A Mixed Signal 65nm CMOS Implementation of a Spiking Neural Network

by

Yangtian Yan

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2022

Author’s Declaration

Abstract

Spiking neural networks (SNNs) are an emerging class of biologically inspired Artificial Neural Networks implemented in machine learning and artificial intelligence. Current state-of-the-art small- and large-scale SNNs are mainly implemented as digital hardware with time-multiplexing techniques to achieve power efficiency. In this thesis, a 65 nm CMOS mixed signal asynchronous SNN implementation was designed and simulated. The proposed design reduces hardware and timing complexity over existing implementations and opens opportunities for further larger-scale implementations.

Acknowledgements

I appreciate my supervisor, Dr. Derek Wright, for taking on the research journey with me throughout my graduate studies and providing tremendous support in academics and life. The past two years have been challenging for us all, and his wit, resilience, diligence, and curiosity have inspired me since the first time I met him, and I look forward to continuing working with him in the near and far future.

I appreciate Prof. Manoj Sachdev and Prof. Lan Wei for teaching me the key and valuable knowledge in the courses throughout my undergraduate and graduate learning, as well as contributing their time and effort in reviewing my thesis.

I appreciate my team’s Ph.D. members, Hugo Stangherlin and Yugal Maheshwari, for providing teaching and research assistance during my graduate studies.

I appreciate my old and new friends who are from and are now situated everywhere in the world for sharing their living experiences with me, keeping my world vital and hopeful.

I appreciate my neighbours, Shirley, Joe, and Bai, for sharing fun times during the lockdowns. Bai is an American Eskimo, he does not need to read or author a research paper, and a part of me is envious of that.

I appreciate my family’s unconditional love, trust, and support. They brought a curious and unique mind into the world and have been nurturing me to explore and continuously become a better person.

Table of Contents

List of Figures

1 Introduction

1.1 Introduction

Since their debut, computers have efficiently assisted tasks in precise numerical calculations, data processing, and communications. Modern programs are constructed with instructions and run in finite states, a paradigm known as the “Universal Turing Machine”. These programs are operated on general-purpose hardware based on von Neumann architecture, where program instructions and data are stored in the memory. They are fetched and processed in Arithmetic Logic Units (ALU)s within the Central Processing Unit (CPU). This software and hardware architecture is efficient in executing various complex programs, so long as the states of the programs are well defined and can be executed in primarily sequential order.

Traditional computing architectures quickly achieved superhuman speed and precision in executing sophisticated sequential programs, but they struggle to solve problems requiring human-like perception, intuition, and cognition. These problems require decision-making in a complex environment with unspecified parameters and conditions. Neuromorphic computing, inspired by natural neurobiological intelligence, is an efficient and practical approach to these complex and ambiguous problems. Its applications span agriculture, science, education, finance, management, engineering, and art [1].

While the current research and applications of neuromorphic computing show promising results across industries, it is incredibly power-hungry and inefficient compared to biological intelligence. On average, the human brain consumes 20 W, whereas a recent machine learning algorithm that learned to solve a Rubik’s Cube consumed 2.8 GWh of energy [2]. The significant power consumed in training and decision-making impedes many applications of this new form of computing, leading to hardware and energy scarcity and related concerns about environmental impacts. High energy consumption per neuron prohibits low-energy applications, like battery-powered IoT devices. Local machines may possess insufficient processing power for security-sensitive information, requiring risky cloud connections and data transfers.

This research explores the levels of abstraction that shape neuromorphic computing in light of these limitations. It focuses on spiking neural networks (SNNs), which seek to mimic biological intelligence better than prevalent approaches. It describes a scalable mixed-signal custom integrated circuit (IC) that relies primarily on a standard CMOS digital workflow with minimal custom blocks and avoids exotic devices like RRAMs.

1.2 Thesis Organization

The remainder of this work is organized as follows: Chapter 2 provides background on artificial intelligence and pinpoints the present research in mixed-signal SNN hardware. Chapter 3 presents the theoretical and architectural considerations proposed to tackle some current limitations of prevalent approaches. Chapter 4 details the circuit design and simulation results. Finally, Chapter 5 discusses the results and enumerates potential improvements and subsequent research directions.

2 Background and Motivation

2.1 Artificial Intelligence and Machine Learning

Artificial Intelligence (AI), Machine Learning (ML), and neural networks are closely related subjects frequently found in this research area.

Artificial Intelligence "is the theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages." [3] Traditional computing architectures struggle to execute AI tasks since they were initially designed and optimized for fast and precise calculation of organized data rather than making human-like abstract decisions and analyses.

Machine Learning is a subset of AI in which the computer processes and analyzes data without explicit instructions to infer rules for classifications and decision making. Whereas AI can be an extensive set of manually programmed conditions, ML derives these conditions automatically and makes decisions based on these learning processes.

2.2 Neurons and Artificial Neural Networks

2.2.1 Biological Neurons

Neurons are the unit building blocks of any neural network in biological or electronic forms. Various types of neurons exist throughout the human body, but their physical structures generally consist of three major components: Dendrites, somas, and axons. Figure 2-1 shows the structure of a typical neuron.

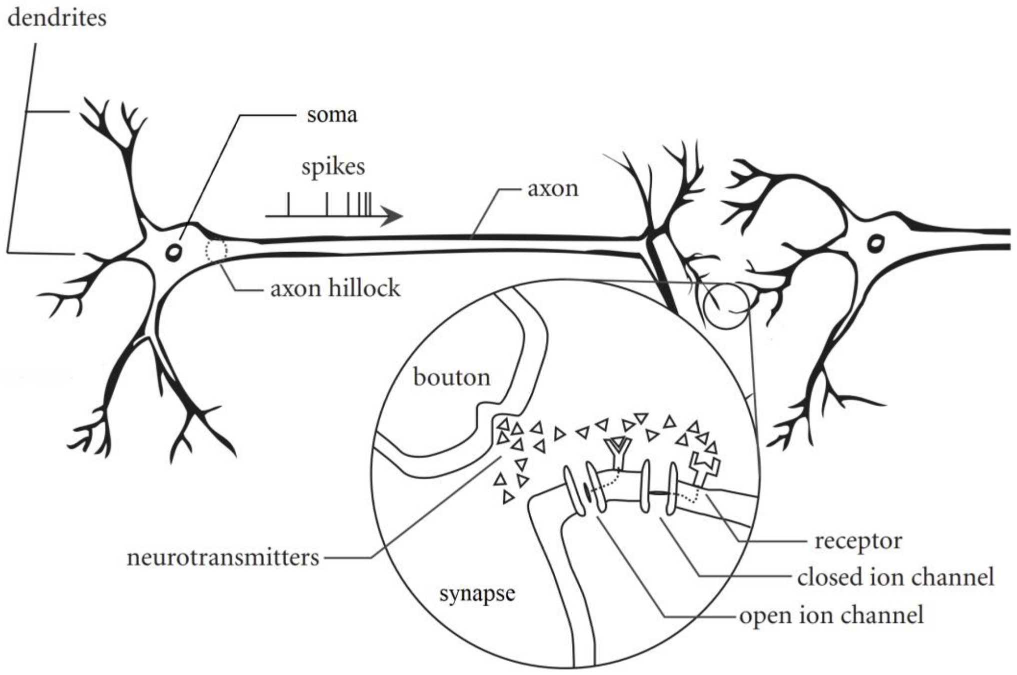


Figure 2-1 Biological Neuron Autonomy and Connection

At a cell's resting state, due to the differences between intracellular and extracellular ions, the potential within a cell membrane is lower than that outside of a cell membrane, resulting in a measurable resting potential of –70 mV.

When a spike arrives at the axon located at a synapse, the axon releases a signalling molecule into the synapse. The signal molecule then binds with a receptor of a gated ion channel on the dendrite of the neuron receiving the signal. Such molecules can be excitatory neurotransmitters, which enable the ion channel, or inhibitory blockers, which occupy the binding site, preventing other neurotransmitters from activating the channel. There are two types of gated ion channels at the dendrite. Upon receiving neurotransmitters, one allows positive Sodium (Na^+) ions into the cell. The other allows positive potassium (K^+) ions out of the cell. Notably, initially, the rate of sodium entering the cell exceeds potassium, resulting in a rise of membrane potential toward the positive range in the **depolarization** process. During the same time as depolarization, potassium channels also increase their passages of positive ions out of the cell. If the potassium transfer rate exceeds the sodium transfer rate, the membrane potential drops back toward resting potential, resulting in **repolarization**. If under enough external stimuli from the neurotransmitters, the sodium ion concentration increase to above a threshold of 15 – 20mV above the resting potential, a group of additional voltage-controlled sodium channels are opened, creating a rapid increase in membrane potential. The membrane potential spikes to + 20mV named **action potential**. After the brief peak of potential, the repolarization continues until the membrane potential reaches and drops below the resting potential to the –90mV range, resulting in **hyperpolarization**. After the spiking event, the cell uses continuously operating active pumps to restore the ion concentrations to the resting potential. The time it takes to restore the ion concentration to the resting level causes a refractory period in which the cell is not responsive and unable to spike from external stimuli. While repolarization below the threshold remains local and degrades over time, an action potential spike travels down the axon.

If observation is taken at the axon of a neuron, only the action potential spikes can reach the output. This behaviour can be modelled mathematically as a form of non-linear filter which blocks the input until a certain threshold is reached, then outputs a brief spike. The output spike frequency increases after continuous inputs, forming a spike train. The velocity of the spike within a neuron can vary from 0.5 – 2.0 m/s [4], significantly slower than electric currents.

2.2.2 Conventional Artificial Neurons and Artificial Neural Networks

2.2.2.1 Artificial Neurons

In a conventional artificial neural network (ANN), neurons can be mathematically represented by the model shown in Figure 2-2.

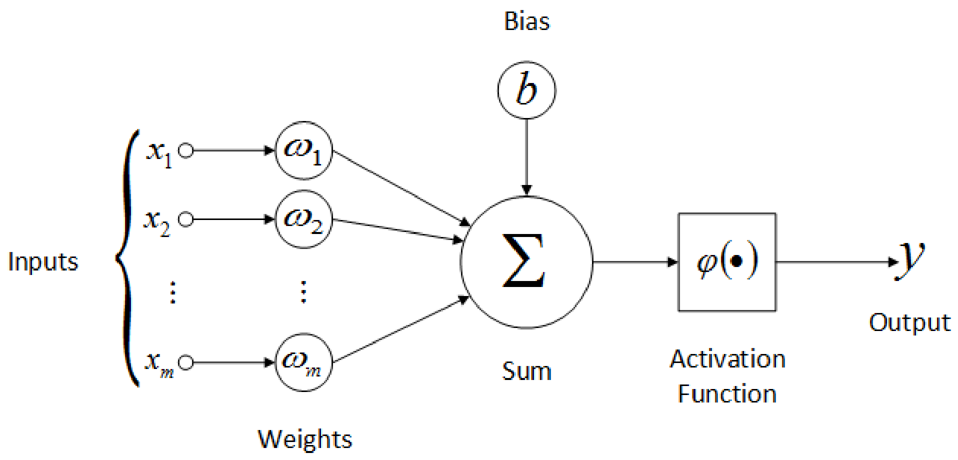


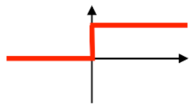
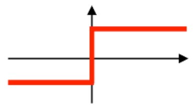
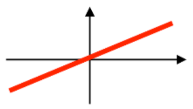
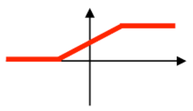
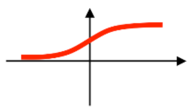
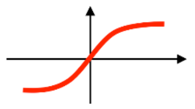
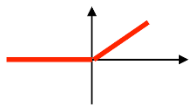
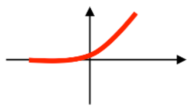
Figure 2-2 – A mathematical representation of an artificial neuron [5]

The inputs, x, represent the incoming signals from the external axons to the neuron's dendrites. Each input is multiplied by a weight, ω, determining its impact on the neuron's membrane potential. All input weight products are then fed into a summation function and a non-linear activation function. The activation function is non-linear, with numerous models of activation functions in table 1. Generally, they produce no or little output until the input reaches a threshold value, then significantly increased output above it. The output can be mathematically represented as in (1), where x is the input value, w is weight, and φ is the activation function.

$y = \varphi(\sum_{i=m}^{i=0} x_i\omega_i + b\ )$	(1)

The most commonly used activation functions are shown in Table 2-1. All but the linear function are non-linear, matching the biological neuron's non-linear response. It shows little or no output until the summed input and bias exceeds the threshold.

Table 2-1 - Neuron activation functions.

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus Copyright © Sebastian Raschka 2016 (http://sebastianraschka.com)	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

2.2.2.2 *Types of Network Connection*

There are numerous types of connections between the neurons in an ANN. The most simplistic is a feedforward network, where the signals only pass from one neuron layer to the next, unlike networks with a short-term memory effect where previous outputs influence subsequent outputs. Feedforward networks consist of input, hidden, and output layers, which can be fully or sparsely connected. It is considered a deep neural network (DNN) if hidden layers exist. Each neuron in a fully connected network connects to every neuron in the adjacent layers. On the other hand, only some of the neurons are connected between layers in sparsely connected networks. Figure 2-3 shows a simple three-layer, fully connected feedforward neural network [6].

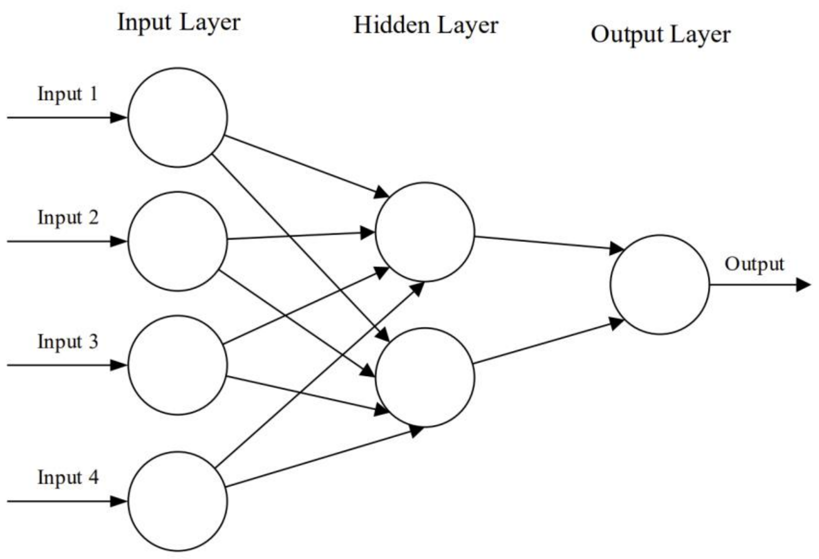


Figure 2-3 A three-layer fully connected feedforward neural network [6]

Feedforward networks with appropriately set weights are effective in classifying input data. However, they become resource intensive for applications such as visual recognition. For instance, Figure 2-4 shows images from the MNIST database of 28x28 pixel grayscale images of handwritten digits [7]. Effectively categorizing these images with a feedforward network requires 28x28 = 784 input neurons, 128 hidden layer neurons, and 10 output neurons. A fully-connected solution requires $784 \times 128 + 128 \times 10 = 101{,}632$ synapses.

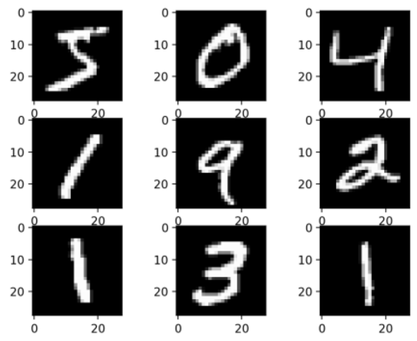


Figure 2-4 Examples of MNIST handwriting images

The number of required multiplications is proportional to the square of the image size, so processing large images becomes prohibitively expensive. Increased weight and input data precision require even more processing resources (e.g., floating-point weights or coloured images).

A specific type of neural network called a convolutional neural network (CNN) is an effective and efficient approach to using network resources. Instead of simultaneously processing all inputs, a smaller-sized kernel filter matrix is scanned across the image in the first layers of the network. Using a kernel significantly reduces the number of network connections and performs better feature recognition for a smaller slice of the image. To further reduce the amount of data processed, pooling is also used to down sample the images. Max pooling represents an area of pixels with the highest valued single pixel found within, while average pooling reduces an area of pixels into a single pixel with their average value. Figure 2-5 is an example of a CNN topology [8].

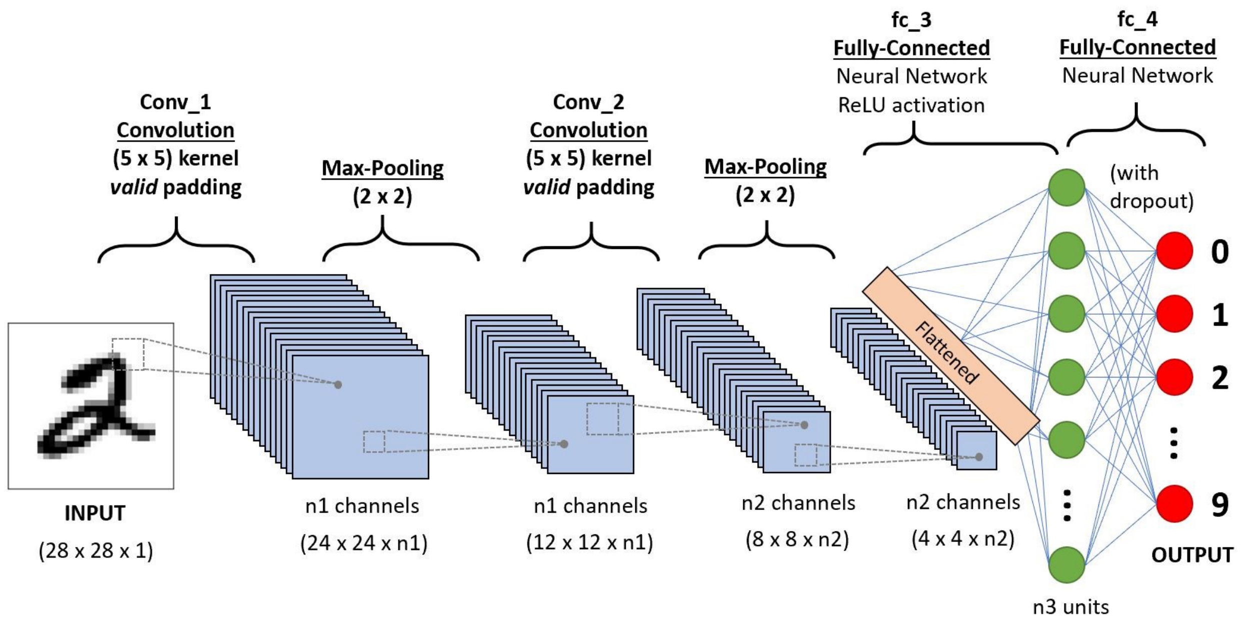


Figure 2-5 A CNN implementation for MNIST digit recognition. The network goes through two convolutions and two max pooling layers to reduce size before a fully connected network [9]

2.2.2.3 Training and Inference

There are two broad types of training for neural networks. In supervised learning, the input and its expected output are provided in the training dataset. The weights of a given ANN are optimized using The combination of backpropagation and gradient descent algorithms. Since the desired output is given in a supervised network, an error can be derived from the output, showing how much the network deviates from the ideal output. The previous layer's output-weight product influences the error function. Gradient descent is a method to adjust the values of the weights of the previous layer to minimize the loss function of the current layer. Backpropagation is the recursive traverse of the layers from output to input, as well as the generation of the loss functions. The network may be optimized and trained with one example at once (stochastic gradient descent), a whole dataset (batch gradient descent), or subsets of the dataset (mini-batch gradient descent).

In unsupervised learning, only input values are provided, and the network then categorizes input data into output classes. Due to the lack of an output feedback factor, the neurons reconfigure their weight and threshold based on current or previous network activities. One example is winner-take-all training, where the neuron receiving the most input signals remains active while others remain largely or entirely inactive.

Inference in an ANN is determining which output responds to a given input. The input signals propagate down the network. At each layer, the neurons multiply with the corresponding weights and sum together. In a synchronous network, each layer of neuron output is generated simultaneously, and the largest output is selected. In an asynchronous network, the neuron outputs may propagate through the network at different times, and a trade-off can be made to select the earliest active outputs or to wait for the highest output value.

2.2.2.4 Layers of Abstraction in Conventional Artificial Neural Networks

Conventional ANNs can be materialized in various application, software, and hardware forms. At the application level, the properties of the network are defined, such as its layer types and sizes, input and output types, and learning methods.

The software level bridges the application and hardware levels. The most well-known software frameworks for ANN implementation are PyTorch, TensorFlow, and Keras. These software frameworks provide APIs for quickly adapting a network design and can use hardware (GPU) acceleration.

At the hardware level, traditional Von-Neumann architectures are inefficient for neural network implementations because operations remain largely sequential, whereas neural network computations are mainly parallel. The limited number of arithmetic units and the memory bottleneck cause significant delays in fetching and calculating a network’s weight values and summations. Graphics processing units (GPUs) are designed to perform parallel computations, and neural networks can use their matrix multiplication capabilities to execute ANN weight calculations and summations efficiently. Due to rapid advances in GPUs, they have been used extensively to support software frameworks. Field programmable gate arrays (FPGAs) are also candidates for improved neural network computing. Their configurable processing cores allow for more flexible data routing and latency, reduced I/O bottlenecking, and better-optimized sensor interfaces [10]. Application-specific integrated circuits (ASICs) allow for even higher flexibility but have limited scalability and significant research and development cost. Consequently, they are not as widely used for hardware acceleration as GPUs and FPGAs but can allow for more biologically accurate neuron simulations.

2.3 Spiking Neural Networks (SNN)

2.3.1 Limitations of Conventional Artificial Neural Networks

Despite achieving great success in computational accuracy, the power efficiency of conventional ANN implementations falls short compared to their biological counterparts. Even with hardware optimized to execute vast quantities of arithmetic calculations quickly, they are power and hardware resource intensive. In nature, the ion molecules across the neuron membrane are mostly preserved and reused, whereas each clock cycle of a digital circuit dumps charge to ground, creating heat in the process. The matrix dimensions grow exponentially with the numerical precision of neuron parameters, such as weights and thresholds. The neuron inputs must sum large numbers of synapse outputs using multiple layers of adders. This many-input addition consumes time and area if implemented with tree adders or significant delay if implemented with multiplexed adders. In addition, conventional ANNs consume power regardless of their input levels because an arithmetic logic unit (ALU) does not consume less power when its inputs are low value. These factors limit the potential power savings of ANNs when the network is standing by for an event-driven input or experiencing low-level input.

2.3.2 The Emergence and Potential Advantages of SNNs

Spiking neural networks (SNN) emerged to overcome the limitations of a mathematically simulated conventional ANN. SNNs are a class of biologically inspired, neuromorphic neural networks. In an ANN, the output of a neuron is a numerically represented non-linear function. In an SNN, the neuron’s behaviour mimics the biological form of the neuron that generates individual pulses (described in chapter 2). The spiking output of an SNN neuron may be encoded with a voltage waveform, a firing rate, or a simple Boolean spike, which is the simplest form. Switching from numerical-based outputs to spiking brings advantages to the network. The

Boolean spike only requires a single-bit bus. Matrix multiplications to calculate the weighted synapse outputs are reduced to simply passing only the weight. The resting power consumption of the SNNs can also be significantly reduced. Another perk of SNNs is their fault tolerance. It has been proven that injecting random noise into a neuron's membrane potential during software simulation of an SNN improves the simulation results [11]. This fact opens the possibility of noise-tolerant mixed signal networks that benefit from circuit noise.

2.3.3 Spiking Neuron Models

2.3.3.1 Leaky Integrate-and-fire Neural Models

Leaky integrate-and-fire (LIF) neurons are widely used in SNNs. LIF neurons are biologically inspired models that use electrical component characteristics to mimic a neuron's spiking behaviour. The cell membrane separating the ions responsible for the membrane potential is modelled as a capacitor. The permeability of the cell membrane due to the passive ion channels causes the membrane potential to approach the resting voltage over time, which is represented by a resistor in parallel with the membrane capacitor. The LIF input is modelled as a current source, which charges the capacitor and dissipates through the resistor. As the voltage reaches the firing threshold, a voltage source spikes in parallel with the capacitor and resistor. While the ideal delta function generator produces an instantaneous spike with infinite magnitude and zero duration, a realistic model produces a narrow spike reaching the supply voltage briefly and then discharges the membrane potential to resting voltage. Figure 2-6 An analog implementation of a spiking neuron. Shows the schematic of an analog LIF neuron.

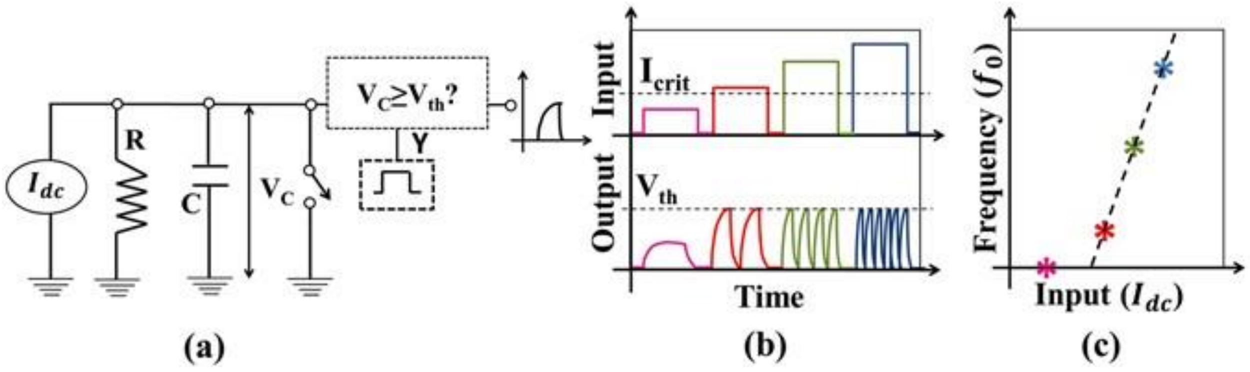


Figure 2-6 An analog implementation of a spiking neuron. (a) LIF neuron circuit model, (b) For input (Idc < Icrit), V(t) never exceeds Vth- hence neuron never spikes. However, for Idc ≥ Icrit, the neuron will fire when V(t) ≥ Vth and immediately reset i.e., V(t) = EL, (c) With higher input (e.g. Idc ≥ Icrit), firing rate or the frequency increases like a biological neuron while for low input

(Idc < Icrit), frequency is zero. The output frequency (fo) vs. input is the signature neuronal function to be mimicked artificially.[12]

2.3.4 SNN Coding and Connection Topologies

2.3.4.1 SNN coding methods

Due to the later emergence of the SNN architectures, most of the training and testing datasets for neural networks have been well adapted for conventional ANN applications. MNIST, or any image, for instance, being a grey scale image dataset, can be quickly normalized to a bit-encoded colour depth for conventional numerically represented ANNs. However, there requires a need to convert and feed existing data into a spiking neural network. There are four primary ways of coding an SNN: rate coding, time-to-first spike coding, Phase coding, and Burst coding. Their representations are displayed in Figure 2-7.

Rate coding normalizes the input signal intensity from none to an application/hardware-specific upper rate limit, with Poisson random intervals within the spike train signal.

Time-to-first-spike coding is more suitable for more real-time applications, such as tactile sensory input. This coding is purely temporal-based.

Phase coding assigns the input value into a binary spike representation with a specified period. The bits of a binary string are coded into each spike train period. Within each period, the individual bits of a binary string also weight differently, corresponding to the binary encoding.

Burst coding is a mixture of rate and temporal coding, where an input signal would generate a series of pulses, and the pulse number and frequency are correlated to the magnitude of the input signal.

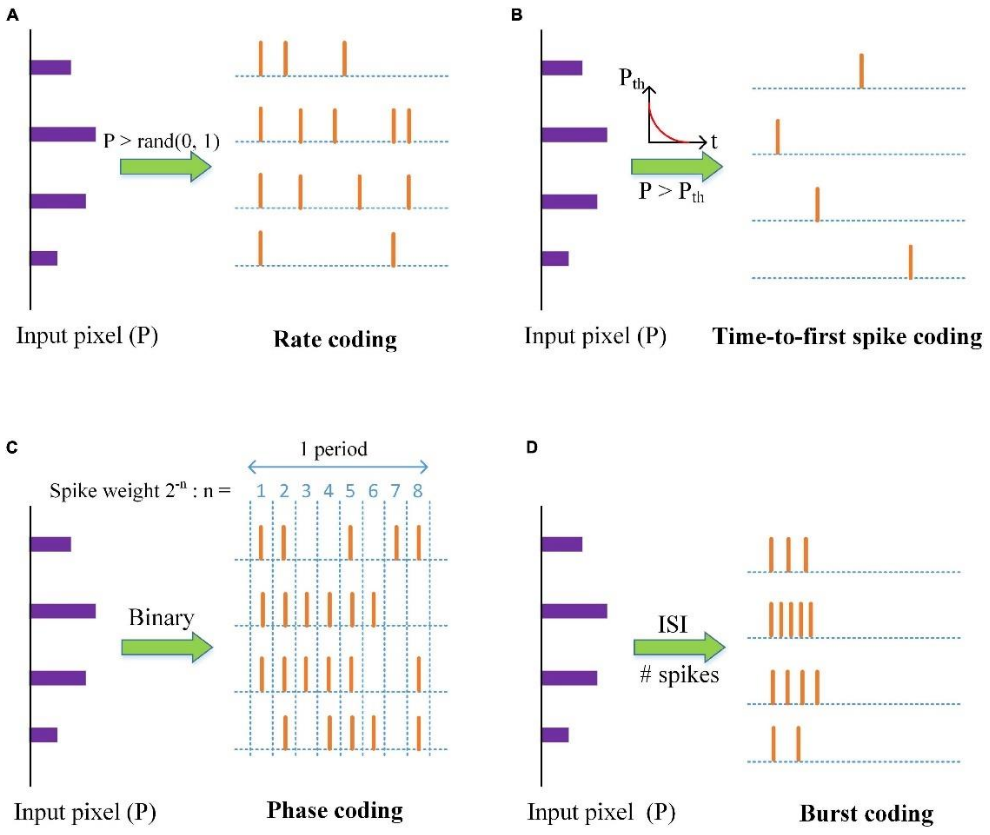


Figure 2-7 An illustration of neural coding schemes; (A) Rate coding, (B) Time-to-first spike coding, (C) Phase coding, and (D) Burst coding. P is the value of an input pixel. ISI is the inter-spike interval. [13]

2.3.4.2 SNN Hardware Configurations and Connectivity

2.3.4.2.1 SNN Neuron Hardware Implementations

SNN connections may vary significantly as the network scales up. On an application level, it's similar to conventional ANNs. Regardless of the initial input coding of the spiking signal, within the network, the synaptic connections are between the last neuron's spiking output bit to the weight bit of the current synapse. On the hardware level, there are numerous potential ways to realize and connect an SNN circuit. For example, a LIF neuron can be implemented digitally and through a mixed-signal method.

The neuron weight, membrane potential, and threshold can be numerically encoded for a digital neuron. An adder circuit can be used to sum all synaptic input weights. The sum is then stored in a register to be compared with a pre-programmed threshold. The two magnitudes can be compared with digital logic, with a clocked spike output. Once the spike is generated, the membrane potential register is reset to zero. A delay counter can be programmed for the neuron to ignore or reduce the weight of incoming spikes during the refractory period.

For a mixed signal or analog neuron, a portion or the entirety of a neuron can be implemented with analog circuitry. Due to the capacitive nature of a LIF's membrane potential mentioned above, the input spike, the charging, and discharging can be implemented through a current input at each dendrite, a charging capacitor, and a discharging resistor at the synapse. The weight of each synapse affects the amount of electric charge being applied to the capacitor per input spike. This can be realized in a combination of varying current magnitude and the length of the charging process. The threshold voltage can be stored both digitally in a register or as an analog voltage in a capacitor. In the former case, the membrane potential can be digitally encoded via an ADC, then compared digitally with the threshold. In the latter, an analog comparator can be used directly. The spiking output of a neuron can be generated digitally with a clocked output or through a comparator providing rail-to-rail output. The spiking and membrane potential resetting in a mixed signal neuron can be both synchronous, with a latched comparator regularly evaluating the membrane potential and a digital spike or an unlatched comparator triggering a spike. In the case of an asynchronous circuit, the spike signal can be used as feedback to signal the discharge of the membrane potential. The rise/fall rates and the duration of the discharge and spike can be influenced by the charging/discharging transistor sizes and the neuron membrane capacitance. The delay used in realizing the refractory period after neuron firing can be implemented through a synchronous digital counter, or an asynchronous delay block, for instance, an inverter chain.

2.3.4.2.2 SNN Hardware Connectivity Configurations

Departing from the traditional Von Neumann architecture, SNN applications utilize various approaches to utilize hardware space and power and minimize delay efficiently. When implementing a hardware SNN network, the nature of a circuit and biological neurons differ in many ways. One great advantage of a biological circuit is its spatial and energy efficiency, which allows neurons to calculate in parallel and fit a large number of neural connections into a small 3D space. One advantage of an electric circuit is the availability of high frequency, high precision clocks, allowing for an efficient task and time multiplexing.

Effective implementation of SNN should efficiently utilize the advantage of the precision of artificial hardware while implementing biologically inspired parallel computing and learning methods. Since modern hardware operates at a significantly higher clock rate than the spiking rate of an average neuron, a single processing unit within a chip can calculate a neuron's spiking and weights very quickly or the spiking of multiple neurons through time-multiplexing. The trade-off is between having how many neurons per processing unit. Each neuron in a spiking network has parameters such as threshold, refractory delay, and learning rate. Moreover, the synaptic weights occupy a greater area with a more extensive, densely connected network. An efficient processing core that utilizes time-multiplexing is placed with such parameters not too far or numerous within the chip to prevent memory bottlenecking. One additional downside of multiplexing is the clock cycles required for transporting data between storage and the processing unit, thus the additional power usage. Scaling up the network, the clusters of neural processing units form a core within a chip, and a network of chips may be populated on the same hardware platform.

For connecting between the cores within a chip or chips across a board in a more extensive scale system, computer network inspired router structures are commonly used due to their advantage in scalability [14]. AER, or Address-event Representation protocol is widely adapted, in which local spikes are encoded with both the neuron's source and destination addresses, as well as the event time of the spike onto a bus. The protocol is advantageous in its scalability within and across the cores, and chips in a system.

One additional parallel computing method is in-ram computing, allowing efficient data modification locally within each neuron's RAM storage. Many of these applications are realized through a crossbar connection to spread the spiking signals and novel memristor devices to store more than one bit of data per ram cell for the neuron parameters and states.

Due to the similarities between SNN and conventional ANN in terms of high-level architectures and connection topology, many existing weight training methods and values can be adapted into an SNN application when solving a similar problem. A comprehensive survey of deep learning with spiking neural networks [15] summarizes five main approaches to SNN training.

They apply to SNNs with various similarities to their ANN counterparts, from minimally modifying existing training methods and weightings to more biologically inspired learning methods.

1. Binarization of ANNs: Conventional DNNs are trained with binary activations but maintain their synchronous mode of information processing.
2. Conversion of ANNs: Conventional DNNs are trained with backpropagation, and all analog neurons are converted into spiking ones.
3. Training of constrained networks: Before conversion, conventional DNN training methods are used together with constraints that model the properties of the spiking neuron models.
4. Supervised learning with spikes: Directly training SNNs using variations of error backpropagation.
5. Local learning rules at synapses, such as STDP, are used for more biologically realistic training.

Method 1 applies to a minimally modified ANN, replacing each neuron's output level with a binary spike value. This type of network benefits from the reduced complexity of weight multiplication for synapse weight summing, but the circuit remains digital and synchronous. Method 2 and 3 are similar in converting an ANN into SNN; however, the difference is that in 2, the SNN's training uses the pre-converted ANN's weight entirely; whereas in 3, some spiking neuron parameters are modified into the ANN first before the training, the network is then fully converted into SNN for the inference. For example, the output of a neuron during training can be normalized into a firing possibility between 0 and 1. In method 4, the training is entirely done on an SNN implementation. The major obstacles to this approach are the non-linear, spiking delta function is impossible to differentiate and therefore poses a significant challenge for the traditional back propagation and gradient descent methods for supervised training. One walk-around is to use a differentiable function to approximate the spiking function for a neuron during training, such as a smoothed and continuous spike rather than a discontinuous delta. In method 5, it has biologically inspired, unsupervised, localized learnings, namely Spike-timing-dependent plasticity (STDP). The principle of STDP is that a synapse should become more significant if it receives a spike before the neuron fires. Therefore, a synapse's weight increases if the neuron fires after receiving input. If a synapse receives a spiking input after the neuron fires, its connection is deemed less significant and thus reduces its weight. STDP can be applied effectively to unsupervised problems and the clustering of input data into groups. Another optimization strategy is to increase the threshold of a neuron after its frequent firings. This encourages other neurons' specialization in their roles in classifying other features.

The wide ranges of methods above show different benefits. An ANN to SNN system conversion may show good accuracy and benefits from the convenience of existed ANN training methods and data. A stand-alone SNN implementation may be designed to take advantage of a more power-efficient hardware platform, adapting efficient temporal-based spiking methods.

2.4 State-of-the-art SNN System Implementations

For numerous general purposes, large-scale SNNs have been developed with different approaches in their implementations.

SpiNNaker [16] aimed to simulate brain activities in a massively-parallel multi-core computing system using up to 1,036,800 ARM9 cores and 7Tbytes of RAM in its hardware clusters. Each core on a PCB within the system is connected through a serial interface, controlled, and multiplexed by Xilinx FPGAs, as shown in Figure 2-9. Each ARM core has access to its own shared 13-bit precision synaptic memory array and time-multiplex spiking signals.

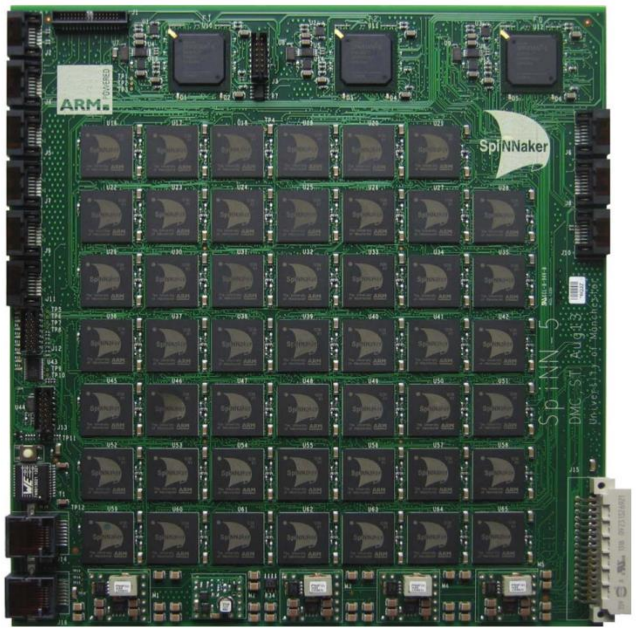


Figure 2-8 A 48-node SpiNNaker PCB. This circuit board incorporates 48 SpiNNaker packages (center) with a total of 864 ARM968 processor cores, three FPGAs (top) for high-speed inter-PCB communications through serial advanced technology attachment connectors (top left and right), with onboard power regulation (bottom) [17]

The PCBs within the system communicate through Ethernet protocol. One great advantage of using general-purpose ARM cores and ethernet protocols is the flexibility in application developments and the system's scalability. SpiNNaker supports the emulation of user-defined neuron and synapse models and can be easily adapted to various sizes. However, the digital emulation of neural models comes with a cost of lower power efficiency. The digital implementation also does not allow for a real-time simulation of a neural system.

TrueNorth [18] is a purely digital implementation aimed at minimizing energy usage and maximizing scalability. The system utilized a single-bit binary weight design for the synapses, which allows the axons and dendrites to be implemented in an efficient crossbar connection, allowing for efficient local neuron integration and fire operations with good parallelism. The neuron-synapse connections are represented in Figure 2-9.

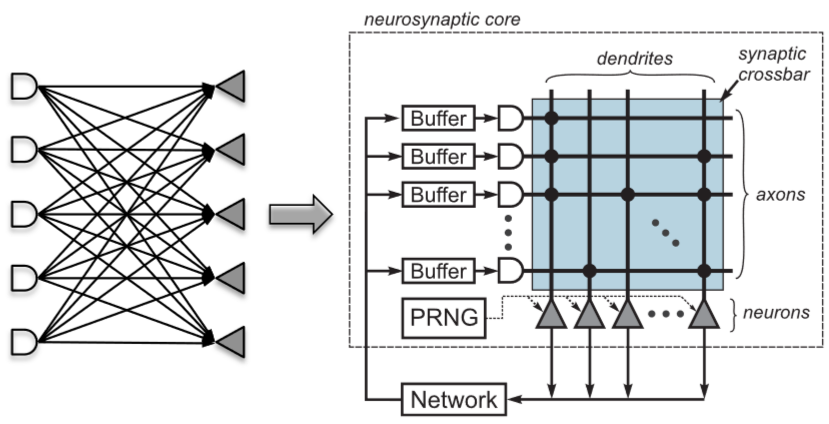


Figure 2-9 Bipartite graph of a neural network (left), with arbitrary connections between axons and neurons and the corresponding logical representation of a TrueNorth core (right). [18]

Like the application mentioned above, TrueNorth operates synchronously and uses time multiplexing to compute neuron states. The difference in connection is that SpiNNaker's chip wise communication is multi-cast, whereas TrueNorth is Unicast. The most significant advantage of TrueNorth is achieving the lowest energy usage out of the examples summarized here; however, it is limited to a single LIF neuron model, a low, single-bit synapse weight precision, and supports no on-chip learning.

Loihi [19], Intel's most recent addition to the SNN system, is similar to TrueNorth's time-multiplexed digital design and the unicasting network between cores on the chip. It, however, supports various configurable neuron models and on-chip learning capabilities. It's also manufactured on a more advanced process (14nm FinFet vs 28nm of the TrueNorth). As a result, although still more energy is used per spike than TrueNorth, it's significantly more power efficient than other digital and analog applications.

BrainScaleS [20], and its evolutionary successor, BrainScaleS – 2 [21], are mixed signal implementations of a SNN system. Within a processing core, they implement a mixed signal neuron model shown in Figure 2-10.

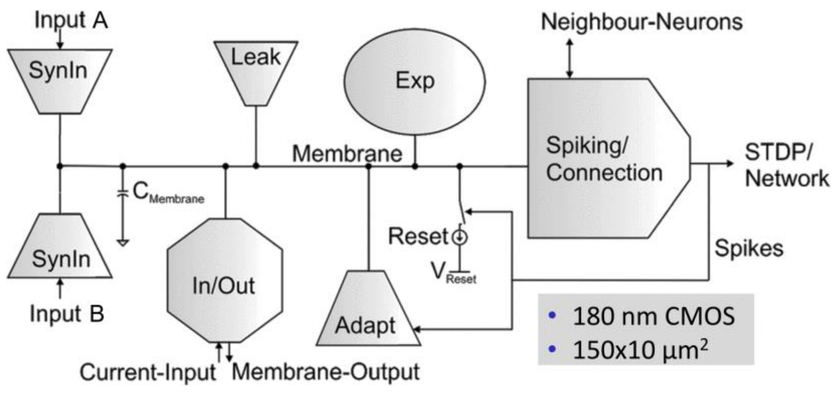


Figure 2-10 Schematic Diagram of BrainScaleS' adaptive exponential Integrate-and-fire neuron circuit [20]

In this system, the membrane potential is represented by the voltage potential stored in the MiM capacitor implemented physically above the circuitry of each neuron. The neuron supports on-chip STDP learning in the first generation, and a more complex, programmable learning method is available for the second generation. It is also notable that the system operates synchronously, with digital spike outputs and synapse inputs.

2.5 Current SNN Limitations and Motivations for Improvements

Despite being the most modern and biologically inspired approach to ANN applications, SNNs face numerous challenges. Despite operating at a much quicker rate than biological circuits, electrical circuits consume power at each clock cycle for synchronous applications and dump charge into the ground with each charge and discharge cycle of any capacitive components. The state-of-the-art SNNs have yet to exceed the classification accuracy of their non-spiking ANN counterparts, and the inference energy efficiency still has room for improvement [14].

All energy-efficient large-scale applications reviewed above used either digital or locally mixed-signal, but system-wise synchronous, digital network implementations. With the fault tolerance and asynchronous nature of biological neurons, there is potential for an efficient large-scale mixed-signal network with a greater degree of asynchronous, analogous device implementation.

3 Architectural Design of an SNN system

3.1 Top-down System Design

The design and experiment process followed the following route:

1. Identifying the goal of the system based on the existing state-of-the-art implementations
2. Selecting an appropriate application for the design for benchmarking the result with
3. System design from a top-down abstraction level
4. Utilize software simulation to create a baseline behaviour and performance expectations of an SNN
5. Identify, utilize, and design hardware architectures that efficiently execute parallel computing and realize a biologically inspired/plausible operation in the digital or mixed signal domain
6. Simulation and result analysis

3.1.1 Goal and design principles

Comparing the state-of-the-art implementations summarized in section 2.4 [14], TrueNorth achieved the lowest energy consumption per spike by utilizing a digital, time-multiplex system with minimal (binary) synaptic weight, at 27pJ/SOP (synaptic operation) with no on-chip learning capability. BrainScaleS utilizes a relatively low 5-bit synapse resolution and a mixed signal design, enabling STDP on-chip learning, and achieved a 174 pJ/SOP energy consumption. The learning capability is appealing; however, the BrainScaleS' increased power consumption may result from multiple factors. It used an older process (180 vs 28nm) and a dedicated capacitor for each neuron's membrane capacitance, which may have contributed to wasting additional charges. This work aimed to adapt the efficient digital and analog designs inspired by the systems above, such as simple crossbar connection of the synapses, a low-bit synaptic weight, as well as analog neurons.

3.1.2 Degree of Biological plausibility

The neuron model used for this work was determined as Leaky Integrate-and-fire (LIF), as it is a relatively simple model for SNNs. It was implemented in the majority of the state-of-the-art digital as well as mixed-signal large-scale applications, and its relative simplicity for analog and mixed-signal hardware realization.

3.1.3 Problem and Application Identification

As a commonly recognized application of testing and benchmarking, a digit classification of the MNIST handwriting database was selected as the target for developing software and hardware application. An MNIST-driven, fully connected SNN consists of three layers, an input layer with 784 (28^2) neurons, a hidden layer with 128 neurons, each connecting with 784 synapses, and an output layer of 10 neurons classifying the digits, each connected to 128 neurons. This presents a practical problem with a respectably sized SNN to present both the advantages and limitations of digital and analog implementations. The wide use of the database also allows for comparison with existed systems.

3.2 Software Simulation Packages

3.2.1 Nengo and Nengo-DL SNN simulation libraries

Nengo and Nengo DL [22] are simulation libraries built to implement SNN-based simulation, inference, and training. The advantage of the Nengo library was that it supported a wide range of simulations, from the membrane potential and spike generation of a single neuron to the simulation of an entire network of neurons, as well as weight training and inferencing of a spiking network. The result of a single neuron operation can be compared to and assist in analyzing the digital implementation in an FPGA system and the mixed signal implementation in an ASIC system. The simulation of a densely connected network can be compared to and assist the debugging of the HDL network simulation and the probed ASIC network simulation/chip level operation. In addition, Nengo and Nengo DL support the conversion of conventional ANN and CNN into SNN with its back propagation training with good classification accuracies. The weights generated from a trained network on Nengo can be converted, normalized, and implemented into the HDL and ASIC networks. Finally, being the highest abstract level of SNN programming, a Nengo network is easier to modify, compile, and train than the other lower-level applications in this work, allowing for a rapid early-stage simulation and network design as optimization. The simulation results of single neuron and network training of Nengo are featured in Chapter 4.

3.2.2 Identifying Opportunities for Hardware implementation

Nengo supports runtime based on both simulating through CPU and GPU acceleration, taking advantage of the parallel computation capabilities of the CUDA Cores on Nvidia GPUs. However, being a general-purpose library, the feature-rich Nengo operates on digitizing high-precision floating point calculations. Examining the works summarized in Chapter two, it was evident that a significantly lower precision and energy-efficient SNN can be implemented. Therefore, configurable hardware with excellent parallel computing became an appealing option. The next development step continued onto an FPGA platform and its HDL programming design.

3.3 HDL Architecture Design

3.3.1 System overview

The DE0-Nano-SoC kit was selected for this research, which consists of an Altera Cyclone V FPGA module and an ARM Cortex-A9 processor working in conjunction to offer convenient onboard IO support. The kit also supported a serial camera port to allow for the future development of real-time image classification tasks.

3.3.2 Neuron Programming

Three types of neurons are used in the HDL SNN: input, hidden layer, and output neurons. A universal LIF neuron model is created and programmed into different configurations with their input/output modes for serving each of the three types within the network. The parameters and input signals are presented in the table below. In HDL programming, all declared components in the code are physically allocated on an FPGA. The parameters determine the component’s size and some initial internal thresholds. At the same time, the input and output ports are open for connection to other components.

Table 3-1 HDL Neuron Parameter designations

Parameter Name	Type	Default Value	Description
N_synapse_in	Natural #	8	Number of Synapses (ports) at the input
N_synapse_bits	Natural #	4	
N_threshold_bits	Natural #	8	Threshold and membrane potential counter bit width
Threshold_IV	Natural #	128	Initial threshold default value

The neuron entity consists of two subcomponents, a Leaky Integrate-and-fire (LIF) neuron and a weighted summer. The LIF neuron adds the number from the summer upon every clock cycle and decrements the counter upon receiving a leaking control signal. The counter is compared to the threshold value, and a single cycle spike is generated once the counter exceeds the threshold and the counter resets to zero.

The weight summer receives all the synaptic weights from the synapses and executes the summation for each clock cycle. The weight summer was programmed in two different modes. The first design allows all synaptic weights to be summed within one clock cycle. This code allocates a full adder tree, which does not consume additional clock cycle numbers, but may limit the clock period due to the depth of the adder tree. This adder form also consumes significant logic units on the FPGA when the neurons are densely connected to many synapses. The second design iterates the synaptic weight input and incrementally adds all weights to the LIF input. The advantage of this configuration is that much hardware space is saved in a dense network since the hardware allocated is a multiplexer for sorting the input source, an adder, and a counter register to keep track of the current weight sum. However, the summer would require its clock signal to iterate through all the synapses. This method of time multiplexing the FPGA requires the LIF operation to wait for all synapse weights to be fully iterated, consuming additional time.

3.3.3 Synapse Programming

Due to the bulk of computation being executed in the neurons, the synapses are relatively simple in their structures. Each synapse allocates a physical connection between the output of a neuron in the previous layer to the input of the neuron in the next layer. Upon receiving the spike signal, the synapse enables the weight it carries, which is read from the shift register to the neuron input.

3.3.4 Identifying Opportunities for Custom Hardware Implementation

After synthesis, it was apparent that the summing logic for each synapse consumes the most hardware resource. In a digitized system, the synapse weight summing introduces significant fan-in to the digital logic, which can only be addressed by a deep tree adder structure, increasing hardware cost. The hardware cost may be compromised through time multiplexing by cycling through the connected synapses and adding them gradually versus all at once into the membrane potential register. This significantly increases the clock cycle between each spike for the entire system. Both methods cause waste and overhead in a clocked synchronous system, especially in a sparsely firing SNN circuit: In the formal method, a tree adder with mostly empty inputs still drains power with each clock input, and in the latter case, the entire system must idle in term of spiking while waiting for the synapse(s) with the most significant number of summations to cycle through all its inputs. The cyclic nature of the summing circuits would also potentially limit the time-competitive inference operations such as winner-take-all for the first neuron that fires in each layer.

In summary, a digital, synchronous circuit on the FPGA is limited in summing many numbers with hardware space and time efficiency. The next step of the work was taken to the ASIC level to explore a potential mixed-signal solution for an efficient neuron and synapse hardware solution.

3.4 ASIC Design

3.4.1 Hardware platform

The ASIC is designed on TSMC 65nm process with a 1V nominal power supply. It consists of a standard library of digital logic templates in the form of normal layout cells and custom layout cells for the analog and mixed-signal components. The design and simulation of the circuit were executed on the Cadence Virtuoso 6.1.8 software.

3.4.2 System Structure

The ASIC consists of two main subsystems: A digital circuit to intake and store the neuron and synapse parameters from the controlling FPGA and a mixed signal spiking neuron and synapse network.

The digital parameters, the controls, and the inputs of the ASIC system are taken from the FPGA's digital signal. The parameters are the initial weight carried in each synaptic connection within the network and the initial spiking threshold of each neuron. The network at the current stage of the work is fully connected by wire. However, disconnection can be achieved by setting a Zero-weight parameter to the corresponding synapse, so a spike from the previous layer does not affect the membrane potential of a given neuron. To populate the parameters, a scan chain consisting of D-flip flops was used to pass them in. The control signals consist of the shift signal to the scan-chain and reset signals for the synapse weights, membrane potential, and thresholds.

The input signal of the network is generated jointly by the ARM/FPGA board and the IC digital modules. The former converts the input signal, a pixel from the MNIST database, in this case, into a normalized intensity scale to a binary weight, which is then converted to a spiking period. The spiking period is fed to the SNN through the scan chain, and the input conversion on the ASIC uses a fixed clock signal and a digital counter to execute the spike for a fixed length after the determined period has passed.

Past the input layer on the SNN, the network operates primarily asynchronously. The clock-timed input spike is multiplied by weights on each synapse as it propagates further into the network. Each neuron operates independently, integrates the membrane potential from input spike-weight products as a form of charges charging up the membrane potential, actuates a spike and self-reset the membrane potential upon reaching the threshold voltage. The only synchronous aspect of a neuron is a globally timed leaking signal, dumping charges from all membranes at an interval configured externally.

A system block diagram of the mixed signal synapse and neuron structure is displayed in Figure 3-1.

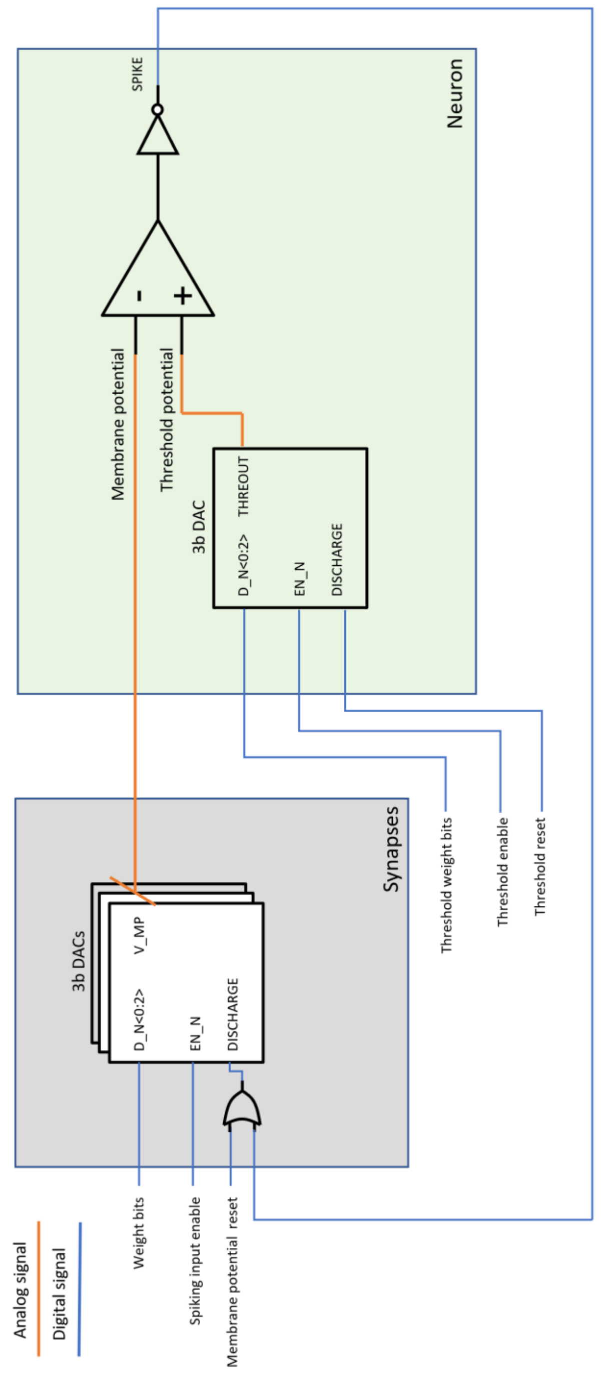


Figure 3-1 ASIC Synapse-Neuron system block diagram

3.4.3 Subcomponent design and schematics

The main philosophy of the ASIC design in this work is to achieve a minimized area and energy consumption with the given process and implement a localized, minimally time-multiplexed circuit for potential scalability.

3.4.3.1 Scan Chain Register

The scan chain registers were implemented through the D flip-flops provided in the TSMC 65nm standard library. The use of a scan chain in this application was motivated by DFT and the more straightforward hardware design compared to an SRAM array and its required controller to populate and route the signals. Upon system initialization, all SNN subcomponents' parameters are pipelined in from the FPGA interface synchronously.

3.4.3.2 Synapse DAC and Bus Capacitance

The synapse is located on each of the neuron's input points. It takes the input spiking signal and addresses it with an appropriate weight, then outputs to a summing membrane potential. The unique aspect of this circuit is that the membrane capacitance is implemented through the gate and shared bus capacitance of all the connected synapses. By doing so, the circuit avoids additional unnecessary charge dumping with dedicated capacitors and minimizes the area. The ideal LIF neuron model uses a constant current source to charge the membrane potential. However, in a CMOS capacitor charging circuit, the charge is exponential and slowly approaches the supply voltage due to the VGS decreasing on the charging PMOS' and thus its conductivity. The non-linearity of the charging curve of the membrane potential can be compensated with an equally normalized threshold voltage and limiting the operating voltage of the membrane potential below the taper point of a significantly slowed charging.

The weight multiplication of the input spike signal requires a DAC, as the weight is digitally encoded. The conventional DACs utilized in IC applications consist of a voltage ladder to produce an accurate voltage reference and an amplifier to supply current to the load until the output voltage is reached. The common implementations for small and low-power DACs in VLSI are mostly C-2C – opamp configurations. However, such a design is unsuitable for the current system for several reasons: Firstly, despite achieving a smaller area than the R-2R design on an

IC level, the minimal area of MiM capacitors is still significant. Secondly, the opamp used in the conventional DAC often uses dozens of transistors. An amplifier's hardware usage and complexity are suitable for a time-multiplexing circuit but too expensive to implement to each synapse. Thirdly, conventional DACs with an amplifier output current to reach a specific voltage level. The application in the current system requires a scale of charge being injected onto a shared bus. Therefore, a different type of DAC implementation is required to suffice the minimal area and charge output requirements.

The proposed DAC design allows a 3-bit binary input signal to scale the amount of charge dumped from VDD to the shared bus capacitance. The input of the DAC consists of an inverted enabling signal, **EN_N**, used as the spiking input; a three-bit inverted binary signal, **DN<0:2>,** to encode the weight as a **DISCHARGE** signal to dump bus charge to the ground. The schematics of the DAC are shown in Figure 3-2.

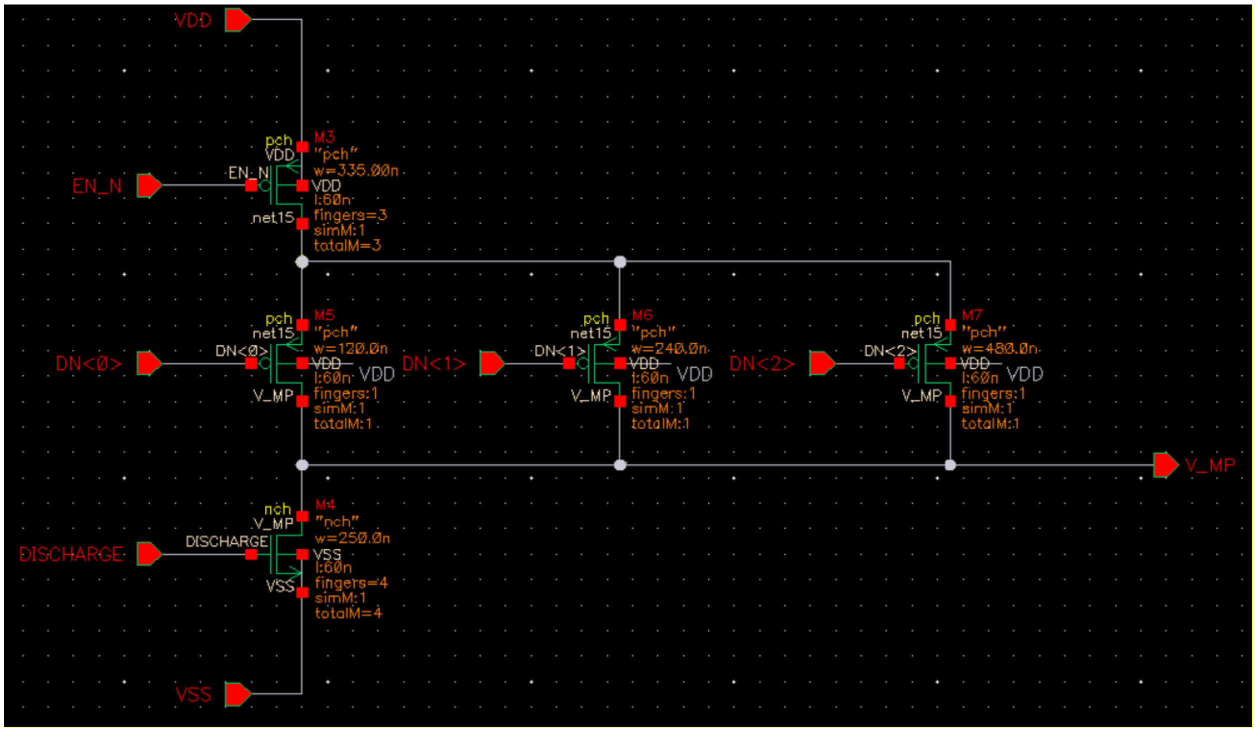


Figure 3-2 Initial 3-bit DAC Schematics

To take advantage of the minimally placed bus capacitance used for the membrane potential storage, a layout was created to analyze the parasitic capacitance of each DAC. The layout was created in similar dimensions to the TSMC 65nm standard cells to allow for easier routing for future work, as shown in Figure 3-3.

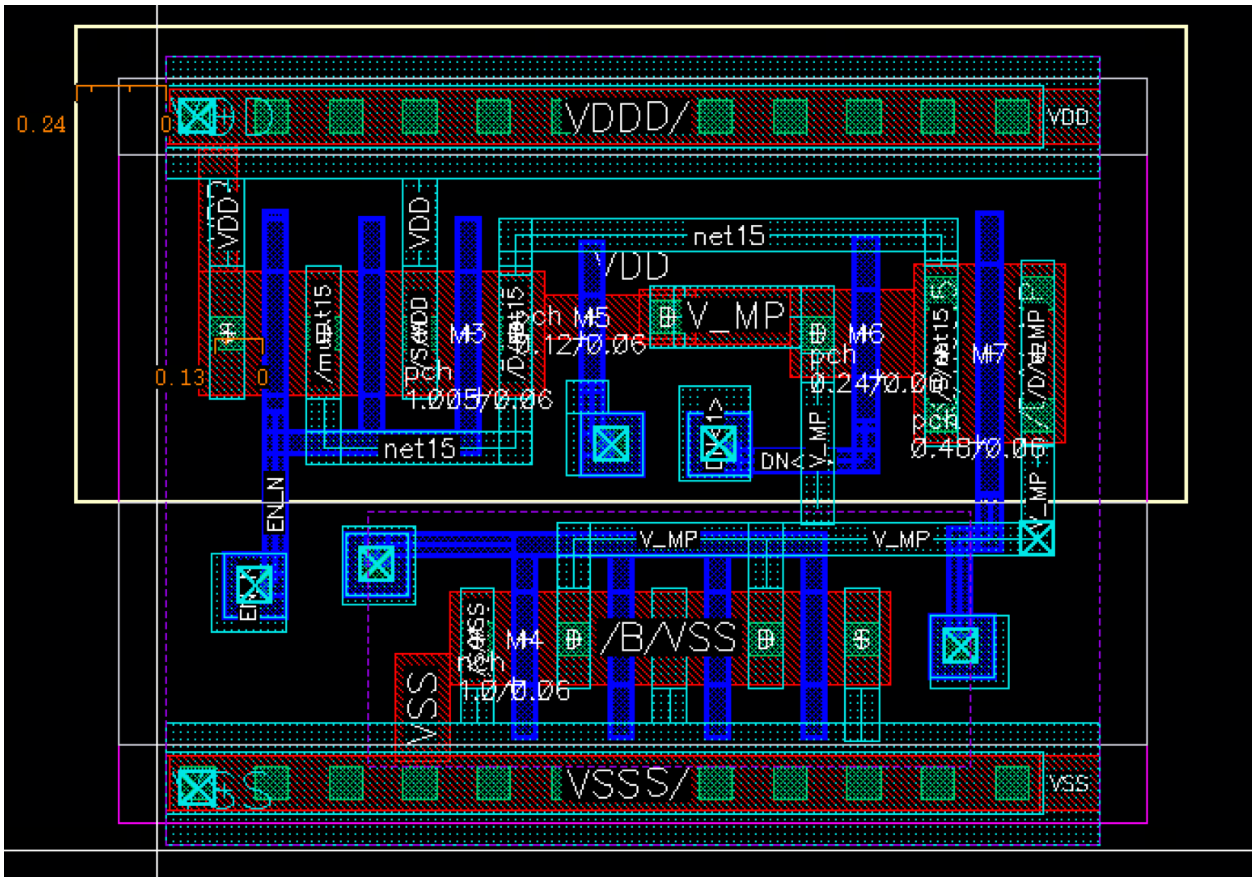


Figure 3-3 Initial DAC layout

After the initial simulation, a revision was introduced to the design, splitting the top large enable PMOS into three individual PMOS' with identical width with their corresponding weight, further improving the linearity of the current output, as shown in Figure 3-4.

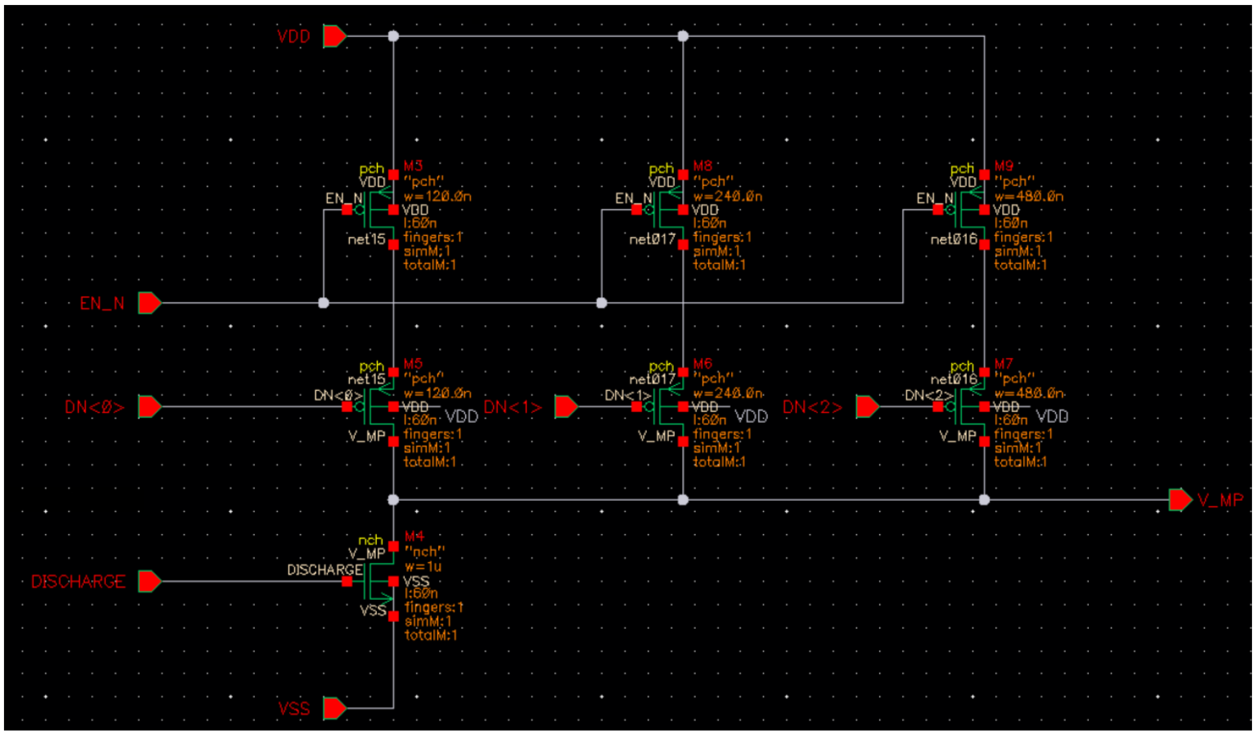

VDD
EN_N
DN<0>
DN<1>
DN<2>
DISCHARGE
VSS
V_MP
M3 "pch" w=120.0n l:60n fingers:1 simM:1 totalM:1
M8 "pch" w=240.0n l:60n fingers:1 simM:1 totalM:1
M9 "pch" w=480.0n l:60n fingers:1 simM:1 totalM:1
M5 "pch" w=120.0n l:60n fingers:1 simM:1 totalM:1
M6 "pch" w=240.0n l:60n fingers:1 simM:1 totalM:1
M7 "pch" w=480.0n l:60n fingers:1 simM:1 totalM:1
M4 "nch" w=1u l:60n fingers:1 simM:1 totalM:1
net15
net017
net016

Figure 3-4 Improved 3-bit DAC Schematics

Except for the input to the first layer of input neurons, all the synapse spike inputs are driven asynchronously.

3.4.3.3 Neuron

The Neuron consists of a DAC to set the threshold, a comparator for initiating the spike, and a feedback signal loop to reset the voltage. The schematic of the neuron is shown in Figure 3-5.

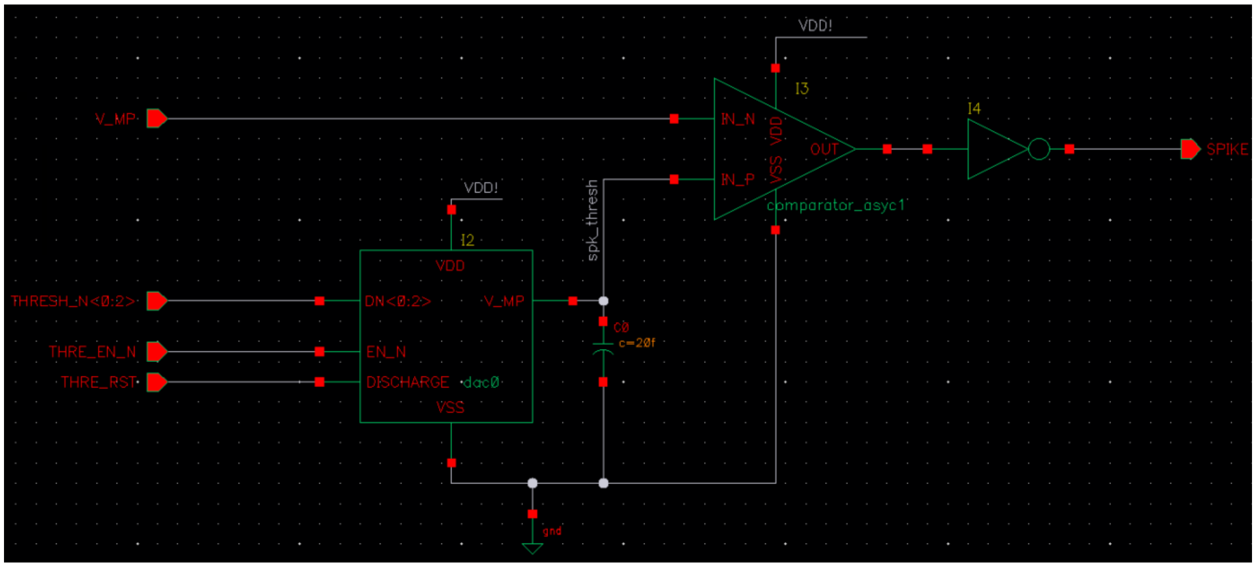


Figure 3-5 ASIC Neuron schematics consist of a 3-bit DAC, a 20f F capacitor for threshold voltage, a comparator, and an output inverter buffer.

3.4.3.3.1 DAC

The DAC in the neuron is identical to the synapse above. One noticeable difference is its 3-bit input is replaced by an externally determined 3-bit value from the scan chain. Its enable and reset signals are externally provided. The reset and enable signals are synchronous but operate much slower than the input neuron spiking signals. They refresh the threshold voltage, which slowly decreases over time due to leaks. Since the DAC in the neuron outputs directly to the input of the comparator, it requires a dedicated capacitor to convert its current into voltage. Due to the small number of neurons relative to the synapses, a dedicated MiM capacitor is used in the layers directly above or below the neuron module to provide adequate capacitance with minimal area consumption.

3.4.3.3.2 Comparator

The comparator compares the membrane potential voltage with the threshold voltage with a resting output at 0V. It generates a pulse at VDD until the membrane potential is completely discharged back to 0V. Due to the low bit-precision of the DAC, the comparator did not require a precise input offset, so long the offset is consistent throughout the operating voltage of the membrane potential and the threshold. However, due to the need to create a spiking output, it needed to operate rail-to-rail. Lastly, the frequency required for the comparator is loose due to the lack of AC signal being presented to the inputs; additionally, the ramp-up of input voltage for the membrane potential is often slow for a sparsely firing SNN.

The most simplistic inverter pair comparator is minimal in size but does not provide enough gain to achieve a quick rail-to-rail voltage swing required for the spike. Adding additional stages to the comparator would provide sufficient gain but required biased current sources throughout the circuit, undesirably constantly draining current and thus power.

A latched comparator was the next option under consideration. A StrongARM latch comparator, shown in Figure 3-6, was chosen for the design as it provides two main advantages: it does not consume static power and produces rail-to-rail output.[23]

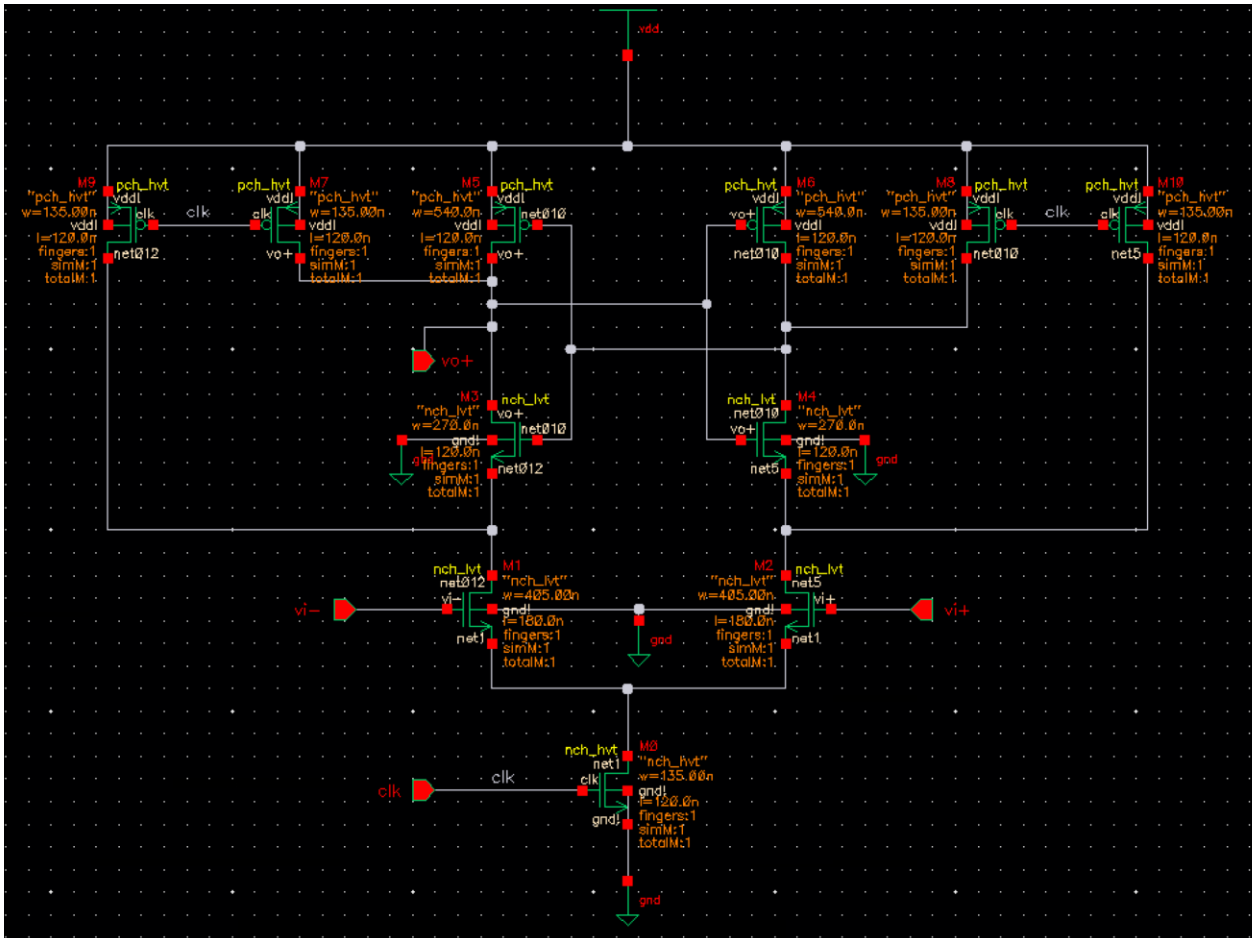


Figure 3-6 Schematic diagram of the StrongARM latched comparator

The downside of a latched comparator in this application is that the output evaluation requires a clocked input. This produces two main disadvantages to the circuit. First, routing a clock signal throughout the neurons costs additional power. Second, the clock signal must be toggled quick enough if the circuit operates asynchronously. If the evaluation is not triggered during a rise of membrane potential due to the incoming spike(s), the comparator effectively suffers greatly in input offset.

Upon further investigation, a more simplistic design was proposed, a non-latched comparator, as seen in Figure 3-7.

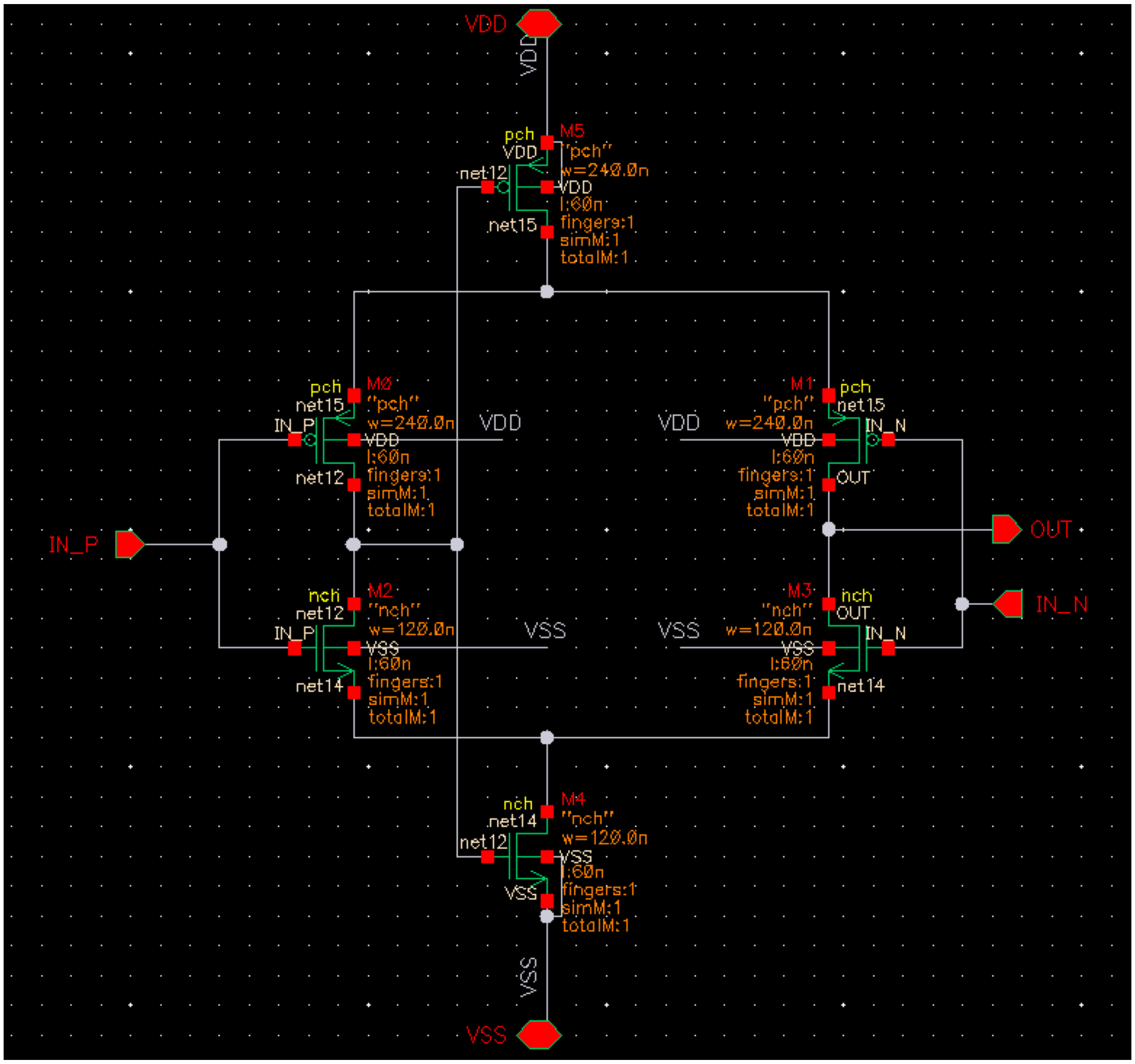


Figure 3-7 Schematic Design of the "pseudo latched" comparator

This design of comparator does require a clock signal; however, it does require the input signals to pre-set such that IN_P (positive) signal is above IN_N (negative) and the output “pre-latch” to negative. The comparator only toggles as IN_N exceeds IN_P, so the output swings to high. This behaviour is desirable for the SNN application, where the neuron output is resting at 0V for most of the time and only briefly spikes.

3.5 Identifying Opportunities for further Hardware design and improvements

3.5.1 Floorplan and system integration

The most imminent next step in ASIC development for this application is the construction of a densely connected SNN resembling the same network simulated in both Nengo and HDL implementations. The development process should provide vital information such as the complete picture of the synapse bus capacitance for each neuron, the potential for compensation of the synaptic weights, thresholds, and even the need for additional, dedicated on-silicon capacitors. Another direction of exploration is the potential hindrance of scaling up the analog network. As each synapse grows in connection number, the spiking neuron’s fan-out increases and requires additional layers of buffer CMOS, creating delay and extra hardware cost to the circuit.

3.5.2 Inhibitory Weights

The simulation of Nengo-bio neurons in chapter four shows the crucial role of inhibitory weights and synapses in the inference accuracy of a neuron cluster output. In software simulation, inhibitory synapses carry negative weight. In contrast, in real life, inhibitory synapses can cause a decrease in membrane potential by opening leaky ion channels on the membrane or inhibiting neighbouring dendrites from reducing the effect of incoming excitatory signals. Both forms of inhibition can be implemented in the design. For a decreasing membrane potential, a weighted NMOS section of the DAC discharges the membrane potential proportionally to the negative weights available to the neuron. For inhibition of other synapses on the same neuron, a non-zero, positive bias voltage may be adjusted and applied to the gates of the weighted charging PMOS’ on the DAC for a specific duration.

3.5.3 Localized STDP Learning

The current design does not provide on-chip learning. One advantage of SNN is the ease of biologically resembling local STDP learning. It is worth exploring the area and energy efficiency of implementing an energy and area-efficient STDP learning circuitry on a non-time-multiplexing circuit. The core principle of STDP is that a synapse's weight is increased if a pre-synaptic spike occurs before the post-synaptic spike. The synapse has a positive influence on generating the spike and thus has its weight increased.

Similarly, if the pre-synaptic spike occurs after the post-synaptic spike, the synapse has a reduced influence on generating the neuron's spike, thus decreasing its weight. To implement STDP locally on each neuron, a digital, asynchronous circuitry can be edge triggered by pre and post synaptic spikes and compare the order to increment or decrement the synaptic weight. Another learning method to optimize the network's overall efficiency is to adjust the threshold. Once a neuron frequently fires, its firing threshold can be increased. Once a neuron's threshold increases, it fires less, promoting the activity of other less firing neurons in the network by creating easier signal paths.

4 Simulation and Analysis

4.1 Nengo Software Simulations

4.1.1 Single Neuron Simulation

Nengo simulates and allows for probing of a single neuron's operation in LIF mode. One advantage of Nengo as a software simulation platform is its ability to both simulate ensembles of neurons and allow the probing of the firing and membrane potential of individual neurons within a network. A single LIF's firing with a sinewave increasing input weight is shown in Figure 4-1.

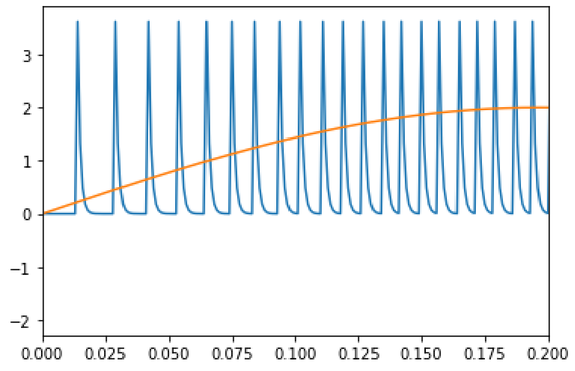


Figure 4-1.a Weight and Membrane Potential

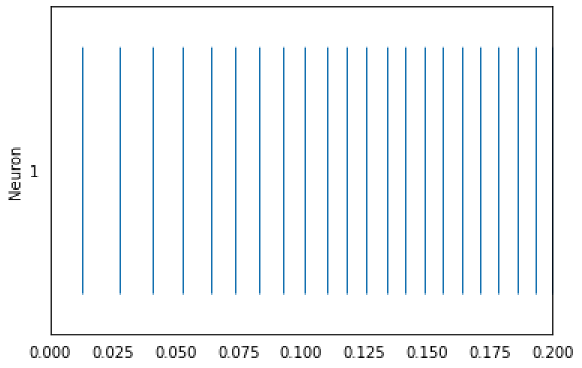


Figure 4-1.b Isolated Neuron Spikes

Figure 4-1 Nengo Single LIF Neuron Simulation

However, adapting Nengo simulations to the application in this work had its cavities. For instance, the high abstraction of the programming did not allow setting the physical voltage of the membrane potential. In addition, it is notable that Nengo encodes spikes more toward a rate-based method than a temporal or event-based one. Two parameters tune Nengo's rate-based encoding neurons: The intercept, the threshold for the input current at which the neuron starts firing, and the Maximum firing rate, for which the firing rate of the neuron reaches as the input current continues to rise. A typical neuron tuning curve with a constant intercept and varying maximum firing rate is presented in Figure 4-2.

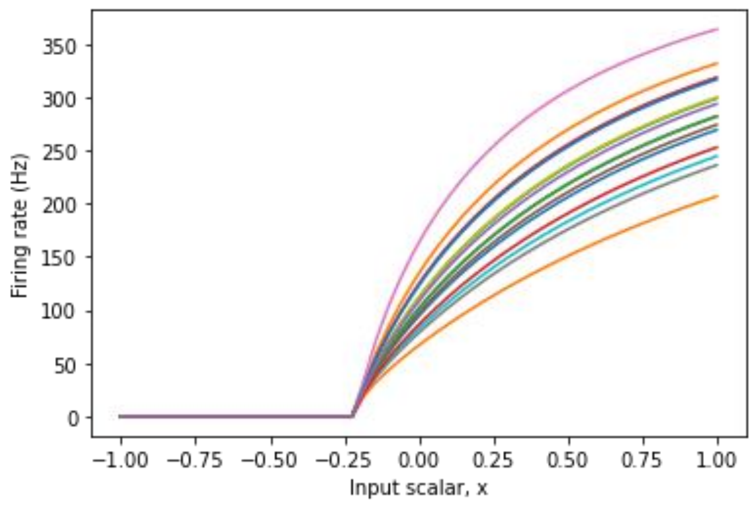


Figure 4-2 Nengo's Rate-based Neuron Tuning Curve [24]

Despite the difference in the SNN encoding, Nengo's neuron simulation was valuable in setting the foundation of the Network. Fundamentally both rate and temporal encoding could realize an effective network simulation. Although a non-standard practice, Nengo's neurons can be converted to threshold-based tuning[25] if necessary in future works.

4.1.2 HDL Synthesis and Simulations

The HDL components were implemented in VHDL language in Altera Quartus 20.0 software. Upon synthetization and simulation, two significant findings surfaced:

When the synapse module was synthesized as a single clock cycle summer, the hardware allocation became inaccessibly large for a three-layer, fully connected network. A fully connected MNIST network consists of a 28 by 28 (784) neuron input layer, a 128-neuron hidden layer, and a 10-neuron output layer. The required full-adder modules in a full-adder tree configuration exceed the capacity of the DE-0 FPGA. This showed the necessity of time multiplexing in synapse summing in digital logic, as well as an opportunity for an ASIC implementation to efficiently add a large number of digits in parallel.

Another finding was a limitation in the precision of comparing the membrane potential to the threshold voltage in a synchronous implementation in which time-slices the incoming synapse weight sums. Consider a situation where the membrane potential was approaching but not reaching the threshold voltage upon the arrival of the next summed synaptic input. Both a small and a large magnitude of addition causes the same spiking effect on the neuron. This may lead to decreased precision of synapse inputs if the time slice is too imprecise and hinder pre-post synaptic spike order-dependent learning methods, such as STDP.

However, the high configurability and parallel signal processing/handling properties of the FPGA showed promising capabilities in interfacing with the ASIC signals, including timed input conversion and storing, accessing, modifying, and sending control parameters to the ASIC board.

4.2 ASIC Simulations

4.2.1 DAC Simulation

The 3-bit DAC is used in two applications, charging the membrane potential and the threshold voltage. A simulation of the DAC charging a capacitor with incrementally increasing weight is shown in Figure 4-3 and a zoomed-in Figure 4-4.

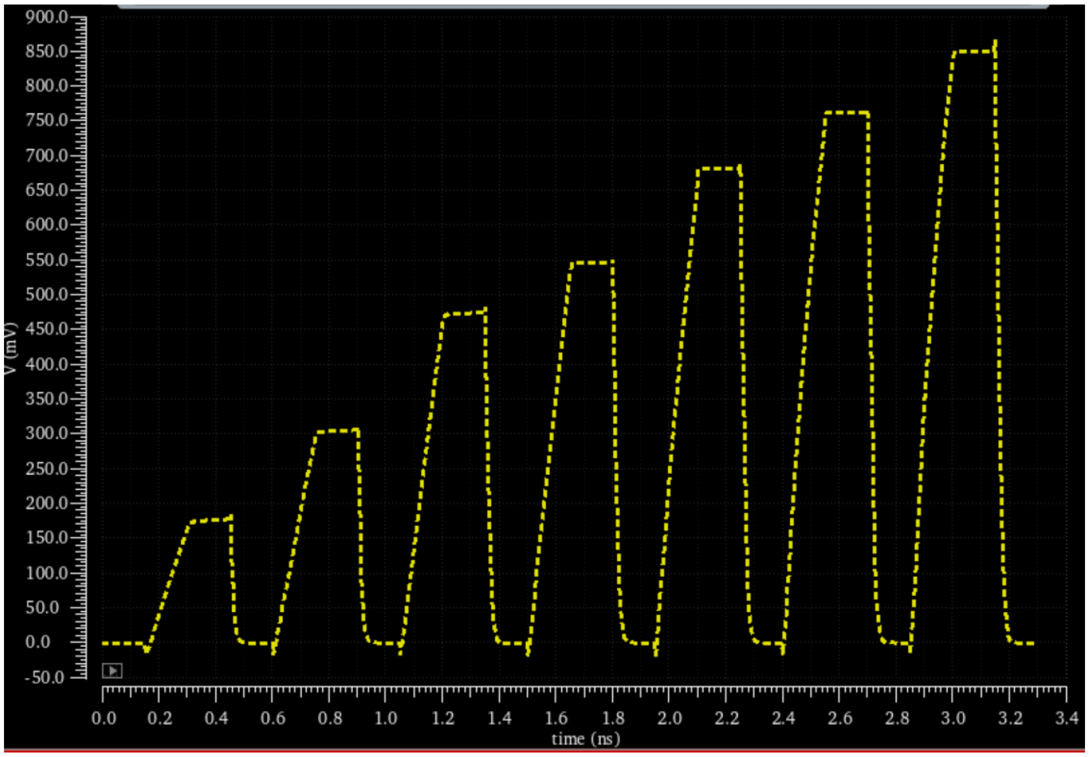


Figure 4-3 DAC incremental weight voltage simulation

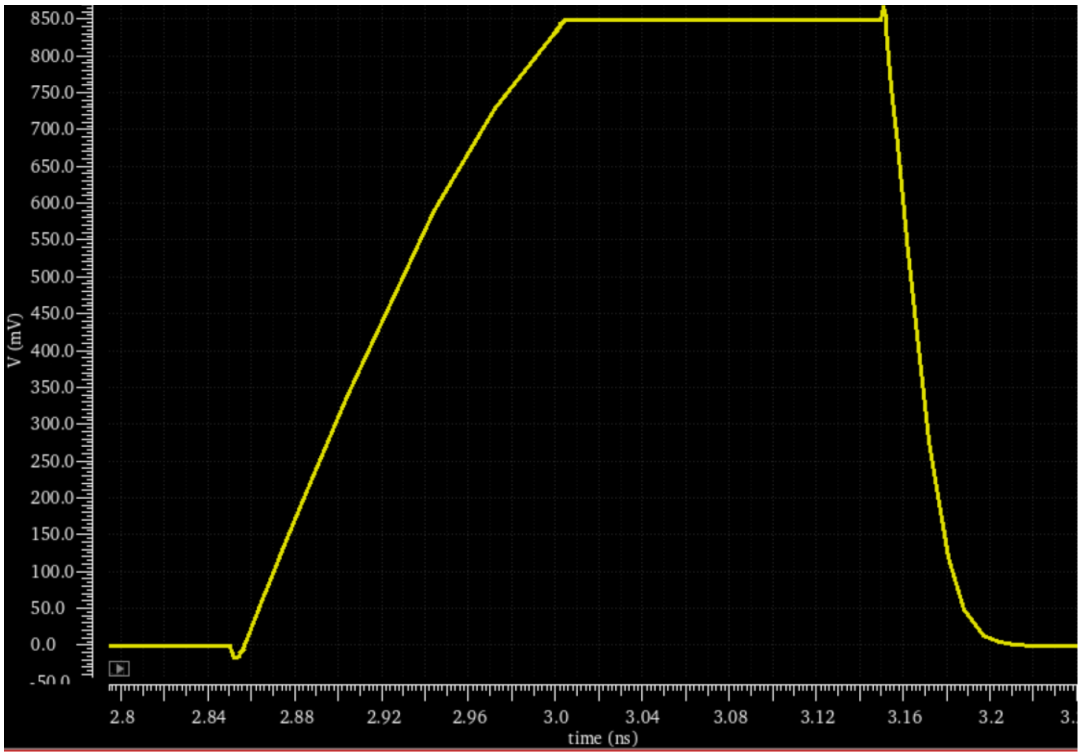


Figure 4-4 DAC Bus Capacitance Charging Curve

The DAC was supplied with 1V for power and signal inputs for the simulation setup. The 3-bit weight input was incremented from value 1 to 7. Each conversion's rise and reset time was set to 150ps, with a 150ps hold period. The DAC contains 1f F output parasitic capacitance, and the DAC was connected to a simulated bus capacitance of 20f F.

The operation principle of this DAC is different from the common DACs such as C-2C and R-2R. In the common C-2C or R-2R designs, a reference voltage is generated by switching the capacitor or resistor circuit. The reference voltage is then fed into a comparator amplifier with the feed back voltage from the output. The DAC in this design operates by scaling the current supplied to a charging capacitor. The output voltage depends heavily on the charging time and the capacitance. Both are variables in the design. One significant advantage of this DAC design is the almost instantaneous setup time and no overshoot after conversion since the charging is dependent entirely on the timing of gate voltage into the PMOS. The simulation above showed an extreme case with minimal charge time, which provided a generous margin in the minimal timing of the spike.

In the 3-bit configuration, the simulation shows the substantial monolithic property of the DAC. However, there are two notable behaviours:

Firstly, due to the non-linearity voltage of a capacitor charged by a constant voltage, the charging slows down exponentially as the membrane potential increases. This causes the gradual slowdown of charging and thus a non-linearity in the DAC operation. The non-linearity in the DAC's operation should not pose a significant negative impact because, at any given membrane potential, the DAC's output current scales linearity with the weight provided when the operation region of the PMOS' remains the same.

Secondly, The DAC output does not support the full rail-to-rail range well because as the membrane potential approaches 1V of the supply, the cascading PMOS shifts out of the saturation region, slowing output. This effect compounds the previous slowed charging of the capacitive load of the membrane potential bus.

These two non-ideal properties of the DAC suggested a limited range of the membrane potential and the threshold voltage in the neuron with which the membrane potential is compared to preserve the overall voltage linearity.

4.2.2 Neuron Simulation

The neuron consists of a comparator amplifier and a DAC to set the threshold voltage. The same DAC used in the synapse module was utilized, except for a dedicated MiM capacitor module instead of sharing the bus capacitance for the synapse; for the comparator, two designs were proposed and designed, a synchronous, latched comparator and an asynchronous, non-latched comparator.

4.2.2.1 Latched Comparator Simulation

The transient simulation of the StrongARM latched comparator is shown in Figure 4-5**.** The comparator was set to drive a 1f F load capacitor while evaluating a 50mV offset input. During low clk input, the comparator resets output to a constant high. Upon receiving a positive clk, the comparator evaluates the input and generates output accordingly.

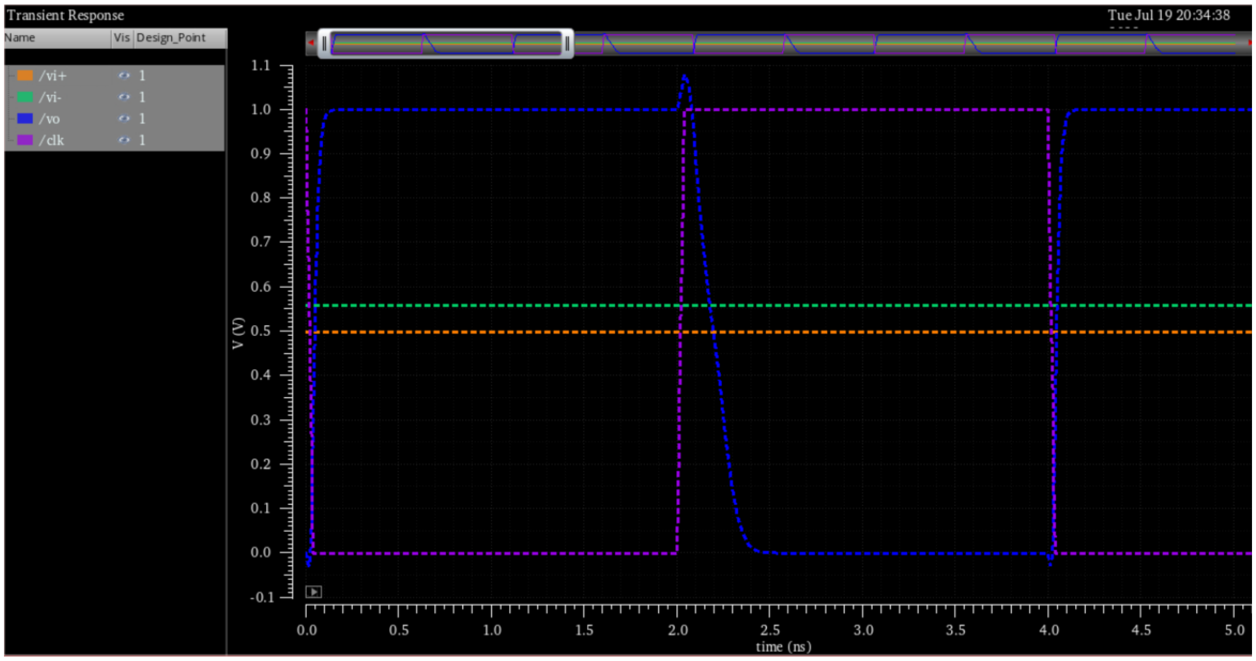


Figure 4-5 Latched Comparator Transient Simulation

One limitation of a latched comparator in an asynchronous SNN is that during the latch reset period, the comparator could not evaluate any incoming spike signals that could increase the membrane potential above the threshold and delay the spike. This effect is like the issue faced by the digital synchronous HDL neuron. Shortening the latch clock frequency mitigated this effect; however, the reset and evaluation period could not be shortened indefinitely. A minimum of 150ps reset and 300ps evaluation period was required to sufficiently reset the latch and provide rail-to-rail voltage output for the spike. It is also worth noting that the evaluation period of the latched comparator also determines the spike width of the network.

4.2.2.2 Non-Latched Comparator Simulation

The non-lathed comparator design did not rely on a locked signal to pre-charge the comparator for evaluation. Instead, it relies on the signal at the negative input to be sufficiently lower than the positive terminal to pre-set the output to low, at which it continuously evaluates the inputs. This behaviour was suitable for the SNN application because the output of a LIF neuron is typically low and generates a brief sparse spike which then resets the membrane voltage to zero. Figure 4-6 shows the non-latched comparator's transient response to a ramp input.

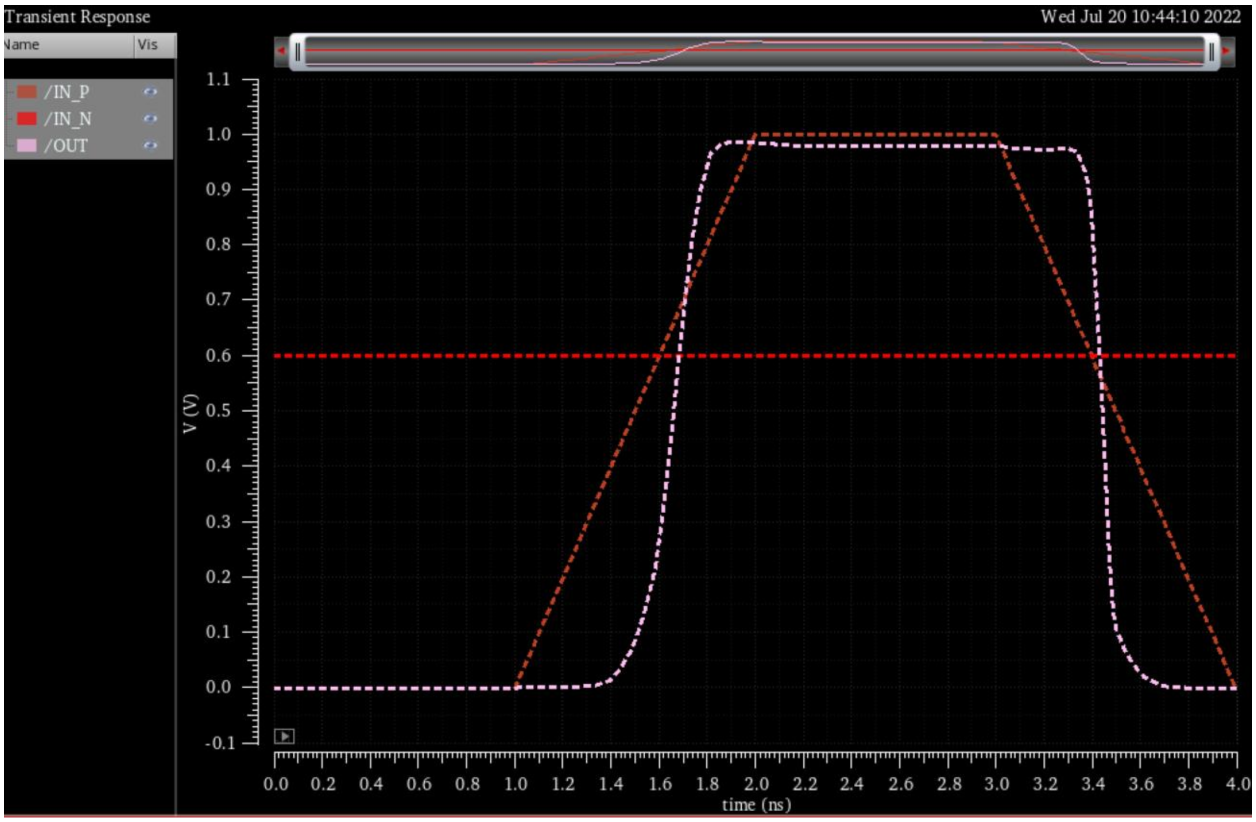


Figure 4-6 non-latched comparator transient response to ramp input

Without a clocked reset period, the non-latched continuously compares the membrane potential at the positive input with the threshold voltage at the negative input. Apart from reduced routing and energy consumption from the clock than the latched comparator, it is also more responsive toward an asynchronous increase of the membrane potential, thus improving spiking accuracy.

The non-latched comparator also had a more straightforward design which used fewer transistors than the StrongARM design; however, two notable trade-offs were observed. Firstly, the rise time is higher than the previous latched design. Fortunately, the spike width does not affect clock timing performance in an asynchronous network. The increased spike period results in a longer charging time in the subsequent layers' synapses. This increased charging duration can be compensated by reducing the weights of the synapses. In addition, since the spike input is an inverted signal at the synapse, an inverter can be used to both invert the spike signal and increase the slew rate. A more impactful issue of the non-latching comparator was its rail-to-rail operation property only held when the threshold voltage was within 0.3 to 0.7 volts, with an even further slowed slew rate at the ends of the voltage range. This is due to the comparators' reduced driving force without a dedicated latching mechanism and reduces the effective range of the threshold parameter of the neuron. Further improvement to the comparator design is required to increase its input operation range and allow for a higher threshold voltage range.

4.2.3 Integrated System timing simulation

To evaluate the feasibility of the asynchronous network operation, a circuit was constructed to simulate the hidden and output layers of a three-layer fully connected network used to classify MNIST digits. For the said MNIST network, the hidden layer contained 128 neurons connected to 784 synapses, and the output layer contained 10 neurons connected to 128 synapses. One of the outputs and hidden layer neurons was simulated to simplify the simulation. The hidden layer was continuously charged by 5 (out of 784) input synapses firing at half full-scale weight and 150ps input periods. The goal of the simulation was to examine and evaluate how spikes propagate in the multi-layered, sparsely firing network. The simulation result is shown in Figure 4-7. The orange trace shows the spike threshold generated from the neurons' DAC; The purple and the pink traces show the membrane potential of the hidden and the output layers; The green and the blue traces show the spiking signals of the hidden and the output layers.

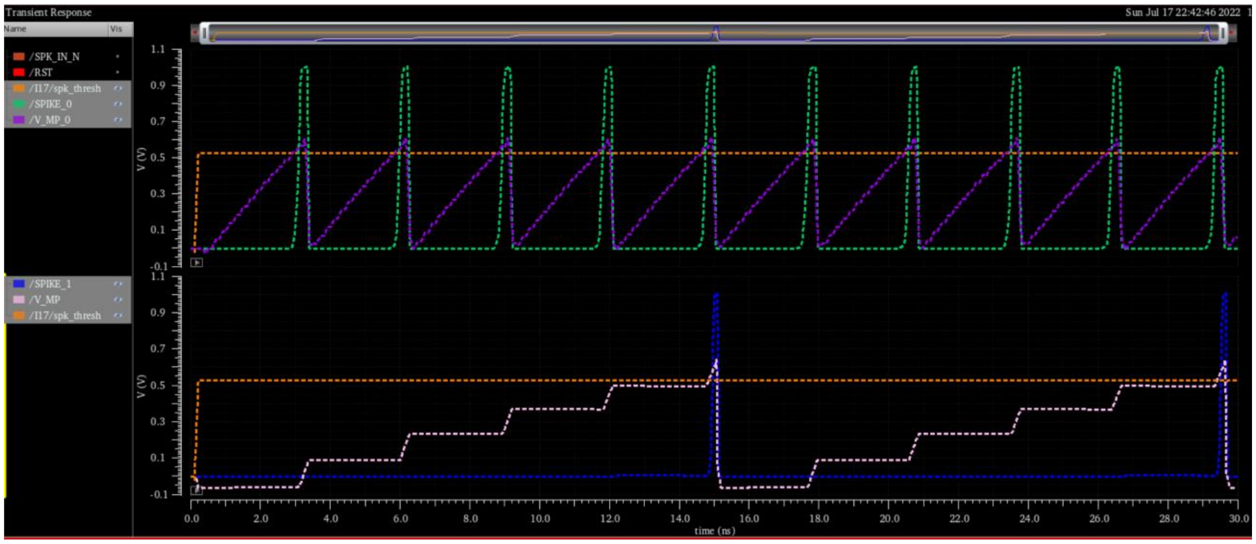


Figure 4-7 Two-Layer integrate-and-fire neuron Simulation

The simulation showed an effective operation of a three-layer asynchronous network with two layers containing integrate-and-firing neurons.

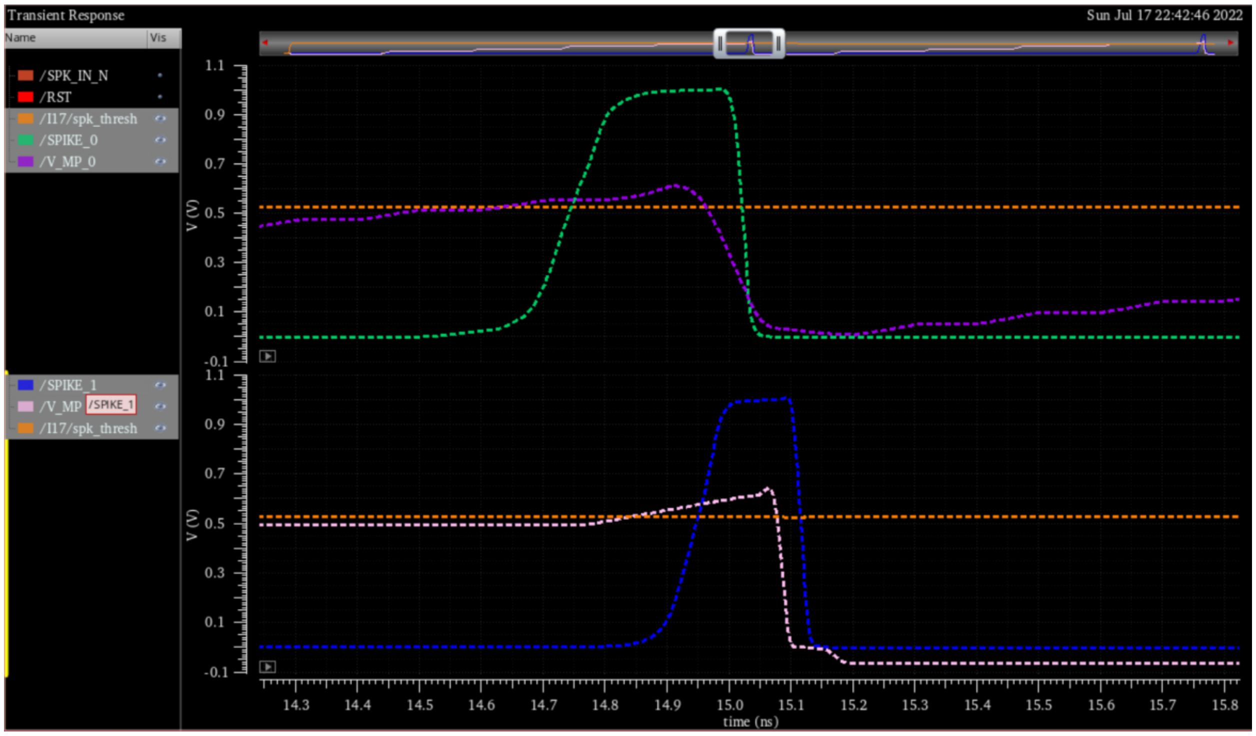


Figure 4-8 Zoomed-in voltage curve of a single spike

The rising slew rate of the spike is marginally higher on the output neuron than on the hidden layer neuron. This is likely due to the higher rise rate of the membrane potential on the lower capacitance synapses on the output layer (which consists of 128 synapses' shared capacitance as opposed to the 784 synapses of a hidden layer neuron). However, a more significant difference is that the spike width of the hidden layer neuron was almost double the output neuron. This is due to the large fan-out of the spike feedback signal used to reset the membrane potential. The inverter buffer at the hidden layer neuron's spiking output was not providing as much current to each synapse's discharge NMOS to drive the membrane potential down to zero quickly, which slowed the comparator reading a lowered input and thus leaving the spike output on for longer. This revealed a need to address the fan-out of resetting signal in a network with many synapse connections per neuron. The fan-out can be mitigated by using layers of inverter buffer. While the delay of the inverter buffers does not pose a significant issue in an asynchronous circuit, they occupy an area and create an additional difficulty in scaling up the layout in the floorplan.

5 Conclusion and Recommendations

5.1 Conclusion

The software suite used for simulation, Nengo, Nengo DL, and Nengo Bio, was proven to be an effective tool for both adapting a widely used MNIST ANN model into an SNN simulation, executing training, achieving good image classification accuracy, as well as generating helpful network parameters such as weights and thresholds for the network.

The HDL SNN was initially designed to be a digital realization of the software SNN model but was faced with a tremendous challenge in digitally adding many digits for each synapse in a densely connected network. The finding raised the need for a mixed signal ASIC synapse and neuron implementation.

The mixed signal ASIC neuron and synapse model was designed and simulated, with the novel design of representing membrane potential by charging the parasitic capacitance of wires connecting synapses to a neuron, drastically saving area and charge used per spike. The network operated asynchronously and showed a multi-layer spiking neural network operation.

5.2 Recommendations For Future Work

5.2.1 Software Future Work

With the ASIC neuron and synapse design developed, the software model can be tuned to more closely resemble them. With a more aligned neuron and synapse behaviour simulated in the software, it can provide the hardware with more accurate weight and threshold values during off-chip training, as well as re-evaluating the efficiency of a new hardware layout before committing effort to produce the chip floorplan.

5.2.2 FPGA Future Work

With the implementation of the mixed signal ASIC network, the primary purpose of the FPGA becomes supplying appropriate input, parameter setups, and output readout from the ASIC. In addition, a camera module interface can be developed on the FPGA to enable real-time object classification for future projects.

5.2.3 ASIC Future Work

The next step for the ASIC design is to upscale the synapse and neuron models into a network equipped with shift registers for input values and neuron parameters to be interfaced with the FPGA. The network should then take inputs and parameters from the Nengo network simulation to evaluate its effectiveness. Another feature that can be implemented into the ASIC design is local, unsupervised STDP training hardware.

References

[1] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, “State-of-the-art in artificial neural network applications: A survey,” Heliyon, vol. 4, no. 11, p. e00938, Nov. 2018, doi: 10.1016/j.heliyon.2018.e00938.

[2] W. Knight, “AI Can Do Great Things—if It Doesn’t Burn the Planet,” Wired. Accessed: Jun. 15, 2022. [Online]. Available: https://www.wired.com/story/ai-great-things-burn-planet/

[3] “Artificial Intelligence (AI): What is it and how does it work? | Lexology.” https://www.lexology.com/library/detail.aspx?g=5424a424-c590-45f0-9e2a-ab05daff032d (accessed Jun. 13, 2021).

[4] “Myelin, Membrane | Learn Science at Scitable.” https://www.nature.com/scitable/topicpage/myelin-a-specialized-membrane-for-cell-communication-14367205/ (accessed Jun. 16, 2022).

[5] “Fig. 2. Mathematical model of artificial neuron.,” ResearchGate. https://www.researchgate.net/figure/Mathematical-model-of-artificial-neuron_fig1_320270458 (accessed Jun. 23, 2022).

[6] K. O’Shea and R. Nash, “An Introduction to Convolutional Neural Networks.” arXiv, Dec. 02, 2015. Accessed: Jun. 23, 2022. [Online]. Available: http://arxiv.org/abs/1511.08458

[7] “MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges.” http://yann.lecun.com/exdb/mnist/ (accessed Feb. 24, 2022).

[8] S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, “Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications,” Jan. 2018.

[9] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,” Medium, Dec. 17, 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (accessed Jul. 21, 2022).

[10] A. intelligenceis evolving rapidly et al., “FPGA vs. GPU for Deep Learning Applications,” Intel. https://www.intel.com/content/www/us/en/artificial-intelligence/programmable/fpga-gpu.html (accessed Jun. 28, 2022).

[11] C. Eliasmith and C. H. Anderson, Neural engineering: computational, representation, and dynamics in neurobiological systems, 1. MIT Press paperback ed. Cambridge, Mass.: MIT Press, 2004.

[12] S. Dutta, V. Kumar, A. Shukla, N. R. Mohapatra, and U. Ganguly, “Leaky Integrate and Fire Neuron by Charge-Discharge Dynamics in Floating-Body MOSFET,” Sci Rep, vol. 7, no. 1, Art. no. 1, Aug. 2017, doi: 10.1038/s41598-017-07418-y.

[13] W. Guo, M. E. Fouda, A. M. Eltawil, and K. N. Salama, “Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems,” Frontiers in Neuroscience, vol. 15, 2021, Accessed: Jun. 30, 2022. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2021.638474

[14] M. Bouvier et al., “Spiking Neural Networks Hardware Implementations and Challenges: A Survey,” J. Emerg. Technol. Comput. Syst., vol. 15, no. 2, pp. 1–35, Jun. 2019, doi: 10.1145/3304103.

[15] M. Pfeiffer and T. Pfeil, “Deep Learning With Spiking Neurons: Opportunities and Challenges,” Front. Neurosci., vol. 12, 2018, doi: 10.3389/fnins.2018.00774.

[16] K. D. Fischl, A. G. Andreou, T. C. Stewart, and K. Fair, “Implementation of the Neural Engineering Framework on the TrueNorth Neurosynaptic System,” in 2018 IEEE Biomedical Circuits and Systems Conference (BioCAS), Oct. 2018, pp. 1–4. doi: 10.1109/BIOCAS.2018.8584720.

[17] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, “The SpiNNaker Project,” Proceedings of the IEEE, vol. 102, no. 5, pp. 652–665, May 2014, doi: 10.1109/JPROC.2014.2304638.

[18] J. Sawada 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, Oct. 2015, doi: 10.1109/TCAD.2015.2474396.

[19] M. Davies et al., “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning,” IEEE Micro, vol. 38, no. 1, pp. 82–99, Jan. 2018, doi: 10.1109/MM.2018.112130359.

[20] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, “A wafer-scale neuromorphic hardware system for large-scale neural modeling,” in Proceedings of 2010 IEEE International Symposium on Circuits and Systems, Paris, France, May 2010, pp. 1947–1950. doi: 10.1109/ISCAS.2010.5536970.

[21] C. Pehle et al., “The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity,” Frontiers in Neuroscience, vol. 16, 2022, Accessed: Jul. 06, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2022.795876

[22] CTNWaterloo, Nengo summer school lecture 1: Introduction & Things you can do with Nengo, (Jul. 23, 2020). Accessed: Dec. 15, 2020. [Online Video]. Available: https://www.youtube.com/watch?v=Lg7LJgd6Ra8&list=PLYLu6sY3jnoXTrkfA_WMn0nF_FGz-MaiJ&index=1

[23] A. Almansouri, A. Alturki, A. Alshehri, T. Al-Attar, and H. Fariborzi, “Improved StrongARM latch comparator: Design, analysis and performance evaluation,” in 2017 13th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME), Giardini Naxos - Taormina, Italy, Jun. 2017, pp. 89–92. doi: 10.1109/PRIME.2017.7974114.

[24] “Tuning curves — Nengo 3.2.1.dev0 docs.” https://www.nengo.ai/nengo/examples/usage/tuning-curves.html (accessed Jul. 17, 2022).

[25] “Spiking threshold as a parameter - General Discussion,” Nengo forum, Nov. 04, 2018. https://forum.nengo.ai/t/spiking-threshold-as-a-parameter/677/2 (accessed Jul. 17, 2022).