Project Neuron: Simulating superconducting nanowire neuron networks with Python and SPICE

Jesus Lares
EECS, MIT, Cambridge, Massachusetts

Abstract

In her 2020 thesis, Emily Toomey introduced a superconducting nanowire computing element for use in neuromorphic computing (3). Toomey demonstrated a circuit level model for the neuron, which required intricate knowledge of nanowire circuit dynamics to work with. To create spiking neural networks (SNNs) from this model required tuning individual circuit parameters in SPICE by hand; this method of creating networks was not generalizable and required knowledge of superconducting physics and circuit dynamics, which is not strictly necessary for the computer scientist creating spiking neural networks. To solve this issue, we created a layer of abstraction between the network and the hardware implementation by developing the Project Neuron Python package. This package acts as an interface for creating spiking neural networks by providing a graphical SNN framework which compiles into and evolves as a network composed of abstract superconducting neurons. This package allows a computer scientist to create SNNs using a standard graphical model and systematically compiles the network into superconducting hardware such that the hardware network evolves as expected by the graphical model. We used the basic compositional model for spiking neural networks created by Nancy Lynch and Cameron Musco (4) as the graphical model which the user would interface with. In order to systematically compile this graphical model into hardware, an abstract model for the nanowire neuron was developed through further characterizations of the neuron using python and SPICE simulations. The package was successfully used to create generalizable spiking neural networks which evolved as Lynch and Musco's model would predict.

Keywords: Neuromorphic Computing; Superconducting Nanowires.

1 Introduction

The demand for computational power is increasing exponentially due to the ongoing 'data revolution', with new applications like IOT and artificial intelligence demanding high throughput and large amounts of data (1). This problem is compounded by the fact that traditional computing architectures are flat-lining in performance, and may not be able to meet these

demands in the future (2). This energy crises will grow in importance in the coming years, and therefore both alternative computational models and alternative hardware architectures are being considered to increase energy and computational efficiency. Two major innovations which have been proposed as bases for alternative computation are neuromorphic computing and superconducting electronics.

1.1 Alternative Architecture: Neuromorphic Computing

Neuromorphic computing is an approach to computation which attempts to mimic the brain. The brain operates as an enormous network of spiking electrical units named neurons, which interact electro-chemically with one another through synapses. Neuromorphic computation replicates this paradigm by creating artificial spiking units and synapses, then connecting each unit together in a type of artificial neural network called a spiking neural network. This computational architecture is an alternative to the traditional von Neumann model of computation, and has been implemented previously using CMOS in IBM's TrueNorth chip and simulated using analog circuits by Stanford's Neurogrid (5).

1.2 Alternative Hardware: The Superconducting Electronics

Using the effects of kinetic induction in paired electrons within superconducting nanowires, it was found that physical spiking behavior could be evoked from paired wires. This behavior is essential for the modeling of neurons, given that neurons within the brain function as electric spiking units.

1.3 Project Neuron: Abstracting the Nanowire Neuron

A new approach to computing was introduced by Emily Toomey in her 2020 thesis which combines both neuromorphic architectures with superconducting hardware. Toomey merged these two novel concepts by creating an electronic device which behaves as a new computational element. This device is called the superconducting nanowire neuron, which behaves as a spiking unit and replicates important behaviors of real neurons (3).

While Toomey created the computational device in her 2020 thesis and replicated ba-

sic networks as a proof of concept, a systematic method for creating networks from these neurons was never specified. Without a systematic method, network creation required knowledge of superconducting physics and the nanowire neuron circuit dynamics. This is because each neuron would have to be manually placed and all network parameters, of which there were tens per neuron, would have to be chosen by hand. This required thorough calculations, an understanding of how the neuron behaves, and generally increased the time and effort it took to create networks in the nanowire hardware. This lack of abstraction greatly limited the complexity of networks which could be created with the nanowire neuron, as well as who could create them. These problems is further amplified by the fact that the neuron, as an electronic device, is specified in SPICE code (which is a common description and simulation language for electronics). While SPICE is often known by electrical engineers, the computer scientists who usually create spiking neural networks are generally less familiar with the language. This placed another barrier as to who could be creating and simulating networks with the nanowire neuron.

General models which describe spiking neural networks already do exist, and are more commonly known by computer scientists within the field of neuromorphic computing. These models specify how spiking neural networks should behave and provide a systematic method for the creation of complex networks. In order to solve the aforementioned problems limiting the complexity and use of our nanowire networks, we set out to create an abstraction at the level between network specification and hardware implementation. We leveraged the existing spiking neural network models by creating a software suite which allows users to specify a network using a common model, then have their network automatically compiled into a hardware implementation using nanowire neurons. This hardware description could then go on to be easily fabricated, or simulated using LTSpice (an open source SPICE simulation tool).

1.3.1 Project Neuron Python package

The software suite we created is a python package called Project Neuron. The use cases for project neuron can be split into three main areas:

1. Provides a graphical spiking neural network model which a user can use to specify a graphical neural network, as well as input sequences to the network.

2. Contains functionality to simulate the time evolution of the network defined by the user. The evolution will be calculated using a phenomenological model of the nanowire neuron which will be further discussed in methods.

3. Contains functionality which compiles and stores the SNN model as a hardware implementation, specifically using the nanowire neuron.

This package structure and use of the package will be further discussed in the methods section.

2 Related Works

To understand the purpose of Project Neuron, it is necessary to understand the Superconducting Nanowire Neuron. Namely, it is necessary to understand the fundamentals of the electrical device model and how the model is used as a computational unit in spiking neural networks.

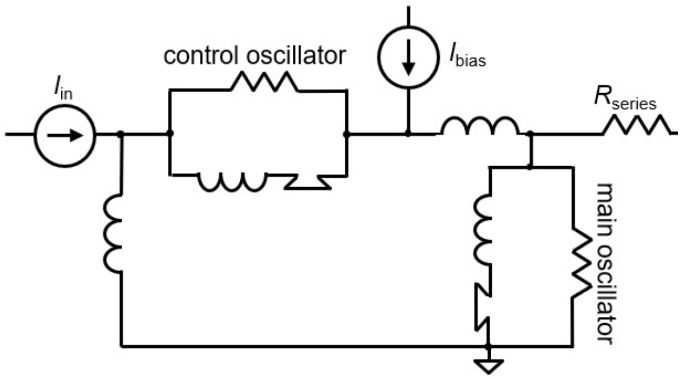

Figure 1: Simplified model of the superconducting nanowire neuron, consisting of two nanowire oscillators, an input current, a bias current, and an output current. Figure taken from Ref. (3).

2.1 Brief Overview of the Superconducting Nanowire Neuron

The superconducting nanowire neuron replicates the spiking behaviors of a biological neuron. It does this by coupling two nanowire oscillators together, and biasing both of them with a current. For the purposes of project neuron it is necessary to understand that, depending on the device parameters, a certain amount of input current will cause the unit to output a spike in current and voltage - the spike is shown in figure 2. It is also apparent in figure 1 that, in this simplified model, there are already 10 parameters to set per neuron; a more detailed model would reveal that the two nanowires also necessitate the tracking of several more. Toomey fully describes the electrical dynamics of this device in the 2020 thesis (3).

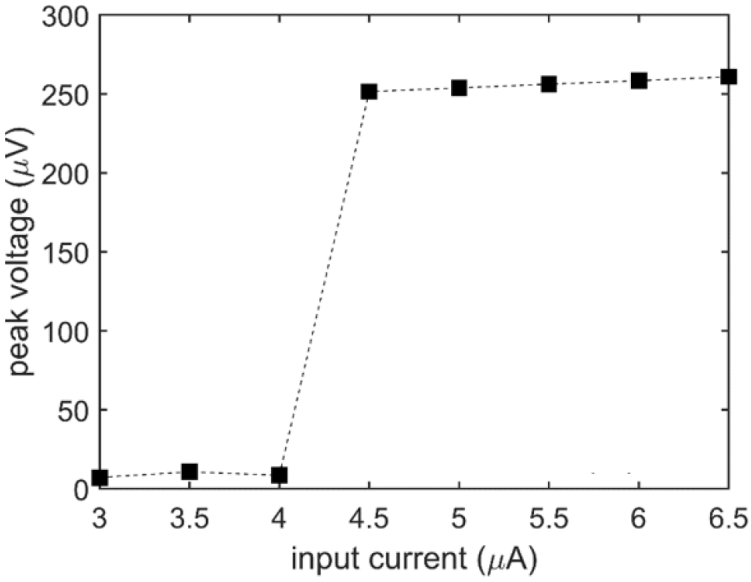

Figure 2: A graph plotting peak output voltage as a function of input current, with appropriately set circuit parameters and a bias current $I_{bias} = 58.6\mu A$. Figure taken from Ref. (3).

2.2 Brief Overview of the Neuron Synapses

Nanowire neurons are connected to each other through a synapse, which introduce several more parameters to track per connection. The synapses mimic the slow release of neurotransmitters in biological neurons, which cause their ion-channels to open and thus begin

to acquire charge and eventually fire. The synapses will become activated by an upstream neuron firing, will charge up, and then slowly release charge, which eventually causes the downstream target neuron to which they are connected to fire. To example synapses which mimic this behavior were demonstrated by Toomey, and are shown as examples in figure 3. Project Neuron uses a different type of synapse in its hardware implementations in order to achieve a higher fan-out than these synapses.

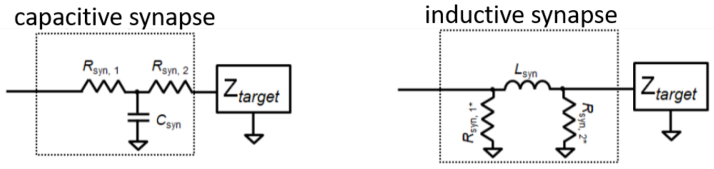

Figure 3: A simplified model of the capacitive and inductive synapses depicted in Toomey's thesis. In a network, Z_{target} would usually represent the input of a downstream neuron. Figure taken from Ref. (3).

2.3 Compositional Graphical Model of Spiking Neural Networks

Nancy Lynch and Cameron Musco introduced a compositional graphical model for spiking neural networks in their 2021 paper (4). This model can fully describe networks as a set of neurons (nodes) which are connected to each other through synapses (edges). Each neuron in the graph has an internal bias, temperature, and potential, which essentially determine the amount of input 'current' which is required to make them spike at any given timestep. Similarly, each edge has an edge strength which determines how much 'current' is delivered to a downstream node when the connected upstream node fires.

This model is highly useful for the purposes of Project Neuron for two reasons: the aforementioned behaviors of the model are analogous to the behaviors of the superconducting hardware neurons, and these graphical models for spiking neural networks are commonly familiar among computer scientists working on neuromorphic algorithms. These reasons make the model an invaluable tool in creating an abstraction between SNN specifications and SNN hardware implementations. Project Neurons will allow the user designing an algorithm to work directly with this abstract model, while converting their network into

superconducting hardware automatically, with no user input required.

3 Methods

As mentioned in the introduction, Project Neuron has three primary functionalities:

1. Provides a graphical SNN for users to specify networks.
2. Provides simulation capabilities to solve the time evolution of the network according to the abstract nanowire neuron behavior.
3. Compiles and stores the SNN as a hardware implementation.

This section will discuss how Project Neuron achieves these functionalities.

3.1 Specifying the Spiking Neural Network

Project Neuron allows users to specify a network in python by providing a neuron, synapse, and spiking neural network class. The spiking neural network class represents the network by maintaining a set of all nodes and edges, with nodes being neuron objects and edges being synapse objects.

The network specification in Project Neuron follows Lynch and Musco's model closely. Neuron objects have temperature and bias parameters, and synapses have synapse strength parameters. These neurons and synapses are then appropriately converted to their abstract hardware counterparts for simulation and hardware descriptions, which will be described in sections 3.2 and 3.3.

3.1.1 General Graphical Model

Our graphical model closely follows Nancy and Musco's (4); a brief overview of there model will be provided here. The graphical model provided by project neuron consists of a network with a set of nodes and a set of edges between the neurons. The nodes are neurons and the edges are synapses, with a synapse between any two connected neurons. This model is inherently probabilistic - this is because biological neurons also behave probabilistically.

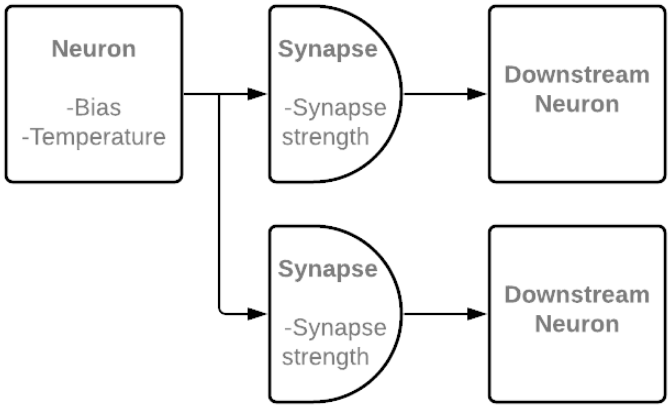

Figure 4: An example of a basic network created by through Project Neuron. Each neuron has a bias and temperature parameter and each synapse has a strength parameter. This network consists of one upstream neuron connected to two downstream neurons.

Furthermore, the model evolves in discretized timesteps.

Each neuron maintains an internal potential at a given timestep which changes in the next timestep depending on the state of the upstream neurons which it is connected to. If enough upstream neurons are firing, and the synapse strengths are large enough, then this neuron will fire in the next timestep and affect the neurons downstream to itself.

Specifically, a neuron evolves according to the model specified below. At each timestep, a neuron u has a potential:

$$potential(u) = [\sum_{(v,u)\in E} C'(v) strength(v,u)] - bias(u)$$

Where (u, v) is an edge between some node v and the node in question u with synapse strength $strength(v, u)$. $C'(v)$ is an indicator variable which is equal to one if v fired in the previous timestep, and zero otherwise. This potential is then transformed into a firing probability for that timestep using the following relationship:

$$p_u = \frac{1}{1+e^{\frac{-potential(u)}{\lambda}}}$$

Where λ is the internal temperature parameter for the neuron u. For more detailed information about the model, refer to Nancy and Musco's 2021 paper (4).

3.2 Compiling the Network into Hardware

While the graphical model provided to users for network specification follows Nancy and Musco's model, the abstract hardware model does not follow it as closely. The main difference between the two is that the graphical model is designed to be probabilistic, while the superconducting nanowire neuron simulations are not. There is a caveat to this, however, because real superconducting nanowires have been shown to exhibit probabilistic behavior (3). That behavior, however, has not yet been implemented into the superconducting nanowire electrical model. For now, the hardware representation will remain deterministic, however this is open for change in the future.

In order to bridge this gap, Project Neuron translates any firing probability over 50% to deterministically fire in the hardware model, while any probability under 50% results in the neuron not firing.

3.2.1 Abstract Neuron and Synapse Models

In order to systematically translate neurons and synapses from the graphical model to the hardware model, a baseline hardware neuron and synapse were be chosen for further characterization. The behavior of these model hardware units were used to create the abstract hardware model.

The baseline abstract neuron is specified as the hardware nanowire neuron depicted in figure 5 with the variable parameter being I_{bias}, which is set to a baseline value of $58\mu A$. With these parameters set, the neuron model has a firing threshold of $3.72\mu A$. The nanowire used in this model is simulated using the spice model developed in reference (6), and any parameters not specified are left as default.

The baseline abstract synapse is specified as shown in figure 6. The synapse used differs from Toomey's model, as it uses an Htron in its design. This design choice came from

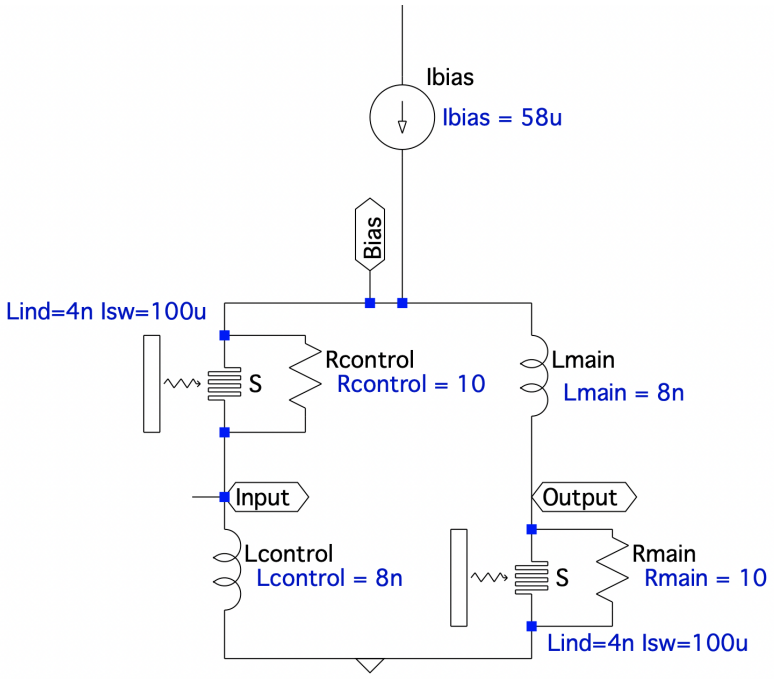

Figure 5: SPICE model for the baseline nanowire hardware neuron. Units for inductances are are H, resistances Ω, and currents A. Unit prefix u means micro and n means nano.

simulations we ran which demonstrated that an Htron synapse results in higher fanout. The Htron synapse is connected to the upstream neuron through the Htron itself, which is switched through heat which is produced in its resistor via Joule heating. Enough output voltage from the upstream neuron can produce enough heat to switch the synapse, and thus produce a spike for the downstream neuron.

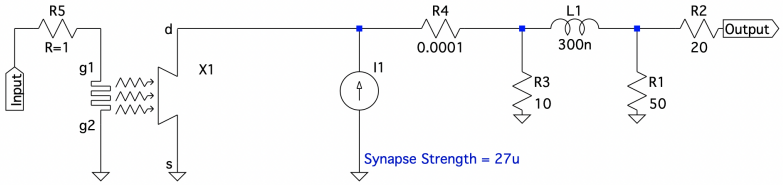

Figure 6: The baseline Htron synapse. In the abstract model, the variable parameter is the synapse strength, which is defaulted to $27\mu A$.

Both of these baseline models were fully characterized in spice, and relationships between their firing threshold, output strengths, and internal bias currents are explained in results. These linear characterization are what Project Neuron uses to translate from the graphical

model into the hardware implementation. The current translation schema is as follows:

Graphical Model	**Nanowire Hardware Model**
0 neuron bias is mapped to	Bias current of $60\mu A$
each additional unit of bias	$.1\mu A$ decrease in neuron bias current
0 synapse strength is mapped to	Bias current of $27\mu A$
Each additional unit of strength	$1.16\mu A$ increase in synapse bias current

This schema is derived from the the linear equations for the neuron and synapse found in the results section. It is designed such that, at any timestep, a neuron in the graphical model with over 50% probability to fire will fire in the hardware implementation.

3.2.2 Translating Abstract Models into Hardware

To translate these abstract models into hardware is a simple procedure. SPICE subcircuits were made for both the baseline modes with the relevant variable parameters being exposed. Project Neuron takes the user defined graphical model and writes the network into a SPICE netlist according to the schema provided in the previous section. This netlist fully specifies the user-created network in nanowire hardware and is capable of being simulated as-is in LTSpice.

3.2.3 Storing Hardware Implementations

The hardware implementation for the network is encoded in SPICE and stored in the netlist data format. The nanowire neurons and synapses are formatted as subcircuits in the network which we created ourselves, and nested within these subcircuits are nanowire subcircuits created by reference (6). Further additions to project neuron will add functionality to translate the neurons into verilog.

3.3 Simulating the Spiking Neural Network

There are two ways to simulate networks created with Project Neuron. The purpose of the package is to create nanowire hardware implementations of neurons, and therefore the

hardware simulation is most pertinent. However, an evolution of the network according to the basic compositional model in (4) is also provided in order to benchmark the results of the hardware implementation.

3.3.1 Graphical Model Simulation

In order to simulate the graphical model using Nancy and Musco's model, a simulation object is created which is set to use the phenomenological model described in section 3.1.1. This simulation will then evolve the graphical model itself and return the firing history for each neuron in the graph.

3.3.2 Hardware Model Simulation

In order to simulate the hardware implementation, currently the compiled netlist must be ran in LTspice. LTspice then returns the firing history for not only each neuron, but each electrical node in the network. This kind of detail is entirely unnecessary, and is prohibitively slow. The next step in Project Neuron is to create a behavioral model for the neuron in python which will allow for a great simplification in computation the ability to simulate nanowire hardware networks much faster.

4 Results

The methods discussed above successfully generalized to the point where two example spiking neural networks were generated systematically from the superconducting nanowire hardware. This was possible due to the fact that we successfully found a linear regime in which the neurons and synapses act compostionally. The networks we implemented are the identity network and the n-neuron AND network which are described in the Lynch and Musco model (4); the hardware implementation for both networks matched the expected behavior according to their model.

4.1 Finding a Compositional Regime in the Neuron

Once the standard neuron and synapse parameters were set, experiments were ran on both the neuron and synapse to determine the voltage regimes in which they would act

compositionally.

4.1.1 Firing Threshold

A compositional region was found for the firing threshold of the neuron around $58\mu A$ of bias current, which fires at around $3.72\mu A$ of input current. Furthermore, it was found that the linear relationship between bias current and firing threshold is as follows:

$$\phi_{firing} = \phi_0 - 1.85(I_{bias} - I_0)$$

ϕ_{firing} is the firing threshold for the neuron, measured as the input current required to spike the neuron (in μA).
I_{bias} is the current biasing the neuron.
ϕ_0 is the firing threshold around which this approximation holds, and is equal to $3.72\mu A$.
I_0 is the bias current around which this approximation holds, and is equal to $58\mu A$.

This approximation is meant to be used around ϕ_0 and I_0, however it has been experimentally shown to hold with biases which are tens of μA away from I_0. The relationship provides a simple, systematic, way to bias a neuron in a linear fashion similar to Lynch and Musco's model. If a neuron must be biased in such a way that to demand a higher input, the bias must accordingly made smaller.

4.1.2 HTron Synapse

A compositional region was also found for the HTron synapse which is used for connections between individual neurons. The parameters of relevance here are the bias to the synapse, also referred to as the 'synapse strength', and the output current of the synapse.
We chose to center the linear region around a synapse strength of $27\mu A$ because that strength resulted in an output current of $3.75\mu A$ on a discharge, which is just enough to spike a downstream neuron biased with $58\mu A$. Of course, as previously mentioned, the firing threshold for the downstream neuron can be altered to require more than $3.72\mu A$ of input current, however for the purposes of this approximation we assumed that the average use case would see currents around these levels. The linear relationship between synapse

strength and output current on discharge is as follows:

$$I_{out} = I^0_{out} + .16(I_{bias} - I^0_{bias})$$

I_{out} is the current output current if the synapse discharges.
I^0_{out} is the discharge output current around which this approximation holds, and is equal to $3.75\mu A$.
I_{bias} is the current synapse strength.
I^0_{bias} is the synapse strength around which this approximation holds, and is equal to $27\mu A$.

It was also found that synapse outputs behave compositionally around these current ranges. What this means is that the output currents of multiple synapses going to the same neuron input will add without any significant complications from the rest of the circuit. This is integral to the general creation of networks from our hardware, and is a principle reason why the general AND network shown below can function.

4.2 Identity Network

The output neuron simply replicates the behavior of the input neuron: if the input is spiking then the output will spike, if it does not spike then the output will not spike. This example serves as a sanity check for project neuron, the neuron and synapse models, and the linear regimes.

4.3 And Network

This is an AND network which behaves identically to the AND network depicted in Lynch and Musco's model (4). If, at any time, all 3 inputs are firing then the output will fire. If less than 3 inputs are firing, then the output will not fire. Three inputs was chosen arbitrarily, and this generalizes to an N-input AND gate by increasing the firing threshold of the output neuron. This can be easily accomplished by using the firing threshold linear approximation.

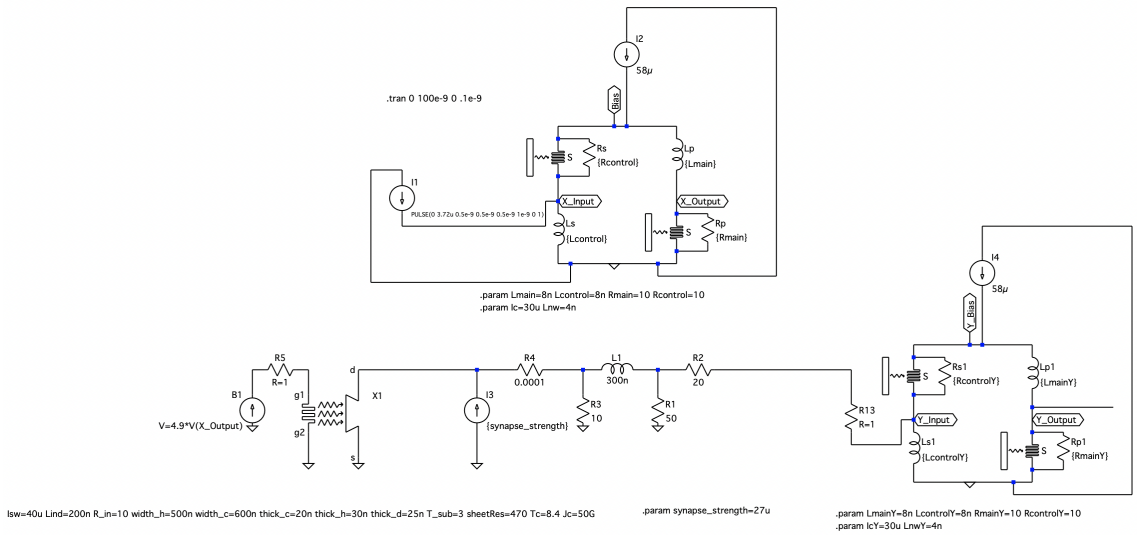

Figure 7: The LTSpice structure of the identity network. The neuron on top is the input, the synapse runs along the bottom, and the neuron on the right is the output. The behavior from the top neuron is ran through the synapse with a behavioral voltage source to increase computational efficiency in LTSpice (instead of directly connecting the two components). Both neurons have a firing threshold of $3.72\mu A$, and the synapse is biased with $27\mu A$.

4.4 Project Neuron

The python package, while still in development, was of great use in reaching the insights demonstrated in this paper. The project neuron package was used to run and analyze LTSpice simulations sequentially to find the compositional regimes in the neuron and was also used to create and simulate the identity network as well as the AND network. This package will be placed online and made freely available for any who want to simulate spiking neural networks, either with or without any hardware implementations. It will be maintained and updated to provide more functionality for both LTSpice compilation and analysis, as well as the simulation of spiking neural networks.

Acknowledgements

I acknowledge the invaluable help of Andres Lombo with LTSpice, as well as the guidance of the entire QNN.

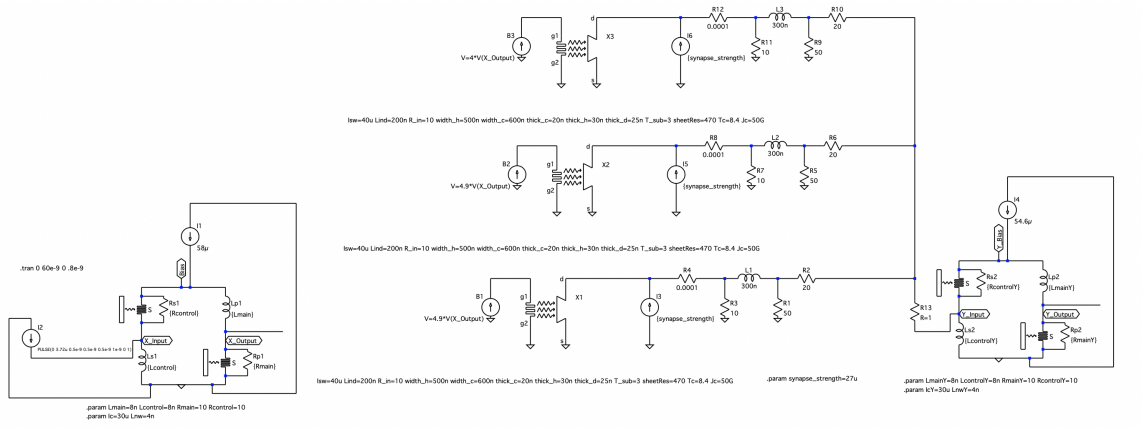

Figure 8: The LTSpice structure of the AND gate. The left neuron is the input, the three synapses are in the middle and are all connected to the right neuron, which is the output. All 3 synapses are biased with $27\mu A$ (which results in a current discharge of $3.75\mu A$), the input neuron has a firing threshold of $3.72\mu A$, and the output neuron has a firing threshold of $9.9\mu A$. Note that while there is only one input neuron, its voltage is replicated and scaled in each synapse input with a behavioral source to replicate 3 input neurons (this increases LTSpice computation speed).

Bibliography

[1] dcadmin. *Rebooting the IT Revolution: A Call to Action.* Semiconductor Industry Association. url: https://www.semiconductors.org/resources/rebooting-the-it-revolution-a-call-to-action-2/.

[2] Nihar R. Mahapatra and Balakrishna Venkatrao. "The processor-memory bottleneck: problems and solutions". *In: XRDS: Crossroads, The ACM Magazine for Students 5.3* (Apr. 1, 1999), 2–es. issn: 1528-4972. doi: 10.1145/ 357783.331677. url: https://doi.org/10.1145/357783.331677.

[3] Emily Toomey. "Superconducting nanowire electronics for alternative computing". *Submitted to the Department of Electrical Engineering and Computer Science at MIT.* (May 14, 2020).

[4] Nancy Lynch and Cameron Musco. *A Basic Compositional Model for Spiking Neural Networks.* (February 14, 2021).

[5] Steve Furber. “Large-scale neuromorphic computing systems”. In: *Journal of Neural Engineering* 13.5 (Aug. 2016), p. 051001. issn: 1741-2552. doi: 10. 1088/1741-2560/13/5/051001. url: https://doi.org/10.1088%2F1741- 2560%2F13

[6] Karl K Berggren, Qing-Yuan Zhao, Nathnael Abebe, Minjie Chen, Prasana Ravindran, Adam McCaughan, and Joseph C. Bardin. “A superconducting nanowire can be modeled by using SPICE”. In: *Superconductor Science and Technology* 31 (2018), p. 055010. url: https://doi.org/10.1088/1361-6668/aab149