Universitat de Valéncia
Escuela Técnica Superior de Ingeniería
Programa de Doctorado en Ingeniería Electrónica

TESIS DOCTORAL
Spiking Neural Networks models targeted for implementation on Reconfigurable Hardware.

Autor: Taras Iakymchuk
Director: Alfredo Rosado Muñoz
Mayo 2017

Declaration of Authorship

I, Taras Iakymchuk, declare that this thesis titled, 'Spiking Neural Networks models targeted for implementation on Reconfigurable Hardware' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____

Date: ____

UNIVERSITY OF VALENCIA

Abstract

Departamento de Ingeniería Electrónica
Escuela Téchnica Superior de Ingeniería

Doctor of Philosophy

Spiking Neural Networks models targeted for implementation on Reconfigurable Hardware.

by

This thesis describes a novel architecture of the Spiking Neural Networks implemented in hardware using Field-Programmable Gate Arrays. By starting from the state of the art theoretical and practical works, a new approach to the problem is proposed. The presented work is dealing with both software and hardware topics such as:

- Spiking neural models with focus on their performance and feasibility in hardware. A novel simplified neuron model is created and tested.
- Learning of SNNs in software and hardware. The well-known learning algorithms are implemented and tested with the simplified neuron model.
- Data representation and conversion in spiking neural systems. A new version of Address-Event Representation protocol is proposed, effectively allowing the finite automata approach to the SNN implementation. A novel hardware architecture to encode images is presented.
- Hardware platforms' resources and their usability for SNN implementation. The latest commercial FPGA devices are evaluated as the prospective platform for large-scale SNN implementation.
- Spiking perceptron and spiking Restricted Boltzmann machine implementation. Two popular network models are implemented and tested, utilizing the proposed neuronal model.
- Neural network learning in hardware. The previously studied algorithms are implemented in the hardware.

The aforementioned material was partially published in two journal and five conference papers. The system has been fully developed and tested using public domain datasets.

Acknowledgements

This thesis would be never finished without many people, helping me on my journey. I owe all of you.

I am very grateful to prof. Bernabé Linares-Barranco and prof. Giacomo Indiveri for inviting me to visit their fantastic insitutions.

The guy from UV who was making a seminar about ELMs in 2014. I don't remember your name but while trying to solve your problem I came to a vectored architecture. If you read this, thank you!

Karolina Zuniga gave me motivation to finish this work when I was ready to give up.

I will be forever indebted to Alfredo, for taking me into this wonderful journey and patience to my horrible bureaucratic paperwork skills.

My parents and my sister, thank you for your support and understanding. I know, it took me quite a time.

Contents

List of Figures

List of Tables

Abbreviations

AER	**A**ddress-**E**vent **R**epresentation
ANN	**A**rtificial **N**eural **N**etwork
CD	**C**ontrastive **D**ivergence
CLB	**C**ommon **L**ogic **B**lock
DSP	**D**igital **S**ignal **P**rocessor
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
GRF	**G**aussian **R**eceptive **F**ield
IVA	**I**nterleaved **V**TS**A**ER
LFSR	**L**inear **F**eedback **S**hift **R**egister
LIF	**L**eaky **I**ntegrate and **F**ire
MLP	**M**ulti-**L**ayer **P**erceptron
MSE	**M**ean **S**quared **E**rror
PSP	**P**ost**S**ynaptic **P**otential
RBM	**R**estricted **B**oltzmann **M**achine
ReLU	**Re**ctified **L**inear **U**nit
RF	**R**eceptive **F**ield
SNN	**S**piking **N**eural **N**etwork
SRM	**S**pike **R**esponse **M**odel
STDP	**S**pike **T**ime **D**ependent **P**lasticity
TSE	**T**ime**S**tep **E**vent
TU	**T**ime **U**nit
VTSAER	**V**ariable **T**ime**S**lot length **AER**

Resumen de la tesis doctoral

”Spiking Neural Networks models targeted for implementation on Reconfigurable Hardware – Modelos de redes neuronales de espigas para implementación en hardware reconfigurable”

Autor: Taras Iakymchuk

La tesis presentada se centra en la denominada tercera generación de redes neuronales artificiales, las Redes Neuronales Spiking (SNN) también llamadas ‘de espigas’ o ‘de eventos’. Este campo de investigación se convirtió en un tema popular e importante en la última década debido al progreso de la neurociencia computacional. Las Redes Neuronales Spiking, que tienen no sólo la plasticidad espacial sino también temporal, ofrecen una alternativa prometedora a las redes neuronales artificiales clásicas (ANN) y están más cerca de la operación real de las neuronas biológicas ya que la información se codifica y transmite usando múltiples espigas o eventos en forma de trenes de pulsos. Este campo ha ido creciendo en los últimos años y ampliado el área de ingeniería neuromórfica cuya principal área de trabajo es el uso de VLSI analógicos, digitales, mixtos analógico/digital y software que implementa modelos de sistemas neuronales spiking.
Esta tesis analiza las Redes Neuronales Spiking desde la perspectiva de Aprendizaje Automático, donde la plausibilidad biológica no es el objetivo principal, pero la capacidad de crear algoritmos de inteligencia artificial basados en SNN es uno de los objetivos principales, junto con su viabilidad de implementación de hardware. Con el fin de cumplir con los objetivos, varios modelos neuronales y topologías de red son revisados y comparados. La codificación de picos o la representación de datos con los picos también se discute en este trabajo.
El desarrollo de topologías SNN y algoritmos capaces de proporcionar capacidades de inteligencia artificial basadas en espigas de entrada al sistema es uno de los principales temas de esta tesis. Sin embargo, se hace también hincapié en su implementación hardware ya que existen modelos complejos para SNN que en muchos casos no son viables para sistemas en tiempo real y requieren de sistemas de alta capacidad computacional para ser ejecutados.
El tema principal de la investigación en este trabajo es la evaluación de algoritmos

existentes y el desarrollo de nuevos algoritmos, estructuras de datos y métodos de codificación para la implementación hardware de las redes neuronales de spiking, especialmente dirigidas a FPGA (Field-Programmable Gate Arrays). Los dispositivos FPGA son elegidos debido a sus excelentes capacidades de cálculo paralelo masivo, bajo consumo de energía, baja latencia y versatilidad. En los últimos años, las FPGA se convirtieron en una popular plataforma para tareas clásicas de aprendizaje de máquinas, tales como reconocimiento de imágenes, control automático, predicción de series temporales, robótica, etc. Así, la tesis investiga todas las cuestiones relacionadas con el despliegue de un sistema completo de hardware basado en espigas, desde la codificación de información externa como entradas hasta la salida final de un sistema de inteligencia artificial basado en SNN, incluida la optimización en la transmisión de datos, y todo ello implementado en arquitecturas hardware que optimizan el rendimiento y permiten la implementación de redes spiking de un elevado número de neuronas.

Se propone una nueva arquitectura simplificada de neuronas de tipo LIF (Leaky Integrate-and-Fire). La neurona se evalúa para redes de tipo Perceptron y Restricted Boltzmann Machine (RBM) para probar su rendimiento. Además, las capacidades de aprendizaje de las redes propuestas se desarrollan mediante la definición de un procedimiento optimizado para el aprendizaje de STDP (Spike Time Dependent Plasticity). Las propuestas de optimización en software son completadas por nuevas arquitecturas de hardware, especialmente diseñadas para la implementación de FPGA.

En lo que se refiere a las arquitecturas de hardware, esta tesis define la llamada "neurona autómata", basada en un formato de representación de espigas novedoso también y definido en esta tesis, llamado 'Variable Timeslot Length Address-Event Representation' (VTSAER). Este formato tiene una mayor versatilidad que anteriores propuestas de AER, eliminando la necesidad de marcas de tiempo y permitiendo un verdadero sincronismo de cualquier número arbitrario de eventos. La estructura del VTSAER permite procesar la información en las neuronas de espigas como un autómata finito alimentado por eventos. Este nuevo enfoque ayuda a separar el estado del sistema de la tasa de entrada de datos y reducir el número de canales de entrada/salida.

Otra novedad propuesta en esta tesis es una arquitectura vectorizada de capas de las redes neuronales. Esta arquitectura permite calcular el estado de cualquier número arbitrario de capas reutilizando los mismos bloques neuronales de hardware varias veces. Este concepto de procesamiento vectorial de datos se puede aplicar no sólo en las redes neuronales de espigas, sino también en redes neuronales clásicas no-spiking de tipo

ANN y otros algoritmos de aprendizaje automático. Con la arquitectura vectorizada y la neurona autómata, el factor limitante para el tamaño de la red es sólo la cantidad de memoria en el FPGA, lo que es una mejora significativa a las implementaciones anteriores. En cuanto a los algoritmos de aprendizaje para SNN, esta tesis describe una nueva aplicación del algoritmo de aprendizaje de Spike Timing Dependent Plasticity. STDP sigue siendo el algoritmo de aprendizaje más popular para las redes neuronales spiking, derivado de las observaciones de los fenómenos biológicos. Implementaciones de hardware digital de la STDP rara vez se encuentran dado que el algoritmo está utilizando causalidad de sincronización hacia atrás que requiere un empleo significativo de recursos de hardware. La nueva implementación propuesta en esta tesis está resolviendo el problema de causalidad con una sobrecarga de hardware muy pequeña. La versión mejorada de STDP se puede utilizar en redes de número arbitrario de neuronas. El proceso de actualización de pesos es independiente para cada neurona y no afecta al flujo global de entrada de espigas.

La implementación FPGA de algoritmos de codificación visual también se cubre en esta tesis. Se describe la codificación de campos receptivos visuales tipo Gabor y se presentan dos implementaciones de hardware. El método de codificación de campo receptivo es muy similar a la operación de convolución utilizada en redes neuronales no-spiking. Los campos específicos de orientación de Gabor son importantes en el procesamiento de imágenes, ya que son fenómenos bien estudiados observados en la corteza visual de mamíferos y se desempeñan bien en el procesamiento de imágenes y en las tareas de codificación de espigas. Las dos propuestas de implementación en FPGA son arquitectura paralela y vectorizada. La comparación se realiza utilizando tamaños de campo receptivo típicamente usados en tareas prácticas que muestran las posibilidades de aplicación para cada una de las propuestas de implementación.

Además, la implementación del hardware digital de algoritmos requiere la adaptación de la aritmética, ya que la aritmética de punto fijo se utiliza para evitar la complejidad adicional dada por los cálculos de coma flotante. Por lo tanto, se realiza un extenso estudio de la aritmética de punto fijo en el hardware de codificación y procesamiento de spikes para probar que el punto fijo es capaz de proporcionar la exactitud y precisión requeridas a un menor costo computacional y de recursos. Todos los algoritmos y arquitecturas propuestos se prueban resolviendo problemas clásicos con bases de datos abiertos (open source) para poder hacer una comparación con otros autores: los conjuntos de datos SEMEION e Iris se utilizan en este caso. Con respecto a los resultados

de hardware, las arquitecturas digitales propuestas permiten una alta frecuencia de operación de reloj, cercana al máximo permitido por el dispositivo FPGA (alcanza hasta 387MHz). Los algoritmos y arquitecturas propuestos también permiten SNN de tamaño arbitrario, limitándose sólo a la capacidad del dispositivo.

Todas las cuestiones antes mencionadas forman una compleja solución novedosa para la implementación de redes neuronales de espigas en hardware FPGA con velocidad de procesamiento varios cientos de veces más rápido que las simulaciones de software y una precisión comparable. Los bloques de hardware propuestos son versátiles, capaces de implementar una amplia gama de modificaciones de los algoritmos descritos y adaptar múltiples topologías SNN con diferentes números de entradas, número de capas, número de neuronas por capa, número de salidas, longitud de bits y, en general, aquellos parámetros que permiten implementar múltiples formas de SNN.

En total, utilizando los bloques de hardware desarrollados en esta tesis, es posible construir un sistema neuromórfico masivo autosuficiente con un ciclo de procesamiento completo hecho dentro de un chip. De este modo, los sistemas neuromórficos podrían ser implementados a un costo menor en términos de desarrollo y tiempo de diseño, junto con placas de hardware más simples.

Esta tesis doctoral, además del primer capítulo introductorio, está estructurada en varios capítulos.

0.1 Capítulo 2

Se realiza una descripción general de las redes neuronales comenzando por las clásicas redes neuronales artificiales para continuar por los modelos de redes neuronales pulsantes o 'spiking', sus topologías y las metodologías de aprendizaje, haciendo especial énfasis en las técnicas de aprendizaje basadas en la plasticidad. Se describen los modelos biológicos así como los modelo simplificados de Hodgkin-Huxley, Izikevich, LIF (Leaky-Integrate and Fire) y su generalización denominada Spike Response Model (SRM) que define un potencial interno de la neurona (Post Synaptic Potential – PSP) basado en los estímulos recibidos en el transcurso de un tiempo y, si el potencial supera un determinado umbral, la neurona genera un estímulo de salida.

Tras la descripción de los modelos generalmente empleados, esta tesis propone un modelo simplificado de neurona que facilita el cálculo del potencial con vistas a su implementación hardware, reduciendo el uso de recursos lógicos y optimizando la velocidad de operación del sistema. Finalmente, se describen las redes neuronales pulsantes de tipo Multiplayer Perceptron y Restricted Boltzmann Machine (RBM) incluyendo los modelos de aprendizaje STDP y evtCD, que serán utilizadas en capítulos posteriores.

0.2 Capítulo 3

Los sistemas de codificación de la información en estímulos (también llamados pulsos, o eventos) resultan de gran importancia para el posterior análisis de los mismos y un correcto diseño de las redes neuronales que los analizan. Se describen los modelos de codificación en frecuencia, Gaussian Receptive Fileds (GRF) y campos de recepción visuales. Por otra parte, dado el alto número de datos que se generan, se describen modelos de transmisión de la información como Address Event Representation (AER). En este sentido se propone una variante que se ha denominado 'Variable Time Slot Length AER' (VTSAER) que optimiza y mejora la recepción de los eventos por parte de las neuronas, especialmente a la hora de tratar los eventos recibidos para el cálculo del potencial interno. Como parte final del capítulo, se describe el proceso desarrollado para poder codificar los estímulos en el formato VTSAER propuesto.

0.3 Capítulo 4

Esta parte de la tesis evalúa el comportamiento de los nuevos modelos propuestos mediante la aplicación de diferentes conjuntos de datos de entrada y su comparación con otros modelos. Para ello, se toma el conjunto de imágenes de dígitos manuscritos, se le aplica la codificación de campos receptivos de 5x5 en una capa de 256 neuronas de codificación y posteriormente se utiliza una capa de 16 neuronas LIF. Posteriormente, se realiza un entrenamiento STDP con estrategia 'winner depresses all' en lugar de la comúnmente empleada de 'winner takes all'. Tras el entrenamiento, se observa que 10 de las 16 neuronas de la capa LIF son capaces de generar una frecuencia de salida de pulsos específicamente asociada a cada uno de los dígitos (0 al 9). En comparación con otros modelos con el mismo resultado, el tiempo de ejecución es mucho menor para el modelo propuesto (del orden de 15 a 20 veces menor, en función del número de neuronas). Como segunda comprobación, se plantea una red de tipo Boltzmann (Restricted Boltzmann Machine, RBM) con aprendizaje de tipo evtCD para resolver la clasificación del conjunto de datos MNIST consistente en imágenes de tamaño 28x28 píxeles. Por tanto, se tiene una capa de entrada de codificación de eventos de 784 neuronas. Se plantea una segunda capa de 100 neuronas LIF. En comparación con otros modelos LIF, el modelo simplificado propuesto en este trabajo presenta un porcentaje más bajo que el modelo no simplificado (71% y 80%, respectivamente). En cambio, se observa que los resultados en fase de validación son similares en ambos modelos, lo que plantea la opción de realizar el entrenamiento con un modelo no simplificado para posteriormente aplicar los pesos obtenidos a las neuronas simplificadas; esta posibilidad es factible en numerosas aplicaciones donde se realiza un entrenamiento 'off-chip' para posteriormente realizar la ejecución del modelo en aplicaciones de tiempo real 'on chip' con el modelo simplificado.

0.4 Capítulo 5

Dado que los nuevos modelos propuestos están orientados a su utilización en sistemas hardware de tiempo real, este capítulo detalla las arquitecturas hardware planteadas para su implementación eficiente. Los actuales dispositivos FPGA contienen, además de recursos configurables, unidades de cálculo tipo ALU (denominadas bloques DSP) y bloques de memoria RAM distribuida internamente (BRAM). Por ello, la implementación de los modelos neuronales propuestos tienen en cuenta el uso intensivo de estos recursos para optimizar la arquitectura hardware completa, reduciendo el uso de recursos lógicos a emplear y aumentando la velocidad de operación del hardware. En primer lugar, se asume que la entrada de las neuronas estará codificada según el algoritmo VTSAER, de modo que para todas las neuronas de una misma capa, la misma entrada es aplicada a todas ellas en un instante determinado, procesándose las entradas de modo serie (una tras otra). Por otra parte, cada neurona se puede considerar un autómata de estados finitos donde el estado de la neurona depende del estado en el instante anterior y de la entrada, existiendo una evolución de los estados para generar un estímulo de salida

en caso de que el potencial interno de la neurona supere el umbral definido. Para la arquitectura interna de la neurona, se proponen dos modelos: 'binary shift-based LIF' y 'multiplication-based LIF'. La diferencia entre estos modelos radica en el modo de cálculo del potencial de neurona dado que, en un caso, el cambio de potencial ante la recepción de estímulos se realiza en base al desplazamiento de un valor potencia de dos (equivalente a una multiplicación) y en el otro caso, se utiliza directamente la multiplicación, siendo más preciso el cálculo del potencial, pero requiriendo del multiplicador.

Este capítulo describe también la implementación del algoritmo de aprendizaje STDP para las neuronas de estímulos, raramente implementado en hardware debido a la complejidad en el cálculo de tiempos entre la aparición de estímulos de entrada y salida, haciendo necesario el empleo de un hardware complejo. Esta tesis propone en cambio un sistema de entrenamiento STDP con el empleo de un bajo número de recursos y asumiendo, como en el caso de las neuronas LIF simplificadas, el uso de entradas neuronales codificadas mediante VTSAER con una señal de marcación (interleaved VTSAER) para indicar la llegada de entradas retardadas, o bien actuales, permitiendo la obtención de tiempos entre estímulos de modo más directo y realizando el entrenamiento mediante un procedimiento realmente sencillo. La implementación hardware de una neurona de este tipo se realiza de modo que es posible optar por su implementación incluyendo aprendizaje o bien sólo cálculo de entradas, pesos, potencial y generación de salida. Recordemos que numerosas aplicaciones no requieren de entrenamiento 'on-chip', con lo que resulta conveniente que la neurona lo permita, pero se pueda configurar antes de implementar el sistema, con el consiguiente ahorro de recursos lógicos. La neurona está basada en el empleo de un bloque interno de memoria BRAM para almacenamiento de pesos de cada entrada de la neurona, una ALU que contiene un bloque DSP para cómputo aritmético del potencial. Se plantean cuatro variantes: función de potencial lineal, función de potencial con desplazamiento, función de potencial con multiplicación, y función de potencial con multiplicación optimizada en base a la combinación resta-multiplicación. Además, se realiza un análisis del tipo de curva de potencial obtenido para cada una de las versiones.

Para concluir el capítulo, se describe la arquitectura completa de una red neuronal con varias capas, mostrando implementaciones de hasta mil neuronas por capa, siendo importante remarcar que el uso de varias capas supone únicamente un incremento en la memoria empleada para el almacenamiento de los pesos ya que el resto del hardware se reutiliza. Se realiza además un análisis de las limitaciones del uso de sistemas de punto

fijo como es el caso y de los errores producidos en base al tamaño de bits empleado. Con todos los bloques de implementación descritos en este capítulo, resulta posible el desarrollo completo de un sistema neuromórfico capaz de procesar las entradas y generar a un flujo de datos de salida de tipo VTSAER tras su procesado por parte de una red neuronal.

0.5 Capítulo 6

Para los casos en los que las entradas al sistema no sean de tipo 'spike', este capítulo ofrece el sistema hardware que permite la codificación en pulsos según los algoritmos descritos en el capítulo 3, mediante neuronas de codificación. Se proponen cuatro posibles arquitecturas de neuronas de codificación: neurona IF ('integrate and fire'), y neurona con campo receptivo 8x8 Frobenius Inner Product (FIP) implementada de tres formas diferentes (con módulo DSP, con bloques lógicos CLB y con bloques CLB sin posibilidad de reparametrización. Para el cómputo completo de una capa de neuronas de codificación, se describen dos tipos de arquitecturas hardware, una de tipo vectorial y otra de tipo paralelo comparando los resultados a nivel de eficiencia y de precisión de cálculo en función del número de bits empleados.

0.6 Capítulo 7

Este capítulo presenta las conclusiones del trabajo y realiza una comparativa con los resultados obtenidos por otros autores, mostrando los tipos de trabajos presentados hasta el momento en este campo y las posibilidades que la implementación de sistemas neuromórficos ofrece en el futuro, mediante el uso de plataformas hardware adaptables a múltiples tipos de redes neuronales, topologías de interconexión, configuración y entrenamiento para los pesos de las conexiones, etc. En general, se abre el camino para la implementación de sistemas neuromórficos con un alto número de neuronas en dispositivos FPGA de bajo consumo y alta velocidad de operación.

Chapter 1

Introduction

1.1 Goals and aims of this thesis

In this thesis, I studied the feasibility of implementing Spiking Neural Networks (SNNs) in reconfigurable hardware. The main research is focused on efficient computation models for the SNNs and the hardware architecture that will boost the performance of those models. The stress was put on the practical applications of the SNNs, rather than on a biological plausibility. The targeted platform are the widespread Field-Programmable Gate Array (FPGA) devices.

In the recent years, the field of Spiking Neural Networks and Neuromorphic Hardware is the area of special interest. The advantages of this approach: low power, low latency combined with the outstanding cognitive capabilities of the spiking networks are attracting more and more researchers. Neuromorphic hardware finds the way to the commercial market, with the companies like IBM developing TrueNorth spike-operating neuroprocessor and Qualcomm with Zeroth neuroprocessor. However, a number of topics in the hardware implementation of a spike-based system still needs to be researched. The dramatic difference between spiking computation paradigm and the classic ones requires novel approach for maximum utilization of its benefits.

To develop the proposed architectures, an extensive study of existing neuronal and networks models was performed. As the data representation plays important role in the properties and performance of the network models, various types of data conversion and representation were studied. Extra effort was dedicated to Address-Event Representation (AER) protocol, as this data representation is widely used in the existing spiking hardware. In addition, several learning algorithms for SNNs were evaluated from the perspective of their implementation in hardware, since the existing computing models and algorithms are not suited for direct implementation in reconfigurable hardware. The conducted research deals with the hardware implementation problem globally: neural coding, neural models, network topology, computer arithmetic and network connectivity protocols. A novel neuron model is proposed, optimized for the fixed-point arithmetic. The new type of vector architecture, based on the given neuron model, is created and tested to prove its feasibility. A novel version of Address-Event Representation protocol is proposed to be used for this implementation, this new version has several advantages over the existing realizations. The proposed hardware and software components can be used to build massive parallel SNNs on a single chip. Networks with several thousands of neurons are possible to implement within one modern FPGA. The developed system is a full neuromophic processing system: from raw data encoding to the classified values output.

1.2 Structure of this thesis

The current thesis is structured as follows:

- Chapter 2 gives a brief overview of the neural networks field, starting from classic ANNs. Spiking neural models, plasticity and neural network topologies are discussed. This chapter introduces the novel linear simplified spiking model, proposed in this thesis.
- Chapter 3 describes the spike encoding, data representation types for spiking neural networks. A large part of this chapter describes the encoding of graphical information since image recognition tasks are one of the most used applications for neural networks. The Address-Event Representation (AER) format is discussed and a novel variation, called Variable Time Slot length AER (VTSAER) is described.
- Chapter 4 contains the evaluation of the simplified neural model in software. Two types of spiking networks, a feedforward perceptron with Gabor-like Receptive Fields encoding and a Restricted Boltzmann Machine are trained and evaluated.

- Chapter 5 describes the hardware implementation of the proposed neural model and data representation. This chapter describes several novel models of spiking neurons, a vectored large-scale neural architecture and a new implementation of the STDP learning algorithm in hardware. The developed components are analyzed from the perspective of speed, occupation, and accuracy in comparison to software-implemented models. The benefits and the limitations of the contemporary FPGA devices and fixed-point arithmetic are discussed.

- Chapter 6 is dealing with the hardware implementation of Gaussian Receptive Field (GRF) encoding, described in Chapter 3. Two approaches, parallel and vectored, are implemented, tested and compared.

- Chapter 7 contains the summary of the thesis and comparison with the state-of-the-art implementations.

Chapter 2

Neural networks and models: a brief overview

In the middle of 20^{th} century, a combination of biological studies with the mathematical approach to the theory of control created a new scientific discipline: artificial neural networks. An artificial neural network is generally defined as a mathematical model, loosely based on the behavior of the animal neural systems. Such models usually consist of computation units (neurons), connected by data paths (synapses) and are able to solve the vast variety of recognition, control, and system identification problems. The pioneering work in this discipline was made by Warren McCulloch and Walter Pitts [1]. They succeeded in creating a mathematical description of an integrating neural element and the model of connections in the networks of such elements. In their work, they demonstrated that a network of simple non-linear integrators can successfully approximate a number of functions. However, the algorithm to design such networks remained unclear until 1949, when Donald Hebb discovered the learning rule, named after him.

"When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased" Hebb, D.O. (1949), "The organization of behavior", New York: Wiley, p.62 [2]

"When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell." Hebb, D.O. (1949), "The organization of behavior", New York: Wiley, p.63

"The general idea is an old one, that any two cells or systems of cells that are repeatedly active at the same time will tend to become 'associated', so that activity in one facilitates activity in the other." Hebb, D.O. (1949), "The organization of behavior", New York: Wiley, p.70

The first self-learning artificial neural network was the Mark 1 Perceptron machine, created in 1958 by Frank Rosenblatt [3]. It featured two layers (input and output, no hidden layer) of McCulloch-Pitts neurons with binary function and was able to learn the linearly separable classes. The early success of relatively simple hardware and mathematical models draw interest into the neural networks field, but it quickly went down in 1969 after a publication of Minsky and Papert, who described the limitations of the perceptron model and its learning rule. The influence of this book was so strong, that very little research was done for the next 20 years, and several important works in this field were unnoticed, including the backpropagation algorithm. Note the complexity of wiring of the Perceptron hardware, shown in Figure 2.1. The connectivity problem (number of connections grows exponentially with the number of neurons) remains an important problem even now, 60 years later. The architecture described in this thesis is created to avoid this problem by taking the vectored approach to the computations necessary.

The backpropagation algorithm was rediscovered in 1986, and the application of this method for the neural network training by Rumelhart, Hinton, and Williams ([4]) allowed to successfully train multilayer perceptrons, capable of solving complex nonlinear problems. From the 90s, the neural networks field became active again, what lead to the development of a broad diversity of topologies, learning algorithms, coding methods, and applications.

2.1 Biological neurons and neural models

Wolfgang Maass [5] proposed the three-generation classification scheme for the neural networks. The first generation of neural networks works only with binary [0,1] signals and uses a step function to provide a binarized [0,1] output. The perceptrons belong to the first generation. A perceptron computation unit consists of an adder and a threshold

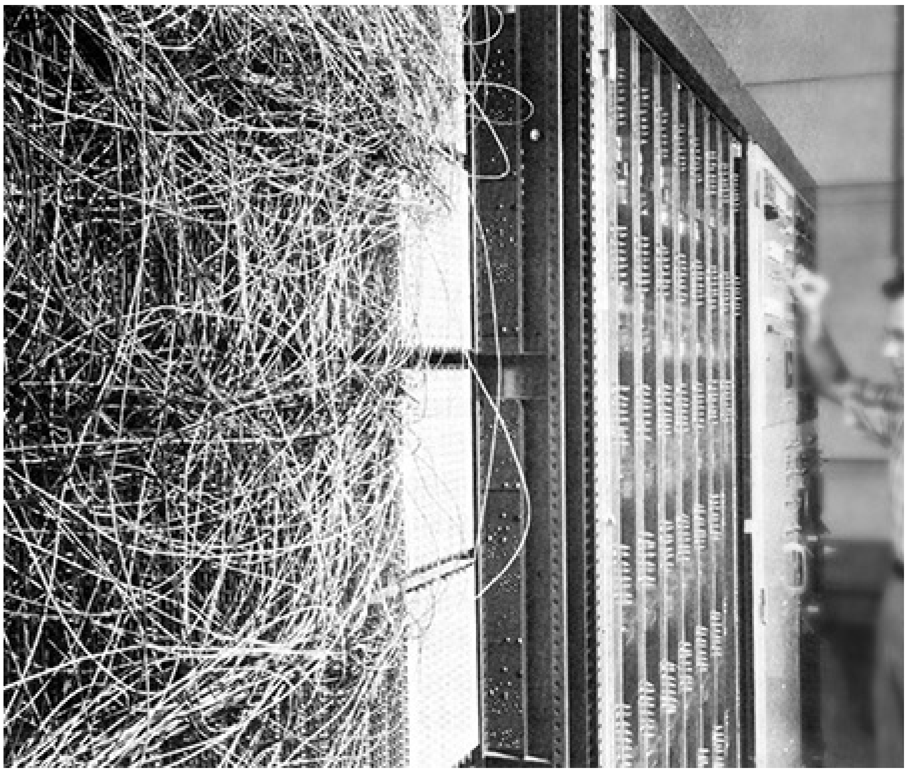


FIGURE 2.1: Mark I perceptron, the first neural network device. Note the amount and complexity of wiring. The connectionist problem (number of connections per node) remains an important issue even now, in 2017.

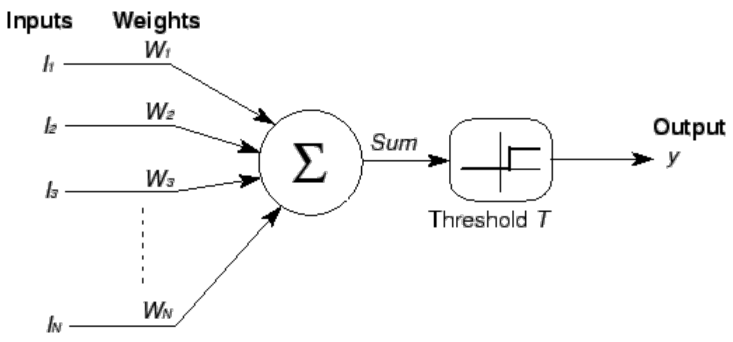


FIGURE 2.2: First generation McCulloch-Pitts perceptron schematic.

comparator, as shown in Figure 2.2. If the sum of weighed binary values is greater than the selected threshold, the output switches to 1.

The second generation works with continuous analog values, usually scaled to the small numeric range and uses a non-linear activation function evaluator. The computing unit, or neuron, is similar to the McCulloch-Pitts neuron. It consists of a multiply-accumulate unit computing the weighted sum of the inputs and the non-linear, usually sigmoid activation function. The output of the sigmoid function is usually bound between [0..1] for the logistic function or [-1..1] for the hyperbolic tangent. In recent years, a novel

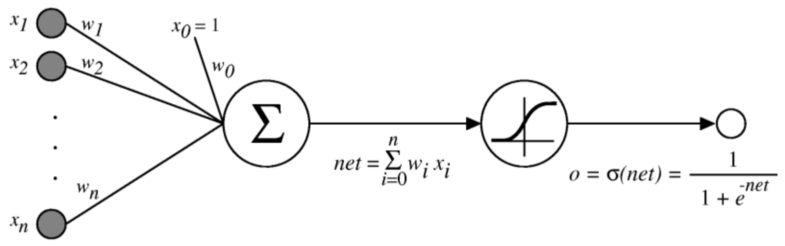

x_1
x_2
x_n
w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

FIGURE 2.3: Sigmoid function neuron schematic. Image courtesy: Tom M. Mitchell. Machine Learning.McGraw Hill. 1997. [7]

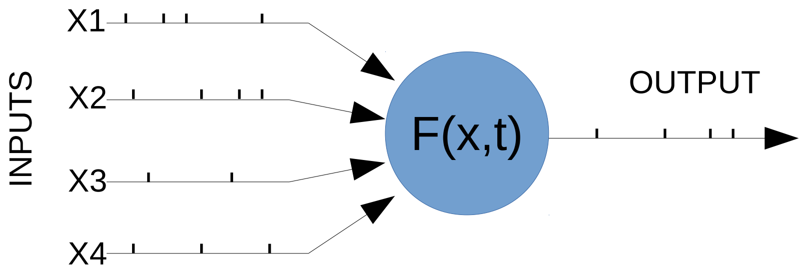


FIGURE 2.4: Spiking neuron model. The output sequence of spikes is computed as a function of input X and time t.

simplified Rectified Linear Unit (ReLU) activation function became popular [6]. The ReLU neurons are computationally simpler and provide a resulting accuracy comparable or even better than traditional sigmoid ones.The schematic of the generic second generation neuron is shown in Figure 2.3.

This work is focused on the third generation of the neural networks, the spiking neural networks [5]. The concept of a spiking neuron is more complex compared to second generation artificial neuron, as the output of the neuron is not only the function of its input (spatial dependency) but also the time. The spiking neuron, shown in Figure 2.4 can change its state even in absence of input stimuli. As the spiking network has both spatial and temporal dependency, its computing capabilities now can include complex processes and transitions. In the spiking neural network, the data are encoded in the series of short pulses, or spikes, transmitted along the data lines, called synapses. The neuron can generate a spike under certain conditions, and these conditions are described by its mathematical model.

The neuron model has a number of tunable parameters, which are important to its functioning. We can change the synaptic weights, transmission delays, stimulation threshold and post-spike response, as well as a number of other parameters. The proper choice of all mentioned variables is crucial for the proper network functionality. The neural models used in this work are described in section 2.3.

2.2 Neural plasticity

There would be a little use of neural networks if they would not have the ability to learn new things by tuning their parameters. The process of adaptation of the network parameters in such form that improves the network response to a given stimuli is called learning. The neural network learning is quite different from the colloquial meaning of this word. S.Haykin [8] defines it as: *Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.* Generally speaking, neural network learning algorithm can be described by the following cycle:

1. The network is given the input, containing the sample of the signal.
2. The network generates an output.
3. The network output is evaluated using some kind to metrics to determine the current fitness of the network.
4. The parameters of the network are tuned with regard of the metrics output from the previous step.
5. The steps 1-4 are repeated until the desired fitness is reached.

There are several types, or paradigms, of learning methods: supervised, unsupervised and reinforced. The most well-known method is the **supervised** learning. During the supervised learning, the network gets input stimuli paired with the desired output response. Knowing the output necessary to be produced, the network can adapt its parameters to fulfill it. Usually, the process of the adaptation is done iteratively, in a number of small steps. The classic example of this type of learning is the previously mentioned backpropagation algorithm applied for the multilayer perceptrons, or Hebbian learning. There are several works regarding the implementation of supervised learning in spiking neural networks (Ponulak and Kasinski, 2011 [9]).

The **unsupervised** learning is done by giving the network only the input stimuli and defining some kind of metrics to estimate the network performance. Such metrics can be the fidelity of the sample reconstruction from the current network state, commonly used for Autoencoder networks, or Restricted Boltzmann Machines. The most common spiking learning rule, the Spike-Time Dependent Plasticity (STDP) is an unsupervised learning algorithm as well [10, 11]. STDP will be further discussed in details.

The **reinforcement** learning requires providing a discrete signal, to inform the network about its performance. The proper outcome of the network is positively reinforced, or rewarded, and the erroneous is punished. The difference between reinforcement and supervised learning is that the network doesn't know the expected outcome, it changes its parameters only in a "reward-punishment" scale. As the neural network has multiple parameters, the learning can be represented as a high-dimensional optimization task. Usually, the learning algorithms are tuning the neuronal weights; however, some of them modify the neuronal thresholds (Diehl et al. [12]). The changes of the neural topology are usually done by eliminating the neurons with the weakest response. This technique is called pruning and is optimizing the network performance as well as its computational complexity.

2.3 Neural models

Due to a very high complexity of living neural cell, spiking neural nets are rarely implemented with high-precision biological models. The most important parts such as potential dynamics, signal propagation, and weight plasticity can be successfully modeled by more simple mathematical descriptions. Multiple spiking neural models exist, but only selected ones are presented in this chapter. The descriptions included here are mostly based on the book *Spiking neuron models: Single neurons, populations and plasticity* by Gerstner and Kistler [13].

The selection of the proper model is a compromise between the degree of biological plausibility and the available computation resources. Classic biological models are focused on the plausibility of the described phenomena and the feasibility of the model to predict the behavior of the model components. Computational neuroscientists are more interested in the model plasticity and large-scale simulation capabilities. In the neuromorphic engineering field, the choice is also limited by the intended hardware platforms

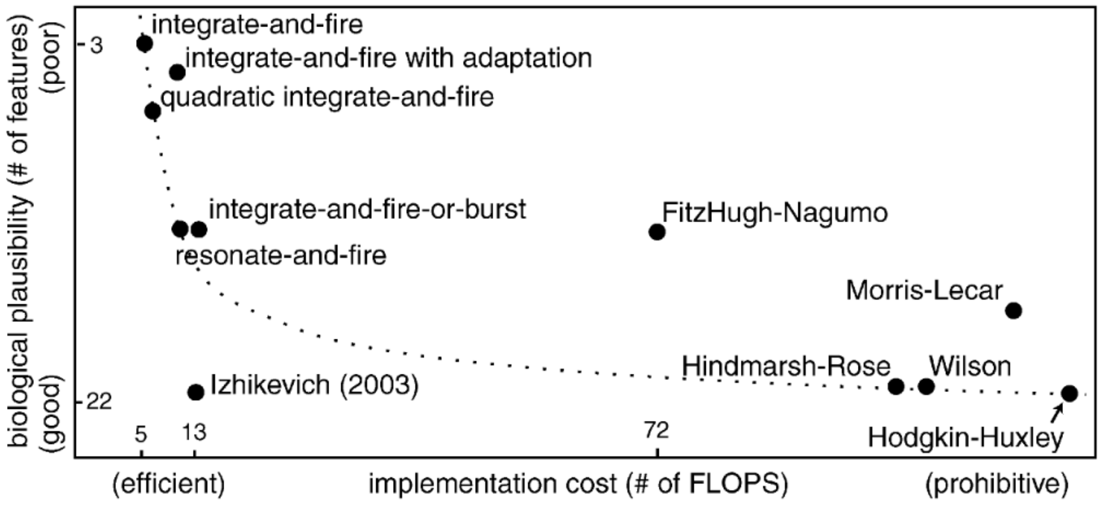


FIGURE 2.5: Plausibility-complexity comparison. FLOPS scale is for digital implementations [14].

as some models with complex behavior can be efficiently modeled only by software. The selected hardware also implies additional limitations: some models are easy to model in analog hardware but have prohibitively high resource occupation for the digital implementation. A plausibility-complexity comparison was made by Izhikevitch [14] and is shown on Figure 2.5. Note that the "biological plausibility" is describing only neuronal behavior, not the internal structure. For example, Izhikevich model, while showing dynamics very close to biological neurons, remains a purely mathematical concept.
This work is treating the spiking neuron as a computing primitive, prioritizing the hardware complexity and learning properties.

2.3.1 Biological spiking neuron

The biological neural cell is a complex dynamic system able to process information delivered by chemicals and electric potential gradient. The complexity of the mammal neural system is overwhelming: the average human body contains around 86 billions of neurons connected by $1.5 * 10^{14}$ synapses. The human brain itself contains between 19 and 23 billions of neurons. The neurons differ by their functions, connectivity type, shape, size and other parameters; however, they all are sharing the same main elements: soma, dendrites, and axon [13], as shown in Figure 2.6.

The **soma** is the main part of the neuron. It contains the cell nucleus as other organelles and is considered the main signal electro-chemical processing unit. Soma is responsible

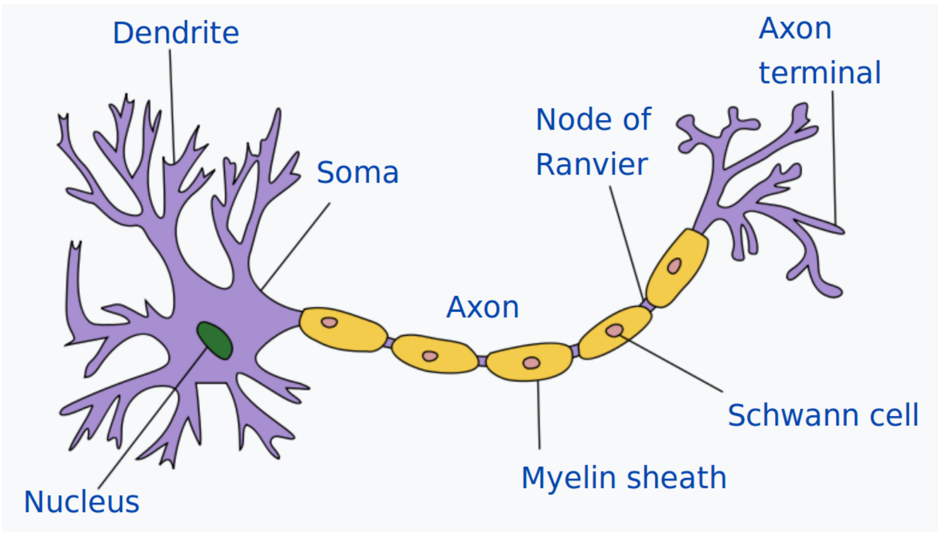


FIGURE 2.6: The most common neuron types and their structure. Image courtesy of [15].

for performing the basic cell functions necessary for the neuron well-being.

The **dendrites** are the extensions of the soma and form together a dendritic tree. The dendrites are attached to the adjacent neural cells via small junctions, called synapses. The dendrites provide inputs to the soma.

The long fine fiber of conductive tissue is called the **axon**. The neuron has only one axon, so it can give only one output at the time. The axon endings can be split into multiple terminals with synapses so that one neuron can be connected via the axon to multiple neighbors. The signal propagation is performed by changing the polarization of the cellular membrane along the axon. The increase of the neurons electric potential opens the Na+ ion channels in the membrane, changing the polarity of the charges on the membrane surface. The area of the oppositely polarized membrane travels along the axon. The saturation of the Na+ ion channels opens the K+ ion channels that restore the membrane to normal, or resting potential. The axon is usually covered by myelin sheath to boost the propagation speed of the signal. The maximum observed signal propagation velocity is 120 m/s in a human neural system.

The **synapse**, shown in Figure 2.7, is the special zone where the communication between the neurons happens. The axon terminal contains molecules of neurotransmitters held by protein membranes, called vesicles. The depolarization of the axon membrane opens voltage-gated Ca+ channels, allowing calcium ions to enter the terminal. This opens the vesicles and neurotransmitters are released into the space between the terminal and the dendrite of the adjacent cell (called postsynaptic cell). The dendrite membranes

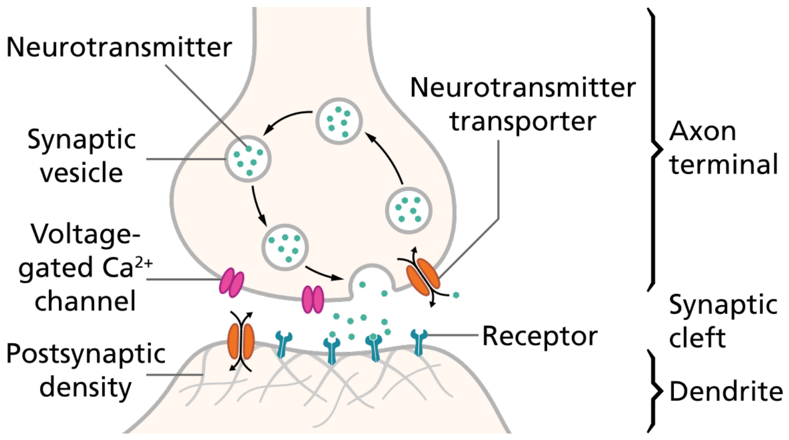


FIGURE 2.7: Biological synapse structure. Image courtesy of [16].

contain the compounds known as neurotransmitter receptors that capture the released neurotransmitters, what changes the chemical equilibrium of the postsynaptic cell. The information is propagated from one cell to another.

The described process, while complex, is very repetitive. As the sodium-potassium channels can be opened only by certain membrane threshold potential, the change of voltage during this event is the same regardless of the frequency of it. This short transition of the membrane potential is called an action potential. As the concentration of ions cannot be restored immediately, the neuron can produce the action potential only once per certain period of time, being irresponsive for the short period after the depolarization-polarization cycle. The rapid change of the potential is also called a spike.
To be able to model the described behavior, a mathematical model of the processes is necessary. There are multiple models with a different approach to the biological plausibility and accuracy. As in this work, the main focus is not on the biological plausibility only the selected models, important for further understanding will be described. The presented models are commonly used in neuromorphic hardware, having a good compromise between computational complexity and performance.

2.3.2 Hodgkin-Huxley Model

Hodgkin and Huxley were the biologists conducting experiments on the giant squid axon. Based on the observations, they created the first electrical model of the neuron that can be expressed mathematically (Hodgkin and Huxley, 1952 [17]).

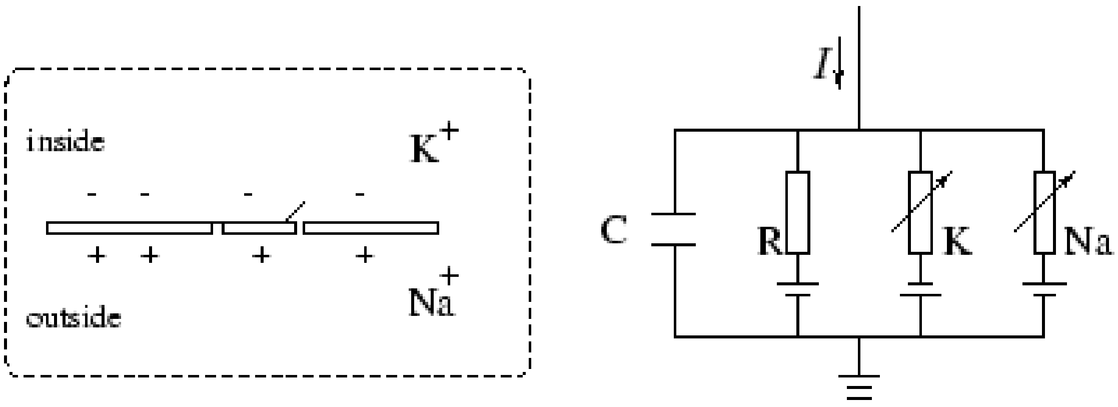


FIGURE 2.8: Hodgkin-Huxley model circuit. Left: simplified synapse membrane schematics with K-Na ion distribution. Right: equivalent electric circuit.

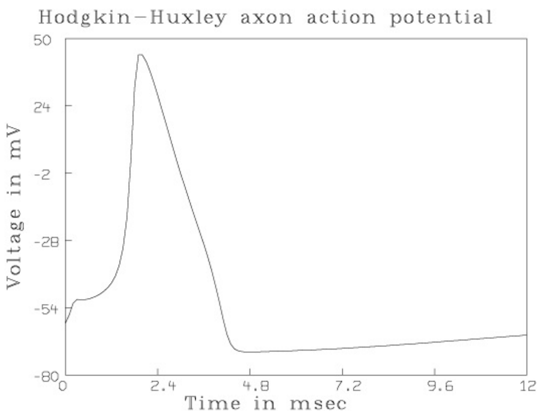


FIGURE 2.9: Hodgkin-Huxley axon action potential vs time in the presence of constant stimulus.

They treated the cell membrane as an electric circuit, where input current I is charging the capacitor C and is leaking through resistors representing the membrane channels. The equation 2.1 has three leakage resistors: a potassium channel, sodium channel and non-specified general leakage.

$$I(t) = I_C(t) + \sum_{k=1}^{3} I_k(t) \tag{2.1}$$

The current flowing through the membrane includes all channels. From the equation 2.1 we can obtain the potential change from the equation 2.2. This equation is describing the general potential dynamic in presence of stimulus.

$$C\frac{du}{dt} = -\sum_{k=1}^{3} I_k(t) + I_C(t) \tag{2.2}$$

Now we know the rule how the membrane potential is changing in time under stimulation.

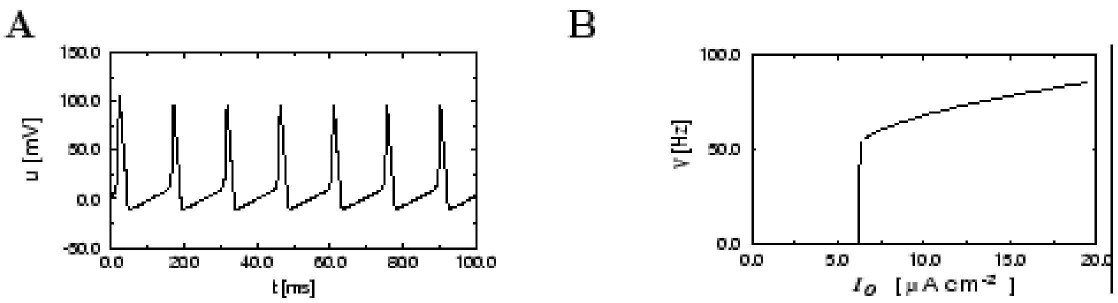


FIGURE 2.10: Hodgkin-Huxley spiking model. A: Sequence of spikes caused by constant stimulation. B: Intensity-frequency plot.

The plot of the du/dt graph is in Figure 2.9. On this plot, the sufficient input current creates a rapid increase in potential up to the threshold level and a rapid discharge after. The time immediately after is called the refractory period. In Figure 2.10 we can see the spike train obtained from the constant stimulus and intensity-frequency plot. The firing rate in the Hodgkin-Huxley model is defined by the pulse duration and the refractory period. These parameters are chosen based on the real biological neuron behavior. The typical membrane potentials are within -80-40 mV range and synaptic currents have 1-20 nA, and the average firing rate range in 1-200 Hz [18].

Hodgkin-Huxley model, being a compromise between biological plausibility and computational complexity is a useful tool for simple neuronal simulations.

2.3.3 Leaky Integrate and Fire (LIF) model

The leaky integrate and fire model is behavioral: it is modeling not the neuron structure, but its behavior [13]. Its relative simplicity makes the LIF a popular choice for the machine learning-oriented SNN simulators, even while it is not biologically plausible. The schematic of the LIF model is presented in Figure 2.11. It consists of an RC integrator (shown in the circle on the right part of the diagram), a threshold switch and a low-pass filter that controls the discharge process.

The input current $I(t)$ is charging the RC circuit. The voltage $u(t)$ on the capacitor is compared with the threshold v by the threshold gate. If $u(t) \geq v$, a gate opens, generating an output pulse $\delta(t - t_i^{(f)})$, which passes through the low-pass filter, generating current $\alpha(t - t_j^f)$. The input driving current $I(t)$ is split into I_R and I_C components. The I_R component corresponds to the current passing through the resistor R. It can be calculated from the Ohm's law as $I_R = u/R$ where u is the voltage across the

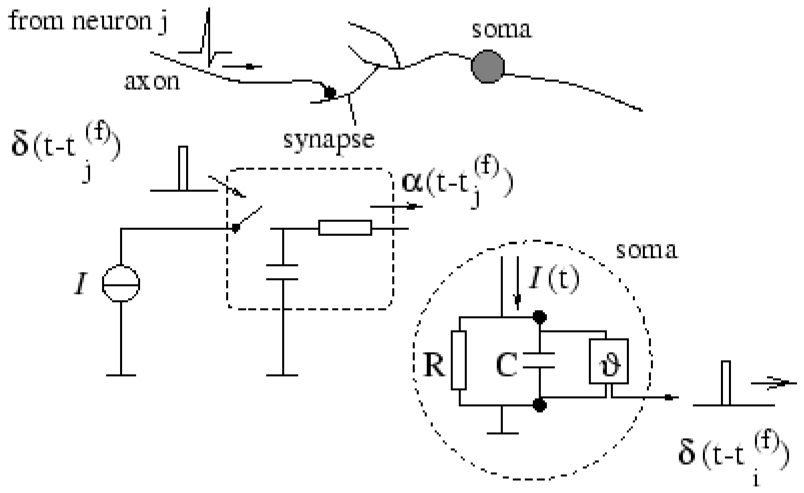


FIGURE 2.11: Leaky integrate-and-fire (LIF) model.

resistor. The I_C component charges the capacitor C. The voltage u is the same on both components. From the capacity equation $C = q/u$, where q is the charge, the capacitive current is equal to $I_C = C\frac{du}{dt}$. The resulting driving current equals to Equation 2.3.

$$I(t) = \frac{u(t)}{R} + C\frac{du}{dt} \tag{2.3}$$

Parameters R and C are constant, we can introduce a time constant $\tau_m = RC$ and yield a general LIF neuron equation 2.4

$$\tau_m \frac{du}{dt} = -u(t) + RI(t) \tag{2.4}$$

Voltage u is the membrane potential and τ_m is the membrane time constant of the neuron.

The spike concept in the LIF is a short current pulse created when a membrane potential reaches the threshold. Immediately after it, the membrane potential is reset to the resting potential and remains unchanged during the refractory period. The main parameters of the LIF model are the threshold υ, membrane time constant τ_m and the refractory period duration T_{ref}. As the LIF model is described by a system of simple linear equations, it is easy to model it numerically or implement in analog hardware. While the model is developed for continuous stimulation (injected) currents, it works with spikes also. The spikes incoming to the neuron are called *presynaptic*, and the outgoing spikes are considered *postsynaptic*. We can see in the Figure 2.12 how the membrane potential is changed with stimuli. While LIF model does not describe cellular membrane

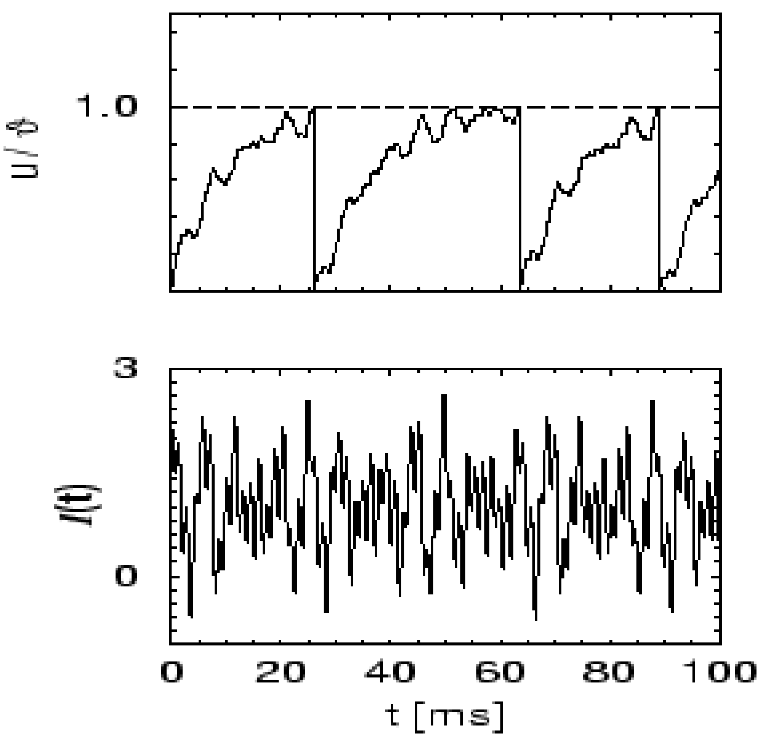


FIGURE 2.12: Leaky integrate-and-fire model dynamics of the neuron driven by irregular current so the membrane positive charge change and leakage can be observed. Injected current is on the bottom plot, membrane potential is on the top plot. When the driving current is negative, the membrane potential decreases.

biophysics with ion channels, LIF model is similar to Hodgkin-Huxley by its time dynamics. Both models do not have the capabilities to express all types of spiking behavior encountered in real neurons but are widely used because of their simplicity. Both models do not have long-term plasticity, and it is usually described as a separate phenomena.

2.3.4 Simplified Spiking Neural model

The classic Leaky Integrate-and-Fire (LIF) model and its generalized form, Spike Response Model [19], are widely used in computational neuroscience. However, LIF spiking neuron models are computationally complex since non-linear equations are used to model the membrane potential. However certain simplifications might be defined in order to reduce computational complexity by proposing a simplified membrane model. The common implementation of the non-linear SRM discards the complex PSP in favor of instant potential change as in Equation 2.5. This work proposes a linear simplified model with similar behavior but more efficient in a digital hardware implementation.

$$P_t = \begin{cases} P_{t-1} + \sum_{i=1}^{K} W_i S_i - \eta P_{t-1}, & \text{if } P_{t-1} < P_{threshold} \\ P_{refract}, & \text{if } P_{t-1} > P_{threshold} \end{cases} \tag{2.5}$$

To avoid confusion with circuit-based models, electrical terms like current and voltage are replaced with mathematical abstract definitions. Let us describe the membrane potential P_t as a function of time and incoming spikes. Time units are counted in discrete time form as the model is intended to be used in digital circuits. For an n-input SNN, during the non-refractory period each incoming spike $S_i, i = [1..K]$ increases the membrane potential P_t by a value of a synapse weight W_i. In addition, the membrane potential is decreasing by a constant value D, every time instant. This process can be described by Equation 2.6, which corresponds to a simplified version of Equation 2.3 in a LIF model. As the membrane potential is decreased by the constant value, a lower boundary condition has to be included.

$$P_t = \begin{cases} P_{t-1} + \sum_{i=1}^{k} W_i S_i - D, & \text{if } P_{min} < P_{t-1} < P_{threshold} \\ P_{refract}, & \text{if } P_{t-1} > P_{threshold} \\ R_p, & \text{if } P_{t-1} < P_{min} \end{cases} \tag{2.6}$$

At each time instant t, if membrane potential P_t is bigger than the resting potential $R_p = 0$, it degrades by a fixed value $P_t = P_{t-1} - D$. The resulting postsynaptic potential(PSP) function will be a saw-like linear function, which is easily implemented by a register and a counter, contrary to classic non-linear PSP models based on look-up tables or RAM/ROM for non-linear equations. The value of constant D is chosen relevant to the maximum presynaptic spike rate and the number of inputs. An example of membrane potential dynamics of the proposed model is shown in Figure 2.13. When $P_t > P_{threshold}$, the neuron fires a spike lasting one minimal time unit, the membrane potential becomes $P_t = P_{refract}$ (resting potential) and a refractory period counter starts. Instead of a slow repolarization of the membrane after the spike, the neuron is blocking its inputs for time $T_{refract}$, and holds membrane potential at a $P_{refract}$ level during this time. To avoid strong negative polarization of membrane, its potential is limited by P_{min}. Despite the model of the neuron is linear, the network can produce a non-linear response by tuning the weights of previous layer inputs, similar to the way ReLU function is used in non-spiking networks. The output spike frequency is bound within given range [20].

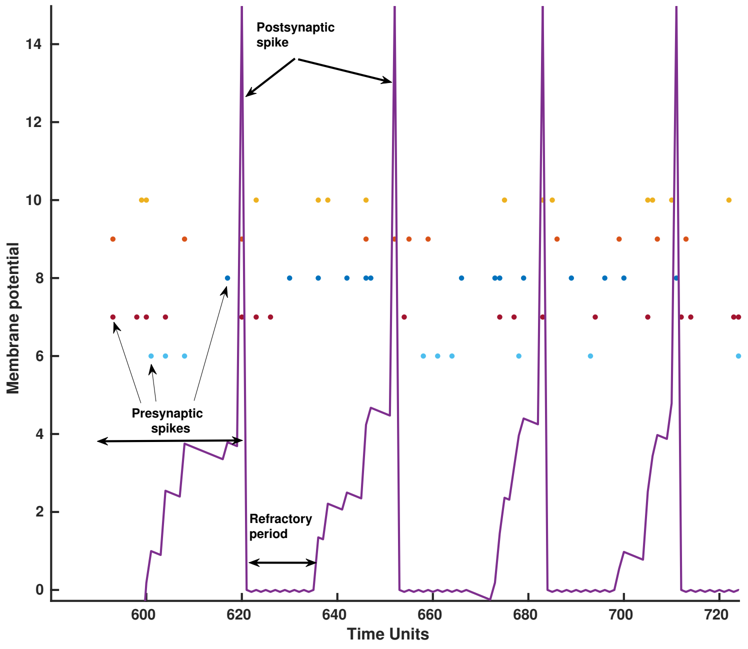


FIGURE 2.13: Membrane potential dynamics of a single neuron with simplified membrane model. After several incoming spikes, the membrane potential surpasses threshold and neuron fires a postsynaptic spike. For better visibility, neuron potential is increased three times for one TU after spiking. During refractory period, neuron does not change its potential. Presynaptic spikes are shown with colored dots.

2.4 Long-term plasticity

In the SNN scope, the long-term plasticity is defined as the ability of synapses to strengthen or weaken over time depending on increases or decreases in their activity [21]. In the simplified phenomenological models described earlier, this corresponds to the changes of the synaptic weights. Thus, the learning in the spiking networks is commonly referred as long-term plasticity.

The most widespread algorithm is called **Spike-Time Dependent Plasticity** (STDP). STDP was observed in biological neurons and is considered to be a nature's equivalent to Hebbian learning rule. The STDP was formally described by Markram in 1997 [22] based on previous observations. The main idea of STDP is that the causality of neural spikes determines the importance of the repeating inputs to discriminate the given pattern. If the presynaptic spike occurred *before* postsynaptic, its synaptic weight increases, and if the presynaptic spike happens *after* the postsynaptic, the weight is decreased. This means that a neuron can become reactive to the specific spiking input, or pattern if the presynaptic spikes have good conformity. This multiple coincidence detection and facilitation mechanism is believed to be the universal biologically plausible learning algorithm.

The strength of the weight change is a function of time between presynaptic and postsynaptic spike events. The function used for STDP learning, shown in Figure 2.14, is a synthetic curve-fitted and does not describe any biochemical process. The classic asymmetric reinforcement curve is shown in Figure 2.14. The function shown is taking arbitrary Time Units (TUs) as an argument. The learning function is described in Equation 2.7 where A^{-} and A^{+} are constants for negative and positive values of time difference Δt between presynaptic and postsynaptic spikes, determining the maximum excitation and inhibition values: τ^{-}, τ^{+} are constants characterizing the steepness of the function.

$$STDP(\Delta t) = \Delta w = \begin{cases} A^{-} \exp^{*}(\frac{\Delta t}{\tau^{-}}), & \text{if } \Delta t \leq -2 \\ 0, & \text{if } -2 \geq \Delta t \leq 2 \\ A^{+} \exp^{*}(\frac{\Delta t}{\tau^{+}}), & \text{if } \Delta t \geq 2 \end{cases} \tag{2.7}$$

The learning rule (weight change) is described by Equation 2.8. The desired distance between presynaptic and postsynaptic spike is unity when the postsynaptic spike occurs one time unit after the given pattern of postsynaptic spikes arrived. The weight change rate σ controls the weight adaptation speed.

$$w_{new} = \begin{cases} w_{old} + \sigma \Delta w, & \text{if } \Delta w \geq 0 \\ w_{old} + \sigma \Delta w, & \text{if } \Delta w \leq 0 \end{cases} \tag{2.8}$$

In computational neuroscience, STDP-type learning describes a broad range of weight update rules where the value of the weight update is determined by the causality of the

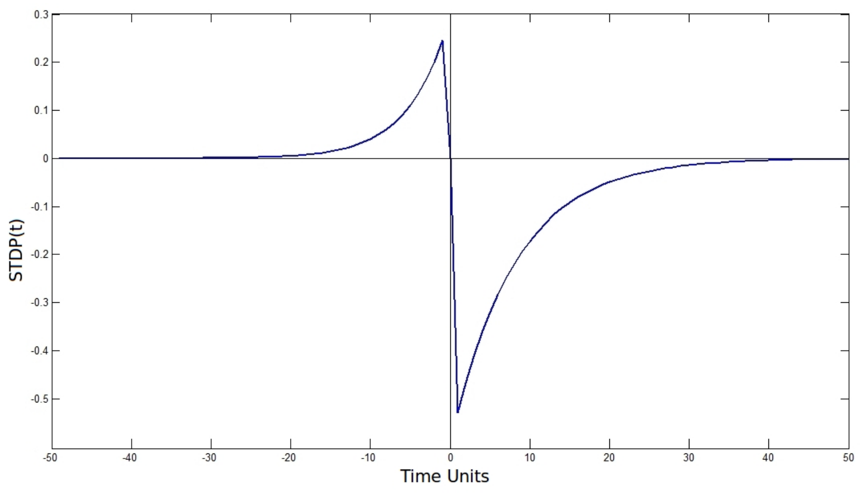


FIGURE 2.14: STDP curve used for learning. This type of curve has stronger depression value than potentiation, increasing specificity. $A^+ = 0.6, A^- = 0.3, \tau^+ = 8, \tau^- = 5$.

presynaptic and postsynaptic events. The represented function is describing only one type of the STDP learning.

2.5 Neural networks types

In this section, two types of neural networks are presented, used with classic and spiking neurons as well. Multi-layer Perceptron and Restricted Boltzmann Machine are examples of different network topologies and have different learning algorithms. They belong to one of the most well-known types of the networks and are widely used in various Machine Learning (ML) tasks. These topologies are widely used in the classic and spiking variations.

2.5.1 Multilayer Perceptron

The multilayer perceptron neural network is undoubtedly the oldest and the most well-studied type of neural networks. The concept can hardly be credited to only one person or team, but MLPs became common after the Rumelhart, Hinton and Williams work [4], after they presented an efficient algorithm for learning them.

Mathematically, MLP is a feedforward neural network with fully connected one or two hidden layers (the layers that are not input or output ones) of neurons with non-linear

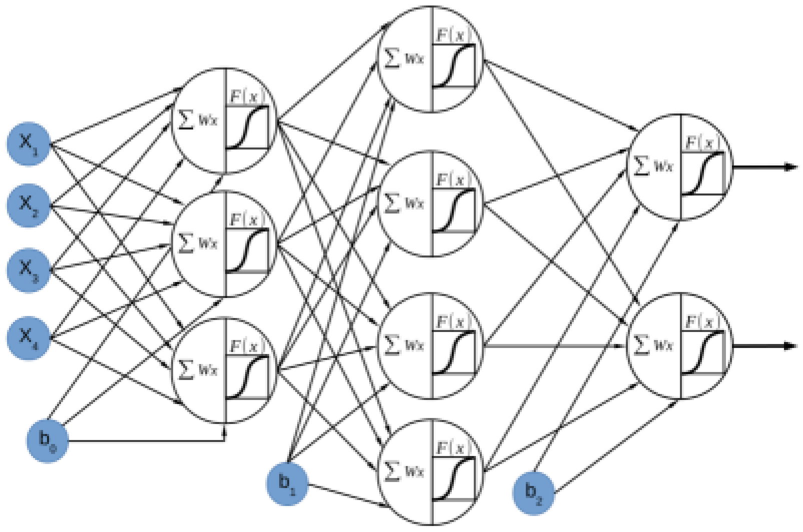


FIGURE 2.15: Multilayer perceptron schematic.

activation functions. The topology of the MLP is presented in Figure 2.15. The main features of the MLP architecture are:

1. No connections exist among the units in the same layer.
2. The output of a layer is a function of the previous layer inputs and a bias.
3. All units in a layer share the same activation function and can be computed independently. Different layers can have different activation functions or it can be skipped as for the last layer.

Every neuron in one the hidden layers of MLP is performing the function described by Equation2.9:

$$Y = F(\sum(\vec{W} * \vec{X}) + b) \tag{2.9}$$

Where $\vec{W}$ is a vector of weights of the neuron, $\vec{X}$ is an input vector and b is a bias value. Note that multiplication is performed elementwise. The F is a non-linear activation function of the neuron.
The whole network can be described as a massively parallel computation of Equation 2.9 for each neuron in the network. The difference for each unit lies in the inputs and weight values, which are those shown in Equation 2.10 where $\vec{Y_j}$ is the computed layer output obtained from the previous layer $(\vec{Y_{j-1}})$ outputs or input layer $\vec{X}$; $\mathbf{W^j}$ and $\vec{b_j}$ are

the weight matrix and bias vector of a given layer j, respectively. The total number of layers is i, and the activation function is F.

$$\begin{aligned}
\vec{Y_0} &= \vec{X} \\
Y_{11} &= F(\sum_j (W^1_{1j} * Y_{0j}) + b_{11}) \\
Y_{12} &= F(\sum_j (W^1_{2j} * Y_{0j}) + b_{12}) \\
&\vdots \\
Y_{i1} &= F(\sum_j (W^i_{1j} * Y_{(i-1)j}) + b_{i1}) \\
Y_{i2} &= F(\sum_j (W^i_{2j} * Y_{(i-1)j}) + b_{i2})
\end{aligned} \tag{2.10}$$

The activation function F serves as a nonlinear regularizer, limiting the information passed between the layers. The traditionally used hyperbolic tangent and logistic sigmoid functions are now often replaced by Rectified Linear Unit (ReLU) activation function [6] due to the less computational complexity and good learning properties.

The classic MLP learning algorithm is a backpropagation; however other methods do exist as well.

Spiking feed-forward networks, such as perceptrons or autoencoders are the most popular types of the SNNs. However, the number of layers is usually much less compared to classic MLPs, as the efficiency of known spike training algorithms is much less for deep networks. The existing implementations of spiking perceptrons can be split into three groups: the networks that use biologically plausible iterative learning (such as STDP) [10, 11, 23, 24], SNNs that are trained using mathematical methods (backpropagation, evolutionary algorithms, etc) [9] and the networks that are a product of conversion from classic ANN into SNN[12, 25]. Very often, SNNs that are trained through iterative algorithms, while having feedforward signal propagation, also feature lateral neural inhibition, i.e. a negative weight connection between all neurons in a layer. Due to the nature of spiking learning algorithms, the addition of inhibitive connections makes the training more efficient. It is implemented either as negative weights, or an array of mirroring inhibitory neurons (every excitatory neuron in the layer has its inhibitory counterpart). The addition of purely inhibitory neurons is a biologically plausible well-known mechanism [26].

2.6 Restricted Boltzmann Machine

The Restricted Boltzmann Machine (RBM) is a stochastic neural network, invented by Paul Smolensky in 1986 [27], but it gained the popularity only after the classic Hinton and Salakhutdinov paper on efficient training algorithm for RBMs [6, 28]. The RBM is a generative model, which means that the network tries to recreate the input state from the superposition of hidden layer activations.

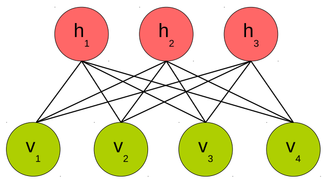


FIGURE 2.16: Restricted Boltzmann Machine network.

Classic RBM consists of two layers: visible v (input) and hidden (output) h, connected by bilateral synapses. It means that the weight matrix $\mathbf{W}$ is shared for both layers, and the state of the input layer can be recreated from the state of the hidden one. Hinton [28] proved that the probabilistic RBMs have better performance and are more robust. In the probabilistic RBM, the output of the neuron is computed by the formula 2.11

$$\begin{aligned} P(h_j = 1|v) &= \sigma(b_j + \textstyle\sum_{i=1}^{m} w_{i,j} v_i) \\ P(v_i = 1|h) &= \sigma(a_j + \textstyle\sum_{j=1}^{n} w_{i,j} h_j) \end{aligned} \tag{2.11}$$

In this equation, the $P(h_j = 1|v)$ is the probability that the j-th neuron in the hidden layer h will have the state of 1, given the input vector v. σ is the logistic sigmoid function, and m is the size of vector v. The $P(v_i = 1|h)$ is the probability that the i-th neuron in the visible layer v will have the state of 1, given the h as the input vector. The size of the hidden vector is n, a and b are the bias vectors for the visible and hidden layers.

The training of RBM is a process to find the matrix $\mathbf{W}$ that provides the best reconstruction of visible vectors from the given dataset. The networks are usually designed such that $n < m$, thus the reconstruction is performed from the smaller vector. Thus, an RBM can serve as linear dimensionality reductor, outperforming many traditional methods [29, 30]. The most popular method of training RBMs is the Contrastive Divergence. The basic version of the algorithm is the following:

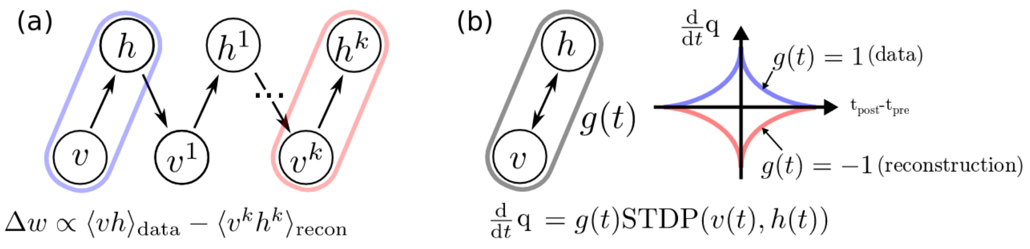

(a)
h
h^1
h^k
v
v^1
v^k
$\Delta w \propto \langle vh \rangle_{\text{data}} - \langle v^k h^k \rangle_{\text{recon}}$
(b)
h
v
$g(t)$
$\frac{d}{dt}q$
$g(t) = 1$ (data)
t_{post}-t_{pre}
$g(t) = -1$ (reconstruction)
$\frac{d}{dt}q = g(t)\text{STDP}(v(t), h(t))$

FIGURE 2.17: Event-based CD algorithm. The RBM is unrolled into Monte-Carlo Markov Chain. The correlation between visible and hidden layer causes weight potentiation; the correlation between reconstruction of layers causes depression Source: [31].

1. Assign the input vector values to the visible vector $\vec{v}$.
2. Compute the states of the hidden vector based on current values of v.
3. Compute the outer product **pos** of vectors $\vec{v}$ and $\vec{h}$. $\vec{pos} = \vec{v}\vec{h}^T$
4. Compute the *reconstruction* of the visible vector v' from the current values of h.
5. Compute the *reconstruction* of the hidden vector h' from the current values of v'. $neg = v'h'^T$.
6. Compute the outer product **neg** of the vectors $\vec{v}'$ and $\vec{h}'$.
7. Update the weight matrix W with $\mathbf{\Delta W} = \eta(\mathbf{pos} - \mathbf{neg})$. η is the learning rate.
8. Update the biases to visible and hidden layers: $\Delta a = \eta(v - v')$,$\Delta b = \eta(h - h')$.

The use of binarized states of the layers makes the RBM similar to spiking neural networks, and the weight update rule is somehow similar to the STDP: the weight is facilitated when both visible and hidden neurons are activated, and depressed when their reconstructions are both activated. This fact is used for creating the Spiking RBMs. The similarity between RBM binarized data processing and spike network naturally lead to use the RBM topology and learning methods for the spike-represented data. Two implementations are important to be mentioned in this work. First spiking RBM implementation with STDP-derived learning rule was presented by Neftci et al. in 2013 [31]. The main idea of his algorithm, called event-based CD (evtCD) is presented in Figure 2.17. This work uses LIF neurons with sigmoid transfer function organized in separate layers for data-driven input and for the reconstruction, so topologically the network becomes feedforward. For weight update the symmetric STDP-derived rule is used, when simultaneous (within certain time window) firings of visible and hidden layer cause potentiation and simultaneous firings in the reconstruction layers cause depression. With such rule, the authors were able to obtain up to 92% accuracy on MNIST dataset. Another STDP-based approach to the RBM learning was proposed by Daniel Neil in his thesis [32]. He also unrolled RBM into 4-layer feedforward network with shared weights and used LIF neurons for it. The weight update rule has fixed potentiation/depression step as shown in Equation2.12.

$$\Delta W_{ij}^{+} = \begin{cases} \eta, if h_i = 1, v_j = 1 \\ 0, otherwise \end{cases} \qquad \Delta W_{ij}^{-} = \begin{cases} -\eta, if h_i' = 1, v_j' = 1 \\ 0, otherwise \end{cases} \tag{2.12}$$

These implementations show that the RBM is a viable topology for spike networks and can be used for practical classification tasks.

Chapter 3

Neural encoding

The discrete or analog information needs to be converted into sequences of spikes to be processed with the SNN. A number of sensors with spike-generating output exist [33–35], and only such sensors can be directly used with the neuromorphic hardware. The most popular types of such sensors are retinas, cochleas, and physical position sensors. Usually, such sensors combine the rate-based coding with the Address-Event Representation (AER) protocol. This approach is the de-facto standard for the neuromorphic sensory hardware and allows seamless data fusion within one network. For example, the work of [36] is combining the visual information with sound as a proof of concept of a multimedia cognitive system. However, such sensors are still not very common yet and the problem of converting data into spikes is very important. The neural encoding defines not only the way the which information is stored, but also influences the processing algorithm and the hardware implementation possibilities.

3.1 Rate-based encoding

The traditional coding scheme directly encodes analog values into spike trains with the intensity proportional to the analog value, as shown in Figure 3.1. Such encoding can be encountered in biological systems [37], but it has limited usability in the machine learning field, because of the disparity of intensity between classes. SNNs naturally favor more spike-intense signals, thus, the normalization of the input is important. It can be successfully used in different applications as [38].

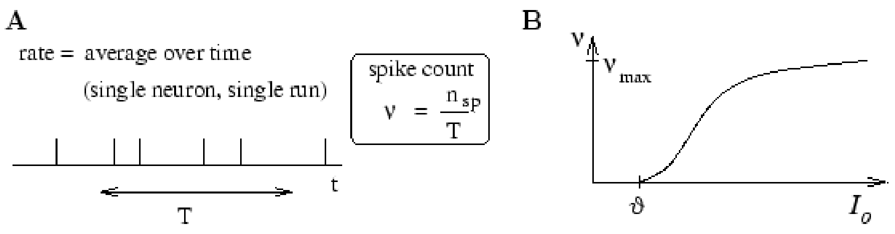


FIGURE 3.1: Rate-based coding of analog stimulus (plot B) into a spike train (plot A). Image source: [13].

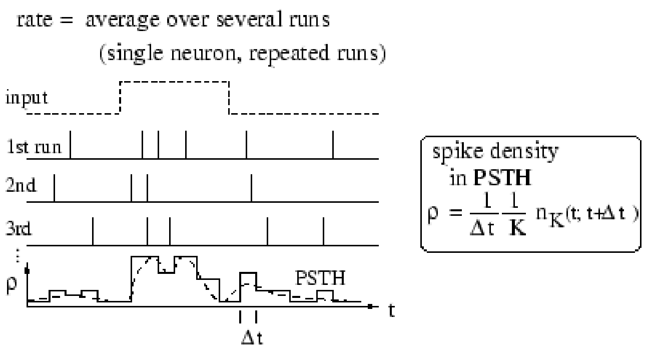

rate = average over several runs
(single neuron, repeated runs)
input
1st run
2nd
3rd
ρ
PSTH
t
Δt
spike density
in PSTH
$\rho = \frac{1}{\Delta t}\frac{1}{K} n_K(t; t+\Delta t)$

FIGURE 3.2: Average spike density coding of analog stimulus. Image source: [13].

In order to improve the spike response and robustness to noise in large neuronal populations, a more complex average rate-based encoding is used. It can be considered as an averaged merging of several stochastic spike trains over the same stimulus. The example of such encoding is shown in Figure 3.2. The resulting spike train represents the average spike density over time for given stimulus.

3.2 Position coding

In the position coding, every input value is encoded into several spike trains, providing not only frequency but also a spatial distribution of the spikes. Sometimes it is called Gaussian Receptive Fields coding (GRF), because of the widespread usage of Gaussian spike distribution. A set of points $\vec{x}$ is evenly distributed within the stimuli input range $[Min..Max]$. Every point corresponds to a separate spike train. The spike frequency for the specific spike train is selected by the Gaussian normal distribution function described by the Equation 3.1), where a is the maximum spiking frequency, x_0 is the nth point within the input range and σ is the Gaussian kernel width. This type of coding is shown in Figure 3.3. Five overlapping GRFs are generating five spike trains with the firing rate proportional to the distance between the current input and the corresponding GRF

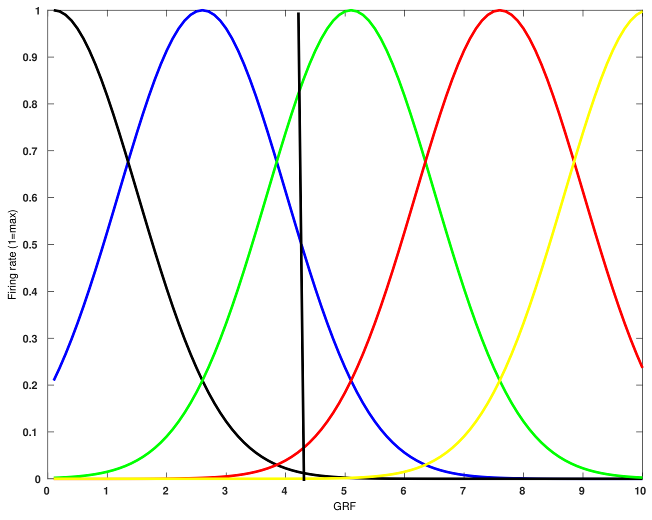


FIGURE 3.3: GRF encoding of the [0...10] range with 5 fields (5 spike trains). The black vertical line denotes one analog input, and the points of crossing with Gaussian curves give the corresponding spiking frequency normalized in [0..1] range.

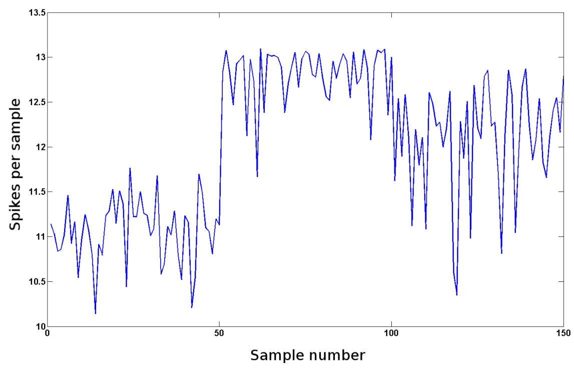


FIGURE 3.4: Average amount of spikes per sample for Iris dataset. First 50 samples are for the *Setosa*, samples from 51 to 100 are for *Versicolor* and the last 50 samples are the *Virginica* species. Note that the variation of the spike density is less than 20% between the classes.

center. The wider fields are, the more spike trains are generated by a single input value.

$$f(x) = a \exp -(\frac{x - x_0}{2\sigma})^2 \tag{3.1}$$

This coding provides the average spike density roughly equal for all dataset regardless of the input values, what is important for the training. For example, the average spike frequency for the samples from the Iris dataset [39] is shown in Figure 3.4.

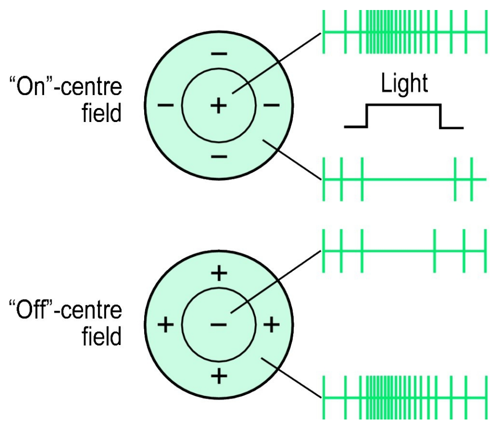


Figure 3.5: Off-centered and on-centered neural receptive field and corresponding spike trains. *Source:* [41].

3.3 Visual receptive fields

The visual information is more complex to encode, as it is usually represented in 2D with several input channels per pixel (color, luminosity). With the bio-inspired algorithms, it is possible to get more efficient spike encoding than with basic rate-based models.

The visual neural cortex is one of the best studied parts of the brain. The receptive field (RF) of a visual neuron is the specified part of the image affecting the neural input. The size and shape of receptive fields vary heavily depending on neuron position and purpose. By modifying the receptive field one can make a neuron sensitive to an object position, orientation, and shape. On each subsequent layer of visual cortex, receptive fields of the neurons cover bigger and bigger regions, convoluting the outputs of the previous layer.
Mammalian retinal ganglion cells located at the center of vision, in the fovea, have the smallest receptive fields and those located in the visual periphery have the largest receptive fields [40]. The large receptive field size of neurons in the visual periphery explains the poor spatial resolution of our vision outside the point of fixation (other factors are photoreceptor density and optical aberrations). Only a few cortical receptive fields resemble the structure of thalamic receptive fields, while others have elongated subregions that respond to either dark or light spots, others respond similarly to light and dark spots through the entire receptive field and others do not respond to spots at all.

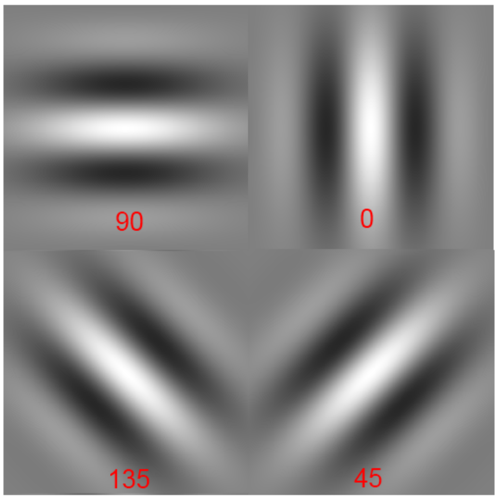


FIGURE 3.6: Different types of Gabor RF.

3.3.1 Receptive field neuron response

The neurons in the receptive or sensory layer generate a response defined by Equation 3.2.

$$R_{RF} = \sum I_{ij} * W_{ij} \tag{3.2}$$

I_{ij} defines input intensity and the matrix W_{ij} defines a receptive field of the neuron, where i is X-axis resolution and j is Y-axis resolution. The field can be off-centered or on-centered as it was shown in Figure 3.5, or it can describe a Gabor-like filter, shown in Figure 3.6. These RFs can be used as line detectors, small shape detectors, performing feature extraction for higher layers. Simple classification tasks (i.e. the inclination of the line, circle or non-circle object) can be performed by single-layer receptive field neurons. Having normalized input and weights, the maximum excitation will be achieved when the input exactly matches the weight matrix.

Sensory layer neurons generate spikes at a frequency proportional to their excitation. As the neuron firing frequency can not be infinite, the maximum firing rate is limited, and thus, the membrane potential is normalized. The spiking response firing rate for the nth spiking input (FR_n) is described by Equation 3.3, where R_n is the receptive field response, $RPmax$ is the defined minimum refractory period and $max(R)$ is the maximum possible value of membrane potential.

$$FR_n = \frac{1}{RPmax * \frac{R_n}{max(R)}} \tag{3.3}$$

3.3.2 Gabor filters

Gabor filter, named after Dennis Gabor, is a band-pass filter widely used in digital signal processing in its time domain form and in image analysis in its space domain form. A family of two-dimensional Gabor functions was proposed by Daugman [42] as the mathematical models of simple receptive fields in visual cortex cells [43]. These functions are described by Equation 3.4

$$\begin{aligned} g_{\lambda,\Theta,\phi,\sigma,\gamma}(x,y) = exp(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2})cos(2\pi\frac{x'}{\lambda}+\phi) \\ x' = xcos\theta + ysin\theta \\ y' = -xsin\theta + ycos\theta \end{aligned} \tag{3.4}$$

The shape and orientation of such receptive field can be selected by modifying the parameters of Equation 3.4. Standard deviation σ of the Gaussian determines the linear size of the receptive field. γ determines the spatial aspect ratio, i.e. the ellipticity of the RF. λ is the wavelength and by modifying σ/λ ratio one can modify the number of inhibitory and excitatory zones in the RF. ϕ responds for the symmetry of the RF and θ specifies the orientation of the RF. x' and y' are imaginary components of the filter response along the XY axis.
Gabor fields have good orientation selection properties and are widely used for image decomposition. For example, the image of a car in Figure 3.7 is converted by 4 Gabor field filters with orientations of 45°, 135°, 0°, and 90°. It is clearly seen in Figure 3.8 that the lines in the direction corresponding to the receptive field are promoted, while other lines are depressed.

3.4 Address-Event Representation

The spike trains, regardless of its encoding, have to be stored in some format understandable by the software or hardware it will be using. The most simple way is using a 2D array, usually called spike raster plot (Figure 3.9). The traditional wide raster plot

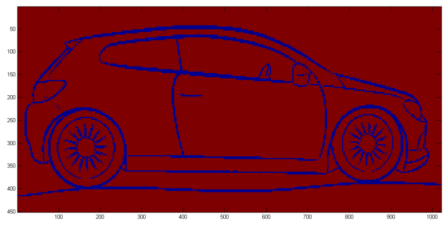


FIGURE 3.7: Sample monochromatic image of a car.

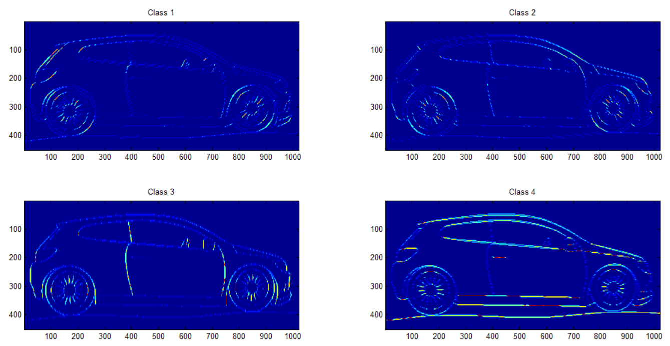


FIGURE 3.8: Sample image, converted with 4 different receptive fields with orientation of 0° ,45° , 90° and 135°.

representation is quite bulky, the memory occupation is static and is determined by a number of neurons and the spike train length regardless of the number of spikes in the plot. In the hardware, the closest representation of the raster plot will be a parallel bus accepting spikes in real time, what is impractical for a high number of inputs.

In 1992, Misha Mahowald in her PhD thesis described a novel interface tailored to the hardware needs [44]. The AER interface became a de-facto standard in the neuromorphic hardware field, having numerous uses [33, 34, 45, 46]. There are several variations of the AER. In the original AER every spike is encoded as the address of its source, so the spike train is converted into a sequence of $log_2(N)$-bit words, where N is the number of event sources (neurons) in the system [47]. Every spike occupies a single time "slot" and the absence of spikes is not signaled on the line. Such encoding does not allow for true spike synchronicity (two spikes cannot share one time slot) and being asynchronous protocol, requires additional data lines to ensure data integrity (usually REQ/ACK handshake). Original AER communication scheme is shown in Figure 3.10. The transmitter raises the REQ signal when the data on the bus is ready, and waits for the high ACK signal, raised by the receiver. Receiving the acknowledge releases REQ back to low, and this is driving ACK low too. This ends the asynchronous data transfer.

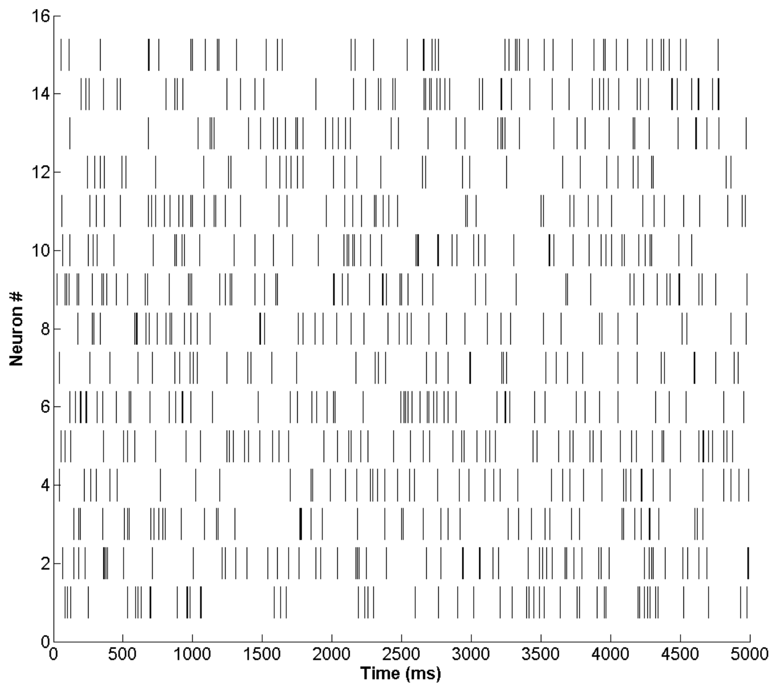


FIGURE 3.9: Sample spike raster plot of 15 neurons spiking over 5000 ms.

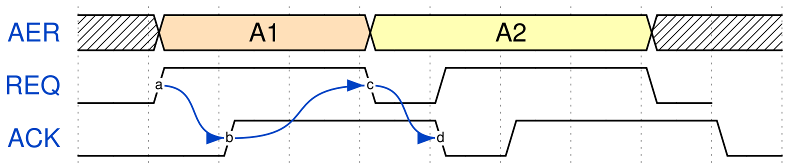


FIGURE 3.10: Asynchronous AER data communication. The transmitter sets the data on the **AER** bus, raises the request signal **REQ** high and waits for the acknowledge signal **ACK** from the receiver. After this, **REQ** goes low, driving *ACK* low. The cycle can be repeated.

The lack of synchronicity is partially solved by *timestamped AER* [48]. In this version of the protocol, every spike or event bears the emission timestamp. The timestamp can be either absolute or relative, equal to the time passed from the previous event. Two spikes from different sources with the same timestamps are considered to be simultaneous. Timestamped AER can efficiently transmit compressed streams as the timestamps can be arbitrarily increased and there is no need to transmit null events. However, it has its limitations: timestamps reduce bandwidth and have the continuity problem. For the 2-byte timestamp where minimum time change Δt is 1 ms, the timestamp will be repeated after 16,635 s what can be incorrectly interpreted by the receiving hardware. Increasing the timestamp field length reduces bandwidth even more. Not all systems

can process timestamped AER, especially when the events are sparse. Usually, the processing is done in realtime or accelerated real time, when timestamped AER is used only in the data transmission and storage protocols [45]. The methods of AER encoding and decoding are extensively studied in [49].
In this work, an AER variation is developed to simplify the design of the efficient hardware neural system.

3.4.1 Variable timeslot length AER (VTSAER)

This work describes the modification of the AER protocol to better fit hardware spike processing systems. It consists of a non-timestamped protocol where a special Timeslot Separator Event (TSE) word is denoted as the separator of the discrete time slots. The spike stream is an uninterrupted sequence of events, and all spikes between two TSEs are considered simultaneous. The TSE denotes the end of one timeslot and the beginning of the next one. The absence of spikes will be encoded as two consecutive TSEs without any events between them. An example of VTS AER is shown in Figure 3.11. The events with address **A1** and **A2** are occurring simultaneously in time slot #3, in the next time slot no spikes occur, event sources **A2**,**A3** spike in time slot #6, followed by the events **A1** and **A3** in time slot #8. The synchronous AER waveform demonstrates how an asynchronous stream is sampled and perceived by the synchronous receiver with the minimum acknowledge time is one clock cycle. By using a TSE, it is possible to construct the spiking neuron as a finite automaton solely driven by the input, without internal counters. This simplifies its hardware structure, as it will be shown in Chapter 5. Also, the VTSAER protocol can be expanded by introducing the null event, which is equivalent to the NOP operation in processor. The null event is the event which synaptic weight is zero; Such input does not change the neuronal state. As non-TSE events in the VTSAER do not bear timing information, they are not triggering the membrane decay also. With null event, it is possible to control the streaming speed to avoid system congestion, how it will be explained in the next section 3.4.2.

For the spike train of N sources of length T with X active spikes, the VTS stream has $X + T$ data words of size $log_2(N)$, while parallel stream has fixed NxT bit size. The processing of the VTSAER is simpler as there is no need to scan the spike stream for spikes, every data word forces a change in membrane potential. It is very easy to

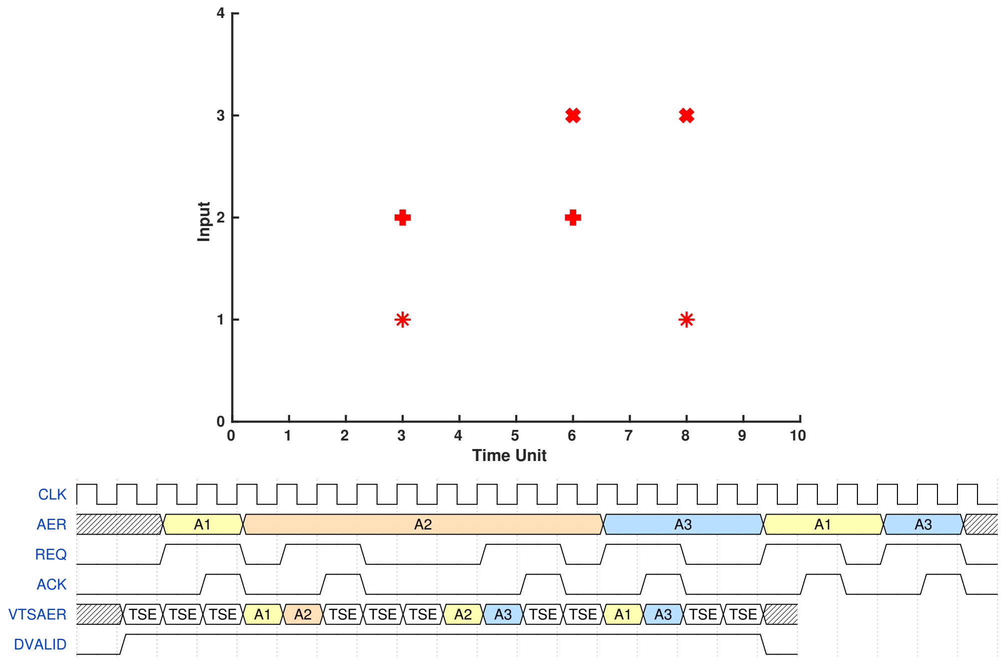


FIGURE 3.11: Sample of a 6-spike raster plot from inputs A1,A2,A3 and its corresponding asynchronous AER and VTSAER streams. Asynchronous AER stream is shown from the perspective of the synchronous receiver.

convert the asynchronous non-timestamped AER system into a VTSAER by adding a timer AER source and merging streams, as shown in Figure 3.12. This schematic will not harm the bandwidth of the data stream if the single event processing time of priority multiplexer will be lower than single event generation time.

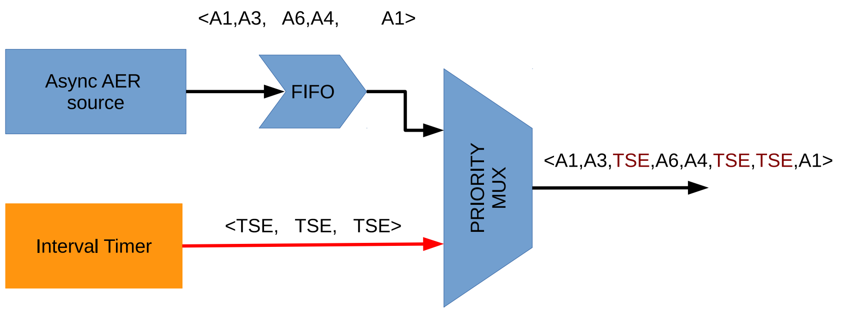


FIGURE 3.12: Schematic of asynchronous AER to VTSAER converter. Priority multiplexer reads the privileged (red) channel first, while the AER source stream is passing through the 1-element FIFO to avoid data loss.

3.4.2 Online event encoding into VTSAER

The conversion of the pre-recorded spike stream into VTSAER is trivial. However, the real-time hardware encoding of the network response into the VTSAER poses a challenge. The output of the network must be encoded in hard real time to avoid data loss. This subsection describes the hardware structure of the encoder, developed for this task.

VTSAER, due to its compressed nature, sets high requirements for the input processing time. When a new meaningful input is guaranteed to arrive every clock period, the processing elements of the network have to process their internal states and communicate the output spikes on the same rate. Of course, the throughput will have some limitations. Consider a layer of N oversensitive neurons, with the refractory period of R. The synaptic weights of the neurons are the same and are high enough to cause a postsynaptic spike on any incoming event. Let the input VTSAER stream consists of a single event every time slot. Then, for M incoming events, the layer will generate $\frac{NM}{R}$ output events. It is obvious, for $N > R$, that the output event stream has to be produced at a higher rate than one event per clock. Of course, such scenario is very unrealistic, as real SNNs are sparsely activated, with one postsynaptic spike occurring after a tens, if not hundreds, presynaptic ones. However, depending on the method of the event collection from the system, these limitations must be considered.
One of the solutions to the described problem is the input flow control. Flow control in the VTSAER can be performed using Null events. If some event has the corresponding weight equal to zero, it is not changing the state of the system, thus, an arbitrary number of null events can be injected into the VTSAER stream. For a complex network, Input-to-Output event ratio cannot be predicted and a dynamic flow control is necessary. If the AER source cannot be paused (live stream), a FIFO of reasonable length has to be added.
The proposed encoding method is shown on Figure 3.13. Output events are passed to the encoder, where they are encoded at the rate of one event per clock cycle. The output VTSAER should have the same timings as the input stream, thus TSEs are passed through the layer in form of virtual "timestep neuron" output. This output has the highest priority and is encoded first. If the coder detects the event overflow, it raises the **PAUSE** signal. Flow control block, after the detection of this signal, stops reading

the input VTSAER FIFO and starts sending Null events to the neurons, giving the encoder necessary time to encode the rest of the active neuron outputs. After all the neuronal outputs are encoded, the **PAUSE** signal goes low, and the input VTSAER is processed.

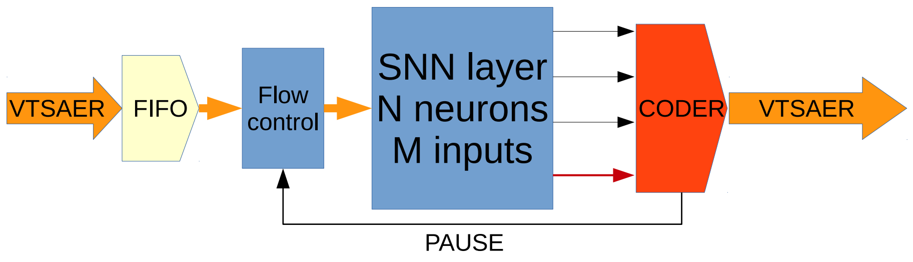


FIGURE 3.13: Block schematic of the VTSAER data flow through the single-layer SNN. Orange arrows show the VTSAER paths, black arrows correspond to binary real-time output. Red arrow corresponds to the ”timestep neuron” output.

The problem of multiple spike handling can be treated as multiple interrupt service problem. The polling approach, when all neuronal outputs are sampled sequentially has fixed and guaranteed time of encoding but is very inefficient. A priority encoder is used often as an interrupt controller, but due to the relative logical complexity, the signal propagation time grows exponentially with the number of interrupt sources. This work proposes a ”passing token” approach than can be described next:

- All neurons are connected in a daisy-chain manner through the line of **AND** gates as shown in Figure 3.14. The neurons also are connected to a shared address bus.
- The encoder is running when the global *READ ENABLE* signal is high.
- When the neuron is not spiking, it is not driving the address bus and the internal AND driving signal *REQOUT* is high, causing the gate to pass the unaltered *REQ* signal.
- If the neuron generates a spike, it waits until the *REQ* input is high. After this, the internal AND driving signal *REQOUT* goes low, preventing all neurons further along the daisy-chain to drive the address bus.The neuron is placing its internally stored neuron address on the address bus.

SPIKEIN	REQ	STATE	STATE(t+1)	REQOUT	OVERDUE
0	0	0	0	1	0
1	X	0	1	0	0
1	1	1	0	1	0
0	0	1	1	0	1

TABLE 3.1: Truth table for the single neuron VTSAER encoder. STATE is the internal variable of the encoder, SPIKEIN and REQ are inputs, REQOUT and OVERDUE are outputs.

- The first block in the daisy-chain is the Time Step Event generator. It places the TSE address on the bus when the TSE in the input data stream is detected.
- If the *REQ* signal is high on the end of the daisy-chain, it means that there are no active neurons on the line. This blocks the writing of the address bus value in the FIFO.
- If the neuron will not encounter *REQ* signal during its refractory period, it raises the *OVERDUE* signal, which is a driver to the global *PAUSE* signal effectively freezing the network state until all active neurons will not be served.

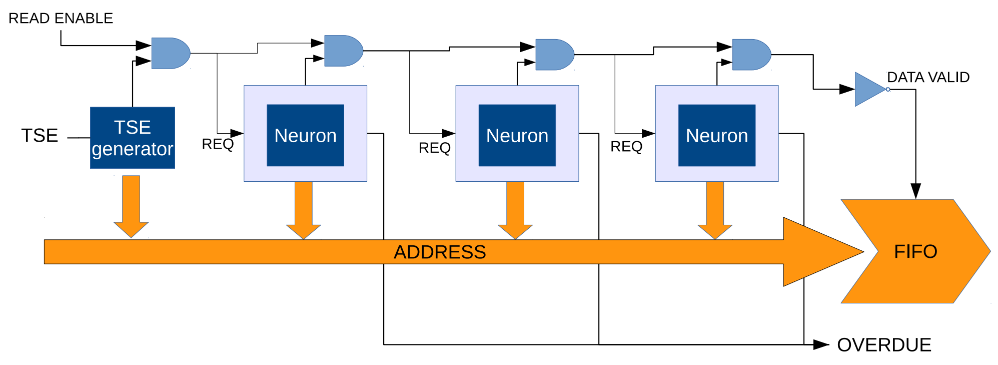


FIGURE 3.14: Hardware architecture of the VTSAER encoder. Spike train, generated by the neuron cores, is converted into the AER stream.

Such architecture pushes the leftmost active neuron number into FIFO at the rate one per clock cycle. TSE generator has the priority and is always served first. This causes some spikes to be placed with the delay of one time window if the TSE occurs when there are unserved spikes, but such error is negligible in practical applications.

The sample timing diagram of its work is shown in Figure 3.15. *SPIKEIN* input is held high during the duration of the refractory period. Table 3.1 describes the logic states

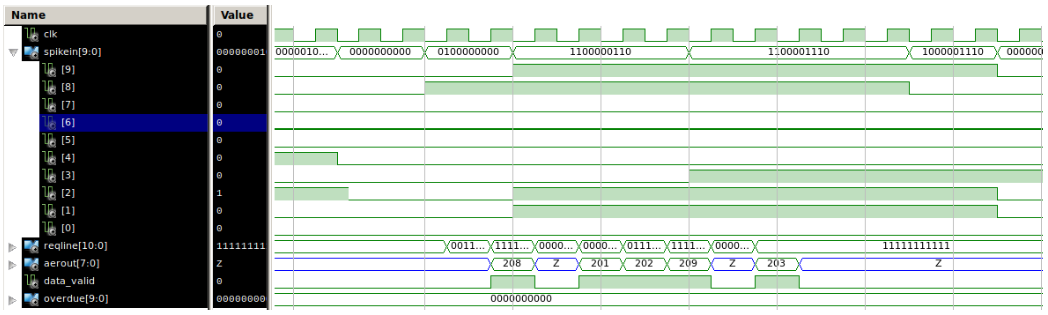


Figure 3.15: VTSAER encoder operation. The neurons 8,9,1,2 and 3 generate spikes that are served by the encoder and neuron addresses (208,201,202,203,209) are put on the bus *AEROUT*. *DATA_VALID* signal is connected to the last *REQOUT* output and is used to push the valid event addresses into the FIFO. Note that the neurons are served in priority order from low numbers to high.

	20 neurons	100 neurons	200 neurons
Registers	81	400	800
LUTs	125	629	1120
Frequency, MHz	741.785	446.07	385.469

Table 3.2: Hardware occupation of the VTSAER encoder without FIFO. The synthesis was made for Virtex 7 xc7vx485t device.

of the encoder. It has one internal memory element *STATE* that is latched high by *SPIKEIN* rising edge and reset to low by the high state of the *REQ* input.

The weak point of this architecture is the long line of AND gates which propagation time limits the maximum clock frequency. When the length of the daisy-chain line is less than 200, the frequency of the encoder is higher than the frequency of the vectored neural core and do not pose a bottleneck in the design. The hardware occupation and the maximum frequency for the encoder are shown in Table 3.2.

To serve large networks efficiently, AER encoders can be cascaded. Several encoders can be connected to one merger. The AER merger block executes the next algorithm:

1. Find the first FIFO with non-TSE address on the output.
2. Read out the addresses from that FIFO until TSE is found.
3. Find the next FIFO with non-TSE address on the output.
4. Repeat steps 2-3 until the last FIFO with non-TSE address is read.

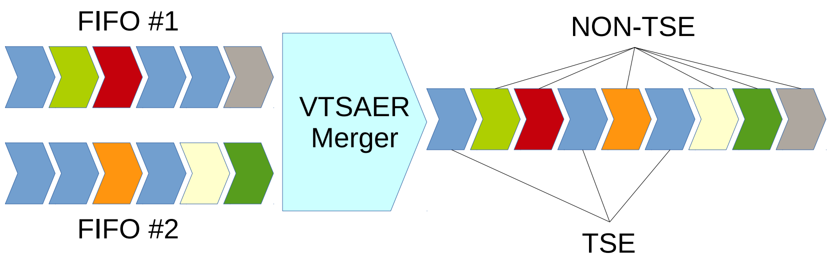


FIGURE 3.16: VTSAER merger operation. TSE events are shown in light blue, non-TSE events shown in other colors. Two streams from FIFOs are merged into one stream.

5. Add TSE to the output VTSAER and start from the beginning.

Graphically this concept is represented in Figure 3.16. As the merger output can be treated as an input to the next merging cascade, a network of an arbitrary number of VTSAER sources can be built and efficiently served. It can be appreciated that the merger can unite VTSAER streams from different sources if their timing references match.

As a summary, VTSAER is a versatile event format suitable for storing and transmitting spike trains from single and multiple sources as well. It can provide any timing resolution necessary, allows true event synchronicity and easy data management. Hardware complexity of VTSAER encoders is low, they can be synthesized in low-cost FPGAs or CPLDs.

Chapter 4

Evaluation of the simplified model

A subset of Semeion handwritten digit dataset [50] was used to test the new algorithms and proof the validity of simplifications. Matlab software was used. The dataset consists of 1593 samples of black and white images of handwritten digits 0-9 (160 samples per digit), 16×16 pixels size as shown in Figure 4.1. The training set consisted of 20 samples for each class (each digit) with 5% of uniform random noise added to every sample fed into the SNN.

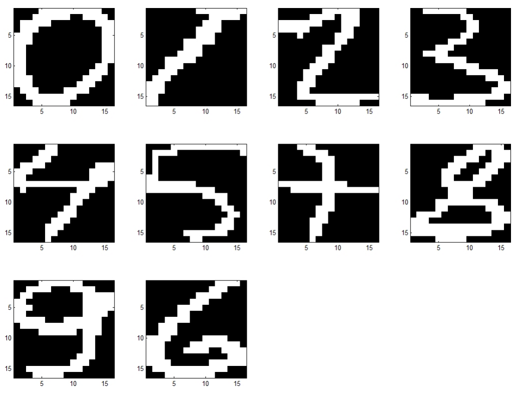


FIGURE 4.1: Patterns for network training of 10 handwritten digits (Semeion dataset).

4.1 Image encoding

In the described experiment, a 5x5 on-centered receptive field was used. This receptive field was weighted in a [-0.5,..,1] range according to Manhattan distance to the center of the field. A 16x16 black and white (binarized) pixel input is processed by a 16x16 encoding neuron layer (256 neurons), obtaining a potential value for each input which will be further converted into spikes. The 2D receptive field coding process, described in section 3.3, using the 5x5 receptive field is shown in Figure 4.2 A and B. The neural response, shown in Figure 4.2C is the membrane potential map, further converted into spike trains whose spiking frequency is proportional to such potential, as shown in Figure 4.2. The same procedure is repeated for all input neurons. This visual data encoding is described in detail in a separate paper [51]. The receptive fields of the neurons are overlapping, an example of three receptive fields is shown in Figure 4.3 where, in case C, a part of the RF lay outside the input space, and thus, that part is not contributing to membrane potential.

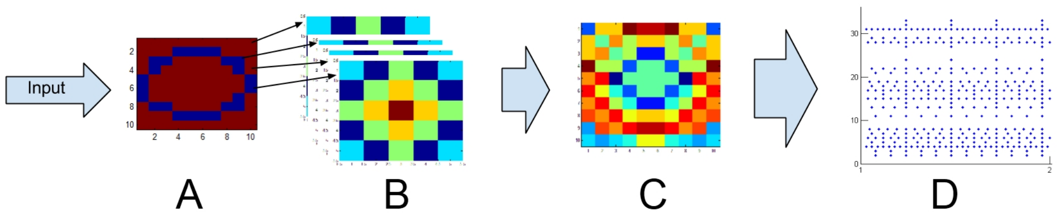


FIGURE 4.2: Image to spike train encoding dataflow. Input image **A** is processed with RFs of encoding neurons **B**, and the result **C** is received by encoding neurons, generating the spike trains **D** where spike frequency is proportional to the intensity of the corresponding pixel and its surroundings.

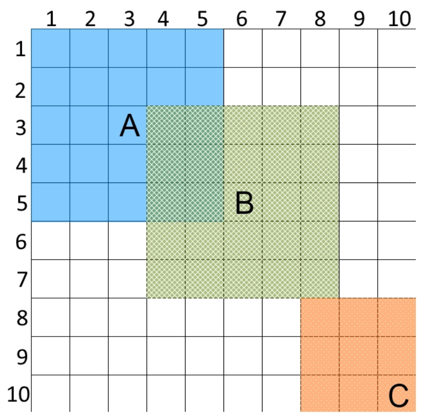


FIGURE 4.3: Three receptive fields in a 10x10 input space. Blue field corresponds to the neuron A (3,3 in input matrix). Green field corresponds to neuron B (6,5) and orange corresponds to neuron C (10,10). Note that only the active part or RF is shown.

4.2 Network architecture

The proposed SNN consists of 2 layers, an encoding layer of 256 neurons with an on-centered 5x5 pixel RF and the second layer of 16 neurons using the simplified LIF neurons proposed in Section 2.3.4. Experimental testing showed that, for proper competitiveness in the network, the number of neurons should be at least 20% greater than the number of classes. For this reason, 16 neurons were implemented; if the number of neurons is insufficient, only the most spike-intensive patterns are learned. Each sample was presented to the network during 200 time units (TUs). With a refractory period of encoding neurons of 30 TUs, the maximum possible amount of spikes is 200/30=6. STDP parameters for learning were $A^{+} = 0.6, A^{-} = 0.3, \tau^{+} = 8, \tau^{-} = 5$. The maximum weight change rate σ was fixed to 0.25 of maximum STDP value, $\sigma = 0.0625$.
Instead of using 'winner-takes-all' strategy, a modification is done by using a 'winner-depresses-all' strategy, where the first spiking neuron gets a weight increase and all other neuron potentials are depressed by 50% of the spike threshold value. Thus, strongly stimulated neurons can fire immediately after the winner, which adds plasticity to the network. The whole network structure is shown in Figure 4.4.

For the classic LIF neuron, a table-based PSP function of 30 points was used while the simplified model uses constant decrease as PSP and does not require table based functions. For both classic and simplified LIF models, STDP function was table-based with 30 positive and 30 negative values. All algorithms (classic and simplified model) were written using atomic operations without the usage of Matlab vector and matrix arithmetic. Such coding style provides more accurate results in performance tests when modeling hardware implementation.

4.3 Results

In order to prove noise robustness, input spike trains were corrupted by randomly inverting the state of 5% from all TU slots. Thus, some spikes were missing and some other random spikes were injected into the spike trains. Five training epochs were run before the evaluation. The implemented network successfully learned all patterns. In Figure 4.5, the membrane potential change during training is shown. Initial values of

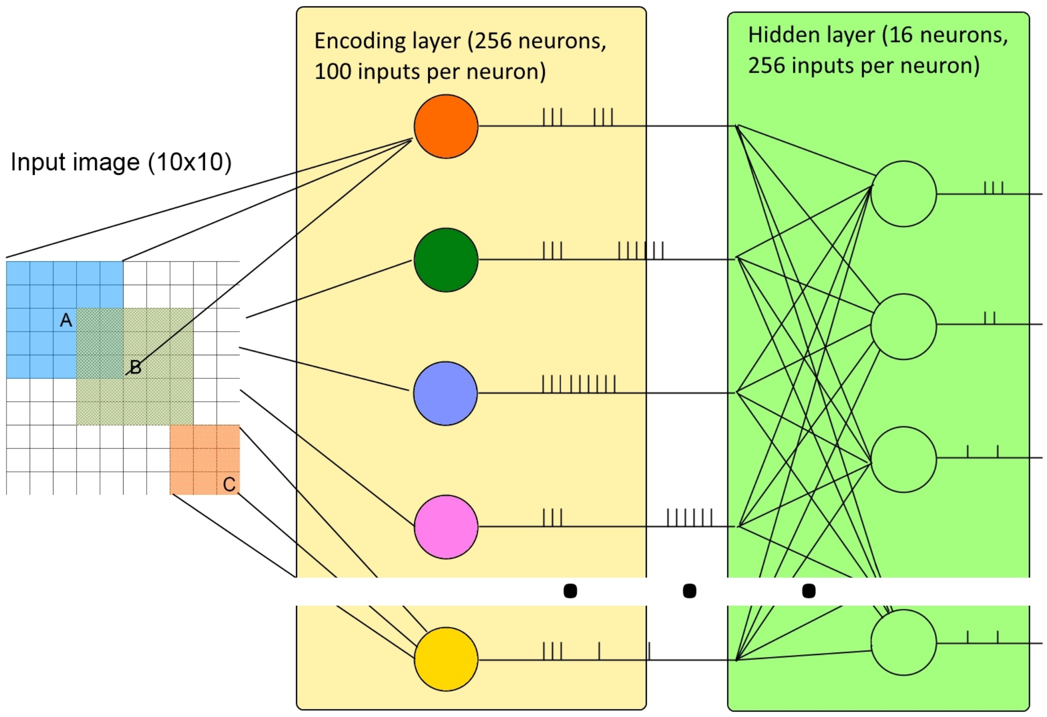

Encoding layer (256 neurons, 100 inputs per neuron)
Hidden layer (16 neurons, 256 inputs per neuron)
Input image (10x10)
A
B
C

FIGURE 4.4: Network structure used in the simulation. Input space of 10x10 is converted into a spike train by a matrix of 10x10 input neurons with the 5x5 receptive field. The generated spike train is fed to the hidden layer of 16 simplified LIF neurons with training. Not all connections between the input space and encoding layer are shown.

the membrane potential are quite small, but with the training, the membrane potential becomes more and more polarized with strong negative values on the classes that are not recognized by the selected neuron. It can also be appreciated that six neurons (numbered 8,10,13,14,15,16) remained almost untrained, with random weights.

Training evolution can be observed by the spike rate diagrams shown in Figure 4.6. Each graph represents one neuron, with classes along the X-axis. Before training, every neuron is firing in several classes and after the training, each neuron has a discriminative high spike rate only in one class. As a result, the final weight maps of neurons become similar to the presented stimuli as Figure 4.7 depicts. The successful separation of patterns 2-5 and 1-6 proves that network can solve problems with partially overlapping classes. The performance of learning between classic LIF and simplified LIF models can be measured with the MSE (Mean Square Error) for normalized weights after training. The training error for a single pattern (class 0) can be seen in Figure 4.8. The graph shows very similarly learning dynamics and performance of both models. Starting from 5000 TU, both models tend to increase the error showing over-training.

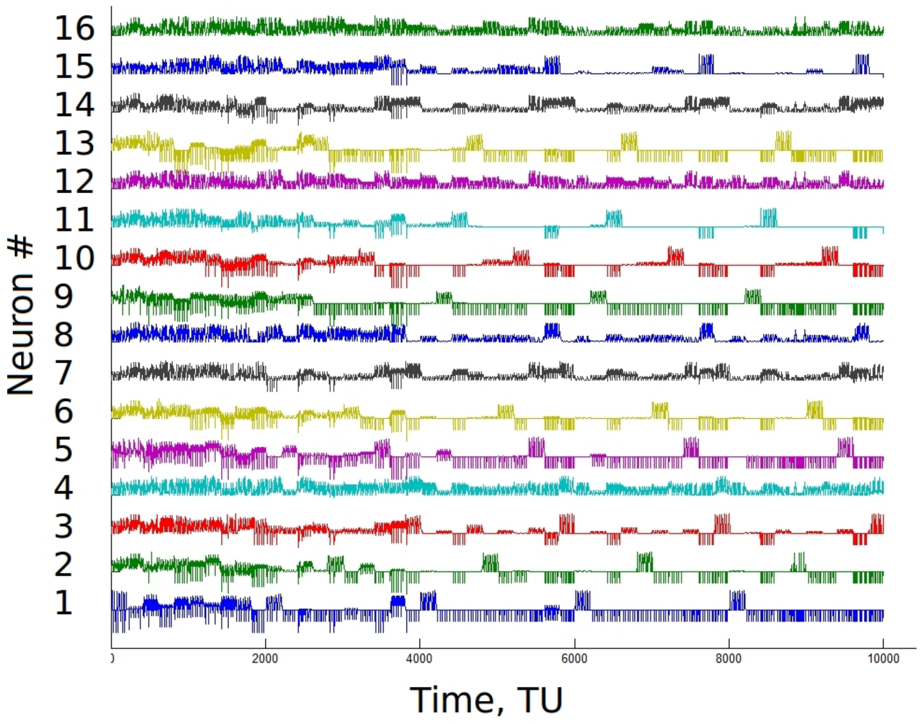


FIGURE 4.5: Membrane potentials of neurons during training. At the beginning, neuronal reactions are chaotic. The training leads to sharp individual neuronal reactions, and neurons become specific to one pattern. The most intensive weight shaping occurs between 3000 and 4000 TUs.

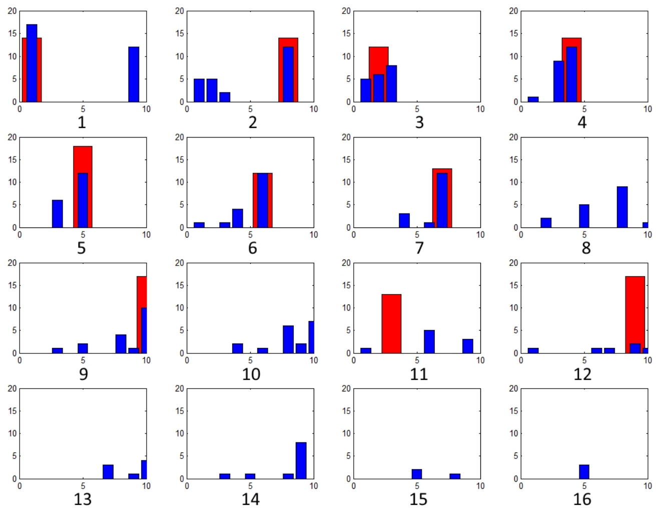


FIGURE 4.6: Spike rate per sample before and after training. Blue bars are spike rate before training and red ones represent the spike rate after the training. Neurons 8,10,13,14,15,16 did not learn any pattern.

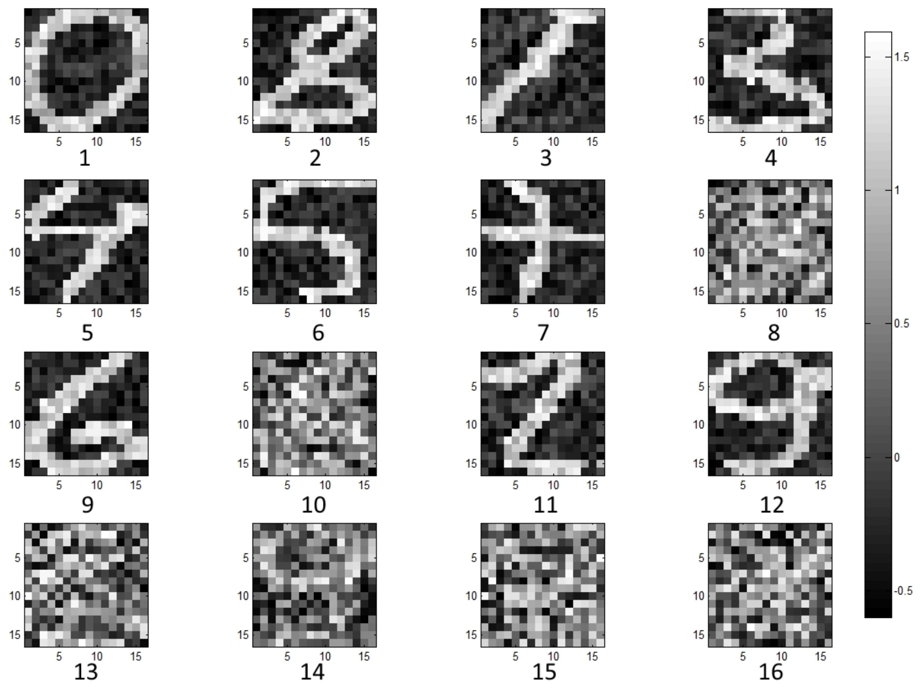


FIGURE 4.7: Neurons weights representation after STDP training. Ten out of sixteen neurons learned to discriminate all ten numbers in the SEMEION dataset.

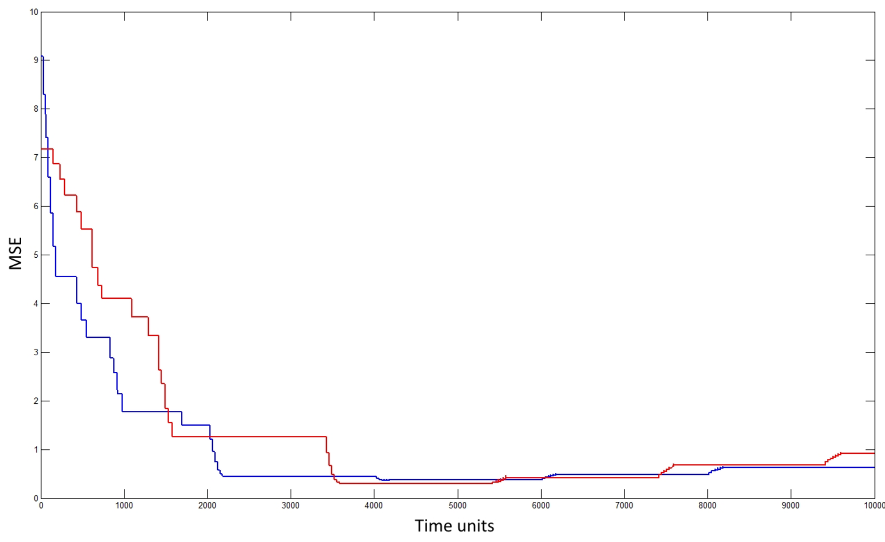


FIGURE 4.8: MSE for single pattern during learning. Red line represents the simplified model, blue represents classic LIF. It can be seen that, after 5000 TU, neuron becomes overtrained for both models and MSE is similar.

To compare the simulation time, three synthetic datasets from Semeion samples with 5, 6 and 12 classes were prepared (12 classes dataset as digits 1 and 0 were represented with 2 classes each). Every class was repeated 30 times to test different network sizes (8, 9, 16, 50 and 100 SNN neuron size in hidden layer were tested). Time of Matlab simulation in Table 4.1 shows an improvement over 20 times when comparing the simplified and classic LIF in these tests. The simulation was done on a 64-bit OS system with 6GB of RAM and an Intel i7-2620M processor.

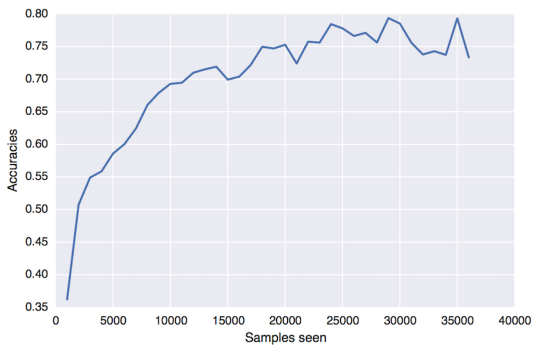
TABLE 4.1: Simulation speed of classic LIF and simplified LIF model networks. All data are obtained on synthetic datasets taking the mean values of 5 runs.

	Classic, sec	Simplified, sec
5 classes, 8 neurons, 15000 time units	48.1	2.25
6 classes, 9 neurons, 15000 time units	48.9	2.4
6 classes, 16 neurons, 15000 time units	66.1	3.9
6 classes, 100 neurons, 15000 time units	137.1	13.03
12 classes, 50 neurons, 96000 time units	1327.12	80.28

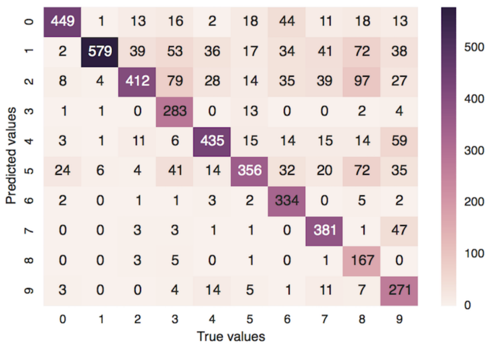
4.4 Evaluation of the RBM based on the simplified model

To demonstrate the feasibility of the evtCD learning, an RBM was created with the simplified LIF neurons. The event-encoded version of the classic MNIST dataset was used, with uniform noise added to the spike train. The spiking RBM had 784 (28×28) spiking inputs. Parameters used for training were:

- Max spikes per sample=200
- Number of neurons = 100
- Training set size = 35000
- Test set size = 5000
- Learning rate = 0.001
- Uniform noise amount=0.03
- Refractory period=0.02
- Threshold=1



(a) MNIST dataset learning curve.



	0	1	2	3	4	5	6	7	8	9
0	449	1	13	16	2	18	44	11	18	13
1	2	579	39	53	36	17	34	41	72	38
2	8	4	412	79	28	14	35	39	97	27
3	1	1	0	283	0	13	0	0	2	4
4	3	1	11	6	435	15	14	15	14	59
5	24	6	4	41	14	356	32	20	72	35
6	2	0	1	1	3	2	334	0	5	2
7	0	0	3	3	1	1	0	381	1	47
8	0	0	3	5	0	1	0	1	167	0
9	3	0	0	4	14	5	1	11	7	271

(b) MNIST dataset confusion matrix.

FIGURE 4.9: evtCD RBM with classic LIF neuron learning curve and confusion matrix.

- Minimal membrane potential =-1
- STDP length=-5..5
- Membrane decay =0.1 (for classic LIF test)

First, a classic LIF neuron RBM was trained with given parameters. After the training, the network had an accuracy of 80%, very close to the original implementation [32]. Unlike the original implementation, the maximal accuracy was achieved earlier, roughly after half of the original dataset. This number was used as a baseline reference for tests. The learning curve and confusion matrix are shown in Figure 4.9. The examples of the trained weights are in Figure 4.10. It is clearly seen that the weights of the spiking RBM are resembling the weights of the classic one and represent the selected features of the presented digits.

The implementation tested is very robust and converges in a similar manner with variable network parameters values. The main tested parameters were the learning rate (Figure 4.11), STDP length (Figure 4.12) and membrane decay constant (Figure 4.13). The network with long STDP length (stdp_ lag=0.04) has worse performance, while membrane decay rate has no influence on the resulting accuracy. This proves that the obtained error is typical for the given network size and encoding method.

Changing the neuron model from LIF to simplified LIF, however, resulted in reduced accuracy during the training. Learning curves for different linear membrane decay values are shown in Figure 4.14. Nevertheless, the network is learning the presented samples. The best classification rate obtained with linear decay is around 71.5%. However, the

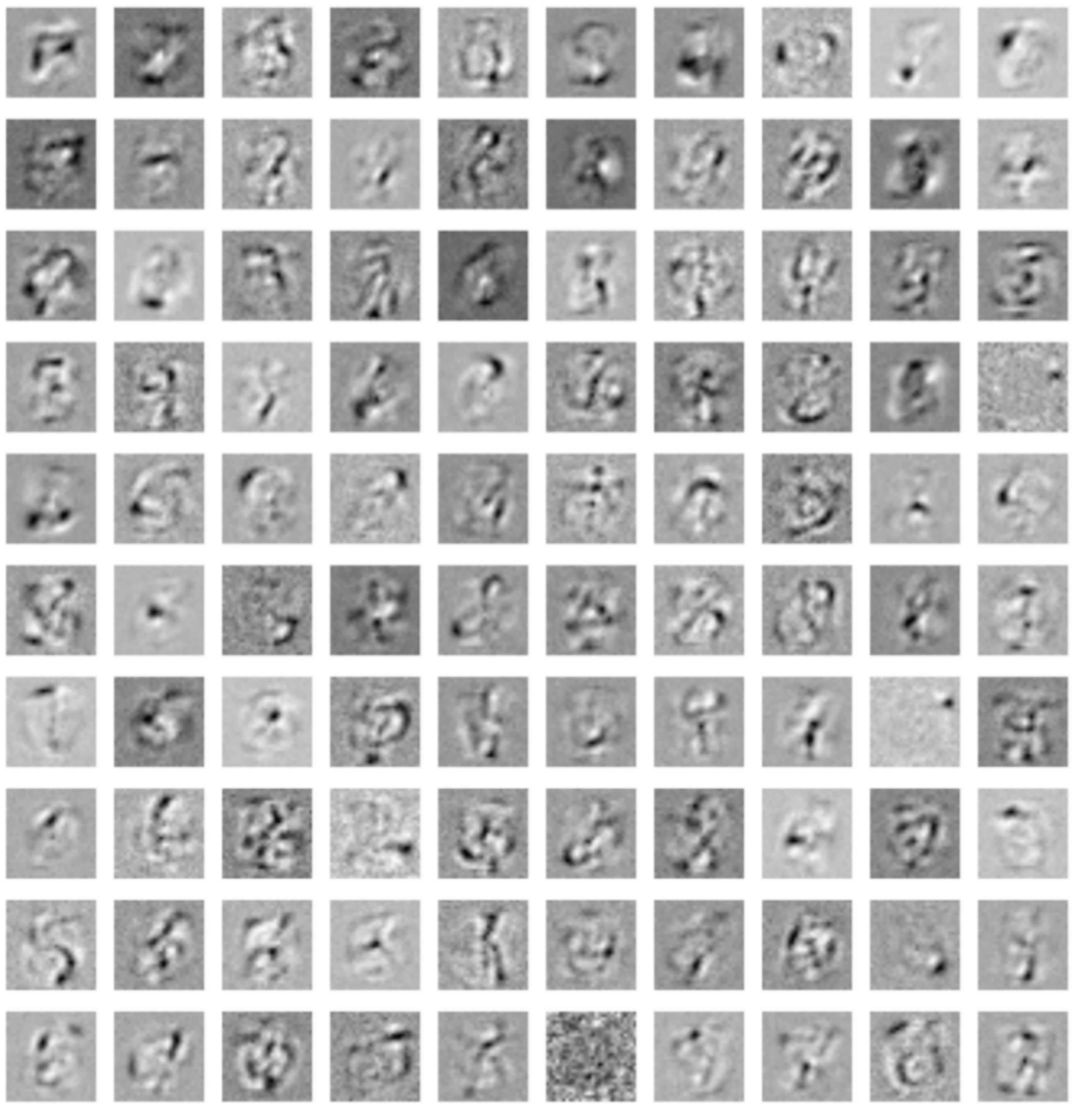


Figure 4.10: Final learned weights for the evtCD RBM with LIF neuron. The weights are similar to weights learned by a classic RBM or autoencoder.

network trained with the classic LIF and evaluated with simplified LIF actually shows better results than the baseline implementation. The learning curve for such combination is shown in Figure 4.15, reaching almost 85% accuracy on the training set. The effect was observed for various other network parameters.

As shown, the proposed simplified LIF model can be successfully used in SNNs with STDP training. In the perceptron test it performed similar to the classic LIF and in the RBM test, it showed up to 71.5% accuracy on MNIST dataset in contrast to classic LIF which obtained 80.1%. After the training, classic LIF neuron can be replaced with the

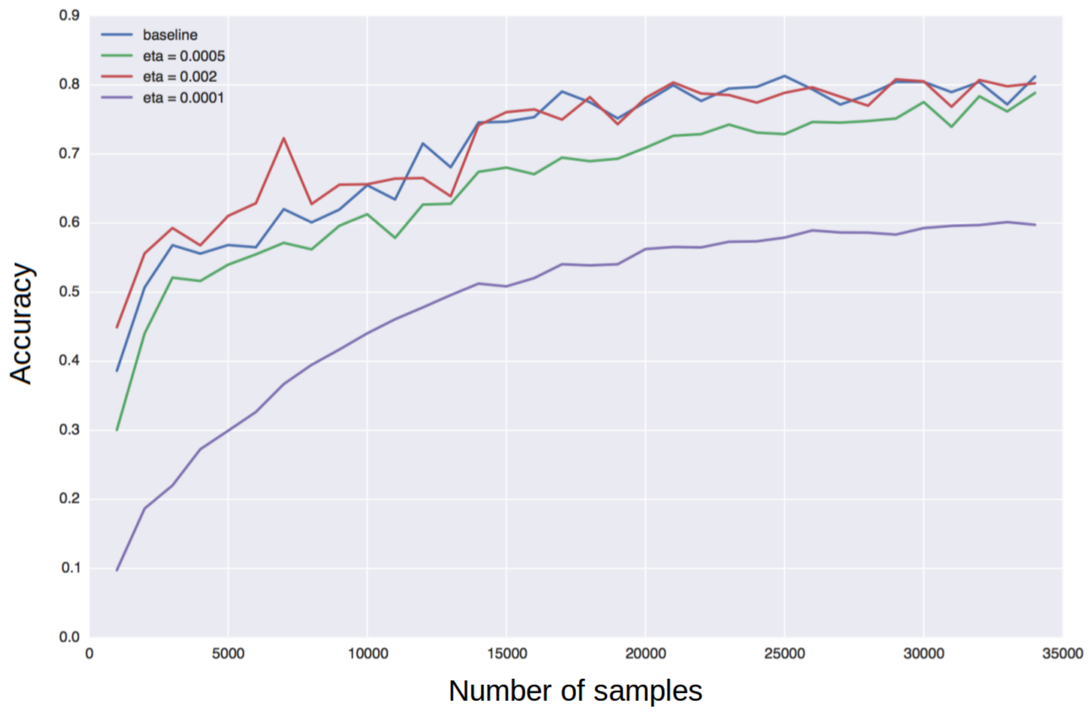


FIGURE 4.11: evtCD RBM with LIF neuron learning curve for different learning rates.

simplified one without loss of accuracy.The network preserves its learning and classification properties while computational and memory complexity is reduced by eliminating the PSP table in each neuron. Learning is stable and robust, the trained network can recognize noisy patterns. A simple, yet effective visual input encoding was implemented for this network. The simplification is beneficial for reconfigurable hardware systems, keeping generality and accuracy.

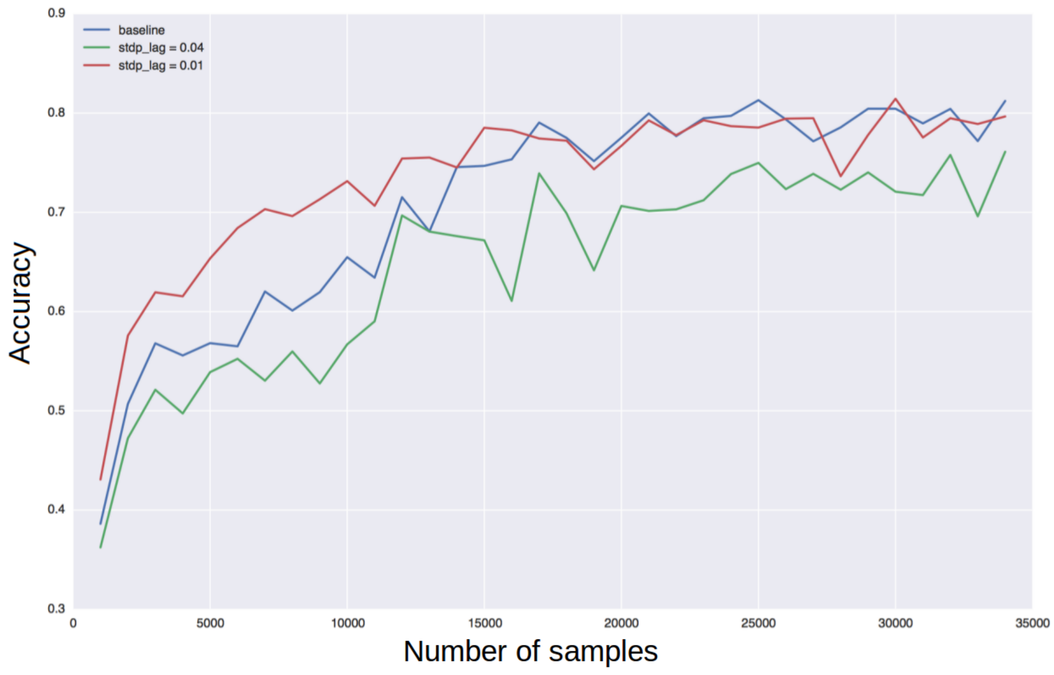


FIGURE 4.12: evtCD RBM with LIF neuron learning curve for different STDP length.

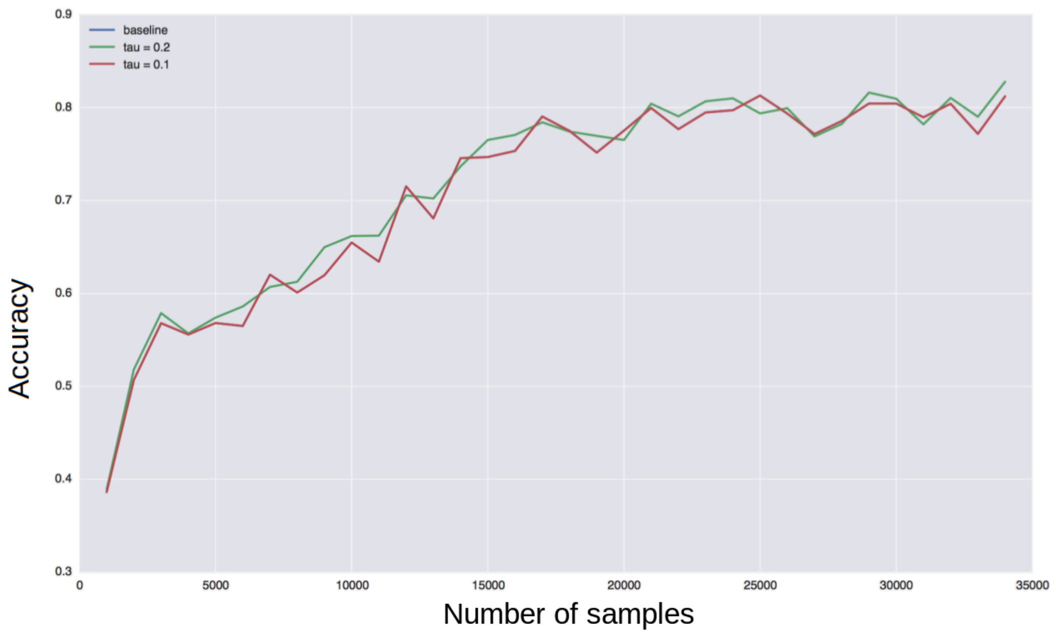


FIGURE 4.13: evtCD RBM with LIF neuron learning curve for different membrane decay constant.

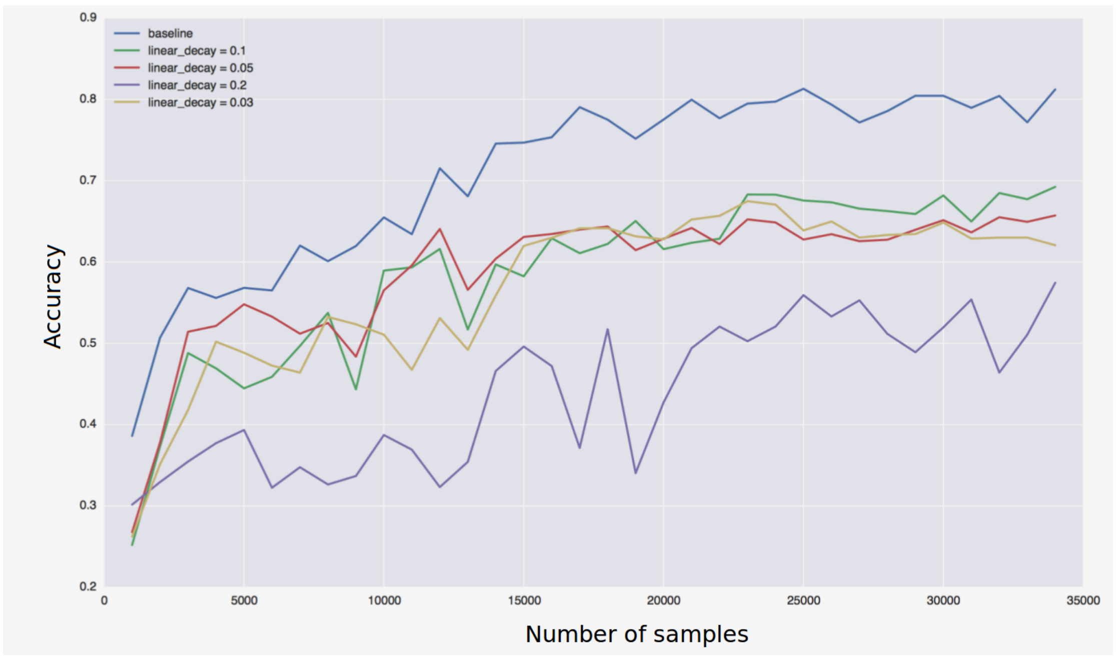


FIGURE 4.14: evtCD RBM with simplified LIF neuron learning curve for different membrane decay constant.

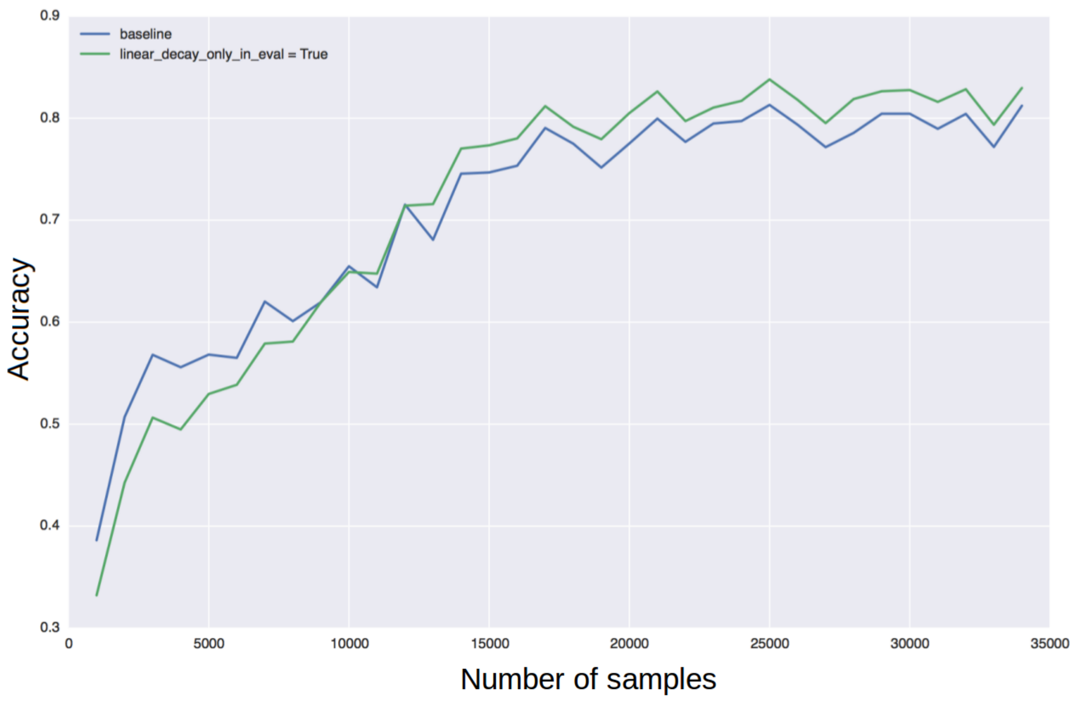


FIGURE 4.15: Learning curve for evtCD RBM trained with classic LIF and evaluated with the simplified model.

Chapter 5

Spiking neuron as a finite automaton and its hardware implementation

This section describes the spiking neuron architecture, suitable for the classic and simplified LIF models described before and optimized for hardware massive parallel implementation. The primary considered platform for this topology is FPGA. Internal structure and capabilities of modern FPGA families defined the algorithmic implementation, thus it is important to discuss the hardware first.

5.1 FPGA in scope of the neuromorphic hardware

Field-Programmable Gate Arrays have evolved from compact ”glue logic” discrete ICs replacements into the top performance computation accelerators during the last 30 years. The recent rise of Machine Learning applications has also influenced the reconfigurable logic market. Having massive parallel processing capabilities, highly developed interconnect and distributed memory, FPGAs provide a viable platform for neuronal computations. Spiking neural networks were successfully implemented on FPGA numerous times. Some of the most important implementations are worth to be mentioned here. In 2003, an obstacle avoidance robot, guided by FPGA-implemented 64 neuron SNN was presented [38]. The neurons utilized a LIF model with truncated weights. Cassidy et al. [52] presented in 2007 a LIF-based implementation of a 32-neuron network with STDP learning, however, the learning wasn't tested on real datasets. A 2009 work by Thomas and Luk [53] was using floating-point Izhikevich model and was able to place up to 1024 neurons per single FPGA. Another work, presented by Rice et al. [54] from the same year mentions over 9000 Izhikevich neurons on a single chip, however, it uses weight storage on external RAM. Two remarkable works were made with real-life tasks. Zamarreno-Ramos et al. [55] is describing an event-based AER-driven mesh network-on-chip of convolutional modules and Daniel Neil [56] presents a Minitaur, an event-driven RBM implementation on low-cost FPGA board. However, these implementations lack learning capabilities and are limited by the memory bandwidth.

The implementation of neural networks in FPGA can be divided into *fine-grained* and *course-grained*. Fine-grained implementations have every element of the network implemented separately, directly mapping the topology into the FPGA fabric. The biggest problem of such approach is the number of connections, growing exponentially with the network size. The coarse-grained implementations use more complex processing nodes, and every node is computing separate parts of the network. This makes the topology of the system simpler, but the internal data processing in the nodes is more complex since different serializations in computations must be done.

Two main resources that are crucial for neural network implementation are the memory and the arithmetic-logic units (ALU), that can be synthesized inside the FPGA. The following description is relevant for the FPGAs of Xilinx brand. Intel FPGAs (former Altera) have comparable resources. In fact, proposed architectures can be easily adapted to different manufacturers or even implemented as ASIC.

5.1.1 Block RAM

To store the synaptic weights inside FPGA, a RAM has to be synthesized. As the memory primitive is widely used in hardware design, the modern FPGAs family have the significant amount of distributed memory blocks included, also called ”Hard IPs”, as they are physically present in the chip and are not synthesized from common logic blocks (CLB). Xilinx memory primitives are named Block RAM and are true dual-port memories of 1024 18-bit words. The ”true dual-port” feature means that the memory has two pairs of independent read and write ports with internal order of operation arbitration. They can be configured either as two independent 1K×9-bit or one 1K×18-bit memory. Each port can be used with a separate clock so that the memory can be used as a dual-clock FIFO, what can be utilized for the external weight update, for example. The Virtex 7 series has up to 3600 separate BRAMs or 68 megabits of internal distributed memory per chip.

5.1.2 DSP48 block

Another widely used primitive included in the FPGA is the sophisticated arithmetic-logic unit. Due to its popularity in DSP, arithmetic hard IPs in the FPGAs are commonly referred as DSP blocks. Xilinx 7th generation DSP48E1 block has the following properties [57]:

- 25×18 two's complement multiplier.
- 48-bit accumulator
- Pre-adder.
- Pattern detector for rounding.
- Dedicated cascading C 48-bit port for cascading the DSPs
- Depending on the device family and speed grade the block can work on frequency up to 550 MHz fully pipelined.

The simplified block schematic of DSP48E1 is shown in Figure 5.1.

The schematic explains the possible and forbidden operations with this DSP block. Input A can be multiplied by B, but not by C or D. Input C, dedicated to connecting the output of the adjacent DSP can be used only for the same operations as the accumulator. The ALU block, used in the SNN implementations described in this work, requires the following operations.

- P=0. Clearing the accumulator.
- P=P+A. Basic MAC operation.
- P=A+B. Weight update operation.

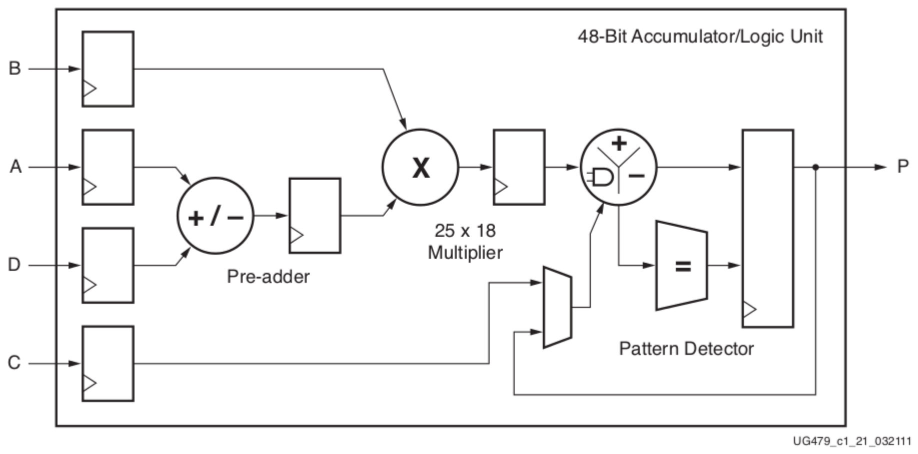


FIGURE 5.1: DSP48E1 block schematic.Source: [57].

- P=C+A. Cascaded MAC to compute RBMs.

All the mentioned operations are supported by the DSP48E1 block natively and it can be efficiently used in complex designs.

5.1.3 Fixed-point arithmetic and accuracy in FPGA

The performance of the fixed-point arithmetic is several times higher than floating-point [58] and the hardware occupation is significantly lower. This makes fixed-point arithmetic a common choice for the algorithms implemented in FPGA. However, the choice of the proper arithmetic size and data format is very important for the performance of the system. On the other side, in floating-point representation, the accuracy is a trade-off for numeric range, while fixed-point arithmetic has the same accuracy across the whole data range. The typical fixed point format to represent fractional data is called Q-arithmetic, or Q-notation. The numbers are stored in QX.Y format, where X is a decimal part and Y is a fractional part. The number of bits for X part limits the data range to $(0..2^X)$ for unsigned and $(-2^{X-1}..2^{X-1})$ for signed numbers. The Y part size defines the numerical accuracy as $\frac{1}{2^Y}$. For an 8-bit fractional part, the accuracy will be ≈ 0.004. When considering fixed-point, inevitable loss of accuracy exists in arithmetic operations. In case of multiplication $A_{X.Y} * B_{X.Y} = C_{X+Y.Y}$, the result has to be rescaled (shifted) by Y bits to the right. Thus, the maximum error of each operation is $\pm 2^{-Y}$. When performing MAC on N arguments, the result is a sum of N multiplications, and,

a maximum single element error is $\pm N * 2^{-Y}$. The error is exponentially reduced with the increase in the size of the fractional part and grows with the size of the matrix and the number of chained matrix multiplications.
Xilinx DSP blocks are designed with a 48-bit accumulator and carry input. Then, there is no reason to limit the MAC operation precision below this number if the DSP block will be used in the design. Another key parameter is the number of bits per weight. As the memory in Xilinx FPGAs is organized in 36-bit columns, divisors of 36 are preferred for data size, especially 18 and 9 bits, natively supported by manufacturers IP-core. Intel FPGAs have 9-bit standard BRAM width and endorse using similar data types.

5.2 VTSAER as a basis for neuron architecture

Simplified spiking model and Variable Timeslot AER described earlier can be used for the dramatic improvement of large SNN architectures. Combined with the FPGA efficient design guidelines, the new spiking architecture can be described by the following terms:

- The inputs are processed serially.
- All network neurons in the layer share the same input.
- The state of the neuron is only dependent on its neuronal input and its state in the previous time unit.

Thus, the elementary spiking neuron can be treated as a finite automaton. Using VTSAER, control, timing, and event words can be processed simultaneously in all neurons in the layer. After the initial weight loading, the neuron internal state is driven only by its input.
The block diagram of the basic algorithm for a neuron with linear membrane potential decay is shown in Figure 5.2.

The abbreviations used in the schematic are:

- REF: Refractory period flag.
- TSE: TimeStep Event.

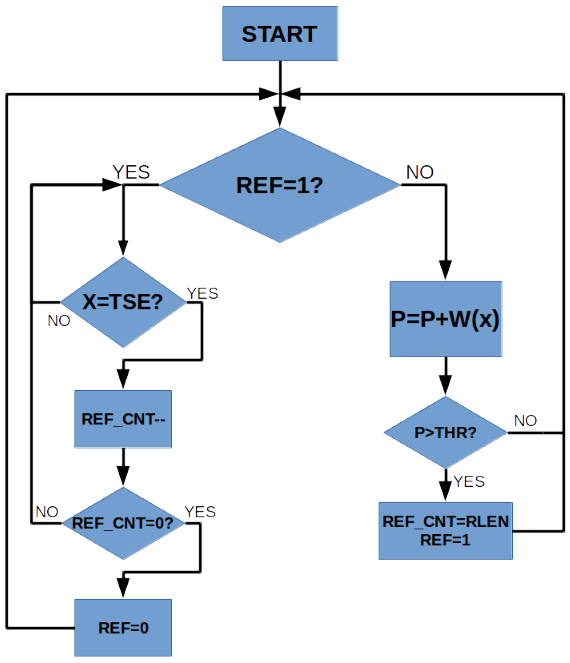


FIGURE 5.2: Simple neuron block schematic.

- P: Membrane potential.
- X: current AER input value.
- REF_CNT: refractory period counter of RLEN maximum length.
- THR: membrane threshold.

The input of the neuron is a VTSAER stream. Every event has the corresponding weight in the neuron weight memory. When the input event arrives, the adder accumulates the input spikes' weights in the accumulator P until the threshold THR is reached. The threshold can be either static, or modified by an external signal (threshold plasticity), or be tied to a random number generator (stochastic spike generation). Note that, when not in the refractory period, the timestep event is treated as a normal event. The weight corresponding to the TSE address is the membrane degradation rate. This neuron does not require a multiplier, free-running counters, PSP function generators and other elements present in other implementations [52, 59].
By assigning multiple TSE addresses, an arbitrary PSP function can be implemented. The presented algorithm realizes only linear decay function; By assigning a part of

weight memory as a PSP lookup table, one can construct more complex piecewise linear membrane potential degradation functions at the cost of increased BRAM memory occupation.

5.2.1 Binary shift-based LIF and multiplication-based LIF neurons

The proposed approach can be used to construct neurons with different nonlinear PSP. Classic LIF function is usually realizing the function described by Equation 5.1, where dP and dt are small values usually in thr range [0.001..0.1]. The resulting multiplicand $e^{\frac{dt}{dP}}$ lays within [0.9..0.99] range. This equation can be rewritten in form $P_{i+1} = P_i - P_i * (1 - e^{-\frac{dt}{dP}})$. It is possible to choose the multiplicand $(1 - e^{-\frac{dt}{dP}})$ equal to some power of 2, and the multiplication by the power of 2 can be replaced by binary shift. This leads to Equation 5.2, where D is a small number, in range of [8..64]. For example, in the RBM tested in Section 4.4, the chosen decay parameters were: $dt = 0.1, dP = 0.005$, what gives the decay $P_{i+1} = P_i * 0.9512$. The multiplierless equation 5.2 for $D = 16$ gives the decay value $P_{i+1} = P_i * 0.9375$. As it was shown in Figure 4.13, small weight decay changes do not influence the accuracy and learning rate.

$$P_{i+1} = P_i * e^{-\frac{dt}{dP}} \tag{5.1}$$

$$P_{i+1} = P_i - P_i >> (|D|) \tag{5.2}$$

By adding one additional conditional operator and one subtractor it is possible to construct a nonlinear membrane potential decay. The updated block schematic of such neuron is shown in Figure 5.3. The algorithm remains almost the same as for the linear decay neuron, the only change is that the TSE has to be detected not only in the refractory period but during normal membrane potential computation, too. TSE will alter the ALU opcode, from P=P+A to P=P+B operation.

If a sufficient number of DSP blocks is available, a classic LIF function can be implemented with the multiplier added to the neuron. The multiplier, necessary for the

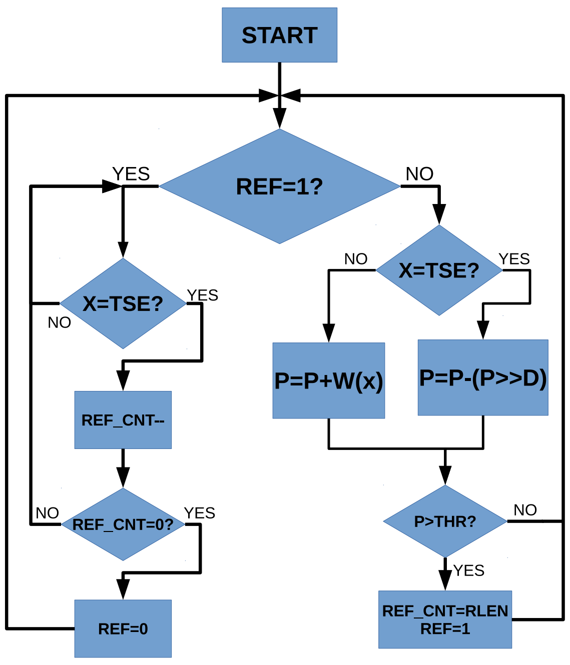


FIGURE 5.3: Neuron with shift-based decay block schematic. D is a constant value controlling the membrane potential decay 2^{-D}.

calculation of the Equation 5.1 is replacing the shift block in the neuron schematic. More detailed analysis of the multiplier-based implementation is in section 5.4.

5.3 STDP on-chip learning for the Automata neuron

True on-chip STDP learning is rarely used as it requires significant hardware resources and no satisfactory implementations of STDP for large networks exist. Using VTSAER, it is possible to add synaptic plasticity with little hardware overhead.

The most important problem of STDP is the backward causality: when a spike occurs, the weights of multiple previous inputs have to be facilitated. As the neuron cannot predict the moment of the spike, it can not update the weights *before* the spike. It is relatively easy to depress the weights corresponding to spikes arriving during the refractory period, but for the positive half of the STDP function, an additional input event memory is required.

As all presynaptic spikes are the same for all neurons in a layer, this memory can be shared for all neurons. The best solution proposed in this work is to use a second VTSAER stream, delayed by **RLEN** time units. It will be a variable length FIFO of a maximum size of $N * RLEN$ words, where N is the number of inputs. After a postsynaptic (neuron output) spike, during the refractory period, STDP block handles both facilitation and depression parts, adjusting the weights accordingly. There is no need to manage the AER flow speed or play the stream backward every time the spike occurs. However, handling two streams simultaneously can be costly in hardware. The memory will require true a dual port operation, and the weight update calculation will use two adders. This work proposes the use of interleaved VTSAER stream. In the interleaved stream, the timeslots in the VTSAER are mixed between normal and delayed, as shown in Figure 5.4. The modulation signal **DTM** defines a normal (0) or delayed (1) timeslot.

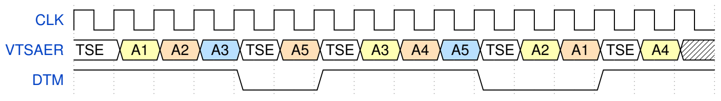
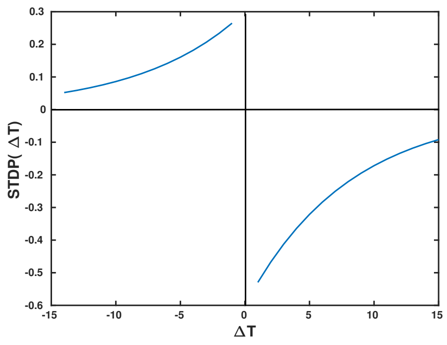


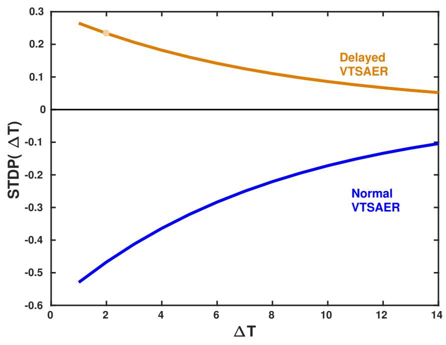
FIGURE 5.4: Interleaved VTSAER stream.

This solution requires to half the data rate because every event will be processed in two clocks instead of one. However, due to the elimination of the need to store the event history, this implementation is more convenient in terms of resources.
Having interleaved VTSAER (IVA), the STDP algorithm becomes easy to implement. Classic and modified one-sided STDP functions are shown in Figure 5.5. Figure A shows typical asymmetric STDP function (depression is stronger than facilitation). This function facilitates presynaptic spikes (negative ΔT) and depresses the postsynaptic spikes (positive ΔT). Part B shows the modified STDP function, that is operating only on postsynaptic spikes, facilitating the spikes arriving through the delayed VTSAER stream.

The block diagram of the new STDP algorithm is shown in Figure 5.6. **STDP** is a LUT-implemented STDP function, **REF** is the refractory period signal, active when the neuron is in the refractory period. Note that the whole STDP weight update process is using only three conditional operators. If the STDP is symmetrical, it halves the size of LUT for this function. Also, there are successful implementations of a single-value ramp STDP, when the weights are changed by the fixed value regardless of the ΔT [31, 32]. Such STDP implementation reduces the neuronal complexity even more.



(a) An asymmetric STDP function for normal spike train.



(b) Converted asymmetric STDP function for interleaved VTSAER.

FIGURE 5.5: The conversion of STDP function for IVA , with only positive values of Δt.

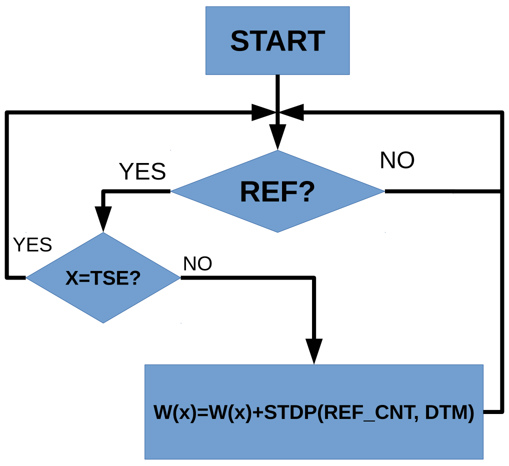


FIGURE 5.6: Block schematic of STDP. Note that STDP is a function of counter *REF_CNT* and delayed/normal stream flag *DTM*.

5.4 FPGA implementation of the Automata neuron

The schematic of the linear decay neuron implemented is shown in Figure 5.7. Some signal delay blocks necessary for data synchronicity are omitted.

Shown signals are:

- VTSAER - main AER data stream input. The width of this signal is $log_2(N)$, where N is the number of inputs.
- DTM (Delayed Timeslot Marker) - high level of the signal corresponds to the delayed timeslot. Allows for interleaved VTSAER data input.

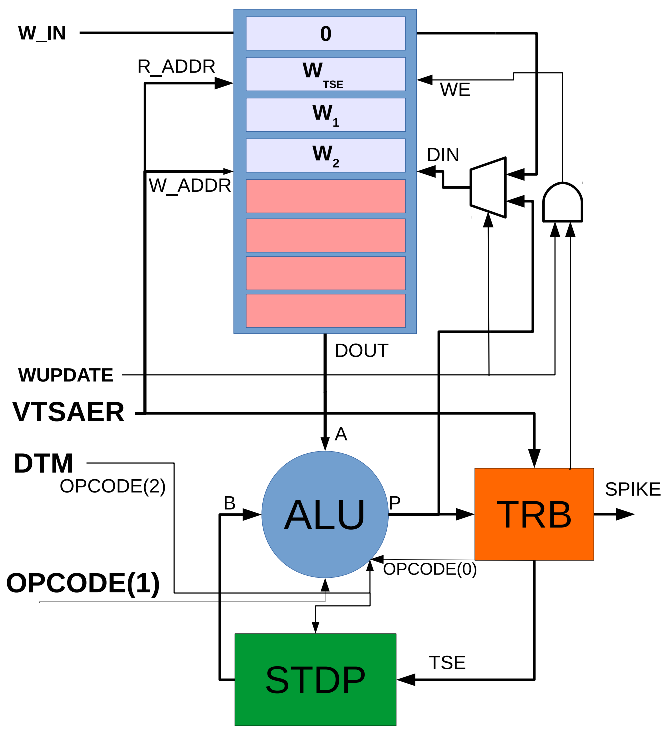


FIGURE 5.7: FPGA implementation of linear decay neuron. The STDP block and DTM (Delayed Timeslot Marker) input are not used in a version without learning. The untitled top block is the weight memory (containing 0 as null event weight and TSE membrane leakage weight). The saturated arithmetic ALU is in the center.

- WE - Write Enable. Used for weight initialization and online weight update.
- R_ADDR - address for reading from the weight memory.
- W_IN - data for weight initialization.
- W_ADDR - address for writing into the weight memory.
- DIN - weight data input.
- DOUT - weight data output. The weights are coming into port A of ALU.
- BSEL - bank select input, used for batch learning. With bank switching, it is possible to update the weights in one bank, while still using the unchanged weights from the second bank for the inference.
- WUPDATE - weight external update signal, serves as WE for data from W_IN. Used for weight initialization.
- SPIKE - spiking binary output.

- OPCODE - the type of operation in ALU. Three-bit signal, composed by external OPCODE, internal OPCODE from TRB and DSM signal.
- TSE - timestep event signal for STDP block that is switching the STDP internal counter. Not used when STDP is constant.

As it is shown in the figure, the STDP block on the bottom and DTM signal are not used if the learning is not required. Corresponding hardware can be excluded from the neuron then.
The untitled block on the top is the weight memory, implemented as a dual-port BRAM. ALU performs the operation of Add and Add-Accumulate.Basic hardware implementation of the neuron does not require a DSP48 block, the ALU is synthesized from the CLBs. Weights are stored in BRAM configured in a true dual-port mode, where the initial weight setup is made through the port A and online weight update is done through the port B. ALU in this neuron uses saturated arithmetic, bounding the membrane potential P within certain range. This prevents potential numerical overflow problems.

The schematic of the shift-based nonlinear decay neuron described earlier in Section 5.2.1, is shown in Figure 5.8. Some signal delay blocks necessary for data synchronicity are omitted. As it was mentioned earlier, this neuron requires one binary shift and one additional comparison. Some possible decay rates for this neuron are: 0.125 ($\frac{1}{8}$), 0.0625 ($\frac{1}{16}$),0.0312 ($\frac{1}{32}$), 0.0156 ($\frac{1}{64}$). The block on the bottom of the schematic is constantly producing the shifted value $P >> |D|$. TSE, detected by TRB, modifies the ALU opcode to perform $P = P - B$ operation instead of $P = P + A$. The membrane potential is decayed according to the Equation 5.2.

This automata neuron can realize the multiplication-based LIF function described by Equation 5.1 as well. For this, the binary shifter block is replaced by a multiplier with normalizer. It reduces the accuracy of the representation according to the formula for fixed-point arithmetic, shown in Equation 5.3, where Y is the fractional part width. This neuron bears significantly higher hardware complexity, as it requires a multiplier now. In addition, the maximum working frequency of this circuit will be lower, because of the comparison, multiplication, and assignment which have to be done in one clock cycle.

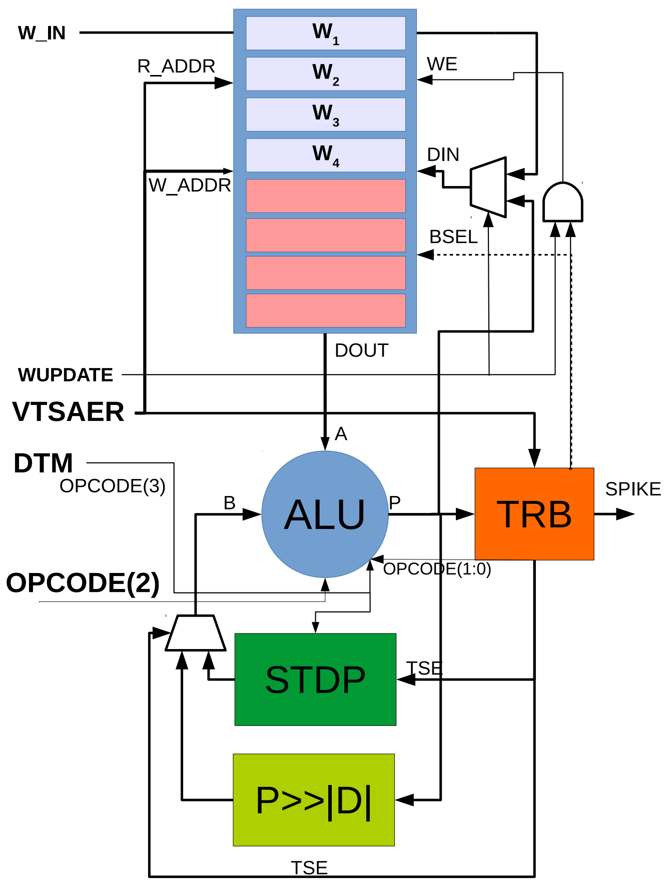


FIGURE 5.8: FPGA implementation of shift-based nonlinear decay neuron. The TRB is detecting TSEs, switching the multiplexer on ALU port B and changes the ALU opcode from $P = P + A$ to $P = P - B$ operation. Block on the bottom is a binary shifter.

$$P_{i+1} = floor((P_i * round(e^{\frac{dt}{dP}} * 2^Y)) >> Y) \tag{5.3}$$

Multiplication-based decay neuron can be optimized at the cost of slight reduction of accuracy. Equation 5.1 can be rewritten into a multiplication-subtraction form as in Equation 5.4. This equation is still containing two references to P_i, but a subtrahend is now a small number. If we replace $P_i*(1-e^{\frac{dt}{dP}})$ by $P_{i-1}*(1-e^{\frac{dt}{dP}})$, the difference will be less than $(1-e^{\frac{dt}{dP}}))\%$ of the current P value, but the subtrahend can be computed in the previous clock cycle. The resulting formula for the fixed-point arithmetic is described in Equation 5.5. The *floor* function in Equation 5.3 and 5.5 is rounding down to the nearest integer what is equal to truncation in fixed-point arithmetic. Other fixed-point rounding strategies exist [60], but they are not included in this research due to the

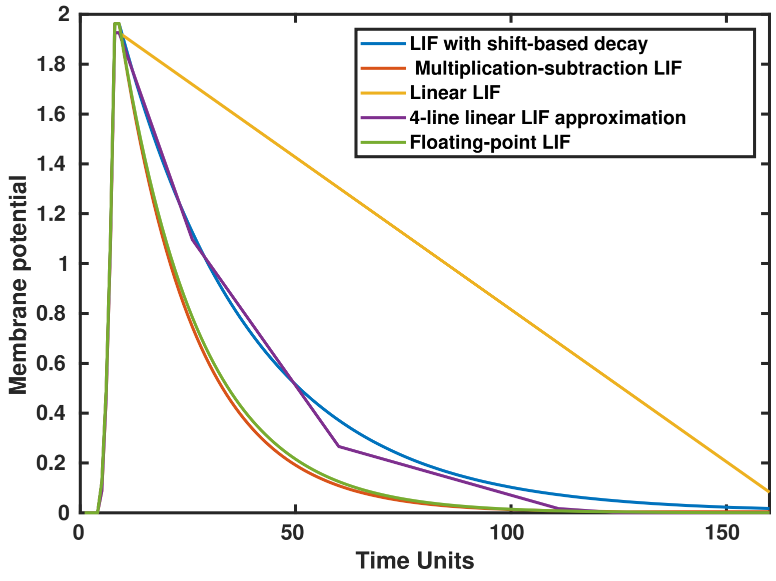


FIGURE 5.9: Comparison of decay curves for various membrane potential decay algorithms. The reference curve is the green one, computed with 64-bit floating point. All other algorithms were using Q6.12 fixed-point arithmetic.

negligible effect for this application.

$$P_{i+1} = P_i - (P_i * (1 - e^{\frac{dt}{dP}})) \tag{5.4}$$

$$P_{i+1} = P_i - floor((P_{i-1} * round(1 - e^{\frac{dt}{dP}} * 2^Y)) >> Y) \tag{5.5}$$

The evaluation of the presented membrane decay algorithms is presented in Figure 5.9. The simulation was made for the neurons with membrane threshold of 3. The weights were slightly different for shift-based decay neuron, and for this neuron there is a small initial discrepancy. After several events that raised the membrane potential to ≈ 1.97, the membrane potential decay process starts. The neuron used Q6.12 fixed-point arithmetic, thus the minimal quantization error is $\frac{1}{4096}$. Four neurons were tested: single line linear-decay neuron, 4-line linear decay, shift-based decay ($D = 5$, equal to $\frac{P_i}{32}$ decay) and the optimized multiplication-subtraction LIF. The floating-point simulated curve is shown in green, the decay value was 5.25%. The 4-line approximation was done for the $\frac{P_i}{32}$ and the graph shows that besides the linear decay neuron, all other curves are close to each other.

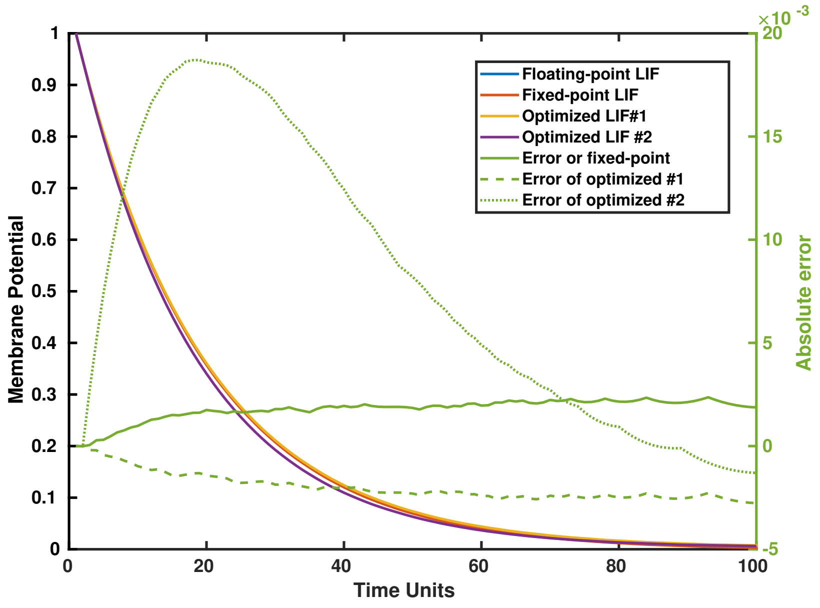


Figure 5.10: Comparison of decay curves and absolute error for subtraction-multiplication based decay algorithms.

A more detailed evaluation of the subtract-multiply algorithm is presented in Figure 5.10. This graph is comparing the decay of multiplication-only (5.3) (orange line, marked as **Fixed-point LIF**), instant subtraction-multiplication (5.4) (yellow line, marked as **Optimized LIF#1**) and the two-step subtraction-multiplication $P_{i+1} = P_i - (P_{i-1} * e^{\frac{dt}{dP}}$ (violet line, marked as **Optimized LIF#2**) algorithms. The absolute error of these algorithms compared to the floating-point implementation is shown in green solid, fine-dotted and dashed lines with the scale on the right side of the graph. All algorithms were using Q6.12 arithmetic. All implementations show almost overlapping decay curves, providing the same behavior. The error of fixed-point implementation (solid green line) is stable and matches the quantization error, while error of the Optimized LIF#1 (dashed line) is slowly increasing due to the truncation of small numbers. The error of Optimized LIF#2 is high initially but after 60 TUs decreases to even less than the Optimized LIF#1 error as the decay values decrease.

5.4.1 Neuron functional blocks description.

Beside ALU and BRAM, an automata neuron contains two complex functional blocks: TRB (Threshold/Refractory Block) and the STDP block. They are realized with CLB logic and do not require any vendor-specific IPs or components.

The TRB (Threshold/Refractory Block), shown in detail in Fig5.11, contains a comparator to compare ALU output with the hardcoded threshold. After detecting the membrane potential higher than programmed threshold, the comparator generates an output spike and enables the refractory counter *REF_CNT*. In addition, TRB alters the *OPCODE* to switch the ALU from Adder-accumulator into Adder mode, necessary for the STDP. Every incoming TSE is detected by pattern detector and the output is driving the Refractory Counter *REF_CNT*. WE signal for weight update can be driven high either by the external WUPDATE or internal signal from the TRB (STDP weight update). The *REF_CNT* is downcounting from *REF_PERIOD* constant value to 0. While the *REF_CNT* is counting (counter value greater than zero), the signal *ZERO* is low. *WE* output is a negated sum (NAND) of the output of *REF_CNT* and TSE detector. It is done to avoid updating the membrane decay constant.
The STDP block structure varies depending on the type of STDP used. For the constant value STDP, it is one-value register and the DTM is used as a sign bit. No TSE signal is used. For the symmetric STDP, TSE is driving the counter with the halved output (because two TSE will occur in the IVA for one present timeslot), the counter output is driving the ROM address, where STDP values are held and the DTM is used as a sign bit. The most complex asymmetric STDP block is shown in Figure 5.12.

The weight memory and weight update process are the most complex algorithmically and the simplest in hardware. Two weight update scenarios are possible. In the first case, the weights are initialized from the external source and are written through **W_IN** port with **VTSAER** as address line when **WUPDATE** is high. The second case is

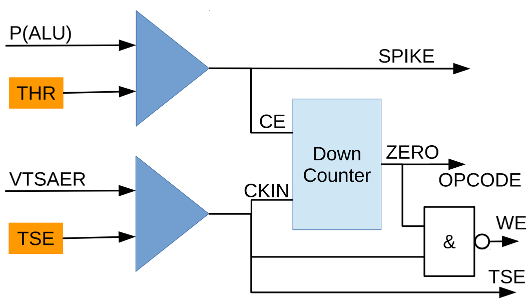


FIGURE 5.11: Threshold and Refractory Block schematic (TRB). It consists of a comparator, pattern detector, a counter and a 2NAND gate to generate a WE signal for the STDP weight update.

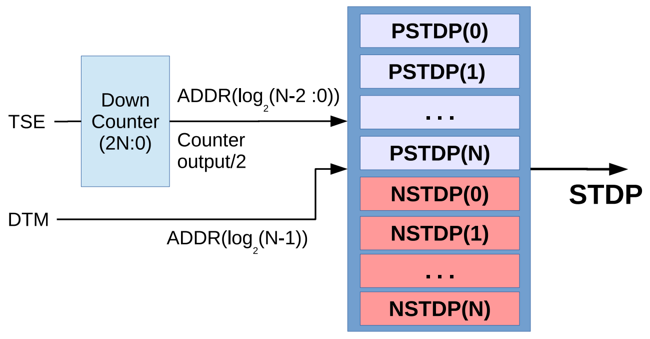


FIGURE 5.12: STDP generation block structure. STDP values are held in ROM, with positive values in the upper half and the negative in the lower half of the memory. ROM address is composed with the halved output of the TSE counter and DTM as the most significant address bit. With such configuration, ROM can be treated as two-bank memory, where DSM is the bank select bit.

the STDP operation: while Xilinx true dual-port BRAM allows reading and writing the same address in one clock cycle with configurable policies (Write after Read or Read after Write) [61], in practice, sometimes it is more favorable to separate the source and destination. In automata neuron, the separation between weight read, modification and write is made by a pipelined architecture. BRAM has one cycle input-to-output delay. ALU performs the calculation in two clock cycles. It means that the updated weight will be stored only on the 4th cycle. This means that the **W_ADDR** memory input signal must be delayed for three clock cycles to synchronize with the data. Delays are not shown in Figure 5.7 for the simplicity of the schematic.

During the refractory period, TRB drives **WE** high, and subsequent **VTSAER** input reads the corresponding weight from the memory and passes it to the ALU. The updated weight is stored by the same address after four clock cycles.

It is also possible to implement batch learning with this architecture. Multiple studies show that batch learning when the update is done once per certain number of samples, gives better results than constant weight update. For batch learning, the weight memory is two times bigger than the number of weights so that weights occupy half of the RAM. Two halves will be referred as memory banks (upper and lower), and **VTSAER** is used as address inside one bank, i.e., **VTSAER** range is N for 2N memory. During the first batch, the weights are read from the upper bank. In the refractory period, WE signal from the TRB also drives the BSEL port (signal shown with the dotted line on the schematic). It modifies the most significant bit of the **VTSAER**, causing the

Decay type	Linear	Shift-based	Multiplication-based	Optimized sub-mul
Slice Registers	85	93	83	83
Slice LUTs	97	97	117	115
BlockRAM	1	1	1	1
DSP48E	0	0	1	1
Frequency, MHz	387.372	387.372	287.993	365.724

Table 5.1: Hardware occupation of single automata neuron.

readdressing of read and write operations to a second bank. As the second bank contains zeroes after the reset, during the update it will contain only weight changes results or ΔW. In the end of the batch, the second bank will be filled with the sums of weight updates. The update of the main weights can be made after the batch by computing the sum of the weights from the lower and upper bank with storing the result in the upper bank.

ALU used in the multiplierless designs performs three operations: Clear ($P = 0$), Add-Accumulate ($P = P + A$) and Add ($P = A + B$). Add-accumulate is used in membrane potential calculation and Add is used in the STDP weight update computation. Both operations use saturated arithmetic to prevent overflow. This is more safe practice in fixed-width integer computations. To increase the speed, both operations are made pipelined with pipeline length of two. External **OPCODE** signal is used for resetting the neuron and for future functionality expansion.

The hardware occupation of the implemented neurons is shown in Table 5.1. The synthesis was made in Xilinx ISE 14.7 for Virtex 7 xc7vx485t-3 device with default synthesis rules, no DSP48E block usage. Weight width is 18 bits, the number of weights is 1024. The synthesis was made without a policy of keeping block hierarchy, what lead to better logic optimization. In the synthesized RTL, counters in the TRB and STDP blocks are merged into one and some secondary logic functions are also optimized.

The comparison shows that optimized subtraction-multiplication algorithm can be synthesized with almost 100 MHz higher frequency for the same device, and the hardware complexity is lower than the multiplication-based LIF implementation. Shift-based and linear decay neurons have the lowest hardware occupation.

The choice between multiplierless and multiplier-using models should be done with regard to the intended hardware resources. Multiplierless automata neurons have such low logic hardware occupation that the number of neurons in FPGA is limited only by the amount of the block memory available in the chip. The accuracy of this architecture is limited only by the arithmetical accuracy of the chosen data type bit width. Membrane potential computation, threshold detection, and weight plasticity do not differ from the software realization within its numerical accuracy limits.

5.4.2 Simulation of Automata neuron functioning

Multiple tests were conducted to test the proposed architecture. The simulation was made with the Q6.12 numeric format. The waveforms shown in Figures 5.13 and 5.14 describe the weight update process in hardware.

Figure 5.13 shows in detail the "synthetic" single-sided (only potentiation) weight update: one short series of events (AER 1-10) is driving the neuron until fired after the event #6. Neuron comes into the refractory period (**REF=1**) and weight update starts. The next three events: 7,8,9 are treated as the presynaptic events and every corresponding weight (**RAM[7,8,9,10]**) is potentiated by 100.

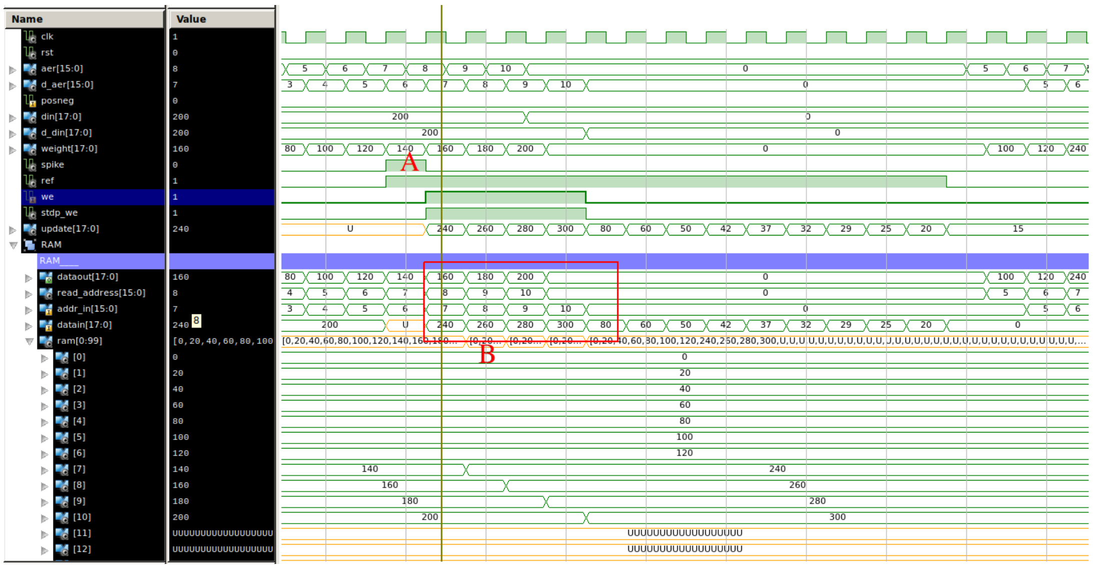


FIGURE 5.13: The simulation of single STDP series. The spike, marked as red A, causes the potentiation of the weights #7,8,9,10. The addresses and values are highlighted by a red rectangle, marked as B.

Figure 5.14 shows a more realistic scenario of multiple interleaved time frames, processed during the refractory period. Events #1 to #4 are arriving in current, or direct time frame and are depressed, while events #6 to #10 are arriving in the delayed time frame and are potentiated. Event #5 is present in both time frames and is sequentially depressed and potentiated in the corresponding time frame. TSE with the address #0 (null events are not used in this simulation) changes the value of the STDP function (from 29 to 25) what can be traced by comparing the updates of the weight #1. After several hundreds of clock cycles, the neuron becomes responsive only to the inputs #6 to #10, expressing the desired weight plasticity.

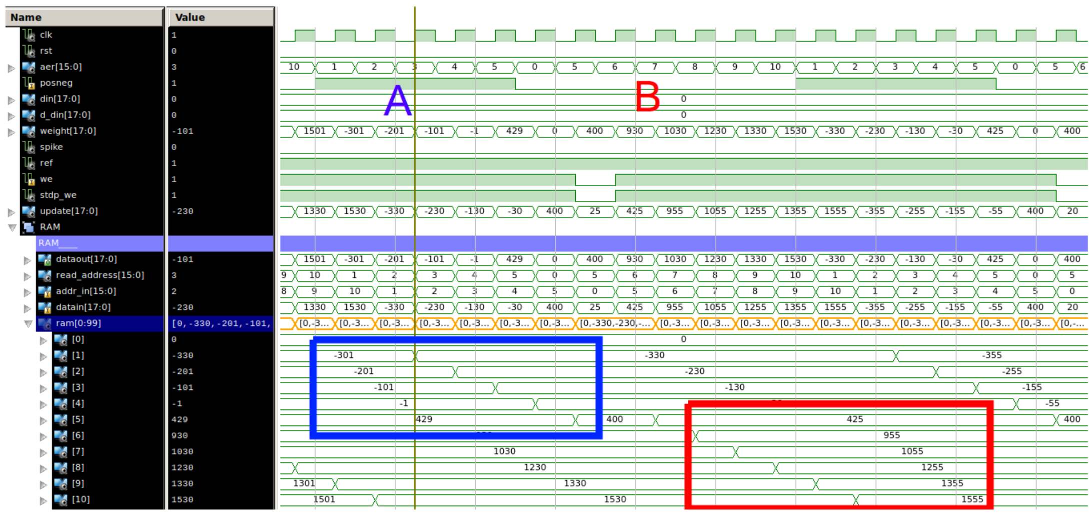


FIGURE 5.14: Simulation of multiple STDP series. Memory values are at the bottom. In the depression phase, marked with blue A (**posneg** signal is high), the weights #1 to 5 are depressed (blue rectangle). In the potentiation phase, marked with red B, the weights #5 to 10 are potentiated (red rectangle). Note that the weight #5 is both depressed and potentiated during one refractory period.

5.5 Composing a layer of Automata Neurons. Multi-layered networks

The proposed neuron structure can be easily connected in parallel to form a layer of arbitrary width. All elements in the layer will share most of the signals, having only **WUPDATE** input separately driven, necessary for the individual initial weight loading, and the **SPIKE** output signals will also be independent. As the number of connections in the layer is a linear function of the number of elements, the hardware complexity per element will not be high even in very large layers, and the working frequency will remain almost unchanged. It is easy to notice that the number of hardware components is changing linearly with the size, with very little logic overhead spent in connections. Moreover, multiple layers of spiking neurons can be computed reusing the same hardware. Consider the weight memory with multiple memory banks, one for each layer.

Membrane decay type	Linear			Shift-based		
Number of neurons	100	500	1000	100	500	1000
Slice Registers	4442	43000	83000	5739	48000	96000
Slice LUTs	7226	49000	96000	7225	48500	97000
BlockRAM	50	250	500	50	250	500
DSP48E	0	0	0	0	0	0
Frequency,MHz	387.372	387.372	387.372	387.372	387.372	387.372

TABLE 5.2: Hardware occupation of layers of different size.

Such architecture can compute the neuronal states in an arbitrary number of layers. The proposed multilayered network architecture is shown in Figure 5.15. All neurons are connected to the same signal buses, only **WUPDATE** inputs and **SPIKE** outputs are individual signals. The proposed architecture has excellent scaling possibilities, what is proven by the synthesis results for networks of different size, shown in Tables 5.2, 5.3. The tables contain the synthesis results for the four types of automata neurons. Also, a detailed synthesis report presented in Table 5.4 proves that the synthesis results match the proposed topology. The data in this table are taken from RTL synthesis, not actual FPGA placement that can alter the component usage due to technology limitations.

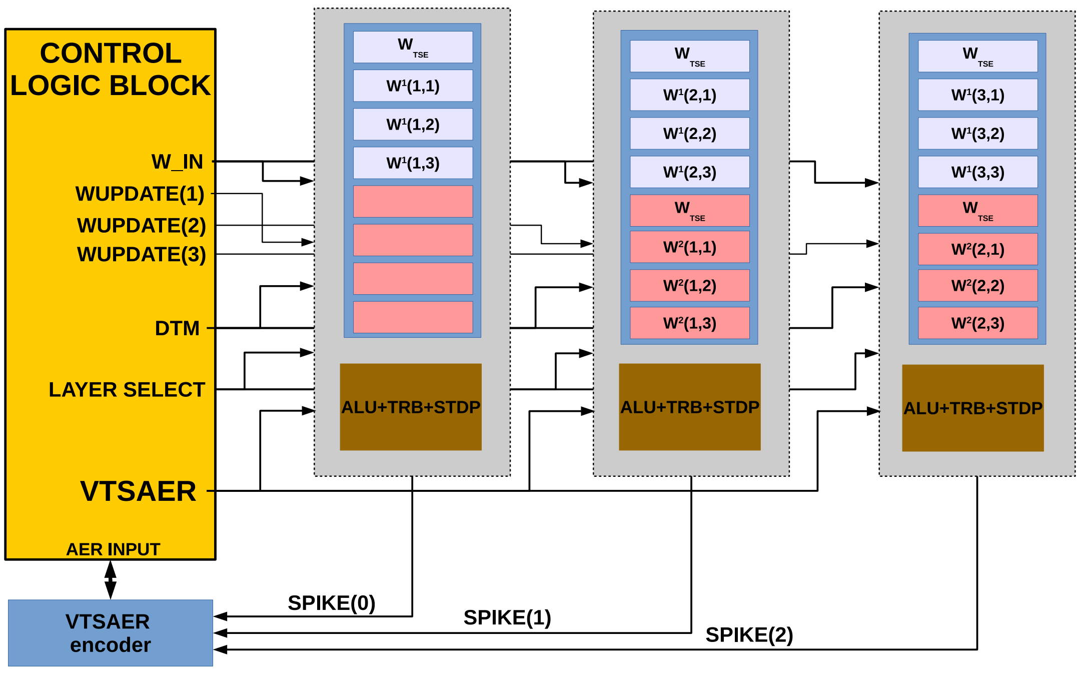


FIGURE 5.15: 3-3-2 network schematic. Hidden layer weights are stored in the upper memory bank, and output layer weights are stored in the lower memory bank. All signals beside SPIKE and WUPDATE are common for all neurons.

Membrane decay type	Multiplier-based			Optimized sub-mul		
Number of neurons	100	500	1000	100	500	1000
Slice Registers	4439	41500	83000	4439	41500	83000
Slice LUTs	9424	49000	116000	9324	57500	96000
BlockRAM	50	250	500	50	250	500
DSP48E	100	500	1000	100	500	1000
Frequency,MHz	287.994	287.994	287.994	365.070	364.511	364.511

Table 5.3: Hardware occupation of layers of different size.

Layer size	1000	1
1000x18-bit dual-port block RAM	1000	1
18-bit addder/subtractor	1000	1
4-bit subtractor	1000	1
4-bit down counter	1000	1
18-bit up accumulator	1000	1
Registers	80000	80
Flip-Flops	80000	80
Comparators	2000	2
18-bit comparator greater	1000	1
4-bit comparator greater	1000	1
Multiplexers	1000	1
18-bit 2-to-1 multiplexer	1000	1

Table 5.4: Hardware components of 1000-neuron layer and a single neuron. Neurons implemented are linear decay simplified LIF. The numbers are from the RTL logical synthesis, not from the actual FPGA placement. FPGA Place and Route numbers are in tables 5.2 and 5.3.

Summarizing, the proposed hardware-friendly Automata neuron can successfully implement various LIF spiking neuron models with STDP plasticity. The architecture is versatile and can host a wide range of PSP and STDP functions. An array of Automata neurons will have size linearly dependent on the number of neurons, as there are no fully connected elements that lead to the exponential growth of the hardware occupation. This means that this architecture is not affected by the connectionist problem, which has been the important issue over the years.

5.6 The limitations of the fixed-point models

While fixed-point arithmetic can reach arbitrary precision in the given numeric range, there are practical limitations for the implementations. In the machine learning field, it is more practical to work with dimensionless units, thus, parameters like time, conductance, current, voltage, and capacitance have to be proportionally converted. Choosing

W	Weight range	[-1..1],[-10..10]
ΔW	minimal weight update	[0.005, 0.03]
dt	Time step	[0.005..0.05]
$1 - e^{\frac{dt}{dP}}$	Membrane decay	[0.01..0.1]
Thr	Threshold	[1..10]

Table 5.5: LIF neuron parameters of the selected implementations.

proper numeric format is crucial for the maximum efficient memory utilization.Also, knowing the limits of numerical resolution and range is necessary to tune the learning algorithm for the network. To study such limitations, a set of network parameters used in practice was chosen.

One of the limitations is the minimum weight change. The Automata neurons were built with the goal of maximum simplicity and performance in the hardware, so the weights are stored and computed in the same numeric format. In general, 9 or 18-bit size is a preferable data width for optimal BRAM utilization. The second limitation is the minimal time unit dt. If the chosen TU is too small, the membrane potential decay $\frac{dt}{dP}$ can be smaller than the selected numeric format allows. If the chosen TU is too large, the STDP becomes nondiscriminative and the possible refractory period values are limited. The parameter ranges listed in Table 5.5 were taken from the different practical SNN implementations [31, 32, 56, 62, 62, 63].

Consider a 9-bit Q2.7 format for example. Numeric range of Q2.7 is [-1.9922..1.9922], and resolution is 0.0078. The values of minimal weight update and time step can be lower than the resolution, and some networks may have the weights out of the numeric range. The 18-bit Q6.12 format has a numeric range of [- 63.9998.. 63.9998] and a resolution of 2.4414e-04, what is satisfying all the encountered conditions. However, there are studies showing that after the learning, the weight resolution can be significantly reduced without compromising the accuracy [31]. Therefore, Q6.12 can be recommended as a basic format for spiking networks with learning and the Q2.7 can be used as a weight-optimized data format for pretrained network implementations.

5.7 Complete neuromorphic data processing system design with the vectored architecture

The described multilayer SNN core, shown in Figure 5.15, is a self-sufficient spike processing unit. For the users' convenience, it is reasonable to add a microprocessor for data initialization and processing management, as well as common bus interface to enhance the compatibility with the existing peripherals. The possible configurations of the proposed neuromorphic data processing systems are shown in Figures 5.16 and 5.17. In

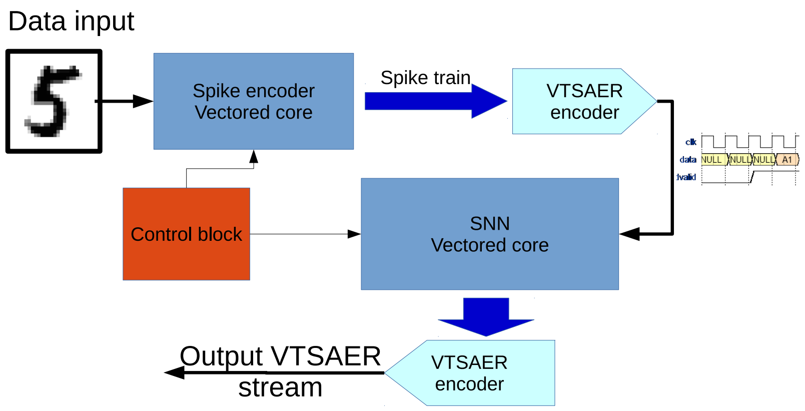


FIGURE 5.16: Neuromorphic real-time processing system layout. The non-spiking input is encoded into spikes with the vectored RF architecture described in 6. The result is encoded into VTSAER using mapper block, and the encoded VTSAER stream is processed by the dedicated SNN core.

the first case, the system is working in true real-time mode, encoding non-spiking input on the fly and processing it with the separate SNN core. VTSAER encoding protocol is described in Section 3.4.2.

The second figure represents the combined vectored core for the encoding and network inference. The vectored RF encoding architecture is described in Section 6.3. The combined architecture reuses the same hardware for encoding and SNN inference or training. Initially, the core works in the encoder mode, accepting non-spiking data and generating the spike train, encoded into VTSAER with the corresponding block. A VTSAER FIFO memory is added to hold the VTSAER stream of the encoded input. After all inputs for the given sample are encoded, the core is switched into the SNN mode, and the data from VTSAER FIFO are taken as an input to the core. The process is repeated for the next input sample. VTSAER FIFO can be also used as an interlayer memory, allowing

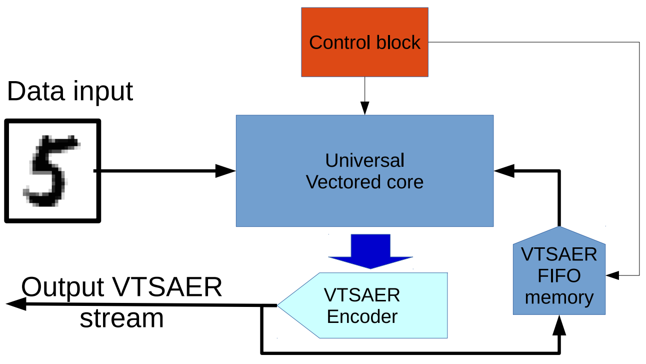


FIGURE 5.17: Neuromorphic processing system layout with one universal core. The similarity in the RF encoding and SNN architectures make the combined vectored core possible. The core is encoding the input sample first and the resulting spike train is encoded into the VTSAER stream and stored in FIFO. After that, the VTSAER is read from the FIFO and processed in the universal core in the SNN mode.

to compute multiple layers with the same SNN core.

The global layout of the proposed System-On-Chip is shown in Figure 5.18. The backbone of the system is the interconnect bus; standard interconnect type for Xilinx is an AXI4 bus. Processor, executing user code, is using the SNN core as an accelerator. The core is configured by the processor. Spike stream can be transmitted either through the SoC interconnect, or directly into the core. A multi-chip system can be build utilizing dedicated high speed AER links [45, 64].

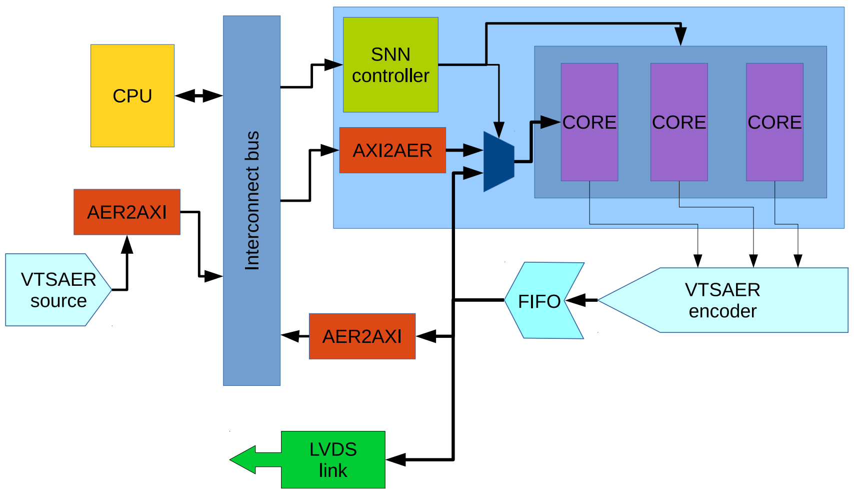


FIGURE 5.18: Possible SoC schematic utilizing the proposed core.

LIF Neuron type	DSP	Memory usage	Remarks
Linear	0	1	The simplest model, worse results in training
N-line piecewise linear	0	N	Additional TSE detector necessary, TSAER requires special generation
Shift-based	0	0	Limited choice of decay values
Sub-mul	1	1 (CLB)	Most versatile type
Multiplier-based	1	0	Worst speed

Table 5.6: General comparison of the described neurons. Sub-mul neuron uses one additional register. Memory usage refers only to the memory necessary to implement the selected PSP function.

As a summary, this chapter described four main novelties of this thesis: a flexible spike encoding protocol, a multiplierless neuron architecture, a novel vectored topology with linear hardware complexity and STDP mechanism modification. It was shown that spiking neural networks implemented in software can be ported into the FPGA. The proposed topology and neuron architecture describes the universal blocks to create various feed-forward networks. The proposed neuron architecture has flexible membrane potential decay options, with positive and negative sides described in table 5.6. With the results presented, the Sub-mul optimized architecture is recommended for the systems where DSP blocks are available and the precise membrane potential decay is necessary. Shift-based neuron offers very good area, speed, and precision for the selected decay rates.

Chapter 6

Receptive field encoding and spike train generation

Visual receptive fields described in Section 3.3 pose a significant computation task and then, the hardware acceleration can speed up the whole system. In this section, two RF hardware architectures will be presented and compared. The first architecture was developed at the beginning of the work on the thesis and was published in 2014 [65]. It is an example of fully parallel design. The second architecture is the vectored type architecture and shows superior performance in tests.

The neurons in the receptive or sensory layer generate a response defined by Equation 6.1, which is the Frobenius inner product (FIP) of input I and weight W; matrices are of size $i \times j$.

$$R_{RF} = \sum I_{ij} * W_{ij} \tag{6.1}$$

The matrix W_{ij} defines a receptive field of the neuron, where i is X-axis resolution and j is Y-axis resolution. These RFs can be used as line detectors, small shape detectors, performing feature extraction for higher layers. Simple classification tasks (i.e. the inclination of the line, circle or non-circle object) can be performed by single-layer receptive field neurons. Having normalized input and weights, the maximum excitation will be achieved when the input exactly matches the weight matrix.

As stated earlier, sensory layer neurons generate spikes at a frequency proportional to their excitation level. As the neuron firing frequency can not be infinite, the maximum firing rate is limited. The spiking response firing rate (FR_n) is described by Equation 6.2, where η_{min} is the defined minimum refractory period, R_n is current RF response and R_{max} is the maximum possible value of receptive field response.

$$FR_n = \frac{1}{\eta_{min} * \frac{R_{max}}{R_n}} \tag{6.2}$$

Image-to-spike encoding task can be split into two blocks: Frobenius Inner Product (FIP) modules used for calculation of the receptive field response, and integrate-and-fire or stochastic neurons generating spike trains with firing rate proportional to the RF response. The VTSAER encoder, described in Section 3.4.2 can be connected to serve the VTSAER-based systems. The overall schematic is shown in Figure 6.1, a control logic to coordinate the system is based on a Finite State Machine and is responsible for the image loading and storing intermediate results. For testing purposes, a hardware system based on Xilinx PLB-IPIF architecture with Microblaze softcore processor (not shown in Figure 6.1) to link with the developed module was used.

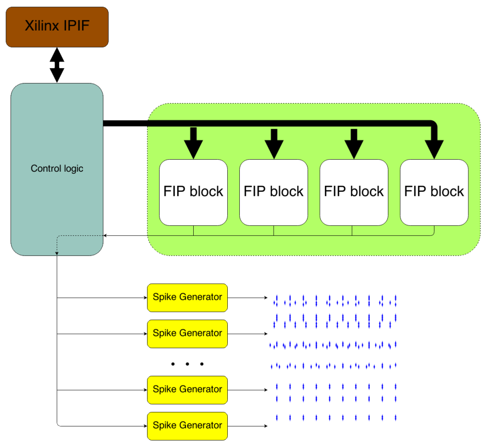


FIGURE 6.1: Implemented system schematic.

Spike train can be generated in deterministic and stochastic manner. A simple Integrate-and-Fire (IF) (without leakage) neuron can generate deterministic spike train. The deterministic neuron schematic is shown in Figure 6.3A. The neuron consists of an accumulator and a comparator with the constant threshold value. The receptive field response *RF(x)* is accumulated until the threshold is crossed. When this happens, the comparator output zeroes the accumulator and the cycle starts from the beginning. The

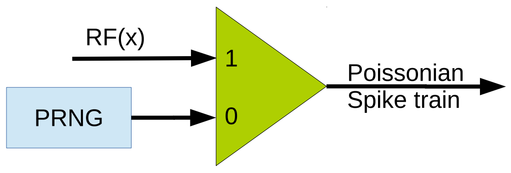


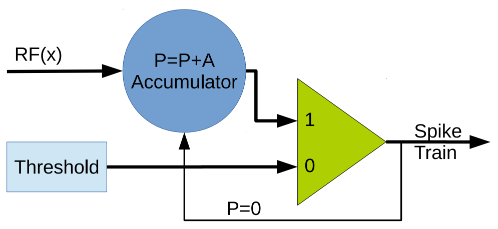
FIGURE 6.2: Poissonian spike generation with the PRNG and comparator.

firing rate has no lower limit, and the upper limit is equal to the maximum possible *RF(x)* divided by the threshold value.

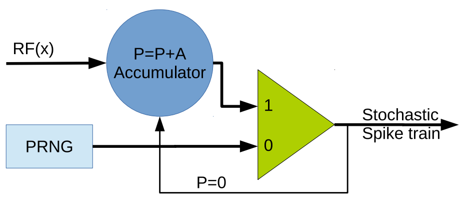
Stochastic spike train generation is more complex. Traditionally, stochastic spike trains are modeled with the Poisson process [66]. The Poissonian spike train has the average spike rate per time proportional to the input stimuli, while the spikes themselves are random and mutually independent. Such spike train can be generated by a the random number generator and a comparator, as shown in Figure 6.2. However, most of the LIF and IF models include refractoriness, which means that the spikes are not mutually independent and the probability of firing a spike is the function of input stimulus and the time passed from the previous spike. This process can be modeled by replacing the constant threshold with the random number generator. Now the neuron has a probability of spiking with any membrane potential, and this probability is increasing with every time unit passed after the previous spike, as the membrane potential increases. The schematic of this type of IF neuron is shown in Figure 6.3B.

Implementing true random number generator is a challenge, as it should have some external source of randomness: radio noise, radioactive decay etc. However, for the stochastic spike train generation task, the requirements are not so strict and a procedural Pseudo-random Number Generator (PRNG) with normal output distribution is sufficient. PRNG, used in this implementation is a 64-bit linear feedback shift register (LFSR), resulting in a total period of 2^{64} clock cycles.

Typically, LFSRs are widely used as a pseudo-random number generators in embedded systems as they are very hardware-efficient and fast, with sufficient randomness for non-critical applications. For better randomness, LFSR is initialized with a non-zero input value to the neuron or an arbitrary 8-bit number provided by parent IP-core. LFSR is constantly running and, after the regeneration request caused by reset or output spikes, the new current threshold value is set on the block output.



(a) Deterministic spiking neuron.



(b) Stochastic spiking neuron.

FIGURE 6.3: Integrate-and-Fire neuron for deterministic and stochastic spike train generation.

6.1 Frobenius inner product calculation implementation

For the base module of 8x8 RF with 8-bit data width, the block consists of 64 parallel 8-bit signed multipliers and one 65-input 8-bit signed adder. For partially parallel computation of big RF response, the block is designed with a HOLD signal that reroutes the previous adder value to the 65^{th} input, allowing to calculate Frobenius product for the big RFs. This block calculates the output in 2 clock cycles, but the processing can be pipelined, i.e. calculation of the sum of the multiplication product can be done in the same cycle as new data arrive to the multiplier. The receptive field to make convolution can be hardcoded or loaded dynamically to the module. The hardcoded block requires about two times fewer resources compared to pure CLB implementation, then, it is favorable if no reconfiguration is required. A general schematic of FIP block is shown in Figure 6.4. The block supports RF sizes up to 8x8 and 512-bit input data width. If receptive fields of bigger size are necessary, FIP is used in serial mode, calculating the field response in several cycles. As a single 8x8 Frobenius norm calculation is done in 2 clock cycles, a 32x32 RF response is calculated in $16 * 2 = 32$ clock cycles. In existing implementations [67, 68], RF size varies between 5 and 9; thus a base size of 8x8 for the FIP module was chosen. All modules use saturated signed fixed-point arithmetic with user-defined data width. All simulations and area occupation calculation were made with 8-bit signed arithmetic.

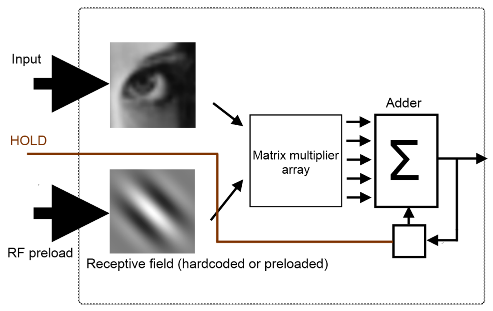


FIGURE 6.4: Frobenius inner product module.

6.2 Hardware occupation and speed of fully parallel FIP implementation

Hardware resource occupation is presented in Table 6.1. Three versions of FIP module were implemented. The multipliers can be either synthesized from CLBs or from the hard DSP48 blocks embedded in the FPGAs. The dedicated DSP48 blocks provide better speed and scalability, but their number is limited. If possible, DSP48 blocks can be used as multipliers. Also, weight loading interface increases the complexity of the module, and the hardcoded version with constant weights should have less hardware occupation. The synthesis was made in Xilinx ISE 14.2 software for Xilinx Spartan 6 family of devices with default optimization settings (optimization goal: speed, optimization effort: normal, Use DSP Blocks: Auto). The largest device in the family, XC6-SLX150T contains 180 DSP48 blocks, which means that it can contain a minimum three of the FIP modules without significant utilization of other logic resources. This is described in the table as ”FIP on DSP48”.

	LUT	LUT-FF pairs	DSP48	Max_f(MHz)
Stochastic IF neuron	54	83	0	222.82
Neuron layer(1K neuron)	55242	83886	0	222.82
8x8 FIP on DSP48	1242	1242	64	40
8x8 FIP on CLB	6618	6618	0	35.84
8x8 FIP CLB/hardcoded	3642	3642	0	41.2

TABLE 6.1: Implementation Results for the proposed blocks: Integrate&Fire neuron, a neuron layer of 1024 units and three different Frobenius Inner Product (FIP) encoding block implementations

It can be seen that hardcoded FIP RF is smaller than normal RF FIP block, but a DSP48 implementation is the smallest in terms of logic occupation. With pipelined operation and hardcoded RFs, a single 8x8 RF-FIP block can process up to 40 Mpix per second, utilizing around 7% of an XC6-SLX150T device. It can be efficiently implemented in parallel architectures counting thousands (more than 3000 for an XC6-SLX150T device) of encoding neurons. The processing speed is sufficient to encode stereo video stream with XGA resolution (1024x768@24 FPS) in real time.

6.3 Vectored FIP computation

The previous model, while being able to compute the 8×8 RF FIP in two clock cycles, requires 64 DSP blocks and runs on a maximum frequency of 40 MHz. Also, the dedicated DSP blocks cannot be shared within the design to be used for other functions. Another complex spot is the input data bus. It assumes that the input image can be read in blocks of 8x8 pixels, and with 8 bits per pixel the data bus should be 512 bit wide. The vectored architecture utilizes narrow data bus and is optimized for serial pixel input with multiple RFs calculated simultaneously. Let **W** be an N-element array of receptive fields of size $M \times M$. The input image patch I and the RF can be represented as vectors $\vec{I}, \vec{W^n}$ of length M^2. Then the corresponding RF response in vector form is equal to Equation 6.3.

$$R_{RF^n} = \vec{I} \cdot \vec{W^n} \tag{6.3}$$

The whole array of RF can be computed by the series of equations: 6.4.

$$\begin{aligned} R_{RF^1} &= I_1 * W_1^1 + I_2 * W_2^1 + \ldots + I_{M^2} * W_{M^2}^1 \\ R_{RF^2} &= I_1 * W_2^1 + I_2 * W_2^2 + \ldots + I_{M^2} * W_{M^2}^2 \\ &\ldots \\ R_{RF^N} &= I_1 * W_1^N + I_2 * W_2^N + \ldots + I_{M^2} * W_{M^2}^N \end{aligned} \tag{6.4}$$

It is easy to note that computation can be parallelized on N nodes as vector-by-scalar multiplication. Input I_m can be shared for simultaneous computation. Array W is split into M independent memories with a common address bus and the R_{RF^n} sum of products is calculated by MAC operations.

After the computation of Frobenius inner product, the response of the given RF must

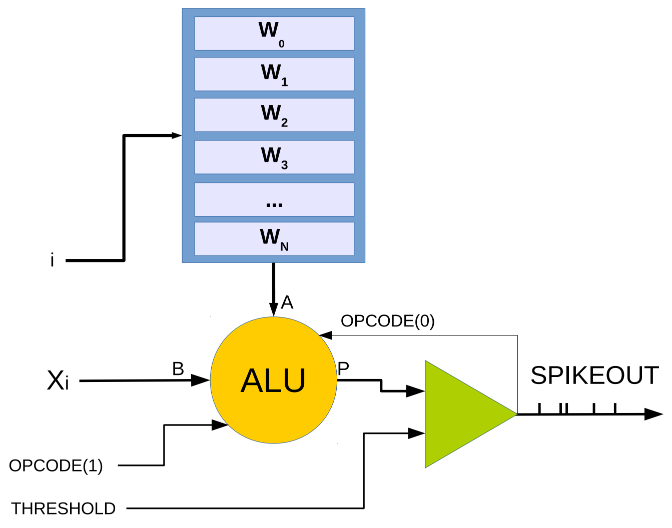


FIGURE 6.5: Vectored implementation of RF neuron.

Slice Registers	39
Slice LUTs	43
BlockRAM	1
DSP48E1	1
Frequency	325.256 MHz

TABLE 6.2: Hardware occupation of single RF Integrate-and-fire neuron.

be converted into a spike train. Frequency coding used in the IF neuron described in the fully parallel implementation uses add-accumulate operation. The vectored model can use the same neuron, but as MAC and add-accumulate operations can be both implemented in the same DSP48 block, the design can be optimized. The hardware design of a Integrate-and-Fire neuron with vectored Receptive Field encoding is shown in Figure 6.5.

The main elements of the neuron are the BRAM-ALU pair. ALU is modified to have two accumulators: one for the FIP calculation, second for Integrate&Fire membrane potential. The threshold in this implementation is a constant value; for the stochastic spike train generation, an RNG can be easily added to the design.
Due to its simplicity, the implementation has higher working frequency. Synthesis results for this neuron are shown in Table 6.2.

For maximum efficiency, the implementation is pipelined. Reading from memory takes one clock cycle, MAC operation is pipelined into 2 cycles. The waveform of the working neuron is shown in Figure 6.6.

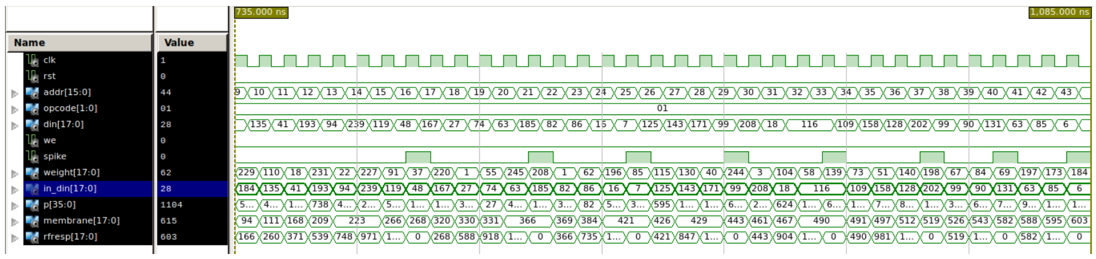


FIGURE 6.6: Hardware simulation of a neuron. Easy to notice how ISI is changing from the infinity (no spikes, response is below threshold) to 4 and then to 3 clock cycles.

6.4 On numerical accuracy of RF computation and spiking response.

As this module requires multiplication, a normalization for Q-format is necessary. Normalization increases the numerical error, but for the spike generation in discrete time units it is not important. Also, normalization discards small weights, increasing the sparsity of the reaction. By discarding very small RF response, the spike frequency has a minimum limit. It may be undesired on very large receptive fields, but in practice, RFs larger than 13×13 are rarely used. Consider a 13×13 RF with 8-bit fractional width weights. The minimal possible response for this RF is $2^{-8} = 0.0039$. If the whole RF response is normalized to 1, the possible output range is limited to 99.61% of numeric range. If the weights' values are normalized to 1 ($R_{max} = 169$), the possible output range is almost 100%. As the synthesized spike streams are not intended to be biologically plausible, weight range, timing units and refractory period can be arbitrary.

However, if all input values for the separate pixels are below some threshold, their individual RF responses will be too weak and will be rounded to 0 with normalization. The minimum single multiplication response is equal to 1×2^{2Y} for QX.Y format, and, as it was explained in Section 5.1.3, the overall maximum RF response error is $\pm N \times 2^{-Y}$. For the given RF it is 0.6602. Obviously, it is an inappropriate error for an RF response normalized to 1, but for the $R_{max} = 169$, the error will only be 0.39%. Numerical accuracy and maximum error for various RFs and fractional part sizes are presented in Table 6.3.

Results illustrate the influence of the numerical accuracy on the spike response. Consider an RF neuron with $\eta_{min} = 10\ TU$, $R_{max} = 1$ (Equation 6.2). The input I and RF response are normalized to 1 and stored with 8-bit precision. Interspike interval

(ISI) will vary between 10 (maximal response) and 15 (minimal response) TUs. For the $R_{max} = 169$, ISI will vary between 10 and 2560. Such difference in ISI boundaries shows how important is the proper data format selection.

Fractional part, bits	1×1	3×3	5×5	9×9	13×13
6	1.5625e-02	1.4062e-01	3.9062e-01	1.2656e+00	2.6406e+00
8	3.9062e-03	3.5156e-02	9.7656e-02	3.1641e-01	6.6016e-01
10	9.7656e-04	8.7891e-03	2.4414e-02	7.9102e-02	1.6504e-01
12	2.4414e-04	2.1973e-03	6.1035e-03	1.9775e-02	4.1260e-02
14	6.1035e-05	5.4932e-04	1.5259e-03	4.9438e-03	1.0315e-02
16	1.5259e-05	1.3733e-04	3.8147e-04	1.2360e-03	2.5787e-03

Table 6.3: Accuracy and maximal error for different RF size and fractional part width.

6.5 Comparison of fully parallel and vectored FIP implementation.

In this section, two different architectures were presented. They both represent classical "time vs size" tradeoff with linear dependability in terms of processing cycles. However, working frequency of FPGA circuits is highly dependent of routing and functions with multiple arguments (64-input adder) work significantly slower. Vectored implementation has the same hardware complexity (if the weights are stored in BRAM) for the RFs up to 1024 elements. Single BRAM48 can be configured as $4 \times 9 - bit 1 K element$ independent single-port RAMs, thus it can be shared among 4 RF neurons. Fully parallel architecture requires multiple DSP blocks, and weights are stored in CLB-synthesized RAM. Let's compare the 8×8 RF neurons. The fully parallel architecture will compute the output in 2 clock cycles on 40 MHz, or 0.05 μs. The vectored architecture will use 64 cycles for computation, and 3 more cycles for passing through the pipeline. Total execution time of 67 clock cycles on 325 MHz is 0.206 μs. While using 64 times more resources, the fully parallel architecture is only 4 times faster. In typical tasks there are several RF used, and 4 parallel RF vectored neurons vastly outperform fully parallel neurons. Proposed architecture, based on BRAM-ALU pair, provides a versatile computating platform for various algorithms that can be represented as a sequence of vector-by-scalar operations. A variation of such architecture was successfully applied for matrix multiplication [60]. Automata neuron implementation, presented in the previous chapter is also built using this schematic. It is possible to create a universal neural block that can be used as

RF encoder or spiking neuron by merging the vectored RF architecture with spiking elements of Automata neuron.

Chapter 7

Final remarks and conclusions

7.1 Comparison with the state of the art hardware implementations

It is hard to directly compare experimental systems, as almost every architecture reported in the scientific literature has unique features and was made using different hardware. The architecture, described in this work can process up to 370 million input events per second with the spiking network of 1 to 10000+ neurons in parallel, and any arbitrary number of neurons serially. The main limitation is the amount of on-chip block memory available. The new generation Xilinx Ultrascale+ Virtex devices host up to 95 Mb of BRAM and up to 360 Mb of UltraRAM, which is also compatible with the architecture. Even cost-optimized Kintex family contains up to 35 Mb of BRAM and 36 Mb of UltraRAM. A single medium-scale Kintex FPGA can contain more than 2 million 18-bit weights or 4 million 9-bit ones. This is a dramatic improvement over the previous FPGA generations. Older architectures that use external DDR memory for weight storage become less competitive with such resources, and the parallel event processing speed becomes more important. Minitaur, a hardware-implemented spiking RBM offers up to 18.73 million of synaptic operations per second with 75 MHz equivalent clock speed; more recent architecture described in [69] goes up to 196 million of synaptic updates (operating on internal BRAM), and has the STDP capability. Both architectures are coarse-grained, with large neural processing cores. SPEEDS architecture [70] is a mix of coarse and fine-grained architectures with external memories and performs up to 72 millions of synaptic operations per second. Coarse-grained architectures provide a more realistic comparison. Some examples of single spiking neuron occupation are shown in Table 7.1. The proposed architecture outperforms other implementations in terms of speed and occupation, with hardware complexity less that 10 times compared to previous works.

How big the implemented network can be? What is the maximum performance? The answers to these questions can be only speculative. The vectored architecture maintains the high clock speed regardless of the size of the layer, and then the maximum throughput can be deducted from the number of elements in parallel. One processing node performs 387 millions of synaptic computations per second. A layer of thousand units will perform 387 billions of synaptic computations, and 10000 units go up to 3.87 trillion of synaptic computations in one second. However, the throughput of the existing hardware interfaces will reduce these theoretical numbers by several times. The limitations are not within the architecture itself but on the amount of memory available and the capability to provide equally fast data input and output. Also, the largest practically used spiking architectures nowadays rarely exceed 1000 neurons. It is more useful to focus on efficient implementations dedicated for smaller, cost-efficient chips and the development of high-bandwidth spike transmission interfaces. The vectored architecture can offer excellent processing speed and latency on all contemporary FPGA families, from mid-range Spartan-6 up to high-end Virtex Ultrascale+.

Device and implementation	LUT	FF	DSP48A	Clock speed(MHz)
Spartan-6, [71]	723	808	0	322.82
Zynq, [72]	3000	n/a	9	200
Virtex-2, [73]	602	491	0	204.311
Spartan-6, [74]	1221	864	0	128.7
Spartan-3, [75]	543	250	1	69
Virtex-7, this work (linear)	97	86	0	387.372
Spartan-6, this work (linear)	95	83	0	272.046

Table 7.1: Comparison of the selected fine-grained spiking neuron hardware implementations.

7.2 Conclusions

FPGA usage for SNN implementation is not a new topic. Significant efforts were made and a number of systems of various complexity was built. However, most of the implementations were made by porting software algorithms into hardware or designed to have very narrow usage. FPGAs, being in the middle between dedicated ASICs and general-purpose CPU-based computing systems seldom were perceived as self-sufficient neuromorphic platforms. The presented work is an effort to join the scattered elements of the neuromorphic hardware into a versatile, universal architecture beneficial for modern commercial hardware. The accepted neuronal models and networks are much closer to the classic Machine Learning field than to the biological models, used in neuroscience. My goal was not to build just another large-scale emulation circuit, but a working system that has a practical value.
The main novelties, proposed in this thesis are:

- A new modification of AER protocol. The proposed variation offers true synchronicity of the events, elimination of timestamp overflow and uniform processing procedure.
- Novel LIF neuronal model implemented as a finite automaton. This approach allowed me to dramatically simplify the logic circuitry with the preservation of the desired functionality. By converting the input data stream into a control sequence, I removed the internal state change logic from the neuron and eliminate the *time-dependency* of the system. The states and outputs of Automata neurons are independent of the system clock and are defined only by the input VTSAER stream. Neurons can be synchronous or asynchronous, they accept variable data rate. Developed PSP models are efficient and have good accuracy compared to the software implementations. The neuronal functionality was verified with neural networks trained on publically available datasets.
- A novel approach to the STDP realization. By using interleaved VTSAER, I was able to create a small, independent STDP block with finite automaton behavior. The proposed method allows to implement various types of causality-driven synaptic plasticity, and learning is done with half of the neural computation speed on the interrupted data input stream.

- Vectored network architecture, allowing to implement very large networks with an arbitrary number of layers. The erformance of the proposed architecture outperforms significantly exisiting designs, and versatility allows to implement virtually any type of feedforward network as well as receptive field encoding layer. The benefits and features of this architecture go beyond the SNN field and I have successfully applied the developed architecture for the matrix multiplication and classic ANN computation tasks. Regarding the spiking computations, a vectored architecture implemented on modern FPGA device can provide a true single-chip low-power neuromorphic processing. The overall performance will allow to process information from multiple neuromorphic sensors, and low latency makes it a viable option for the real-time controlling systems.

- Vectored approach to the visual receptive fields data encoding. The proposed algorithm and architecture also goes beyond the SNN field and can be used for convolutional neural networks as well.

The philosophy of lightweight, massively parallel computation with "hardware before software" thinking gave me the possibility to explore previously omitted paths in the neuromorphic hardware field. The capabilities of spiking neural networks grow every day, and undoubtedly SNNs will find a proper place in robots, drones and embedded systems, performing real tasks. And I believe FPGAs will play an important role in it. I hope my work will help others to go further, to port more and more SNNs into hardware, accelerating them hundreds of times.

Bibliography

[1] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[2] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory.* Psychology Press, 2005.

[3] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[4] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

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

[6] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[7] Thomas M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

[8] Simon Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998. ISBN 0132733501.

[9] Filip Ponulak and Andrzej Kasiński. Supervised learning in spiking neural networks with resume: Sequence learning, classification, and spike shifting. *Neural Comput.*, 22(2):467–510, February 2010. ISSN 0899-7667. doi: 10.1162/neco.2009.11-08-901. URL http://dx.doi.org/10.1162/neco.2009.11-08-901.

[10] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9:99, 2015.

[11] Timothée Masquelier, Rudy Guyonneau, and Simon J Thorpe. Competitive stdp-based spike pattern learning. *Neural computation*, 21(5):1259–1276, 2009.

[12] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015.

[13] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[14] Eugene M Izhikevich. Which model to use for cortical spiking neurons? *IEEE transactions on neural networks*, 15(5):1063–1070, 2004.

[15] Wikimedia Commons. Biological neuron structure, 2017. URL https://en.wikipedia.org/wiki/File:Neuron-no_labels2.png.

[16] Thomas Splettstoesser. Biological synapse structure, 2017. URL https://commons.wikimedia.org/wiki/File:SynapseSchematic_lines.svg.

[17] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[18] Allan L Hodgkin and Andrew F Huxley. The dual effect of membrane potential on sodium conductance in the giant axon of loligo. *The Journal of physiology*, 116(4): 497, 1952.

[19] Wulfram Gerstner. A framework for spiking neuron models: The spike response model. *Handbook of Biological Physics*, 4:469–516, 2001.

[20] Taras Iakymchuk, Alfredo Rosado-Muñoz, Juan F. Guerrero-Martínez, Manuel Bataller-Mompeán, and Jose V. Francés-Víllora. Simplified spiking neural network architecture and stdp learning algorithm applied to image classification.

EURASIP Journal on Image and Video Processing, 2015(1):4, 2015. ISSN 1687-5281. doi: 10.1186/s13640-015-0059-4. URL http://dx.doi.org/10.1186/s13640-015-0059-4.

[21] John R Hughes. Post-tetanic potentiation. *Physiological reviews*, 38(1):91–113, 1958.

[22] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, 275 (5297):213–215, 1997.

[23] Olivier Bichler, Damien Querlioz, Simon J Thorpe, Jean-Philippe Bourgoin, and Christian Gamrat. Unsupervised features extraction from asynchronous silicon retina through spike-timing-dependent plasticity. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 859–866. IEEE, 2011.

[24] Thomas J Strain, LJ McDaid, Liam P Maguire, and T Martin McGinnity. A supervised stdp based training algorithm with dynamic threshold neurons. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 3409–3414. IEEE, 2006.

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

[26] Abigail L Person and David J Perkel. Unitary ipsps drive precise thalamic spiking in a circuit required for learning. *Neuron*, 46(1):129–140, 2005.

[27] P. Smolensky. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Information Processing in Dynamical Systems: Foundations of Harmony Theory, pages 194–281. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. URL http://dl.acm.org/citation.cfm?id=104279.104290.

[28] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

[29] Y. Bu, G. Zhao, A.-l. Luo, J. Pan, and Y. Chen. Restricted Boltzmann machine: a non-linear substitute for PCA in spectral processing. *aap*, 576:A96, April 2015. doi: 10.1051/0004-6361/201424194.

[30] C. C. Tan and C. Eswaran. Performance comparison of three types of autoencoder neural networks. In *2008 Second Asia International Conference on Modelling and Simulation (AMS)*, pages 213–218, May 2008. doi: 10.1109/AMS.2008.105.

[31] Emre Neftci, Srinjoy Das, Bruno U. Pedroni, Kenneth Kreutz-Delgado, and Gert Cauwenberghs. Event-driven contrastive divergence for spiking neuromorphic systems. *CoRR*, abs/1311.0966, 2013. URL http://arxiv.org/abs/1311.0966.

[32] Daniel Neil. *Online Learning in Event-based Restricted Boltzmann Machines*. PhD thesis, Institute of Neuroinformatics, 2013.

[33] Rafael Serrano-Gotarredona, Matthias Oster, Patrick Lichtsteiner, Alejandro Linares-Barranco, Rafael Paz-Vicente, Francisco Gómez-Rodríguez, Luis Camuñas-Mesa, Raphael Berner, Manuel Rivas-Pérez, Tobi Delbruck, et al. Caviar: A 45k neuron, 5m synapse, 12g connects/s aer hardware sensory–processing–learning–actuating system for high-speed visual object recognition and tracking. *IEEE Transactions on Neural Networks*, 20(9):1417–1438, 2009.

[34] Tobi Delbrück, Bernabe Linares-Barranco, Eugenio Culurciello, and Christoph Posch. Activity-driven, event-based vision sensors. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 2426–2429. IEEE, 2010.

[35] Shih-Chii Liu, André Van Schaik, Bradley A Minch, and Tobi Delbruck. Event-based 64-channel binaural silicon cochlea with q enhancement mechanisms. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 2027–2030. IEEE, 2010.

[36] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbruck, and Michael Pfeiffer. Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in neuroscience*, 7, 2013.

[37] Edgar Douglas Adrian. The impulses produced by sensory nerve-endings: Part 4. impulses from pain receptors. *The Journal of physiology*, 62(1):33, 1926.

[38] D. Roggen, S. Hofmann, Y. Thoma, and D. Floreano. Hardware spiking neural network with run-time reconfigurable connectivity in an autonomous robot. In *NASA/DoD Conference on Evolvable Hardware, 2003. Proceedings.*, pages 189–198, July 2003.

[39] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[40] Luis M Martinez and Jose-Manuel Alonso. Complex receptive fields in primary visual cortex. *The neuroscientist*, 9(5):317–331, 2003.

[41] Dictionary of optometry and visual science, 7th edition. *Clinical and Experimental Optometry*, 92(5):465–465, 2009. ISSN 1444-0938. doi: 10.1111/j.1444-0938.2009.00388.x. URL http://dx.doi.org/10.1111/j.1444-0938.2009.00388.x.

[42] John G Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *JOSA A*, 2(7):1160–1169, 1985.

[43] S. Marĉelja. Mathematical description of the responses of simple cortical cells$*$. *J. Opt. Soc. Am.*, 70(11):1297–1300, Nov 1980. doi: 10.1364/JOSA.70.001297. URL http://www.osapublishing.org/abstract.cfm?URI=josa-70-11-1297.

[44] Misha Mahowald. *VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function.* PhD thesis, California Institute of Technology, Pasadena, California, 5 1992.

[45] T. Iakymchuk, A. Rosado, T. Serrano-Gotarredona, B. Linares-Barranco, A. Jiménez-Fernández, A. Linares-Barranco, and G. Jiménez-Moreno. An aer handshake-less modular infrastructure pcb with x8 2.5gbps lvds serial links. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1556–1559, June 2014. doi: 10.1109/ISCAS.2014.6865445.

[46] Juan Antonio Leñero-Bardallo, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. A five-decade dynamic-range ambient-light-independent calibrated signed-spatial-contrast aer retina with 0.1-ms latency and optional time-to-first-spike mode. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(10):2632–2643, 2010.

[47] Shih-Chii Liu, Tobi Delbruck, Giacomo Indiveri, Rodney Douglas, and Adrian Whatley. *Event-based neuromorphic systems.* John Wiley & Sons, 2015.

[48] Rafael Serrano-Gotarredona, Matthias Oster, Patrick Lichtsteiner, Alejandro Linares-Barranco, Rafael Paz-Vicente, Francisco Gomez-Rodriguez, Håvard Kolle Riis, Tobi Delbrück, Shih-Chii Liu, S Zahnd, et al. Aer building blocks for multi-layer multi-chip neuromorphic vision systems. In *NIPS*, pages 1217–1224, 2005.

[49] Alejandro Linares-Barranco, Gabriel Jimenez-Moreno, Bernabé Linares-Barranco, and Antón Civit-Balcells. On algorithmic rate-coded aer generation. *IEEE Transactions on Neural Networks*, 17(3):771–788, 2006.

[50] UCI Machine Learning Repository. *Semeion Handwritten Digit Dataset.* 2014. URL http://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit.

[51] Taras Iakymchuk, Alfredo Rosado-Munoz, Manuel Bataller-Mompeán, Juan F Guerrero-Martínez, and Jose V Francés-Víllora. Frequency spike encoding using gabor-like receptive fields. *IFAC Proceedings Volumes*, 47(3):701–706, 2014.

[52] A. Cassidy, S. Denham, P. Kanold, and A. Andreou. Fpga based silicon spiking neural array. In *2007 IEEE Biomedical Circuits and Systems Conference*, pages 75–78, Nov 2007. doi: 10.1109/BIOCAS.2007.4463312.

[53] D. Thomas and W. Luk. Fpga accelerated simulation of biologically plausible spiking neural networks. In *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, pages 45–52, April 2009. doi: 10.1109/FCCM.2009.46.

[54] K. L. Rice, M. A. Bhuiyan, T. M. Taha, C. N. Vutsinas, and M. C. Smith. Fpga implementation of izhikevich spiking neural networks for character recognition. In *2009 International Conference on Reconfigurable Computing and FPGAs*, pages 451–456, Dec 2009.

[55] C. Zamarreno-Ramos, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco. Multicasting mesh aer: A scalable assembly approach for reconfigurable neuromorphic structured aer systems. application to convnets. *IEEE Transactions on Biomedical Circuits and Systems*, 7(1):82–102, Feb 2013. ISSN 1932-4545. doi: 10.1109/TBCAS.2012.2195725.

[56] D. Neil and S. C. 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. ISSN 1063-8210. doi: 10.1109/TVLSI.2013.2294916.

[57] Xilinx. 7 series dsp48e1 slice user guide (v1.8). Technical Report UG479, Xilinx, Inc, November 2014. URL http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.

[58] Intel. Understanding peak floating-point performance claims, February 2017. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01222-understanding-peak-floating-point-performance-claims.pdf.

[59] Andrew Cassidy, Andreas G Andreou, and Julius Georgiou. A combinational digital logic approach to stdp. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pages 673–676. IEEE, 2011.

[60] Taras Iakymchuk, Alfredo Rosado-Munoz, Manuel Bataller Mompéan, Jose Vicente Frances Villora, and Emmanuel Ovie Osimiry. Versatile direct and transpose matrix multiplication with chained operations: An optimized architecture using circulant matrices. *IEEE Transactions on Computers*, 65(11):3470–3479, 2016.

[61] Xilinx. 7 series fpgas memory resources user guide (v1.11). Technical Report UG473, Xilinx, Inc, November 2014. URL http://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf.

[62] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J. Thorpe, and Timothée Masquelier. Stdp-based spiking deep neural networks for object recognition. *CoRR*, abs/1611.01421, 2016. URL http://arxiv.org/abs/1611.01421.

[63] Milad Mozafari, Saeed Reza Kheradpisheh, Timothée Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh. First-spike based visual categorization using reward-modulated stdp. *arXiv preprint arXiv:1705.09132*, 2017.

[64] A Yousefzadeh, M Jabłoński, T Iakymchuk, A Linares-Barranco, A Rosado, LA Plana, T Serrano-Gotarredona, S Furber, and B Linares-Barranco. Multiplexing aer asynchronous channels over lvds links with flow-control and clock-correction for scalable neuromorphic systems.

[65] T Iakymchuk, A Rosado-Munoz, M Bataller-Mompean, JF Guerrero-Martínez, JV Francés-Villora, M Wegrzyn, and M Adamski. Hardware-accelerated spike train generation for neuromorphic image and video processing. In *Programmable Logic (SPL), 2014 IX Southern Conference on*, pages 1–6. IEEE, 2014.

[66] David Heeger. Poisson model of spike generation. *Handout, University of Standford*, 5:1–13, 2000.

[67] C.A. Perez, C.A. Salinas, P.A. Estevez, and P.M. Valenzuela. Genetic design of biologically inspired receptive fields for neural pattern recognition. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 33(2):258–270, Apr 2003. ISSN 1083-4419. doi: 10.1109/TSMCB.2003.810441.

[68] J.A. Perez-Carrasco, Bo Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, Shouchun Chen, and B. Linares-Barranco. Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward convnets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(11):2706–2719, 2013. ISSN 0162-8828.

[69] Byungik Ahn. Special-purpose hardware architecture for neuromorphic computing. In *SoC Design Conference (ISOCC), 2015 International*, pages 209–210. IEEE, 2015.

[70] G. Séguin-Godin, F. Mailhot, and J. Rouat. Efficient event-driven approach using synchrony processing for hardware spiking neural networks. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2696–2699, May 2015. doi: 10.1109/ISCAS.2015.7169242.

[71] E. Jokar and H. Soleimani. Digital multiplierless realisation of a calcium based plasticity model. *IEEE Transactions on Circuits and Systems II: Express Briefs*, PP(99):1–1, 2016. ISSN 1549-7747. doi: 10.1109/TCSII.2016.2621823.

[72] Lei Wan, Yuling Luo, Shuxiang Song, J. Harkin, and Junxiu Liu. Efficient neuron architecture for fpga-based spiking neural networks. In *2016 27th Irish Signals and Systems Conference (ISSC)*, pages 1–6, June 2016. doi: 10.1109/ISSC.2016.7528472.

[73] H. Soleimani, A. Ahmadi, and M. Bavandpour. Biologically inspired spiking neurons: Piecewise linear models and digital implementation. *IEEE Transactions on*

Circuits and Systems I: Regular Papers, 59(12):2991–3004, Dec 2012. ISSN 1549-8328.

[74] M. Heidarpour, A. Ahmadi, and R. Rashidzadeh. A cordic based digital hardware for adaptive exponential integrate and fire neuron. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(11):1986–1996, Nov 2016. ISSN 1549-8328.

[75] T. Iakymchuk, A. Rosado, J. V. Frances, and M. Bataller. 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, July 2012.