AFRL-RI-RS-TR-2023-013

DESIGN APPROACHES FOR EFFICIENT RECONFIGURABLE NEUROMORPHIC SYSTEMS

UNIVERSITY OF TENNESSEE, KNOXVILLE

JANUARY 2023

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-2023-013 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
JOSEPH E. VANNOSTRAND
Work Unit Manager

/ S /
GREGORY J. HADYNSKI
Assistant 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

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
JANUARY 2023	FINAL TECHNICAL REPORT	FEBRUARY 2019	AUGUST 2022

4. TITLE AND SUBTITLE

DESIGN APPROACHES FOR EFFICIENT RECONFIGURABLE NEUROMORPHIC SYSTEMS

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
N/A	FA8750-19-1-0025	61102F
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
		R2R4

6. AUTHOR(S)

Garrett S. Rose

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)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Air Force Research Laboratory/RITB 525 Brooks Road Rome NY 13441-4505	AFRL/ RI	AFRL-RI-RS-TR-2023-013

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

RAVENS was both about refining circuit-level designs for more robust integration and also pointing toward how these memristive mixed-signal elements can be integrated into full neuromorphic systems. To that end, this work also included the development of standard cells, memory generators, refined I/O cells, interconnect circuitry along with other components and tools necessary for realizing full system-on-chip designs using hybrid CMOS-memristive technologies. This work included necessary add-on features and modifications to the process design kit (PDK) from SUNY Poly to reflect the added layers in the manufacturing process used to realize both memristors and CMOS on the same substrate. Since the SUNY Poly process and associated PDK were being developed as we worked through other design requirements, we also prototyped pure digital components (e.g. scan chain logic, basic memory, RISC-V core logic) using a commercial CMOS process. This commercial tapeout allowed us to test some of the key design elements required for the more digital parts of the RAVENS system, including the automated design flows developed to more efficiently integrated CMOS and memristive components. The culmination of this work was a CMOS-memristive system design that has recently been fabricated by SUNY Poly.

15. SUBJECT TERMS

Memristor, FPGA, CMOS, nanoelectronics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	SAR	77

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
JOSEPH VANNOSTRAND	N/A

PREVIOUS EDITION IS OBSOLETE.

Table of Contents

List of Figures

List of Tables

1.0 Summary

This final technical report summarizes the R&D efforts for the AFRL project "Design Approaches for Efficient Reconfigurable Neuromorphic Systems," referred to here as "RAVENS," which occurred from February 2019 through August 2022. This research project enables future generations of neuromorphic computing systems by (1) leveraging memristor-based analog memory elements for efficient synaptic circuits and (2) considering the efficient integration of these synaptic elements into larger mixed-signal systems. In several respects, this work build on the lessons learned from a prior AFRL funded effort, "Design of a Memristive Dynamic Adaptive Neural Network Array (mrDANNA)," or simply mrDANNA (AFRL Contract Number: FA8750-16-1-0065). The mrDANNA project worked toward a greater understanding for how hafnium-oxide (HfO_2) memristors can be integrated with CMOS using test circuits that could be integrated into a neuromorphic arrays. As an extension of those earlier efforts, RAVENS has focused on (1) leveraging early lessons for more robust synaptic circuit designs, (2) developing design flows, both custom and automated, for integrating large-scale mixed-signal CMOS-memristor neuromorphic systems, (3) demonstrating the benefits of neuromorphic features and parameters at the architectural level, and (4) developing and refining a software stack for a full RAVENS stack that incorporate accurate simulation models for the overall architecture, thus allowing performance and fitness evaluation across a wide variety of applications.

The RAVENS project has been very vertical in how we worked to explore CMOS-memristor neuromorphic system design. At the lower levels, we collaborated with both AFRL and SUNY Polytechnic Institute (SUNY Poly) to refine memristor models that better reflect the behavior observed in the lab. A key component to this low-level device/circuit work was testing HfO_2 memristive devices made available from the mrDANNA project, in conjunction with 65 nm CMOS transistors. Also at the circuit-level, we redesigned the memristor-based synapse considered for mrDANNA, moving from a voltage-controlled "inline" memristive element to a current-controlled synaptic circuit. This current-controlled memristive synapse design element is a key product that has resulted from RAVENS, leading to a recent invention disclosure with hopes of eventually patenting the design. Another circuit-level design and test structure for programmable leak in the CMOS neurons integrated into RAVENS neuromorphic cores.

RAVENS was both about refining circuit-level designs for more robust integration and also pointing toward how these memristive mixed-signal elements can be integrated into full neuromorphic systems. To that end, this work also included the development of standard cells, memory generators, refined I/O cells, interconnect circuitry along with other components and tools necessary for realizing full system-on-chip designs using hybrid CMOS-memristive technologies. This work included necessary add-on features and modifications to the process design kit (PDK) from SUNY Poly to reflect the added layers in the manufacturing process used to realize both memristors and CMOS on the same substrate. Since the SUNY Poly process and associated PDK were being developed as we worked through other design requirements, we also prototyped pure digital components (e.g. scan chain logic, basic memory, RISC-V core logic) using a commercial CMOS process. This commercial tapeout allowed us to test some of the key design elements required for the more digital parts of the RAVENS system, including the automated design flows developed to more efficiently integrated CMOS and memristive components. The culmination of this work was a CMOS-memristive system design that has recently been fabricated by SUNY Poly.

RAVENS, like mrDANNA before it, is a project aimed toward understanding the potential of leveraging transition metal-oxide memristors (HfO_2) in full neuromorphic architectures. It is important to note that the CMOS-memristive manufacturing process is also in early research and development stages, along with development of associated design tools. Thus, it is important that we find ways to emulate our neuromorphic architecture using tools such as field programmable gate arrays (FPGAs) and also develop accurate simulation models that can be used to explore the use of these systems in neuromorphic applications. To this end, the RAVENS architecture was designed, as best as possible, such that it could be implemented as either a full digital system or as a mixed-signal, even memristive system. While we have two potential implementation styles (digital and mixed-signal), the architecture itself is consistent across both. This provides a better representation of the RAVENS system when emulating using FPGAs and using it in applications such as robot control. Note that this more seamless convergence of the mixed-signal and digital systems into one architectural model is an improvement from mrDANNA in the sense that the two implementation styles are much more similar for RAVENS.

At the software level, we have further improved the TENNLab software framework with the development of a base architectural model that has been leveraged for simulation models for both the digital and mixed-signal memristive implementations of RAVENS. This base architecture, a Reduced Instruction Spiking Processor (RISP), is very simple by design, meaning it has the bare minimum features for a neuromorphic system. This allows us to easily explore how the addition of various features at both microarchitectural and architectural levels can impact the performance of the neuromorphic system. Further, various applications has been developed as part of the TENNLab software framework, many of them basic control problems or spatiotemporal classification, that has allowed us to benchmark RAVENS architecture. We have performed studies exploring how each feature added to RISP to implement RAVENS (e.g. leak, context switching) impacts the performance for different applications. The simulation model is also key to our ability to generate neural networks for different applications as the simulations are required for both training and testing.

2.0 Introduction

In recent years, the semiconductor industry has begun to experience a significant slowdown in the performance improvements gained from technology scaling. While this is due in part to the end of Moore's Law scaling, power consumption has also become a critical limiting factor for the level of performance achievable. The research proposed here aims to enable 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. It is worth noting that neuromorphic or neuro-inspired computer architectures are particularly worthwhile given the increasing number of applications requiring techniques and systems that can capture knowledge from the environment, learn from it and respond in accordance. Furthermore, it is important to also consider such neuromorphic systems through the lens of the ubiquitous computer systems, including the internet of things (IoT) paradigm and deployable, autonomous systems. Thus, the proposed Reconfigurable and Very Efficient Neuromorphic System (RAVENS) aims to be an energy-efficient neuromorphic architecture specifically tailored for control and other spatio-temporal applications commonly implemented with resource constrained computer systems.

The ultimate objective of this work is to develop approaches that lead to low-power, reconfigurable, high-efficiency brain-inspired computing hardware for embedded control applications, and other applications dealing with dynamic and spatio-temporal data streams. The application domains that frame this work are specifically related to resource constrained systems with limited size, weight, and power (SWaP), necessitating lightweight approaches for implementing brain-inspired, neuromorphic networks. This project has leveraged prior results and lessons from work on memristive dynamic adaptive neural network arrays (mrDANNA). Building on these earlier efforts, we selected and refined optimally performing neuron/synapse configurations (CMOS-memristor circuits) ready for integration into fully functional memristive neuromorphic processors capable of efficiently implementing command/control, navigation and avoidance, as well as other spatio-temporal data processing applications. Memristors (or “memory resistors”) are nanoscale electronic switching devices, leveraged here in combination with more conventional CMOS transistor technology to maximize circuit density and reduce overall energy consumption. This effort will provide neuromorphic computing power that meets stringent SWaP metrics for the Air Force that can be leveraged for UAVs, aircraft, satellites, and other deployable autonomous systems.

In our previous efforts we have demonstrated functional CMOS-memristor hybrid circuits that are based on hafnium oxide memristors that have multi-level resistance capability. This multi-level resistance capability is paramount for encoding the synaptic weights in neuromorphic circuits. A key challenge that still needs to be met is to improve memristor performance from the standpoint of reliability and power consumption. To this end, our focus at the circuit-level is to design memristor-based circuits that provide better resolution but importantly are more robust and reliable. The analog memristive memory element, used to store synaptic weights, should be consistent from device to device and also consistent cycle to cycle. Our collaborators at SUNY Poly have simultaneously worked to improve device level characteristics, leading to more consistent multi-level resistance levels and improved reliability. This collaborative work, both in terms of improved devices and more robust circuits, has led to a novel memristive synapse based on current-control, more analogous to how we are able to reliably test the devices in the lab.

Leveraging emerging device technologies in still emerging computer architectures, such as neuromorphic, comes with a long list of challenges that we have been working through. Memristive devices have great potential in terms of providing improved physical density and more efficient storage of analog weights, in the context of neuromorphic. However, these memristive devices and circuits need to be implemented such that the technologies are compatible with CMOS, allowing the overall CMOS-memristive system to leverage the best of memristors in some instances and the best of CMOS in others. Seamless integration of CMOS and memristive technologies, while beneficial, has remained a challenging undertaking for the field. From the circuit-level perspective, we require memristive devices that operate with threshold voltage levels within the range of what is acceptable for CMOS and also resistance levels more compatible with the CMOS channel resistance. Many memristive device technologies are not compatible with CMOS in these ways, meaning the threshold are larger than the VDD supply voltages allowable for CMOS or the resistance levels are too small. Similarly, the more experimental memristive devices that exist today still require a forming operation where the forming threshold voltage is larger than that needed to switch a formed device.

The HfO_2 devices available in the SUNY Poly process do tend to provide lower thresholds for SET and RESET operations, even below the 1.2 V supply commonly used for 65 nm CMOS. These threshold levels are lower than what has been reported for other transition metal-oxide devices, providing an opportunity for improved compatibility. However, even HfO_2 requires forming and the forming voltages tend to be around 2.5 V to 3.0 V, larger than what can be delivered through CMOS transistors with 65 nm gate lengths, meaning larger thick oxide transistors must be used in these early stages. In terms of resistance levels, we know from our work on mrDANNA and this project that resistance values for HfO_2 can range anywhere from a few 100 Ohms up to the range of 100 kOhms. There is potential here for high on/off ratios and a wide range of resistance levels that can represent synaptic weights. However, our work in collaboration with SUNY Poly has shown that the reliable operating range is closer to a few kOhms until 10s of kOhms, meaning variability is much lower for these ranges. We also cannot operate memristive devices whose resistance is lower than just 1 kOhm for the simple reason that such resistances are lower than the channel resistance of the CMOS devices, meaning CMOS cannot control switching of very low resistance devices. Thus, for improved CMOS compatibility, it has been important for to design CMOS-memristive synapses where the resistance levels of the memristive memory element can be reliably managed to operate within a range more compatible with what CMOS can handle while mitigating the effects of increased variability as the memristor approaches the highest resistance states. **We accomplish this level of control using a current-controlled, as opposed to voltage-controlled or "inline" memristive synapse. This current-controlled synapse improves reliability, allows for better power management and similarly allows CMOS control of the memristor at DC such that system clock frequencies and duty cycles are more easily adjusted**.

As mentioned, the aim of RAVENS is to consider the ability of integrating neuromorphic systems using hybrid CMOS-memristive technology and to extend that integration to large scale implementations. The CMOS-memristive synapse, managed by a current-controlled mechanism, has provided an example for how memristors can be integrated with CMOS such that all operations (read, SET, RESET, and form) are managed with CMOS circuitry. This is critical, but it is also important that we work toward integrating these mixed-signal memristive circuits into full systems consisting of on the order of millions of devices and more. Here we consider such scalability by developing the means to implement larger scale CMOS-memristive systems. This requires a design flow that tightly manages custom analog circuit design, where CMOS-memristive circuits such as

the synapse are optimized for integration at a higher level. This work has also required refinements to the SUNY Poly PDK so that it includes drawing layers for the memristors and tools for verifying connections in drawn layouts of these circuit blocks. Analog, including CMOS-memristive circuit blocks are also designed with digital I/O and minimal analog references that can't be locally generated. This allows these lower-level mixed-signal blocks to be automatically placed and routed, alongside standard cell based digital circuits. Since the SUNY Poly PDK for this process (CMOS-memristor) is a modified to include memristors, it differs, a standard cell library of basic digital logic gates was developed at UTK to enable automatic place & route. The design flow for this larger scale of integration thus requires careful attention to digital cell placement, custom mixed-signal placement, routing between digital and custom blocks, proper inclusion of volatile memory blocks (i.e. SRAM), and clock routing. All using a PDK and associated technology files that have been modified and augmented to include hybrid CMOS-memristive components. **Through this RAVENS project we have developed the means to integrate CMOS-memristive systems at scale and have made available our libraries, scripts, memory compilers and PDK updates on a shared GitLab repository maintained by SUNY Poly**.

For our CMOS-memristive circuits and systems, most attention is placed on the lower-level circuit designs. Here, we've implemented various test circuits that allow us to form, SET, RESET and read memristors through very simple circuits and work through more complex circuits where CMOS is used for each required operation. The more complex test circuits are CMOS-memristive synapses and eventually full neuromorphic cores (synapses plus neuron) that can be integrated into full systems as noted above. **The test chip includes two reconfigurable neuromorphic arrays (RNAs) constructed using CMOS-memristive cores, both 4x4 or 16 cores in size. Both are synthesized and implemented using automated place & route tools and flows, with the mixed-signal memristive blocks placed in a semi-custom fashion**. One 4x4 RNA is implemented using simpler cores that do not include local analog reference generation, but instead have the reference voltages supplied globally from off-chip connections. The second 4x4 RNA takes integration further by using cores that do include local reference generation, allowing more seamless integration using the automated placement and route tools. Both flavors of memristive RNA are included for some risk mitigation, but each RNA is constructed with many smaller components (e.g. reference generation circuits) that must be tested individually. Small circuit test structures are thus also included to verify and test operations such as reference generation.

A digital RNA has also been developed for RAVENS and an implementation of this digital RNA has been included in the RAVENS test chip. While it does not include memristive elements, the digital RAVENS design does allow us to consider the full architecture and even implement the system using FPGAs. Through such FPGA-based emulation of RAVENS, we are able to explore neuromorphic applications at scale and consider how various architectural features impact system performance. The digital RAVENS implementation, like the digital components of the memristive RAVENS, was implemented using the SystemVerilog hardware description language. Our goal through this process was to make sure the SystemVerilog code was synthesizable both to FPGA and to ASIC/SOC implementations of RAVENS based systems.

One feature that we found was more easily implemented using the digital RAVENS was context switching over a time-division multiplexed network between RNA cores. By context switching, we refer to the ability of one physical neuron to be used as multiple virtual neurons through time-division multiplexing. For the memristive RAVENS implementation, the charge stored on analog neurons is difficult to save without conversions to digital memory which bring

unnecessary overhead. However, we did consider such context switching for digital RAVENS and hope to further explore ways to realize this feature in mixed-signal systems in future efforts. What this context switching allows is virtualization such that the 16 physical cores included in the digital RNA on the test chip can be used like 64 virtual cores, as each neuron has been implemented to execute as up to 4 virtual neurons.

In order to enable full system integration, various other components are needed. Specifically, we have designed and implemented a variety of I/O, interface options, and related peripheral circuits: circuitry for serial peripheral interface (SPI), a universal asynchronous receive transmit (UART) module, a serial scan chain with a module addressing scheme for programming the RNA, a full 32-bit RISC-V processing core for on-chip spike encode/decode, and a digital dot product engine (DPE). In the RAVENS test chip the scan chain is connected to everything as it is used to load neural network configurations and can be used for testing. Also, each RNA and the RISC-V are connected to SPI modules for off-chip communication. However, the three RNAs (2 analog, 1 digital) and the RISC-V are all separate components, ready for individual testing but not fully integrated in silicon. **What is critical in this work is that explore how such components can be integrated efficiently, for example how memristors can be effectively programmed and read from using a serial scan chain**. This scan chain is an important component in our system integration portfolio, but it is also not something that is commonly used to program non-volatile memory elements based on transition metal-oxides (e.g. HfO_2).

FPGA emulation and software simulation using the digital RAVENS architecture has been done to build a better understanding for the performance impact of various neuromorphic features. We have specifically conducted studies using three different control system applications included in the TENNLab software framework: (1) inverted pendulum, (2) an archery game known as "Bowman" and (3) a version of the Asteroids game. In terms of I/O channels and overall neural network size, these three application vary in complexity with the pendulum being the simplest and Asteroids being the most complex. Thus, these applications allow us to benchmark the performance of the RAVENS architecture across different levels of complexity for spatiotemporal control applications. It is important to note that RAVENS, like mrDANNA, is specifically designed to support spiking recurrent neural networks (SRNNs) that can be particularly useful for classifying spatiotemporal data and control. **Relative to the base TENNLab architecture developed for this project, RISP, we found that the various features added (e.g. leak, context switching) do improve application fitness for simpler applications but there is no significant improvement when the application is more complex, thus leading to larger neural networks with denser synaptic connections**. This can likely be explained by limitations of the training algorithm used, evolutionary optimization for neuromorphic systems (EONS). For simpler applications, we believe EONS can explore a larger section of the design space and thus find useful optimizations that leverage the newer RAVENS features. Larger neural networks, on the other hand, are likely too complex for EONS to fully explore the space and optimize much beyond that which is achievable for the base RISP architecture. These findings point to interesting algorithmic considerations that could be explored in future work.

Any complete computing system, including neuromorphic, must include sound methods for some form of software development and execution. For RAVENS, and the mrDANNA project before it, we have developed and refined the TENNLab software framework for (1) efficient training of SRNNs, (2) application development, and (3) neuroprocessor architecture modeling for system simulation. All three of these components work hand-in-hand. For example, training

leverages the objectives specified during application development to evaluate neural networks, simulated using neuroprocessor models, for application fitness. While all three parts work together to train and evaluate neural networks for a particular architecture, such as RAVENS, their modular development allows one part to be modified while others remain the same. For RAVENS, we have not been specifically concerned with the training algorithms themselves, instead opting to use algorithms (usually EONS) already available in the framework. We have continued to develop different applications useful for benchmarking RAVENS, mostly because spatiotemporal applications are often not as widely available in the greater community. So we have developed our own applications that are more suitable to evaluations of these systems as they would be used, for control or similar applications that run on resource constrained platforms. A key project specific development for the TENNLab framework has been the neuroprocessor models used to simulate and train RAVENS neural networks. These neuroprocessor models have been developed in the back-end of the framework using C++ for improved performance. We have actually developed and evaluated three separate models: RISP (the baseline with few features), digital RAVENS (includes context switching), and memristive RAVENS (incorporates constraints due to use of memristors and analog neurons).

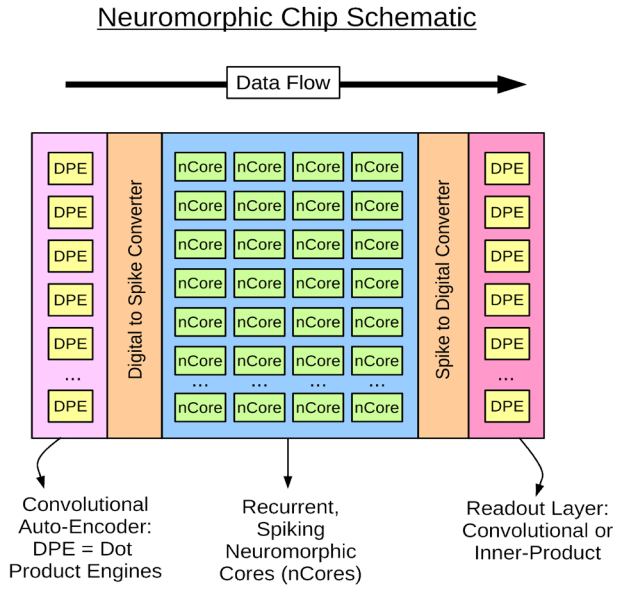

Figure 1. System level view of the proposed reconfigurable neuromorphic array, complete with spiky neuromorphic cores (nCores) and dot product engines (DPEs).

The remainder of this report provides more detail and results for the various items summarized above.

3.0 Methods, Assumptions and Procedures

3.1 Design Kit Updates, Standard Cells, and CMOS-memristor Design Flows

The aim of RAVENS is to consider methods for integrating neuromorphic systems using hybrid CMOS-memristive technology and to extend that integration to large scale implementations. We consider such scalability by developing the means to implement larger scale CMOS-memristive system-on-chip designs that include (1) full-custom analog/memristor-based circuits, (2) semi-custom blocks such as SRAM or memristive RNA, and (3) digital logic blocks synthesized from standard cells.

The design flow requires tightly managed custom analog circuit design, to aid in production of CMOS-memristive circuits optimized for integration at a higher level. This work has also required refinements to the SUNY Poly PDK so that it includes drawing layers for the memristors and tools for verifying connections in drawn layouts of these circuit blocks. For this PDK, we added two new drawing layers, updated from what was used for mrDANNA: (1) the V0 layer for drawing the bottom tungsten contact between the first metal layer and the memristor and (2) the RR layer for drawing the HfO2 memristor layer itself. This leads to a modification in the design kit such that two parallel via stacks between M1 (first metal layer) and M2 (second metal layer) are allowed:

$$M1 – V1 – M2$$

$$M1 – V0 – RR – V1 – M2$$

The first stack (M1-V1-M2) is identical to that of the original IBM 10lpe PDK, representing the via connection between the first and second metal layers. In the updated kit, the original 10lpe stack is maintained such that the M1-V1-M2 is still used and properly recognized as the direct connections between the two metal layers with no memristors. What is new is the additional, parallel via stack (M1-V0-RR-V1-M2) that works as a secondary option for electrically connecting these first two metal layers, but with a memristor (V0-RR) in between. Both the original non-memristive and the recently updated memristive via stacks are recognized for layout editing, tapeout, and verification (DRC and LVS). These memristor-specific layout drawing layers are incorporated into the technology files used for custom layout design (Cadence Virtuoso) and top-down place-and-route (Cadence Innovus). For verification, we have created additional lines of code for design rule verification (DRC) and recognizing a memristive device in the layout when the V0-RR stack is present (Extraction). Furthermore, we've appended the rules files for layout versus schematic (LVS) verification so that we can verify layouts with memristors present against memristor-based circuit schematics. All of these augmented technology files and rules files have been uploaded to the SUNY Poly GitLab to share with collaborators.

Analog, including CMOS-memristive circuit blocks are also designed with digital I/O and minimal analog references that can't be locally generated. This allows these lower-level mixed-signal blocks to be automatically placed and routed, alongside standard cell based digital circuits. Our goal for any custom block (e.g. memristive neuromorphic core) that is to be routed automatically should mostly connect to the top-level system through digital connections (analog behavior is mostly self-contained in the custom block/core). There are some reference voltages that must be provided. For these, it is critical that the layout exchange file (LEF) representation of the custom block properly mark such analog pins as just that, analog. This will allow the routing tool to automatically connect to analog reference voltages without inadvertently placing digital buffers along the routed path. Automated routing also requires that all pins be placed at well

specified intervals such that the pins are on the routing grid. Anything off-grid will be difficult to route, and may even lead to no connection to the wider system. For the CMOS-memristive RAVENS chip, our routing grid is defined by a spacing of 0.34 um, meaning all pins should be some integer factor 0.34 um apart and similarly a factor of 0.34 um from the cell origin in both the X and Y dimensions. The track height for standard cell placement is 4.76 um. For semi-custom and analog/mixed-signal custom blocks, we also tried to make sure these blocks are some factor of 4.76 um tall.

Since the SUNY Poly PDK for this process (CMOS-memristor) is modified to include memristors, a standard cell library of basic digital logic gates was developed at UTK to enable automatic place-and-route. The design flow for this larger scale of integration thus requires careful attention to digital cell placement, custom mixed-signal placement, routing between digital and custom blocks, proper inclusion of volatile memory blocks (i.e. SRAM), and clock routing. All using a PDK and associated technology files that have been modified and augmented to include hybrid CMOS-memristive components. Given the constraints of our CMOS-memristive technology, we specifically developed: (1) a new standard cell library of basic digital logic gates, (2) a library of standard I/O or pad cells for building custom pad frames, and (3) an SRAM compiler with associated libraries for SRAM memory cells and other SRAM peripheral logic.

A summary of all standard cells created and used for this project is provided in Appendix I. This standard cell library, along with all other libraries and tools mentions, has been uploaded to the SUNY Poly GitLab repository so that it is also available to our collaborators.

3.2 Small All-CMOS Test Chip – Verifying Core Digital/CMOS Components

In late July 2020, we submitted a small test chip design that included a few test structures that will help us work out the kinks in our design process before diving into the full CMOS-memristor RAVENS system design. A layout of this chip can be seen in Figure 2. This chip was submitted for fabrication through Muse Semiconductor, using a 65 nm CMOS process that isn't exactly like that used by SUNY Poly but similar in many key ways.

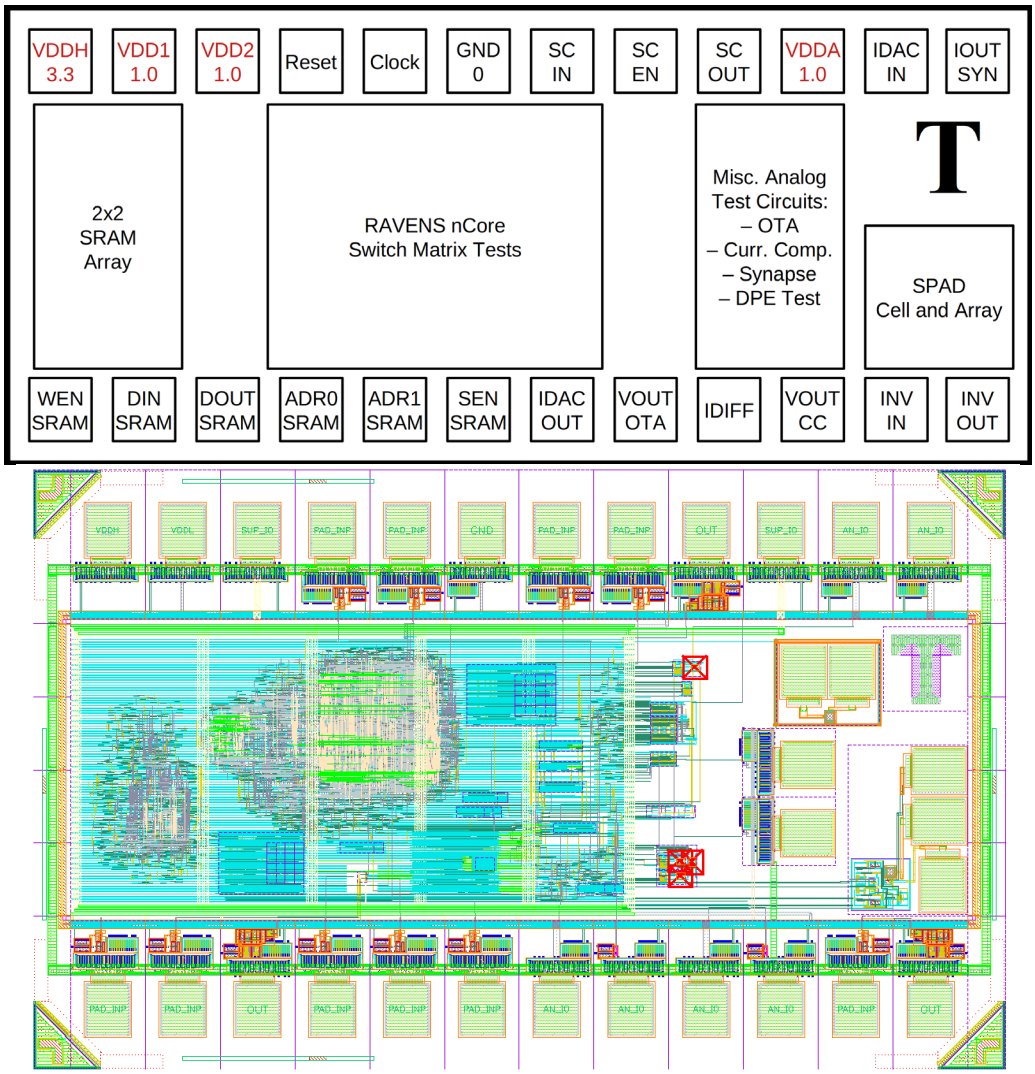

Figure 2. Block diagram (including pinout) of the RAVENS test chip (top) as well as the final layout as submitted for fabrication (bottom).

Due to the outbreak of the COVID-19 and the subsequent lockdown of our campus at the University of Tennessee, Knoxville, we have had to adjust how we work and what elements of the project we focus on while working from home. More specifics regarding the impact of COVID-19 over these last few months are provided at the end of this report. This is mentioned here because one specific response for was to simply focus on design.

Our proposed effort is primarily focused on a somewhat aggressive goal of designing and implementing a full RAVENS system (Figure 1) using the SUNY Poly hybrid CMOS-memristor process. This remains the goal of our project and everything we're working on is to help maximize chances that that CMOS-memristor implementation is successful. To that end, we have also planned an all CMOS implementation of RAVENS to be fabricated through a commercial foundry. The architecture is the same, from a high level, as what will be included on the CMOS-memristor implementation. The all CMOS version is there to (1) help us mitigate risk in the final design with another round of testing for CMOS components and (2) provide an all CMOS version of the system for more "apples to apples" comparisons toward the end of this effort.

The all CMOS RAVENS effort is also a big project with ambitious goals. To further mitigate risk, we've decided to take the budget item set aside for commercial fabrication and split this into

designs: one a small design with CMOS test circuits used to just "get things right" and a second one later in calendar year 2020 for the full "all CMOS" RAVENS system. Plans for taping out the CMOS-memristor version of RAVENS have that happening in early 2021.

The design tapeout through Muse Semiconductor was submitted end of July 2020 with test chips received in October 2020. This was done to provide enough time to test before committing designs for the larger CMOS-memristor RAVENS test chip. Some of the test circuits, including example layouts are shown here to provide reporting on the test structures included on this first iteration of RAVENS.

3.2.1 Current Steering CMOS Dot Product Engine

One differentiating feature of RAVENS, relative to mrDANNA, that is clear in Figure 1 is the inclusion of optional dot product engines (DPEs) that can be used to configure a "read-out layer" in an application based on reservoir computing. In our prior reports, we've shown designs for a hybrid CMOS-memristor crossbar-based DPE. For the memristive version, binary spikes enter the crossbar inputs as voltages that are "multiplied" with memristive weights to produce current representations of weighted inputs. These currents (weighted inputs) are then summed along the columns via KCL to yield a dot product between the vector of spiking inputs and the matrix of weights. A memristive DPE.

To better emulate and compare the memristive DPE using an all CMOS counterpart, we have implemented a CMOS DPE based on current steering (Figure 3).

The primary workhorse of the CMOS DPE is a digital to analog converter (DAC) that includes (1) a binary representation of the weight stored in *N*-bit register, (2) an AND gate for binary multiplication of weight with corresponding input spike, and (3) a series of appropriately sized current mirrors for converting the weighted inputs to current ready for summation along the DPE column. A circuit schematic for the current steering DAC used in the all CMOS DPE is shown in Figure 4.

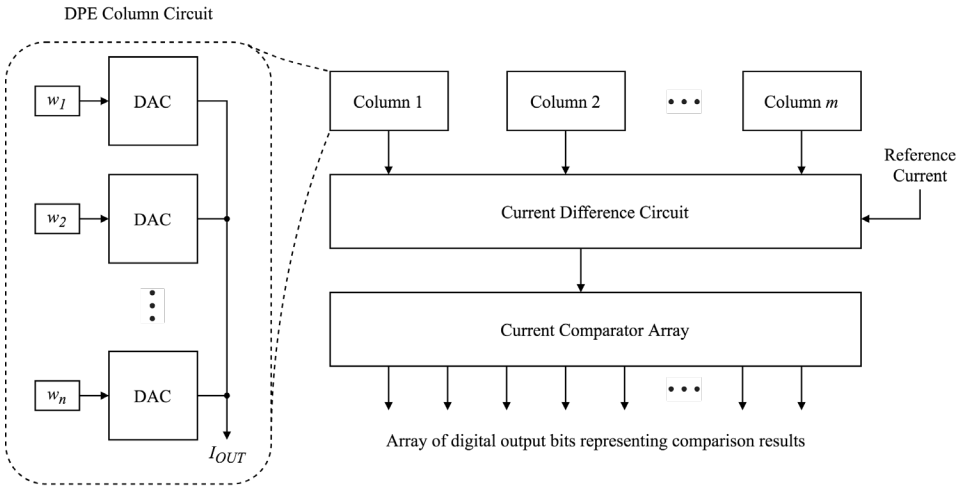

Figure 3. Complete "dot product engine" based on current steering. The idea is to construct a CMOS based dot product engine that more closely emulates the memristive couterpart, using currents for matrix-vector multiplication.

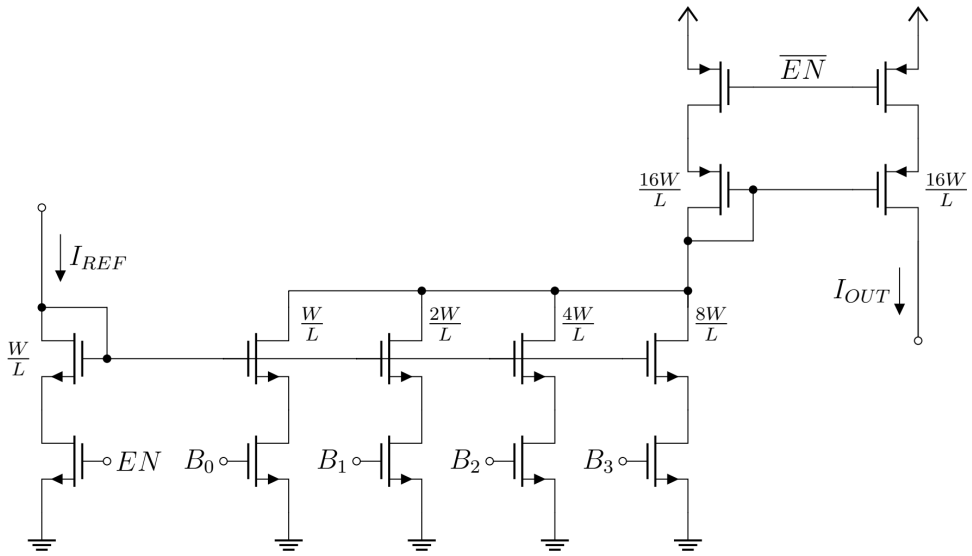

Figure 4. Schematic of current steering DAC used to convert weighted inputs (B_i) to currents ready for summation.

Simulation results for the DAC can be seen in Figure 5. Here it is clear that various digital combinations for B_i (weighted inputs) are properly converted to the desired current levels. Next steps for further verification include corner analysis and Monte Carlo simulations that factor in process variability.

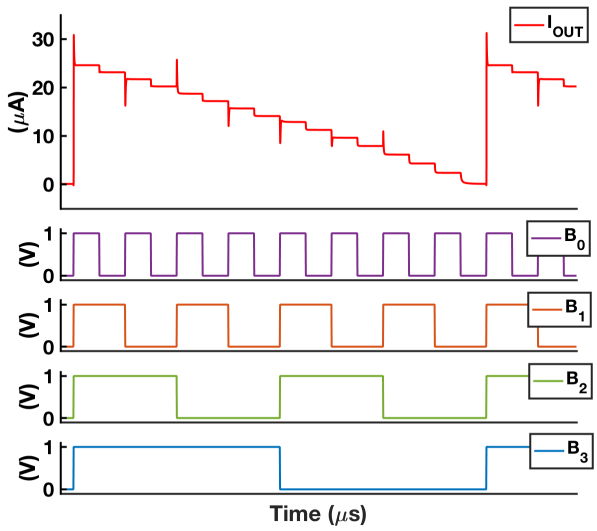

Figure 5. Simulation results for the current steering DAC shown in Figure 4.

The layout for the DAC is shown in Figure 6.

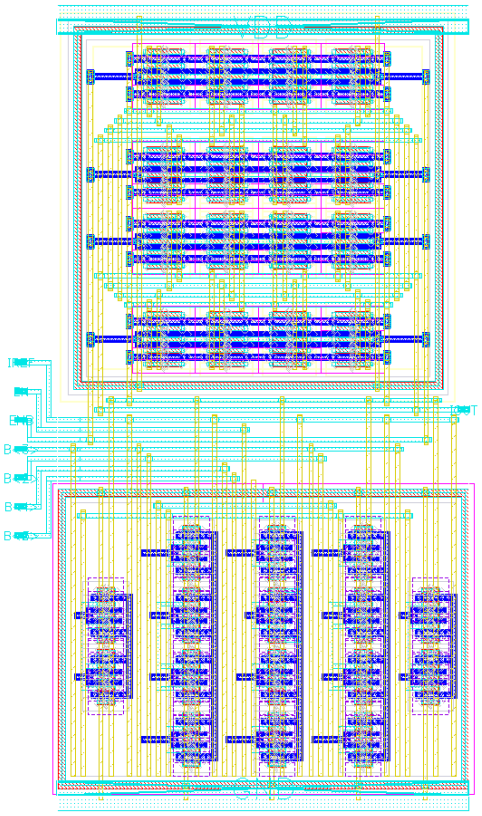

Figure 6. Layout of current steering DAC. A centroid layout matching technique is used for placing transistors in a way that mitigates the impact of mismatch variations.

Simulation results for the complete analog/mixed-signal CMOS DPE can be seen in Figure 7.

3.2.2 Switch Matrix Design for RAVENS Connectivity (FPGA-like)

A core concept in RAVENS is that it is a configurable system, leading to something akin to a "neuromorphic FPGA." As such, the connectivity between neuromorphic cores is meant to be somewhat fixed during run-time though reconfigurable. A common component used for connectivity between configurable logic blocks (CLBs) in an FPGA is the switch matrix, a matrix of switches that define how various CLBs are connected in order to realize customized logic. In the case of RAVENS, the switch matrix connects various neuromorphic cores (nCores) to customize the neural network to the desired application.

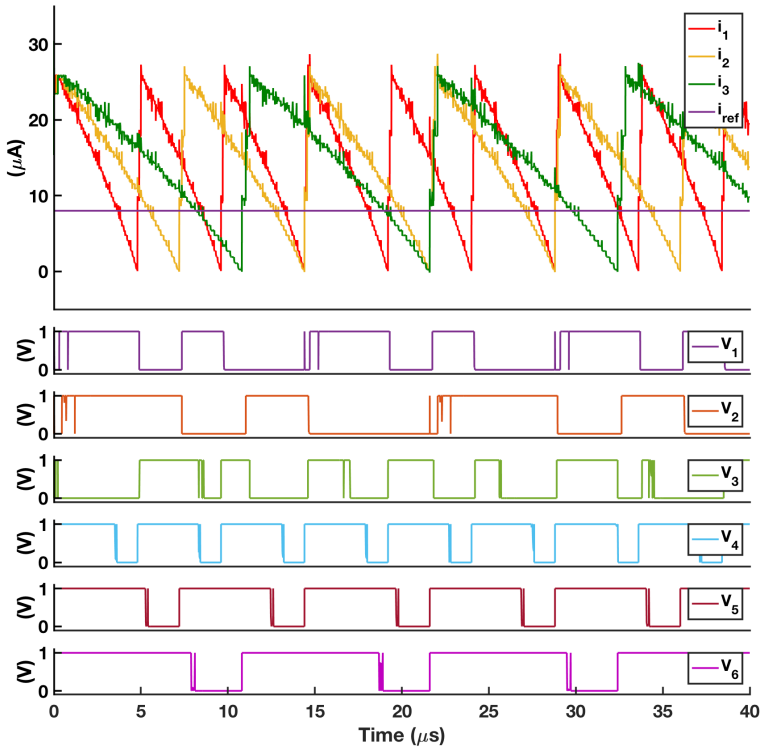

Figure 7. Simulation results for the CMOS DPE test structure.

As seen in Figure 8, each crosspoint in the switch matrix is made up of 6 switches, defining all possible connections between North (N), South (S), East (E), and West (W) input/output lines. Figure 8 specifically shows a switch matrix based on pass transistors. The gates for each pass transistor are driven by a binary memory cell (one configuration bit), implemented using SRAM in the all CMOS implementation but memristive ReRAM in the CMOS-memristor implementation.

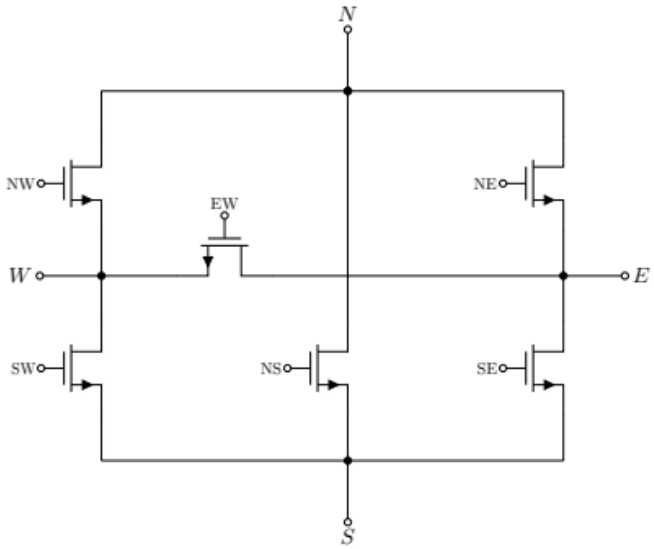

Figure 8. Schematic view of one crosspoint of a pass transistor based switch matrix.

It is worth noting that the switch matrix approach to connectivity is another difference between RAVENS and mrDANNA, which used a nearest neighbor approach to connectivity. With the switch matrix approach a larger number of cores can more easily be connected across chip.

Layouts have been completed for pass transistor and transmission gate implementations of the switch matrices. The pass transistor option is smaller while the transmission gate is more robust in the face of noise and potential glitches. An example layout for a 4x4 array of pass transistor switches (4x4 switch block) can be seen in Figure 9.

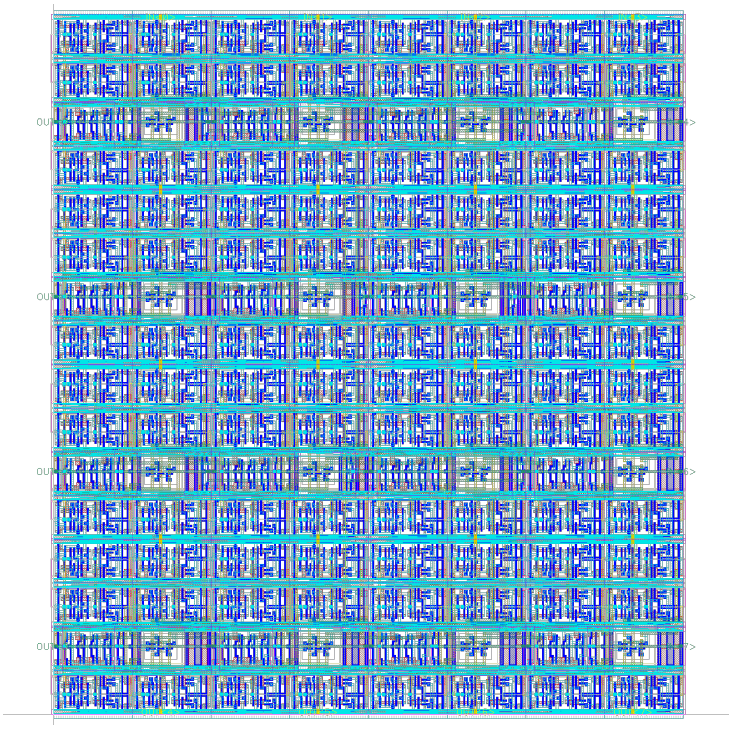

Figure 9. Layout view of a 4x4 pass transistor based switch matrix.

To be clear, the switch matrix in Figure 9 shows the all CMOS layout for the test structure to be fabricated with the test run at the end of July 2020. This all CMOS version uses CMOS registers and potential CMOS SRAM to store configuration bits.

One aim of all of this work is to develop structures that can be reused in the final CMOS-memristor RAVENS design. The switch matrix is something we can fabricate using a commercial foundry now, test, and then refine as needed for the CMOS-memristor implementation. The key difference is use of memristive ReRAM to store configuration bits in the CMOS-memristor version. All other logic is the same such that this commercial run is an early test of what will be included in the final RAVENS implementation.

3.2.3 Other Test Structures for July 2020 RAVENS Tapeout

The upcoming RAVENS test chip will include several other test structures that will help us refine the design for the later, full RAVENS implementations. Test structures included:

- SRAM bit cells and small SRAM array (all UTK designed)
- at least one full neuromorphic core (nCore) – multiple synapses, neuron
- current mode test circuit for testing memristors – uses resistors for first iteration testing

- a Linear Feedback Shift Register – test and verify sequential logic using UTK standard cell library
- a standalone carry save adder (CSA) – key component in refined nCore designed
- scan chain for testing and configuring – similar version on CMOS-memristive chip

3.3 RAVENS Reconfigurable Neuromorphic Array (RNA) Digital Core

When we first targeted this early test chip tapeout, we had envisioned including a slightly modified design of the DANNA 2 design, as a way to evaluate standard cells and other important circuits. However, we have since realized that more work could be done to better align the digital implementation (formerly DANNA) with the memristive implementation (formerly mrDANNA) of our neuromorphic architecture. **In this way, the RAVENS architecture aims to be more unified, with possible full-CMOS digital, full-CMOS analog, and CMOS-memristive mixed-signal implementations of the same architectural specification**.

Thus, we have made quite a few strides updating the RAVENS architecture itself. As illustrated in Figure 1, the full RAVENS architecture includes (1) a input to spike conversion layer, (2) a reconfigurable neuromorphic array (RNA), and (3) a layer of dot product engines. To simplify our implementation and allow for flexibility in spike encoding options, we have decided to use an in-house RISC-V core for the spike encoding layer. As such, the July 2020 RAVENS small chip (CMOS-only) run includes a RISC-V pipeline test circuit. The DPE is a matrix-vector multiplier, with the memristive option essentially based on a memristive crossbar structure.

The spiking RNA is where a lot of updates have occurred, especially with respect to our digital implementation that can be synthesized to an FPGA or to an ASIC/SOC. Like mrDANNA, the RNA for RAVENS consists of configurable neuromorphic cores, or nCores. All nCores are in turn connected via an FPGA-like connectivity fabric built from switch matrices. To better emulate the high-level functionality of the memristive implementation, the digital nCore now includes an accumulator that performs the multiply-and-accumulate function in a single cycle. Where this is done digitally in the digital RNA, the analog implementation works by leveraging Ohm's law for an analog multiply, followed by Kirchoff's law for summing or accumulating analog currents. At a high level, both implementations are identical since they are performed in a single cycle.

A feature that was part of DANNA but not mrDANNA was charge injection, which allows values to be injected directly into a neuron accumulator/integrator as charge. Realizing the power of this particular feature for faster processing of input data from active sensors, we included charge injection in the RAVENS RNA design, for all implementations. For the memristive implementation, we have two options for charge injection depending on which neuron is implemented. For a purely analog integrate-and-fire neuron, charge injection is possible but likely requires a DAC and possibly some sample and hold circuit to "inject" an analog voltage onto the neuron's feedback capacitor. For a digital neuron, where a simple ADC is included between synapse array and neuron accumulator, "charge" can be simply added to the accumulator as it will

store a digital value. Regardless of implementation, we feel that charge injection is an important and distinctive feature included in the RAVENS RNA architecture.

Figure 6 illustrates the RNA nCore design, as implemented on the July 2020 RAVENS test chip. It is worth pointing out that the nCore is reconfigurable, meaning initial weights and thresholds can be tailored to a particular application. Further, some key parameters for the accumulator include: configurable "leak" for a leaky integrate-and-fire response, configurable absolute and relative refractory periods.

A final feature now included in the RNA design is the virtual neuron. This is particularly useful for digital hardware where the neuron implementation may be somewhat costly in terms of area. By virtual neuron, we mean that each physical neuron in a given core is allowed to alternate between multiple virtual neurons, swapping parameters on alternate clock cycles in a form of time division multiplexing. The same TDM approach is applied within the RNA connectivity network, where the network alternates between different fixed connections over different clock cycles. Altogether, the design allows any virtual neuron to connect to any other, regardless of which TDM cycle is used for each one.

Figure 10 provides a high-level illustration of the digital RAVENS nCore design. This new RNA design is a key element in the RAVENS system design. The memristive components in the CMOS-memristive RAVENS implementation would be the "synapse units" shown in Figure 10.

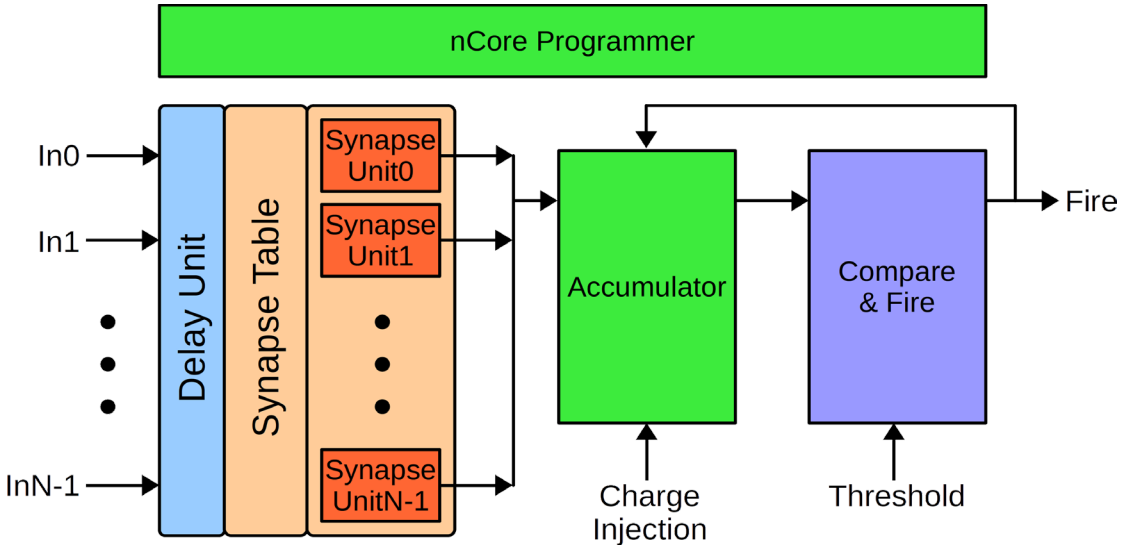

Figure 10. High level view of the RAVENS RNA nCore architecture, complete with configurable accumulator, delay, synapses and compare & fire units.

A more detailed view of the digital nCore design is shown in Figure 11, where the various multiplexing operations are explicitly called out. Some logic, such as the Compare and Fire of the digital neuron, is shared for the entire core. Context switching using the multiplixers shown allows multiple states of the synapses and stored charges of virtual neurons to be simultaneously

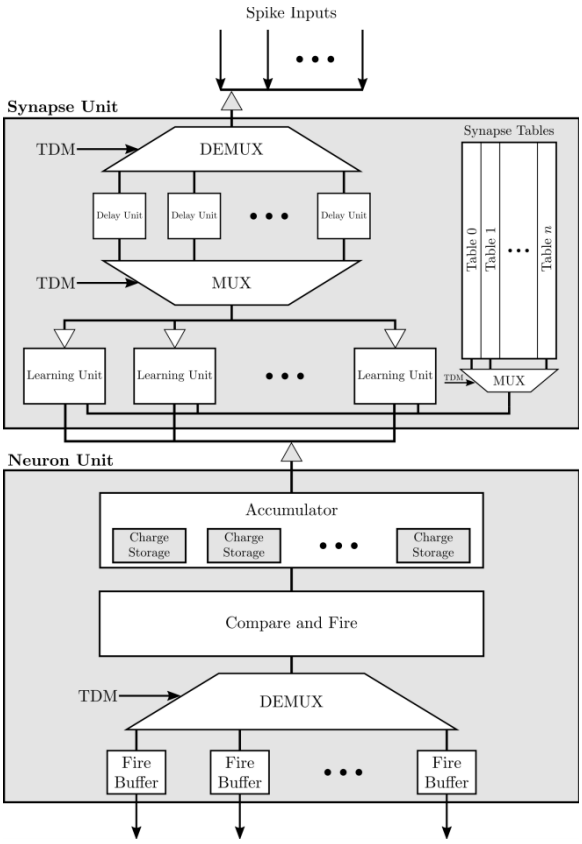

Figure 11. A more detailed look at the digital RAVENS core design, including elements for synaptic delay weights and multiplexing logic used for TDM based context switching. TDM allows logic such as the Compare and Fire portion of the neuron to be shared across multiple virtual neurons.

stored in memory while other virtual neurons and their corresponding synapses are in operation. TDM is used here to switch between different synapse/neuron sets (virtual neurons) are used during different clocking phases. For example, an implementation with two TDM cycles/phases might switch to using virtual neuron 1 on the odd clock cycles and then switch to neuron 2 on all even clock cycles. At a higher level in the system, counters are used to keep track of what TDM cycle the system is currently on, thus driving the TDM select lines shown in Figure 11.

A digital version of the RAVENS RNA is included in the test chips, including the one fabricated by SUNY Poly. This digital RNA is interconnected using a CMOS-based switch fabric, like that of a modern FPGA. The memristive RNA is also integrated onto this same chip in two different flavors: one with reference voltages driven from off-chip and a second where we generate the reference voltages on-chip. Ideally, we'd like to use the reference generator circuits but wanted a failsafe in case this option did not work as expected. The memristive RNA in constructed using a multi-stage network that simplifies connectivity implementation but still allows for a larger number of potential connections between cores, allowing for the definition of various recurrent network topologies.

3.4 The Memristive Neuromorphic Core with Current-Controlled Synapses

Figure 12 shows the circuit design of a new memristive synapse that our group has been working on for some time. A test circuit for the CMOS parts of this circuit was included in our July 2020 CMOS run. While that was useful to help work through the necessary functionality of the circuit, the updates design with memristors and 10LPe CMOS for forming, programming and learning is included in our memristive reconfigurable neuromorphic array (mRNA) that we are sending to SUNY Poly for fabrication. This new synapse is the centerpiece of our new RAVENS design, as it builds on lessons learned from mrDANNA and provides a more reliable mechanism for realizing synaptic weights with the HfO_2 memristors as they exist today. We continue to explore the impact of other memristive materials, including different potential synaptic circuits. However, the design shown in Figure 12 is designed specifically for the HfO_2 memristors as they exist today.

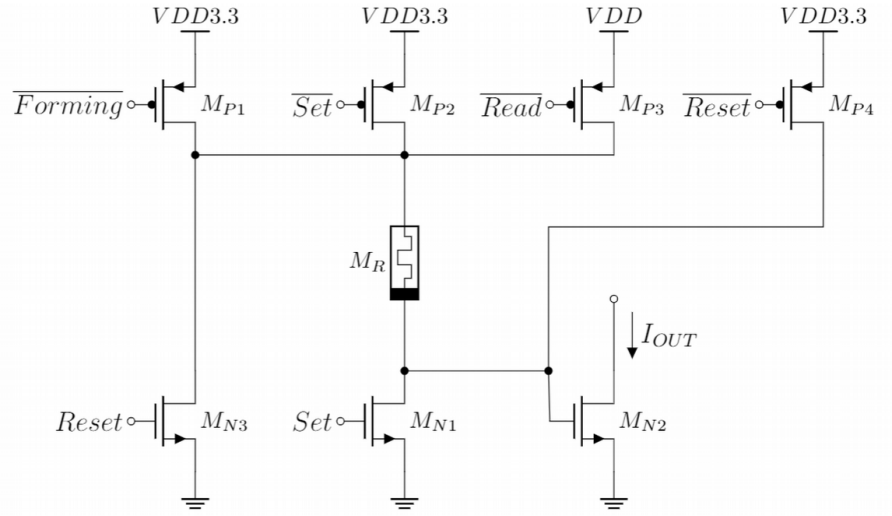

Figure 12. New CMOS/memristor synapse schematic, illustrating all circuitry needed to manage forming, set and reset. The forming and set operations are managed by current compliance through the M_{N1} transistor. This technique for programming memristors has proven most reliable based on prior testing.

In contrast to the circuit shown in Figure 12, our mrDANNA synapse options were based on the idea of driving well timed pulses, or spikes, across the memristor to realize changes in device resistance (forming, set, nudges for learning). The mrDANNA design was dubbed a "twin memristor synapse" by our group as it incorporated two memristors to realize the synapse. One memristor would drive positive current into the post-synaptic neuron, the other pulled current away from the neuron. In this way, the two mrDANNA synapses worked together to realize either positive or negative weights. The new design consists of one memristor as opposed to two. Further, much of the CMOS needed to form the synapse can be shared in either 1D or 2D array structures. However, minimizing the dependence of the CMOS circuitry continues to be an aspect of our research. It is worth noting that the CMOS is required for either the Figure 12 synapse or the "twin memristor" synapse as larger 3.3V voltages are required for forming.

The circuit in Figure 12 works by using the M_{N1} transistor, driven by *Set* on its gate, for providing a clear current compliance while the memristor is switching. Considering transistors M_{P2} and M_{N2}, together with the memristor M_R, this design could be considered a sort of 2T1R

circuit. Again, some of the other transistors can be shared. The *Set* voltage is applied such that M_{N4} remains in saturation during a SET or FORMING operation. Transistors MN1, MP1, and MP2 are all sized such that MN4 will remain in saturation during these operations. If the memristor is in an HRS state, then most of the voltage drop during the programming or learning event will be across the memristor itself. However, throughout the process, the current will remain roughly constant since MN4 is in saturation. As the voltage drop across the memristor is held high, the resistance changes due to the memristive properties of the device, and more voltage is now across MN4. Since current is constant, this causes the process to stabelize at a near low resistance (LRS) state, but not the true lowest LRS state. What near LRS state is achieved is a function of the Set voltage, providing our programming/training mechanism.

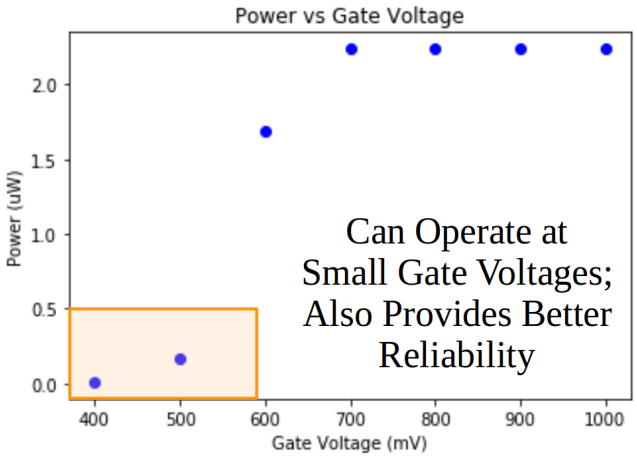

Figure 13. Power consumption of the current compliance memristive synapse as a function of gate voltage.

We have explored the power consumption of this new memristive synapse. Since the current can be well controlled, the power through the circuit is also controllable, especially during a read operation. Since power is always quadractic with respect to voltage, our goal is to keep voltage as low as possible. Further, we find that this circuit shows a clear drop off in power consumption for smaller voltages, as shown in Figure 13. This drop-off occurs right around the transistor threshold, which makes sense. However, it shows that we do not need to operate in an extremely sub-threshold region to save power. In fact, just above and certainly just below threshold leads to a significant savings in power for the read/inference operation.

We see this near-threshold operation as a key method to save power without the need to run too slow. The RAVENS synapse design is expected to operate in the range of a few MHz to a few 10's of MHz, in contrast with sub-threshold or "biologically plausible" alternatives that operate in the kHz or lower range. We see this speed range as an opportunity to fill a need for neuromorphic systems that are particularly well-suited for deployable embedded devices, in the same vein as IoT style devices.

Another key advantage of the synapse shown in Figure 12 is reliability. More specifically, our mrDANNA research, in collaboration with SUNY Poly has found that the devices are more reliable during a read operation when the voltage is a few 100 mV lower than the threshold. This makes sense as the rate of memristive change is not an ideal abrupt switch at the threshold point. Rather,

as one would expect with any memristive system, a gradual almost linearly increasing rate of resistance change is experienced as the voltage is increased toward the threshold. Once the applied voltage hits the threshold, the rate of resistance change increases exponentially. What this means is that there is some read disturbance that we can expect from these devices, even when the read voltage is small. Applying lower voltages will thus not only save power but also minimize the impact of read disturbance over a long period of operation.

3.4.1 Read-Out Circuit – Sensing Positive/Negative Weights to Drive Neuron Input

Considering the second stage of the current-controlled synaptic circuit in Figure 12, the synapse sinks an output current (Iout) that must be directed toward the input of an integrate-and-fire neuron. Per the architectural model used for RAVENS, and in this instance also mrDANNA, synapses can store both positive and negative weights. From the perspective of this output current, that means either driving a positive current into the neuron input or driving a negative current, analogous to pulling current away from the neuron. Thus, we need a synaptic output buffer that determines the sign of this current and can drive it from synapse to neuron.

The circuit shown in the left of Figure 14 is a synaptic buffer, sensing an input current through Mp1 and Mp2 and driving a corresponding output by comparing this input to a negative current mode (NCM) reference into the bottom of the circuit. This NCM reference needs to be chosen such that it refers to a synaptic weight of zero, meaning the synaptic buffer will drive 0 A of current when the input current matches this reference. If the input current is less than the NCM reference, then the output current is negative. On the other hand, an input current higher than NCM will lead to a positive output from this synaptic buffer. Thus, the synaptic buffer (left of Figure 14) performs two functions: it steps down the current level to a magnitude that can efficiently be read by the neuron and it also allows the synapse to store either a positive or negative weight in the form of its resistance value.

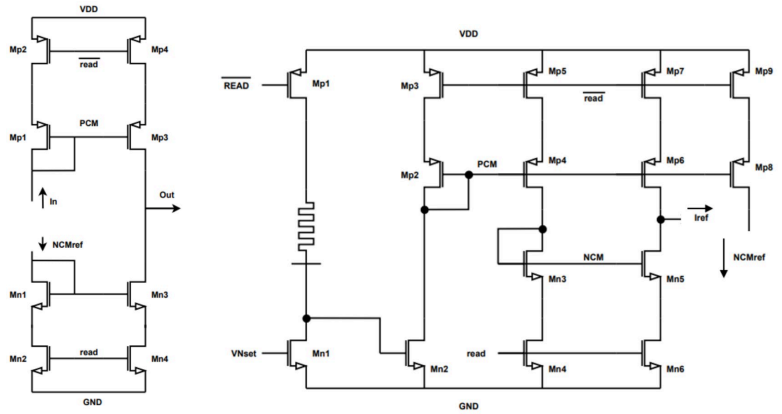

Figure 14. (Left) Synaptic read-out buffer that takes Iout from the synaptic element in Figure 12 and drives an output (Out) current that is either positive or negative into the neuron input. Whether the output is deemed positive or negative is determined by a negative current mode (NCM) reference. (Right) Memristor-based circuit for generating the reference (NCMref), assuming the memristance is at value representing a synaptic weight of zero.

The NCM reference can be driven from an off-chip pin, meaning the current that represents zero weight would need to be generated off-chip and steered into every synapse in the system. One challenge is driving or steering this reference throughout the RNA such that the reference current is identically reproduced for every synapse. Another issue is that manufacturing variability is such that the required NCM for the memristive synapse will vary as the memristance varies. If the memristors in a particular run see a shift in the resistance levels, either positive or negative, it would be difficult to know exactly what the NCM reference will need to be for that particular run. Thus, it is better to generate this NCM locally on-chip with a circuit like that shown in the right side of Figure 14. Here, a sample memristor is included that should be initially SET to the resistance value that should represent a weight of zero. For example, if we're using a resistance range of 5kOhm to 15kOhm to represent our weights with HfO2 memristors, we should initialize the sample memristor to 10kOhm, the midpoint of our resistance range. The NCM reference generation circuit then generates a current representation of this 10kOhm. Importantly, we do not need an NCM reference generation circuit for every synapse. We can locally distribute the reference across multiple synapses to save area and power due to reference generation. For the cores in the RAVENS test chip, one NCM reference circuit is included for every 16 synapse, or one per core.

3.4.2 Reliable Current-Control for Programming and Read

The Set input in current-controlled synapse (Figure 12) is to be used for three different operations: Forming an uninitialized memristor, Setting (programming) a memristor into a desired resistance state, and reading the resistance/weight of the synaptic memory element. All three of these operations require very different voltages in order for the circuit to work properly. For forming, a large current (high Set voltage) on the order 100 uA must be driven through the memristor. This is a one time operation, but the Set input must allow for this forming event. Likewise, the SET operation of driving the resistance down to a specific near low resistance state (LRS) will require a larger current than what is used to read. For SET, the Mn1 transistor gate in Figure 12 (input Set) should be in the range 900 mV to 1.1 V to drive a current large enough to

program the device in the range of something like 3kOhm to 15kOhm. For read, this current should be much smaller, and it is chosen to be about 1 uA in the circuits used for RAVENS. For this level of read current, the Set voltage must be 600 mV. Thus, a circuit is needed to select what operation drives the synapse operation (form, set/program, read).

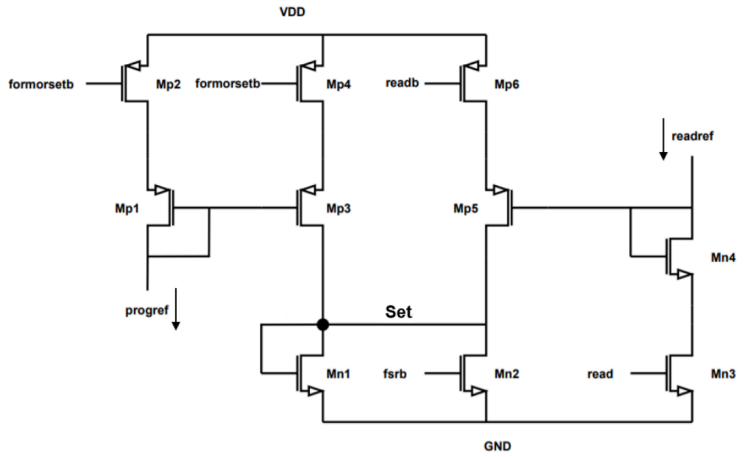

Figure 15. Current-control circuit for selecting between the operations form, set/program and read. Current references for these different operations must also be supplied for proper operation.

The circuit in Figure 15 also requires references, either from off-chip or generated on-chip. Assuming our circuit includes the NCM reference generation circuit shown in Figure 14, we can also use this to help generate the readref signal on-chip. Notice that the circuit in the left side of Figure 14 includes an output labeled Iref. This second output reference is particularly useful for the read reference current generation, but we need to compare it to the mid-rail voltage supplied to our chip. For RAVENS, VDD for 65 nm CMOS is 1.2 V with ground anchored at 0 V. Thus, the mid-rail voltage is 0.6 V, an extra supply voltage that is so critical to the operation of our synapses and neurons that it is provided as an additional supply voltage. The circuit shown in Figure 16 essentially compares the NCM output Iref with the mid-rail supply and generates a readref reference current adequate for efficient reading. When supplying this read reference current to the right side of the circuit shown in Figure 15, that circuit is well situated for driving a reliable read signal during a normal read operation, meaning a spike has arrived. Just as is the case for the NCM reference generation circuit (Figure 14), the circuit shown in Figure 16 can be shared across multiple synapses. For RAVENS, we have included one of these reference generation circuits for every 16 synapses.

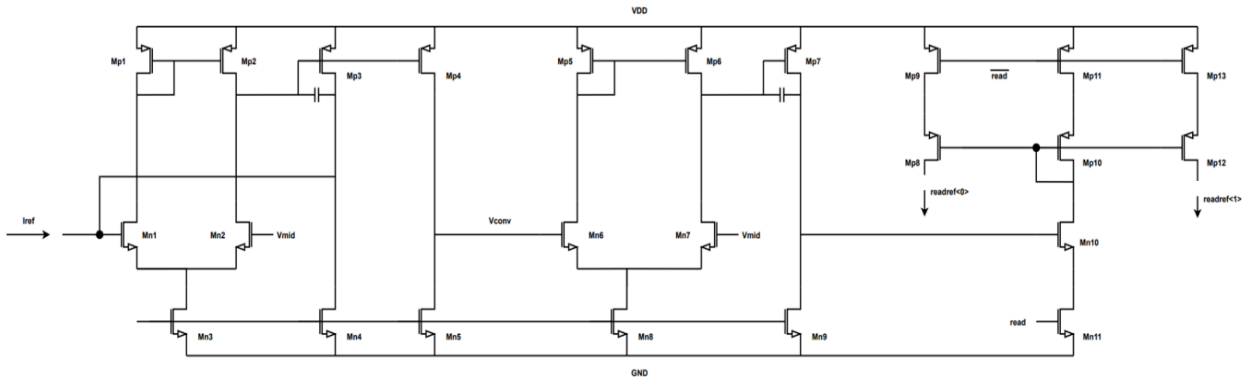

Figure 16. On-chip reference generation for reliable read operation using the current-controlled synapse.

In addition to forming and reading, the Set signal driving the MN1 transistor in the current-controlled synapse (Figure 12) must allow for programming. Here, programming is actually a two step process due to the variability of the high resistance state (HRS) and relative stability of anything near LRS. The near-LRS states, unlike their near-HRS counterparts, can also provide relatively linear operation across the resistance ranges used. For RAVENS, using the HfO2 memrsitive synapse, the near-LRS resistance range used is about 3kOhm to 15kOhm. To program the device into any value within this range, the memristor is first RESET to HRS and then SET to the desired resistance by controlling the current. For RESET, we simply drive the device with as much current as possible in the reverse direction. The RESET operation is digital using the two RESET transistors shown around the memristive synapse. For the second step, the SET operation, we must now drive a specific current through the memristor by controlling the Set gate voltage. To achieve something in the range 3kOhm to 15kOhm, we've found that the Set voltage to produce the desired current should be in the range 900 mV to 1.1 V, higher than the 600 mV used to read.

During a programming event (SET), a current-steering DAC (Figure 17) is used to convert a 3-bit digital value into the corresponding current (progref) used to program the memristive synapse. This progref output node would be connected to the corresponding node in the left side of the circuit shown in Figure 15, used for managing form, set, and read. The DAC works by supplying some current based on the significance of each one bit in the 3-bit input. The currents from each corresponding bit (Mp3, Mp5, and Mp7) are summed together and then mirrored to drive the output progref. In the RAVENS RNA test circuits, the 3-bit digital input comes from a register loaded by the scan-chain during a configuration event. Since the scan-chain is serial, and configuration is expected to take some time, the same DAC is alternatively used for all 16 synapses in a neuromorphic core.

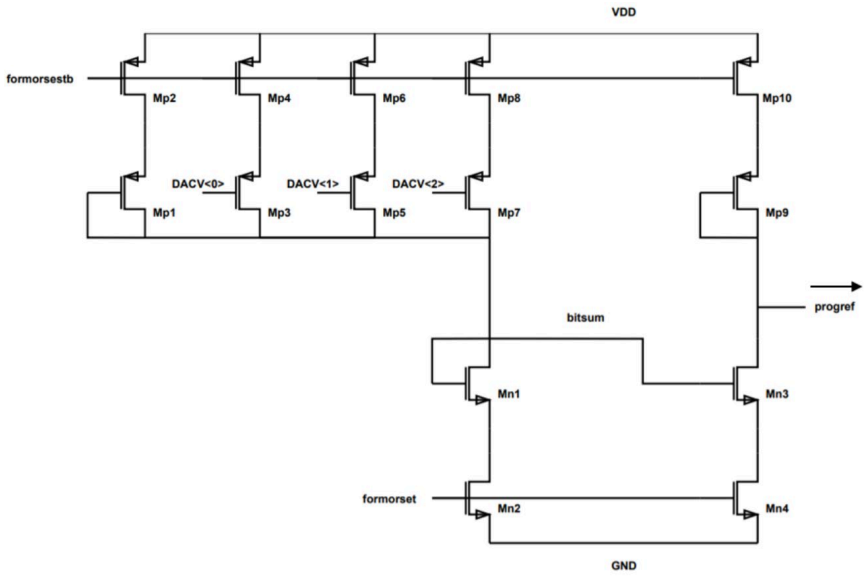

Figure 17. A current-steering DAC for programming a synapse to a desired resistance/weight. The 3-bit DAC input is digital and driven into a 3-bit register using the scan-chain during (re)configuration.

3.4.3 STDP for Online Learning with Current-Controlled Synapses

Online learning (meaning run-time learning on-chip) continues to be a goal for neuromorphic computing. We find that online learning can be beneficial in some cases, but is most useful with the neural network topology is dense and even layered. Furthermore, early work has suggested that online learning based on spike-time-dependent-plasticity (STDP) is most useful for the last layer in a deep neural network or the read-out layer of a reservoir computer. Here, STDP can help refine the classification function that occurs in such densely connected layers. What still isn't clear is how much STDP benefits neurons within sparse SRNNs. Likely, for most applications at least, STDP is not useful everywhere and is especially not useful within sparse recurrent neural networks. There are two points here: STDP is not useful everywhere in a neural network but it can be very beneficial for learning in real-time if it is applied to the right set of neurons. Where to include STDP remains an open question, but the design of STDP circuits for any neuromorphic system certainly remains worthwhile.

For the current-controlled memristive synapse, STDP is not as straightforward as it would be for an inline, voltage-controlled synapse. This is why mrDANNA opted for the inline design. However, an important lesson from mrDANNA is that HfO_2 memristors experience switching at much faster rates in one direction that they do in the other. Thus, the fidelity required to manage STDP in a system built using inline memristors (e.g. basic 1T1R crossbar arrangement) is actually impractical. Thus, we are now focused on a current-controlled memristive synapse, not inline, and are faced with the challenge of how to implement STDP for this system.

Figure 18 illustrated the logic level design of an STDP control circuit that can be used for online learning with current-controlled synapses. This circuit consists of a series of sample-and-hold circuits, used to read the value of the memristor at various clock cycles. The analog sample-

and-hold circuit is leaky, and the leakage rate is leveraged as a measure of time from the last spiking event on the input of the synapse. This leaky stored voltage of prior spiking events can be subtracted from a voltage representative of an output spiking event to determine how much a memristive synapse must be modified to implement the STDP rule. As can be seen from Figure 18, the STDP circuit is costly in terms of area and power. This provides another reason one should be selective regarding where this circuit is included and will likely not want to include it everywere.

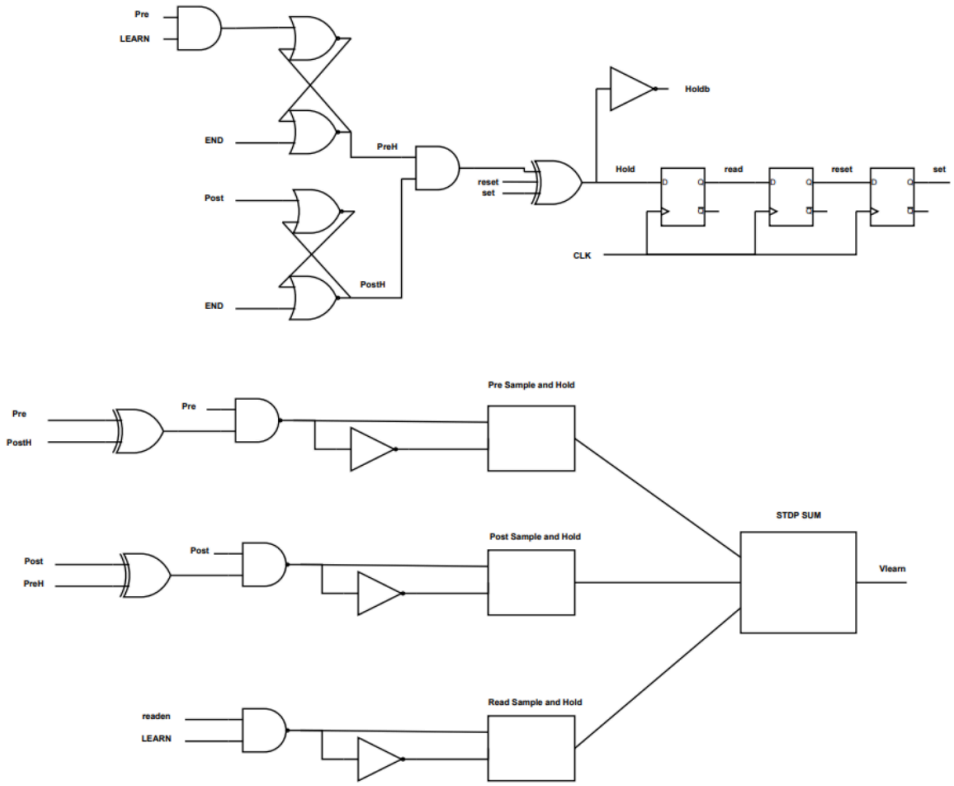

Figure 18. Components required for implementing spike-timing-dependent-plasticity STDP using the current-controlled memristive synapse design presented here.

Shown in Figure 19 and 20 are examples of potentiation and depression events, respectively, using the STDP circuit shown in Figure 20. An important look at the performance of any STDP implementation is the relationship between change in synaptic weight and time differences between output and input spike events. If the output spike event comes after an input spike, the corresponding synapse will potentiate (weight increase) by an amount corresponding to the time between the spikes. Depression likewise occurs for output spikes that occur before input spiking event, where the weight is reduced proportional to the time between spikes. This behavior is shown for the CMOS-memristor based synapse in Figure 21.

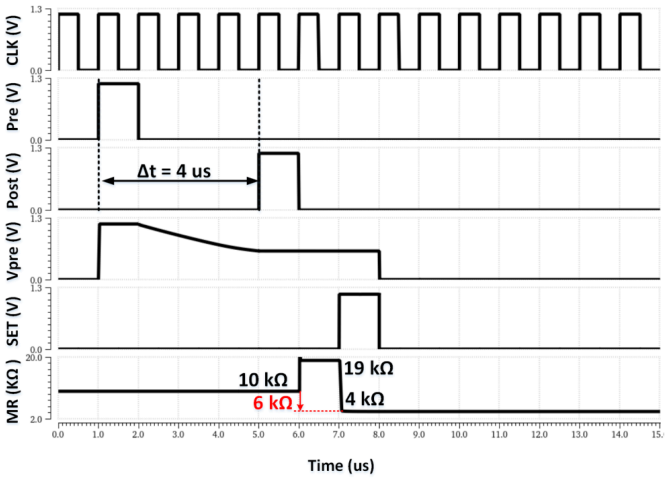

Figure 19. Simulation showing the voltage time relationships for a potentiation.

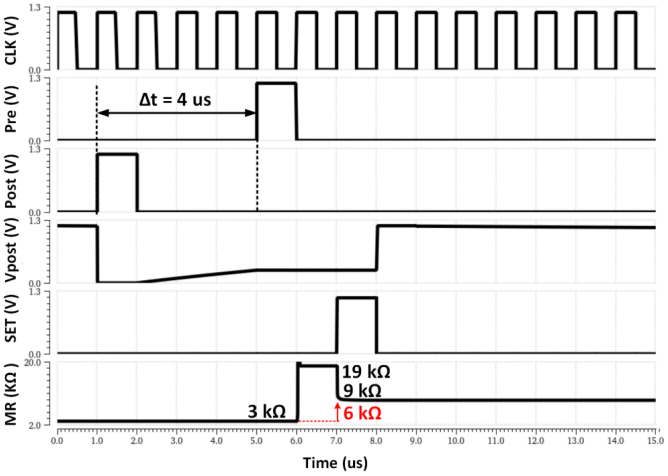

Figure 20. Simulation showing the voltage time relationships for a depression.

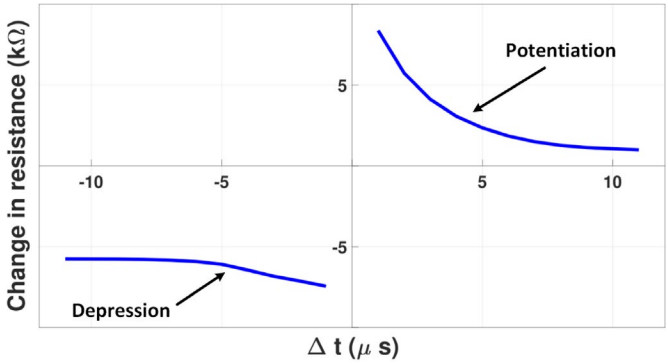

Figure 21. Plot showing change in memristor resistance (synaptic weight) as a function of the time difference between output and input spike events (Δt).

3.5 Memristive RAVENS RNA Design and Layout

Figure 22 shows our "base" memristive RNA (mRNA) layout that has been included on the CMOS-memristor test chip fabricated by SUNY Poly. What we mean by base is that this version does not include a reference generation circuit for determining the middle voltage point which in

turn determines the zero point of our memristive synapse (Figure 12). Instead, the base implementation has all references driven from off-chip, providing some tuning capability of these voltages during testing. We've also included an identical (4x4) memristive RNA on the test chip that does include reference generation, based on the NCM and readref feedback mechanisms described in the prior section. These feedback circuits are expected to be robust in the face of noise. Further, the base version of the mRNA does include a leaky integrate-and-fire neuron, based on our mrDANNA work, but the leak rate is not programmable in the base implementation. We have also implemented an experimental circuit that provides a configurable leak rate for the LIF neuron. This implementation is included on the test chip as a standalone core connected to our in-house scan chain for testing.

The full mRNA shown in Figure 22 consists of 16 cores, each with 16 synapses and a single LIF neuron, centered in the core layout. This is the exact same configuration used for the version

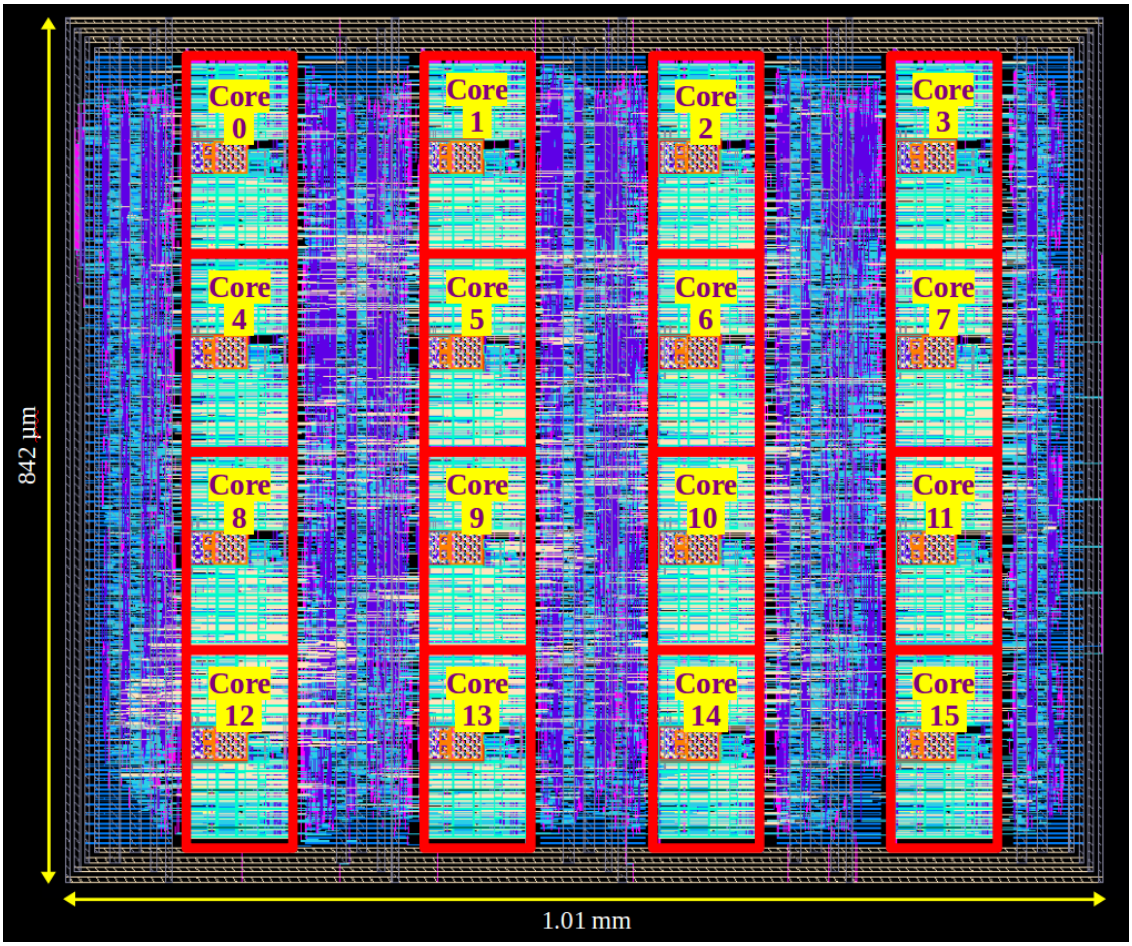

Figure 22. Layout of the 16 core RAVENS memristive Reconfigurable Neuromorphic Array (RNA). Each neuromorphic core (nCore) has 16 synaptic inputs, each implemented with the CMOS/memristive synapse shown in Figure 2. On-chip communication is implemented as a programmable (set once) multi-stage network and off-chip communication is through an in-house scan chain for programming and a serial peripheral interface (SPI).

that includes a feedback reference generation circuit. On-chip mRNA communications is accomplished with a 16x256 multi-stage network, providing a large amount of connectivity but with few crosspoints (thus, little memory). The on-chip communications fabric is simple, in that connections are defined upon programming the device via the scan chain. Each switching stage in the network will then have a fixed configuration, defining neural network topology, until the device is later reconfigured for a different application. Such an FPGA-like approach to connectivity allows spikes to be routed as spikes through this neuromorphic cluster or RNA. **There is no address encoding or packetization of spikes as they traverse the RNA connectivity network. There are simply pulses or binary spikes**.

For off-chip communication with the mRNA, we have integrated a standard serial peripheral interface (SPI) with both master and slave communication capabilities. For testing, spikes will be driven in and out of the chip through SPI, courtesy a FIFO that will serially load spike trains on the inputs, to be sequentially driven through the mRNA, and another FIFO that receives spikes from the outputs and then drives them off-chip. The in-house scan chain, based on a design that resulted from our mrDANNA project, is used to program the memristors and the RNA multistage network switches that determine neural network topology.

3.6 Memristive Switch Options for Dense On-Chip Reconfigurable Connectivity

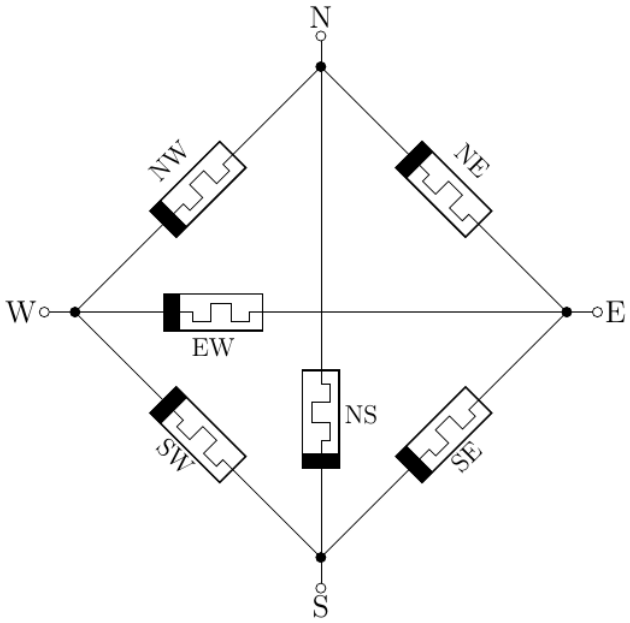

Figure 23. Ideal schematic view of memristor-based switch block.

Another potential use of memristors in neuromorphic systems that we've begun to explore through this RAVENS research is as a switching element in the interconnect that glues together multiple neuromorphic cores. For our RAVENS switch fabric we have considered an FPGA-like architecture, built from switch blocks that allow connections in any of the North, South, East and West directions. Switch blocks that can connect vertical (N/S) to vertical (S/N) tracks and also vertical (N/S) to horizontal (E/W) tracks. In addition to switch blocks, this FPGA-like connectivity

includes connection blocks that work more like a crossbar in the sense that vertical (N/S) can be connected to horizontal (E/W) tracks, but a connection block won't connect across vertically or horizontally.

Memristors are very promising candidates for the switching elements that build up such FPGA-like connectivity. Not only do the hold the potential for more dense connectivity, due to the fact that they are small two-terminal switches, but we find that they are very promising in terms of potential power savings. This is critical as interconnect, not logic, is often the main culprit for power dissipation in modern integrated circuits. Our inline memristor based switch block design is shown in Figure 23.

The key to how the Figure 23 switch block can save power is the fact that the memristor acts as both the memory element that stores the configuration and simultaneously the switch that connects two I/O points. This is distinctly different from an all-CMOS implementation where a separate memory element (typically SRAM) is needed for each switch to store the configuration. Typically, the outputs of the SRAM cells drive their respective pass transistors, which are the elements that connect the I/O points. In the inline memristor switch block (Figure 23), these functions are all performed by the two-terminal memristor.

Figure 24 shows the energy of a comparable SRAM-based switch block and several such blocks implemented with the inline memristive switch cell shown in Figure 23. A variety of memristor material stacks are considered. The HfO_2 could potentially be competitive with pure CMOS implementations when the array size is large. However, HfO_2 has a relatively low LRS and also a somewhat small on-off ratio, as compared to other memristor materials. Thus, the real

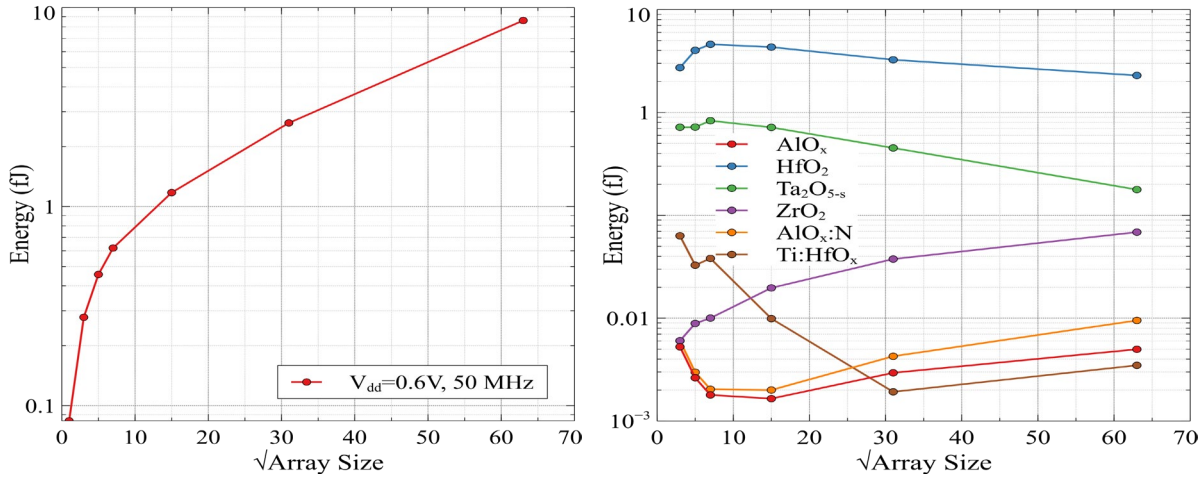

Figure 24. Energy comparison of SRAM-based (left) and several memristor-based (right) switch block cells.

potential for implementing for energy-efficient on-chip communications using memristors will come with the inclusion of other material stacks, or at least improved versions of existing options.

For the fabricated RAVENS chip on the SUNY Poly 10LPe CMOS/memristor process, we will not include the memristor based interconnect, at least not as the main interconnect for the memristive RNA. For that, we opted to go with a more traditional multi-stage network whose configuration would be held fixed by configurable registers on a scan chain, similar to an FPGA. While the connectivity won't be as dense as a full switch block implementation and we don't expect to achieve the density of a memristor-based approach, we felt an all-CMOS option for interconnect was safe and improved chances of success for a memristor based RNA.

3.7 RAVENS CMOS-memristive Chip Design and Standalone Test Structures

In late August 2021, we submitted the full RAVENS test chip (Figure 25), including various standalone test structures, to SUNY Poly for fabrication. The RAVENS test chip is ready for packaging, in the sense that it is enclosed in a 110-pin pad frame with both digital and analog I/O. Since the RAVENS test chip is based on a CMOS-memristor process that essentially works as an extension of the IBM 10lpe CMOS process, in the sense that HfO_2 memristors are included, we have developed in-house standard I/O cells to build the pad frame for this and other chips like it. The UTK standard I/O library developed for the SUNY Poly CMOS-memristor process is summarized in Appendix II.

The RAVENS test chip shown in Figure 25 includes the following major modules:

- 32-bit RISC-V Processing Core (I/O: UART, SPI, Scan-Chain)
- Digital, Instruction-based Dot Product Engine (I/O: SPI, Scan-Chain)
- Digital RAVENS RNA (16 Physical, 64 Virtual Cores; I/O: SPI, Scan-Chain)
- Memristive RNA1 – No Reference Gen. (I/O: SPI, Scan-Chain)
- Memristive RNA2 – Incl. Reference Gen. (I/O: SPI, Scan-Chain)
- Standalone Memristive nCore with Leak (Analog/Mixed-Signal Connections)
- Standalone 1kB SRAM Block (Direct Digital I/O)
- Standalone Neuron and Synapse for Testing
- Standalone Inverters – One with GPIO Connections; One using Input/Output Pads
- 9 12x2 Standalone Test Structures – Not Connected to Pad Frame

The 9 12x2 test structures are included for early probe station testing of basic synapses and other key components, before packaging the chip.

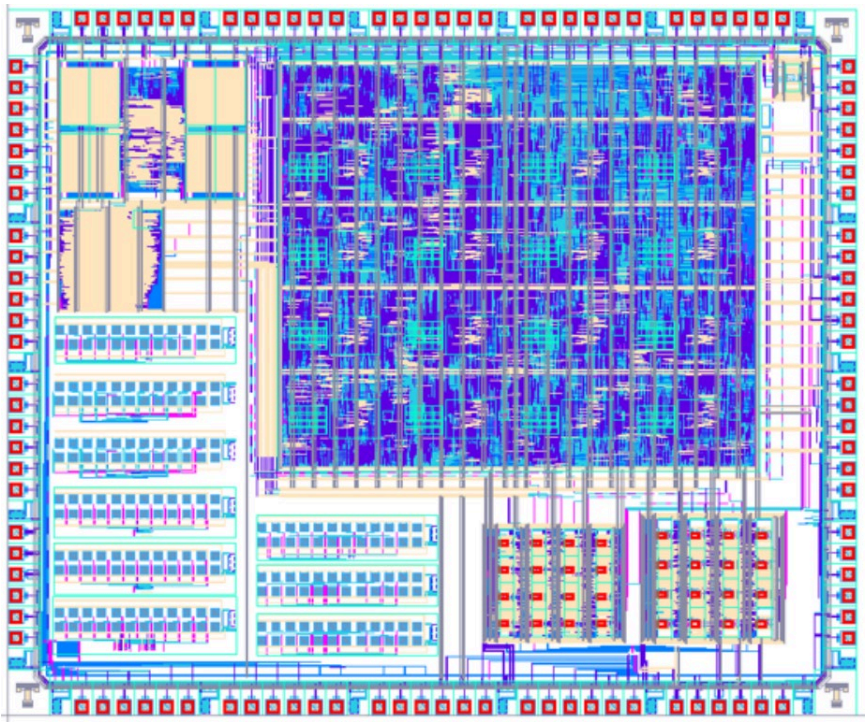

Figure 25. Layout view of full RAVENS test chip, minus some "off-chip" standalone test structures. This layout was submitted to SUNY Poly for fabrication August 2021.

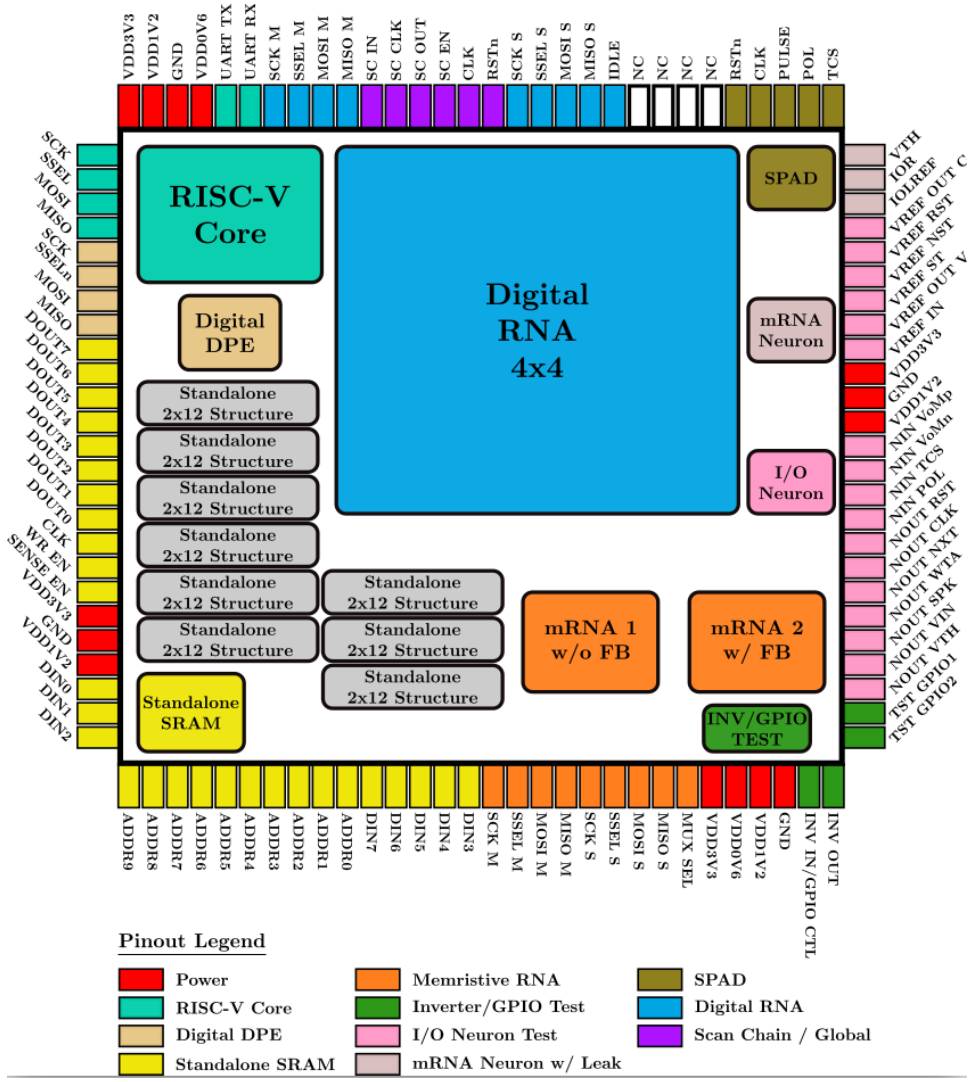

Figure 26. Pinout of the full RAVENS 2021 test chip, with the various modules highlighted.

Figure 26 shows the pinout and high-level layout of the full RAVENS 2021 test chip, with all major modules clearly marked. Once all standalone structures are tested, and given we've confirmed the pad frame is operational, we will work with collaborators at SUNY Poly to have this test chip packaged for further testing of components that are primarily accessed via digital I/O.

3.8 TENNLab Software Framework and RAVENS Processor Models

Figure 27 shows an illustration of the TENNLab Software Framework, including all the various pieces used for neuroprocessor modeling, neural network training/optimization, and application development. This illustrates the various pieces of the framework in its current version as of November 2022. For example, under the heading of "processors," are 4 different neuroprocessor models: Caspian (developed in collaboration with ORNL), GNP (generic model originally used as the base for mrDANNA), RISP (Reduced Instruction Spiking Processor), and RAVENS. The RISP model is our most recent "base" in that it can be expanded to include additional features for simulating more complex neuroprocessor architectures such as RAVENS. Thus, the RAVENS neuroprocessor model is based on RISP.

While Figure 27 lists one simulation model for RAVENS, with a note that this is targeted for memristor-based implementations. We have two versions of the RAVENS neuroprocessor model included in the software framework: one representative of the digital implementation (includes TDM) and the second more representative of the analog/memristive implementation (no TDM, resolution matched to memristors). At the architectural level, these various models are essentially the same. However, the neuroprocessor simulation models include more detail as we can consider them somewhat microarchitectural, meaning the nuances of a memristor-based implementation are fairly well captured for high-level, fast simulation.

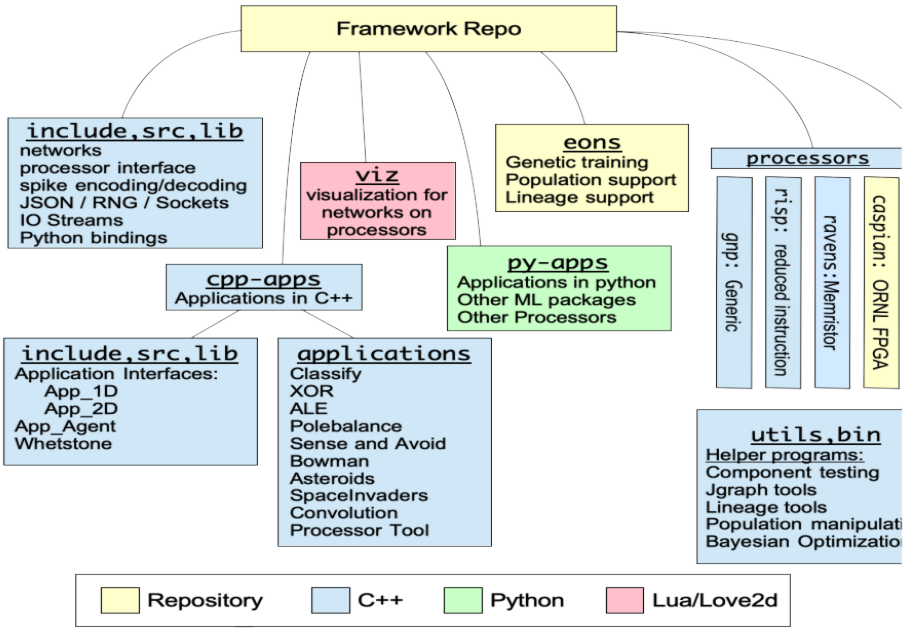

Figure 27. Structure of the TENNLab Software Framework, used to model, simulate, and train RAVENS specific neural networks. As noted under "processors," the framework includes several other neuroprocessor models as well.

There are various other components in the overall framework, as illustrated by Figure 27. We have C++ and Python bindings ready to leverage for application development, including I/O specification and clear definitions of objective functions. Various applications have been developed over the past few years, including through the duration of the RAVENS project. Often application development is a good way for our undergraduate students to get started with TENNLab. Several of these applications are included in the framework as part of a library that can be used for benchmarking purposes. These applications range in type across spatial classification problems, spatiotemporal classification, and control.

The software framework also include support for a few different training options. In fact, the entire framework allows one to work at the algorithmic level, exploring how different training approached perform across the different neuroprocessor models, for example. The most common training algorithm used for RAVENS continues to be EONS, which is a genetic algorithm that generates, evolves and evaluates SRNNs over several epochs. EONS not only trains the weights of the neural networks, but also explores the space to optimize the topology of the SRNN for a particular application. This is an important feature for EONS, in that sparse SRNNs are not layered and thus not particularly well suited for machine learning algorithms such as back propagation or even back propagation through time (BPTT). While an algorithm such as BPTT can be applied to spiking neural networks, it presumes the network topologies are layers. Sparse SRNNs are

typically not layered, but involve many recurrent connections across various neuron levels. Furthermore, the application of BPTT or any other gradient-descent approach to spiking neural networks requires some way to differentiate the spike itself or the information represented in a spike train. Based on these various challenges, we find EONS to be the more reliable choice for training sparse SRNNs typically mapped to RAVENS, mrDANNA or other neuroprocessors designed for resource constrained environments.

In RISP, each neuron has two configurable parameters: a threshold, and whether or not the neuron leaks. As with most spiking neuroprocessors, each neuron stores an action potential, whose value may be increased or decreased by spikes through incoming synapses. Processing is accomplished in discrete timesteps, where incoming spikes are integrated over the timestep. The action potential of a neuron will then determines whether or not the neuron fires at the end of the timestep. When neuron leak is enabled, assuming the neuron does not fire at the end of the integration cycle, its action potential is reset to zero. Without leak, neurons retain their action potentials across time steps.

Due to the discretized integration cycle of neurons, synapses only have unit delays. Like neurons, synapses may be configured to either have discrete weights or analog weights. There is no restriction on connectivity, or the number of synapses that may come into or go out of a neuron.

Neurons can be designated as input neurons that may receive spikes from the outside world. Similarly, neurons may be specified as output neurons, whose spikes are driven into the outside world.

To be precise, a RISP network is defined by 8 sets:

1. N Neurons: $V = \{v0, v1, ...vN-1\}$
2. N Thresholds: $T = \{t0, t1, ...tN-1\} | ti \in R \text{ or } Z$
3. N Leaks: $L = \{l0, l1, ...lN-1\} | ti \in \{T, F\}$
4. M Synapses: $E = \{e0, e1, ...eM-1\} | ei = (vj \rightarrow vk)$
5. M Weights: $W = \{w0, w1, ...wM-1\} | wi \in R \text{ or } Z$
6. M Delays: $D = \{d0, d1, ...dM-1\} | di \in Z$
7. Input neurons: $I \subseteq V$.
8. Ouptut neurons: $O \subseteq V$.

An example neural network implemented for RISP is shown in Figure 28. This network calculates which input neuron, IX or IY, spikes more over an interval of t timesteps. If IX spikes more, then OX spikes exactly once, at timestep t + 1. If IY spikes more, then OY spikes exactly once, at timestep t + 1. If they spike equally, then neither spikes, although it is a straightforward matter to add an extra synapse to break ties in favor of IX or IY. There is a bias neuron that is required to spike once at timestep 0. The network clears itself, and may be reused for more inputs at timestep t+1. This type of network is useful for composing spiking neural networks, perhaps created by other methodologies (e.g., back propagation). The network in Figure 28 performs this interpretation, which can in turn be used as inputs to other spiking neural networks, or even as control signals for a physical device.

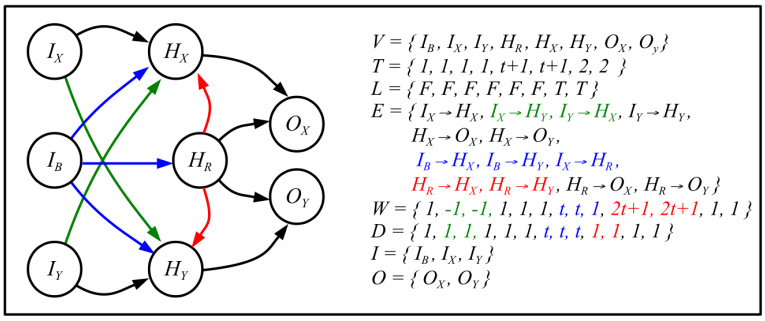

Figure 28. RISP neural network for computing which input fires more over an interval of t timesteps. The synapses are colored help facilitate reading elements of sets E, W, and D.

The RAVENS neuroprocessor models are built on the base RISP framework, meaning neural networks can be constructed in a similar manner as what is described here for RISP. Among the distinguishing features for RAVENS are programmable leak, axonal delay, absolute/relative refractory, and in the case of the digital implementation virtualization. The memristive RAVENS implementation is currently between the full RAVENS model and RISP, in the sense that features such as relative refractory and virtualization are left for future work.

Header files for the digital and memristive implementations of RAVENS are provided in Appendices III and IV, respectively. These provide some insight into the various components of the neuroprocessor models. The full neuroprocessor models, as well as the TENNLab Software Framework, are available as shareable Git repositories.

4.0 Results and Discussion

4.1 Test Results for Small CMOS-Only RAVENS 2020 Chip

The small test chip (CMOS-only) was received in late October 2020. One test, used to validate the I/O pad cells and to some extent our basic digital logic gates, we setup a simple test for an inverter that was included on chip. The test of this inverter was conducted using a portable oscilloscope, Salaea logic analyzer, Digilent Arty FPGA board and other miscellaneous equipment. The test setup can be seen in Figure 29. Note that these tests were conducted during the COVID-19 pandemic.

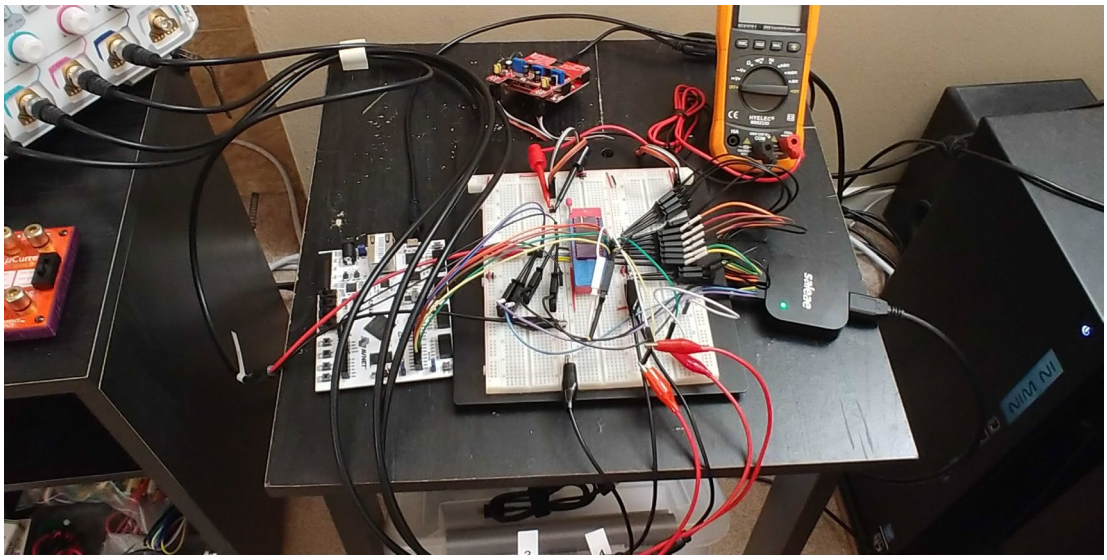

Figure 29. Test setup for early testing of the small chip.

The inverter test is simple and thus not terribly impressive. However, this test does validate that the pad cells we used for this design (full custom by our group) do in fact perform as expected. We tried several setups, including one with a 47 pF coupling capacitor. The eye diagram results from this particular inverter test can be seen in Figure 30.

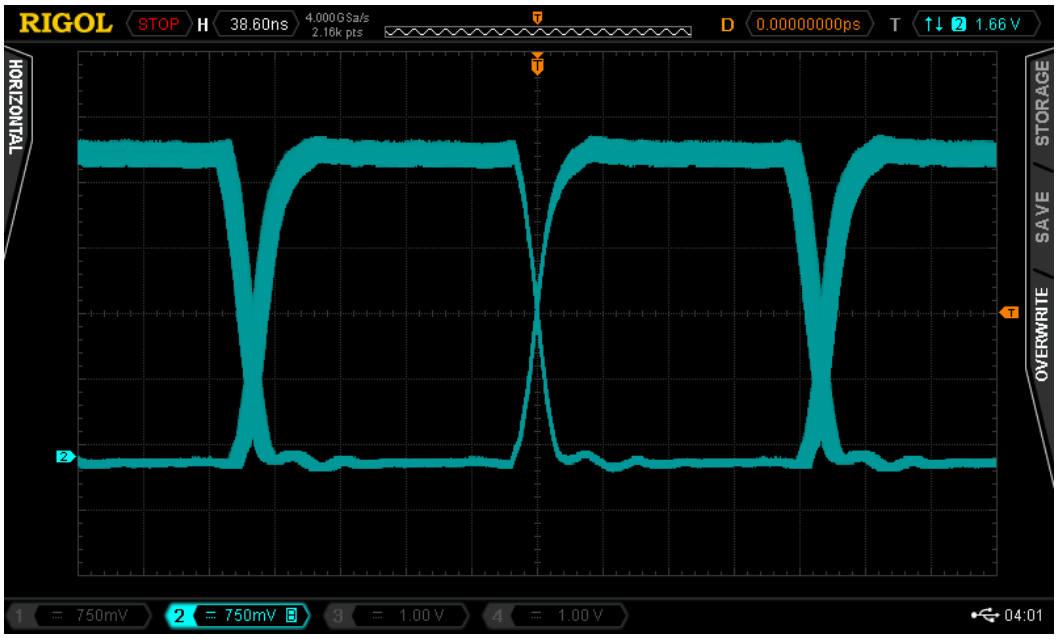

Figure 30. Eye diagram measurement of an inverter included on the small July RAVENS test chip.

Another component that has been tested was a small scale SPAD (single-photon avalanche diode) that could potentially be used as part of an event-based camera. The idea here is that the sensor and the neural network that processes images from these sensors can potentially be co-located on the same silicon. **To the best of our knowledge, this is the first implementation of a SPAD using a 65 nm CMOS process**. We also included a small SPAD array circuit that can quickly translate photon detection events into spikes (inputs to a neuromorphic array).

Testing was somewhat limited for our SPAD and other analog components, primarily due to limited access to the lab where our probe station is located. We share the space with another group and, unfortunately, a COVID-19 case in the other group led to our lab being closed for some time. Even before closing the lab, we had to limit access to the space in order to maintain social distancing and protect the members of our research group. Given the requirements of the RAVENS project, we opted to test what we could outside the lab and then move on with other research tasks.

We were able to perform testing on the SPAD device itself, showing that it does perform as expected, even when implemented using a 65 nm process. These results have been included in a short conference paper and were used to develop models for a full circuit design, including the neural network that would process SPAD generated information. The latter have been compiled for a journal submission.

Results from SPAD testing can be seen in Figure 31.

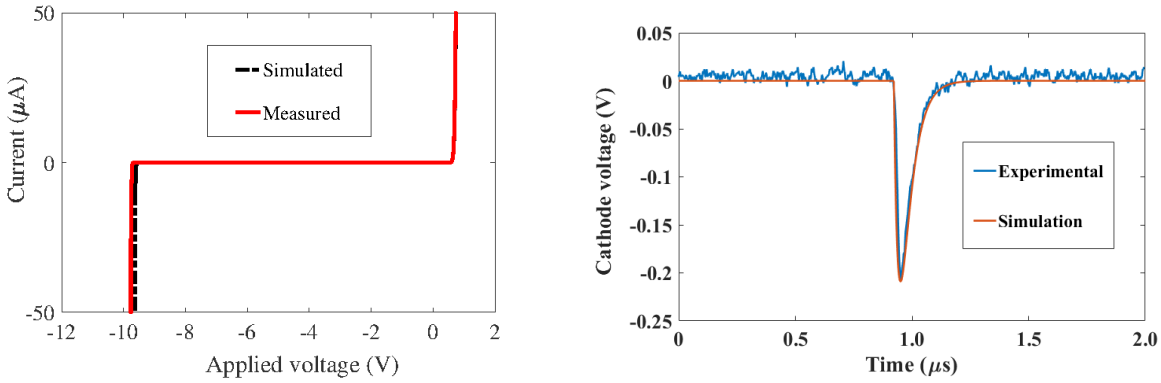

Figure 31. For SPAD device fabricated in a 65 nm CMOS process: (left) Simulated and measured IV characteristics. (right) Simulated and measured cathode voltage at photon event (around 1μs) and quenching behavior.

We were also able to validate the scan chain included in our small RAVENS 2020 chip, though more testing is needed to understand more complex behavior. This scan chain is a CMOS implementation of a similar design we've completed for programming memristive memory elements. The scan based memristor programming circuit led to a patent application.

Figure 32 shows results from one test of the scan chain, showing two bits are scanned in via the SC_IN input and then several cycles later the same 2 bits are scanned out through the SC_OUT port. There is one extra bit that is scanned out through SC_OUT but this occurs after the scan is disabled, as indicated by SC_EN going low. We're still looking into why SC_OUT might go high at any point when scan is disabled. However, the behavior while scan is enabled is as expected, validating the scan chain design and the standard cells (developed in-house at UTK) used to implement it.

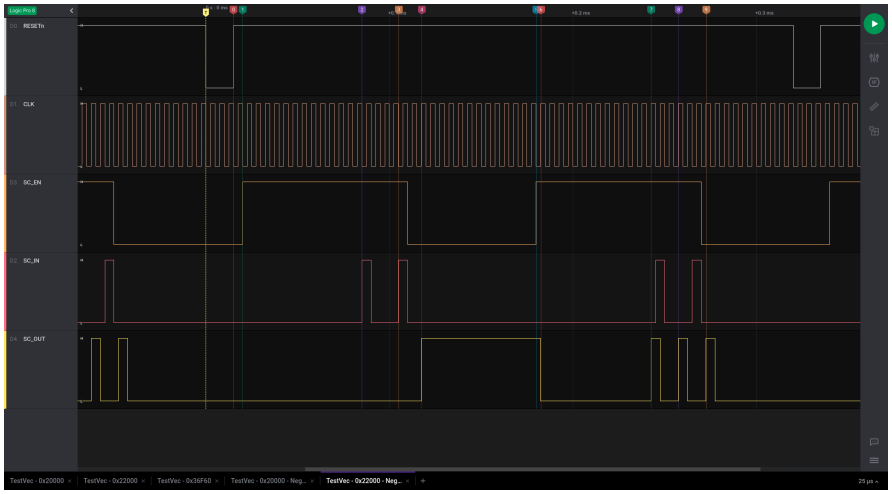

Figure 32. Measured results of simple scan in and scan out test using the scan chain on the July RAVENS test chip.

Another test structure that was tested but showed mixed results was the small SRAM included on the small RAVENS 2020 chip. Based on our results, we're confident that we could write data to the SRAM but the read was not optimal. Specifically, it was clear the sense amp was trying to drive the correct output but also wasn't strong enough to drive that value through the pad cells. The flaw in the design is that there was no buffer between the sense amp and the large pad cell circuit, which must drive relatively large off-chip loads. We have simulated this likely scenario to

determine that this would lead to the behavior observed and have corrected this design in the SRAM included in the full RAVENS 2021 system design.

4.2 Architectural Comparisons: Benchmarking RAVENS and RISP

To better understand the impact of various architectural features included in RAVENS, it was benchmarked against the base neuroprocessor architecture, RISP. The additional features that differentiate RAVENS at an architectural level are: programmable leak, axonal delay, an absolute refractory period and relative refractory period. For the digital implementation of RAVENS, the architecture also allows for TDM based context switching, or neuron virtualization. Table 1 below lists the various features that can be included with a RAVENS configuration alongside which features can be included in the base RISP architecture.

Table 1. Configuration options for RISP and RAVENS neuroprocessors.

RAVENS* & RISP† Configurations	
Configuration	**Description**
Base* †	Includes only basic I&F neuron functionality
Delay w/o Leak* †	Enables axonal delay
Delay w/ Leak (No Refractory)* †	Enables delay and linear leakage only
Delay w/ Leak (Absolute Refractory)*	Delay and leak enabled plus absolute refractory period only
Delay w/ Leak (Full Refractory)*	Delay and leak enabled plus the full absolute and relative refractory periods
Delay w/ Leak (Full Refractory Limited)*	Delay and leak enabled plus the full absolute and relative refractory periods. Refractory resting potential limited to less than the standard resting potential

A more bio-inspired feature included in RAVENS that is not available in other like architectures is a separation between absolute and relative refractory periods. Figure 33 illustrates the refractory period observed in biology and also shows the behavior as specified for RAVENS.

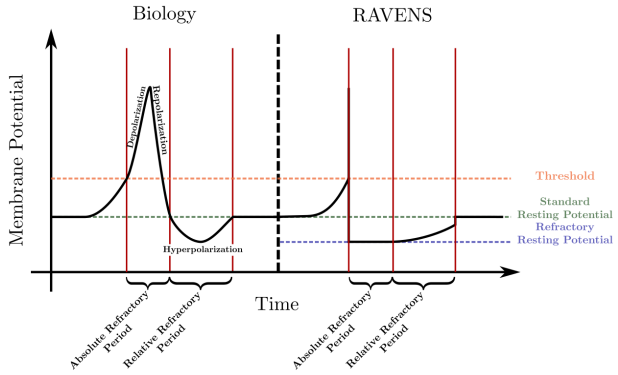

Figure 33. (Left) Spike waveform for a biological neuron with absolute and relative refractory clearly marked. (Right) RAVENS charge accumulation showing how this behavior is mimiced.

A spike in biology is defined by three distinct regions: depolarization as the neuron potential increases, repolarization as the potential comes back down toward rest, and hyperpolarization where the tail of the spike is below the resting potential. During the depolarization and repolarization phases, the neuron is in an absolute refractory period, meaning the neuron is not firing and accumulation of incoming spiking information does not occur. During this absolute refractory period, not only does the neuron not fire again, as it is already in the process of firing, but it is also ignoring all incoming spikes. The hyperpolarization phase shows an overcorrection just after the primary spiking event where the potential is less than the resting potential. During this relative refractory period, synonymous with hyperpolarization, the neuron will accumulate new input spiking information but it will not fire.

Since a spike is often marked by a single event, in an artificial neuron model the refractory is represented by the time after the spike where inputs are effectively ignored. What the RAVENS model allows for is a separation of the refractory period into two parts: one representing absolute refractory and another representing relative refractory (right of Figure 33). For RAVENS, both refractory periods can be programmable, meaning one can specify the different times for a given application. For both refractory periods, the RAVENS neuron will not fire. However, unlike the absolute refractory, inputs driven into the neuron will accumulate during the relative refractory period. This provides a model more inline with biological behavior.

To better understand how these various features (leak, delay, absolute and relative refractory) impact application performance, we compared results for RISP and RAVENS for three different control applications. The control applications used for this study were: Polebalancing, a Bowman game, and a version of the Asteroids video game. The applications range from low complexity (few parameters, small neural networks) for Polebalancing to relatively high complexity (more parameters, larger neural networks) for Asteroids.

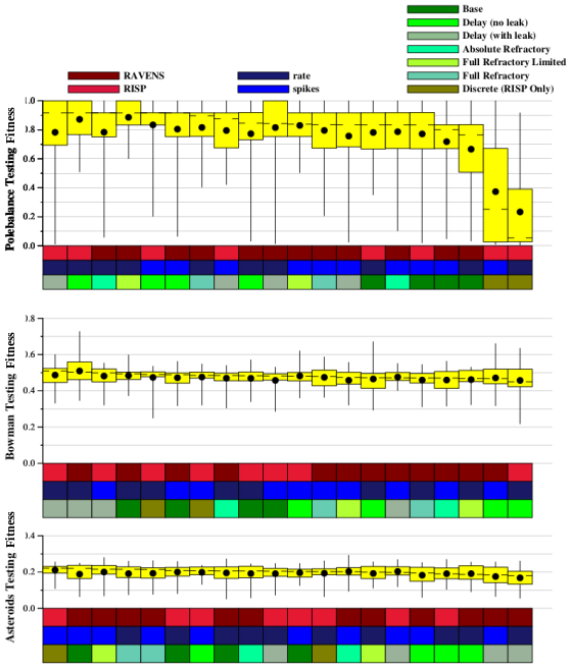

Figure 34. Test fitness results for all three applications considered. The box colors along the x-axes indicate the parameter combination used for each fitness result.

Figure 34 shows results comparing RAVENS and RISP for the three applications considered across implementations with different combinations of parameters (delay, leak, etc.). The top plot in Figure 34 shows results for Polebalancing, the center is for Bowman and the bottom for Asteroids. Here, fitness varies per application and is not normalized in these results. For example, with a fitness of 0.2 we find the player (neural network) can remain alive for an extended period of time while achieving many successful kills of incoming asteroids. Thus, 0.2 is considered a good fitness score for Asteroids. The Figure 34 results are also separated based on the type of spike encoding used for each test. The dark blue boxes represent instances that used rate-based (frequency of spike train) encoding while the light blue boxes represent spike-timing (time between spikes) encoding. For the discrete RISP implementation (simplest architecture, fewest parameters) executing Polebalancing the results are not as good as they are for other

implementations. The results suggest that the added parameters can be beneficial, but for most tests the results are equally good.

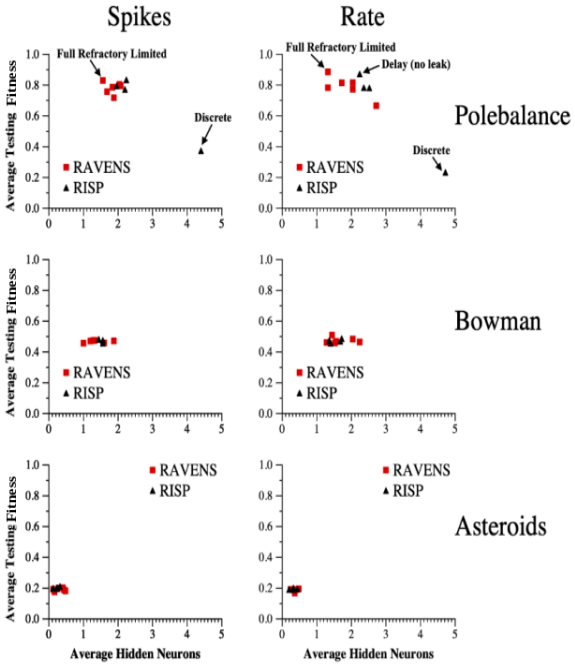

Figure 35. Test fitness results for all three applications considered. This plot shows the network size as the number of neurons required along the x-axes. The two sets of results were also determined for spike-timing (left) and rate-based (right) spike encoding.

Another view of the results from this study is shown in Figure 35. Here, the fitness is again plotted but as a function of the number of neurons (a measure of network size/complexity) included for the given application. From these results, at least for Bowman and Asteroids, there isn't much variance for the different implementations and parameter sets considered. In fact, for the most complex application, Asteroids, all architectural configurations not only perform with very similar fitness but require similarly sized neural networks. Polebalancing, the simplest among the applications in terms of control parameters, provides some more interesting results. For both spike-timing and rate-based encoding, the smallest neural networks for Polebalancing were those implemented for RAVENS and utilized both absolute and relative refractory period parameters. Furthermore, these same implementations are either the highest or among the highest performing in terms of fitness. Thus, the simpler application seems to be able to utilize the added complexity per neuron to produce well-performing neural networks with minimal cost, in terms of neuron count.

An important consideration that isn't highlighted in the results shown is the impact of the training algorithm. In actuality, the neuroprocessor doesn't take advantage of neuron features to produce smaller neural networks. The training algorithm determines the size of the neural network that is mapped to the neuroprocessor for efficient execution of the application. Thus, it is the training algorithm that somewhat takes advantage of relative refractory in the case of Polebalancing to produce a smaller yet higher performing neural network. On the other hand, it is also the training algorithm that is not able to take full advantage of the feature set to further optimize, either for fitness or network size, for Asteroids or Bowman. As far as network size is concerned, it is important to point out that Asteroids and Bowman, while complex in terms of

control parameters and degrees of freedom, actually consistently yield small neural networks with what appears to be a minimal set of hidden neurons. Thus, it appears the training algorithm did about as good a job as it could for those applications. For all implementations considered in this particular study, we utilized the evolutionary optimization for neuromorphic systems (EONS) training algorithm. Further works could certainly be done considering different training approaches and a wider variety of benchmark applications.

4.3 Test Results for CMOS-memristor RAVENS 2021 Chip

Though submitted for fabrication around August 2021, manufacturing delays were experienced such that we did not receive any testable wafers from SUNY Poly until late October 2022. As such, we have only been able to complete some limited testing to meet the final reporting deadline. Since the RAVENS design is being leveraged for another AFRL funded effort, further testing will be completed as part of that project. We summarize our early results here.

For the full 110-pin RAVENS test chip, we included a very simple inverter on the pad frame that could be used to verify that all supplies were driven properly into the core and that the digital input and output pad cells worked properly. This simple inverter is our sanity test to verify the UTK developed I/O library (Appendix II), specifically the following pad cells: pad_vdd3v3 (for off-chip 3.3 V), pad_vdd1v2 (for 1.2 V core supply), pad_gnd (for ground), pad_dig_inp2 (for digital input), and pad_dig_out2 (for digital output). Both input and output pad cells include level shifters that shift between the 3.3 V needed for off-chip PCB connections and the 1.2 V used for core CMOS-memristor logic. Note that the 3.3 V supply is also used for forming memristors in this CMOS-memristor process. Finally, this simple inverter sanity test proves the inverter (from our UTK standard cell library) also functions as expected.

Figure 36 shows a waveform of an input square wave signal into the RAVENS test chip, oscillating between 0 and 3.3 V (off-chip supply). The output shows the proper inverted signal, also between 3.3 V and 0, as this would be a signal delivered off-chip. In order for this signal to be produced as shown, the level-shifter from 3.3 V to 1.2 V in the pad_dig_inp2 cell must be able to deliver a proper 0 to 1.2 V to the small (65 nm) standard cell inverter on-chip. Likewise, the inverter generates a 1.2 V to 0 output, since it is a core standard cell. So, the pad_dig_out2 cell level shifts the on-chip signal from 1.2 V to 3.3 V. Further, pad_dig_out2 includes a tappered buffer capable of driving around 70 pF of off-chip load capacitance. As simple as this result is, it verifies the functionality of these four critical I/O pad cells, thus showing that our pad frame works as expected.

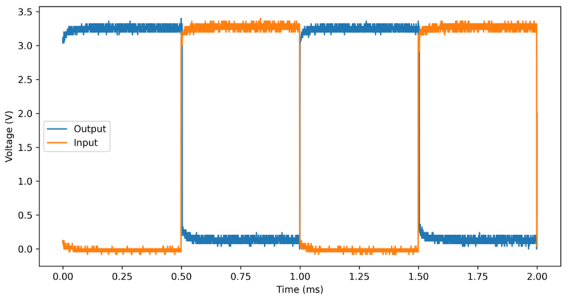

Figure 36. Measured waveform for transient test of single inverter on RAVENS test chip.

Another helpful test of the inverter in the 110-pin RAVENS test chip is a voltage transfer curve (VTC). Figure 37 shows results of a VTC for this inverter using a triangle wave input signal.

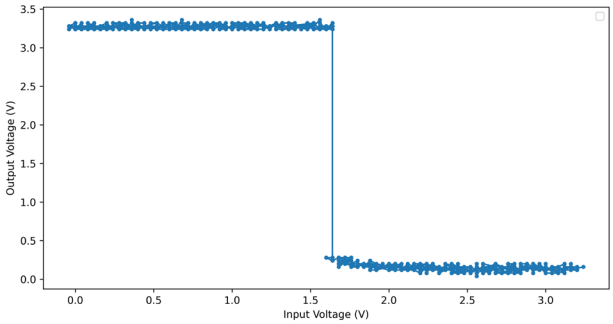

Figure 37. Measured VTC of 65 nm inverter driving off-chip output pin. Input was triangle wave.

Satisfied that our pad frame and the basic inverter used to test are behaving as expected, we have shared these results with SUNY Poly and begun the process to have the 110-pin test chips packaged. The UTK standard I/O cell library has already been used by other research groups to construct pad frames for their own test chips using the SUNY Poly CMOS-memristor process. With these positive results, their chips are now cleared for packaging. Given that most components on the larger 110-pin chip require testing of several digital I/O pins, we will need to wait for shipment of the fully packaged chips to test more complex on-chip logic.

Also in the two weeks since receiving RAVENS wafers from SUNY Poly we have begun probe station testing of standalone test circuits that were placed in 12x2 pad frames. These standalone structures were designed to test each part of the current-controlled synapse, starting with a 1T1R test circuit within a 12x2 frame. This test is a bit different from tests of the many 1T1R structures used to characterize memristive devices in that the frames include ESD protection. Thus, we are testing the memristor with not just one NMOS, but also all ESD circuitry and the RC clamp. We have found that we can successfully form the memristor in these 1T1R structures, which is a very positive result. However, the 1T1R is limited when trying to perform a RESET operation because of the ESD. The RC clamp in particular clamps any incoming voltage to one of the rails, either VDD (4 V in early testing) or ground. Since RESET using a 1T1R requires the application of a negative node, the voltage gets clamped and thus is not applied across the memristor. However, we can see that forming works for the 1T1R with ESD, so we moved to the next simplest test structure.

The standalone pad frames also include the central 2T1R stack included in the current-controlled synapse, with an NMOS used for current compliance at the bottom of the memristor and the top connected to a PMOS that drives voltages across the memristor for forming, SET, RESET and read. These 2T1R structures also include pads directly connected at both the top and bottom of the memristor, allowing a direct read after any operation (e.g. form) to determine success or failure. We have formed a few of these devices successfully and have been working through our test setup to move through RESET then SET. Figure 38 shows results of a read after form event using the 2T1R circuit. Importantly, this demonstrates a forming event controlled by on-chip CMOS, instead of any pads directly connected to either memristor terminal.

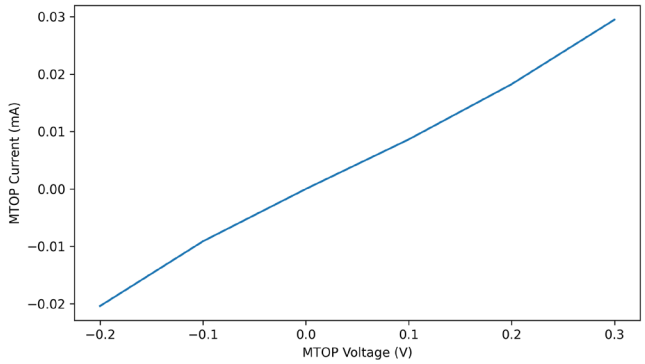

Figure 38. Measured results for a read after form, using a source meter to sweep voltages from -0.2 V to 0.3 V (small enough to not change the resistance). The measured resistance after the forming event in this case is approximately 10 kOhm. Preforming resistance values were found to be in the range of 10 MOhm.

We have seen results that suggest RESET works as expected, but we are continuing the refine the test setup to make sure our results are reliable and repeatable. Some challenges we are currently facing are related to minor differences in the device parameters for the HfO_2 memristors in this first batch. Specifically, SUNY Poly was able to actually improve yield, significantly from what we've seen, which in turn has had the effect of increasing the forming voltage. For this current batch of test wafers, the forming voltage is in the range of 3.5 V, compared to something between 2.0 to 2.5 V for the mrDANNA chips. Since our circuits were designed with a I/O VDD of 3.3 V in mind, this leads to some inconsistencies that are likely related to how the transistors are optimized. For forming to work, we have been applying a high VDD of 4 V to the standalone 12x2 frames. Our understanding is that there are ways to bring down the forming voltage, but we will need to wait for additional fabrication runs for that. In the meantime, we are further characterizing devices and plan to continue testing of the RAVENS chips as it is a solid pre-cursor for the aforementioned follow-on project.

Further testing is needed on various fronts, including for power. The RAVENS current-controlled synapse was designed for better regulation of power and is expected to operate during read cycles with 1 uA or less current through the memristor. For the core VDD of 1.2 V, this amounts to a static power of 1.2 uW. When not operating (no form, SET/RESET, or incoming spike), the synapse is also powered off to minimize idle power. Thus, we anticipate favorable power characteristics, but testing is required to demonstrate what is possible in the actual circuit.

Appendix V provides pinouts and descriptions of the standalone 12x2 pad frame test structures included in the RAVENS chip.

5.0 Conclusions

In this project, we have designed and developed a neuroprocessor architecture, RAVENS, that is well equipped for mapping and executing sparse spiking recurrent neural networks, such as those produced by EONS for control and other applications dealing with spatiotemporal data. RAVENS is intended for implementation using a CMOS-memristor process technology under development at SUNY Poly. As such, we've had to refine the design kit to include layers for defining memristors, develop standard I/O pad cells, develop a standard cell library and also work through a top-down design flow ready for mixed-signal integration. The goal here was to push the limits of how memristors can be used, enabling their inclusion in full system integration of CMOS-memristor systems-on-chip. Early testing of the developed pad cells and some standard cells show that the components we've developed for these various libraries function as expected.

RAVENS incorporates several novel architectural features that can be leveraged for more efficient implementation of spiking neural networks. Our work has included consideration for relative and absolute refractory as well as a TDM virtualization method for improving the functional density of neurons. Some of these features are implemented in our digital RAVENS and only considered for memristive RAVENS through simulation. The memristive implementation does include test circuits for leak, context switching using analog neurons is a challenge left for future work. Our architectural exploration of RAVENS, from a generalized perspective not specific to either implementation, found that the features added could useful to improve neural network performance but such advantages are likely limited by the ability of the training algorithm to leverage them.

It is also important to note that RAVENS, both digital and memristive implementations, is built around an RNA (reconfigurable neuromorphic array) architecture where neurons fire spiking events through a network and into other neurons, all without ever being converted into AER or other packetized representations. Our architecture is novel, and as best we can tell unique, in that we route spikes, not packets. The allows us to consider the potential of spikes on power consumption for the full neuromorphic array as we are not always converting from spike trains to digital and back again.

Two of the three years of the RAVENS project were completed during the COVID-19 pandemic. This has had an impact on much of what we do. When the pandemic started in 2020, our IC design team decided to focus on VLSI design over testing of prior chips. For the most part this worked fine, but there were points where communication through email, Slack, Zoom and other remote tools were not as good as face-to-face discussion. Moreover, several students graduated and post-doctoral assistants found work elsewhere. Such transitions are normal for a university, but training each new generation was limited by the fact that training started in a virtual environment. Furthermore, recruitment was difficult for a wide variety of reasons. Overall, we

were able to weather the storm that was the pandemic and have produced two chips that seem functional for the testing we've been able to complete so far.

The main challenge in the end is that we have not been able to test chips as fully as hoped before submission of this final report. This is due to manufacturing delays that are beyond our control and beyond the control of our collaborators at SUNY Poly. Development of this novel technology, and all various tools that must support it, is a complex task that has been largely made possible by the work of students and other performers in academia. Regardless of these delays and challenges, what we have seen thus far is that yields for the memristors used in RAVENS have dramatically improved, the memristors so far seem to mostly play well with CMOS and the design libraries developed in-house for this effort are behaving well. While more testing is certainly required to fully assess everything included on the RAVENS test chip, our results thus far provide good reason to be optimistic.

6.0 Publications Resulting from This Project

6.1 Journal Papers

1. G.S. Rose, M.S.A. Shawkat, A.Z. Foshie, J.J. Murray, and M.M. Adnan, "A System Design Perspective on Neuromorphic Computer Processors," *IOP Neuromorphic Computing and Engineering*, vol. 1, no. 2, November 2021. (Online: September 2021)
2. A. Zeumault, S. Alam, Z. Wood, R.J. Weiss, A. Aziz, and G.S. Rose, "TCAD Modeling of Resistive-Switching of HfO2 Memristors: Efficient Device-Circuit Co-Design for Neuromorphic Systems," *Frontiers in Nanotechnology*, vol. 3, no. 71, October 2021.
3. M.M. Adnan, S. Sayyaparaju, S.D. Brown, M.S.A. Shawkat, C.D. Schuman, and G.S. Rose, "Design of a Robust Memristive Spiking Neuromorphic System with Unsupervised Learning in Hardware," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 17, no. 4, article 56, June 2021.
4. S. Sayyaparaju, M.M. Adnan, S. Amer, and G.S. Rose, "Device-Aware Circuit Design for Robust Memristive Neuromorphic Systems with STDP-based Learning," *ACM Journal of Emerging Technologies in Computing Systems*, vol. 16, no. 3, article 28, May 2020.
5. K. Beckmann, W. Olin-Ammentorp, G. Chakma, S. Amer, G.S. Rose, C. Hobbs, J. Van Nostrand, M. Rodgers, and N.C. Cady, "Towards Synaptic Behavior of Nanoscale ReRAM Devices for Neuromorphic Computing Applications," *ACM Journal of Emerging Technologies in Computing Systems*, vol. 16, no. 2, article 23, April 2020.
6. A.R. Young, M.E. Dean, J.S. Plank, and G.S. Rose, "A Review of Spiking Neuromorphic Hardware Communication Systems," *IEEE Access*, vol. 7, pp. 135606-135620, September 2019.

6.2 Conference Papers

7. R. Weiss, H. Das, N.N. Chakraborty, and G.S. Rose, "STDP Based Online Learning for a Current-Controlled Memristive Synapse," in *Proceedings of IEEE International Midwest Symposium on Circuits and Systems* (*MWSCAS*), Fukuoka, Japan (Virtual), August 2022.
8. M. Rathore, R.J. Weiss, M.S.A. Shawkat, G.S. Rose, M. Liehr, M. Abedin, S. Rafiq, and N.C. Cady, "A Compact Model of the Variable Switching Dynamics of HfO_2 Memristors," in *Proceedings of IEEE International Midwest Symposium on Circuits and Systems* (*MWSCAS*), Fukuoka, Japan (Virtual), August 2022.
9. M.M. Adnan, M.S.A. Shawkat, R.D. Febbo, and G.S. Rose, "On-Chip Interface for Event based Sensor and Spiking Neuromorphic Processing," in *Proceedings of IEEE International Midwest Symposium on Circuits and Systems* (*MWSCAS*), Fukuoka, Japan (Virtual), August 2022.
10. A.Z. Foshie, C. Rizzo, H. Das, C. Zheng, J.S. Plank, G.S. Rose, "Benchmark Comparisons of Spike-based Reconfigurable Neuroprocessor Architectures for Control Applications," in *Proceedings of the ACM Great Lakes Symposium on VLSI* (*GLSVLSI*), Irvine, CA, June 2022.

11. C. Schuman, J. Plank, R. Patton, and G. Rose, “A Framework to Enable Top-Down Co-Design of Neuromorphic Systems for Real-World Applications,” in *Proceedings of the Neuro-Inspired Computational Elements Conference*, Virtual, March 2022.
12. N.N. Chakraborty and G.S. Rose, “Programmable Refractory Period Implementations in a Mixed-Signal Integrate-and-Fire Neuron,” in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, Austin, TX, May 2022.
13. S.D. Brown, M.M. Adnan, M.S.A. Shawkat, and G.S. Rose, "Capacitor-Less Memristive Integrate-and-Fire Neuron with Stochastic Behavior," in *Proceedings of IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Lansing, MI, August 2021.
14. J.J. Murray, A.Z. Foshie, M.S.A. Shawkat, and G.S. Rose, "Scaling Constraints for Memristor-based Programmable Interconnect in Reconfigurable Computing Arrays," in *Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Tampa, FL, July 2021.
15. A.Z. Foshie, N.N. Chakraborty, J.J. Murray, T.J. Fowler, M.S.A. Shawkat, and G.S. Rose, "A Multi-Context Neural Core Design for Reconfigurable Neuromorphic Arrays," in *Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Tampa, FL, July 2021.
16. S. Sayyaparaju, M.S.A. Shawkat, and G.S. Rose, “Robust Implementation of Memristive Reservoir Computing with Crossbar Based Readout Layout,” in *Proceedings of IEEE Dallas Circuits and Systems Conference (DCAS)*, Dallas, TX, November 2020.
17. S. Sayyaparaju, M.S.A. Shawkat, M.M. Adnan, and G.S. Rose, “Circuit Techniques for Efficient Implementation of Memristor Based Reservoir Computing,” in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, Seville, Spain, October 2020.
18. J. Plank, J. Zhao, and B. Hurst, “Reducing the Size of Spiking Convolutional Neural Networks by Trading Time for Space,” in *Proceedings of IEEE International Conference on Rebooting Computing* (*ICRC*), Glasgow, Scotland, December 2020.
19. M.S.A. Shawkat, S. Sayyarparaju, N. McFarlane, and G.S. Rose, “Single Photon Avalanche Diode based Vision Sensor with On-Chip Memristive Spiking Neuromorphic Processing,” in *Proceedings of the IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Springfield, MA, August 2020.
20. C. Schuman, J.P. Mitchell, M. Parsa, J. Plank, S. Brown, G. Rose, R. Patton, and T. Potok, “Automated Design of Neuromorphic Networks for Scientific Applications at the Edge,” in *Proceedings of The International Joint Conference on Neural Networks (IJCNN)*, Glasgow, Scotland, July 2020.
21. J. Ambrose, A. Foshie, M. Dean, J. Plank, G. Rose, J. Mitchell, C. Schuman, and G. Bruer, “GRANT: Ground-Roaming Autonomous Neuromorphic Targeter,” in *Proceedings of The International Joint Conference on Neural Networks* (*IJCNN*), Glasgow, Scotland, July 2020.

22. A. Young, A. Foshie, M. Dean, J. Plank, G. Rose, J. Mitchell, and C. Schuman, “Scaled-up Neuromorphic Array Communications Controller (SNACC) for Large-scale Neural Networks,” in *Proceedings of The International Joint Conference on Neural Networks* (*IJCNN*), Glasgow, Scotland, July 2020.

23. M.M. Adnan, S. Amer, S. Sayyaparaju, M.S. Hasan, and G.S. Rose, "A Scan Register Based Access Scheme for Multilevel Non-Volatile Memristor Memory," in *Proceedings of IEEE International Conference on Electronics, Circuits and Systems* (*ICECS*), Genova, Italy, November 2019.

6.3 Technical Reports & Preprint Articles

24. J.S. Plank, C. Zheng, B. Gullett, N. Skuda, C. Rizzo, C.D. Schuman, and G.S. Rose (2022, June). “The Case for RISP: A Reduced Instruction Spiking Processor”. ArXiv:2206:14016 [cs.NE].

6.4 Invention Disclosures & Patent Applications

25. R.J. Weiss and G.S. Rose, “Current-Controlled Analog Memory Circuits Built from Non-Volatile Memory Elements,” U.S. Provisional Patent Application, 2022.

Appendix I – UTK Standard Cell Library Summary

(For SUNY Poly 65 nm CMOS-memristor Process Technology)

Below is a summary of all digital cells/gates included in the UTK standard cell library for the SUNY Poly 65 nm CMOS-memristor process. These cells are CMOS only, though implemented such that they can be placed-and-routed with an implementation that includes memristive circuit elements.

All cells were implemented using Cadence Virtuoso, with schematic, layout, extracted and abstract views included for each. Layouts were converted to an LEF description for use in Cadence Innovus for automated place-and-route. Timing of all cells was characterized using Cadence Liberate, with the timing information included as a Liberty (.lib) file. The timing file was implemented for use with Cadence Innovus (place-and-route) and Synopsys Design Compiler (synthesis).

RTL (register transfer language) simulation is also possible with behavior Verilog representations of all gates, allowing high-level simulation of post-synthesis and post-place-and-route gate-level netlists. The simulator typically used in our work was Mentor Graphics (now Siemens) QuestaSim.

Cell Type	Name in Library	Width (um)	Height (um)	Area (um^2)
Full Adders	addf_x1	6.80	4.76	32.3680
	addf_x2	8.16	4.76	38.8416
	addf_x4	16.32	4.76	77.6832
	addf_x8	31.28	4.76	148.8928
addfi	addfi_x1	6.12	4.76	29.1312
	addfi_x2	7.48	4.76	35.6048
	addfi_x4	14.96	4.76	71.2096
	addfi_x8	29.24	4.76	139.1824
addh	addh_x1	4.08	4.76	19.4208
	addh_x2	5.44	4.76	25.8944
	addh_x4	9.52	4.76	45.3152
	addh_x8	17.00	4.76	80.9200
2-Input AND	and2_x1	1.70	4.76	8.0920
	and2_x2	1.70	4.76	8.0920
	and2_x4	2.38	4.76	11.3288
	and2_x8	3.40	4.76	16.1840
3-Input AND	and3_x1	2.04	4.76	9.7104
	and3_x2	2.04	4.76	9.7104
	and3_x4	3.40	4.76	16.1840
	and3_x8	4.42	4.76	21.0392
4-Input AND	and4_x1	2.38	4.76	11.3288
	and4_x2	2.38	4.76	11.3288
	and4_x4	4.08	4.76	19.4208
	and4_x8	5.44	4.76	25.8944

AND-OR-INV	aoi21_x1	1.36	4.76	6.4736
	aoi21_x2	2.38	4.76	11.3288
	aoi21_x4	3.40	4.76	16.1840
	aoi21_x8	6.46	4.76	30.7496
AND-OR-INV	aoi22_x1	1.70	4.76	8.0920
	aoi22_x2	3.06	4.76	14.5656
	aoi22_x4	4.42	4.76	21.0392
	aoi22_x8	8.84	4.76	42.0784
AND-OR-INV	aoi2bb1_x1	2.04	4.76	9.7104
	aoi2bb1_x2	2.04	4.76	9.7104
	aoi2bb1_x4	4.76	4.76	22.6576
	aoi2bb1_x8	8.84	4.76	42.0784
AND-OR-INV	aoi2bb2_x1	2.38	4.76	11.3288
	aoi2bb2_x2	3.40	4.76	16.1840
	aoi2bb2_x4	6.12	4.76	29.1312
	aoi2bb2_x8	11.22	4.76	53.4072
Buffers	buf_x1	1.02	4.76	4.8552
	buf_x2	1.02	4.76	4.8552
	buf_x4	1.02	4.76	4.8552
	buf_x8	1.70	4.76	8.0920
	buf_x16	3.06	4.76	14.5656
	buf_x32	5.78	4.76	27.5128
	buf_x64	11.22	4.76	53.4072
Tri-State Buffers	inv_x1	2.04	4.76	9.7104
	inv_x2	2.72	4.76	12.9472
	inv_x4	4.76	4.76	22.6576
	inv_x8	8.84	4.76	42.0784
	inv_x16	17.00	4.76	80.9200
Clock Enable	clken_x1	6.12	4.76	29.1312
	clken_x2	6.12	4.76	29.1312
	clken_x4	9.86	4.76	46.9336
	clken_x8	13.94	4.76	66.3544
D Flip-Flop	ff_x1	7.82	4.76	37.2232
	ff_x2	9.18	4.76	43.6968
	ff_x4		4.76	0.0000
	ff_x8		4.76	0.0000
ff_ref	ff_ref_x1	7.48	4.76	35.6048
	ff_ref_x2		4.76	0.0000
	ff_ref_x4		4.76	0.0000
	ff_ref_x8		4.76	0.0000
ffhq	ffhq_x1	6.80	4.76	32.3680

	ffhq_x2		4.76	0.0000
	ffhq_x4		4.76	0.0000
	ffhq_x8		4.76	0.0000
ffr	ffr_x1	7.48	4.76	35.6048
	ffr_x2		4.76	0.0000
	ffr_x4		4.76	0.0000
	ffr_x8		4.76	0.0000
ffrhq	ffrhq_x1	6.80	4.76	32.3680
	ffrhq_x2		4.76	0.0000
	ffrhq_x4		4.76	0.0000
	ffrhq_x8		4.76	0.0000
ffrs	ffrs_x1	8.16	4.76	38.8416
	ffrs_x2		4.76	0.0000
	ffrs_x4		4.76	0.0000
	ffrs_x8		4.76	0.0000
fft	fft_x1	6.12	4.76	29.1312
	fft_x2		4.76	0.0000
	fft_x4		4.76	0.0000
	fft_x8		4.76	0.0000
ffthq	ffthq_x1	5.44	4.76	25.8944
	ffthq_x2		4.76	0.0000
	ffthq_x4		4.76	0.0000
	ffthq_x8		4.76	0.0000
fftr	fftr_x1	6.80	4.76	32.3680
	fftr_x2		4.76	0.0000
	fftr_x4		4.76	0.0000
	fftr_x8		4.76	0.0000
fftrhq	fftrhq_x1	6.12	4.76	29.1312
	fftrhq_x2		4.76	0.0000
	fftrhq_x4		4.76	0.0000
	fftrhq_x8		4.76	0.0000
Inverters	inv_x1	0.68	4.76	3.2368
	inv_x2	0.68	4.76	3.2368
	inv_x4	0.68	4.76	3.2368
	inv_x8	1.02	4.76	4.8552
	inv_x16	1.70	4.76	8.0920
	inv_x32	3.06	4.76	14.5656
	inv_x64	5.78	4.76	27.5128
Tri-State Inverters	inv_x1	1.36	4.76	6.4736
	inv_x2	2.38	4.76	11.3288
	inv_x4	4.08	4.76	19.4208

	inv_x8	7.48	4.76	35.6048
	inv_x16	14.28	4.76	67.9728
Gated D-Latch	latchgd_x1	3.74	4.76	17.8024
	latchgd_x2	3.74	4.76	17.8024
	latchgd_x4	6.80	4.76	32.3680
	latchgd_x8	9.52	4.76	45.3152
SR Latch	latchgsr_x1	3.74	4.76	17.8024
	latchgsr_x2	3.74	4.76	17.8024
	latchgsr_x4	6.80	4.76	32.3680
	latchgsr_x8	9.52	4.76	45.3152
2-to-1 MUX	mx2_x1	2.72	4.76	12.9472
	mx2_x2	3.74	4.76	17.8024
	mx2_x4	6.80	4.76	32.3680
	mx2_x8	12.92	4.76	61.4992
mxi2	mxi2_x1	2.04	4.76	9.7104
	mxi2_x2	3.40	4.76	16.1840
	mxi2_x4	6.12	4.76	29.1312
	mxi2_x8	11.90	4.76	56.6440
2-Input NAND	nand2_x1	1.02	4.76	4.8552
	nand2_x2	1.02	4.76	4.8552
	nand2_x4	1.70	4.76	8.0920
	nand2_x8	2.38	4.76	11.3288
nand2b	nand2b_x1	1.70	4.76	8.0920
	nand2b_x2	1.70	4.76	8.0920
	nand2b_x4	2.38	4.76	11.3288
	nand2b_x8	3.06	4.76	14.5656
3-Input NAND	nand3_x1	1.36	4.76	6.4736
	nand3_x2	1.36	4.76	6.4736
	nand3_x4	2.72	4.76	12.9472
	nand3_x8	3.40	4.76	16.1840
nand3b	nand3b_x1	2.04	4.76	9.7104
	nand3b_x2	2.04	4.76	9.7104
	nand3b_x4	3.40	4.76	16.1840
	nand3b_x8	4.08	4.76	19.4208
4-Input NAND	nand4_x1	1.70	4.76	8.0920
	nand4_x2	1.70	4.76	8.0920
	nand4_x4	3.40	4.76	16.1840
	nand4_x8	4.42	4.76	21.0392
nand4b	nand4b_x1	2.38	4.76	11.3288
	nand4b_x2	2.38	4.76	11.3288
	nand4b_x4	4.08	4.76	19.4208

	nand4b_x8	5.10	4.76	24.2760
nand4bb	nand4bb_x1	3.06	4.76	14.5656
	nand4bb_x2	3.06	4.76	14.5656
	nand4bb_x4	4.76	4.76	22.6576
	nand4bb_x8	5.78	4.76	27.5128
2-Input NOR	nor2_x1	1.02	4.76	4.8552
	nor2_x2	1.02	4.76	4.8552
	nor2_x4	2.38	4.76	11.3288
	nor2_x8	4.42	4.76	21.0392
nor2b	nor2b_x1	1.70	4.76	8.0920
	nor2b_x2	1.70	4.76	8.0920
	nor2b_x4	3.06	4.76	14.5656
	nor2b_x8	5.10	4.76	24.2760
3-Input NOR	nor3_x1	1.36	4.76	6.4736
	nor3_x2	2.72	4.76	12.9472
	nor3_x4	4.76	4.76	22.6576
	nor3_x8	8.84	4.76	42.0784
nor3b	nor3b_x1	2.04	4.76	9.7104
	nor3b_x2	3.40	4.76	16.1840
	nor3b_x4	5.44	4.76	25.8944
	nor3b_x8	9.52	4.76	45.3152
OR-AND-INVERT	oai21_x1	1.36	4.76	6.4736
	oai21_x2	2.38	4.76	11.3288
	oai21_x4	3.06	4.76	14.5656
	oai21_x8	6.12	4.76	29.1312
OR-AND-INVERT	oai22_x1	1.70	4.76	8.0920
	oai22_x2	3.40	4.76	16.1840
	oai22_x4	4.42	4.76	21.0392
	oai22_x8	8.84	4.76	42.0784
oai2bb1	oai2bb1_x1	2.04	4.76	9.7104
	oai2bb1_x2	2.04	4.76	9.7104
	oai2bb1_x4	3.40	4.76	16.1840
	oai2bb1_x8	4.76	4.76	22.6576
oai2bb2	oai2bb2_x1	2.38	4.76	11.3288
	oai2bb2_x2	3.40	4.76	16.1840
	oai2bb2_x4	5.78	4.76	27.5128
	oai2bb2_x8	9.86	4.76	46.9336
2-Input OR	or2_x1	1.70	4.76	8.0920
	or2_x2	1.70	4.76	8.0920
	or2_x4	3.06	4.76	14.5656
	or2_x8	5.44	4.76	25.8944

3-Input OR	or3_x1	2.04	4.76	9.7104
	or3_x2	3.40	4.76	16.1840
	or3_x4	5.44	4.76	25.8944
	or3_x8	9.86	4.76	46.9336
TIE-HI (VDD)	tiehi	0.68	4.76	3.2368
TIE-LO (GND)	tielo	0.68	4.76	3.2368
2-Input XNOR	xnor2_x1	3.06	4.76	14.5656
	xnor2_x2	4.42	4.76	21.0392
	xnor2_x4	7.14	4.76	33.9864
	xnor2_x8	12.58	4.76	59.8808
2-Input XOR	xor2_x1	6.12	4.76	29.1312
	xor2_x2	8.84	4.76	42.0784
	xor2_x4	14.28	4.76	67.9728
	xor2_x8	25.16	4.76	119.7616
3-Input XNOR	xnor3_x1	5.78	4.76	27.5128
	xnor3_x2	8.50	4.76	40.4600
	xnor3_x4	13.94	4.76	66.3544
	xnor3_x8	24.82	4.76	118.1432
3-Input XOR	xor3_x1	5.78	4.76	27.5128
	xor3_x2	8.50	4.76	40.4600
	xor3_x4	13.94	4.76	66.3544
	xor3_x8	24.82	4.76	118.1432

Appendix II – UTK Standard I/O Library Summary

(For SUNY Poly 65 nm CMOS-memristor Process Technology)

Described below are the primary I/O padcells included in the "stdio_10lpe" library developed at UTK for the SUNY Poly process. These cells include ESD protection, including a few options for RC clamps that are useful for large or small integrated circuits. All digital and analog cells have been simulated for ESD events and expected delay/timing characteristics. The pad for each cell is 100 um across and the area beneath for ESD and other logic (e.g. tapered buffers) leads to a pitch of 150 um between neighboring pads. This somewhat larger pitch is expected to allow for lower cost packaging options, considering these cells are designed specifically for research focused test chips such as RAVENS.

Cell Type	Name in Library	Purpose
Power Supply	pad_gnd	Supplies Ground
	pad_vdd_0v6	600 mV Mid-Rail Supply
	pad_vdd_1v2	1.2 V Supply (Core Logic Supply)
	pad_vdd_2v5	2.5 V Supply (Off-Chip/PCB I/O Supply)
	pad_vdd_3v3	3.3 V Supply (Off-Chip/PCB I/O Supply)
Digital I/O	pad_dig_inp2	Digital Input (Off-Chip: 3.3 V; Core: 1.2 V)
	pad_dig_out2	Digital Output (Off-Chip: 3.3 V; Core: 1.2 V)
	pad_dig_gpio	GPIO (Off-Chip: 3.3 V; Core: 1.2 V)
	pad_dig_inp3v3	Digital Input (Off-Chip: 3.3 V; Core: 3.3 V)
Analog	pad_analog2	Analog Pin WITH Current Limiting Resistance
	pad_analog_nores	Analog Pin WITHOUT Current Limiting Resistance
RC Clamps	pad_rcclamp_cdm2	RC Clamp for Full Pad Frames
	pads_rcclamp_small	RC Clamp for Small 12x2 Standalone Structures
Corners	corner_nologo_bot	Bottom Corner Cell – No Logo
	corner_nologo_top	Top Corner Cell – No Logo
	corner_utk_bot	Bottom Corner Cell – Includes Power T
	corner_utk_top	Top Corner Cell – Includes Power T
Frames (Examples)	frame_utk_trial	110-Pin Pad Frame – Used for RAVENS
	pads_2x12_esd_rcclamp	2x12 Small Frame – Includes ESD
	pads_2x12_raw	2x12 Small Frame – No ESD Protection

Appendix III – Digital RAVENS Processor Model (Header)

```
#pragma once
#include "framework.hpp"
#include "gnp.hpp"
#include "event.hpp"
#include "processor.hpp"

namespace Ravens
{
  using neuro::Network;
  using neuro::Spike;
  using neuro::PropertyPack;
  using neuro::EdgeMap;
  using neuro::NodeMap;
  using neuro::Property;
  using event::Event;
  using event::EventSystem;
  using gnp::Leak;
  using gnp::STDPType;
  using std::vector;

  class Processor;

  class RavensSTDP: public STDPType
  {
    public:
    /**
     * @brief the set of fires to consider
     *
     * the set of fires to consider, primarily indexed on time since fire,
     * secondarily indexed on which synapse id fired at that time
     */
    std::vector<std::vector<size_t> >fireGrid;
    /**
     * @brief time last modified
     *
     * the last time this stdp object was updated (should be the last time
     * the neuron was updated)
     */
    double lastModified;
    /**
     * @brief The last time the attached neuron fired
     *
     */
    double lastFire;
    /**
     * @brief stdp length
     *
     * the number of cycles to consider for stdp operations -- the length of fireGrid
     */
    size_t numCycles;
    /**
     * @brief stdp curve parameter
     *
     * how much the adjustment of the synapse is altered, as decayRate^(deltaCycles -1)
     */
    double decayRate;
    /**
     * @brief a link to the processor object
     *
     */
    Processor *sim;

    /**
     * @brief Construct a new Adjustable S T D P object
     *
     * @param cycles the number of cycles to consider pre/post fire
     * @param dev the parent gnp processor
```

```
   */
  RavensSTDP(size_t cycles, gnp::Processor *dev);
  /**
   * @brief Construct a new Adjustable S T D P
   *
   * @param cycles the number of cycles to consider pre/post fire
   * @param decayRate decay of adjust strength
   * @param dev the parent gnp processor
   */
  RavensSTDP(size_t cycles, double decayRate, gnp::Processor *dev);
  /**
   * @brief Destroy the Adjustable S T D P object
   *
   */
  virtual ~RavensSTDP();

  /**
   * @brief Record an accumulation event
   *
   * @param e the event
   * @return true
   * @return false
   *
   * Accumulate events are stored in the fireGrid by how many cycles since the last update
   */
  virtual bool Accumulate(Event *e);
  /**
   * @brief record a neuron fire
   *
   * @param e the neuron fire event
   * @return true
   * @return false
   *
   * A neuron fire event triggers adjustment events in recently fired synapses,
   * and sets up to cause depression events for the next stdp_length cycles
   */
  virtual bool Neuron_Fire(Event *e);
  /**
   * @brief update the tracker
   *
   * @param time the current time to adjust to
   * @return true
   * @return false
   *
   * Updating the tracker will move some of the stored events,
   * and may cause events to fall out of the tracker if they are too old.
   */
  virtual bool update(double time);
  /**
   * @brief Is the neuron in the refractory period of 2*stdplen after firing?
   *
   * @param time the current time in question
   * @return true yes, the neuron is in stdp
   * @return false no, the neuron is not in STDP-based refractory period,
   * but may be in refractory for other reasons
   */
  virtual bool STDP_Refractory(double time);
  /**
   * @brief reset all runtime info
   *
   */
  virtual void Reset();
};

class Neuron: public gnp::Neuron
{
  public:

  Neuron(neuro::Node *n, Processor *dev, size_t _id, size_t _eonsid);
  virtual void addCharge(Event *e);
  virtual void endAccum(Event *e);
```

```
    virtual void fire(Event *e);
    virtual void update(double time);
    virtual void Reset();
    virtual void ClearTracking();
    static void neuronProperties(nlohmann::json &settings, PropertyPack &pp);
    virtual double getPreFire();
    virtual double getPostFire();

       /* After fire, enter abs refrac, where no charge is accumulated.
     * After abs refrac, enter rel refrac, where charge can be accumulated,
     * but neuron resting potential is set to refractory resting potential.
     */
    double abs_refrac_period = 0.0, rel_fefrac_period = 0.0;
    bool abs_refrac = false, rel_refrac = false;
    double refrac_resting_potential = 0.0;
    double resting_potential = 0.0;

    // Amount of charge leaked every time step.
    double leak_amt = 0.0;
  };


  class Synapse: public gnp::Synapse //placeholder
  {

    /** Synapse constructor, now given the different pointer for framework 6 */
    Synapse(std::unique_ptr<neuro::Edge> e, Processor *dev, SynapseType *st, size_t from,
            size_t to, bool keepWeight, size_t id);
    virtual ~Synapse();

    /** This is where the synapse half of the work of adjust events occur,
     *  given signaling from the STDP system of the post-synaptic neuron */
    virtual void adjust(Event *e);

    /** This causes the synapse to process a fire from the pre-synaptic neuron
     * Note that the Ravens synapse has to check the output voltage from the neuron's type
     */
    virtual void getFire(double time);

    /** have the synapse restore its weight and any other internal information */
    virtual void Reset();
  };

  /**
   * @brief the Ravens simulator0
   *
   */
  class Processor: public gnp::Processor
  {
    public:

    double cycleTime;
    double curSimTime;

    bool CrossbarMode;

    Processor(nlohmann::json &settings);
    virtual ~Processor();
    virtual bool load_network(Network *n, int network_id = 0);
    virtual void run(double duration, int network_id=0);
    virtual void clear(int network_id=0);
    virtual void clear_activity(int network_id=0);
    virtual PropertyPack get_properties();

    protected:
    virtual bool Apply_Pulse(int _network_id, int input_id, double val, double time);
  };
}
```

Appendix IV – Memristive RAVENS Processor Model (Header)

```
#pragma once

#include <map>
#include "framework.hpp"
#include "io_stream.hpp"
#include "mravens_network.hpp"
#include "nlohmann/json.hpp"

using namespace neuro;
using namespace std;

namespace mravens {

  class Processor : public neuro::Processor
  {
    public:

    Processor(json &params);
    ~Processor();

    bool load_network(neuro::Network* n, int network_id = 0);
    bool load_networks(std::vector<neuro::Network*> &n);
    void clear(int network_id = 0);

    /* Queue spike(s) as input to a network or to multiple networks */

    void apply_spike(const Spike& s, int network_id = 0);
    void apply_spike(const Spike& s, const vector<int>& network_ids);

    void apply_spikes(const vector<Spike>& s, int network_id = 0);
    void apply_spikes(const vector<Spike>& s, const vector<int>& network_ids);

    /* Run the network(s) for the desired time with queued input(s) */

    void run(double duration, int network_id = 0);
    void run(double duration, const vector<int>& network_ids);

    /* Get processor time based on specified network */
    double get_time(int network_id = 0);

    /* Output tracking.  See the markdown for a detailed description of these.  */

    bool track_output_events(int output_id, bool track = true, int network_id = 0);
    bool track_neuron_events(uint32_t node_id, bool track = true, int network_id = 0);

    /* Access output spike data */

    double output_last_fire(int output_id, int network_id = 0);
    vector <double> output_last_fires(int network_id = 0);

    int output_count(int output_id, int network_id = 0);
    vector <int> output_counts(int network_id = 0);

    vector <double> output_vector(int output_id, int network_id = 0);
    vector < vector <double> > output_vectors(int network_id = 0);

    /* Spike data from all neurons. */

    vector <int> neuron_counts(int network_id = 0);
    vector <double> neuron_last_fires(int network_id = 0);
    vector < vector <double> > neuron_vectors(int network_id = 0);

    /* Remove state, keep network loaded */
    void clear_activity(int network_id = 0);

    /* Network and Processor Properties.  The network properties correspond to the Data
       field in the network, nodes and edges.  The processor properties are so that
```

```
      applications may query the processor for various properties (e.g. input scaling,
      fire-on-threshold vs fire-over-threshold. */

   PropertyPack get_network_properties();
   json get_processor_properties();

   protected:

   mravens::Network* get_mravens_network(int network_id);
   map <int, mravens::Network*> networks;

   double min_weight;
   double max_weight;

   double min_threshold;
   double max_threshold;

   uint32_t min_delay;
   uint32_t max_delay;

   double min_standard_resting_potential;
   double max_standard_resting_potential;

   double min_refractory_resting_potential;
   double max_refractory_resting_potential;

   int min_absolute_refractory_period;
   int max_absolute_refractory_period;

   int min_relative_refractory_period;
   int max_relative_refractory_period;

   double min_leak;
   double max_leak;

   int spike_value_factor;
   int max_input_edges_per_node;
   int max_output_edges_per_node;

   int clock_time_unit;

   double stdp_rate;

   bool integration_delay;

   vector <double> stdp_potentiation;
   vector <double> stdp_depression;

   IO_Stream log;
  };
}
```

Appendix V – Pinouts for Standalone 12x2 Test Structures

Test Structure:1

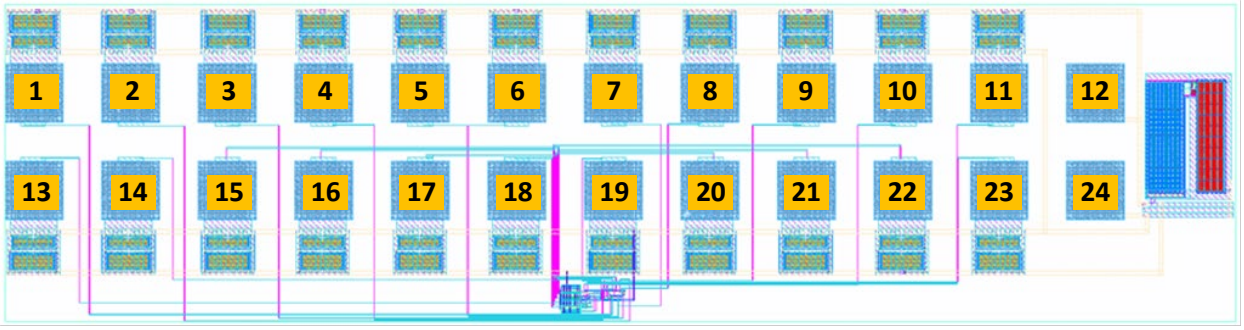

Table 1: Pad frame details for test structure 1

Pin Number	Pin Name	Pin Type	Pin Description
1	Vnset2	Analog Output	Current compliance for memristor2
2	FormB2	Digital Input	Forming memristor2
3	Mtop2	Analog Input	Memristor2 top electrode
4	Mbot2	Analog Output	Memristor2 bottom electrode
5	ResetB2	Digital Input	Rest memristor2 0V-3.3V enable
6	SetB2	Digital Input	Set memristor2 3.3V-0V enable
7	Reset2	Digital Input	Reset memristor2 3.3V-0V enable
8	Out2	Analog Output	Memristor2 output current
9	Vnset3	Analog Input	Current compliance for memristor3
10	Mtop3	Analog Input	Memristor3 top electrode
11	Mbot	Analog Output	Memristor1 bottom electrode
12	VDD3v3	Power	3.3V power
13	VDD1v2	Power	1.2V power
14	Vncm	Analog Reference	Reference for synapse output current
15	Form	Digital Input	Form memristor1 0V-1.2V enable
16	Set	Digital Input	Set memristor1 0V-1.2V enable
17	Reset	Digital Input	Reset memristor1 0V-1.2V enable
18	Read	Digital Input	Read memristor1 output current enabled
19	Synout	Analog Output	Current output from memristor1
20	Vnset	Analog Input	Current control node for memristor1
21	Vrst	Analog Reference	Reset voltage applied across memristors
22	Vst	Analog Reference	Set voltage applied across memristors
23	Mtop	Analog Output	Memristor1 top electrode
24	GND	Power	Ground

Test Structure:2

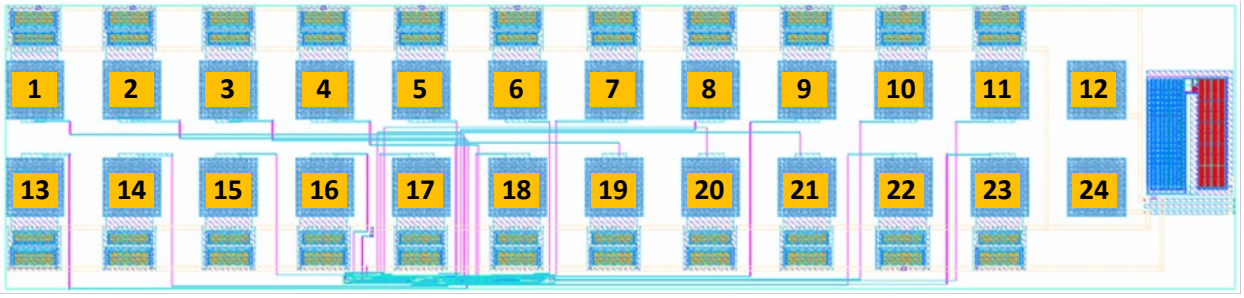

Table 2: Pad frame details for test structure 2

Pin Number	Pin Name	Pin Type	Pin Description
1	DACV<0>	Digital Input	Bit for set/form current compliance
2	DACV<1>	Digital Input	Bit for set/form current compliance
3	DACV<2>	Digital Input	Bit for set/form current compliance
4	Readref	Analog Reference	Reference voltage determining read current compliance
5	SYNOUT	Digital Output	Clocked synapse output
6	CLK	Digital Input	Clock for synapse control
7	Vnset	Digital Input	Current control node
8	PGR	Analog Output	Voltage node after memristor cell
9	OUT<0>	Digital Output	Output current level digitized
10	OUT<1>	Digital Output	Output current level digitized
11	OUT<2>	Digital Output	Output current level digitized
12	VDD3v3	Power	3.3V power
13	VDD1v2	Power	1.2V power
14	Vncm	Analog Reference	Reference for synapse output current
15	Form	Digital Input	Form memristor 0V-1.2V enable
16	Set	Digital Input	Set memristor 0V-1.2V enable
17	Reset	Digital Input	Reset memristor 0V-1.2V enable
18	Read	Digital Input	Read memristor output current enabled
19	syn_out	Analog Output	Current output from memristor
20	Mbot	Analog Output	Memristor bottom electrode
21	Vrst	Analog Reference	Reset voltage applied across memristors
22	Vst	Analog Reference	Set voltage applied across memristors
23	Mtop	Analog Output	Memristor top electrode
24	GND	Power	Ground

Test Structure:3

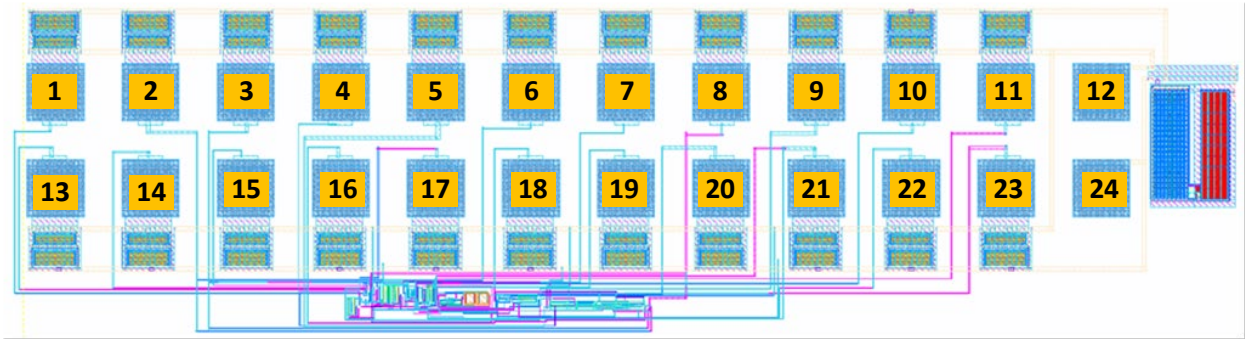

Table 3: Pad frame details for test structure 3

Pin Number	Pin Name	Pin Type	Pin Description
1	VDD1v2	Power	1.2V power
2	Form	Digital Input	Form memristor 0V-1.2V enable
3	Set	Digital Input	Set memristor 0V-1.2V enable
4	Reset	Digital Input	Reset memristor 0V-1.2V enable
5	Read	Digital Input	Read memristor 0V-1.2V enable
6	Formref	Digital Input	Form reference memristor 0V-1.2V enable
7	Setref	Digital Input	Set reference memristor 0V-1.2V enable
8	Prog<0>	Analog Output	Voltage generated from DACV values
9	Vrst	Analog Reference	Reset voltage applied across memristors
10	Vst	Analog Reference	Set voltage applied across memristors
11	Vmid	Analog Reference	Midrail voltage reference 0.6V
12	GND	Power	Ground
13	DACV<0>	Digital Input	Bit for set/form current compliance
14	DACV<1>	Digital Input	Bit for set/form current compliance
15	DACV<2>	Digital Input	Bit for set/form current compliance
16	CLK	Digital Input	Clock for synapse control
17	Resetref	Digital Input	Reset reference memristor 0V-1.2V enable
18	NCMref	Analog Output	Reference for synapse output current
19	Readref	Digital Input	Read reference memristor 0V-1.2V enable
20	Vnset	Analog Output	Current control node
21	Ifb	Analog Output	Internal feedback voltage from reference
22	Refread	Analog Reference	Reference voltage determining read current compliance
23	Synout	Analog Output	Clocked synapse output
24	VDD3v3	Power	3.3V power

Test Structure:4

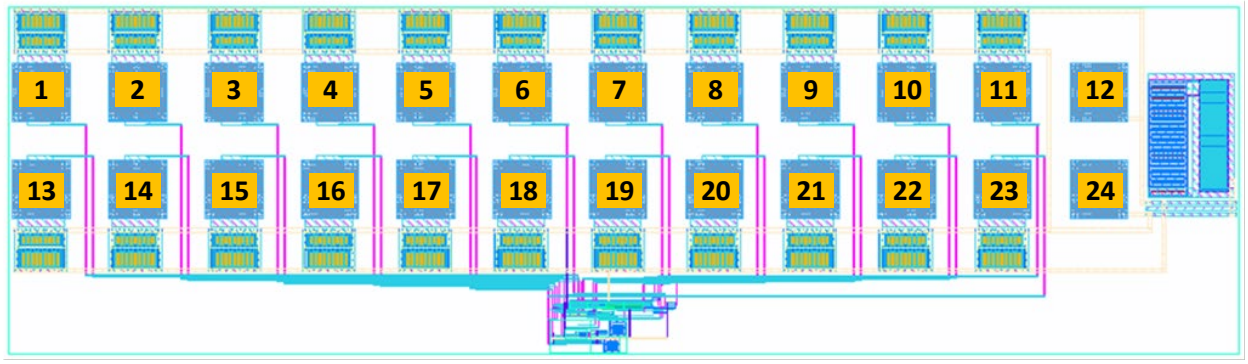

Table 4: Pad frame details for test structure 4

Pin Number	Pin Name	Pin Type	Pin Description
1	DACV<0>	Digital Input	Bit for set/form current compliance
2	DACV<1>	Digital Input	Bit for set/form current compliance
3	DACV<2>	Digital Input	Bit for set/form current compliance
4	CLK	Digital Input	Clock for synapse control
5	Post	Digital Input	Reset reference memristor 0V-1.2V enable
6	Mtop	Analog Output	Memristor top electrode
7	PGR	Analog Output	Voltage node after memristor cell
8	vlearn	Analog Output	Summed voltage for update programming
9	readref	Analog Reference	Reference voltage determining read current compliance
10	Prog1	Analog Output	Voltage generated from DACV values
11	Mbot	Analog Output	Memristor bottom electrode
12	VDD3v3	Power	3.3V power
13	VDD1v2	Power	1.2V power
14	Form	Digital Input	Form memristor 0V-1.2V enable
15	Inset	Digital Input	Set memristor 0V-1.2V enable
16	Inreset	Digital Input	Reset memristor 0V-1.2V enable
17	Pre	Digital Input	Read memristor 0V-1.2V enable
18	NCM	Analog Output	Reference for synapse output current
19	Vnset	Analog Output	Current control node
20	SYNOUT	Analog Output	Clocked synapse output
21	Vrst	Analog Reference	Reset voltage applied across memristors
22	Vst	Analog Reference	Set voltage applied across memristors
23	LEARN	Digital Input	STDP learning control 0V-1.2V enable
24	GND	Power	Ground

Test Structure:5

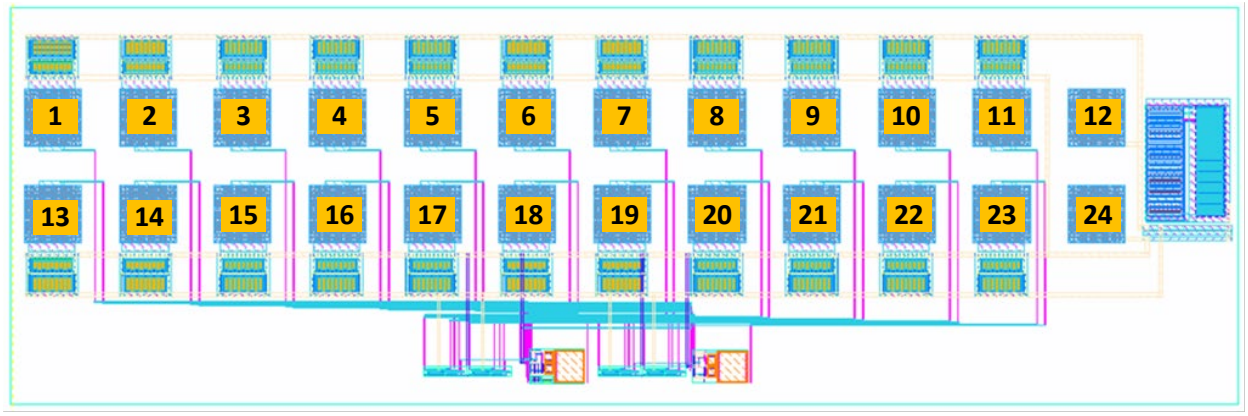

Table 5: Pad frame details for test structure 5

Pin Number	Pin Name	Pin Type	Pin Description
1	Ior		
2	ioIref		
3	Leak<0>		
4	Leak<1>		
5	Leak<2>		
6	Leak<3>		
7	VDD1v2	Power	1.2V power
8	Read3		
9	Read4		
10	Vout1		
11	Vout2		
12	VDD3v3	Power	3.3V power
13	En		
14	Reset		
15	Vth		
16	Vmid	Analog Reference	Midrail voltage reference 0.6V
17	CLK	Digital Input	Clock for synapse control
18	In1		
19	In2		
20	NCM	Analog Output	Reference for synapse output current
21	Vnset	Analog Output	Current control node
22	Read1		
23	Read2		
24	GND	Power	Ground

All Test Structure in the chip


TS_1_2
TS_3_1
TS_3_2
TS_4_1
TS_4_2
TS_5
TS_1_2
TS_2_1
TS_2_2

7.0 List of Acronyms

AFRL: Air Force Research Laboratory
ASIC: Application Specific Integrated Circuit
BPTT: Back Propagation Through Time
CMOS: Complementary Metal Oxide Semiconductor
DANNA: Dynamic Adaptive Neural Network Array
DRC: Design Rule Check
EO: Evolutionary Optimization
EONS: Evolutionary Optimization for Neuromorphic Systems
ESD: Electrostatic Discharge
FIFO: First In First Out
FPGA: Field Programmable Gate Array
HRS: High Resistance State
IC: Integrated Circuit
I/O: Input/Output
LIF: Leaky Integrate-and-Fire
LRS: Low Resistance State
LTD: Long Term Depression
LTP: Long Term Potentiation
mrDANNA: Memristive Dynamic Adaptive Neural Network Array
NIDA: Neuroscience Inspired Dynamic Architecture
NMOS: n-Type Metal Oxide Semiconductor
PCB: Printed Circuit Board
PMOS: p-Type Metal Oxide Semiconductor
RAVENS: Reconfigurable and Very Efficient Neuromorphic System
R&D: Research & Development
ReRAM: Resistive Random-Access Memory
RISC-V: Reduced Instruction Set Computer - 5
RISP: Reduced Instruction Spiking Processor
RNA: Reconfigurable Neuromorphic Array
SOC: System on a Chip
SPICE: Simulation Program with Integrated Circuit Emphasis
SRAM: Static Random Access Memory
STDP: Spike-Timing-Dependent Plasticity
SUNY: State University of New York
TDM: Time Division Multiplexing
UAV: Unmanned Aerial Vehicle
UTK: The University of Tennessee, Knoxville

VHDL: VHSIC Hardware Description Language
VHSIC: Very High Speed Integrated Circuit
VLSI: Very Large Scale Integration