Directed Research Summer 2021 Final Report

A Neuromorphic Framework with Hardware-software Codesign for Vision Tasks

Haoqin Deng

(haoqinde@usc.edu)

Supervised by

Dr. Alice Parker

(parker@usc.edu)

Acknowledgements

All the work presented in this report is done under the supervision of Professor Alice Parker, in the University of Southern California. I am extremely grateful that Dr. Parker shared her expertise in the field of neuromorphic computing without reservation.

The work as presented here is still incomplete. While organizing the materials for this report, I realized that there is some knowledge gap between previous DR students and later ones, that is, I find it difficult to directly build upon the work done by previous DR students due to the lack of detailed textual description of their implementations. In addition, I realized that the general lack of educational materials in the field of neuromorphic computing makes it even more difficult for newcomers to grasp established work. Hence, I decided to write this report in a thorough and even pedagogical manner, such that future DR students or interested researchers can directly build upon the unfinished work. Although I currently do not possess the knowledge (requiring taking EE577A) to continue the project, I nevertheless hope that my unfinished work can become useful in someone else' hands.

All materials, including Python code, circuit files, and figures, will be uploaded to the google drive of Dr. Parker's group and my github repository: https://github.com/haoqindeng/DR-_SUMMER2021.

Abstract

In this report, we represent a summer's work on developing a neuromorphic-computing framework for solving image-classification problems, with hardware-software codesign. The hardware design involves developing analog circuits that emulate the functions of various components of human brains, including ***excitatory synapse, inhibitory synapse, axon hillock, edge detector, dopamine reward***, all of which are building blocks of a large neural network. The software design involves ***simulating*** a spiking neural network (SNN) with realistic circuit parameters and eventually ***synthesizing*** SNN circuits.

Note 1: the last part of "synthesizing SNN circuits" is not yet done as the author currently does not possess the knowledge of using the Python-Spice interface, which is taught in EE577A.

Note 2: We assume that the readers already possess some basic knowledge about CMOS transistors , neural science, and neural networks. For those unfamiliar with these topics, we refer them to first watch some crash course tutorials, which are abundant online.

Note 3: The figures presented below may be too small to see clearly. Full figures can be found in the Appendix 1.2.

1. Introductions

1.1 Neuromorphic Computing

Modern-day electronics, which mostly adopts Von-Neumann architecture, are inherently bottlenecked by data transfer between memory and computing units. Neuromorphic computing provides a more efficient computing scheme by combining memory with computing, therefore avoiding the bottleneck. In addition to having higher computing speed, neuromorphic circuits also consume less energy because most of their building transistors operate in subthreshold regime and information is encoded as sparse spikes.

Neuromorphic computing is essentially an analog computing scheme, which builds circuits that emulate the functions of various components of human brains such as neurons and synapses. This is somewhat analogous to the concept of Application Specific Integrated Circuits (ASIC) in digital computers, where we trade general purpose for special optimization of a particular function. However, due to its architectural advantages, neucomputing has great potential for efficiently implementing machine-learning related on-chip modules.

1.2 Circuit Components

We design and implement circuit components that each emulates the functions of human-brain components, including ***excitatory synapse***, ***inhibitory synapse***, ***axon hillock***, ***edge detector***,

Dopamine-Noise-STDP synapse. These are the most basic components of human brains, and there are many more such as astrocytes and dendrites, but for now they are sufficient for the purpose of constructing a simplified neural network model. We can always develop more circuit components if we decide to include more brain features in our model. We will discuss each of the components mentioned above later in this report.

1.3 Spiking Neural Network

We organize those circuit components into a system that is the spiking neural network (SNN). SNN is in many aspects similar to the artificial neural network (ANN) that we are familiar with, as they share a similar network topology and many computing mechanisms such as matrix-vector multiplication, thresholding, convolution etc. Their major difference is that SNN has a unique temporal dimension as it encodes information, or spikes, in the time domain. There have been various lines of research in the field of SNN: conversion from ANN to SNN [1,2]; directly training SNN using backpropagation with surrogate gradients [3,4]; training SNN with STDP rule and reward modulation [5]. More comprehensive reviews can be found here [6,7]. We will describe the architecture of the SNN that we use in detail later in this report.

2. CMOS Implementation of Circuit Components

In this section, we present the schematics of each circuit component, describe in detail their working principles, and analyze wave functions we obtain from Cadence simulation.

2.1 Excitatory Synapse

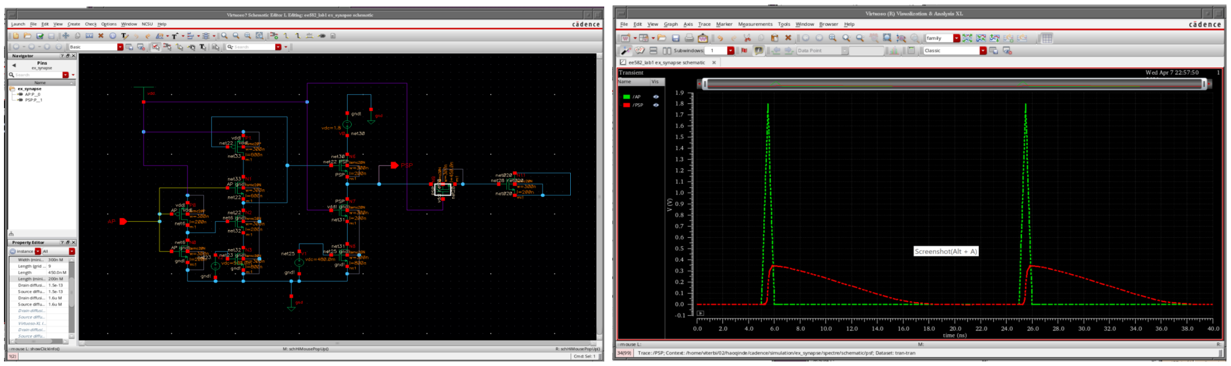


Figure 1. (a) Schematic of Excitatory Synapse (b) Waveform of Cadence simulation

The function of an excitatory synapse is to take an input action potential (AP), which is a spike, and output a more attenuated but prolonged Excitatory Postsynaptic Potential (EPSP or PSP). The process lowers the signal strength and extends the overlapping window for multiple PSPs for fault tolerance in analogue computing.

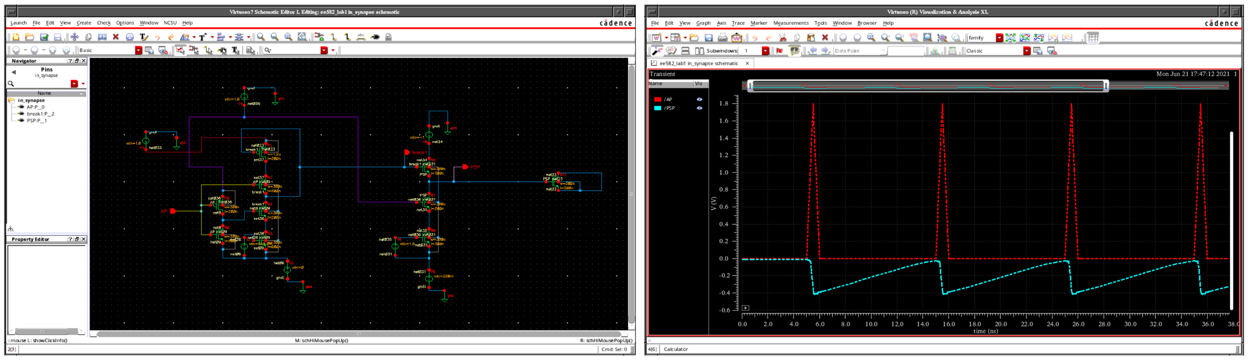
Figure 1.a shows the schematic of the excitatory synapse. The working principle is as follows:

When the input action potential (AP) rises: Vnet6 decreases because it's essentially AP inverted, and closes N2; meanwhile, AP opens N1; As a result, Vnet22 gets pulled up by VDD to a limited extent controlled by P1; Vnet22 opens N6, causing PSP to rise.

When the input action potential (AP) decreases: the mirror of the process described above takes place. Eventually, PSP gradually gets pulled to ground.

Figure 1.b shows the waveform of Cadence simulation. We clearly see a sharply shaped AP (green signal), which is a spike, resulting in a more attenuated but prolonged EPSP (red signal).

2.2 Inhibitory Synapse



Virtuoso® Schematic Editor L Editing: ee582_lab1 in_synapse schematic
Virtuoso (R) Visualization & Analysis XL
cadence
Transient
/AP
/PSP
V (V)
time (ns)

Figure 2. (a) Schematic of Inhibitory Synapse (b) Waveform of Cadence simulation

The function of the inhibitory synapse is similar to that of the excitatory synapse. It takes in an AP but outputs a negative PSP. This negative PSP has the effect of lowering the sum of multiple PSPs, hence the name "inhibitory synapse".

Figure 2.a shows the schematic of the excitatory synapse. The working principle is almost identical to that of the excitatory synapse, except that after Vbreak1 opens N6, PSP gets pulled up to vdc=-1, a negative voltage. We also introduce a biasing voltage vdc=220mV between net031 and gnd! to offset the voltage drop across N7 due to the leakage current, even when N7 is closed.

Figure 2.b shows the waveform of Cadence simulation. We clearly see a sharply shaped AP (red signal), which is a spike, resulting in a more attenuated, prolonged and negative PSP (cyan signal).

2.3 Edge Detector with Voltage Adder

In image processing, edge detection is simply convolving a filter with the image. Here, we use the traditional sliding-kernel as opposed to the FFT approach because the size of kernels is small [8] . Using the sliding-kernel approach, we simply need to employ a voltage adder to sum up weighted, or filtered, PSPs in the particular sliding window.

It’s tempting to think that by simply converging all PSPs into a node can we sum them up. However, physics law tells us that only currents get summed up this way. In fact, it is a bit more troublesome to implement a voltage adder. Previous DR students used *current mirror voltage adder*, which requires some effort for parameter tuning. Here, I use the most straightforward *non-inverting Op-amp voltage adder*. Note that the resisters involved can simply be replaced by pass transistors.

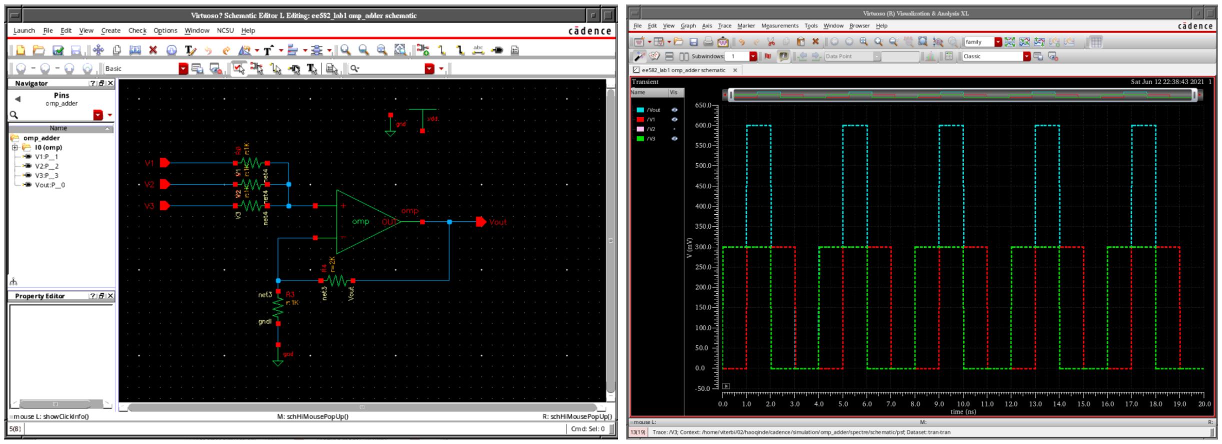

Virtuoso® Schematic Editor L Editing: ee582_lab1 omp_adder schematic
Launch File Edit View Create Check Options Window NCSU Help
cādence
Navigator
Pins
omp_adder
V1:P__1
V2:P__2
V3:P__3
Vout:P__0
Property Editor
V1
V2
V3
omp
Vout
net3
net4
gnd
Virtuoso (R) Visualization & Analysis XL
Transient
/Vout
/V1
/V2
/V3
V (mV)
time (ns)

Figure 3. (a) Schematic of omp adder (b) Waveform of Cadence simulation

Figure 3.a shows the schematic of the voltage adder. It is a standard non-inverting voltage adder. It takes in three PSPs and outputs their summation. In principle, we can sum up an arbitrary number of PSPs simply by adding more input ports.

Figure 2.b shows the waveform of Cadence simulation. We clearly see that the output voltage (cyan signal) is the sum of two input PSPs (red & green signal). Note that the exact voltage may deviate from the ideal equation and may vary with a different number of inputs. It takes a bit of parameter tuning over the values of resistors.

2.4 Axon Hillock Circuits

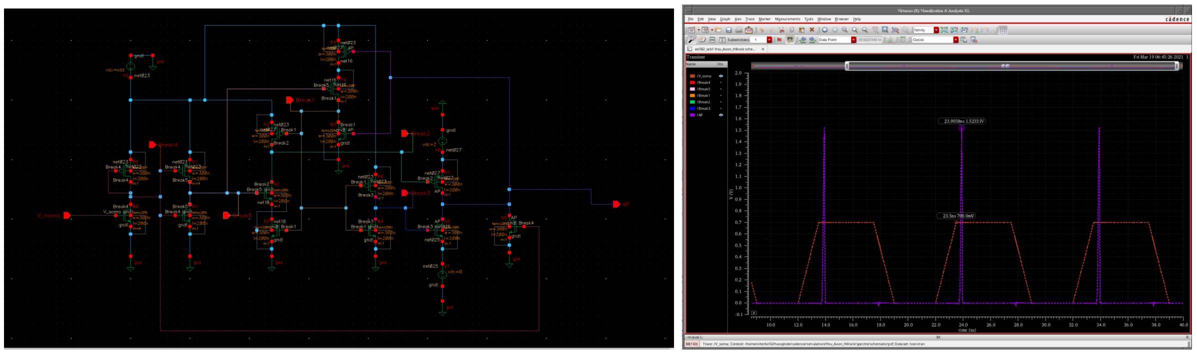


Figure 4. (a) Schematic of Axon Hillock (b) Waveform of Cadence simulation

The Axon Hillock circuit reverses the input and output relationship of the excitatory synapse circuit. It takes in PSP and outputs a spike if the amplitude of PSP exceeds a certain threshold. This circuit provides the nonlinearity that is crucial to generalization ability of neural networks. **Figure 4.a** shows the circuit schematic. The working principle is described in *Appendix 1*.

Figure 4.b shows the waveform of Cadence simulation. We clearly see that the input PSP (orange signal) with 500mV amplitude exceeds the threshold of the NMOS transistor and triggers the Axon Hillock circuit to make an output spike.

2.5 Cascading Circuits

Now that we have excitatory synapse circuit that takes input AP and outputs PSP, and Axon Hillock circuit that takes input PSP and outputs AP, we need to match their waveforms such that the output of one circuit is in the proper operating input range of the other circuit, and vice versa. Only this way can we construct a cascade of circuits, or multi-layer neural networks, in which signals propagate with little attenuation or distortion.

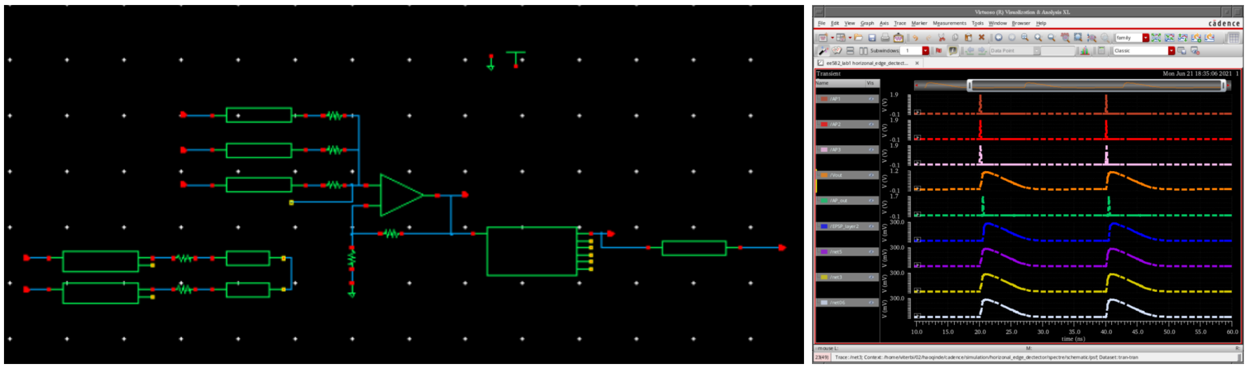


Figure 5. (a) schematic of cascading circuits (b) Waveforms of cascading circuits

Figure 5.a shows the schematic of cascading circuits. The signal propagates as follows: three artificial input spikes (AP1, AP2, AP3) goes through three excitatory synapses and produces EPSP1, EPSP2, EPSP3, which gets summed up by the voltage adder and goes into the axon hillock circuit, producing AP_OUT, which is then fed into excitatory synapse circuit to produce EPSP_layer2.

Figure 5.b shows the waveform after cascade synapse circuits and Axon Hillock circuits. We see that the output EPSP (blue signal) of the second-layer axon hillock circuit produces the same EPSP as the first layer synapses (first three signals) do. It means that our *Input_Spike->Synapse->EPSPs->Neuron->Realistic_spike->Synapse->realistic_EPSP* signal chain now forms a closed loop with little attenuation or distortion while propagating through multi-layers. With this verified, we can proceed to construct multi-layer neural networks.

Note that we have modified some parameters of the original excitatory synapse and axon hillock circuits presented above to match their input and output operation range.

2.6 STDP circuit

The excitatory and inhibitory synapse described above has only fixed weights. We still need a variable synapse that can adapt its weight based on the STDP learning rule. We first design the circuit component that emulates STDP features. Tutorials online have often assumed a simplified version of the STDP rule. Here, we adopt the version used in Kun Yue's thesis [9], which captures richer neural dynamics, as shown in **Figure 6.a**.

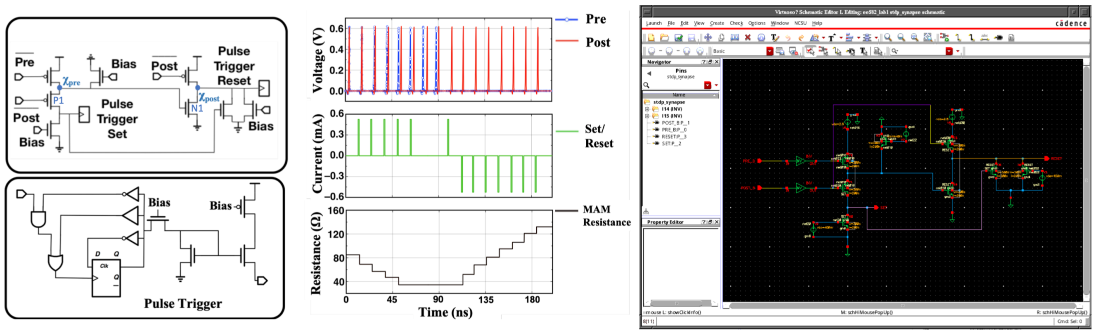


Figure 6. (a) Schematic of STDP circuit in Kun's thesis (b) Schematic of STDP circuit we implement

When a presynaptic potential arrives, it pre charges Xpre. Then, several scenarios may occur:

If postsynaptic potential arrives immediately afterwards, it blocks the Xpre brom discharging through P1, which maintains *set* signal, and allows Xpre to keep N1 open, which discharges Xpost and avoids triggering the *reset* signal. This is <u>scenario 1</u> in Kun's thesis.

If postsynaptic potential arrives afterwards after a long time, Xpre already leaks away, so there won't be any *set* signal. The low Xpre cannot keep N1 open, so *the* reset signal cannot be pulled down and gets pulled up, or triggered, by the post signal from above. This is <u>scenario 3</u> in Kun's thesis.

If postsynaptic potential arrives after a somewhat long time, between the range in scenario1 and scenario 3, we can naturally interpolate between the two extreme scenarios, that is, the signal strength of the *set* and *reset* signal linearly vary with the time difference. This is <u>scenario 2</u> in Kun's thesis.

Note that Kun further discretizes the *set* and *reset* signal by feeding them into a pulse trigger. This is because he needs the pulses to control the resistance of memristors, which is the way to store memory in his setup. Since we only store memories in voltage nodes (shown later) as opposed to resistance of memristors, we do not need the pulse trigger and can directly utilize the raw *set* and *reset* signal.

Furthermore, this STDP circuit is not a synapse circuit by itself; rather, it only generates *set* and *reset* signals, which can be considered to be opposing signals that determine the direction and extent of changing synaptic weights. We need to append the circuit to an actual synapse circuit.

Knowing the working principle of STDP circuit, we implement it on our own. **Figure 7.b** shows the schematic of our STDP circuit.

Figure 7 shows a series of waveforms corresponding to varying time differences between presynaptic signal and postsynaptic signal, ranging from {1, 2, 3, 4, 5, 15}ns. We clearly see that as time difference increases, the *Set* signal gets weaker (blue signal) and that *Reset* signal gets stronger (purple signal). Note that the raw *set* and *reset* signal may need to be amplified later to exert notable effects on a voltage node.

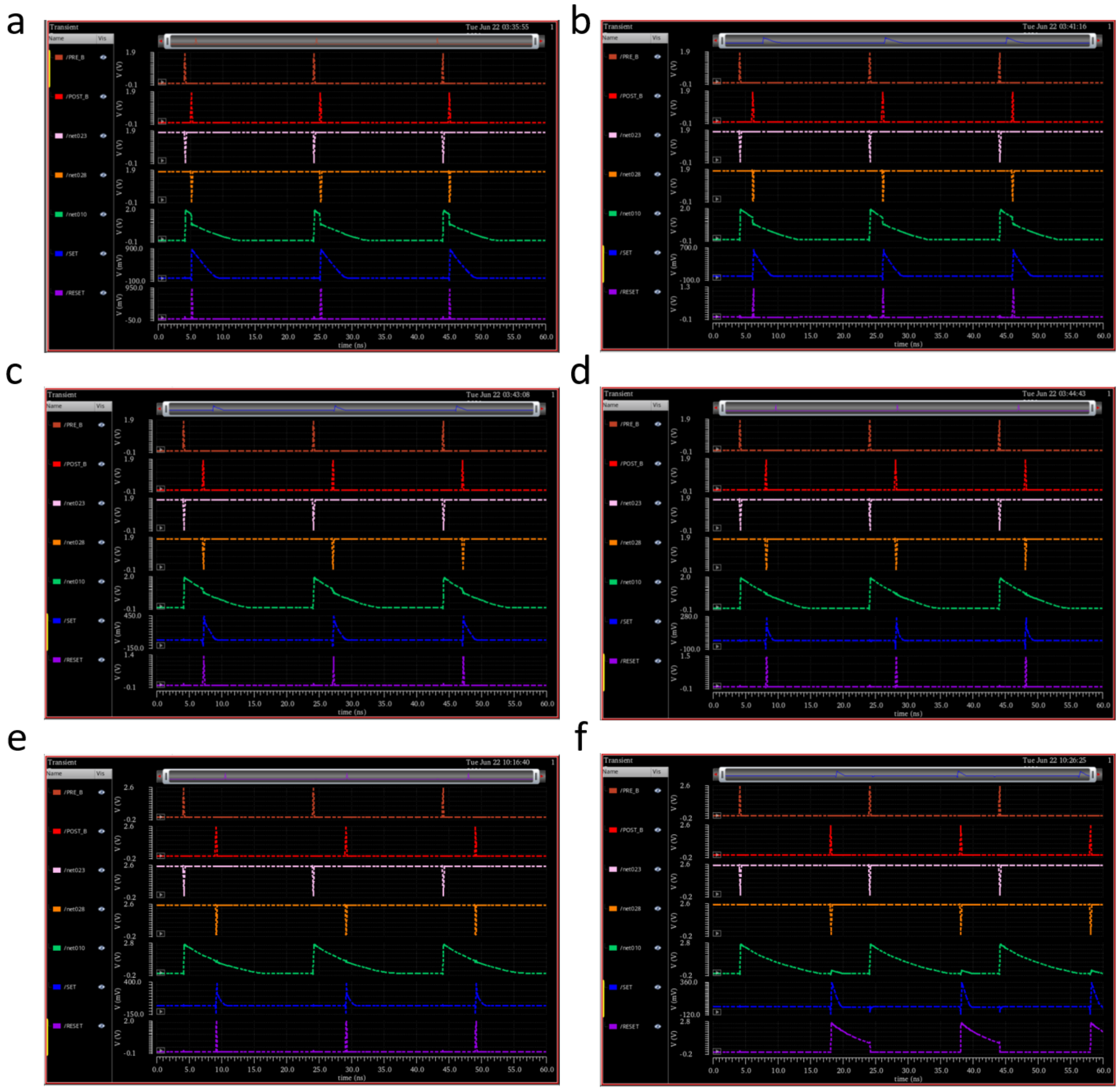


Figure 7. Waveforms of STDP circuit simulations

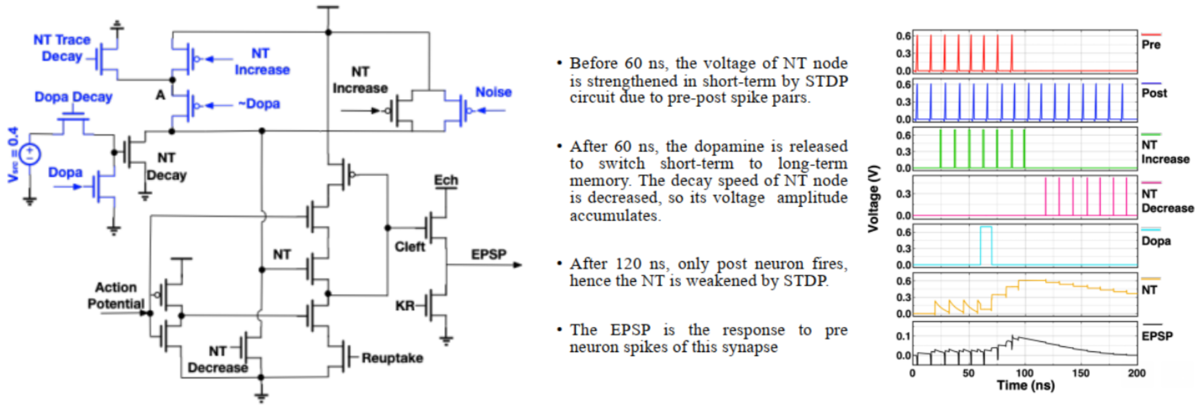
2.7 STDP-Dopamine-Noise (SDN) Synapse Circuit

Now that we have the STDP circuit to give *set* and *reset* signals, we need to incorporate it into a synapse circuit. The STDP-Dopamine-Noise Synapse Circuit has multiple functions: it receives STDP signals (*set* and *reset*) and changes the synaptic weight accordingly; it receives dopamine signals and responds accordingly; it receives noise signals, which is important to kickstart neural activities.

Figure 8.a shows the schematic of this multi-purpose circuit, which is adapted from Kun's thesis. The working principle is fairly straightforward: we simply add to the original excitatory circuit (black colored) with extra control circuits (blue colored) that take in external STDP (NT Increase / Decrease), Dopamine, and Noise signals. The synaptic strength is stored at the NT voltage node.

However, notice that there are some differences between the original excitatory synapse circuit and the black colored circuit shown here, with the main difference being that in the black colored circuit there is an NT node that controls the transistor conductance, or the synaptic weight. **Figure 9.a** shows the schematic of our implementation of the black-colored part of the circuit, which we call NT synapse. **Figure (b)-(d)** shows the waveforms of simulation of NT synapse with various NT voltages {1.8V, 0.5V, 0V}. We clearly see that when NT=1.8V the NT synapse functions just like the original excitatory signal, and that as NT decreases, the amplitude of EPSP (purple signal) also decreases. Hence, by varying NT node voltage with external signals (STDP, dopamine, noise) can we directly change the synaptic weight.

Note: the implementation of this SDN synapse circuit is not fully done, as it is a bit troublesome to implement the Noise circuit, which generates Noise signals. The simulation in Kun's paper achieves this functionality through random initialization in software. The Dopamine circuit should be designed in tandem with the network architecture and will be described later in the report. It is a challenging yet worthwhile task to implement a full SDN circuit that integrates the STDP circuit, Noise circuit, Dopamine circuit into the excitatory synapse, and that operates in a compatible voltage range with the rest of the circuit. We leave the work to future DR students.



- Before 60 ns, the voltage of NT node is strengthened in short-term by STDP circuit due to pre-post spike pairs.
- After 60 ns, the dopamine is released to switch short-term to long-term memory. The decay speed of NT node is decreased, so its voltage amplitude accumulates.
- After 120 ns, only post neuron fires, hence the NT is weakened by STDP.
- The EPSP is the response to pre neuron spikes of this synapse

Figure 8. (a) Schematic of multi-purpose synapse circuit in Kun's thesis. (b) Waveforms of corresponding simulation in Kun's thesis.

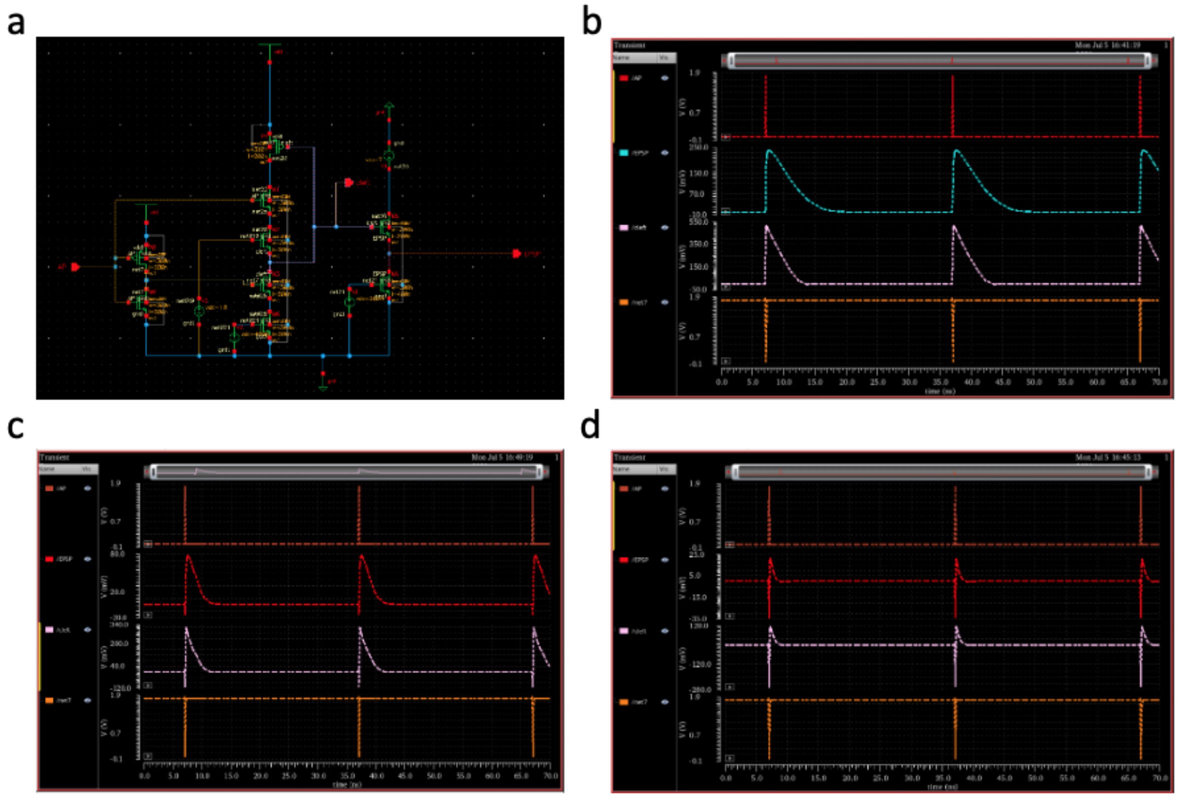


Figure 9. (a) Schematic of NT synapse. (b)-(d) Waveforms of Cadence simulations of NT synapse circuit with various NT voltages.

3. Python Simulation of SNN

Now that we have all the circuit components, we can draw the circuit schematic of a full neural network out of these building blocks. However, we immediately see that the task is quite challenging, if not unrealistic, because of two reasons: first, it is extremely laborious and error-prone to connect all connections by hand, especially when the network size is large; second, it is very slow (sometimes days) for a software such as Cadence to simulate a large neural network, much slower than running a neural network written in code, which makes it impossible to actively test and revise the neural network back and forth. Hence, we must adopt the approach of using software for designing and testing neural networks then synthesizing the hardware circuit based on the verified software model. Hence, this approach is named "*software-hardware codesign*".

3.1 SNN Architecture

Figure 10 shows a toy SNN architecture. Blue blocks represent a layer of neurons while green blocks synapses. The overall workflow is as follows:

In the current setup, we have input images of size 3*3. The first layer of synapses are four 2*2 edge detectors in ↑, ↖, ←, ↙ directions, resulting in 4*2*2 outputs for each input image. This layer of synapses exactly corresponds to the convolutional layer in traditional machine learning. The 4*2*2 neurons are flattened and enter the STDP synapse layer, where synapses are initialized with random weights and update themselves according to the STDP rule. This layer functions like a self-organizing map and gives rise to distinctive output patterns for each class of input. The outputs at this pattern layer are then fed into the DNS layer, which uses Dopamine feedback signal to force those patterns into the one-hot fashion at the output layer, where each neuron represents a particular class. The function of dopamine signals is to inhibit the natural synapse decays for synapses that contribute to elevating the potentials of the correct output neurons. Note that the edge detection synapse layer has all synaptic weights fixed, the STDP layer self-evolves without supervision, and the DNS layer is somewhat similar to the reinforcement learning.

Table 1 shows the pseudocode of the algorithm. Note that the SNN is trained layerwise for computational efficiency. More details of implementation can be found in the code file, which may be slightly different from the pseudo code in terms of coding manner.

Note that all the components involved in this SNN architecture have their CMOS counterparts. Hence, it is a natural next step to directly map the software model onto a SPICE model that can be run by Cadence.

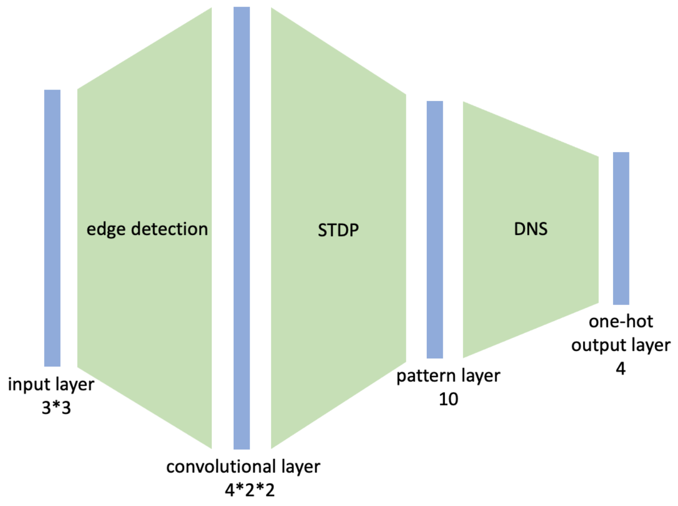


Figure 10. Diagram of a mini SNN architecture

3.2 Python Implementation

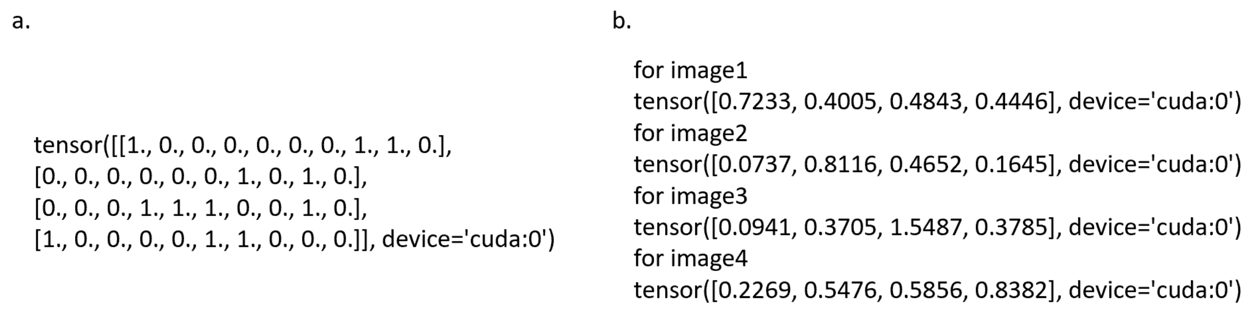
We implemented the SNN using PyTorch library to allow for convenient GPU acceleration and thus better scalability. The neural network is constructed layerwise, and all values are stored in torch.tensors for potential GPU deployment. The code contains Neurons, Synapses, Network classes as building blocks, and the mini SNN architecture described above is implemented in Main.py.

```
Algorithm:
Train():
  input_neurons = GetImage() # four 3*3 images
  image_num = input_neurons.shape[0]
  # compute conv layer
  kernels = GetKernels() # four 4*4 kernel
  for idx = 0 to image_num:
    conv_neurons[idx] = conv2d(input_neurons, kernels) # 4*2*2 for each images
  Flatten(conv_neurons)
  # train stdp layer
  stdp_synapses = Random()
  for iter = 0 to 12:
    idx = iter % image_num
    pattern_neurons[idx] = mul(conv_neurons, stdp_synapses)
    stdp_sypases.STDP()
  # train dopa layer
  dopa_synapses = Random()
  for iter = 0 to 10:
    for idx = 0 to image_num:
      dopa_synapses.Dopa(idx)
      dopa_synapses.Decay()
Test(idx):
  input_neurons = GetImage()[idx] one 3*3 image
  output_neurons = mul(mul(conv2d(input_neurons, kernels), stdp_synapses), dopa_synapses)
```
Table 1. Algorithm for training the mini-SNN

3.3 Code Simulation Results

Since everything before the STDP layer has fixed parameters, it is not interesting to analyze this part other than verifying its correctness. For the STDP layer outputs, **Figure 11.a** clearly shows that for each of the four input images, there is a unique output firing pattern. **Figure 11.b** shows the effect of forcing those patterns into one-hot output neurons using dopamine signals. We clearly see that for each image, the neuron with the highest output potentials correctly corresponds to the class of the image.

Note that we only developed a mini-SNN that takes in 4 classes of 3*3 images, just for the sake of demonstrating the correctness of its working principles. The prospect of extending the mini-SNN to operate on larger images dataset seems promising. Also, we have made some arbitrary choices for certain parameters, which may not exactly correspond to our CMOS circuits. We need to synchronize them in the end .



a.

```
tensor([[1., 0., 0., 0., 0., 0., 0., 1., 1., 0.],
[0., 0., 0., 0., 0., 0., 1., 0., 1., 0.],
[0., 0., 0., 1., 1., 1., 0., 0., 1., 0.],
[1., 0., 0., 0., 0., 1., 1., 0., 0., 0.]], device='cuda:0')
```

b.

```
for image1
tensor([0.7233, 0.4005, 0.4843, 0.4446], device='cuda:0')
for image2
tensor([0.0737, 0.8116, 0.4652, 0.1645], device='cuda:0')
for image3
tensor([0.0941, 0.3705, 1.5487, 0.3785], device='cuda:0')
for image4
tensor([0.2269, 0.5476, 0.5856, 0.8382], device='cuda:0')
```

Figure 11. (a) output at the pattern layer: for each of the four input images, there is a unique output firing pattern as represented by each row (b) neural potentials at the one-hot output layer

4. Future Work

In this section, I provide some suggestions for future DR students about possible directions of proceeding with the unfinished project.

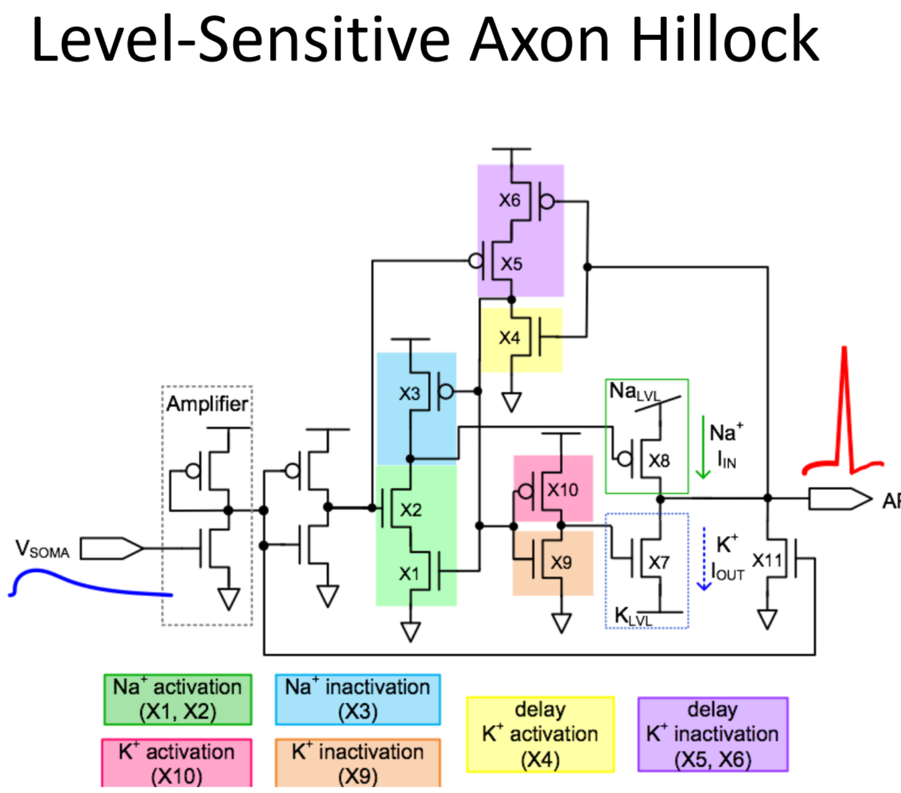
- **Hardware**:
  - As has been explained in the note of *section 2.7 STDP-Dopamine-Noise (SDN) Synapse Circuit*, we need to have a compatible, fully integrated SDN synapse.
  - For now all the circuit components only have *schematics*. We need to implement their corresponding *layouts* as well.
  - *Optional*: it may be interesting to have a STDP circuit that has an adjustable synaptic-weight range spanning both the positive and negative regime. It may not be biological but may be useful if we need such a feature in the software model.
- **Software**:
  - As has been explained in the start of section *3. Python Simulation of SNN*, we need to learn how to use the Python-Spice interface so that we can automate the generation of the spice model of circuits and directly feed it into Cadence for simulation. This is the approach Kun used in his thesis. This topic can be learned and practiced in EE577A.
  - While one may relatively easily master the technique of automatically generating the spice model of circuits if he/she has already taken EE577A, it may be a lot more difficult to automate the generation of layout for the whole neural network. Again, it is impossible to draw all connections by hand because it is too laborious and error prone. From my limited knowledge, it may be an active area of research in the machine learning field.
  - We need to enrich the features of the Python library. For now, it only deals with binary input values. We need to modify the code to be compatible with more quantization levels, which may require rate or temporal encoding. It would be nice if the code is simultaneously adaptable to both. I personally think such a library is extremely useful to the general neuromorphic community.
  - We need to scale up the neural network such that it is deeper (maybe more convolutional layers) and can process larger image sets such as MNIST and CIFAR-10.

References

[1] Cao, Yongqiang, Yang Chen, and Deepak Khosla. "Spiking deep convolutional neural networks for energy-efficient object recognition." International Journal of Computer Vision 113.1 (2015): 54-66.
[2] Cao, Yongqiang, Yang Chen, and Deepak Khosla. "Spiking deep convolutional neural networks for energy-efficient object recognition." International Journal of Computer Vision 113.1 (2015): 54-66.
[3] Bohte, Sander M., Joost N. Kok, and Johannes A. La Poutré. "SpikeProp: backpropagation for networks of spiking neurons." ESANN. Vol. 48. 2000.
[4] Mostafa, Hesham. "Supervised learning based on temporal coding in spiking neural networks." IEEE transactions on neural networks and learning systems 29.7 (2017): 3227-3235.
[5] Mozafari, Milad, et al. "First-spike-based visual categorization using reward-modulated STDP." IEEE transactions on neural networks and learning systems 29.12 (2018): 6178-6190.
[6] Pfeiffer, Michael, and Thomas Pfeil. "Deep learning with spiking neurons: opportunities and challenges." Frontiers in neuroscience 12 (2018): 774.
[7] Tavanaei, Amirhossein, et al. "Deep learning in spiking neural networks." Neural Networks 111 (2019): 47-63.
[8] Smith, Steven W. "The scientist and engineer's guide to digital signal processing." (1997), Chapter 18.
[9] https://github.com/haoqindeng/DR-_SUMMER2021

Appendix

1. Axon Hillock Circuit Tracing



The initial conditions are: X6 is on; X5 is on; X4 is off; B is high, so X1 is on; C is high, D is low.

When Vsoma rises, A rises, turning on X2, which pulls down C, which turns on X8, causing AP to rise.

When AP rises X6 is off and X4 is on, which lowers B, which increases D through the inverter, which turns on X7, which pulls down AP; also, the lowered B increases C, which turns off X8, which pulls down AP.

When Vsoma is lowered, A is lowered, so is the gate of X11, making AP to stay low, which makes B high, C high, and D low, as is the initial condition

2. Full Figures

All full figures can be found here: https://github.com/haoqindeng/DR-_SUMMER2021