A Scalable Hardware-Efficient CSNN Accelerator with Binarized STDP and STDG-based Hybrid Learning

by

Hamid Rahimian Kalatehbali

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

GRADUATE PROGRAM IN
ELECTRICAL AND COMPUTER ENGINEERING
YORK UNIVERSITY
TORONTO, ONTARIO, CANADA

JULY 2025

Abstract

Spiking neural networks (SNNs) offer a biologically inspired, energy-efficient computing paradigm that processes information through sparse, event-driven spike signals. In this work, we present Binarized spike-timing dependent plasticity-gradient (BSTDPG), a novel SNN framework that combines Time-to-first-spike (TTFS) latency encoding with integrate-and-fire (IF) neuron model, binarized convolutional layers with local Spike-timing dependent plasticity (STDP) learning rule, and a gradient-based classifier. To support hybrid learning, the final output layer is trained using either a Support Vector Machine (SVM) or gradient-based backpropagation, allowing flexible deployment in both unsupervised and supervised settings.

The proposed architecture consists of two spiking convolutional layers followed by max-pooling and a classifier. A key innovation is the zero-skip optimization technique, which dynamically prunes inactive spike regions based on bounding boxes generated from TTFS-encoded spike trains. This drastically reduces unnecessary computations during training and inference. For instance, processing sample digit '0' during its first timestep without optimization results in full 28×28 convolution and 7860 ns latency. With zero-skip enabled, the latency drops to 1710 ns, achieving a $4.6\times$ speedup and 78.2% processing time reduction. A similar improvement is observed at the full network level, where zero-skip reduces prediction latency from 1459 μs to 1212 μs, saving 246 μs. We evaluate BSTDPG on Modified National Institute of Standards and Technology database (MNIST) and Fashion-MNIST (F-MNIST) across both shallow and deep network configurations. The model achieves accuracies of 99.1% and 91.0%, respectively, outperforming or matching several state-of-the-art SNNs such as BANN, BS4NN, S4NN, and even deep temporal SNNs, despite using fewer layers.

Together, these innovations highlight BSTDPG as a promising architecture for low-latency, low-power neuromorphic applications including edge-AI, robotics, and event-based vision, offering an effective trade-off between biological plausibility, efficiency, and learning performance.

Acknowledgements

I would like to acknowledge my supervisor, Professor Amirali Amirsoleimani, for providing the opportunity to conduct this research within his group and for his role in overseeing its completion. I also extend my thanks to my thesis committee members — Professor Arash Habibi Lashkari, Professor Hossein Kasiri, and Professor Razieh Salahandish — for their time and valuable feedback. I am also grateful to my colleagues and friends for their encouragement and for fostering a collegial environment.

Finally, I am deeply indebted to my family and close friends for their patience, understanding, and unwavering support throughout this journey. Their encouragement has been an invaluable source of motivation.

Dedication

To my family, for their endless love and sacrifices.

Table of Contents

List of Tables

List of Figures

Chapter 1

Overview

1.1 Background

SNNs represent a more biologically realistic approach to neural modeling. Unlike Artificial neural networks (ANNs), which use continuous activation functions, SNNs incorporate the concept of time and discrete spike events to transmit information [4]. This temporal dimension allows SNNs to process information more efficiently and with lower energy consumption [5]. The importance of SNNs lies in their potential to bridge the gap between artificial and biological neural systems, by more closely mimicking the behavior of biological neurons [6]. Additionally, the event-driven nature of SNNs makes them attractive for neuromorphic systems [7] and well-suited for processing temporal data and real-time sensory information, such as in robotics and autonomous systems [8]. SNNs are also being explored for low-power neuromorphic hardware implementations, potentially revolutionizing edge computing and Internet of things (IoT) devices [9]. Moreover, SNNs have shown promise in tasks involving temporal data, such as speech recognition [10], and object detection [11].

In contrast, ANNs [12] are computational models inspired by biological neural systems, consisting of interconnected neurons that process and transmit information to approximate solutions for complex problems in a manner similar to how the brain performs specific tasks [13]. They learn to perform tasks by analyzing large datasets and adjusting the strength of connections between neurons to optimize performance [14], making them applicable to decision-making systems such as computer vision [15], and natural language processing [16]. While both SNNs and ANNs aim to replicate aspects of biological information processing, they offer distinct advantages and face different limitations. ANNs benefit

from mature training algorithms, such as Backpropagation (BP) [17], and have achieved state-of-the-art results across a wide range of tasks. However, their reliance on dense, continuous computations leads to high energy consumption and limited biological plausibility. In contrast, SNNs provide a more faithful model of neuronal activity by leveraging sparse, event-driven communication and temporal dynamics, enabling lower power consumption and suitability for neuromorphic hardware.

1.2 Motivation

When comparing SNNs and ANNs in terms of training methods, a key distinction lies in the maturity and efficiency of available algorithms. ANNs are typically trained using gradient-based optimization techniques such as BP, which have been well-established and highly optimized through decades of research. In contrast, SNNs face significant challenges due to the non-differentiable nature of spike-based communication, resulting in a lack of standardized and efficient training methods. Consequently, SNNs often lag behind ANNs in terms of performance on large-scale benchmarks.

At the same time, the development of hardware-based SNN accelerators is driven by the growing need for real-time, low-power computing in neuromorphic systems. Traditional von Neumann architectures [18], while powerful, are poorly suited for the sparse, event-driven, and highly parallel nature of SNNs [19]. Hardware-based SNN accelerators address these challenges by enabling massively parallel, asynchronous computation that mirrors biological neural systems [19, 20]. Their event-driven operation reduces energy consumption by performing computations only when spikes occur, making them ideal for edge computing and mobile applications [9]. Moreover, their inherent compatibility with continuous, real-time sensory data opens up new possibilities for efficient and scalable neuromorphic processing.

These limitations in both training algorithms and hardware architecture motivate the need for a unified approach that combines a novel, hardware-friendly SNN training method with an event-driven accelerator specifically designed to support it. In this thesis, we propose an efficient training method for SNNs and present the design and hardware implementation of a modular, event-driven accelerator customized for neuromorphic computing. The accelerator integrates multiple spike encoding schemes and feature extraction mechanisms, including a novel fast convolution method, to evaluate their computational efficiency and functional performance. It supports a range of encoding methods, such as rate coding [21], TTFS [22], and phase encoding [23] as well as various neuron models, including IF [24] and Leaky integrate-and-fire (LIF) neurons [25, 26, 27].

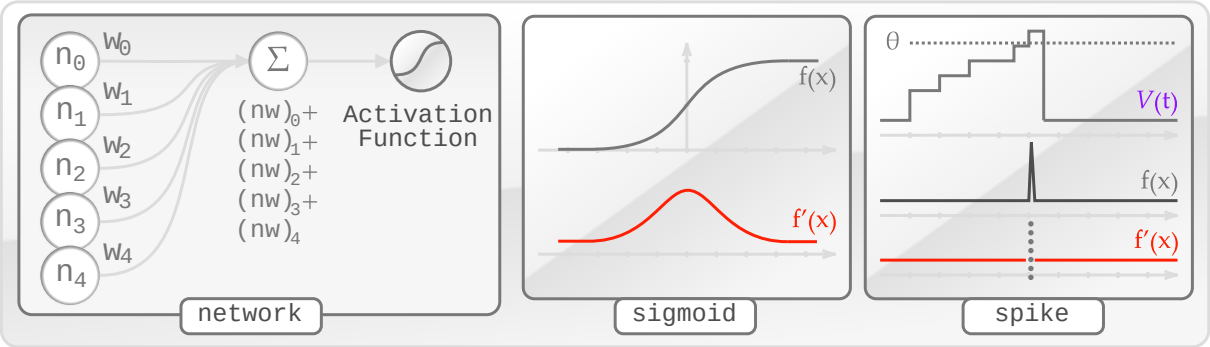


Figure 1.1: Comparison of differentiable ANN sigmoid function and non-differentiable SNN spike function.

1.3 Problem Statement

SNNs offer significant advantages in biological plausibility and energy-efficient computation by communicating via sparse, asynchronous spike events [28]. This makes them promising candidates for neuromorphic computing and edge applications where power constraints are critical. However, the fundamental challenge in training SNNs arises from the discrete, binary nature of spike function, which is inherently non-differentiable [29] at the spike point as shown in Figure 1.1. The jump state in spike function prevents the direct application of traditional gradient-based optimization methods, such as BP, that rely on differentiable activation functions. While the Backpropagation-through-time (BPTT) algorithm [30] has been adapted for SNNs, it suffers from substantial computational complexity and memory consumption, limiting its scalability and practicality in real-world neuromorphic systems [31]. On the other hand, biologically inspired learning rules such as unsupervised STDP [1] offer a more hardware-friendly and local alternative, as they rely solely on the relative timing of pre- and post-synaptic spikes. However, traditional unsupervised STDP often struggles to scale effectively for deep network training, which limits its accuracy on complex classification tasks.

These challenges highlight the need for novel training techniques that can efficiently bypass the non-differentiability of spikes while maintaining or improving the accuracy of SNN models. Approaches that leverage temporal locality, spike timing relationships, and binarized representations have shown promise in addressing these issues [32, 33, 2]. Still, there remains a gap in developing lightweight, fast, and hardware-friendly learning schemes suitable for large-scale SNN deployment. In this context, the problem this thesis addresses is the design and implementation of an efficient, scalable training method customized for binarized SNNs, combined with a highly reconfigurable accelerator in hardware.

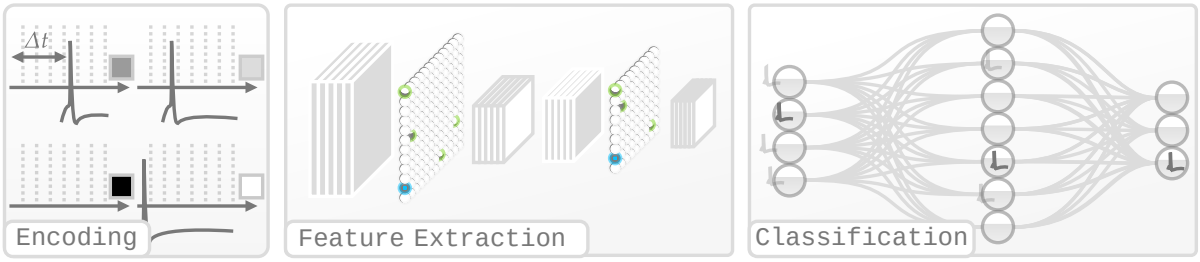


Figure 1.2: Overview of the proposed binarized SNN framework and its hardware accelerator.

1.4 Objectives

The primary objective of this thesis is to develop an efficient, hardware-friendly training method for binarized SNNs, and to design and implement a highly reconfigurable hybrid clock- and event-driven accelerator that supports the proposed learning method. To achieve this, the research focuses on three tightly connected goals as shown in Figure 1.2. First, we employ temporal coding technique and unsupervised STDP learning in feature extraction step to update synaptic weights in convolutional layers. This temporal encoding method provides a compact and precise representation of input features, reduces spike redundancy per timestep, and preserves temporal dynamics. The training algorithm uses binarized weights, enabling compatibility with low-power neuromorphic systems and reducing memory overhead. Second, for classification step we develop a novel supervised training scheme for binarized SNNs that integrates a sparse coding approach based on latency coding with a simple IF neuron model. To evaluate the trade-offs between computational complexity and accuracy, we explore the use of a gradient-based classifier at the output layer. Finally, we design and implement a modular event-driven accelerator architecture customized to support these unsupervised and supervised training methods. The accelerator includes configurable spike encoders, feature extraction modules, and supports both IF and LIF neuron dynamics. It is highly reconfigurable in terms of bit width, neuron count, memory bank structure, and Address-event representation (AER) encoders, making it adaptable for various network topologies and application constraints.

1.5 Contributions

To address current limitations in SNNs training and neuromophic hardware design, this thesis introduces the following novel contributions:

1. **Integrated on-chip spike encoding:** The accelerator features an on-chip encoding module capable of converting raw input samples into spike trains using either rate or TTFS coding schemes. This hardware-level integration not only streamlines the dataflow by eliminating the need for external encoding but also enables a fully event-driven, end-to-end neuromorphic pipeline.

2. **Hybrid feature extraction and classification approach:** We employ unsupervised STDP learning in the convolutional layers to update kernel weights during the forward pass, while for the backward pass, we propose a biologically plausible and lightweight supervised training algorithm that uses STDP principles to update binarized weights in the fully connected layers. This approach eliminates the need for non-differentiable gradient approximations and avoids the computational overhead of BPTT, making it highly suitable for efficient hardware implementation.

3. **A modular and reconfigurable event-driven accelerator architecture:** We implement a custom SNN accelerator in hardware that is modular and highly reconfigurable. The architecture allows configurable weight bit width, neuron count, weight memory bank size, and AER encoders per output neuron, enabling experimentation across various network configurations and resource constraints.

4. **On-chip sparse convolution for feature extraction:** A novel sparse convolution module is integrated into the accelerator to support real-time and fast feature extraction. To exploit the sparse nature of SNN, the proposed sparse convolution module performs computation only over the active regions of the input at each timestep. Specifically, a bounding box is dynamically generated around the non-zero pixels, and convolution operations are restricted to this localized region. By avoiding unnecessary computations over inactive areas, this approach significantly reduces computational load and increases processing speed, making it particularly well-suited for energy-efficient, real-time neuromorphic systems.

5. **Evaluation on standard benchmarks:** We validate the proposed training method on Caltech101 Face/Motorbike [34], MNIST [35], and F-MNIST [36] datasets. The results demonstrate competitive accuracy, spike sparsity, and energy efficiency, showing the practicality of the proposed solution in neuromorphic applications.

1.6 Structure

The remainder of this thesis is organized as follows:

- **Chapter 2** provides a comprehensive introduction to spiking neural networks, including neuron models, spike encoding schemes, and learning mechanisms.
- **Chapter 3** provides a comprehensive literature review, summarizing existing work on SNNs, and neuromorphic hardware.
- **Chapter 4** presents an overview of SNN training methods, outlining the challenges of training SNNs and reviewing commonly used algorithms such as STDP and BPTT. Also, it coveres the proposed supervised training method, detailing the binarized, biologically plausible learning scheme and its integration with temporal coding.
- **Chapter 5** describes the proposed SNN accelerator architecture, highlighting its modular, reconfigurable design and support for various encoding and neuron models.
- **Chapter 6** discusses the implementation and simulation results, including performance evaluations on benchmark datasets.
- Finally, **Chapter 7** concludes the thesis and outlines potential directions for future work, including architectural improvements and broader applications of the proposed system.

Chapter 2

Introduction

2.1 Spiking Neural Networks

Spiking Neural Networks (SNNs) are inspired by the way information is processed in the brain, and represent the third generation of neural network models, offering a biologically inspired paradigm that employes temporal dynamics, sparse communication, and event-driven processing [37]. SNNs have attracted significant attention due to their potential to be biologically plausible, energy-efficient, and well-suited for event-driven computation [5, 38]. Unlike conventional ANNs, where neurons continuously process and transmit information, SNN neurons emit discrete spikes only when their membrane voltage exceeds a certain threshold [39, 40]. This event-based mechanism inherently reduces power consumption by activating computation only at the moments when spikes occur, in contrast to the constant activity seen in ANNs [8]. Due to this sparsity and asynchronous communication, SNNs more closely mimic the operational principles of biological neural systems [41]. Moreover, their spike-based processing makes them highly compatible with neuromorphic hardware, offering the potential to achieve comparable or even superior performance to ANNs in certain tasks, particularly under strict energy and latency constraints [42, 43]. The timing of these spikes carries almost all the information about sensory inputs and motor commands. This temporal coding scheme is the foundation upon which SNNs are built.

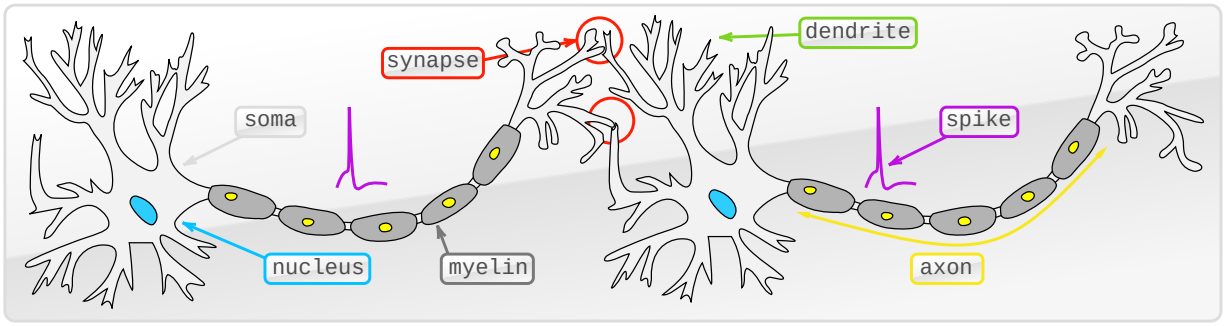


Figure 2.1: Neuron structure with the connection to another neuron through its dendrite and synapses.

2.2 Neuron

In the central nervous system, the fundamental processing units are neurons, which are interconnected in highly complex and structured networks. A typical biological neuron consists of three main components: dendrites, soma (cell body), and axon. Dendrites serve as the primary input structures, receiving signals from other neurons and transmitting them to the soma [44]. The soma functions as the computational core, integrating incoming signals and executing a nonlinear processing operation: when the accumulated input exceeds a specific threshold, the neuron generates an output signal. This output, known as an action potential or spike, is propagated along the axon, which serves as the output pathway to downstream neurons [4]. Communication between neurons occurs at specialized junctions called synapses, where the axon terminal of the presynaptic neuron releases chemical neurotransmitters in response to an incoming spike [28]. These neurotransmitters cross the synaptic cleft and influence the postsynaptic neuron, thereby enabling signal transmission across the network [44]. The action potential is a brief voltage pulse, typically lasting $1-2$ milliseconds with an amplitude of approximately $50-100$ millivolts [45] and can be measured by placing a fine electrode near the soma or axon of the neuron [46]. Figure 2.1 shows the details of a neuron connected to another neuron via its dendrite and synapses.

2.2.1 Neuron Models

A fundamental component of SNNs is the spiking neuron model, which controls the dynamics of membrane potential and spike generation. The most commonly used models include:

- **Integrate-and-Fire (IF)** [24]: When Input variable passes current, it might cross the threshold and if it does, a spike will be generated and then internally shorts out the circuit to discharge the capacitor to be ready for another cycle.
- **Leaky Integrate-and-Fire (LIF)** [25]: This model integrates incoming current over time with a leakage term. When the membrane potential exceeds a threshold, a spike is emitted, and the membrane potential is reset.
- **Izhikevich Model** [47]: A biologically plausible model that captures various firing patterns (bursting, regular spiking, etc.) while maintaining computational efficiency.
- **Hodgkin-Huxley Model** [48]: A biophysically detailed model based on ion channel dynamics, often used for neuroscientific simulations rather than machine learning tasks due to its computational complexity.

In this work, we adopt the IF, and LIF neuron models as the core computational elements of the proposed spiking neural architecture. These models are selected primarily due to their simplicity, mathematical tractability, and hardware-friendly characteristics, which make them particularly well-suited for efficient digital and neuromorphic implementations. The IF model, as the most basic spiking neuron model, captures the essential spiking behavior by accumulating input until a fixed threshold is reached, at which point a spike is generated [24]. The LIF model extends the IF model by introducing a leakage term, enabling a more biologically realistic representation of membrane potential decay over time [25].

Compared to more complex neuron models such as the Izhikevich and Hodgkin-Huxley models, IF and LIF offer significant advantages in terms of computational and memory efficiency, especially when deployed in large-scale neuromorphic systems. While the Izhikevich model is capable of replicating a wide range of biologically observed spiking patterns with relatively low computational overhead, it still involves non-linear differential equations that increase design complexity [47]. Similarly, the Hodgkin-Huxley model provides the highest level of biological fidelity through detailed ion channel modeling but is computationally restrictive for real-time applications or hardware implementations [48]. Thus, by leveraging the simplicity of IF and LIF neurons, this work achieves a favorable trade-off between biological plausibility, computational efficiency, and implementation feasibility.

2.2.2 Spike Trains

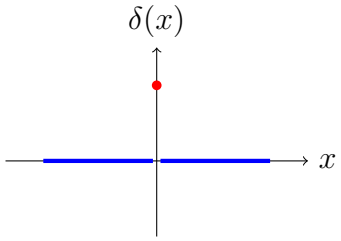
The neuronal signals pulses or spikes typically lasting $1-2$ milliseconds with an amplitude of approximately $50-100$ millivolts. The form of the pulse does not change as the spike

propagates along the axon. A chain of spikes emitted by a single neuron is called a spike train $\boldsymbol{S_i(t)}$ and defined in Equation (2.1). Spike train is a sequence of stereotyped (similar in their appearance and characteristics) events which occur at regular or irregular intervals. The form of a spike from a neuron does not provide any information since all spikes from the same neuron look the same [49]. Instead, the important factors are the timing and quantity of spikes [46]. Formally, we may denote the spike train of a neuron $\boldsymbol{i}$ as the sequence of firing times:

$$S_i(t) = \sum_f \delta(t - t_i^{(f)}) \tag{2.1}$$

where $\boldsymbol{\delta(x)}$ use the Dirac Delta function which is used to represent an infinitely narrow pulse at a specific point in space. As shown below, the Dirac Delta function is defined such that it is zero for all values of $\boldsymbol{x}$, except at $\boldsymbol{x = 0}$, where it is infinite. Therefore, spikes are represented as discrete points in time, which is why they are referred to as spikes.

$$\delta(x) = \begin{cases} 0, & x \neq 0 \\ \text{undefined}, & x = 0 \end{cases} \quad \text{with} \quad \int_{-\infty}^{\infty} \delta(x)\, dx = 1$$



Spikes in a spike train are usually well separated, and even with very strong input, it is difficult to generate (excite) a second spike during or immediately after the first one. The absolute refractory period is the shortest time interval between two spikes in which the neuron cannot be stimulated to excite another spike, no matter how strong the input is. After the absolute refractory period, there is a phase of relative refractoriness in which it is more difficult to stimulate the neuron to excite another spike, but still possible with a stronger input [46].

For example, as illustrated in Figure 2.2, the TTFS encoding process applied to a single MNIST digit. The original digit image is first downsampled from 28×28 to 14×14 pixels to reduce the dimensionality. The left side of the figure presents the resulting 14×14 grayscale image, with pixel intensities ranging from 0 to 255. On the right, the spike raster plot illustrates the TTFS-encoded spike trains corresponding to the 196 input neurons. Each red dot in the plot represents the firing time of a corresponding neuron along a 200 milliseconds simulation window. The x-axis denotes time in milliseconds, while the y-axis corresponds to the neuron index in a row-major order. The figure highlights the sparse

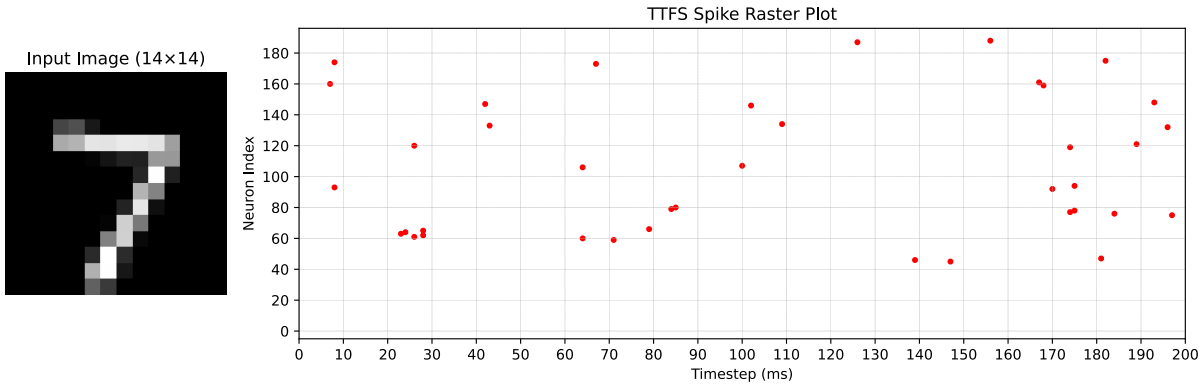


Figure 2.2: Spike raster plot of a 14×14 downsampled MNIST digit encoded using TTFS.

and temporally distributed nature of TTFS coding, where temporal information implicitly encodes input intensity. Therefore, each row represents the spike train of a single neuron, while each column corresponds to a discrete timestep, indicating which neurons fire at that moment.

2.2.3 Synapses

A synapse is the junction where the axon terminal of a presynaptic neuron connects to the dendrite or soma of a postsynaptic neuron. Between these two structures lies a narrow space known as the synaptic cleft. When a spike reaches the presynaptic terminal, it triggers the release of neurotransmitters into the synaptic cleft. These neurotransmitters then bind to receptors on the postsynaptic membrane, causing a change in the membrane potential of the postsynaptic neuron. This transformation from a chemical to an electrical signal is referred to as the postsynaptic potential [46].

2.2.4 Postsynaptic Potentials

The effect of a spike on a postsynaptic neuron can be observed using an intracellular electrode, which measures the voltage difference $\boldsymbol{u(t)}$ between the inside of the cell and its external environment. This voltage difference is referred to as the membrane potential. The membrane potential $\boldsymbol{u(t)}$ of neuron $\boldsymbol{i}$ at time $\boldsymbol{t}$ is determined by the cumulative effect of postsynaptic potentials generated by incoming spikes from presynaptic neurons. Each presynaptic neuron $\boldsymbol{j}$ may emit multiple spikes at times $\boldsymbol{t_j^{(f)}}$, with each spike contributing

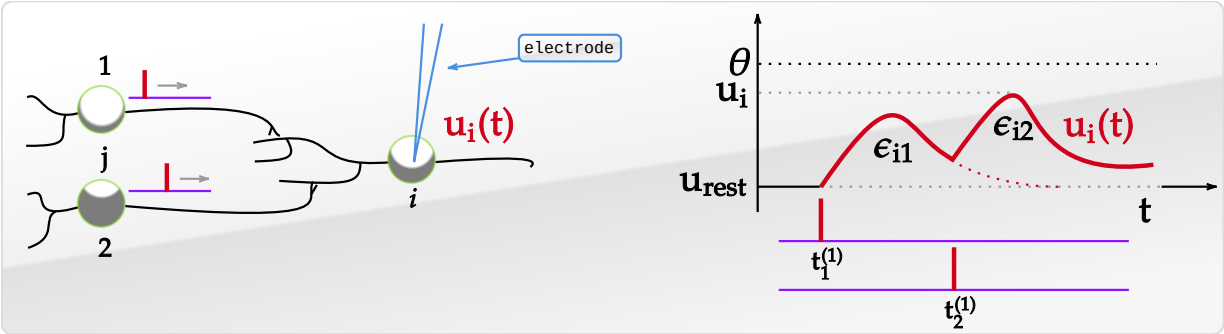


Figure 2.3: The excitory postsynaptic potential of neuron $\boldsymbol{i}$ caused by a spike from neuron $\boldsymbol{j}$.

to the postsynaptic membrane potential through a synaptic response kernel $\boldsymbol{\epsilon_{ij}(t - t_j^{(f)})}$, which characterizes the temporal profile of the postsynaptic response to an individual spike. The total membrane potential is calculated by summing over all such contributions from all presynaptic neurons and their respective spike times, as shown in Equation (2.2).

$$u_i(t) = \sum_j \sum_f \epsilon_{ij}(t - t_j^{(f)}) + u_{\text{rest}} \tag{2.2}$$

As illustrated in Figure 2.3, in the absence of any incoming spikes, neuron $\boldsymbol{i}$ remains in a resting state characterized by a constant membrane potential $\boldsymbol{u_{rest}}$. Upon receiving a spike from a presynaptic neuron $\boldsymbol{j}$, the membrane potential deviates from its resting value and, in the case of a LIF model, gradually decays back to $\boldsymbol{u_{rest}}$ over time [49]. The nature of the synapse determines the direction of this change: if the membrane potential increases (i.e., $\boldsymbol{u_i(t) - u_{rest} > 0}$), the synapse is classified as excitatory; otherwise, it is considered inhibitory. At rest, the membrane typically maintains a negative polarization around -65 millivolts. An excitatory input decreases this negative polarization, a phenomenon referred to as depolarization, whereas an inhibitory input increases the polarization further, resulting in hyperpolarization [50].

2.2.5 Firing Threshold

In a case where two presynaptic neurons, $\boldsymbol{j = 1, 2}$, transmit spikes to a common postsynaptic neuron $\boldsymbol{i}$. The first presynaptic neuron fires spikes at times $\boldsymbol{t_1^1}$, $\boldsymbol{t_1^2}$, ..., and the second neuron fires at times $\boldsymbol{t_2^1}$, $\boldsymbol{t_2^2}$, ... Each spike makes a postsynaptic potential in the target neuron, denoted by $\boldsymbol{\epsilon_{i1}}$ and $\boldsymbol{\epsilon_{i2}}$, respectively. The total membrane potential

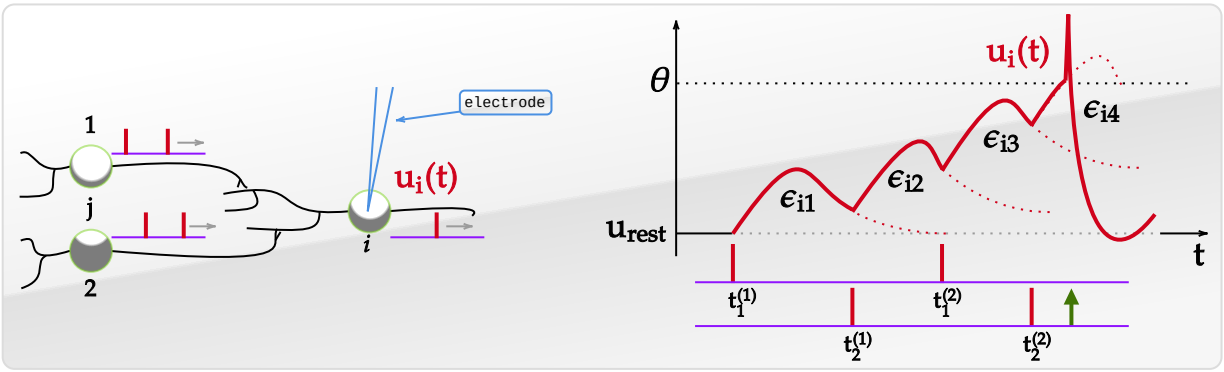


Figure 2.4: An action potential is triggered when $u_i(t)$ surpasses the threshold θ.

of the postsynaptic neuron is approximately the linear sum of these individual potentials. As shown in Figure 2.4, this summation continues in a linear fashion until the membrane potential surpasses the firing threshold θ, at which point the neuron generates a spike. When the accumulated potential reaches the firing threshold θ, this linearity breaks down and the neuron generates a spike with an amplitude of approximately 100 millivolts. The spike then propagates along the axon of neuron i to its synaptic targets. Following spike generation, the membrane potential drops below the resting potential u_{rest}, entering a brief period of hyperpolarization. Typically, the threshold θ is about $20 - 30$ millivolts above u_{rest}, and in most neurons, about $20 - 50$ presynaptic spikes must arrive within a short time window before postsynaptic spikes are triggered [51].

2.3 Spike Encoding Methods

In SNNs, input data, such as images, audio signals, or sensory information must first be converted into spike trains [52]. As already mentioned, neurons in SNNs operate using discrete events rather than continuous values. This conversion is referred to as spike encoding, and the choice of encoding scheme plays a crucial role in determining the performance and computational efficiency of the network. Several encoding strategies have been proposed in litrature, each with its own advantages and trade-offs [22, 49, 42].

In image classification task, different encoding schemes, such as rate coding [21, 53], temporal coding [22, 54], and population coding [55, 56], can be utilized to encode pixel values of a sample into spike trains. Rate coding is often considered less efficient because it does not take advantage of the temporal information that can be encoded in the precise

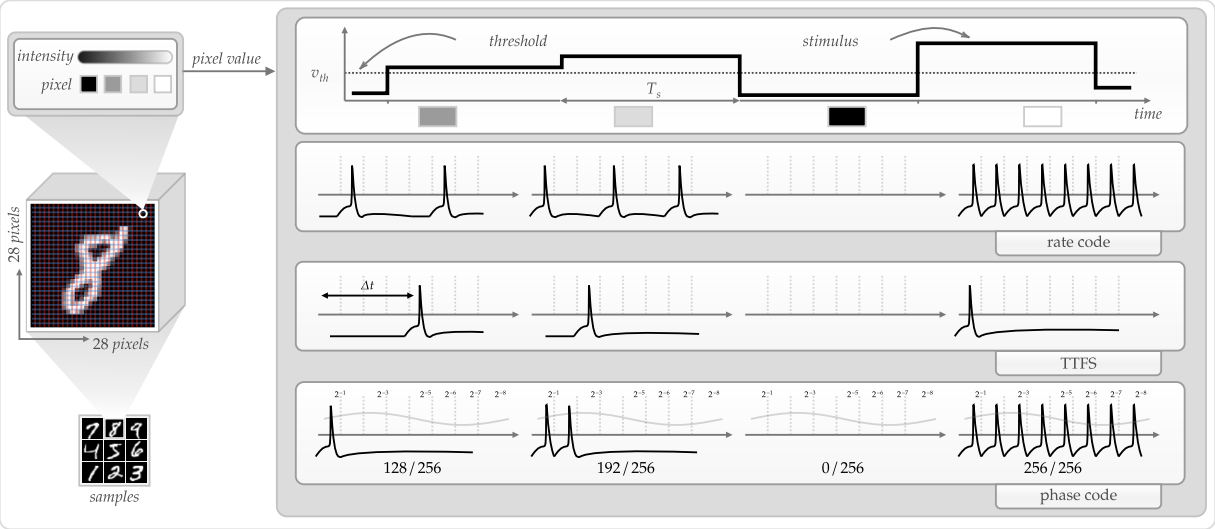


Figure 2.5: Visualization of different spike encoding schemes applied to an MNIST digit `'8'`.

timing of spikes within spike trains [57, 21]. Temporal coding includes phase [23], burst [58], and TTFS [22] coding, each relying on different timing aspects like a global oscillator, intervals between spikes, or precise spike timing. Figure 2.5 provides visual examples of several encoding schemes, which are discussed in the following subsections.

2.3.1 Rate Coding

In rate coding scheme, information is encoded in the firing rate of a neuron over a time window. A higher stimulus intensity corresponds to a higher firing frequency. Mathematically, the firing rate of neuron is defined as Equation (2.3), where for a given intensity value $I \in [0, 255]$ and total number of simulation steps $\boldsymbol{t_{sim}}$, the number of spikes $\boldsymbol{N_s}$ generated by a pixel is calculated as:

$$N_s = \left\lfloor \frac{I}{I_{max}} \times t_{sim} \right\rfloor \tag{2.3}$$

2.3.2 Temporal Coding

In temporal coding, information is conveyed through the precise timing of individual spikes rather than their frequency. A prominent example is the TTFS scheme [22], where the latency of the first spike encodes the input stimulus. This approach reflects how quickly a

neuron responds to a given input, and it means stronger stimuli result in earlier spike times. TTFS is particularly advantageous for neuromorphic systems, as it significantly reduces energy consumption and computational complexity by allowing each neuron to fire only once per stimulus [59]. This single-spike constraint also simplifies hardware implementation and reduces memory usage. The TTFS firing time is inversely related to input intensity and can be computed as Equation (2.4):

$$t = \left\lfloor \frac{I_{max} - I}{I_{max}} \times t_{sim} \right\rceil \tag{2.4}$$

where $\boldsymbol{t}$ denotes the spike time, $\boldsymbol{I}$ is the pixel intensity, $\boldsymbol{I_{max}}$ is the maximum possible intensity, and $\boldsymbol{t_{sim}}$ is the total simulation duration. The result is rounded to the nearest integer to ensure discrete time steps. This formulation enables rapid, efficient processing with minimal spiking activity.

2.3.3 Phase Coding

In phase coding, spikes are generated relative to an underlying oscillatory cycle, such as a global clock. The information is encoded in the phase at which the spike occurs within this cycle. This approach is biologically motivated by brain rhythms and can be used to coordinate activity across distributed networks. However, it may require synchronization mechanisms, making hardware implementation more complex [23].

2.3.4 Population Coding

Population coding is a neural computation strategy where information is encoded across multiple neurons rather than relying on individual neuronal responses, enabling more accurate representation of sensory and motor variables through population averaging [55]. This method is inspired by biological neural systems and is particularly useful for representing continuous values or multi-dimensional inputs.

2.3.5 Rank Order Coding

In rank order coding, each pixels is ranked by intensity, and neurons fire in the order of brightness. Therefore, each pixel is given a spike time based on how bright it is compared to the others. Brighter pixels fire earlier, and dimmer ones fire later. The exact intensity

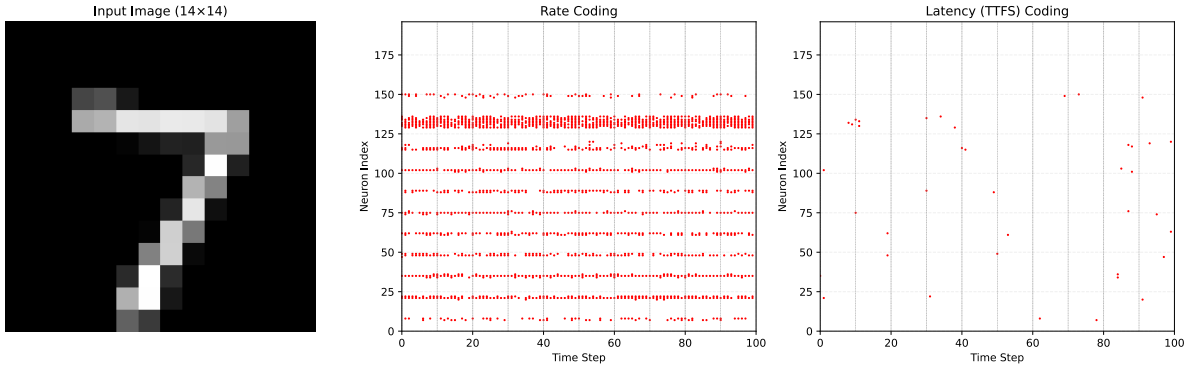


Figure 2.6: Comparison of rate coding and TTFS spike coding schemes applied to MNIST digit `'7'`.

does not matter, and only the order of brightness is used. However, it loses the exact intensity values, can be unclear when many pixels have similar brightness, and needs a way to decide which pixel fires first when the intensities are the same [57].

2.3.6 Comparison and Selection

The choice of neural encoding method depends on the specific application, hardware constraints, and the desired trade-offs between speed, accuracy, and energy consumption. Rate coding is robust to noise and easy to implement, making it suitable for noisy or redundant environments. However, it often requires long time windows to accurately represent input values, which can limit computational speed and energy efficiency. In contrast, temporal coding schemes such as latency and rank order coding offer faster and more energy-efficient alternatives by relying on fewer spikes, but they are more sensitive to noise and require precise spike timing mechanisms in hardware [60]. Population and phase coding provide more nuanced representations but typically involve more complex decoding mechanisms. Phase and burst coding, for example, use different strategies for temporal encoding and may require more spikes or offer less precision [59].

Figure 2.6 shows the encoding of MNIST digit `'7'` using two spike-based neural coding schemes that is implemented in this work. The middle plot shows rate coding, where spike frequency reflects pixel brightness, and the right plot shows latency coding, where brighter pixels cause earlier spikes. Each row represents spike train of a neuron corresponding to a pixel, and each red dot is a spike at a specific timestep. It is evident that in rate encoding, the spike patterns across consecutive time steps show significant repetition. This occurs because pixels that are near black or white produce nearly constant spike probabilities,

resulting in almost identical spike patterns at each timestep. Only pixels with intermediate gray values contribute to variability, generating stochastic spikes that differ over time. Consequently, much of the computational effort in rate coding is spent reprocessing largely redundant spikes from previous time steps, with meaningful temporal variation arising primarily from the gray pixels.

In this work, we implemented both rate coding and TTFS coding. However, TTFS was ultimately used for the image classification task due to its high efficiency, as it produces the fewest spikes among the considered methods.

2.4 Learning Mechanisms

As already mentioned, SNNs represent a biologically inspired paradigm that mimics the way real neurons communicate through discrete events or spikes. This event-driven communication model enables SNNs to operate with high energy efficiency, making them attractive for low-power and real-time applications. However, the training of SNNs remains an open challenge due to the non-differentiable nature of spike-based communication, resulting in a lack of standardized and efficient training methods [5, 33]. Therefore, the precise mechanisms through which biological neural systems learn are still not fully understood, and this make the development of effective training algorithms for SNNs considerably more complex than for conventional ANNs.

By contrast, the remarkable success of ANNs can be largely attributed to the BP algorithm [17], as it gained when it was demonstrated to effectively train deep architectures for classification and recognition. However, BP is widely criticized for its lack of biological plausibility. Several factors support this reason: first, biological neurons are thought to perform both linear and nonlinear operations, whereas BP relies on linear approximations through gradients. Second, BP assumes that signals can travel both forward and backward through connections, and it needs a global error signal that passes through all the layers. However, in the brain, learning happens locally and continuously, without needing a global signal to guide it [61]. More importantly, BP updates weights in fixed steps, while SNNs process information asynchronously through time-dependent spike trains [33].

Given these limitations, applying BP directly to SNNs poses substantial challenges, and training of deep SNNs is not straightforward primarily due to their complex temporal dynamics and non-differentiable spike function [31, 49]. As network size and depth increase, the computational and memory requirements also grow, with complexity scaling proportionally to the number of simulation time steps [61, 62]. These constraints preventing the

scalability and practicality of SNN training for real-world applications. Therefore, the development of more efficient and biologically plausible direct training strategies customized specifically for SNNs is highly desirable.

One approach involves surrogate gradient [63] methods to approximate the spike function, where the non-differentiable spike function is replaced by a smooth, differentiable approximation during the backward pass [64, 65, 66]. This allows SNNs to benefit from gradient-based optimization while preserving spike-based dynamics during inference. Several successful examples have been employed in the evolution of SNNs, such as using surrogate gradients to approximate the spike function in rate-coded SNNs [63, 64, 67], using rectified linear postsynaptic potential function [68, 69], and bypassing the spike non-differentiability by taking the derivative at spikes only in latency-coded SNNs [32, 70, 71]. Following the successful attempts in directly training SNNs, researchers are exploring more energy reduction options such as binarization of synaptic weights [72]. Although the use of binarized weights is not a new concept [73], there has been limited research in the context of binarized SNNs with latency coding.

Another option is to train a conventional ANN and then convert it into a SNN by mapping continuous activations to spike rates [74, 75]. This conversion approach leverages the strengths of deep learning while producing a network that can run with spiking neurons, though it often sacrifices the temporal coding advantages and leads to inefficient spike timing [76]. While several studies have investigated the conversion of pre-trained ANNs into functionally equivalent SNNs [77, 67], this process typically sacrifices biological plausibility and discards the temporal dynamics inherently encoded in spike-based processing [3].

A third option is employing an unsupervised learning rule where synaptic strength is updated based on the precise timing of pre- and post-synaptic spikes [1]. In this work, an online learning approach based on STDP-inspired Hebbian learning rule [78] is employed to train the convolutional layers of the proposed convolutional SNNs, as applying biologically plausible learning rules in convolutional layers has shown promising results in various studies [79, 80, 81, 82]. STDP is a biologically inspired synaptic learning rule that adjusts synaptic weights based on the precise timing relationship between pre- and post-synaptic spikes [1]. This mechanism enables the network to self-organize and extract meaningful spatiotemporal features from input data without requiring labeled supervision [83].

2.5 Address Event Representation

Traditional image processing techniques operate in a frame-based manner, processing the entire image at each frame and repeatedly accessing individual pixels, which incurs signif-

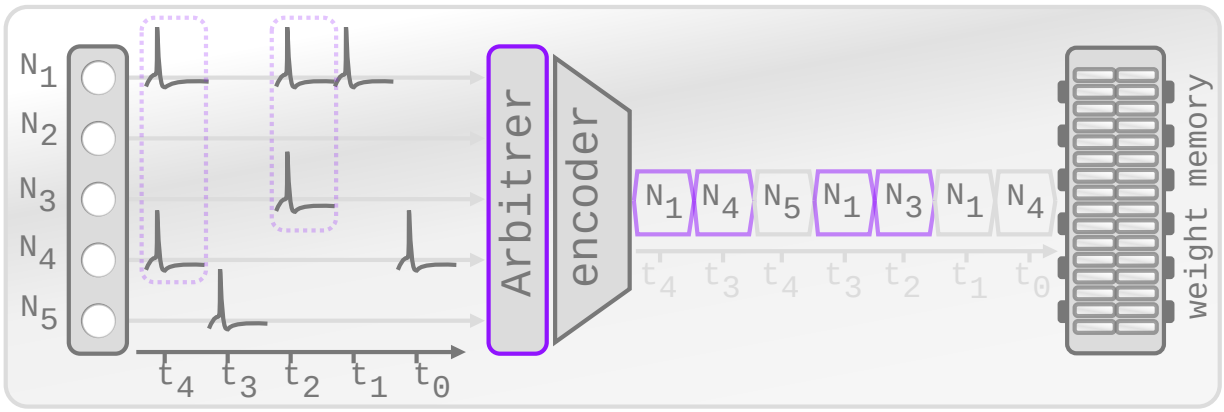


Figure 2.7: Multiplexing different neurons onto a single communication channel using AER encoders.

icant memory communication bandwidth [84]. In contrast, event-based image processing operates asynchronously by responding only to changes in brightness at individual pixels, rather than processing the full image frame [85]. This approach generates events only when meaningful changes occur, making the data inherently sparse and temporally precise. Each event encodes the unique address of the originating pixel along with its timestamp, allowing for efficient representation and processing of spatiotemporal visual information.

In hardware implementations, directly connecting thousands of neurons from one layer to the next layer, requires an large number of physical wire connections, which becomes increasingly difficult to implement on silicon due to routing congestion and signal conflicts. Each connection demands physical space for wires, vias, and associated routing resources, ultimately making the chip area extremely large. Moreover, driving signals across thousands of dedicated wires leads to significant power consumption and resource utilization. The connection complexity between fully connected layers grows as $\boldsymbol{\mathcal{O}(N^2)}$, where $\boldsymbol{N}$ is the number of neurons. For instance, with 1024 neurons in the hidden layer and 10 neurons in the output layer, a direct connection scheme would require $1024 \times 10 = 10{,}240$ physical wire connections. While 10,240 connections might seem manageable compared to the million wire connection in larger architectures, this is still challenging in hardware implementation.

To address this challenge and support concurrent processing of asynchronous spikes, the AER protocol employs arbitration and address encoding schemes to multiplex simultaneous events onto a shared communication bus, as illustrated in Figure 2.7. This approach reduces the physical connection complexity from $\boldsymbol{\mathcal{O}(N^2)}$ to approximately $\boldsymbol{\mathcal{O}(\log N)}$ in terms of bus width requirements [86], enabling more efficient routing, lower power consumption, and

a significantly smaller silicon footprint. The arbiter determines the order in which neurons gain access to the shared bus when multiple neurons attempt to transmit their addresses simultaneously. Various strategies have been developed to address bus conflicts, including random access arbitration, bus sensing that discards spikes when the bus is busy, tree-based arbitration that queues and orders access sequentially, and ring-based arbitration that uses a rotating token to allocate access rights [86].

Chapter 3

Literature Review

Spiking Neural Networks (SNNs) are biologically inspired networks that attracted growing interest as an alternative to traditional artificial neural networks (ANNs), offering advantages in energy efficiency, event-driven computation, and biological plausibility. However, realizing their full potential requires advances in both algorithmic training methods and hardware architectures. The development of effective learning rules that operate within the constraints of spike-based communication remains a major research challenge. At the same time, implementing these models on neuromorphic hardware demands specialized circuit designs that support sparse, event-driven, and asynchronous processing. This section reviews the state-of-the-art in both domains, highlighting several key approaches in SNN training as well as architectural strategies for neuromorphic computing systems.

3.1 Training Algorithms

In recent years, there has been a growing interest in applying the STDP algorithm within convolutional layers [80, 81, 82], demonstrating encouraging outcomes by using latency-coded networks on popular benchmarks such as the MNIST and Caltech101 [87] datasets. However, as networks grow in size and complexity, the need for more energy-efficient algorithms, such as weight binarization, is becoming increasingly important.

The authors in [88] presented a hardware implementation of STDP using $1-$bit synaptic weights for SNNs. Instead of applying continuous weight updates during training, their approach switches the synaptic weight directly between binary states (0 or 1) based on a computed probabilistic rule. When an update is triggered, the computed probability

determines whether the 1−bit synaptic weight should increase or decrease. To ensure stable learning, the method also incorporates weight normalization and lateral inhibition. The system was implemented on a Spartan−6 FPGA, where STDP units were shared across multiple neurons to reduce resource usage. Their design achieved a classification accuracy of up to 95.68% using 6,400 neurons in the feature extraction layer, which is comparable to other STDP-based methods that use full-precision weights. For the FPGA implementation, their system consumed 333 milliwatts at a 100 MHz clock frequency for a network with 1−bit synapse connections.

The authors in [89] proposed a 4−bit weight-quantized SNN that uses STDP for online learning. During training, full-precision weights were used for STDP updates to preserve learning accuracy, while quantized weights were employed during forward propagation to reduce computational cost. The architecture was implemented in hardware and synthesized using 45 nm (GSCLIB045) technology. The performance of this approach remained limited, achieving less than 94% accuracy on the MNIST dataset with 1−bit binarized weights, and under 95% with 4−bit and 8−bit quantized weights.

The authors in [90] introduced a deep residual convolutional SNN that employs binary synaptic kernels to significantly reduce memory usage and improve efficiency in neuromorphic computing. The architecture includes convolutional layers with LIF neurons, pooling layers, and a fully connected layer for classification. They trained the convolutional layers using a novel unsupervised Hybrid-STDP learning rule, a probabilistic STDP algorithm combining Hebbian and anti-Hebbian mechanisms. Also, in the final fully connected layer, they applied a supervised backpropagation step to preserve learning accuracy. For the result, they achieved up to 98.54% on MNIST compared to 32−bit full-percision weights. While this approach offers certain benefits, it also adds complexity to the training process by requiring each synaptic weight to use double the memory, increasing storage to 2−bit weights and preventing the network from being fully binarized.

The authors in [91] introduced BinaryConnect, a technique that enables BP in BANNs by using full-precision weights during the backward pass while binarizing them in the forward pass using a sign function. Building on this, XNOR-Net [92] introduced a per-layer scaling factor multiplied by binary weights to better approximate full-precision weights. This idea was further refined by [93], who proposed using learnable scaling factors instead. The BS4NN framework [94] extended these concepts to BSNNs, combining both binarized and full-precision weights. To address the non-differentiability of spikes during training, BS4NN incorporates the S4NN [32] training technique, which allows gradient-based learning while maintaining binary weights in the forward pass. BS4NN also employs learnable scaling factors for better approximation of full-precision parameters.

Most of these implementations rely on deep network architectures to achieve high prediction accuracy, as they are typically based on rank-order encoding [57, 60], which requires a substantial number of spikes to encode sufficient information. Although rank-order coding can transmit more information than rate coding [21, 53], it is still less efficient than precise temporal coding. Additionally, these methods often use complex neuron models such as the Spike Response Model (SRM) [95] or the LIF model, both of which impose greater hardware costs compared to the simpler IF model.

In this work, we adopt the binary synaptic weight approach proposed in [91, 88], where a sign function toggles weights between 0 and 1 during synaptic potentiation or depression. For training, we employ a hybrid strategy: direct supervised learning for the fully connected layer, following [32], and an unsupervised Hebbian learning rule for the convolutional layers. To reduce hardware complexity while maintaining functional accuracy, we utilize a simple IF neuron model across both convolutional and fully connected layers.

3.2 Neuromorphic Hardware Accelerators

Hardware accelerators for neuromorphic computing, particularly for SNN implementations, have been mainly realized using Field programmable gate arrays (FPGAs) and Application specific integrated circuits (ASICs). While FPGAs offer flexibility and rapid prototyping, ASICs typically provide superior performance, energy efficiency, and area utilization. Accordingly, this work is also implemented using an ASIC design. However, several notable FPGA-based implementations are also reviewed to highlight the design trade-offs and to provide a comprehensive overview of existing approaches.

3.2.1 FPGA Accelerators

Pani et al. [96] proposed an FPGA-based architecture using the Izhikevich [47] neuron model, and the equations are converted to fixed-point to balance performance and resource efficiency. The modular system supports up to $1,440$ neurons, with pipelining and parallelism enabling 0.1 ms delta-cycle real-time updates. The design reproduces biologically plausible spiking behaviors and is suitable for closed-loop neurophysiological experiments.

LaCSNN [97] implements real-time simulation of conductance-based SNNs using piecewise linear approximations of the Hodgkin-Huxley [48] model. Built on six Stratix III FPGAs, it simulates one million neurons and 60 million synapses at 50 Hz firing rate. The platform achieves 95% reduction in memory access, dissipating 10.578 W.

Han et al. [98] designed a hybrid time-stepped/event-driven SNN accelerator using LIF neurons on FPGA. Supporting 16,384 neurons and 16.8 million synapses, it consumes only 0.477 W. The accelerator achieves 97.06% accuracy on MNIST, offering significant power savings over Graphics processing unit (GPU) implementations.

Spiker [99] is an edge-focused FPGA accelerator using simplified LIF neurons and firing rate coding. Optimizations include removing $\boldsymbol{V_{rest}}$, using shift-based approximations, and weight quantization. Implemented on Artix-7, it classifies MNIST digits with 73.96% accuracy using 400 neurons.

SIES [100] accelerates spiking Convolution neural networks (CNNs) using a systolic array for membrane potential updates, spike generation, pooling, and buffering. Implemented on XCVU440 FPGA, it supports 4,000 neurons and 256 k synapses.

Cerebron [101] is an SNN accelerator using systolic arrays and adder-tree structures to reduce memory and improve energy efficiency. It achieves up to 20× speedup and 9.6× lower energy consumption on segmentation and classification tasks. Adding online learning and automating layer mapping are future directions.

3.2.2 ASIC Accelerators

The event-driven nature of SNNs enables the development of highly efficient systems featuring co-located memory and processing units, massive parallelism, and low energy consumption [102]. Several well-known neuromorphic platforms include Intel Loihi [9], IBM TrueNorth [103], SpiNNaker [104], SpiNNaker2 [105], ODIN [106], ReckOn [107], NeuroGrid [108], BrainScaleS2 [109], and DYNAPs [110].

SpiNNaker [104], implemented in 130 nm CMOS technology, is a massively parallel architecture capable of simulating up to 1% of the human brain, approximately one billion neurons in real time. It integrates up to one million ARM9 processor cores operating at 200 MHz, with 7 TB of distributed Random-access memory (RAM) across 57 k nodes, each containing 18 cores. The platform supports 36 k neurons and 2.8 M synapses per core, which adds to 2.5 billion neurons and 200 billion synapses in total.

The ODIN [106] chip, fabricated in 28 nm CMOS, integrates 256 neurons and 64 k 4−bit synapses controlled by the Spike-driven synaptic plasticity (SDSP) rule. IBM TrueNorth [103] features one million neurons and 256 million binary, non-plastic synapses per chip. Intel Loihi [9], a fully asynchronous neuromorphic architecture, supports up to 180 k neurons and 114 k to one million synapses per chip, with synaptic precision ranging from 9−bit to 1−bit.

A few recent designs, such as μBrain [111] and Speck [112], are fully event-driven and eliminate the need for global or per-layer clock synchronization. μBrain, fabricated in 40 nm CMOS technology, comprises 336 spiking neurons and 18.2 k synapses, achieving 91.17% accuracy on the MNIST dataset. Speck, implemented in 65 nm technology, integrates 327.6 k neurons and 272 KB of synaptic memory, achieving 98.56% MNIST accuracy. In both architectures, neurons can emit spikes immediately once their membrane potential exceeds a defined threshold, without waiting for a synchronization signal or timing clock.

Learning capability is a critical function in brain-inspired computing. However, most neuromorphic chips are limited to inference, with training typically performed offline on GPUs. This is suboptimal, as GPU architectures are not well-suited for the asynchronous and sparse nature of SNN workloads. To address this limitation, some neuromorphic systems integrate on-chip learning mechanisms. ODIN [106] supports the SDSP learning rule, while Intel Loihi [9] and SpiNNaker [104] incorporate modules for implementing STDP. More recently, architectures such as H2Learn [113] and SATA [114] have been proposed to enable on-chip training using BPTT for SNNs.

In this work, online unsupervised learning is employed in the convolutional layers using STDP for feature extraction. The final fully connected layer is trained offline on a GPU using a supervised learning algorithm. However, the proposed hardware architecture includes a dedicated interface to support supervised online learning via STDG updates in the fully connected layer. This interface is designed to enable future extensions of this work, where the entire network, including both feature extraction and classification, can be trained directly on-chip in a fully event-driven and online manner.

Chapter 4

Proposed Training Method

Spiking Neural Networks (SNNs) have attracted significant attention due to their potential for biological plausibility, energy efficiency, and suitability for event-driven computation [37, 5, 38]. Unlike traditional ANNs, which process all inputs at every time step, SNNs only respond to inputs that exceed a certain threshold [39, 40]. This event-based processing allows SNNs to communicate via discrete spikes rather than continuous activation values, thereby reducing computational load [111]. However, this same characteristic presents a major challenge for training deep SNNs, as the spike generation function is non-differentiable [31, 49].

As previously discussed, one training approach for SNNs is based on STDP, a biologically inspired unsupervised learning rule that adjusts synaptic weights based on the timing differences between pre- and post-synaptic spikes. While STDP preserves biological realism, its performance in terms of optimization and convergence is limited compared to models trained with gradient-based methods such as BP [17, 115]. Therefore, there is a growing need for direct training strategies that enhance SNNs without sacrificing biological plausibility or disregarding the temporal dynamics inherently encoded in spiking activity.

In this work, we adopt a hybrid training strategy: unsupervised Hebbian learning for convolutional layers and a direct supervised learning method for the fully connected layer. We propose a fast and energy efficient supervised learning rule that accounts for presynaptic and postsynaptic spike timings, which avoids BPTT in the backward pass entirely and is invariant to the number of simulation steps. The supervised method is derived from the S4NN [32] and BS4NN [3] frameworks, enabling effective training while maintaining low computational cost and temporal efficiency. The method uses full-precision weights during the backward pass while binarizing them in the forward pass using a sign function.

4.1 Model Architecture

As previously shown in Figure 1.2, the proposed SNN architecture consists of three main components: an encoding stage, a feature extraction stage, and a classification stage. In the first stage, a TTFS encoding scheme is employed to convert pixel intensities of the input MNIST image into spike trains. Each pixel is mapped to a single spike whose firing time is inversely proportional to its intensity, allowing the network to encode richer temporal information while ensuring sparse spiking activity. The resulting spike trains are then passed to the feature extraction stage, which comprises two spiking convolutional layers followed by a max pooling layer. These convolutional layers are designed to detect spatial features from the input spike patterns, with the max pooling layer reducing the spatial resolution and the number of active neurons, thereby improving computational efficiency. Finally, the extracted features are fed into the classification stage, which is composed of a fully connected layer of spiking neurons that performs decision making based on the temporal patterns of incoming spikes. This hierarchical structure enables the network to perform end-to-end spike-based image classification while preserving temporal precision and maintaining compatibility with neuromorphic hardware.

4.1.1 Encoding Method

The first step for image processing with SNN is to encode the image pixel intensities to spike trains. The choice of neural encoding method depends on the specific application, hardware constraints, and the desired trade-offs between speed, accuracy, and energy consumption. Rate coding is robust to noise and easy to implement, making it suitable for noisy or redundant environments. However, it often requires long time windows to accurately represent input values, which can limit computational speed and energy efficiency. In contrast, temporal coding schemes such as latency coding offer faster and more energy-efficient alternatives by relying on fewer spikes, but they are more sensitive to noise and require precise spike timing mechanisms in hardware [60]. In this work, we implement both rate coding and TTFS coding. However, TTFS was ultimately used for the image classification task due to its high efficiency, as it produces the fewest spikes among the considered methods.

To encode input information using a temporal coding scheme, each neuron is constrained to fire exactly one spike within the defined learning window, with the spike timing conveying the encoded information. In this work, the input MNIST image is encoded using the TTFS method, where pixels with higher intensities are assigned earlier firing times.

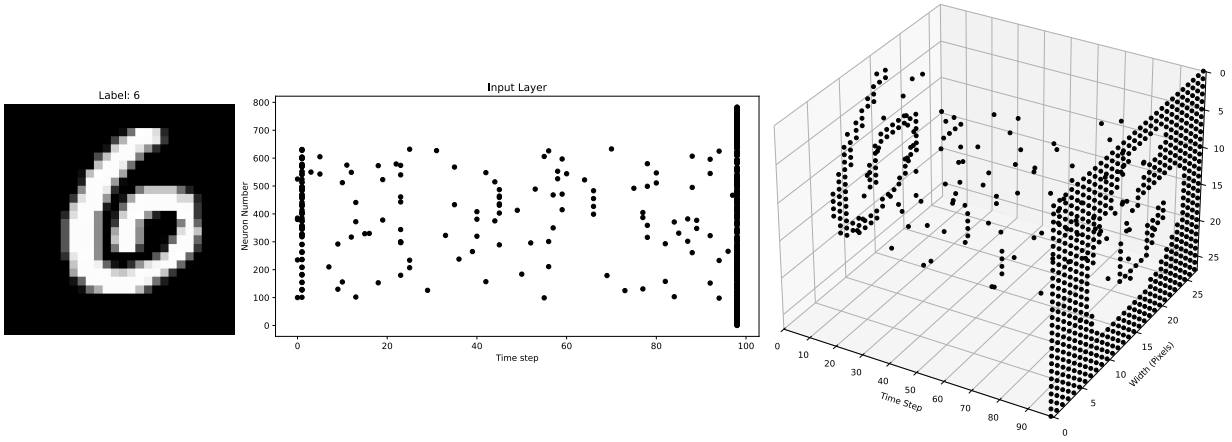


Figure 4.1: Visualizing MNIST digit into 2D and 3D spike trains from all time steps.

Specifically, the pixel with the highest intensity fires first, followed by others in descending order of intensity. This results in a sparse and temporally informative spike train that aligns well with the dynamics of spiking neurons. The encoding process is visualized in Figure 4.1 and is formalized in Equation (4.1), where $\boldsymbol{t}$ represents the firing time of a given pixel, $\boldsymbol{I}$ denotes the pixel intensity, $\boldsymbol{I_{max}}$ is the maximum pixel intensity in the image, and $\boldsymbol{t_{sim}}$ is the total simulation time.

$$t = \left\lfloor \frac{I_{max} - I}{I_{max}} \times t_{sim} \right\rceil \tag{4.1}$$

As illustrated in Figure 4.1, the original 28×28 grayscale image from the MNIST dataset is first fed into the TTFS encoder. The encoder produces a spike train in which brighter pixels generate earlier spikes and darker pixels spike later, over a simulation window of 100 time steps. A two-dimensional raster plot is also generated using a flattened view of the input layer, where each black dot represents a spike emitted by a specific neuron at a specific time. This visualization offers a concise and intuitive representation of how spikes are distributed across neurons and time. Additionally, a three-dimensional scatter plot is provided for improved interpretability. In this plot, each black dot corresponds to a spike and is positioned according to its pixel location and firing time, enabling a spatiotemporal analysis of the spike distribution. The 3D view effectively visualizes the temporal evolution of the spike matrix across the simulation window.

As illustrated in Figure 4.2, the encoded pixel values, represented as input layer, are

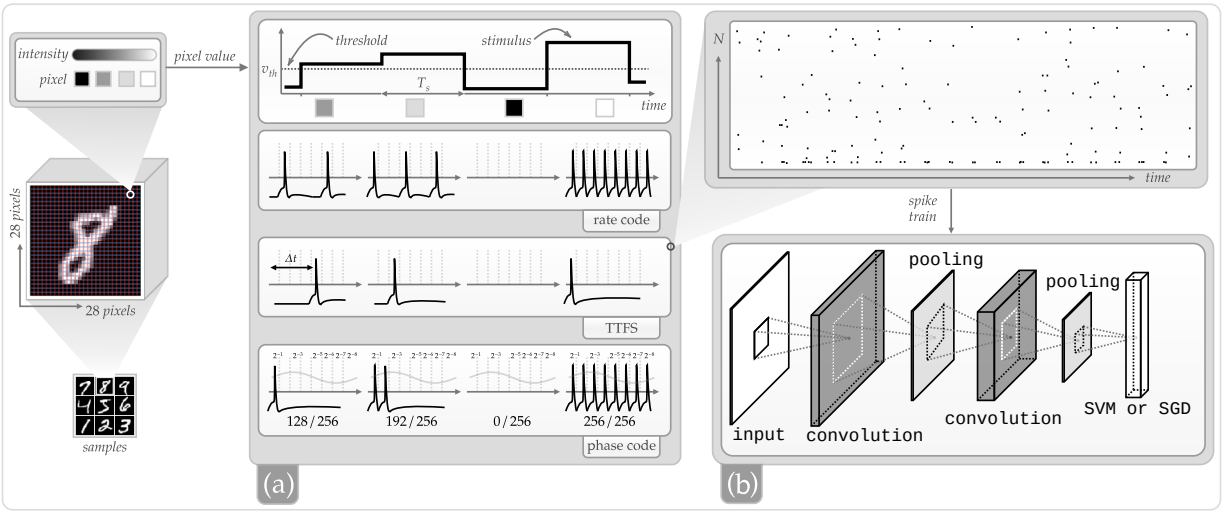


Figure 4.2: The workflow of the encoder and feature extraction stages of the proposed training method.

then passed to the first spiking convolutional layer in the feature extraction stage. In a spiking convolutional layer, each neuron is equipped with a set of input synaptic weights that define its feature selectivity, enabling it to respond to specific patterns such as edges, corners, or curves. These neurons are grouped into multiple maps, with each map dedicated to detecting a particular type of visual feature. Within each map, all neurons share the same synaptic weights and are arranged in a retinotopic layout, meaning they maintain the spatial structure of the input. This configuration ensures consistent feature detection across the visual field. The neurons in each map receive input from localized regions of the previous layer, and by using highly overlapping input windows, the network can detect features regardless of their exact position in the image. As a result, the spiking convolutional layer can extract increasingly complex patterns by building upon simpler features detected in earlier layers. All spiking convolutional layers in the feature extraction stage consist of IF neurons, which accumulate input spikes from presynaptic neurons and fire a spike when their membrane potential exceeds a predefined threshold.

Equation (4.2) defines the update rule for the membrane potential $\boldsymbol{U_i(t)}$ of the $\boldsymbol{i^{th}}$ neuron at each time step $\boldsymbol{t}$. In this equation, $\boldsymbol{j}$ indexes the presynaptic neurons, $\boldsymbol{\beta_{ij}}$ denotes the binarized synaptic weight between the $\boldsymbol{j^{th}}$ and $\boldsymbol{i^{th}}$ neurons, and $\boldsymbol{F_j(t)}$ is the spiking function, which takes the value 1 if the $\boldsymbol{j^{th}}$ neuron fires at time $\boldsymbol{t}$, and 0 otherwise. The membrane potential $\boldsymbol{U_i(t)}$ accumulates incoming spikes from presynaptic neurons, and when it reaches or exceeds a predefined firing threshold $\boldsymbol{v_{th}}$, the neuron emits a spike. Under the TTFS encoding scheme, each neuron is allowed to fire only once, and any subsequent

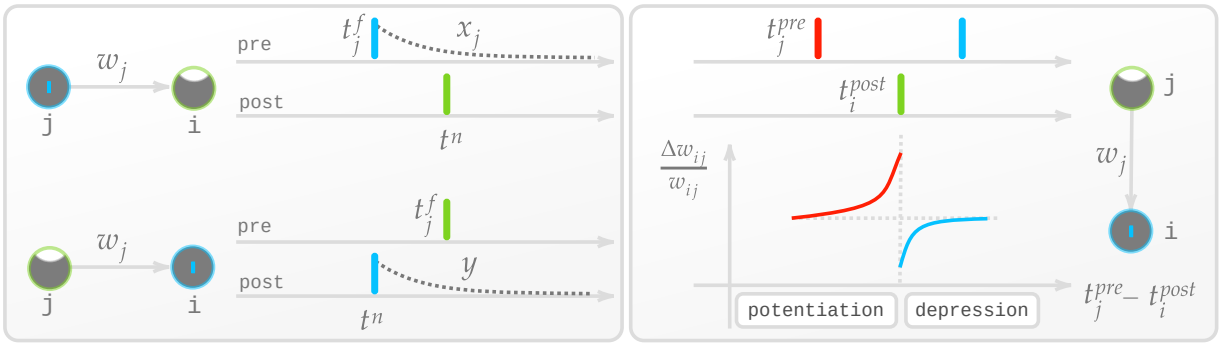


Figure 4.3: The STDP weight update rule that shows the change of synaptic connections, redrawn from [1].

inputs after spiking do not affect the simulation. To enable binarized inference, two sets of weights are used: full-precision weights $\boldsymbol{w}$ during training, and binarized weights $\boldsymbol{\beta}$ during forward propagation, where $\boldsymbol{\beta} = \boldsymbol{sign}(\boldsymbol{w})$.

$$U_i(t) = U_i(t-1) + \sum_j \beta_{ij}\, F_j(t-1) \tag{4.2}$$

4.2 Spike-timing Dependent Plasticity

STDP is an unsupervised, biologically plausible learning rule that adjusts synaptic weights based on the relative timing of spikes between pre- and post-synaptic neurons [1]. It is often considered as a temporal extension of Hebbian learning, summarized by the principle that `neurons that fire together, wire together` [116]. As illustrated in Figure 4.3, in STDP mechanism, if a presynaptic neuron fires shortly before a post-synaptic neuron, the synaptic weight is increased (potentiation). On the other hand, if the presynaptic spike occures after the post-synaptic spike, the weight is decreased (depression) [1]. This temporal asymmetry allows the network to capture causal relationships in spike patterns and encourage the formation of feature detectors. Mathematically, STDP is typically implemented using an exponential weight update rule, where the amount of change depends on the time difference $\boldsymbol{\Delta t} = \boldsymbol{t}_{\text{post}} - \boldsymbol{t}_{\text{pre}}$, often defined as Equation (4.3):

$$\Delta w = \begin{cases} +A^+ \,.\, exp(-\frac{\Delta t}{\tau_{pre}}) & \text{if } \Delta t > 0 \\ -A^- \,.\, exp(+\frac{\Delta t}{\tau_{post}}) & \text{if } \Delta t < 0 \end{cases} \tag{4.3}$$

where $\boldsymbol{A^+}$ and $\boldsymbol{A^-}$ are the learning rates for potentiation and depression, and $\boldsymbol{\tau_{pre}}$, $\boldsymbol{\tau_{post}}$ are the respective time constants.

While the mentioned classical STDP models offer a biologically inspired mechanism for learning based on the precise timing of spikes, their implementation often involves additional complexity, such as computing exponential decay update functions. These requirements, although manageable in software simulations, can become resource-intensive or impractical for real-time neuromorphic hardware systems. Moreover, the need to approximate exponential functions in hardware can further reduce accuracy and increase implementation complexity. To address these limitations, this work adopts a simplified yet biologically Hebbian-based learning rule [78]. This learning rule preserves the core principles of STDP synaptic strength based on spike timing, while eliminating the need for complex exponential calculations.

4.2.1 Hebbian-based STDP Learning Rule

In this work, a STDP-inspired Hebbian learning rule is used to implement synaptic plasticity in an unsupervised manner. The rule is a variant of STDP that modifies synaptic weights based on the temporal order of spikes but does not rely on an exponential timing window [83, 117]. Instead, it uses a simple yet biologically principle to drive weight updates based on the relative firing times of pre- and post-synaptic neurons. This rule enables layer-wise unsupervised training, beginning from the input layer and progressing sequentially through the network without the need for global error signals or BP [118]. As a result, each layer learns to extract relevant features from the spike trains it receives and updates its synaptic kernel weights accordingly.

The weight update of STDP-inspired Hebbian learning rule is defined in Equation (4.4). It ensures that synaptic weights stay within a fixed range and uses a multiplicative form that depends on the current weight value. This helps the network learn in a stable way over time.

$$\Delta W_{ij} = \begin{cases} A^+(W_{ij} - W_{\min}) \times (W_{\max} - W_{ij}) & \text{if } T_j \leq T_i \\ A^-(W_{ij} - W_{\min}) \times (W_{\max} - W_{ij}) & \text{if } T_j > T_i \end{cases} \tag{4.4}$$

Here, $\boldsymbol{W_{ij}}$ is the synaptic weight between the presynaptic neuron $\boldsymbol{j}$ and the postsynaptic neuron $\boldsymbol{i}$. $\boldsymbol{T_j}$ and $\boldsymbol{T_i}$ represent their respective spike times, and $\boldsymbol{W_{min}}$, $\boldsymbol{W_{max}}$ define the minimum and maximum values for the full-precision weights. The learning rates $\boldsymbol{A^+}$ and $\boldsymbol{A^-}$ control the magnitude of potentiation and depression. This method increases the

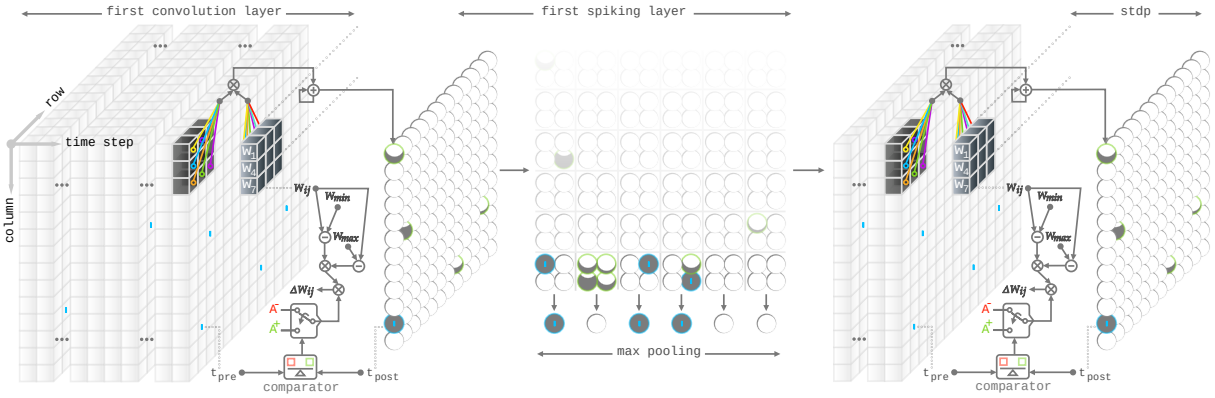


Figure 4.4: The STDP-based synaptic weight update process in a spiking convolutional layer.

weight if the presynaptic neuron fires before or at the same time as the post-synaptic neuron $(\boldsymbol{T_j \leq T_i})$, and decreases the weight otherwise $(\boldsymbol{T_j > T_i})$. The multiplicative terms $(\boldsymbol{W_{ij} - W_{min}})$ and $(\boldsymbol{W_{max} - W_{ij}})$ ensure that the updates decay as the weight approaches its limits, helping to prevent saturation. As illustrated in Figure 4.4, at each time step, spike-encoded inputs are convolved with the kernel weights of the layer, contributing to the membrane potentials of the post-synaptic neurons. If a neuron membrane potential crosses the firing threshold, it generates a spike. The STDP-based Hebbian rule is then applied: if a corresponding presynaptic spike is detected at the same spatial location (i.e., the neuron from the same receptive field region fired earlier or simultaneously), the synaptic weight is increased, otherwise it is decreased. The resulting weight update is denoted by $\boldsymbol{\Delta W_{ij}}$.

Durin training, input and output spike trains are used to guide synaptic updates. For each input, the top k output neurons are selected based on their earliest spike. In cases where multiple neurons spike at the same time, priority is given to those with higher membrane potentials. To promote sparsity in the learned features, lateral inhibition is applied across feature maps: when a neuron fires, it suppresses the activity of neighboring neurons within a fixed spatial radius r, preventing redundant feature learning in close proximity. This mechanism ensures that only the strongest neurons contribute to learning, enforcing competition among nearby neurons. The parameters k and r are crucial for the network behavior and must be tuned based on the specific characteristics of the dataset.

Moreover, a common challenge during training is that some weights may experience sudden sign flips (e.g., from $+1$ to -1), causing certain output channels or feature maps to become inactive, a phenomenon known as dead channels. Instead of discarding these inactive channels, a reactivation mechanism is employed: if a feature map shows no spike

activity or insufficient response over the last n training samples, its associated weights are sign flipped to bring the neuron closer to the firing threshold, encouraging re-engagement in the learning process.

During inference, the use of binarized weights introduces a potential challenge: the underlying full-precision weight W can saturate at its upper or lower bound, preventing further updates. To mitigate this issue, weight normalization is performed after processing each mini-batch of images. This step ensures that weights remain within a functional range, preserving their ability to adapt throughout training.

This STDP-based Hebbian learning rule is highly attractive for hardware implementation. It avoids the need for computationally expensive operations like exponential decay or continuous time integration. Instead, it relies on simple operations—such as comparisons, multiplications, and additions—which are well-suited to digital logic circuits. Additionally, since updates depend only on the local spike timing and synaptic state, the rule satisfies the locality constraint critical for event-driven, parallel neuromorphic systems. In this work, the rule is used to train the convolutional layers of the SNN, enabling the network to learn useful spatiotemporal features from spike-encoded inputs in a fully unsupervised manner.

4.2.2 Local Max Pooling

After training the convolutional layers with the input dataset, we apply spatial pooling to the spike maps. This technique merges adjacent pixels within the convolutional feature map, effectively reducing its dimensions while retaining the most important features [119]. Specifically, max pooling with 2×2 kernels were used featuring unit weights and stride lengths of 2, as shown in Figure 4.4. In the pooling layer, neurons perform max pooling by extracting the highest value from 2×2 windows within the feature map from the previous layer. This is achieved through pooling IF neurons, which are configured with input synaptic weights and thresholds both set to one. As a result, these neurons generate an output spike immediately upon receiving an input spike. Utilizing the TTFS encoding scheme, where each neuron produces at most one spike, allows us to propagate only the initial spike to represent the maximum value for the pooling operation.

4.2.3 Second Convolutional Layer

The layer-wise training strategy for convolutional layers employs the outputs of previously trained layers as inputs to subsequent ones. To enhance feature selectivity and learning

efficiency, we extend the use of lateral inhibition to both the input and output spike trains. During the pre-processing phase, for each input channel, only the earliest spike at each spatial location is preserved. In cases where multiple spikes occur simultaneously, the spike with the highest membrane potential is selected. This approach effectively suppresses weaker spikes, encouraging competition among neurons for distinct features extraction. It is important to note that lateral inhibition is applied only during training, and no such pre-processing is performed during inference.

4.3 Proposed Supervised Training Method

Following the unsupervised feature extraction phase, in which convolutional layers learn to detect important spatiotemporal patterns through spike-based activity, the network transitions to a supervised learning stage focused on classification. In this stage, the extracted spike-based representations are passed to a classifier layer initially trained using Stochastic gradient descent (SGD) algorithm [120]. However, due to the non-differentiability of spike generation functions, which poses challenges for gradient-based optimization, we propose a novel supervised learning rule, Spike-Timing Dependent Gradient (STDG) [33]. Inspired by the biological principles of STDP, STDG introduces a temporal error-driven gradient approximation that incorporates label supervision into the weight update process. This allows the network to effectively learn discriminative features for classifying MNIST digits despite the discrete nature of spikes.

The high-level architecture of the binarized STDG [2] network is shown in Figure 4.5, which has been redrawn for clarity and consistency with this work. In this design, the input layer receives spike trains coming from the flattened output of the second max pooling layer. These spike trains are then propagated through the subsequent layers of the network. The propagation continues until the spikes reach the output layer, where a comparison is made between the actual and target firing times to compute temporal errors. These errors are then used to update the synaptic weights using a form of temporal error BP.

To fully understand the operation of the binarized STDG method, it is essential to examine the underlying mechanisms of both the forward and backward passes in detail.

4.3.1 Forward Pass

Similar to ANNs, an SNN consists of input and output layers that are responsible for encoding the input signals and decoding the output responses. An SNN is composed

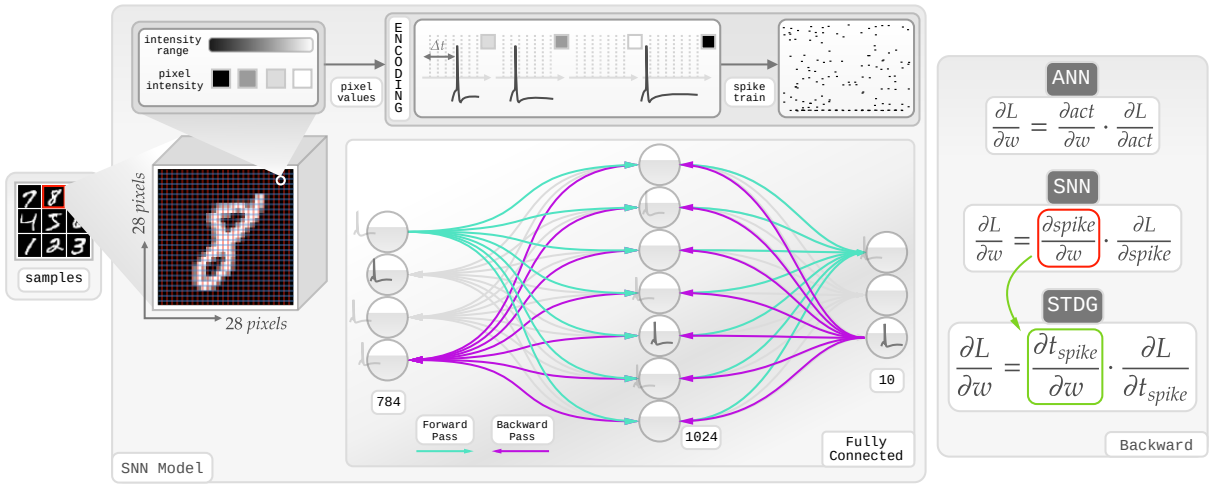


Figure 4.5: The network structure and workflow of binarized STDG, redrawn from [2].

of interconnected spiking neurons that communicate by transmitting discrete spikes to neurons in subsequent layers [8]. Input signals are integrated over time, and a neuron generates a spike when its membrane potential exceeds a predefined threshold.

The hidden layers comprise spiking neurons that process incoming spike trains and generate spike-based outputs. Information processing within the network is controlled by synaptic weights, which determine the strength of the connections between neurons. In our approach, we utilize two sets of weights: binarized weights, denoted by $\boldsymbol{\beta}$, and full-precision weights, represented by $\boldsymbol{w}$, where $\boldsymbol{\beta} = \boldsymbol{sign(w)}$. As shown in Figure 4.6, during the forward pass, the membrane potential of each neuron is computed using the binarized weights $\boldsymbol{\beta}$, which is more energy efficient. In the backward pass, however, the full-precision weights $\boldsymbol{w}$ are updated using temporal error BP. Finally, the output layer produces the network response based on the spike patterns generated by the preceding hidden layer.

During the simulation, the membrane potential of each neuron accumulates over time as incoming spikes arrive, as defined in Equation (4.5):

$$U_i^l(t) = U_i^l(t-1) + \sum_j \beta_{ij} X_j^{l-1}(t) \tag{4.5}$$

where $\boldsymbol{U_i^l(t)}$ represents the membrane potential of neuron $\boldsymbol{i}$ in layer $\boldsymbol{l}$ at time $\boldsymbol{t}$. $\boldsymbol{\beta_{ij}}$ denotes the binarized synaptic weight between presynaptic neuron $\boldsymbol{j}$ and postsynaptic neuron $\boldsymbol{i}$, and $\boldsymbol{X_j^{l-1}(t)}$ indicates the spike activity of neuron $\boldsymbol{j}$ in the previous layer at time $\boldsymbol{t}$.

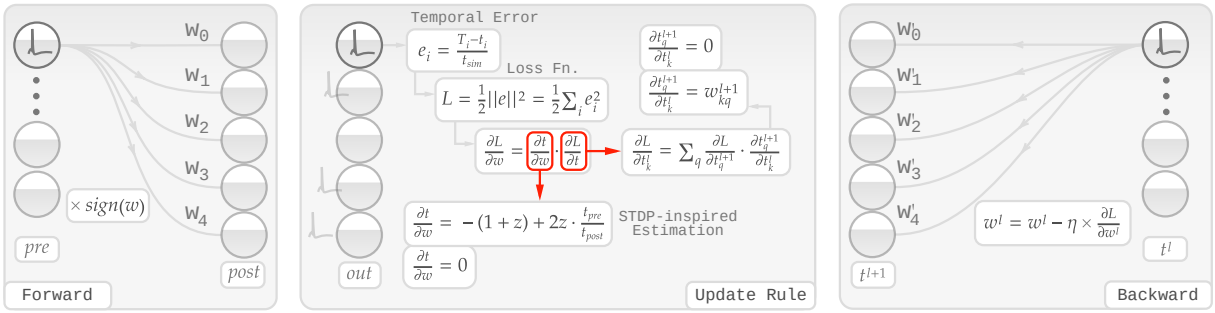


Figure 4.6: Forward and backward passes in STDG method using binarized weights for inference and full-precision weights for learning.

The neuron integrates these incoming spikes, and once its membrane potential exceeds a predefined threshold, it generates a spike and transmits information to the next layer. After firing, the output neurons are no longer monitored for the remainder of the simulation, even if additional spikes arrive.

In the output layer, each neuron is assigned to a distinct class out of 10 MNIST classes, and may fire at different time steps during the forward pass. The binarized STDG method makes a classification decision by selecting the output neuron that fires first or the one with the earliest spike time. Once a spike is emitted by any output neuron, the simulation can be terminated, as the network has reached a decision.

4.3.2 Backward Pass

In a classification task with a finite number of classes, such as the 10−digit categories in the MNIST dataset, each output neuron is assigned to a distinct class label. Once the forward pass is completed, the winning output neuron, the one that fires earlier than the others, is identified as the predicted class for the input image. As illustrated in Figure 4.6, a temporal error function $\boldsymbol{e_i}$ is defined for each output neuron to enable supervised learning, as shown in Equation (4.6):

$$e_i = \frac{T_i - t_i}{t_{\text{sim}}} \tag{4.6}$$

where $\boldsymbol{T_i}$ denotes the target firing time for output neuron $\boldsymbol{i}$. Actual firing time of that neuron is $\boldsymbol{t_i}$, and $\boldsymbol{t}_{\text{sim}}$ represents the total simulation time. This normalized temporal error quantifies how early or late a neuron fired relative to its target, and serves as the basis

for adjusting synaptic weights during training. Specifically, it captures the discrepancy between the target firing time $\boldsymbol{T}$ and the actual firing time $\boldsymbol{t}$ of each output neuron $\boldsymbol{i}$. To minimize this error, the SGD algorithm is employed during the learning phase to iteratively adjust the synaptic weights.

Since the network is expected to produce an earlier spike for the correct class, a minimum temporal margin $\boldsymbol{\gamma}$ is introduced to enforce a separation between the correct neuron firing time and those of all other output neurons. This ensures that the desired neuron fires at least $\boldsymbol{\gamma}$ time units before the incorrect ones. The target firing times are assigned according to Equation (4.7), where $\boldsymbol{c}$ denotes the correct class label and $\boldsymbol{t}_{\min}$ represents the earliest actual firing time observed among all output neurons.

$$T_i = \begin{cases} t_{\min} & \text{if } i = c \\ \max\left(\gamma + t_{\min},\ t_i\right) & \text{otherwise} \end{cases} \tag{4.7}$$

The loss function $\boldsymbol{L}$ is defined as the Mean squared error (MSE) between the target and actual firing times, as shown in Equation (4.8):

$$L = \frac{1}{2}\|e\|^2 = \frac{1}{2}\sum_i e_i^2 \tag{4.8}$$

To update the synaptic weights in the output layer, the gradient of the loss function with respect to the weights is computed. This is expressed in Equation (4.9) using the chain rule:

$$\frac{\partial L}{\partial \beta} \approx \frac{\partial L}{\partial w} = \frac{\partial L}{\partial t} \cdot \frac{\partial t}{\partial w} \tag{4.9}$$

However, calculating the term $\boldsymbol{\partial t / \partial w}$ is non-trivial, as it involves differentiating through the Heaviside step function [121] used in spike generation, which is non-differentiable. To address this challenge, we introduce a gradient approximation method called STDG, which provides a biologically inspired and computationally tractable way to estimate the gradients.

4.3.3 Spike-Timing Dependent Gradient

Since the partial derivative of spike timings with respect to the synaptic weights, $\boldsymbol{\partial t / \partial w}$, is inherently non-differentiable, we address this challenge by introducing an adaptive gradient rule specifically customized for spike timing dynamics. Given that the network decisions

depend primarily on the earliest spikes, these initial spikes carry more useful information and apply a stronger influence on the final decision. To fully leverage this idea, we assume that the contribution of a neuron to the gradient is proportional to the ratio between its input spike timing $\boldsymbol{t}_{\text{pre}}$ and output spike timing $\boldsymbol{t}_{\text{post}}$. To introduce flexibility in the gradient behavior and accommodate various data characteristics, we incorporate a hyperparameter $\boldsymbol{z}$ in Equation (4.10) to modulate this relationship:

$$\frac{\partial t}{\partial w} = \begin{cases} -(1+z) + 2z \times \frac{t_{\text{pre}}}{t_{\text{post}}} & \text{if } t_{\text{pre}} < t_{\text{post}} \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

The hyperparameter $\boldsymbol{z}$ is a learning coefficient constrained to the interval $[0, 1]$, which modulates the strength of the spike-time dependent learning rule by either amplifying or attenuating its effect. Assuming that the spike timing ratio $\boldsymbol{t_{pre}/t_{post}}$ is uniformly distributed over $[0, 1]$, Equation (4.10) yields an expected value of -1, which can be further scaled by the network learning rate $\boldsymbol{\eta}$.

To propagate the error gradient generated by the STDG through hidden layers l, it is essential to compute the gradient of the loss with respect to the spike timing of neurons in the preceding layer, denoted as t^{l+1}. This is achieved by expanding the gradient $\boldsymbol{\partial L/\partial W^l}$ as formulated in Equation (4.11):

$$\frac{\partial L}{\partial W^l} = \frac{\partial L}{\partial t^{l+1}} \cdot \frac{\partial t^{l+1}}{\partial t^l} \cdot \frac{\partial t^l}{\partial W^l} \tag{4.11}$$

Since spike timings $\boldsymbol{t}$ are seen as differentiable variables, we can directly propagate the error signal through the network layers without the need to compute gradients at each individual time step. The intermediate terms in this chain rule can be derived as Equation (4.12) and Equation (4.13), where $\boldsymbol{k}$ and $\boldsymbol{q}$ represent the indices of neurons in layers $\boldsymbol{l}$ and $\boldsymbol{l+1}$, respectively:

$$\frac{\partial L}{\partial t_k^l} = \sum_q \frac{\partial L}{\partial t_q^{l+1}} \cdot \frac{\partial t_q^{l+1}}{\partial t_k^l} \tag{4.12}$$

$$\frac{\partial t_q^{l+1}}{\partial t_k^l} = \begin{cases} w_{kq}^{l+1}, & \text{if } t_q^{l+1} > t_k^l \\ 0, & \text{otherwise} \end{cases} \tag{4.13}$$

When employing binarized weights in a neural network, the discrete nature of the weights can lead to unexpectedly transitions between -1 and 1. These sudden changes

introduce challenges in training the network while maintaining performance and accuracy. To mitigate this issue and enable the network to better adjust its firing times, the firing threshold $\boldsymbol{\theta}$ for each layer $\boldsymbol{l}$, comprising $\boldsymbol{N}$ neurons, is updated during the backward pass based on the gradient direction of the loss, as described in Equation (4.14):

$$\frac{\partial L}{\partial \theta^l} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial L}{\partial t_n^l} \cdot \frac{\partial t_n^l}{\partial \theta^l} \tag{4.14}$$

The neuron firing threshold $\boldsymbol{\theta}$ determines the specific time at which a neuron generates a spike in response to its input. A higher threshold delays the spike, whereas a lower value accelerates it. Due to this inverse relationship, the derivative $\boldsymbol{\partial t / \partial \theta}$ can be approximated as 1. However, to enhance the accuracy of this estimation, only effective synaptic inputs, those that actively contribute to the neuron firing are taken into account. This refined computation is shown in Equations (4.15) and (4.16):

$$\frac{\partial t_n^l}{\partial \theta^l} = \frac{1}{M} \sum_{m=1}^{M} \xi_{mn} \tag{4.15}$$

$$\xi_{mn} = \begin{cases} 1, & \text{if } t_n^l > t_m^{l-1} \\ 0, & \text{otherwise} \end{cases} \tag{4.16}$$

Here, $\boldsymbol{M}$ denotes the number of neurons in the preceding layer, and $\boldsymbol{m}$ and $\boldsymbol{n}$ represent the neuron indices in layers $\boldsymbol{l-1}$ and $\boldsymbol{l}$, respectively.

After completing the backward pass, the synaptic weights are updated using the gradient descent algorithm. Specifically, the weights and firing thresholds are adjusted using their respective learning rates, $\boldsymbol{\eta}$ for the synaptic weights $\boldsymbol{w}$, and $\boldsymbol{\eta_{th}}$ for the thresholds $\boldsymbol{\theta}$. The update rules are given in Equations (4.17) and (4.18):

$$\theta^l = \theta^l - \eta_{th} \cdot \frac{\partial L}{\partial \theta^l} \tag{4.17}$$

$$w^l = w^l - \eta \cdot \frac{\partial L}{\partial w^l} \tag{4.18}$$

To mitigate overfitting and improve the generalization capability of the network, we normalize the computed gradients and apply $\boldsymbol{L_2}$ regularization into the training objective.

Consequently, the original loss function (4.8), is modified to include a regularization term $\boldsymbol{\lambda}$, yielding the regularized loss function shown in Equation (4.19).

$$L = \frac{1}{2}\|e\|^2 + \sum_{ij} \lambda w_{ij}^2 \tag{4.19}$$

Here, $\boldsymbol{\lambda}$ is the regularization coefficient that controls the trade-off between minimizing the temporal error and constraining the magnitude of the synaptic weights. As a result of this regularization, the weight gradient during BP includes an additional term. Specifically, each weight $\boldsymbol{w_k}$ is reduced by an amount proportional to $\boldsymbol{2\lambda w_k}$, effectively modifying the gradient update step accordingly.

Chapter 5

SNN Accelerator Architecture

The proposed Spiking Neural Network (SNN) accelerator represents a neuromorphic computing architecture designed specifically for efficient image classification tasks, with MNIST digit recognition as the primary application. This architecture fundamentally differs from traditional ANNs by processing information through discrete spike events rather than continuous activation values, enabling ultra-low power consumption and event-driven computation. The accelerator implements a complete end-to-end pipeline that transforms input pixel data into temporal spike trains, processes these spikes through two spiking convolutional laters and two spiking layers of IF neurons, and produces classification outputs through spike-based decision making.

The architecture adopts a hybrid approach that combines the biological plausibility of spiking neurons with the structured organization of convolutional neural networks. This design enables the accelerator to use the spatial feature extraction capabilities of spiking CNNs while maintaining the temporal dynamics and energy efficiency characteristics of SNNs. The accelerator operates on 28×28 grayscale images from the MNIST dataset, processing each image through a spike encoding, convolution, pooling, and classification stages. The entire system is designed for hardware implementation with careful consideration of memory bandwidth, computational complexity, and real-time processing requirements.

5.1 Design Architecture

As illustrated in Figure 5.1, the top-level architecture of the proposed SNN accelerator is designed to communicate with external memory or a software interface to retrieve MNIST

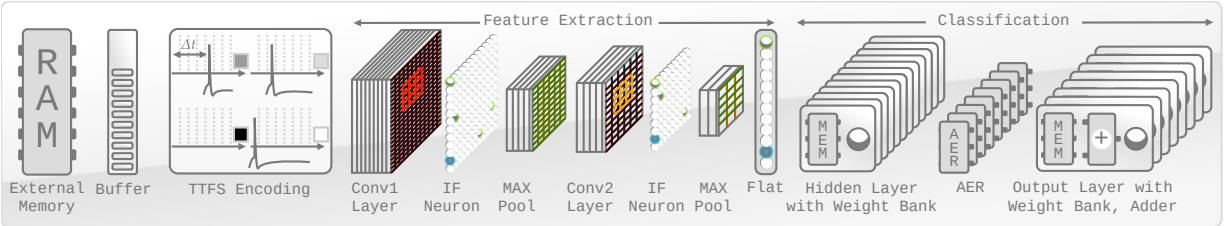


Figure 5.1: The top view of the proposed SNN accelerator interfacing with the external memory.

image samples and classify the input into 10 classes. Once a sample is fetched, it is transferred to the accelerator internal buffer for preprocessing and encoding. The buffer acts as the initial stage, sequentially reading 784 pixel values of the 28×28 input image. The encoder processes one sample at a time by generating spikes for the current time step and forwarding them to the spiking convolutional layers for feature extraction. This procedure is repeated for each time step until the complete simulation window is covered.

After completing the full temporal encoding and feature extraction process, the resulting spike-based representations are forwarded to the classifier module. The network implements a classification network with 1024 hidden neurons and 10 output neurons, corresponding to the number of classes in the MNIST dataset. The design implements IF neuron models with AER encoders to achieve efficient spike-based computation. Each hidden layer IF neurons, equipped with dedicated weight storage implemented using block RAM and instantiated with 64 weighted values from the previous layer. The layer receives weighted inputs from the maxpooling stage, which are processed sequentially rather than in parallel. This sequential processing eliminates the need for AER encoding between the input and hidden layers, as temporal conflicts between simultaneous spikes do not occur. However, in the output layer neurons, there are AER encoders equiped to select weight and adders to compute the cumulative weights send it to the output layer neuron as input. The erliest output neuron spike will be considered as the prediction and the rest spikes will be avoided.

As illustrated in Figure 5.1, the top-level architecture of the proposed SNN accelerator is designed to interface with external memory or a software host to retrieve MNIST image samples to classify each input into one of 10 classes. Once a sample is fetched, it is transferred to an internal buffer for preprocessing and encoding. This buffer serves as the first stage in the pipeline, sequentially reading the 784 pixel values of a 28×28 input image. The encoder processes one sample at a time by generating spikes at each time step and forwarding them to the spiking convolutional layers for feature extraction. This process is repeated across multiple time steps until the entire simulation window is covered.

After completing the temporal encoding and feature extraction stages, the resulting spike-based representations are passed to the classifier module. The classification network consists of 1024 hidden neurons and 10 output neurons, corresponding to the MNIST class labels. The design utilizes IF neuron models along with AER encoders to enable efficient spike-based computation. Each hidden-layer IF neuron is equipped with dedicated weight storage, implemented using block RAM, and instantiated with 64 weights from the previous layer. The hidden layer receives input from the max-pooling stage and processes the incoming spikes sequentially rather than in parallel. This sequential processing removes the need for AER encoding between the input and hidden layers, as there are no temporal conflicts arising from simultaneous spikes.

In contrast, the output layer incorporates AER encoders that dynamically select the appropriate weights and control the adders used to compute the cumulative input weight for each output neuron. The earliest spike generated by the output neurons is interpreted as the final prediction, and any subsequent spikes are ignored. In cases where multiple output neurons spike simultaneously, the neuron with the highest membrane potential is selected as the predicted class, ensuring a deterministic and conflict-free decision.

5.2 SNN Accelerator Design

The accelerator operation is controlled by a Finite state machine (FSM) that employs a 4−bit state encoding to coordinate nine distinct operational states: `IDLE`, `INIT`, `LOAD`, `ENCD`, `PREP`, `CONV`, `NEXT`, `SNN`, and one more state for future work as `STDG`. This FSM serves as the central control unit that control the complex interactions between different processing modules and ensures proper timing synchronization throughout the computational pipeline for convolutional layers. The state transitions are carefully designed to maximize hardware utilization while maintaining deterministic processing behavior, with each state having specific responsibilities and clear transition conditions that depend on completion signals from various processing modules.

5.2.1 Proposed Finite State Machine

The FSM architecture as shown in Figure 5.2, enables efficient pipelining of operations by overlapping computation and data movement phases. The designed FSM adopts a structured approach, in which each state is responsible for specific computational tasks, such as memory access, input encoding, convolution and max pooling, and classification.

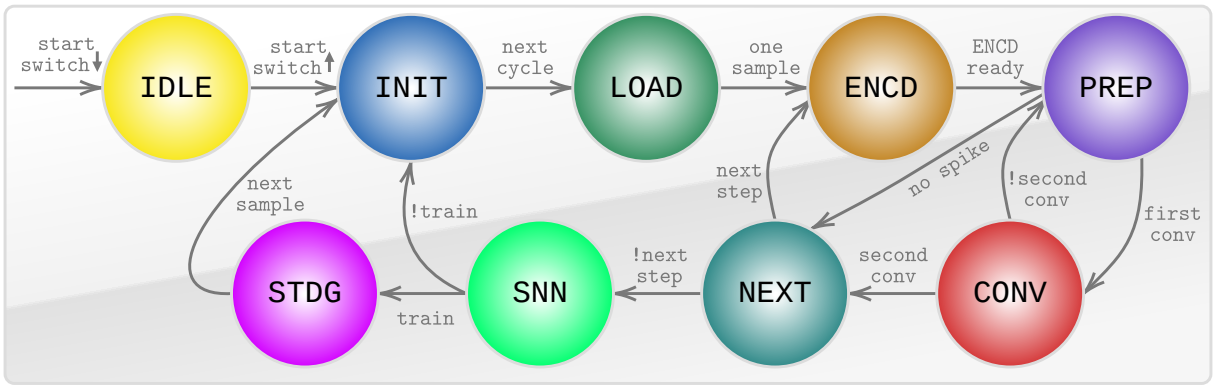


Figure 5.2: Finite state machine architecture for spiking neural network hardware implementation.

The transitions to the next state is also based on predefined completion conditions and system configuration.

🟡 `IDLE`: The `IDLE` state serves as the system standby mode, waiting for an external trigger to initiate processing. During this state, the system remains in a low-power configuration with all computational units disabled. The FSM monitors the start switch signal (`start_sw`), which is active low, to determine when to begin processing. When the start signal is deasserted, indicating a request to process new data, the system transitions to the 🔵 `INIT` state and calculate the memory address, and transitions to the 🟢 `LOAD` state.

🔵 `INIT`: The `INIT` state performs system initialization and prepares the hardware for memory data loading operations. Key operations include calculating the memory address and offset, enabling the memory read signal to allow sequential pixel data transfer from external memory, and clearing the convolution switch flag to ensure proper layer sequencing. This state also initializes various control signals and prepares the memory buffer subsystem for the incoming data transfer. The 🔵 `INIT` state always transitions to the 🟢 `LOAD` state on the subsequent clock cycle, ensuring rapid progression through the initialization sequence.

🟢 `LOAD`: The `LOAD` state manages the sequential transfer of input image pixel data from memroy to the internal memory buffer. During each clock cycle, when the read valid signal is asserted, the system copies data from the memory output to the internal buffer starting from the first address location. The address counter is incremented after each transfer, progressing through the entire image pixels. For MNIST digit recognition, this process involves loading 784 pixels corresponding to a 28×28 image. The state continues until the address counter reaches the buffer depth parameter, at which point it transitions to

the ENCD state. This sequential loading approach ensures reliable data transfer while maintaining synchronization with the external memory interface.

ENCD: The ENCD state implements spike encoding algorithms that convert pixel intensities into temporal spike patterns. The system supports multiple encoding schemes selected by the 2−bit enc_fn parameter. In rate coding mode, each pixel intensity is mapped to a number of spikes distributed over the simulation time window as defined in Equation (2.3). In the TTFS encoding scheme, pixel intensities maps to specific time delays, where higher intensities spike earlier in the temporal window using Equation (2.4).

During encoding, the system performs bounding box computation when the z_skip optimization is enabled. This involves tracking the minimum and maximum row and column indices of active spikes to define a region of interest for subsequent convolution operations. Each generated spike is stored in a zero-padded buffer enc_buff that includes boundary padding to handle edge cases during convolution. The ENCD state processes all 784 pixels in raster scan order, incrementing column and row counters appropriately. After processing the entire image, the state sets the cnv_prep flag and transitions to the PREP state to configure convolution parameters.

PREP: The PREP state configures convolution parameters and establishes processing boundaries for the subsequent convolution operations. The system calculates the convolution region start and end coordinates based on the bounding box information computed during encoding. When z_skip optimization is enabled, the convolution region is restricted to the area containing active spikes, reducing computational overhead. The convolution switch cnv_sw is toggled to alternate between processing the first or second convolutional layers. When cnv_sw is low, the system processes the first convolutional layer operating on the original 28×28 spike data. When cnv_sw is high, the system processes the second convolutional layer operating on the 14×14 max-pooled data from the first convolution layer. The state also resets bounding box registers for the next time step and clears validation flags to prepare for the next processing cycle. Depending on the current layer configuration and spike validity, the system either proceeds to the NEXT state to encode the next time step or proceeds to the CONV state to begin convolution operations.

CONV: The CONV state performs the core computational operations of the SNN, including convolution, max pooling, and synaptic weight updates. The convolution operation implements a 3×3 kernel that slides across the input feature map, computing the weighted sum of inputs within each receptive field. The system processes different input sources depending on the convolution switch setting: encoded spikes for the first layer and max-pooled results for the second layer.

Each convolution result is fed to leaky IF neurons arranged in a two dimensional grid

corresponding to the feature map dimensions. The first convolutional layer contains 784 IF neurons arranged in a 28×28 grid, while the second layer contains 196 neurons in a 14×14 configuration. The IF neurons integrate the weighted input currents and generate output spikes when their membrane potential exceeds the threshold voltage.

Max pooling operations are performed on the spike outputs using 2×2 windows with a stride of 2. This reduces the spatial dimensions by half while preserving the most significant features. The max pooling results are stored in dedicated buffer `mpl1_buff` for subsequent processing.

During the training phase, the system employs STDP-based Hebbian learning to update synaptic weights. Upon detecting a postsynaptic spike, the system evaluates the presynaptic activity within the corresponding receptive field. If presynaptic spikes occur before the postsynaptic spike, the associated synaptic weights are potentiated; otherwise, they are depressed controlled by a predefined learning rate. The `CONV` state transitions are determined by completion of processing regions and spike detection in the output layer. Depending on the convolution switch `cnv_sw`, the system either returns to the `PREP` state to begin second convolution operations or proceeds to the `NEXT` state to encode the next time step.

`NEXT`: The `NEXT` state manages temporal progression through the processing pipeline. The system increments the time step counter and evaluates whether the current temporal window has been completed. For each image, the system processes multiple time steps defined by `TIMESTEP` parameter to capture the temporal dynamics of the spike-based computation. When the time step counter reaches the maximum value, indicating that temporal processing is complete, the system transitions to the `SNN` state for event-driven classification. If additional time steps remain, the system returns to the `ENCD` state to continue temporal processing with the next time step.

`SNN`: The `SNN` state handles event-driven classification and MNIST prediction. The fully connected SNN involves 1024 hidden neurons with banked weight memory organization. AER encoders are used to efficiently fetch the synaptic weights and process sparse spike patterns. The system accumulates weighted contributions from all active inputs using a two-phase summation approach that handles both Most significant bits (MSBs) and Least significant bits (LSBs) separately. The output layer consists of ten IF neurons corresponding to the $10-$digit classes in MNIST dataset. These neurons receive weighted inputs from the 1024 hidden layer neurons through a fully connected architecture. The system implements a `winner-take-all` selection mechanism using priority encoding. When multiple output neurons spike simultaneously, the priority encoder selects the neuron with the highest membrane potential as the predicted class.

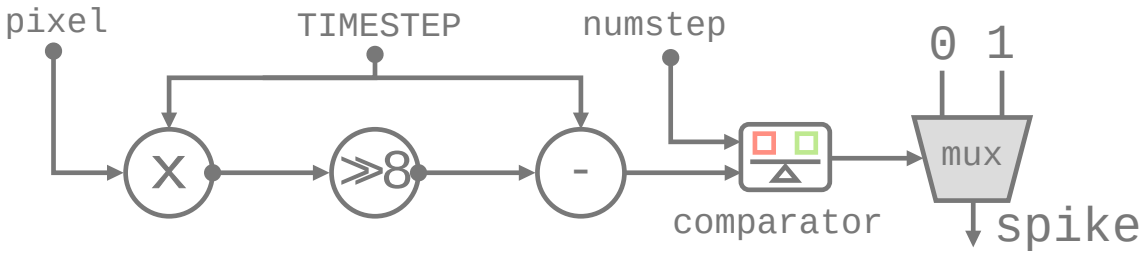


Figure 5.3: Schematic of approximated TTFS module.

When the output layer generates spikes (`snn_index > 0`), indicating that a prediction has been made, the system transitions based on the current operating mode. In training mode, it proceeds to ● `STDG` to perform synaptic plasticity updates using the supervised learning rule. In inference mode, it returns to ● `INIT` to process the next input sample.

● `STDG`: The `STDG` state implements STDG weight updates for the classification layer connections. The STDG learning rule adjusts synaptic weights based on the relative timing of actual firing time and target firing time as introduced in Equation (4.6).

5.2.2 TTFS Spike Encoder Design

The spike encoding module serves as the interface between the memory buffer containing pixel intensities and the digital domain of hardware processing. This module supports TTFS encoding scheme as defined in Equation (2.4). In TTFS encoding, each neuron is constrained to generate exactly one spike within a predefined simulation window, with the spike timing encoding the input intensity, earlier spikes indicate higher intensities.

To implement TTFS encoding in hardware, a synchronous Verilog module was developed to map 8−bit pixel intensities to corresponding spike times within a fixed `TIMESTEP`. To ensure efficient hardware deployment, the division by $\boldsymbol{I_{max}}$ which is 255, was approximated using a bitwise right shift by 8 bits, which reduces hardware complexity by eliminating the need for a dedicated division unit.

The schematic of the TTFS module is illustrated in Figure 5.3. The design comprising a single 8−bit multiplier, a subtractor, a comparator, and a small number of control registers. Specifically, the design utilizes one multiplier for computing the spike timing, and a comparator for detecting when the current time step equals the calculated spike time. Due to its low logic footprint, the module is well suited for scalable instantiations across large arrays of input neurons or pixels in neuromorphic systems.

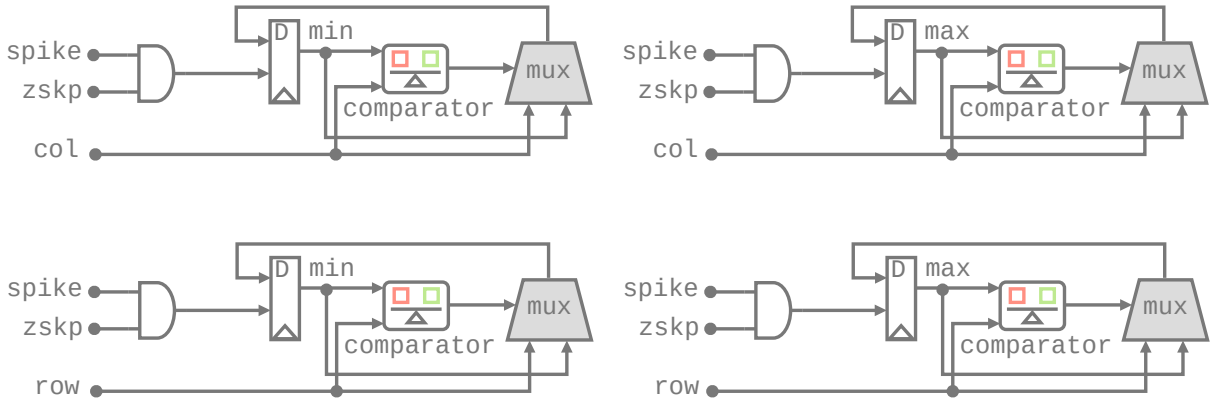


Figure 5.4: Bounding box circuits to define a rectangular region over active area.

5.2.3 Zero-skip Bounding Box

The bounding box module, depicted in Figure 5.4, implements this optimization by tracking the spatial extent of spike activity. It maintains four key coordinates: `min_row_img` and `max_row_img` define the vertical bounds, while `min_col_img` and `max_col_img` define the horizontal bounds. These coordinates dynamically form a rectangular region that bounds all spike locations. The implementation is highly efficient, requiring just four $5-$bit registers for 28×28 images and basic comparison logic, making it particularly well-suited for neuromorphic systems.

The bounding box computation is integrated with the ● `ENCD` state, executing in parallel with the encoding process. Whenever a spike is generated, and the `zero-skip` feature is enabled, the module compares the current spike row and column indices with the existing boundary values. If a spike lies outside the current boundaries, the module updates the variables accordingly, thereby maintaining a rectangle that bounds all spiking activity. Because this update process occurs in real-time as spikes are encoded, no additional computation phase is required.

After identifying the active region for each time step of the current sample, the system dynamically computes the convolution boundaries with appropriate padding for both convolutional layers. This dynamic region computation is essential due to the reuse of shared convolution and max pooling circuitry across two processing stages. A dedicated control `switch` signal determines whether the current operation related to the first or second convolutional layer and whether the `zero-skip` optimization is enabled for the current context. The process involves calculating the exact start and end coordinates for the con-

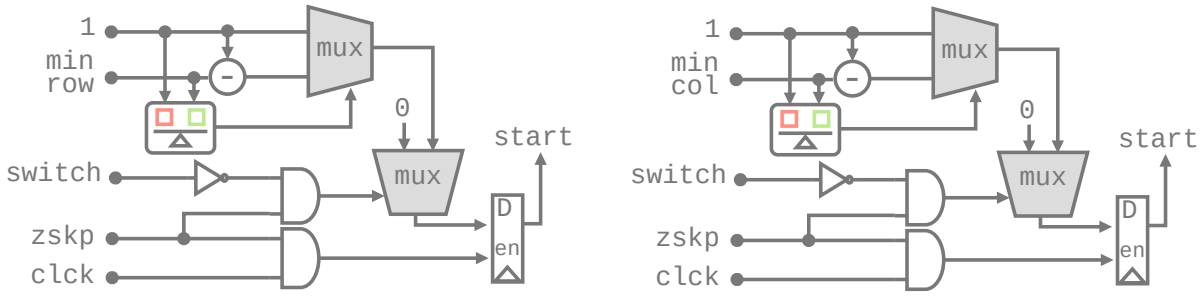


Figure 5.5: Hardware architecture of convolution start boundary calculation by applying padding for correct alignment.

volutional operation based on the previously identified bounding box, while adhering to padding requirements. Padding calculations are particularly important for maintaining consistent feature map dimensions and handling edge conditions, even when operating on a spatially reduced region.

As illustrated in Figure 5.5 and Figure 5.6, the circuit effectively translates the spike-based bounding box into appropriately padded convolutional regions for both first and second layers. This translation preserves the correctness of the network pipeline while improving computational efficiency.

Figure 5.5 shows the hardware logic for calculating the dynamic start coordinates of the convolution region. This logic operates in combinational fashion and is activated when the `zero_skip` optimization is enabled. The row start coordinate is computed as `min_row-PAD` if `min_row` is greater than or equal to the padding value. Otherwise, it defaults to `1` to ensure safe indexing. The column start coordinate follows the same principle. If the `zero_skip` feature is disabled or the operation related to the second convolution layer, both coordinates default to 0, indicating that the full image or feature map is processed without spatial reduction.

Figure 5.6 illustrates how the system calculates the end coordinates of the convolution window. These coordinates are derived by adding the padding value to the bounding box maximum row and column positions. The result is then checked against upper bounds to ensure it does not exceed the image limits, which is 27 for the 28×28 input image. When `zero_skip` is disabled, the end coordinates default to their maximum values: 27 for the full image in the first convolution layer and 13 for the 14×14 feature map in the second layer. This guarantees that convolution operations remain valid and fully cover the active region, even in boundary conditions.

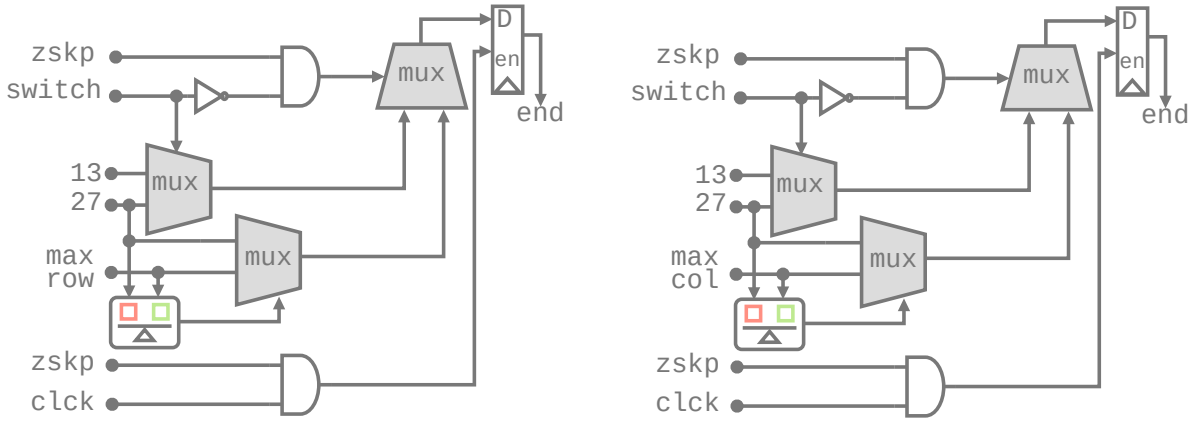


Figure 5.6: Hardware architecture of convolution end boundary calculation by applying padding for correct alignment.

5.2.4 Spiking Convolution Layer

To perform conditional convolution in hardware, a Verilog module was developed to compute the dot product between a 3×3 kernel and a localized region of the input feature map during the ● `CONV` state. As shown in Figure 5.7, this module supports two modes of operation, determined by the `switch` control signal. When `switch` is high, the module operates on the encoded spike buffer `enc_buff` using weights from the first kernel bank `swght1`. When `switch` is low, the input is taken from the max-pooled feature map buffer `mxpol1`, with corresponding weights from the second kernel bank `swght2`.

In both modes, the module performs a 3×3 convolution by scanning the local input region and computing the sum of products between the kernel weights and corresponding input values. The accumulated result is stored in a signed register `result`. To reduce unnecessary computation, each Multiply-accumulate (MAC) operation is gated by a conditional check that ensures both the input value and the corresponding kernel weight are non-zero. This gating mechanism is particularly effective in the non-active zones within the bounded active region, where no spike is present. The buffer indexing logic is aligned with the minimum convolution window determined during the encoding stage, enabling the convolution operation to be spatially localized and computationally efficient.

Operating in parallel with the ● `CONV` state, the design incorporates a set of 784 IF neurons that process convolution outputs in real-time. Each IF neuron corresponds to a spatial location in the feature map and receives the convolution result at that location as input current on every clock cycle. This fully parallel architecture enables simultaneous

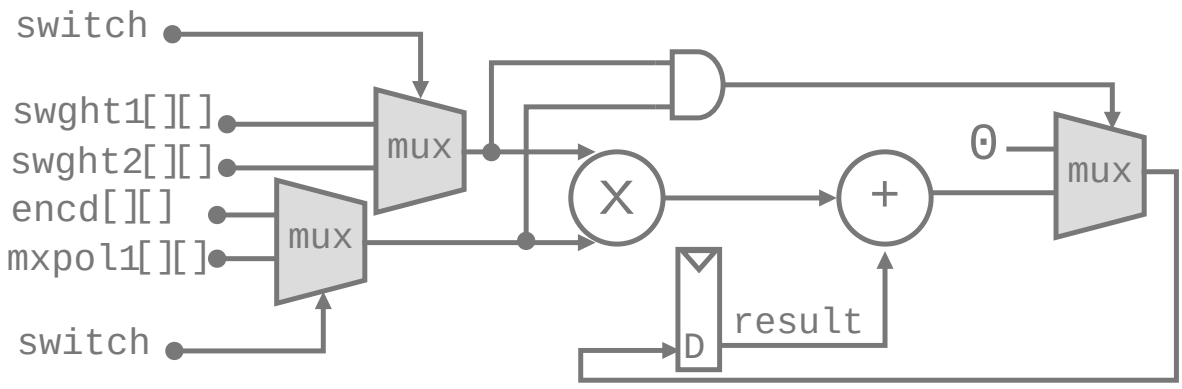


Figure 5.7: The first stage of conditional convolution unit that supports two modes.

processing of all spatial positions, thereby maximizing computational throughput while preserving the temporal characteristics of spike-based processing. The IF neurons integrate the incoming convolution values as membrane potentials, which increase cumulatively with each positive input. When a neuron membrane potential exceeds a predefined threshold, it generates an output spike, effectively encoding the convolution result as a discrete temporal event. This parallel neuron array eliminates the need for sequential processing, significantly reducing overall processing latency.

The integration of the bounding box optimization with the parallel IF neuron array enhances computational efficiency across the processing pipeline. Since convolution is performed only within the spatially bounded active region, only the IF neurons corresponding to those regions receive meaningful input currents, while neurons outside remain idle. This selective activation minimizes unnecessary computation and reduces power consumption by restricting processing to relevant regions of the feature map. Consequently, the bounding box optimization not only accelerates convolution but also streamlines downstream neuron activity, yielding substantial efficiency gains.

5.2.5 Integrate-and-fire Neuron

The IF neuron implementation employs a parameterized design that supports configurable fixed-point arithmetic to balance computational precision with hardware efficiency. The neuron utilizes an $8-$bit fixed-point representation with 7 fractional bits, providing sufficient precision for membrane potential calculations while maintaining manageable hardware complexity. As shown in Figure 5.8, the `enable` signal provides selective activation

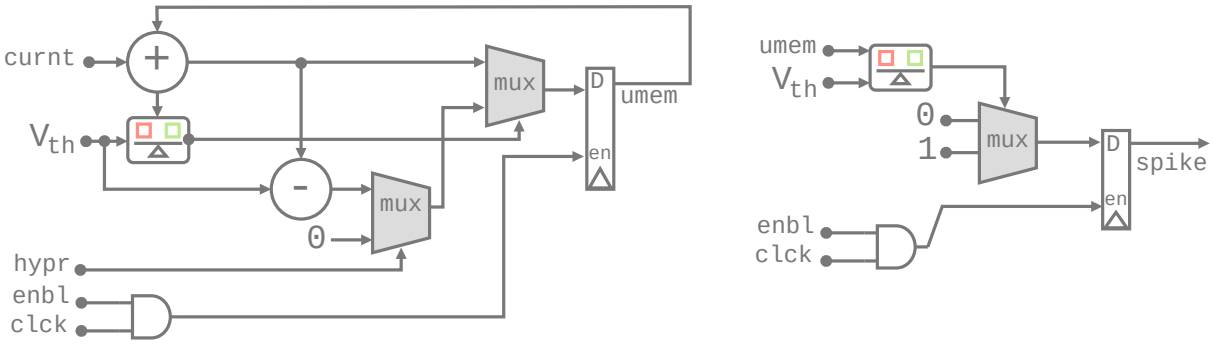


Figure 5.8: Hardware implementation of the IF neuron featuring both hard and soft reset mechanisms.

control, allowing individual neurons to be activated or deactivated based on processing requirements. When enabled, the neuron performs membrane potential integration by accumulating the input current `curr` with the previous membrane potential state. This integration process implements the fundamental IF dynamics, where each input current contributes to the growing membrane potential.

The threshold comparison mechanism forms the core of the spike generation process. When the accumulated membrane potential `u_mem` exceeds the threshold value `v_th`, the neuron generates an output spike and resets its membrane potential according to the specified reset mechanism. The implementation supports two distinct reset strategies controlled by the `hyper` signal. When `hyper` is asserted, the neuron performs a hard reset, setting the membrane potential to zero immediately after spike generation. Alternatively, when `hyper` is deasserted, the neuron implements a soft reset by subtracting the threshold value from the current membrane potential, preserving any excess potential for subsequent processing cycles. This dual reset mechanism provides flexibility in modeling different neural behaviors and optimizing network performance. The hard reset mode mimics the absolute refractory period observed in biological neurons, while the soft reset mode ensures that no membrane potential information is lost during the reset process. The soft reset approach is particularly beneficial for maintaining temporal precision in TTFS encoding schemes, where small potential differences can significantly impact spike timing.

5.2.6 STDP-based Hebbian Learning Rule

After feeding the IF neurons with the convolution outputs at each cycle, any neuron stores its output in a ring buffer for real-time max pooling using two $2-$bit OR gates. The max-pooled result is then written into a 14×14 feature buffer. Simultaneously, synaptic weights

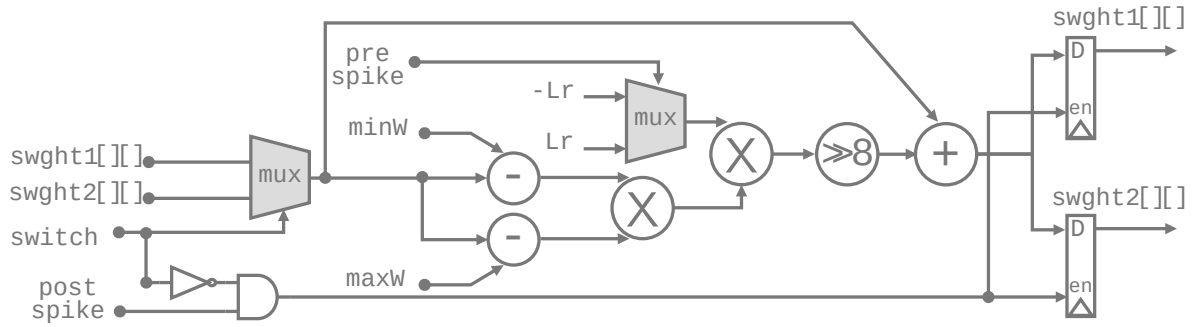


Figure 5.9: Hardware implementation of STDP-based Hebbian learning for real-time synaptic weight update.

are updated using an STDP-based Hebbian learning rule (4.4). Figure 5.9 illustrates the hardware implementation of the STDP-based Hebbian learning mechanism. The update logic is activated only during training, when the `inference` signal is low. It monitors post-synaptic spiking activity within two spike maps corresponding to the first and second convolutional layers at the relevant neuron coordinates.

Upon detecting a post-synaptic spike, the module initiates a weight update loop over a 3×3 convolution kernel window. If the corresponding pre-synaptic neuron was active, the associated synaptic weight is updated using the following rule:

$$\Delta W_{ij} = A \cdot (W_{ij} - W_{\min}) \cdot (W_{\max} - W_{ij}), \tag{5.1}$$

where A is the learning rate, and $W_{\min}$ and $W_{\max}$ define the allowed weight range.

The source of pre-synaptic activity depends on the control signal `switch`. When `switch` is low, the encoded spike buffer `enc_buff` is used to update weights in `swght1`. Otherwise, the second-layer feature map `mxpol1` is used to update `swght2`. This learning mechanism enables local, online, and unsupervised weight updates within the convolutional layers, efficiently leveraging the temporal correlation between pre- and post-synaptic spikes.

Need to mention that all values in this computation are represented in fixed-point format. Therefore, to restore the final product to the intended Q-format, the result is arithmetically shifted right by the number of fractional bits. This truncated value represents $\boldsymbol{\Delta W_{ij}}$, the computed weight change in fixed-point format, ready to be applied to the original weight. The delta is then added to the current weight to produce the new synaptic weight. To ensure that the updated value remains within valid bounds, optional saturation logic may be employed to clip the result to $\boldsymbol{W_{\min}}$ or $\boldsymbol{W_{\max}}$ if necessary. Although this rule promotes weight stability, it involves addition, two subtractions, two multiplication

operations, and a right-shift operation, making it computationally expensive in terms of both logic resources and energy consumption. This module thus represents one of the most power-intensive blocks in the design due to the fixed-point arithmetic and the frequency of updates during training.

5.2.7 SNN Layer Design and Implementation

The proposed SNN architecture comprises two convolutional layers that serve as the feature extraction stage, followed by two fully connected layers forming the classification stage. The convolutional layers extract spike-based feature maps, which are subsequently processed by the classification layers to predict the output class. The classification network includes 1024 hidden neurons and 10 output neurons, corresponding to the 10 classes in the MNIST dataset. The hidden-layer IF neurons are supported by a centralized weight bank that stores synaptic weights between the input and hidden layers. In contrast, the output-layer IF neurons are equipped with AER encoders, adders, and multiple sub weight banks to manage the connections between the hidden and output layers with cummulative weight.

Each hidden-layer neuron is equipped with a dedicated weight storage, instantiated with 64 weighted values corresponds to the total number of pixels in the max pooling layer. When a max-pooled pixel generates a spike, the corresponding weights are fetched from the weight bank for all hidden neurons. These neurons then receive their respective weighted inputs and integrate them into their membrane potentials. A neuron generates a spike when its membrane potential exceeds the threshold. Since the max-pooling module operates sequentially, each hidden neuron receives only one spike input at a time. Consequently, the weight bank fetches one weight per cycle, eliminating the need for AER encoding in this layer because no temporal conflicts between spike inputs occur. However, hidden neurons may spike simultaneously in response to their weighted input. Therefore, a mechanism is required to handle concurrent spike events generated by multiple neurons, ensuring proper transmission and processing at the output stage.

5.2.8 Address Event Representation Encoding

Due to the possibility of simultaneous spike generation in the hidden layer, an AER encoding scheme is employed between the hidden and output layers. In contrast, direct wiring can be used between the input and hidden layers, as the input data is sequential and the input neurons do not produce simultaneous spikes. However, implementing direct wiring in hardware remains costly in terms of power, area, and circuit complexity. Figure 5.10

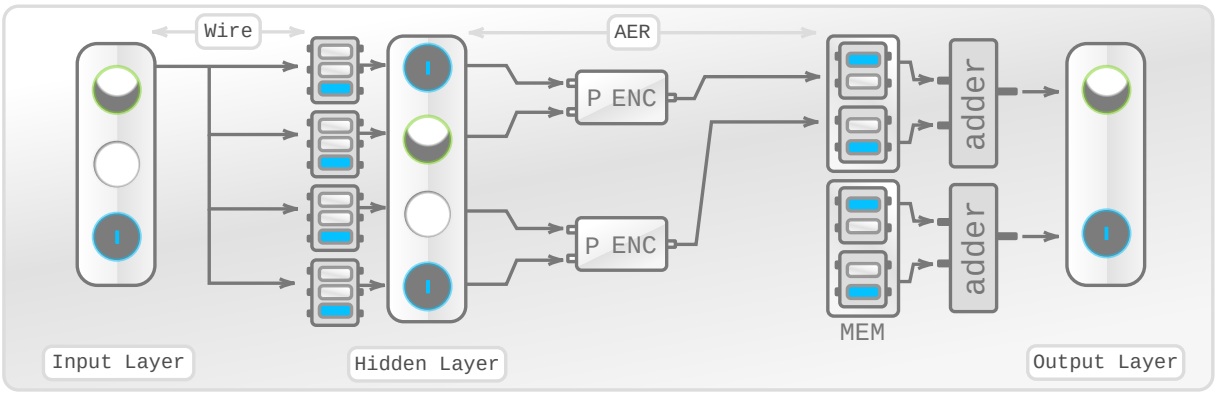


Figure 5.10: Memory organization and communication schemes between layers: direct wire and AER.

shows an example with three input neurons and four hidden neurons, along with their associated memory organization and communication schemes. In this configuration, each hidden neuron has a memory block containing three entries, corresponding to the three input neurons. Similarly, for two output neurons, there are two memory blocks, each containing four entries corresponding to the four hidden neurons.

When the input layer generates a spike, the corresponding weight is fetched from each hidden neuron memory and sent directly to the neuron. However, in the case of AER communication from the hidden to the output layer, if multiple hidden neurons generate spikes simultaneously, a priority encoder arbitrates one spike at a time. The corresponding weights are then fetched from each cluster of output neuron memory. This process is repeated for the remaining spikes, with the fetched weights summed using an adder. The resulting accumulated weight is then delivered to the output neurons.

As illustrated in Figure 5.11, the shared AER module comprises 64 parallel high-priority encoders. Each encoder monitors a cluster of 16 consecutive hidden-layer neurons and arbitrates at most one spike per cluster during each cycle. The architecture is fully parameterized and supports recursive expansion to finer granularities down to 2 neurons per encoder allowing up to 512 parallel priority encoders. Each output neuron is connected to all 64 shared encoders through a banked weight storage system, which distributes synaptic weights across multiple smaller block RAM modules rather than relying on a large uniform memory. Weight processing employs a dual-phase summation scheme to handle $8-$bit weights using $4-$bit datapaths: the lower 4 LSB bits and upper 4 MSB bits are processed sequentially using a faster internal clock to meet timing constraints without stalling throughput. Each pair of sub-weight banks is connected to a dedicated adder that computes the partial sum of synaptic weights associated with the triggered spikes. These

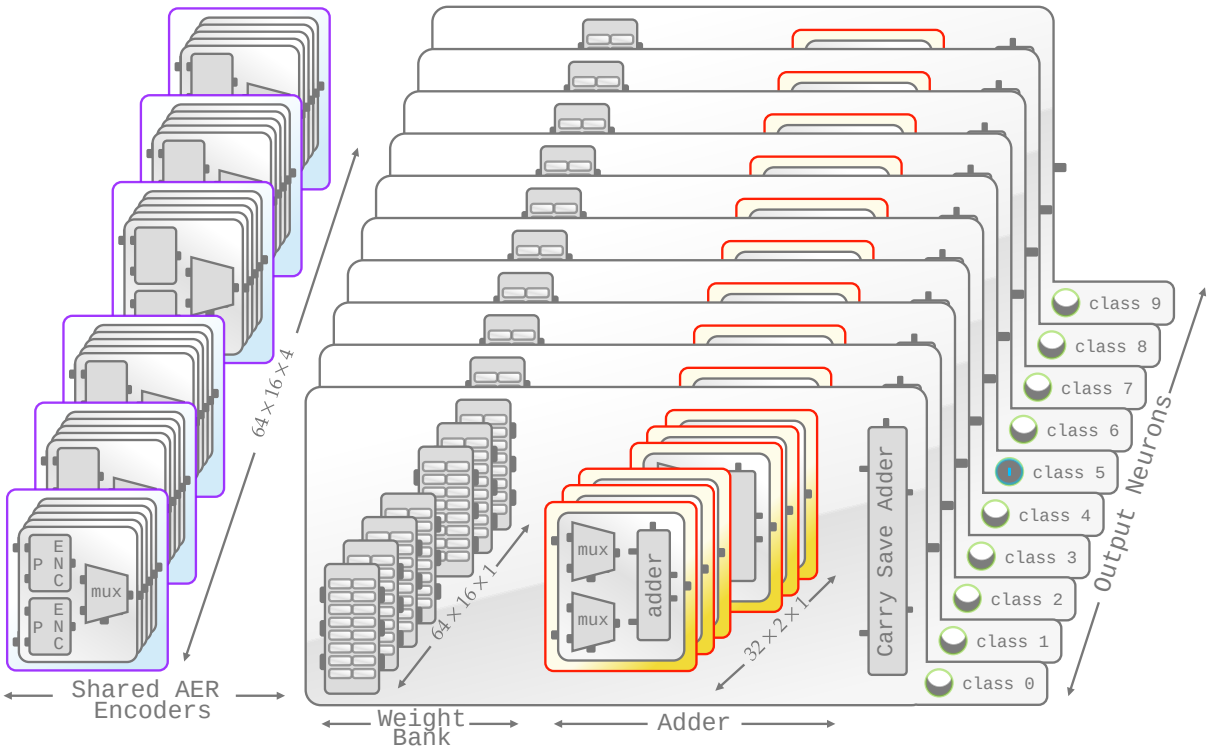


Figure 5.11: Hardware implementation of AER and weight distribution.

partial sums are forwarded to the output-layer neurons, where each neuron includes an adder tree that aggregates the received weights to compute the cumulative input current. The membrane potential of each output neuron is updated based on this cumulative current, and if it exceeds a predefined threshold, the neuron generates a spike, indicating a prediction event.

This parallel encoding and summation approach ensures precise membrane potential updates while maintaining spike-timing fidelity. By replacing dense, fully connected wiring with a shared AER bus and localized computation units, the proposed architecture significantly reduces routing complexity and chip area. The primary trade-off is the additional circuitry required for implementing the localized adders associated with each encoder cluster and output neuron.

Chapter 6

Experimental Results and Hardware Implementation

This chapter presents the evaluation results of the proposed STDG training algorithm on the MNIST dataset and other benchmarks, followed by the hardware implementation outcomes of the designed architecture. The evaluation aims to assess the classification performance of the algorithm and demonstrate its practical feasibility on digital hardware platforms. The first section details the classification accuracy achieved on the MNIST and other benchmark datasets using three configurations of the proposed STDG algorithm: a fully connected network, a binarized fully connected network, and a binarized convolutional fully connected network. The second section outlines the synthesis results, including area utilization, power consumption, and timing performance of the hardware design, highlighting the trade-offs between accuracy, efficiency, and resource utilization. Collectively, these results validate both the effectiveness and hardware implementability of the proposed approach for neuromorphic computing applications.

6.1 Benchmarking

To evaluate the effectiveness, robustness, and temporal efficiency of the proposed learning framework, we conduct experiments on two widely used benchmark datasets: Caltech101 Face/Motorbike [34] and MNIST [122]. These datasets present distinct challenges in visual complexity and variability, enabling a comprehensive assessment of the model classification performance and response latency.

6.1.1 Caltech101 Dataset

The publicly available Caltech101 Face/Motorbike [34] dataset contains a diverse set of images featuring faces and motorbikes. To enhance generalization, the dataset is augmented by randomly flipping images horizontally, adding salt-and-pepper noise, and applying random rotations within a $\pm 5°$ range. These augmentation techniques expand the original 436 face and 799 motorbike images to 3600 samples per class. All images are converted to grayscale and resized to 28×50 pixels. For evaluation, 200 images from each class are randomly selected for testing, while the remainder are used for training.

6.1.2 MNIST Dataset

The MNIST [35] dataset is a standard benchmark, consisting of $70,000$ grayscale images of handwritten digits $(0-9)$, each with a resolution of 28×28 pixels. The dataset is divided into $60,000$ training and $10,000$ testing samples.

6.1.3 Fashion-MNIST Dataset

The F-MNIST [36] dataset consists of $70,000$ grayscale images, each with a resolution of 28×28 pixels. It includes 10 distinct classes representing various clothing items: `T-shirt/Top`, `Trouser`, `Pullover`, `Dress`, `Coat`, `Sandal`, `Shirt`, `Sneaker`, `Bag`, and `Ankle Boot`. Compared to the original MNIST dataset, F-MNIST poses a greater challenge due to the increased complexity and variability in shape, texture, and color intensity of the clothing items. As a result, it has become a widely adopted benchmark for evaluating the performance and generalization capabilities of image classification models.

6.2 Fully Connected STDG Model

The high-level network architecture of the first configuration of the proposed STDG algorithm is illustrated in Figure 4.5. The input layer employs the TTFS scheme to encode input data into a spike train, which is then propagated through the network. At the output, the actual firing times of neurons are compared with the target firing times to compute temporal errors to update synaptic weights using temporal error BP.

The performance evaluation of this configuration was conducted on the MNIST and Caltech101 datasets, following the testing protocols adopted in comparable prior works. In

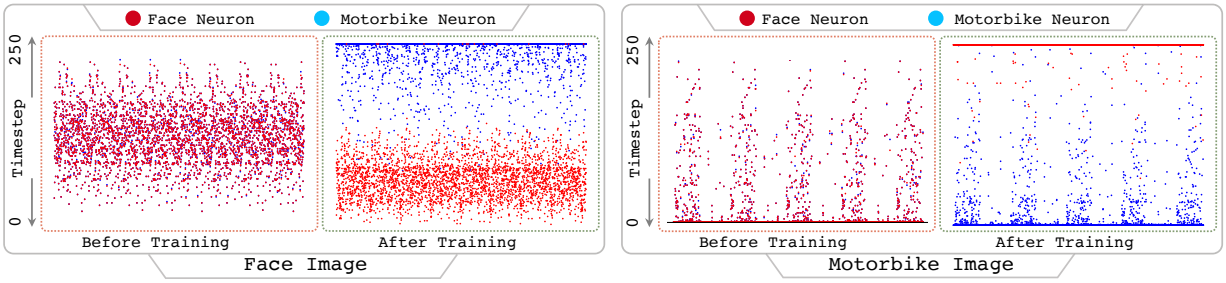


Figure 6.1: Firing times of two output neurons in response to Face/Motorbike images at different stages.

this setup, the synaptic weights in each layer were initialized using a uniform distribution, with value ranges customized to each dataset.

6.2.1 Caltech101 Dataset Result

Figure 6.1 illustrates the firing times of the output neurons responsible for detecting faces and motorbikes in both training and validation samples. Before training, the distributions of the firing times of both output neurons are widely scattered, resulting in poor classification accuracy. However, as the learning phase proceeds and the network gradually learns to solve the task, the firing times of the correct output neuron tend to occur earlier, leading to an improvement in accuracy. For the Caltech101 dataset, STDG achieved an accuracy of 99.5%, outperforming existing SNN-based methods. STDG employed a three-layer architecture of $1400 - 200 - 2$ neurons, initialized with weights drawn from two uniform ranges, namely $[0, 1]$ and $[0, 37]$, and used a constant threshold of 100 for both layers. The hyperparameters $\boldsymbol{\gamma = 17}$, $\boldsymbol{z = 0.5}$, and $\boldsymbol{\lambda = 1 \times 10^{-5}}$ were found to be optimal for this dataset. As mentioned earlier, $\boldsymbol{\gamma}$ represents the minimum required temporal gap between the firing time of the correct output neuron and all other output neurons, $\boldsymbol{z}$ is the learning rate, and $\boldsymbol{\lambda}$ denotes the regularization coefficient applied to each synaptic weight to prevent overfitting and promote sparsity.

6.2.2 MNIST Dataset Results

Figure 6.2 (**A**) illustrates the membrane potential of each output neuron over time during the classification of digit `'9'`. The plot captures the cumulative effect of incoming spikes from timestep 0 to 200. Initially, the membrane potentials exhibit noisy behavior, and the digit image remains indistinct before the 25^{th} timestep. As spike accumulation progresses,

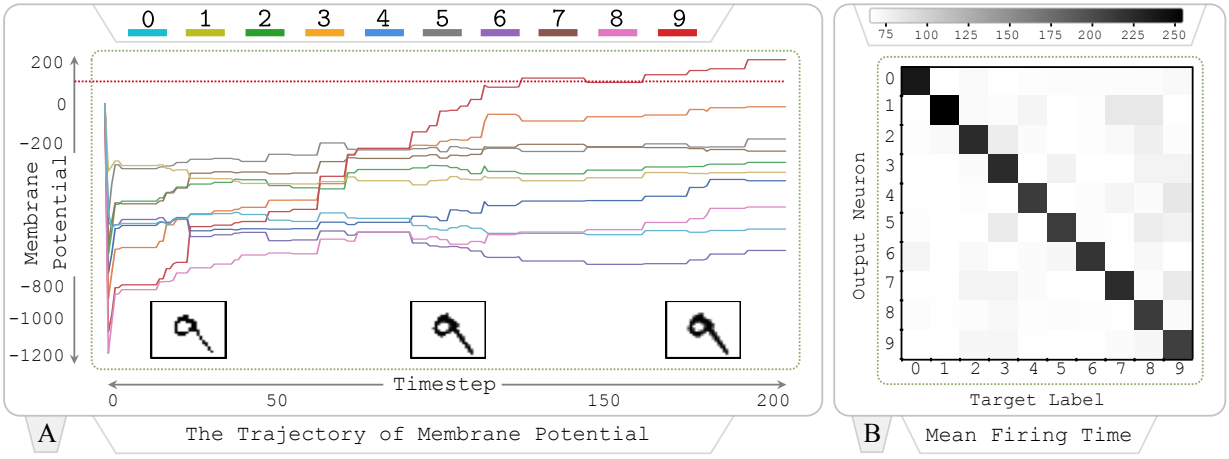


Figure 6.2: (A) Membrane potential evolution for digit '9'; (B) Mean firing time for all digit classes.

the correct neuron crosses the firing threshold of 100 after approximately the 125^{th} timestep, indicating a confident classification.

Figure 6.2 (**B**) presents the mean firing times of output neurons across all digit classes. The correct output neuron for each class consistently fires earlier than the others. This pattern is evident from the grayscale colormap, where the intensity reflects the difference in average firing time between the target neuron and the rest. The STDG algorithm reliably selects the first-spiking output neuron as the prediction, enabling early and accurate classification.

On the MNIST dataset, the proposed STDG method achieved a competitive classification accuracy of 98.2%, surpassing several well-known temporal BP-based SNNs. For instance, Bp-STDP [123] attained 96.6%, while S4NN [32] and Comsa [124] reported 97.4% and 97.9% respectively. Although the method by Zhang [69] achieved a slightly higher accuracy of 98.4%, it relied on a larger network with 800 hidden neurons. In contrast, STDG used a more compact architecture of $784 - 600 - 10$, offering a good trade-off between accuracy and complexity. The network was trained using temporal coding and BP through time with hyperparameters set as $\boldsymbol{\gamma = 3}$, $\boldsymbol{z = 0.5}$, and $\boldsymbol{\lambda = 5 \times 10^{-7}}$.

6.2.3 Model Accuracy Result

Table 6.1 summarizes the key experimental configurations and corresponding classification results of the proposed STDG algorithm on both benchmark datasets. The algorithm achieves a classification accuracy of 99.5% on Caltech101 and 98.2% on the MNIST dataset,

Table 6.1: Comparison of STDG Settings and Performance

Dataset	Model	Structure	Coding	Learning	Initial Weights	Threshold	γ, z, λ	Accuracy
MNIST	[123]	784-1000-10	Rate	STDP-based BP	–	–	–	96.6%
	[32]	784-600-10	Temporal	Temporal BP	–	–	–	97.4%
	[124]	784-340-10	Temporal	Temporal BP	–	–	–	97.9%
	[69]	784-800-10	Temporal	Temporal BP	–	–	–	98.4%
	STDG	784-600-10	Temporal	Temporal BP	[0, 5], [0, 48]	[100], [100]	3, 0.5, 5×10^{-7}	**98.2**%
Caltech101	[80]	–	–	STDP	–	–	–	99.1%
	[32]	–	–	Temporal BP	–	–	–	99.2%
	[125]	–	–	STDP-based	–	–	–	99.3%
	STDG	1400-200-2	Temporal	Temporal BP	[0, 1], [0, 37]	[100], [100]	17, 0.5, 1×10^{-5}	**99.5**%

surpassing prior models under comparable experimental setups and reaching performance levels on par with state-of-the-art approaches.

6.3 Binarized Fully Connected STDG Model

The BSTDG model [2] adopts the same overall architecture as the original configuration illustrated in Figure 4.5. The key distinction lies in the use of binarized synaptic weights during the forward pass, which enhances inference efficiency and reduces computational overhead. Despite this change, the backward pass remains identical to the original model, utilizing full-precision gradients for learning.

This configuration employs two sets of weights: binarized weights, denoted by $\boldsymbol{\beta}$, and full-precision weights, represented by $\boldsymbol{w}$, where $\boldsymbol{\beta} = \text{sign}(\boldsymbol{w})$. During the forward pass, each neuron membrane potential is computed using $\boldsymbol{\beta}$, enabling efficient spike-based inference. In the backward pass, the temporal error BP algorithm updates the full-precision weights $\boldsymbol{w}$, and then the sign function is applied to obtain the updated binarized weights for the next iteration.

We evaluated the performance of this version of STDG on the Caltech101, MNIST, and F-MNIST. In our experiment, we use a single hidden layer and a total simulation time of 256 timesteps. For the Caltech dataset, the number of training samples was fixed at 200, with initial synaptic weights randomly initialized in the ranges $[0, 5]$ for excitatory and $[0, 37]$ for inhibitory synapses. The firing threshold values were set to 50 for excitatory and 25 for inhibitory neurons. The temporal scaling factor $\boldsymbol{\gamma}$ was set to 17, and the weight decay constant $\boldsymbol{\lambda}$ was 1×10^{-5}. For the MNIST dataset, 600 training samples were used, with initial excitatory and inhibitory weights drawn from $[0, 5]$ and $[0, 48]$, respectively.

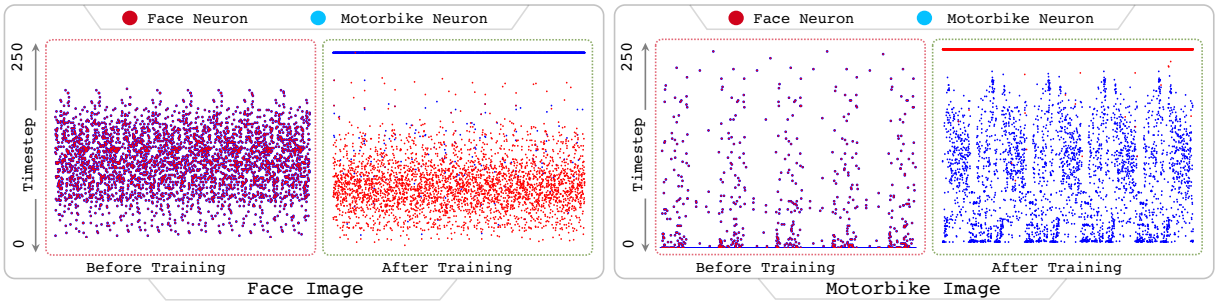


Figure 6.3: Firing times of two output neurons in response to Face/Motorbike images at different stages.

Thresholds were set to 25 and 50, $\boldsymbol{\gamma}$ was set to 3, and $\boldsymbol{\lambda}$ was 5×10^{-9}. Finally, for the F-MNIST dataset, the number of training samples was also set to 600, with both excitatory and inhibitory weights initialized in the range $[0, 1]$. Threshold values were fixed at 70 and 90, $\boldsymbol{\gamma}$ was set to 8, and $\boldsymbol{\lambda}$ was 5×10^{-8}.

6.3.1 Caltech101 Dataset Result

Figure 6.3 illustrates the firing times of the output neurons responsible for detecting face and motorbike classes in both training and validation samples. Before training, the firing times of the two output neurons are widely scattered, as shown in Figure 6.3, resulting in low classification accuracy. As training progresses and the network learns the task, the correct output neuron begins to fire earlier, creating a clearer temporal separation and improving classification performance.

Furthermore, as summarized in Table 6.2, the BSTDG model, using a fully connected architecture of $784-200-10$ and binarized synaptic weights, achieves an impressive 99.0% accuracy on the test set. Compared to previously proposed models, the performance of the BSTDG remains highly competitive, validating its effectiveness even with reduced weight precision.

6.3.2 MNIST Dataset Results

Figure 6.4 (**A**) presents the mean firing times of the output neurons with the correct labels for both BS4NN [3] and the BSTDG model. During simulation, inference terminates as soon as the first spike is emitted in the output layer. Notably, the BSTDG achieves a shorter average response time of 107.46 timesteps, compared to 112.9 timesteps for BS4NN. Ideally,

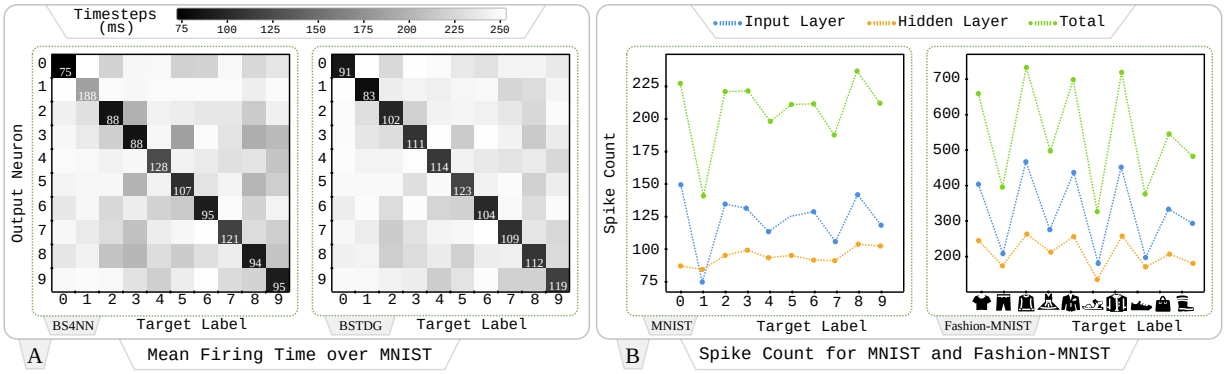


Figure 6.4: (A) Mean firing times between BSTDG and BS4NN [3] over MNIST, (B) Spike count for MNIST and F-MNIST.

the diagonal entries in the heatmap should appear darker, indicating that the correct output neuron fires first. However, the results show that BS4NN struggles particularly with digit '1', requiring nearly 100 additional timesteps to reach a decision. It also displays confusion between similar-looking digits such as '3' vs. '2' and '5' vs. '3'. In contrast, the BSTDG exhibits consistently early firing across all digit categories, as indicated by the more uniform and diagonally dominant patterns.

In a subsequent experiment, we measured the average number of spikes required for classification, as shown in Figure 6.4 (**B**). For all digits except '1', the BSTDG utilizes approximately 105 spikes in the input layer and 80 in the hidden layer. Due to its simpler structure, digit '1' requires fewer spikes—averaging around 150 in total. Although BS4NN achieves a comparable inference latency, it produces approximately 325 spikes across both layers—nearly twice the 180 spikes required by the BSTDG. These findings underscore the proposed model's efficiency and sparse spike utilization in temporal computation.

Table 6.2 compares the classification accuracies of recent SNN models trained using error BP on the MNIST dataset. After 200 epochs, the BSTDG reaches an accuracy of 97.3%, representing a 0.3% improvement over BS4NN. Additionally, the BSTDG, despite using quantized weights, significantly outperforms S4NN [123], which operates with full-precision parameters. A binarized convolutional SNN model [126] achieves a competitive accuracy of 99.1% using direct current input and threshold adaptation; however, it is not a single-spike model. To extend our evaluation, we implement a convolutional variant of the BSTDG with architecture 784–32C5–P2–800–10. This model achieves 98.7% accuracy, a 0.5% drop compared to state-of-the-art full-precision, latency-coded deep networks [127, 128]. Despite using only a single convolutional layer with binarized kernels and weights, the results remain competitive and demonstrate the strength of the proposed approach.

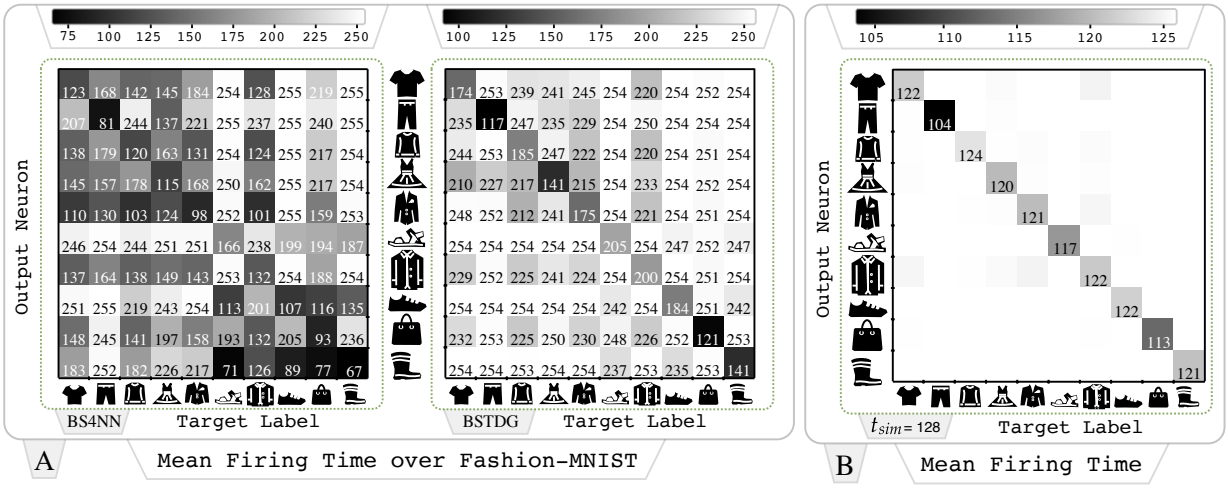


Figure 6.5: Mean firing times of the output neurons over (A) Fashion-MNIST, (B) Convolutional BSTDG at $t_{sim} = 128$.

6.3.3 Fashion-MNIST Dataset Results

Figure 6.5 (**A**) compares the mean firing times of output neurons across F-MNIST categories for both BS4NN [3] and the BSTDG model. Compared to the MNIST dataset, the response times are notably longer in this benchmark. On average, the BSTDG requires more than 100 timesteps for a neuron to spike. Among all categories, `Bag` and `Trouser` are predicted most efficiently, while categories like `Shirt` and `Sandal` demand over 200 timesteps on average. This observation is reasonable, as `Shirts` often share visual similarities with `Pullovers`, `T-shirts`, or `Coats`, making classification more challenging and time-consuming.

Despite the relatively high latency, the neurons corresponding to the correct class consistently exhibit the earliest firing times within their respective columns, indicated by the darker diagonal patterns in the heatmap. In contrast, although BS4NN can respond in as few as 67 timesteps, its heatmap lacks clear diagonality, reflecting poor class separation. BS4NN frequently confuses `Pullovers` with `Coats` and `Dresses`, and `Sandals` with `Ankle Boots`, as well as `Bags` with `Sneakers`, due to overlapping firing times. These findings suggest that while BS4NN offers faster responses, its classification performance is inferior to that of BSTDG.

Figure 6.5 (**B**) shows the results for the convolutional version of the BSTDG. The addition of feature extraction layers enables more consistent performance across all categories. In this case, neurons associated with the correct label typically fire within 120 timesteps,

Table 6.2: Configuration and performance comparison of STDG and BSTDG across different datasets.

Dataset	Model	Structure	Coding	Neuron	Learning Method	Accuracy (%)
MNIST	[123]	784-1000-10	Rate	IF	STDP-based BP	96.6
	BANN [3]	784-600-10	N/A	BSigmoid	BP with Adam	96.8
	BS4NN [3]	784-600-10	Temporal	IF	Temporal BP	97.0
	[129]	784-800-10	Temporal	IF	Temporal BP	97.2
	S4NN [32]	784-600-10	Temporal	IF	Temporal BP	97.4
	[128]	784-40C5-P2-1000-10	Temporal	IF	Temporal BP	99.2
	[127]	784-32C5-P2-16C5-P2-10	Temporal	IF	Temporal BP	99.3
	[126]	784-16C5-P2-64C5-P2-1024-10	–	Leaky-IF	BP	99.1
	BSTDG	784-600-10	Temporal	IF	Temporal BP	97.3
		784-32C5-P2-800-10			STDP + Temporal BP	**98.7**
F-MNIST	[130]	784-6400-10	Rate	Leaky-IF	Dopamine STDP	85.3
	BANN [3]	784-1000-10	N/A	BSigmoid	BP with Adam	86.4
	BS4NN [3]	784-1000-10	Temporal	IF	Temporal BP	87.3
	[131]	28×28-32C3-32C3-P2-128-10	Rate	Leaky-IF	Rate BP	89.0
	[132]	784-400-400-10	Rate	Leaky-IF	Spike-sequence BP	89.5
	[68]	28×28-16C5-P2-32C5-P2-800-128-10	Temporal	IF	Temporal BP	90.1
	[126]	784-16C5-P2-64C5-P2-1024-10	–	Leaky-IF	BP	87.0
	BSTDG	784-1000-10	Temporal	IF	Temporal BP	87.5
		784-32C5-P2-1000-10			STDP + Temporal BP	**90.1**
Caltech101	[81]					98.2
	[133]					99.2
	[125]					99.3
	BSTDG					**99.0**

while incorrect neurons rarely reach the firing threshold. Since the training simulation time is limited to 128 timesteps, further extending t_{sim} could potentially enhance performance even more.

6.3.4 Model Accuracy Result

Table 6.2 presents a comprehensive comparison of the proposed BSTDG learning method across multiple benchmark datasets, including Caltech, MNIST, and F-MNIST. The results demonstrate that BSTDG achieves competitive or superior performance compared to existing state-of-the-art models, reaching 99.0% on Caltech, 98.7% on MNIST, and 90.1% on F-MNIST.

6.4 Convolutional Fully Connected STDG Model

The high-level architecture of the BSTDPG model as another version of STDG is depicted in Figure 5.1. The input layer utilizes the TTFS scheme to encode input data into spike trains, which are then passed through two convolutional layers before propagating through the fully connected SNN. At the output layer, the actual firing times of neurons are compared against target firing times to compute temporal errors, which are used to update synaptic weights via a gradient-based learning rule.

This configuration was evaluated on the MNIST and F-MNIST datasets, with the simulation time $\boldsymbol{t_{sim}}$ fixed at 128 timesteps. We evaluated the performance of the BSTDPG model under three structural configurations.

– **One Spiking Convolution Layer with STDP and SVM Classifier:** The first variant of convolutional STDG adopts a shallow architecture consisting of a single convolutional layer with 32 filters of size 5×5, followed by a 2×2 max-pooling layer. Designed for low computational overhead and biological plausibility, this compact model employs local STDP learning in the convolutional layer to extract meaningful spatiotemporal features from temporally encoded inputs. These features are subsequently classified using an external SVM [134].

– **Two Spiking Convolution Layers with STDP and SVM Classifier:** The second variant increases architectural depth by stacking two convolutional layers, the first with 64 filters of size 5×5, and the second with 128 filters of size 2×2, each followed by a pooling layer. Both convolutional layers are trained using local unsupervised STDP, which enhances the network capacity for hierarchical spatiotemporal feature extraction. As in the shallow model, an external SVM is used for final classification, leveraging the more expressive features developed by the deeper structure.

– **Two Spiking Convolution Layers with STDP and Gradient-based Classifier:** The third variant extends the second variant by incorporating two fully connected layers: a hidden layer with 800 neurons for MNIST and 1000 neurons for F-MNIST and ten neurons in output layer. While the convolutional layers retain local BSTDPG for unsupervised learning, the fully connected layers are trained using temporal BP. This hybrid learning scheme effectively combines local unsupervised feature extraction with global supervised refinement of spike timing, leading to improved classification performance without compromising training efficiency.

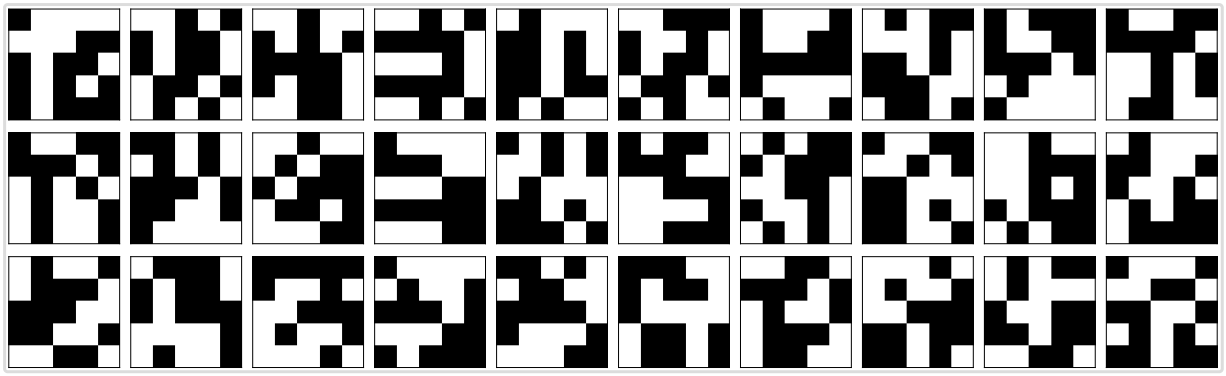


Figure 6.6: Learned convolutional kernels from selected feature maps after 15 training epochs, captured after the pooling layer.

Before the training phase begins, the synaptic weights of the network are initialized using a normal distribution with a mean of 1 and a variance of 0.1, introducing mild stochasticity while maintaining a consistent initialization baseline. During training, the spiking threshold is set to 15, encouraging selective neuron activation and the learning of discriminative features. In the testing phase, this threshold is reduced to 12, allowing for more permissive spiking and improved recognition accuracy. For the final classification stage, an SVM is employed with a regularization parameter of $\boldsymbol{C} = 0.05$, which mitigates overfitting and enhances the models generalization performance.

Since the weights are initialized from a uniform distribution prior to training, the initial convolution outputs do not produce meaningful feature maps. However, after feeding the convolutional layer with TTFS-encoded spike trains and enabling local STDP learning, the synaptic weights of each kernel are adjusted based on the temporal correlation of pre- and post-synaptic spikes. This unsupervised learning mechanism allows the kernels to specialize and extract discriminative features from the input data. Figure 6.6 displays the learned convolutional kernels for selected feature maps after 15 epochs of training, taken from the layer following pooling.

6.4.1 MNIST Dataset Results

Figure 6.7 illustrates the feature maps extracted from the first variant of the BSTDPG model, highlighting its learning behavior. In the shallow architecture comprising one convolutional layer with the structure of 32C5-P2, the local unsupervised STDP rule enables the network to develop meaningful features even in the absence of explicit supervision.

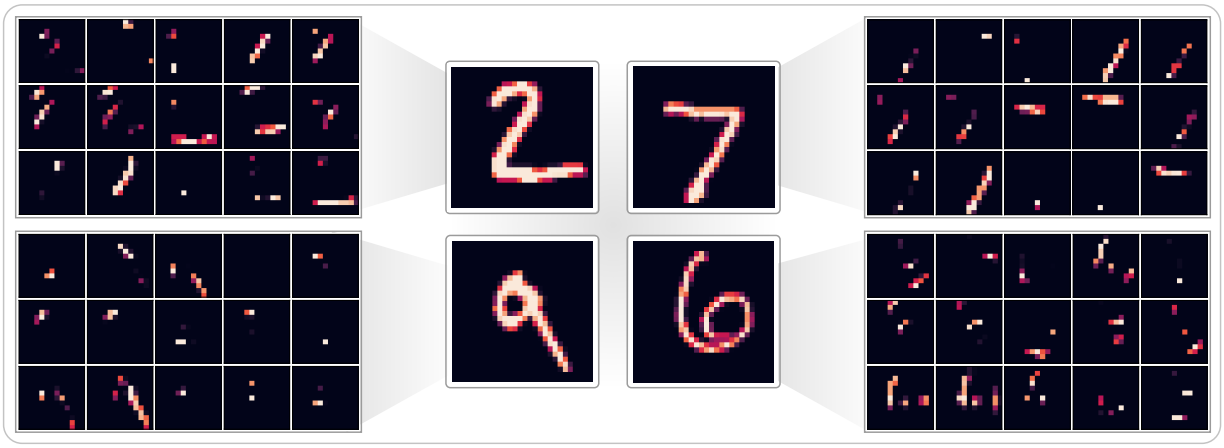


Figure 6.7: Learned features in the first variant of BSTDPG over selected MNIST digits.

The max pooled outputs from selected neurons reveal localized activation patterns that correspond well to structures in the MNIST digits. These features, learned purely through temporal spike correlations, provide robust representations for the downstream SVM classifier, facilitating competitive classification accuracy.

The proposed BSTDPG framework demonstrates competitive, and in many cases superior, performance when compared to state-of-the-art models, including those trained with BP. As listed in Table 6.3, even the shallowest variant of BSTDPG, featuring only a single convolutional layer, achieves an accuracy of 98.3% on the MNIST dataset, surpassing models such as S4NN and Comsa [124], both of which rely on temporal BP. While Kheradpisheh [80] and Gaspard [69] report slightly higher accuracies, 98.4% and 98.6% respectively, these gains come at the cost of greater architectural complexity and computational burden. For instance, Kheradpisheh *et al.* [80] employed Difference of Gaussian (DoG) [135] filters and deeper full-precision networks, while Gaspard *et al.* [69] utilized 70 convolutional filters of size 7×7, resulting in significant memory and compute overhead.

In contrast, the proposed BSTDPG utilizes binarized weights throughout the network, yielding substantial reductions in memory usage and computational complexity. The deeper variant featuring two convolutional and pooling layers achieves an accuracy of 98.8% using only a lightweight SVM classifier, outperforming several full-precision models. Furthermore, extending the architecture with a fully connected layer trained via temporal BP elevates the accuracy to 99.1%, establishing a new benchmark among biologically inspired models employing binarized synapses.

For the deeper architecture with the structure of 64C5-P2-128C2-P2-800-10, the hier-

Table 6.3: Performance Comparison over MNIST and F-MNIST Datasets

Model	Structure	Coding	Learning	Accuracy (%)	
				MNIST	Fashion-MNIST
Bp-STDP [123]	1000-10	Rate	STDP-based BP	96.6	—
S4NN [32]	600-10	Temporal	Temporal BP	97.4	—
Comsa *et al.* [124]	340-10	Temporal	Temporal BP	97.9	—
Kheradpisheh *et al.* [80]	30C5-P2-100C5-P2	Temporal	STDP + SVM	98.4	—
Gaspard *et al.* [69]	70C7-P2	Temporal	VDSP + SVM	98.6	—
Hao *et al.* [130]	6400-10	Rate	Dopamine-mod. STDP	—	85.3
BANN [3]	1000-10	N/A	BP with Adam	—	86.4
BS4NN [3]	1000-10	Temporal	Temporal BP	—	87.3
Ranjan *et al.* [131]	32C3-32C3-P2-128-10	Rate	Rate BP	—	89.0
Zhang *et al.* [132]	784-400-400-10	Rate	Spike-sequence BP	—	89.5
Zhang *et al.* [68]	16C5-P2-32C5-P2-800-128-10	Temporal	Temporal BP	—	90.1
BSTDPG	32C5-P2	Temporal	BSTDP + SVM	98.3	90.0
	64C5-P2-128C2-P2			98.8	90.8
	64C5-P2-128C2-P2-800-10	Temporal	BSTDP + Temporal BP	**99.1**	—
	64C5-P2-128C2-P2-1000-10			—	**91.0**

archically stacked convolutional and pooling layers significantly enrich the extracted representations. The STDP-trained convolutional layers capture increasingly complex and abstract features from the raw inputs, while the temporal BP mechanism in the fully connected layers further refines the decision boundaries. Visualizations of the pooled feature maps from deeper layers demonstrate robust pattern selectivity and spatial invariance, showcasing how the network builds high-level representations in a fully spike-based and largely unsupervised manner.

6.4.2 Fashion-MNIST Dataset Results

Figure 6.8 presents the feature maps extracted from the first variant of BSTDPG model, highlighting its unsupervised learning dynamics. This shallow architecture, comprising a single convolutional layer with the structure of 32C5-P2, leverages local unsupervised STDP to learn meaningful features in the absence of explicit supervision. The max-pooled outputs from selected neurons exhibit localized activation patterns that correspond closely to structural elements in the F-MNIST dataset. These spike-timing-based features provide effective representations for the downstream SVM classifier, supporting competitive classification performance.

On the more challenging F-MNIST dataset, BSTDPG consistently outperforms biologically plausible approaches such as the dopamine-modulated STDP model [130] and

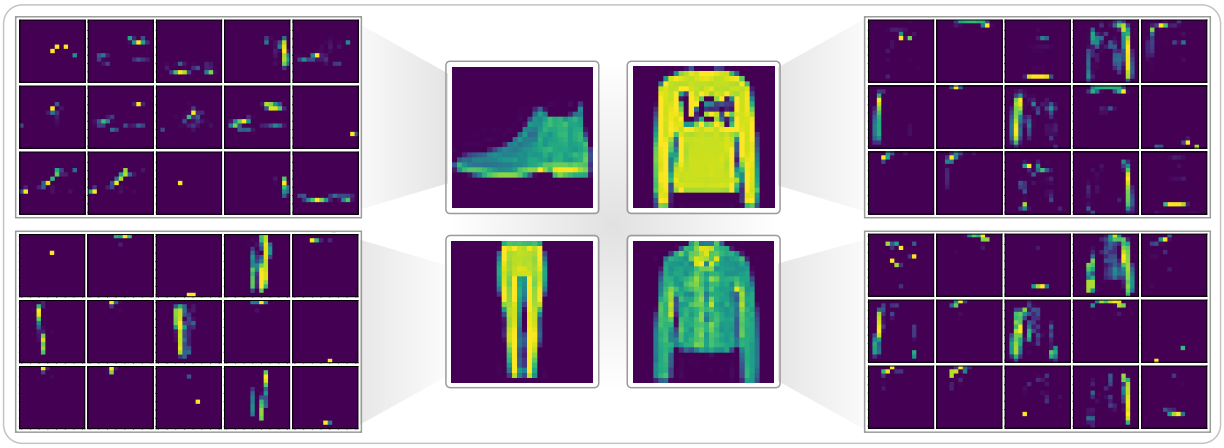


Figure 6.8: Learned features in the first variant of BSTDPG over selected F-MNIST samples.

BS4NN [3], while also surpassing several gradient-based SNN models. The extended BSTDPG model achieves an accuracy of 91.0% on F-MNIST, demonstrating its strong generalization capabilities. These results underscore the effectiveness of combining BSTDPG with temporal coding and confirm the scalability and efficiency of the proposed framework across different visual recognition tasks.

6.4.3 Model Accuracy Result

Table 6.3 summarize the BSTDPG model performance on MNIST and F-MNIST datasets, outperforming other SNN models. Even the simplest BSTDPG model with one convolutional layer achieves strong accuracy with less complexity and lower computational cost compared to other methods that use filters or large convolutional kernels. The deeper BSTDPG versions further improve accuracy, reaching over 99% on MNIST and 91% on F-MNIST. These results confirm the strength of combining unsupervised local BSTDPG learning in convolutional layers with temporal error-driven learning in deeper layers.

6.5 Simulation Results

All hardware modules developed in this work were simulated using Questa Intel FPGA Edition (ModelSim) to verify their functional correctness. This industry-standard HDL simulator enables cycle-accurate waveform analysis, allowing detailed observation of signal

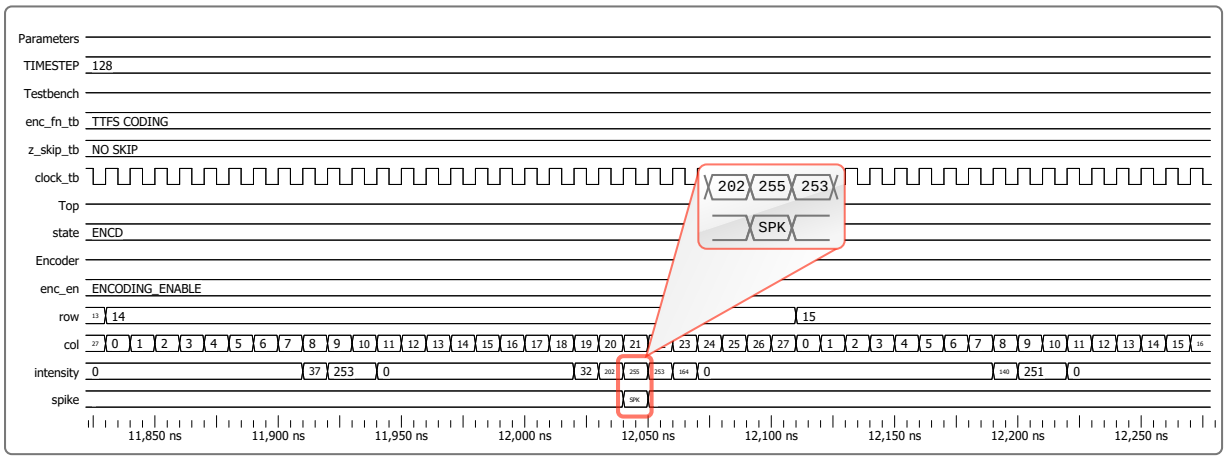


Figure 6.9: Verilog simulation result of the approximated TTFS module.

transitions and timing behavior within each module. The simulation waveforms presented in this section provide visual confirmation that the designs operate as intended under representative input stimuli.

6.5.1 TTFS Spike Encoder Design

As shown in Figure 6.9, the Verilog implementation of the TTFS encoding module was functionally verified. The waveform demonstrates the spike generation process for a single MNIST image sample. In this specific test case, the image includes a pixel with an intensity of 255 at 12,040 ns located at `Pixel[14][21]`, and another with an intensity of 253 at 12,050 ns at `Pixel[14][22]`. According to the TTFS encoding scheme, only a pixel with a maximum intensity of 255 corresponds to the earliest time step in the encoding window. This confirms the correct functionality of the module, wherein high-intensity pixels are translated into earlier spike events, aligning with the time-to-first-spike encoding principle.

6.5.2 Zero Skip Optimization

The bounding box mechanism significantly enhances system performance. First, it improves computational efficiency by narrowing the convolution scope to only the active regions, thereby avoiding unnecessary operations across the full 28×28 input. This results in early termination of convolution loops in inactive areas and reduces memory accesses.

Second, it improves energy efficiency by minimizing activity, which is especially valuable for low-power neuromorphic systems. Latency is also improved, as processing a smaller region allows convolution operations to complete more quickly. The reduced computation window accelerates data propagation through the pipeline and shortens the overall critical path. Performance gains vary with input sparsity. For example, small and centered digits such as digit `'1'`, typically activate a region of just 10×15 pixels, reducing convolution operations at least approximately 75% for a few time steps, and by more than 86% for most timesteps, since many contain only a single spike or no spike across each time step. Even for spatially larger digits such as `'0'`, most timesteps still involve sparse activity, maintaining computational savings throughout the temporal window. These results underscore the efficiency and scalability of the bounding box approach for TTFS-based SNN.

The only limitation of the bounding box optimization arises when spikes are distributed across distant locations, such as the opposite corners of the input image. In such cases, the bounding box must span a large rectangular region, potentially including a significant number of inactive pixels. However, this scenario is rarely encountered in MNIST digit recognition tasks, where the active pixels forming each digit are typically connected or closely clustered. Even in the unlikely event that spike activity appears in widely separated regions, the inherent temporal sparsity of TTFS encoding ensures that most time steps still contain very few spikes. As a result, the computational benefits of the bounding box approach are largely preserved, since the convolution operations remain focused on sparse regions in the temporal domain.

As illustrated in Figure 6.10, an experiment was conducted on an MNIST digit `'0'` sample during its first timestep to evaluate the impact of the `zero-skip` optimization. When the `zero-skip` feature was disabled, the system processed the full 28×28 pixel array regardless of spike activity, resulting in a convolution latency of 7860 nanoseconds. In contrast, enabling the bounding box optimization allowed the system to complete the same convolution stage in just 1710 nanoseconds. This represents a 78.2% reduction in processing time, translating to a 4.6× speedup and a savings of 6150 nanoseconds for that timestep alone. These results highlight the substantial efficiency gains achievable by exploiting the spatial sparsity inherent in TTFS-encoded spike patterns.

6.5.3 Integrate-and-fire Neuron

To validate the behavior of the implemented LIF neuron, which also supports IF mode through parameter control, we conducted functional simulations. The design employs an $8-$bit fixed-point arithmetic format with 7 fractional bits, offering a balance between

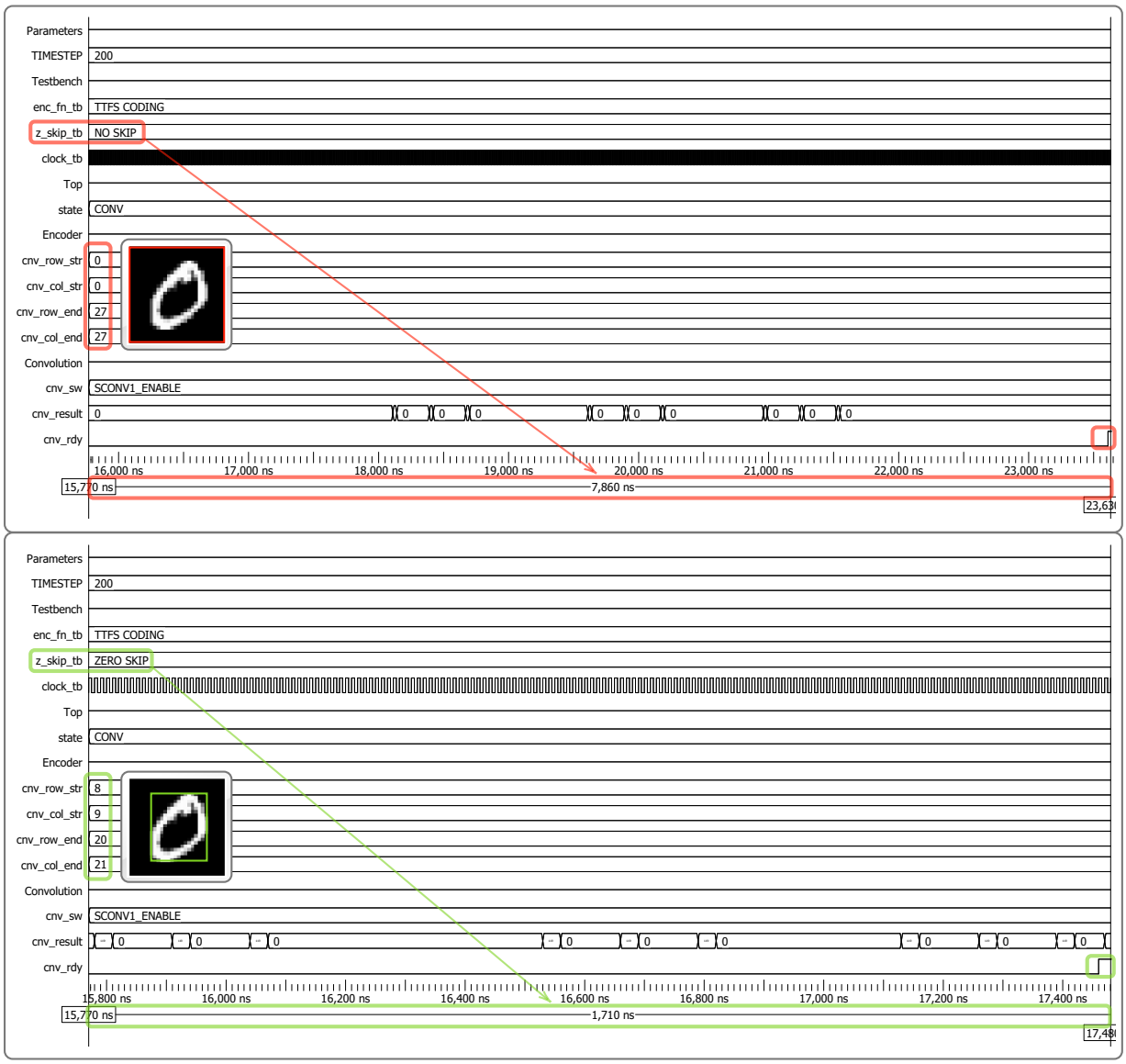


Figure 6.10: Comparison of simulation waveforms for a single timestep with zero-skip feature.

numerical precision and hardware efficiency. The neuron receives input currents over time, and the simulation captures the evolution of the membrane potential along with the spike generation events.

The simulation of the LIF dynamics is shown in Figure 6.11. The neuron integrates the

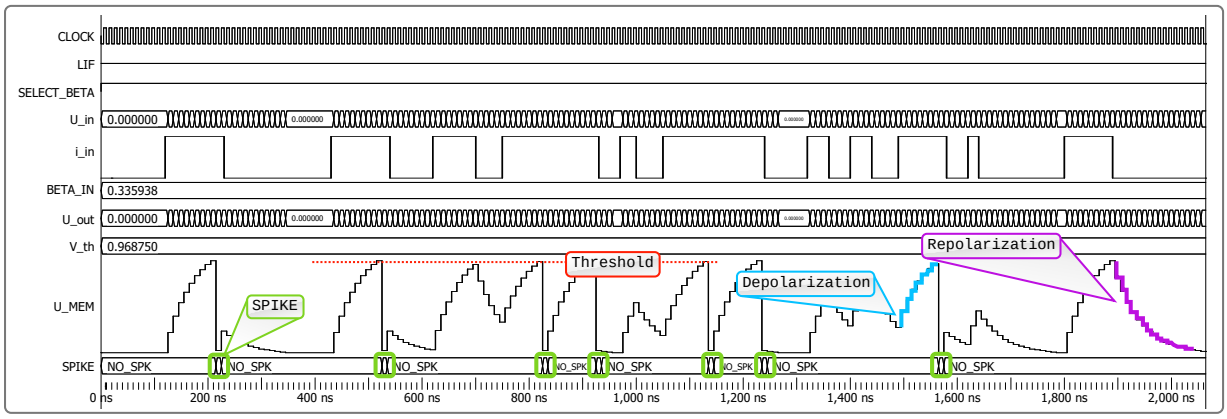


Figure 6.11: Verilog simulation result of the LIF neuron.

incoming current when the `enable` signal is asserted, following the classical IF dynamics. Once the membrane potential `u_mem` surpasses the defined threshold `v_th`, a spike is emitted and the potential is reset based on the selected reset mode. The `hyper` control signal configures the reset mechanism: in hard reset mode, the potential is reset to zero, while in soft reset mode, the threshold is subtracted from the accumulated value. This dual-mode capability allows for flexible modeling of neural behavior and enhances the suitability of the design for spike-timing-dependent tasks. The waveform demonstrates the expected firing pattern and membrane potential trajectory, confirming the correctness of the neuron implementation under randomized input stimuli.

6.5.4 STDP-based Hebbian Learning Rule

To verify the functional correctness of the proposed STDP learning module, we performed simulations using synthetic spike activity across two convolutional layers. As illustrated in Figure 6.12, the simulation demonstrates several synaptic weight updates corresponding to spike timing relationships between pre- and post-synaptic neurons. In the first convolutional layer, multiple updates are observed in `stdp1_w`, indicating that several pairs of spikes satisfy the temporal constraints for STDP. These updates reflect the local plasticity rule, where synaptic weights are potentiated or depressed based on the relative timing of the spikes. In contrast, the second convolutional layer exhibits only a single update event in `stdp2_w`, as only one valid pre-post spike pair occurs within the specified time window. This behavior aligns with the layer activity pattern, where higher-level feature extraction results in more sparse and selective spiking. The simulation results confirm that the STDP

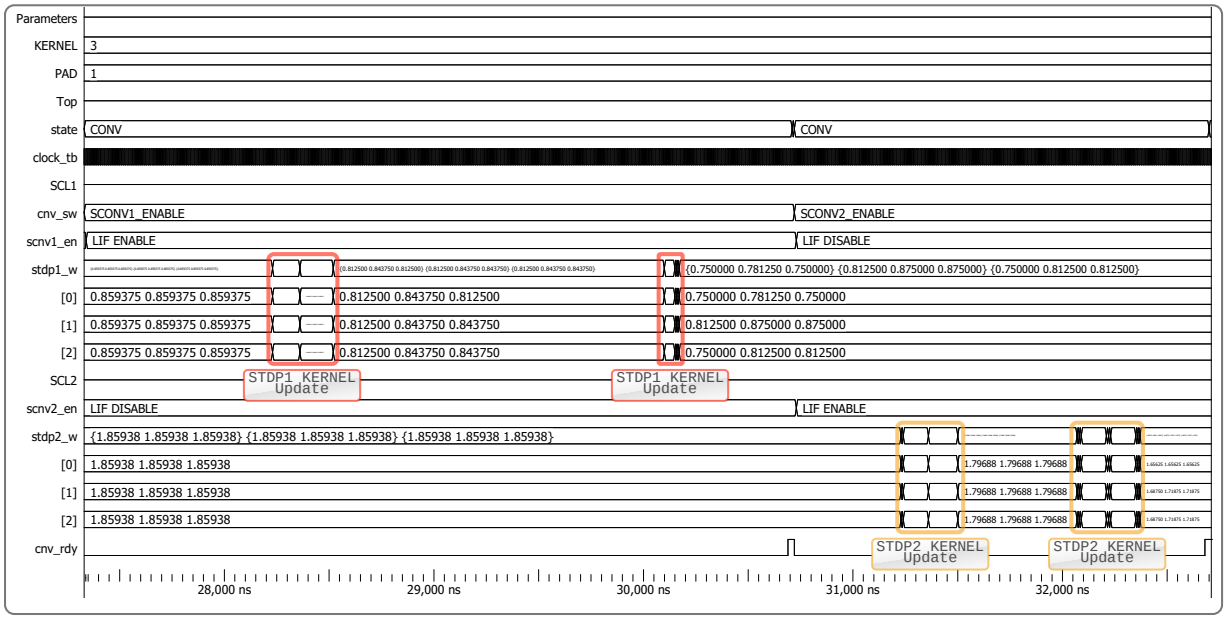


Figure 6.12: Verilog simulation result of the STDP module in both convolutional layers.

module correctly implements layer-wise plasticity, enabling unsupervised feature learning through temporal correlations in spiking activity.

6.5.5 SNN Layer Design and Implementation

To evaluate the inference pipeline of the BSTDPG, we perform simulation using pre-initialized normalized weights across all layers. The architecture consists of two spiking convolutional layers, each followed by a max-pooling layer, resulting in a compact feature map of 7×7 pixels. This flattened output is fed sequentially into the input layer of the SNN. As spikes are generated in the input layer, the AER module identifies the corresponding pre-synaptic address and retrieves the associated synaptic weights from `weight_bank_3`. These weights are then forwarded to the output layer neurons, which perform membrane potential integration based on the incoming spikes. The first neuron to reach the threshold and generate a spike determines the prediction. As shown in Figure 6.13, the network completes inference within a simulation window of 128 timesteps. Without the zero-skip optimization, the prediction is produced at $1,459,240$ ns. However, enabling the zero-skip feature, reduces the inference time to $1,212,270$ ns, demonstrating a significant improvement in runtime efficiency without altering the network structure or accuracy.

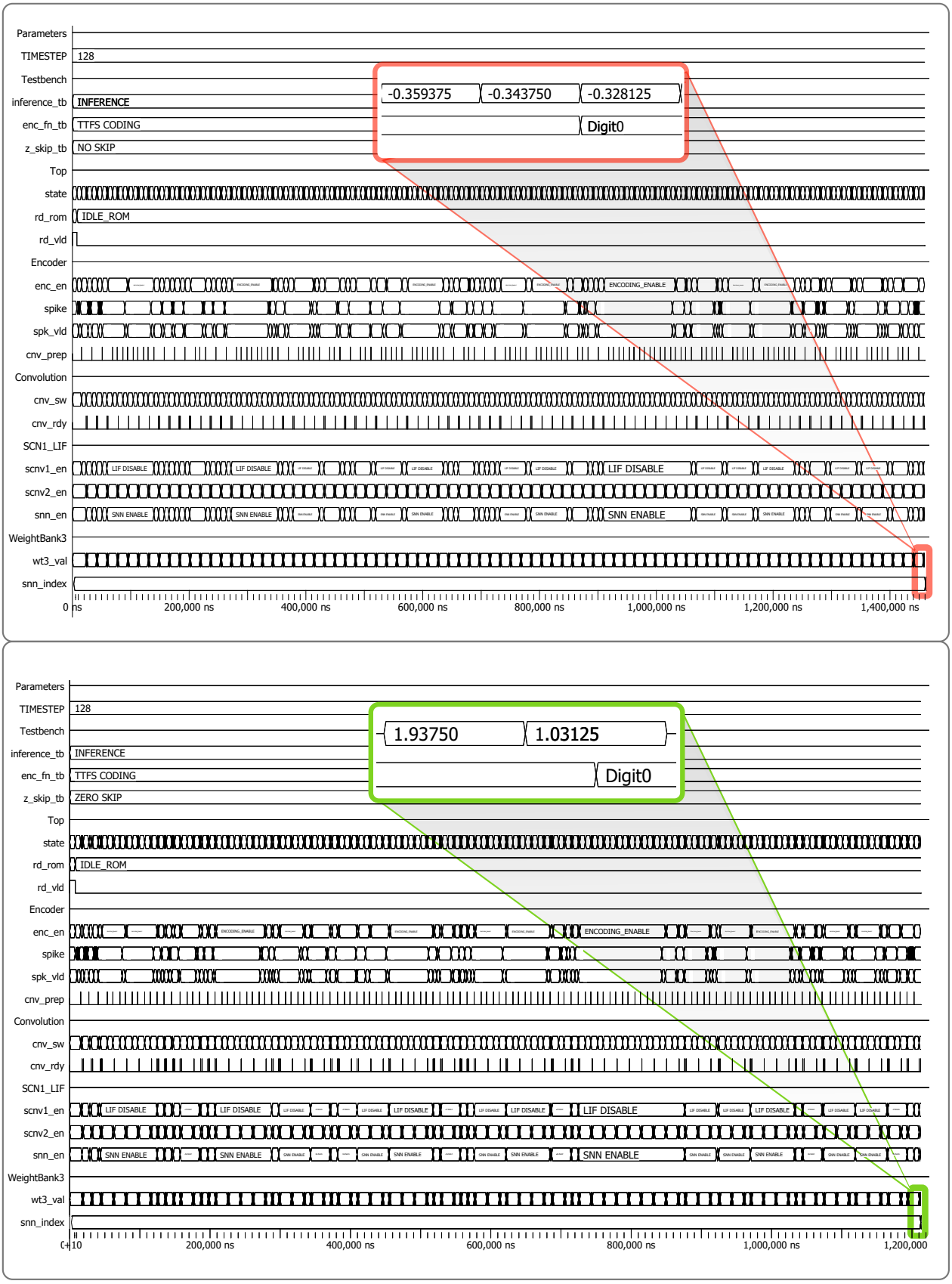


Figure 6.13: Verilog simulation result of the BSTDPG module in both optimization modes.

Table 6.4: Comparison of BSTDPG synthesis results with state-of-the-art works.

Dataset	Model	Accuracy (%)	Neuron/ Synapses	Technology (nm)	Power	Area (mm^2)	Resolution	Clock Frequency
MNIST	[111]	91.7	74 / 17k	40	73 μW	2.68	4-bit	Event-driven
	[9]	96.4	10 / 7840	180	400 μW	43.79	2-bit	Event-driven
	[106]	91.4	10 / 2.5k	28	35–447 μW	0.086	4-bit	75 MHz
	[137]	91.6	2048 / 149k	40	46.6 mW	2.56	2/3-bit	110 MHz
	[136]	97.9	1546 / 666k	10	94 mW	1.7	7-bit	105 MHz
	[138]	97.8	410 / 199k	65	23.6 mW	10.08	>10-bit	100 MHz
	BSTDPG	**99.1**	2223 / 77k	65	13.6 mW	4.89	1-bit	Hybrid–100 MHz

6.6 Accelerator Results

Based on the synthesis results shown in Table 6.4, the proposed BSTDPG accelerator demonstrates competitive performance when compared to state-of-the-art implementations. Synthesized using Cadence Genus in 65nm technology at 100 MHz, the BSTDPG achieves the highest MNIST classification accuracy of 99.1%, surpassing all compared works. The design utilizes $2,223$ neurons and $76,756$ synapses with $1-$bit resolution, operating in a hybrid mode at 100 MHz clock frequency. While the power consumption of 13.66 mW is higher than ultra-low-power designs like Stuijt *et al* [111] and Frenkel *et al* [106], this increased power consumption is attributed to the hybrid clock-driven and event-driven nature of BSTDPG, whereas these ultra-low-power implementations are purely event-driven systems. Despite this higher power consumption, the BSTDPG remains significantly more efficient than higher-capacity processors such as Chen *et al* [136] and offers a favorable trade-off considering the superior accuracy achieved. The area footprint of 4.89 mm^2 positions the BSTDPG in the middle range compared to other implementations, demonstrating reasonable area efficiency while maintaining high classification performance. The 1-bit resolution represents the most aggressive quantization among the compared designs, yet still delivers the best accuracy, highlighting the effectiveness of the proposed architecture design principles.

In summary, the proposed BSTDPG outperforms prior works in MNIST accuracy while maintaining competitive power and area efficiency. The hardware metrics for BSTDPG were obtained from synthesis results generated using Cadence Genus Synthesis Solution targeting a 65 nm standard-cell library. These synthesis-level estimates reflect the architecture hardware cost and efficiency prior to physical design, without routing effects.

Chapter 7

Conclusion and Future Works

This thesis has presented a comprehensive investigation into the development of an efficient, hardware-friendly training method for binarized SNNs, accompanied by the design and implementation of a highly reconfigurable hybrid clock- and event-driven accelerator. The work addresses critical training challenges in neuromorphic computing through the novel BSTDPG framework and its corresponding hardware implementation. The research successfully achieved the three tightly connected primary objectives outlined in section 1.4.

First, we developed a temporal coding-based feature extraction approach that employs unsupervised STDP learning in convolutional layers with binarized weights. This temporal encoding method provides compact and precise input feature representation, reduces spike redundancy per timestep, and preserves essential temporal dynamics while maintaining compatibility with low-power neuromorphic systems. Second, we introduced a novel supervised training scheme for the classification step that integrates sparse coding based on latency coding with a simple IF neuron model, complemented by gradient-based classification at the output layer. Third, we designed and implemented a modular, highly reconfigurable event-driven accelerator architecture that supports both unsupervised and supervised training methods through configurable spike encoders, feature extraction modules, and support for both IF and LIF neuron dynamics.

7.1 Conclusion

A key contribution of this work is the introduction of the zero-skip optimization technique, which represents a significant advancement in reducing computational overhead in spike-based processing. The experimental results demonstrate remarkable efficiency gains, with

latency reductions of up to 78.2% for individual convolution operations and meaningful improvements at the network level. This optimization exploits the inherent sparsity of spike trains generated through latency encoding, dynamically pruning inactive regions based on bounding box analysis. The proposed BSTDPG architecture achieves competitive performance across standard benchmarks, attaining 99.1% accuracy on MNIST and 91.0% on F-MNIST while utilizing fewer convolutional layers than state-of-the-art approaches. These results demonstrate that the framework successfully balances biological plausibility, computational efficiency, and learning performance.

The modular event-driven accelerator architecture presented in Chapter 5 demonstrates the practical feasibility of deploying BSTDPG in neuromorphic systems. The highly reconfigurable design supports variable bit widths, neuron counts, memory bank structures, and AER encoders, making it adaptable for various network topologies and application constraints. The proposed FSM architecture, combined with TTFS encoding, IF neuron dynamics, and local STDP learning rule, provides a flexible foundation for efficient hardware acceleration of binarized SNNs.

7.2 Future Work

While this thesis has made significant contributions to the field of SNNs, several possible directions emerge for future research and development.

7.2.1 Architectural Enhancements

Future research could explore advanced binarization techniques that preserve higher precision in critical network components while maintaining memory and computational advantages. This may include investigating mixed-precision approaches, where certain layers or connections retain higher bit-widths based on their contribution to network performance, while less critical components use $1-$bit precision weights.

7.2.2 Reconfigurable Architecture Extensions

Building upon the modular accelerator design, future research could explore dynamic layer reconfiguration capabilities that allow the hardware to adapt its architecture during runtime based on workload characteristics and power constraints. This could include

developing self-optimizing hardware that can automatically adjust bit widths, encoding method, neuron model, and memory configurations.

7.2.3 Accelerator Architecture Optimization

Developing specialized memory hierarchies and interconnect architectures optimized for the specific access patterns of binarized SNNs could yield significant performance improvements. This could involve designing custom cache structures for spike data, optimizing AER encoding/decoding pipelines, and developing efficient routing mechanisms for event-driven communication.

7.2.4 Reconfigurable Hardware Validation

Comprehensive validation of the reconfigurable accelerator across different kernel, structures, bit-widths, and neuron models would provide important insights into the trade-offs between flexibility and performance. This could include developing automated design space exploration tools for optimal hardware configuration selection.

7.2.5 Event-Based Vision Processing

The temporal processing capabilities of BSTDPG make it particularly suitable for event-based vision applications. Future work could focus on adapting the framework for dynamic vision sensors and exploring applications in autonomous navigation, object tracking, fall detection, and gesture recognition.

7.2.6 Real-Time Robotics Applications

Deploying BSTDPG in real-time robotics applications, particularly for sensorimotor control and adaptive behavior learning, could demonstrate the practical benefits of the framework in dynamic, real-world environments.

7.2.7 Adaptive Binarization

Investigating dynamic binarization techniques that can adapt weight precision during training based on layer importance, gradient magnitudes, or performance requirements

could provide better accuracy-efficiency trade-offs. Also developing theoretical frameworks to analyze the convergence properties of binarized STDP learning combined with gradient-based training could provide important insights into the stability and optimality of the hybrid approach.

7.3 Final Remarks

The work presented in this thesis marks a significant advancement in the development of efficient, hardware-friendly training methods for binarized SNNs. The proposed BSTDPG framework, combined with its reconfigurable accelerator architecture, demonstrates that competitive learning performance can be achieved while preserving the energy efficiency benefits of neuromorphic computing enabled by temporal coding and weight binarization. The modular and reconfigurable design of the accelerator offers a practical solution to the challenge of developing flexible neuromorphic hardware that can adapt to diverse network topologies and application requirements.

As neuromorphic computing continues to progress toward real-world deployment in edge computing, autonomous systems, and real-time processing applications, the principles and techniques introduced in this work provide a solid foundation for future innovation. The proposed hybrid learning approach presents a promising direction for developing adaptive learning systems capable of efficiently processing temporal information in resource-constrained environments. The reconfigurable architecture supports exploration of novel training algorithms and network structures, while the binarized design ensures compatibility with energy-efficient hardware implementations. Future research building on this foundation holds the potential to unlock new capabilities in intelligence systems that are computationally efficient, biologically inspired, and practically deployable in real-world scenarios.

References

[1] Jesper Sjöström and Wulfram Gerstner. Spike-timing dependent plasticity. *Scholarpedia*, 5(2):1362, 2010. [cited: x, 3, 18, and 30].

[2] Zhengyu Cai, Hamid Rahimian Kalatehbali, Ben Walters, Mostafa Rahimi Azghadi, Amirali Amirsoleimani, and Roman Genov. Spike timing dependent gradient for direct training of fast and efficient binarized spiking neural networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 13(4):1083–1093, 2023. [cited: xi, 3, 34, 35, and 61].

[3] Saeed Reza Kheradpisheh, Maryam Mirsadeghi, and Timothe Masquelier. Bs4nn: binarized spiking neural networks with temporal coding and learning. *Neural Processing Letters*, 54(2):1255–1273, 2022. [cited: xii, 18, 26, 62, 63, 64, 65, 69, and 70].

[4] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997. [cited: 1 and 8].

[5] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothe Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural networks*, 111:47–63, 2019. [cited: 1, 7, 17, and 26].

[6] Chetan Singh Thakur, Jamal Lottier Molin, Gert Cauwenberghs, Giacomo Indiveri, Kundan Kumar, Ning Qiao, Johannes Schemmel, Runchun Wang, Elisabetta Chicca, Jennifer Olson Hasler, et al. Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in neuroscience*, 12:891, 2018. [cited: 1].

[7] Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990. [cited: 1].

[8] Filip Ponulak and Andrzej Kasinski. Introduction to spiking neural networks: Information processing, learning and applications. *Acta neurobiologiae experimentalis*, 71(4):409–433, 2011. [cited: 1, 7, and 35].

[9] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018. [cited: 1, 2, 24, 25, and 77].

[10] Jibin Wu, Emre Yılmaz, Malu Zhang, Haizhou Li, and Kay Chen Tan. Deep spiking neural networks for large vocabulary automatic speech recognition. *Frontiers in neuroscience*, 14:199, 2020. [cited: 1].

[11] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *Proceedings of the IEEE*, 111(3):257–276, 2023. [cited: 1].

[12] John Hopfield. Artificial neural networks. *IEEE Circuits and Devices Magazine*, 4(5):3–10, 1988. [cited: 1].

[13] Simon Haykin. *Neural networks and learning machines, 3/E*. Pearson Education India, 2009. [cited: 1].

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. [cited: 1].

[15] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018(1):7068349, 2018. [cited: 1].

[16] Karen Sparck Jones. Natural language processing: a historical review. *Current issues in computational linguistics: in honour of Don Walker*, pages 3–16, 1994. [cited: 1].

[17] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. [cited: 2, 17, and 26].

[18] John Von Neumann. First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75, 1993. [cited: 2].

[19] Giacomo Indiveri and Shih-Chii Liu. Memory and information processing in neuromorphic systems. *Proceedings of the IEEE*, 103(8):1379–1397, 2015. [cited: 2].

[20] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014. [cited: 2].

[21] Jacques Gautrais and Simon Thorpe. Rate coding versus temporal order coding: a theoretical approach. *Biosystems*, 48(1-3):57–65, 1998. [cited: 2, 13, 14, and 23].

[22] Roland S Johansson and Ingvars Birznieks. First spikes in ensembles of human tactile afferents code complex spatial fingertip events. *Nature neuroscience*, 7(2):170–177, 2004. [cited: 2, 13, and 14].

[23] Jaehyun Kim, Heesu Kim, Subin Huh, Jinho Lee, and Kiyoung Choi. Deep neural networks with weighted spikes. *Neurocomputing*, 311:373–386, 2018. [cited: 2, 14, and 15].

[24] Charles Franklin Stevens. *A quantitative theory of neural interactions: theoretical and experimental investigations.* The Rockefeller University, 1964. [cited: 2 and 9].

[25] Jianfeng Feng and David Brown. Integrate-and-fire models with nonlinear leakage. *Bulletin of Mathematical Biology*, 62(3):467–481, 2000. [cited: 2 and 9].

[26] Larry F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin*, 50(5-6):303–304, 1999. [cited: 2].

[27] Bruce W Knight. Dynamics of encoding in a population of neurons. *The Journal of general physiology*, 59(6):734–766, 1972. [cited: 2].

[28] Jilles Vreeken et al. Spiking neural networks, an introduction. *reports*, 2003. [cited: 3 and 8].

[29] Youngeun Kim and Priyadarshini Panda. Visual explanations from spiking neural networks using inter-spike intervals. *Scientific reports*, 11(1):19037, 2021. [cited: 3].

[30] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 2002. [cited: 3].

[31] Manon Dampfhoffer, Thomas Mesquida, Alexandre Valentian, and Lorena Anghel. Backpropagation-based learning techniques for deep spiking neural networks: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2023. [cited: 3, 17, and 26].

[32] Saeed Reza Kheradpisheh and Timothe Masquelier. Temporal backpropagation for spiking neural networks with one spike per neuron. *International journal of neural systems*, 30(06):2050027, 2020. [cited: 3, 18, 22, 23, 26, 60, 61, 65, and 69].

[33] Zhengyu Cai, Hamid Rahimian Kalatehbali, Ben Walters, Mostafa Rahimi Azghadi, Amirali Amirsoleimani, and Roman Genov. Stdg: Fast and lightweight snn training technique using spike temporal locality. In *2023 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–5. IEEE, 2023. [cited: 3, 17, and 34].

[34] Fei-Fei Li, Marco Andreeto, Marc'Aurelio Ranzato, and Pietro Perona. Caltech 101, Apr 2022. [cited: 5, 57, and 58].

[35] Yann LeCun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. [cited: 5 and 58].

[36] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. arXiv, 2017. [cited: 5 and 58].

[37] Matthias Oster, Rodney Douglas, and Shih-Chii Liu. Quantifying input and output spike statistics of a winner-take-all network in a vision system. In *2007 IEEE International Symposium on Circuits and Systems*, pages 853–856. IEEE, 2007. [cited: 7 and 26].

[38] Mostafa Rahimi Azghadi, Ying-Chen Chen, Jason K Eshraghian, Jia Chen, Chih-Yang Lin, Amirali Amirsoleimani, Adnan Mehonic, Anthony J Kenyon, Burt Fowler, Jack C Lee, et al. Complementary metal-oxide semiconductor and memristive hardware for neuromorphic computing. *Advanced Intelligent Systems*, 2(5):1900189, 2020. [cited: 7 and 26].

[39] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in neuroscience*, 12:774, 2018. [cited: 7 and 26].

[40] Mostafa Rahimi Azghadi, Corey Lammie, Jason K Eshraghian, Melika Payvand, Elisa Donati, Bernabe Linares-Barranco, and Giacomo Indiveri. Hardware implementation of deep network accelerators towards healthcare and biomedical applications. *IEEE Transactions on Biomedical Circuits and Systems*, 14(6):1138–1159, 2020. [cited: 7 and 26].

[41] Nikola K Kasabov. *Time-space, spiking neural networks and brain-inspired artificial intelligence*. Springer, 2019. [cited: 7].

[42] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019. [cited: 7 and 13].

[43] Aboozar Taherkhani, Ammar Belatreche, Yuhua Li, Georgina Cosma, Liam P Maguire, and T Martin McGinnity. A review of learning in biologically plausible spiking neural networks. *Neural Networks*, 122:253–272, 2020. [cited: 7].

[44] Casey Henley. Synapse structure. *Foundations of Neuroscience*, 2021. [cited: 8].

[45] Jahangir Moini, Anthony LoGalbo, and Raheleh Ahangari. *Foundations of the mind, brain, and behavioral relationships: understanding physiological psychology*. Elsevier, 2023. [cited: 8].

[46] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002. [cited: 8, 10, and 11].

[47] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003. [cited: 9 and 23].

[48] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952. [cited: 9 and 23].

[49] Jason K Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*, 111(9):1016–1054, 2023. [cited: 10, 12, 13, 17, and 26].

[50] Filipe Rodrigues, Carlos Cardeira, and JMF Calado. Neural networks applied to short term load forecasting: A case study. In *Smart Energy Control Systems for Sustainable Buildings*, pages 173–197. Springer, 2017. [cited: 12].

[51] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014. [cited: 13].

[52] Stephen M Schuetze. The discovery of the action potential. *Trends in Neurosciences*, 6:164–168, 1983. [cited: 13].

[53] Edgar D Adrian. The impulses produced by sensory nerve endings: Part i. *The Journal of physiology*, 61(1):49, 1926. [cited: 13 and 23].

[54] Andreas K. Engel, Peter König, Andreas K. Kreiter, Thomas B. Schillen, and Wolf Singer. Temporal coding in the visual cortex: new vistas on integration in the nervous system. *Trends in Neurosciences*, 15(6):218–226, 1992. [cited: 13].

[55] Bruno B Averbeck, Peter E Latham, and Alexandre Pouget. Neural correlations, population coding and computation. *Nature reviews neuroscience*, 7(5):358–366, 2006. [cited: 13 and 15].

[56] Quentin JM Huys, Richard S Zemel, Rama Natarajan, and Peter Dayan. Fast population coding. *Neural computation*, 19(2):404–441, 2007. [cited: 13].

[57] Simon Thorpe and Jacques Gautrais. Rank order coding. In *Computational Neuroscience: Trends in Research, 1998*, pages 113–118. Springer, 1998. [cited: 14, 16, and 23].

[58] John E Lisman. Bursts as a unit of neural information: making unreliable synapses reliable. *Trends in neurosciences*, 20(1):38–43, 1997. [cited: 14].

[59] Wenzhe Guo, Mohammed E Fouda, Ahmed M Eltawil, and Khaled Nabil Salama. Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems. *Frontiers in Neuroscience*, 15:638474, 2021. [cited: 15 and 16].

[60] Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. Spike-based strategies for rapid processing. *Neural networks*, 14(6-7):715–725, 2001. [cited: 16, 23, and 27].

[61] João D Nunes, Marcelo Carvalho, Diogo Carneiro, and Jaime S Cardoso. Spiking neural networks: A survey. *IEEE access*, 10:60738–60764, 2022. [cited: 17].

[62] Simon Davidson and Steve B Furber. Comparison of artificial and spiking neural networks on digital hardware. *Frontiers in Neuroscience*, 15:651141, 2021. [cited: 17].

[63] Friedemann Zenke and Tim P Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural computation*, 33(4):899–925, 2021. [cited: 18].

[64] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to

spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019. [cited: 18].

[65] Guobin Shen, Dongcheng Zhao, and Yi Zeng. Backpropagation with biologically plausible spatiotemporal adjustment for training deep spiking neural networks. *Patterns*, 3(6), 2022. [cited: 18].

[66] Chunming Jiang and Yilei Zhang. Klif: An optimized spiking neuron unit for tuning surrogate gradient function. *Neural Computation*, 36(12):2636–2650, 2024. [cited: 18].

[67] Yuhang Li, Yufei Guo, Shanghang Zhang, Shikuang Deng, Yongqing Hai, and Shi Gu. Differentiable spike: Rethinking gradient-descent for training spiking neural networks. *Advances in Neural Information Processing Systems*, 34:23426–23439, 2021. [cited: 18].

[68] Malu Zhang, Jiadong Wang, Jibin Wu, Ammar Belatreche, Burin Amornpaisannon, Zhixuan Zhang, Venkata Pavan Kumar Miriyala, Hong Qu, Yansong Chua, Trevor E Carlson, et al. Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks. *IEEE transactions on neural networks and learning systems*, 33(5):1947–1958, 2021. [cited: 18, 65, and 69].

[69] Malu Zhang, Jiadong Wang, Zhixuan Zhang, Ammar Belatreche, Jibin Wu, Yansong Chua, Hong Qu, and Haizhou Li. Spike-timing-dependent back propagation in deep spiking neural networks. *arXiv preprint arXiv:2003.11837*, 2020. [cited: 18, 60, 61, 68, and 69].

[70] Sander M Bohte, Joost N Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37, 2002. [cited: 18].

[71] Fangxin Liu, Wenbo Zhao, Yongbiao Chen, Zongwu Wang, Tao Yang, and Li Jiang. Sstdp: Supervised spike timing dependent plasticity for efficient spiking neural network training. *Frontiers in Neuroscience*, page 1413, 2021. [cited: 18].

[72] Jason K Eshraghian, Xinxin Wang, and Wei D Lu. Memristor-based binarized spiking neural networks: Challenges and applications. *IEEE Nanotechnology Magazine*, 16(2):14–23, 2022. [cited: 18].

[73] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016. [cited: 18].

[74] Nguyen-Dong Ho and Ik-Joon Chang. Tcl: an ann-to-snn conversion with trainable clipping layers. In *2021 58th ACM/IEEE design automation conference (DAC)*, pages 793–798. IEEE, 2021. [cited: 18].

[75] Ruohong Zhou. A method of converting ann to snn for image classification. In *2023 IEEE 3rd International Conference on Electronic Technology, Communication and Information (ICETCI)*, pages 819–822. IEEE, 2023. [cited: 18].

[76] Yi Jiang, Sen Lu, and Abhronil Sengupta. Stochastic spiking neural networks with first-to-spike coding. In *2024 International Conference on Neuromorphic Systems (ICONS)*, pages 24–31. IEEE, 2024. [cited: 18].

[77] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, and Michael Pfeiffer. Theory and tools for the conversion of analog to spiking convolutional neural networks. *arXiv preprint arXiv:1612.04052*, 2016. [cited: 18].

[78] Wulfram Gerstner. Hebbian learning and plasticity. *From neuron to cognition via computational neuroscience*, pages 0–25, 2011. [cited: 18 and 31].

[79] Chankyu Lee, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in Neuroscience*, 12, 2018. [cited: 18].

[80] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J Thorpe, and Timothe Masquelier. Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67, 2018. [cited: 18, 21, 61, 68, and 69].

[81] Milad Mozafari, Saeed Reza Kheradpisheh, Timothe Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh. First-spike-based visual categorization using reward-modulated stdp. *IEEE transactions on neural networks and learning systems*, 29(12):6178–6190, 2018. [cited: 18, 21, and 65].

[82] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, Simon J Thorpe, and Timothe Masquelier. Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *Pattern recognition*, 94:87–95, 2019. [cited: 18 and 21].

[83] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. *Spike-timing dependent plasticity*. Frontiers E-books, 2012. [cited: 18 and 31].

[84] Tobi Delbruck et al. Frame-free dynamic digital vision. In *Proceedings of Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society*, volume 1, pages 21–26. Tokyo, 2008. [cited: 19].

[85] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J Davison, Jörg Conradt, Kostas Daniilidis, et al. Event-based vision: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(1):154–180, 2020. [cited: 19].

[86] Shih-Chii Liu, Tobi Delbruck, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas. *Event-based neuromorphic systems*. John Wiley & Sons, 2014. [cited: 19 and 20].

[87] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 conference on computer vision and pattern recognition workshop*, pages 178–178. IEEE, 2004. [cited: 21].

[88] Amirreza Yousefzadeh, Evangelos Stromatias, Miguel Soto, Teresa Serrano-Gotarredona, and Bernab Linares-Barranco. On practical issues for stochastic stdp hardware with 1-bit synaptic weights. *Frontiers in neuroscience*, 12:665, 2018. [cited: 21 and 23].

[89] SG Hu, GC Qiao, TP Chen, Qi Yu, Y Liu, and LM Rong. Quantized stdp-based online-learning spiking neural network. *Neural Computing and Applications*, 33(19):12317–12332, 2021. [cited: 22].

[90] Gopalakrishnan Srinivasan and Kaushik Roy. Restocnet: Residual stochastic binary convolutional spiking neural network for memory-efficient neuromorphic computing. *Frontiers in neuroscience*, 13:189, 2019. [cited: 22].

[91] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015. [cited: 22 and 23].

[92] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Computer*

Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV, pages 525–542. Springer, 2016. [cited: 22].

[93] Wei Tang, Gang Hua, and Liang Wang. How to train a compact binary neural network with high accuracy? In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017. [cited: 22].

[94] Saeed Reza Kheradpisheh, Maryam Mirsadeghi, and Timothe Masquelier. Bs4nn: Binarized spiking neural networks with temporal coding and learning. *Neural Processing Letters*, 54(2):1255–1273, 2022. [cited: 22].

[95] Wulfram Gerstner. Spike-response model. *Scholarpedia*, 3(12):1343, 2008. [cited: 23].

[96] Danilo Pani, Paolo Meloni, Giuseppe Tuveri, Francesca Palumbo, Paolo Massobrio, and Luigi Raffo. An fpga platform for real-time simulation of spiking neuronal networks. *Frontiers in neuroscience*, 11:90, 2017. [cited: 23].

[97] Shuangming Yang, Jiang Wang, Bin Deng, Chen Liu, Huiyan Li, Chris Fietkiewicz, and Kenneth A Loparo. Real-time neuromorphic system for large-scale conductance-based spiking neural networks. *IEEE transactions on cybernetics*, 49(7):2490–2503, 2018. [cited: 23].

[98] Jianhui Han, Zhaolin Li, Weimin Zheng, and Youhui Zhang. Hardware implementation of spiking neural networks on fpga. *Tsinghua Science and Technology*, 25(4):479–486, 2020. [cited: 24].

[99] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. Spiker: an fpga-optimized hardware accelerator for spiking neural networks. In *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 14–19. IEEE, 2022. [cited: 24].

[100] Shu-Quan Wang, Lei Wang, Yu Deng, Zhi-Jie Yang, Sha-Sha Guo, Zi-Yang Kang, Yu-Feng Guo, and Wei-Xia Xu. Sies: A novel implementation of spiking convolutional neural network inference engine on field-programmable gate array. *Journal of Computer Science and Technology*, 35:475–489, 2020. [cited: 24].

[101] Qinyu Chen, Chang Gao, and Yuxiang Fu. Cerebron: a reconfigurable architecture for spatiotemporal sparse spiking neural networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(10):1425–1437, 2022. [cited: 24].

[102] Bipin Rajendran, Abu Sebastian, Michael Schmuker, Narayan Srinivasa, and Evangelos Eleftheriou. Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches. *IEEE Signal Processing Magazine*, 36(6):97–110, 2019. [cited: 24].

[103] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon 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*, 34(10):1537–1557, 2015. [cited: 24].

[104] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014. [cited: 24 and 25].

[105] Christian Mayr, Sebastian Hoeppner, and Steve Furber. Spinnaker 2: A 10 million core processor system for brain simulation and machine learning-keynote presentation. In *Communicating Process Architectures 2017 & 2018*, pages 277–280. IOS Press, 2019. [cited: 24].

[106] Charlotte Frenkel, Martin Lefebvre, Jean-Didier Legat, and David Bol. A 0.086-mm ˆ 2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos. *IEEE transactions on biomedical circuits and systems*, 13(1):145–158, 2018. [cited: 24, 25, and 77].

[107] Charlotte Frenkel and Giacomo Indiveri. Reckon: A 28nm sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3. IEEE, 2022. [cited: 24].

[108] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, Jean-Marie Bussat, Rodrigo Alvarez-Icaza, John V Arthur, Paul A Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014. [cited: 24].

[109] Christian Pehle, Sebastian Billaudelle, Benjamin Cramer, Jakob Kaiser, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Aron Leibfried, Eric Müller, and Johannes Schemmel. The brainscales-2 accelerated neuromorphic system with hybrid plasticity. *Frontiers in Neuroscience*, 16:795876, 2022. [cited: 24].

[110] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE transactions on biomedical circuits and systems*, 12(1):106–122, 2017. [cited: 24].

[111] Jan Stuijt, Manolis Sifalakis, Amirreza Yousefzadeh, and Federico Corradi. μbrain: An event-driven and fully synthesizable architecture for spiking neural networks. *Frontiers in neuroscience*, 15:664208, 2021. [cited: 25, 26, and 77].

[112] Ole Richter, Yannan Xing, Michele De Marchi, Carsten Nielsen, Merkourios Katsimpris, Roberto Cattaneo, Yudi Ren, Yalun Hu, Qian Liu, Sadique Sheik, et al. Speck: A smart event-based vision sensor with a low latency 327k neuron convolutional neuronal network processing pipeline. *arXiv preprint arXiv:2304.06793*, 2023. [cited: 25].

[113] Ling Liang, Zheng Qu, Zhaodong Chen, Fengbin Tu, Yujie Wu, Lei Deng, Guoqi Li, Peng Li, and Yuan Xie. H2learn: High-efficiency learning accelerator for high-accuracy spiking neural networks. *IEEE transactions on computer-aided design of integrated circuits and systems*, 41(11):4782–4796, 2021. [cited: 25].

[114] Ruokai Yin, Abhishek Moitra, Abhiroop Bhattacharjee, Youngeun Kim, and Priyadarshini Panda. Sata: Sparsity-aware training accelerator for spiking neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(6):1926–1938, 2022. [cited: 25].

[115] Lei Deng, Yujie Wu, Xing Hu, Ling Liang, Yufei Ding, Guoqi Li, Guangshe Zhao, Peng Li, and Yuan Xie. Rethinking the performance comparison between snns and anns. *Neural networks*, 121:294–307, 2020. [cited: 26].

[116] Sara Elkerdawy, Mostafa Elhoushi, Hong Zhang, and Nilanjan Ray. Fire together wire together: A dynamic pruning approach with self-supervised mask prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12454–12463, 2022. [cited: 30].

[117] Ami Citri and Robert C Malenka. Synaptic plasticity: multiple forms, functions, and mechanisms. *Neuropsychopharmacology*, 33(1):18–41, 2008. [cited: 31].

[118] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. A history of spike-timing-dependent plasticity. *Frontiers in synaptic neuroscience*, 3:4, 2011. [cited: 31].

[119] Hossein Gholamalinezhad and Hossein Khosravi. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*, 2020. [cited: 33].

[120] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. [cited: 34].

[121] Eric W Weisstein. Heaviside step function. https://mathworld.wolfram.com/, 2002. [cited: 37].

[122] Yann Lecun. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/, 1998. [cited: 57].

[123] Amirhossein Tavanaei and Anthony Maida. Bp-stdp: Approximating backpropagation using spike timing dependent plasticity. *Neurocomputing*, 330:39–47, 2019. [cited: 60, 61, 63, 65, and 69].

[124] Iulia M Comsa, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala. Temporal coding in spiking neural networks with alpha synaptic function. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8529–8533. IEEE, 2020. [cited: 60, 61, 68, and 69].

[125] Mengting Lan, Xiaogang Xiong, Zixuan Jiang, and Yunjiang Lou. Pc-snn: Supervised learning with local hebbian synaptic plasticity based on predictive coding in spiking neural networks. *arXiv preprint arXiv:2211.15386*, 2022. [cited: 61 and 65].

[126] Jason K Eshraghian and Wei D Lu. The fine line between dead neurons and sparsity in binarized spiking neural networks. *arXiv preprint arXiv:2201.11915*, 2022. [cited: 63 and 65].

[127] Shibo Zhou, Xiaohua Li, Ying Chen, Sanjeev T Chandrasekaran, and Arindam Sanyal. Temporal-coded deep spiking neural network with easy training and robust performance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11143–11151, 2021. [cited: 63 and 65].

[128] Maryam Mirsadeghi, Majid Shalchian, Saeed Reza Kheradpisheh, and Timothe Masquelier. Stidi-bp: Spike time displacement based error backpropagation in multilayer spiking neural networks. *Neurocomputing*, 427:131–140, 2021. [cited: 63 and 65].

[129] Hesham Mostafa. Supervised learning based on temporal coding in spiking neural networks. *IEEE transactions on neural networks and learning systems*, 29(7):3227–3235, 2017. [cited: 65].

[130] Yunzhe Hao, Xuhui Huang, Meng Dong, and Bo Xu. A biologically plausible supervised learning method for spiking neural networks using the symmetric stdp rule. *Neural Networks*, 121:387–395, 2020. [cited: 65 and 69].

[131] Joshua Arul Kumar Ranjan, Titus Sigamani, and Janet Barnabas. A novel and efficient classifier using spiking neural network. *The Journal of Supercomputing*, 76(9):6545–6560, 2020. [cited: 65 and 69].

[132] Wenrui Zhang and Peng Li. Temporal spike sequence learning via backpropagation for deep spiking neural networks. *Advances in Neural Information Processing Systems*, 33:12022–12033, 2020. [cited: 65 and 69].

[133] Timothe Masquelier and Simon J Thorpe. Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS computational biology*, 3(2):e31, 2007. [cited: 65].

[134] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998. [cited: 66].

[135] SS Barsov and VV Ul'yanov. Difference of gaussian measures. *Journal of Soviet Mathematics*, 38(5):2191–2198, 1987. [cited: 68].

[136] Gregory K Chen, Raghavan Kumar, H Ekin Sumbul, Phil C Knag, and Ram 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*, 54(4):992–1002, 2018. [cited: 77].

[137] Sung-Gun Cho, Edith Beigné, and Zhengya Zhang. A 2048-neuron spiking neural network accelerator with neuro-inspired pruning and asynchronous network on chip in 40nm cmos. In *2019 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4. IEEE, 2019. [cited: 77].

[138] Jeongwoo Park, Juyun Lee, and Dongsuk Jeon. 7.6 a 65nm 236.5 nj/classification neuromorphic processor with 7.5% energy overhead on-chip learning using direct spike-only feedback. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 140–142. IEEE, 2019. [cited: 77].