High-Level Simulation for Spiking Neuromorphic Computing Systems

Nicholas D. Skuda
EECS Department
University of Tennessee
Knoxville, Tennessee
Email: nskuda@vols.utk.edu

Catherine Schuman
Computational Data Analytics
Oak Ridge National Laboratory
Oak Ridge, Tennessee
Email: schumancd@ornl.gov

Gangotree Chakma, James S. Plank,
and Garrett S. Rose
EECS Department
University of Tennessee
Knoxville, Tennessee
Email:[gchakma, garose, plank]@vols.utk.edu

***Abstract*—Neuromorphic computing systems are alternatives to conventional microprocessors, often built from unconventional hardware. Designing and evaluating these systems requires multiple levels of simulation, from the device level to the circuit level to the system level. In this paper, we describe the system level simulator of a neuromorphic computing system based on memristors. We compare it to a circuit level simulator of the same system, both verifying its accuracy and demonstrating its performance improvement. We argue that system level simulation is an essential part of the design process of neuromorphic systems.**

I. Introduction

With the end of Moore's law and Dennard scaling, a variety of alternative computer architectures are being explored, one of which is neuromorphic computers. Neuromorphic computers are computers in which the underlying architecture and function is inspired by biological brains. Thus, the functional units of neuromorphic computers tend to be neurons and synapses, while the operation of such systems tend to be characterized by massively parallel computation with event-driven, spike-based communication [1].

The current primary way to program neuromorphic systems is to utilize training or learning algorithms from neural network literature, including back-propagation algorithms and Hebbian learning-style algorithms such as spike-timing dependent plasticity (STDP). Though many neuromorphic chips include implementations of STDP, most do not include a notion of on-chip supervised learning algorithms such as back-propagation. However, they may be utilized in training for the inference step of some algorithms (e.g., the forward pass step of the back-propagation algorithm) using a chip-in-the-loop approach. At present, as neuromorphic devices are still being developed, it is often not feasible to utilize the device itself even as a part of a chip-in-the-loop training algorithm. Circuit-level simulators such as Cadence Spectre can provide very detailed estimates of how an actual device will behave, but the simulations are typically too slow to do anything meaningful with respect to training or learning. In order to truly understand how a neuromorphic system will or should train and/or learn, we need to develop high-level simulators of neuromorphic devices.

In this work, we discuss our neuromorphic hardware system, which is a mixed analog-digital hardware implementation that utilizes memristors to realize the synapses. We present an implementation of a high-level simulator of our neuromorphic hardware system, and we compare and verify the results from this simulator using low-level circuit simulation software. Using this high-level simulator, we can more quickly produce "programs" for our neuromorphic devices, in the form of instances of spiking neural networks, and validate the performance for the hardware on those real applications.

II. Related Work

A. Memristor-based neuromorphic systems

Memristors, since their official "discovery" in 2008 [2], have become an increasingly popular component of neuromorphic systems. Memristors have been commonly used to implement synapses in neuromorphic systems for a variety of reasons, including that they allow for high density and high connectivity of synapses and their ease of integration in a learning mechanism such as STDP [3], [4]. Since memristor-based hardware is relatively rare in today's physical hardware implementations, we rely primarily on experimental results from physical memristors and simulated results of larger circuits [5], [6].

B. Software Simulators for Neuromorphic Systems

Compared to the overall number of neuromorphic hardware implementations, there have been relatively few software simulators developed. However, software simulators for neuromorphic systems can play a variety of vital roles in the development and use of neuromorphic systems. For example, neuromorphic simulators have been developed to verify and validate the behavior of hardware, to train or develop programs or neural networks for neuromorphic hardware, and to help in the co-design process of the system. IBM's TrueNorth hardware has had multiple simulators, developed both inside IBM from the same developers as the hardware [7] and externally, with the development of the NeMo simulation platform [8]. These simulators were developed to understand the capabilities of the chip on various applications and to estimate performance on those applications. Memristor-based neuromorphic hardware simulators have also been developed, both for co-design and validation and verification purposes [9], [10]. As noted above, utilizing neuromorphic simulators for co-design is a popular use case, because design decisions can be made in the simulator and evaluated before actually implementing them in hardware [11], [12]. Finally, neuromorphic simulators have been commonly used to help understand how to train or program neuromorphic systems, as well as to perform off-line training to produce programs that can be deployed onto the device [13], [14].

III. Hardware Description

In the literature, several implementations [15], [16], [17] of neuromorphic hardware have been described. Our hardware implementation is based on a memristive mixed-signal circuit design. As mentioned in Section II-A, memristors have been popular in neuromorphic computing architectures, specifically in implementing synapses. Memristors are nano-scale devices with the feature of non-volatility, making them suitable to leverage in synapse design. In our hardware, we are following a twin memristive synapse architecture [18] because of its advantage of implementing both positive and negative synaptic weights and also its ability to include an on-line learning mechanism (STDP). The memristive synapse is initially trained off-line with our high-level simulator, and then while testing, we use STDP to make the network more efficient by leveraging on-line, reinforcement learning. The memristor model used in this paper follows that described in [19]. The model considers different LRS and HRS values of the memristor along with positive (negative) threshold, V_{tp} (V_{tn}) and positive (negative), t_{swp} (t_{swn}) switching time of the memristors.

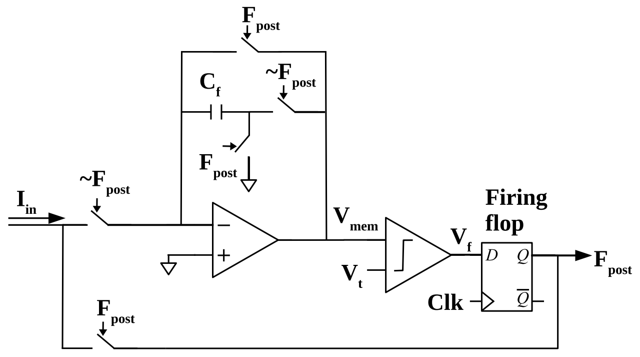


Fig. 1: Analog Integrate and Fire (IAF) Neuron.

The neuron model shown in Fig. 1 is based on an analog integrate and fire neuron [20] in which the integrator accumulates all the incoming charges from the connected synapses as the input current I_{in} and generates a firing spike, F_{post} when the accumulated charge, V_{mem} crosses the specified threshold, V_t. After firing, the neuron resets itself and goes back to the initial stage to accept new input signals from the synapses. The neuron implementation also serves a crucial part in implementing STDP by providing feedback signals to the synapse inputs post-fire.

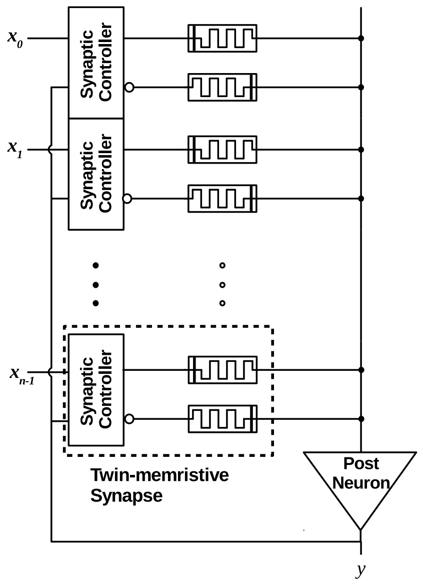


Fig. 2: Single memristive neuromorphic core consisting of single neuron and multiple synapses.

With these analog neurons and memristive synapses, we have designed a complete neuromorphic system consisting of several neuromorphic cores. Each core shown in Fig. 2 has a single neuron and multiple input synapses connected to the neuron. At present, we assume eight synapses per neuron, but this may be updated as the co-design process continues. The cores are connected to each other so that the signals can be sent and received in a neural network.

IV. Hardware Simulation

In order to accurately model the behavior of the device, but also be able to evaluate the performance of the device on real applications, we utilize low-level simulation (Spectre) and implement a high-level simulation.

A. Low-Level Simulation

For low-level simulation, we use Cadence Spectre as our circuit-level simulator. To initialize the simulation, we use a network generated from the high-level simulator which has been trained off-line. We create the schematic netlist and initialize the synaptic weights based on the application specific network. The model files for memristor and the transistors have also been included while setting up the simulation. We create a stimulus file to send inputs to the network and track circuit activity. We can also analyze the energy consumed by each neuron and synapse per spike from the low-level simulation by using the current flowing through each of the component.

B. High-Level Simulation

Our high-level simulator uses data gathered from the low-level simulator to simulate the behaviors seen in the low-level simulator. By abstracting away the individual devices to focus on neurons and synapses, the high-level simulator runs much more quickly, allowing for learning to happen at a practical pace. The high-level simulator is written in C++, and uses a discrete-event simulation to simulate the behavior of the device. As such, we only simulate activity as it occurs, rather than simulating idle behavior of all neurons and synapses in the device.

In order to evaluate the device on real applications, we include it as a neuromorphic model in a broader neuromorphic software framework. This framework includes both a training component (based on evolutionary optimization), as well as a set of well-established applications [21], [22]. In order for this memristor-based spiking neuromorphic system to be included in the software framework, we implement a variety of functions that enable other software to interact with the model. For example, we implement the ability to apply input to the simulation of the device, read output from the simulation, and run the simulation to execute real behavior. The application codes can then interact with the model through these functions. We also implement the appropriate functions to enable evolutionary optimization to operate on the model, including reproduction operations that will take one or two networks in the model and produce one or two new networks through recombination and/or mutation.

For verification purposes, the high-level simulation of the hardware creates and outputs detailed event logs to match the circuit activity tracked in the low-level simulator for essential elements such as the accumulation capacitor in each neuron and the two memristors in each synapse. These event logs can then be processed to produce output images that indicate the activity in a run of the network on any given input combination or to collect into larger datasets for broad analysis.

V. Verification Setup

To perform verification of the high level simulator compared to Cadence Spectre's simulation results, both simulators are programmed with the same network, a small classification network used for solving Iris flower classification (Fig. 3). This is a well-established application and is ideal for verification purposes [22]. In this verification setup, each of the input and output neurons are connected via non-learning synapses to connections off-chip acting as network-level IO. A single test case is used to reduce runtime and the inputs are programmed as in the high-level application through the neuromorphic library. Those inputs spikes are then fed into both networks, in this case all simultaneously at the beginning of simulation.

The network that was generated used only 7 of the 12 possible inputs used in Iris classification (where the 12 inputs come from splitting each of the four input values into three bins); furthermore it includes a vestigial hidden neuron that cannot fire as it does not have any source of input spikes. Each of the synapses, identified in Fig. 3 by directional lines, uses a delay of one cycle before the spike arrives at the post neuron. Note that although there are no functional hidden neurons, there are connections between input neurons, so some of those neurons are used as both input and "hidden" neurons. Memristor resistances are preset to the same values in both simulations to ensure identical conditions.

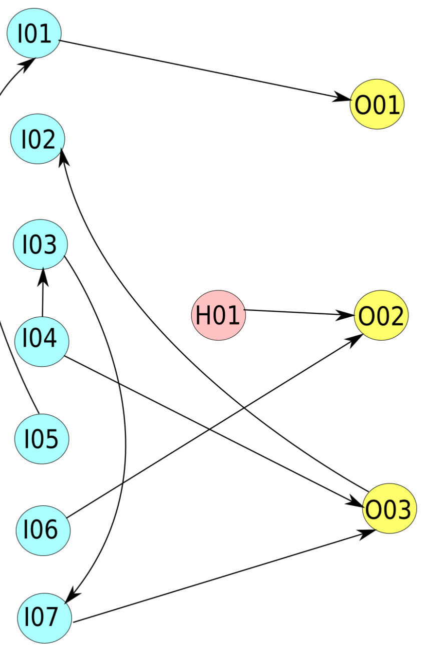


Fig. 3: Iris classification network. The input neurons noted with an I are input neurons connected to a non-learning input synapse. Output neurons are marked with an O and they have outgoing synapses that connect off of the network. Internal synapses are the directional lines. This network does not connect to 5 of the input synapses and has a vestigial hidden neuron that does not receive inputs

Runtime for each simulation is determined by using the high precision timers built into the operating system, running on identical systems with 4th generation Intel i7 processors. Output graphs for Cadence Spectre are produced after runtime is finished. Output graphs for the high level simulator are constructed via Python script by processing the event log from the simulator. Both logs and graphs are verified against one another to ensure that events such as firing, delay, on-line learning, and accumulation occur correctly in both simulators at the same time.

Accuracy of the network itself is judged using the high-level model in accordance to parameters by which the network was trained (Table I). In this case inputs come in as four decimal values, and are then split into one of three input synapses depending on that value. Output values are judged by the highest number of output spikes, with ties being associated with the lower-numbered output. The activity of either model is then used to develop an energy usage based on design information from Cadence Spectre.

TABLE I: Verification run setup

Input Spikes	IO2, IO3, IO4
Input Values	6.1 3.0 4.9 1.8
Expected Output	Most spikes come from O03
Output Classification	Iris Virginica

VI. Results

The most dramatic difference between the two simulators is runtime – Cadence Spectre ran for 632.6 seconds across 8 processing cores and the high level simulator completed in 5 milliseconds on one core. This strongly supports the hypothesis that the high level simulator will run much faster and provide accurate findings and thus will be very useful in quickly performing supervised learning.

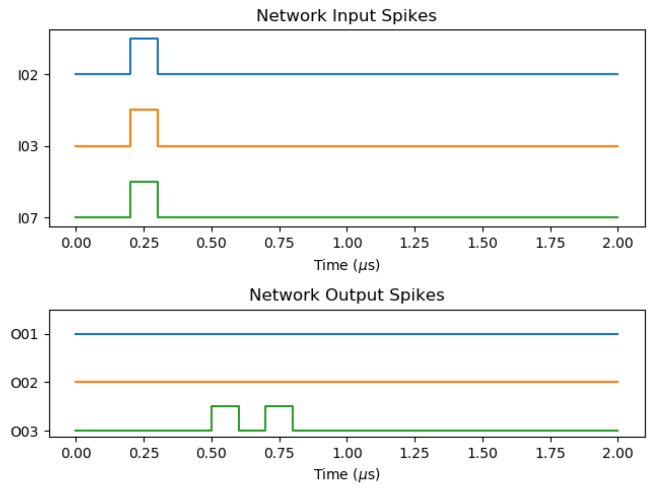
For accuracy, one obvious difference between the high-level and low-level simulators is that the low-level simulator is extremely time-accurate when showing events as they occur. Conversely the high-level simulator is cycle accurate (Fig. 4); events are processed by cycle and then by group, such that all events that are grouped together have no influence upon one another and thus can be processed in batches. This greatly speeds execution time but provides less detail on the graphs as each event visually takes up the entire cycle.

All pre- and post-neuron fires occur on the same cycle in both the Cadence Spectre results and the high-level simulator results. Both simulators fired twice on Output 3 (Fig. 4), correctly classifying the flower as Iris Virginica. By entering the event information from each simulation, both simulators agreed with an energy usage estimate of 7.45 pJ for this run.

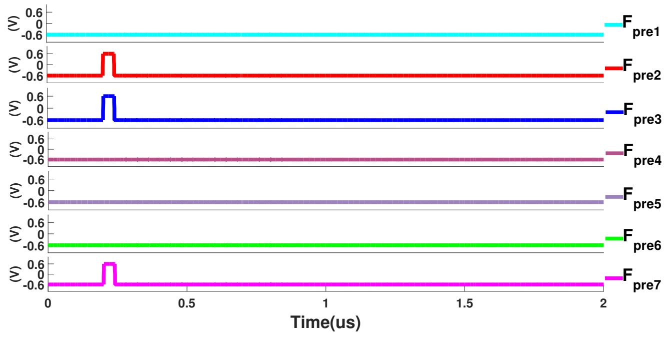
VII. Future Work & Conclusions

We will continue verifying the performance of both simulators using larger networks, more inputs, and longer runs. The high-level simulator will be expanded to work on additional neuron and synapse designs as well as different network setups. Additionally, we will work on using the fast high-level simulator to generate large datasets to better understand changing memristor characteristics as new devices are unveiled. Finally, we plan on exploring deeper integration of power usage estimates within the simulator, and potentially even the learning system to optimize towards networks with lower power usage as well as strong application performance.

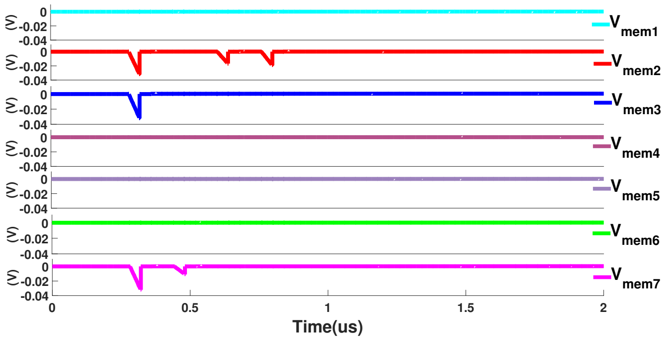
Performance is a key consideration when performing any operation on a neural networks, especially learning; this drives the need for high-speed simulation of neuromorphic hardware. Our high-level simulation outperformed the circuit simulation by a factor of over 100,000. These faster run times allow for much faster iteration while training networks using evolutionary optimization and network verification. Cycle-level event simulation lacks the resolution of a direct hardware simulation in sub-cycle time intervals, but on a cycle level our high-level model directly matched the output of the low-level simulation.



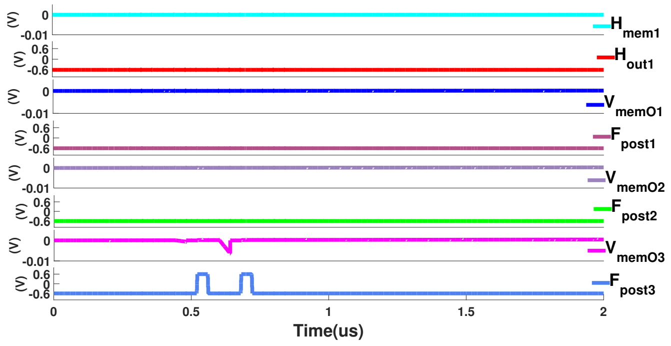
(a) High Level Simulation Inputs and Output



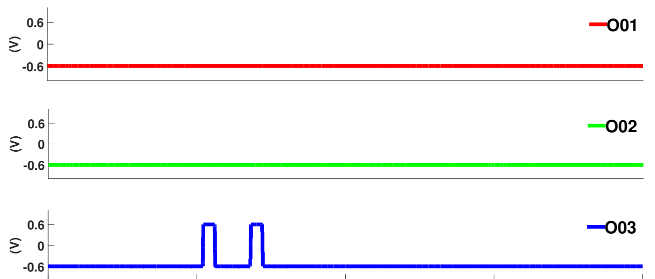
(b) Cadence Spectre Output



(c) Cadence Spectre Output



(d) Cadence Spectre Output



This accuracy allows us to utilize the much faster process to enable quick results in generating networks.

Acknowledgment

The authors would like to thank Dr. Mark Dean, Md. Musabbir Adnan, Sagarvarma Sayyaparaju, Sherif Amer and Ryan Weiss from the University of Tennessee, Knoxville for interesting and useful discussions on this topic.

This material is based in part on research sponsored by Air Force Research Laboratory under agreement number FA8750-16-1-0065 and the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Reserach, under contract number DE-AC05-00OR22725.

References

[1] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," arXiv:1705.06963, May 2017. [Online]. Available: https://arxiv.org/abs/1705.06963

[2] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[3] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[4] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, "Stdp and stdp variations with memristors for spiking neuromorphic learning systems," *Frontiers in neuroscience*, vol. 7, p. 2, 2013.

[5] R. B. Jacobs-Gedrim, S. Agarwal, K. E. Knisely, J. E. Stevens, M. S. van Heukelom, D. R. Hughard, J. Niroula, C. D. James, and M. J. Marinella, "Impact of linearity and write noise of analog resistive memory devices in a neural algorithm accelerator," in *IEEE International Conference on Rebooting Computing (ICRC 2017)*, Washington, DC, November 2017, pp. 160–169.

[6] W. Olin-Ammentorp, K. Beckmann, J. E. Van Nostrand, G. S. Rose, M. E. Dean, J. S. Plank, G. Chakma, and N. C. Cady, "Applying memristors towards low-power, dynamic learning for neuromorphic applications," in *42nd Annual GOMACTech Conference*, Reno, NV, March 2017.

[7] R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha, "Compass: A scalable simulator for an architecture for cognitive computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 54.

[8] M. Plagge, C. D. Carothers, and E. Gonsiorowski, "Nemo: A massively parallel discrete-event simulation model for neuromorphic architectures," in *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*. ACM, 2016, pp. 233–244.

[9] O. Bichler, D. Roclin, C. Gamrat, and D. Querlioz, "Design exploration methodology for memristor-based spiking neuromorphic architectures with the xnet event-driven simulator," in *Nanoscale Architectures (NANOARCH), 2013 IEEE/ACM International Symposium on*. IEEE, 2013, pp. 7–12.

[10] L. Xia, B. Li, T. Tang, P. Gu, P.-Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang, "Mnsim: Simulation platform for memristor-based neuromorphic computing system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[11] Y. Ji, Y.-H. Zhang, and W.-M. Zheng, "Modelling spiking neural network from the architecture evaluation perspective," *Journal of Computer Science and Technology*, vol. 31, no. 1, pp. 50–59, 2016.

[12] M. Khalil-Hani, V. P. Nambiar, and M. Marsono, "Co-simulation methodology for improved design and verification of hardware neural networks," in *Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE*. IEEE, 2013, pp. 2226–2231.

[13] A. Disney, J. Reynolds, C. D. Schuman, A. Klibisz, A. Young, and J. S. Plank, "Danna: A neuromorphic software ecosystem," *Biologically Inspired Cognitive Architectures*, vol. 17, pp. 49–56, 2016.

[14] M. Kolasa and R. Długosz, "An advanced software model for optimization of self-organizing neural networks oriented on implementation in hardware," in *Mixed Design of Integrated Circuits & Systems (MIXDES), 2015 22nd International Conference*. IEEE, 2015, pp. 266–271.

[15] J.-s. Seo, B. Brezzo, Y. Liu, B. D. Parker, S. K. Esser, R. K. Montoye, B. Rajendran, J. A. Tierno, L. Chang, D. S. Modha *et al.*, "A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *Custom Integrated Circuits Conference (CICC), 2011 IEEE*. IEEE, 2011, pp. 1–4.

[16] B. Linares-Barranco, E. Sanchez-Sinencio, A. Rodriguez-Vazquez, and J. L. Huertas, "A cmos analog adaptive bam with on-chip learning and weight refreshing," *IEEE Transactions on Neural networks*, vol. 4, no. 3, pp. 445–455, 1993.

[17] C. Schneider and H. Card, "Analog cmos synaptic learning circuits adapted from invertebrate biology," *IEEE transactions on circuits and systems*, vol. 38, no. 12, pp. 1430–1438, 1991.

[18] S. Sayyaparaju, G. Chakma, S. Amer, and G. S. Rose, "Circuit techniques for online learning of memristive synapses in cmos-memristor neuromorphic systems," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, ser. GLSVLSI '17. New York, NY, USA: ACM, 2017, pp. 479–482. [Online]. Available: http://doi.acm.org/10.1145/3060403.3060418

[19] S. Amer, S. Sayyaparaju, G. S. Rose, K. Beckmann, and N. C. Cady, "A practical hafnium-oxide memristor model suitable for circuit design and simulation," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*.

[20] G. Chakma, S. Sayyaparaju, R. Weiss, and G. S. Rose, "A mixed-signal approach to memristive neuromorphic system design," in *Proceedings of IEEE International Midwest Symposium on Circuits and SystemsMWS-CAS*.

[21] J. S. Plank, G. S. Rose, M. E. Dean, C. D. Schuman, and N. C. Cady, "A unified hardware/software co-design framework for neuromorphic computing devices and applications," in *IEEE International Conference on Rebooting Computing (ICRC 2017)*, Washington, DC, November 2017.

[22] C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds, "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *International Joint Conference on Neural Networks*, Vancouver, July 2016.