NeuroVision: A Neuromorphic Computing Accelerator for Image Processing

1st Franck Belemkoabga
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA, USA
fnbelemk@mit.edu

***Abstract*—In image processing, the need for energy-efficient and real-time solutions is important. NeuroVision explores the integration of neuromorphic computing principles in image processing tasks using Spiking Neural Networks (SNNs) on an FPGA platform. By leveraging the asynchronous and event-driven nature of SNNs, NeuroVision aims to deliver efficient and scalable solutions for tasks such as simple pattern recognition in images. NeuroVision seeks to bridge the gap between traditional image processing techniques and the emerging field of neuromorphic computing. By employing an SNN implemented on an FPGA, the project aims to do image-processing tasks in a way that mimics the energy efficiency and parallel processing of the human brain.**

***Index Terms*—Spiking Neural Networks, Neuromorphic Computing, Leaky Integrate-And-Fire, Convolution, MaxPool, Fully-Connected layer**

I. Introduction

In the field of digital image processing, the demand for efficient and high-speed processing systems has never been greater. Traditional computing methods, while effective, are increasingly facing challenges in keeping up with the amount of data and computational complexity of advanced signal processing tasks. These challenges set the stage for neuromorphic computing: an approach that emulates the human brain's neural structure and processing techniques.

NeuroVision aims to develop a neuromorphic computing accelerator tailored for image processing applications. Neuromorphic systems, characterized by their remarkable efficiency and speed, offer a compelling solution to the limitations of conventional computing methods in handling large-scale and complex image datasets.

NeuroVision seeks to harness the unique advantages of spiking neural networks (SNNs) - a key component of neuromorphic computing - to enhance the efficiency of image processing while reducing power consumption. Our objective is to design and implement a hardware accelerator that optimizes computational tasks and is scalable and adaptable across various imaging applications.

This report outlines the conceptual framework of NeuroVision, details our methodology, presents the preliminary results from the testing of individual modules, and sets forth the roadmap for the project's future development, including plans for FPGA implementation.

II. Methodology

A. Leaky Integrate-And-Fire Neuron

The LIF neuron model is a simplified representation of a biological neuron and serves as a fundamental unit in the architecture of Spiking Neural Networks (SNNs). In our project, NeuroVision, we employ the LIF neuron model due to its biological relevance and computational efficiency.

The LIF neuron integrates incoming electrical signals and generates a spike (output signal) when its membrane potential exceeds a certain threshold. Post-spike, the membrane potential resets, emulating the refractory period of biological neurons [1]. Key parameters of the LIF model in our implementation include the membrane resistance and decay time constant, which influence how the neuron integrates signals over time, and the threshold potential, determining the spiking behavior. The mathematical equation for the LIF Neuron is shown:

$$S[t] = \begin{cases} 1, & \text{if } U[t] > U_{\text{thr}} \\ 0, & \text{otherwise} \end{cases}$$

$$U[t+1] = \underbrace{\beta U[t]}_{\text{decay}} + \underbrace{WX[t+1]}_{\text{input}} - \underbrace{\beta S[t] U_{\text{thr}}}_{\text{soft reset}}$$

Fig. 1. LIF Neuron Equation

U[t] is the membrane potential, W is the weight, X[t] is the input data, S[t] is the output spike, Beta is the decay rate [1].

SNNs are a class of artificial neural networks that more closely mimic the functioning of the human brain compared to traditional neural networks. NeuroVision leverages SNNs for their efficiency and potential in processing complex image data. SNNs operate using discrete spikes, making them inherently more energy-efficient and suitable for real-time processing tasks [1].

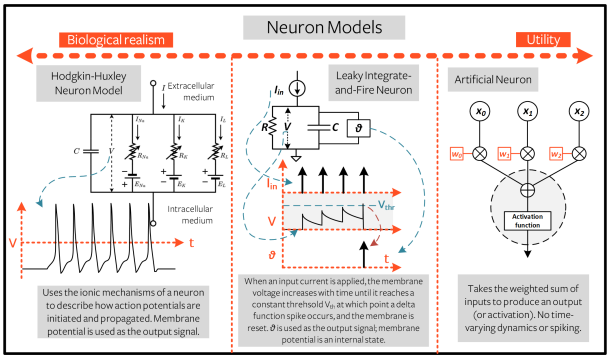

Fig. 2. Neuron Models

In NeuroVision, we construct the SNN using multiple layers of LIF neurons. The network topology and inter-neuron connectivity patterns are designed to optimize image processing tasks, with a focus on feature extraction and pattern recognition.

B. SNNTorch Framework

For the development and training of our SNN model, we utilize snntorch, a Python library designed to simulate and train SNNs. snntorch is chosen for its seamless integration with PyTorch, facilitating the use of GPU acceleration and a wide range of optimization tools.

We use snntorch to train our SNN model on relevant image datasets. The training process involves adjusting synaptic weights based on learning algorithms, ensuring the network effectively learns to recognize patterns and features in the input data.

The performance of the SNN during training is evaluated using metrics such as accuracy and loss, ensuring that the network generalizes well to new, unseen data.

Post-training, we extract crucial parameters, namely the synaptic weights and biases, from the snntorch model. These parameters are integral in defining the behavior of the SNN on hardware.

The extracted weights and biases are converted into a format suitable for FPGA implementation. This conversion process addresses challenges such as adapting floating-point weights from snntorch into fixed-point representations for efficient hardware utilization.

C. Convolutional Layer

The Conv layer serves as the cornerstone of our image processing approach in NeuroVision, primarily tasked with feature extraction from the input images.

The Conv Layer operates by applying multiple filters or kernels to the input image. These filters are designed to detect specific features, such as edges or textures. As they convolve across the image, they generate feature maps – representations that capture essential aspects of the image.

The effectiveness of the Conv Layer in extracting detailed features makes it instrumental for the network's ability to recognize and interpret complex visual patterns.

In NeuroVision, the Conv Layers are configured with considerations tailored to the nature of our input – small-sized images (like 8x8 pixels). We employ filters with smaller kernel sizes to suit these dimensions, ensuring an optimal balance between feature detection capability and computational efficiency.

D. MaxPoolLayer

MaxPool Layer In conjunction with the Conv Layers, the MaxPool Layer plays a critical role in our network by reducing the dimensionality of the feature maps.

The MaxPool Layer in NeuroVision functions by down-sampling the feature maps generated by the Conv Layers. It employs a window (typically 2x2) to scan through the feature map and selects the maximum value within each window. This process significantly reduces the spatial dimensions, leading to a decrease in computational complexity and model parameters. This layer enhances the network's ability to generalize and reduces its sensitivity to minor variations and distortions in the input image.

Implementing the MaxPool Layer on an FPGA presents unique challenges, particularly in handling parallel data streams and efficient memory utilization. In NeuroVision, we address these by employing optimized pooling algorithms that maximize throughput while minimizing resource consumption.

The design ensures that pooling operations are performed rapidly and accurately, a critical factor in maintaining the overall processing speed of the network.

E. Fully-Connected Layer

The Fully Connected (FC) Layer is a crucial component in NeuroVision, serving as the final stage in the neural network where the high-level reasoning based on the extracted features occurs.

The FC Layer in our neural network architecture takes the flattened output from the previous layers and performs classification tasks. Each neuron in this layer is connected to every neuron in the preceding layer, allowing it to integrate the global information extracted by earlier convolutional and pooling layers.

This layer is instrumental in making final decisions or predictions based on the cumulative knowledge gained through the network. It is particularly adept at recognizing patterns and making classifications.

In our implementation, the FC Layer is tailored to suit the specific requirements of our image-processing tasks. Given the compact nature of our input data (like 8x8 images), the layer is designed to handle a smaller number of input features while still being capable of accurate classification. We carefully select the number of neurons in the FC Layer to balance between computational efficiency and the network's ability to learn complex patterns.

The integration of the FC Layer with the spiking neural network architecture poses unique challenges, particularly in translating the continuous values into spike patterns suitable

for SNN processing. We employ specific encoding mechanisms to ensure this translation is efficient and lossless.

Implementing the FC Layer on FPGA involves optimizing for speed and resource usage. Given that fully connected operations can be computationally intensive, we utilize strategies like parallel processing and efficient memory access patterns to maximize the throughput.

F. Leaky SNN Layer

The Leaky Spiking Neural Network (SNN) Layer is an important component of the NeuroVision architecture, playing an important role in handling the temporal dynamics of neural information processing. This layer is specially designed to mimic the leaky integrate-and-fire mechanism seen in biological neurons, making it highly effective for tasks involving temporal data patterns, such as in video or real-time image processing scenarios.

In the layered structure of NeuroVision, the Leaky SNN Layer is adeptly positioned to receive and process inputs from earlier stages, typically following the initial feature extraction through convolutional and pooling layers. Its ability to manage the temporal aspect of the incoming data is a key requirement for our image-processing tasks.

The layer is characterized by neurons with a 'membrane potential' that leaks over time, simulating the gradual loss of electrical charge typical in biological neurons. When this potential reaches a certain threshold due to incoming spikes, the neuron fires, thereby generating a spike and resetting its potential. This dynamic allows the Leaky SNN Layer to process sequences of spikes over time, integrating temporal information into the network's decision-making process.

G. Flattening Layer

The Flattening Layer in the NeuroVision project plays a fundamental role in transitioning from the spatially organized layers, like convolutional and pooling layers, to the Fully Connected (FC) Layer. It acts as a critical intermediary that reshapes the data for subsequent processing.

In our neural network, this layer takes the multi-dimensional output (such as 2D arrays from image data) from previous layers and converts it into a one-dimensional array. This conversion is essential because the FC Layer, which follows, requires input in a linear format to perform classification tasks.

The transformation by the Flattening Layer is a simple yet vital step. It allows the network to move from understanding spatial features in the image, like edges and textures captured by convolutional layers, to interpreting these features in a high-level, holistic manner in the FC Layer.

This layer does not alter the data itself but reorganizes it, ensuring that the intricate patterns learned in the convolutional layers are not lost but are instead made accessible in a format suitable for the complex pattern recognition tasks carried out by the FC Layer.

H. Full Network Architecture for 28 by 28 MNIST Image Dataset

This neural network architecture is a convolutional spiking neural network as depicted in Figure 3. The network processes 28x28 pixel images from the MNIST dataset, following a structured series of layers to extract and interpret features for image recognition tasks.

Input Layer: The network begins with an input layer representing the 28x28 MNIST image, serving as the foundational data input.

First Convolutional Layer (Conv1): This layer applies eight filters of size 5x5 to the input image. Due to the filter size, the output dimension is reduced to 24x24, capturing essential spatial features from the image.

First MaxPooling Layer: Following Conv1, a MaxPooling layer is applied, which effectively reduces the dimensions by half, resulting in a 12x12 output for each of the eight feature maps. This pooling process helps in reducing the computational load while retaining significant features.

First Leaky Integrate-and-Fire (LIF) Layer: Next, the network includes a LIF layer, maintaining the dimensions of 12x12x8. This layer introduces the spiking neural network dynamics, crucial for temporal feature processing.

Second Convolutional Layer (Conv2): A second convolutional layer is employed, utilizing 16 filters of size 5x5. This layer further processes the data, reducing the output dimensions to 8x8, enhancing the extraction of complex features.

Second MaxPooling Layer: This layer again halves the dimensions, resulting in a 4x4 output for each of the 16 feature maps, further concentrating the important features and reducing data volume.

Second LIF Layer: The network then includes another LIF layer, with an output dimension of 4x4x16. This layer continues the process of temporal feature integration.

Flattening Layer: The output from the second LIF layer is transformed by the Flattening Layer, which converts the 4x4x16 tensor into a single vector. This step is crucial for preparing the data for the final fully connected layer.

Fully Connected Layer with LIF Output: The network concludes with a Fully Connected (FC) layer, integrated with a LIF output layer. This combination allows the network to perform final classification tasks based on both spatial and temporal features extracted throughout the layers.

For testing and adaptability, this network architecture is also reconfigured to accommodate an 8x8 input dataset, demonstrating the flexibility and scalability of our design.

I. Full Network Architecture for 8 by 8 MNIST Image Dataset

This neural network processes 8x8 pixel images, using a series of layers specifically designed for smaller datasets.

Input Layer: The network begins with an 8x8 pixel image, setting the stage for initial data processing.

Convolutional Layer (Conv1): The first layer employs eight 3x3 filters for convolution. This smaller kernel size is ideal for the 8x8 input, allowing the network to capture spatial features

effectively. The output from this layer is a set of feature maps with reduced dimensions.

MaxPooling Layer: Following Conv1, a MaxPooling layer halves the dimensions of the feature maps, reducing them further for efficient processing. This layer helps concentrate essential features while minimizing data size.

Leaky Integrate-and-Fire (LIF) Layer: Next, the LIF layer introduces the temporal processing characteristic of spiking neural networks. It maintains the dimensions from the MaxPooling layer, ensuring temporal features are integrated without increasing data volume.

Flattening Layer: This layer converts the output from the LIF layer into a one-dimensional vector. It prepares the multi-dimensional feature maps for the final classification layer, ensuring no critical information is lost in the transformation.

Fully Connected Layer with LIF Output: The network concludes with a Fully Connected layer, adjusted to handle the output size from the Flattening layer. Integrated with a LIF output layer, it ensures effective classification based on the spatial and temporal information processed through the network.

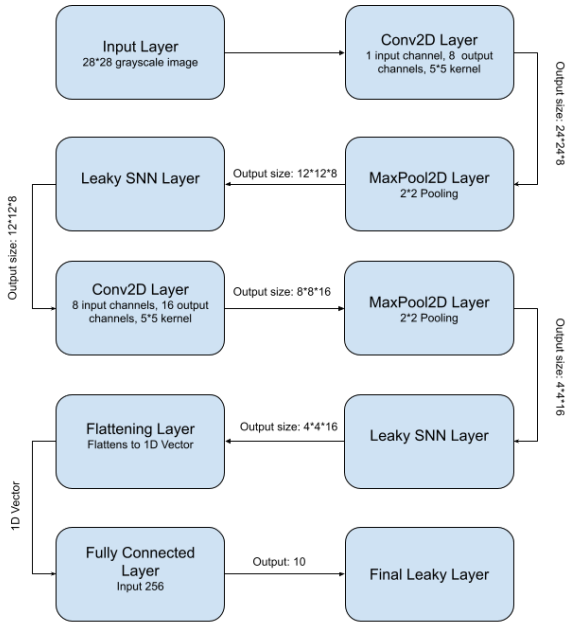

Fig. 3. Convolutional Spiking Neural Network

III. Results and Discussion

In this section, we detail the results obtained from the individual testing of each module within the NeuroVision project. These modules include the Leaky Integrate-and-Fire (LIF) Neuron, Convolutional Layer (ConvLayer), MaxPool Layer, and the Fully Connected Layer. The focus of our testing was to validate each module's functionality against theoretical models and mathematical expectations.

A. Leaky Integrate-and-Fire (LIF) Neuron Module

To validate the functionality and accuracy of the LIF neuron module, we conducted tests using predefined input parameters. One example of The test involved stimulating the LIF neuron with specific input values and observing its response. The parameters used for this test were as follows:

Input Spike (x = 0.500): 32'd26214 (representing an example of synaptic input)

Synaptic Weight (w = 0.4): 32'd32768 (representing the strength of the synaptic connection)

Decay Factor (beta = 0.819): 32'd53673 (determining the rate at which the neuron's potential decays over time)

Firing Threshold (threshold = 1): 32'd65535 (the potential at which the neuron fires)

Note that each number is in fixed point representation of float value using 16 bits for the integer side and 16 bits for the fractional bits. Later on, 1 bit is used for the integer part and 15 bits for the fractional bits as the values range between 1 and -1. Upon applying these inputs to the LIF neuron module, the observed responses were recorded. The behavior of the neuron, including its membrane potential dynamics and spiking behavior, was found to be consistent with the expected theoretical outcomes based on the mathematical models of LIF neurons in Figure 1.

The results from this test demonstrated that the LIF neuron module in our NeuroVision system operates in alignment with established theoretical models of spiking neurons. This consistency is crucial as it validates the reliability and accuracy of our neuron model implementation

B. Convolution Layer

The ConvLayer in our NeuroVision system was rigorously tested to validate its convolutional processing capabilities. This testing involved specific, predefined inputs and kernel weights to simulate a typical scenario where the layer processes image data. The setup for the test was as follows: The input data was a 2x2 matrix with fixed-point values representing a small section of an image. The values used were: 0.47, 0.78, 0.31, 0.59 Two sets of kernel weights were used, each forming a 2x2 matrix in Q1.15 format, with values such as -0.3, 0.8, 0.6, -0.5, 0.1, -0.7, 0.4, and 0.2. Biases were set to 0.01 and -0.01 in two's complement binary format.

The ConvLayer's performance was evaluated based on its ability to correctly apply these kernel weights to the input data and add the respective biases. The results were as follows:

The output feature maps generated by the layer were in precise agreement with the expected results which are respectively 0.384 and -0.267, calculated using the specified input, weights, and biases.

This included correct handling of the fixed-point arithmetic in the Q1.15 format, ensuring accurate convolution operations. The testing confirmed that the ConvLayer is capable of accurately performing convolutions with fixed-point arithmetic. This is crucial for the layer's role in feature extraction from image data within the NeuroVision system.

The ConvLayer's ability to process inputs with precise adherence to predefined weights and biases underscores the reliability of our system in handling image data. This accuracy is fundamental for tasks that rely on detailed feature recognition and extraction.

C. MaxPool Layer

The MaxPoolLayer in the NeuroVision system was tested using a feature map with values normalized between -1 and 1. This test aimed to validate the layer's downsampling accuracy under typical operating conditions. A 4x4 matrix with values within the range of -1 to 1 was utilized to simulate a realistic output from a preceding convolutional layer. Example input (in normalized format): [[-0.5, 0.2, 0.8, -0.1], [0.7, -0.6, 0.4, 0.9], [-0.3, 0.5, -0.2, 0.1], [0.2, -0.8, 0.6, -0.4]]

A standard 2x2 max pooling operation with a stride of 2 was applied to reduce the spatial dimensions. The MaxPoolLayer accurately processed the input feature map. The resulting 2x2 matrix was consistent with the maximum values within each 2x2 window of the input. The output matrix was [[0.7, 0.9], [0.5, 0.6]]

These results confirm that the MaxPoolLayer is effectively performing max pooling on normalized data. The test validated that our MaxPoolLayer can accurately downsample feature maps with values in the typical range of -1 to 1, an important aspect considering the normalization of data in neural networks. The accuracy of the MaxPoolLayer in processing normalized data ensures that the subsequent layers receive correctly processed inputs, which is vital for the overall functionality of the NeuroVision system.

D. Fully-Connected Layer

The Fully Connected Layer (FCL) in the NeuroVision system was subjected to a test using input data normalized between -1 and 1. This test aimed to assess the layer's ability to accurately integrate and classify such data. A small, flattened array representing normalized feature map outputs was used as the input to the FCL. The input data is [-0.5, 0.7, 0.2, -0.3, 0.6]. The layer was configured with predefined, normalized weights and biases. The weights are [[-0.2, 0.4, -0.6, 0.8, -0.1], [0.5, -0.3, 0.2, -0.4, 0.7]] and the biases are [0.1, -0.1]. For simplicity, let's assume a small number of output neurons, say 2, and corresponding weights and biases.

The FCL processed the input data by applying the weights and adding the biases, producing an output that was then compared with calculated expectations.

The output from the FCL, based on the given input, weights, and biases, was computed and found to align with the expected results which are respectively [0.19, 0.27] after calculation.

This test confirmed that the FCL is performing accurately with normalized input data, effectively integrating the inputs and producing an expected classification output. The ability of the FCL to handle normalized data is crucial for its role in making final decisions in the NeuroVision system. This test demonstrates the layer's reliability and precision in a typical neural network context.

E. Full Network Simulation

Following the rigorous testing and verification of each individual module's functionality, the complete network system was implemented and simulated. This comprehensive simulation, executed within the Vivado simulation environment, leveraged the 8x8 MNIST dataset. The input data, along with the network weights and biases, were hard-coded into the simulation. These components were represented in fixed-point format, comprising 16 bits in total, with 14 bits dedicated to the fractional part. This level of precision was chosen to strike a balance between computational efficiency and the accuracy necessary for effective neural network operations. The purpose of this simulation was to assess the integrated performance of all modules operating together, ensuring that the system is working. As illustrated in Figure 4, the output of each layer within the simulation is displayed. These outputs were meticulously compared with the corresponding outputs of the SNNTorch neural network, on a layer-by-layer basis. It was observed that each layer's output within the simulation aligns accurately with the outputs generated by the SNNTorch layers, confirming the fidelity of the implemented model in SystemVerilog.

IV. Interpreting Predicted Ouput Through Spike Counting

In the final stage of processing within the convolutional SNN implemented in SNNTorch, the output is represented in the form of spikes. Each neuron in the output layer generates spikes whose frequency or count is indicative of the network's decision-making process. To translate these spike patterns into a meaningful predicted output, a method known as spike counting is employed.

Spike counting is a technique used to interpret the output of SNNs. It involves counting the number of spikes that each neuron in the output layer generates over a certain period. This count is directly related to the neuron's activation level; the more spikes a neuron generates, the higher its activation for a specific input.

In the context of image classification tasks, like those involving the MNIST dataset, each neuron in the output layer typically corresponds to a specific class or digit. The class represented by the neuron that generates the highest spike count is considered the predicted class for the given input image. For example, in a network trained on the MNIST dataset, if the neuron corresponding to the digit '3' generates the highest spike count when an image is inputted, the network predicts that the image represents the digit '3'.

Spike counting is a natural and effective way to decode the output of SNNs because it leverages their inherent temporal dynamics. Unlike traditional neural networks, where the output is a continuous value representing class probabilities, SNNs communicate information via discrete spikes over time. Therefore, counting these spikes provides a straightforward and intuitive measure of the network's output.

In the simulation environment, such as the one illustrated in Figure 4, spike counting shows the output of each layer. For

example, the highest spike here is 37, and counting the index from 0 to 9, one can see that the network predicted number 7. Compared with the real data, the number is also 7.

The Spike count data is then graphically depicted as a histogram, where each bar's height is proportionate to the spike count from a specific neuron as shown in Figure 5. This approach enables a real-time, visually intuitive insight into the neural network's functioning.

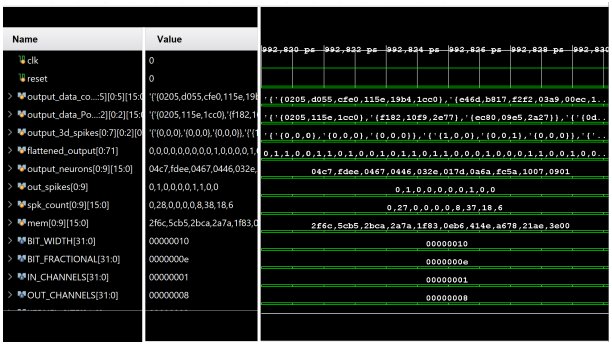

Fig. 4. Simulation of 8 by 8 network in Vivado simulation of the first version of design

Fig. 5. Snapshot of Histogram Display in HDMI. Over time the bars grow showing the neural activity. Image tilted due to phone incline

V. Optimization and Results

The initial design aimed to implement a digital processing system comprising multiple layers, including Convolutional Layers (ConvLayer), MaxPooling Layers (MaxPoolLayer), and others. While simulation results showed accuracy of this first design, it faced critical challenges: High Resource Utilization: Preliminary synthesis indicated that the design's complexity led to excessive usage of logic cells and memory blocks, surpassing the FPGA xc7s50csga324-1 capacity. The initial design used 685% of LUTs, and 14% of Slice Registers as shown in Figure 6. These challenges required a strategic optimization approach to make the design feasible.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	223233	0	0	32600	684.76
LUT as Logic	223233	0	0	32600	684.76
LUT as Memory	0	0	0	9600	0.00
Slice Registers	8660	0	0	65200	13.28
Register as Flip Flop	8660	0	0	65200	13.28
Register as Latch	0	0	0	65200	0.00
F7 Muxes	0	0	0	16300	0.00
F8 Muxes	0	0	0	8150	0.00

Fig. 6. LUTs, Slice Registers and Muxes Utilization for Second Version of Design

To address these issues, an optimization strategy was used: Transitioning from parallel to sequential execution in certain layers effectively reduced resource demands, particularly in the ConvLayer and MaxPoolLayer, without significantly impacting overall performance. Pipelining Adjustments: Reworking the pipelining stages in the data processing flow helped balance the load, improving timing characteristics and throughput. Simplifying the Verilog code and removing redundant logic operations contributed to a more resource-efficient design. Certain algorithmic modifications were made to ensure computational tasks were more hardware-friendly, focusing on reducing complexity and enhancing parallelism where feasible.

After implementing these optimizations, the final design fit within the FPGA resource constraints. As shown in figure 7, This design used 38% of LUTs as logic, 15% of Slice Registers. And 4% of Muxes. Compared to the first design, there is a reduction of 95% in LUTs usage.

A. Design

The FPGA design's core is governed by a finite state machine (FSM), important for making the sequential flow of operations. The FSM's primary role is to manage the execution of various processing layers, ensuring efficient use of the FPGA's resources.

The FSM is structured with several distinct states, including IDLE, STARTCONV, WAITCONV, STARTPOOL, WAITPOOL, and so forth, culminating in a DONE state. Transitions between these states are triggered by specific conditions, such as the completion of a task in a layer or a change in input signals (like user-operated switches).

In the IDLE state, the FSM awaits a start condition, typically a specific switch configuration. Upon this trigger, it transitions to STARTCONV, initiating the convolution layer's processing. The subsequent WAITCONV state ensures the FSM remains in a holding pattern until the convolution operation completes, indicated by a doneconv signal, before proceeding to the next layer.

The FSM is implemented in SystemVerilog using always ff blocks, ensuring synchronous operation with the FPGA's clock. Transitions and state maintenance are managed using case statements, providing clear and efficient state progression.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	12314	0	0	32600	37.77
LUT as Logic	12314	0	0	32600	37.77
LUT as Memory	0	0	0	9600	0.00
Slice Registers	9752	0	0	65200	14.96
Register as Flip Flop	9737	0	0	65200	14.93
Register as Latch	15	0	0	65200	0.02
F7 Muxes	616	0	0	16300	3.78
F8 Muxes	12	0	0	8150	0.15

Fig. 7. LUTs, Slice Registers and Muxes Utilization for Second Version of Design

VI. Future Exploration

One of the primary areas of future exploration involves scaling the network to handle more complex and larger datasets. This could entail increasing the depth and breadth of the network, experimenting with different layer configurations, or integrating more advanced types of neural layers. Scaling up the network has the potential to enhance its capability to process more intricate and higher-resolution images, making it suitable for a broader range of applications.

Further optimization of the network is important for future work. This could involve fine-tuning existing layers, exploring different learning algorithms, or employing novel methods of spike coding and decoding. Optimization efforts would aim to improve the accuracy, learning efficiency, and overall performance of the network, making it more effective and practical for real-world applications.

Another critical aspect for future research is the evaluation of the efficiency of spiking neural networks in comparison to traditional neural networks. This includes assessing aspects such as computational cost, energy consumption, and processing speed. Detailed studies could be conducted to quantify the benefits of SNNs in terms of their biologically-inspired processing mechanisms, particularly in tasks where temporal dynamics are crucial.

VII. Acknowledgments

Firstly, thanks to Professor Steinmeyer, whose guidance, and insights have been instrumental. Thanks to all the Teaching Assistants and Lab Assistants for their help and support.

VIII. Conclusion

NeuroVision uses the emerging field of neuromorphic computing to accelerate and optimize computational tasks. This project aims to design and develop a specialized hardware accelerator based on neuromorphic principles. Individual modules, each integral to the whole system, have been implemented and tested. These modules have been tested in simulation environments for functionality and performance, with preparations underway for subsequent FPGA implementation. Going forward, these individual modules will be put together for a fully functional neural network system.

A significant achievement was the adaptation of these designs to fit within the resource constraints of the Xilinx Spartan-7 XC7S50 FPGA. This was accomplished through a series of optimizations, including sequential execution and code refactoring, which addressed initial resource utilization challenges. Future works will explore further optimizations, new neural network architectures, and more complex applications.

References

[1] 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) September 2023. Jason K. Eshraghian et al. report on their work training spiking neural networks using lessons from deep learning, in Proceedings of the IEEE (September 2023).

[2] S. Barchid, J. Mennesson, J. K. Eshraghian, C. Djéraba, M. Bennamoun, “Spiking Neural Networks for Frame-based and Event-based Single Object Localization”, Neurocomputing, Sep. 2023.

[3] F. Ottati, C. Gao, Q. Chen, G. Brignone, M. R. Casu, J. K. Eshraghian, L. Lavagno, “To Spike or Not to Spike: A Digital Hardware Perspective on Deep Learning Acceleration”, arXiv preprint arXiv:2306.15749, June 2023.

[4] A. Mehonic, J. K. Eshraghian, “Brains and bytes: Trends in neuromorphic technology”, APL Machine Learning, 1(2), June 2023.