Recurrent Spiking Neural Networks in FPGA for Signal Processing Applications

Master's Thesis Report

Anand Sankaran

Committee Members:

Marco Gerards	-	Committee Chair, UT
Federico Corradi	-	External Committee Member, IMEC
Nikolaos Alachiotis	-	Committee Member, UT
D V Le Viet Duc	-	Committee Member, UT

January 28, 2022

Abstract

Gyro is a deep spiking event-based deep belief network architecture used in a range of Edge-AI applications. The existing Gyro architecture consists of a neuron model based on a timer and input encoded with temporal information. The model was not wholly event-driven, instead using complex, time-dependent calculations to calculate each neuron's membrane potential. The current system also utilises BRAM for storing its synaptic weight values which can pose the issue of lower maximum value compared to the use of alternative memory types. The synaptic weight storage is done in a parameterized manner, allowing the user to configure the throughput of the network. The use of round-robin schedulers to arbitrate the input and output was also found to be memory inefficient for large networks. Furthermore, Gyro does not currently support fully event-driven spiking neurons and recurrent connections, limiting the potential accuracy that can be achieved by the system for a relatively lower number of physical neurons.

Inspired by the capabilities of a spiking neural network, this work aims to develop an efficient architecture to run an event-based recurrent spiking neural network on an FPGA platform. The availability of the large and fast on-chip memory enables implementation of networks with a large number of neurons and synapses in a parallel manner. This project focuses on improving the existing deep spiking event-based deep belief network architecture named Gyro. Since the initial version of Gyro's Edge-AI architecture, the scope of improvement has been broad, including but not limited to enhanced functionalities, reduced memory consumption, reduced complexity of the design and computations, improved energy efficiency and reduced power consumption.

Exploration of different types of memory elements on an FPGA provided with possibilities on how these synaptic weights can be stored using on-board memory, but not limited with an upper boundary on the available memory as low as that of BRAMs. An investigation into how the network can be redesigned to incorporate recurrently spiking neurons was conducted, making the system truly event-triggered. This meant that time-dependent calculations were removed along with redundant computations, making the neuron model much simpler. The recurrent spiking functionality was backed up by a synaptic weight memory mapping which supported recurrent synaptic weights. Moreover, round-robin arbiters were replaced with sequential schedulers.

The resulting architecture was functionally verified on a Register Transfer Level, simulations and analysis of which concluded the following. The neuron model was made fully event-triggered and memory efficient, without any redundant computations. The network supports recurrently spiking neurons which is achieved by studying and using different types of on-board memory. Memory requirements were also reduced due to the replacement of round-robin schedulers. The change in the network at the fundamental level meant that the network inputs and outputs are less complex. The system also attains higher throughput compared to its previous iteration.

Acknowledgement

The research included in this master's thesis could not have been performed if not for the support, patience, and advice of many individuals. I would like to express my gratitude first and foremost to my thesis supervisor Federico Corradi for mentoring me over the course of my project duration. He has helped me through extremely difficult times over the duration of the project and for that I sincerely thank him for his support during that period and for his confidence in me. The completion of this project would have been extremely difficult, if not impossible, without his advises and suggestions as a supervisor. His insight into the project and feedback on a weekly basis were significant for the progress. I would also like to thank him for accepting my shortcomings in the neuromorphic domain and for helping me with understanding and learning it even more. His technical help was also crucial to completion of the project. I would additionally like to thank Paul Detterer for his technical advice and help during the design phase of the project.

I would also like to express my thanks to Nikolaos Alachiotis, whose guidance, comments, and questions regarding the methods helped me understand the project better and keep a track of the progress. His feedback on a weekly basis and the corrections and suggestions on this report helped me to attain a better presented project report. Furthermore, I would like to express my sincere regards for his help with the project from the University side and for supervision throughout the duration of the project.

This project would not have been successful if not for the personnel at University of Twente, Netherlands and IMEC, Netherlands, and for their assistance and help and co-operation. I would like to thank Wendi Klieverik, UT and Marsha Loomans, IMEC for their co-operation and help during the initial phase and for the starting of the project.

Finally, I would like to extend my deepest gratitude to my dearest family members and friends, without whose love, support, and understanding, I could never have completed this project.

Contents

Chapter 1

Introduction

Neuromorphic computing or neuromorphic engineering is a field that focuses on the synthetic equivalent of neuro-biological networks present in the nervous system, implemented using VLSI (Very Large-Scale Integration) systems. It aims to imitate how a nervous system works by incorporating and realizing the parameters that affect a biological neural architecture. Neuromorphic architectures emerged as early as the 1980s as a more feasible alternative for neurological models that compare well to the biological brain. The main advantage of neuromorphic computing is its ability to provide an energy-efficient way to enable intelligent algorithms on hardware [26]. Factors like optimal architectures for efficient computation, learning, development and adapting to local changes in inputs are critical deciding aspects in neuromorphic computing. These can vary depending on the field of application, spanning multiple disciplines like biology and chemistry [4], data science and image processing [30], and robotics [35].

The realization of different network architectures over the years has various applications, for which they are optimized. While the conventional hardware implementations are based on massive parallelism for neuromorphic computing, Edge-based applications have constantly been evolving. These are applications where the system that uses a machine learning algorithm processes data locally on the device. Memory requirements and massive computations are the challenges for efficient implementations since these are dependent on the hardware. Architectural optimizations can further reduce latency and power consumption without a reduction in computational accuracy or throughput. This is where FPGAs (Field-Programmable-Gate-Arrays) come into play, as FPGAs are versatile, which suits well for the realization of a neural network with sparse activity on them and the architecture specific custom hardware optimizations. Hardware implementations involving off-the-shelf components, opt for FPGAs because of its efficiency. Furthermore, inherent parallelism and scalability of neural networks can be exploited on FPGAs, without demanding high resource.

Current technological leaps in artificial intelligence using machine learning and deep learning signify the importance of brain-inspired neuromorphic computing elements [29]. Machine learning algorithms have evolved to implement neural networks that have become increasingly complex over the years. However, the requirements for such computing elements in energy consumption, area or computing power have low tolerances. In a traditional Von Neumann architecture, time and power are required in the extremities since the algorithms have increased computational density. This bottleneck is overcome using novel architectures and running these neural networks on dedicated hardware, which can provide massive parallelization in computing [15]. In this project, one such implementation of neural network, Gyro, on a dedicated hardware is being explored with the aim of design optimization and performance improvement.

Gyro implements a Spiking Neural Network (SNN), which is an event-driven neural network with sparse activity, modelled after biological neural networks. Implementation on FPGA provides the advantages like flexibility and parallelism in terms of architecture design. On-chip memories are used to store the synaptic weights to ensure a high memory bandwidth as opposed to off-chip memory. It uses a parameterized method to map the synaptic weights to on-chip memories facilitating both feed-forward and feed-back networks. The weight memory mapping parameters offer a configurable trade-off between forward and backward throughput and hardware resource utilization. Furthermore, a hardware efficient LIF neuron model has been implemented to avoid the necessity of neuron time-multiplexing. The neuron model uses time-dependent input for its computations. Gyro achieved a classification accuracy of 99,3% on the MNIST dataset using a network of 2954 neurons which is among the highest of the related SNN implementations. Gyro paved a way for deploying SNNs on power and energy constrained systems, and achieved an energy efficiency superior to all the related works.

1.1 Problem Statement

The focus of this work is the Gyro architecture [5], which is implementation of a spiking neural network on a FPGA, supporting forward and backward spike propagation in a time-dependent manner. While this implementation leverages custom hardware, parallel processing, on-chip memory and performs adequately in terms of efficiency, the main goal of this project was to improve memory and power requirements, efficiency of computations, and functionality. This is mainly because of how the application of Gyro consisted of Edge deployment. Specifically, the time-dependency of the computations in the neuron model meant the system is not event-driven in nature. This is crucial when it comes to memory requirements of FPGA implementation. The current Gyro architecture also only supports feed-forward networks - the lack of recurrent connections can decrease the predictive power of the network. Gyro is designed for edge-computing applications such as drones, which are defined by their operation at or near the source of the data, and therefore the need to operate independently of a central data center once deployed. This means that there are limited resources available, requiring that Gyro needs to be further optimized for computational efficiency.

The research question being *"Can the sparsity in time and space of a neural network be exploited to reduce power consumption and memory requirements?"* Exploration of potential solutions meant significant changes to the architecture in terms of performance of the existing Gyro architecture.

This project therefore focuses on the following sub-questions and exploring the pathways:

- Event-triggered systems are preferred for designs where resource utilization is required to be optimized [28]. Moreover, event-triggered systems are scalable when compared against time-triggered systems [14]. How can the system be made event-driven in nature and thereby reduce memory consumption? How can the event-driven nature of the SNNs be exploited to implement a much more memory efficient architecture?

- Spiking neural networks are sparse in nature. What is the potential of exploiting this sparse nature that could result in an improvement in terms of performance gains or memory requirements?
- Current architecture of Gyro supports feed-forward and feedback connections in the network. It does not support recurrent connections. Can the architecture be modified such that it supports recurrent connections? How does this change affect the predictive power of the network?
- Gyro focuses on Edge-AI applications. How can the memory and power requirements be reduced since computations are done locally on the system?
- Computations consume memory and power, and increase the complexity of the design for hardware implementation. Does the system contain redundant computations, and if so, how can it be removed to reduce the complexity of the design?
- Both the network architecture and the hardware have the potential to support parallelization in its implementation. Is the parallel nature of the architecture fully utilized? If not, how can it be implemented using the custom hardware?
- The architecture supports networks of different sizes. Does the system benefit from the scalability of the network? How can this prove to be useful for making the system more dynamic?

1.2 Scientific Contributions

Overall improvement of the architecture includes improved efficiency in terms of power and computations by taking advantage of the said event-driven and sparse nature of Spiking Neural Networks. The design changes were made in such a way that it does not increase latency, reduce throughput, or reduce accuracy, of the system.

This work, the second iteration of Gyro architecture, therefore contains the following improvements in terms of design and functionality:

- Removal of timestamp and tracking of time, and making the neuron model event-driven in nature by using a trigger signal
- Reduced memory requirements in architecture for implementation of a network that compares based on the same application test cases
- Elimination of redundant computations, thereby reducing the design complexity and playing a crucial role in reduced resource consumption
- Extended capability to handle recurrent spikes using a weight memory mapping patterns for synaptic weight storage
- Increased throughput by handling recurrently spiking neurons
- Reduced memory overhead using sequential schedulers instead of round-robin arbiters

1.3 Section Overview

The report structure for the master's thesis is as follows. Relevant background information on spiking neural networks and FPGA implementations, for understanding the work is described in chapter 2 along with a glimpse into the results of the first version of Gyro architecture. Literature study is chapter 3 which contains a brief description of similar implementations to the Gyro architecture, mainly FPGA implementations. The design of the modified Gyro architecture is explained in detail in chapter 4. VHDL implementation and simulation results are in chapter 5. The evaluation results of synthesized design are explained in chapter 6. Chapter 7 is conclusion of the work along with possible improvements that can be implemented in future.

Chapter 2

Background Material

There exists different types of artificial neural networks, with different models and algorithms to implement the core computational element. There are also different platforms when it comes to hardware implementation of neural networks. Specific model of computing element, the type of neural network, relevant hardware implementation and a brief description of the focus of this project is explored in this chapter.

The neuron model chosen for Gyro has its advantages when it comes to hardware implementation due to the simplicity of the algorithm. Spiking neural networks are event-driven in nature and closely resembles or mimics a human brain, and is the type of network used by Gyro. Deep spiking neural networks define a network topology using the spiking neural network as its backbone. When it comes to hardware implementations of deep spiking neural networks, current technology offers different platforms, each unique with its own advantages and disadvantages. Among these platforms, FPGAs are custom hardware with different types of memory elements available, having different use cases. Gyro uses a combination of these aspects for its FPGA implementation.

2.1 Leaky Integrate-and-Fire Model

Neurons are the main computing elements in a neuromorphic computing systems. The realization of neurons can be done with different complexities, each model being optimized for a specific application or network configuration. Among these models, the Leaky Integrate-and-Fire (LIF) is one of the simplest models with ease of simulation and analysis. In its simplest form, the neuron is modelled as a RC-filter, a circuit with a resistor and capacitor in parallel. A LIF neuron model is described as:

$$\tau_m \frac{dVm(t)}{dt} = -V_m(t) + R_m I(t) \tag{2.1}$$

where V is the membrane potential at time t, R_m is the membrane resistance and C_m is the membrane capacitance, with τ_m=R_m*C_m being the time constant, and I is the total input current. The temporal evolution of membrane potential of a typical LIF neuron model is shown in figure 2.1 along with the input and output spikes.

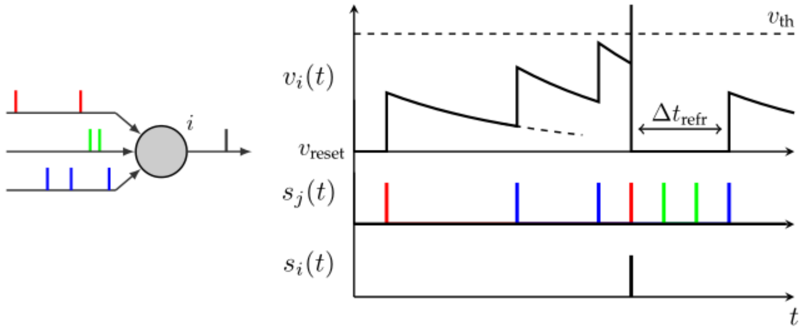


Figure 2.1: Membrane potential temporal evolution of a LIF neuron [23]

When a neuron receives an input spike, the synaptic weight associated to the spike is integrated to the membrane potential. When this membrane potential is above the threshold voltage V_{th}, the neuron fires an output spike, and the membrane potential is reset. The neuron retains this state for a fixed amount of time, known as refractory period, Δt_{refr}, during which time period input spikes are not integrated and output spikes are not fired. Membrane potential retains the reset value during this period and will not change. This is where the time-dependent data arises in the neuron model. The leakage is an exponential decay of membrane potential which is observed between every input spike and in refractory period. The neuron needs to store the previous input spike time and is compared to the current input spike time to calculate the decay voltage. The output is decided after comparing the potential to the threshold value and can be defined as:

$$x_j^l(t) = \begin{cases} 1, & \text{if } V_m^l(t) > V_t \text{ and } t - t_{spike} > T_r \\ 0, & \text{otherwise.} \end{cases} \tag{2.2}$$

where t_{spike} is the last time step at which neuron fired and t is time. T_r is refractory period. V_m and V_t are membrane potential and threshold voltage, respectively.

2.2 Spiking Neural Networks

Spiking Neural Network is directly inspired by a naturally occurring neural networks, mainly a biological brain. SNNs easily enable low power hardware evaluation. The event-driven nature of SNN makes it efficient for both computation and communication [24]. Its operating model consists of neuronal state and synaptic state, in which neurons transmit information only when the membrane potential overcomes a threshold voltage. The information is propagated between neurons in the form of spikes. The neuron receives input spikes from other neurons and integrates them into its membrane potential. Each neuron fires a spike when the membrane potential reaches the threshold value. The spike is then propagated to other neurons depending on the network configuration, like feed-forward, recurrent network, etc. The connections between the neurons are implemented as synapses weighted according to the network topology. A post-synaptic neuron receives the weighted activation w value, which is multiplied with the pre-synaptic value x.

$$y = \sum_{i=1}^{n} x_i * w_i \tag{2.3}$$

This is a linear perceptron, and non-linearity is what plays a crucial role in neural networks. To implement that, a non-linear function F known as activation function is applied onto y. The network can learn non-linear dependencies because of the activation function.

$$y = \phi(\sum_{i=1}^{n} x_i * w_i) \tag{2.4}$$

A neural network can have different network configurations. With each layer containing a specific number of neurons or computing elements and the number of layers depending on the network size, the interconnect of neurons decides the network topology, which in turn affects the neuron architecture and its computations [3]. A Fully Connected Network will have all neurons in one layer connected to every other neuron in its adjacent layers. These connections are synapses, which means that for two layers of n_1 and n_2 number of neurons in a fully connected configuration, they will have $n_1 * n_2$ synapses. Other configurations include Feed-Forward Networks, Recurrent Neural Networks (RNN), Boltzmann machines, Deep Belief Network (DBN), etc. The combination of feed-forward and feed-back network topology is advantageous compared to a strictly feed-forward network. It can integrate multi-sensory inputs and do predictive error correction, made possible by the recurrent connections.

SNNs also take advantage of event-driven nature by updating the neurons only when there is a spiking activity. In contrast to synchronous systems, the neurons are not updated at every time step. The amount of computation required for such synchronous systems are much higher when compared to event driven spiking neural networks.

Spiking Neural Networks require equal or lesser number of computing elements to realize a function, when compared to artificial neural networks or convolutional neural networks, depending on the function. Since the information propagated also contains the element of time in some format, it enables SNNs to extract temporal information from this time dependent data, much more efficiently.

2.3 Deep Spiking Neural Networks

A Deep Neural Network (DNN) is a type of Artificial Neural Network composed of many layers between the input and output. These have the same components as other neural networks, like a neuron, synapse, axons, weights, etc. DNNs can model complex non-linear relations.

DNNs are usually designed as feed-forward networks. Data flows to the output layer from the input layer without looping back. The extra layers have proven to be much more effective in approximating sparse multivariate polynomials than shallow networks [25]. At first, the DNN assigns random weights to the synapses, which is multiplied with the input to return an output between 0 and 1. If the pattern is not recognized accurately, the algorithm adjusts the weights. This way, some parameters become more influential until the correct mathematical manipulation to process the data entirely is determined.

Varying architectures of DNNs are used depending on application. Recurrent Neural Networks are configured in such a way that data flow is enabled in every direction, not limited to only forward connections, and is used for applications involving language modeling. Convolutional Deep Neural Networks are particularly effective for computer vision and natural language processing.

A Spiking Neural Network is a DNN with neuron models inspired by the biological neuron model. These neuron models are designed to mimic the nervous system, which transmits sharp electrical potentials across the cell membrane, which are called action potentials or spikes. These spikes are transmitted across the axon and synapses to many other neurons, the primary information processing units. LIF neuron models are such spiking neuron models that are mathematically simpler. The membrane voltage is described as a function of input current, without being dependent on biophysical processes that decide the action potential.

2.4 Hardware Implementation of Neural Networks

Hardware implementation of a neural network is a task that depends on the targeted application and the resource requirements. When comparing a Spiking Neural Network and an Artificial Neural Network (ANN), spiking hardware is intrinsically sparse in its activity. Low power machine applications benefit from this, along with better performances in accuracy and computation cost on event-driven datasets. Therefore, the hardware requirements differ in terms of area, power, flexibility, etc.

Relevant applications include neuromorphic sensor processing, and signal analysis, such as silicon retina, silicon cochlea, sonar, radar, lidar, etc. The SNN for this application focuses mainly on basic building blocks useful to run the streaming tasks as in [38], but as a first step, the accelerators designed are benchmarked on the MNIST digit recognition dataset (Modified National Institute of Standards and Technology) [5]. DBNs are composed of many processing elements and even more synaptic connections. With all the neurons operating parallelly, the number of computations is Giga Synaptic Operations per Second (GSOPS). This does not suit the sequential nature of a general CPU, and the resource constraints and energy requirements are not met on a CPU. GPUs, on the other hand, can handle the computation due to the parallel computing architecture. Still, it is a power-hungry system that does not suit the application and the event-driven nature of SNN.

Implementation of SNN on custom hardware has proven to improve the network's final accuracy while reducing the memory requirements [6], [16]. This is because a custom hardware design can provide the versatility required for this specific need. It can be a parallel computing architecture optimized for event-driven configurations and meets the required power constraints. This increases the efficiency in terms of computation, energy, and latency. The event-driven approach also does not need a digital clock since it is asynchronous. When designing the custom hardware on a Field Programmable Gate Array (FPGA), the system configurations' changes can be quickly adapted without incurring additional hardware costs.

2.5 Memory Elements in FPGA

Memory Efficiency is defined as the throughput achieved per unit on-chip memory consumed. High throughput means more data bits can be transferred per unit time or clock cycle. When it comes to an FPGA, different types of memory are available, each having its own pros and cons. Some of the memory elements in FPGAs are Flip-flop, Distributed RAM, Block RAM (BRAM) and Ultra RAM.

Flip-flops are the default memory elements for saving a digital state. A single flip-flop stores a single bit and are synchronous in nature. They are ideal for smallest memories since their numbers are limited. Flip-flops are also spread throughout the FPGA which makes it difficult to route larger memories. Flip-flops also do not support multiple ports for read and write operations. Distributed RAM is built with Look-Up Tables (LUTs) and are mainly employed in creating the logic of a design. A LUT with 6 inputs can store upto 64 bits. It supports read from upto four ports but writes are limited to a single port (depending on the FPGA). Distributed RAM is ideal for fast buffers because it supports asynchronous read and values can be used immediately, but are not suited for large memories.

Block RAM is implemented using dedicated circuitry on FPGAs and are better than distributed RAM for larger memories. BRAMs support true dual-ports (2 read/write ports with independent clocks) and are flexible in organisation [9]. For example, 36 Kb BRAM blocks can have data widths of 1, 2, 4, 9, 18 and 36 bits, of varying memory depths, depending on which the amount of data that needs to be stored could be increased or decreased. If usage is restricted to simple dual-port (one read and one write port) then data width can be upto 72 bits. For a reduced maximum data width, 36Kb block can be split into two blocks of 18Kb each. Ultra RAM has bigger capacity (usually 8 times regular BRAM) compared to BRAM but less agile. Ultra RAM also does not support independent clocks when dual-ported.

2.6 Gyro

Gyro is a custom event-driven architecture that implements a spiking DBN on an FPGA, and it is the focus of this project. The usage of SNN in Gyro implies that the network is capable of high computation and that power and energy requirements are kept in check. The input spikes are received from event-driven neuromorphic sensors. The network topology includes both feed-forward and feed-back connections based on layered structure of fully connected neurons, with the weight connections, mapped to on-chip memories. The neuron model is a LIF neuron, and this being a simplified model, the hardware resource requirements are reduced [3]. Calculating membrane potential requires exponentiation, which is approximated using bit-shifts. All neurons in a layer are updated simultaneously. Using an event-driven algorithm means that unnecessary computations are eliminated. The main focus of Gyro is on flexibility, and it can deploy arbitrary sized SNNs with a configurable trade-off between performance and hardware resource requirement.

Gyro uses a digital spiking neuron model based on an event-driven LIF neuron algorithm as its computing element. All the neurons in a layer are fully connected and are therefore updated simultaneously. The implemented neuron model also avoids negative values for the membrane potential. A time-driven algorithm loops through the time-steps for the input spikes, while an event-driven algorithm loops through the spike events, ensuring that computations happen only on demand. The membrane potential leakage is implemented using hardware efficient bit-shifts as opposed to using Tailor series, which can be resource heavy and time consuming. Usage of refractory time period means that the membrane potential remains zero after an output spike, even if it has received new input spikes. A behavioral model of the LIF neuron model can be seen in figure 2.2.

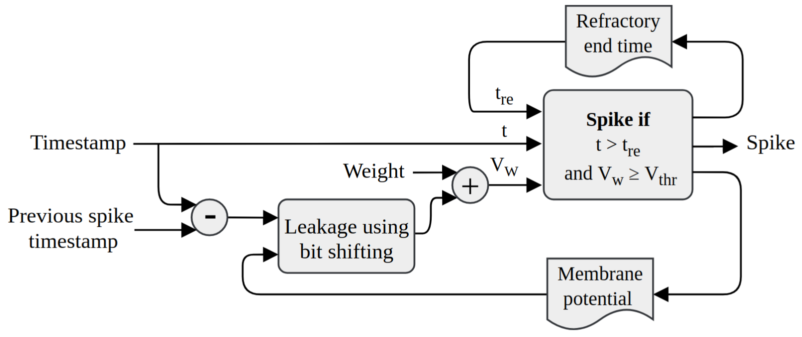


Figure 2.2: LIF neuron model

The synaptic weight are distributed over the on-chip memories in such a way that it determines the memory bandwidth and the system throughput. The synaptic weight matrices contain the weights for both forward and backward connection. Trade-off can be made between forward and backward weight accesses, depending on how they are mapped in the memory. For example, in figure 2.3, a simple 4 by 4 network can be seen, along with 2 patterns in which the synaptic weights can be stored. In the first pattern, when a neuron in the visible layer V_n spikes, all the weights can be read in a single read. This gives a throughput of 4 neuron updates per clock cycle. However, when a neuron in the hidden layer H_n spikes, 4 reads are required to fetch all the weights, and the throughput is one neuron update per clock cycle. In the second pattern, only two reads are required for both the scenarios, and the resulting throughput is 2 neuron updates per clock cycle.

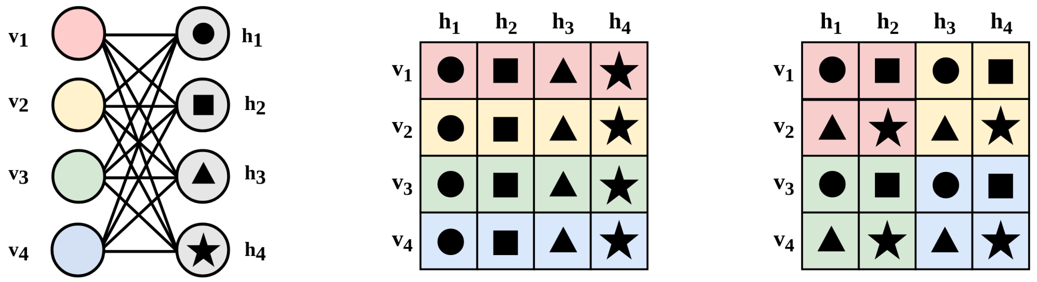


Figure 2.3: Sample network and weight memory mapping patterns

Since gyro implements networks in layered structure, seen in figure 2.4, there is a layer module for each layer of the SNN. It consists of spike queues, which receives one or two streams of input. The weight controller module fetches the synaptic weights from the memory, depending on the source of the input spikes. The neuron wrapper contains the LIF neuron cluster along with the spike arbitration. The timer module provides temporal information, which is required for the calculations in the neurons. The AXI4-lite interface is used to configure the parameters of the network with a set of registers. The main input of the network are spikes that stream into the spike queue at the input of the first layer module. The spikes can either be streamed in directly or they can be generated using mean-rate spike generators. The output of the network are the spikes at the output layer. There are three ways to monitor the behavior of the output layer. The literal spikes can be streamed through a buffer and the Inter-Spike Interval (ISI), and a low-pass filtered membrane potential is computed for each neuron at the output layer. Each weight memory contains one or more true dual-port on-chip memories.

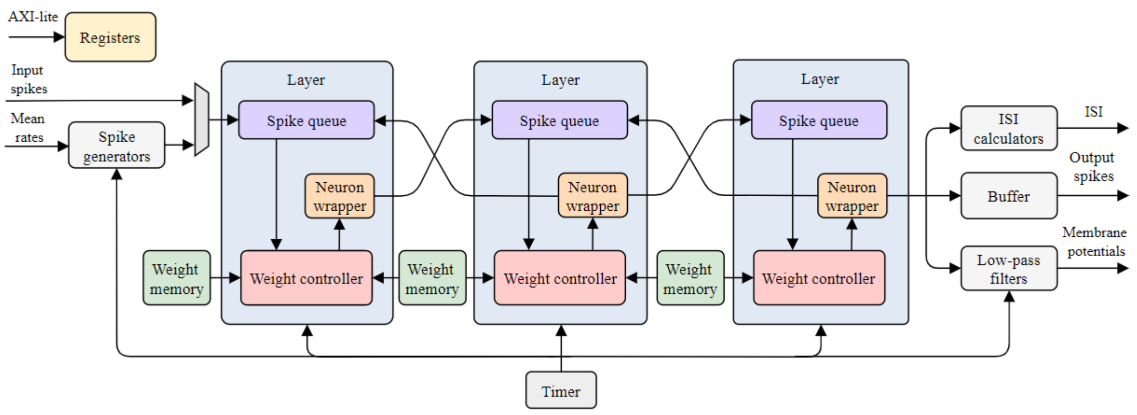


Figure 2.4: Gyro SNN architecture

The performance metrics [5] for the Gyro architecture is as follows. A SNN of 784-330-330-10 (i.e. an SNN with four layers, layer 1 has 784 neurons, layer 2 and 3 has 330 neurons each, and output layer has 10 neurons, each layers having fully connected neurons between them) configuration uses 53466 LUTs, 48310 Flip-Flops, 100 BRAMs, and 10 DSPs in terms of resource consumption and performance. This network has a clock frequency of 250 MHz and uses 2008 mW of power. A feed-forward network of 2954 neurons and 1,608,440 synapses achieved a peak throughput of 40,71 GSOPS. The energy efficiency varies depending on network size, mapping parameters, and clock frequencies, from 0,050 nJ/SO to 0,151 nJ/SO. Gyro achieved a classification accuracy of 99,3 percent on the MNIST dataset using a network of 2954 neurons and 99,7 percent on the sensory fusion task of cropland classification.

Chapter 3

Related Work

Many implementations of SNNs on FPGA, including the current state of the art and other hardware realizations were explored to understand the working principles and trade-offs of previous realizations. The optimization of various architectures was usually for performance gains and in some cases, application dependent. Different neuron models were also used, and some architectures focus on bio-realistic implementations. Some notable ones are listed in table 3.1.

An architecture with hybrid updating algorithm [10], that is a combination of time-stepped algorithm and event-driven updating, eliminates the need for sorting of events with same timestamp. The model shows shows a reduced runtime latency by avoiding sorting operations required for even queues. An event queue is sorted according to the event timestamps Q_n in event-driven algorithm and in time-stepped algorithm, each neuron is updated at fixed time intervals. For the hybrid model, multiple event queues with timestamps are used, which eliminates the need for sorting of events with same timestamps. These event queues are managed using a global time-steps, where at each time-step, queue tagged with Q_0 is processed and current time-step finishes, after which, tags of all queues is decremented by 1. This model has a low power and memory requirements, and therefore focuses on applications with strict power constraints.

The minitaur [19] and n-minitaur [12] architectures use conventional event-driven models. When there is an input spike, it checks for firing after the membrane potential has been updated and time of update is stored. Membrane decay is calculated based on the time difference calculated from this stored time and is summed with the current input. Emission of spikes from a layer does not happen until all inputs to that layer have been evaluated. Spike generation is a performance-limiting step and determining recipient neurons is a memory-intensive task in minitaur. ROM lookup is table used for neuron decay. In n-minitaur however, the design complexity was reduced, and system latency was improved. The interface block was modified to handle events from multiple sensors as input. Redesign of state machines for spike heap, neurocore blocks, and weighing caching strategy provided 2.5-fold improvement. Core frequency was increased from 75 MHz to 105 MHz and operation frequency of RAM was increased to 264 MHz. Total performance improvement is 4-fold compared to minitaur. The applications vary; minitaur is suitable for embedded robotics and n-minitaur is optimized for sensory fusion, which is relevant for this project.

uCaspian [17] is an event-driven pipelined network. The components operate as a separate stage of the pipeline with global time synchronization. BRAM utilization allows the design to be flexible and modular. Within each time-step, the system controller does not proceed to its next step until every module of pipeline reports idle, after which time is incremented, and a sync pulse is sent. This method of pausing the system progress is of interest since a spike queue that should be emptied can be implemented, which requires a trigger to be processed. The highlights of this implementation are low power requirements and low cost of implementation due to utilization of off-the-shelf components. The application of uCaspian is best suitable for edge deployment applications, which is also relevant for this project.

Siegert approximation for LIF neurons for an event-driven SNN can be suitable for hardware implementation [20]. It also facilitates the demonstration of simulation of sensory fusion of silicon retina and silicon cochlea. The DBN was trained to associate the visual and auditory stimuli and integrating both input streams. The advantage is the ability to include inputs from different domains, possibly noisy or ambiguous inputs. The combination provides a more conclusive evidence of the true label rather than a single modality.

FDF architecture [36] is based on a reconfigurable neural layer, which is implemented using time-multiplexing. The neuron model performs a leaky integration of the post synaptic currents to calculate membrane potential. A single physical neuron is time multiplexed up to 20,000 virtual neurons. A trade-off between speed and network size makes it possible to make the network run faster. Limitation of time multiplexing is data storage requirement. The on-chip memory used in this architecture is limited in size, while off-chip memory is limited by bandwidth. The latency of the design is reduced by implementing a 7 stage pipelining. An address buffer is used, which generates synaptic weight on the fly without using memory.

Fast SNN [11] is a serial feed-forward network architecture. It uses inter-spike time for internal calculations and is independent of number of inputs due to the architecture being serial, which greatly reduces memory requirements. Compared to a parallel architecture, it has a disadvantage of non-parallelism only for very large networks. Temporal or threshold spike coding is used, which converts active inputs into a spike train according to the parameters set. It also uses a pipelined neuron design, which processes n inputs in $n + 1$ clock cycles, compared to the usual of one input in two clock cycles. The neuron model requires only one multiplier and 2 memory blocks for weight storage and post-synaptic fucntional model. The benchmarking was done using a pattern classification problem.

SNAVA [31] is a bio-inspired simulation platform implemented on FPGA for prototyping SNNs faster than CPUs/GPUs by exploiting the FPGA re-configurability and minimizing implementation time. A program flexible SNN architecture for efficient SNN simulation with scalability is one of the main features offered by SNAVA, which allows the simulation of large scale SNN by connecting multiple FPGA boards.Its features of interest include time-multiplexed neural computations and a distributed memory system utilizing BRAMs. Specific time slots are used for sending spikes via the shared channel to obtain the minimum error rate, thereby avoiding spike contention. The program flexibility is shown by the simulation of three SNN models with different levels of computational complexity. These features enable the simulation of large scale SNN models at high processing and communication speed.

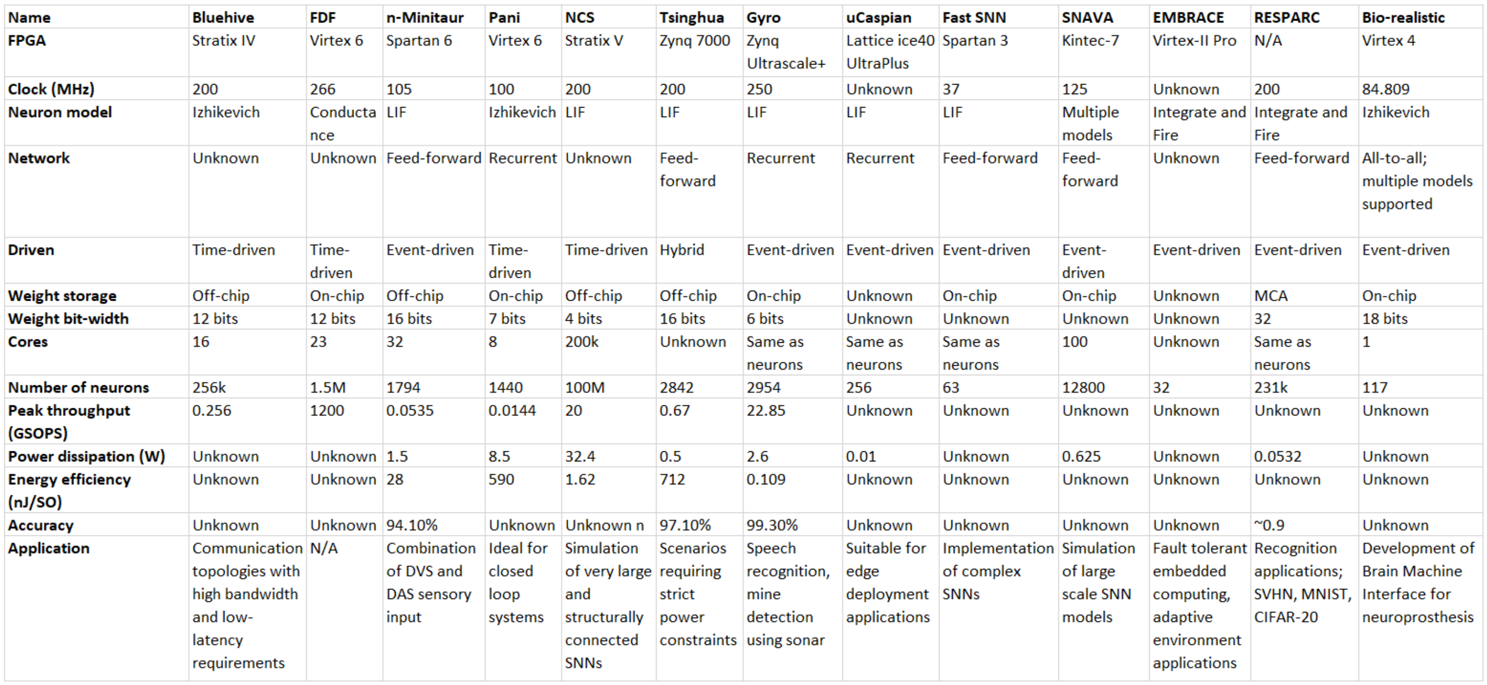
EMBRACE-FPGA [21] is a flexible, reconfigurable architecture with time-multplexed inter-neuron communication channels. The interconnects are also scalable. Neuron implementation is visualized as a 2D $N * M$ array of interconnected SNN neural tiles, each connected in 4 directions. Communication between tiles achieved by routing data packets through round-robin based ports. Network routers enable propagation of spikes from source tile to destination tile using time-multiplexing of connection lines.Spikes are queued by buffering of neuron fire requests. Reconfiguration of synapses and synaptic weight modification of different tiles is easily possible, which is facilitated by the Network-on-Chip (NoC) routers. The design has proved to be compact with low power requirements, with high synaptic densities compared to other hardware implementations. This architecture is more applicable towards fault tolerant embedded computing, adaptive environment applications, etc.

Bluehive architecture [18] is a scalable architecture for simulation of SNN in real-time with a reconfigurable communication topology. It focuses on a communication centric approach rather than computation centric design, allowing simulation of large networks. Network topologies demanding high bandwidth and low-latency communication are suitable for Bluehive. It uses Izhikevich spiking neuron model and evaluates inputs in continuous time using floating point arithmetic.

The adaptive spiking recurrent neural network (SRNN) [38] achieves a state-of-the-art performance compared to other SNNs and in some cases that of classical RNNs while exhibiting sparse activity, even though it is not a FPGA implementation. The result is attained by modeling standard and adaptive multiple-timescale spiking neurons as self-recurrent neural units. The adaptive spiking neurons were effective because of training the individual time-constants of the neurons. The adaptive spiking neuron maintains a multiple-timescale memory due to having two time constants, and this allows the memory in the network to be adapted to the temporal dynamics of the task. The network was tested with significant energy efficiency, in applications such as sequential and streaming classifications benchmarks, QTBD waveform classification of ECG, and other applications on low power always-on edge computing devices.

RESPARC [2] is a reconfigurable architecture that utilizes Memristive Crossbar Arrays (MCA) for its energy efficient implementation of deep spiking neural networks. The crossbars store the network weights which enables in-memory processing.The event-driven nature of SNNs is also utilized for achieving the energy efficiency. Different spiking network topologies with applications such as digit recognition, house number recognition, object classification etc. are mapped onto RESPARC. The design also highlights the importance of weight bit precision and how it contributes towards the energy consumption and area of the memristive device. The power and memory bottlenecks of modern computing systems are circumvented in this architecture by combining energy benefits of post-CMOS technologies and event-driven nature of bio-inspired SNNs.

The bio-realistic model [1] focuses on implementation of a bio-realistic spiking neural network composed of Izhikevich neurons which works in hard real-time, i.e., it keeps the same biological time of simulation at the millisecond scale. The architecture of network implementation allows working on a single computation core. Different networks ranging from independent-neuron configuration to all-to-all configuration or a mix of several independent small networks can be configured. This model requires only a single multiplier for its computational core, which can reproduce the required firing activities. It also allows to freely scale and configure the network, including different neuron models and different network connectivity. This architecture is used for development of Brain Machine Interface for neuroprosthesis. It is also efficient in resource consumption, and allows scaling of the network due to low resource requirements.



Name	Bluehive	FDF	n-Minitaur	Pani	NCS	Tsinghua	Gyro	uCaspian	Fast SNN	SNAVA	EMBRACE	RESPARC	Bio-realistic
FPGA	Stratix IV	Virtex 6	Spartan 6	Virtex 6	Stratix V	Zynq 7000	Zynq Ultrascale+	Lattice ice40 UltraPlus	Spartan 3	Kintec-7	Virtex-II Pro	N/A	Virtex 4
Clock (MHz)	200	266	105	100	200	200	250	Unknown	37	125	Unknown	200	84.809
Neuron model	Izhikevich	Conducta nce	LIF	Izhikevich	LIF	LIF	LIF	LIF	LIF	Multiple models	Integrate and Fire	Integrate and Fire	Izhikevich
Network	Unknown	Unknown	Feed-forward	Recurrent	Unknown	Feed-forward	Recurrent	Recurrent	Feed-forward	Feed-forward	Unknown	Feed-forward	All-to-all; multiple models supported
Driven	Time-driven	Time-driven	Event-driven	Time-driven	Time-driven	Hybrid	Event-driven	Event-driven	Event-driven	Event-driven	Event-driven	Event-driven	Event-driven
Weight storage	Off-chip	On-chip	Off-chip	On-chip	Off-chip	Off-chip	On-chip	Unknown	On-chip	On-chip	Unknown	MCA	On-chip
Weight bit-width	12 bits	12 bits	16 bits	7 bits	4 bits	16 bits	6 bits	Unknown	Unknown	Unknown	Unknown	32	18 bits
Cores	16	23	32	8	200k	Unknown	Same as neurons	Same as neurons	Same as neurons	100	Unknown	Same as neurons	1
Number of neurons	256k	1.5M	1794	1440	100M	2842	2954	256	63	12800	32	231k	117
Peak throughput (GSOPS)	0.256	1200	0.0535	0.0144	20	0.67	22.85	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown
Power dissipation (W)	Unknown	Unknown	1.5	8.5	32.4	0.5	2.6	0.01	Unknown	0.625	Unknown	0.0532	Unknown
Energy efficiency (nJ/SO)	Unknown	Unknown	28	590	1.62	712	0.109	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown
Accuracy	Unknown	Unknown	94.10%	Unknown	Unknown n	97.10%	99.30%	Unknown	Unknown	Unknown	Unknown	~0.9	Unknown
Application	Communication topologies with high bandwidth and low-latency requirements	N/A	Combination of DVS and DAS sensory input	Ideal for closed loop systems	Simulation of very large and structurally connected SNNs	Scenarios requiring strict power constraints	Speech recognition, mine detection using sonar	Suitable for edge deployment applications	Implementation of complex SNNs	Simulation of large scale SNN models	Fault tolerant embedded computing, adaptive environment applications	Recognition applications; SVHN, MNIST, CIFAR-20	Development of Brain Machine Interface for neuroprosthesis

Figure 3.1: Spiking neural network FPGA implementations

Chapter 4

Design

This chapter presents the design changes implemented to the existing Gyro architecture. The neuron model is described in the first section with the event-driven network update algorithm. Following that, the architecture is described in a bottom-up manner, covering the weight memory mapping, network layer, finite state machine, spike queue, weight controller, neuron wrapper, and the external interface.

4.1 Neuron Model

The basic neuron model is based on LIF model described in section 2.1. The LIF neuron model with minimal changes was implemented.For every spike from a neuron i in the visible layer to neuron j in the hidden layer, the membrane potential of neuron j is updated. It is increased or decreased with a step of size W^{ij}. The event-driven network update algorithm was adapted for improved functionality and reduced resource consumption. Previously, every input spike was represented as a combination of a timestamp and the neuron source address. In the updated neuron model for modified Gyro architecture, the time element is removed from every calculation. Timestamp was also used to calculate the decaying membrane voltage of the neuron. Removing timestamp allows the system to take advantage of its sparsity in time and thereby reduces the resource consumption and reduces the number of mathematical operations performed per input spike.

Figure 4.1 shows the block diagram for the updated neuron model. The dotted elements and gray modules were removed. These were timestamp input, refractory time calculation block and membrane voltage decay module.

The neurons in a layer are updated simultaneously since it is a fully connected network. To include the functionality of neuron evaluation upon a trigger signal, the algorithm is modified by removing time-dependent values and calculations. The membrane potential is set to zero if it goes to a negative value. Since it is an event-driven algorithm, the neuron model performs the computations only upon spiking activity.

Algorithm 1 Event-driven LIF neuron update independent of time

```
Require: set of Source neurons $S_{src} \in 1..N$ corresponding to every Input spike $I_s$
Require: set of Destination neurons $S_{dst} \in 1..M$
Require: set of synaptic weights $W \in R^{NxM}$
Require: active high toggle signals $tick$ and $decay$
  for $f$ in $I_s$ do
      for $s$ in $S_{src}$ do
          for $d$ in $S_{dst}$ do
              $V_m^i \leftarrow V_m^i + W^{S_{src}^k,i}$
              if $tick = 1$ then
                  if $decay = 1$ then
                      $V_m^i \leftarrow V_m^i \cdot 0.5$
                  end if
                  if $V_m^i > V_{thr}$ then
                      Spike()
                      $V_m^i \leftarrow V_{reset}^i$
                  else if $V_m^i < 0$ then
                      $V_m^i \leftarrow V_{reset}^i$
                  end if
              end if
          end for
      end for
  end for
```

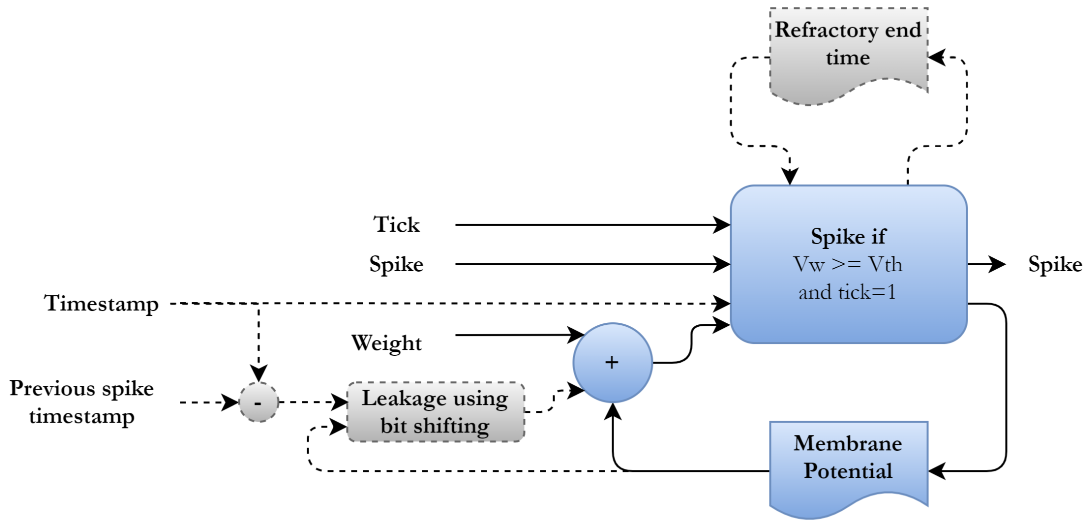


Figure 4.1: Revised neuron model

The neuron now evaluates input spikes based on an internal trigger signal. This signal, 'tick', triggers the neuron to compare its membrane voltage to the threshold value depending on which an output spike is produced. This provides more control over the system since this signal can also be modified to be a user input and/or an external signal. Until a ‘tick’ signal is received, the neuron accumulates its membrane voltage. So, in definition, the time-period between two ‘tick’ signals are considered as a chunk of time and all the input spikes that is inside that period is accumulated until the ‘tick’ signal is received, upon which an output spike is produced depending on whether the membrane voltage is above the threshold voltage or not.

The neuron model follows the algorithm in Algorithm 1. No action is performed until an input spike or tick signal is received. Upon receiving a spike, it updates the membrane voltage by adding the input synaptic weight to the current membrane voltage. This cycle is followed and membrane voltage is incremented until a tick signal is received. This triggers the comparison of the membrane voltage against the threshold voltage and an output spike is produced depending on the comparison result.

The decay of membrane voltage is implemented using a shift register which calculates a 50 percent decay before evaluation against threshold. Since this is a fixed decay and not a time-dependent value, no dynamic calculation is needed. Decay of the neuron voltage can be enabled or disabled depending on the use case scenario. Decay is not calculated for every input spike, thereby eliminating redundant computation. The neuron model also discards usage of refractory time since the time element itself is removed from the design. This enables immediate processing of input spikes without any delay, and no input spikes will be unaccounted for.

4.2 Architecture

One of the many advantages of using custom hardware is the versatile nature of the different types of available on-chip memories. In FPGAs, the on-chip memories being close to computations achieve lower latency. Utilizing on-chip memories also provides higher memory bandwidth when compared to off-chip memory. Gyro architecture therefore opted on-chip memories for storing the connection weights, and the way in which continuous weights are stored determines the memory bandwidth which affects the performance of the entire network. Gyro focused on storage of connection weights in Block RAMs or BRAMs. BRAMs were initialised upon synthesis with the trained network connection weights. In this modified iteration of Gyro, usage of flip-flops for storage of weights are explored. The advantage using flip-flops is that the required memory can be immediately accessed depending on how it is stored, thus improving the parallelism of weight storage and fetching. There no significant increase in resource consumption due to the sparse nature of the network and most of the values are zeroes.

The second iteration of Gyro maintains the layered structure as in the previous version, which was based on layered structure of DBNs. The network has fully connected layers, i.e., there is an all-to-all connection between neurons in one layer to the neuron in the previous and next layer. The neurons within a layer are also fully connected which constitutes the recurrent connection of each layer. What this means for the network is that when a neuron spikes, it is sent to all the neurons in the next layer and also to every neuron within the same layer, depending on whether recurrent connections are enabled or not. The design mainly consists of two modules: the layer and weight memory. To facilitate proper timing and processing of spikes, the layer is designed to have proper interface at both the input side and output side, either of which could also be another layer. A layer module is instantiated for every layer except the input layer since it does not require any computations. Between every layer module, a weight memory module is required to store the connection weights between the two layers. So for a network of N layers, there are N-1 layers and N-1 weight memories.

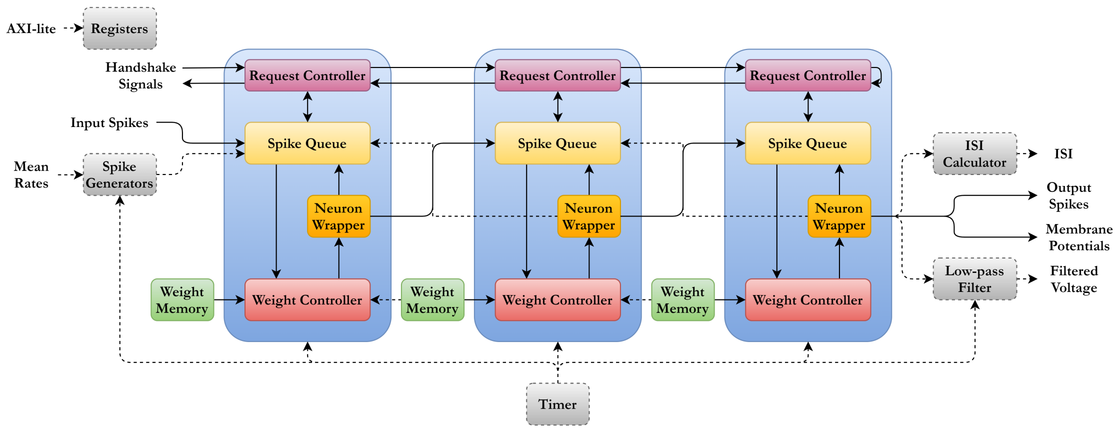


Figure 4.2: Network instance with 3 hidden layers

A sample network instance of 3 layers with its sub-modules can be observed in figure 4.2. The layer module contains the following sub-modules: input spike queue, weight controller, neuron cluster and output spike buffer. The working of layer module and interfacing at the input and output side is controlled by a finite state machine or request controller. The input spike queue acts as a buffer to store the incoming spikes until they are ready to be processed. It contains separate queues for forward spikes and recurrent spikes. The weight controller fetches the right weights from the weight memory by decoding the memory address depending on the source address of the input spikes. The neuron addressing module inside the weight controller generates weight valid signals for each cluster of neurons to facilitate activating the neuron corresponding to the connection weight. The neuron cluster, which has the LIF neurons, and output spike buffer is what constitutes the neuron wrapper. The parallel output from the neurons are converted to serial and then are sent out as the output of the layer upon request from the output side. The layer module is further elaborated in section 4.4. The timer module present in Gyro architecture is removed in its entirety since the modified architecture of Gyro has no dependency on time for its calculation of membrane potential or voltage decay.The dotted signals and gray modules in figure 4.2 were removed from the Gyro architecture, namely, the backward spike propagation, backward BRAM access and timer module. The network interfaces at the input and output were also modified such that AXI-lite registers, spike generator, ISI calculator and low-pass IIR filter were also removed, since these modules did not fit with the working principles of the new design.

The main input of the network is received as a stream of spikes by the first layer, with the output spikes being the same data type, received at the output side of the network. The behaviour of the output layer can be monitored in two ways. The literal spikes can be streamed through a buffer, and membrane potential computed for each neuron at the output layer. The weight memory module remains as a dual-port on-chip memory. However, 2 layers are not simultaneously accessing the same weight memory since the architecture now supports recurrent connections, and the weights for which are stored in the same weight memory.

4.3 Weight Memory Mapping

Weight memory mapping is the key enabler of Gyro and Gyro II. The pattern in which weights are stored and read from the on-chip memory is crucial when it comes to the latency of the system and memory bandwidth. Each of the weight memory modules stores a weight matrix that represents the connections between two layers and the recurrent connections.

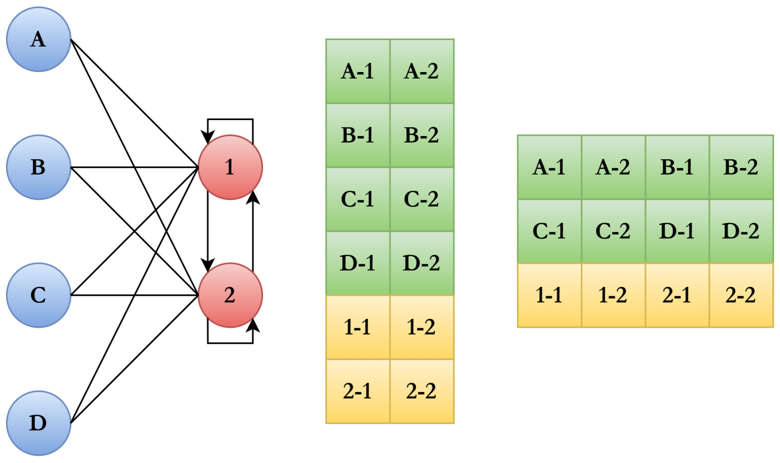


Figure 4.3: A sample network of two layers and 2 weight mapping patterns

Consider the example of a simple four by two network with the corresponding weight mapping patterns shown in figure 4.3. There are 4 pre-synaptic neurons and 2 post-synaptic neurons. Forward and recurrent connections are present to the post-synaptic neurons. In the weight mapping patterns in the figure, one row of weights can be read per clock cycle. For the first mapping pattern, when a neuron in the visible layer spikes, each of the two neurons in the hidden layer needs to be updated and the corresponding weight can be read from one row in one clock cycle. When a neuron in the hidden layer spikes, the same procedure applies, where the weights for two neurons can be read in a single clock cycle. The resulting throughput is therefore two neuron updates per 6 clock cycles for both forward and recurrent spikes, or two neuron updates per 4 clock cycles for a feed-forward network, as it effectively reads one weight per clock cycle.

For the second mapping pattern, when a neuron in the visible layer spikes, 2 weights can be read from one row in one clock cycle. When a neuron in the hidden layer spikes, the weights for two neurons can be read in a single clock cycle. The resulting throughput is two neuron updates per 3 clock cycles for both forward and recurrent spikes, or two neuron updates per 2 clock cycles for a feed-forward network, as it effectively reads two weights per clock cycle.

To make the weight mapping generalized, a parameterized method is opted. For storing the weight matrices between V visible neurons and H hidden neurons, the memory is visualized as a three-dimensional block with x, y and z coordinates as in figure 4.4. X represents the memory width of a single weight. Y represents the number of weights in a single row of the memory, or memory width, and Z represents the number of distinct memories, where all the values x, y and z are integers. The memory width is represented in number of weights and can give you the width in number of bits when multiplied by the bit-width of a single weight.

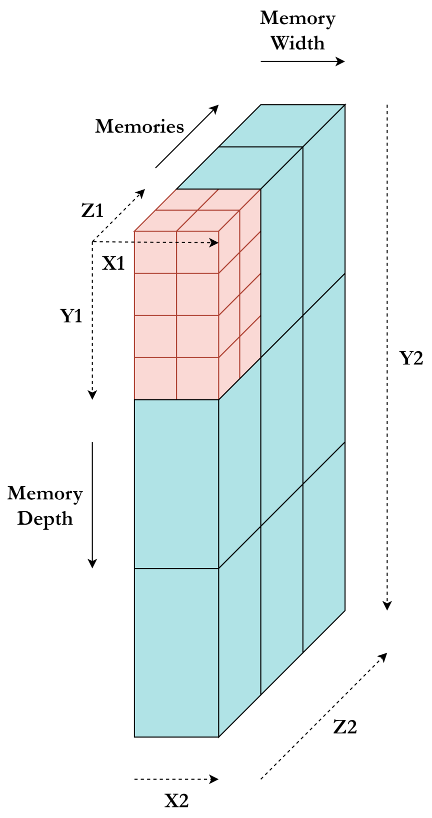


Figure 4.4: 3-dimensional visualization of memory

When visualized in a three-dimensional manner, as in figure 4.4, the first cube with dimensions x_1, y_1, z_1, contains the set of H weights corresponding to a single visible neuron V. The volume of this cube is the number of hidden neurons H, $x_1 \cdot y_1 \cdot z_1 = H$. This cube is repeated x_2 times in x dimension, y_2 times in y dimension and z_2 times in the z dimension, for the entire forward weight matrices between the 2 layers. The number of repetitions gives the amount of visible neurons, since there is a single cube for each forward spiking neuron. That is, $x_2 \cdot y_2 \cdot z_2 = V$. To update a set of hidden neurons upon spiking of a forward neuron, x_1 by y_1 weights are read from z_1 memories.

The recurrent connections are stored with an offset in the y dimension but in the same pattern. That is, the cube for a single recurrent spiking neuron is represented in the same way as a forward spiking neuron, with x_1, y_1, z_1, and is repeated x_2 times in the x dimension, y_2_r times in the y dimension and z_2_r times in the z dimension. y_2_r and z_2_r is derived from the previous parameters, where y_2_r is minimum of y_1 and z_1 times x_1, and z_2_r is maximum of y_1 and z_1. Maximizing z parameter utilizes more distinct memories and increases the number of weights read parallelly in a single clock cycle and effectively increasing the memory bandwidth. To update a set of hidden neurons upon spiking of a hidden neuron, x_1 by y_1 weights are read from z_1 memories like for forward spikes, but difference being the memory locations and weights themselves.

The parameters x_n and z_n indicates effective weights read per clock cycle, which means that the number of neurons that are simultaneously updated per clock cycle is $x_n \cdot z_n$, which is the neuron cluster size C. The weight memory mapping parameters and variables are listed in table 4.1 along with the constraints. It is written as an optimization problem and the objective is maximum throughput for both forward and recurrent connections.

Variable	**Type**	**Description**
$x_1, y_1, z_1, x_2, y_2, z_2, y_2\_r, z_2\_r$	Integer	Memory mapping parameters
C_1	Integer	Forward neuron cluster size, i.e., $x_1 \cdot z_1$
C_2	Integer	Recurrent neuron cluster size, i.e., $x_2 \cdot z_2$
M_x	Integer	Maximum memory width in bits
M_y	Integer	Maximum memory depth
M_z	Integer	Maximum number of distinct memories
V	Integer	Number of neurons in visible layer
H	Integer	Number of neurons in hidden layer
W	Integer	Weight bit width

Table 4.1: Variables related to weight mapping

$$x_1 \cdot y_1 \cdot z_1 >= H, \tag{4.1}$$

$$x_2 \cdot y_2 \cdot z_2 >= V, \tag{4.2}$$

$$x_1 \cdot x_2 <= M_x/W, \tag{4.3}$$

$$y_1 \cdot y_2 <= M_y, \tag{4.4}$$

$$z_1 \cdot z_2 <= M_z, \tag{4.5}$$

$$y_2\_r = Min(y_1, z_1) \cdot x_1, \tag{4.6}$$

$$z_2\_r = Max(y_1, z_1) \tag{4.7}$$

$$x_2 = 1 \tag{4.8}$$

4.4 Layer

The layer is the computational core of the design. The structure of the layer with its sub-modules can be seen in figure 4.5. It contains the state machine, input spike queue, weight controller, the LIF neuron cluster and output spike queue, along with the external interfaces. Each of the sub-modules and its functions are explored in the sections below. In the figure 4.5, the gray modules and dotted line connections represents connections and modules which was removed. They are backward BRAM access and round-robin arbiters for scheduling queue reads.

The interface of the layer at the input and output sides were modified from the Gyro architecture to suit the needs. A 4-phase handshake is implemented to implement a proper processing of spikes. The internal structure was changed to accommodate recurrent connections instead of backward connections. The output spikes are buffered and source addresses for output spikes are generated inside the neuron wrapper using a parallel to serial shift register.

4.4.1 Finite State Machine

The finite state machine controls the internal functioning of the layer and manages and observes the external interface. The finite state machine takes into account all the possible states the entire layer system can attain and controls the sub-modules through the control signals. It manages the external interface based on its internal state, and control signals from the input and output interface, which could also be interface of another layer. The transition diagram of the state machine can be seen in figure 4.6.

The state machine contains 5 states: Idle, process, communicate, acknowledge, and finish. The state diagram can be observed in figure. Each state and its function, along with control signals are as follows:

- **Idle**: In idle state, the layer is ready to receive input spikes from the user input or the previous layer. It accumulates the spikes in the spike queue, and a maximum of V spikes in forward spike queue and maximum of H spikes in recurrent spike queue are accumulated. These spikes are represented as neuron address in the queue. All the control signals are reset or initialised in idle state.
- **Process**: When a request is received from the input or previous layer, it transitions from idle to process state. In process state, the neuron spikes are processed, and the membrane voltage is accumulated with each of the input spike received. The spikes in the queue received before the request signal are processed until both recurrent and forward spike queue are empty. The spikes received after receiving a request signal are not processed until the next request is received and are buffered in the queue until then. The transition occurs when the input queues are emptied.

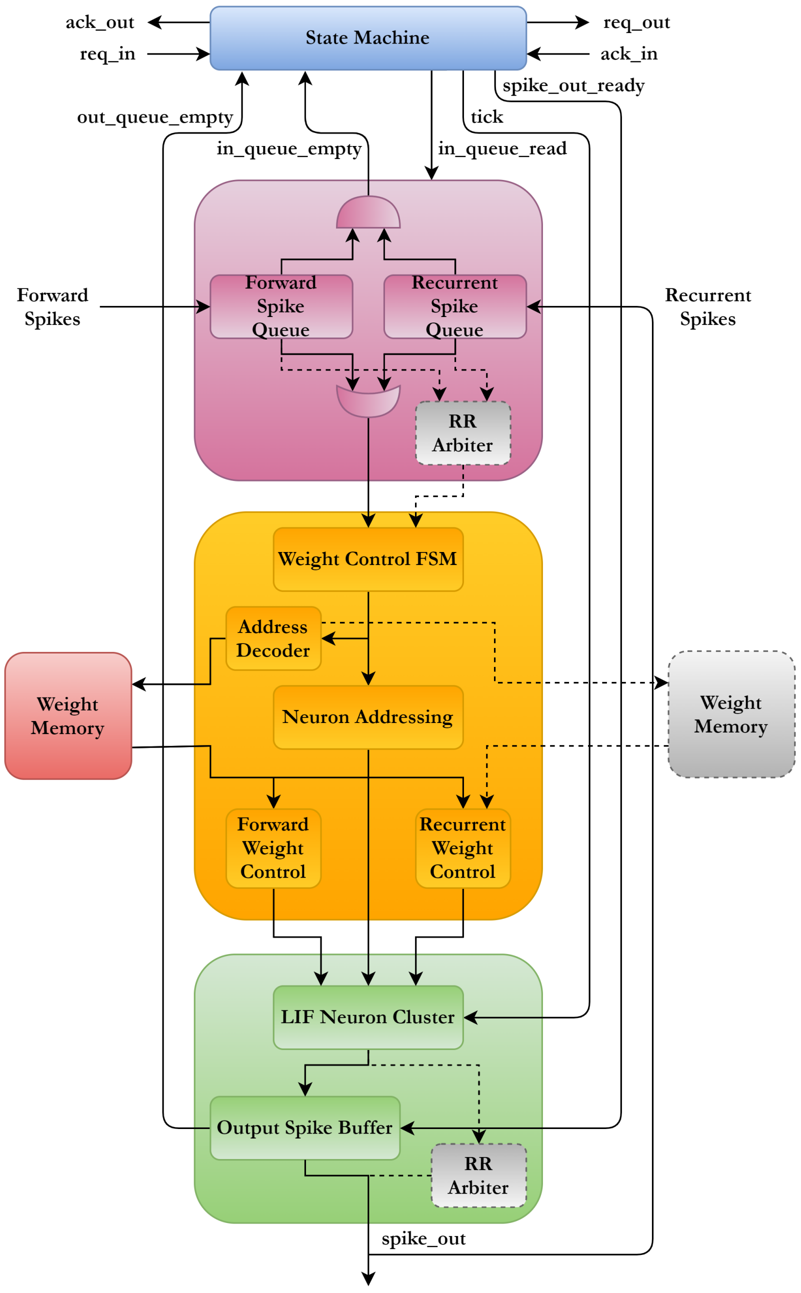


Figure 4.5: Structure of a network layer

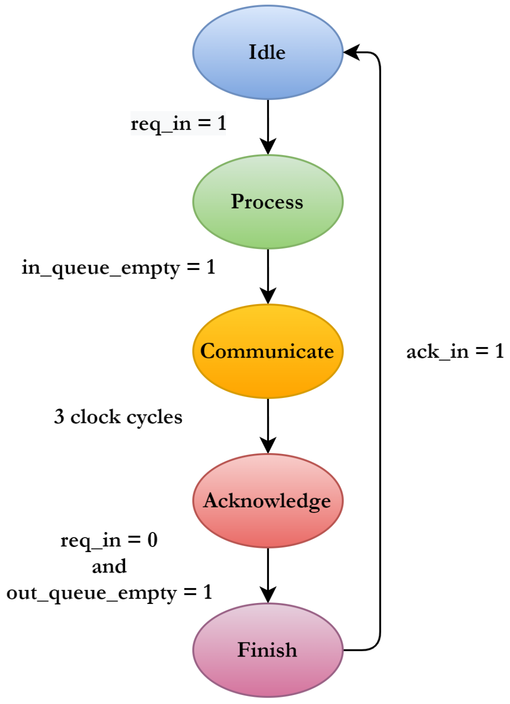


Figure 4.6: Finite state machine of a layer

- **Communicate**: once the input spike queues are emptied and neuron voltages updated, the state machine triggers the threshold evaluation of the LIF neurons through the 'tick' signal and thereby conveys to the neurons that the spikes are ready to be evaluated. This evaluates the neuron membrane voltage against the threshold voltage and generates an output spike depending on whether it is beyond the threshold voltage or not. If a neuron spikes, the output spike is stored in the output spike queue as the address of the neuron that spiked. This buffering of output spike enables the layer to send a burst of output spikes upon receiving a 'ready-to-receive' signal from the output or next layer.

- **Acknowledge**: once the neurons complete evaluation of spikes against threshold, which takes 3 clock cycles in communicate state, the layer sends the output spikes out. This is signalled by setting the 'ack' signal as high to the input side to convey that data processing is complete (to complete the request-acknowledge 4-phase handshake). The output spikes are sent as a burst of spikes to the output or next layer (which is in its idle state, accumulating input spikes). Acknowledge state is maintained until the output spike queue is empty and the request signal is zero at the input side.

- **Finish**: in finish state, the request signal for the next layer is set as high, which triggers the next layer into the same state transitions through which it processes the input spikes received from the current layer. Upon receiving an acknowledge signal from the next layer or output, the layer goes back to idle state to reset its control signals. This completes the request-acknowledge handshakes on both input and output sides of the layer and thereby processing of a set of input spikes.

4.4.2 Spike Queue

The spike queue receives two streams of spikes, one for forward spikes from the previous layer and one for recurrent spikes from the same layer. A spike is represented by the address of the neuron the spike originated from, represented as S_{src}. Each stream is separately buffered using a FIFO. The input queues are emptied based on triggers from the state machine. This is done in such a way that the recurrent spikes are processed first, followed by the spikes in the forward spike queue. The maximum number of spikes in both the queues are monitored to ensure that no input spikes are left out from being processed.

Since it is the state machine that schedules the reading of the spike queues, the round-robin arbiter is no longer required in the design. Arbiter also reads the queues in parallel which is not suitable for the design. Recurrent spikes and forward spikes need to be processed separately and sequentially. Removal of the round-robin arbiter is also beneficiary since it is resource heavy for queues with large values of queue depth.

The input of recurrent spike queue is an internal signal. The output from the neuron wrapper is sent to both the output of the layer and to the recurrent queue. The maximum length of the queues are also parameterized. The maximum queue length for forward spike queue is same as the number of inputs, since the maximum number of input spikes at a time cannot exceed the total number of inputs. The maximum queue length for recurrent spike queue is same as the number of neurons in the same layer, since the maximum number of recurrent spikes at a time cannot exceed the total number of neurons in the layer.

4.4.3 Weight Controller

The weight controller performs the function of fetching the right weights from the memory depending on the spiking neuron. The behaviour of the weight control module is controlled by a finite state machine (figure 4.7 shows the state transitions), which synchronizes the sub-modules and manages its interface with other modules within the layer. The spike source address and spike direction are latched from the spike queue in the initial state. During decode 1 and decode 2, memory address is derived from source address by the address decoder. The y index of the weight is derived in forward or backward state to multiplex the weights that are read from the memories.

The structure of weight controller can be seen in figure 4.8. The state machine latches the spike source address and the direction to the address decoder. The address decoder also receives the y-indices from the state machine, and it generates the memory addresses for forward and recurrent weights. The weight control modules fetches the weights as vectors and then provides the right weights to the neurons for spike processing, while the neuron addressing module makes sure that the right neuron receives the intended weight value.

Compared to the Gyro architecture, the weight memory mapping is same for forward connections. However, the recurrent weights are stored in a different pattern in the current architecture,

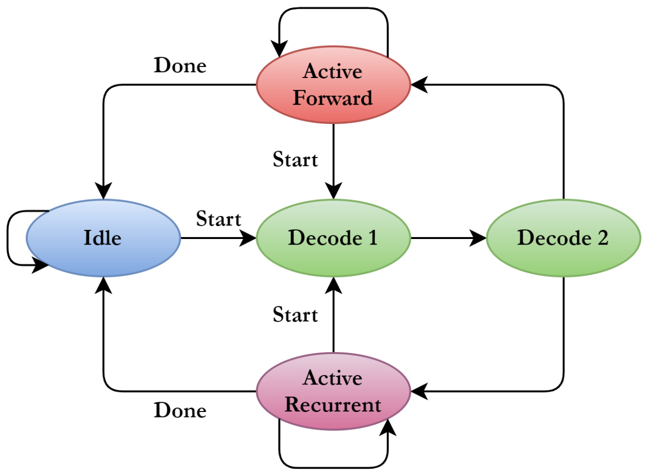


Figure 4.7: Finite state machine of weight controller

and it follows the same pattern as the forward connections. The difference is in the y_2 and z_2 parameters. In the memory, the recurrent weights are also stored 'beneath' the forward weights, i.e., the y indices of recurrent weights have an offset of $y_1 \cdot y_2$.

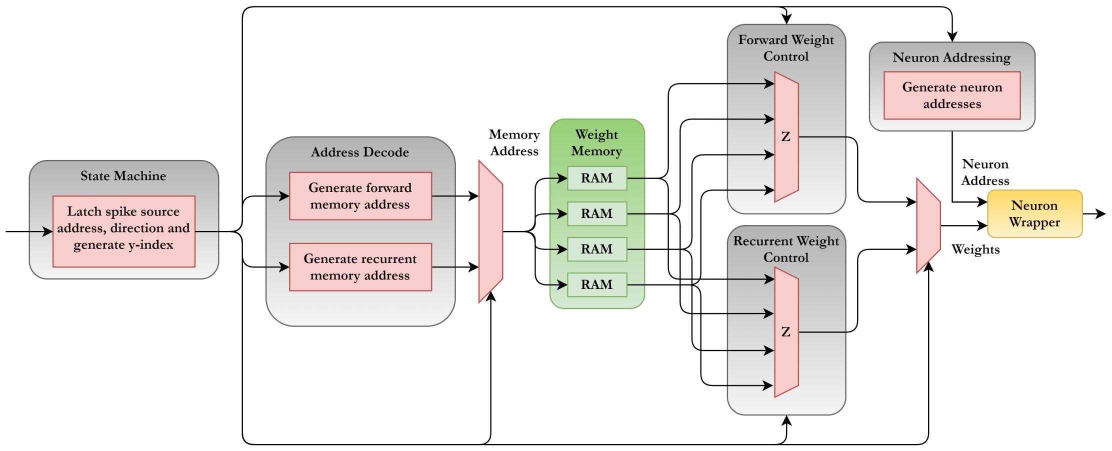


Figure 4.8: Weight controller structure

The weights corresponding to a forward spike are the weights within a virtual cube. $x_1 \cdot z_1$ weights are read in parallel, and the address decoder module iterates through the rows, i.e., the y index, until the entire cube is read. Index y_f from state machine counts from 0 to $y_n - 1$, going row by row. To obtain the weights from a cube in y dimension, y offset is added to the y index. Dividing S_{src} by z_2 gives the cube offset in the y dimension and multiplying this by cube height y1 gives the row offset within the cube. The forward address is thus generated as:

$$\text{Forward memory address} = y_f + \frac{S_{src} \cdot y_1}{z_2} \tag{4.9}$$

For a recurrent spike, the same calculation applies, with y offset being $y_r = y_f + y_1 \cdot y_2$. The recurrent address is therefore generated as:

$$\text{Recurrent memory address} = y_f + y_1 \cdot y_2 + \frac{S_{src} \cdot y_1}{z_2} \tag{4.10}$$

In forward and recurrent weight control modules, data is received from memories and is multiplexed to filter out relevant weights. The cube offset in z dimension is calculated in the same way for both forward and recurrent weights as:

$$z_f = S_{src} \bmod z_2 \tag{4.11}$$

$$z_r = S_{src} \bmod z_{2\_}r \tag{4.12}$$

There is no multiplexing in x dimension since x_2 is fixed to 1. However, the weight offset in x dimension is calculated as:

$$x = S_{src} \bmod x_1 \tag{4.13}$$

The neuron addressing module generates weight valid signals to control the neurons and the weight each neuron receives. It is generated from a counter which counts from 0 to $max(y_1, y_2)$ with step size of 1, i.e., reading weights row by row in the memory. Depending on the weight valid signals, the respective neuron cluster is activated.

By implementing a weight memory mapping in this format, the Gyro architecture was corrected. Previous implementation consisted of a backward connection between layers which was not optimal for the use case scenario. This was removed and instead, a recurrent connection was implemented. This however meant an increase in resource consumption because recurrent connection requires to store the weights in a square matrix of dimension equal to the number of neurons in the hidden layer, compared to a one-to-one connection to the previous layer.

4.4.4 Neuron Wrapper

Neuron wrapper consists of a set of LIF spiking neurons along with parallel to serial converter to buffer and serialize the output spikes, as can be seen in figure 4.9. This is because the neurons function in parallel and the output spikes from a layer has to be a serial stream of spike source addresses.

To serialize the output, a N bit parallel in serial out shift register is implemented, where N is the total number of neurons in the layer. It takes in the output spikes from N neurons as N bit vector. The shift register then shifts the vector to the right and depending on the bit value, it determined whether neuron number n spiked or not. For every neuron that spiked, an output spike is generated as the address of the neuron that spiked. The output spikes are sent out one per clock cycle.

For a single request signal at the input side, the maximum number of spikes that can generate from a layer is the total number of neurons, since each neuron is evaluated against the threshold only once per request. This therefore eliminates the need to multiplex the output spikes between the neuron clusters.

Gyro architecture implemented output spike arbitration using a round-robin scheduler. This was resource heavy in terms of flip-flops and amount of logic required to implement the arbiter for large values of N, and thus lowers the maximum clock frequency. This was overcome by using a parallel to serial shift register.

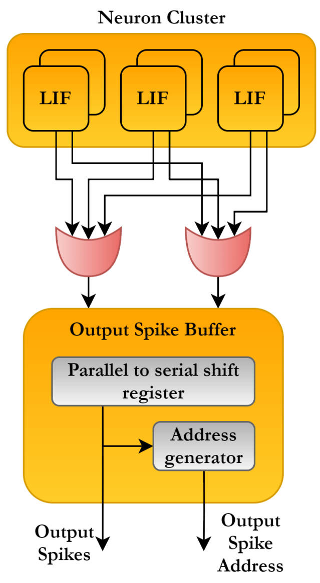


Figure 4.9: Neuron wrapper with output spike buffer

4.5 External Interface

The network interface at both input and output sides were modified from Gyro since the core computing element itself changed in terms of how the input is received and processed. This also changes how the output has to be interpreted at the network output interface. The interfaces will be covered in the following sub-sections.

4.5.1 Network Input

The input spikes of the first layer are encoded as a combination of a binary valid signals and a source address of the input neuron. The binary valid signal avoids processing of any unwanted or erroneous input for source address. Source address conveys which input neuron spiked, just as the name implies. For each valid input source address, the spike valid signal should have a rising edge since the source address is sampled or latched by the system at every rising edge of the spike valid signal. So for a continuous stream of input spikes, the spike valid signal will resemble a clock signal, where it oscillates or completes a cycle for every valid source address. If the spike valid signal maintains a binary value of '1' and does not have rising edge for a continuous stream of valid source addresses, the input spike queue would simply treat it as one input spike, with the source address being the source address during the rising edge of the spike valid signal.

The processing of the spikes in the queue is triggered using a request signal, and an acknowledge signal is received as a confirmation of receiving the request. These inputs handle the functioning of the network from the input side and does not require any timer input for providing the input. Every spike is sequentially sent, which will be stored in the queue, and the request signal processes these spikes. Every spike that are sent between 2 request signals are basically treated as a set of input spikes under a 'chunk' of time. Because of this model of working of the network and that of the spike generation, a timer module is redundant while providing inputs. For a network with 28 input neurons with an input of 28*28 spikes, this method spike generation requires 784 clock cycles or 7.84 μs at 100 MHz clock frequency just for the providing the inputs. This does not take into consideration the latency of the system to process the spikes, which is the time taken between providing a request signal and receiving an acknowledge signal.

The main difference of the network input interface compared to that of Gyro is that the input spikes does not require temporal coding or mean rate coding. Temporal coding represented how spikes were represented between network layers in Gyro. Temporal coding involved implicit coding of time, since timestamp was also used to represent an input spike. Because of this, communication latency and jitter had to be limited. Latency had no effect on functional correctness but rather affected the feasibility of real-time requirements. An increase in communication latency meant an increase in response time of the whole system. Large values of jitter resulted in erroneous time related calculations, for instance, neurons might leak more than they should or it is incorrectly decided that the refractory period has expired. Avoidance of time-related calculations as a whole meant that the system no longer had these disadvantages of temporal coding.

Mean rate coding, on the other hand, had no dependence on communication latency or jitter. But it required additional logic to convert mean rate for each input neuron into a sequence of spikes and was dependent on timestamp for its generation of input spikes. This was done using a spike generator module with mean rate as input, which represents the probability that an input neuron has spiked, and sets registers for the input accordingly. The spike generator module was time-multiplexed, which meant that multiple instances were required depending on the number of input neurons. The mean rate encoding method is no longer required since input spike generation is not dependent on timestamp at all.

4.5.2 Network Output

Evaluation of the proper functioning of the network is done by monitoring the neurons of the last layer of the network. The output spikes can be observed in 3 ways mainly. The first option is a stream of sequential output spikes along with the source addresses of the spiking neurons. The second option is a parallel bus with the bit positions representing the address of the neuron that spiked. The third possibility is to monitor the individual membrane voltages of every neuron as and how it is processed.The sequential spikes is what facilitates the spike propagation between the layers and that is how the spikes are represented between the layers. The parallel bus of output spikes enable the user to monitor which neuron spiked at every request. Since the threshold evaluation of every neuron happens in parallel, the output spikes are sent out at the same instant and can be observed as a burst of output spikes. Monitoring the membrane voltage of every neuron enables the verification of proper functioning at the lowest layer of the network.

The previous design, Gyro, used 2 additional methods along with output spikes to monitor the network which are Inter-Spike Interval (ISI) and Low-pass filtered voltage. ISI was a reliable measure for high spike rates. It computes the time interval between 2 output spikes, as had a highest limit of the period of the clock signal used. For a time triggered approach, ISI values proved useful, but for a request triggered system, it does not apply. This is because the neurons are processed in parallel and all neurons will send output spikes when it gets the request signal. What results is a burst of output spikes, and when serialized, it will have an ISI value equal to that of the time period of the clock signal. This makes the ISI module redundant.

For spike rates below ISI value, membrane potential was monitored by passing it through a low-pass filter. This provides an average potential without high frequency variations. It used a first-order IIR filter, which sampled data based on a time trigger, and a sampling frequency derived from the time resolution from the timer module. Since it is a time-dependent calculation, it was not used and the design relies on monitoring the raw values of membrane voltages instead.

Chapter 5

VHDL Implementation

The VHDL implementation of the modified Gyro architecture was executed with a bottom-up approach and are synthesized for Xilinx FPGAs. Starting from the smallest element, which is the neuron, then the weight control module, output spike buffer, layer and ending with a network with multiple layers. The hierarchy of the SNN can be observed in figures 4.2 and 4.5. The changes brought about in the architecture is easier to ripple throughout the system with a bottom-up approach.In a bottom-up design methodology, the building blocks are first identified. Using these building blocks, bigger cells are built. These cells are then used for higher-level blocks until the top-level block in the design is reached. This design flow results in a library of all necessary building block elements required to complete the design. Since there are no restrictions on the top-level design, top-down methodology is not really required for the design of this system.

Using bottom-up approach, each module is written and verified separately with block-level simulations. This was followed by integration of these modules which were verified using system level simulations. The simulations and synthesis were done using Xilinx's Vivado. The parameters in table 5.1 along with the memory mapping parameters are specified in the VHDL implementation for realising the recurrent networks. The weight values are specified using ASCII text files containing binary data. Vivado parses these files during synthesis, which initializes the weight memory accordingly. These ASCII text files are generated using a python script which converts the weight matrices to the weight values in the same pattern intended for the BRAM, using the weight mapping parameters.

The VHDL simulation was performed for a single neuron, a network of 2 neurons and a network of 96 neurons for the simulation of MNIST dataset. Throughout the design flow, verification of weight fetching, output spike buffer and the recurrent spike functionality was observed and its functional correctness was verified manually against the weights generated using python script.

5.1 Leaky Integrate-and-Fire Neuron

The LIF neuron model is the core computing element of the SNN. It constitutes the LIF neuron cluster in figure 4.5. The functional correctness of the neuron model is crucial for the proper working of the network itself. The expected behaviour of the redesigned LIF neuron is that a series of

Parameter	Data Type	Default Value
Number of Layers	Integer	-
Recurrent Connections	Boolean	True
Input spike queue depth	Integer	64
Bit width of neuron address	Integer	10
Bit width of neuron membrane potential (V)	Integer	9
Bit width of weights	Integer	4
Threshold voltage	Integer	$2^V - 1$

Table 5.1: Network Parameters

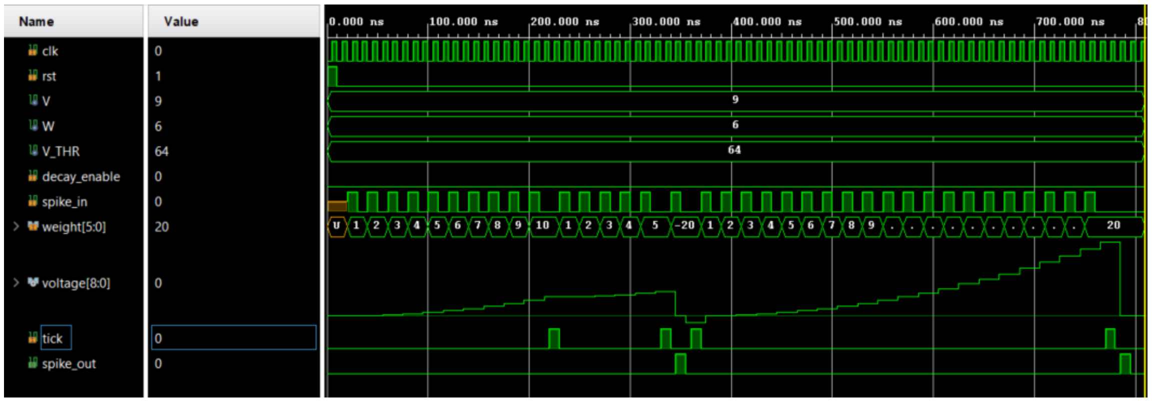


Figure 5.1: RTL simulation of event-driven LIF neuron model

input spikes are accumulated over time until a flag signal is received, upon which it is evaluated against the threshold voltage. The VHDL simulation can be observed in figure 5.1 where the neuron behaves exactly as intended. The input provided to the neuron model are spikes with different weight values, such that every scenario of the type of input can be observed. This includes normal input spikes, negative weight values, accumulation of membrane potential above threshold, accumulation of membrane potential below zero, threshold evaluation at positive and negative values of membranes potential etc.

The incoming spikes are processed immediately. Evaluation against the threshold is done at every 'tick' signal. If the membrane voltage is greater than the threshold voltage, an output spike is produced. If the membrane voltage is greater than zero and less than threshold value, then no output spike is produced and the membrane voltage value is held. For negative values of weight for input spikes, the voltage goes below zero and an evaluation against threshold value resets the membrane voltage to zero (as can be observed in figure 5.1), without producing any output spikes. Fixed membrane decay reduces the complexity of the design when membrane decay is enabled. It is implemented using a left bit-shift to apply a 50 percent decay of membrane voltage.

5.2 Simple Network

Through simulation of a simple 2 neuron network with a single input, the transfer function of the LIF neuron can be obtained. Furthermore, this also helps in verifying other sub-modules of a layer from figure 4.5. The design flow includes comparison of the VHDL implementation against an identical network implemented and simulated in python. This enables comparison of the network with expected behaviour.

The synaptic weights are kept identical for all forward connections, which is '1', and '2' for recurrent connections. This is to verify the proper working of the weight control module. The simulated input is that a single input spike and a request to process the spike is provided in a sequentially repeating manner. The expected output of the neuron membrane voltage is a constant step incremental climb up (step value of '1', which is the forward weight) until the threshold voltage. After threshold evaluation, the membrane voltages reset to zero, since an output spike is produced. This can be observed in figure 5.2, where the 2 signals with a slope are the membrane voltages of the 2 neurons.The output spike in-turn produces recurrent spikes within the layer which is accumulated afterwards. Proper propagation of the recurrent spikes with the corresponding weights being fetched can be observed in the same figure (5.2). The step increment in membrane voltage after threshold evaluation is steeper, since the recurrent weights have value '2', and is accumulated initially. This is identical to the expected behaviour in as can be observed in figure 5.3 where the membrane potential of the neuron goes through a step increment. The first increment after firing an output spike is higher because of the higher synaptic weight values for the recurrent spikes. The steep fall in membrane voltage at every time-step is due to the neuron firing an output spike and resetting the membrane potential to zero.

Processing of recurrent spikes is an added functionality in the second iteration of Gyro design. The output spikes from a layer is not only passed to the output (or next layer, depending on the network) but also to the input spike queue of the same layer. The recurrent spike queue enables this. The working of this functionality can be observed in figure 5.2, where output spikes are forwarded to the recurrent spike input queue of the layer. Enabling recurrency of a layer is what triggers the processing of recurrent spikes in the layer.

The output spikes are produced as a burst and are read in parallel. This is converted to serial spikes using a N to 1 parallel to serial converter, where N is the number of neurons in the layer and only one output spike is sent at a given moment. The source addresses are generated in the neuron wrapper module. Working of the neuron wrapper module can be seen in figure 5.4, where output spikes originating from the neuron cluster is converted to serial output spikes along with the generation of the address of the neuron from which the output spike was generated.

The transfer function of the neuron can therefore be inferred from this model. It can be observed in figure 5.5 for different ratios of weight bit-resolution against the threshold voltage.

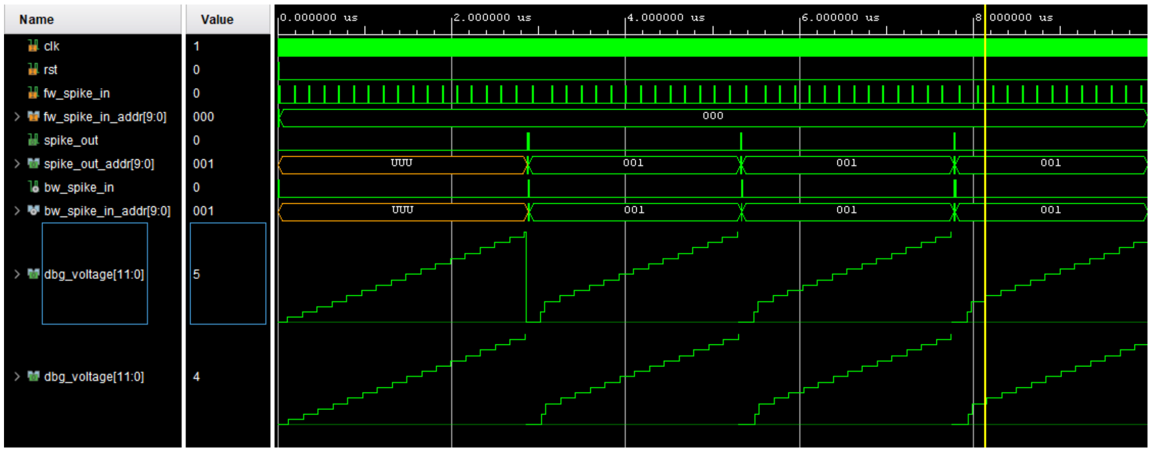


Figure 5.2: RTL simulation of layer with 2 neurons

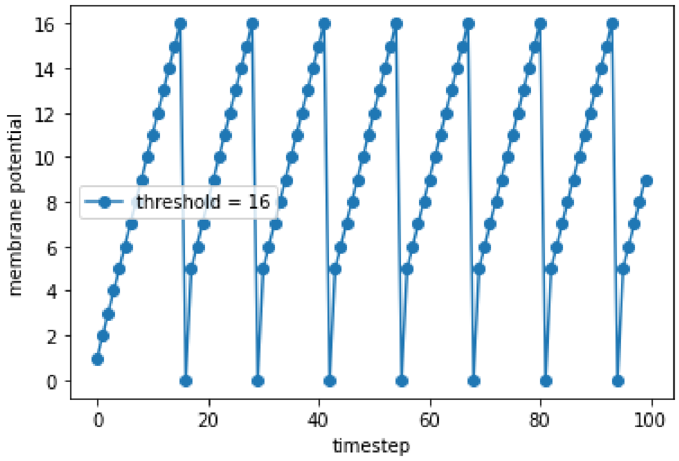


Figure 5.3: Membrane voltage temporal evolution of the neuron model

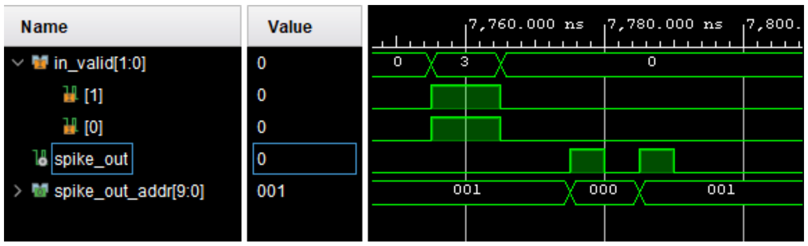


Figure 5.4: Conversion of parallel spikes to serial spikes

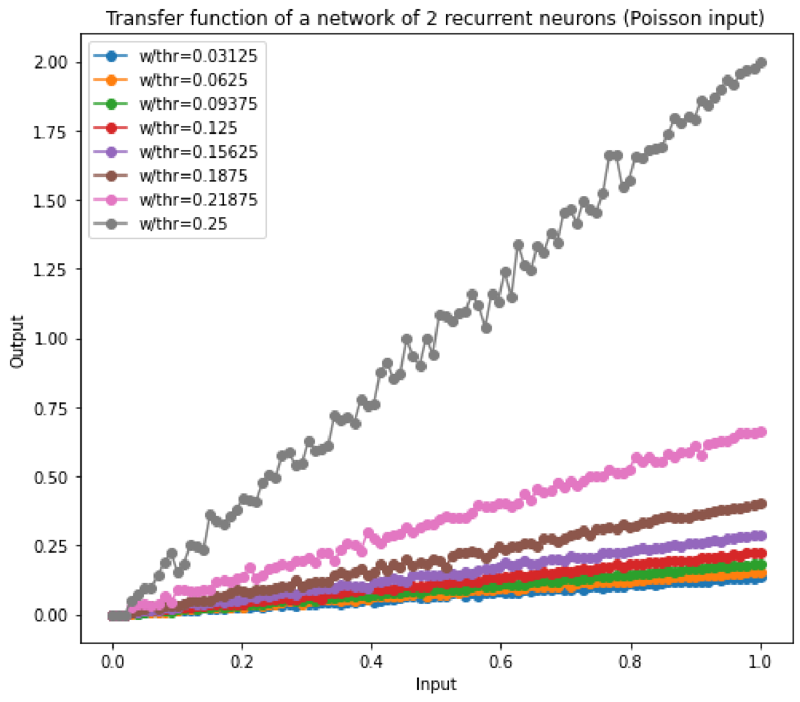


Figure 5.5: Transfer function of the neuron model

5.3 MNIST Dataset Simulation

After the functionality of each of the sub-modules are verified through the implementation of a simple network, a bigger network of the configuration 28-64-32 (28 inputs, 64 neurons in layer 1, 32 neurons in layer 2) with synaptic weights from a trained network is simulated. The weights are generated from a python script, which are binary files with synaptic weights in them. Weight memory module simulates BRAM modules and reads the weights from the binary files. The weight control module reads the weights from this "BRAM" and performs the accumulation. The input spikes are a series of 28 inputs. These simulate the handwritten digits dataset provided by MNIST. It can be used as a base for comparison against other related implementations. It contains 60,000 training images and 10,000 test images. The images are black and white and is represented in 28x28 pixels, which in binary, is a matrix of 28 rows and 28 columns (figure 5.6). The testbench reads the test images in a binary format from the binary file and simulates the input for the network. Each row of pixels are read and sent to the first hidden layer of 64 neurons with the source address (from 1 to 28). After each row, the request signal is provided to process the first row of inputs. This is repeated 28 times to process one image. The results are observed at the output from the first 10 neurons of the 32 neurons in the second hidden layer or the output layer.

The simulation was first done in python for a trained network. The synaptic weights provided to both the python simulation and RTL simulation were identical, which was for a network that included recurrent connections. The expected output is a one-to-one matching of output spikes and membrane voltages between both the simulations, with the python simulation acting as a reference. In the RTL simulation in figure 5.7, it can be observed that the output spikes from second layer are only produced by the first 10 neurons. This is identical to the result from the python simulation in figure 5.8, where output spikes from both layers can be observed at every time-step. The first block represents the first layer and the second block represents the second layer, of output spikes. Only first 10 neurons are spiking in the second layer which is the result of the classification. Furthermore, it was verified that the output spikes were identical for the first layer and that the output spike addresses were correct for both the layers. In figure 5.9 proper signal propagation from input, first hidden layer, second hidden layer and output can be observed. The signals in order are simulated input spikes and addresses, forward and recurrent spikes with their addresses, tick signal, output spikes and their addresses. These signals from the first layer are followed by the same signals from the second layer.

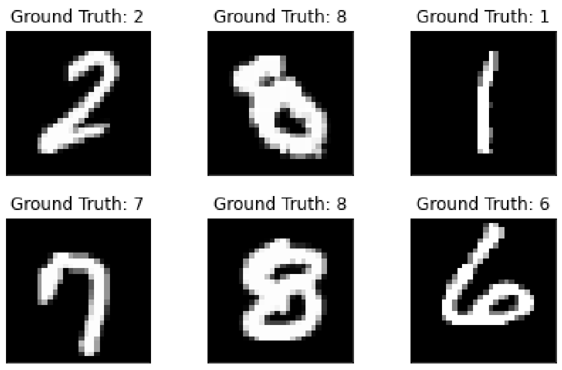


Figure 5.6: MNIST input digits

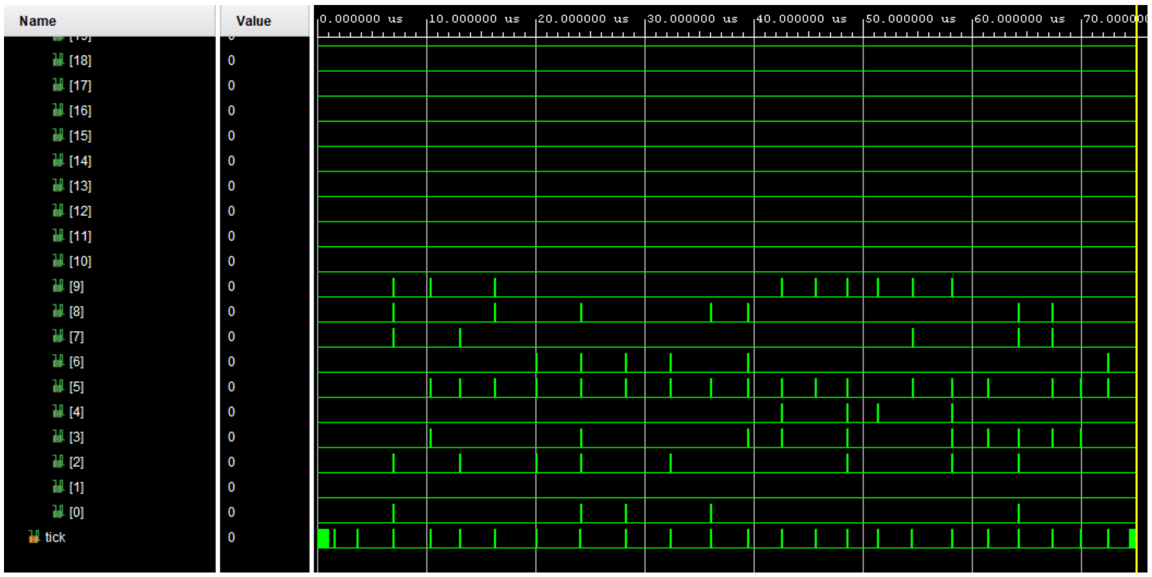


Figure 5.7: Output spikes from second layer

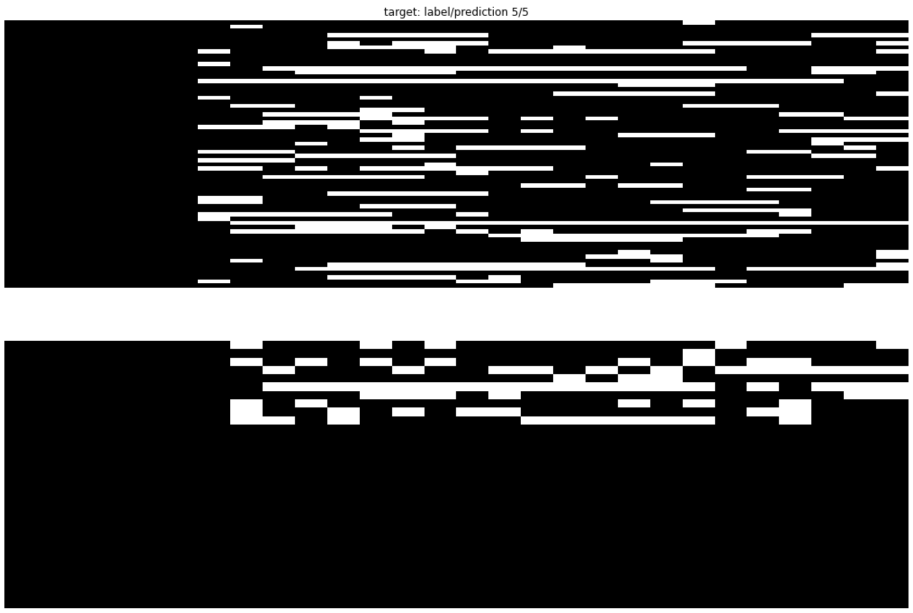
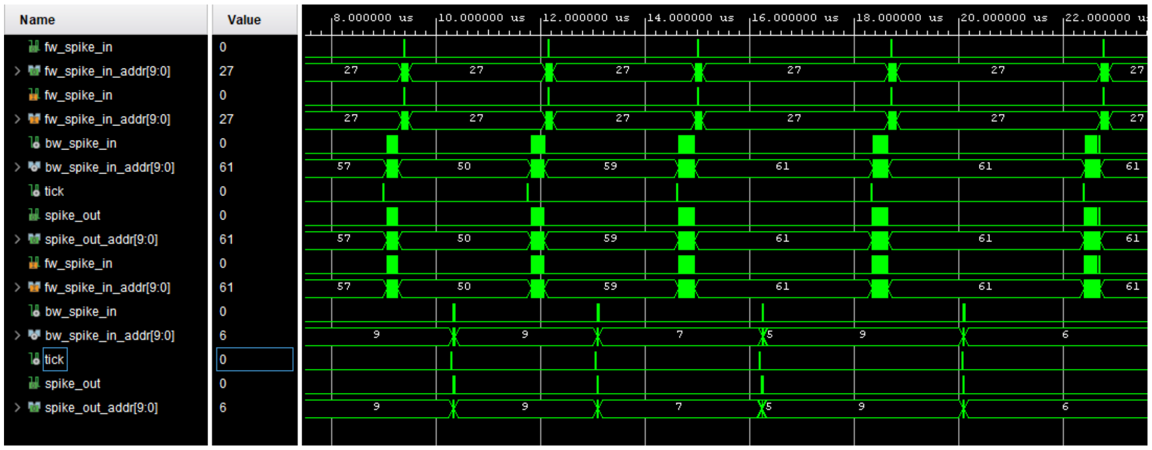


Figure 5.8: MNIST output spikes from first and second layers respectively



Name	Value
fw_spike_in	0
fw_spike_in_addr[9:0]	27
fw_spike_in	0
fw_spike_in_addr[9:0]	27
bw_spike_in	0
bw_spike_in_addr[9:0]	61
tick	0
spike_out	0
spike_out_addr[9:0]	61
fw_spike_in	0
fw_spike_in_addr[9:0]	61
bw_spike_in	0
bw_spike_in_addr[9:0]	6
tick	0
spike_out	0
spike_out_addr[9:0]	6

Figure 5.9: RTL simulation of a network running MNIST test case

Chapter 6

Results

The assessment of results obtained from the design modifications of Gyro are divided into subsections as follows: Characterization, Case studies and Comparisons. Section 6.1 presents quantitative results in terms of throughput and resource utilization. Section 6.2 evaluates the accuracy and functional correctness of the implementation for the test cases. Comparison against other related works will be discussed in section 6.3.

6.1 Characterization

This section presents a theoretical analysis of the design on throughput and latency. The results of the implementation is also explored in regards to the accuracy and resource utilization. Networks are represented in the format I-L_1-L_2-...-L_n where for instance, 28-64-32 represents a network with 28 inputs, 64 neurons in the first hidden layer and 32 neurons in the output layer.

6.1.1 Throughput

For artificial neural networks, throughput is defined as number of synaptic operations per second (SOPS). When a neuron generates an output spike to N neurons, it causes N synaptic operations. The peak throughput is decided by the weight mapping parameters and the clock frequency. The weight mapping parameters define the memory size and layer size, as can be seen in section 4.3. Upon receiving a forward input spike, the layer does $x_1 * z_1$ synaptic operations per clock cycle. The layer requires y_1 clock cycles plus one clock cycle overhead to update all neurons in the layer. The maximum amount of forward input spikes the layer processes per time unit is therefore:

$$\frac{1}{t_{clk}(y_1 + 1)} \tag{6.1}$$

Thus, the peak throughput in synaptic operations per time unit for a layer is:

$$\frac{x_1 \cdot y_1 \cdot z_1}{t_{clk}(y_1 + 1)} \tag{6.2}$$

Summing this over all the individual layers gives the peak throughput of the whole SNN. Only the hidden layers are taken into accounting when calculating the throughput, since the input layer does zero calculations. The peak throughput of the SNN therefore is:

$$\sum_{l=2}^{L} \frac{x_1^l \cdot y_1^l \cdot z_1^l}{t_{clk}(y_1^l + 1)} \tag{6.3}$$

where l indicates the hidden layers, L is the total number of hidden layers, and x_1^l, y_1^l, z_1^l are the parameters of the respective layer.

The calculation of the peak throughput for recurrent spikes follows the same method, only difference being the parameters will be having suffix r. One key factor here is that both the forward and recurrent synaptic weights are stored in the same memory bank, which is in contrast with Gyro, where synaptic weights for forward connections and backward connections (not recurrent connections) were stored in different memory blocks. So, for peak throughput of the SNN for recurrent input spikes are:

$$\sum_{l=2}^{L} \frac{x_1\_r^l \cdot y_1\_r^l \cdot z_1\_r^l}{t_{clk}(y_1\_r^l + 1)} \tag{6.4}$$

where l indicates the hidden layers, L is the total number of hidden layers, and $x_1\_r^l, y_1\_r^l, z_1\_r^l$ are the recurrent parameters of the respective layer. The peak throughput for forward spike and recurrent spike are virtually the same since the parameters are identical (section 4.3). Essentially, both types of input spikes update the same number of neurons with the same throughput.

From these equations,it is evident that the throughput is maximized by minimizing y_1 and y_1_r. However, to maximize the utilization of the memory in terms of memory depth, it is favourable to maximize $y_1 * y_2$ and $y_1\_r * y_2\_r$ to the maximum limit of M_y, which is the maximum available memory depth. So a trade-off can be made by minimizing y_1 and y_1_r to increase the throughput, and increasing y_2 and y_2_r proportionately to maximize the utilization of memory in terms of depth.

Both the test cases used to evaluate Gyro II had recurrent connections by default. The parameters for the network that implemented MNIST test case are as in table 6.1.

Parameter	**Layer 1**	**Layer 2**
x_1 / x_1_r	1	1
y_1 / y_1_r	16	8
z_1 / z_1_r	4	4
x_2 / x_2_r	1	1
y_2	7	16
z_2	4	4
y_2_r	4	4
z_2_r	16	8

Table 6.1: Weight mapping parameters for 2 layers of MNIST test case network

The network configuration is 28-64-32 with a clock frequency of 250 MHz. Throughput for each layers are calculated using equation 6.3 or 6.8 as follows:

$$Layer1 : \frac{x_1^l \cdot y_1^l \cdot z_1^l}{t_{clk}(y_1^l + 1)} = \frac{64}{4 \cdot 10^{-9}(16+1)} = 941.17MSOPS \tag{6.5}$$

$$Layer2 : \frac{x_1^l \cdot y_1^l \cdot z_1^l}{t_{clk}(y_1^l + 1)} = \frac{32}{4 \cdot 10^{-9}(8+1)} = 888.88MSOPS \tag{6.6}$$

This gives a total of 1.83 GSOPS for the MNIST test case network. The throughput can be further increased by increasing the x_1 parameter, but was kept to 1 for this test case for ease of functional verification.

Now consider the cropland classification test case. Table 6.2 contains the parameters for the network for cropland classification test case.

Parameter	**Layer 1**	**Layer 2**
x_1 / x_1_r	1	1
y_1 / y_1_r	20	1
z_1 / z_1_r	30	8
x_2 / x_2_r	1	1
y_2	1	75
z_2	17	8
y_2_r	20	1
z_2_r	30	8

Table 6.2: Weight mapping parameters for 2 layers of cropland classification test case network

The network configuration is 17-600-8 with a clock frequency of 250 MHz. Throughput for each layers are calculated in the same way as follows:

$$Layer1 : \frac{x_1^l \cdot y_1^l \cdot z_1^l}{t_{clk}(y_1^l + 1)} = \frac{600}{4 \cdot 10^{-9}(20+1)} = 7.14GSOPS \tag{6.7}$$

$$Layer2 : \frac{x_1^l \cdot y_1^l \cdot z_1^l}{t_{clk}(y_1^l + 1)} = \frac{8}{4 \cdot 10^{-9}(1+1)} = 1GSOPS \tag{6.8}$$

This gives a total of 8.14 GSOPS for the cropland classification test case network. From the above throughput numbers, it can be concluded that the bigger the value of the cluster size, i.e., $x_1 \cdot y_1$, higher will be the throughput of the network. Gyro attained a throughput of 83.3 MSPS for 240 neurons, which is relatively low, considering how it was only a feed-forward network. For a network of 240 neurons with backward spike propagation, Gyro attained a throughput of 20.8 MSPS. Comparing the throughput with that of Gyro, the stark difference is that in Gyro, throughput is significantly lower for backward synaptic connections when compared to forward connections. This was overcome in Gyro II by opting for a synaptic weight storage pattern that is the same for both the forward and recurrent connections.

6.1.2 Resource Utilization

Resource utilization of the design is measured in terms of the amount of FPGA resources that are required, mainly the different types of memories associated with it. They are Lookup Tables (LUTs), Flip-Flops (FF) and BRAMs. Table 6.3 shows the numbers associated to the resource utilization for the implementation of the networks for the two test cases, MNIST dataset classification and cropland classification. The resource utilization of the networks can be observed in table 6.3. This does not include the external registers, resets or the AXI interface logics.

Network	FPGA	LUTs	FFs	BRAMs
28-64-32 (Recurrent)	ZU7EV	7,559	3,604	16
17-600-8 (Recurrent)	ZU7EV	54,516	21,756	312

Table 6.3: FPGA resource utilization for test cases

For both the MNIST network and cropland network, the weight memory was initialised in such a way that they will be implemented as LUTs. That is, in the RTL design and the synthesis of the design, the synaptic weights were specified to be used as LUTs and not as BRAMs, since LUTs are suited for fast access of the values. The MNIST network uses 7,559 LUTs and 3,604 FFs. The network logic is what mainly contributes towards these numbers. It also requires 16 BRAMs for the implementation. The Cropland network has a significantly higher number of neurons and therefore increased number of synaptic weights. 54,516 LUTs and 21,756 FFs were required to implement the network, along with 312 BRAMs. The tool also optimizes the resource utilization depending on the sparsity of the network, i.e., for synaptic weight values of zero, which occurs frequently, the realization would not require any dedicated logic on the FPGA. Plots of how the resources are allocated within the network can be seen in figure 6.1.

The resource utilization is predictably directly proportional to the number of neurons in a layer. More neurons require more configurable logics for implementation. This can be seen in figure 6.2, where the neuron cluster takes up the majority of resources in a layer of the network. The relatively large number of FFs is due to the storage of membrane potential. After dividing the total resources used by a neuron cluster by the number of neurons, the amount of logic required to implement a single neuron can be obtained. A single neuron uses 62 LUTs, 30 FFs, no BRAMs, no DSPs. Furthermore, the weight controller requires its own combinatorial logic for the proper fetching of neuron weights and for memory address decoding. Compared to the weight controller and neuron cluster modules, the input and output spike queue uses very small quantities of the resources. This is due to a simplistic approach for emptying the queue using the state machine rather than using a round-robin arbiter. A layer of neurons do not require any DSPs due to removal of redundant calculations and simplification of others. Additionally, removal of the modules timer, ISI monitor, low-pass filter, and spike generators also greatly reduced the complexity and resource utilization of the design.

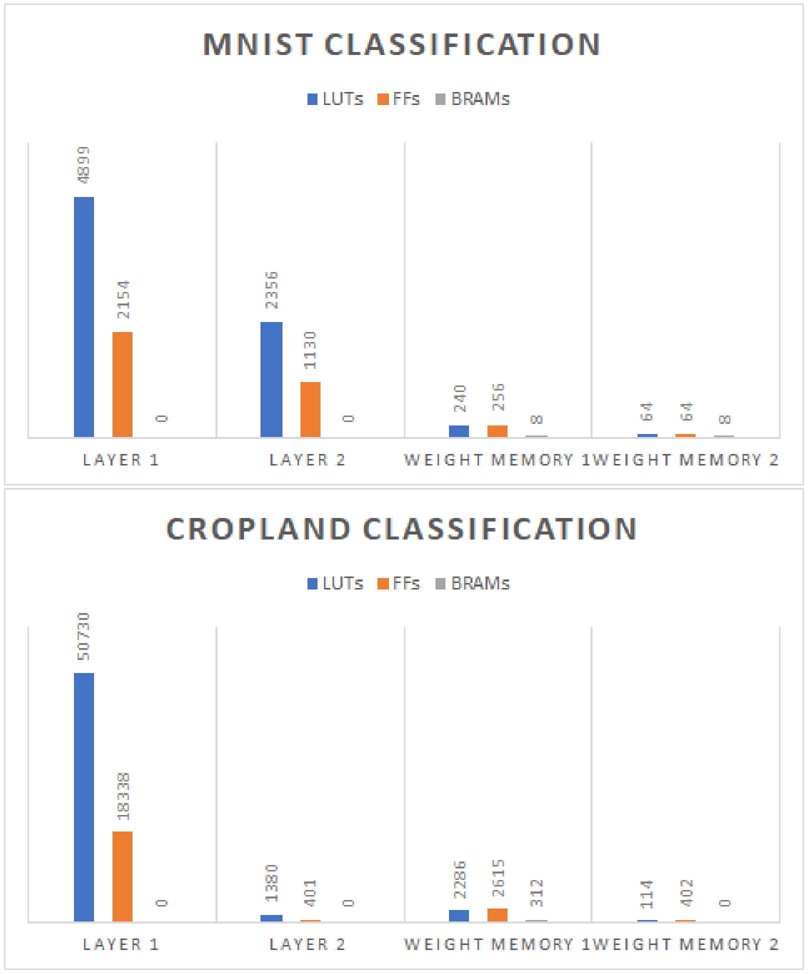


Figure 6.1: Plots of resource distribution within the network

6.2 Case Studies

Using the case studies for reference, Gyro is compared against Gyro II, mainly in terms of resource consumption. Two case studies were performed to evaluate the design; MNIST dataset classification and cropland classification. The functional correctness of the implemented networks for these test cases were discussed in chapter 5. They also provide quantifiable measures that can be used to compare the design to other works. The networks were trained using python and the synaptic weights were converted to 4-bit signed values for RTL implementation. Table 6.4 shows the results obtained after running the test cases. It can also be observed that the networks implemented using Gyro II architecture contains significantly lesser number of neurons. This is because the added functionality of recurrently spiking neurons enables networks of smaller sizes have comparable performance figures for the same application or case, while significantly reducing the resource consumption of

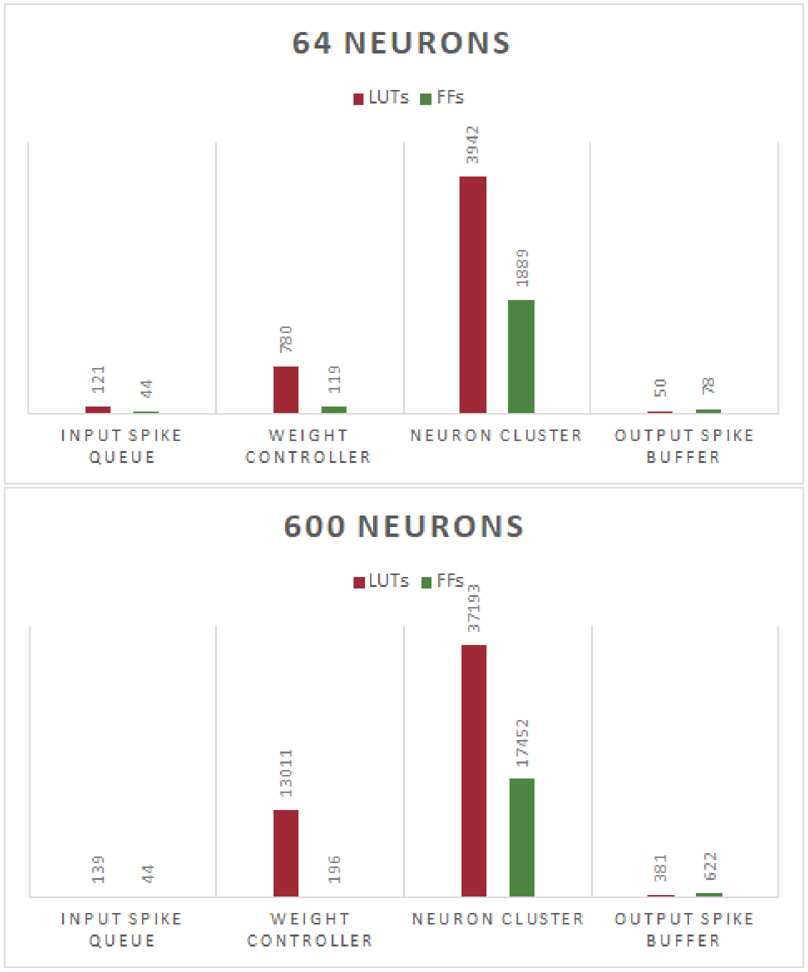


Figure 6.2: Plots of resource distribution within a layer of the neurons

the implementation.

Test case	**Network**	**Neurons**	**Accuracy**
MNIST	28-64-32 (Recurrent)	96	93.5%
Cropland Classification	17-600-8 (Recurrent)	608	95%

Table 6.4: Accuracy of test case networks

6.2.1 MNIST Classification

MNIST dataset contains 60,000 training images and 10,000 test images. The images are black and white and are represented in 28x28 pixels, which in binary, is a matrix of 28 rows and 28 columns. The network of 64-32 neurons uses 28 inputs in 28 time-steps to classify the image. From the results obtained during simulation of MNIST dataset test case, it was observed that the RTL implementation has identical results as that of the python simulation. This means that the accuracy for classification of the RTL implementation is same as that of the python version and hence, performance metrics can be measured using the python version of the network. The network was trained for a weight bit-resolution of 4-bits. The weight matrices occupies the BRAM or the weight memory. For a weight bit-resolution of 4-bits, the implemented network for MNIST test case achieved an accuracy of 93.5%. Figure 6.3 shows a plot of accuracy against different weight bit resolutions.

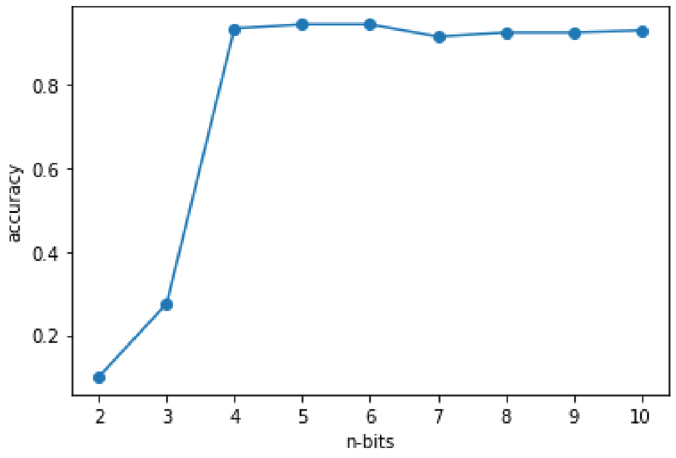


Figure 6.3: Plot of accuracy against different weight bit resolutions for MNIST test case (network 28-64-32).

Network	**Neurons**	**LUTs**	**FFs**	**BRAMs**	**Accuracy**
784-720-720-720-10 (Gyro, feed-forward)	2954	140,206	131,977	306	99.3%
28-64-32 (Gyro II, recurrent)	96	7,559	3,604	16	93.5%

Table 6.5: Comparison between Gyro and Gyro II for MNIST classification cases

For the MNIST classification application, Gyro required a network of 2954 neurons with 6-bits weight precision to attain accuracy of 99.3%, whereas the second iteration achieved an accuracy of 93.5% with just 96 neurons and a weight bit resolution of 4-bits. Gyro required all the inputs to be provided parallelly, i.e., for an input image of 28x28 pixels, it required 784 inputs, while for Gyro II, the time-step approach meant only 28 inputs were required, to which 28 set of inputs were provided in a sequential manner. When it comes to resource utilization, Gyro required 140,206 LUTs, 131,977 FFs and 306 BRAMs. Gyro II required 7,559 LUTs, 3,604 FFs and 16 BRAMs for its implementation. A comparison of the networks implemented in Gyro and Gyro II can be seen in table 6.5.

6.2.2 Cropland Classification

Cropland classification uses optical radar data from a large set of numerical features collected by an Uninhabited Aerial Vehicle Synthetic Aperture Radar (UAVSAR) over agricultural regions. The dataset contains labels of 8 crop types. A network of 600-8 neurons was implemented for this classification. It uses 17 inputs in 6 time steps to provide a full image for classification. The implemented network achieved an accuracy of 95%, for a weight bit-resolution of 4-bits. Figure 6.4 shows a plot of accuracy against different weight bit resolutions.

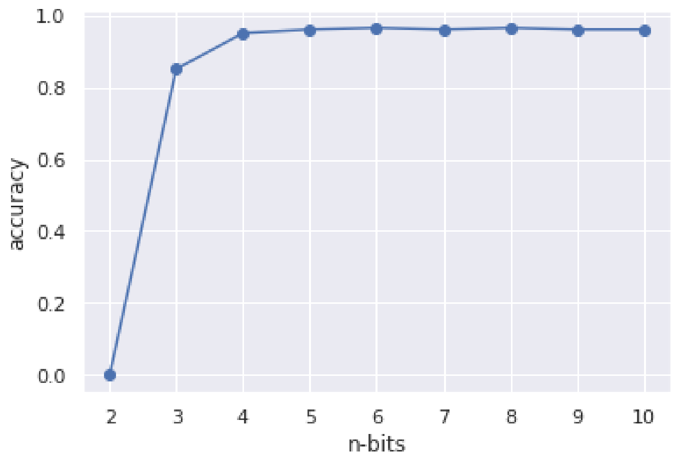


Figure 6.4: Plot of accuracy against different weight bit resolutions for Cropland classification

Network	**Neurons**	**LUTs**	**FFs**	**BRAMs**	**Accuracy**
102-600-600-600-7 (Gyro, feed-forward)	1,807	101,583	104,738	170	99%
17-600-8 (Gyro II, recurrent)	608	54,516	21,756	312	95%

Table 6.6: Comparison between Gyro and Gyro II for cropland classification cases

For the cropland classification test case, Gyro required a network of 1,807 neurons with 6-bit weight precision bits to achieve an accuracy of 99%. It required 101,583 LUTs, 104,738 FFs and 170 BRAMs. For the same test case, Gyro II was able to achieve an accuracy of 95% using a network of 608 neurons with 4-bits weight precision bits. This network required 54,516 LUTs, 21,756 FFs and 312 BRAMs. A comparison of the networks implemented in Gyro and Gyro II can be seen in table 6.6. The number of BRAMS used in Gyro II is higher because the xilinx tool automatically optimizes the memory elements and therefore allocates BRAMs for the entirety of weight memory 1, as can be seen in the resource distribution in figure 6.1, and not for any other module.

6.3 Inference

When compared against Gyro, Gyro II has the added functionality of a recurrently spiking neurons, whereas the peak accuracy achieved by Gyro was a feed-forward network. The reduced resource utilization observed in tables 6.5 and 6.6 is a combined result of removal of redundant computations, simplification of neuron model and converting it to an event-driven model, removal of redundant modules like round-robin arbiters, ISI calculators, low-pass filters, etc. Furthermore, the recurrency feature enables realization of much smaller networks with comparable performance while significantly reducing the memory overhead of the implementation. And from this comparison of Gyro II against its first iteration Gyro, the classification accuracy is relatively low. Even though this is not a significant dip in performance, the reason for this is the low weight bit-resolution of 4-bits and reduced network size. However, the trade-off is that these smaller networks handled the application testcases sufficiently without taking up too much memory, when compared to the networks implemented on Gyro. The accuracy of the network can be increased by a mere 1% to 96% if the weight bit-resolution is increased to 10 bits. However, this will dramatically increase the resource utilization, which is not favorable.

An increase in the weight bit-resolution beyond 4-bits does not significantly contribute to the accuracy, as can be seen in figures 6.3 and 6.4. The network configurations along with the accuracies can be seen in table 6.4. It can therefore be concluded that a relatively low bit precision of 4 bits is not a limiting factor in achieving high accuracy. However, sufficient number of neurons in the hidden layers are significant for reducing the error rate for low weight bit precision. Gyro II also handles inputs in a parallelized manner, thereby elimination the number of inputs received by the first layer of the network. Time-stepped approach of providing inputs was possible because the event-driven nature of the implemented neuron model. This allowed the design to incorporate inputs provided in a sequential time-multiplexed manner. This was done without removing the parameterized network implementation of Gyro, meaning that any network topology with a large number of neurons can be configured depending on the requirement and resource availability of the FPGA being used.

The peak throughput of Gyro II is heavily depending on network size. For a network of 28-64-32 neurons, a peak throughput of 1.83 GSOPS is achieved, which includes both forward and recurrent connections. For a network of 17-600-8 neurons, a peak throughput of 8.14 GSOPS is achieved. The relation between the cluster size or the parameters and the throughput can be taken into account when configuring the weight mapping, such that a higher throughput can be obtained. The numbers attained using Gyro II is not comparable to other related works since the networks usually implemented has significantly larger number of neurons and layers. However, compared to Gyro, there is a substantial increase in the throughput considering how the network is considerably smaller and has the added functionality of recurrent spiking neurons.

The presented results are also compared against similar works (table 6.7 and 6.8) to get a bigger picture of how well the second iteration of Gyro performs, despite its implementation of smaller networks. All the other networks in table 6.7 contains large number of neurons, upwards 2000. Gyro II achieved this accuracy for significantly reduced number of neurons, which is 608. Except Gyro, all the other implementations mentioned in table 6.7 used time multiplexed neurons. This means that the number of physically implemented neurons are lower than the actual number of neurons in the layer, which reduces memory overhead and enables realization of really large networks. The second iteration of Gyro does not have time multiplexed neurons, which contributes to the LUT usage. However, this does not hinder Gyro II from achieving a high accuracy compared to other works. A comparison of Gyro II against some other related works from chapter 3 can be seen in table 6.8. The implementation of networks using Gyro II architecture is also much more convenient, given how the inputs provided to the network has changed from Gyro. This is mainly because of the absence of temporal information in the input and output, and how output interpretation does not require analog filters or ISI counters.

Model	**Year**	**Architecture**	**Accuracy**
Neil [13]	2016	DBN	94.1%
Stromatias [33]	2015	DBN	94.9%
Stromatias [32]	2015	DBN	95%
Han [10]	2020	SNN	97.1%
Gyro [5]	2020	DBN	99.3%
Gyro II (this work)	2021	Recurrent SNN	96%
Esser [8] [7]	2015	Deep SNN	99.4%
Rueckauer [27]	2017	Spiking CNN	99.4%

Table 6.7: Spiking neural networks MNIST accuracy [34]

A benefit of the current implementation of the architecture is that it supports a range of layers and network sizes, Depending on the size and availability of resources on the FPGA board, many layers can be stacked on top of each other. The parameterized weight mapping patterns also makes it possible to make a trade-off between resources and throughput of the layer. Optimisation of these parameters depend on the network size, application and the FPGA board used. The comparison of test cases also gives an insight into how efficient the network is in terms of resource utilization compared to Gyro. For achieving a high accuracy, the networks implemented were recurrent networks with significantly lesser number of neurons.

Name	Bluehive	FDF	n-Minitaur	Pani	NCS	Tsinghua	Gyro	Gyro II
Reference	[18]	[36]	[12]	[22]	[37]	[10]	[5]	N/A
Year	2012	2014	2016	2017	2018	2020	2020	2021
FPGA	Stratix IV	Virtex 6	Spartan 6	Virtex 6	Stratix V	Zynq 7000	Zynq Ultrascale+	Zynq Ultrascale+
Clock (MHz)	200	266	105	100	200	200	250	250
Neuron model	Izhikevich	Conductance	LIF	Izhikevich	LIF	LIF	LIF	LIF
Network	Unknown	Unknown	Feed-forward	Recurrent	Unknown	Feed-forward	Recurrent	Recurrent
Driven	Time-driven	Time-driven	Event-driven	Time-driven	Time-driven	Hybrid	Event-driven	Event-driven
Weight storage	Off-chip	On-chip	Off-chip	On-chip	Off-chip	Off-chip	On-chip	On-chip
Weight bit-width	12 bits	12 bits	16 bits	7 bits	4 bits	16 bits	6 bits	4 bits
Cores	16	23	32	8	200 k	Unknown	Same as neurons	Same as neurons
Number of neurons	256 k	1,5 M	1794	1440	100 M	2842	2954	608
Peak throughput (GSOPS)	0,256	1200	0,0535	0,0144	20	0,67	22,85	8.14
Power dissipation (W)	Unknown	Unknown	1,5	8,5	32,4	0,5	2,6	Unknown
Energy efficiency (nJ/SO)	Unknown	Unknown	28	590	1,62	712	0,109	Unknown
MNIST accuracy	Unknown	Unknown	94,1%	Unknown	Unknown	97,1%	99,3%	93.5%
Application	Communication topologies with high bandwidth and low latency requirements	N/A	Combination of DVS and DAS sensory inputs	Ideal for closed-loop systems	Simulation of very large and structurally connected SNNs	Scenarios requiring strict power constraints	Speech recognition, Mine detection using sonar	Cropland classification

Table 6.8: FPGA-based spiking neural network implementations in chronological order

Chapter 7

Conclusion

This work presents Gyro II, an architecture to deploy Recurrent Spiking Neural Networks on FPGAs for Edge-AI applications. It is the second iteration of Gyro, which itself was an architecture for Spiking Deep Belief Networks. FPGAs were chosen due to the versatile nature it offers. It facilitates parameterized designs which can be altered or adapted according to the network size required for the application. Furthermore, FPGAs provide multiple options when it comes to the usage of on-chip memories required for storing the synaptic weights of the network. In Gyro, synaptic weights were significantly lesser compared to Gyro II due to only having feed-forward or feedback networks. Gyro II has the new included functionality of having networks with recurrent connections, which requires the storage of an increased number of synaptic weights. A method is presented to map these synaptic weights to the on-chip memories.

The mapping of the synaptic weights to the on-chip memories is realized in a parameterized manner, which enables the user to configure how the weights are stored in the memory. This plays a crucial role when it comes to the bandwidth of the weight memory, which affects the throughput of the system as a whole. The configurable nature also allows the user to prioritize the hardware utilization over throughput, which results in having a trade-off between hardware utilization and latency of the system. When more memories can be used, a higher memory bandwidth and ultimately higher throughput can be attained.

A hardware efficient, truly event-triggered, LIF neuron model is used in Gyro II. The conversion of the conventional LIF neuron model to an event-triggered model required elimination of time-triggered calculations. This also avoids the necessity of neuron time-multiplexing. Removal of time-dependent calculations from the neuron model also proved useful when it came to resource efficiency, by reducing the memory requirement significantly.

Looking back at the research question in section 1.1, the resulting architecture has successfully exploited the sparsity in time to make the system more event-driven and reduced the memory requirement significantly. In FPGAs, reduced memory consumption inherently implies reduced power and energy requirements. The implementations also showed that for the same use case scenario, a smaller network works just as well, due to the recurrent nature of the network means the design takes up lesser space for implementation in the FPGA. Furthermore, a glimpse into the sub-questions from section 1.1 concludes the following:

- The system was converted into a truly event-driven model by removing time-based calculations. Calculations are simplified and memory requirements were reduced upon making the design event-triggered. This was possible only due to the event-driven nature of SNNs, which made it possible to make the architecture more memory efficient.
- The architecture now supports recurrent connections within a layer. This made it easier for a smaller network to attain similar performance for handling the same task, whereas the previous architecture required a significantly bigger network, and therefore higher memory requirement.
- Memory requirements were significantly reduced by removing redundant sub-modules and calculations in the design. Reduced memory requirement implies reduced energy requirement in FPGAs. Eventhough no direct results were attained regarding power consumption, theoretical analysis shows how the system demands lesser power. This can prove to be helpful considering how Gyro II is geared towards Edge-AI applications.
- The parameterized design makes it possible to alter the network size and also enables configuring the synaptic weight storage mapping in the memory. This saves time when it comes to implementing multiple networks of different sizes. The architecture is scalable due to this aspect of the design.
- The parallel nature of the architecture is taken into account while making the system request based. The network now supports input in a time multiplexed manner.

The architecture is implemented using VHDL and the proper functioning was verified on different hierarchical levels, through both theoretical and empirical analyses. Theoretical analysis on throughput shows that a recurrently connected network of 96 neurons achieves a peak throughput of 1.83 GSOPS and a network of 608 neurons achieves a peak throughput of 8.14 GSOPS. This is a merit of using on-chip memory and configuring the parameters for weight memory in such a way that it maximizes throughput. These throughput numbers were also attainable due to the recurrent nature of the network, which was not possible in the first iteration, i.e., Gyro.

Two case-studies were performed, handwritten digit recognition and cropland classification. These prove the applicability and functional correctness of the architecture. Gyro II achieved a classification accuracy of 93.5% on the MNIST dataset using a recurrent network of 96 neurons which is impressive considering the network contains relatively lesser number of neurons compared to similar works of SNN implementations. Gyro paved the way for deploying spiking DBNs on power and energy constrained systems, and Gyro II improved upon it in terms of memory requirement and improved its functionality. Research and development in the area of deep spiking neural networks are nowhere near the end, and Gyro II is a link in one of the many possible improvements and iterations in the domain.

7.1 Future Work

This proposed architecture is the second working iteration which can be improved in many ways. Before and during the execution of the project, a list of desired features were prioritized and only a select few were implemented. This section discusses some possible improvements that can be brought to Gyro II.

- **Reconfigurable Weights** : The weights of the network are set by initialising the FF memory or BRAM with the synaptic weights during synthesis of the design. The values are not adapted afterwards. FF memory cannot be changed dynamically, but BRAMs and UltraRAMs can be changed while the SNN is running on the FPGA, due to their true dual-port nature. This has the potential to avoid long synthesis and implementation times between updates of weight values where the network topology remains same but the synaptic weights change. UltraRAM memory blocks also have larger width and depth compared to BRAM, enabling storage of more weights. A key difference is that UltraRAM memory contents are initialized to zeros at power up, and therefore weight reconfiguration functionality is required to use them.
- **Pipelining** : Currently, the finite state machines of the layer and the weight controller works in sequential manner. The state machine of the layer accumulates spikes first, then processes them, evaluates the threshold voltage and so on. The state machine of the weight controller finishes fetching all the weights before it starts decoding the next spike source. This sequential flow of data can be optimized by handling multiple steps at the same time, but multiplexed in time. For example, the layer state machine can trigger the weight fetching once spikes are present in the queue, and upon receiving the request signal, the weights are ready to be sent to the LIF neuron cluster. The weight controller can be pipelined to start address decoder for the next spike before it finishes fetching weights. This could eliminate overhead clock cycles for each spike source address that is processed and increase the throughput.
- **Weight Memory Mapping** : The current weight memory mapping is a paremeterized, straightforward pattern where the weights are considered as blocks and clustered together in the memory. Optimization techniques can be explored where the weight storage is handled in a pattern such that the clusters are stored in a pattern that maximizes throughput even more. Current pattern stores recurrent weights at an offset in ’y’ dimension. Offsetting in ’z’ dimension is a viable option but it will require larger area of implementation on the FPGA, eventhough it increases the throughput. This can also facilitate parallel fetching of forward and recurrent weight, and thereby provide a means to empty both the forward and recurrent spike queues simultaneously. Currently, the emptying of the queues and the weight fetching is executed in a sequential manner.
- **Neuron Time Multiplexing** : From figure 6.2 it is evident that the neuron cluster is the major contributor to LUT usage. Time multiplexing of neurons can therefore help in reducing the amount of physical neurons. Since memory bandwidth is the common bottleneck for performance, having a subset of neurons will be sufficient to achieve maximum performance. When it comes to time-multiplexing the neurons, the vast majority of the LUTs will be used to store the multiplexed voltages only. For the previous design, it would have required multiplexing of voltages and refractory end times, but since time-based calculations have been removed, time multiplexing of neurons could be beneficial in reducing memory requirements even further.
- **Parallel Weight Fetching** : Current design uses both ports of the true dual-port memory during weight fetching. However, the weights are still being read in a sequential manner, since forward synaptic weights and recurrent synaptic weights have to be fetched separately.

Fetching these weights in parallel could increase the throughput significantly, and thereby reduce the latency. This can be achieved by assigning multiple clock domains to the forward and recurrent weights, which can be incorporated with pipelined design where weights are fetched beforehand for processing.

Bibliography

[1] M. Ambroise, T. Levi, Y. Bornat, and S. Saighi. Biorealistic spiking neural network on fpga. In *2013 47th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6, 2013.

[2] Aayush Ankit, Abhronil Sengupta, Priyadarshini Panda, and Kaushik Roy. Resparc: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks. pages 1–6, 06 2017.

[3] Maxence Bouvier, Alexandre Valentian, Thomas Mesquida, Francois Rummens, Marina Reyboz, Elisa Vianello, and Edith Beigne. Spiking neural networks hardware implementations and challenges. *ACM Journal on Emerging Technologies in Computing Systems*, 15(2):1–35, Jun 2019.

[4] Hugh M. Cartwright. *Artificial Neural Networks in Biology and Chemistry—The Evolution of a New Analytical Tool*, pages 1–13. Humana Press, Totowa, NJ, 2009.

[5] FEDERICO CORRADI, GUIDO ADRIAANS, and SANDER STUIJK. Gyro: A digital spiking neural network architecture for multi-sensory data analytics. 2021.

[6] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, January 2018.

[7] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V. Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1117–1125. Curran Associates, Inc., 2015.

[8] Steven K. Esser, Paul A. Merolla, John V. Arthur, Andrew S. Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J. Berg, Jeffrey L. McKinstry, Timothy Melano, Davis R. Barch, Carmelo di Nolfo, Pallab Datta, Arnon Amir, Brian Taba, Myron D. Flickner, and Dharmendra S. Modha. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, 113(41):11441–11446, 2016.

[9] Paulo Garcia, Deepayan Bhowmik, Robert Stewart, Greg Michaelson, and Andrew Wallace. Optimized memory allocation and power minimization for fpga-based image processing. *Journal of Imaging*, 5(1), 2019.

[10] J. Han, Z. Li, W. Zheng, and Y. Zhang. Hardware implementation of spiking neural networks on fpga. *Tsinghua Science and Technology*, 25(4):479–486, 2020.

[11] T. Iakymchuk, A. Rosado, J. V. Frances, and M. Batallre. Fast spiking neural network architecture for low-cost fpga devices. In *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pages 1–6, 2012.

[12] I. Kiselev, D. Neil, and S. Liu. Event-driven deep neural network hardware system for sensor fusion. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2495–2498, May 2016.

[13] Ilya Kiselev, Daniel Neil, and Shih-Chii Liu. Event-driven deep neural network hardware system for sensor fusion. *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2495–2498, 2016.

[14] H. Kopetz. *Event-Triggered Versus Time-Triggered Real-Time Systems*, volume 563, pages 86–101. 10 2006.

[15] L.P. Maguire, T.M. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin. Challenges for large-scale implementations of spiking neural networks on fpgas. *Neurocomputing*, 71(1):13–29, 2007. Dedicated Hardware Architectures for Intelligent Systems Advances on Neural Networks for Speech and Audio Processing.

[16] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, aug 2014.

[17] Parker Mitchell, Catherine Schuman, and Thomas Potok. A small, low cost event-driven architecture for spiking neural networks on fpgas.

[18] S. W. Moore, P. J. Fox, S. J. T. Marsh, A. T. Markettos, and A. Mujumdar. Bluehive - a field-programable custom computing machine for extreme-scale real-time neural network simulation. In *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pages 133–140, April 2012.

[19] D. Neil and S. Liu. Minitaur, an event-driven fpga-based spiking network accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(12):2621–2628, Dec 2014.

[20] P. O'Connor, D. Neil, S. Liu, T. Delbruck, and M. Pfeiffer. Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in Neuroscience*, 7:178, 2013.

[21] Sandeep Pande, Fearghal Morgan, Seamus Cawley, Tom Bruintjes, Gerard Smit, Brian McGinley, Snaider Carrillo, Jim Harkin, and Liam McDaid. Modular neural tile architecture for compact embedded hardware spiking neural network. *Neural processing letters*, 38(2):131–153, 2013.

[22] 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.

[23] Federico Paredes-Vallés, Kirk Scheper, and Guido Croon. Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: From events to global motion perception. 03 2019.

[24] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience*, 12:774, 2018.

[25] David Rolnick and Max Tegmark. The power of deeper networks for expressing natural functions. In *International Conference on Learning Representations*, 2018.

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

[27] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11:682, 2017.

[28] Fabian Scheler and Wolfgang Schroeder-Preikschat. Time-triggered vs. event-triggered: A matter of configuration? pages 1 – 6, 04 2006.

[29] Catherine D. Schuman, Thomas E. Potok, Robert M. Patton, J. Douglas Birdwell, Mark E. Dean, Garrett S. Rose, and James S. Plank. A survey of neuromorphic computing and neural networks in hardware, 2017.

[30] Neha Sharma, Vibhor Jain, and Anju Mishra. An analysis of convolutional neural networks for image classification. *Procedia Computer Science*, 132:377–384, 2018. International Conference on Computational Intelligence and Data Science.

[31] Athul Sripad, Giovanny Sanchez, Mireya Zapata, Vito Pirrone, Taho Dorta, Salvatore Cambria, Albert Marti, Karthikeyan Krishnamourthy, and Jordi Madrenas. Snava—a real-time multi-fpga multi-model spiking neural network simulation architecture. *Neural Networks*, 97:28 – 45, 2018.

[32] Evangelos Stromatias, Dan Neil, Francesco Galluppi, Michael Pfeiffer, Shih-Chii Liu, and Steve Furber. Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on spinnaker. 07 2015.

[33] Evangelos Stromatias, Dan Neil, Michael Pfeiffer, Francesco Galluppi, Steve Furber, and Shih-Chii Liu. Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms. *Frontiers in Neuroscience*, 9, 07 2015.

[34] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47–63, Mar 2019.

[35] S.G. Tzafestas. Neural networks in robotics: state of the art. In *1995 Proceedings of the IEEE International Symposium on Industrial Electronics*, volume 1, pages 12–20 vol.1, 1995.

[36] R. Wang, T. J. Hamilton, J. Tapson, and A. van Schaik. An fpga design framework for large-scale spiking neural networks. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 457–460, June 2014.

[37] Runchun M. Wang, Chetan S. Thakur, and André van Schaik. An fpga-based massively parallel neuromorphic cortex simulator. *Frontiers in Neuroscience*, 12:213, 2018.

[38] Bojian Yin, Federico Corradi, and Sander M. Bohté. Effective and efficient computation with multiple-timescale spiking recurrent neural networks, 2020.