmann, A. Basu, S. Brink, C. Petre, B. Degnan, S. Ramakrishnan, P. Hasler, and A. Balavoine, "Hardware and software infrastructure for a family of floating-gate based FPAAs," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 2794–2797, iSSN: 2158-1525.

[20] J. Gray, R. Robucci, and P. Hasler, "The design and simulation model of an analog floating-gate computational element for use in large-scale analog reconfigurable systems," in *2008 51st Midwest Symposium on Circuits and Systems*, Aug. 2008, pp. 253–256, iSSN: 1558-3899.