Evaluating Neuron Models for a 130-nm Spiking Neural Network Hardware Accelerator

Amy Liu
Electrical and Computer Engineering
University of Michigan
Ann Arbor, MI
amyzliu@umich.edu

Finn Moore
Electrical and Computer Engineering
University of Michigan
Ann Arbor, MI
fdm@umich.edu

Kristen Schang
Electrical and Computer Engineering
University of Michigan
Ann Arbor, MI
keschang@umich.edu

Isaac Schuster
Electrical and Computer Engineering
University of Michigan
Ann Arbor, MI
isaaczzz@umich.edu

Emily Su
Electrical and Computer Engineering
University of Michigan
Ann Arbor, MI
syuchien@umich.edu

***Abstract*—In recent years, the investigation of neural networks has attracted a large audience of researchers in various fields. Neural networks have wide-ranging applications in computer vision and machine learning, and can be useful for low-power applications such as surveillance systems or wearable devices. One type of neural network, spiking neural networks (SNNs), uses binary spikes as inputs and outputs to each node, or "neuron," in a network. This characteristic allows SNNs to have less power consumption than traditional neural networks. Furthermore, hardware accelerators for neural networks offer significant performance speed-ups and increased energy efficiency over their software counterparts. In a digital hardware implementation of an SNN, neurons can be modeled in many different ways; the way they are modeled determines how a neuron triggers an output spike and has ramifications on the power and accuracy of an SNN implementation. The neuron models investigated in this work are Integrate-and-Fire (IF); Leaky-Integrate-and-Fire (LIF) with exponential decay; and Leaky-Integrate-and-Fire with linear decay (LLIF), which is used to approximate an exponential decay factor. Digital hardware implementations of an SNN with each of the neuron models were designed in SystemVerilog, and the accuracy, power, and area among each of these implementations were compared. It was found that the use of digital exponential leak logic creates significant power and area overhead with marginal accuracy benefits; linear leak logic is more feasible for a hardware implementation. Overall, our implementation achieved an accuracy of 95.0% when tested on 10,000 inputs from the MNIST dataset, and has a throughput of 31.35K images/second with 13.71 mW power consumption and 7.51 mm^2 area. In general, for simple data sets like MNIST, IF neurons, without the overhead of any leak logic, may be the most viable solution in terms of balancing performance, efficiency, and accuracy.**

I. Introduction

The investigation of spiking neural networks (SNNs) is an emerging subdivision of machine learning. Inspired by the biology of real neurons, SNNs are a unique take on neural networks with promise for low-power applications [1]. SNNs are made of a large number of individual nodes called neurons, and information is passed from neuron to neuron through spikes – similar to a single binary number 0 or 1. These signals encode information in the frequency of the spikes, as well as a weight assigned to the spike source. Each of the weights, depending on the input spikes, is summed in the membrane potential of each neuron. If this potential crosses a set threshold, the neuron sends out a spike and resets its potential. Based on the neuron architecture used in the SNN, implementation complexity or accuracy can be traded for lower power or area in a desired application.

A large portion of the energy savings SNNs can provide comes from their sparse nature and absence of multiplication operations. They are well-suited to serve sparse inputs where data is received in small bursts, such as when input data to the model has been rate-encoded. Rate encoding means that instead of having one image, a set of binary spike trains represents a single time step; each pixel is represented by a rate instead of a singular value. For example, a pixel that is brighter will spike more frequently than a pixel that is dark. Figures 1 and 2 show these rate-encoded spikes, which are also temporally encoded as they enter and leave a neuron.

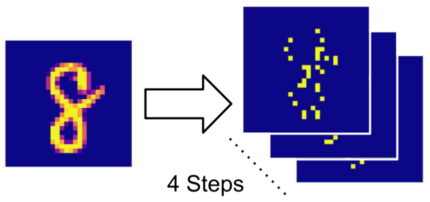

Fig. 1. Image intensity rate encoding of MNIST dataset.

Using spikes as the fundamental unit in computation, SNNs are able to utilize less power-intensive operations to offer the same performance as traditional neural networks. This computational framework offers a unique opportunity to create hardware that is specifically designed for SNNs. The chal-

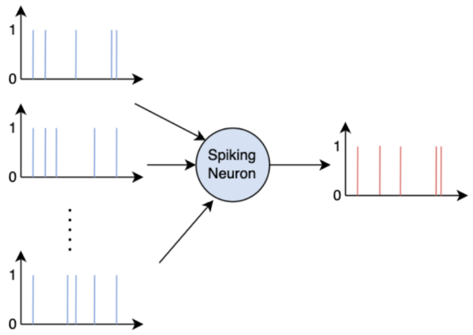

Fig. 2. Spiking neural networks (SNNs) encode information with temporal binary spikes. The single spiking neuron takes in binary spikes as inputs, performs computations, and returns binary spikes as outputs.

lenge in developing SNN hardware comes from selecting the most optimal combination of features – like learning rules, STDP algorithms, and type of neuron – that minimizes power consumption without sacrificing correctness. One issue that is particularly relevant for a hardware-based implementation is the tradeoff between different models of leak logic in each neuron. For example, an Integrate-and-Fire (IF) neuron model, which saves the membrane potential in a neuron each cycle without decay, and different types of Leaky-Integrate-and-Fire models (LIF or LLIF), which have the membrane potential in a neuron decay each cycle, are types of neuron models that are commonly used in SNNs. IF neuron models are simpler to implement but often come at the cost of some accuracy; meanwhile, an LIF neuron model may be more accurate, but hardware implementations of the LIF or LLIF neuron models are more complex and may be overly power-intensive. Neurons with decay are generally expected to be more accurate than those without decay, as there is an implicit bias towards more recent input than older inputs. This paper aims to answer the question of which neuron model or implementation of leak logic in hardware is most worthwhile. The goal of the project is to determine an optimal hardware implementation for SNNs that balances power, performance, and accuracy, by experimentally developing and testing different combinations of neuron models in a hardware-based SNN implementation.

Additionally, the hardware implementation of an SNN investigated in this paper can be especially advantageous for applications which require low power consumption [2]. Most current hardware implementations of SNNs implement adaptive learning algorithms such as spike-timing dependent plasticity (STDP), which are not required for applications in which models can be trained separately from the hardware, and their corresponding weights can be pre-computed and stored [3]. The implementation proposed in this paper forgoes this functionality, as it may be extraneous for simple applications that do not require real-time adaptive learning, and thus saves on the power that this feature would require and lowers the complexity of the hardware. In these applications, battery life is often a primary concern. For example, in use cases such as traffic surveillance or wearable devices, a simpler implementation that forgoes power-intensive training techniques in favor of a simpler model with a longer battery life is advantageous. In these cases, the implementation described in this paper, which has low power consumption and hardware complexity compared to traditional models, can be more desirable than other implementations of spiking neural networks. Furthermore, choosing a neuron model that prioritizes power at the tradeoff of some accuracy may be beneficial for these applications.

The rest of this paper is as follows: Section III is Design and Section IV is Testing Methodology. Our results are in Section V, and Section VI is the Conclusion. Lastly, we wrap up our paper in Section VII with Team Member Contributions.

II. Background

Advancements in SNN algorithms and emulation tools have facilitated the application of SNNs to numerous fields. SNNs provide the potential to bring low-overhead AI models to areas where they were previously unavailable. As well, continuing advances in accelerators for SNNs allow for much more research into this field. Previous implementations of neuromorphic chips that support spiking neural networks have reported power savings. Some examples include Loihi [3], TrueNorth [4], and Chen et al. [5]. There have also been multiple FPGA-based implementations of SNNs [6], [7]. These implementations use techniques such as approximate computing and parallel processing to optimize power and speedup. To test and develop an SNN, many implementations use the Modified National Institute of Standards and Technology (MNIST) database, a large set of handwritten digits that can be used to train models for image classification. Many of these image classification SNN-based accelerators have proven to be energy-efficient while high in accuracy, and our hardware-based spiking neural network implementation will be compared against these models. Additionally, these existing implementations do not analyze the effects of using different neuron architectures. As such, using differing models of the decay functionality in the LIF unit, or instead using an Integrate-and-Fire (IF) unit, has power, area, and accuracy tradeoffs which remain to be evaluated.

A. ISSC'19

Park et al. [8] developed a 65nm on-chip neuromorphic processor that allowed for energy-efficient on-chip learning and used a direct spike-only feedback mechanism. Their hardware consists of an input layer containing the spike converters, two hidden layers containing 200 neurons each, and an output layer containing 10 target neurons. An SRAM is shared by each of five neurons in the hidden layer for storing weights and enhancing area efficiency. At the training stage, weight updates are made out-of-order for forward passes to proceed without waiting for weight updates to complete and are skipped when deemed redundant. When trained on the MNIST dataset, their processor achieves a 97.83% accuracy rate with only approximately 7.5% energy overhead for inference. The design consumes 23.6 mW in power, with a throughput of 100K img/s and a total area of 10.08 mm^2.

B. VLSI'17

Buhler et al. [9] designed a 40nm digital-analog hybrid spiked-based neural network processor. Their hardware implements the Locally Competitive Algorithm (LCA) that relies on 512 analog LIF neurons, which are 3x smaller and 7.5x less power consuming than the digital counterpart. The LIF neurons are organized into two 256-neuron cores that each consist of 64 neurons connected through a bus-ring topology with shared SRAM for storing weights. The design allows for spikes from any neuron to reach other neurons within 8 clock cycles and offers flexible core configurations. Their chip achieves approximately 88% accuracy when trained on MNIST, with a throughput of 1.7M img/s and a total area of 1.31 mm^2.

C. μBrain

μBrain [10], developed by Stuijt et al., is a 40nm fully digital, SNN-based, and event-driven neuromorphic processor that is ultra-low-power. The design uses a non-Von-Neumann architecture, replacing global clocks with a delay-cell based oscillator system. The processor's architecture consists of a recurrent fully-connected layer of 256 IF neurons and two fully connected layers of 64 and 16 IF neurons. Memory and processing are co-located with each neuron, allowing local spike computations without centralized memory accesses. When trained on MNIST, μBrain achieves an accuracy of 91.7% while consuming only 73 μW of power. The chip has a total area of 2.82 mm^2 and a throughput of approximately 238 img/s.

III. Design

A. Software Implementation

The snnTorch [11] Python library was used for the simulation and training of SNNs, with both a neuron architecture that did not implement leak logic and one that implemented exponential leak logic. The training was done using backpropagation through time (BPTT) with surrogate gradients. Each model was trained for handwritten digit classification on the MNIST dataset, rate encoded over 4 steps.

B. Neuron Models and Architecture

At the core of an SNN design is the neuron architecture; each neuron, of which there may be hundreds, is woven together to create the overall network. At a high level, each neuron receives an array of inputs, either from the previous layer or from the input to the system. Each of the weights stored in the neuron is summed with the corresponding neuron bias and stored in the neuron's membrane potential if its corresponding input spike is high. If the membrane potential exceeds a threshold, then the neuron will fire a spike to its output. The intricacies in the neuron architecture come from the storage of the membrane potential; to better simulate a real neuron and the concept of "memory," logic for the decay of the stored membrane potential can be added to the neuron architecture.

Our design compares three different neuron types: Integrate-and-Fire (IF); Leaky-Integrate-and-Fire (LIF); and Linear-Leaky-Integrate-and-Fire, i.e., LIF with a linear decay model, to approximate the exponential decay factor (LLIF). What differentiates the different models is the ability for the neuron to simulate decay in the membrane potential. In the human brain, if a membrane potential doesn't reach a threshold, it will decay over time [11]. This characteristic allows old inputs to have less of an impact on the output spikes than newer inputs do. An IF neuron doesn't perform any decay. An LIF neuron has exponential decay in its stored membrane potential every cycle. An LLIF neuron simulates this decay linearly; a constant amount is subtracted from the stored membrane potential every cycle. Table I shows the different decay types among the investigated models, and the corresponding estimated accuracies and power consumptions for the implementations.

TABLE I
Expected performance of neuron models

Neuron Models	Decay Type	Est. Accuracy	Est. Power
Integrate-and-Fire (IF)	None	High	Lowest
Leaky Integrate-and-Fire (LIF)	Exponential	Highest	Highest
LIF with Linear Decay (LLIF)	Linear	High	Low

Before inference can be performed, the neurons must load their corresponding weights and bias. The bias, which is treated as a single weight that is always added to the membrane potential, is read in first. Next, the weights are read in cycle by cycle until all the weights have been read in. After that, inference can happen, and the input spikes are sent to each neuron. One input spike is consumed every cycle. For example, in a layer with 100 inputs, it takes 100 cycles for one time step to be completed. The benefit to doing this step over the span of many cycles, as opposed to reading all the input spikes in at once, is to reuse hardware components in the neuron and lower the power consumption and area of the design. An initial design, in which all inputs were consumed in one cycle, was implemented. However, the amount of hardware in each neuron was too large to be practical. Instead, we opted for a minimal amount of hardware at the cost of some performance. Future work may include finding a more optimal balance between performance and hardware costs (i.e., area and power). After the input spikes are sent to each neuron, it will send a corresponding output spike to the next layer.

The architecture for the IF neuron, as seen in Figure 3, contains logic for implementing the functionality mentioned above. It accumulates all the weights that have a high input spike and the bias, performs a comparison against a threshold, and outputs a spike if the condition is met. The architecture for the LIF and LLIF neurons, as seen in Figures 4 and 5, respectively, shows the mechanisms for decay in the membrane potential as well. One note about the LIF neuron is that the hardware it synthesizes to may be different depending on the decay rate chosen; for example, decaying by 50% each cycle would be able to synthesize as a single shift. Other decay factors would require more hardware which would further increase area used. We chose to use an arbitrary decay factor (e.g., divide the membrane potential by 5 every cycle), that would not synthesize as a single shift to show the effects of a more complicated hardware in the LIF neuron.

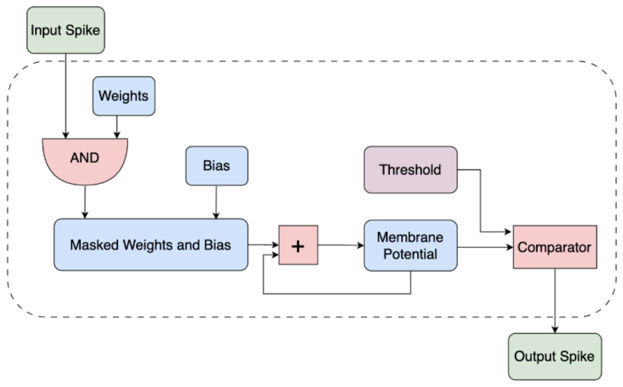

Fig. 3. Architectural layout of the Integrate-and-Fire neuron. The membrane potential is updated every cycle by adding the new masked weights and biases with no leaky logic.

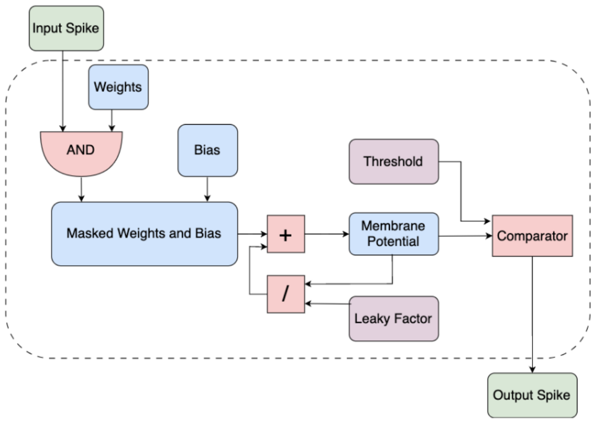

Fig. 4. Architectural layout of the Leaky-Integrate-and-Fire neuron. The membrane potential is divided by the leaky factor every cycle and added by the masked weights and biases.

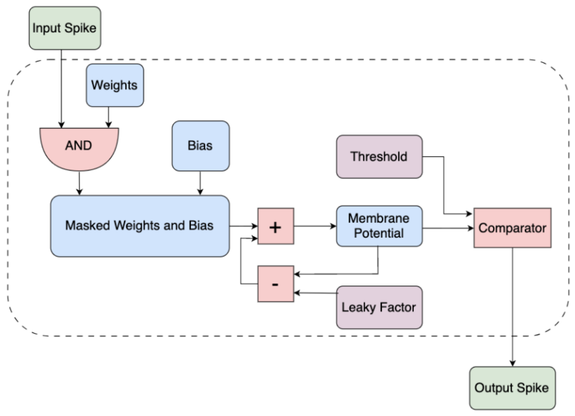

Fig. 5. Architectural layout of the Leaky-Integrate-and-Fire neuron with a linear decay model. The membrane potential is subtracted by the linear-decay leaky factor every cycle and added by the masked weights and biases.

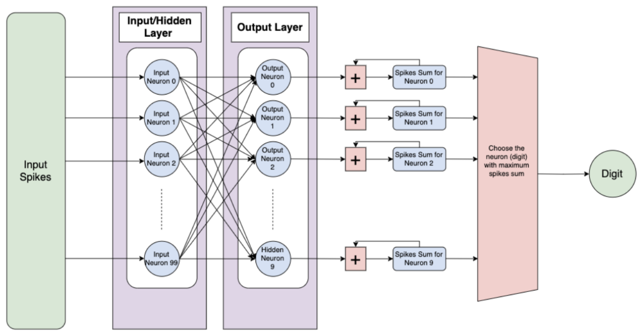

Fig. 6. Full architectural layout of the spiking neural network.

C. Layer and Model Architecture

As described in Subsection B, the neurons in the network are responsible for storing weights and biases. However, our overall architecture, as shown in Figure 6, contains two layers: one input/hidden layer with 100 neurons and one output layer with 10 neurons.

The layer is created as an array of neurons and passes the weights and bias values into the neurons, each of which contains memory to store each value. In the initial start-up phase of our SNN, weights and biases are read in one at a time and then passed into each layer every clock cycle. The layer manages a one-hot bit encoding that represents the neuron it is feeding the weights and biases into. At each clock cycle, the layer updates the one-hot bit encoding and sends the weight or bias value into all of the neurons in the layer. The neuron that corresponds to the bit-encoded value then stores the corresponding weight or bias. Once all the weights and biases are read and passed into all the layers, the initial start-up phase is complete, and inference begins.

During inference, the rate-encoded input spikes are passed in one at a time at every clock cycle into the layers, which are propagated through the two layers. The output spikes are passed into an output module that performs the classification. Over each time step, the output spikes for each of the 10 neurons from the output layer are added, and the neuron with the maximum sum of spikes is identified as the final classified digit. The purpose of the output module is to perform the scorekeeping of the last layer of the network, so that a clean digit can be the output of the hardware.

IV. Testing Methodology

First, unit testing for each module, i.e., the neuron, layer, and output modules, was performed. The testing here was done first with simple test cases to prove functionality, and then done with corner cases for edge case testing. After this, each of the pieces was wired together to create the whole model.

After implementing the versions of the network using IF, LIF, and LLIF neurons in SystemVerilog, the accuracy of each implementation was tested for 10,000 cases. The first step to test each model was to read in the weights and bias, which were obtained through training the snnTorch model in software. To do the training, we ran behavioral simulations for each implementation using Synopsys VCS on 10,000 input streams.

Each design was then synthesized in the IBM 130 nm process using Synopsys Design Compiler with ARM SRAM IP to store the weights. During this step, we optimized to reduce area and power, so the clock cycle remained higher that the minimum value to meet slack, for a final clock period of 10.17 ns, resulting in a clock frequency of 98 MHz. Power and area results were obtained from the synthesis report.

The power, performance, area, and accuracy were evaluated against other published neuromorphic processor designs and were found to have comparable specifications. Finally, the specifications for the models with IF, LIF, and LLIF neurons were compared among each other, and the tradeoffs among them were analyzed.

V. Results

The hardware-based SNN implementation created in this work achieved final specifications of 16.16 mW, 7.51 mm^2 area, 95.0% classification accuracy, and a throughput of 31.35K images per second. This performance is comparable with some other SNN implementations, such as ISSC'19, which has higher throughput but higher power and area [8]. To compare each design, an overall score metric normalized for the process node was used. The equation to calculate the model's score is shown in Equation 1.

$$\text{Score} = \frac{\text{Throughput} \times \text{Tech} \times \text{Accuracy}}{\text{Power} \times \text{Area}} \tag{1}$$

The scores were normalized and are shown in Table II. Our design has a 40.8% increase in this metric compared to ISSC'19. Other implementations, such as VLSI'17 and μBrain, which prioritize throughput or power, have higher scores in this metric than our implementation does. However, the accuracies of their implementation are 88% and 91.7%, respectively, which is much lower than the accuracy of our implementation (95%) [9], [10]. For applications that require a fair amount of accuracy (e.g., around 95%), our design strikes an optimal balance between power, area, throughput, and accuracy. Table II and Figure 7 detail the comparison between other neuromorphic processor implementations and the SNN developed in this project.

Overall, we found that the exponential decay logic in the LIF neuron had significant power and area overhead; area increased by around 80%, and power increased by around 30% over the IF implementation. The overhead for the LLIF implementation, which implemented constant linear decay each cycle, was much lower; power only increased by around 3%, and area only increased by around 8% in comparison to the IF implementation. The benefits in accuracy with the leak logic were marginal; when modeled in software, the accuracy of the models with leak logic only achieved around 0.5% classification accuracy over the simpler IF implementation.

TABLE II
Comparison with other neuromorphic processors for MNIST

	Power (mW)	Hardware Accuracy	Throughput (img/s)	Area (mm^2)	Tech (nm)	Score
This Work (IF)	13.71	95.0%	31.35K	7.51	130	1
ISSC'19 [8]	23.6	97.83%	100K	10.08	65	0.71
VLSI'17 [9]	87.0	88%	1.7M	1.31	40	13.96
μBrain [10]	0.073	91.7%	238	2.68	40	1.19

These results indicate that an implementation of linear leak logic is more feasible than exponential leak logic, if one chooses to implement leak logic at all. Additionally, one thing to note is that the implementation of exponential leak decay described in this paper was done using a leak factor that cannot synthesize as a single shift in each neuron; with tuning decay numbers such that the exponential leak synthesizes using simpler hardware, exponential leak logic may be more realistic. For low-power applications that do not require an extremely high degree of accuracy, an IF neuron architecture will likely suffice.

TABLE III
Power, area, and software accuracy of the different neuron architectures

Neuron Model	Power (mW)	Area mm^2	SW Accuracy
IF	13.71	7.51	97.37%
LIF	17.47	13.66	97.83%
LLIF	14.21	8.16	N/A

Another observation made in our work was that memory was an important design consideration for power consumption. As seen in Figure 9, the power breakdown of the IF design shows that 21.4 percent of the power usage is attributed to memory. This proportion was likely due to the number of values stored, as the weights and biases for each neuron take a lot of memory to store. Future work in optimizing power or area for an SNN design could include investigating optimizations for the implementation of memory.

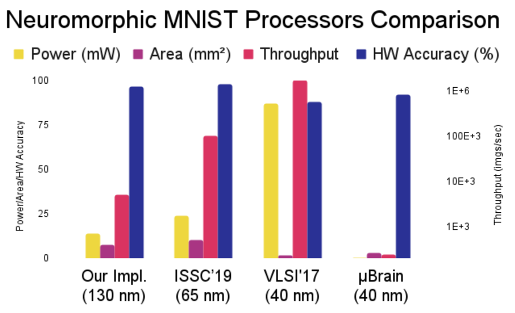

Fig. 7. Comparison of our model with existing neuromorphic processors trained on MNIST.

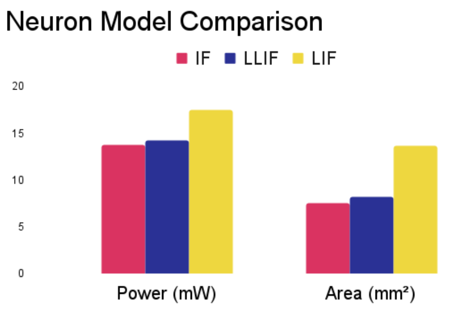

Fig. 8. Power and area comparisons for our three neuron models: Integrate-and-Fire, Leaky-Integrate-and-Fire, and Linear Leaky-Integrate-and-Fire.

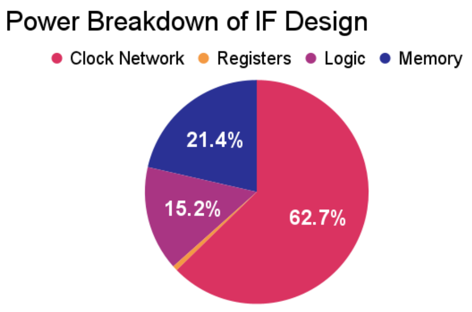

Fig. 9. Power breakdown of the Integrate-and-Fire neuron.

VI. Conclusion

SNNs offer area and power savings as opposed to traditional neural networks. This work explores the intricacies of implementing an SNN in hardware; it serves to analyze the tradeoffs among different choices of neuron architecture in terms of power, area, and accuracy. Ultimately, current published designs, such as those described in ISSC'19 and VLSI'17, have comparable performance to our 130-nm implementation, with some tradeoffs in terms of power, area, and throughput. Our work sought to analyze the differences in performance, power, and area of different neuron models, as well as create a low-power, high-accuracy hardware implementation of an SNN. The decrease in accuracy between the SNN with the IF neuron model and SNN with neurons with leak logic was small; however, the IF neuron implementation saw a significant decrease in power and area when compared to an exponential decay implementation. Ultimately, when implementing an SNN in hardware, an IF neuron may be the most feasible for applications that prioritize lower power, complexity, and area over accuracy. These findings have applications in edge AI, such as in surveillance systems or wearable devices. Future work includes further investigating the effects of different leak factors (i.e., ones that synthesize more cleanly to shifts in hardware), and exploring different ways of storing weights and biases in memory so as to further optimize the power of the design.

VII. Team Member Contributions

Every group member was a key contributor to this project. Most tasks were done as a full group; small tasks were done in a divide-and-conquer fashion. Every team member had a role to play in creating the final product. Below is a breakdown of just a few of the things that each team member did.

- Finn - SNN training and weight quantization, SRAM compilation and synthesis, debugging
- Amy - Implementation of different neuron models/layer architecture and corresponding testbenches, debugging
- Emily - Output module implementation, neuron/layer implementation, debugging, schematic drawing
- Kristen - Output module implementation, neuron/layer implementation, debugging
- Isaac - Neuron/layer implementation, debugging, experimenting with different neuron implementations

The above list is just a brief overview of the contributions of each member. Each group member was a driving force during the debugging process, testing, and report/milestone writing. Overall, everyone pulled their weight and saw the project to fruition.

References

[1] N. Rathi, I. Chakraborty, A. Kosta, A. Sengupta, A. Ankit, P. Panda, and K. Roy, “Exploring neuromorphic computing based on spiking neural networks: Algorithms to hardware,” *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–49, 2023.

[2] A. Javanshir, T. T. Nguyen, M. P. Mahmud, and A. Z. Kouzani, “Advancements in algorithms and neuromorphic hardware for spiking neural networks,” *Neural Computation*, vol. 34, no. 6, pp. 1289–1328, 2022.

[3] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, “Loihi: A neuromorphic manycore processor with on-chip learning,” *Ieee Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[4] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam *et al.*, “Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip,” *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[5] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, “A 4096-neuron 1m-synapse 3.8-pj/sop spiking neural network with on-chip stdp learning and sparse weights in 10-nm finfet cmos,” *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, 2018.

[6] Q. Wang, Y. Li, B. Shao, S. Dey, and P. Li, “Energy efficient parallel neuromorphic architectures with approximate arithmetic on fpga,” *Neurocomputing*, vol. 221, pp. 146–158, 2017.

[7] Z. Kuang, J. Wang, S. Yang, G. Yi, B. Deng, and X. Wei, "Digital implementation of the spiking neural network and its digit recognition," in *2019 Chinese Control And Decision Conference (CCDC)*. IEEE, 2019, pp. 3621–3625.

[8] J. Park, J. Lee, and D. Jeon, "A 65-nm neuromorphic image classification processor with energy-efficient training through direct spike-only feedback," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 108–119, 2020.

[9] F. N. Buhler, P. Brown, J. Li, T. Chen, Z. Zhang, and M. P. Flynn, "A 3.43tops/w 48.9pj/pixel 50.1nj/classification 512 analog neuron sparse coding neural network with on-chip learning and classification in 40nm cmos," in *2017 Symposium on VLSI Circuits*, 2017, pp. C30–C31.

[10] J. Stuijt, M. Sifalakis, A. Yousefzadeh, and F. Corradi, "μbrain: An event-driven and fully synthesizable architecture for spiking neural networks," *Frontiers in neuroscience*, vol. 15, p. 664208, 2021.

[11] J. K. Eshraghian, M. Ward, E. O. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.