Universitat
de les Illes Balears

DOCTORAL THESIS
2021

HARDWARE IMPLEMENTATION OF MACHINE LEARNING AND DEEP LEARNING SYSTEMS ORIENTED TO IMAGE PROCESSING

Christiam Camilo Franco Frasser

Universitat
de les Illes Balears

DOCTORAL THESIS
2021

Doctoral Programme of Electronic Engineering

HARDWARE IMPLEMENTATION OF MACHINE LEARNING AND DEEP LEARNING SYSTEMS ORIENTED TO IMAGE PROCESSING

Christiam Camilo Franco Frasser

Thesis Supervisor: José Luis Rosselló Sanz
Thesis Supervisor: Miquel Jesús Roca Adrover
Thesis Tutor: José Luis Rosselló Sanz

Doctor by the Universitat de les Illes Balears

To Christian and Matias.

Acknowledgements

First and foremost, I wish to thank the main advisor and mentor of this PhD work José Luís Rosselló Sanz, who has guided me and encouraged me along the way. From the very first time, he manifested his confidence in myself, which really keeps me on track despite it was not an easy road. I have learned a lot of things from him, which I could not enumerate in this section; from his invaluable advice, patience, kindness and indeed experience. It has been an honor from beginning to end, to be his student.

I want to express my gratitude to Professor Miquel Roca for the support throughout this work. His suggestions and helpful advice came in crucial moments. A special thanks to Dr. Vicente Canals Guinand for his valuable experience in the field of knowledge. His expertise in Stochastic Computing really made me improve the different designs presented in this work. His great sense of humor encouraged me in moments where I really needed to.

I am thankful as well to my colleagues Alejandro Morán and Erik Skibinsky Gitlin, research members of the team, with whom I have had a lot of great discussions. Their different points of view in several moments were essential for the successful outcomes.

I wish to thank as well my family members, especially my mother and father for everything they have given to me.

But above all, I want to say "thank you" to my dear wife, Diana Marcela Rocha, who encouraged me to start the dream of my life: being a PhD. Her strength in supporting me while parenting two kids was invaluable.

Finally, I thank to my God and Savior: Jesus Christ.

Resumen

El aprendizaje de máquinas (ML) y aprendizaje profundo (DL) han experimentado un creciente interés en la comunidad científica en la última década. Desafortunadamente, la implementación de dichos algoritmos en hardware sigue siendo un gran reto. El gran número de operaciones que ocupan mucha área, como es el caso de la multiplicación, ha llevado a los investigadores a pensar en nuevas metodologías para poder operar. Una de las técnicas más prometedoras es la computación estocástica (SC). En la computación estocástica, un número es descrito como la probabilidad de encontrar un uno lógico a través de una cadena de bits, con esto, ocupando solamente 1 hilo en el bus de datos. Debido a que opera en el dominio probabilístico, operaciones complejas como la multiplicación son reducidas a compuertas lógicas individuales. Aun así, algunos desafíos siguen estando presentes, tal como: (a) los altos recursos utilizados para implementar generadores de números aleatorios (RNG) independientes, (b) la degradación de la precisión producida por los efectos de la correlación entre señales, y (c) la realización de otras funciones esenciales para la aplicación en cuestión.

En lugar de reducir el fenómeno de correlación, en este trabajo, estudiamos y explotamos los efectos que produce sobre las señales. Esto nos permite realizar operaciones esenciales en aplicaciones de DL como es la función máxima. Además, llevamos a cabo las pruebas de casos de uso en implementaciones digitales reales, no solo simulaciones, empleando plataformas FPGA. Para el modelo más complejo de la presente tesis (una red neuronal convolucional, CNN), se realiza una implementación completamente paralela utilizando un solo chip de FPGA. La síntesis VLSI de dicho circuito se compara con otros trabajos relevantes encontrados en la literatura, superando el rendimiento de todos ellos.

Resum

Els aprenentatges automàtic (ML) i profund (DL) han mostrat dins la darrera dècada un molt alt interès dins de la comunitat científica. Malauradament la implementació dels algorismes en hardware continua representant a dia d'avui un gran repte tecnològic. El gran nombre d'operacions involucrades en el procés impliquen la necessitat de molta àrea (com passa en el cas dels blocs multiplicadors molt presents dins els algorismes esmentats). Aquesta situació ha fet que els investigadors intentin desenvolupar noves metodologies per poder fer aquestes operacions entre moltes altres. Una de les tècniques que destaca pel seu interès és la computació estocàstica (SC). La computació estocàstica considera un senyal descrit com la probabilitat de trobar un 1 lògic a través d'una cadena de bits emprant, per tant, un únic fil del bus de dades. El fet de que la computació estocàstica operi en el domini probabilístic, operacions complexes com la multiplicació, abans esmentada, es redueixen a l'ús de portes lògiques simples (una única porta en el cas de la multiplicació). Tot i així, alguns reptes segueixen estant presents, tals com: (a) els recursos utilitzats per implementar generadors de nombres aleatoris (RNG) independents, (b) la disminució de la precisió que es produeix degut als efectes de la correlació entre senyals, i (c) la implementació d'altres funcions essencials per a les aplicacions basades en ML i DL.

En lloc d'intentar evitar el fenomen de correlació, en aquest treball ho empram degut als efectes que produeix sobre els senyals. Això ens permet fer operacions essencials en aplicacions de DL com és el cas de la funció màxim. A més, es duen a terme les proves d'aplicacions reals en implementacions digitals, i no només simulacions si no també emprant plataformes hardware basades en FPGA. Pel model més complex desenvolupat en la present tesi (una xarxa neuronal convolucional, CNN), es realitza una implementació completament paral·lela emprant una única FPGA. La síntesi VLSI d'aquest circuit es compara amb altres treballs rellevants trobats a la literatura, a on

es mostra que en el present treball es supera el rendiment de tots ells.

Abstract

Machine Learning (ML) and Deep Learning (DL) have experienced a booming interest from the research community in the last decade. Unfortunately, the implementation of such algorithms in hardware keeps being a total challenge. The vast number of area-hungry operations, such as the multiplication, has led researchers to think about new methodologies to operate. One of these promising techniques is Stochastic Computing (SC). In SC, a number is described as the probability to find a logic one along a bit-stream, therefore, occupying only 1-wire. Since it operates in the probabilistic realm, complex operations such as multiplication are reduced to single logic gates. However, some concerns are to be addressed in the quest to take ML and DL to SC: (a) the high resources used to implement independent Random Number Generators (RNGs), (b) the accuracy degradation produced by the correlation effects between signals, and (c) the SC realization of essential functions for the considered application. In this thesis, we tackle all these concerns. We present a deep analysis of the more employed RNG in SC: the LFSR. We provide two real case implementations in which only two LFSRs are needed to operate a complete DL model. Instead of mitigating the correlation phenomenon, as it is normally attempted by researchers, we study the correlation phenomenon and exploit the effects that it produces over signals. This allows us to realize essential operations in DL applications such as the maximum function. In addition, we conduct the use-case tests in real digital implementations, not only simulations, using FPGA platforms. For the more complex model in the present thesis (a Convolutional Neural Network), a fully-parallel implementation is realized on single FPGA chip. The VLSI synthesis of such a circuit is compared with other relevant works found in literature, surpassing the performance of all of them.

Contents

ACRONYMS

AE	Absolute Error
AF	Activation Function
AI	Artificial Intelligence
ALM	Adaptative Logic Module
ANN	Artificial Neural Network
APC	Accumulative Parallel Counter
AAPC	Approximate Accumulative Parallel Counter
ASIC	Application Specific Integrated Circuit
B2S	Binary to Stochastic
BRAM	Block RAM
BS	Bit-Stream
BSC	Binary to Stochastic Converter
CE	Computing Efficiency
CNN	Convolutional Neural Network
DFF	D-type Flip-Flop
DL	Deep Learning
DSP	Digital Signal Processor
FA	Full Adder
FEB	Feature Extraction Block
FF	Feed Forward
FLOP	Floating point Operation
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
IoT	Internet of Things
LAB	Layer Auto-scale Block
LD	Low Discrepancy
LFSR	Linear Feedback Shift Register

LSZ	Least Significant Zero
LUT	Look Up Table
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi Layer Perceptron
MP	Max Pooling
MPE	Molecular Pairing Energy
MSE	Mean Square Error
MSO	Most Significant One
NH	Neuromorphic Hardware
NN	Neural Network
OP	Operation
PC	Parallel Counter
PTS	Push To Sigma
RBF	Radial Basis Function
ReLU	Rectifier Linear Unit
ROC	Receiver Operating Characteristic
RNG	Random Number Generator
S2B	Stochastic to Binary
SNG	Stochastic Number Generator
SC	Stochastic Computing
SQF	Stochastic Quadratic Function
TC	Traditional Computing
TDP	Thermal Design Power
TOPS	Tera Operations Per Second
TRNG	True Random Number Generator
TSMC	Taiwan Semiconductor Manufacturing Company
VLSI	Very Large Scale Integration
VS	Virtual Screening
WET	Wire Exchanging Technique

Chapter 1

INTRODUCTION

The present chapter provides the motivation, the main objectives, as well as a summary of the present dissertation.

1.1 Motivation

Moore's Law has been considered as death in the last decade [1]. This law says that the number of transistors on a microchip doubles every two years. The high temperatures of transistors, their leakage currents, and the hurdles faced in the manufacturing process, eventually would make impossible to create smaller devices. This in fact has made the production and test of such devices increase the costs steadily with every new generation of chips. As transistors approximate to the atomic scale, which in turn they are, it becomes difficult to shrink them further. Moore himself proclaimed in a 2005 interview: "...the fact that materials are made of atoms is a fundamental limitation and it's not that far away...We're pushing up against some fairly fundamental limits so one of these days we're going to have to stop making things smaller" [2]. This has aroused an urgent research topic in the quest to rise more technological advances.

Machine Learning (ML) is an increasing area of research in the last decade. It comes as a subdivision of Artificial Intelligence (AI), which by means of algorithms, gives the capacity to computational machines of identifying patterns in massive data. This is done without being deterministically programmed to do it so. The process of learning such patterns from the input data is referred as *training*. After training, the system is able to predict or

categorize unseen input data by itself.

Deep Learning (DL) is at the same time a leaf of the ML tree. It focuses on a type of ML algorithm called Neural Networks (NNs). The term *deep* keeps being a dispute among experts because of disagreements about how deep must be the network to consider it is *deep*. In any case, generally speaking, it is normally referred as *deep* network that model which has more than one hidden layer. NNs or ANNs (Artificial-NNs) are one of the most important areas of study in the computing research field. They possess some advantageous characteristics as the non-linearity, flexible configuration, and self-adaptability, which make them the preferred approach for different practical problems. They have been used in classification problems [3], system control [4], feature extraction [5], organic chemistry [6] and computer vision [7]. Specifically, in the organic chemistry arena, the vast possibility for constructing chemical compounds is an enormous data space. This lead to an ever increasing number of compounds included in different chemical databases, which can approximate the billions. Actually, the aim of any drug discovery process is rooted in the identification of molecular compounds that can cure and treat disease. This application field is a perfect one where ANNs can be exploited for accelerating the compound searching process. On the other hand, the vision system is the richest sense that we as human have. This has aroused a great interest from the scientific community for years, with good improvements. However, it was not until technological capabilities allowed Convolutional Neural Networks (CNNs) to appear as the main solution for visual pattern recognition, that the computer vision community started to boom. Nowadays, CNNs are implemented in a wide range of applications such as the medical field [8], natural language processing [9], autonomous navigation [10], image captioning [11] and image recognition [7].

Interest in unburdening the work of CPU for ML applications by means of hardware accelerators has arisen as a consequence. The large amount of computation required for training and inference has lead researchers to look for solutions to leveraging the CPU work. Some candidates for doing this job have been proposed, from which GPUs, FPGA devices and ASIC chips stand out. Nevertheless, the exponential growing in model sizes creates the need for the developing of more efficient accelerators. Simultaneously, the proliferation of Internet of Things (IoT) in the daily life has attracted the community to run ML models *at the edge*. The term *at the edge* burns from contrasting how it is normally done in power and resource constrained devices: *in the cloud*. Running these models in the cloud rises its own short-

comings such as network coverage requirements, latency, data privacy and time access. That is why a localized scheme to process data is therefore required for cloud independence. However, the enormous amount of computation required, the limited resource budget, and the power constraints of such devices have restricted a seamless path. It is thereby a serious need to boost the researcher's efforts for improving the efficiency of such model's performance for edge devices. In addition to this, the CMOS down-scaling pushed by the semiconductor industries have resulted in more noise sensitive devices. The supply voltage reduction to avoid oxide breakdown and improve power consumption comes with the trade-off of producing a loss in noise robustness [12]. This has caused that the reliability of the systems drop, being critical for high precision applications like airplanes or medicine.

Stochastic Computing (SC) has been suggested to overcome the preceding drawbacks. Contrary to traditional binary computing (TC), SC circuits occupy a small footprint area and require a low power consumption. Many essential arithmetic operations such as addition, subtraction, and multiplication are implemented with very simple logic [13]. Moreover, SC possess the faculty of being inherently noise tolerant [14]. For these reasons, SC can facilitate to implement noise-tolerant ML algorithms with very low area usage and power consumption. This allows both, maximize parallelization for accelerator applications and implement complex models at the edge. Nevertheless, SC keeps being an immature technology. Different challenges keep being over the table, such as the inherently not exact outcomes, the long latency, the abundance of Random Number Generators (RNGs) employed and the correlation effects over signals. All of these mentioned hurdles rise an increasing research study in the scientific realm which motivated the current thesis dissertation.

1.2 Objectives

The present work is our research and contribution to circumvent the aforementioned hurdles. Its principal aim is to implement specific ML and DL algorithms employing SC in digital hardware platforms such as FPGA and ASIC devices. For this purpose, different secondary objectives must be accomplished:

1. Analyse the SC implementation of the main operations used in ML algorithms.

2. Find an efficient RNG scheme to reduce the amount of blocks used in the system without impacting the outcome precision and test it in real use cases.

3. Design an efficient SC-based Neural-Network framework and implement it in hardware.

4. Design an efficient SC-based Convolutional-Neural-Network framework and implement it in hardware.

Once the different objectives are enumerated, we proceed to indicate how the main chapters of this work are ordered and how the objectives are related to them.

1.3 Thesis Contributions

This subsection details the contributions of the present work to the state-of-the-art. For the sake of clarity, they are organized in four groups: contributions to international indexed journals, patents, conferences, and projects in which this work is framed.

1.3.1 Contributions to International Journals

The published contributions to international indexed journals directly related with the present PhD thesis are:

1. **Frasser, Christiam F.**, Miquel Roca, and Josep L. Rosselló 2021. "Optimal Stochastic Computing Randomization" Electronics 10, no. 23: 2985. IMPACT FACTOR: 2.397 [15]. This paper is the summary and results obtained from Chapter 3.

2. **Frasser, Christiam F.**, Carola de Benito, Erik S. Skibinsky-Gitlin, Vincent Canals, Joan Font-Rosselló, Miquel Roca, Pedro J. Ballester, and Josep L. Rosselló 2021. "Using Stochastic Computing for Virtual Screening Acceleration" Electronics 10, no. 23: 2981. IMPACT FACTOR: 2.397 [16]. This paper focuses in the SC-NN design and use-case presented in Chapter 4.

Additionally, one more paper is in review process in a high-impact indexed journal:

1. **Christiam F. Frasser**, Pablo Linares-Serrano, Alejandro Morán, Joan Font-Rosselló, V. Canals, Miquel Roca, T. Serrano-Gotarredona and Josep L Rosselló - "Fully-parallel Stochastic Computing Hardware Implementation of Convolutional Neural Networks for Edge Computing Applications", submitted to "IEEE Transactions on Neural Networks and Learning Systems" journal. IMPACT FACTOR: 10.451. This paper is the culmination and experimental results showed in Chapter 5.

Other papers published in high-impact indexed journals, which are not directly related to the present thesis, but which were written throughout the course of the thesis are the following:

1. Vicent Canals, **Christiam F Frasser**, Miquel L Alomar, Antoni Morro, Antoni Oliver, Miquel Roca, Eugeni Isern, Víctor Martínez-Moll, Eugeni Garcia-Moreno, Josep L Rosselló - "Noise tolerant probabilistic logic for statistical pattern recognition applications", published in a high-impact journal named: Integrated Computer-Aided Engineering, 24(4), 351-365 [14].

2. Miquel L. Alomar, Erik S. Skibinsky-Gitlin, **Christiam F. Frasser**, Vincent Canals, Eugeni Isern, Miquel Roca and Josep L Rossello, "Efficient parallel implementation of reservoir computing systems." in Neural Comput and Applic 32, 2299–2313. https://doi.org/10.1007/s00521-018-3912-4.

3. A. Morán, **C. F. Frasser**, M. Roca and J. L. Rosselló, "Energy-Efficient Pattern Recognition Hardware With Elementary Cellular Automata," in IEEE Transactions on Computers, vol. 69, no. 3, pp. 392-401, 1 March 2020, doi: 10.1109/TC.2019.2949300.

4. Alejandro Morán, Vincent Canals, Fabio Galan-Prado, **Christian F Frasser**, Dhinakar Radhakrishnan, Saeid Safavi and Josep L Rosselló, "Hardware-Optimized Reservoir Computing System for Edge Intelligence Applications" in Cognitive Computation, doi: 10.1007/s12559-020-09798-2.

1.3.2 Patents

Due to the high interest and booming research nowadays in developing hardware accelerators oriented to DL applications, the work developed in Chapter

5 was patented as:

1. **C. F Frasser** and J. L. Rosselló. "Elemento de generación de señales estocásticas, neurona estocástica y red neuronal a partir de esta". Spanish patent application number ES 2 849 123 A8, PCT number WO 2021/160914 A1, Feb 2021.

1.3.3 Contributions Published in International Conferences or in Public Repositories

Finally, the contributions in international conferences or in public repositories related to the present thesis are:

1. **Christiam F. Frasser**, Pablo Linares-Serrano, Alejandro Morán, Joan Font-Rosselló, V. Canals, Miquel Roca, T. Serrano-Gotarredona and Josep L Rosselló - "Exploiting Correlation in Stochastic Computing based Deep Neural Networks". In Design of Circuits and Integrated Circuits (DCIS) 2021.

2. **Frasser, C. F**., Linares-Serrano, P., Canals, V., Roca, M., Serrano-Gotarredona, T., and Rossello, J. L. (2020). Fully-parallel Convolutional Neural Network Hardware. arXiv preprint arXiv:2006.12439.

3. **Frasser, C. F.**, de Benito, C., Canals, V., Roca, M., Ballester, P. J., and Rossello, J. L. (2020). Stochastic-based Neural Network hardware acceleration for an efficient ligand-based virtual screening. arXiv preprint arXiv:2006.02505.

Other contributions published in international conferences or in public repositories where I have contributed but which have not been directly included in my thesis are:

1. Erik S Skibinsky-Gitlin, Miquel L Alomar, **Christiam F Frasser**, Vincent Canals, Eugeni Isern, Miquel Roca, Josep L Rosselló. "Cyclic reservoir computing with FPGA devices for efficient channel equalization". In International Conference on Artificial Intelligence and Soft Computing, ICAISC 2018.

2. Erik S Skibinsky-Gitlin, Miquel L Alomar, Vincent Canals, **Christiam F Frasser**, Eugeni Isern, Fabio Galán-Prado, Alejandro Morán,

Miquel Roca, Josep L Rosselló. "Fpga-based echo-state networks". In International Conference on Time Series and Forecasting (2018).

3. Morán, A., **Frasser, C. F.**, and Rosselló, J. L. (2018). Reservoir computing hardware with cellular automata. arXiv preprint arXiv:1806.04932.

1.3.4 Private and Public Projects in which this Thesis is Framed

1. Project TEC2017-84877-R, "Desarrollo de sistemas de computación no convencional de alto rendimiento y sus aplicaciones prácticas", financed by: Ministerio de Economía, Industria y Competitividad. Ips: José Luis Rosselló Sanz; Miguel Jesús Roca Adrover. (Universidad de las Islas Baleares). From 01/01/2018 to 30/09/2021. Amount financed: 204,490€.

2. Project "Design of a Neural Network/Reservoir Computing system for speech-recognition", financed by: Bridgewest Capital Management LLC. IP: Dr. Josep Luis Rosselló Sanz. (Universidad de las Islas Baleares). From 09/2018 to 01/2019. Amount financed: 45,360€.

1.4 Summary

This section is a summary of how this thesis is organized.

Chapter 2 is dedicated to introducing SC, its advantages against Traditional Computing (TC) and how we convert from TC to SC and vice-versa. Then, the correlation phenomenon is exposed for finally analysing the main SC-based operations performed in Machine Learning algorithms. In this chapter, the first objective is covered.

Chapter 3 presents an analysis of the Random Number Generators (RNGs) presented in the literature for SC, highlighting the LFSR circuit as the best alternative. The LFSR circuit has lately been discarded due to the correlation effects introduced when it is employed. However, we show that if seeded correctly, LFSR outperforms other RNGs, recovering the first place as the best choice for generating stochastic signals in hardware. By means of experimental results, we prove that LFSR performs better than other RNGs for the SC-based quadratic function, scaled addition and multiplication. Finally,

an application in the image processing realm is carried out: an image edge detection. This chapter covers the second objective.

Chapter 4 introduces the design of the SC-based ReLU neuron and how it is connected in the network using the more efficient blocks introduced in Chapter 2. Moreover, different solutions are presented for overcoming two relevant difficulties exhibited in SC-NN implementations: the constrained value range problem and the weight distribution problem. Finally, a virtual screening accelerator for drug discovery applications is implemented as a use case over a high-end FPGA. Objective three is covered here.

Chapter 5 provides the design of a fully parallel CNN hardware realization based on SC. A survey of some relevant works carried out in SC targeting CNNs is presented to finally conduct a comparison of our method versus theirs. As a proof of concept, the architecture is implemented in an FPGA and compared it with other FPGA works presented in the literature. The VLSI synthesis of the digital design is introduced and contrasted with other VLSI works. This chapter covers the fourth objective.

The conclusions and future work of the present dissertation are exposed in Chapter 6.

Lastly, an appendix where some relevant information from Chapter 3 is added: the best seeds for different use cases depending on the bit precision and the tap configuration of the LFSRs for different precision. Moreover, the VHDL code for the LFSR implementation is provided.

Chapter 2

STOCHASTIC COMPUTING

2.1 Introduction

There has been an increasing interest in the different non-deterministic computing architectures traditionally known as Von Newman. This interest comes from the fact of finding high parallelism and resource saving solutions to solve problems of high computational demand as is the case of machine learning. Stochastic Computing (SC) is one of this non-deterministic architectures that arises as a promising technique for machine learning hardware implementations and is becoming extensively used in neuromorphic applications [17]. It is based on the most simple mathematical codification that one can imagine. Unlike Traditional Computing (TC) codifications, where the bit weight (2^i) is proportional to 2 powered to its relative position 'i' in the bit stream; in SC, all the data bits processed have the same weight, providing several benefits as explained in this section.

In SC, data is defined by bit-streams that represent the probabilities of observing a logic high ('1') at an arbitrary bit position throughout the sequence of bits. For instance, the number 0.75 could be represented by a bit-stream in which the probability of finding a logic '1' along the bit-stream is 75%. Neither the length nor the pattern of the bit-stream require to be fixed, there are infinite approaches to represent the same number in Stochastic domain. Take as an example the number 0.75, which could be represented as $(1, 1, 0, 1)$ for a four bit-stream or $(0, 1, 1, 0, 1, 1, 1, 1)$, for an eight bit-stream. As can be noted, the order in which the ones are fixed in the stream does not matter, as long as in the whole bit-stream the ones and

zeros proportion keeps the same. This property makes the SC methodology noise-tolerant. The main source of noise in electronic devices comes from electromagnetic waves, which produces an effect of flipping one or more bits in the bus. This effect could be catastrophic in TC, due to the probabilities of flipping a weighted bit. Suppose a flip in the most significant bit of the data. This causes a relative error of $2^{N-1}/2^N = 1/2$ (where N is the bit-stream length), which in most of the applications is prohibit. Now, suppose the same happens in an SC system. The relative error becomes just $1/2^N$, which is quite small in comparison, and gets smaller as N grows.

Another worth noting advantage of SC is what has been called the *early termination* property [18]. In order to achieve the same accuracy as in TC, SC needs 2^N cycles to operate, adding an exponential latency to the system as the bit resolution grows. But depending on the application requirements and the quality of the bit-streams generated, the time to achieve the desired level of accuracy can be reduced, allowing to terminate earlier the computation (fewer cycles). For this to be accomplished, random number generators with low discrepancy sequences are commonly used [19], as will be studied later on in Chapter 3. Nevertheless, the advantage of TC is that the available information (I, that would be related to the precision of the operations) is coded with the minimum number of data bits (N_b), so that the ratio I/N_b is always maximum and equal to 1 (where $I \equiv -log_2(p)$, being p the probability of any feasible category included in the codification, assumed all to be equally probable). In the case of SC, its codification has a poor efficiency rate since $I/N_b = log_2(1+N_b)/N_b$. This rate vanishes to zero as soon as N_b is increased. As a counterpart, the poor codification efficiency of SC is compensated with a more simple hardware compared with TC.

Let us define the hardware packing efficiency as the ratio of N_b with respect to the expected number of gates used in a specific operation. A comparison between both methodologies is shown in Table 2.1, for the special case of the multiplication operation. Given that more gates are needed when constructing a TC-binary multiplier than on a SC one (both in a parallel way), the hardware efficiency is always 1 for SC, while for TC vanishes to zero as N_b is increased. For the selection of the best computing technique for a specific computation, we can define the Information Processing Efficiency (or computing efficiency CE) as the ratio between the information to be processed (related to the precision required in the operation) and the number of gates needed (last row of Table 2.1). For a better understanding of these results, we can plot the ratio of both efficiencies (CE_{SC}/CE_{TC}), as a function

of the required precision (see Fig. 2.1). As it can be appreciated, SC provides much better efficiency than TC for low-precision operations (less than 9 bits). For the case of pattern recognition, in which a relatively small set of categories must be distinguished, it is well known that the precision used in the operations is not a critical requirement. Therefore, it is expected that SC should provide a greater processing efficiency than TC for deep learning applications.

Table 2.1: Comparison between stochastic and traditional computing techniques for the product operation. The first row represents the number of logic gates needed per operation to obtain N_b coded bits. The second row is the information I contained in each coded bit. Third row is the coded bits processed per gate. The last row is the information I processed per gate.

Metric	**Equation**	**Stochastic Computing**	**Traditional Computing**
Gates	$\#Gates$	N_b	$6N_b^2 - 8N_b$
Codification efficiency	I/N_b	$\frac{log_2(1+N_b)}{N_b}$	1
Hardware efficiency	$N_b/\#Gates$	1	$\frac{1}{6N_b-8}$
Information efficiency	$I/\#Gates$	$\frac{I}{2^I-1}$	$\frac{1}{6I-8}$

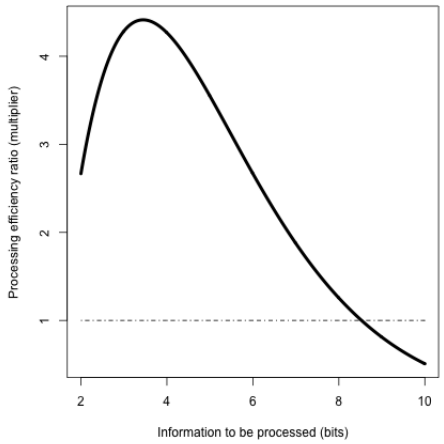


Figure 2.1: Information processing efficiency ratio between SC and TC as a function of the number of bits to be processed for the case of the product operation.

2.1.1 Stochastic Computing Codifications

The probabilistic interpretation of SC only allows to represent positive numbers (between 0 and 1) and is normally known as *unipolar* codification. However, it is possible to represent negative values (between -1 and 1, known as *bipolar* codification) by just using a variable change from the *unipolar* codification. In *bipolar* codification, the number of zeros (FALSE state) are subtracted from the number of ones (TRUE state), and finally divided by the total number of bits in the stream, giving: $p^* = (N_1 - N_0)/(N_0 + N_1)$, where N_0 is the number of zeros, N_1 is the number of ones, and the * symbol denotes *bipolar* codification. The variable change from the *unipolar* coding (p) to obtain *bipolar* coding (p^*) would be: $p^* = 2p - 1$.

One of the main advantages of using SC is the low cost in hardware resources of implementing complex functions. Take for instance the multiplication operation, which is implemented in SC using just a single logic gate: an AND gate for *unipolar* codification, or an XNOR gate for *bipolar* codification. Fig. 2.2 shows how the multiplication can be obtained using different logic gates for the two stochastic logic codifications. As can be noted in Fig. 2.2(b) and Fig. 2.2(d), the input bit-stream pattern for the gates is the same, but the value represented is not. In the case of x, the value for the bit stream is 0.5, unlike x^*, which is 0.0. That comes from scaling the *unipolar* stochastic number x to its *bipolar* representation $x^* = 2x - 1$. Note that the output bit-stream pattern z and z^* are different, representing properly the product between its inputs.

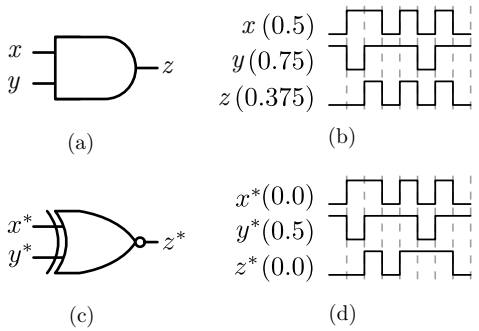


Figure 2.2: Difference in stochastic multiplication using different codification techniques with the same input bit-streams: (a) *unipolar* multiplication gate, (b) time diagram for *unipolar* multiplication circuit, (c) *bipolar* multiplication gate, (d) time diagram for *bipolar* multiplication circuit.

2.1.2 Conversion Circuits

In order to operate in stochastic domain, a converter circuit must be implemented from the X binary value (from now, binary data will be represented in capital letters) to its stochastic representation x (stochastic signals will be represented in lowercase letters). It is important to mention that the X binary value will represent a *fraction* of the space representation (maximum value X can take in the interval). So for the case of Fig. 2.3, which space representation is defined as 2^b (where b is the bit precision), X represents the fraction $4/8 = 0.5$, for an evaluation period of $N = 8$ clock cycles.

The most commonly used circuit in the literature [13, 20, 21, 22] is based on a single comparator and a Random Number Generator (RNG). Each TC value X is converted to its serialized stochastic counterpart $x(t)$ by using a RNG ($R(t)$ in Fig. 2.3). Whenever X is greater than $R(t)$, the output $x(t)$ is set to '1', otherwise, is set to '0'. If the RNG $R(t)$ is uniformly distributed in the interval of all possible values of X, the probability x of the stochastic signal $x(t)$ is proportional to X. To recover the X value, a digital counter is incremented every high pulse of the bit-stream during the evaluation period. Fig. 2.3 shows a binary to stochastic conversion, and a recovery of the binary signal for an eight bit-stream. As can be noted, the probability $x = P(x(t) = 1) = 4/8$ is proportional to the X value since $R(t)$ is uniformly distributed for *all* of the possible X values. The register is triggered at the 9'th clock cycle, giving the original X value in the output signal Z.

The stream length over which the evaluation is performed is related to the conversion error, so that the longer the stream, the lower the error obtained. This phenomenon could be observed in Fig. 2.4, where the Mean Square Error (MSE) is calculated for different bit-stream lengths using an eight-bit precision input ($b = 8$). As shown, the MSE for a 4 bit-stream is 6.4×10^{-2}, whereas for a 4092 bit-stream is 5.8×10^{-5}, a thousand times more accurate. This is due to only a subset of the probabilistic real numbers can be represented exactly in SC. For instance, a *unipolar* N bit-stream is only able to represent the exact values : $[\frac{0}{N}, \frac{1}{N}, \frac{2}{N}, ...\frac{N}{N}]$. Nevertheless, this feature of SC can be exploited since the precision needed can be adjusted without hardware modifications. Thus, if the application does not require high precision (as is the case of pattern recognition systems), we can compute in less time, decreasing latency and power.

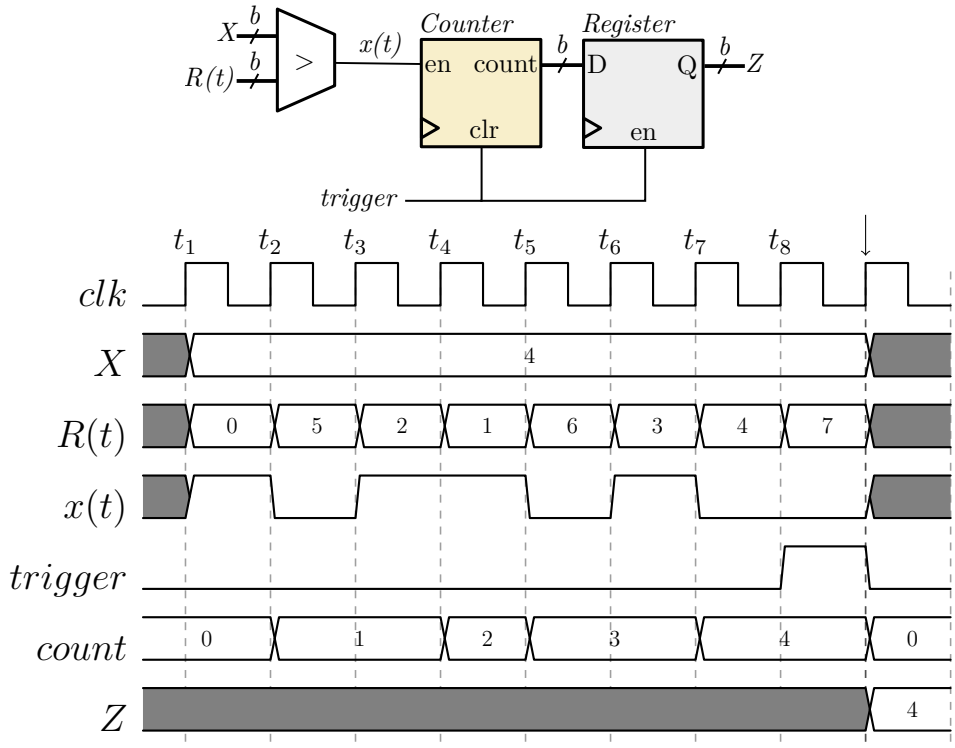


Figure 2.3: Conversion circuits between TC and SC for a bit-stream of $N = 8$, using a $b = 3$ bit precision. After conversion, X value is recovered at the output Z.

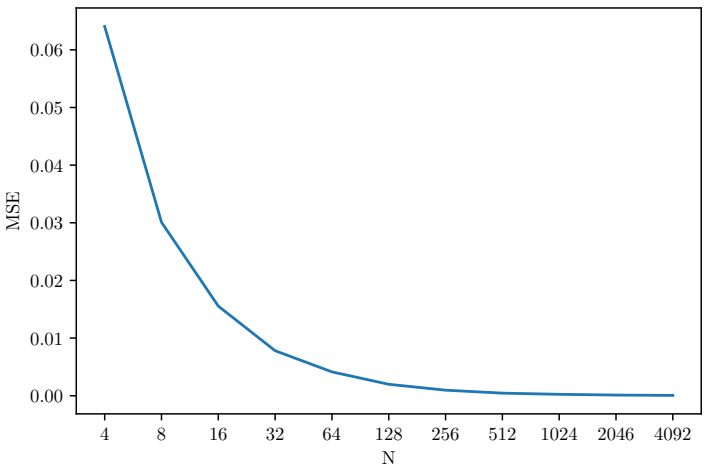


Figure 2.4: MSE for different bit-stream lengths using the conversion circuit of Fig. 2.3 with an $x = 0.5$.

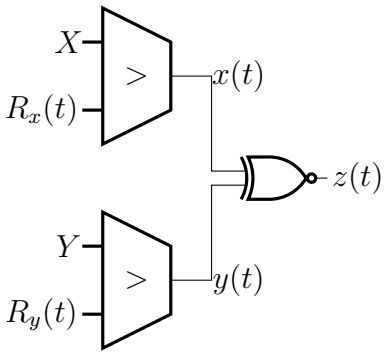
2.2 Stochastic Correlation

In the context of stochastic computing, the correlation is said to be the statistical similarity between two or more bit-streams [23]. Formally, the normalized stochastic correlation of two bit-streams codified in *bipolar* can be expressed as:

$$C(x^*, y^*) = \frac{Cov\big(x(t), y(t)\big)}{1 - \mid x^* - y^* \mid - x^* y^*}, \tag{2.1}$$

where the function Cov is the covariance between the two time-dependent stochastic signals $x(t)$ and $y(t)$ using *bipolar* coding, while parameters x^*, y^* are their averaged values (bounded between -1 and +1). The case $C(x^*, y^*) = +1$ implies maximum correlation, produced when both signals are generated from the same RNG; whereas $C(x^*, y^*) = 0$ implies a minimum correlation, produced when both signals are generated from two independent RNGs.

Most of the errors produced by SC systems come from operating the stochastic signals with an undesired degree of correlation. Many researchers have tried to avoid the stochastic correlated imprecision by generating all the stochastic bit-streams with independent RNGs. Unfortunately, this approach employs a high amount of hardware resources in the conversion cir-



z^*	
$R_x \neq R_y$	$R_x = R_y$
x^*y^*	$1 - \lvert x^* - y^* \rvert$

Figure 2.5: Correlation impact over stochastic operations. Correlation between signals changes the operation computed by the logic gate. Stochastic signals $x(t)$ and $y(t)$ are said to be completely correlated when they share the same RNG ($R_x = R_y$), producing the function $1 - |x^* - y^*|$; otherwise, if $R_x \neq R_y$, they are said to be uncorrelated and the output function is different: x^*y^*.

cuits, limiting the advantage that SC offers for hardware implementations. Nevertheless, despite some operations (such as the multiplier) need the use of uncorrelated signals, there are some cases where correlation can be exploited [23]. Consider the circuit shown in Fig. 2.5. The XNOR gate, in the presence of two uncorrelated signals (produced by two different RNG, $R_x \neq R_y$), performs the product operation in *bipolar* codification (x^*y^*); but in presence of two correlated signals (produced by sharing the same RNG, $R_x(t) = R_y(t)$), performs a totally different operation ($1 - |x^* - y^*|$), which would be costly if implemented using uncorrelated signals.

For obtaining the analytical expression, we can use the diagrams shown in Fig. 2.6. When two different RNGs are used to generate bipolar signals x^* and y^*, we can use the diagram shown in Fig. 2.6(a). Since both R_y and R_x are oscillating independently, we put them in two independent axes. Then, the possible values (x^*, y^*) can be easily delimited. For the case of using an XNOR gate as in Fig. 2.5, the shadowed area of Fig. 2.6(a) represents the case when $z(t) = 1$ (when inputs are (0,0) and (1,1)), and consequently, the non-shadowed area represents the case when $z(t) = 0$. Therefore, the bipolar signal $z^* = (N_1 - N_0)/(N_1 + N_0)$ can be easily obtained estimating these areas, providing the result $z^* = x^*y^*$. For the case of complete correlation ($R_x = R_y$) we can follow a similar reasoning but using one single axis instead of two. Without loosing generality, we can draw $x' = max(x^*, y^*)$ and $y' =$

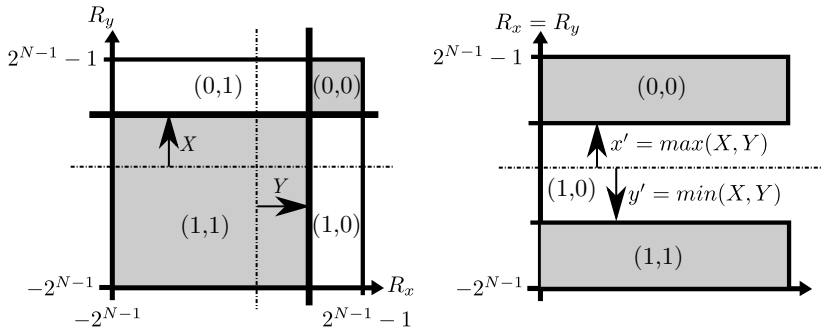


Figure 2.6: Stochastic diagrams for the estimation of the stochastic function of a logic gate depending of input signals correlation: (a) uncorrelated, and (b) completely correlated.

$min(x^*, y^*)$, and identify the different areas related to the pair of signals (x', y') as shown in Fig. 2.6(b). For the case of the example of the XNOR gate, only the shaded area will be related to a high output $(z(t) = 1)$ which corresponds to the bipolar value of $z^* = (N_1 - N_0)/(N_1 + N_0) = 1 - (x' - y') = 1 - |x^* - y^*|$. For the cases of the AND and OR gates we can follow a similar procedure, obtaining:

$$AND(x^*, y^*) = \begin{cases} \frac{1}{2}\left(x^* y^* + x^* + y^* - 1\right) & R_x \neq R_y \\ min(x^*, y^*) & R_x = R_y \end{cases} \tag{2.2}$$

$$OR(x^*, y^*) = \begin{cases} \frac{1}{2}\left(1 + x^* + y^* - x^* y^*\right) & R_x \neq R_y \\ max(x^*, y^*) & R_x = R_y \end{cases} \tag{2.3}$$

Once we obtain the analytical expressions when the signals are totally correlated and uncorrelated, we can express the output of any combinational gate as a function of the correlation between its inputs. So for the case of the AND, OR and XNOR gates in *bipolar* coding, we have:

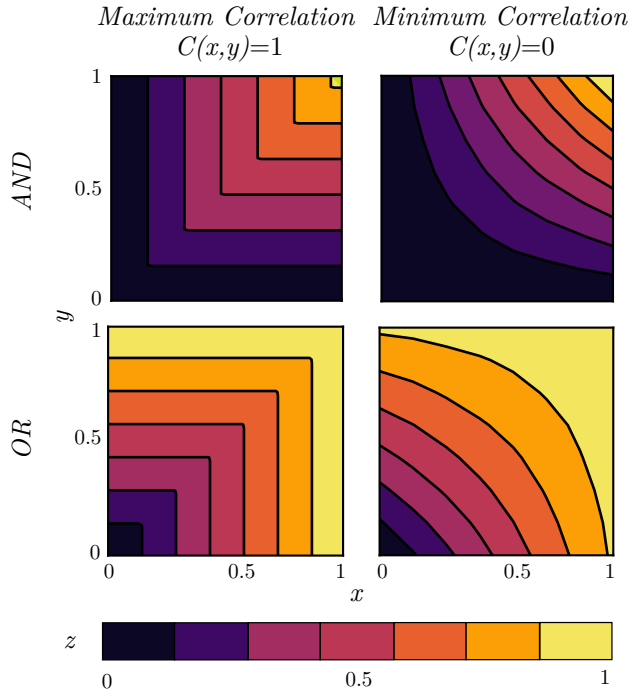


Figure 2.7: FPGA outcome difference when operating with the maximum and minimum correlation inputs for the AND and OR gates, using eight-bit precision in the *unipolar* coding.

$$\begin{aligned} AND(x^*,y^*) = \;& \tfrac{1}{2}\big(x^*y^* + x^* + y^* - 1\big)\big(1 - C\left(x^*,y^*\right)\big) \\ & + min\left(x^*,y^*\right) C\left(x^*,y^*\right) \\ OR(x^*,y^*) = \;& \tfrac{1}{2}\big(1 + x^* + y^* - x^*y^*\big)\big(1 - C\left(x^*,y^*\right)\big) \\ & + max\left(x^*,y^*\right) C\left(x^*,y^*\right) \\ XNOR(x^*,y^*) = \;& x^*y^*\big(1 - C\left(x^*,y^*\right)\big) \\ & + \big(1 - \mid x^* - y^* \mid\big) C\left(x^*,y^*\right) \end{aligned} \tag{2.4}$$

Fig. 2.7 shows the FPGA outcome when operating with the maximum correlation $C\left(x,y\right) = 1$ and minimum correlation $C\left(x,y\right) = 0$ for the AND and OR gates, using eight-bit precision for the *unipolar* coding. For the case of the *unipolar* code, the stochastic correlation is defined as $C(x,y) = Cov(x,y)/(min(x,y) - xy)$. As can be appreciated, we can have totally different results depending on the correlation level of the input signals. The case of the correlated OR gate circuit is of special interest for the implementation

of the ReLU transfer function used in neural networks, and the Max-Pooling block used in convolutional neural networks, as Chapter 5 explains in depth.

2.3 Stochastic Operations

Since the formulation of SC in the earliest 60's, different stochastic circuits have been presented for the implementation of common mathematical operations. Multiplication schemes were firstly presented by Gaines [13], using single AND gates for *unipolar* codification, and single XNOR gates for *bipolar* codification. In his relevant work, he comprises the square function, the addition, the integration, the division, and the square root function. Many of his contributions keep being the state of the art for stochastic circuits nowadays. More recently, T. Chen and J. Hayes proposed a novel division technique called CORDIV, which exploits correlation among inputs [24]. Kim et al. proposed an Approximate Accumulative Parallel Counter (AAPC) [25] to accomplish the addition operation minimizing the resources employed. Lunglmayr et al. proposed a novel shift-register-based architecture for a stochastic max/min function [26]. And V. Canals [27] presented different stochastic circuit implementations for different mathematical operations such as: the inverse operation $f^{-1}(p)$, the exponential negative function $e^{-k \cdot p}$, and the Gaussian function, among others.

This section discusses the basic operations for neural network applications in stochastic computing such as: the multiplication, addition and the argmax.

2.3.1 Stochastic Multiplication

Stochastic circuit multiplication is the best example of the saving resource contribution by stochastic computing logic (see Table 2.1). Gaines introduced the basic circuit in his work [13] for the two more used codifications : *unipolar* and *bipolar*. We will briefly discuss the two main circuits used in the literature to accomplish the multiplication operation.

The AND Gate

As discussed previously, the AND gate performs the product operation if the inputs are uncorrelated and the *unipolar* coding is chosen. Consider the circuit shown in Fig. 2.2(a), where the input bit-streams are $x(t)$, $y(t)$, with

probabilities x, and y respectively. All bit-streams denote numerical values in the range $[0, 1]$, corresponding to the probability of finding a high logic in the whole bit-stream. Assuming $x(t)$ and $y(t)$ are uncorrelated, we have:

$$\begin{aligned} z &= P(z(t) = 1) \\ &= P(x(t) = 1 \text{ and } y(t) = 1) \\ &= P(x(t) = 1)P(y(t) = 1) \\ &= xy \end{aligned} \tag{2.5}$$

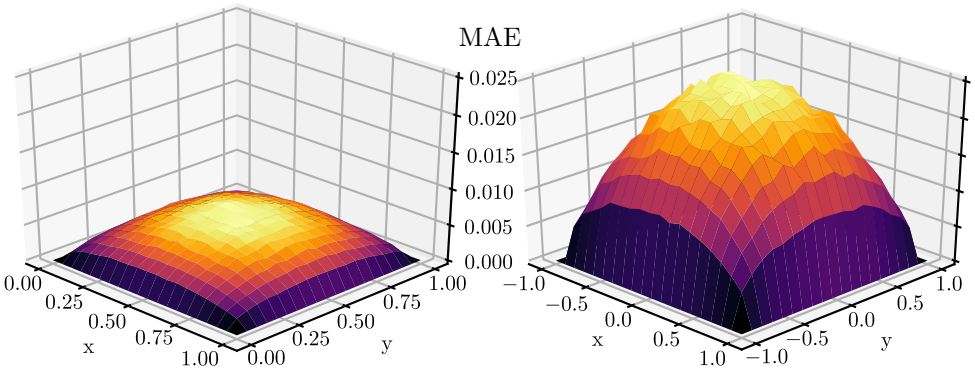
The XNOR Gate

For the case of the *bipolar* coding multiplication, the circuit employed is the XNOR gate. Using the notation of Fig. 2.2(b), and considering the change of variable $p^* = 2p - 1$, where p is the *unipolar* representation of the bit-stream, and p^* is the *bipolar*, we have:

$$\begin{aligned} z &= P(z(t) = 1) \\ &= P(x(t) = 1)P(y(t) = 1) + P(x(t) = 0)P(y(t) = 0) \\ &= xy + (1 - x)(1 - y) \\ \frac{z^* + 1}{2} &= \frac{x^* + 1}{2} \cdot \frac{y^* + 1}{2} + (1 - \frac{x^* + 1}{2})(1 - \frac{y^* + 1}{2}) \\ z^* &= x^* y^* \end{aligned} \tag{2.6}$$

Comparison

Fig. 2.8 shows a comparison of the Mean Absolute Error (MAE) for the two different multiplication circuits presented. The error produced by the binary to stochastic conversion is omitted; only the error introduced by the stochastic operation is considered. For each point, an averaged over 1000 iterations has been done, using a 1024 bit-stream length. As can be noted, the peak error occurs when the input signals are in the maximum variability: when both signals are 0.5 in the *unipolar* coding, and 0 for the *bipolar* coding, both at the center. It is worth noting that the error measured by the *bipolar* multiplication case is almost 4 times higher than that of the *unipolar* one. This phenomenom can be explained by the implicit lower precision in the *bipolar* coding, by reason of the scaling factor in the conversion ($\frac{p^*-1}{2}$).



(a) AND gate error (b) XNOR gate error

Figure 2.8: Mean absolute error comparison for the two different stochastic multiplication circuits. XNOR *bipolar* gate has 4 times higher error than the AND *unipolar* gate.

As Table 2.2 exposes, doubling the bit-stream length decreases the multiplication error by 25%. A latency and energy cost have to be assumed if precision is the target. Comparing the *bipolar* multiplication with the *unipolar* one, *bipolar* needs 16 times longer bit-streams in order to equate the results of the *unipolar* coding. Take for instance the 64 bit-stream measure, where AND gate has 13.1×10^{-3} MAE; the XNOR gate needs a bit-stream of 1024 cycles to take similar results.

Table 2.2: Stochastic multiplication error comparison for different bit-stream lengths (N). The comparison is based on the averaged MAE, the standard deviation and the maximum error.

N	**Avg MAE**($\times 10^{-3}$)		**std**($\times 10^{-3}$)		**max**($\times 10^{-3}$)	
	AND	**XNOR**	**AND**	**XNOR**	**AND**	**XNOR**
32	17.3	69.1	12.3	49.1	36.0	144.0
64	13.1	52.4	8.1	32.4	25.5	102.0
128	9.4	37.6	5.7	22.9	18.3	73.2
256	6.7	26.8	4.0	16.1	13.1	52.3
1024	3.4	13.4	2.0	8.0	6.4	25.4

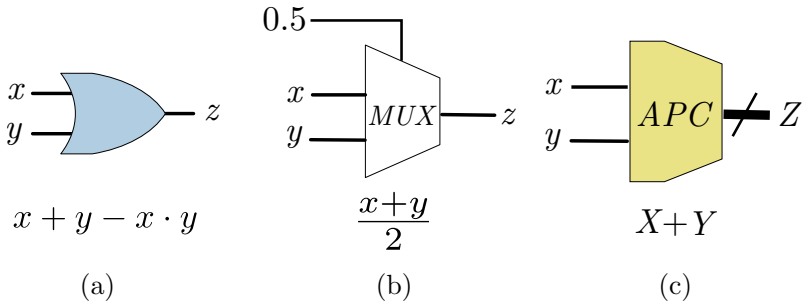


Figure 2.9: Stochastic addition circuits: (a) Stochastic addition using an OR gate, where $x \cdot y$ must be close to zero in order to compute the addition accurately. (b) Stochastic scaled addition using a multiplexer, where the accuracy outcome is dependent on the number of inputs. (c) Stochastic addition using an Accumulative Parallel Counter (APC), where the accuracy is not degraded.

2.3.2 Stochastic Addition

Contrary to common-sense expectations, addition has been one of the most challenging circuits to implement in SC. The fact resides in that the result must fit in the boundaries of the representation range: [0,1] for unipolar and [-1,1] for bipolar coding. Therefore, an exact addition for all input range is unfeasible, forcing a weighted sum ($kp_1 + (1-k)p_2$, where $0 <= k <= 1$) circuit to be designed. Different circuits in the literature have been proposed, but in this section we will focus on three of the more used: The OR gate, the multiplexer circuit (MUX), and the Accumulative Parallel Counter (APC) (see Fig. 2.9).

The OR Gate

The OR gate is the most basic circuit one can imagine it could implement the addition operation. It certainly does, but with big limitations. Examine the Fig. 2.9(a), where a 2-input OR gate is depicted. Assuming unipolar stochastic inputs $x(t)$ and $y(t)$ are uncorrelated, we have:

$$
\begin{aligned}
z &= P(z(t) = 1) \\
&= P(x(t) = 1 \text{ or } y(t) = 1) \\
&= P(x(t) = 1) + P(y(t) = 1) - P(x(t) = 1)P(y(t) = 1) \\
&= x + y - xy
\end{aligned}
\tag{2.7}
$$

As shown, in order to get an accurate addition, the probabilities x and y must be close to zero to get rid of the xy factor. Moreover, the OR gate circuit is high sensitive to correlation between its inputs. As explained later, all these considerations make the use of OR gate practically null for almost any application, despite it proposes the smaller and less power approach.

The MUX Circuit

The multiplexer is one of the most popular circuits to achieve the addition in both unipolar and bipolar codification cases. The circuit is low cost in terms of area and the precision is not affected by the correlation among its inputs. Unlike the OR gate, the MUX circuit has a third signal to take into account (the selector), which must be uncorrelated respect to the input signals. This selector signal contributes drastically to the output result of the operation, being mandatory that its probability value be proportional to the number of inputs. This effect is necessary in order to select all possible input signals with the same proportion of time in the evaluation period. Thus considering the circuit of Fig. 2.9(b), where the selector input has a constant value of 0.5, we have:

$$
\begin{aligned}
z &= P(z(t) = 1) \\
&= P(s(t) = 1)P(x(t) = 1) + P(s(t) = 0)P(y(t) = 1) \\
&= \frac{1}{2}P(x(t) = 1) + \frac{1}{2}P(y(t) = 1) \\
&= \frac{x + y}{2}
\end{aligned}
\tag{2.8}
$$

As shown, the MUX circuit realizes the weighted sum, averaging the input signals with the selector signal s. The problem is that it requires an extra stochastic signal for each addition in the circuit, a shortcoming for the design, considering it must be uncorrelated with the input signals. Moreover, as we

will see later on, it demands a small number of inputs and long periods of N to compute accurately.

The APC Circuit

The APC (Fig 2.9(c)) calculates the stochastic addition by adding and accumulating the ones of each stochastic input stream for the whole evaluation period N. This produces an output size of $k = log_2(N(i+1))$, where i is the number of input bit-streams. The whole circuit is composed of a Parallel Counter (PC) unit and an accumulator circuit (see Fig. 2.10).

As detailed in [28], many methods have been proposed to implement the PC. Some of them use techniques such as: two-level gate network, ROM memory, and a Full Adder (FA) network. Fig. 2.10 shows a 15-input FA network PC, formed by 2 PC blocks of 7 inputs and a standard 3-bit adder. The 7-input PC block consists of a 7-line input FA network tree that produces a 3-bit output result (7,3) [29]. This circuit is the base unit block for building higher input PC (as shown in the figure to build a (15,4) block). As shown, the inputs of the (7,3) PC are grouped in a set of 3 lines each (for each FA) and the remainder input is connected to the second layer of the network to complete the whole operation. In this way, the whole number of FA are efficiently employed. To hold this optimization, the number of inputs must be fixed to:

$$i = 2^k - 1 \tag{2.9}$$

Therefore, if a higher number of inputs have to be operated, we add as many (7,3) blocks as required. If an arbitrary number of inputs needs to be added, we can ground the remainder non-used inputs (see the 15th input signal in Fig. 2.10).

The number of FAs needed to implement a k output PC is calculated taking into account the line reduction a FA produces: from three lines to two lines. In this way, we can express the number of FAs as:

$$\mathrm{FA}_{Total} = i - k \tag{2.10}$$

Assuming the delay of one single FA as the base unit delay δ, we can calculate the delay path for the different PC inputs. As shown in Fig. 2.10, the result of the first layer FA-tree is ready after one δ. The second layer output will be ready after the first layer output is stable, and the carry output of the FAs from the current layer are set. This adds an extra of 2δ in the

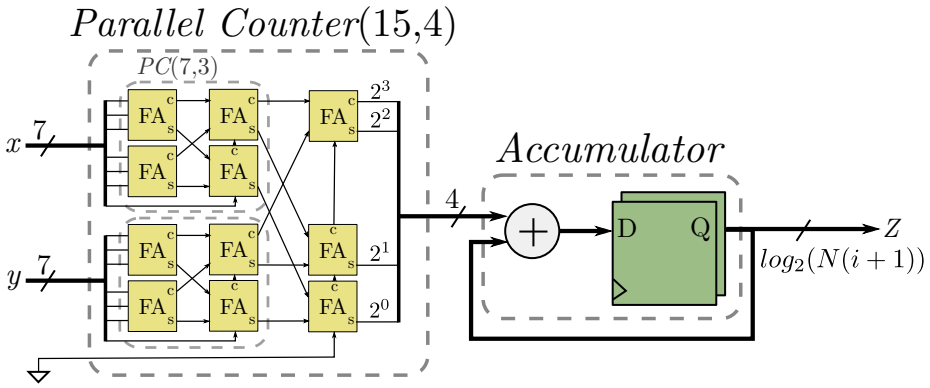


Figure 2.10: 15-input Accumulative Parallel Counter (APC) block. The output binary signal size is dependant of the number of inputs i and the evaluation period N.

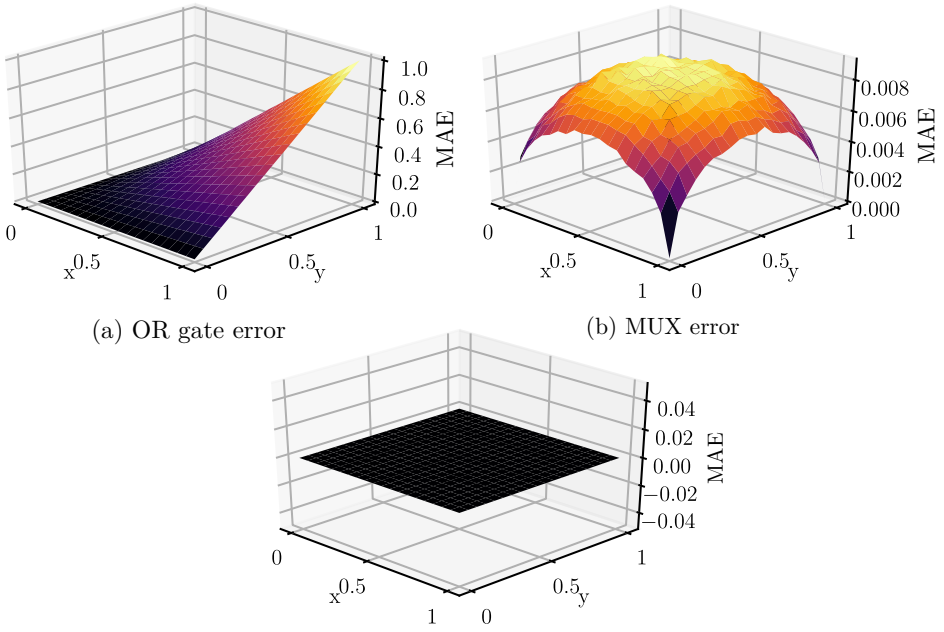
path for every layer. Considering this analysis, we can express the delay for the least significant bit of the PC output as $(k-1)\delta$. For the worst case, which is that of the most significant bit, we have a delay of:

$$\delta_{max} = (2k-3)\delta \tag{2.11}$$

The great advantage provided by the APC circuit is that all inputs with high value are counted up for the whole evaluation time, consequently, producing error-free results. Nevertheless, the APC circuit is the most area-hungry, and the longer path-delay approach for the cases studied in this section, as Eq. 2.10 and 2.11 demonstrate.

Comparison

The MAE for the different stochastic addition circuits explained previously are shown in Fig. 2.11. The MAE values are calculated for a 1024 bit-stream length. As it can be seen, the OR gate approach has the highest inaccuracy for all instances. As the xy factor gets higher, the error increases considerably. When both inputs are in the max value, it shows an error of 1, which clearly makes it useless for practically any application. Unless we can guarantee the xy factor is small, it is not advisable to use the OR gate for adding in stochastic computing. On the other hand, the MUX circuit presents smaller error and different behaviour than that of the OR gate. Its peak value occurs when both inputs are in the center of the graph (when the



(a) OR gate error

(b) MUX error

(c) APC error

Figure 2.11: Mean absolute error comparison for the three different stochastic addition circuits. OR gate (a) needs input signals close to zero in order to be accurate. MUX circuit (b) presents higher error when its inputs are in the middle. APC circuit (c) has no implicit error in its calculus.

signals fluctuates the most). As the inputs tend to go to the extremes the error decreases to 0. Finally, as explained before, the APC circuit presents no error in the operation. It is shown only for comparison purposes.

Table 2.3 shows different statistics errors for the OR gate and the MUX circuit varying the bit-stream length. The OR gate demonstrates that as the bit-stream length increases the error decreasing is poor. MUX circuit by its part, it is 36 times more accurate than the OR gate for a 1024 bit-stream. When bit-stream length doubles the error decreases in 30%. MUX approach is considered the most used solution for addition, by virtue of its small footprint and low error property. The main disadvantage MUX presents is what is called the *vanishing phenomenon*. This is that as the number of inputs grow, the output signal vanishes, tending to be null. Therefore, we need to increase the evaluation period N in order to see an accurate portion of every signal, thus producing higher latency in the system.

Table 2.3: Error comparison for the OR gate and the MUX circuit, using different bit-stream lengths (N). The comparison is based on the averaged MAE, the standard deviation and the maximum error.

N	**Avg MAE** ($\times 10^{-3}$)		**std**($\times 10^{-3}$)		**max**($\times 10^{-3}$)	
	OR	**MUX**	**OR**	**MUX**	**OR**	**MUX**
32	279.8	37.6	250.7	9.8	1000	52.4
64	264.5	27.0	242.1	6.3	1000	37.3
128	257.2	19.2	238.3	4.4	1000	25.5
256	253.5	13.6	236.0	3.1	1000	18.3
1024	250.9	6.8	234.1	1.5	1000	9.1

2.3.3 Stochastic Maximum Function

As described previously, SC offers an area reduction for some complex operations such as addition and multiplication. However, some operations employed in applications such as neural networks do not benefit from this peculiarity. That is the case of the maximum function. The max function returns the highest value from a list of input arguments, executing the function: $max(x_1, x_2, ..., x_n)$. In TC, this is not an issue, since the instantaneous input values can be accessed at any time. But in the case of SC, we have to integrate the N cycles of the evaluation period in order to calculate the exact value of the input streams. That brings a significant extra delay, therefore, increasing the energy consumption. Different techniques have been purposed in literature with the purpose of solving this matter. In this section, we will see the relevant approaches purposed in the literature for overcoming the max function problem, and finally, we will introduce how the correlation phenomenon can be exploited to implement the maximum function in the most efficient way: using a single gate.

Max Function Based on Segment Slicing

As previously mentioned, in SC domain, we can calculate the maximum value among different bit-streams only after counting the total number of ones in the stream (considering uncorrelated signals). This is done by employing high-cost digital counters for each input argument, which in turn, produces an increment in the area footprint. For alleviating the cost, Ao Ren et.al [30] present a hardware-oriented Max function block founded on bit-stream segment slicing. the concept is shown in Fig. 2.12. The process is as follows.

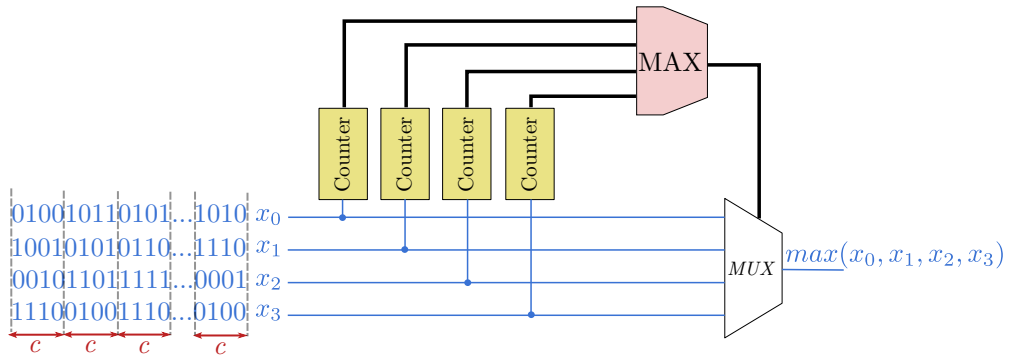


Figure 2.12: Max function circuit based on segment slicing. The whole evaluation period N is divided in k segments of c bit length. Blue lines represent SC signals, whereas black ones represent binary TC.

We divide the total evaluation period (N cycles) in k segments of time, and integrate (count-up) each bit-stream only for that slice of time. The highest bit-stream value evaluated on the sliced segment $c = N/k$ has the highest probability to be the global maximum bit-stream (considering all ones are randomly distributed in the bit-streams). Once the maximum is determined, we select it as the most probable output of the circuit for the next c-bit sliced segment. To simply put, the previous segment selects the current segment chosen as output. In the case of the first segment, the bit-stream is selected randomly. This approach introduces some accuracy loss depending on the slice size, therefore, we can not select a slice c so short, to keep an acceptable accuracy in the system. The bigger the c slice, the more accurate will be the operation, but at the expense of longer latency.

Max Function Based on Tanh-FSM

Another interesting approach to calculate the maximum function between uncorrelated bit-streams was proposed by Joonsang Yu et.al, [31]. They present a SC Max function circuit based on a Finite-State-Machine (FSM), a multiplexer (MUX), and an OR gate (see Fig. 2.13(b)). Their design is an improving circuit of the work presented in [32], which occupies a high area due to RNG circuits (about 48% of the circuit area, using linear-feedback-shift-registers). With the improvement suggested, the reduction in area is up to 52%. The basic concept is based on an FSM, that calculates the *tanh* function over a stochastic bit-stream (as explained in [33]), but only when the

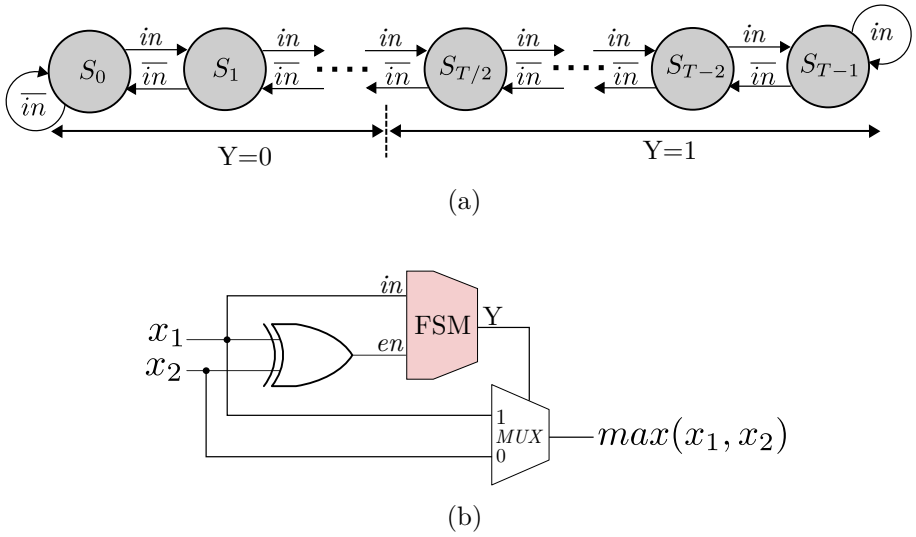


Figure 2.13: Stochastic max function circuit based on a *tanh* FSM. The state diagram of the FSM is shown in (a). The complete circuit implementation is depicted in (b).

two input signals are different. Fig. 2.13(a) shows the state diagram for the FSM implementing the *tanh* function. The differentiation is accomplished by an XNOR gate, which enables the FSM to advance the current state, otherwise, the state is held. The effect is that the FSM tends to be on the high state ($Y = 1$) when x_1 is bigger than x_2 (that is when x_1 is 1, and x_2 is 0). Contrary, the FSM stays in the low state ($Y = 0$). Finally, the MUX selects the x_1 signal, if the FSM is in the high state side (higher than half of the total states T), or x_2 signal, if the FSM is in the low state side. This operation produces the final effect of seeing the higher value signal at the MUX output. This solution, despite is smaller than that of the segment slicing, produces a dependence between precision and the number of states of the FSM, which in turn, it represents a significant area augmenting if accuracy is demanded in the application. Moreover, the FSM complexity increases as the number of input grows in the operation.

Max Function Based on OR Gate

As explained in the foregoing approaches, the dilemma of the stochastic max function implementation is due to the value uncertainty of uncorrelated bit-

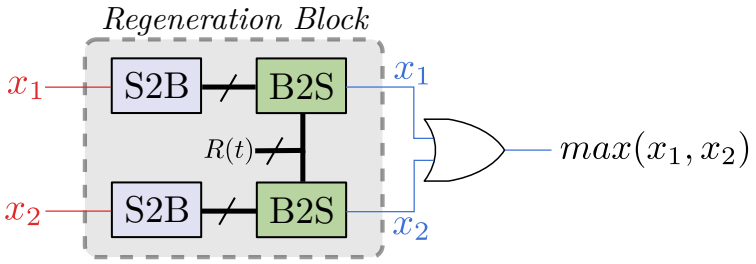


Figure 2.14: Regeneration of uncorrelated signals to compute stochastic maximum function using a single OR gate. Uncorrelated signals are denoted in red, and totally correlated in blue.

streams. This issue has led designers to propose high-area solutions that are not totally efficient, overshadowing the advantages offered by SC. Nevertheless, as previously mentioned in Section 2.2, we can take advantage of the correlation property of SC, and correlate the input signals of the function to produce the expected result. As demonstrated in Eq. 2.3, when we use the same RNG for converting signals from binary to stochastic, a single OR gate is enough to implement the max function without the need of integrating for a period of time. That means we can have the exact result with no clock delay and with no error introduced. We will show in the next section that this approach is the most efficient in terms of latency, accuracy and area. But consequently, it has the difficulty to force the input signals to be correlated. Dependant on the application, this could be an issue, considering that perfect correlation on signals only can be guaranteed during the binary to stochastic conversion phase. Therefore, *if coming from uncorrelated signals*, a regeneration circuit must be implemented [34] (see Fig. 2.14, where S2B and B2S blocks denote stochastic-to-binary and binary-to-stochastic converters respectively) every time we need to operate the maximum function, incurring in an extra area, latency and power investment. But otherwise (if the signals come from correlated sources), we can have all the benefits previously mentioned.

Comparison

The experimental measures of the MAE for the different SC-Max function circuits referred in this section are showed in Table 2.4 and Fig. 2.15. The OR gate circuit results are omitted as the operation is totally accurate, that is,

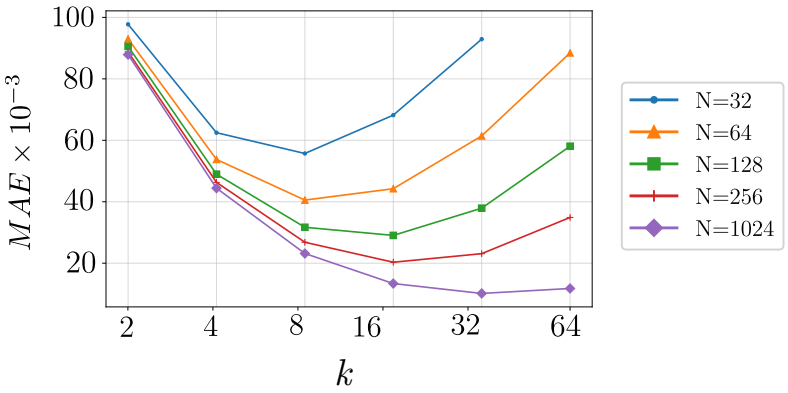
for all cases its MAE is 0. We used an 8-bit precision for the conversions and 1000 iterations for each point calculated. For the case of the slice segment method, different number of slices k are taken into account for different lengths of bit-streams N. Similar case is carried out with the tanh-FSM approach, varying the number of states T. As can be seen, the tanh-FSM approach performs better than the slice segment method for all cases. Take for instance the 32 bit-stream length case (first row), where for a $k = 32$ segments (that is a slice of 1 bit) the MAE is 92.91×10^{-3}; whereas for the similar setting in tanh-FSM approach, where the number of states T is 1, the MAE is $59.19 \times 10{-}3$. That is 1.56 times more accurate, with practically the same hardware used (a counter of 1-bit). And as the value of N increases, the difference in the MAE between the two solutions rises exponentially.

Fig. 2.15 plots the Table 2.4 values for the different cases. One might expect that as the number of slices k increases in the slice-segment method, the error decreases for all values of k. However, as shown in Fig. 2.15(a), there exists an optimum value of k for which it performs the best, which is settled around $2^{(log_2(N)+1)/2}$. So as we move away from this center point, the error increases exponentially. In contrast, the tanh-FSM solution presents an exponential decrease of MAE as we add more states T to the FSM. Once in the the stabilisation point (which again is around $2^{(log_2(N)+1)/2}$), the error decreasing is nearly imperceptible. In general, we can see that for most applications, an 8-state FSM can make the deal.

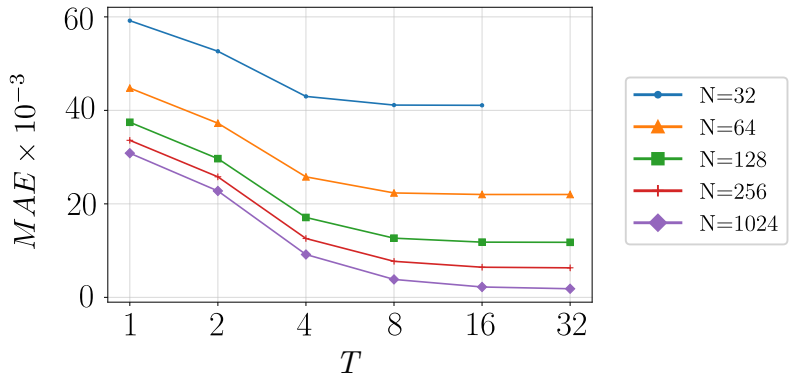
So definitely, the tanh-FSM is the better solution if we want to calculate the maximum function in stochastic computing for uncorrelated bit-streams. They are cheaper in area and more accurate. Nevertheless, if the signals are correlated, the OR gate keeps being the preferred approach, considering the precision and the small area employed.

Table 2.4: MAE for slice segment method and tanh-FSM method. The MAE presented is scaled by 10^3.

N	**Slice Segment (k)**						**tanh-FSM (T)**					
	2	**4**	**8**	**16**	**32**	**64**	**1**	**2**	**4**	**8**	**16**	**32**
32	97,75	62,48	55,71	68,19	92,91	–	59,19	52,64	42,99	41,13	41,08	41,08
64	93,01	53,77	40,54	44,25	61,42	88,45	44,77	37,25	25,78	22,33	22,00	22,00
128	90,58	48,98	31,69	29,05	37,91	58,07	37,46	29,67	17,09	12,66	11,81	11,77
256	88,77	46,31	26,82	20,32	23,11	34,89	33,60	25,78	12,59	7,70	6,44	6,31
1024	87,89	44,43	23,16	13,38	10,16	11,78	30,84	22,76	9,17	3,83	2,22	1,84



(a) MAE for SC-Max function using slice segment method.



(b) MAE for SC-Max function using tanh-FSM method.

Figure 2.15: MAE for different SC-Max function circuits. The slice method (a) is measured for different number of segments k. The tanh-FSM method (b) is measured for different number of states T.

2.4 Summary

In this chapter, we analysed the SC implementation of the main operations needed for ML and DL applications. We explained what SC is and how to convert from the traditional binary logic to the stochastic one. The correlation effect was studied and explained, showing the drawbacks of its existence. Moreover, we show some positive effects of the correlation in specific circuits, which in turn, can be leveraged to produce exact results on some op-

erations such as the maximum function. Correlation phenomenon is always a headache for SC designers. But if handling wisely, we can produce better outcomes in some operations than with uncorrelated signals. In addition, we showed different ways in which state-of-the-art approximate the main functions required for ML and DL, such as the multiplication, the addition and the maximum function. Finally, we compared the different circuit proposals for such functions and presented the advantages and disadvantages with the aim of helping SC designers to improve their circuits.

Chapter 3

SELECTING A RNG - A BEST CASE STUDY

3.1 Introduction

In section 2.1.2, we introduced briefly the most commonly used circuit in literature for operating in Stochastic Computing (SC) domain. As mentioned therein, the circuit performance is highly dependant on the Random Number Generator (RNG). The reasons are twofold. Firstly, the area employed for this part of the system is way higher than that of the computational part, making them to occupy around 80% or even 90% of the total footprint of the system [35]. This is especially critical for applications requiring a high fan-in. But secondly, the randomness quality can significantly affect the precision of those operations requiring non-correlated signals, such as the stochastic multiplication. For these reasons, finding the best RNG in terms of area and low correlation is a major concern to address when designing real SC applications.

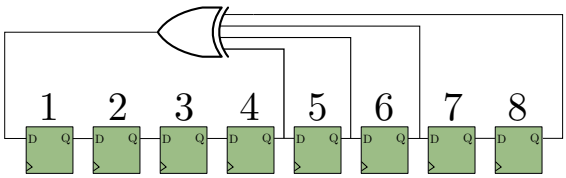
Different approaches have been proposed to tackle these issues. Low-discrepancy sequences such as Halton [36], or Sobol sequences [19] deal with the low-correlation matter. These sort of sequences produce pulse signals with ones and zeroes *uniformly* spaced, which mitigates the random fluctuations in the bit-streams (BSs) generated. The problem they face is the area employed for generating such sequences. Normally, different base counters [36] or least significant zero detectors plus storage arrays [19, 37] are utilized, thus increasing the hardware overhead. On the other hand, area

saving mainly comes in a single flavor: the Linear-Feedback-Shift-Register (LFSR) circuit.

An LFSR is a circuit based on a shift register and a linear function of its previous state connected to its input. Normally, the linear function is produced by connecting exclusive OR gates to different points (known as taps) in the state registers. The way of connecting these taps in the circuit is known as *primitive polynomials*, which can be expressed using two different notations: polynomial or binary notation. In polynomial notation the tap connections are expressed as $1 + \sum_{i \in [T]} x^i$, where T are the register taps selected. In binary notation, the taps connected are expressed as $\sum_{i \in [T]} 2^{i-1}$. Fig. 3.1 shows an LFSR circuit where the inputs of the XOR gate are connected to registers $8, 6, 5$ and 4; its polynomial is thereby expressed as $x^8 + x^6 + x^5 + x^4 + 1$ or 184 in binary notation. Different polynomials allow different deterministic streams with different lengths to be produced. There exists a finite number of polynomial configurations (depending on the LFSR resolution) that produce a maximal length sequence of $N = 2^b - 1$ cycles, where b is the number of registers used in the circuit (bit-resolution). For instance, a 10-bit LFSR has at least 60 different polynomials that produce maximal length sequences [38]. Those polynomials which generate a maximal stream length are known as primitive polynomials. The reason the LFSR sequence has one cycle missed is because there is a prohibitive state, where the LFSR is locked-up in case it enters (when all bit registers are zeros in case XOR gate is used).

The starting value of the LFSR is named as the *seed*. Fig. 3.1 shows an LFSR sequence with a seed set to 255. After N clock cycles, the sequence will roll over again from the starting seed point. This is one of the reasons LFSRs are not truly RNGs; they are, rather, pseudo-random numbers, since we can predict the next state of the sequence if we know the current state (something uncertain in a true RNG).

LFSR is the easiest and smallest circuit to produce pseudo-RNG (from now referred as RNG; we explicitly use the adjective *True* when meaning True random), albeit presenting a high correlation behavior [39]. Different works have been proposed in the literature for exploiting the advantages of LFSR while mitigating its shortcomings. Basically, most of the works presented focus on sharing the same LFSR between different Stochastic Number Generators (SNGs) but alleviating the correlation effects that this method raises. The work carried out by Z. Li et al. [40] is a good example of this; their approach is based on a DFF insertion technique (adding a DFF to



clock cycles	LFSR value
0	255
1	254
2	252
...	...
254	127
255	255
256	254
257	252

Figure 3.1: 8-bit LFSR circuit with a primitive polynomial configuration equal to $x^8 + x^6 + x^5 + x^4 + 1$. Every $N = 255$ cycles, the sequence is repeated periodically, starting from the seed value (blue color).

the line to uncorrelate the BS with itself), where the DFF circuit aims to uncorrelate one of the BS from the other. This technique performs successfully when employing a True Random Number Generator (TRNG), but for stand-alone LFSRs this is not the case. The LFSR shows a high level of autocorrelation if only isolated with a single DFF, as demonstrated in [39]; this approach is therefore not efficient. Hideyuki Ichihara et al. [41] suggested a technique for sharing as many LFSRs as possible by employing a circular shift at the LFSR output. This method allows generating two non-correlated signals with no hardware overhead; therefore, reducing the area employed and dealing with the error at the same time. Along the same line, another relevant study was proposed by H. Joe and Y. Kim [42]. They dug deeper into the different ways of connecting the same LFSR to produce the smallest correlation impact between signals using a wire exchanging technique. Their results showed better performance in comparison with other LFSR sharing methods. Another approach, introduced by F. Neugebauer et al. [39], tries to increase the randomness of the LFSR outcome by adding a nonlinear boolean circuit. This method decreases the correlation impact at

the cost of a hardware overhead.

J.Anderson et al. [43] explored an interesting approach based on the effect of seeds over the accuracy of SC systems when employing LFSRs. They demonstrated that an efficient selection of seeds in the circuit improves the accuracy. For this, the authors explored the whole space to find the best seeding set; something affordable as long as the exploration space is bounded (small bit-precision). However, for more complex analysis cases (as more complex systems or higher bit-precision), a higher-level procedure must be employed (such as meta-heuristic techniques [44, 45]). Nevertheless, as previously introduced, SC advantages are displayed when small bit-precision is used (≤ 8 bits, for the case of multiplication). Therefore, the whole exploration process is reasonable and no need of higher-level heuristic is demanded.

Despite the fact that some solutions have come about, none of them guarantees good accuracy and small footprint area at the same time for correlation-sensitive circuits such as the SC quadratic function, the scaled addition, and the multiplication. These circuits are the driving force in the SC realm, and high demand applications like image processing or DL employ them.

In this section, we explore in more detail how the LFSR circuit could be better exploited as a RNG source in SC by making a careful selection of the seeds employed, with the purpose of finding the best BS generator technique for different application requirements.

3.2 Seeding Impact on Correlation

LFSRs are valuable in different applications such as fast digital counters [46], whitening sequences [47], cryptography [48], circuit testing [49] and indeed, SC circuits. Despite its advantages, the use of LFSR sequences for SNG raises some difficulties. Unlike in the case of TRNGs, in that of LFSRs, the premise that all bits in the stochastic stream are independent of each other does not apply anymore. Consecutive values of the LFSR sequence are highly dependent, as they are a shifted version of past states. Each bit in the sequence, if taken separately, possesses good randomness characteristics, but when it is seen as a whole binary number (as it is normally used in SC), the ideal randomness quality disappears. This is a real issue for commonly used stochastic functions such as the quadratic function x^2, which is carried out with a single AND gate (*unipolar* coding) and a single D-FF register.

The purpose of the D-FF is to uncorrelate the stochastic signal from itself by adding a delay cycle. This is true for stochastic signals generated by TRNGs, but it is not fulfilled for the LFSR case. Moreover, when operating multiple BS, different seed combinations produce different results, and since the sequence is periodic, the same error is observed in each cycle. This contrasts with the TRNG, in which the error fluctuates, converging to zero as the integration time increases. For these reasons, finding optimum seed combinations is a major issue when employing LFSRs as the random number source of the circuit.

Despite the fact that LFSR has been the preferred RNG in SC real implementations, previous works do not provide a careful analysis of the LFSR seeding; still less do they offer a method to efficiently choose the best seeds to operate. The work that comes closest to doing so is that carried out by J.Anderson et al. [43]. They explore if there exists a suitable set of seeds for improving the accuracy of stochastic computing systems. They demonstrate that a good selection of seeds increases the accuracy of SC circuits for the same bit precision, and that shorter streams with optimum seeding have the same or better accuracy than longer bit streams with random seeding. Nevertheless, although they present empirical evidence for their results, the authors do not provide what those seed combinations are and how to select them *a-priori*, i.e, without iterating the whole *seed sweeping* computation (Monte-Carlo way) for the application. One of the problems with their method is that we need to do a trial and error procedure: a quick task if small circuits are evaluated, but a heavy task for a big implementation. In this section, an analysis of the seeds of the LFSR is presented, the aim being to overcome the aforementioned shortcomings.

Suppose two BSs are generated from two independent LFSRs with the same polynomial, but having different seeds. Suppose these two BSs are multiplied. The question that arises is: does any different couple of seeds generate the same result? and if not, how does the seeding affect the overall outcome? Take for instance the operation shown in Fig. 3.2. The x signal represents the value 4/15, while y represents 5/15. The same LFSR polynomial is used to generate each signal but with different seeds. Two versions of y (y_1 and y_2) are generated for comparison purposes. For the y_1, we picked the next value in the sequence of the LFSR_x (see LFSR_{y1} with the seed highlighted in red) as the seed. For y_2, we picked the fourth value in the sequence of the LFSR_x (see LFSR_{y2} with the seed highlighted in green). As seen, due to the fact that the LFSRs have the same polynomial, the y_1

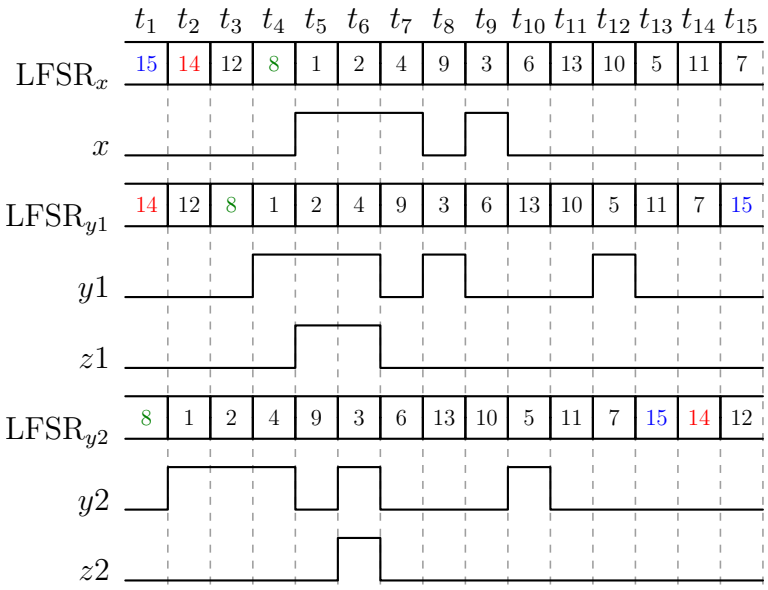


Figure 3.2: Impact of seeding for stochastic multiplication ($x \times y$, where $x = 4/15$ and $y = 5/15$). Depending on the seed used to generate the stochastic BS (y_1 or y_2), different results are produced (z_1 or z_2).

stream is a shifted version of y_2. This makes the x signal 1's match the y_1 and y_2 1's in different times, producing different outcomes in the final operation. In order to see the impact of seeding, the AND operation between x and y_1, y_2 is presented in z_1, z_2, respectively. For the case of $z_1 = x \wedge y_1$, the result is $2/15 \approx 0.13$, whereas for $z_2 = x \wedge y_2$, the result is $1/15 \approx 0.06$. As shown, z_2 represents more accurately the ideal value (≈ 0.08), with an absolute error of ≈ 0.02. This fact shows that y_2 is less correlated than y_1, respect to x. This short and simple example demonstrates how seeding has an impact on the overall results in SC correlation sensitive circuits (as is the case of multiplication), so careful selection of seeds must be carried out to get the most accurate results.

Expanding the preceding example, Fig. 3.3 shows the Mean Absolute Error (MAE) for different seeds, when x and y are multiplied considering all possible values. Once again, the same polynomial is employed for the two LFSRs. Two different bit resolutions are taken into account in the analysis: 6-bit and 8-bit. Instead of doing the analysis by taking as reference the seed values as in the previous example, this time we took the difference between

the seed position (*seed index*) in the sequence $\Delta idx = idx_{\text{SEEDy}} - idx_{\text{SEEDx}}$. Taking Fig. 3.2 as an example, x is generated with the seed index 0, y_1 is generated with the seed index 1, and y_2 is generated with the seed index 3. In essence, the seed index corresponds to the value taken by the LFSR sequence at time $t_{index+1}$. Formally, the MAE for every seed index is calculated as:

$$MAE = \frac{1}{N}\sum_{j}|y_j - \hat{y}_j| = \frac{\sum_{X=0}^{2^b-1}\sum_{Y=0}^{2^b-1}|x \wedge y - \bar{x}\bar{y}|}{2^{2b}}, \tag{3.1}$$

where b is the bit-precision, x and y are the BS generated when converting X and Y to SC domain, and $\bar{x}$, $\bar{y}$ the expected value of x and y, respectively.

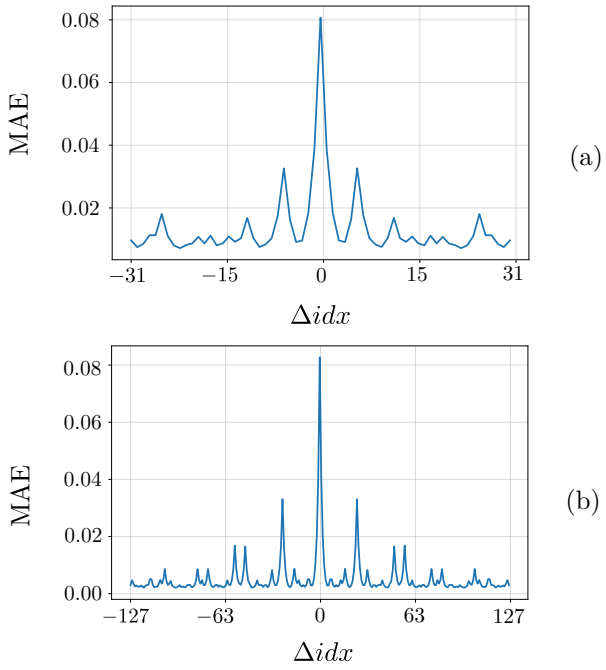


Figure 3.3: MAE for different seed indexes measured from a 6-bit LFSR (a), and an 8-bit LFSR (b), for the stochastic multiplication. LFSR polynomials are shown in Appendix Table A.2.

As shown in both plots, the maximum error occurs when x, and y are generated with the same seed ($\Delta idx = 0$). However, as the idx_{SEEDy} moves away from the idx_{SEEDx}, the error tends to diminish (with some resonant error peaks throughout), until we get to the *further* seed index $\Delta idx = \pm(2^{b-1}-1)$.

The behaviour can be seen as a mirror if we take the center as the point of reference. The takeaway from these figures is that *the difference between both seed indexes* $|\Delta idx|$ *is the real issue, not the LFSR seed values.*

It is worth noting that as we move away from $\Delta index = 0$, some seed indexes present an error resonant behaviour (high peaks in the plots); but as we get closer to the further index, the peaks are mitigated in an exponential way. The reason for this phenomenon can be better understood by observing Fig. 3.4. The blue line shows the normalized value of the 8-bit LFSR sequence for the first 127 cycles. The orange, in turn, shows the MAE for the first 127 seed indexes. Since both variables share the same axis ($idx = cycles$), we can plot them in the same figure. As shown, the LFSR sequence presents similar patterns periodically (see arrows in the plot). Considering that the LFSRx seed is at $idx = 0$, if the LFSRy seed coincides with one of these initial-pattern values, then a resonant error occurs, indicating that both sequences (the one starting with seed index 0, and the one starting with the initial-pattern) have a high degree of correlation, i.e, they are similar. Therefore, if noncorrelated operations are to take place (as is the case of the multiplication), it is mandatory to avoid these seed values for the generation of the second BS.

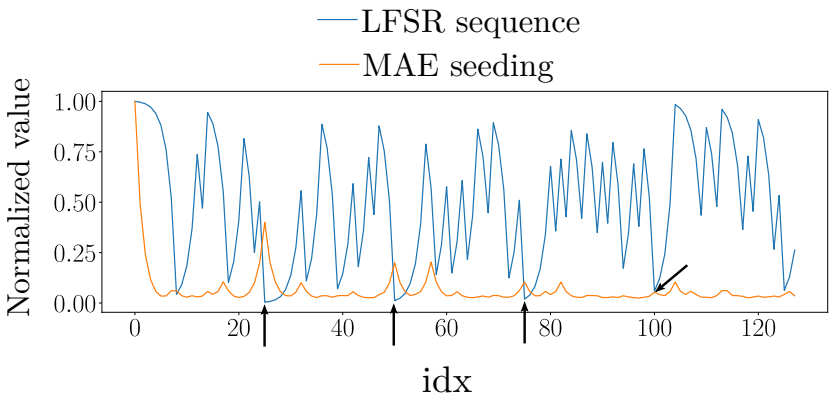


Figure 3.4: Resonant seeds (marked by arrows) produced by values which are more correlated respect to the first seed index. There exist some correlated patterns in the LFSR sequence (blue line) which produce high correlation between stochastic signals, thereby inducing higher MAE (orange line).

Fig. 3.5 shows the MAE histogram for the 8-bit LFSR and a 10-bit LFSR implementation. As can be appreciated from the LFSR-8 instance,

selecting random seeds can lead us to have more than twice the error than if the seeds are selected intentionally. According to the measurements, there exists a 79% probability of choosing an inaccurate seed (seeds with a MAE greater than 0.002 in Fig. 3.5 for the LFSR-8 case). Moreover, 90% of the LFSR-10 seeds produce an MAE of less than 0.002, which is the same MAE we obtain when the seeds are efficiently chosen for the LFSR-8. We can thereby achieve similar accuracy if the pairing seed selection is carried out deliberately for lower resolution LFSR, instead of doing it in a random way for higher resolution LFSR, saving hardware resources, latency and power.

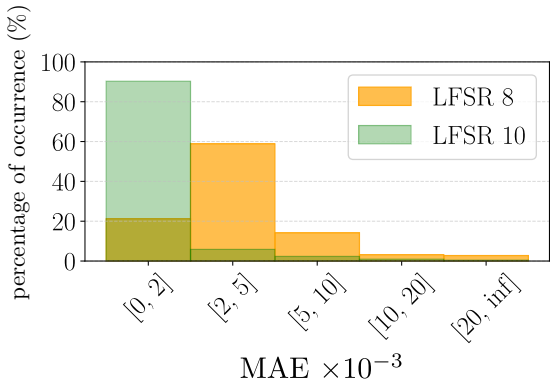


Figure 3.5: MAE histogram for the 8-bit, and 10-bit LFSR examples.

The Absolute Error (AE) for different seed indexes when varying x and y are presented in Fig. 3.6. The worst case (Fig. 3.6(a)) occurs when we generate the y BS with seed index 0, producing maximum correlation between both input BSs. As seen, the maximum error is produced around the center, when the variance of the signal is maximum ($x = y = 0.5$), taking maximum error values of up to 0.25. The second plot (Fig. 3.6(b)), shows the error behaviour for the first resonant peak of Fig. 3.4 (seed index 25). On this occasion, the maximum error is spotted when one of the signals is 0.5 and the other is at 0.25 and 0.75, raising error values of up to 0.12, almost half of the worst case. The best seed (index = 97) is presented in Fig. 3.6(c), where the maximum error rises to no more than 0.011, an order of magnitude less than the first resonant peak. Finally, the further seed index (idx = 127) is shown in Fig. 3.6(d). As can be seen, its behavior is very similar to the best seed index case (Fig. 3.6(c)), presenting a maximum value of 0.016; 0.005 higher than the best seed case. As shown, the outcome pattern varies depending on the seed employed; the seeding effect is therefore a major concern when

utilizing LFSRs as the random source for generating stochastic BSs.

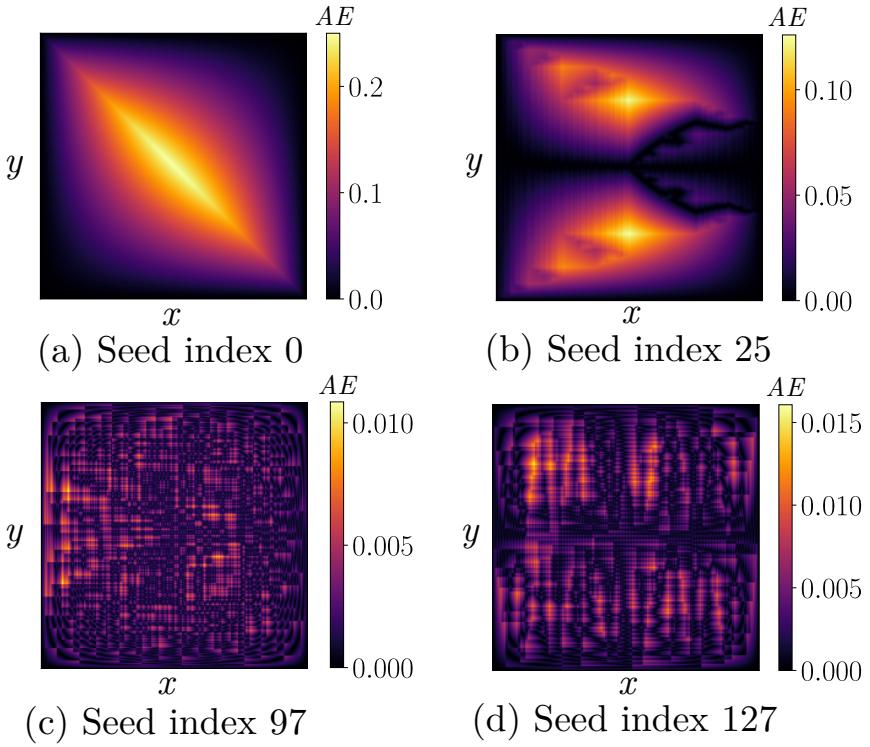


Figure 3.6: AE in the multiplication when varying x, y (from 0 to 1) using different seeds in the 8-bit LFSR example. Seed index = 0, which is the worst case, is shown in (a). Seed index = 25, which is one of the resonant error peaks, is shown in (b). Seed index = 97, which is the best case, is shown in (c). And seed index = 127, which is the further seed index is shown in (d).

3.3 Seeding Impact on Autocorrelation

The study of LFSR seeding can be extended to an essential area of research in SC: The autocorrelation. A BS is said to be non-autocorrelated when there exists a low dependency between its consequent bits. When autocorrelation occurs, an isolator circuit could be used to uncorrelate the BS with itself, allowing operations such as the stochastic square function (x^2) to be done. In other words, the autocorrelation measures how well an isolator circuit is able to uncorrelate the BS with a delayed version of itself [39]. The most

common circuit employed in the literature as an isolator is the DFF [13, 50]. Inserting a DFF in the BS line uncorrelates the BS with itself or with another BS *generated with the same RNG*, as explained by Z. Li et al. [40]. This is something especially exploited in their work in the quest to reduce the area overhead produced by the RNG circuits, since inserting a single DFF in the line is much more simple than inserting a complete independent RNG. Nevertheless, although they claim that their technique works for any BS generated with a circuit generator structure made of any RNG (LFSR method included) and a comparator, the truth is that for the LFSR case, this property does not apply, as will be analyzed in this section. The LFSR circuit has a high degree of autocorrelation, as demonstrated in [39], and a single DFF isolator insertion is insufficient to uncorrelate the BS with itself.

Let us see firstly how the LFSR behaves compared to other RNG found in literature using an autocorrelation metric. F. Neugebauer et al. [39] define an autocorrelation metric based on the Box-Jenkins function [51], which is employed in the NIST Engineering Statistics Handbook [52]. The definition they provide is as follows: Let x be a BS with a sequence : $x_1, x_2, ..., x_N$, where N is the BS length; and let $\bar{x}$ be the expected value of x. The autocorrelation A_k of x will be:

$$A_k = \frac{\sum_{i=1}^{N-k}(x[i] - \bar{x})(x[i+k] - \bar{x})}{\sum_{i=1}^{N}(x[i] - \bar{x})^2}, \tag{3.2}$$

where k is the number of cycles delayed (the number of DFFs inserted in the line). Autocorrelation values close to 0 indicate a good independence of the BS with its k delayed version, whereas a high absolute value indicates a bad independence.

Fig. 3.7 shows a comparison of the autocorrelation between the LFSR for different k values and other RNG methods. The measure is taken for different x values using 8-bit precision. As seen, LFSR for $k = [1, 2]$ presents an autocorrelation value higher than that of the TRNG (TRandom). The average value of the LFSR A_1 (LFSR k=1, blue line) is 0.29, showing an increasing in A as x moves away from the middle point. That is why inserting a single DFF when using an LFSR produces poor precision results when the BS value moves away from the center point ($x = 0.5$). For the case of inserting 2 DFF in the line (LFSR k=2), the average value decreases to 0.12, but still shows high peaks of more than 0.2, performing still poorly for most applications. The ideal case, which is the TRandom, presents an average A value of -0.038, measured from 1000 different random samples (in the figure,

one of the samples chosen arbitrarily is plotted), having better autocorrelation value than the LFSR for $k = 1, 2$. This conclusion is supported by [39] in their work, discarding the LFSR standalone circuit for stochastic operations such as the quadratic function, and proposing the SBoNG method as an approach to circumvent the problem. The SBoNG method is based on connecting a nonlinear Boolean function to the output of the LFSR. The function is performed by a combinational circuit called SBox [53]. The SBox circuit is normally implemented as a Look-Up-Table (LUT) (although it can be implemented using logic gates) and is constrained to 4-bit inputs, limiting its use to 4-bit-multiple RNGs. In their paper, the authors compare the SBoNG method with the LFSR circuit, but only for k=1,2,3; concluding that SBoNG performs better for stochastic operations demanding low autocorrelated BSs. Nevertheless, as Fig. 3.7 shows, the LFSR with the precise number of delay elements ($k = 5$) performs much better than the TRandom and the SBoNG implementations, with an average A value of 0.007; 5.1, and 2.2 times better than the TRandom and the SBoNG methods, respectively. It must be said that for the case of the SBoNG measures, our results differ from the ones presented in the original paper [39]. This is because they evaluated the SBoNG circuit by varying randomly, for every iteration of the test, the LFSR seed and the initial-state of the circuit (see details in original paper). However, to conduct real digital circuit implementations without increasing the amount of resources to generate random values for every iteration, we fixed the seed and initial-state values. These values were found by running 1000 tests and selecting the best case result, i.e, the LFSR-seed and initial-state couple which performed the lowest autocorrelation average value.

The reason LFSR with 5 DFF performs better, can be understood by analyzing the MAE values of the first seed indexes in the 8-bit LFSR multiplier. Table 3.1 shows the numerical values. A k delayed version of x is equivalent to taking the seed index k (see the example of Fig. 3.2). That is why the $k = 5$ version has a low autocorrelation value, because the seed index 5 is the first minimum MAE, as can be observed in the table. Additionally, the first two seed indexes, which represent $k = 1, 2$ in Fig. 3.7, have a high MAE, corresponding to a high autocorrelation level. It is therefore expected that if we employ only one or two isolators, the LFSR will perform poorly, since it corresponds to employing the two first seed indexes for operating. However, if we embed the correct number of DFF, we can have accurate results.

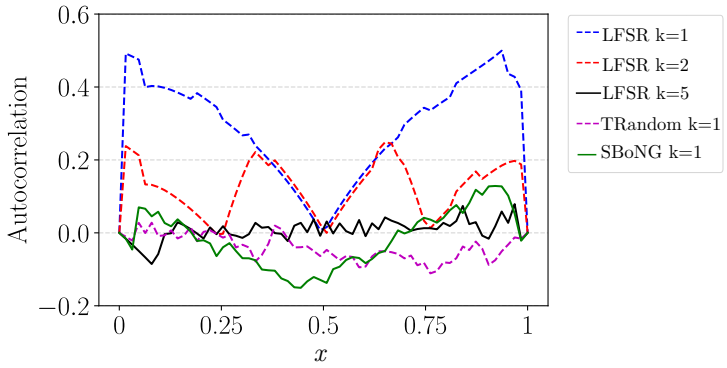


Figure 3.7: Autocorrelation comparison for the LFSR, TRNG (TRandom) and SBoNG methods when initial input x value takes different quantities.

Table 3.1: MAE for the first 9 seed indexes of the 8-bit LFSR circuit.

Seed Index	MAE
1	0.04089
2	0.02006
3	0.00972
4	0.00479
5	**0.00287**
6	0.00290
7	0.00503
8	0.00507
9	0.00299

3.4 Experimental Results

In this section, we conduct different experiments to test the LFSR seeding impact in SC applications. We first test three important operations employed in the SC realm, and which are correlation sensitive: the quadratic function, the scaled addition and the multiplication. Finally, we implement a real image processing application over an FPGA device: an edge detection circuit.

Under otherwise noted, the polynomials employed for the different bit-precision LFSR implementations are the ones described in Appendix Table A.2. We have selected these primitive polynomials arbitrarily from all possible ones, since the variation of the best seed MAE for each of them are negligible (see the MAE std column in Table 3.2).

Table 3.2: MAE statistics for the best seeds of each polynomial sequence considering 4 to 8 bit precision. MAE was calculated for the multiplication operation.

Bit precision	Possible Primitive Polynomials	MAE Avg	MAE Std ($\times 10^{-3}$)
4	[9, 12]	0.021	2.971
5	[18, 20, 23, 27, 29, 30]	0.012	0.873
6	[33, 45, 48, 51, 54, 57]	0.007	0.238
7	[65, 68, 71, 72, 78, 83, 85, 92, 95, 96, 101, 105, 106, 114, 119, 120, 123, 126]	0.004	0.141
8	[142, 149, 150, 166, 175, 177, 178, 180, 184, 195, 198, 212, 225, 231, 243, 250]	0.002	0.057

3.4.1 Quadratic Function

For comparison purposes, we evaluated an important operation in SC, namely, that of the Stochastic Quadratic Function (SQF). Its use is extended to very interesting applications such as the implementation of the SC-Gaussian function employed by V. Canals [27] for describing the probability density of a continuous random variable. Additionally, it has recently been used by J. Li et al. in the normalization block circuit for a SC neural network implementation [54]. We evaluated this function circuit as an example, in an effort to see the effect of correctly isolating the BS generated by the LFSR, and compare it with other RNG approaches. The SQF is built by a multiplication gate (AND gate for unipolar coding and XNOR for bipolar coding), and an isolation block to uncorrelate the input signal with itself, as shown in Fig. 3.8(a).

Table 3.3 presents the MAE for the *bipolar* SQF using different RNG methods proposed in literature. We vary the number of isolation elements for different bit widths, where the number next to the D termination corresponds to the number of elements employed (DFFs). The RNG methods evaluated were: the LFSR method, the Sobol sequence method [19], the SBoNG method [39], and the TRNG method (TRandom). For a fair comparison, the table presents the same number of isolation circuits for Sobol and LFSR methods, whereas for SBoNG and TRandom methods only one isolator is employed, since their best results are obtained in this manner [39]. We measured the MAE for a complete LFSR period ($2^b - 1$ cycles), where b is the bit-precision. The best result of each column is highlighted in bold. For the case of the SBoNG and TRandom methods, we followed the same experimental setting described in Section 3.3 for measuring the auto-correlation. A graphical comparison for the 8-bit precision case is shown in Fig. 3.8(b).

Observing the table and the Fig. 3.8(b), the one with the lowest precision is that of the Sobol sequence. For 8-bit precision, its MAE can rise up to

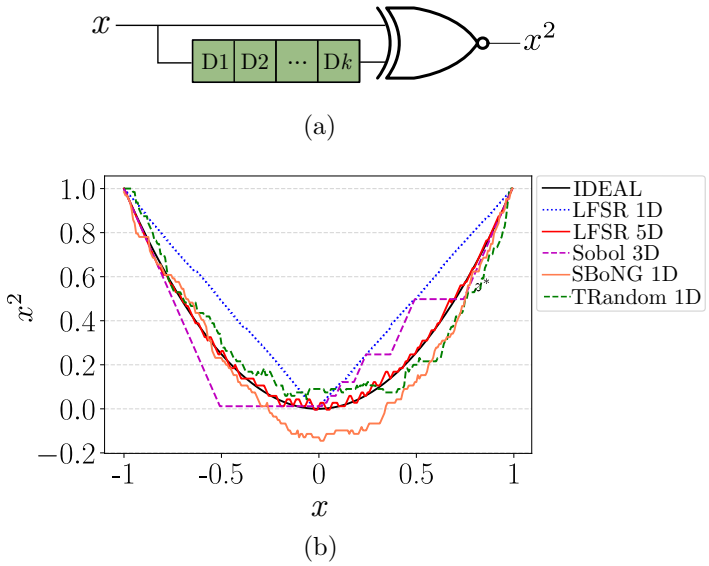


Figure 3.8: Bipolar Stochastic Quadratic Function (SQF). The circuit implementation is shown in (a), for different k DFF. The operation outcome comparison is shown in (b) for different RNGs: LFSR, Sobol, SBoNG and TRNG (TRandom).

0.415 (Sobol 8D), which is 2.5 times worse than the LFSR with only one DFF (LFSR 1D), and 36 times worse than the best LFSR case (LFSR 5D). Moreover, its best case (Sobol 3D), is worse than all of the other methods, excluding the 1 delay instance of the LFSR (LFSR 1D). Observing the plot, we see that for negative input values the behaviour is rather imprecise, tying the output to 0 for input values between -0.5 and 0, and behaving far from ideally even for the positive range. The reason for this is that Sobol sequence is a deterministic low-discrepancy sequence for which every bit in the stream is highly dependent on the past bits, leading to high autocorrelation values.

For the SBoNG method, we could measure only the MAE for 4-bit and 8-bit precision, as the method is restricted to 4-bit multiples. As regards 8-bit precision, SBoNG has similar performance to LFSR when 2 isolators are employed (LFSR 2D). This is an interesting observation which can help designers to save a good deal of resources, as 1 extra DFF (for the LFSR case), is cheaper than the whole SBoNG circuit with fix seeds. Observing its behaviour in Fig. 3.8(b), the output can take negative values when the input value is near 0. This is caused by the imprecision of the BS generated by

Table 3.3: MAE comparison for the bipolar SQF using different RNGs. The best result for each column is highlighted in bold.

RNG method	Bit precision				
	4	**5**	**6**	**7**	**8**
LFSR 1D	0.111	0.134	0.149	0.158	0.162
LFSR 2D	**0.084**	0.067	0.068	0.074	0.079
LFSR 3D	0.089	**0.047**	0.036	0.036	0.038
LFSR 4D	0.089	0.054	**0.030**	**0.022**	0.019
LFSR 5D	0.124	0.080	0.083	0.030	**0.012**
LFSR 6D	0.116	0.068	0.112	0.085	0.012
LFSR 7D	0.122	0.053	0.067	0.118	0.016
LFSR 8D	0.122	0.064	0.046	0.061	0.019
Sobol 1D [19]	0.091	0.080	0.083	0.086	0.088
Sobol 2D [19]	0.311	0.323	0.328	0.331	0.332
Sobol 3D [19]	0.100	0.084	0.083	0.085	0.087
Sobol 4D [19]	0.222	0.256	0.265	0.269	0.270
Sobol 5D [19]	0.127	0.105	0.098	0.094	0.092
Sobol 6D [19]	0.244	0.290	0.312	0.323	0.328
Sobol 7D [19]	0.202	0.121	0.096	0.090	0.089
Sobol 8D [19]	0.202	0.318	0.373	0.401	0.415
SBoNG 1D [39]	0.162	–	–	–	0.079
TRandom	0.360	0.126	0.106	0.047	0.025

this RNG, because the sequence, for a complete period of 2^b, does not cover all the possible values of x.

Finally, the LFSR method has the best performance overall when selecting the correct number of isolation elements. It is on average more than twice as good as the second best method for all bit-widths. As far as 8-bit precision is concerned, it performs 7.3, 6.6, and 2.2 times better than the best Sobol, SBoNG, and TRandom methods, respectively.

Fig. 3.9 shows the best outcomes of each method. As seen, the Sobol method barely changes its MAE for different bit widths. When $b > 6$ TRandom becomes better than Sobol, approaching LFSR performance. Therefore, if accuracy is the main requirement of our application, the LFSR remains the best option if we analyze the data for a complete BS period.

3.4.2 Scaled Addition

The stochastic addition is essential for almost any SC application. The most commonly used circuit is based on a MUX [13]. This circuit needs the input signals to be uncorrelated with the selector signal to have accurate results.

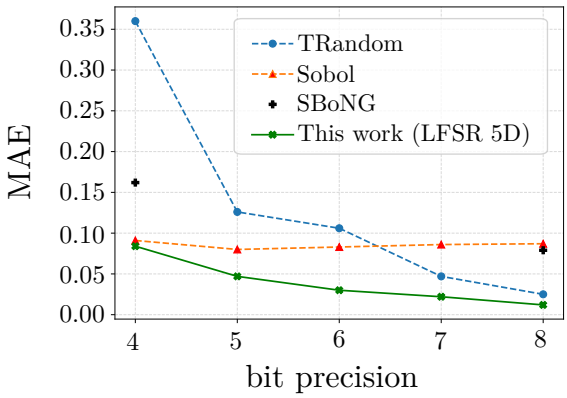


Figure 3.9: MAE for the best outcomes of each method employing the SQF.

We analyse the performance of different RNGs proposed in the literature to mitigate the correlation error of the scaled addition, and compare them with the LFSR seeding method put forward in this work.

We examined 7 different RNG methods. The first one attempts to tackle the correlation problem effect using LFSRs with different polynomials (different taps, defined here as DTaps). Ideally, we would expect that the LFSR with different polynomials would be totally uncorrelated, and thus that a precise outcome would be guaranteed. The second one is based on the DFF technique insertion proposed by Z. Li et al. [40]. In order to reduce both the computation latency and area overhead, they propose using a single LFSR to generate multiple BSs, but inserting DFFs in their lines, assuming that a single DFF will do the task of uncorrelating. The third one is the circuit suggested by H. Ichihara et al. [41], where again, it is based on the LFSR sharing technique. However, in this method, the authors suggest making a circular shift to the LFSR bit lines that are connected to one of the converters. In their experiments, they conclude that the best shuffle choice is when a circular shift of $k = b/2$ is chosen, where b is the bit precision. The next method is an improvement of the circular shift method. H. Joe and Y. Kim [42] proposed an efficient technique to permute the lines in the LFSR; they call it Wire Exchanging Technique (WET). They demonstrate that a fixed setting of permutation reduces the relative error compared with other LFSR sharing techniques, saving area and power. The WET swaps the bits by pairs and then exchanges the wires symmetrically as follows. Suppose the LFSR output bit order is $[b_{n-1}, b_{n-2}, ..., b_1, b_0]$, where b_n is the bit at position

n, being n the bit-precision. The WET is defined as:

$$Swap(b_{n-1}, b_{n-2}, ..., b_1, b_0) = b_{n-2}, b_{n-1}, ..., b_0, b_1$$
$$Exchange(b_{n-2}, b_{n-1}, ..., b_0, b_1) = b_1, b_0, ..., b_{n-1}, b_{n-2}.$$

In this manner, only using one LFSR, but permuting the wires, the correlation is mitigated. The fifth method taken into account is an approach that has aroused the interest of the SC community lately: the low-discrepancy sequence, especially the Sobol sequence [19]. Low discrepancy (LD) sequences have been commonly used in speeding-up Monte-Carlo simulations [37], but S. Liu and J. Han [19] employed it for the purpose of lowering the computing latency on SC. The LD sequences distribute the highs and lows along the BS uniformly, allowing the outcome to converge quicker to the expected result, thus lowering the latency. The Sobol sequence has been employed in different studies in the SC realm, such as in SC edge detection circuits [55], as well as in SC convolutional neural networks [56]. It is based on an address generator circuit formed by a Least Significant Zero (LSZ) detector circuit and a storage array, normally implemented as a LUT. As a sixth method, we introduce the proposed work, which consists in the stand-alone LFSR circuit with the best couple of seeds for addition (see Appendix Table A.1). Finally, a ground truth instance is added using a TRNG. Similarly to the previous experiment, this method is evaluated running 1000 different iterations.

The circuit employed for the experiment is set out in Fig. 3.10(a). All of the tests were done varying all possible binary inputs X, Y and the MAE is calculated with respect to the ideal outcome (similar to Eq. (3.1), but with the scaled addition operation).

Table 3.4 shows the MAE results for the different RNG evaluated with different bit-precision. We measured the MAE for a complete LFSR period ($2^b - 1$ cycles). The polynomials employed for the second LFSR (RNG2 in Fig. 3.10(a) were [9, 18, 33, 83 ,175] for the 4, 5, 6, 7, and 8 bit precision, respectively.

As shown, the 1-DFF insertion method [40](LFSR 1D) has the worst performance for bit widths greater than 6. The correlation introduced by inserting only one DFF impacts the scaled addition to the extent of producing an approximate constant MAE value of 0.04 for all bit-widths. As the bit precision increases, the error gap in relation to other methods grows exponentially, with a factor of 42 with respect to the best method for 8 bits. This is expected, as only 1 DFF insertion is insufficient to decorrelate two

Table 3.4: MAE for the scaled addition circuit using different RNGs. The best result for each column is highlighted in bold.

RNG method	Bit precision				
	4	**5**	**6**	**7**	**8**
LFSR 1D [40]	0.0458	0.0433	0.0424	0.0420	0.0418
TRandom	0.0964	0.0654	0.0461	0.0321	0.0226
LFSR CShift [41]	0.0250	0.0222	0.0109	0.0106	0.0053
LFSR DTaps	0.0307	0.0163	0.0102	0.0062	0.0039
LFSR WET [42]	0.0297	0.0134	0.0063	0.0031	0.0015
Sobol [19]	0.0250	0.0121	0.0060	0.0030	0.0015
This work	**0.0167**	**0.0081**	**0.0040**	**0.0020**	**0.0010**

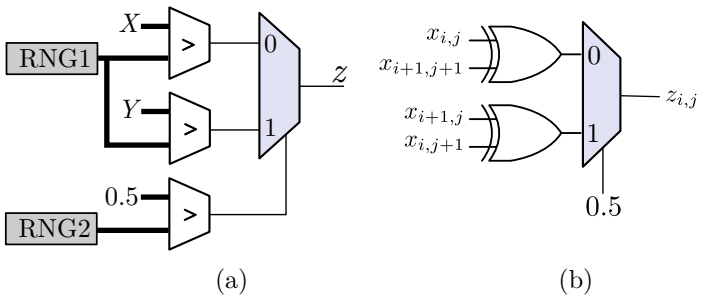


Figure 3.10: (a) Scaled addition test setting. (b) SC Roberts edge detection circuit exploiting correlation.

BS generated with the same LFSR, as discussed in Section 3.3 and verified in experiment 3.4.1.

The TRNG (TRandom) method presents the second worst performance. However, as in experiment 3.4.1, when the bit width increases, the MAE decreases exponentially.

It can be noted how the gap between LFSR-WET and Sobol methods decreases as the bit-precision gets higher.

Finally, the best-seed LFSR method (this work) outstrips all other methods for all bit-precision cases; a 1.5 factor is observed with respect to the second best method (Sobol [19]) for all the bit-widths. Contrary to intuition, using the same LFSR taps produces better results than using different taps.

3.4.3 Multiplication

In this experiment, we evaluate how the different RNGs perform for the stochastic multiplication using the same RNG conditions used in experiment

3.4.2. Table 3.5 shows the MAE comparison. The test circuit is displayed in Fig. 3.11, calculating the MAE as in Eq. (3.1). For the proposed LFSR method, the seeds employed are the best found for the multiplication operation (see Table A.1 in Appendix).

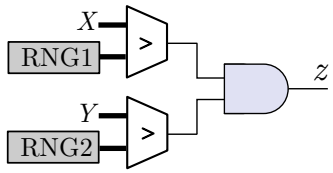


Figure 3.11: Multiplication test setting.

Table 3.5: MAE for different RNG circuits evaluated for different bit-precision in the multiplication operation. The best result of each column is highlighted in bold.

RNG method	**Bit precision**				
	4	**5**	**6**	**7**	**8**
LFSR 1D [40]	0.0330	0.0365	0.0388	0.0401	0.0409
TRandom	0.0762	0.0514	0.0356	0.0252	0.0174
LFSR CShift [41]	0.0284	0.0243	0.0172	0.0138	0.0095
LFSR DTaps	0.0217	0.0136	0.0102	0.0071	0.0046
LFSR WET [42]	0.0284	0.0177	0.0115	0.0066	0.0040
Sobol [19]	0.0255	0.0129	**0.0071**	**0.0038**	0.0022
This work	**0.0190**	**0.0108**	0.0072	0.0040	**0.0020**

Similar behavior to that obtained in experiment 3.4.2 is observed for the different methods (being ordered similarly to Table 3.4). However, for the multiplication the Sobol method outperforms LFSR-WET in all bit-precision. Moreover, it does slightly better than this work for the 6 and 7 bit-widths. Nevertheless, it only performs better by a factor of 1.01 and 1.05, respectively; whereas ours performs better by a factor of 1.3, 1.2, and 1.1 for the 4, 5 and 8 bit cases respectively.

Stochastic multiplication is the core operation for measuring the correlation between two BSs [57]. We can therefore conclude that our method produces less correlated signals than state-of-the-art RNG when analyzing a complete integration period. This claim is confirmed when implementing a real use case application where SC has been widely used, as detailed in the next experiment.

3.4.4 Edge Detection

Different image processing algorithms have been proposed in the literature where SC has been demonstrated to perform better than traditional computing with an error increase imperceptible [58, 59]. One of them is the Roberts' cross algorithm for edge detection. The algorithm computes the input image in a 2x2 moving pixel window by the following formula:

$$z_{i,j} = 0.5(|x_{i,j} - x_{i+1,j+1}| + |x_{i+1,j} - x_{i,j+1}|),$$

where $x_{i,j}$ represents the pixel value at location (i, j), and $z_{i,j}$ represents the outcome pixel value.

Fig. 3.10(b) shows the SC circuit, considering that all input BSs are correlated, and the selector signal is uncorrelated with respect to the inputs [60]. Using this circuit, we measured the MAE for the different RNG methods introduced in the preceding experiments 3.4.2, 3.4.3, except for the TRNG, since a real implementation on an FPGA is carried out. The FPGA employed is a Cyclone V 5CSEBA6U23I7 included on the DE10-Nano board from Terasic, running at 50 MHz. The input BSs is generated with the first RNG of each method, and the selector with the second RNG (see Fig. 3.10(b)).

The MAE for each method is shown in Table 3.6. A noteworthy pattern displayed is that as the bit precision gets higher, the proposed work presents a higher improvement factor than the others. For instance, for the 4-bit precision column, an improvement of 1.04, and 1.06 times is observed compared to the Sobol and LFSR-WET methods, respectively, while a 1.3 times increase is observed for the 8-bit precision.

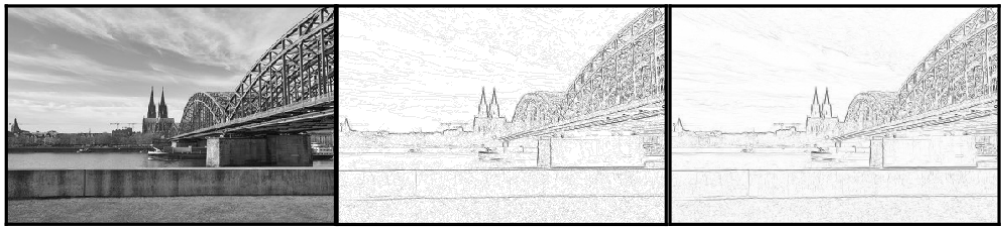
Fig. 3.12 shows the edge detection results for the 4-bit and 8-bit precision using the proposed LFSR seeding method. These results demonstrate that a good seeding in the LFSR presents the most accurate results for real image processing implementations.

3.5 Summary

We have presented a solid base for holding the LFSR as the best RNG circuit in the SC domain if computed for a complete sequence period. We have demonstrated that LFSR block is an appropriate circuitry for improving accuracy in key SC operations. If compared with other RNG methodologies applied to SC, the proposed method shows better results for different

Table 3.6: MAE for different RNG circuits evaluated for different bit-precision in the Roberts edge detection circuit. The best result of each column is highlighted in bold.

RNG method	**Bit precision**				
	4	**5**	**6**	**7**	**8**
LFSR 1D [40]	0.0262	0.0152	0.0089	0.0055	0.0034
LFSR CShift [41]	0.0261	0.0174	0.0105	0.0088	0.0053
LFSR DTaps	0.0261	0.0169	0.0100	0.0063	0.0043
LFSR WET [42]	0.0259	0.0149	0.0082	0.0043	0.0020
Sobol [19]	0.0256	0.0146	0.0080	0.0042	0.0020
This work	**0.0245**	**0.0134**	**0.0069**	**0.0035**	**0.0015**



(a) (b) (c)

Figure 3.12: Edge detection outcome using SC Roberts edge detection circuit for different bit-precision: Input image (a), output image for 4-bit precision (b), and 8-bit precision (c), using the proposed LFSR seeding method.

SC operations such as the quadratic function, the scaled addition, and the multiplication (for this last case, it presents a similar performance than the Sobol method [19]). Furthermore, the proposed method does better than other RNG performances in real case applications such as an edge detection circuit. We obtained these results using both simulations and FPGA implementations. To conclude, we offer to SC designers the guidelines for setting their LFSRs for different case applications, in this way, saving them a great deal of time, while simultaneously assuring good results.

Chapter 4

STOCHASTIC COMPUTING NEURAL NETWORK

4.1 Introduction

Neural Networks (NNs) are one of the most important areas of study in the computing research field, as they are used in many artificial intelligence and machine learning applications. Their non-linear behaviour, flexible configuration ability, and self-adaptability makes them the most common approach to different problems such as feature extraction [5], classification [3], system control [4], and machine learning [61]. They born from the idea of mimicking how the biological brain, full of interconnected neural networks, processes the information, solves complex problems, and acquires new knowledge through a training process with a low amount of energy [62]. The number of neurons only in the human cortex is counted in billions [63]. The neurons in the network are linked by inter-neuron connections, named synapses. They are in charge of passing out the information among neurons as an electrical or chemical signal. Fig. 4.1 shows a simplistic model of the neuron anatomy. As shown, the neuron has a cell body which works as a processor of the input signals received by other neurons in the network through the dendrites, which are connected to the synaptic terminals. The core produces a signal electrical stimulus based on the information processed to the following neurons connected to itself via the axon. According to Hebbian theory [64], neurons can exhibit in its synaptic connections an excitatory or inhibitory behaviour, contributing as a weighting form to the response of all of the neu-

rons connected to them. The delivered signal presents a specific non-linear function when they are activated [65].

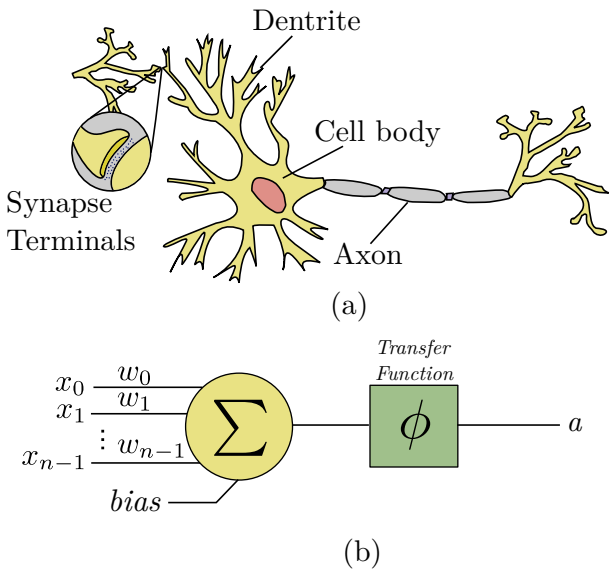


Figure 4.1: Neuron model. The real neuron (a) behaviour is modeled in (b), employing the dot product between the input x (the output activation of the previous layer a^{l-1}) and the weight synapses w for finally passing a non-linear transfer function.

Although some advances on the neuron studies were taken place, it was not until 1944, when Warren McCullough and Walter Pitts proposed the first advances on Artificial Neural Networks (ANN) modelling. They made the neuron inception model with some simple functions, mimicking the biological behaviour of the neuron. In other words, they established the base to what now is usually called the *perceptron*, albeit Rossenbelt developed the concept later on [66]. The perceptron imitates the behaviour of the biological neuron (which adds the potentials of the post-synaptic membrane in the body), by doing a dot product between the inputs and a scalar value (representing the synapse strength), referred as weight, and applying a non-linear function to the outcome (see Fig. 4.1(b)). The weights in the network are the parameters which are to be optimized in order to have a trained model, increasing or decreasing the strength of each neuron input signal (emulating the synapses force connection). Depending on the application, there are different types of neural networks. They are usually categorized depending on its neuron

connectivity pattern, being the most common : Feed Forward (FF) and the feed-backward, also known as recurrent. The FF are the simplest type of NN. They propagate only in forward direction, i.e, from the input layer to the output layer. Therefore, they do not form a loop connection as opposed to recurrent NN. For this reason, they do not allow us to implement a model with memory capacities.

The process of searching the best fitting for weights to accomplish a determined task, is known as learning. Learning involves adjusting the weights to approximate the expected outcome, thus, increasing the accuracy of the model. In FF networks, the process to do this is by minimizing the observed errors, mainly, done by an algorithm named *back-propagation*, popularized in the 80's by Rumelhart, Hinton and Williams [67]. The algorithm keeps being effectively used to train a NN through a mathematical method called *chain rule*. Simply stated, the algorithm firstly runs a forward pass through the network (evaluation), retaining the neuron activation values to adjust the model's parameters later on in a backward pass. In the backward pass, the parameters are updated by calculating the gradient of a cost function with respect to those parameters, where the gradient of the cost function $C(w_1, w_2, ..., w_m)$ at point w is a vector with the partial derivatives of C in w:

$$\frac{\delta C}{\delta w} = \left[\frac{\delta C}{\delta w_1}, \frac{\delta C}{\delta w_2}, ..., \frac{\delta C}{\delta w_m}\right]$$

This gradient tells us how much the parameter w needs to be modified to minimize C. In order to compute those gradients, a technique called *chain rule* is employed. The chain rule says that to compute the derivative of a composite function, we can multiply its partial derivatives:

$$\frac{\delta y}{\delta x} = \frac{\delta y}{\delta u}\frac{\delta u}{\delta x}$$

So for a single weight w^l_{jk}, its gradient is:

$$\frac{\delta C}{\delta w^l_{jk}} = \frac{\delta C}{\delta z^l_j}\frac{\delta z^l_j}{\delta w^l_{jk}}$$

where,

$$z^l_j = \sum_{k=1}^{m} w^l_{jk} a^{l-1}_k + \beta^l_j,$$

with m being the number of neurons of layer $l-1$, j the number of neurons in the current layer l, a the neuron activation of layer $l-1$, and β the bias. Therefore, by differentiation:

$$\frac{\delta z_j^l}{\delta w_{jk}^l} = a_k^{l-1}$$

giving the final outcome:

$$\frac{\delta C}{\delta w_{jk}^l} = a_k^{l-1} \frac{\delta C}{\delta z_j^l}$$

The gradients give us a rate of change to optimize our parameters in each iteration:

$$\Delta w_{jk}^l = \alpha \frac{\delta C}{\delta w_{jk}^l}$$

where α is referred as *learning rate*, which determines how the gradient impacts the parameter updating. This process is done iteratively, upgrading the parameter values every E passes of the network, called *epochs*, until the accuracy achieves a required threshold. We say the learning is concluded when the error rate does not decrease significantly after further samples are presented.

NNs have proved to be optimal to solve complex problems in which traditional methods found a stumble block. Their non-linearity behaviour is one of the main reasons, as complex problems convey complex non-linear information, impossible to be approached with only linear-systems. A second advantage presented is their fault tolerant. NNs are a distributed system, which means that they allow the result to be insignificantly affected in case of some of the main elements (neurons) fail. This fact is in contrast with the classical computational method, in which a minimum error in the system could cause unacceptable errors. Moreover, they offer what is commonly named adaptability. Due to their capacity to tweak the synapses parameters (weights), NNs provide a broad adaptability for different problems that can rise in the real implementation, making them the desired solution [68]. Nevertheless, the adaptability is limited. Stephen Grossberg introduced this phenomenon in his work "Adaptive Resonance Theory" [69], where he explains that as the network parameters are more dependant on the input environment changes, the system could get in an instability that makes it

useless. However, and although limited, the adaptability of such networks represents one of their highlights. But if there is one thing that stands out from NNs, is that they can *generalize* [70, 71]. After the training parameter adjustment, considering the data used for training was selected correctly, the model is able to infer unseen relationships upon unseen input data, therefore, predicting correct outcomes from data not presented in the training process. Let us present an example for this. Suppose we train the network with 10 different dog breeds, and test the model using 2 breeds not presented in the training set. If the model is able to recognize the 2 extra breeds for the testing data, then we can say that our model has a good generalization. As opposed to this, if the model can perform good with the training data, but it is not able to recognize the two extra breeds, then we say that the model has been overfitted, i.e, has a bad generalization behaviour.

The aforementioned advantages have made of NNs one of the favorite approaches for many applications in different fields. Take for instance the forecasting need in the everyday business decision (e.g, financial allocation for products, sales, stock market, and the finance general). The forecasting problems are not trivial, e.g, predicting stock prices has a a lot of underlying factors. But NNs have shown to be powerfully used in this ambit [72, 73, 74, 75], as they can approximate any continuous function. In the field of modeling, applications where the response of the system can be modeled for a predictive control [76] have been employed successfully. The medicine arena is a spotlight. NNs have contributed in different areas in this ambit. For instance, new methods for sickness diagnosis, as some heart diseases could be found [77, 78], and some tumor detection through medical images [79, 80]. In the signal processing field, we can find several applications. Due to the ability to infer complex non linear relationships among their inputs, NNs are having a huge amount of applications on the audio and image pattern recognition realm. From audio noise cancellation [81, 82, 83], speech recognition [84, 85, 86], voice detection [87, 88, 89], and computer vision [7, 90, 91], mainly employed in self-driving cars [92, 93, 94], up to unmanned aircraft systems [95, 96], NNs keep being applied to more and more different pattern recognition fields. One interesting application realm is in the organic chemistry, more specifically for constructing chemical compounds [6, 97, 98, 99, 100, 101]. Further on, in Chapter 4.5, we will show an application of stochastic computing for accelerating the inference in the construction of chemical compounds using virtual screening methods by means of NNs.

Many efforts have been made for accelerating NNs through hardware.

Basically, two main categories dominate: analog and digital implementations. On the analog side, some circuits have been proposed [102, 103, 104]. In general, analog circuits can lead to more area and power efficiency [105, 106], but the lack of flexibility, reconfigurability, automation and scalability converts them in a very challenging option to implement for real cases. That is the reason why they are primarily focused on emulating the behaviour of real biological neurons [107, 108]. Conversely, digital circuits offer flexibility as one of its leading profits [109]. Their main shortcoming is the amount of transistors required compared to their analog counterpart. Overall, the non-linear transfer function implementation is one of the causes, needing of extra area to be effective [110]. In addition, the size issue worsens as the number of inputs grow in the circuit, growing exponentially as the number of neurons increase. Here is where Stochastic Computing (SC) appears as a possible solution alternative. The high efficiency in terms of area and energy, as showed in Chapter 2, makes it ideally suited as an approach for the task of accelerating NNs [111, 112, 113, 114].

In the rest of this chapter, we explain the details of the implementation of a SC-NN, based on the more used activation function lately: the Rectifier-Linear-Unit (ReLU). Finally, a case of study is introduced, targeting the organic chemistry field.

4.2 The ReLU Neuron

As introduced previously, the command center of any neural network is the neuron. Since its inception in the 40's, different artificial neuron models have been introduced [115, 116], although the more common is the one which computes an Activation Function (AF) of its weighted inputs, so that for the i^{th} neuron, the output activation is given by:

$$a_i = \phi\Big(\sum_j \omega_{ij} x_j + \beta\Big), \tag{4.1}$$

where ω_{ij} represents the weight assigned to the j^{th} input x_j (with all inputs coming from the previous neural layer), and ϕ is the non-linear AF computed.

Different non-linear AFs have been used in literature, offering different advantages according to the case.

Let's first start with the *sigmoid* function (or Fermi function in some contexts). Fig. 4.2(a) shows the sigmoid function and its derivative. The

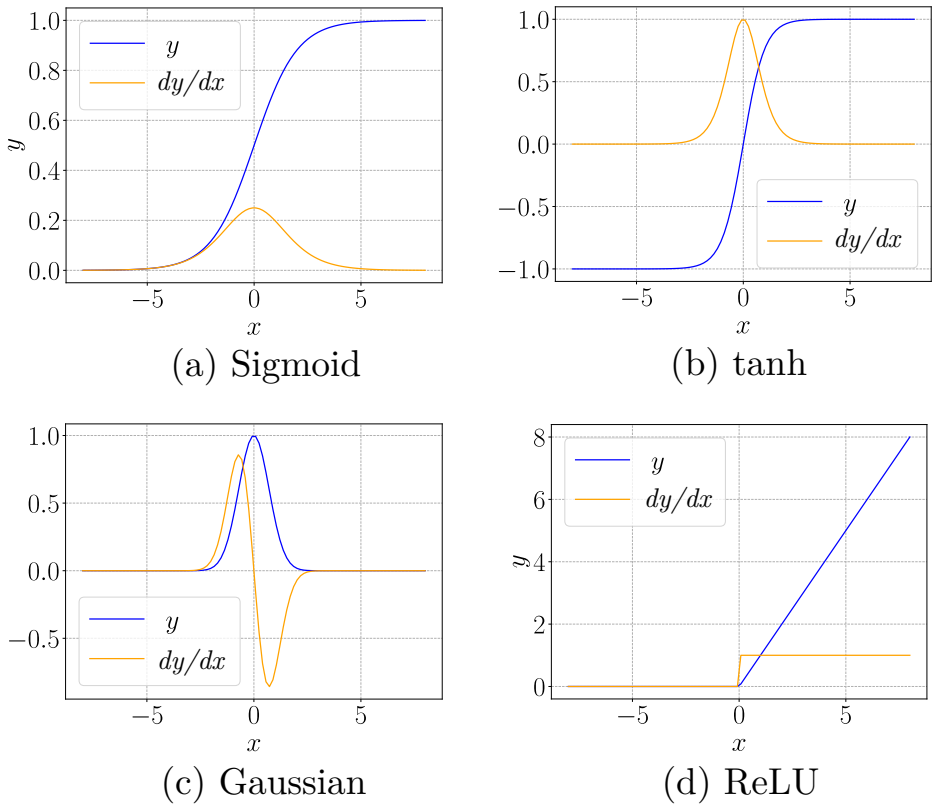


Figure 4.2: Different activation functions employed in NNs with their respective derivatives.

function maps the input value into a range between 0 and 1, denoted by the operation: $y = 1/(1 + e^{-x})$. It is normally used because of the easiness of its derivative, which is calculated knowing the original sigmoid result $dy/dx = y(1 - y)$. The simplicity of its derivative allows us to efficiently perform the back-propagation algorithm without using approximations. The fact that this function is smooth, continuous and bounded makes the back-propagation algorithm work effectively. However, the flaw is that in some cases, when training with back-propagation, the weights receive an update vanishingly small (due to the small partial derivative), effectively preventing the weight to change the value. This could cause that the neural network stops the learning completely and never approximate to a satisfactory outcome.

The hyperbolic-tangent (*tanh*) is very similar to the sigmoid function, and practically has the same properties. In fact, they both have the same

shape in the output form (see Fig. 4.2(b)). The difference lies in the output boundaries, where *tanh* allows the output to be in between -1 and 1 (not inclusive of those), being its output $y = tanh(x)$, and its derivative $dy/dx = 1 - tanh(x)^2$. This allows the function to apply a penalty to the node instead of just prohibit the node to fire at all (as it happens in the case of the original sigmoid function). Again, as the behaviour of the *sigmoid*, its derivative is continuous, letting us to train with the back-propagation algorithm. Nevertheless, as its predecessor, it presents the same shortcoming of the vanishing gradient, limiting its use for some cases.

A third (not so common used) non-linear function is the Gaussian. Fig. 4.2(c) depicts its outcome $y = e^{-x^2}$ and its derivative $dy/dx = -2xe^{-x^2}$. As shown, this function presents a symmetric behaviour, giving the same output for the same absolute value of the input. It presents its maximal output when the input is zero, and as the absolute value of the input grows, the output tends to zero. It is commonly applied when the output value depends mainly on the distance between the input and some fixed point, as is the case of the Radial Basis Function (RBF) [117, 118]. Apart from it they have no common use, since the computation is expensive; normally, needing approximation functions, something that will detriment the performance on the whole system.

Finally, the Rectifier Linear Unit (ReLU) [119, 120] is an activation function that has raised its population in NN applications [121]. As shown in Fig. 4.2(d), it allows to pass the input only when it is positive, otherwise, it gives a null value ($y = max(0, x)$). It presents two great advantages:

- it is more computationally efficient than the previous mentioned functions, as it just needs to do a comparison instead of doing expensive exponential operations. The same happens when calculating its derivative, as is 1 or 0 depending on the input.

- it deals with the vanishing problem introduced by the sigmoid family functions during the backward propagation in the training phase [122], allowing deeper networks to be trained more efficiently. According to Krizhevsky et al. [90], the networks trained with ReLU tend to display better convergence performance.

All of these reasons makes the ReLU function to be the most commonly used function when dealing with dense layer networks, as is the case of the

FF-NN, and Convolutional Neural Networks (CNN), which are the main focus of the present thesis.

4.3 The SC-Neural Network

NNs are constructed of several interconnected layers of neurons. As previously said, the most common model in deep NNs is composed of a scalar product block and a ReLU activation function. Therefore, in order to implement the SC counterpart of the ReLU-Neuron, it is mandatory to embed the following operations: multiplication, addition and maximum function using SC. In Chapter 2, such operations were implemented using SC, manipulating the correlation phenomenon. On the other hand, different stochastic neuron designs have been proposed in literature [30, 123, 111, 124], although none of them have exploited properly both phenomenons: the signal correlation and decorrelation. If properly used, both phenomenons can be exploited to offer a neuron design approach which could simplify the hardware implementation considerably.

Fig. 4.3 shows the proposed stochastic neuron design that exploits both correlation and decorrelation. For the sake of clarity, the bias term has been omitted since its effect can be emulated by tying one of the inputs to logic one (maximum SC value). The incoming SC vector $\mathbf{x}$ (formed by n elements of b bits) is generated by using the output of one RNG circuit $R_x(t)$, whereas the SC weight vector $\mathbf{w}$ (formed by n elements of b bits) is generated by using the output of a second RNG circuit $R_w(t)$. The BSC-array blocks denote Binary to Stochastic Converter circuits, which are employed using digital comparators, as detailed in Chapter 2.1.2. As a result, and considering a *bipolar* codification, the n-XNOR-gate array calculates the stochastic product between neuron inputs and weights. Bipolar codification is employed since the weight parameters include negative values. Then, these product signals must be added. For the addition, we consider to use the APC circuit approach, as is the most accurate of all of the possible adder circuits (see Chapter 2.3.2 for details). The APC yields a b-bit two's complement number at the output, representing the SC dot product between x and w. Once we have the stochastic inner product in digital representation, we must convert it into stochastic domain again to operate in the following layers. It is in this point, where the correlation phenomenon is fully exploited with the purpose of implementing the max function.

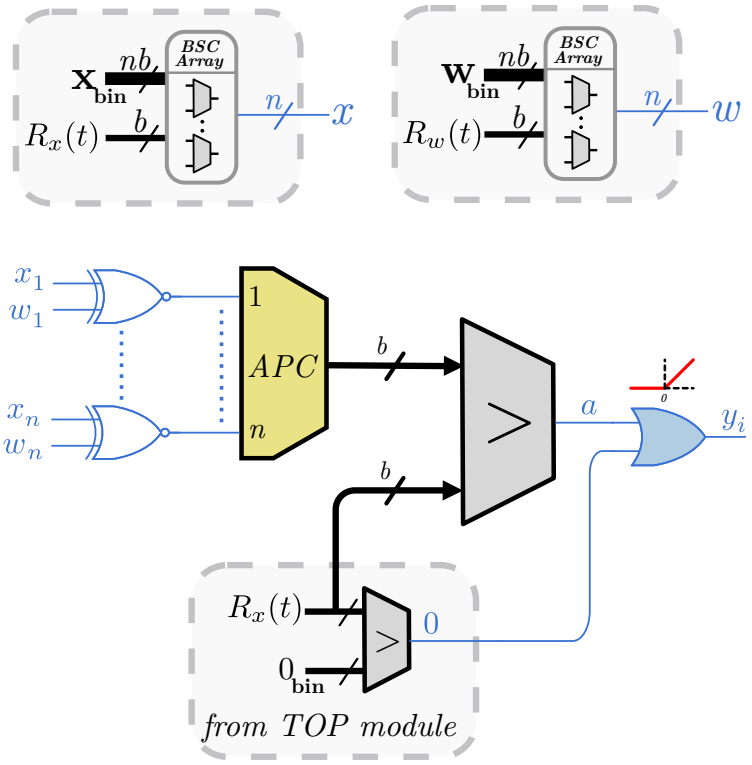


Figure 4.3: Stochastic neuron design exploiting correlation to reduce area cost. The SC-APC output a and zero-*bipolar* (0) are generated with the same RNG ($R_x(t)$) to produce total correlation between them, returning the ReLU function on the output with a single OR gate. Stochastic signals are depicted in blue. Tag **bin** implies *standard binary* format

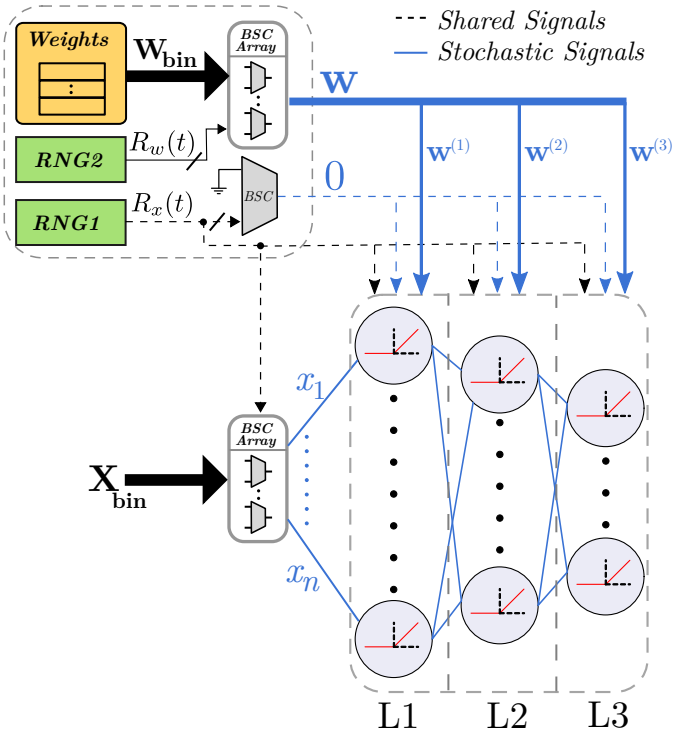


Figure 4.4: Neural Network implementation using two RNG for the whole system. RNG1 is used to generate the input stochastic vector $\mathbf{x}$, the zero *bipolar* signal 0 and the APC outcome inside each neuron. RNG2 is used to generate the stochastic weight vector $\mathbf{w}$. Shared signals among the different NN cores are depicted with dashed lines. Signals with 2's-complement b-bit precision are depicted in black, whereas stochastic signals are noted in blue color. Tag **bin** implies *standard binary* format, and upper index indicates the layer for weight distribution.

The ReLU function has lately been the target for the SC designers in literature. The different approaches struggle to implement this operation in order to hold the non-correlated throughput in the network [30, 31, 125]. This fact causes to be one of the challenging blocks if one is to implement a whole SC-NN in a parallel fashion, since following layers will be severely impacted by its imprecision. Nevertheless, the max function can be easily implemented using an OR gate if the inputs are totally correlated (see Chapter 2.3.3). The issue is try to find a way of guarantying a maximum correlation between the OR gate inputs, that for our SC-neuron case is the APC output and a zero signal reference. We can actually produce this effect by generating the SC-zero-reference signal (0 in Fig. 4.3) and the SC-APC signal (a) with the same RNG ($R_x(t)$ in the figure). In this way, the ReLU activation function is performed with a single OR gate, achieving the operation: $y_i = \max(0, \sum_{j=1}^{n} x_j \cdot w_{ij})$.

One of the benefits of the proposed stochastic-ReLU function approach, is its normalized reproduction of the standard-ReLU function used by the machine learning community. This means that the weights obtained after the training process of the ANN can be easily adapted to the hardware, due to the expected activation function is not disturbing. This is in clear contrast with respect to other published studies, in which the function outcome is distorted, as is the case of references [123, 31]. In these particular cases, the stochastic implementation of the ReLU function is high area-consuming, and moreover, it results in a ReLU clipped version, degrading the inputs for the following layers. In our simple ReLU proposal, the OR gate implementation computes the maximum function without clipping or distorting the signal, and therefore, the weights can be incorporated directly to the hardware after a simple process of normalization (as noted on further applications in Chapters 4.5 and 5.5). Moreover, approaches suggested by the literature do not control the correlation level of the outcome, being of unpredictable level. This is a problem for following layers, where full decorrelation must be guaranteed in order to achieve the multiplication operation; thereby leading to accumulate inaccuracies when the signal goes from layer to layer. But in the approach suggested, the correlation is intentionally manipulated to exploit the best of the two worlds (correlation and decorrelation), and at the same time is leveraged for saving resources.

Fig. 4.4 depicts the implementation of a 3-layer NN using the SC approach proposed. As shown, only two RNG are used for the whole system. The first one ($R_x(t)$ in the figure), is used for generating the input stochastic

vector $\mathbf{x} = [x_1, x_2, .., x_n]$, the zero *bipolar* signal (used as reference for the SC-ReLU), and the stochastic APC output. The second RNG ($R_w(t)$), is used for generating only the stochastic weight vector $\mathbf{w}$, which is connected along the whole network. This scheme accomplishes two main objectives. Firstly, it guaranties that for every single neuron, all of the SC input signals are totally uncorrelated respect to the SC weight signals, since all of the previous layer outputs are generated with the same RNG $R_x(t)$ (see Fig. 4.3 for details). This fact allows the achievement of an accurate SC multiplication on each individual neuron, regardless of the layer depth. But secondly, it asserts a total correlation for the ReLU operation inside each neuron, performing the *max* operation accurately.

Our proposed design is in contrast with the different designs presented in literature [30, 31, 125, 126], where the SC-ReLU function occupies a considerable amount of hardware resources and the correlation level of the signals generated are not manipulated between layers, performing an agnostic behaviour as the network gets deeper. Moreover, our design saves plenty of resources in the most hungry-resource block: the stochastic signal generator, as only two RNGs are employed for the whole system.

4.4 Working on the Accuracy in SC-NN

Although SC-NN implementations mostly incur in minimum error degradation in most of the cases, there exists some particular applications in which a high accuracy is demanded. For such applications, we can not convert directly from the original floating point weights model to SC; but rather, it is necessary to have some important considerations into account and proceed to do some optimization process. The cause for such a degradation in precision comes from different sources, but most of them share four roots. Firstly, the *quantization process*. SC operates with fixed point representation. So, there are natural losses in the process of converting from higher original precision (mainly, floating-point) to lower SC-precision in fixed point. This issue is something inherent from the process, and it can not be avoided. Secondly, the *stochasticity behaviour on signals*. As the signals are operated in a non-deterministic way, they perform with fluctuations, and this in turn, produces expected inaccuracies. Although stochasticity can be mitigated by means of controlling the correlation of signals in some way, this phenomenom, as the previous one, is inherent from the SC domain, and the price must be

assumed. Nonetheless, stochasticity can be mitigated by means of using deterministic RNG [127], as is the case of the full-period LFSR, among others [128, 36]. Thirdly, we have the *constrained value range*. SC signals present a constrained range of representation in between 0 and 1 for unipolar codification, and in between -1 and 1 for bipolar representation. This constraint, in addition to the quantization feature, produces limited options for the neurons to represent the original value of the output. Lastly, we have the *weight distribution problem*. As detailed in Chapter 2.3.1, in SC there is a range of input values where the multiplication incurs higher error. If the weight distribution of the trained model fall in this range, then the error produced by the SC multiplication will rise as a consequence.

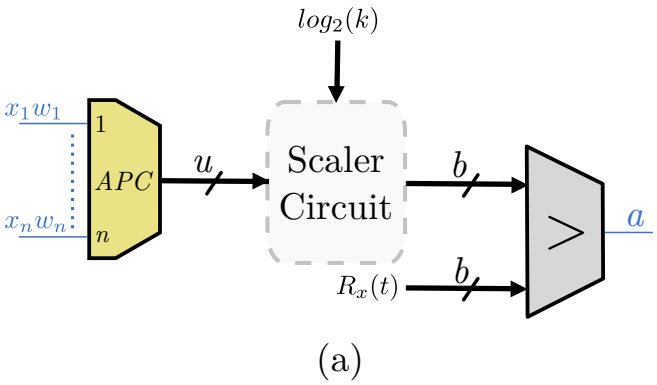
In the following two sections, we will address the two latter issues: the *constrained value range* and the *weight distribution problem*, as the two former are inherent from the representation.

4.4.1 The Constrained Value Range Problem

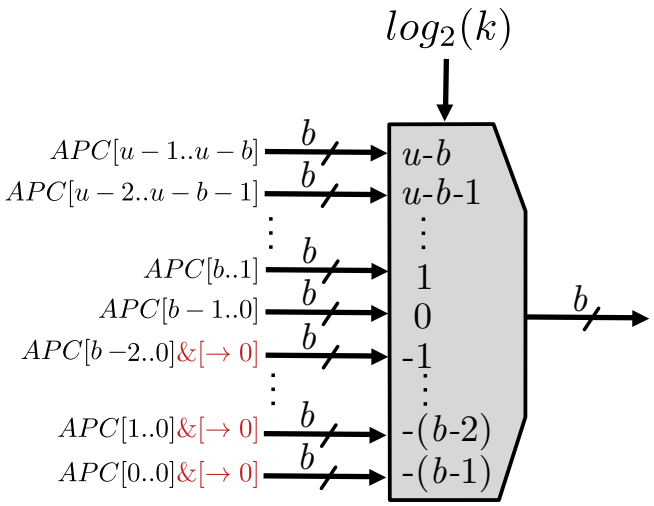
Intrinsically, SC has a constrained range of representation. Because of this, the ReLU-neuron outcome is greatly affected when implemented in SC. One of the natural hurdles we face, is that the original ReLU function has no boundary in its positive range, but the SC ReLU does have it. Therefore, when outputs grow beyond positive one (which occurs frequently) the APC output must be constrained to represent the value in SC domain. There are two ways of guarantee this to happen. (1) By means of *clipping the value* when the value exceeds one. This is something frequently done in literature, as can be seen in the work of A. Ren et al. [123, 30]. The difficulty with this way of approaching the problem is that we lose all the neurons firing values greater than one, inevitably losing precision in the network. But if rather, (2) we *normalize* the APC output, then we will not drop those firing neurons, which normally are the ones that convey the relevant information.

For achieving this behaviour, the APC output must be scaled down to fit the current bit-precision of the implementation. This can be done using a MUX circuit at the APC output and selecting a fix scale factor k (in power of two) for the signal to pass (connected to the MUX selector). In this way, we obtain at the MUX output the APC scaled outcome $\frac{APC}{k}$.

Fig. 4.5(a) shows the scale factor circuit added to the APC output, and Fig. 4.5(b) shows the scaler circuit details. As seen, only a portion of b-bit precision (being b the current application bit-precision) from the total APC



(a)



(b)

Figure 4.5: Scaler circuit added to the APC output in order to normalize the outcome. APC output precision $u = log_2(n+1)$ is normalized to b-bit precision using a scale factor k. The MUX output is a scaled version of the APC input : $\frac{APC}{k}$.

bit-precision $u = log_2(n+1)$ (being n the number of inputs in the APC) can be passed to the binary-to-stochastic conversion circuit, making the APC normalization effect. The MUX selector is controlled by the calculated k scale factor. Note that all the MUX inputs are b-bit wide, starting from the most significant APC output bits, up to the least significant. When k is less than 1, the input b-bits are completed by padding with 0's the least significant bits (see red indicators in Fig. 4.5(b)).

Now, the question that arises is how much do we need to scale the output in order to have the best results? and more importantly, how to calculate that scale factor? For answering these questions, we need to settle some properties of the implementation model.

1. *Every neuron of the same layer must be scaled by the same amount.* This property has to be maintained with the purpose of not affecting the next layer inputs, and for the SC model behaves as a scaled version of the original model. If this is not accomplished, the trained weight values in the original model will not work when used in the SC implementation. Therefore, if we scale the neuron output a_i by some constant k, all of the neurons presented in the same layer l must have the same k scaling. So that for every next layer neuron a_i^{l+1}, every input a_j^l is seeing as a k version of the original trained model.

$$\begin{aligned} a_i^{l+1} &= \phi\Big(\sum_j a_j^l \omega_j + \beta\Big), <= \text{original model} \\ \frac{a_i^{l+1}}{k} &= \frac{1}{k}\phi\Big(\sum_j a_j^l \omega_j + \beta\Big), <= \text{scaled version} \\ &= \phi\Big(\sum_j \frac{a_j^l \omega_j}{k} + \frac{\beta}{k}\Big) \end{aligned} \tag{4.2}$$

   If this can be guaranteed, then the scale factor does not affect the network behaviour, and the quantized version weights from the original trained model can be employed in the implementation.

2. *The k value must let pass the most amount of firing neurons.* The k value can not take a massive value, such that the neuron output vanishes. Since we have a limited quantized range of values to represent

our data, a high value in the scale factor will provoke a null value in the neuron output. But at the same time, the scale value must not be so low to provoking saturation to the max value possible (clipping) in all of the neurons firing high values. This will incur in a drop on accuracy. Therefore, it is important to find the best k value for every layer, such that it let us pass the maximum amount of firing neurons. The trade-off is between having a large enough value for k to reduce the chance of overflow (avoiding clipping), but at the same time, a small enough value to reduce the quantization error.

Based on the aforementioned properties, we propose two possible solutions:

1. *Finding a fix scale factor.* One possible solution which does not require extra hardware resources is to calculate the best possible k value in every layer for the data-set in question. We can have a rough estimation passing a representative portion of the data-set to the network and observing the neuron activation histogram per layer. Afterwards, we analyze which are the most frequent activation values and determine what is the optimum k that allows to pass such range of data. Fig. 4.6 shows graphically the concept. Once we have the neuron activation histogram for a concrete layer, we select a representative percentage of the whole activation setting (found empirically) and establish the max activation value for such a set, which we name as the activation threshold a_T. Recalling that k must be a power of two, we can easily calculate it as:

$$k = \zeta \left\lceil a_T \right\rceil, \tag{4.3}$$

where ζ is the next-power-of-two function:

$$\zeta(x) = 2^n, \text{if } 2^{n-1} < x \leq 2^n. \tag{4.4}$$

2. *Building a dynamic scale factor.* Although the fix method is appropriate because no extra hardware is required, it incorporates some limitations. Since the k factor is fixed (hardwired) in the network, input samples that under-excite or over-excite the neurons will considerably affect the outcomes, since they are vanished or clipped by a factor of k,

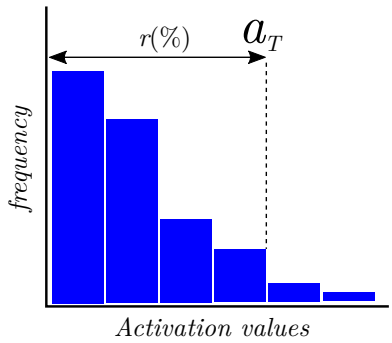


Figure 4.6: Histogram of neuron activations for the same layer. We pick a representative $r(\%)$ amount of data to pass using the activation threshold a_T.

respectively. Therefore, depending on the accuracy requirements of the application, the fix scale factor can not be effective. As a consequence, for accurate demanding applications, we need a *dynamic* normalization factor; some type of circuit which for the same layer, normalizes the outcome of every neuron with respect to the highest value at that time. In this way, the network performance becomes independent of the layer activation distribution. We introduce the circuit devoted to do this task: the Layer Auto-scale Block (LAB).

Basically, the LAB measures the APC value of every neuron in the current layer and finds the Most Significant One (MSO) bit, excluding the negative APC values (only positive values are evaluated as the negative ones become zero with the ReLU activation function). Once found, this value is encoded to be understood by the scaler circuit inside each neuron, as can be seen in Fig. 4.7. The LAB adjusts the layer scale factor such that the maximum activation value for the current layer can be represented with the full-range precision of the system. This is accomplished by controlling the MUX selector of the APC output, as explained previously.

As shown in the Fig. 4.7, a LAB per layer is needed. This adds some area usage for the circuit, but presents great advantages: Firstly, the process of calculating the best fix scale value is discarded, with the benefit of calculating it in real time by the circuit. Secondly, the network becomes independent of the data variation between samples,

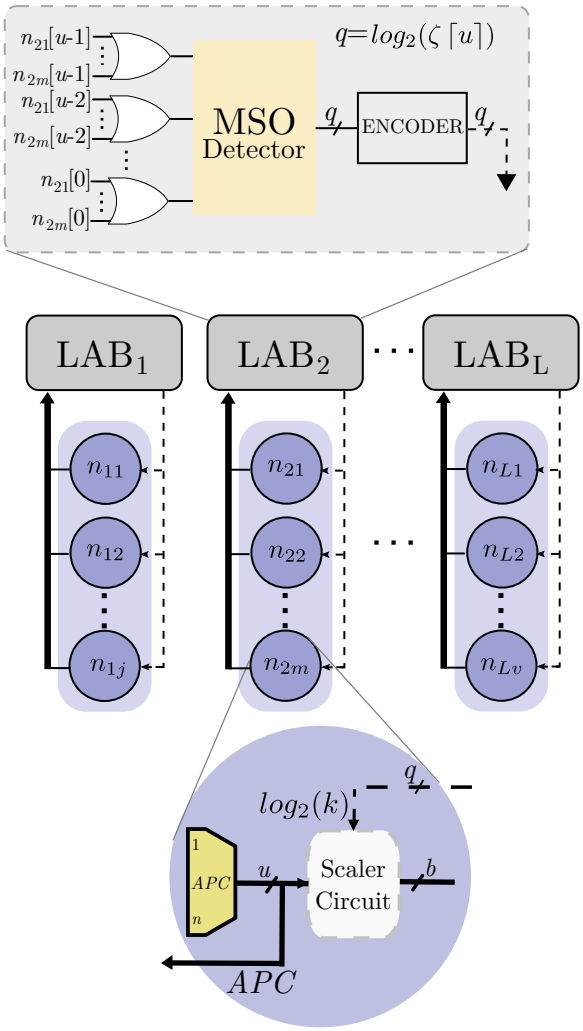


Figure 4.7: Dynamic scale factor circuit. The Layer-Auto-scale Block (LAB) calculates the maximum value activation for each layer by means of a Most Significant One (MSO) detector and an encoder. The LAB output bus width is $q = log_2(\zeta\lceil u\rceil)$, where ζ is the next-power-of-two function. The output of each neuron is normalized respect to the maximum activation value for the current layer.

Table 4.1: Fix and dynamic scale comparison for the use case presented in Section 4.5

Scale Factor Method	**Accuracy**	**Area (ALMs)**
Fix	0.65	16937
Dynamic	0.78	17418

as it calculates dynamically the scale factor for the activation setting produced by the current sample. Thirdly, there is a signal recovery phenomenon produced when the neurons are under-excited. When the highest output of the current layer is lower than 2^{b-1} (with b being the system precision), the LAB produces a scaling factor less than 0, producing a power of two amplification at the output (see Fig. 4.5(b) when the selector MUX value is less than 0). For understanding this issue better, let's take for instance a bit-precision b of 8 bits. Now suppose the maximum output activation of the current layer l is 30_{dec} = 00011110_{bin} (a value $< 2^7$). The LAB circuit detects that only 5-bits are necessary to represent the highest value, and as a consequence, it sets its output to $log_2(k) = -3$, commanding the MUXs to take these 5-bits as the most significant bits. The MUXs will then complete the 8-bits by right padding with zeros (left shifting) and consequently, at the output, instead of having 30_{dec}, we will have 240_{dec} =11110000_{bin}, amplifying the original value by a 2^3 factor. This effect behaves as a sort of recovering precision in SC systems, considering that the signal tends to vanish between layers, due to most of the activation values are near zero.

This proposed method is employed in the use-case implementation described in Chapter 4.5, where a high precision was required. Table 4.1 shows the trade-off when employing different scale factor methods for the use-case implementation. The area usage is reported after synthesis in the Quartus Prime 19.1 tool. Results showed an increase of only 2.8% in the total logic usage when employing the LAB, while achieving 20% more precision (see details in Chapter 4.5 for the accuracy metric).

4.4.2 The Weight Distribution Problem

Working in the SC domain is not an easy journey, as seen in previous section. To make matters worse, there is another hurdle to face which increases the imprecision in the SC-NN system : *the weight distribution problem*. After training the model using the conventional tools in floating point, the weight values tend to have a Gaussian shape around zero, as can be seen in Fig. 4.8 (blue shape). The reasons are twofold: Firstly, from a practical point of view, it is advised to initialize the weights to random numbers with a normal distribution [129], as it seems to help with the back-propagation parameter updating. Moreover, the weights tend to zero during training because of the regularization process, which penalizes the nonzero weights to avoid overfitting [130]. As a consequence, after training, most of the weights become near zero values. This fact introduces an added complication to the SC performance. Why? If we recall what was introduced in Section 2.3.1, the bipolar multiplication presents the highest error behaviour when its inputs are near zero, and manifests the most accurate outcomes when they are near boundaries [-1,1] (see Fig. 2.8). Several inaccuracies, which are not acceptable for most common cases, will occur in the system if applying a direct conversion from the original weights to SC values. Therefore, and although not always possible, forcing a different distribution for the weights in the training phase can be a possible solution to mitigate this effect.

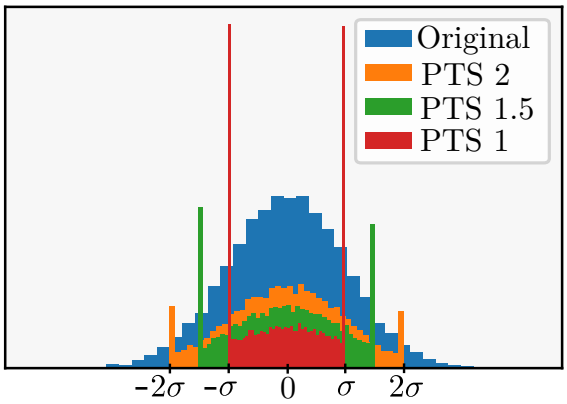


Figure 4.8: Weight distribution instance for a random model. Original weight distribution is depicted in blue, whereas weights after running the PTS technique with threshold values $w_T = 2\sigma, 1.5\sigma$, and 1σ are shown in orange, green and red, respectively.

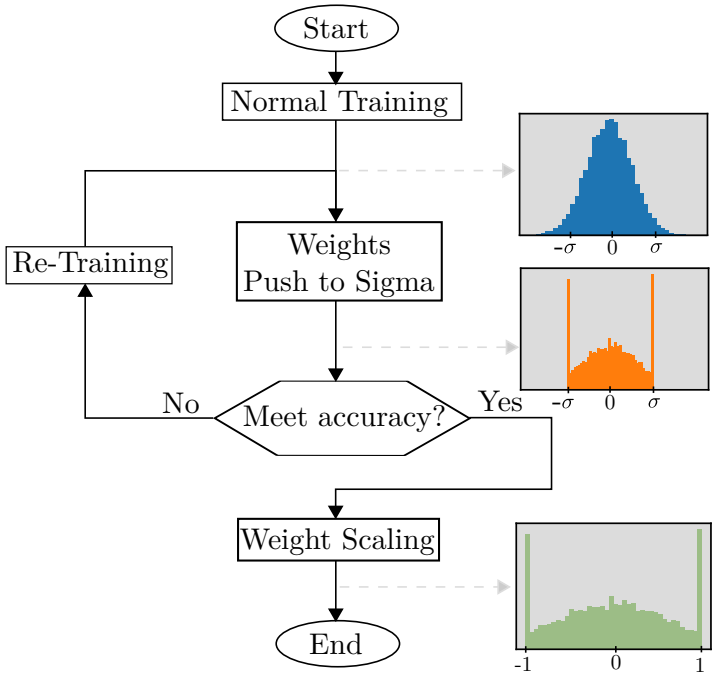


Figure 4.9: Block diagram for the PTS technique training process.

Fig. 4.9 depicts the proposed solution to circumvent the near-zero weight problem, tackling at the training phase. We firstly conduct the initial training until an acceptable accuracy is met. Afterwards, the *Push To Sigma* (PTS) technique loop is carried out. Therein, we calculate the standard deviation σ of the weight distribution layer-wise, and set a threshold value for each layer w_T^l in σ fractions (see Fig. 4.8, where we represent 3 different w_T values : 1σ, 1.5σ, and 2σ). The lower the threshold, the lower the number of weights close to zero, and the higher the number of weights in the boundaries [-1,1] once the weight scaling conversion is done, thus the better the accuracy on the SC-multiplication. Once we selected the threshold (through trial and error, we firstly initialize w_T to one σ), we clip all the weights of layer l to the determined threshold value w_T^l, as:

$$w_i^l = \begin{cases} -w_T^l, & \text{if } w_i^l < -w_T^l \\ w_T^l, & \text{if } w_i^l > w_T^l \\ w_i^l, & \text{otherwise} \end{cases} \tag{4.5}$$

After clipping the weight out layers with Eq. 4.5, we check if the required

application accuracy is met; if it is not the case, we do a retraining process of some epochs and adjust the weights again through PTS. This loop is repeated until we get the needed accuracy. Once the accuracy required is met, the training process is over, and the weights are ready to be converted to SC-domain through a simple weight scaling process. In the case of the accuracy required is not met, then, we can expand the w_T^l value in the process until the accuracy reaches the needed results.

In Section 5.5, we show the utilization of this technique to increase the inference accuracy in a real Deep Learning hardware implementation using SC. The difference with the base-line score (obtained with floating-point software) was substantially decreased compared to a simple direct weight conversion (no-PTS), obtaining for the no-PTS method a -1.04% difference, whereas for the PTS method only a -0.16%; an improvement of near one order of magnitude.

4.5 SC-NN Use Case: Virtual Screening Accelerator

Due to the data explosion it has occurred in all scientific and technological areas, the use of Artificial Intelligence (AI) arises as an optimal and quick strategy to convert raw data into useful information. This data explosion is particularly critical in the area of organic chemistry owing to the truly vast possibilities for constructing chemical compounds [131]. Such a vast chemical space embraces, but it is by no means limited to, the ever-increasing number of compounds from different chemical databases, which is now of the order of tens of billions and publicly available. For the specific problem of the analysis of interactions between different molecules (the aim of any drug discovery process after all), the number of possibilities explodes and becomes unmanageable. This is critical even if machine learning techniques are applied in the context of fast-growing screening libraries [132, 133]. That is why speeding up this type of processes by means of some accelerator methods is highly desirable.

Drug development is a time-consuming process where, on average, more than ten years and hundreds of millions of US dollars are needed. From the early 1990s, huge libraries of compounds have been made available to facilitate compound screening in the drug discovery process. To aid in the

early stages of the drug discovery process, new computing strategies were developed (Virtual Screening, VS) [134]. The primary goal of VS [135, 136, 137] is to find out, from a molecule database, a subset of molecules with a high chance to be chemically active. This subset of compounds is revised by medicinal chemists to improve their preclinical properties. When a 3D structure of the target is available and the binding site is known, this problem is more specifically called structure-based VS [138, 139, 140]. On the other hand, when at least a molecule with activity for the target compound is known, methods for ligand-based VS are used [141, 142, 143]. Certain ligand-based methods used for virtual screening, benefit from a set of molecules [144, 145, 146] instead of a single ligand used as a search template. Such models are typically generated with machine learning algorithms trained on ligands with known activity for the target and chemical properties [147, 148].

4.5.1 Artificial Neural Networks applied to VS

It is well known that the use of advanced machine learning methodologies in VS such as Support Vector Machines or Artificial Neural Networks (ANN) is becoming increasingly popular in ligand-based VS strategies [149]. ANN are inspired by how the brain processes information and have become trendy in many science and technological areas due to their ability to solve many complex pattern matching problems. ANN can be applied to adjust complex relationships without considering the underlying physical links. A sufficiently large dataset of samples suffices to correlate the data, which consists of pairs of an input (a sample) and an output (its class). ANN are normally structured as a set of interconnected elementary processing units, each one implementing a non-linear operation. Because of the large configurability of ANN, a huge number of training examples are required for the adjustment of the connectivity matrices $\Omega^k = \left\{\omega_{ij}^k\right\}$. This adjustment is specially easy to obtain for feedforward neural networks. Given a specific training dataset whose desired network response is correlated with the input, the optimization of Ω is a non-convex problem that can be solved by using the backpropagation algorithm.

Neuromorphic hardware (NH) is a research field which has been propelled by the need of developing high-performance AI systems [150, 151, 152, 153, 154]. NH is able to provide timely responses to those applications which require processing huge amounts of data [155]. Many efforts have been made in implementing neuromorphic digital [156] and analogue [157] circuits. The

great advantage of digital implementation is its ability to create massive ANN devices on a single chip and to easily test them in Field Programmable Gate Arrays (FPGAs) [158, 159]. Nevertheless, the use of large numbers of multipliers, necessary to reproduce the neural weights, constrains its proper parallelization and, as a consequence, it limits the speeding up process.

A possible solution is the use of approximate multipliers that are built by using non-deterministic computing methods. In this particular context, Stochastic Computing (SC) arises as a potential alternative [13, 160]. SC has been used to develop many different complex tasks such as implementation of Bayesian classifiers [161, 162], image processing [163, 60], or implementation of neuromorphic circuits [164, 165]. Data representation within the SC paradigm is performed in a probabilistic way with the use of boolean quantities which switch randomly over time. The probability of a single bit in a given state is used to encode the desired signal, thus reducing the area cost since only one wire is needed to carry the whole numerical value with this probabilistic codification. Hence, complex functions whose implementation requires a high amount of area, such as multiplications [166], are performed by using a single logic gate, with great savings in terms of area and power [167, 168]. However, this area reduction has a cost in terms of precision loss. This is not critical anyway in most machine learning applications (e.g., ANN), where a relatively reduced set of output categories or classes must be discriminated based on generic similarities. Indeed, most of the current machine learning applications use a small number of bits to represent digitized signals [169] because the difference in the final result between using high-precision floating-point signals and low-precision 8/16-bit signals is negligible [170]. Hence, the integration time used to evaluate the result of stochastic operations can be considerably reduced since it is exponentially dependent on bit precision. In addition, as a result of the low area employed in SC, different ANN cores can be implemented in parallel on a single chip, thus increasing the throughput compared with traditional computing methods. Be that as it may, tackling the miscalculation produced by stochastic correlation signals, while minimizing the area usage, remains a challenge for SC designers. The main obstacle is the high number of Random Number Generators (RNG) needed, which are essential for producing the probabilistic bit streams used in SC. Indeed, some operations, such as multiplications, require the inputs to be uncorrelated (statistical independent) if we want to achieve accurate calculations. That is the reason why independent RNGs are necessary to generate each stochastic signal. This fact makes these blocks

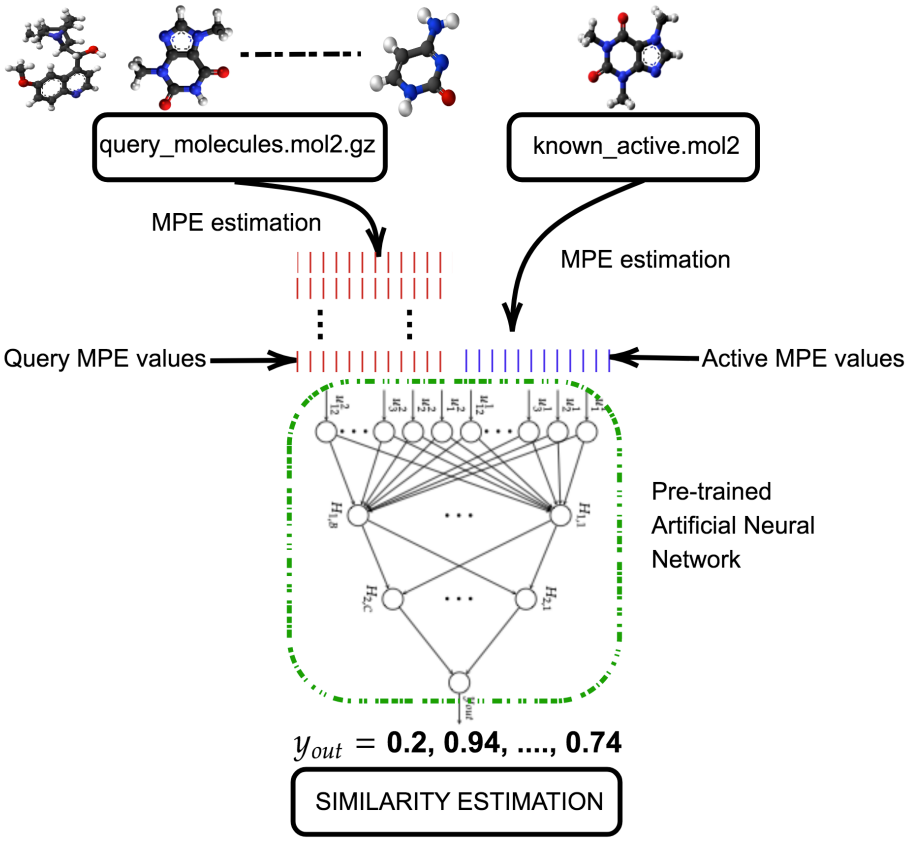


Figure 4.10: Process of similarity comparison between a set of query compounds with unknown chemical properties and a compound with known activity against a specific therapeutic target.

occupy up to the 90% of the whole area of the final circuit footprint [171], limiting the benefits of SC. It is therefore crucial to minimize the use of RNG in SC systems and simultaneously maintain the accuracy.

In this work, we present an ultra-fast Virtual Screening method based on the use of an ANN implemented with Stochastic Computing. The process scheme is shown in Fig. 4.10, where we compare a known compound that presents a certain biological activity, with a set of query compounds with unknown activities (database). The proposed system provides, for each query compound, a likelihood value which measures if the query compound presents the same activity as the active one. In this process it must be considered

that:

- Electron distribution assigned to individual atoms must be taken into consideration along with their spacial distribution. The .MOL2 file type is therefore needed for each compound since this information is included in this format.

- A set of molecular descriptors are estimated from the MOL2 files. Molecular Pairing Energies (MPE) [172], dependent on both charge distribution and molecular geometry, are adopted as descriptors. These descriptors are independent of rotations and translations of the compound, and provide valuable information about its binding possibilities.

- Once the MPE values are obtained from query and active compounds, a preconfigured ANN estimates the similarity between queries and the active compounds.

- The database is finally ordered according to the similarity values provided by the system and, consequently, only the top-most compounds are selected for laboratory assays.

To validate the model, we use the DUD-E dataset of chemical compounds [173]. The DUD-E docking benchmark is a widely used database to evaluate virtual screening methods. It is composed of 102 different pharmaceutical targets that include a crystal ligand, along with a set of active compounds and decoys (assumed as non-actives). The performance of the system is evaluated through the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve. As it will be shown, a competitive accuracy in regard to other ligand-based models present in the literature is obtained, along with a good performance in terms of both processing speed and energy efficiency.

4.5.2 Molecular Pairing Energies descriptors

Virtual Screening consists of comparing each query compound of a database with active ligands against a specific target [147, 148]. This comparison is normally made by using molecular descriptors that can be based on physicochemical properties. In this work, we propose a classification model which

uses Molecular Pairing Energies MPE as the main descriptor [172] together with a neural network discriminator. These pairing energies are defined as

$$E_{ij} = K\frac{q_i q_j}{r_{ij}} \tag{4.6}$$

where q_i and q_j are the partial charges of each atom of the molecule and r_{ij} is the distance between them.

These partial charges are related to the electron distribution that can be assigned to individual atoms through quantum mechanical calculations. Nevertheless, to screen thousands or millions of compounds, there are faster and more efficient methods such as the Partial Equalization of Orbital Electronegativity or the Merck Molecular Force Field [174, 175, 176].

Among all the pairing energies present in a compound, we choose the N highest (positive) and lowest (negative) energies, thereby creating an ordered 2N-dimensional vector for the description of the molecule. If we have less than 2N pairing energies, the vector will be center-padded with zeroes until it reaches the 2N dimensions. In this work we set N to 6, so we use a total of 12 energy descriptors per compound. The most electropositive or electronegative MPE values are related to molecular scattering or the assembly properties of the compound.

In this work, the MPE model is applied to the full DUD-E database, in which the partial charges have been estimated by using the MMFF94 force field [176], that has been implemented within the Openbabel software. Empirically, the MPE model has shown a good capability to cluster those compounds showing similar chemical properties [172], as it can be appreciated in Fig. 4.11, where we have plotted the most positive and most negative pairing energies for five different DUD-E targets. Fig. 4.11 suggests that those compounds with a higher cohesion energy usually have a higher scattering energy. The working hypothesis is that MPE values can be efficiently used to compare chemical activities between compounds. This way, an AI-assisted model, such as the ANN proposed, may take advantage of this representation and obtain good results.

4.5.3 Neural Network implementation

ANNs have emerged as a powerhouse for prediction and classification tasks. In this work, ligand-based virtual screening is considered a classification problem to be solved via an ANN-based algorithm.

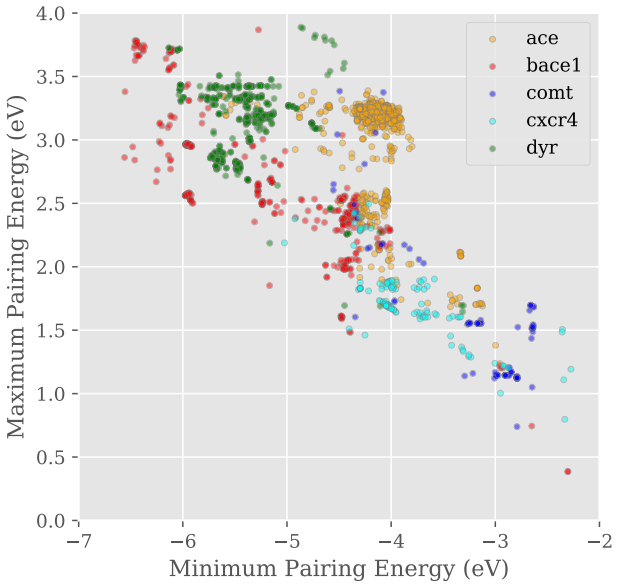


Figure 4.11: Clustering capacity of pairing-energy descriptors when using the most negative and most positive MPE values for five different DUD-E targets.

Each neuron in the network computes a transfer function of its weighted inputs so that the output of the j^{th} neuron, omitting the bias term is:

$$a_j = \phi\Big(\sum_i \omega_{ij} x_i\Big), \tag{4.7}$$

where ω_{ij} is the weight assigned to the i^{th} input x_i and ϕ is the non-linear activation function. It is supposed that all inputs come from the previous neural layer.

Several activation functions have been used in the literature. Among them, the ReLU function ($\phi(x) = max(x, 0)$) is one of the most widely used functions due to its simplicity for both inference and training. The ReLU function has two great advantages. First, it addresses the vanishing gradient problem introduced by other activation functions during the backpropagation training phase [122]; and second, it can be easily implemented in Stochastic Computing by using a reduced set of logic gates.

The main purpose of this work is to develop an accurate and energy-efficient methodology to implement a ligand-based Virtual Screening process. Starting from 24 MPE values (12 per each compound to be compared), we studied several ANN architectures with a single output (i.e., 1 and 0, meaning active and decoy, respectively) in order to estimate target similarity. Fig. 4.12 shows the scheme of the ANN architecture employed, in which the parameter u_j^k stands for the j^{th} component of the k^{th} compound, $H_{l,j}$ stands for the j^{th} neuron in the l^{th} hidden layer, and the output y_{out} is the prediction of the similarity between compounds.

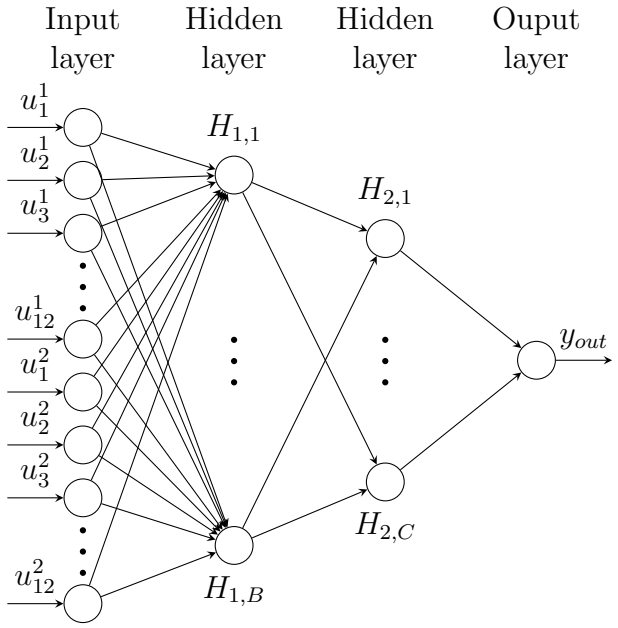


Figure 4.12: ANN architecture employed for the estimation of similarity between two compounds. Each compound u_j^k is described by 12 energy descriptors, where j is the j^{th} component of the k^{th} compound, $H_{l,j}$ stands for the j^{th} neuron in the l^{th} hidden layer; B and C are the number of neurons in the first and the second hidden layer; and y_{out} is the prediction of the model.

4.5.4 Stochastic Computing ANN

Fig. 4.13 shows a general view of the hardware implementation of the proposed VS hardware using the proposed SC neuron. As it can be observed,

only two RNGs are needed for the whole VS acceleration system (LFSR1 and LFSR2 in the figure). The proposed architecture saves a large amount of hardware resources since the RNG circuits are one of the most area-demanding blocks in stochastic computing designs. A Linear Feedback Shift Register (LFSR) circuit is used as a pseudo-RNG. The LFSR1 block generates $R_x(t)$, which is used in the stochastic conversion of the inputs $\mathbf{u}^{1,2}$, the zero reference signal, and the APC binary output of each stochastic neuron. All these signals present maximum correlation. Binary to Stochastic blocks (BSC) are employed to convert from the 2's complement to the stochastic domain using LFSR sequences. The output vector of each layer is denoted as $\mathbf{a}_l$, where l stands for the hidden layer of the network. To attain full decorrelation between neuron inputs and weights (needed for multiplication), LFSR2 is employed as a second pseudo-RNG. It provides the signal $R_w(t)$) which generates the stochastic weight vector $\mathbf{w}(t)$.

As noted, the proposed stochastic hardware implementation exploits the correlation phenomenon between signals and, at the same time, minimizes the area usage by reducing the number of RNG employed in the circuit to only two LFSR blocks. Furthermore, the compounds' stochastic signals $\mathbf{u}^1$ and the stochastic weights $\mathbf{w}(t)$ are shared by the k different cores (see dashed lines), thus allowing to increase parallelism. Moreover, each layer in the network has an LAB block (not shown in the figure), which computes the scale factor dynamically to improve the performance accuracy (see Table 4.1 for details).

The hardware proposal can be embedded in a single chip, thus increasing the acceleration factor this architecture can attain.

4.5.5 Experiments and Results

To evaluate the performance and capabilities of the proposed VS hardware, we used the DUD-E [177] database. This database contains 22,886 active compounds and their affinities to 102 targets.

We built the training set by incorporating the 50% of actives and the 10% of decoys from each target. Each sample of the training set incorporates two compounds as inputs: the crystal ligand of the target ($\mathbf{u}^1$) and an active or a decoy for $\mathbf{u}^2$. We used a total of 162,530 samples for the training set and 1,300,804 ones for the testing set. A learning rate of 0.001 with an Adam optimizer [178] was employed to train each model.

We performed our analysis based on two comparisons. First, we evalu-

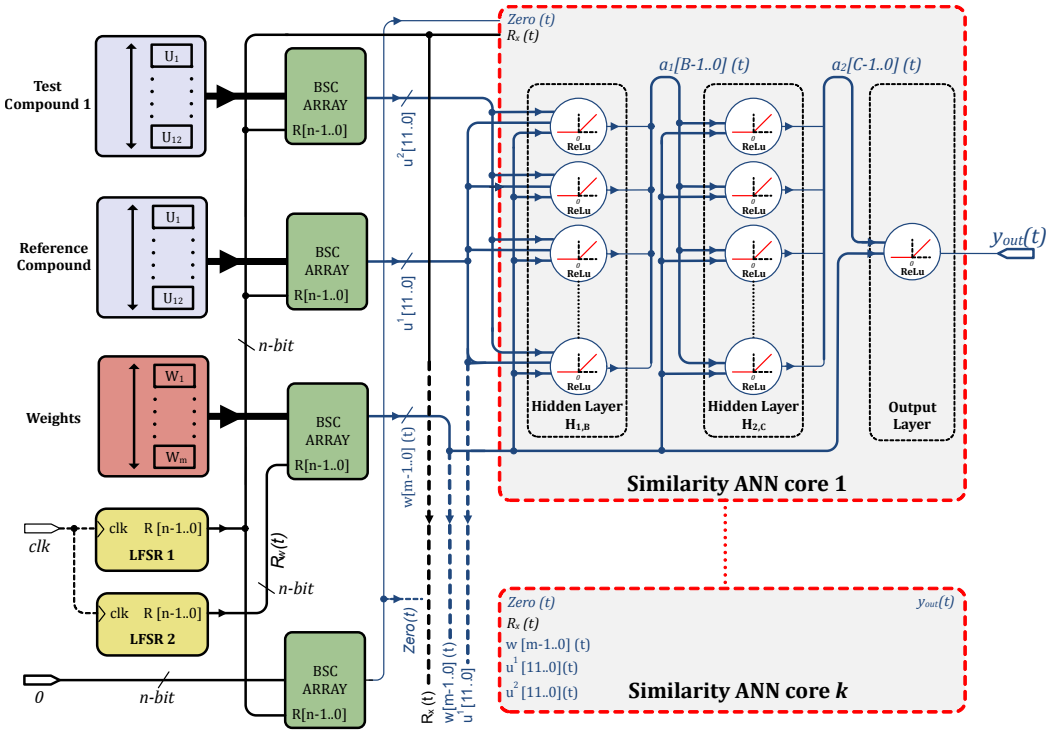


Figure 4.13: Neural Network implementation using two LFSRs for the whole system. LFSR1 is used to generate the input stochastic vector $\mathbf{u}^{1,2}$, the zero *bipolar* signal $zero(t)$ and the APC outcome inside each neuron. LFSR2 is used to generate the weight stochastic vector $\mathbf{w}(t)$. Shared signals among the k ANN cores are depicted in dashed lines. Signals with n-bit 2's-complement are noted in black whereas stochastic signals are noted in blue.

ated three different SC-ANN models and compared them with their software counterparts. Second, we compared the SC-ANN implementations with other ligand-based works present in the literature. The metrics we chose for the performance evaluation were: (a) accuracy measured with the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve, (b) processing speed in inferences per second, and (c) energy efficiency in terms of inferences obtained per Joule invested. The AUC value was calculated for each target and the mean value was reported. The power reported for the SC models was calculated by the Power Analyzer Tool incorporated in the Quartus environment. We took into consideration the worst case scenario for the signal toggle rate (50% of variation). We have reported the total power calculated by the tool. As detailed in Fig. 4.13, we embedded the maximum possible number of ANN cores on the FPGA, occupying the maximum amount of logic resources on the device. This makes the power consumption for the SC implementations be practically the same (see ANN cores and FPGA ALM(%) columns in Table 4.2 for details). As to software, we reported the Thermal Design Power (TDP) of the processor specified by the manufacturer.

Hardware measurements vs software simulations

Table 4.2 compares the performance of SC and software (SW) implementations. The model name points out the number of neurons in the first hidden layer, being 12 for a 24-12-6-1, 24 for a 24-24-12-1, and 48 for a 24-48-24-1 architectures. SW results were produced through an Intel(R) Xeon(R) X5670 processor with a 64-bit floating-point precision running at 2.93 GHz. Concerning the SC implementation, we used the same weights derived from its SW counterpart. SC results were measured by using a Gidel PROC10A board (Fig. 4.14), which contains an Intel Arria-10 10AX115H3F34I2SG FPGA. This device has 427,200 ALMs, built with TSMC's 20 nm process technology, improving the performance in regard to previous FPGA versions. Moreover, it has a PCI Express (PCIe) 3.0 specification (Gen3) IP that allows rapid communication interface. The package is a 35×35 mm^2 1152-FCBGA. We ran the application at a clock frequency of 125 MHz. As to programming, we used the Quartus Prime 19.1 multiplatform environment.

The bit precision for the SC implementation was 12 bits, running a sequence length of $N = 2^{12} - 1 = 4095$ cycles. We embedded as many SC-ANN cores as possible on the device for the different architectures (reported in

Figure 4.14: Gidel PROC10A board with an Intel 10AX115H3F34I2SG FPGA [179] running the 12-bit SC implementation at 125 MHz.

the ANN cores column). The number of FPGA resources employed is also pointed out in the table.

As shown, SC models display an AUC degradation with respect to the SW implementation of 0.04, 0.01, and 0.01 for the 12, 24, and 48 architectures, respectively. Nevertheless, they work 74, 33, and 15 times faster than their SW counterparts. Even more, they are more energy-efficient by factors 333x, 148x, and 66x. Additionally, we measured the AUC for the 12-bit quantized models (not shown in the table), and observed that the SC implementation did not exhibit any accuracy degradation. We thus conclude that the only degradation presented with respect to SW results is because of the quantization process and not due to the stochasticity of the technique.

As it can be observed, no DSP or RAM are employed in the FPGA because a non-conventional approach is used for the calculus (SC), using only the distributed resources of the device. This fact undoubtedly shows the advantages provided by SC.

Table 4.2: Performance comparison between FPGA measurements and software simulations for several ANN architectures.

Model	**AUC Mean**	**Speed (inf/sec)**	**Power (W)**	**Energy Efficiency (inf/Joule)**	**ANN cores**	**FPGA ALM(%)**	**FPGA DSP(%)**	**FPGA BRAM(%)**	**Clk Freq (MHz)**
SC-12	0.62	3,205,128	21	152,625	105	340,305(80%)	0(0%)	0(0%)	125
SC-24	0.71	1,373,626	21	65,411	45	329,715(77%)	0(0%)	0(0%)	125
SC-48	0.78	549,451	21	26,164	18	309,909(73%)	0(0%)	0(0%)	125
SW-12	0.66	43,573	95	459	1	-	-	-	-
SW-24	0.72	42,034	95	442	1	-	-	-	-
SW-48	0.79	37,397	95	394	1	-	-	-	-

Table 4.3: Performance comparison between this work and different ligand-based methods taken from the literature.

Model	**AUC Mean**	**Speed (inf/sec)**
This work (SC-48)	**0.78**	549,451
This work (SC-24)	0.71	1,373,626
eSim-pscreen[180]	0.76	12.3
eSim-pfast[180]	0.74	61.2
eSim-pfastf[180]	0.71	274.9
mRAISE[181]	0.74	–
ROCS[182]	0.60	1820
USR[183, 182]	0.52	$\mathbf{5.0 \times 10^6}$
VAMS[182]	0.56	109,000
WEGA[184]	0.564	1.6×10^{-3}
OptimPharm[184]	0.56	8.7×10^{-4}

Table 4.4: Comparison of percentages linked to the different methods that fit a specific AUC threshold.

Model	**% AUC** $< \mathbf{0.5}$	**% AUC** $\geq \mathbf{0.6}$	**% AUC** $\geq \mathbf{0.7}$	**% AUC** $\geq \mathbf{0.8}$	**% AUC** $\geq \mathbf{0.9}$
This work (SC-48)	**0**	**92**	**74**	**43**	**18**
eSim-pscreen[180]	5	81	69	**43**	17
eSim-pfast[180]	9	82	62	34	14
eSim-pfast[180]	5	79	53	26	6

Comparison with Other Ligand-based Models

Table 4.3 compares the SC models of this work with nine different ligand-based methods from literature: eSim-pscreen [180], eSim-pfast [180], eSim-pfastf [180], mRAISE [181], ROCS [182], USR [183, 182], VAMS [182], WEGA [184] and OptimPharm [184]. SC-48 outperforms all other methods in terms of AUC. It has an improvement of 0.02 with respect to the best AUC (0.76) method of other works (eSim-pscreen [180]. What's more, it is 44,670 times faster. If we compare the SC-24 method with the fastest model in the literature (USR [183, 182]), our proposal is 3.65 times slower, but it yields 0.19 more AUC.

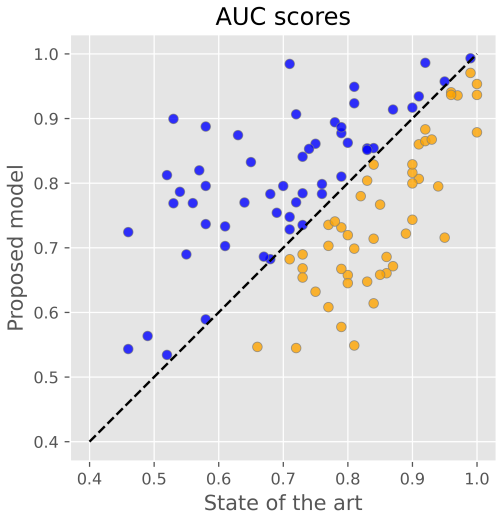


Figure 4.15: Visual representation of the scores shown in Table 4.5. Dots in blue indicate that the proposed VS system performs better, otherwise we highlight the dots in orange colour.

A detailed analysis of the AUC for each individual target (from 102 in total) of the most accurate models in the literature is presented in Table 4.4. Each column presents, for each model, the percentage of targets that fits a specific AUC threshold. The proposed model presents the best results in the table (written in bold).

Table 4.5 shows a per target AUC score comparison between the SC-48 implementation and the best model of [180, 181, 182, 184]. Fig. 4.15 shows a graphical representation of this data. We can appreciate that the proposed model provides the best AUC score values for 53 of the targets present in the database.

Table 4.5: Per target comparison of the current SC implementation versus state-of-the-art models. Data taken from [180]

Target	Proposed model	Max (other)	Target	Proposed model	Max (other)	Target	Proposed model	Max (other)
aa2ar	**0,80**	0,76	fabp4	**0,85**	0,83	mmp13	**0,91**	0,72
abl1	0,69	**0,73**	fak1	0,72	**0,95**	mp2k1	**0,87**	0,63
ace	**0,86**	0,75	fgfr1	**0,98**	0,71	nos1	**0,90**	0,53
aces	**0,81**	0,52	fkb1a	**0,77**	0,72	nram	0,83	**0,9**
ada	0,81	**0,91**	fnta	0,74	**0,78**	pa2ga	**0,85**	0,74
ada17	**0,86**	0,8	fpps	**0,99**	0,99	parp1	0,74	**0,9**
adrb1	**0,80**	0,7	gcr	**0,77**	0,64	pde5a	**0,74**	0,73
adrb2	**0,83**	0,65	glcm	**0,89**	0,78	pgh1	0,65	**0,73**
akt1	**0,74**	0,58	gria2	**0,68**	0,68	pgh2	**0,85**	0,84
akt2	0,55	**0,66**	grik1	0,67	**0,73**	plk1	0,78	**0,82**
aldr	**0,75**	0,71	hdac2	**0,77**	0,53	pnph	0,86	**0,92**
ampc	**0,78**	0,76	hdac8	0,77	**0,85**	ppara	**0,92**	0,9
andr	0,68	**0,71**	hivint	**0,56**	0,49	ppard	**0,92**	0,81
aofb	**0,54**	0,46	hivpr	0,83	**0,84**	pparg	**0,89**	0,79
bace1	**0,79**	0,54	hivrt	**0,73**	0,71	prgr	0,70	**0,81**
braf	0,74	**0,77**	hmdh	0,86	**0,91**	ptn1	**0,82**	0,57
cah2	**0,99**	0,92	hs90a	0,65	**0,8**	pur2	0,95	**1**
casp3	**0,89**	0,58	hxk4	0,82	**0,9**	pygm	**0,80**	0,58
cdk2	0,72	**0,8**	igf1r	**0,73**	0,61	pyrd	0,80	**0,9**
comt	0,97	**0,99**	inha	0,54	**0,72**	reni	**0,81**	0,79
cp2c9	**0,53**	0,52	ital	0,70	**0,77**	rock1	0,58	**0,79**
cp3a4	**0,59**	0,58	jak2	0,55	**0,81**	rxra	0,87	**0,93**
csf1r	0,66	**0,8**	kif11	0,65	**0,83**	sahh	0,94	**1**
cxcr4	0,73	**0,79**	kit	**0,75**	0,69	src	**0,69**	0,67
def	0,66	**0,86**	kith	**0,93**	0,91	tgfr1	0,71	**0,84**
dhi1	**0,78**	0,68	kpcb	0,66	**0,85**	thb	0,72	**0,89**
dpp4	**0,78**	0,73	lck	**0,69**	0,55	thrb	**0,85**	0,83
drd3	**0,72**	0,46	lkha4	0,61	**0,84**	try1	**0,91**	0,87
dyr	**0,96**	0,95	mapk2	0,69	**0,86**	tryb1	0,80	**0,83**
egfr	0,61	**0,77**	mcr	**0,88**	0,79	tysy	0,88	**0,92**
esr1	0,94	**0,96**	met	0,67	**0,87**	urok	**0,95**	0,81
esr2	0,94	**0,97**	mk01	0,67	**0,79**	vgfr2	0,63	**0,75**
fa10	**0,84**	0,73	mk10	**0,77**	0,56	wee1	0,88	**1**
fa7	0,94	**0,96**	mk14	**0,70**	0,61	xiap	0,79	**0,94**

4.6 Summary

This work shows the potential application of the MPE descriptors and Artificial Neural Networks (ANN) to Virtual Screening (VS). We have also shown a new methodology to accelerate the proposed VS process in an energy-efficient way by using Stochastic Computing (SC) hardware design techniques. An FPGA-compatible implementation has been realized, and its performance in terms of accuracy, processing speed and energy efficiency has been estimated. Compared to the state-of-the-art, the proposed model shows an increase by a factor of 44,670 in terms of processing speed and an improvement of a 2% in terms of overall accuracy. We have also demonstrated the benefits attained by employing non-conventional (SC) hardware accelerators for ANN when processing huge databases. In particular, we have introduced a new methodology in which the implementation of the activation (ReLU) function exploits correlation between signals, obtained thanks to the APC isolating characteristics. Overall, only two LFSR blocks are needed for the full ANN design in such a way that the impact of these blocks, historically regarded as the main drawback of SC, is minimized. To summarize, this work demonstrates the potential benefits of using non-conventional (stochastic) accelerators for ANNs when processing huge databases, especially suitable for those computationally costly processing problems such as Virtual Screening.

Chapter 5

STOCHASTIC COMPUTING CONVOLUTIONAL NEURAL NETWORK

5.1 Introduction

Vision is one of the most important human being senses. We rely on what our eyes can capture more than any other sense. From navigating in the physical world, recognizing and manipulating objects, to interpreting facial expressions to understand emotions, the visual sense is a substantial part of the human experience. This has lead the scientific community for decades to put a big effort in trying to understand how it works and model its nature. Back in the '60s, the interest in the neural basis of vision and the characterization of the processing [185], persuaded computer scientists to apply the new findings in neuroscience to artificial vision. David Hubel and Torsten Wiesel [185] working at Harvard observed how the visual cortex reacts in cats. They found that certain neurons in the visual cortex respond to specific patterns and regions of visual stimuli, and more importantly, that there exists a structure of neural layers in the cortex. Since then, different studies have been made to apply Artificial Neural Networks (ANN) to image processing [186, 187]. However, typical images are large, with several variables, which thus, frustrated the use of plain ANN on this topic. Just the first fully-connected layer of an ANN, with, for instance, only 100 hidden neurons, would multiply the number of weight parameters by 100 for each input

pixel. That is incredibly large for computing. Therefore, the application of ANN for vision was eluded for some years. However, in the '90s, Y. LeCun et al. popularized what today is known as Convolutional Neural Networks (CNNs)[188] (based on previous works like [189]). In their work, they applied a CNN for recognizing handwritten digits with important improvements with respect to previous works. Their early version was called LeNet and found a niche market in banking and postal services. Despite that, the constraints in power computation for those days made CNNs remained on the sidelines of visual computing. Nevertheless, as the shrinking of transistors was growing, more robust and efficient processors were developed, which brought a revival upon CNNs interest. The breakthrough was carried out in 2012, when Alex Krizhevsky et al. introduced AlexNet [90], a CNN capable to recognize complex patterns in color images from the ImageNet database[190]. Using the same CNN structure as LeCun, but deeper, with 5 convolutional layers (LeNet had 3 layers), AlexNet achieved a top-5 error of 15.3% in the contest, more than 10.8% better than the runner up. Since then, a huge booming research in this area has emerged up to these days.

CNNs have been applied in many different applications, being the preferable approach when tackling computer vision problems. In the medical field, they have been adopted for detecting lesions and tumour successfully [8]. Alzheimer's disease prediction by means of an image extracted from Magnetic Resonance Imaging (MRI) of a human brain has been carried out by CNNs [191]. Moreover, they have broken through in the drug research realm by predicting molecular properties such as binding or toxicity capacity [192]. In the same manner, AtomNet [193] discovers chemical features, such as aromaticity, sp3 carbons and hydrogen bonding. CNNs have been used for natural language processing as well, performing tasks such as semantic parsing[9], sentence modeling,[194] and prediction [195]. Furthermore, CNNs have been used to add sounds to silent movies [196] and to generate captions for images [11]. In the autonomous navigation domain, they have been successfully employed for drone navigation [10], robotics and particularly in autonomous car vision systems [197].

Recently, increasing attention in employing such models in embedded devices has aroused. But the efficiency of such applications relies on the capacity of computation made by the processors where they run. CNNs are intensive computation and memory demanding. Their several layers, neurons per layer, multiple filters, and filter sizes have opened a new door in researching for taking them to embedded devices. For instance, a standard

CNN network like ResNet50 [7] requires up to 7.7 billion floating-point operations (FLOPs) and 25.6 million weights to classify a 224x224x3 image. Although there have been some advances in taking CNNs to the embedded world, it keeps being a challenge. Most of the embedded solutions rely on the cloud to process the input information, bringing their own shortcomings. Data privacy, network latency, security risk or the huge amount of devices raised in the IoT realm are some of the hurdles cloud computing is addressing. Therefore, developing suitable approaches for running these models locally, on the edge, instead of transferring to the cloud is a must. Furthermore, apart from intensive computation and memory requirements, energy efficiency becomes an added bottleneck since most of the edge applications run from batteries. Some solutions have been introduced in the literature to run CNNs at the edge. One of the most popular optimizations is based on model compression, which includes lower data precision [198], weight pruning [199], weight clustering sharing [200], or specific shrunken models [201]. On the other hand, some works have been focused on reducing the cost of the most expensive and repetitive operation in CNNs: the multiplication.

In this chapter, we introduce Stochastic Computing (SC) as a great candidate for accelerating CNNs. Due to its great advantages in saving power and area, mainly on multiplication, it has the potential to satisfy the low-power requirements for edge CNNs implementations.

5.2 Convolutional Neural Network

In its very base, CNNs are built of three main layers: convolutional layers, pooling layers, and fully-connected layers.

5.2.1 Convolutional Layer

Despite of its known name, the convolutional layer does not perform a convolution operation according to its formal definition. Instead, the operation used is the cross-correlation, which is the same as convolution but without flipping the kernel. The convolutional layer extracts a feature map from the input vector (which for the first layer is the input image) by performing a linear operation (dot product) and a non-linear transfer function (mainly the rectifier-linear-unit, ReLU). The operation is performed between a window of its input, called receptive field, and a kernel, commonly known as filter. The

kernel is a set of trainable parameters of the network (weights), which are shared among neighboring cells; in this way, mimicking how the visual cortex neurons operate [189]. Therefore, the same kernel values are used for several neurons of the same layer, saving plenty of memory space if compared to its Multi-Layer-Perceptron (MLP) counterpart. Moreover, the sharing kernel scheme in the same layer neurons produces the layer to have *equivariance to translation*, which means that a translation on the input features produces an equivalent translation on the output. Therefore, the pattern can be rotated or moved along the image space, producing the same results at the output.

The forward convolution can be seen as an operation with two different inputs: the input data and the kernel. The input data is a three dimensional vector $X \in \mathbb{R}^{HWC}$, being H and W the height and width of the input vector, and C the number of input channels. On the other hand, the kernel is a four dimensional vector $F \in \mathbb{R}^{RSCK}$, being R and S the height and width of the kernel weights, and K the number of kernels employed. The convolution of these two vectors produces a three dimensional output vector $Z \in \mathbb{R}^{PQK}$, being $P = f(H, R, u, pad_h)$ and $Q = f(W, S, v, pad_v)$ dependant functions on the image and kernel height-width, along with the padding and striding choices of the user. The vertical and horizontal striding u, v allow the user to reduce the computational process by moving the receptive field window in different vertical and horizontal steps, respectively. The padding parameters pad_h and pad_v, by their part, specify the number of columns and rows, respectively, appended with value 0 to the input vector. More specifically, P and Q are calculated as:

$$P = f(H, R, u, pad_h) = \left\lceil \frac{H - R + 1 + 2pad_h}{u} \right\rceil$$

$$Q = f(W, S, v, pad_v) = \left\lceil \frac{W - S + 1 + 2pad_v}{v} \right\rceil$$

Therefore, the output vector is calculated as:

$$Z[p, q, k] = \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} \sum_{c=0}^{C-1} F[r, s, c, k] \overset{0}{X}[p + r, q + s, c] + bias \qquad (5.1)$$

, $\forall p \in [0, P)$, $\forall q \in [0, Q)$, $\forall k \in [0, K)$, and $\overset{0}{X}$ being the zero padded version of X. Note that the *bias* term is added as in every standard neuron operation. As can be seen from Eq. (5.1), a 6 nested loop is performed, with 3

accumulation loops for iterating r, s, c and 3 output loops for iterating p, q, k. This causes the convolutional layer to be the most computational demanding.

Fig. 5.1 shows an example of how a feature map is constructed by convolving a $4 \times 4 \times C$ input (transparent cube) with 3 kernels of $3 \times 3 \times C$ (blue, green and red cubes in figure). For clarity purposes the C value is not represented in the figure. As shown, every kernel k produces a 2×2 feature map, which is stacked to form the next layer input. With local receptive fields, kernels can learn elementary visual features, and as the layer gets deeper, more complex features can be learned. As noted, every point in the feature map is the outcome of a standard neuron, which computes not only the dot product but the assigned non-linear activation function $\phi(Z[p, q, k])$, which is normally the ReLU.

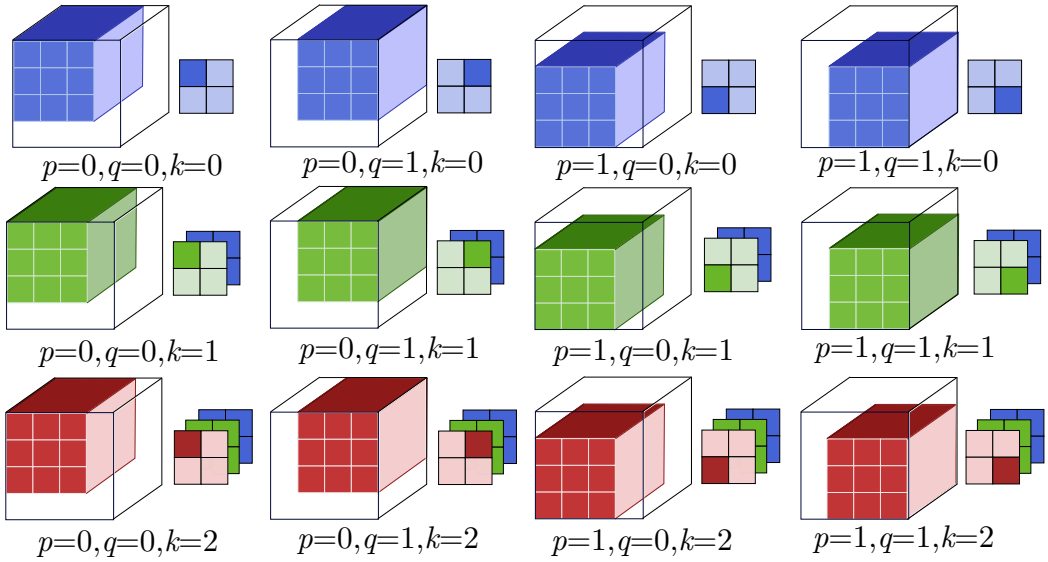


Figure 5.1: Discrete convolution computation for $H = W = 4$, $R = S = 3$, $K = 3$, $u = v = 1$, $pad_h = pad_v = 0$ for an arbitrary value of C.

5.2.2 Pooling Layer

Thereafter, a pooling layer is used. The pooling layer reduces the dimensional space of the feature map, therefore, reducing the computational effort for the next neural layers and reducing the probability of over-fitting when training. Moreover, it introduces a degree of local *translational invariance*,

allowing the networks to be more immune to rotation and movement of the patterns presented in their inputs [202]. This property is not to be confused with the *equivariance to translation* mentioned before, which means that a displacement on input values generates a displacement on output values as well; whereas *translational invariance* means that a displacement on input features produces no changes at all in the output.

Normally, three types of pooling are employed: the L2-Norm, average, or Max-Pooling (MP). All of them are applied to a typical square window of the input, reducing the input square window to a single value. For the L2-Norm, the n elements of the square window $x = [x_1, x_2, ..., x_n]$ are reduced to a single output $m = \sqrt{\sum_{i=1}^{n} x_i^2}$. For the average pooling, the output is calculated as $m = \frac{\sum_{i=1}^{n} x_i}{n}$. Finally, for the MP case, the operation performed is the maximum function $m = \max(x_1, x_2, ..., x_n)$. Empirical results show that the MP operation significantly outperforms other types of pooling [203], therefore, it is the most commonly used in community. Fig. 5.2 depicts how the MP is performed over a 4×4 input image with a 2×2 kernel window and an horizontal and vertical stride $u = v = 2$.

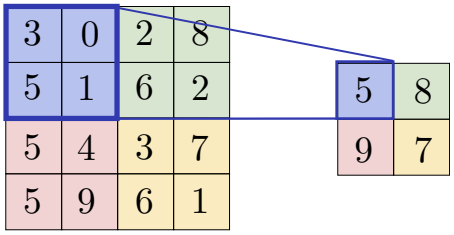


Figure 5.2: Max-pooling of a 4×4 input image. The 2×2 window is moved in steps of 2 (stride $u, v = 2$).

5.2.3 Fully-Connected Layer

After extensive feature extraction layers, convolutional plus pooling layers deliver a flatten feature map representation of the key characteristic of the input image to a categorizer. The fully-connected layer is the preferred choice for main applications. As discussed in Chapter 4, fully-connected layers have connections between current layer neuron inputs and all previous layer activations. This fact contributes to a great part of the amount of weights to be saved in memory by the application.

5.2.4 LeNet-5

Fig. 5.3 shows the LeNet-5 CNN architecture. As shown, it has 2 convolutional layers, 2 pooling layers and a fully-connected layer built by 3 stacked layers. Considering an image from the MNIST dataset, the input is a $28 \times 28 \times 1$ vector. Then, 6 kernels of $5 \times 5 \times 1$ are passed to the input to generate a $24 \times 24 \times 6$ feature map. No padding is applied. The 2×2 MP is then achieved to reduce the space dimension to $12 \times 12 \times 6$. Afterwards, another convolution is carried out with 16 kernels of $5 \times 5 \times 6$. This produces an $8 \times 8 \times 16$ feature map, since once again, no padding is employed. Then, a final 2×2 MP takes place to produce a $4 \times 4 \times 16$ feature map. This feature map is then flatten to 256 and inserted to the fully-connected categorizer, which is made of a stack of 128, 84, and 10 neurons.

Table 5.1 shows the resources and operations (OPs) for the LeNet-5 of Fig. 5.3. The number of weights does not include the bias. The number of operations is calculated taking one multiplication as a single operation and one addition as a single operation. For MP layers, we calculated a two-input maximum comparison as a single operation, having for a 2×2 MP a total of 3 operations.

As shown, 96% of the total neurons are employed in the feature extraction layers (Conv1 + MP1 + Conv2 + MP2), where 59.4% is spent only in the first convolutional layer. This clearly indicates that to increase efficiency in this network, these layers must be parallelized in some way. On the other hand, 94% of the weights are in the fully-connected layers, of which 70% are just in the first dense layer. This shows the clear advantage of using convolutional layers in comparison with fully-connected layers. Since the weights are shared, the parameters for these layers are decreased. Finally, convolutions are the most demanding in computation, conducting 85% of the total operations, with 54% just running in the second convolutional layer.

5.3 Related Works

Different works have been presented in the literature to implement CNNs based on SC. In essence, they differ in the design of the different base blocks carried out in the network. Table 5.2 shows a comparison of different design criteria among relevant implementations of the LeNet-5 using SC. We compared different SC-CNN features in each design, such as the RNG method,

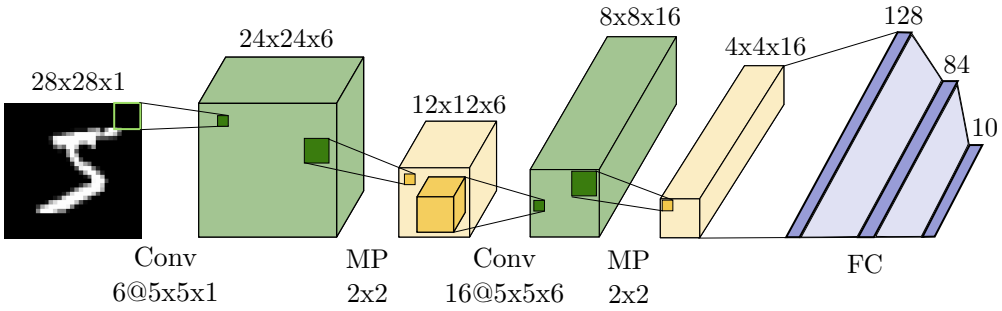


Figure 5.3: LeNet-5 CNN architecture. Two convolutional layers and two pooling layers constitute the feature extraction block. Then, a fully-connected layer classifies the inputs among 10 possible classes.

Table 5.1: LeNet-5 resources and operations per layer.

Layer	**Processing Elements**		**Weights**		**KOPs**	
	count	%	**count**	%	**count**	%
Conv1	3456	59.4	150	0.3	172.8	30.5
MP1	864	14.9	–	–	2.6	0.5
Conv2	1024	17.6	2400	5.4	307.2	54.2
MP2	256	4.4	–	–	0.8	0.1
fc1	120	2.1	30720	69.5	61.4	10.8
fc2	84	1.4	10080	22.8	20.2	3.6
fc3	10	0.2	840	1.9	1.7	0.3
Total	5814	100	44190	100	566.7	100

the multiplication coding (Mul), the addition circuit (Add), the activation function (Act), the pooling technique (Pool), the bit-width (BW), the bit-stream length (BS), the model accuracy (Score %), the testing platform (Pltf), and the number of layers (Layers) implemented. The score is provided for the test-set of the MNIST handwritten dataset, composed of 60k training images and 10k testing images of 28×28 pixels [204].

Back in 2017, there were different proposals for taking SC to reality for CNN applications. A. Ren et al. [30] presented different designs of SC neurons (SCD17 in the table). They analyzed the performance for different combinations of basic function blocks such as pooling circuits, BS lengths, and addition circuits. The best results were obtained by two designs. The first one is based on the average pooling and a mix of MUX and Approximate Accumulative Parallel Counter (AAPC) for addition (not shown in the table).

The average pooling is implemented by the classical scale addition using MUX circuits [13]. This implementation is cheap in SC, but it needs long BSs and a small number of inputs for not losing accuracy. For the addition block, they used MUXes for the first convolutional layer and AAPC for the rest of layers. AAPCs were firstly introduced in [25]. They consist of approximating the first layer of full-adders in the parallel counter, replacing them with combinatorial gates. This method has the advantage of reducing the total area employed for the expensive parallel counter, but with an increase in imprecision. For the activation function, they employed the FSM-based *tanh* [205]. With this design, they achieved a 96.64% of accuracy, only a -1.83% degradation with respect to the software one for a BS of 512. The second design uses the same blocks, but instead of the average-pooling, they used the MP. Their SC-MP is based on counting the ones in a segment of the integration period. In this way, they can guess which BS has the highest value (as explained in Chapter 2.3.3). This method is costly in terms of hardware resources and it is not exact, incurring in accuracy reduction. With this second design, they accomplished only a 0.21% of degradation with respect to the baseline. For both designs, they implemented all layers in SC using simulation; no real implementation was carried out. We displayed the MP design in the table as it approaches more to the standard architecture of nowadays CNNs.

In the same year, Z. Li et al. studied the effects in precision when changing the addition circuit and the arrangement in which the pooling and activation functions are connected [206](SDO17 in the table). For the addition circuit, they varied between MUX and AAPC, while for the pooling-activation arrangement they alternated the order between pooling blocks and the FSM-based *tanh*. For the RNG method, they implemented the LFSR shuffling technique presented in [207]. The best performance is obtained from the AAPC-MP-*tanh* configuration, run for 256 cycles, obtaining an accuracy degradation of 0.12% with respect to the software baseline. The accuracy outcome is acquired from the simulation of all the network layers.

An improvement of the first relevant design of A. Ren et al. [30] (the average pooling design) was introduced in the same year by J. Li et al. [113] (TADNN17 in the table). In the same manner as the A. Ren's work, the AAPC is used for accumulation, but in this occasion, it is employed for all the layers. For this experiment, they varied the BS to shorter lengths. Results showed that using the AAPC in all the layers improved the accuracy to the extent of having a 95.6% with only a 64 BS length. They concluded

that the MUX-based adder design is suitable for area-constraint systems, albeit it incurs a higher error. On the other hand, the AAPC-based design is suitable for energy constrained systems. However, the fact that the activation function kept being the *tanh* was a problem for CNN implementations. This activation function is not recommended for CNNs since it generates the vanishing problem in the training phase, as explained in Chapter 4.2. Moreover, the average-pooling is not generally used by the community as pooling technique; MP block is the preferred one as it shows better accuracy comportment. As in the case of A. Ren's work, they do not provide the RNG employed, which in turn let the open question of whether these works are feasible to implement in real circuits, since as studied in Chapter 3, RNG is a key component block in order to achieve good results. We thereby conclude that taking these designs to real implementation circuits is tough as one of the key components is obviated in the system.

J. Yu et al. provided another approach for implementing SC based CNNs [31] (AESC17 in the table). In this paper, they presented an efficient unipolar neuron with an SC-ReLU and MP design, both based on an FSM. The unipolar coding is selected in their design as it provides better accuracy than the bipolar coding (see Chapter 2.3.1). They took only the magnitude of the input and weights to operate; and since the sign of the multiplication depends only upon the fixed weight found in training (as the inputs are all positive), they separated the positive and negative results to do the accumulation. For addition, unlike previous works, they used the exact APC with no approximations. The maximum function, employed for the MP and ReLU, is based on the FSM-*tanh* design, as explained in Chapter 2.3.3. The RNG employed is the LFSR sharing technique introduced in [41], using a circular shift at the LFSR output to decorrelate the BSs. However, the sharing technique is only used in the convolutional layer and affects the SC-ReLU performance. Thus, to implement the model with accurate results, they needed to prune some connections in the network. The results are presented through simulation as preceding works, with a 99.19% of accuracy; only 0.04% of degradation with respect to the software model.

In the following year, Z. Li et al. proposed HEIF, an efficient SC based inference framework for DNNs [123] (HEIF18 in the table). In this work, they provided several improvements to previous works [30, 113]. They claimed the first SC-based ReLU design. Additionally, they improved the AAPC and optimized the multiplier by replacing XNOR gates with transmission gates and inverse mirror adders. Moreover, they provided a tape-out of a 64-

input feature extraction block for 8-bit and 16-bit precision. However, the test accuracy was provided by means of simulation. For the ReLU and MP design, they used a circuit based on counters and accumulators, clipping the ReLU output to one. For the RNG method, they implemented the LFSR shuffling technique presented in [207]. Their results were provided for an 128 BS length, with a 99.07% of accuracy; only a 0.10% of degradation with respect to the software outcome.

In 2019, S. Faraji et al. exploited the advantages offered by using low-discrepancy sequences for SC operations (EECNN19 in the table). They provided a low-latency and energy-efficient implementation of a CNN; although they only implemented the first convolutional layer in SC. In their work, they employed unipolar coding and an exact APC. The ReLU activation function, the MP, and the rest of the layers were implemented using Traditional Computing (TC) in 8-bit precision. Their results showed a 99.12% of accuracy by means of simulation (they did not provide the software base-line). The interesting feature of their method is that it only needs 8 cycles to operate, saving a significant amount of energy consumption compared to previous designs. Nevertheless, these results are only for the first convolutional layer. This fact leaves the question mark of the performance of the method for the whole network.

Finally, in 2020, P. Muthappa et al. provided a complete SC-CNN encompassing all network layers fully implemented over an FPGA device [208] (SCFPGA20 in the table). The system was run at 60MHz and could classify images in nearly 6ms. They used the unipolar coding for the multiplication and the exact APC for the addition. For the activation function, they implemented the FSM-based SC-sigmoid. The pooling design is the SC-MP introduced in [126], which uses combinatorial gates and counters to implement the maximum function. The RNG technique is the SBoNG method, presented in [39], which is based on an LFSR and a non-linear combinatorial circuit. This circuit decorrelates the LFSR outputs, allowing to be shared in the binary to stochastic conversion. For the design evaluation, they used a 512 BS length. The accuracy was 98.13%, only a 0.54% degradation with respect to the software result. Despite of being the unique of the list in performing a full real implementation, the fact of employing the sigmoid activation makes this approach to be of scarce use, as the ReLU function is the preferred choice in community. Moreover, the implementation only fits the first and the last layer in a fully parallel way, forcing to run the rest of the layers in a serial manner, therefore making the throughput having a low

execution: only 170 Img/sec.

The presented attempts to implement CNNs in SC face different hurdles when employing the commonly used blocks in the literature for real CNNs. Firstly, the amount of resources used to implement different RNGs to mitigate the correlation effects. Secondly, the accuracy degradation produced by the lack of full decorrelation between signals. And lastly, the stochastic maximum function circuit implementation for the SC version of the ReLU and MP. Tackling these issues is not trivial. Furthermore, excluding the P. Muthappa et al. work [208], all results are obtained by software simulations, and no real implementation in hardware is introduced.

In the next section, we propose an efficient reduced-area hardware architecture that overcomes the named hurdles. We exploit both the decorrelation and correlation between SC signals to implement the basic building blocks of a CNN in an efficient manner. To the best of our knowledge (at the time of writing this memory), this is the first time a fully parallel SC-CNN is implemented on a single chip. As a real proof of concept, we implemented our architecture on a single FPGA chip and compared its performance characteristics with different published FPGA implementation works. In addition, we synthesized our architecture over a VLSI circuit using a 250nm UMC and 40 nm TSMC technology, demonstrating an improvement over state-of-the-art VLSI designs of SC-CNN circuits.

Table 5.2: Related SC works for the LeNet-5 implementation.

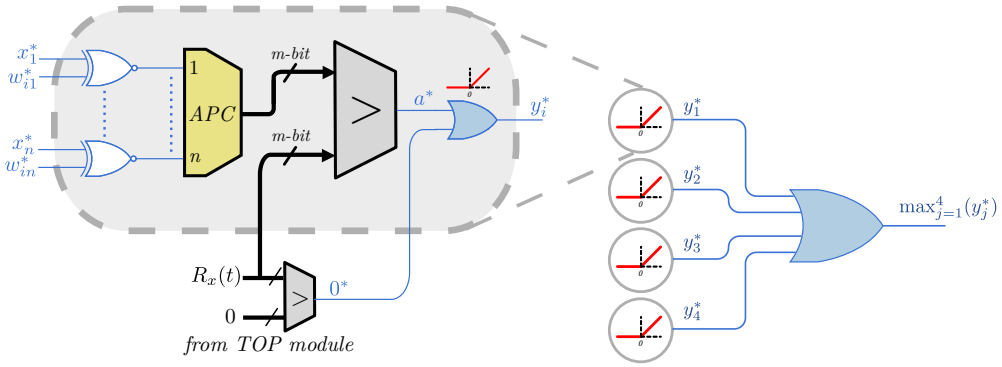
Method	SC-CNN features							Implementation				
								Score (%)			Pltf	Layers
	RNG	Mul	Add	Act	Pool	BW	BS	Sw	SC	Diff		
SCD17 [30]	–	Bipolar	AAPC	SC-Tanh FSM-based	SC-Max Counter-based	–	256	98.47	98.26	-0.21	Sim	All
SDO17 [206]	LFSR Shuffle [207]	Bipolar	AAPC	SC-Tanh FSM-based	SC-Max Counter-based	–	256	99.04	98.92	-0.12	Sim	All
TADNN17 [113]	–	Bipolar	AAPC	SC-Tanh FSM-based	SC-Avg MUX-based	–	64	98.46	95.6	-2.86	Sim	All
AESC17 [31]	LFSR CS [41]	Unipolar	APC	SC-ReLU FSM-based	SC-Max FSM-based	–	–	99.23	99.19	-0.04	Sim	All
HEIF18 [123]	LFSR Shuffle [207]	Bipolar	AAPC	SC-ReLU Counter-based	SC-Max Counter-based	–	128	99.17	99.07	-0.10	Sim	All
EECNN19 [56]	Sobol	Unipolar	APC	ReLU (TC)	Max (TC)	–	8	–	99.12	–	Sim	First
SCFPGA20 [208]	SBoNG [39]	Unipolar	APC	SC-Sigmoid FSM-based	SC-Max Counter-based [126]	9	512	98.67	98.13	-0.54	FPGA	All
This work	LFSR seeds	Bipolar	APC	SC-ReLU OR-based	SC-Max OR-based	8	510	99.05	98.89	-0.16	FPGA	All

5.4 SC-CNN exploiting correlation

As noted from Eq. (5.1), every point $Z[p, q, k]$ is calculated from a dot product between the input X and the weight set F plus a bias. This is the same operation carried out by a single neuron in the fully-connected layers, as noted in Eq. (4.1). Therefore, the convolutional layers are formed by standard neurons sharing the same weight parameters and having as input a $R \times S \times C$ window of the preceding layer. Thus, to implement the convolutional layer in SC domain, the SC neuron introduced in Section 4.3 can be used (see Fig. 5.4(a)). The design of this neuron allows to implement the exact SC-ReLU function with a single 2-input OR gate by exploiting the correlation effect. This is in contrast with previously mentioned works, where the design is focused on non-correlated inputs, generating the drawbacks aforementioned of area and imprecision. For the case of fully-connected layers, the same design is used indeed. This clearly helps the whole architecture design since only one neuron design is employed for the whole system.

If we consider the network architecture shown in Section 4.3, where all output neurons are totally correlated because they are generated by the same RNG, we can keep exploiting the correlation phenomenon for implementing the MP block. The design is straight forward, simple, and moreover, exact. Fig. 5.4(b) shows the SC-MP block, where a single 4-input OR gate is used to calculate the $\max_{j=1}^{4}(y_j)$.

Fig. 5.5 shows how the whole system is connected to reproduce a CNN design (the LeNet-5). As noted, only two unique pseudo-random number generators ($R_x(t)$ and $R_w(t)$) are needed to accomplish the overall calculus, considerably saving area and power in the design. This could be achieved thanks to the stochastic neuron design, which exploits correlation and decorrelation for computing. LFSR1 is used as RNG ($R_x(t)$) for generating the SC image, the zero reference signal, and every neuron APC output in the whole design. The domain conversion is done by means of Binary to SC converters (BSC). LFSR2, at the same time, is used as RNG ($R_w(t)$) for generating the SC weights. In this manner, each stochastic signal generated by the LFSR1 is totally uncorrelated with that generated by the LFSR2, allowing neuron inputs to be multiplied by weights with the highest precision. Moreover, the architecture proposed allows neuron outputs from layer l_i to be connected to the neuron inputs of the next layer l_{i+1} without any risk of signal degradation. Since the l_i neuron outputs are generated from the first LFSR block ($R_x(t)$), and the l_{i+1} weights are generated from the



(a) SC Neuron

(b) SC max-pooling

Figure 5.4: (a) Stochastic neuron design exploiting correlation to reduce area cost. Stochastic signal a^* and zero-*bipolar* (0^*) are generated with the same LFSR ($R_x(t)$) to produce total correlation between them, returning the ReLU function on the output with a single OR gate. (b) Stochastic max-pooling circuit for a spatial window size of k = 2x2. Stochastic neuron outputs y_k^* are totally correlated, allowing the implementation of the maximum function with a single OR gate. The generalization of this design for any window size is trivial. Stochastic signals are shown in blue color.

second LFSR ($R_w(t)$), the error induced from layer to layer by the appearance of uncontrolled correlation between signals is totally avoided. In every convolutional layer output, a stuck of 2×2 neurons plus a SC-MP block is connected, generating the next layer input. As noted by dashed lines, $R_x(t)$ and zero reference are shared through the whole network, saving plenty of resources and enabling all neurons to work simultaneously in parallel. Power consumption plummets since no access to memory for reading and writing intermediate results are necessary since the whole neuron is embedded fully parallel in the chip. Last row of Table 5.2 shows a summary of the block design modules and the implementation results.

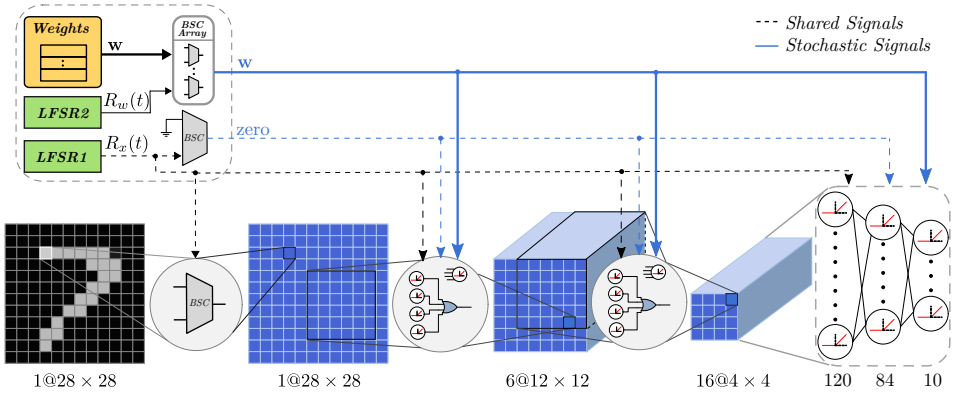


Figure 5.5: Fully-parallel stochastic CNN architecture. Only two unique pseudorandom number generators are employed. All neurons are working simultaneously in parallel thanks to the correlation phenomenon exploitation. 5×5 size kernels are used for convolution layers. Blue lines denote stochastic-coded signals, whereas black lines denote two's complement signals. Shared signals among neurons are depicted by dashed lines.

5.5 Experiments and Results

5.5.1 FEB Evaluation

The Feature Extraction Block (FEB) is defined as the union of convolutional and pooling neurons to generate a single feature point. This block is the base of every single convolutional layer and the minimum block required in case

Table 5.3: FEB performance comparison for pipelined and non-pipelined architectures.

Metric	**Pipelined**		**Non-pipelined**	
	HEIF18[123]	**This work**	**HEIF18[123]**	**This work**
Tech (nm)	45	40	45	40
Clk (ns)	2.08	1.13	2.67	1.8
Area (μm^2)	1569.4	2500	1453.9	1900
Power (μW)	928.4	430	659.9	220
Energy per Clk (fJ)	1931.0	485.9	1762.0	396

that a bigger CNN architecture is needed. For this reason, an efficient FEB will reverberate in the whole efficiency of the network.

As shown in Table 5.2, the proposed FEB design has three operations providing exact results, while occupying a low area in the pooling and activation function blocks since only a single OR gate is needed.

In Table 5.3, we compared the frequency, area, power and energy with respect to reference HEIF18[123] results. The 64-input FEB is made of 4 ReLU neurons of 16 inputs each and a 4-to-1 MP block. The design has been synthesized in TSMC 40nm CMOS technology using the Cadence Genus Tool. We have synthesized two FEB designs: with and without pipelining. The pipeline is accomplished by inserting some DFFs in critical paths of the design to improve the frequency of the system. Pipelining is essential if a more complex architecture is required: it allows to split the whole network into smaller processing elements that can fit in the device and operate in a sequential manner (tiling technique).

Comparing the two proposed FEB designs (pipelined vs non-pipelined), the pipeline optimization achieves 1.6x more clock speed, while increasing 1.3x the area, 1.9x the power, and 1.2x the energy. Comparing the proposed pipelined design with the proposal taken from HEIF18[123], this work presents a 1.8x increment in processing speed and 3.9x more energy efficiency. The advantage is produced mainly due to the exploitation of the correlation phenomenon, achieving an exact ReLU and MP functions while reducing the total circuitry path delay. The difference in area comes from the APC design used in [123] (an Approximate APC (AAPC) that is developed in [25]). The block is considerably smaller than an exact APC although it is more imprecise.

5.5.2 SC-CNN Evaluation

To evaluate the proposed SC design, we have implemented two different CNN architectures: the LeNet-5 and a 30M of operations CNN capable of processing the CIFAR-10 dataset. For the case of the LeNet-5 architecture, a fully-parallel FPGA implementation is carried out.

FPGA LeNet-5 Implementation

LeNet-5 CNN is oriented to processing the MNIST handwriting data set composed of 60k training images and 10k testing images [204]. The CNN architecture consists of two convolutional layers and three fully connected layers, as the original paper [188] by Lecun describes. The baseline score of the trained model was 98.6% (no special optimizations were introduced during training), and the stochastic model could get 97.6%, a 1% accuracy degradation compared to the software version. It means a satisfactory result, considering that no parameter fine tuning process was applied, just a simple weight normalization. We also trained the network employing the push-to-sigma technique explained in Section 4.4.2, using a $\sigma = 1.5$. The results for software and SC were 99.05% and 98.89%, respectively. As seen, only a 0.16% accuracy degradation is observed employing this technique.

We tested the full SC CNN implementation on a GIDEL PROC10A board (Fig. 5.6), which has an Intel 10AX115H3F34I2SG FPGA running the 8-bit SC implementation at 150 MHz. The communications were done through a PCI express bus.

Figure 5.6: Evaluation FPGA Board [179].

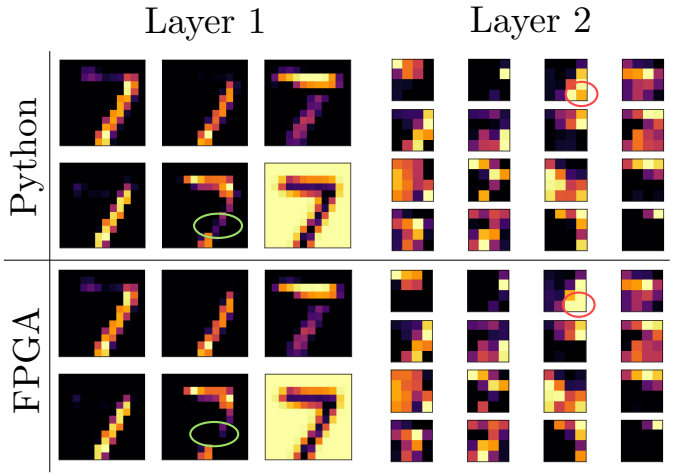


Figure 5.7: Feature maps comparison between software floating point implementation and FPGA measurements.

Fig. 5.7 shows the comparison between software feature maps extracted from the two pooling layers with the feature maps measured in the FPGA board. As shown, the results are practically the same except for some differences that have been highlighted. Foregone differences were expected if we take into consideration that the hardware computes in lower data resolution (8 bits) and the computations are not deterministic. Be that as it may, they do not affect the final outcome.

Table 5.4 shows the comparison between the proposed implementation and some Traditional Computing (TC) and SC FPGA-based CNN accelerators. We compared the methods in terms of throughput, performance, energy efficiency, and computation blocks employed.

Table 5.4: Comparison with other FPGA Lenet-5 Implementations.

Methods	FPGA17[209]	FPGA17[210]	FPGA18[211]	FPGA20[208]	This work
Year	2017	2017	2018	2020	2021
Architecture	Sequential (TC)	Sequential (TC)	Sequential (TC)	Semi-Parallel (SC)	Parallel (SC)
Activation/Weight precision (bits)	16/8	8/8	-	9/9	8/8
FPGA platform	Virtex7 VX690T	Virtex7 485t	Zynq ZC706	Zynq XC7Z020	Arria10 GX1150
Frequency (MHz)	100	-	166	60	150
Throughput (Images/s)	10617	1042	625	170	**294118**
Performance (Images/s/MHz)	106	-	4	2.83	**1961**
Power (W)	25.2	**0.47**	10.98	3.72	21
Energy efficiency (Images/J)	421	2216	57	45.7	**14006**
Logic used K (LUT or ALM)	233	7	40	27779	343
DSP (blocks)	2 907	574	59	0	0
Memory (Mbits)	17.17	12.37	3.49	1.73	0.00

As can be appreciated, the proposed method outperforms other architectures. The results show that the proposed stochastic CNN implementation achieves 28x more throughput and 18x more performance versus VX690T implementation [209]. In terms of energy efficiency, this work presents 6.3x increase compared to Virtex7-485t implementation [210], making it promising for real embedded system applications.

Comparing the proposed system with other SC FPGA implementation [208], the difference is notorious. The proposal is 1730x faster, achieves 692x more performance, and consumes 306x less energy per image. Although both implementations are based on SC, the difference lies mainly in the exploitation of the correlation phenomenon. This fact permits to realize the ReLU and MP functions in a highly compact way, decreasing drastically the area usage, and thus allowing a total parallelization of the model.

It is important to note that the different power estimations of these works only take into account the FPGA chip consumption.

However, the energy invested in memory access, which can be over the 80% of the energy consumed in a CNN accelerator [212], is omitted. Thus, considering that the proposed system is the only one which implements the model in a fully parallel way and without a permanent memory access, its comparison with other works can be considered as a worst-case scenario. This is one of the drawbacks of layer accelerator implementations, where you must save the output of the intermediate computations and then read them back to carry out the whole processing. By contrast, parallel pipelined architectures do not suffer from this phenomenon because all the parameters are embedded in the system, and the intermediate results are directly connected to the next layer, making the RAM transactions needless.

To the best of our knowledge, this is the first time an entire fully-parallel SC CNN is embedded in a single FPGA. This feature is in stark contrast to the studies presented, where the inference operations are realized by using a loop-tiling technique (an optimization approach to use the same hardware resources recursively).

In our design, DSPs blocks are avoided since an unconventional computing technique (Stochastic Computing) is used instead of traditional-binary logic. At the same time, memory blocks are not required since the computation is not performed in a tile-loop manner, thereby getting rid of the principal source of power consumption, which comes from the access operations to the memory.

VLSI LeNet-5 Implementation

The complete SC-CNN architecture has also been synthesized in TSMC 40nm CMOS technology and in UMC 250 nm technology using the Cadence Genus Tool. The implemented design comprises a total number of 913,906 combinatorial elementary cells (NAND, NOR and inverter gates) and 104,317 sequential cells.

The total area of the full design is 10.88 mm^2 in the UMC 250 nm technology node. The design synthesized in the TSMC 40nm takes up a total area of 2.01 mm^2 and consumes a $651mW$ operating at 200MHz.

Table 5.5 summarises and compares the performance of the synthesized SC-CNN LeNet5 with other implementations published in literature. Compared with state-of-the-art implementations of the LeNet5 using non traditional logic, the proposed system achieves 1.4x more computational density evaluated in $MOPS/mm^2$, 1.28x more throughput measured in $TOPS$, 1.58x more energy efficiency expressed in $TOPS/W$, 3.8x more area efficiency expressed in $TOPS/mm^2$, 10.4x more throughput measured in $Images/\mu s$, 2x more energy efficiency expressed in $Images/\mu J$, and 3.6x more area efficiency expressed in $Images/(\mu s \cdot mm^2)$, compared to the best reference of each case. This is due to the compact implementation of the ReLU function and MP operation by adequately exploiting the signal correlations. Furthermore, the use of correlated signals allows to implement the architecture by using a very reduced number of pseudorandom number generators.

Table 5.5: Comparison with other VLSI LENET-5 Implementation.

Methods	[213](4b)	[213](8b)	[30]	[123]	[214]	[215]	This work (2b)	This work (4b)	This work (8b)
Year	2020	2020	2017	2018	2019	2019	2021	2021	2021
Technology	40nm	40nm	45nm	45nm	45nm	65nm	40nm	40nm	40nm
Activation/Weight precision (bits)	4/1	8/1	8/8	7/7	16/16	2/2	2/2	4/4	8/8
Total Operations synthesized	19200	19200	4.59M	4.59M	-	-	566640	566640	566640
Latency(μ**s**)	52.7	842.67	1.28	0.31	-	37.50	0.03	0.15	2.56
Power (mW)	0.03	0.03	3530	2600	-	0.07	651	651	651
Area(mm^2)	0.124	0.124	36.4	22.9	-	0.006	2.01	2.01	2.01
Computational Density(MOPs/mm^2)	0.155	0.155	0.126	0.201	-	-	**0.282**	**0.282**	**0.282**
Normalized Throughput(TOPS)	$3.6 \cdot 10^{-4}$	$2.3 \cdot 10^{-5}$	3.59	14.72	-	0.01[a]	**18.89**	3.77	0.22
Normalized Energy Efficiency(TOPS/W)	12.08	0.76	1.02	5.66	-	18.36	**29.01**	5.80	0.34
Normalized Area Efficiency(TOPS/mm^2)	$2.9 \cdot 10^{-3}$	$1.8 \cdot 10^{-4}$	0.098	0.64	-	2.47	**9.40**	1.88	0.11
Absolute Throughput(Images/μ**s)**	-	-	0.78	3.20	-	0.03	**33.33**	6.67	0.39
Absolute Energy Efficiency(Images/μ**J)**	-	-	0.22	1.23	0.66	24.47	**51.20**	10.24	0.60
Absolute Area Efficiency(Images/(μ**s** $\cdot$ **mm^2))**	-	-	0.02	0.14	0.05	4.60	**16.58**	3.32	0.19

[a] Calculated from the total operations of the network: 536560

VLSI CIFAR-10 CNN Implementation

In addition, we also present the VLSI synthesis for a bigger CNN which can be able to process the CIFAR-10 dataset. CIFAR-10 consists of 60k 32 $\times$32 RGB images. The images are composed of real objects which can be categorized among 10 different classes. For training, we use 50k images and 10k are let for testing. The CNN architecture is formed by two blocks of two convolutional layers plus one MP, and two Fully Connected (FC) layers. All filters have a 3×3 sliding window with a stride of 1. The number of filters per convolutional layer is 32, 32, 64 and 64. No zero-padding is used. The MP is made of 2×2 sliding window with a stride of 2. Fully connected layers are built with 512 and 10 neurons. The accuracy for the SC implementation was 81% using 8-bit precision.

Table 5.6 shows the synthesis results for the CIFAR-10 CNN architecture for each layer using the same CMOS technology. As shown, the second convolutional layer is the most demanding in terms of area and power. This is due to the 47.6% of the total operations are executed in this layer.

We compared the performance of the proposed system for the two CNN architectures (CIFAR-10 and LeNeT-5) in Table 5.7. Unlike LeNet-5 implementation, which could get a 200MHz clock frequency, CIFAR-10 architecture, on account of its higher complexity, could only get 166.6MHz. This leads to a 1.2x longer latency, which consequently affects all metrics that depend on latency. Nevertheless, the throughput measured in $TOPS$ is 45x higher. This can be explained since 54x more operations are achieved in the CIFAR-10 CNN for the same number of cycles. This clearly shows the advantages of operating in SC, where the number of cycles per inference is independent of the number of operations computed (see the absolute throughput for both designs in Table 5.7).

Table 5.6: VLSI synthesis results for CIFAR-10 CNN architecture.

Layer	Conv 1	Conv 2	Conv 3	Conv 4	FC 1	FC 2	Total
Area(mm2)	17.22	54.8[a]	20.8	26.9	17.2	0.038	136.958
Delay(ns)	1.97	3.10[a]	3.07	3.36	5.50	2.73	5.50[b]
Power(mW)	1996[a]	11852[a]	4370	6049	2603	8.3	26878.3

[a] The value is taken from the intermediate synthesis reports.
[b] Taken from the longer path.

Table 5.7: CIFAR-10 CNN performance comparison.

Methods	This work (LeNeT-5)	This work (CIFAR-10)
Technology	40nm	40nm
Activation/Weight precision (bits)	8/8	8/8
Total Operations synthesized	566640	30.36M
Latency(μs)	2.56	3.06
Power (**mW**)	651	26878
Area(mm^2)	2.01	136.96
Computational Density(MOPs/mm^2)	0.282	0.222
Normalized Throughput (TOPS)	0.22	9.92
Normalized Energy Efficiency (TOPS/W)	0.34	0.37
Normalized Area Efficiency (TOPS/mm^2)	0.11	0.07
Absolute Throughput (Images/μs)	0.39	0.32
Absolute Energy Efficiency (Images/μJ)	0.60	0.01
Absolute Area Efficiency (Images/(μs $\cdot$ mm^2))	0.195	0.002

5.6 Summary

Thanks to the advantages of area shrinkage and low power consumption, stochastic computing is presented as a paradigm solution so as to implement deep learning algorithms in hardware for edge computing applications. Nevertheless, many difficulties are still being faced in the quest to achieve good results. In this paper, we present an efficient reduced-area architecture in an attempt to find a solution to the high area consumed by random number generators, the precision degradation produced by correlation between signals, and the stochastic maximum function implementation. For the first time, a fully-parallel stochastic computing-based CNN is embedded in a single FPGA chip, obtaining better performance results than traditional binary logic and other SC implementations, showing the compression effectiveness of the architecture by exploiting the correlation features presented by stochastic signals. The fully parallel SC-CNN has been synthesized in a VLSI circuit, demonstrating improved efficiency over previously reported SC-CNN VLSI implementations. Additionally, the synthesis result for a 30 million of operations CNN is presented, that could still be implemented in a single chip. This fact shows the benefits of the proposed design for implementing relatively complex edge-oriented CNN architectures in a compact and efficient way.

Chapter 6

CONCLUSIONS AND FUTURE WORK

This chapter presents the concluding remarks of the present work, the contributions made to the state-of-the-art, and the possible future work.

6.1 Conclusions

The main conclusions from the present work derive directly from the main objectives introduced in Chapter 1.2. Therein, it is explained in detailed the reasons behind such objectives and the hurdles they intend to circumvent in the area of knowledge.

Every objective was addressed in each chapter, being the main conclusions for each of them:

1. The SC implementation of the main operations needed for Machine Learning (ML) and Deep Learning (DL) applications is analyzed. This analysis was critical for all of the implementations presented, especially, for the Neural Network (NN) and Convolutional Neural Network (CNN) implementations. The correlation effects over SC signals were leveraged to implement high performance circuits and accomplish the remaining objectives.

2. It is demonstrated that the LFSR circuit keeps being the king of the Random Number Generators (RNGs) for real digital implementations in SC. We showed that if properly seeded, the LFSR is more accurate

than other well known RNGs present in the literature. The claiming has been proved for different SC functions and a real image processing application. These results were carried out in FPGA devices, not only simulations. Also, different guidelines are provided to SC designers for setting their LFSRs for different use case applications.

3. An efficient SC-based NN framework is designed and implemented in hardware for an interesting use-case: the drug discovery application. Based on the correlation phenomenon effects studied in Chapter 2, a high performance design is implemented which realized the SC-ReLU function using a single OR gate. The complete architecture needs only two LFSRs to compute the whole network, saving plenty of resources in the stochastic signal generation blocks. The SC-NN proposal for a Virtual Screening (VS) block (a pre-processing phase in the drug discovery realm) has been implemented. The design was tested over an FPGA and compare with some literature works. The proposed hardware design shows an increase in terms of processing speed, energy efficiency and accuracy. This fact shows the benefits of using SC for accelerating ML applications.

4. An efficient SC-based CNN framework is designed, proved it over an FPGA, and calculated its performance for a VLSI implementation. To the best of our knowledge (at the time this memory is written), this is the first time a fully-parallel SC-CNN is embedded completely in a single FPGA. A better performance is obtained with respect to traditional binary and SC FPGA-based implementations. The design was synthesized over a VLSI circuit and compared it against other VLSI chips, showing an improvement in overall performance. Moreover, the synthesis of a 30 million of operations CNN is presented, which could still be implemented in a single chip. This clearly demonstrates that SC is a feasible alternative to take the DL world to edge-computing applications.

We can conclude that the main objective, which was to implement specific Machine Learning (ML) and Deep Learning (DL) algorithms employing Stochastic Computing (SC) was accomplished successfully. Unlike many works from the literature, which present their results based only on simulations, we executed the proof of concept for all designs in real digital hardware platforms such as FPGAs. This shows the reality of the benefits presented in

the proposed designs. For the case of the more complex design in this work (the CNN), we also estimated the performance in VLSI, showing the gains against other works upon a high-demanding area of research as is the case of accelerating CNNs.

6.2 Future Work

Despite the main issues concerning the implementation in hardware of ML and DL algorithms using SC have been addressed, there are still many open doors to keep knocking. In this section, I highlight some possible lines to tackle, which I believe is worth the research effort.

As detailed in Chapter 2, the correlation phenomenon is a complication if it is not wisely manipulated. However, it correctly managed it can bring many advantages. As explained in Eq. (2.4), different correlation levels impacting the input signals in the same logic circuit can produce different intermediate operations. On the other hand, there are some researchers which adventured to train directly the model in SC in order to improve the accuracy [216, 217], showing interesting benefits. V. Lee et al. introduced some circuits to manipulate the correlation level of two stochastic signals [218]. Thus, I think that one exciting idea would be to train the SC model taking into account the correlation level as an extra hyperparameter. This can lead to surprising results with no additional hardware.

The Accumulative Parallel Counter (APC) is the most area consuming block in the neuron design presented in this thesis, with a 68% of the total neuron area invested only in the Parallel-Counter (PC) circuit. There is therefore a crucial necessity in boosting the efficiency of such circuit. One possible solution is to replace the digital PC with an analog counterpart. Lately, memristor technology has broken through the SC realm [219]. The memristor device operates as an accumulating counter driven by a number of electric inputs in a parallel way, similarly as the digital PC circuit. This proposed method could reduce the footprint area of the present SC neuron design, allowing to implement DL applications in small embedded devices.

The early-termination (or progressive-precision) is one relevant feature of SC systems. It says that with no modifications in hardware, we can reduce the latency of the system by terminating the processing earlier (less cycles). This property has led interesting improvements in energy consumption for SC designs [220]. Nevertheless, some inaccuracy is introduced when oper-

ating with- less cycles. Based on the results presented in Chapter 3, it is demonstrated that the best results are obtained when properly seeding the LFSR for different bit-widths, we suggest to design a RNG based on LFSRs. This RNG would be prepared to modify the tap selection of the LFSR based on the bit-width we want to operate. In this manner, with the same hardware, we can reduce the latency as the early-termination property suggests but with better accuracy outcomes.

We see plenty of future work in the line of SC-CNN implementations. Firstly, although the LeNet-5 is a good representative workload for the proposed method, it would be better to add a more generic network which can learn more complex datasets like ImageNet [190]. This will rise probably new difficulties which in turn will open new lines for researching. Secondly, the current SC-CNN implementation aims to leverage the natural parallelization of the NNs by fitting the whole network in an unrolled manner. This, as exhibited in the Chapter 5 results, enhances substantially the performance efficiency. However, for obvious reasons, a larger model like AlexNet [3], or mobileNet [201] could not fit entirely in a parallel form. Therefore, we see an important coarse of research in developing a rolled-type implementation, which would be more flexible to embed any type of network, and at the same time, exploits the advantages presented in the current design.

Finally, a VLSI synthesis of the existing SC-CNN implementation is presented. Nevertheless, we consider to realize a real tape-out of the chip in the quest to perform real measurements and confirm the robustness of the presented work.

Appendices

Appendix A

LFSR Tables

Table A.1 presents the seed indexes and values that must be employed in the second LFSR considering the first LFSR seed index is 0 for different bit-precision. "Best ADD index" column shows the best seed index for the scale addition operation. "Best MULT index" column shows the best seed index for the multiplication operation.

Table A.2 shows the tap configuration used in this work for the LFSR at different bit precision.

Table A.1: Best relative seed index for different use cases depending on the bit precision.

bit precision	Best ADD index	Best MULT index
4	3	2
5	4	3
6	5	23
7	6	52
8	7	97

Table A.2: Tap configuration used in the estimation of the best seeds.

bit precision	Polynomial	Binary Notation
4	$x^4 + x^3 + 1$	12
5	$x^5 + x^3 + 1$	20
6	$x^6 + x^5 + 1$	48
7	$x^7 + x^6 + 1$	96
8	$x^8 + x^6 + x^5 + x^4 + 1$	184

Appendix B

LFSR RTL Code

Listing B.1 shows the VHDL code for the LFSR implementation of this work. Different bit-precision can be synthesized modifying the generic parameter. The LFSR polynomials are set according to Table A.2.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY lfsr_nbits IS
generic (
    BIT_PRESICION : integer:=8 -- Num. within [4,8]
);
PORT(
    -- Inputs
    clk, rst : IN STD_LOGIC;
    i_en : IN STD_LOGIC;
    i_seed: in STD_LOGIC_VECTOR(BIT_PRESICION-1 downto 0);
    --Outputs
    o_sequence : OUT STD_LOGIC_VECTOR(BIT_PRECISION-1 downto 0)
);
END lfsr_nbits;

ARCHITECTURE arch OF lfsr_nbits IS
    SIGNAL lfsr: STD_LOGIC_VECTOR(BIT_PRECISION downto 1);
BEGIN
    PROCESS (clk, rst, i_seed, i_en)
    BEGIN
        IF rst ='1' THEN
            lfsr <= i_seed;
        ELSIF (i_en='1') THEN
```

```
            IF (clk'EVENT AND clk='1') THEN
                -- Shift Register
                for i in 1 to BIT_PRECISION-1 loop
                lfsr(i+1) <= lfsr(i);
                end loop;
                if BIT_PRECISION = 4 THEN
                lfsr(1) <= lfsr(4) XOR lfsr(3);
                elsif BIT_PRECISION = 5 THEN
                lfsr(1) <= lfsr(5) XOR lfsr(3);
                elsif BIT_PRECISION = 6 THEN
                lfsr(1) <= lfsr(6) XOR lfsr(5);
                elsif BIT_PRECISION = 7 THEN
                lfsr(1) <= lfsr(7) XOR lfsr(6);
                elsif BIT_PRECISION = 8 THEN
                lfsr(1) <= lfsr(8) XOR lfsr(6) XOR lfsr(5) XOR
                    lfsr(4);
                end if;
            END IF;
        END IF;
    END PROCESS;
    o_sequence <= lfsr;
END arch;
```

Listing B.1: VHDL code for the LFSR implementation of this work.

Bibliography

[1] L. B. Kish, "End of moore's law: thermal (noise) death of integration in micro and nano electronics," *Physics Letters A*, vol. 305, no. 3-4, pp. 144–149, 2002.

[2] Computer History Museum. (accessed June 17, 2021) Moore's law 40th anniversary with gordon moore. https://www.youtube.com/watch?v=MH6jUSjpr-Q.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[4] R. Fierro and F. L. Lewis, "Control of a nonholonomic mobile robot using neural networks," *IEEE transactions on neural networks*, vol. 9, no. 4, pp. 589–600, 1998.

[5] J. Mao and A. K. Jain, "Artificial neural networks for feature extraction and multivariate data projection," *IEEE transactions on neural networks*, vol. 6, no. 2, pp. 296–317, 1995.

[6] A. Morro, V. Canals, A. Oliver, M. L. Alomar, F. Galán-Prado, P. J. Ballester, and J. L. Rosselló, "A stochastic spiking neural network for virtual screening," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 4, pp. 1371–1375, 2017.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[8] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[9] E. Grefenstette, P. Blunsom, N. De Freitas, and K. M. Hermann, "A deep architecture for semantic parsing," *arXiv preprint arXiv:1404.7296*, 2014.

[10] A. Kouris and C.-S. Bouganis, "Learning to fly by myself: A self-supervised cnn-based approach for autonomous navigation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9.

[11] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3128–3137.

[12] A. Islam, "Technology scaling and its side effects," in *2015 19th International Symposium on VLSI Design and Test*, 2015, pp. 1–1.

[13] B. R. Gaines and Others, "Stochastic computing systems," *Advances in information systems science*, vol. 2, no. 2, pp. 37–172, 1969.

[14] V. Canals, C. F. Frasser, M. L. Alomar, A. Morro, A. Oliver, M. Roca, E. Isern, V. Martínez-Moll, E. Garcia-Moreno, and J. L. Rosselló, "Noise tolerant probabilistic logic for statistical pattern recognition applications," *Integrated Computer-Aided Engineering*, vol. 24, no. 4, pp. 351–365, 2017.

[15] C. F. Frasser, M. Roca, and J. L. Rosselló, "Optimal stochastic computing randomization," *Electronics*, vol. 10, no. 23, p. 2985, 2021.

[16] C. F. Frasser, C. de Benito, E. S. Skibinsky-Gitlin, V. Canals, J. Font-Rosselló, M. Roca, P. J. Ballester, and J. L. Rosselló, "Using stochastic computing for virtual screening acceleration," *Electronics*, vol. 10, no. 23, p. 2981, 2021.

[17] A. Morro, V. Canals, A. Oliver, M. L. Alomar, F. Galán-Prado, P. J. Ballester, and J. L. Rosselló, "A stochastic spiking neural network for virtual screening," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 4, pp. 1371–1375, April 2018.

[18] D. Wu, R. Yin, and J. S. Miguel, "Normalized stability: A cross-level design metric for early termination in stochastic computing," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 254–259.

[19] S. Liu and J. Han, "Energy efficient stochastic computing with sobol sequences," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 650–653.

[20] V. Canals, A. Morro, and J. L. Rosselló, "Stochastic-based pattern-recognition analysis," *Pattern recognition letters*, vol. 31, no. 15, pp. 2353–2356, 2010.

[21] C. Janer, J. Quero, J. G. Ortega, and L. G. Franquelo, "Fully parallel stochastic computation architecture," *IEEE Transactions on Signal Processing*, vol. 44, no. 8, pp. 2110–2117, 1996.

[22] S. Toral, J. Quero, and L. Franquelo, "Stochastic pulse coded arithmetic," in *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 1. IEEE, 2000, pp. 599–602.

[23] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE, 2013, pp. 39–46.

[24] T.-H. Chen and J. P. Hayes, "Design of division circuits for stochastic computing," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 116–121.

[25] K. Kim, J. Lee, and K. Choi, "Approximate de-randomizer for stochastic circuits," in *2015 International SoC Design Conference (ISOCC)*. IEEE, 2015, pp. 123–124.

[26] M. Lunglmayr, D. Wiesinger, and W. Haselmayr, "Design and analysis of efficient maximum/minimum circuits for stochastic computing," *IEEE Transactions on Computers*, vol. 69, no. 3, pp. 402–409, 2019.

[27] V. J. Canals Guinand *et al.*, "Implementación en hardware de sistemas de alta fiabilidad basados en metodologías estocásticas," Ph.D. dissertation, Universitat de les Illes Balears, 2017.

[28] E. E. Swartzlander, "Parallel counters," *IEEE Transactions on Computers*, vol. C-22, no. 11, pp. 1021–1024, 1973.

[29] E. Swartzlander, "A review of large parallel counter designs," in *IEEE Computer Society Annual Symposium on VLSI.* IEEE, 2004, pp. 89–98.

[30] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan, "SC-DCNN: Highly-scalable deep convolutional neural network using stochastic computing," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 405–418, 2017.

[31] J. Yu, K. Kim, J. Lee, and K. Choi, "Accurate and efficient stochastic computing hardware for convolutional neural networks," in *2017 IEEE International Conference on Computer Design (ICCD).* IEEE, 2017, pp. 105–112.

[32] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *2011 IEEE 29th International Conference on Computer Design (ICCD).* IEEE, 2011, pp. 154–161.

[33] P. Li, D. J. Lilja, W. Qian, M. D. Riedel, and K. Bazargan, "Logical computation on stochastic bit streams with linear finite-state machines," *IEEE Transactions on Computers*, vol. 63, no. 6, pp. 1474–1486, 2012.

[34] S. S. Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor, and W. J. Gross, "Majority-based tracking forecast memories for stochastic ldpc decoding," *IEEE Transactions on Signal Processing*, vol. 58, no. 9, pp. 4883–4896, 2010.

[35] Y. Zhang, R. Wang, X. Zhang, Z. Zhang, J. Song, Z. Zhang, Y. Wang, and R. Huang, "A parallel bitstream generator for stochastic computing," in *2019 Silicon Nanoelectronics Workshop (SNW).* IEEE, 2019, pp. 1–2.

[36] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2014, pp. 1–4.

[37] I. L. Dalal, D. Stefan, and J. Harwayne-Gidansky, "Low discrepancy sequences for monte carlo simulations on reconfigurable platforms," in *2008 International Conference on Application-Specific Systems, Architectures and Processors*. IEEE, 2008, pp. 108–113.

[38] P. Koopman, *Maximal length LFSR feedback terms*, (accessed Junuary, 2021). [Online]. Available: https://users.ece.cmu.edu/~koopman/lfsr/

[39] F. Neugebauer, I. Polian, and J. P. Hayes, "Building a better random number generator for stochastic computing," in *2017 Euromicro Conference on Digital System Design (DSD)*. IEEE, 2017, pp. 1–8.

[40] Z. Li, Z. Chen, Y. Zhang, Z. Huang, and W. Qian, "Simultaneous area and latency optimization for stochastic circuits by d flip-flop insertion," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1251–1264, 2018.

[41] H. Ichihara, S. Ishii, D. Sunamori, T. Iwagaki, and T. Inoue, "Compact and accurate stochastic circuits with shared random number sources," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*. IEEE, 2014, pp. 361–366.

[42] H. Joe and Y. Kim, "Novel stochastic computing for energy-efficient image processors," *Electronics*, vol. 8, no. 6, p. 720, 2019.

[43] J. H. Anderson, Y. Hara-Azumi, and S. Yamashita, "Effect of lfsr seeding, scrambling and feedback polynomial on stochastic computing accuracy," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 1550–1555.

[44] M. Abd Elaziz, A. H. Elsheikh, D. Oliva, L. Abualigah, S. Lu, and A. A. Ewees, "Advanced metaheuristic techniques for mechanical design problems," *Archives of Computational Methods in Engineering*, pp. 1–22, 2021.

[45] D. Oliva, M. Abd Elaziz, A. H. Elsheikh, and A. A. Ewees, "A review on meta-heuristics methods for estimating parameters of solar cells," *Journal of Power Sources*, vol. 435, p. 126683, 2019.

[46] A. Ajane, P. M. Furth, E. E. Johnson, and R. L. Subramanyam, "Comparison of binary and lfsr counters and efficient lfsr decoding algorithm," in *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2011, pp. 1–4.

[47] M. Baldi, M. Bianchi, N. Maturo, and F. Chiaraluce, "A physical layer secured key distribution technique for ieee 802.11g wireless networks," *IEEE Wireless Communications Letters*, vol. 2, no. 2, pp. 183–186, 2013.

[48] W. Liang and Long Jing, "A cryptographic algorithm based on linear feedback shift register," in *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, vol. 15, 2010, pp. V15–526–V15–529.

[49] W. B. Jone, J. C. Rau, S. C. Chang, and Y. L. Wu, "A tree-structured lfsr synthesis scheme for pseudo-exhaustive testing of vlsi circuits," in *Proceedings International Test Conference 1998 (IEEE Cat. No.98CH36270)*, 1998, pp. 322–330.

[50] P.-S. Ting and J. P. Hayes, "Isolation-based decorrelation of stochastic circuits," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 2016, pp. 88–95.

[51] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[52] N. A. Heckert, J. J. Filliben, C. M. Croarkin, B. Hembree, W. F. Guthrie, P. Tobias, J. Prinz *et al.*, "Handbook 151: Nist/sematech e-handbook of statistical methods," 2002.

[53] M. Gay, J. Burchard, J. Horácek, A.-S. Messeng Ekossono, T. Schubert, B. Becker, M. Kreuzer, and I. Polian, "Small scale aes toolbox: algebraic and propositional formulas, circuit-implementations and fault equations," 2016.

[54] J. Li, Z. Yuan, Z. Li, A. Ren, C. Ding, J. Draper, S. Nazarian, Q. Qiu, B. Yuan, and Y. Wang, "Normalization and dropout for stochastic computing-based deep convolutional neural networks," *Integration*, vol. 65, pp. 395–403, 2019.

[55] P. Metku, R. Seva, and M. Choi, "Energy-performance scalability analysis of a novel quasi-stochastic computing approach," *Journal of Low Power Electronics and Applications*, vol. 9, no. 4, p. 30, 2019.

[56] S. R. Faraji, M. H. Najafi, B. Li, D. J. Lilja, and K. Bazargan, "Energy-efficient convolutional neural networks with deterministic bit-stream processing," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1757–1762.

[57] H. Hsiao, J. S. Miguel, Y. Hara-Azumi, and J. Anderson, "Zero correlation error: A metric for finite-length bitstream independence in stochastic computing," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 260–265.

[58] P. Ting and J. P. Hayes, "Exploiting randomness in stochastic computing," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–6.

[59] R. K. Budhwani, R. Ragavan, and O. Sentieys, "Taking advantage of correlation in stochastic computing," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[60] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*, ser. DAC '13. New York, New York, USA: ACM Press, 2013, p. 1.

[61] G. P. Zhang, "Neural networks for classification: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 4, pp. 451–462, 2000.

[62] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[63] S. S. Haykin *et al.*, "Neural networks and learning machines/simon haykin." 2009.

[64] D. Hebb, "Organization of behavior. new york: Wiley," *J. Clin. Psychol*, vol. 6, no. 3, pp. 335–307, 1949.

[65] D. Purves, R. Cabeza, S. A. Huettel, K. S. LaBar, M. L. Platt, M. G. Woldorff, and E. M. Brannon, *Cognitive neuroscience*. Sunderland: Sinauer Associates, Inc, 2008.

[66] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[67] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[68] J. Trmal, J. Zelinka, and L. Müller, "Adaptation of a feedforward artificial neural network using a linear transform," in *International Conference on Text, Speech and Dialogue*. Springer, 2010, pp. 423–430.

[69] G. A. Carpenter and S. Grossberg, *Adaptive Resonance Theory*. Boston, MA: Springer US, 2010, pp. 22–35. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_11

[70] J. Liu, G. Jiang, Y. Bai, T. Chen, and H. Wang, "Understanding why neural networks generalize well through gsnr of parameters," *arXiv preprint arXiv:2001.07384*, 2020.

[71] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro, "Exploring generalization in deep learning," *arXiv preprint arXiv:1706.08947*, 2017.

[72] V. Kondratenko and Y. A. Kuperin, "Using recurrent neural networks to forecasting of forex," *arXiv preprint cond-mat/0304469*, 2003.

[73] A.-S. Chen, M. T. Leung, and H. Daouk, "Application of neural networks to an emerging financial market: forecasting and trading the taiwan stock index," *Computers & Operations Research*, vol. 30, no. 6, pp. 901–923, 2003.

[74] A. P. Adewole, A. T. Akinwale, and A. B. Akintomide, "Artificial neural network model for forecasting foreign exchange rate," 2011.

[75] M. Khashei, M. Bijari, and G. A. R. Ardali, "Hybridization of autoregressive integrated moving average (arima) with probabilistic neural

networks (pnns)," *Computers & Industrial Engineering*, vol. 63, no. 1, pp. 37–45, 2012.

[76] J.-Q. Huang and F. L. Lewis, "Neural-network predictive control for nonlinear dynamic systems with time-delay," *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 377–389, 2003.

[77] K.-i. Minami, H. Nakajima, and T. Toyoshima, "Real-time discrimination of ventricular tachyarrhythmia with fourier-transform neural network," *IEEE transactions on Biomedical Engineering*, vol. 46, no. 2, pp. 179–185, 1999.

[78] K. U. Rani, "Analysis of heart diseases dataset using neural network approach," *arXiv preprint arXiv:1110.2626*, 2011.

[79] M. F. Othman and M. A. M. Basri, "Probabilistic neural network for brain tumor classification," in *2011 Second International Conference on Intelligent Systems, Modelling and Simulation*. IEEE, 2011, pp. 136–138.

[80] K. D. Kharat, P. P. Kulkarni, and M. Nagori, "Brain tumor classification using neural network based methods," *International Journal of Computer Science and Informatics*, vol. 1, no. 4, 2012.

[81] M. Stella, D. Begusic, and M. Russo, "Adaptive noise cancellation based on neural network," in *2006 International Conference on Software in Telecommunications and Computer Networks*. IEEE, 2006, pp. 306–309.

[82] L. B. Fah, A. Hussain, and S. A. Samad, "Speech enhancement by noise cancellation using neural network," in *2000 TENCON Proceedings. Intelligent Systems and Technologies for the New Millennium (Cat. No. 00CH37119)*, vol. 1. IEEE, 2000, pp. 39–42.

[83] T. Zhu-Mei and W. Ai-Zhen, "The research of adaptive noise cancellation technology based on neural network," in *2012 International Conference on Computing, Measurement, Control and Sensor Network*. IEEE, 2012, pp. 144–147.

[84] S. Tamura and M. Nakamura, "Improvements to the noise reduction neural network," in *International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 1990, pp. 825–828.

[85] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[86] H. Sak, A. Senior, K. Rao, and F. Beaufays, "Fast and accurate recurrent neural network acoustic models for speech recognition," *arXiv preprint arXiv:1507.06947*, 2015.

[87] H. Wu, J. Soraghan, A. Lowit, and G. Di Caterina, "Convolutional neural networks for pathological voice detection," in *2018 40th annual international conference of the ieee engineering in medicine and biology society (EMBC)*. IEEE, 2018, pp. 1–4.

[88] J. I. Godino-Llorente and P. Gómez-Vilda, "Automatic detection of voice impairments by means of short-term cepstral parameters and neural network based detectors," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 2, pp. 380–384, 2004.

[89] A. Sehgal and N. Kehtarnavaz, "A convolutional neural network smartphone app for real-time voice activity detection," *IEEE Access*, vol. 6, pp. 9017–9026, 2018.

[90] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[91] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 20, no. 1, pp. 23–38, 1998.

[92] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*, 2015.

[93] Q. Rao and J. Frtunikj, "Deep learning for self-driving cars: Chances and challenges," in *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, 2018, pp. 35–38.

[94] S. Chen, S. Zhang, J. Shang, B. Chen, and N. Zheng, "Brain-inspired cognitive model with attention for self-driving cars," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 11, no. 1, pp. 13–25, 2017.

[95] T. Troudet, S. Garg, and W. Merrill, "Neural network application to aircraft control system design," in *Navigation and Control Conference*, 1991, p. 2715.

[96] S. F. Adra, A. I. Hamody, I. Griffin, and P. J. Fleming, "A hybrid multi-objective evolutionary algorithm using an inverse neural network for aircraft control system design," in *2005 IEEE Congress on Evolutionary Computation*, vol. 1. IEEE, 2005, pp. 1–8.

[97] L. Molnár and G. M. Keserű, "A neural network based virtual screening of cytochrome p450 3a4 inhibitors," *Bioorganic & Medicinal Chemistry Letters*, vol. 12, no. 3, pp. 419–421, 2002.

[98] W. Shan, X. Li, H. Yao, and K. Lin, "Convolutional neural network-based virtual screening," *Current Medicinal Chemistry*, 2021.

[99] J. L. Melville, E. K. Burke, and J. D. Hirst, "Machine learning in virtual screening," *Combinatorial chemistry & high throughput screening*, vol. 12, no. 4, pp. 332–343, 2009.

[100] O. Roche, G. Trube, J. Zuegge, P. Pflimlin, A. Alanine, and G. Schneider, "A virtual screening method for prediction of the herg potassium channel liability of compound libraries," *ChemBioChem*, vol. 3, no. 5, pp. 455–459, 2002.

[101] P. J. Ballester and J. B. Mitchell, "A machine learning approach to predicting protein–ligand binding affinity with applications to molecular docking," *Bioinformatics*, vol. 26, no. 9, pp. 1169–1175, 2010.

[102] B. E. Boser, E. Sackinger, J. Bromley, Y. Le Cun, and L. D. Jackel, "An analog neural network processor with programmable topology,"

IEEE Journal of Solid-State Circuits, vol. 26, no. 12, pp. 2017–2025, 1991.

[103] L. Gatet, H. Tap-Béteille, and F. Bony, "Comparison between analog and digital neural network implementations for range-finding applications," *IEEE transactions on neural networks*, vol. 20, no. 3, pp. 460–470, 2009.

[104] C. Geng, Q. Sun, and S. Nakatake, "Implementation of perceptron as an essential element of configurable neural networks," *Sensors*, vol. 20, no. 15, p. 4222, 2020.

[105] P. W. Hollis and J. J. Paulos, "Artificial neural networks using mos analog multipliers," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 3, pp. 849–855, 1990.

[106] G. Indiveri and T. K. Horiuchi, "Frontiers in neuromorphic engineering," *Frontiers in neuroscience*, vol. 5, p. 118, 2011.

[107] S. Ryckebusch, J. M. Bower, and C. Mead, "Modeling small oscillating biological networks in analog vlsi," 1989.

[108] S. Le Masson, A. Laflaquiere, T. Bal, and G. Le Masson, "Analog circuits for modeling biological neural networks: design and applications," *IEEE transactions on biomedical engineering*, vol. 46, no. 6, pp. 638–645, 1999.

[109] R. D. Ledwith and J. F. Miller, "Introducing flexibility in digital circuit evolution: exploiting undefined values in binary truth tables," in *International Conference on Evolvable Systems*. Springer, 2010, pp. 25–36.

[110] T. MS Jr, D. J. Walker, and M. A. Sivilotti, "A digital neural network architecture for vlsi," in *1990 IJCNN International Joint Conference on Neural Networks*. IEEE, 1990, pp. 545–550.

[111] V. Canals, A. Morro, A. Oliver, M. L. Alomar, and J. L. Rossello, "A New Stochastic Computing Methodology for Efficient Neural Network Implementation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 3, pp. 551–564, mar 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7093194/

[112] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[113] J. Li, A. Ren, Z. Li, C. Ding, B. Yuan, Q. Qiu, and Y. Wang, "Towards acceleration of deep convolutional neural networks using stochastic computing," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 2017, pp. 115–120.

[114] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1515–1531, 2017.

[115] M. Nakagawa, "An artificial neuron model with a periodic activation function," *Journal of the Physical society of Japan*, vol. 64, no. 3, pp. 1023–1031, 1995.

[116] J. Lv, C. Guo, Z.-p. Shen, M. Zhao, and Y. Zhang, "Summary of artificial neuron model research," in *IECON 2007-33rd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2007, pp. 677–682.

[117] M. J. Powell, "Radial basis functions for multivariable interpolation: a review," *Algorithms for approximation*, 1987.

[118] B. Fornberg, E. Larsson, and N. Flyer, "Stable computations with gaussian radial basis functions," *SIAM Journal on Scientific Computing*, vol. 33, no. 2, pp. 869–892, 2011.

[119] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[120] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[121] K. Hara, D. Saito, and H. Shouno, "Analysis of function of rectified linear unit used in deep learning," in *2015 international joint conference on neural networks (IJCNN)*. IEEE, 2015, pp. 1–8.

[122] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[123] Z. Li, J. Li, A. Ren, R. Cai, C. Ding, X. Qian, J. Draper, B. Yuan, J. Tang, Q. Qiu *et al.*, "Heif: Highly efficient stochastic computing-based inference framework for deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 8, pp. 1543–1556, 2018.

[124] B. Li, Y. Qin, B. Yuan, and D. J. Lilja, "Neural network classifiers using stochastic computing with a hardware-oriented approximate activation function," *2017 IEEE International Conference on Computer Design (ICCD)*, pp. 97–104, 2017.

[125] J. Li, Z. Yuan, Z. Li, C. Ding, A. Ren, Q. Qiu, J. Draper, and Y. Wang, "Hardware-driven nonlinear activation for stochastic computing based deep convolutional neural networks," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 1230–1236.

[126] F. Neugebauer, I. Polian, and J. P. Hayes, "On the maximum function in stochastic computing," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, 2019, pp. 59–66.

[127] M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing stochastic computation deterministically," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2925–2938, 2019.

[128] P. K. Gupta and R. Kumaresan, "Binary multiplication with pn sequences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 4, pp. 603–606, 1988.

[129] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[130] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *arXiv preprint arXiv:1506.02626*, 2015.

[131] T. Hoffmann and M. Gastreich, "The next level in chemical space navigation: going far beyond enumerable compound libraries," *Drug Discovery Today*, vol. 24, no. 5, pp. 1148–1156, 2019.

[132] Y. O. Adeshina, E. J. Deeds, and J. Karanicolas, "Machine learning classification can reduce false positives in structure-based virtual screening," *Proceedings of the National Academy of Sciences*, vol. 117, no. 31, pp. 18 477–18 488, 2020.

[133] L. Fresnais and P. J. Ballester, "The impact of compound library size on the performance of scoring functions for structure-based virtual screening," *Briefings in Bioinformatics*, vol. 22, no. 3, 06 2020.

[134] A. Lavecchia and C. Giovanni, "Virtual screening strategies in drug discovery: A critical review," *Current Medicinal Chemistry*, vol. 20, no. 23, pp. 2839–2860, 2013.

[135] N. Singh, L. Chaput, and B. Villoutreix, "Virtual screening web servers: designing chemical probes and drug candidates in the cyberspace," *Brief. Bioinform. In Press*, 2020.

[136] E. Glaab, "Building a virtual ligand screening pipeline using free software: A survey," *Briefings in Bioinformatics*, vol. 17, no. 2, pp. 352–366, 2016.

[137] G. Ghislat, T. Rahman, and P. J. Ballester, "Recent progress on the prospective application of machine learning to structure-based virtual screening," *Current Opinion in Chemical Biology*, vol. 65, pp. 28–34, 2021.

[138] H. Li, K.-H. Sze, G. Lu, and P. Ballester, "Machine-learning scoring functions for structure-based virtual screening," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 2020.

[139] M. Batool, B. Ahmad, and S. Choi, "A structure-based drug discovery paradigm," *International Journal of Molecular Sciences*, vol. 20, no. 11, 2019.

[140] L. Pinzi and G. Rastelli, "Molecular docking: Shifting paradigms in drug discovery," *International Journal of Molecular Sciences*, vol. 20, no. 18, 2019.

[141] V. Zoete, A. Daina, C. Bovigny, and O. Michielin, "Swisssimilarity: A web tool for low to ultra high throughput ligand-based virtual screening," *Journal of Chemical Information and Modeling*, vol. 56, no. 8, pp. 1399–1404, 2016.

[142] H. Li, K.-S. Leung, M.-H. Wong, and P. Ballester, "Usr-vs: a web server for large-scale prospective virtual screening using ultrafast shape recognition techniques," *Nucleic acids research*, vol. 44, no. W1, pp. W436–W441, 2016.

[143] A. Kumar and K. Zhang, "Advances in the development of shape similarity methods and their application in drug discovery," *Frontiers in Chemistry*, vol. 6, no. JUL, 2018.

[144] B. Neves, R. Braga, C. Melo-Filho, J. Moreira-Filho, E. Muratov, and C. Andrade, "Qsar-based virtual screening: Advances and applications in drug discovery," *Frontiers in Pharmacology*, vol. 9, no. NOV, 2018.

[145] O. Soufan, W. Ba-Alawi, A. Magana-Mora, M. Essack, and V. Bajic, "Dpubchem: A web tool for qsar modeling and high-throughput virtual screening," *Scientific Reports*, vol. 8, no. 1, 2018.

[146] A. Speck-Planche, V. Kleandrova, F. Luan, and M. Cordeiro, "Chemoinformatics in anti-cancer chemotherapy: Multi-target qsar model for the in silico discovery of anti-breast cancer agents," *European Journal of Pharmaceutical Sciences*, vol. 47, no. 1, pp. 273–279, 2012.

[147] I. Olier, N. Sadawi, G. Bickerton, J. Vanschoren, C. Grosan, L. Soldatova, and R. King, "Meta-qsar: a large-scale application of meta-learning to drug design and discovery," *Machine Learning*, vol. 107, no. 1, pp. 285–311, 2018.

[148] P. Sidorov, S. Naulaerts, J. Ariey-Bonnet, E. Pasquier, and P. J. Ballester, "Predicting synergism of cancer drug combinations using nci-almanac data," *Frontiers in chemistry*, vol. 7, pp. 509–509, 07 2019. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/31380352

[149] A. Lavecchia, "Machine-learning approaches in drug discovery: Methods and applications," *Drug Discovery Today*, vol. 20, no. 3, pp. 318–331, 2015.

[150] M. Azghadi, B. Linares-Barranco, D. Abbott, and P. Leong, "A hybrid cmos-memristor neuromorphic synapse," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 11, no. 2, pp. 434–445, 2017.

[151] C. Frenkel, J.-D. Legat, and D. Bol, "Morphic: A 65-nm 738k-synapse/mm2 quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 5, pp. 999–1010, 2019.

[152] W. Guo, H. Yantır, M. Fouda, A. Eltawil, and K. Salama, "Towards efficient neuromorphic hardware: Unsupervised adaptive neuron pruning," *Electronics (Switzerland)*, vol. 9, no. 7, pp. 1–15, 2020. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85087122671&doi=10.3390%2felectronics9071059&partnerID=40&md5=1546bd3007b2013ef9a05c409ac14753

[153] H. Son, H. Cho, J. Lee, S. Bae, B. Kim, H.-J. Park, and J.-Y. Sim, "A multilayer-learning current-mode neuromorphic system with analog-error compensation," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 5, pp. 986–998, 2019.

[154] M. Kang, Y. Lee, and M. Park, "Energy efficiency of machine learning in embedded systems using neuromorphic hardware," *Electronics (Switzerland)*, vol. 9, no. 7, pp. 1–10, 2020. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85087373839&doi=10.3390%2felectronics9071069&partnerID=40&md5=b83d3cf06b068ecd5ee67fb68f40514e

[155] A. Morro, V. Canals, A. Oliver, M. Alomar, F. Galan-Prado, P. Ballester, and J. Rossello, "A stochastic spiking neural network for virtual screening," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 4, pp. 1371–1375, 2018.

[156] I. Nascimento, R. Jardim, and F. Morgado-Dias, "A new solution to the hyperbolic tangent implementation in hardware: Polynomial modeling of the fractional exponential part," *Neural Computing and Applications*, vol. 23, no. 2, pp. 363–369, 2013.

[157] M. Carrasco-Robles and L. Serrano, "Accurate differential tanh(nx) implementation," *International Journal of Circuit Theory and Applications*, vol. 37, no. 5, pp. 613–629, 2009.

[158] B. Liu, D. Zou, L. Feng, S. Feng, P. Fu, and J. Li, "An fpga-based cnn accelerator integrating depthwise separable convolution," *Electronics (Switzerland)*, vol. 8, no. 3, 2019.

[159] M. Zhang, L. Li, H. Wang, Y. Liu, H. Qin, and W. Zhao, "Optimized compression for implementing convolutional neural networks on fpga," *Electronics (Switzerland)*, vol. 8, no. 3, 2019.

[160] A. Alaghi and J. Hayes, "Survey of stochastic computing," *Transactions on Embedded Computing Systems*, vol. 12, no. 2 SUPPL., 2013.

[161] V. Canals, A. Morro, and J. Rosselló, "Stochastic-based pattern-recognition analysis," *Pattern Recognition Letters*, vol. 31, no. 15, pp. 2353–2356, 2010. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-77956392632&doi=10.1016%2fj.patrec.2010.07.008&partnerID=40&md5=5dd142171e7b952e3241cbc513285252

[162] M. Faix, R. Laurent, P. Bessiere, E. Mazer, and J. Droulez, "Design of stochastic machines dedicated to approximate bayesian inferences," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 1, pp. 60–66, 2019. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85029498627&doi=10.1109%2fTETC.2016.2609926&partnerID=40&md5=1791fedd2ede37d87b0e5edb212a69b8

[163] H. Joe and Y. Kim, "Novel stochastic computing for energy-efficient image processors," *Electronics (Switzerland)*, vol. 8, no. 6, 2019.

[164] S. Xiao, W. Liu, Y. Guo, and Z. Yu, "Low-cost adaptive exponential integrate-and-fire neuron using stochastic computing," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 5, pp. 942–950, 2020.

[165] J. L. Rosselló, V. Canals, and A. Morro, "Hardware implementation of stochastic-based neural networks," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1–4.

[166] J. Tomlinson, M.S., D. Walker, and M. Sivilotti, "A digital neural network architecture for VLSI," in *1990 IJCNN International Joint Conference on Neural Networks*. IEEE, 1990, pp. 545–550 vol.2. [Online]. Available: http://ieeexplore.ieee.org/document/5726723/

[167] G. Moon, M. Zaghloul, and R. Newcomb, "VLSI implementation of synaptic weighting and summing in pulse coded neural-type cells," *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 394–403, may 1992. [Online]. Available: http://ieeexplore.ieee.org/document/129412/

[168] S. Sato, M. Yumine, T. Yama, J. Murota, K. Nakajima, and Y. Sawada, "LSI implementation of pulse-output neural network with programmable synapse," in *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, vol. 1. IEEE, pp. 172–177. [Online]. Available: http://ieeexplore.ieee.org/document/287140/

[169] A. Yousefzadeh, E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, "On Practical Issues for Stochastic STDP Hardware With 1-bit Synaptic Weights," *Frontiers in Neuroscience*, vol. 12, oct 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00665/full

[170] S. Fox, J. Faraone, D. Boland, K. Vissers, and P. H. Leong, "Training Deep Neural Networks in Low-Precision with High Accuracy Using FPGAs," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, dec 2019, pp. 1–9. [Online]. Available: https://ieeexplore.ieee.org/document/8977908/

[171] H. Ichihara, S. Ishii, D. Sunamori, T. Iwagaki, and T. Inoue, "Compact and accurate stochastic circuits with shared random number sources," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, 2014, pp. 361–366.

[172] A. Oliver, V. Canals, and J. Rosselló, "A bayesian target predictor method based on molecular pairing energies estimation," *Scientific Reports*, vol. 7, 2017.

[173] M. Mysinger, M. Carchia, J. Irwin, and B. Shoichet, "Directory of useful decoys, enhanced (dud-e): Better ligands and decoys for better benchmarking," *Journal of Medicinal Chemistry*, vol. 55, no. 14, pp. 6582–6594, 2012.

[174] G. Sliwoski, S. Kothiwale, J. Meiler, and E. Lowe Jr., "Computational methods in drug discovery," *Pharmacological Reviews*, 2014.

[175] J. Gasteiger and M. Marsili, "Iterative partial equalization of orbital electronegativity-a rapid access to atomic charges," *Tetrahedron*, 1980.

[176] T. Halgren, "Merck molecular force field. i. basis, form, scope, parameterization, and performance of mmff94," *Journal of Computational Chemistry*, vol. 17, no. 5-6, pp. 490–519, 1996.

[177] M. M. Mysinger, M. Carchia, J. J. Irwin, and B. K. Shoichet, "Directory of useful decoys, enhanced (dud-e): Better ligands and decoys for better benchmarking," *Journal of Medicinal Chemistry*, vol. 55, no. 14, pp. 6582–6594, 2012, pMID: 22716043. [Online]. Available: https://doi.org/10.1021/jm300687e

[178] A. Yaguchi, T. Suzuki, W. Asano, S. Nitta, Y. Sakata, and A. Tanizawa, "Adam Induces Implicit Weight Sparsity in Rectifier Neural Networks," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, dec 2018, pp. 318–325. [Online]. Available: https://ieeexplore.ieee.org/document/8614079/

[179] Gidel company. (accessed June 10, 2020) Proc10a board image. https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/3/boardimage-us-dsnbk-3-3405483112768-proc10agxplatform.jpg.

[180] A. Cleves, S. Johnson, and A. Jain, "Electrostatic-field and surface-shape similarity for virtual screening and pose prediction," *Journal of Computer-Aided Molecular Design*, vol. 33, no. 10, pp. 865–886, 2019.

[181] M. von Behren and M. Rarey, "Ligand-based virtual screening under partial shape constraints," *Journal of Computer-Aided Molecular Design*, vol. 31, no. 4, pp. 335–347, 2017.

[182] D. R. Koes and C. J. Camacho, "Shape-based virtual screening with volumetric aligned molecular shapes," *Journal of Computational Chemistry*, vol. 35, no. 25, pp. 1824–1834, 2014.

[183] P. J. Ballester and W. G. Richards, "Ultrafast shape recognition to search compound databases for similar molecular shapes," *Journal of Computational Chemistry*, vol. 28, no. 10, pp. 1711–1723, jul 2007. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/jcc.20681

[184] S. Puertas-Martín, J. L. Redondo, P. M. Ortigosa, and H. Pérez-Sánchez, "OptiPharm: An evolutionary algorithm to compare shape similarity," *Scientific Reports*, vol. 9, no. 1, p. 1398, dec 2019. [Online]. Available: http://www.nature.com/articles/s41598-018-37908-6

[185] T. N. Wiesel and D. H. Hubel, "Single-cell responses in striate cortex of kittens deprived of vision in one eye," *Journal of neurophysiology*, vol. 26, no. 6, pp. 1003–1017, 1963.

[186] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[187] Y.-T. Zhou, R. Chellappa, A. Vaid, and B. K. Jenkins, "Image restoration using a neural network," *IEEE transactions on acoustics, speech, and signal processing*, vol. 36, no. 7, pp. 1141–1151, 1988.

[188] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[189] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 826–834, 1983.

[190] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[191] B. Khagi, C. G. Lee, and G.-R. Kwon, "Alzheimer's disease classification from brain mri based on transfer learning from cnn," in *2018 11th Biomedical Engineering International Conference (BMEiCON)*. IEEE, 2018, pp. 1–4.

[192] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1263–1272.

[193] I. Wallach, M. Dzamba, and A. Heifets, "Atomnet: a deep convolutional neural network for bioactivity prediction in structure-based drug discovery," *arXiv preprint arXiv:1510.02855*, 2015.

[194] S. T. Hsu, C. Moon, P. Jones, and N. Samatova, "A hybrid cnn-rnn alignment model for phrase-aware sentence classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, 2017, pp. 443–449.

[195] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of cnn and rnn for natural language processing," *arXiv preprint arXiv:1702.01923*, 2017.

[196] A. Owens, P. Isola, J. McDermott, A. Torralba, E. H. Adelson, and W. T. Freeman, "Visually indicated sounds," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2405–2413.

[197] E. Ackerman, "How drive. ai is mastering autonomous driving with deep learning," *IEEE Spectrum Magazine*, vol. 1, 2017.

[198] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[199] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[200] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[201] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[202] C. Mouton, J. C. Myburgh, and M. H. Davel, "Stride and translation invariance in cnns," in *Southern African Conference for Artificial Intelligence Research*. Springer, 2021, pp. 267–281.

[203] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*. Springer, 2010, pp. 92–101.

[204] Y. LECUN, "The MNIST database of handwritten digits," *http://yann.lecun.com/exdb/mnist/*. [Online]. Available: https://ci.nii.ac.jp/naid/10027939599/en/

[205] B. D. Brown and H. C. Card, "Stochastic neural computation. i. computational elements," *IEEE Transactions on computers*, vol. 50, no. 9, pp. 891–905, 2001.

[206] Z. Li, A. Ren, J. Li, Q. Qiu, B. Yuan, J. Draper, and Y. Wang, "Structural design optimization for deep convolutional neural networks using stochastic computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 250–253.

[207] K. Kim, J. Lee, and K. Choi, "An energy-efficient random number generator for stochastic circuits," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 256–261.

[208] P. K. Muthappa, F. Neugebauer, I. Polian, and J. P. Hayes, "Hardware-based fast real-time image classification with stochastic computing," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 340–347.

[209] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou, and Y. Xu, "Throughput-optimized fpga accelerator for deep convolutional neural networks," *TRETS*, vol. 10, pp. 17:1–17:23, 2017.

[210] Z. Li, L. Wang, S. Guo, Y. Deng, Q. Dou, H. Zhou, and W. Lu, "Laius: An 8-bit fixed-point cnn hardware inference engine," in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, Dec 2017, pp. 143–150.

[211] S.-S. Park, K.-B. Park, and K. Chung, "Implementation of a CNN accelerator on an embedded soc platform using sdsoc," 02 2018, pp. 161–165.

[212] M. Dhouibi, A. K. Ben Salem, A. Saidi, and S. Ben Saoud, "Accelerating deep neural networks implementation: A survey," *IET Computers & Digital Techniques*, vol. 15, no. 2, pp. 79–96, 2021.

[213] A. Sayal, S. Nibhanupudi, S. Fathima, and J. Kulkarni, "A 12.08-TOPS/W all-digital time-domain CNN engine using bi-directional memory delay lines for energy efficient edge computing," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 60–75, 2020.

[214] H. Kung, B. McDanel, and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 821–834. [Online]. Available: https://doi.org/10.1145/3297858.3304028

[215] Y. Zhang, X. Zhang, J. Song, Y. Wang, R. Huang, and R. Wang, "Parallel convolutional neural network (cnn) accelerators based on stochastic computing," in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, 2019, pp. 19–24.

[216] S. Bodiwala and N. Nanavati, "Efficient implementation of stochastic computing based deep neural network on low cost hardware with saturation arithmetic," *Journal of Computer Science*, no. 11, pp. 1570–1584, Nov. 2020. [Online]. Available: https://thescipub.com/abstract/jcssp.2020.1570.1584

[217] A. Ren, Z. Li, Y. Wang, Q. Qiu, and B. Yuan, "Designing reconfigurable large-scale deep learning systems using stochastic computing," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2016, pp. 1–7.

[218] V. T. Lee, A. Alaghi, and L. Ceze, "Correlation manipulating circuits for stochastic computing," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1417–1422.

[219] P. Knag, W. Lu, and Z. Zhang, "A native stochastic computing architecture enabled by memristors," *IEEE Transactions on Nanotechnology*, vol. 13, no. 2, pp. 283–293, 2014.

[220] T.-H. Chen, P. Ting, and J. P. Hayes, "Achieving progressive precision in stochastic computing," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2017, pp. 1320–1324.