EPFL Summer Internship Work Summary

Andrey Kurenkov

August, 2014

Memristor Modeling

In the first week of the internship, I was tasked with creating a VerilogA model of memristors I could use for further simulations. As there is no single predominant model, I explored the range of models used in published works and implemented several in VerilogA. All these models are in the RRAM library, and they all have a name with the prefix memr_model. Below is an explanation of each model that can be found in the library:

-**memr_model_linear**

This was the first model of a memristor implemented in VerilogA, and was written by Davide Sachetto. As seen below, it models memristors with only two distinct resistance states as well as a state in which the resistance is switching from one state to the other. This image was produced with an ADE-L simulation of the memr_model_linear_test schematic, and there are similar test schematics with saved test states for all the other models as well as other circuits. Note that high slope corresponds to low resistance and vice versa. The resistance switches from high to low whenever the voltage drop over the memristor exceeds the threshold, 1 Volt in this simulation. Correspondingly, resistance switches from from low to high if the voltage goes below the negative of the same threshold. Switching begins when the voltage approaches the threshold, which is at 0.9 Volts for this simulation. This is a good initial model to capture the essential traits of memristors, but it is in no way accurate to physical devices. The only modification I made to this model was to add the possibility of randomizing the threshold and resistance states slightly.

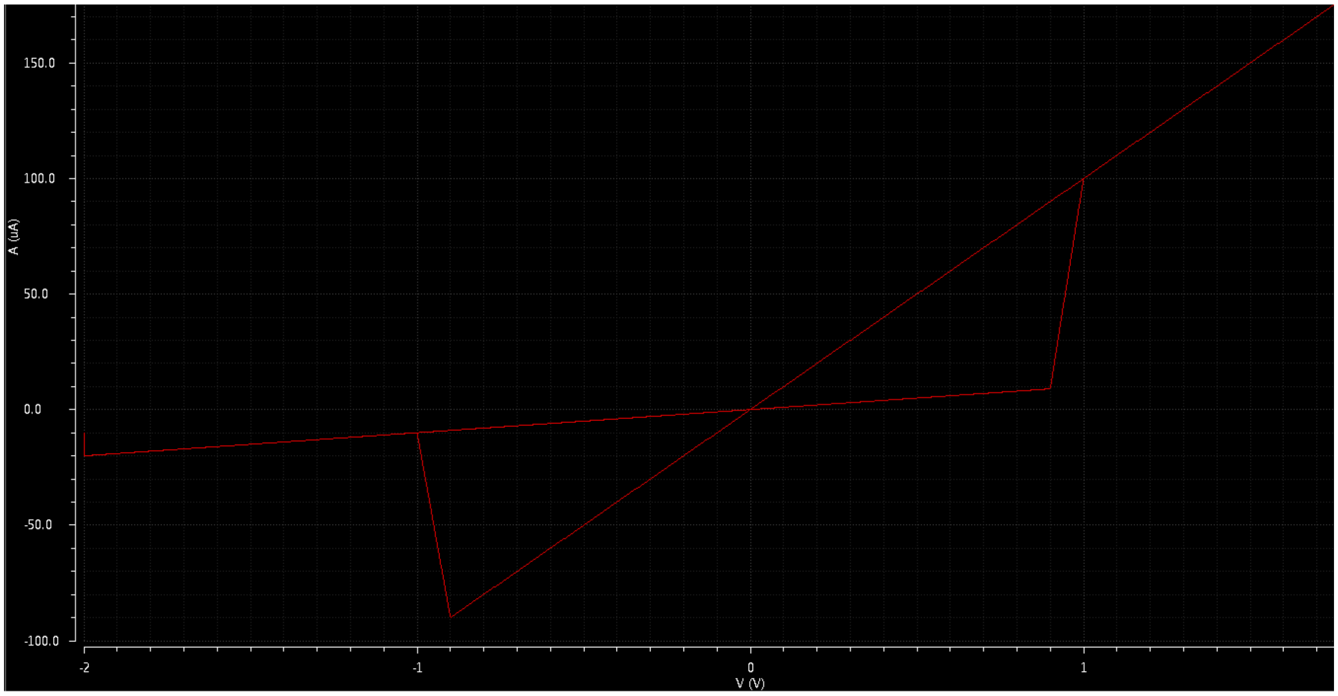


Fig 1. The hysteresis curve of this model, showing a total of 3 distinct resistance states.

-memr_linear_smooth

Although the linear model could be used in a simulation with a single memristor, I soon found that simulations with multiple memristors had convergence failures. This was due to the very abrupt transitions of this model, and so I wrote a slightly updated version of this model with smoothed out transitions that used sigmoids. Another addition in this model was that it output the state, though it should be noted this is not necessary since internal variables of VerilogA models can be saved and plotted after simulations. With this model, there are no convergence failures in simulations with multiple memristors. However, it is still a very idealized model of memristors.

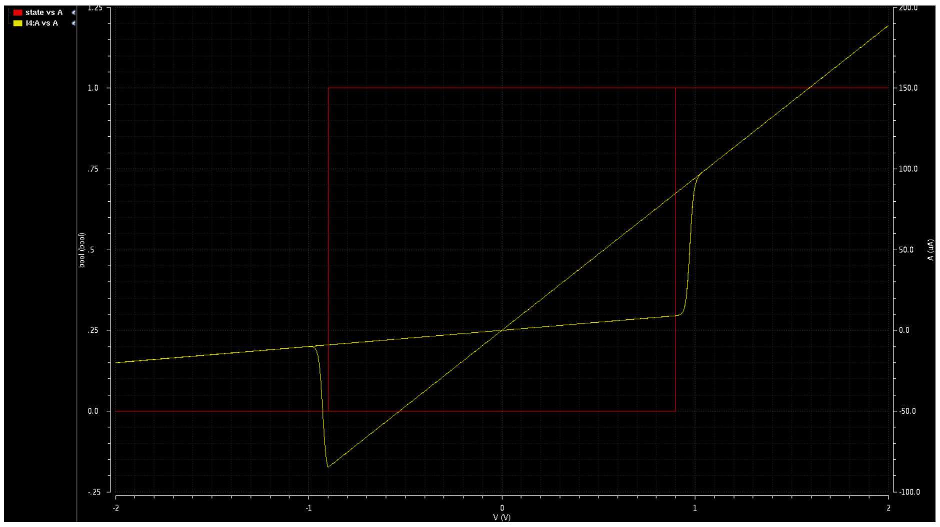


Fig. 2 The smoothed linear memristor model, with the state of the memristor also plotted.

-memr_model_smooth

After getting simulations with the linear model to work by smoothing it, I applied the same smoothing technique in an implementation that mimicked the data gathered from the lab's physical memristor devices. This is a very simple model that just fit some polynomial equations to approximate each section of the observed current versus voltage graph in the data. The real devices exhibit behavior such as frequency-dependence and variability not captured by this model, though this simple model can be tuned for simulation in a specific context.

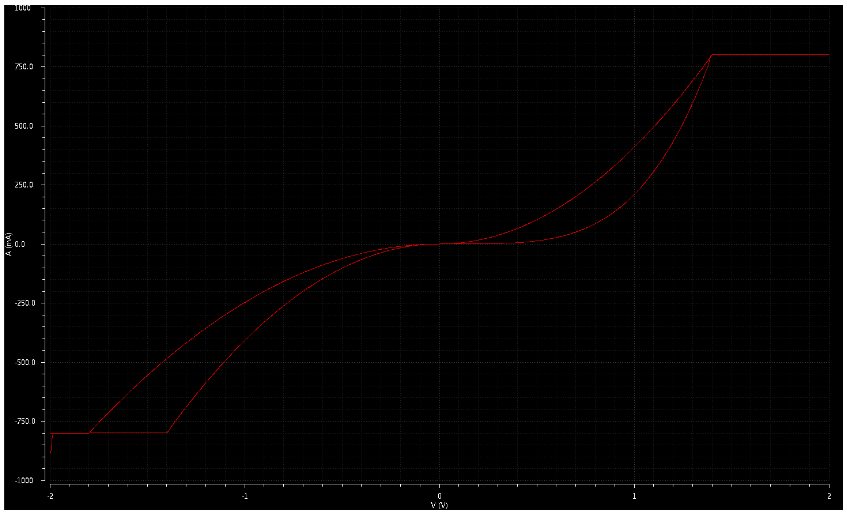


Fig. 3 The IV curve of the smoothed data-approximating model.

-memr_model_DiVentra

Massimiliano Di Ventra and Yuriy V. Pershin have published several papers dealing with memristors, and have proposed their own model of them in those papers. The model is described in “Memristive Model of Amoeba’s Learning” and “Experimental Demonstration of Associative Learning in Memristor Neural Networks.” As will be described later, my first several weeks in the lab were spent thinking about the viewpoint advocated by DiVentra that memristors could be used for logical calculation. Naturally, Di Ventra's own model was something I considered using for this purpose. However, the resulting behavior in simulation behave much like the lab's memristors, and I ultimately chose another more widely used model. However, it should be noted that it is based on real device behavior moreso than the previously described models, and exhibits realistic phenomena such as the size of the hysteresis loop being dependent on the frequency of input.

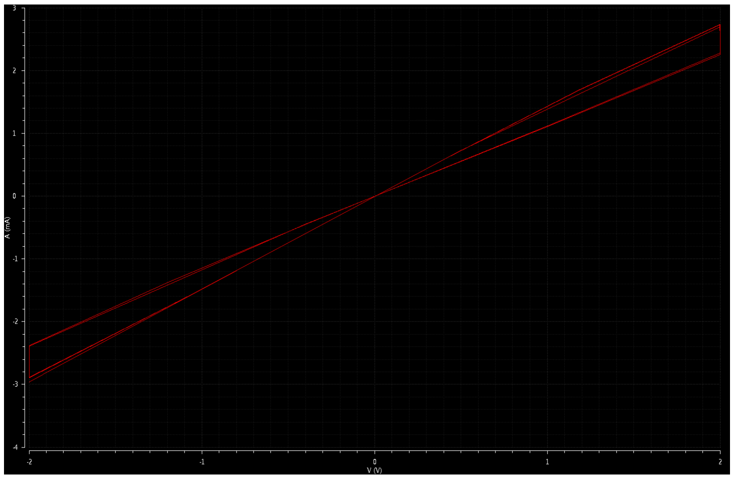


Fig. 3 The Di Ventra model simulated.

-memr_model_TEAM

This is a VerilogA implementation of 4 different memristor models published in the paper “TEAM: ThrEshodl Adapative Memristor Model". The first model is based on the simple linear ion model described in HP's seminal “The Missing Memristor Found,” two others are more complex and newer models called the nonlinear ion model and Simmons tunnel barrier model, and the final model is a new one proposed by the authors for its computational efficiency and flexibility. The linear ion and TEAM models were used for future simulations, as the former was commonly used in discussions of the neural nets and the latter could be parametrized to exhibit the more complex behavior of devices.

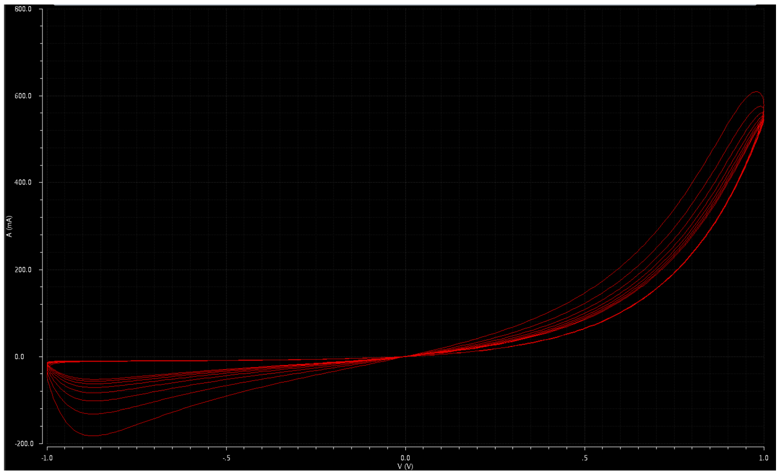


Fig. 4 The TEAM code simulated with a nonlinear ion drift model.

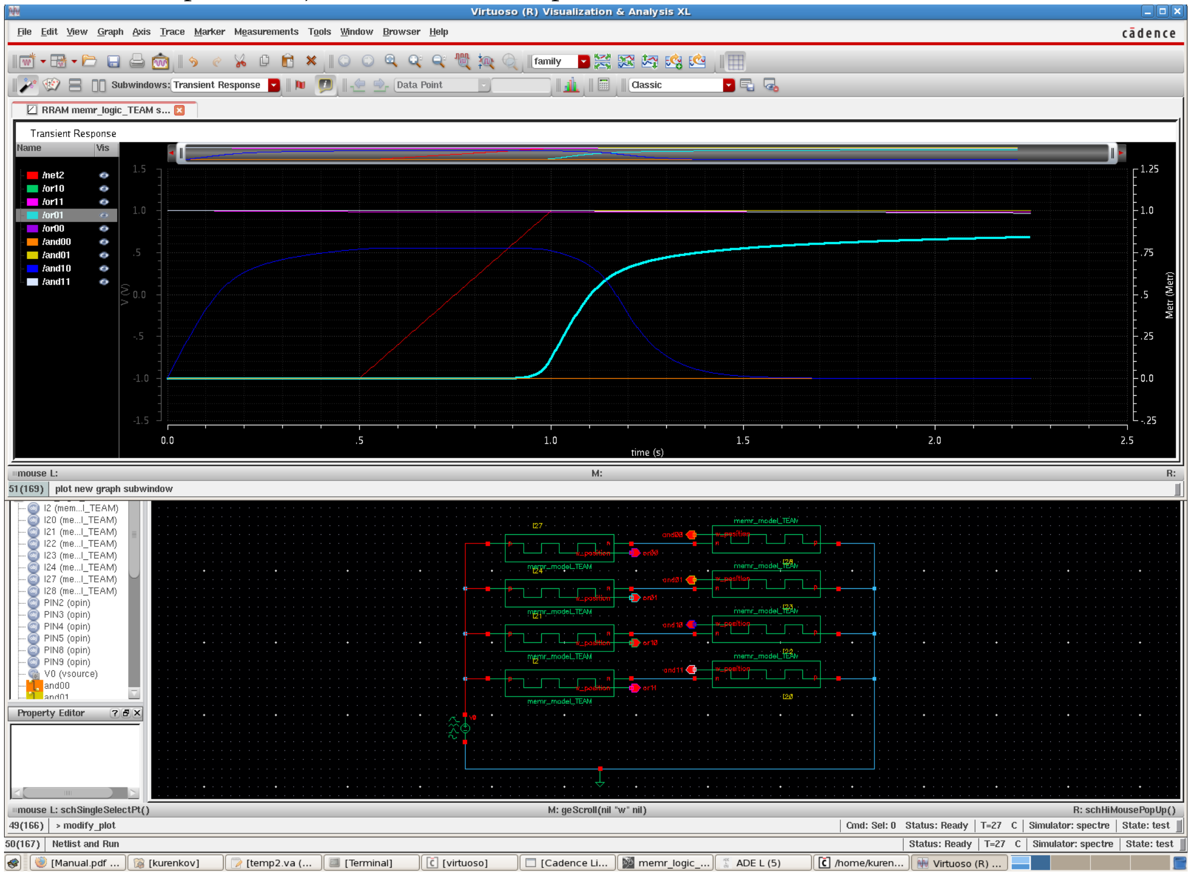
For using the TEAM code, it is important to be aware of the online manual which describes how to parametrize each model as well as the matlab and VerilogA code implementations of TEAM that are freely available online. A good explanation of several of the models in TEAM is presented in the paper "The Elusive Memristor: Properties of Basic Electrical Circuits." The latest summary about the theoretical and purely mathematical definition of the memristor can be found in "Memristor, Hodgkin-Huxley, and the Edge of Chaos," which is a better quick introduction to the concept than Leon Chuah's original 1970s writings on the subject. A more detailed paper concerning the existing physical memory components is Di Ventra's "Memory Effects in Complex Materials and Nanoscale Systems."

The **memr_model_phase** is the last implemented model, and was written according to a slight revision of the linear ion model made to better fit with physical device measurements. Though it was implemented, the expected behavior did not become apparent in simulation and it was not used.

Logic With Memristors

In the same period as I was exploring how best to model the memristor with VerilogA, I was also exploring applications of memristors to computing. My notes from that time concerning "memcomputing" can be read on Google Drive. This research direction was inspired by the fact that memristor-based logic was shown to be possible by Di Ventra et al., as well as others such as HP's Greg Snider, and because such "computation with storage elements" was recently formalized.

In addition to Di Ventra's DCRAM formulation of logic with memristors, there are others such as IMPLY, MRL , and MAGIC. While implementing VerilogA models of memristors, I also implemented several schematics for simulating these different logic types with those models in the same RRAM library. Some of these approaches to logic computation require specific memristor models, so DCRAM (which uses memcapacitance) and MRL did not produce the correct results in the simulations.


Virtuoso (R) Visualization & Analysis XL
File Edit View Graph Axis Trace Marker Measurements Tools Window Browser Help
cādence
Subwindows: Transient Response
Data Point
Classic
RRAM memr_logic_TEAM s...
Transient Response
Name
Vis
/net2
/or10
/or11
/or01
/or00
/and00
/and01
/and10
/and11
V (V)
Merr (Merr)
time (s)
mouse L:
M:
R:
51(169) plot new graph subwindow
Property Editor
memr_model_TEAM
mouse L: schSingleSelectPt()
M: geScroll(nil "w" nil)
R: schHiMousePopUp()
49(166) > modify_plot
50(167) Netlist and Run
Cmd: Sel: 0 Status: Ready T=27 C Simulator: spectre State: test
Fig. 5 The simulation results from one of the logic verification circuits

A more detailed summary of possible computation with memristors is “Logic Design with Memristors”. An additional topic I looked into was logic calculation with complementary resistive switches, and more specifically the possibility of using them for constructing a memristor with half-adders. As can be seen in my notes, the creation of a full adder with memristors was explored with the larger ambition of designing the hardware that could act as the “memprocessor” that carries out O(n) subset-sum solving as theoretically described in Universal Memcomputing Machines.

Though realizing such a “memprocessor” system would be impressive, I soon realized that this was a difficult computer engineering task akin to creating parallel computing processors that have been a subject of research for decades. The conclusions regarding this can be found in more detail in the last page of my notes, as well as in this Google Drive presentation I prepared prior to switching focus to neuromorphic computing with memristors.

Neuromorphic Computing With Memristors, VerilogA

Another application of memristors that has been a subject of many publications in the past several years is neuromorphic computing. Specifically, memristors have a strong potential to act as synapses in artificial neural nets, for reasons covered well in “Nanoscale Memristor Device as Synapse in Neuromorphic Systems.” The interest in hardware neural nets is also strengthened by an increasing understanding of the biological processes that enable learning, and particularly Spike Time Dependent Leaning (STDP). The relation between memristor and STDP is well explained in “STDP and STDP Variations with Memristors for Spiking Neuromorphic Learning Systems”.

Though several labs have demonstrated learning with simulated memristor-based neural nets, some of the most impressive results were done by Damine Querlioz and several others in “Simulation of a Memristor-Based Spiking Neural Network Immune to Device Variations”, which demonstrated learning digit recognition (OCR), as well as “Extraction of Temporally Correlated Features From Dynamic Vision Sensors with Spike-Timing-Dependent Plasticity.” The first paper is notable for achieving quite good character recognition performance with only a single layer neural net leaning in a completely unsupervised fashion, which proved the potential of memristor-based neural nets. The second paper explored STDP learning with multiple layers of neurons for relatively difficult computer vision problems, and once again presented impressive performance. Therefore, replicating the results of these papers in simulation with simulation in Cadence was set as the next goal for me to work on.

The initial task for me to be able to do this was choosing a suitable neuron design and implementing it in Cadence. To do this, I used the **NCSU45** device library I was already familiar with from prior experience. I started with the simple leaky integrate and fire neuron introduced by Mead in 1989, by implementing the schematic and symbol **IAFMead** cell in the NeuronDesigns library.

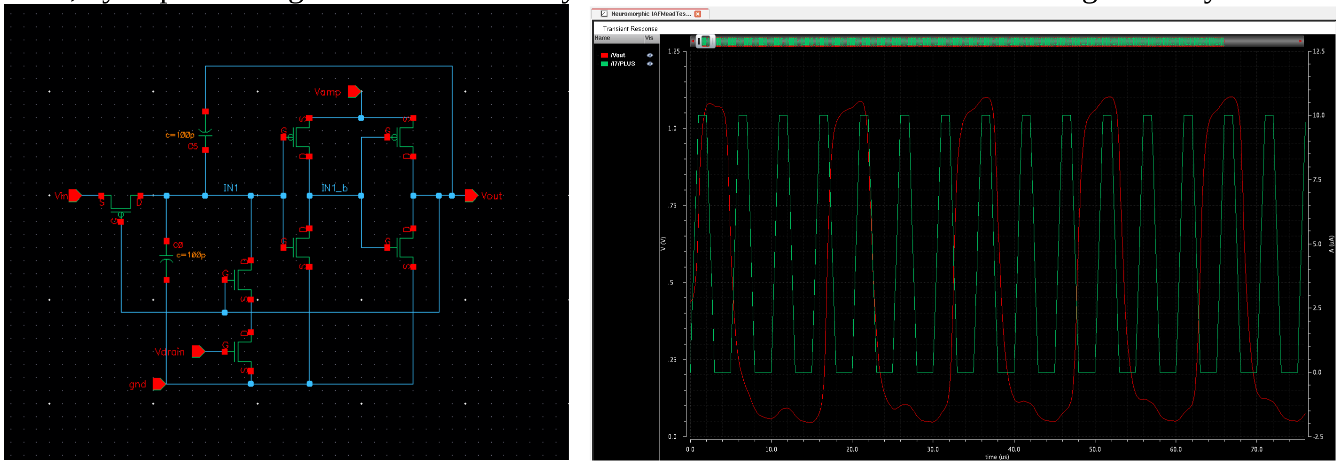


Fig. 6 The Mead circuit and working simulation results

Because the simulation results were at first incorrect, I implemented several other neuron designs such as in **IAFIndiveri** (based on "Neuromorphic Silicon Neuron Circuits") and **IAFWen** (based on "Associative Learning of Integrate-and-Fire Neurons with Memristor-Based Synapses") which had better defined values for the circuit parameters. However, those circuits also did not work in simulation and after more debugging I found that the specific technology I was using was limited to a high voltage of 1.2V rather than the standard 5V. With this detail corrected, the Mead circuit worked as expected.

However, the Mead circuit by itself would not be able to implement STDP learning as described in the 2011 Querlioz OCR paper, which was my goal. Therefore, to meet the goal of quickly trying to reproduce the results of that paper I decided to first implement the paper's neuron model with VerilogA code. I started implementing the basic IAF neuron in **IAFVerilog**, and modified it to exhibit the behavior of the OCR paper with **IAFSTDPVerilog**. By far the largest challenge with getting this software neurion implementation to work is the requirement for only one neuron to ever be spiking at one time. As with memristors before, having a constant change in the value of the inhibition signal that each neuron outputs led to convergence problems. Ultimately I implemented two different solutions for inhibition, which is why there is also **IAFSTDPVerilog2**.

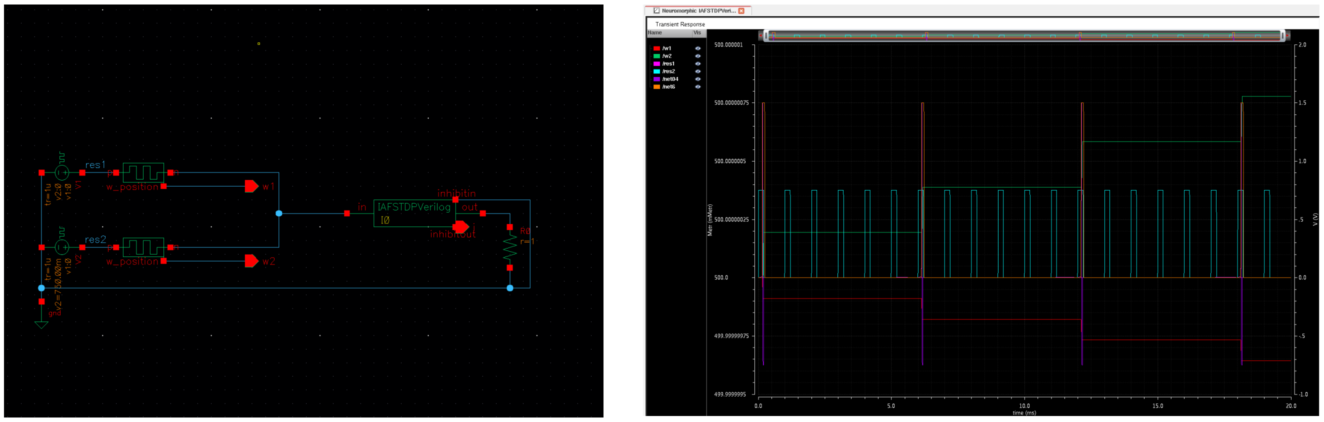


Fig. 7. Demonstration of basic STDP with two synapses and one neuron.

The first test circuit I constructed was **IAFSTDPVerilogTest**, seen above. This allowed me to work out some initial issues and demonstrate that the basic learning scheme described in the paper was implemented in my code. After this, I had the issue of having to test larger neural nets with multiple inputs that changed over time. In order to programatically generate input to the neural larger nets, I

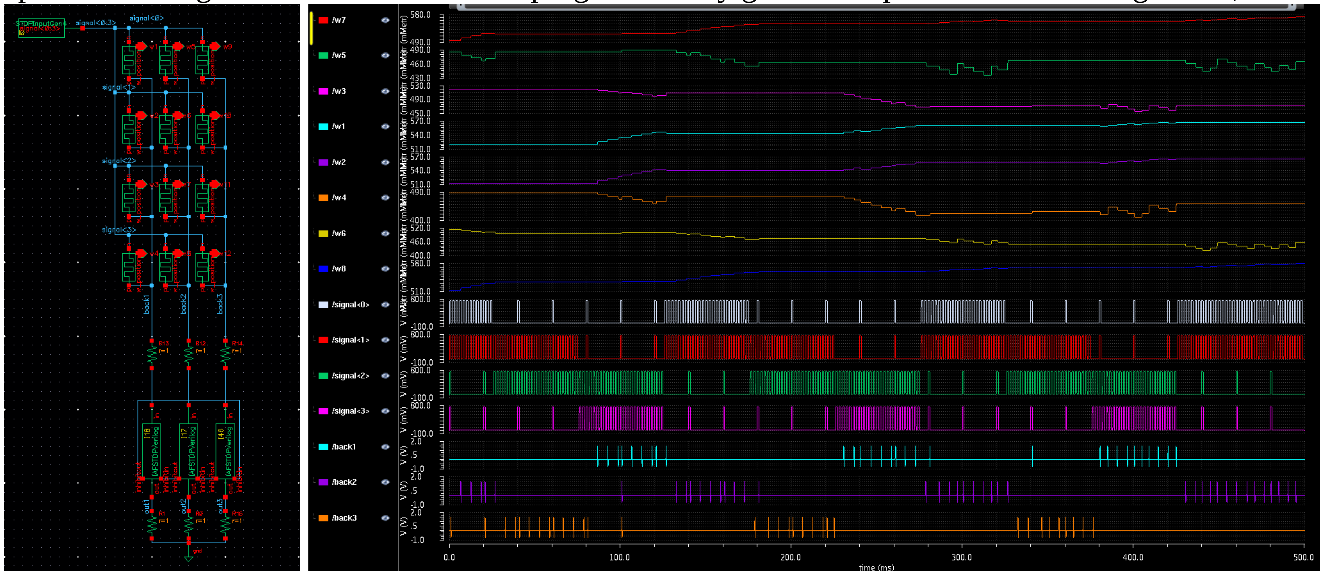


Fig 8. The schematic for a 4 input, 3 output, neural net and corresponding successful learning.

wrote the **InputGen** cell in the Neuromorphic library. An additional version called **InputGen4** was the same except that it produced 4 inputs signals instead of 10. With this, I created the larger neural net test schematic, **ArraySmall(1-3),** and demonstrated successful learning of several patterns by the neural net. A final small neural net that was tested was one with 4 inputs and 4 outputs. The neural net initially did not succeed in learning the 4 patterns, so the homeostasis property from the 2011 paper was added and with it the 4 by 4 neural net successfully learned as well.

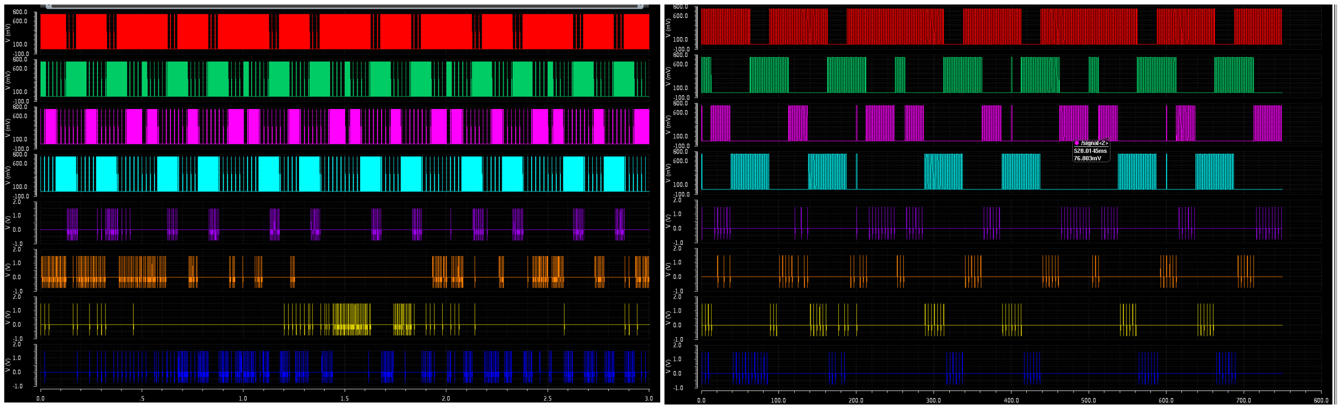


Fig. 9 Results for a hard 4 by 4 learning task, before and after homeostasis.

With successful learning done on these smaller circuits, the task afterward was to attempt to do larger-scale learning on a real dataset. Though the paper we were working off of used the MNIST handwritten digit dataset, the Semeion digit dataset was chosen to start working with initially chosen instead due to its smaller size and simpler format. In order to parse this dataset and use it as input in simulations, the I**nputGetData** cell was created and made to successfully do that. An additional challenge was the fact that the grid of memristors needed for this larger neural net could clearly not be layed out entirely by hand. Therefore, the **MemristorGrid256by10** cell, also in the Neuromorphic library, was created as a compact implementation of the grid of synapses needed for this larger net. **ArrayBig** is the result of combining those cells with ten output neurons.

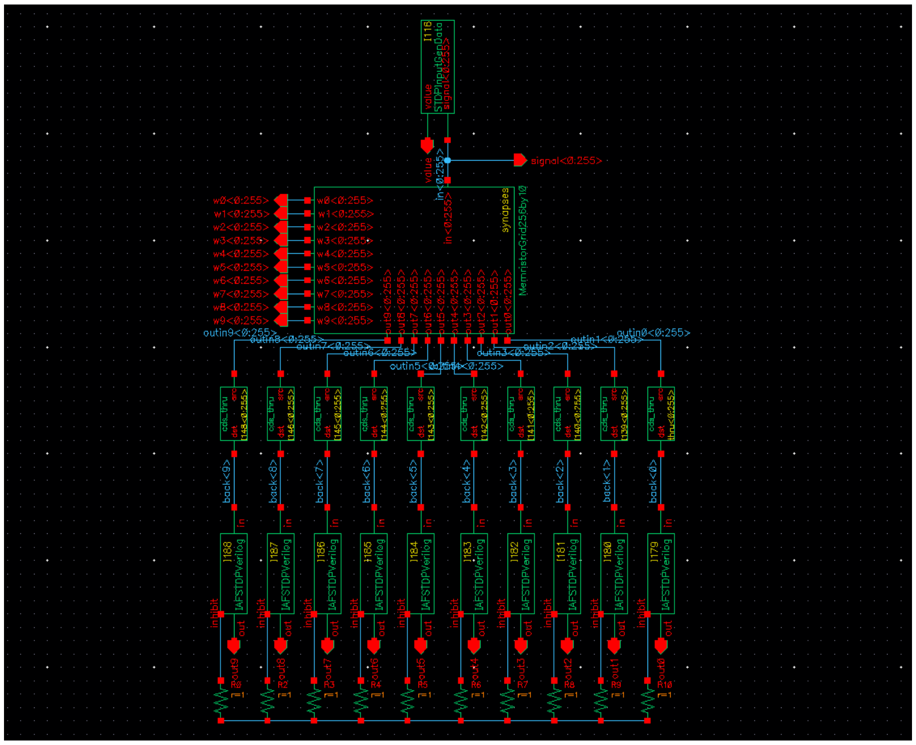


Fig. 10 The larger array composed in a schematic for learning with a dataset

However, because this larger neural net has such a large number of individual components and so much input data, the simulations took much too long to run for the entire dataset. Therefore, since VerilogA is in essence entirely software already, it was decided that the same simulation should be implemented in Matlab to enable much faster simulation and verification of the logic used for it.

Neuromorphic Computing With Memristors, Matlab

The reimplementation of the VerilogA code into Matlab did not take long, and as expected resulted in much faster simulation. At first, the result seen in VerilogA simulation for very small neural nets were replicated with Matlab to confirm the same learning behavior was observed.

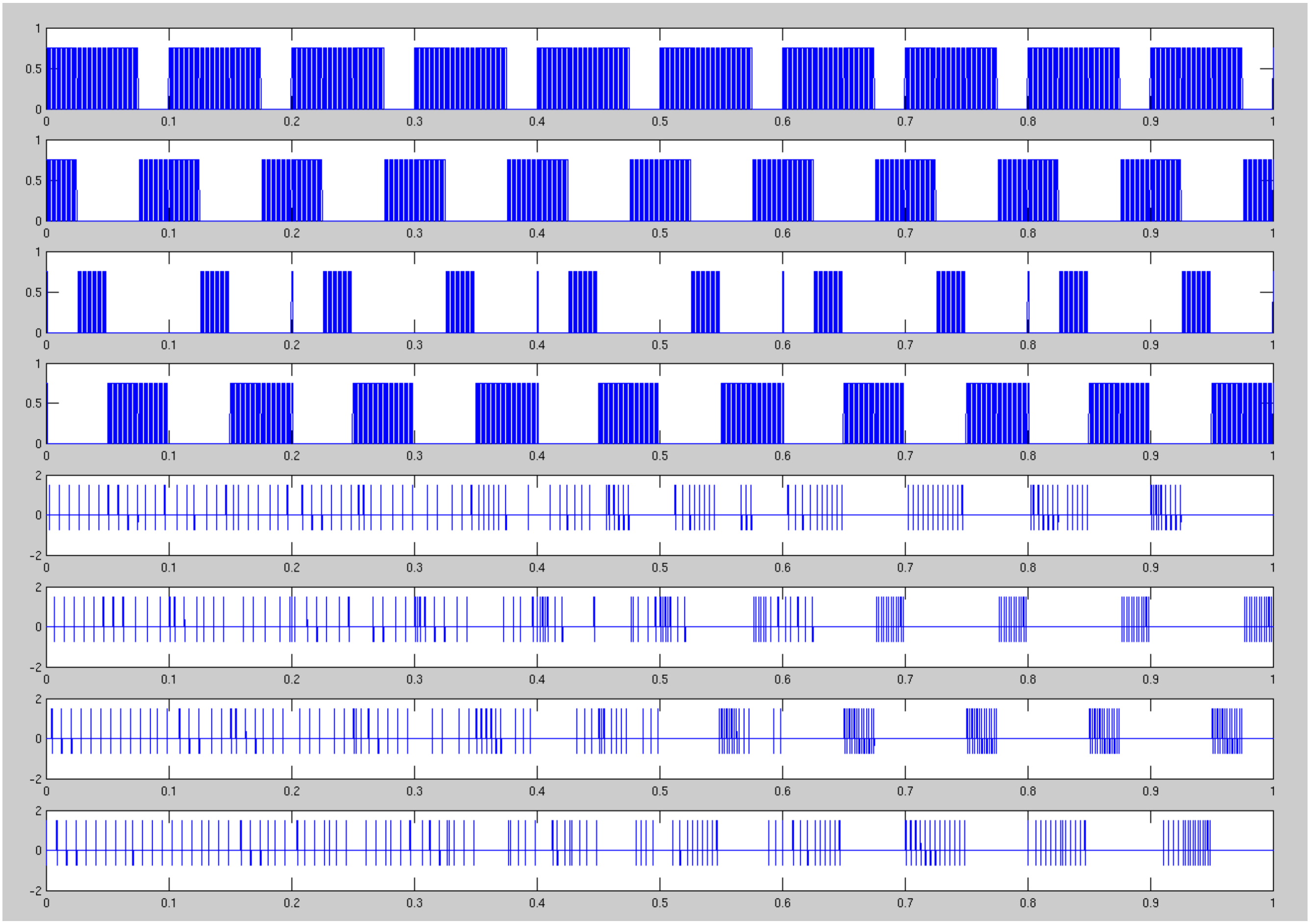


Fig. 11 As before, 4 output neurons learn to distinguish between 4 input patterns.

The Matlab code likewise simulated the larger dataset neural net much faster, though it would still take minutes to run a simulation over the entire dataset. Running such simulations and plotting the resulting synapse conductance clearly showed that the neural net was in fact learning to recognize the digits presented in the input. However, the accuracy seen from these simulations for the training set was at about 50%, far below expectations.

Soon after my developing the initial Matlab code for running these simulations, an additional intern (Nicolas Merlin, nicolas.merlinge@ensam.eu) started work in the lab and took over revising and improving the simulation. Further reports on findings from simulations with Matlab can be found in the "Matlab Results" folder, whereas this report will detail further Cadence cells I developed during my internship.

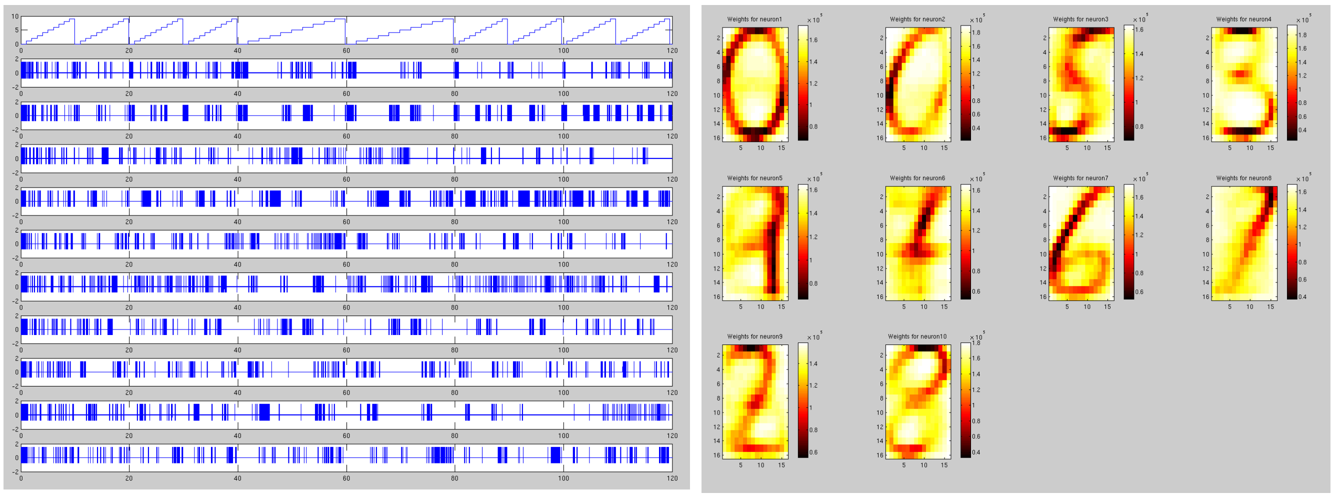


Fig 12. Spike graph from a full simulation of simulation with the dataset input and 10 output neurons, with the final conductances of the synapses of each neuron.

Neuromorphic Computing With Memristors, Simulated Circuits

The focus for the rest of my work was finding appropriate circuits with which to replicate the STDP learning exhibited in the already accomplished simulations. The paper “Neuromorphic Silicon Neuron Circuits” by Indiveri et al. summarizes many existing neuron designs for neuromorphic circuits. The basis for my neuron design was from the paper “Neural Learning Circuits Utilizing Nano-Crystalline Silicon Transistors and Memristors.” This was chosen due to its relative simplicity and the modifications it had made to the original Mead circuit, such as a tuneable leakage rate and spiking duration. However, since the motivation of this work was still to replicate the results of the OCR paper, modifications to the base design were necessary for it to operate in the correct way.

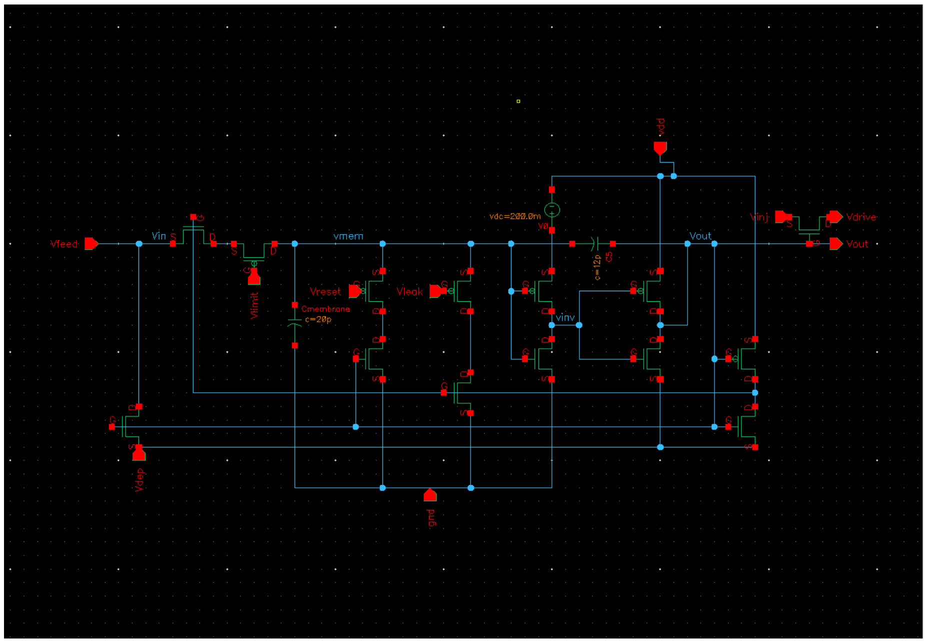


Fig. 13 The original neuron circuit replicated from the Cantley paper.

IAFCantley has the initial neuron design from the paper replicated with the NCSU 45 tech library. It is in the Neuron Designs library, though the CMOS cells (**pmos45**, **nmos45**, etc) used to make it are in the Neuromorphic library, as better library organization was only done later and it is not possible to move cells. This cell was used to verify that the neuron circuit works in the **CantleyArray**

cells, but it was soon clear modifications were necessary for accomplishing the desired functionality.

The design iterations are in the numbered **IAFCantleyModified** cells. The first such cell only added the STDP waveforms, described in the 2011 OCR, for learning. A significant difference from the software model was that the negative voltage could only be of low magnitudes such as -100m or -200m, and so the positive voltage was within this amount of the memristor threshold. Additionally, the voltages from a spiking neuron to the pre and post synapses have the same duration for this iteration of the neuron, unlike the theoretical model in which the pre voltage waveform is significantly shorter.

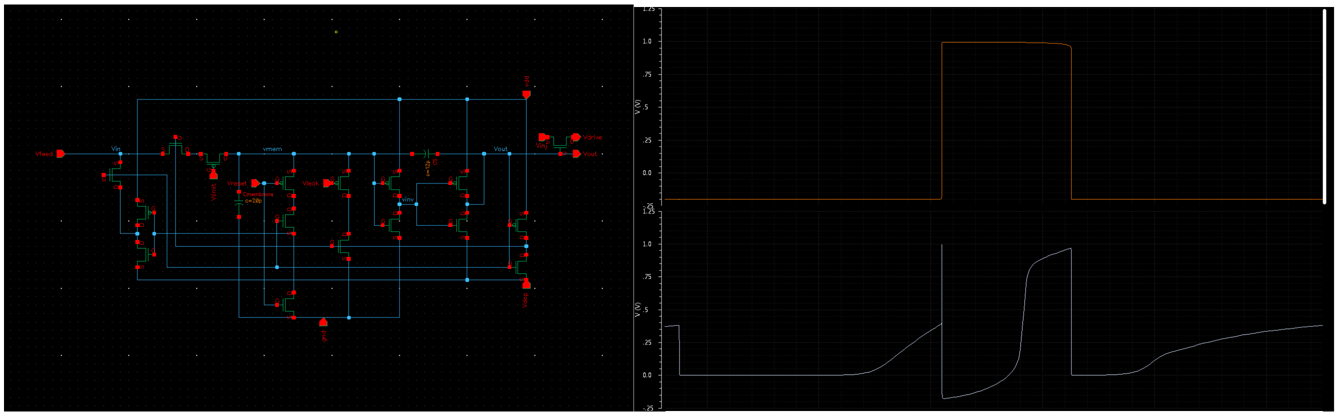


Fig. 14 The modified neuron and the corresponding spiking voltage at the output and input.

However, with a constant on/off input this circuit did exhibit the basic learning that was similar to the results previously seen with software. Further iterations of the design introduced several other important components. **IAFCantleyModified2** added a simple mechanism for homeostasis, which is the change of neurons' spiking thresholds with the purpose of having roughly equal spiking frequencies. The design is a simplification of a much more complicated homeostasis circuit described in "Implementing homeostatic plasticity in VLSI networks of spiking neurons." Many circuits that were experimented with but not used for this are in the NeuromorphicCircuits library.

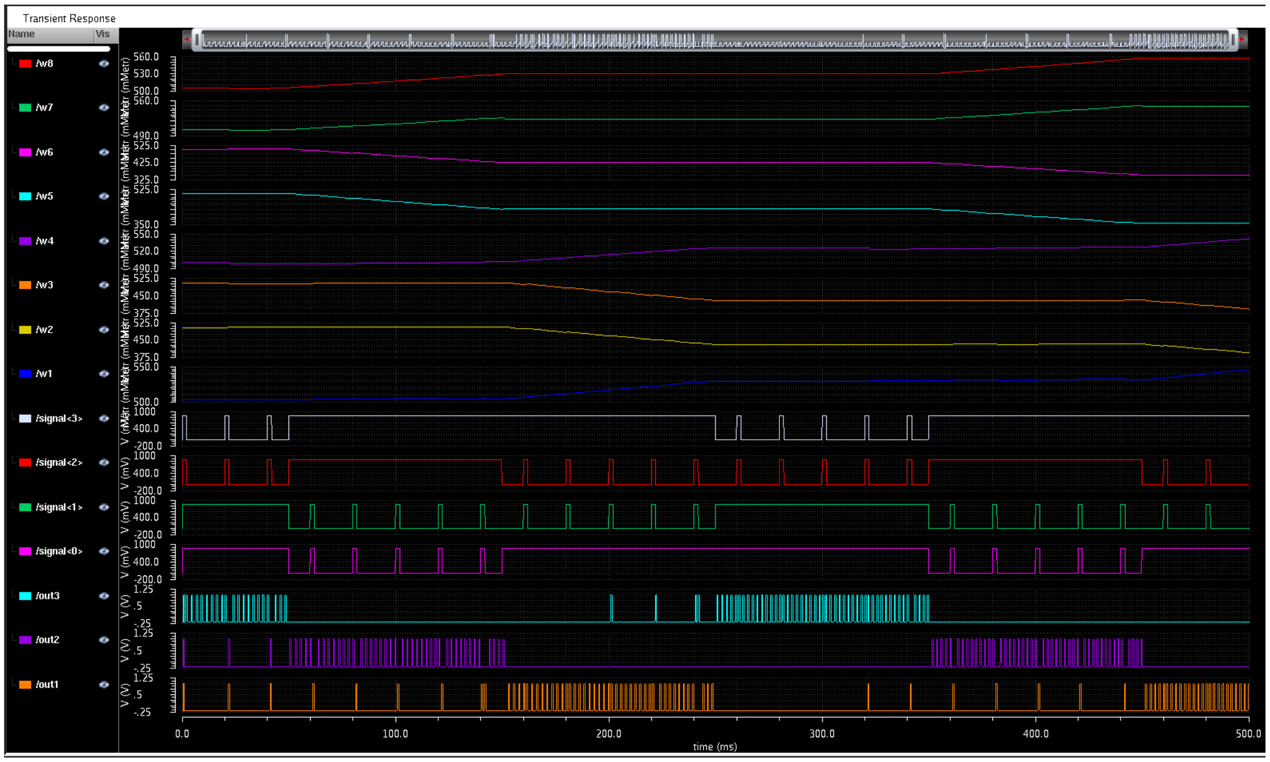


Fig. 15 A demonstration of simple STPD learning with the CMOS neuron circuits.

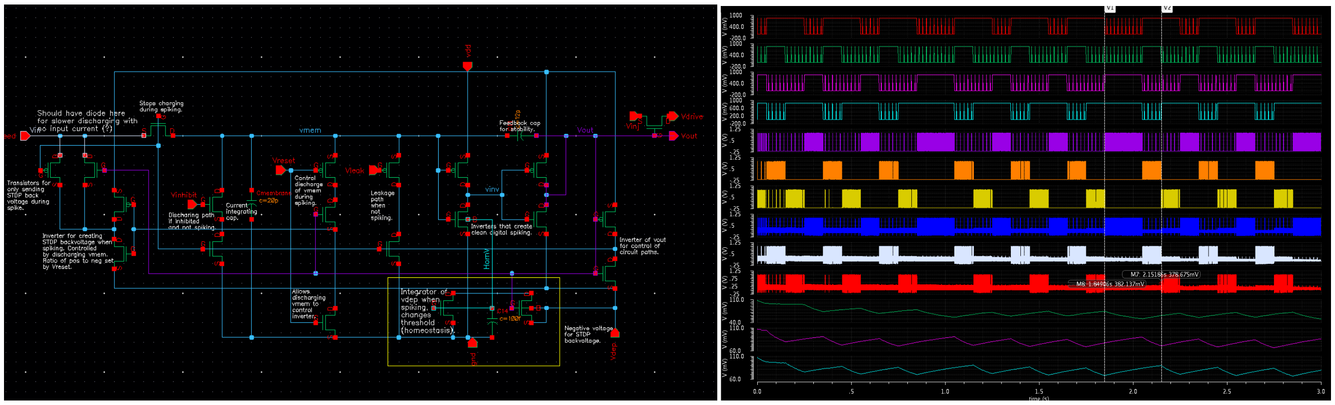


Fig. 16 The modifications made to the circuit for homeostasis, and its operation on the right.

This was followed by the last major revision to the neuron circuit, with **IAFCantleyModified3**. Version **a** of this cell introduced the major change of this revision, which is that some additional circuitry is introduced to make the spiking voltage waveform at the input of the neuron be much shorter in duration than the output voltage waveform. This ensures the when a neuron spikes, the pre and post synapse spiking voltages overlap for learning. The modification to the circuit accomplishes that by lowering the votlage at the input as soon as it reaches its peak, so that most of the spike is spent with the back voltage too low to be greater than the threshold of the memristor.

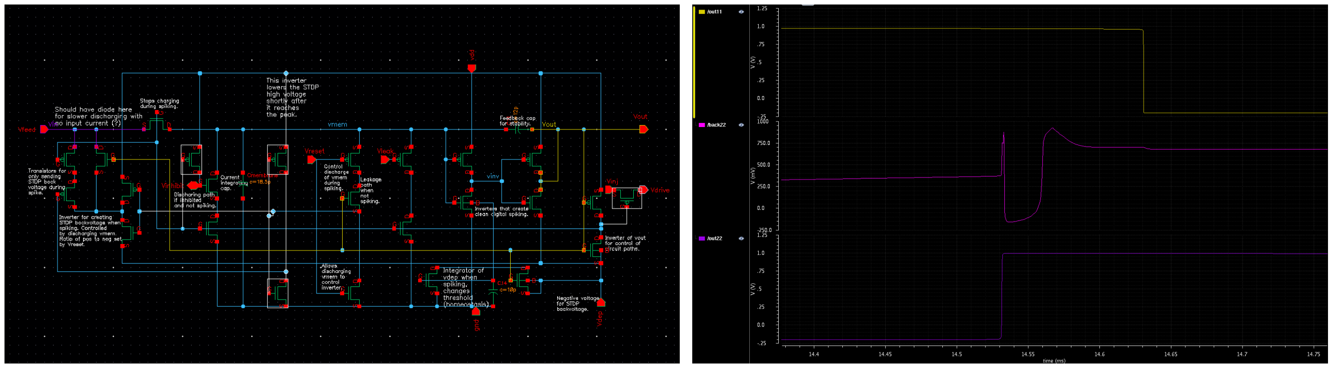
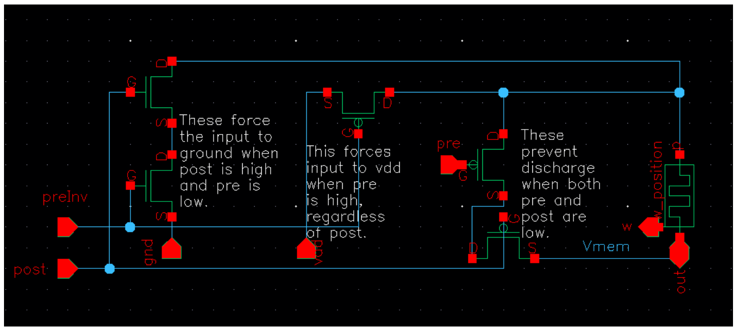


Fig 17. The final iteration of the neuron circuit, and corresponding voltages upon spiking. The yellow signal is an input, and the pink and purple signals are the backward and forward voltage upon spiking.

Following these many design iterations, there was still a significant problem with the neural net circuits because when the input before a synapse was low all the charge on the membrane capacitor would discharge through the input, rather than being blocked and only leaking through the leakage path. This was a crucial problem, since later layers from multilayer neural networks would not be charged to their spiking threshold by neuron spikes from the previous layer. In order to get a working multi layer neural net, multiple iterations of **CantleyModifiedMultiArray** were created, with version **c** working the best. The **STDPSynapse** cell in NeuromorphicCircuits implements the synapse that is used in that cell.


These force the input to ground when post is high and pre is low.
This forces input to vdd when pre is high, regardless of post.
These prevent discharge when both pre and post are low.
preInv
post
pre
gnd
vdd
Vmem
w
w_position
out
Fig. 18 The final synapse design.

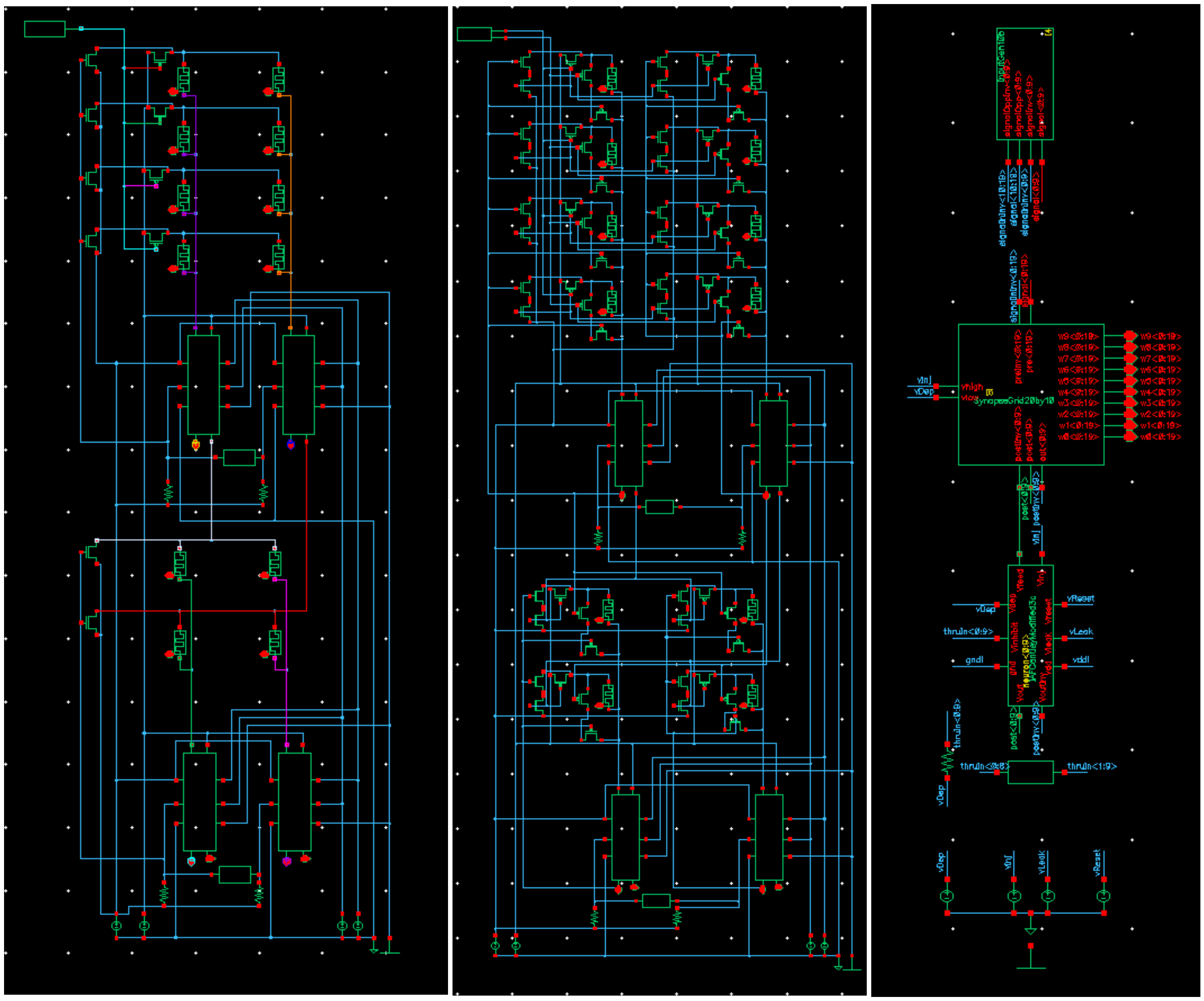


Fig. 19 A first attempt at a multilayer neural net on the left, the final design in the middle, and the schematic for a much larger neural net on the right.

As part of developing these neural nets, **InputGen4b** and **InputGen10b** were developed to output both the input signal and its inverse. Additionally, the latter also has two more signals for the opposite frequency of the set patterns, so that neurons can learn to spike for patterns with mostly low input signals as well as with mostly high input signals. This, and the cell **SynapseGrid20by10**, was used to create the **CantleyModifiedArrayBig** cell that is an easily scalable approach to create schematics for large neural nets. The same cell was modified for the **CantleyModifiedMultiArrayBig** cell, which is the size needed to attempt learning on real data or more complicated patterns.

As before, these larger neural nets take much longer to run than the small neural nets used for development, and so only short simulations can be run. Because of this limitation, the behavior of the larger neural nets could not be fully inspected or tuned. A short simulation did confirm that the schematic is sound, although several undesired behaviors were seen: all neurons spiked at the same time despite there being lateral inhibition, and later on only a single neuron spiked and inhibited the rest. Large patterns have been learned by neural nets in our software simulations, so it must be assumed that the same can be done by either varying parameters or further iterating on the circuit designs.

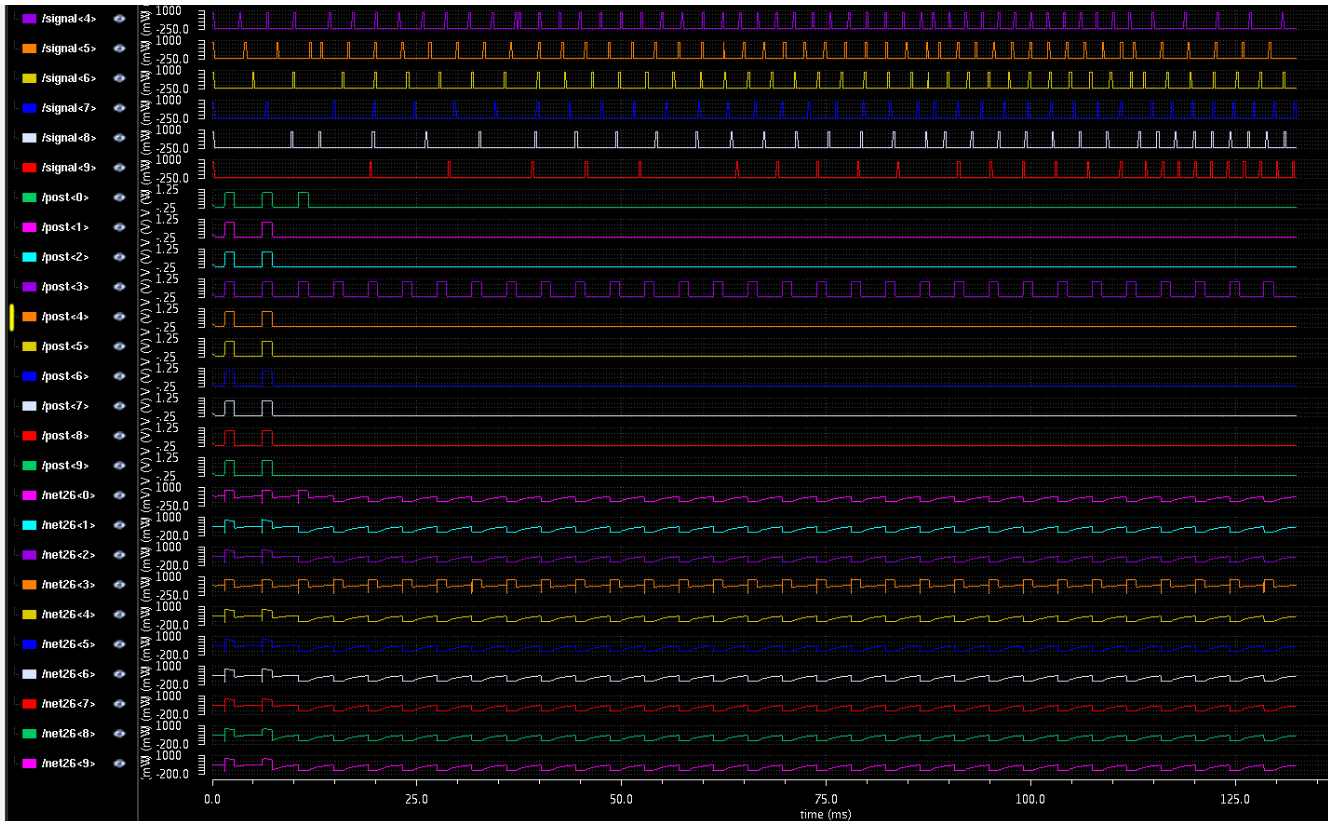


Fig 20. The preliminary simulation result obtained from the CantleyModifiedArrayBig. The failure of inhibition for a larger circuit is a significant problem.

Final Notes

This summary primarily focused on explaining the developed Cadence libraries and cells, as well as providing the sources on which this work is based. The primary aim with regards to hardware design was to implement the same simple learning approach as in the software simulations we already had working, and though the implemented neurons do not behave identically they do qualitatively work the same way. It is important to note that other approaches to STDP, such as in "Spike timing-dependent synaptic plasticity using memristors and nano-crystalline silicon TFT memories", and other approaches to neuron design, such as in "Neuromorphic Silicon Neuron Circuits" and "Associative Learning of Integrate-and-Fire Neurons with Memristor-Based Synapses", have been shown to work and if further studied could reveal a better approach to learning with multilayer neural nets in hardware. Additionally, it is likely further circuitry is needed to implement the sort of multilayer learning described in "Extraction of Temporally Correlated Features From Dynamic Vision Sensors with Spike-Timing-Dependent Plasticity." This paper described successful STDP learning with multiple layers in software simulations, and no hardware implementations of something like this have yet been developed. Though the essential components needed to implement such neural nets have been created in this work, there may be revisions or further iteration required to apply them for learning analogous to what is presented in that paper. Further findings from simulations in MATLAB could be used to explore how this simple learning scheme can be applied to a task such as character recognition, which could possibly clarify any further additions needed in hardware.