AFRL-RI-RS-TR-2019-174

DESIGN OF A MEMRISTIVE DYNAMIC ADAPTIVE NEURAL NETWORK ARRAY (MRDANNA)

UNIVERSITY OF TENNESSEE, KNOXVILLE

AUGUST 2019

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2019-174 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
JOSEPH E. VAN NOSTRAND
Work Unit Manager

/ S /
LISA WEYNA
Acting Technical Advisor, Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
AUGUST 2019	FINAL TECHNICAL REPORT	DEC 2015 – FEB 2019

4. TITLE AND SUBTITLE

DESIGN OF A MEMRISTIVE DYNAMIC ADAPTIVE NEURAL NETWORK ARRAY (MRDANNA)

5a. CONTRACT NUMBER
N/A

5b. GRANT NUMBER
FA8750-16-1-0065

5c. PROGRAM ELEMENT NUMBER
61102F

6. AUTHOR(S)

Garrett S. Rose, Mark Dean, and James Plank

5d. PROJECT NUMBER
T2NR

5e. TASK NUMBER
ND

5f. WORK UNIT NUMBER
ES

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Tennessee, Knoxville
1534 White Ave
Knoxville, TN 37996-1529

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITB
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2019-174

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The objective of this effort was to build affordable, manufacturable, low power, dynamic neuromorphic computing platform for handling spiky, highly variable information/data, as well as develop a low-power, hybrid memristor/CMOS "neuron / synapse" implementation for implementation of a hardware-based dynamic neural network array. The approach was to leverage hybrid CMOS/memristor encryption project to design, fabricate, and test memristor/CMOS "neurons / synapses," and integrate them with existing FPGA implementations for full-scale dynamic neural network array demonstration. In addition, to make a module design kit library available for other circuits and architectures, thus making advancements in this effort useful to future designs that utilize hybrid CMOS/memristor technologies.

15. SUBJECT TERMS

Memristor, FPGA, CMOS, Nanoelectronics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**JOSEPH E. VAN NOSTRAND**
U	U	U	UU	51	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

Table of Contents

List of Figures

Summary

This final technical report summarizes the R&D efforts for the AFRL project "Design of a memristive dynamic neural network array," referred to here as "mrDANNA," which occurred from December 2015 through February 2019. This research project enables future generations of computing systems by (1) leveraging an emerging, power-efficient device technology (i.e. the memristor) and (2) considering an alternative architectural model (i.e. neuromorphic) that promises to overcome many of the performance limitations of conventional von Neumann systems. The specific neuromorphic architecture on which the proposed mrDANNA is based is the Neuroscience-Inspired Dynamic Architecture (NIDA), developed at the University of Tennessee, Knoxville (UTK) as an approach to applying neuromorphic principles to a wide variety of applications. Key features of the NIDA architecture include: (1) a spiky representation of data, (2) the ability for the system to adapt during run-time, and (3) a synaptic representation including delay distance as well as weight information. The structure and simplicity of the NIDA architectural model has been leveraged in the development of a second neuromorphic model, more specifically tailored to digital implementation, called Dynamic Adaptive Neural Network Array (DANNA). DANNA's key feature is a basic neuromorphic element that can be configured to represent either a neuron or a synapse, laid out in a two-dimensional array.

For this project we have leveraged the features of the NIDA architecture in the design of a mixed-mode, analog/digital memristor-based DANNA (mrDANNA) system. The mrDANNA system maintains several important features of the digital representation of DANNA, including real-time dynamic adaptability and synaptic functionality that includes both weight and delay distance information. There are three specific components of the mrDANNA design effort: (1) FPGA prototyping of digital DANNA and emulated mrDANNA architectures, (2) design and verification of mrDANNA test chip to be fabricated in collaboration with AFRL and SUNY Polytechnic Institute (SUNY Poly), and (3) the development and demonstration of the supporting software required to utilize DANNA and mrDANNA systems for various applications.

Introduction

The primary components of the NIDA architecture and corresponding DANNA structure are programmable elements that can be configured for neuronal and synaptic functionality, and be replicated across an “n x m” grid with nearest neighbor or “small world” connectivity. To maximize performance and minimize size and power consumption very simple logic circuits are used, avoiding the use of digital signal processors, floating-point units, arithmetic-logic units, memory arrays and other common microprocessor units.

The neuron’s function is to accumulate “charge” by adding the weighted inputs to an existing charge level until that level reaches a programmable threshold. Upon reaching this threshold the neuron fires its outputs and resets its charge to a predetermined bias level. A weighted-sum threshold activation function is the primary neuron type used due to its simplicity and functionality. However, other activation functions can be implemented (such as linear, sigmoidal or Gaussian).

The synapse function captures selected features of both axons and synapses found in biological neural networks. Specifically, a NIDA/DANNA synapse contains both the associative delay distance between two neurons and the weight (or strength) of the synaptic connection. Two unique characteristics of the synapse model are: 1) the weight value held by the synapse can automatically potentiate (long-term potentiation, LTP) or depress (long-term depression, LTD) depending on the firing condition of its output neuron and 2) a string of firing events can be stored in a “distance FIFO” to simulate a synapse transmitting a set of firing events down its length (thus constituting temporal storage of events).

An evolutionary optimization (EO) environment has been developed at UTK to configure the neural networks in a DANNA. The EO trains over parameters of the network (weights and delay distances on synapses and thresholds on neurons) as well as the structure (the number and placement of neurons and synapses) and the dynamics of the network. The dynamics of the network are directly embedded in the structure itself (delays in the synapse and charges in the neuron). Most other artificial neural network implementations have a predefined, fixed structure rather than one determined by a dynamic optimization method as in our approach. The EO requires the user to specify the inputs and outputs to the network, and a fitness function that rates how well a network's outputs fare with the given inputs. The EO then generates an initial population of random networks, and gradually evolves the population using mutation and crossover operations, until it generates a network that exceeds a predefined threshold for fitness. These operations occur on the direct representation of the network.

An initial prototype, including global functions and the programming interfaces, was implemented and tested using two different Xilinx FPGAs: the XC7VX690T and the XC7VX485T. A square grid of approximately 2500 elements was placed on a 485T FPGA and approximately 5000 elements on a 690T FPGA. With these sizes of neural network arrays, applications such as handwritten character recognition, image pattern recognition, network traffic analysis and small systems controls automation are supported.

The mrDANNA (memristive DANNA) implementation constructed with nanoscale memristors extends the number of elements placed on one chip and thus also extends the application possibilities. Specifically, mrDANNA enables improvements in area utilization as compared to an all CMOS DANNA implementation. This increase in the number of available

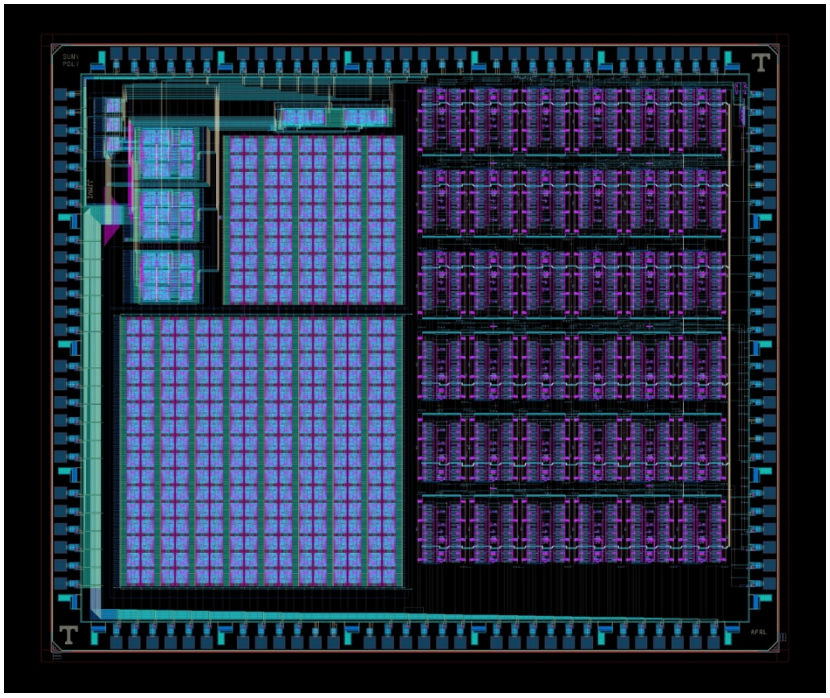

Figure 1. *Final layout of mrDANNA/DANNA test IC. The left side of the chip includes test structures for the digital DANNA system. The right side includes test structures for mrDANNA, including 36 cores that can be "mask programmed" via upper metal.*

elements in the system translates into an increase in the complexity of applications that can be implemented with a single mrDANNA chip.

To implement the mrDANNA test chip, this project leveraged the hybrid CMOS/memristor process developed at SUNY Polytechnic Institute. The process integrates metal-oxide (HfO_2) memristors in the metal layers of the 65 nm 10LPe CMOS process, leading to a seamless CMOS/memristor integration process. The seamless integration of CMOS with memristive technology is a unique feature as compared to most related efforts where memristive devices are integrated post-fabrication on an existing CMOS chip. Further, this project is the first example of a CMOS/memristor design integrated into a process consisting of multiple CMOS options (PMOS and NMOS) and a full metal stack with multiple metal layers (6 total in the case of this process).

The following objectives were outlined for this effort, combining work at both SUNY Poly and UTK. This report specifically details the effort at UTK, which is primarily focused on the design of the mrDANNA test chip, supporting software, and the DANNA-based FPGA prototype. We have also worked with collaborators at SUNY Poly and AFRL on the final physical design (layout) of the mrDANNA chip, used in conjunction with the existing DANNA (FPGA implementation) at UTK. Below are the specific objectives, as outlined in the original research proposal:

- *CMOS/Memristor Design of mrDANNA*
  The primary goal of this research is to demonstrate the feasibility of hybrid CMOS/memristive circuits in the implementation of neuromorphic computing systems. We have specifically designed a CMOS/memristor implementation of the dynamic neural network array (DANNA) architecture that is well suited to spatio-temporal applications. For this project we have designed and verified a memristive DANNA (mrDANNA) system, demonstrating via simulation the projected performance for full scale implementation. Additionally, a scaled version of mrDANNA and the related components (neurons and synapses) were taped-out and fabricated using the CMOS/memristor process developed by collaborators at SUNY Poly. More specifically, the SUNY Poly process offers the ability to integrate HfO_2 memristors with 65 nm CMOS (IBM 10LPe technology node). The end result (shown in Fig. 1) is a physical test chip demonstrating the potential of such CMOS/memristor neuromorphic structures.

- *Digital Design and FPGA Prototyping of DANNA*
  In order to fully demonstrate the advantages of DANNA/mrDANNA neuromorphic systems, we have prototyped DANNA system designs on Xilinx FPGAs. The DANNA prototypes have been optimized for digital implementations, including on FPGAs and as digital ASICs, providing a robust representation of NIDA-inspired neuromorphic systems. In addition to the FPGA-optimized DANNA, we have also developed a DANNA2 system that more closely adheres to the NIDA model and is behaviorally more similar to the mrDANNA system. Development of the FPGA prototype was also included as part of our risk management plan as it adds a second hardware demonstration vehicle for the architecture considered.

- *Development and Demonstration of Supporting Software*
  In order to effectively utilize the DANNA and mrDANNA systems, several software tools were developed for generating application-specific networks, configuring the DANNA or mrDANNA hardware and interfacing with the hardware once configured. Specific tools developed as part of this project include (1) robust simulators (high-level system models) for both DANNA and mrDANNA used for verification and offline training, (2) visualization tools that help further improve verification efforts, and (3) optimization engines used for generating new networks.

Methods, Assumptions and Procedures

HfO_2 Memristor Model for SPICE-Level Simulation

Verilog-A models have been developed in-house at UTK to accurately represent the HfO_2 devices fabricated at SUNY Poly. More recent modifications to our memristor model include (1) parameter adjustment for fitting to more recent device characterization results, (2) improved convergence for faster circuit simulation, and (3) an improved representation that accounts for non-linearities observed in memristor device behavior. Results for the updated model have also been presented at the IEEE International Symposium on Circuits and Systems (ISCAS) and at the IEEE Midwest Symposium on Circuits and Systems (MWSCAS). The inclusion of a quadratic window function was found to adequately represent device behavior while maintaining reasonable circuit simulation performance.

We take an approach which uses the instantaneous resistance as the state variable. While this approach is not purely physics-based, it is well suited for circuit designers looking for a "plug-and-play" model that is (1) simple to understand, (2) designed for direct parameter extraction from experimental data, and (3) converges well in circuit simulation.

$$\frac{dM}{dt} = \begin{cases} -C_{LRS}(\frac{V(t)-V_{tp}}{V_{tp}})^{P_{LRS}} f_{LRS}(M(t)), & V(t) > V_{tp} \\ C_{HRS}(\frac{V(t)-V_{tn}}{V_{tn}})^{P_{HRS}} f_{HRS}(M(t)), & V(t) < V_{tn} \\ 0, & \text{otherwise}, \end{cases} \tag{1}$$

Here C_{LRS} and C_{HRS} are two scaling parameters. P_{LRS} and P_{HRS} are non-linearity coefficients. f_{HRS} and f_{LRS} capture the resistance saturation (commonly referred to as window functions). Equation (2) presents the proposed window function that can be easily fitted to measurable parameters.

$$f(M(t)) = \begin{cases} \frac{1}{1+e^{\frac{M(t)-\theta_{HRS}HRS}{\beta_{HRS}\Delta r}}}, V(t) < V_{tn} \\ \frac{1}{1+e^{\frac{\theta_{LRS}LRS-M(t)}{\beta_{LRS}\Delta r}}}, V(t) > V_{tp} \end{cases} \tag{2}$$

In (2), M(t) is clipped to either LRS or HRS once the resistance hits either boundary to ensure it does not exceed the measurable range of resistance such that:

$$\begin{aligned} If\ (M(t) > HRS),\ then\ M(t) = HRS, \\ If\ (M(t) < LRS),\ then\ M(t) = LRS, \end{aligned} \tag{3}$$

where (3) and (4) are implemented as the device is switching from LRS to HRS and HRS to LRS, respectively. In fact, the user can opt to activate just the clipping function without applying the window function in (2) using window selectors as shown in the following section.

Appendix I of this report provides the Verilog-A code for this model.

Fig. 2 shows results for the updated model fit to recent experimental results from SUNY Poly. Further results illustrating the forming mechanism as implemented using the Verilog-A model are shown in Fig. 3. As can be seen from both figures, the model accurately reflects observed DC and transient behavior. The inclusion of accurate forming characteristics, again based on experimental observations, have also been key to verifying the forming and programming circuits included in the mrDANNA test IC.

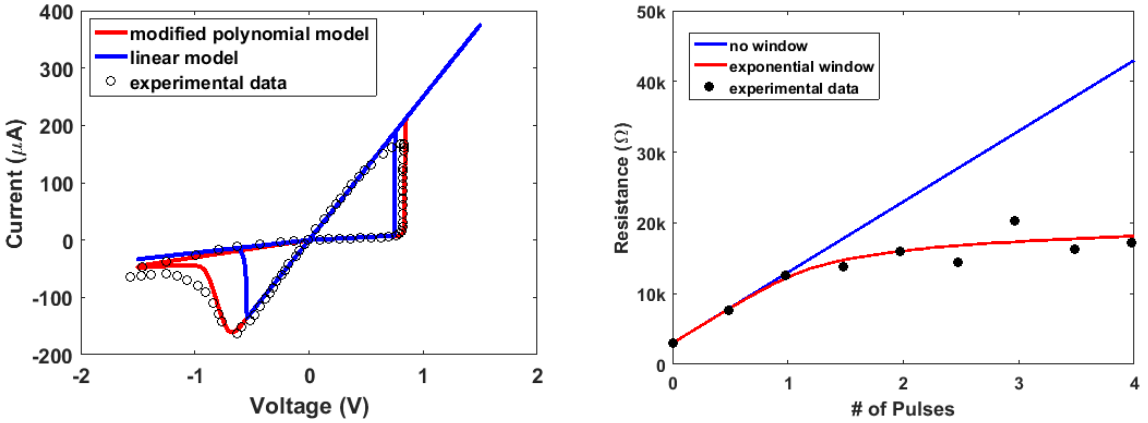

Figure 2*. Improved memristor model I-V curve (left) and corresponding experimental I-V response. Resistance vs. the applied pulses or time of applied bias (right) for both the model and measured results.*

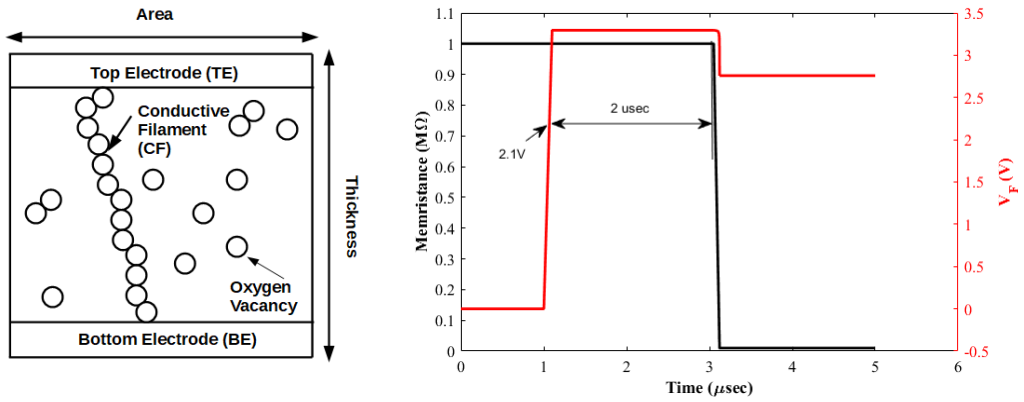

Figure 3. *Illustration of filament formation mechanism (left) and relationship between formation, voltage amplitude and duration for the model developed (right).*

Testing at UTK of wafers obtained from SUNY Poly for our mrDANNA test IC focused on repeating device level tests of the HfO2 based memristors when implemented as part of 1-transistor, 1-memristor (1T1R) ReRAM test cells. In these basic ReRAM structures, a high-voltage n-type DGXFET is used as a selector or, in the case of testing, as a current compliance regulator.

Fig. 4 shows an I-V curve including memristor forming, set and reset behavior obtained at UTK using our probe station and characterization system. It is clear that our result is well aligned with that of our SUNY Poly collaborators and that the device behavior is consistent with the models developed for circuit simulation.

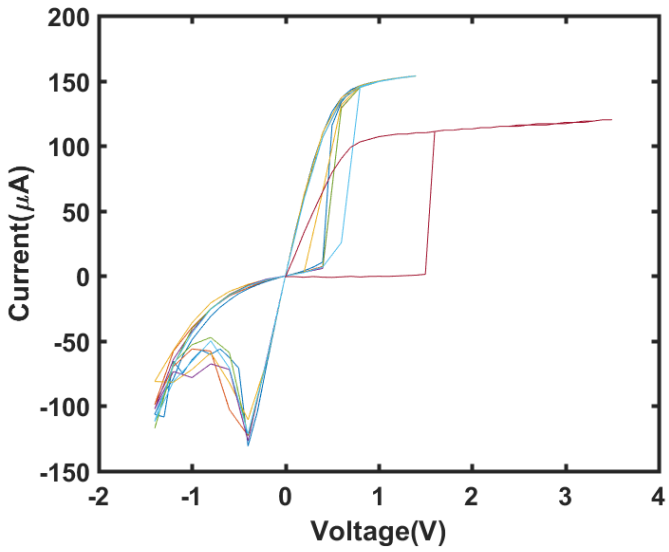

Figure 4. *Experimental results for current-voltage response of an HfO2 memristor tested as part of a 1T1R ReRAM test structure. These results were obtained at UTK and are consistent with model expectations and results obtained by SUNY Poly collaborators.*

mrDANNA Circuit Design and Verification

We have designed, verified and implemented CMOS/memristor circuits for synaptic buffers (used to manage online learning) and mixed-signal neurons. These components have also been integrated into the mrDANNA core design with several cores integrated into the mrDANNA test chip (Fig. 5). More specifically, the mrDANNA test chip includes 36 mrDANNA cores, each comprised of a single neuron and six synapses for our first generation system. While small, the array is sufficient for running small networks capable of executing applications such as controling small autonomous vehicles. The test chip has been fabricated by our collaborators at SUNY-Poly in Albany, NY.

It is worth noting that a follow-on project enables the inclusion of more metal layers in the IC and will allow more rigorous testing of different synapse-neuron combinations, helping us determine the most efficient combination for mrDANNA systems. As part of this follow-on work, the upper metal layers can be redesigned such that all 36 mrDANNA cores are wired for the best configuration. This “sea of gates” approach will allow for the reuse of lower, and much more expensive, design masks in future implementations.

As alluded to, there are two basic design components that must be optimized and fully verified in order to implement a reliable mixed-signal CMOS/memristor neuromorphic system:

the synapse circuit and the neuron circuit. Other circuits that must be included are decoders for programming the mrDANNA system (Fig. 5), a reconfigurable interconnection fabric for defining specific neural networks, and the requisite I/O circuitry. For the fabricated test structures, we have focused primarily on the design and optimization of the neurons and synapses.

Validation Testing & Physical Design

We have completed rigorous validation testing on all circuit schematics and layouts for the components implemented in the mrDANNA test chip. This includes defining (1) clear behavioral validation tests as well as (2) performance targets and simulation results. All simulations are being performed on both schematic views and extracted layouts for each component. Circuit simulations were all performed using existing models for SUNY Poly's HfO_2 memristors and the 65 nm CMOS process design kit.

In addition, we have completed design rule check (DRC) and layout versus schematic (LVS) verification of all layouts included in the mrDANNA system design.

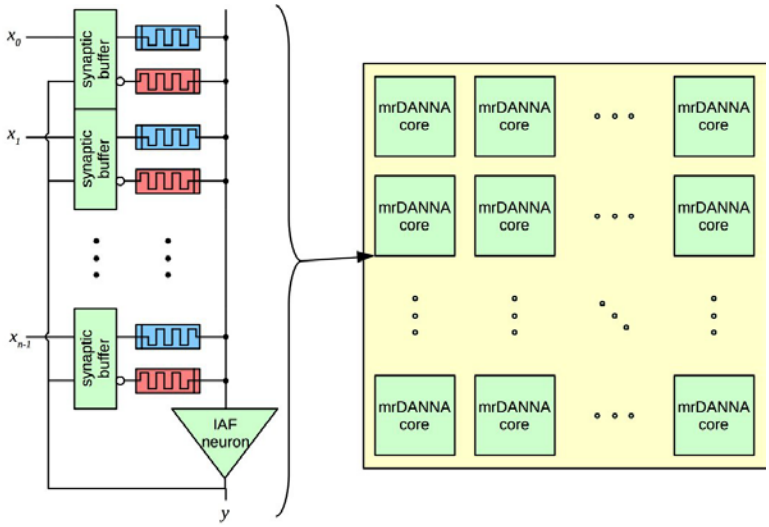

Figure 5. *Illustration of the memristive DANNA (mrDANNA) system designed for this project.*

Synaptic Buffer Design

Three flavors of synaptic buffer are included in the final mrDANNA test IC. First, a "twin memristor synapse" uses two memristors to represent synaptic weights (either positive or negative). For the twin memristor synapse, the switching rates for the underlying memristors are assumed to be symmetric.

There are actually two versions of the twin memristor synapse: (1) symmetric switching twin memristor and (2) asymmetric switching twin memristor. The symmetric switching synapse assumes the memristor switching times for LRS to HRS transitions are approximately identical to switching times for HRS to LRS transitions. Such behavior at the device level drastically reduces the circuit and power overhead of the online learning system. However, these switching times are not identical for most HfO_2 memristive devices explored thus far due to physical asymmetries in

the material stack itself. Since the test IC is an experimental platform, we included the symmetric switching synapse in the event new material stacks are eventually available with the desired symmetric switching behavior. The asymmetric switching twin memristor synapse is designed specifically for online training and does compensate for asymmetric switching at the device level.

The third synapse flavor considered is a "quad memristor synapse" where four memristors are used to represent a single weight (either positive or negative). Similar principles apply in terms of representing weight by an effective conductance where current flows either into or out of the synapse. However, the quad memristor synapse forces all memristance changes during online learning to occur in one direction only. The reason for this is that some memristors have exhibited asymmetric switching rates where switching is faster for switching in one direction relative to the other (e.g. LRS to HRS). In our test circuit, the quad memristor synapse provides an added option for synaptic behavior and online learning.

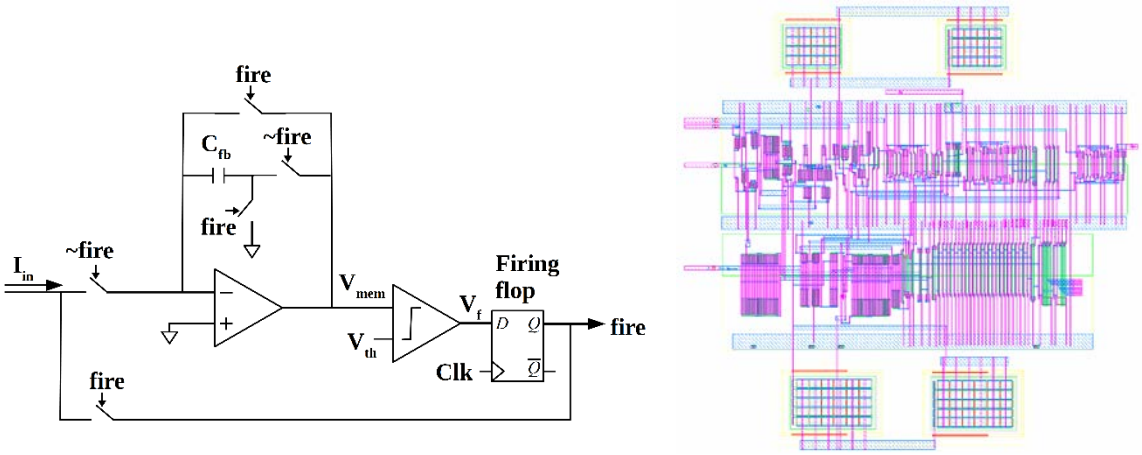

Figure 6. *Leaky integrate and fire neuron schematic (left) and DRC/LVS clean layout view (right) designed with a digital buffering stage for simplicity.*

Neuron Design

Two primary categories of analog neuron are included in the mrDANNA test chip. The first is a classic leaky integrate and fire neuron (LIF) (Fig. 6) consisting of an analog op-amp based integrator followed by an op-amp based comparator. The comparator in this case uses a reference voltage to implement the neuron threshold. Energy consumption for the LIF neuron using the 65 nm CMOS process is estimated to be ~ 9 pJ/Spike.

The second neuron class included in the mrDANNA test circuit is an axon hillock (AH) neuron (Fig. 7). This is a simpler design that relies on a charge pump for integrating/accumulating charge from the synaptic inputs followed by a basic comparator for the thresholding operation. The comparator in this case could be implemented using an op-amp or alternatively a digital buffer (illustrated in Fig. 5). All options are included on the test chip to better understand their behavior when integrated with memristors. The AH neuron can be tailored for different energy-delay requirements with energy consumption thus far found to be as low as 100 fJ/Spike.

The third neuron included in our mrDANNA test IC is a digital neuron, implemented as a simple ADC that converts currents from the memristive synapse array into a digital value added to a neuron register. This digital neuron has been included primarily as another backup to provide a degree of risk management. It is also expected that the digital neuron is more easily tuned for a variety of potential memristive device behaviors (LRS, HRS, thresholds, etc.), thus allowing future exploration of a full mrDANNA system even constructed from memristor stacks other than HfO_2. The schematic and layout of the digital neuron is shown in Fig. 8.

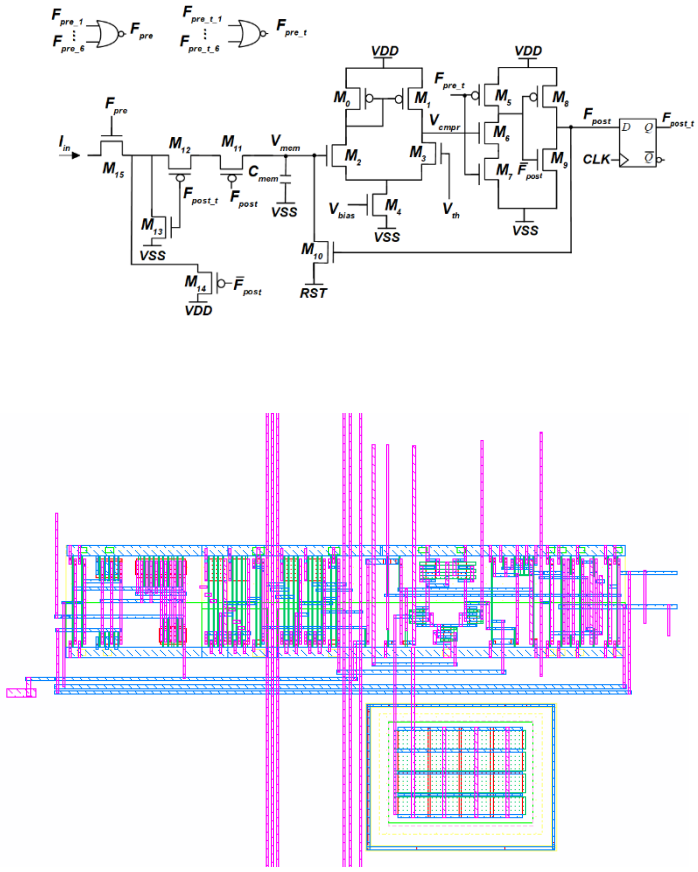

Figure 7. *Axon hillock neuron schematic (top) and associated DRC clean layout view (bottom) designed with a domino buffering stage.*

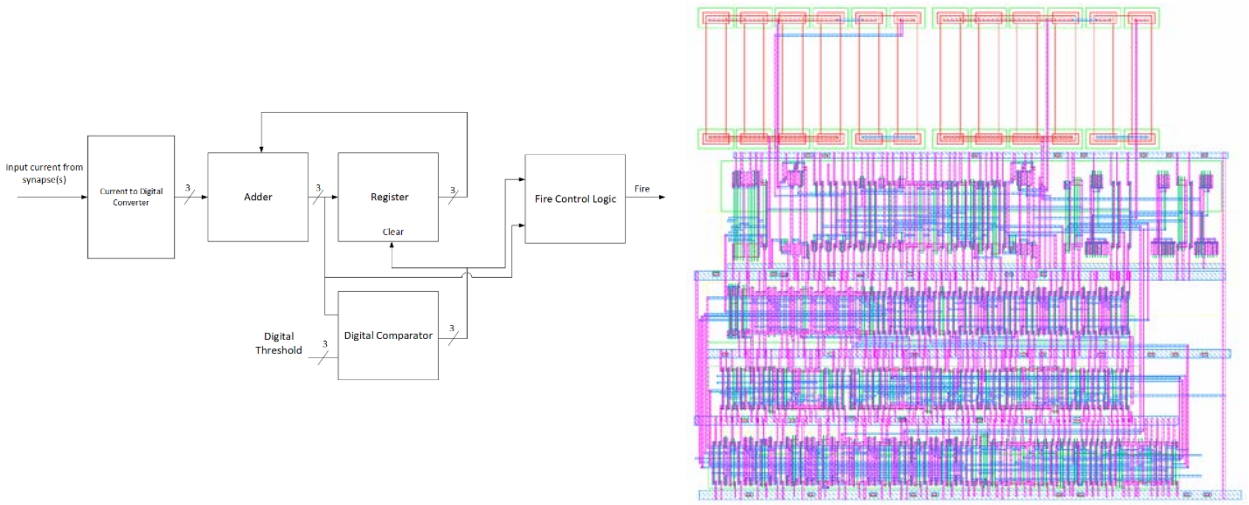

Figure 8. *Digital neuron schematic (left) and associated DRC clean layout view (right).*

Scan Chain based Programming

The full mrDANNA array has been designed with a novel scan chain circuit (Fig. 9) used to shift in both binary configuration bits for defining network topologies and shifting in initial memristor weight values (determined from EO-based offline training). The system works by first assuming that all initial synaptic weights will be configured to extreme values, either -*N*, 0 or *N* where the range [-*N*, *N*] is the range for all possible weights used in the system. These extreme weights (-*N* and *N*) are accomplished in the twin memristor synapse circuit by forcing one memristor in the pair to LRS and the other to HRS. If the memristor driving positive current (R_p) is at LRS while the negative current memristor (R_n) is at HRS, then the total weight value is *N*. Likewise, when R_p = HRS and R_n = LRS, the synaptic weight value is -*N*. Combinations that provide for equal resistance in both memristors ($R_p = R_n$) leads to a weight value of 0.

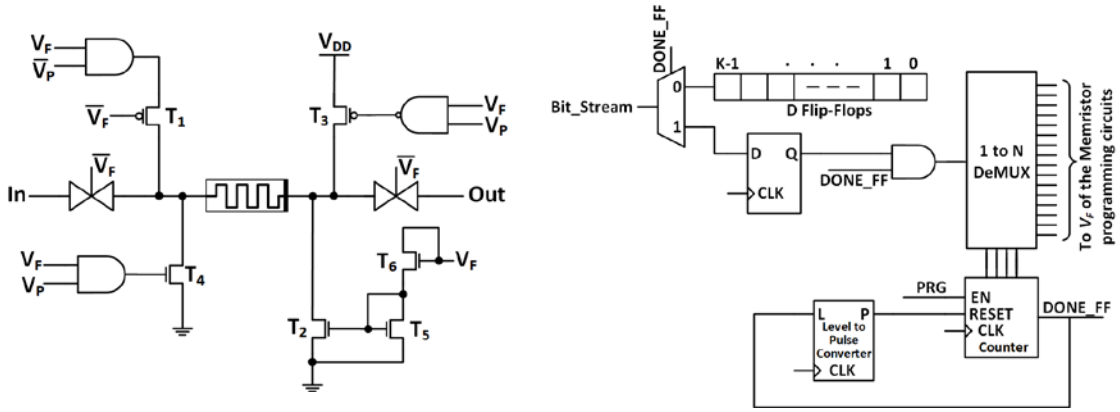

Figure 9. *Forming/programming circuit (left) and full scan chain control schematic (right).*

The rationale for programing initial weight values to their extreme values is two-fold. First, programming a memristive device to either HRS or LRS is relatively easy (requiring little overhead) as compared to programming to “in-between” resistance states. Second, this limited

resolution is expected to ease the process of off-line training, i.e. network initialization, such that a neural network solution for a given application can be determined in relatively few "epochs" of the evolutionary optimization process. Here, we rely on online learning, via an unsupervised STDP-based mechanism controlled by the synaptic buffer, to optimize the network during inference/operation. We find that unsupervised STDP learning in the online environment can effectively improve the network accuracy with time by nudging memristive weights to intermediate states between the extreme -*N* and *N* cases. This effectively means we program with low resolution (via our scan chain) and refine to high resolution using online learning.

Formation/Programming Control Circuit

The HfO_2 memristors leveraged for mrDANNA require a relatively high voltage forming step (~3V) in order to form or initialize the filament used to switch the device. Put another way, a memristor will initially act as a high impedance resistor until the 3V forming pulse is applied, thus causing the device to behave as a memristor from that point forward. Forming only occurs once but is required for the type of memristors currently being considered in this project.

For this one-time forming step, we have designed, verified and integrated a formation/programming circuit (left side of Fig. 9) constructed with high voltage FETs in the 65nm process. Since these transistors need to drive large currents and must withstand large voltages, we use DGXFETs with gate lengths of 500nm – much larger than the normal 65nm variety transistor available in this technology kit. Eliminating forming, if possible, would drastically reduce the overall size and power consumption of the system. Such a device level goal could be worthwhile in future, related projects.

Results of our scan-in system for programming initial network configurations into the mrDANNA array were presented in a May 2018 publication for the IEEE International Symposium on Circuits and Systems (ISCAS). We have also submitted an invention disclosure and pursuing a patent for this scan-in mechanism.

FPGA Prototyping of DANNA – DANNA 2 Design

We have updated the digital DANNA (Dynamic Adaptive Neural Network Array) system to more closely match the NIDA architecture requirements and mrDANNA functionality. Since this is the second major architectural update of DANNA, we refer to this system as "DANNA 2."

Following from NIDA, there was a desire and need to develop an architecture which may be implemented as a programmable hardware array on an FPGA or as a VLSI integrated circuit. This led to the creation of DANNA. DANNA uses a two dimensional NxN grid of elements with element connectivity constrained to a 16 nearest neighbor scheme. Each DANNA element may be a neuron or a synapse. Neurons use an accumulate and fire model with 8-bit integer thresholds, and synapses use 8-bit integer weights and up to 256 cycles of configurable delay. Synapses also have long term potentiation and depression (LTP/LTD) to allow for online learning and

adaptability. LTP allows for a synaptic weight to be increased if the pre-synaptic neuron fires immediately prior to a post-synaptic neuron firing. Similarly, LTD allows for a synaptic weight to be decreased if the pre-synaptic neuron fires immediately after a post-synaptic neuron firing. DANNA is now a mature platform complete with an event based simulator, multiple FPGA deployments, and a VLSI implementation.

Training spiking neural networks for solving a particular task poses significant challenges. There are some results with modifications of backpropagation through time algorithms which attempt to deal with the nondifferentiable nature of the activation function, but these approaches are difficult and cumbersome to apply to all SNN models.

With NIDA and DANNA, the use of genetic algorithms offers one solution. We use a set of genetic algorithms which we call Evolutionary Optimization (EO). With EO, training occurs by initializing a set of random networks called a population. Each network in the population is evaluated with an application defined fitness function which specifies success for a task. Based on the fitness of each network, the networks can be ranked, and the best networks can be preferentially chosen through tournament selection. Then, a new population can be formed by applying mutations and crossover operations to prior networks. The evaluation of each population constitutes a training epoch. This training process is iterative and continues until either an epoch limit is reached or a network achieves the target fitness value.

DANNA2 follows a simple spiking neural network model using neurons and synapses as the network primitives. The model is designed to work well as a digital hardware design or as an event based software simulation.

Neurons

Neurons follow a discretized accumulate and fire model in which each neuron accumulates charge until a defined activation threshold. Neurons acquire charge based upon incoming spikes and the synaptic weights associated with each spike. Once the activation threshold is reached, the neuron emits a spike to each of its down-stream synapses which then will travel to other neurons. Immediately following a fire event, the neuron enters a refractory period in which it may not fire again for some configurable time.

DANNA2's neurons represent activation thresholds as 10 bit unsigned integers up to 1023, and refractory periods may be up to 7 network cycles after the neuron's last fire. DANNA2 also optionally supports linear charge leak in which the neuron's charge may decay back to a resting state (i.e, value of 0) at a programmable linear rate.

The neuron charge function used for DANNA may be updated to take into account the availability of a linear leak, and thus, the function may be expressed as:

$$H_{kj}(t) = \sum_{i=1}^{N} w_i(t)x_i(t) + H_{kj}(t-1) - L_{kj}(t-1) \quad (4)$$

The neuron activation function is unchanged from DANNA and may be expressed as:

$$a_{kj}(t) = f(H_{kj}(t)) = \begin{cases} 1 \text{ if } H_{kj}(t) \geq \theta(t) \\ 0 \text{ if } H_{kj}(t) < \theta(t) \end{cases} \quad (5)$$

Synapses

Synapses serve as a mechanism to connect neurons. Each synapse transmits spikes from a pre-synaptic neuron to a post-synaptic neuron. These spikes are sent with a configurable weight value which corresponds with the amount of charge the post-synaptic neuron should accumulate. There is also a configurable delay between the synapse receiving a spike from the pre-synaptic neuron and when the post-synaptic neuron will receive the spike from the synapse.

DANNA2's synapses use signed 9-bit integer weight values from -256 to 255 and may have a delay up to 15 additional network cycles. Each neuron is allowed up to 24 incoming synapse connections. Synapses may also leverage a simple Spike Timing Dependent Plasticity (STDP) model in which weight values are dynamically adjusted during runtime as a method of online learning.

The DANNA2 STDP model is a discretized lookup table keyed on the time difference between a synapse fire and a post-synaptic neuron fire.

$$\Delta t = t_{\text{synapse\_fire}} - t_{\text{neuron\_fire}} \quad (6)$$

This time value is used to look up a configurable synaptic weight adjustment. Potentiation values are located from $\Delta t = -N + 1$ to $\Delta t = 0$ in the table, and depression values are located from $\Delta t = 1$ to $\Delta t = N$ where 2N is the temporal window size for STDP. When $\Delta t = 0$ for DANNA2, this means the synapse fire occurred the same cycle that the neuron hit its threshold.

The approach allows for different STDP curves to be implemented, and DANNA2 can be optionally configured to use the same value for each potentiation and depression slot which then works similar to DANNA's LTP/LTD mechanism.

Networks

A network is a collection of neurons and synapses which works together to accept input spikes and generate output spikes. DANNA2 networks are typically sparse directed graphs in which the graph nodes are neurons and the edges are synapses. Networks are allowed and often will have recurrent cyclic patterns.

Each network has a maximum width and height dimension M×N, and each neuron must uniquely exist at a single coordinate pair within this specified grid. DANNA2 typically allocates dimensions with a height to width ratio of 2 to 1.

Input/Output

Inputs to the network are fire events with scaled weight values from 0 to 255 corresponding the positive weight range of a single synapse. All of a network's inputs are located on the left side of the network. Outputs from the network are valueless spikes. All of a network's outputs are located on the right side of the network.

DANNA2 Core Architecture

A DANNA2 element is a tightly coupled collection of 24 synapses with one post-synaptic neuron. The digital implementation of this is shown in Fig. 19. A collection of 24 distance registers collects input fires for each respective synapse. During each element cycle, three synapses are sampled by reading from the corresponding distance registers and loading the appropriate synapse parameters from the synapse table. The synapse units are responsible for sending the correct weight value to the accumulator if the synapse is firing, and it also handles the calculation of spike timing dependent plasticity weight adjustments. It is also important to note that one DANNA element may be either a neuron or a synapse while a single DANNA2 element is a neuron with up to 24 synapses.

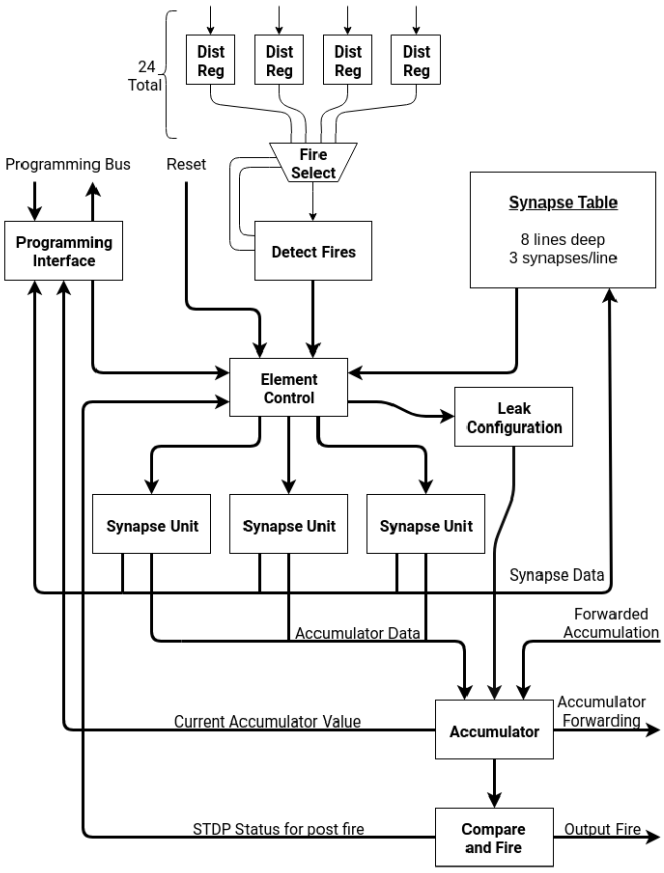

Figure 10. DANNA2 neuron core architecture.

The hardware implementation of DANNA2 uses element cycles and network cycles. An element cycle is a single clock cycle in which the element performs a portion of work. Each element requires 10 total element cycles to process all 24 synapses, fully update the accumulator, and register if there is a fire. These 10 element cycles form a single network cycle. In contrast, DANNA's element accumulator clock is 32 times faster than the network cycle clock. We expect the effective network speed of DANNA2 to be $5 - 10$ times faster than DANNA, but some of these improvements may be reallocated to instead lower power consumption and improve efficiency. This improvement in execution speed is due to logic timing optimization as well as the reduction in the ratio of the element clock to the network clock.

DANNA2 neurons may have a linear leak value with per-neuron granularity. The leak value is applied similar to a negative weight as an input to the charge accumulator.

Synapses have a simplified discrete spiking timing dependent plasticity implementation. Weight adjustment values can be stored in a small look up table in which $t - N + 1...t + N$ values can be stored for a STDP window of 2N. These values are determined at hardware synthesis but are adjustable for exploration of different weight update curves.

An element can operate in a fan-in mode which allows for the 24 synapses to be forwarded to another neuron at the cost of one additional cycle of delay. This allows for a neuron with high connectivity or for two distant neurons to be connected. This functionality is provided by accumulating the synapses as usual, but the value from the accumulator is forwarded to another neuron and then reset for each network cycle. Each element has maximally four fan-in ports in which it can accept forwarded accumulations.

A DANNA2 neural network is executed on an array. DANNA2 supports two primary array types with different complexity and efficiency trade-offs.

A configurable grid array is implemented as a fixed size rectangular grid of elements. Connectivity of elements is dependent on spatial locality to enable the array to scale to thousands or more of elements. Each neuron has configurable synapses connecting it to 24 of its neighbors in a 5 by 5 connectivity grid pattern centered on the neuron. This array style is similar to DANNA and works well as a general purpose neuromorphic architecture for a programmable VLSI chip. One IC could support a very large number of different network configurations and varying applications. We expect DANNA2 grid arrays to be smaller in overall element count as compared to DANNA for a given set of resources. However, we also expect that the increased capability of a single element will result in an effective gain in overall density because fewer DANNA2 elements will be required to solve a particular task.

A sparse array instead only implements the elements and connectivity required for a given network. The advantage of this approach is connectivity is no longer limited to a fixed grid with spatial locality. Any neuron is able to connect to any other neuron with only the limitation of 24 total input synapses per element. Because unused elements and connections are removed, this approach gives way to increased practical density as well as improved power efficiency. However, this approach is primarily viable for FPGAs because the array is no longer externally programmable at runtime. We expect this to allow for lower size, area, and weight of FPGA based deployments. In the results section, we also explore the effect that more flexible connectivity can have on training performance.

Development and Demonstration of mrDANNA/DANNA Software

The mrDANNA/DANNA Software framework provides interfaces and software support for the development and testing of both neuromorphic applications and neuromorphic devices. The programming approach utilizes a genetic algorithm called Evolutionary Optimization of Neuromorphic Systems (EONS), which requires minimal support from the application and the device, but otherwise is a general purpose approach. The framework has already been employed to develop over twenty neuromorphic applications and six neuromorphic devices. One feature of the framework is that applications and devices program to a general model, and therefore applications can run on all architectures, and architectures can support all applications.

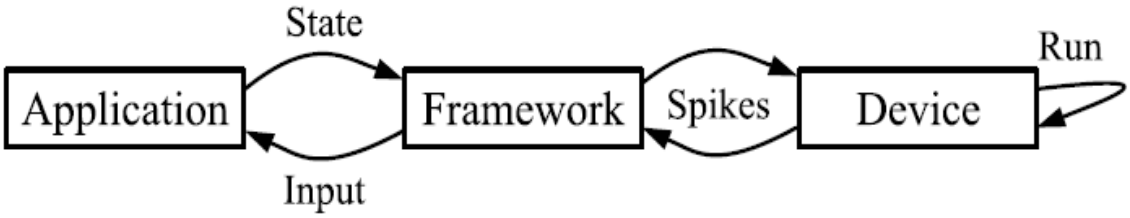

Figure 11*. The main loop of an application running on a neuromorphic device within the TENNLab framework. Applications express their states and interpret their inputs with values, whereas the devices process spikes.*

Fig. 11 displays the main loop of an application running on a neuromorphic device. The application communicates its state, which is composed of values, to the device. This communication is aided by a module in the framework which converts values to spikes and back again. The device accepts input spikes and then processes for a period of time, producing spikes as output. These spikes are converted to values which are then interpreted as input to the application, and the loop continues until the application is complete.

The framework supports many encodings of values to spikes, including the following:

1. **Direct** encoding of the value as spike amplitude.
2. **Binning**, by using multiple input neurons for a value, and partitioning the values into bins, each bin going to a specific input neuron.
3. **Rate Coding** the value into multiple spikes, where higher values are represented with more spikes.
4. **Stochastic Logic**, where values are converted into spike trains, with the decision to spike at each point along the train is assigned randomly with a probability based on the value.
5. **Temporal Coding**, where values are converted into two spikes, where the interval between the spikes is determined by the value.
6. Combinations of **Direct**, **Binning** and **Rate Coding**.

The framework supports the same encodings for output, except there is no direct encoding, because a neuron or synapse's spike value typically does not change. For Binning, multiple output neurons partition each output's value into bins, and the bin that spikes the most is used as the output. Similarly, with Rate Coding, the number of pulses determines the output value. The encodings and their various parameterizations may be assigned by the application at runtime.

Fig. 12 shows the software components that an application must implement within the TENNLab framework, and how the components fit in with the other software modules. The application implements three libraries, which get employed by three separate programs.

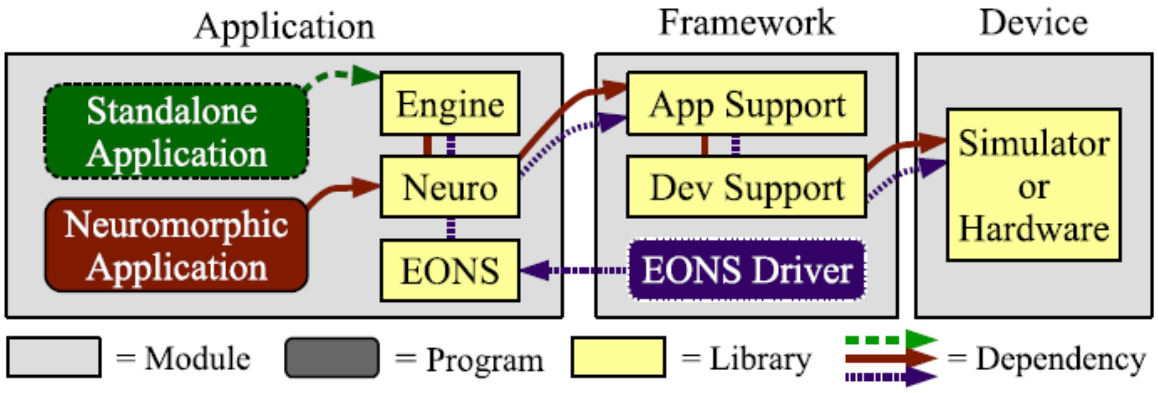

Figure 12. *TENNLab software modules from the application perspective.*

The first library is the Application Engine. This library implements functionality specific to the application that is independent of anything neuromorphic. The engine may be compiled with a Standalone Application program so that it may be executed and tested independent of any neuromorphic components.

The second library is the Neuromorphic Library. This library performs any interaction that the application may have with a neuromorphic device, such as sending state (see Fig. 1), receiving input and instructing the device to run. As depicted in Fig. 12, the application interacts with an Application Support module within the framework, which performs the relevant input/output encodings and interactions with the device. The specification of the neurons, synapses and their relevant parameters is in the form of a network serialization, which may be stored in a file or communicated as a character string. A second program within the application module is a Neuromorphic Application, which loads a one of these networks onto the device, and runs the application using the device. It assumes that the network has been created by EONS or by another means of machine learning.

The last application library is the EONS Library. This library implements a *fitness function*, which receives a network from the framework's EONS Driver, and then runs training tests using the network on the device. It returns a fitness score, which the EONS Driver uses to create further networks for the application. For a typical control application, the fitness function is composed of several independent runs of the application, each time with a different seed. The runs are scored based on their success, and scores are averaged for a resulting fitness value.

Our high-level simulator uses data gathered from the low-level simulator to simulate the behaviors seen in the low-level simulator. By abstracting away the individual devices to focus on neurons and synapses, the high-level simulator runs much more quickly, allowing for learning to happen at a practical pace. The high-level simulator is written in C++, and uses a discrete-event simulation to simulate the behavior of the device. As such, we only simulate activity as it occurs, calculating all state change in a component only when its state again becomes pertinent instead of once or more per cycle.

To encapsulate the difficulties in programming memristors to individual values, two notions of their state must be used – the number of training states achievable from learning signals, and the actual resistance value which may be modified by online learning processes and may be subject to user-configurable cycle and process variation. In addition to these variation parameters, the resistances, threshold voltages and currents, and number of training states can be user-configured for simulation of different memristive devices at runtime (see Appendix II). These runtime configurations also include options for altering online learning length, clock frequency, as well as logfile options and power estimation parameters.

In order to evaluate the device on real applications, we include it as a neuromorphic model in the broader neuromorphic software framework. To include this memristor-based spiking neuromorphic system to be in the software framework, we implement a variety of functions that enable other software to interact with the model. For example, we implement the ability to apply input to the simulation of the device, read output from the simulation, and run the simulation to execute real behavior. The application codes can then interact with the model through these functions. We also implement the appropriate functions to enable evolutionary optimization to operate on the model, including reproduction operations that will take one or two networks in the model and produce one or two new networks through recombination and/or mutation.

For verification purposes, the high-level simulation of the hardware creates and outputs detailed event logs to match the circuit activity tracked in the low-level simulator for essential elements such as the accumulation capacitor in each neuron and the two memristors in each synapse. These event logs can then be processed to produce output images that indicate the activity in a run of the network on any given input combination or to collect into larger datasets for broad analysis. For easy comparison to Cadence output, the simulator can also generate a spike raster table in the form of a .csv file for all input, internal, and output spikes generated by that run of the network.

Results and Discussion

mrDANNA Performance Results and Analysis

For the mrDANNA system design, the potential performance (application fitness, accuracy, energy, etc.) was assessed extensively using circuit-level simulation tools, specifically Cadence Spectre, and associated high-level simulation models developed in C++. The high-level mrDANNA simulation models were verified against transistor/memristor level simulations to ensure accuracy. Details of the simulation framework and associated neuromorphic models are provided in a later section of this report.

In order to demonstrate the performance of mrDANNA at a low level, including detail for all CMOS transistors and memristors included in the design, small circuits were extensively simulated using the SPICE-based simulator Cadence Spectre. Fig. 13 shows a simple two input mrDANNA neural network, complete with two twin-memristor synapses and a single post-synaptic neuron. With this simple network, we are able to assess the performance of the fundamental mrDANNA components (neurons and synapses) and also demonstrate the online learning mechanism.

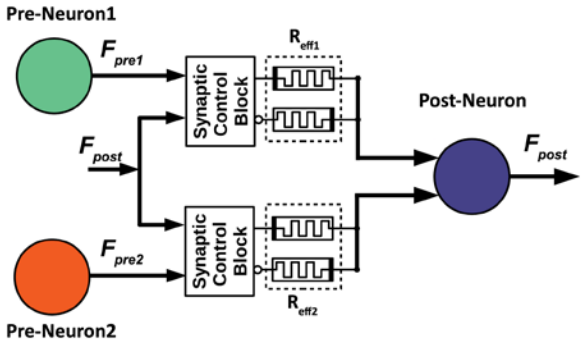

Figure 13. *Small mrDANNA network with two synapses driving a single post-synaptic neuron.*

Online learning in mrDANNA is based on a simplified learning rule that essentially mimics spike-timing dependent plasticity (STDP) observed in biological systems. STDP determines a variable rate of weight change based on the temporal distance between input and output spikes. In order to reduce the overhead required for learning, mrDANNA implements a single cycle form of STDP that only considers events during clock cycles immediately preceding and following an output firing event. This simple learning rule has been dubbed digital long term plasticity (DLTP) as the change is based on binary spikes (either a spike is present or not) that are tracked in finite time.

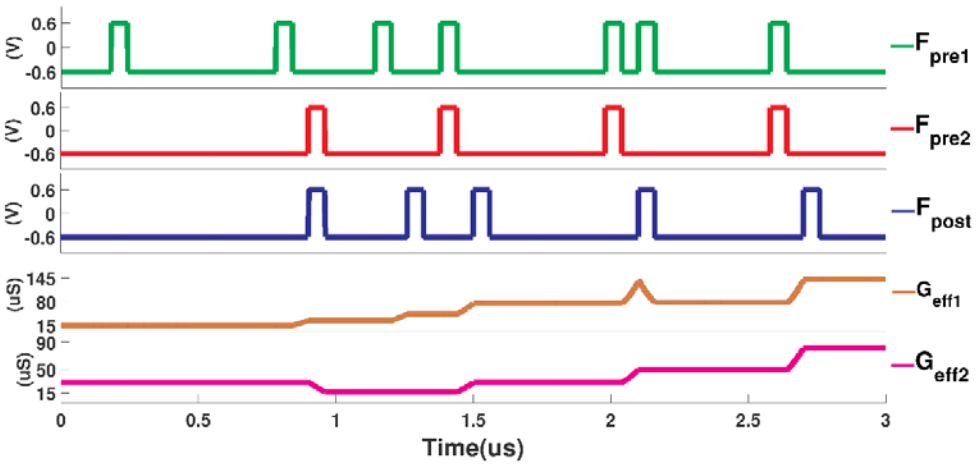

Figure 14. *Simulation result of small mrDANNA network showing both potentiation and depression as examples of online learning.*

Fig. 14 shows a Cadence Spectre simulation results for the simple two input network from Fig. 13. Here, several input spikes arrive through the two inputs over a few microseconds of time. It is clear from the top two input waveforms that each spike is a simple "binary" pulse. The middle waveform (F_{post}) clearly indicates that, at the start of the simulation, two input spikes must be accumulated before the output fires. When F_{post} fires an output spike, the online learning mechanism is activated such that the input synapse that fired just before F_{post} fires (in this case F_{pre1}) is potentiated, as indicated by a slight increase in the effective conductance or weight on that synapse (G_{eff1}). Further, when the second synapse, F_{pre2}, happens to fire at the same time as the output F_{post}, the associated weight (G_{eff2}) is depressed. Thus, online learning is possible in the mrDANNA system as constructed in the recent test chip.

It is important to remember that the mrDANNA approach to neuromorphic computing consists of two forms of training: (1) offline training in a simulated environment and (2) online learning via DLTP. Thus, even without online learning, pre-trained neural networks can be uploaded onto the reconfigurable mrDANNA fabric that are ready to perform a given task.

To demonstrate the effectiveness and efficiency of pre-trained mrDANNA neural networks, consider the example for iris classification shown in Fig. 15. This network has been trained using evolutionary optimization to recognize and classify three different types of iris flowers. The simplicity of this particular application leads to a very small network (11 neurons in this example) that can be simulated using detailed SPICE-like simulators such as Cadence Spectre. The neural network representation also indicates the neuron thresholds (written into the nodes) and the synapse parameters (weight and delay). As represented by the feedback, the iris application leads to a recurrent neural network, as is common for most mrDANNA networks.

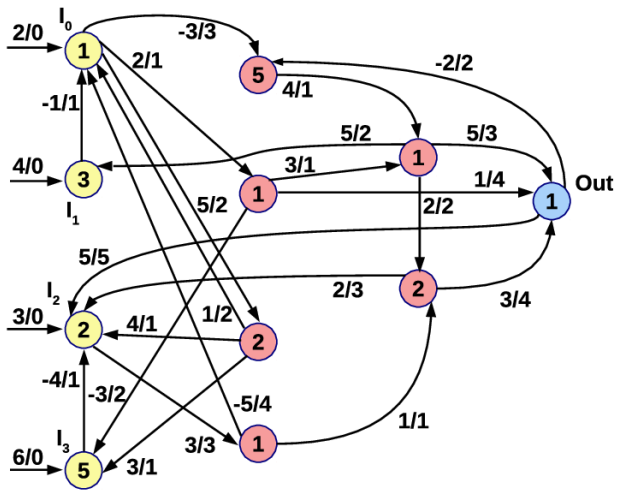

Figure 15. *Neural network view of mrDANNA implementation of iris flower classification application. The nodes indicate neurons with values for associated threshold levels. Edges represent synaptic connections with weight and delay parameters as weight/delay.*

Energy efficiency for the mrDANNA system is illustrated for three different classification applications (iris, breast cancer, diabetes) shown in Fig. 16. The energy numbers presented represent energy consumption for the enitre mrDANNA network over the entire classification time. As memristors are emerging devices, we also considered three different types of memristor, differentiated by their HRS and LRS levels. The result clearly indicates that energy is reduced for increasing LRS but plateus around 30kΩ, an amount achievable for HfO_2 based memristors.

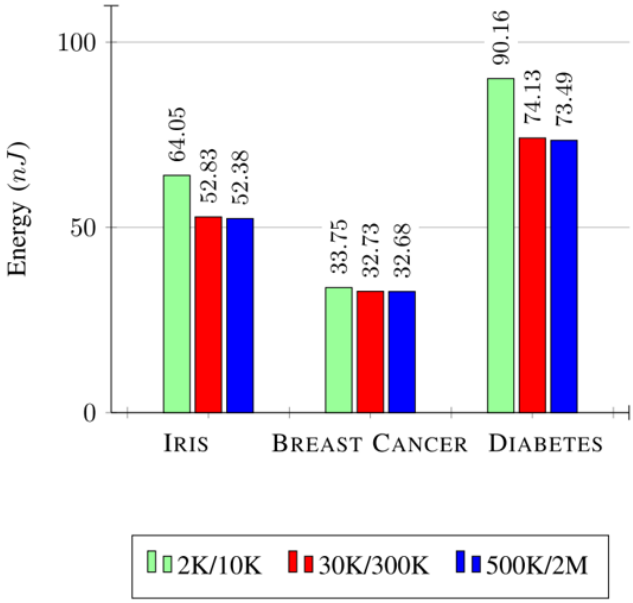

Figure 16. *Total energy consumption of mrDANNA networks implemented for three different classification applications: iris flowers, breast cancer, and diabetes. Impact of high and low resistance levels (HRS and LRS) on energy is also shown for three device types.*

Testing Results of Fabricated mrDANNA Components

The mrDANNA test chip represents an important step toward full integration of HfO_x based memristive devices with a full CMOS process stack (in this case 65nm). It is our understanding that prior work in this area has led to the integration of memristive elements with some CMOS components (e.g. NMOS, 2 metal layers) but this is the first realization of a full process including all CMOS components and a full metal stack (e.g. multiple flavors of NMOS and PMOS, 6 metal layers). Thus, fabrication has been slow at times as we've learned more about what to expect from such processes. Such slowdowns are expected for such a high risk, high reward project. It is our sincere belief that continued development and refinement of this full CMOS/memristor process represents an important technological advantage for our DOD sponsors and our country. Thus, we've been happy to take any and all delays in stride as we learn from the experience and work to ensure this and similar processes can be efficiently leveraged in future projects.

A specific challenge that led to limited test results of our final mrDANNA/DANNA test IC was due to having only 3 of the 6 total metal layers included in the first few wafer runs. For the full IC, all 6 metal layers are required for full system testing. Unfortunately, we do not have a wafer with all layers at the time of this writing. Thus, test results presented here are for test structures that work with 3 or fewer metal layers only. These include: independent memristor device test structures and some versions of the axon hillock neuron described in this report.

Our action taken on the testing front was to test the circuits we could, as extensively as possible. This was accomplished at UTK using a new probe station and characterization system (Fig. 17) acquired in 2018 via a DURIP award. The new system allows us to test all independent memristive test structures (e.g. 1T1R), standalone circuits (e.g. AH neurons), and the full system. It is worth noting that we have a probe card for the full IC system test, ready to use once wafers with all metal layers are fabricated. This capability has been useful for the mrDANNA project and we expect it to continue to be an important capability in future projects as well.

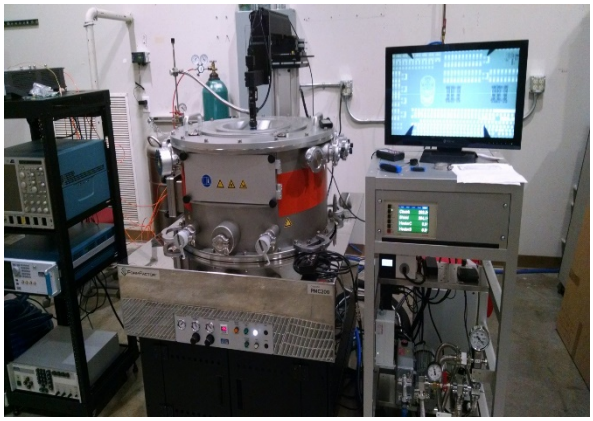

Figure 17. *Device and circuit characterization system at UTK acquired through 2018 DURIP.*

Results for the Axon Hillock Neuron Test Structure

The approach to testing the synchronous axon hillock neuron test structure divides the system into three sections and checks their functionality. These include the resistive input synapse, the analog accumulation and comparison, and finally the synchronous digital output spiking. The initial test is designed to check the functionality of the synchronous output. Once the output is determined to be functioning the analog accumulation and comparison is tested. After verifying the functionality in the output and middle section, the last test checks the complete functional behavior of the circuit.

To first check that the output is working as intended, applied DC voltages are applied that set the neuron into an always firing or never firing state. This is accomplished by applied 1.2V and 0V at the necessary locations. To set the neuron to always fire, the neuron must be turned on and its internal storage capacitor needs to be charged above the threshold. To accomplish this, the reference voltage used in the differential amplifier in the neuron is set to 0V while the tail transistor's gate is set to 1.2V. One positive weight resistive synapse input is set to always high, by applying 1.2V to its control circuit and the synapse positive power rail. The rest of the circuits voltage inputs are set to 0V. In this state, the neuron should always output a high voltage when the output enable signal is set to 1.2V. From simulation, the resulting output matches the output enable signal (Fig. 16). The physical circuit had the same results (Fig. 17). After observing correct behavior in the output from the circuit when in the always firing state, the reference voltage used is brought up from 0V to 1.2V. During this process the output should stay at 0V after the reference is above 0.7V. This was observed in simulation and physical testing. This concludes the first test and determines the output of the neuron is working properly. The output is a domino logic inverter, which outputs high and low voltages depending on the output of the analog accumulate and compare stage and the output enable signal.

The second test uses the same DC voltage setup as the always firing test to power the circuit. However, unlike the always firing test, the reference voltage is now set to something between 0V and 1.2V. This tests the thresholding property of the neuron. The neuron will need some time to charge up and will not fire every time the output is enabled. This further verifies the accumulation and comparison or middle section, of the neuron. To test the input section, which is made of resistive synapses, the synapse enable signal is no longer set to 1.2V. Instead the output enable signal is also fed into the synapse enable signal. This indicates that the neuron is only charging at the same time its output is enabled which increases the time to charge the neuron. From this setup the neuron should not fire after every time the output enable signal goes high. The physical and simulation results vary slightly (Fig. 18 and 19). The simulation results show absolute consistency while the experimental results show different numbers of enable signals (spikes) are needed to see an output spike. Ultimately, the goal of this test is to see output spikes during some but not all output enable signals to verify the synapse inputs, accumulation and reset functionality works.

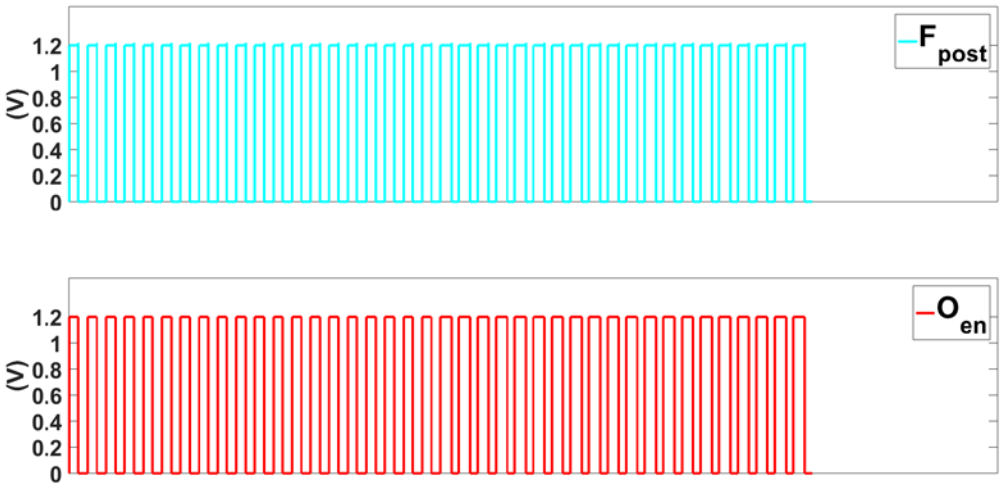

Figure 18. *Simulation results for the always firing neuron test. The top waveform is the neuron output, and the bottom wave form is the output enable signal. The top and bottom waveforms match showing the neuron is always firing.*

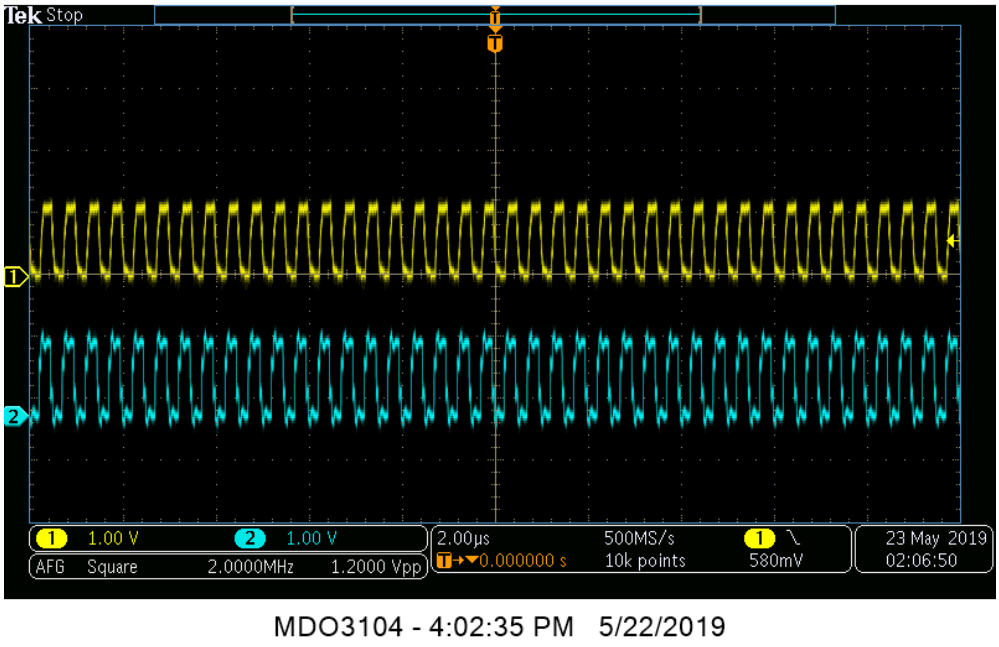

Figure 19. *Experimental results for the always firing neuron test. The top waveform is the neuron output, and the bottom wave form is the output enable signal. The top and bottom waveforms match showing the neuron is always firing.*

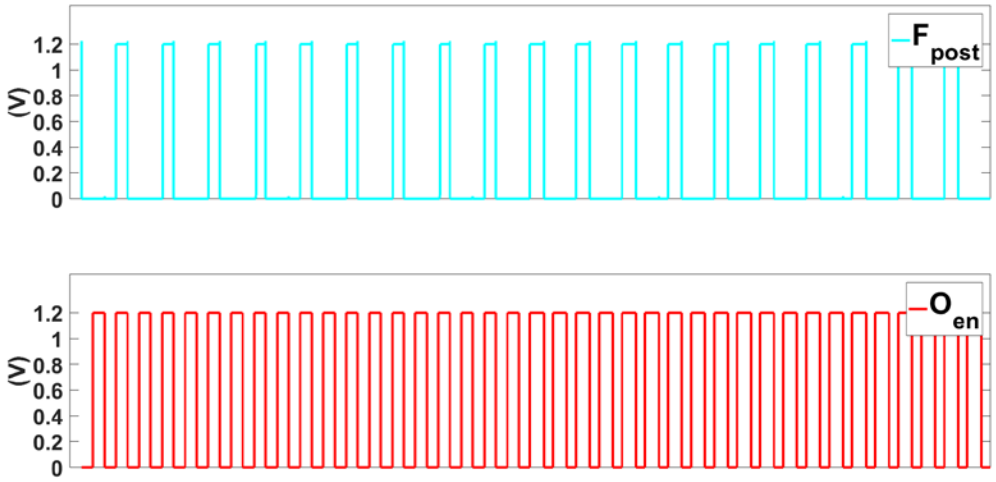

Figure 20. *Simulation results for the pulse firing neuron test. The top waveform is the neuron output, and the bottom wave form is the output enable signal. The output goes high inline with an output enable signal on every other output enable signal.*

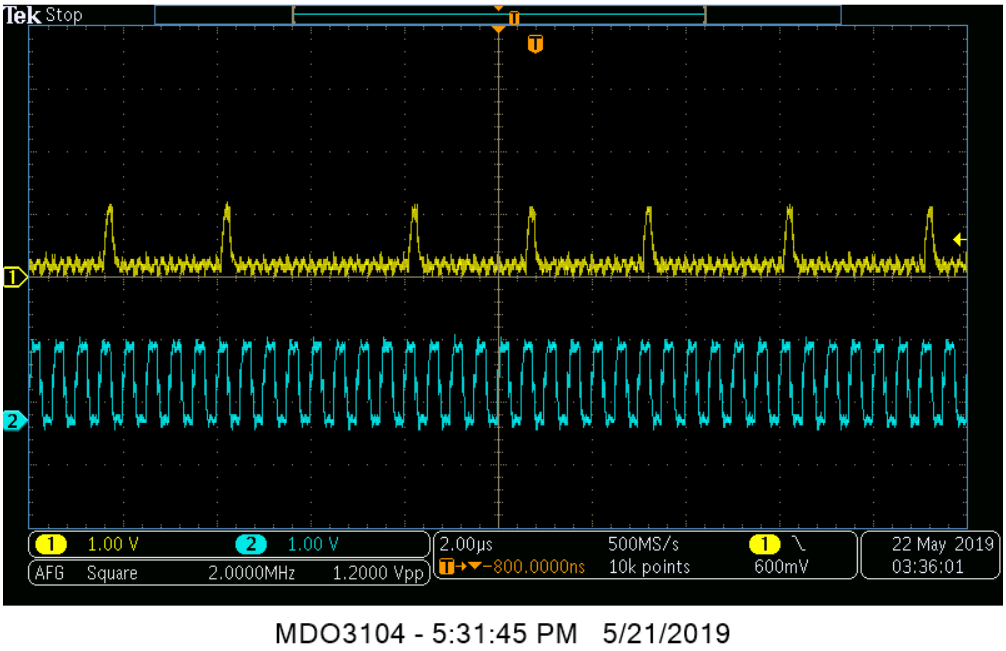

Figure 21. *Experimental results for the pulse firing neuron test. The top waveform is the neuron output, and the bottom wave form is the output enable signal. The output goes high inline with an output enable signal after a few microseconds.*

DANNA2 Evaluation and Results

In order to evaluate the DANNA2 model, we benchmarked it against previous models within the same software framework across both control system and classification use cases. For each application, we ran 100 EO training sessions for each model with different starting seeds for the random number generator. We report the number of epochs, time elapsed, final fitness value, and testing value for each. These benchmarks are intended to be relatively short running tests to

demonstrate differences between approaches. Often, final network training will run for more epochs with larger population sizes to obtain maximal accuracy.

All models are run with the default parameters, and the same EO hyper-parameters are used across all models and all applications. DANNA2 is running without any neuron leak and without STDP enabled. Future work may independently explore the effect of these features and parameters.

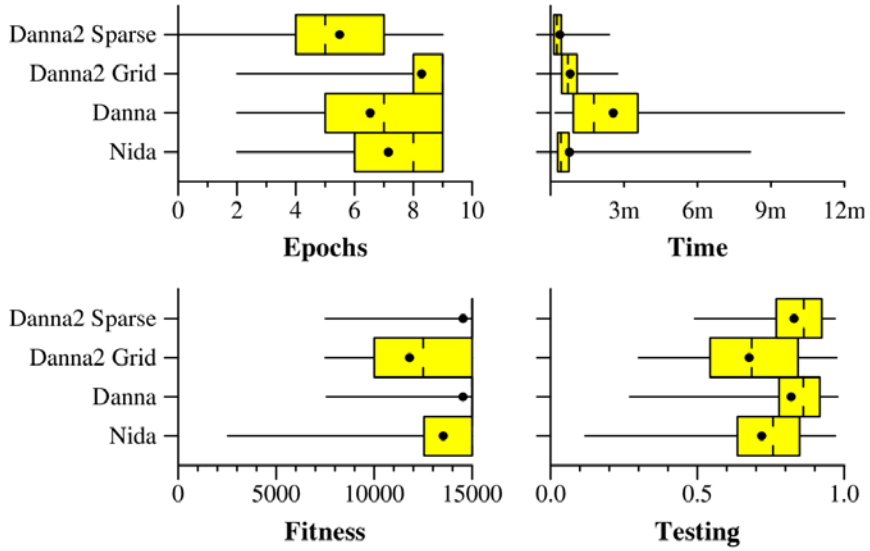

Figure 22. *Benchmark results for the Inverted Pendulum run with a population of 400, trained for a maximum of 10 epochs.*

Inverted Pendulum

The goal of the inverted pendulum application is to keep a pole balanced on cart by applying a force to the left or right side of the cart. The cart must stay in the range [-2.4 m, 2.4 m], and the pole's angle must stay in the range [-12°, 12°]. The cart-pole system has four variables describing the state: the position of the cart, the velocity of the cart, the angle of the pole, and the angular velocity of the pole. Every 0.02 seconds, we apply the state to the network, run the network for 100 cycles, and apply the output of the network to the cart.

The network has eight inputs and two outputs. The outputs correspond to pushing right or left; whichever fires more is the direction the cart is pushed. There are two inputs for each state variable, "Done" indicates a positive value and the other negative. The magnitude of the value is rate-encoded by mapping from that value's range to the range [1 pulse, 10 pulses]. The velocity terms don't explicitly have a range, so we use ±2 m/s and ±2 rad/s.

For training, we use six different starting conditions corresponding to a starting position at -1.2, 0, or 1.2 meters and a start angle of -0.15 or 0.15 radians (8.6°). Each run is cut off after 15,000 timesteps (5 minutes) or when the cart hits the wall or the pole falls past 12°. The fitness is the average of the survival time of the six training cases.

The generalization test runs the system in 1927 different test cases, with the position from -2.3 to +2.3 meters in increments of 0.1 and the pole angle from -0.2 to +0.2 radians in increments of 0.01. The testing score is normalized by dividing by 15,000.

As shown in Fig. 23, DANNA2 Sparse and DANNA perform the best in both the training fitness and the generalization testing. However, DANNA2 is able to reach this level of performance with much less training time. Interestingly, DANNA2 Grid appears to require more epochs to reach a suitable fitness for this application. Because of the stark difference between the configurable grid array and the sparse array performance, it seems the connectivity constraints of the grid array negatively impacted training performance. Notably, both DANNA2 arrays use a 5 × 10 array whereas DANNA uses a 15 × 15 array.

Robot Navigation

The goal is to explore an environment while avoiding obstacles. The robot is equipped with a LIDAR sensor taking measurements at 5 different angles equally spaced from -60°to +60°. It also has two limit switches that detect when there is a drop-off in front of the robot (i.e, the edge of a table or staircase). The LIDAR measurements are applied to the network by firing a single pulse with the charge linearly mapped such that closer distances have a higher charge value. Each limit switch input receives a single pulse if it is activated. For scoring, the environment is divided into tiles, and the robot's score is the percentage of tiles that were touched.

For the Room benchmark, we simulate the robot in a 6m x 6m room that contains spherical obstacles of various sizes. We train with eight different obstacle distributions plus one training run with an empty room. Each run is two minutes of simulated time. If the robot collides with an obstacle, the score for that run is multiplied by 0.6 as a penalty, and the run is prematurely ended. The fitness is the average score over the nine runs. The Room and Table benchmark additionally simulates the robot on a 6m x 6m table with eight different obstacle distributions. The generalization test simulates the robot with fifty different obstacle distributions, giving the score as the average over these runs.

Fig. 23 shows DANNA2 outperforming both NIDA and DANNA in final training fitness and testing generalization performance. DANNA2 Grid achieves the highest average fitness and testing score in both benchmarks while also requiring the least CPU time to train. In this application, we believe the connectivity constraints are reducing the search space of possible solution networks, so unlike the inverted pendulum application, the reduced flexibility results in quicker convergence.

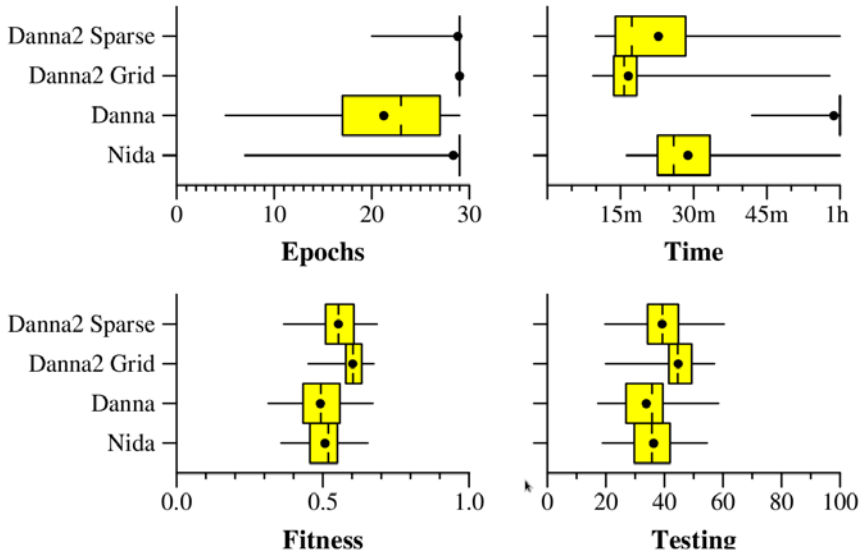

Figure 23. *Benchmark results for the RoboNav (robot navigation) in a room with a population of 200, trained for a maximum of 30 epochs.*

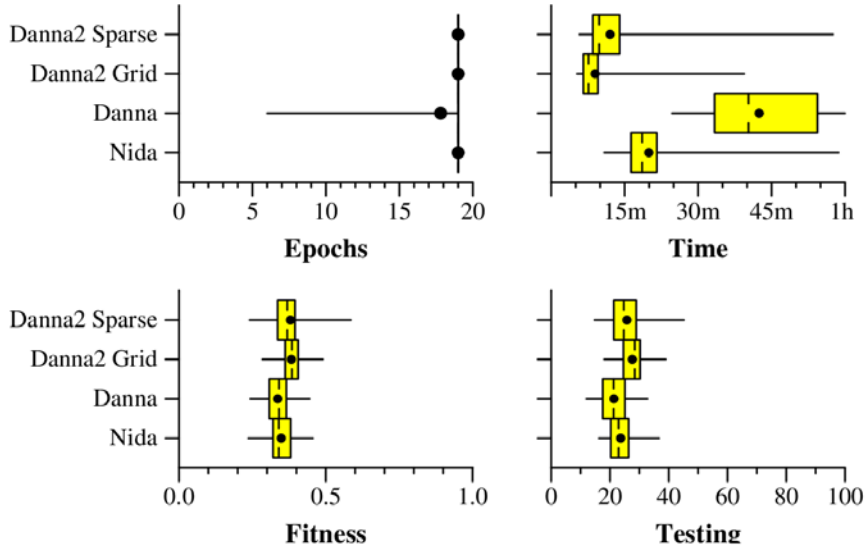

Figure 24. *Benchmark results for the RoboNav (robot navigation) in a room and on a table with a population of 200, trained for a maximum of 20 epochs.*

For the second benchmark shown in Fig. 24, the difficulty of the task in the given number of epochs and time results in little appreciable difference between models for fitness and testing. The key advantage of DANNA2 here is the reduced CPU time required compared to DANNA.

In both benchmarks, DANNA2 is using a 5 × 10 array compared to DANNA which is using a 15 × 15 array. This demonstrates a significant decrease in total array size while also showing improved training performance.

Robot Navigation on Hardware

Shown in Fig. 25, GRANT (Ground Roaming Autonomous Neuromorphic Targeter) is the hardware implementation of the robot navigation application. DANNA2 networks that are trained using the model's simulator can be saved, and a DANNA2 array on an FPGA can have its elements programmed to match that network.

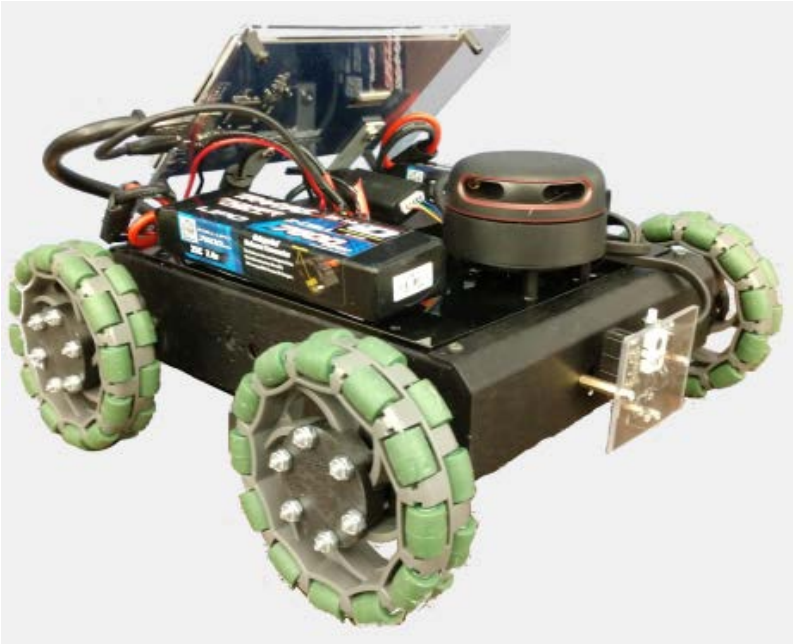

Figure 25. *Image of GRANT robotic demonstration vehicle, controlled by a DANNA2 array.*

GRANT gathers inputs from its environment using an RPLiDAR A2. The same five angles from -60°to +60° are sampled from this LiDAR to be used as input to the network. The limit switches that can be utilized in the training application were disabled for training and excluded from this hardware implementation. Worth noting are the two other network inputs: the BIAS and RANDOM inputs. These two inputs are necessary to encourage desired behaviors in the network. BIAS is used to ensure that the network is always receiving some kind of input even if the LiDAR inputs are not active. The RANDOM input fires into the network with a random weight to encourage non-deterministic behavior that might cause the robot to get stuck in a loop. The outputs of the network are simply "votes" for the left and right side motors to apply power in the forward or backward directions. The network used on GRANT (Fig. 26) trained with 88.83% fitness and only used a total of 45 neurons and 73 neurons.

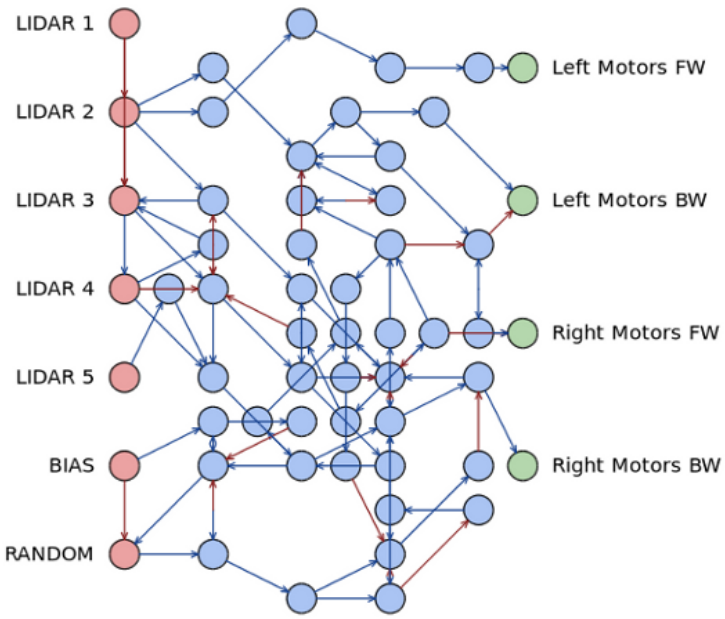

Figure 26. *The RoboNav neural network implemented on GRANT.*

To communicate the information GRANT collects from its environment, a communications architecture was created centered around the AXI-Stream protocol. Fig. 27 shows this architecture.

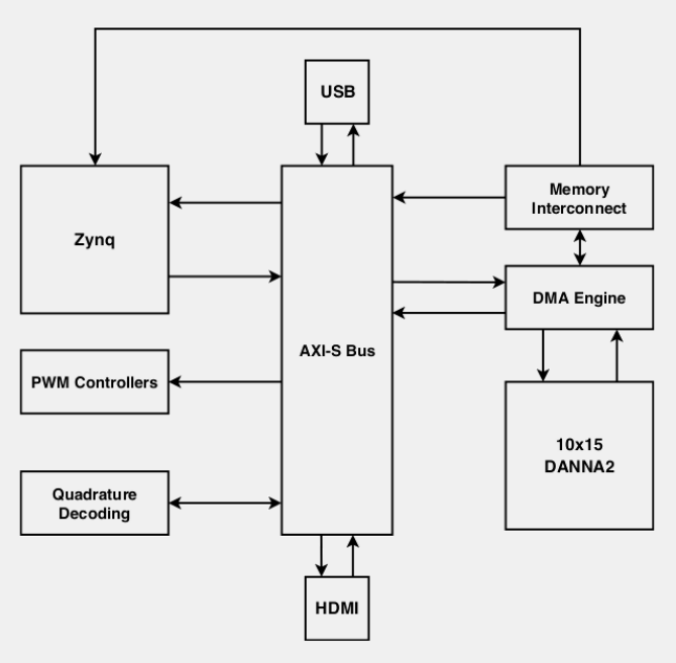

Figure 27. *GRANT communications architecture.*

Using this communication architecture, GRANT is able to use the trained network to navigate an unknown space while successfully avoiding obstacles.

Inverted Pendulum on Hardware

SABR (Self-Adjusted Balancing Robot) is a physical implementation of the inverted pendulum problem. The simulator for the problem can train networks, and these trained networks can then be used on the hardware to allow the robot (shown in Fig. 28) to maintain its balance.

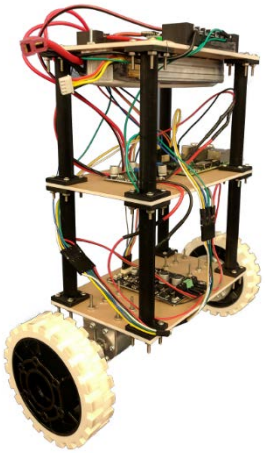

Figure 28. *Image of the self-adjusted balancing robot (SABR), controlled by DANNA2 array.*

The simulator for the inverted pendulum provides inputs corresponding to the position of the robot (relative to its original position), the change in position since the last timestep, the angle of the robot (relative to its original angle), and the change in angle since the last timestep. Using this set of inputs, the network makes decisions on whether it should move right or left. The trained network (illustrated in Fig. 29) used on the robot had an accuracy of 100% and utilized 50 neurons and 170 synapses.

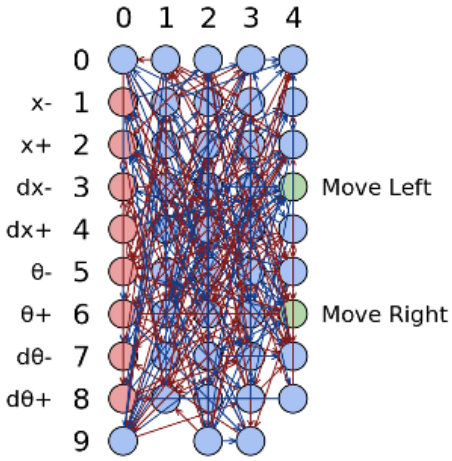

Figure 29. *Inverted pendulum neural network implemented on DANNA2 to control SABR.*

The SABR neural network was used on the robot by running the simulator with this network on a Raspberry Pi and sending the output to the motors. This resulted in the physical robot being able to maintain its balance, even when someone attempts to knock it down.

Conclusions

In this project, we have successfully designed and developed a spiky recurrent neural network (SRNN) based neuromorphic architecture, with both digital (CMOS-only) and mixed-signal (CMOS/memristor) implementations. The digital DANNA and DANNA2 implementations have been successfully demonstrated using FPGAs for a variety of applications, including robot navigation and control for balancing. In fact, an important result of this research is that the SRNN based networks explored are particularly useful for control problems and classifying spatio-temporal data. Further, as evidenced by the small network examples provided in this report, these DANNA and mrDANNA networks tend to work well for even a very small number of neurons (on the order of 10). In comparison, deep neural networks typically require 1,000s if not 10,000s of neuron nodes for similar applications.

We have also partnered with collaborators at SUNY Poly to design and fabricate a CMOS/memristor test chip built from 65nm CMOS and HfO_2 memristors. The design of CMOS/memristor components required improved device models for the HfO_2 devices for more efficient circuit simulation in SPICE-based tools. Thus, we have developed an improved device model (Appendix I) that more accurately reflects experimentally observed behavior for both DC and transient effects. Device level testing on the new chips, performed at both SUNY Poly and UTK, show that these models continue to be accurate and reflect experimental results.

For the mrDANNA test integrated circuit (IC), we were able to test simple test structures for axon hillock neurons and some circuits used for memristor forming/writing. The results of these component tests are promising in that we can form and operate memristors and the axon hillock neuron performs as expected. However, our testing at this point is somewhat limited as the current wafer runs from SUNY Poly include metal layers up to metal3 (M3) but none higher. Thus, the larger circuits (e.g. full mrDANNA neuromorphic cores) cannot be tested at this time. All test structure and the full mrDANNA system array are included in the mask sets, ready for fabrication in future projects. We also designed this system as a "sea of gates" such that different combinations of synapse and neuron can be explored in future work without having to create new masks for silicon and the first 3 metal layers.

A major limitation of most neuromorphic systems developed to date is a lack of reliable software support. In this project, we have successfully developed a full software framework to support SRNN style architectures, particularly the DANNA and mrDANNA architectures. This software is modular in the sense that training, application development and the implementation model can all be swapped out somewhat independently. For example, an application (e.g. robot navigation) originally developed for DANNA can also be reused to generate networks for mrDANNA with relative ease. The software framework in its entirety allows a user to develop an application via specified "fitness" requirements, generate neural networks (typically SRNN type) for the application, simulate and interface with the hardware. Thus, we have developed a full suite of tools that enable users to more easily work with DANNA and mrDANNA neuromorphic computing systems.

Publications Resulting from This Project

Journal Papers

1. J.S. Plank, C.D. Schuman, G. Bruer, M.E. Dean, and G.S. Rose, "The TENNLab Exploratory Neuromorphic Computing Framework," *IEEE Letters of the Computer Society*, vol. 1, no. 2, pp. 17—20, July—December 2018.
2. S. Amer, M.S. Hasan, M.M. Adnan, and G.S. Rose, "," *IEEE Journal of the Electron Devices Society*, vol. 7, pp. 18—25, October 2018.
3. G. Chakma, M.M. Adnan, A.R. Wyer, R. Weiss, C.D. Schuman, and G.S. Rose, "Memristive Mixed-Signal Neuromorphic Systems: Energy-Efficient Learning at the Circuit-Level," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)*, vol. 8, no. 1, March 2018
4. S. Amer, M.S. Hasan, and G.S. Rose, "Analysis and Modeling of Electroforming in Transition Metal Oxide Memristors," *IEEE Electron Device Letters*, vol. 39, no. 1, pp. 19—22, January 2018.

Conference Papers

5. J.S. Plank, C. Rizzo, K. Shahat, G. Bruer, T. Dixon, M. Goin, G. Zhao, J. Anantharaj, C.D. Schuman, M.E. Dean, G.S. Rose, N.C. Cady, and J. Van Nostrand, "The TENNLab Suite of LIDAR-Based Control Applications for Recurrent, Spiking, Neuromorphic Systems," in *Proceedings of the Government Microcircuit Applications and Critical Technology Conference (GOMACTech)*, Albuquerque, NM, March 2019.
6. S. Amer and G.S. Rose, "A Multi-Driver Write Scheme for Reliable and Energy Efficient 1S1R ReRAM Crossbar Arrays," in Proceedings of *International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, March 2019.
7. M.M. Adnan, S. Sayyaparaju, G.S. Rose, B.W. Ku, S.-K. Lim, and C.D. Schuman, "A Twin Memrsitor Synapse for Spike Timing Dependent Learning in Neuromorphic Systems," in *Proceedings of the International IEEE System-on-Chip Conference* (*SOCC*), Washington, D.C., September 2018.
8. A.W. Disney, J.S. Plank, and M. Dean, "Four Simulators of the DANNA Neuromorphic Computing Architecture," in *Proceedings of International Conference on Neuromorphic Systems (ICONS)*, Knoxville, TN, July 2018.
9. J.P. Mitchell, M.E. Dean, G. Bruer, J.S. Plank, and G.S. Rose, "DANNA 2: Dynamic Adaptive Neural Network Arrays," in *Proceedings of International Conference on Neuromorphic Systems (ICONS)*, Knoxville, TN, July 2018.
10. J.J.M. Reynolds, J.S. Plank, C.D. Schuman, G. Bruer, A. Disney, M.E. Dean, and G.S. Rose, "A Comparison of Neuromorphic Classification Tasks," to appear in *Proceedings of International Conference on Neuromorphic Systems (ICONS)*, Knoxville, TN, July 2018.

11. S. Sayyaparaju, R. Weiss, and G.S. Rose, “A Mixed-Mode Neuron with On-Chip Tunability for Generic Use in Memristive Neuromorphic Systems,” in *Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Hong Kong, China, July 2018.
12. A. Young, M. Dean, J. Plank, G.S. Rose, and C. Schuman, “Neuromorphic Array Communications to Support Large-Scale Neural Networks,” in *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN)*, Rio de Janeiro, Brazil, July 2018.
13. M.M. Adnan, S. Amer, and G.S. Rose, “A Novel Scan-In Scheme for CMOS/ReRAM Programmable Logic Circuits,” in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, Italy, May 2018.
14. N. Skuda, C.D. Schuman, G. Chakma, J.S. Plank, and G.S. Rose, “High-Level Simulation for Spiking Neuromorphic Computing Systems,” in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, Italy, May 2018.
15. G. Chakma, N. Skuda, C.D. Schuman, J.S. Plank, M.E. Dean, and G.S. Rose, “Energy and Area Efficiency in Neuromorphic Computing for Resource Constrained Devices,” in *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI)*, Chicago, Illinois, May 2018.
16. S. Sayyaparaju, S. Amer, and G.S. Rose, “A Bi-Memristor Synapse with Spike-Timing-Dependent Plasticity for On-Chip Learning in Memristive Neuromorphic Systems,” in *Proceedings of International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, March 2018.
17. N. Cady, K. Beckmann, W. Olin-Ammentorp, G. Chakma, S. Amer, R. Weiss, S. Sayyaparaju, M. Adnan, J. Murray, M. Dean, J. Plank, G.S. Rose, and J. Van Nostrand, “Full CMOS-Memristor Implementation of a Dynamic Neuromorphic Architecture,” in *Proceedings of the Government Microcircuit Applications and Critical Technology Conference (GOMACTech)*, Miami, FL, March 2018.
18. J. Plank, C. Schuman, M. Dean, G. Rose, and N. Cady, “A Unified Hardware/Software Co-Design Framework for Neuromorphic Computing Devices and Applications,” in *Proceedings of IEEE International Conference on Rebooting Computing* (*ICRC*), Washington, DC, November 2017.
19. J.P. Mitchell, G. Bruer, M.E. Dean, J.S. Plank, G.S. Rose, and C.D. Schuman, “NeoN: Neuromorphic Control for Autonomous Robotic Navigation,” in *Proceedings of the International Symposium on Robotics and Intelligent Sensors*, Ottawa, Canada, October 2017.
20. C.D. Schuman, J.S. Plank, G.S. Rose, G. Chakma, A. Wyer, G. Bruer, and N. Laanait, “A Programming Framework for Neuromorphic Systems with Emerging Technologies,” in *Proceedings of ACM International Conference on Nanoscale Computing and Communications* (*NanoCom*), Washington, D.C., September 2017.

21. R. Weiss, G. Chakma, and G.S. Rose, “A Synchronized Axon Hillock Neuron for Memristive Neuromorphic Systems,” in *Proceedings of Midwest Symposium on Circuits and Systems* (*MWSCAS*), Boston, MA, August 2017.
22. S. Amer, K. Beckmann, N.C. Cady, and G.S. Rose, “Design Techniques for In-Field Memristor Forming Circuits,” in *Proceedings of Midwest Symposium on Circuits and Systems* (*MWSCAS*), Boston, MA, August 2017. (poster)
23. G. Chakma, S. Sayyaparaju, R. Weiss, and G.S. Rose, “A Mixed-Signal Approach to Memristive Neuromorphic System Design,” in *Proceedings of Midwest Symposium on Circuits and Systems* (*MWSCAS*), Boston, MA, August 2017. (special session)
24. A. Wyer, M.M. Adnan, B.W. Ku, S.-K. Lim, C.D. Schuman, R.C. Pooser, and G.S. Rose, “Evaluating Online-Learning in Memristive Neuromorphic Circuits,” in *Proceedings of Neuromorphic Computing Symposium* (*NCS*), Knoxville, TN, July 2017.
25. S. Amer, S. Sayyaparaju, G.S. Rose, K. Beckmann, and N.C. Cady, “A Practical Hafnium-Oxide Memristor Model Suitable for Circuit Design and Simulation,” in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, Baltimore, MD, May 2017.
26. S. Sayyaparaju, G. Chakma, S. Amer, and G.S. Rose, “Circuit Techniques for Online Learning of Memristive Synapses in CMOS-Memristor Neuromorphic Systems,” in *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI)*, Banff, Alberta, Canada, May 2017.
27. W. Olin-Ammentorp, K. Beckmann, J.E. Van Nostrand, G.S. Rose, M.E. Dean, J.S. Plank, G. Chakma, and N.C. Cady, “Applying Memristors Towards Low-Power, Dynamic Learning for Neuromorphic Applications,” in *Proceedings of the Government Microcircuit Applications and Critical Technology Conference (GOMACTech)*, Reno, NV, March 2017.
28. J.S. Plank, G.S. Rose, M.E. Dean, and C.D. Schuman, “A CAD System for Exploring Neuromorphic Computing with Emerging Technologies,” in *Proceedings of the Government Microcircuit Applications and Critical Technology Conference (GOMACTech)*, Reno, NV, March 2017.
29. G. Chakma, M.E. Dean, G.S. Rose, K. Beckman, H. Manem, and N. Cady, “A Hafnium-Oxide Memristive Dynamic Adaptive Neural Network Array,” in *International Workshop on Post-Moore's Era Supercomputing (PMES)*, Salt Lake City, UT, November 2016.
30. M.E. Dean, J. Chan, C. Daffron, A. Disney, J Reynolds, J.S. Plank, G.S. Rose, J.D. Birdwell, and C.D. Schuman, “An Application Development Platform for Neuromorphic Computing,” in *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN)*, Vancouver, Canada, July 2016.
31. C. Daffron, J. Chan, A. Disney, L. Bechtel, R. Wagner, M.E. Dean, G.S. Rose, J.S. Plank, J.D. Birdwell, and C.D. Schuman, “Extensions and Enhancements for the DANNA Neuromorphic Architecture,” in *Proceedings of IEEE SoutheastCon*, Norfolk, Virginia, March 2016.

32. N. Cady, K. Beckmann, H. Manem, M.E. Dean, G.S. Rose, and J. Van Nostrand, “Towards Memristive Dynamic Adaptive Neural Network Arrays,” in *Proceedings of the Government Microcircuit Applications and Critical Technology Conference (GOMACTech)*, Orlando, FL, March 2016.

Patent Applications

33. M.M. Adnan and G.S. Rose, “A Scan In/Out Scheme for Deep Access and Reading of Non-Volatile Memory,” Invention Disclosure, 2019.

Appendix I – Verilog-A Model Code for HfO_2 Memristive Device

```
// Verilog-A for HfO2 based Memristors
//  Developed by: Sherif Amer
//  Institution:  University of Tennessee, Knoxville
//
// This model only includes memrsitor switching equations. It does not include
// any secondary effects such as variations, temperature dependence or aging.
`include "constants.vams"
`include "disciplines.vams"

module memr_TMO_switching(p,n,r);
  inout          p;          // positive pin
  inout          n;          // negative pin
  inout          r;          // not physical, facilitate resistance computaion
  electrical     p, n, r;

  // global parameters
  parameter real window        =  0;

  // model parameters
  parameter real HRS         =  1.5e5; // high resistance state
  parameter real LRS         =  1e4;   // low resistance state
  parameter real Vtp         =  0.75;  // positive threshold voltage
  parameter real Vtn         = -1.0;   // negative threshold voltage
  parameter real tsw_p       =  1e-8;  // time to switch under +V bias
  parameter real tsw_n       =  1e-6;  // time to switch under -V bias

  // window parameters
  parameter real theta_HRS   =  0.85;  // transition boundary at HRS
  parameter real beta_HRS    =  0.2;   // transition sharpness at HRS
  parameter real theta_LRS   =  2.1;   // transition boundary at LRS
  parameter real beta_LRS    =  0.05;  // transition sharpness at LRS

  // fitting parameters
  parameter real CLRS        =  1;     // speed while transitioning to LRS
  parameter real CHRS        =  1;     // speed while transitioning to HRS
  parameter real P_LRS       =  3;     // non-linearity for LRS transition
  parameter real P_HRS       =  3;     // non-linearity for HRS transition

  parameter real Rinit       =  1e4;   // initial resistance

  // internal variables
  real delR;
  real time_last;
  real Vwr;
  real delt;
  real Rm;
  real Rm_tmp;

analog begin
  @( initial_step or initial_step("dc") ) begin
    delt             = 0;
    time_last        = 0;

    Rm               = Rinit;
    delR             = HRS - LRS;
  end

  delt       = $abstime - time_last;
  time_last  = $abstime;
  Vwr        = V(p,n);
```

```
  ///// Model Equations //////////////////////////////////////////////////
  // if window == 0, window function is deactivated
  if (window == 0) begin
    if (Vwr >= Vtp  && Rm != LRS) begin
      // switching equation while transitioning from HRS to LRS
      Rm_tmp = Rm -  delt* CLRS* (delR/tsw_p)*( pow(((Vwr-Vtp)/Vtp), P_LRS));

      if (Rm_tmp <= LRS) begin
       Rm_tmp = LRS; //clipping the resistance at LRS
      end
    end
    else if (Vwr < Vtn  && Rm != HRS) begin
      //switching equation while transitioning from LRS to HRS
      Rm_tmp = Rm + delt* CHRS * (delR/tsw_n) *(pow(((Vwr-Vtn)/Vtn), P_HRS));

      if (Rm_tmp >= HRS) begin
        Rm_tmp = HRS; //clipping the resistance at HRS
      end
    end
    else begin
      Rm_tmp = Rm; // no change
    end
  end

  // window == 1, window function is activated
  if (window == 1) begin
    if (Vwr >= Vtp && Rm != LRS) begin
      Rm_tmp = Rm -  delt* CLRS* (delR/tsw_p)* pow(((Vwr-Vtp)/Vtp), P_LRS) /
               (1+exp((theta_LRS*LRS-Rm)/(delR)/beta_LRS));

      if (Rm_tmp <= LRS) begin
        Rm_tmp = LRS; //clipping the resistance at LRS
      end
    end
    else if (Vwr <= Vtn && Rm != HRS) begin
      Rm_tmp = Rm + delt* CHRS * (delR/tsw_n) *pow(((Vwr-Vtn)/Vtn), P_HRS) /
               (1+exp((Rm-theta_HRS*HRS)/(delR)/beta_HRS));

      if (Rm_tmp >= HRS) begin
        Rm_tmp = HRS; //clipping the resistance at HRS
      end
    end
    else begin
      Rm_tmp = Rm; // no change
    end
  end
  ///// End Model Equations //////////////////////////////////////////////////

  Rm = Rm_tmp;
  I(p,n) <+ Vwr / Rm;
  V(r) <+ Rm;
end         // end analog
endmodule
```

Appendix II – mrDANNA Simulator Instructions

This software is an event-driven simulator of MrDANNA hardware using TennLab's neuromorphic simulation library to easily connect applications to hardware and software models. The current implementation uses Leaky Integrate-and-Fire neurons using memristive synapses.

Feature Overview

1. Simulate mrDANNA hardware quickly with a fast, event-driven simulator
2. Create neuromorphic applications easily, or train with use any of the apps in the TennLab library
3. Measure the effect of different memristive materials and processes on learning and performance as information becomes available
4. Predict the accuracy and yield of manufactured chips by simulating process and operational variations
5. Generate detailed logs and spike rasters to verify with both circuit simulation and physical devices

System Requirements

Linux systems with g++ 4.9 or higher

Installation Instructions

Extract the included archive to the desired location. Run Setup in the main directory, or to manually configure installation run make in the following locations in order:

1. The main neuro directory
2. The neuro/eo subdirectory
3. The neuro/models/mrdanna subdirectory

At this point, the mrDANNA library is built and is located in:

neuro/models/mrdanna/lib/libneuro.a

This is a statically linked library, and may be included in applications following the model API. To install an application, change to that application's root directory and run:

make model=mrdanna

to build with the mrDANNA model.

Running a Simulator

After an application is built, the application's 'bin' directory contains the necessary executables for training and running networks. To train a network, run ***bin/[appname]EOGG*** from within the app directory; for example to run the classification tasks the executable is ***bin/class2dEOGG***. The best network will be saved to ***network.txt***.

To test a saved network, run ***bin/[appname]TestRun*** from within the app directory and follow the directions in the terminal. So for example to test a network file saved to ***network.txt*** in the ***class2d*** directory and make a verification run a user would run:

```
bin/class2dTestFit ../.. bin/ network.txt
```

Simulation Configuration

Many of the simulation variables can be edited for modification using one of several runtime configuration files. These configuration files are similar to JSON files and use the following format:

```
{
        Key1: value1,
        Key2: value2,
}
```

These files are located in the ***mrdanna/defaults*** directory and are copied to the ***neuro/defaults*** directory upon setup or ***make***.

The main configuration files are ***device.txt*** and ***mrdanna.txt***

Configuration File: *device.txt*

This file sets log configurations – make sure these values are false while running EO!

Parameter Key	Parameter Type	Description	Default Value
track_spikes	Bool	Create a spike raster file?	FALSE
track_spikes_file	String	The csv filename for a spike raster	spike_history.csv
print_events	Bool	Output the event log to Stderr	FALSE
log_events	Bool	Output the event log to a file	FALSE
logfile	String	If log_events is set to true, this is the output file	event_log.txt

Configuration File: *mrdanna.txt*

This file controls physical characteristics for the mrDANNA system, energy usage, variability, and designating the file(s) used to describe the memristor types.

Parameter Key	Parameter Type	Description	Default Value
memristor_types	String	This value sets the config file for memristor information	HfO
cycle_time	Double	Clock time per cycle	0.00000002
stdp_length	Int	Number of cycles of STDP to be considered; 0 cycles disables online learning, 1 cycle is LTP/LTD	1
spike_voltage	Double	The voltage of a spike across the memristor	1.2
learn_voltage	Double	The voltage of a learning signal to a memristor	2.0
lrs_variation	Double	The % variation for the low resistance value	0.0
hrs_variation	Double	The % variation for the low resistance value	0.0
change_time_variation	Double	The % variation for the low resistance value	0.0
threshold_variation	Double	The % variation for the low resistance value	0.0
cycle_variation	Double	The % variation for the low resistance value	0.0
energy_usage_neuron_inactive	Double	The per cycle energy usage of an inactive neuron	7.1 * 10^-13
energy_usage_neuron_accum	Double	The per cycle energy usage of a neuron that is accumulating	2.4 * 10^-12
energy_usage_neuron_fire	Double	The per cycle energy usage of a neuron that is firing	2.66 * 10^-12
energy_usage_synapse_delay	Double	The per cycle energy usage of a synapse that is delaying a fire	2 * 10^-13
energy_usage_synapse_learn	Double	The per cycle energy usage of a synapse that is firing	2 * 10^-13
energy_usage_synapse_inactive	Double	The per cycle energy usage of a synapse that is inactive	6.67 * 10^-14

Configuration File: *HfO.txt*

This file sets the memristor settings – along with mrdanna.txt this allows the user to change to different memristive devices without recompiling. While HfO.txt is the file included, a user can have as many memristor files as desired, changing modes by simple adjusting the memristor.txt.

Parameter Key	Parameter Type	Description	Default Value
HRS	Double	The resistance of HRS	15000
LRS	Double	The resistance of LRS	9000
set_time	Double	The time (in seconds) to set (lrs -> hrs)	.000001
reset_time	Double	The time (in seconds) to reset (hrs->lrs)	.00000005
Vset	Double	The voltage threshold to set	1.0
Vreset	Double	The voltage threshold to reset	1.0
Ifail	Double	The current threshold at which a memristor “breaks” and becomes stuck at lrs	.00026666
set_states	Double	The number of states that the memristor can be externally programmed to. 2 means only hrs/lrs can be accurately programmed	2

List of Acronyms

VLSI:	Very Large Scale Integration
IC:	Integrated Circuit
CMOS:	Complementary Metal Oxide Semiconductor
SPICE:	Simulation Program with Integrated Circuit Emphasis
FPGA:	Field Programmable Gate Array
VHDL:	VHSIC Hardware Description Language
VHSIC:	Very High Speed Integrated Circuit
NIDA:	Neuroscience Inspired Dynamic Architecture
DANNA:	Dynamic Adaptive Neural Network Array
mrDANNA:	Memristive Dynamic Adaptive Neural Network Array
DRC:	Design Rule Check
EO:	Evolutionary Optimization
LTP:	Long Term Potentiation
LTD:	Long Term Depression
HRS:	High Resistance State
LRS:	Low Resistance State
GRANT:	Ground Roaming Autonomous Neuromorphic Targeter
SABR:	Self Adjusted Balancing Robot