Preprints.org

Article

Not peer-reviewed version

Complex Exponential Based Bio-Inspired Neuron Model Implementation in FPGA Using Xilinx System Generator and Vivado Design Suite

MARUF AHMAD * , Lei Zhang * , Kelvin Tsun Wai Ng , Muhammad E.H Chowdhury

Posted Date: 6 November 2023

doi: 10.20944/preprints202311.0309.v1

Keywords: Spiking Neural Networks; Neural Encoding; Complex Exponential Neuron; FPGA Implementation

Preprints.org is a free multidiscipline platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

Complex Exponential Based Bio-Inspired Neuron Model Implementation in FPGA Using Xilinx System Generator and Vivado Design Suite

Maruf Ahmad [1], Lei Zhang [1,*] , Kelvin Tsun Wai Ng [1] and Muhammad E. H. Chowdhury [2]

1 Faculty of Engineering and Applied Science, University of Regina, Regina, Canada; mah370@uregina.ca (M.A.); kelvin.ng@uregina.ca (K.T.W.N.)
2 Department of Electrical Engineering, Qatar University, Doha 2713, Qatar; mchowdhury@qu.edu.qa
* Correspondence: Lei.Zhang@uregina.ca

Abstract: This research investigates the implementation of complex exponential-based neurons in FPGA, which can pave the way for implementing bio-inspired spiking neural networks to compensate for the existing computational constraints in conventional artificial neural networks. The increasing use of extensive neural networks and the complexity of models in handling big data lead to higher power consumption and delays. Hence, finding solutions to reduce computational complexity is crucial for addressing power consumption challenges. The complex exponential form effectively encodes oscillating features like frequency, amplitude, and phase shift, streamlining the demanding calculations typical of conventional artificial neurons through levering simple phase addition of complex exponential functions. The article implements such a two-neuron and a multi-neuron neural model using Xilinx system generator and Vivado design suite, employing 8-bit, 16-bit, and 32-bit fixed-point data format representations. The study evaluates the accuracy of the proposed neuron model across different FPGA implementations while also providing a detailed analysis of operating frequency, power consumption, and resource usage for the hardware implementations. BRAM-based Vivado designs outperformed Simulink regarding speed, power, and resource efficiency. Specifically, the Vivado BRAM-based approach supported up to 128 neurons, showcasing optimal LUT and FF resource utilization. Such outcomes accommodate choosing the optimal design procedure for implementing spiking neural networks on FPGAs.

Keywords: spiking neural networks; neural encoding; complex exponential neuron; FPGA implementation

---

1. Introduction

Spiking Neural Networks (SNN) represent a promising approach for information encoding, where the firing rate or number of spikes within a certain time period reflects the oscillation properties of biological neurons. In this context, recent research has proposed a mathematical model based on complex exponential neurons [1] that provides a means of encoding and decoding information in artificial neural network systems by leveraging their oscillation features such as frequency, phase and amplitude. The model offers a significant advantage over existing approaches, as it enables the simplification of complex calculations involving convolution and multiplication to phase addition, which enhances the computational efficiency and performance of Artificial Neural Networks.

The findings of this research are particularly relevant in the context of embedded system design, where there is a need to achieve high precision of neural network output while dealing with limited hardware resources [2,3] such as memory and computational capacity. Thus, this research highlights a promising avenue for improving the performance and efficiency of Spiking Neural Networks, which is a critical factor in enabling their adoption in real-world applications.

© 2023 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

1.1. Related Work

In this literature review section, we present a comprehensive overview of recent advancements in the field of neural network models based on complex exponential functions. One notable research study introduced the construction of a Spiking Neural Network (SNN) model and its hardware accelerator using FPGA utilizing spiking exponential functions [4]. Another study delved into the implementation of a Multi-Valued Neuron (MVN) model [5], also founded on complex exponential functions, highlighting how MVNs can significantly enhance the functionality of individual neurons. Moreover, a recent investigation focused on MVNs introduced a novel derivative-free convolutional neural network [6] learning algorithm, demonstrating its efficacy in accelerating the learning process while improving generalization capabilities. Remarkably, it is worth noting that, to date, there has been a dearth of hardware accelerators designed to facilitate neural network inference based on these innovative models. The inherent simplicity of derivative-free learning algorithms presents an exciting opportunity to explore the development of dedicated hardware accelerators for neural network learning.

Considerable effort has been devoted to the hardware implementation of spiking neural networks, with a particular focus on FPGA-based and Application-Specific Integrated Circuit (ASIC) systems [7,8]. The paper presented in [9] describes the implementation of a spiking neural network model on a Xilinx FPGA evaluation board, utilizing a hybrid updating algorithm that combines conventional time-stepped updating and event-driven updating techniques, while using 16-bit signed fixed-point number representation. Their model consisted of 16,384 neurons and 16.8 million synapses, and achieved an accuracy of 97.06% on the MNIST dataset classification task with a power consumption of only 0.477W.

Another study [10] presents S2N2, a streaming accelerator for spiking neural networks that efficiently supports axonal and synaptic delays, utilizes binary tensors for addressing events, and achieves a minimum tick-resolution of 30 ns with over three orders of magnitude reduction in input buffer memory utilization.

The study in [11] proposed a holistic optimization framework for encoder, model, and architecture design of FPGA-based neuromorphic hardware, which includes an efficient neural coding scheme, training algorithm, and flexible SNN model represented as a network of IIR filters, achieving state-of-the-art accuracy and outperforming various platforms in terms of latency and throughput.

In [12] a simplifed Leaky integrate-and-fire neuron model is used to develop an efficient SNN on Xilinx Virtex 6 FPGA. In a recent study [13], deep convolutional spiking neural networks (DCSNNs) were successfully implemented on low-power FPGA devices, where two backpropagation techniques were compared for object classification tasks. Additionally, another recent work [14] focuses on developing a customizable hardware accelerator for neural network inference models, specifically building a convolutional neural network on an FPGA. This study showcases substantial reductions in power consumption when compared to running the same convolutional neural network model on a laptop processor.

Also in, [14] outlines an optimization scheme aimed at improving the performance of the SNN by adjusting biological parameters, and then proceeds to implement the SNN on FPGA using the Euler and third-order Runge-Kutta (RK3) methods.

Lastly, implementing complex exponential functions is a crucial part of this project, and it has been developed in many research in various ways on FPGA, including polynomial approximation, interpolation, look-up table based methods [15], CORDIC IP core [16], two-dimensional interpolation [17], and floating-point implementation [18]. However, these methods often consume a significant amount of FPGA logic cells, such as Look-up Table (LUT) and D flip-flops. Fortunately, most FPGAs have Block Random Access Memory (BRAM) [19], which can be utilized to implement complex exponential functions and save considerable FPGA resources. This is particularly important for larger projects like neural network development, where efficient resource utilization is crucial.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

In this paper, a complex exponential function-based neuron model [1] is designed in MATLAB Simulink using Xilinx System Generator (SysGen) [20,21] and Xilinx Vivado design suite for FPGA implementation on a ZynQ xc7z020clg484-1 FPGA chip. 8-bit, 16-bit and 32-bit fixed-point number representations are used for the model design.

Xilinx System Generator is an add-on feature for MATLAB Simulink that facilitates using graphical block programming for architecture-level FPGA designs. Xilinx Vivado is an Integrated Development Environment (IDE) [22] for designing digital circuits for Xilinx FPGA and System on Chip (SoC) devices, supporting various design entry methods, including graphical block diagram entry and HDL code entry.

1.2. Outline

The rest of the paper is structured into several key sections. Section 2 offers a mathematical overview of the complex-exponential neuron model, elucidating its foundational principles. Section 3 explores various implementation methods, including BRAM and CORDIC-based designs, realized through Vivado and Simulink, with subsequent analysis. Section 4 presents results and facilitates discussion based on simulations of these implementations. Lastly, Section 5 provides a summarization of the research's core findings and outlines potential future directions, ensuring a logical and coherent flow throughout the paper.

2. Background

2.1. Mathematical model

In the realm of complex numbers, a representation is often employed wherein a complex number $z = x + iy$ residing in the complex plane finds its expression as $z = r \cdot e^{i\phi}$ in the phase domain. This representation entails two essential components: the magnitude (r), which serves as the vector's length and is calculated as $r = e^x$, and the phase angle (ϕ), signifying the angle between the vector and the x-axis ($\phi = y$). Furthermore, Euler's formula remarkably connects complex exponentials with trigonometric functions, manifesting as $e^{i\pi} + 1 = 0$, with a distinctive instance at $\phi = \pi$, intricately bridging imaginary numbers with the transcendental constants π and e. At the core of this representation lie two fundamental equations:

$$|e^{i\phi}| = \sqrt{\cos^2\phi + \sin^2\phi} = 1$$

unveiling the unit magnitude of $e^{i\phi}$, and

$$e^{i\phi} = \cos\phi + i\sin\phi;\ e^{\bar{i}\phi} = \cos\phi - i\sin\phi$$

illustrating the complex exponential and its complex conjugate. These equations encapsulate the essence of complex number representation and its profound mathematical underpinnings.

In the phase plane, an oscillating neuron E(t) can be written by the complex exponential form as

$$E(t) = e^{i\omega t+\theta} = e^{\theta}e^{i\omega t} = e^{\theta}\left(\cos\omega t + i\sin\omega t\right) \tag{1}$$

Where ω is the angular frequency, the real component θ is used to make the oscillation amplitude.

The synaptic weight can also be represented by the complex exponential form as $W = e^{i\phi}$ where ϕ represents the phase delay of the neural connection.

Therefore a weighted input can be represented as

$$E(t)W = e^{i\omega t+\theta}e^{i\phi} = e^{\theta}e^{i(\omega t+\phi)} \tag{2}$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

In neural networks, weights act as scaling factors that adjust data as it passes between connected neurons. Input to a neuron is the sum of outputs from previous layer neurons, each weighted by its respective synaptic connection. This process drives information transformation within the network.Pre-synaptic neurons are multiplied by their corresponding weight and then summed up to the post-synaptic neuron. This summation (S) in the post-synaptic neuron can also be represented by a complex exponential form as in equation (5) [1].

$$S = E_1 W_1 + E_2 W_2 \tag{3}$$

$$= e^{i\omega_1 t + \theta_1} e^{i\phi_1} + e^{i\omega_2 t + \theta_2} e^{i\phi_2} \tag{4}$$

$$= e^{\theta_1} e^{i(\omega_1 t + \phi_1)} + e^{\theta_2} e^{i(\omega_2 t + \phi_2)} \tag{5}$$

Three parameters ω, θ and ϕ are used to calculate the weighted sum.This representation aligns with biological plausibility, where the synaptic weight signifies a phase delay introduced to the input neuron's oscillation, considering minimal signal attenuation over short distances. In biological neural cells, a specialized structure known as myelin plays a vital role in facilitating swift impulse transmission along axons, consequently enhancing the speed of action potential propagation [23]. In this paper, the equation of the complex exponential function is represented by $e^{i\phi} = \cos\phi + i\sin\phi$

2.2. MATLAB Simulation Report

In the MATLAB Simulink, a weighted sum of two neurons model is designed as an example for analyzing the Complex Exponential Neural Network. Figure 1 provides a comprehensive depiction of two weighted inputs and their combined weighted sum, plotted individually. The temporal patterns reveal that weighted inputs introduce phase delays to oscillating neurons while preserving their oscillation amplitude. However, the weighted sum continues to oscillate periodically, but its amplitude is no longer constant; instead, it exhibits periodic variations.

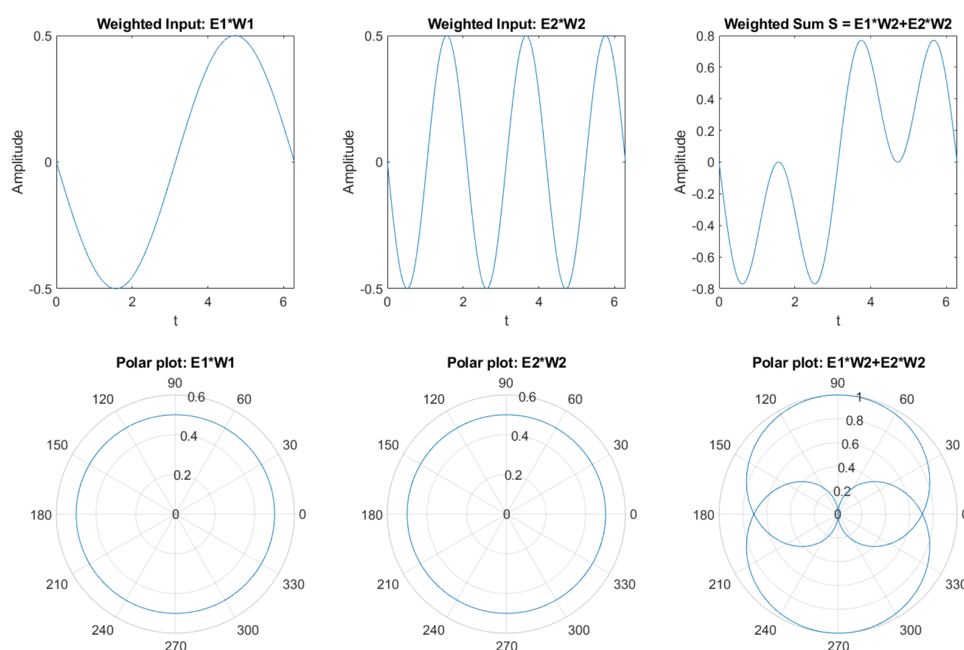

Figure 1. Weighted sum of two complex exponential neurons.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

Figure 2 shows the output of the weighted sum of two neurons $E1W1 + E2W2$ with different parameter values mentioned in the examples below.

- Example 1 in Figure 2 (temporal plot I and Polar plot I) shows the output of the weighted sum of two neurons where $\omega_1 = 1, \omega_2 = 2, \theta_1 = \log(0.5), \theta_2 = \log(1.5), \phi_1 = \frac{\pi}{2}, \phi_2 = \frac{\pi}{3}$ are used.
- example 2 (temporal plot II and Polar plot II) all parameters of the weighted input of two neurons remain the same except $\theta_1 = \log(1)$ is taken. This causes a change in amplitude.
- example 3, θ_1 is changed to $\frac{\pi}{4}$. This changes the phase and orientation of the weighted sum.
- For example, 4 ω_2 is set to 3; this dramatically changes the pattern of the weighted sum, which is shown in the polar plot iv.

These four examples are used to demonstrate the effect of different parameters and how changes of one parameters can distinctively change the oscillation patterns of the output. Therefore any of the parameters can be used to generate unique encoding patterns [24].

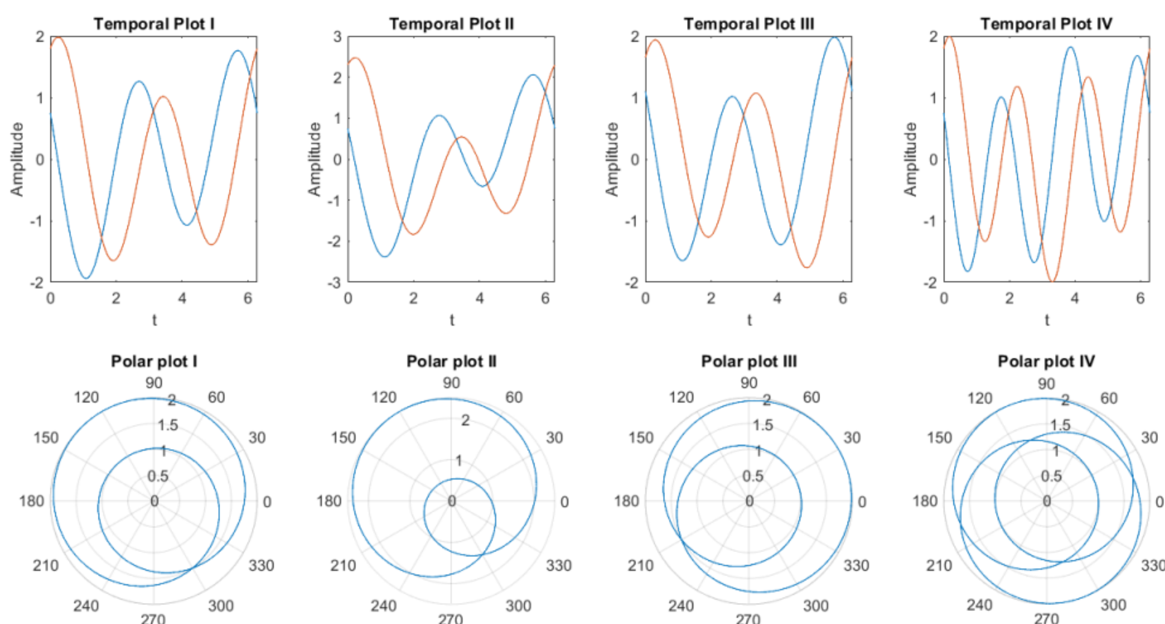

Figure 2. Examples of the Weighted sum of two Complex Exponential neurons with different parameter values.

3. Methodology

3.1. Model design in Simulink and Vivado

In this study, we present four different approaches to implement the weighted sum of two neurons on an FPGA using the Xilinx System Generator (SysGen) of Matlab Simulink and VHDL coding in Vivado Design Suite. The first approach involves utilizing Block-Random-Access-Memory(BRAM) HDL blocks to implement the complex exponential function of the weighted sum of two neurons in Matlab Simulink using SysGen. The second approach employs BRAM IP cores and VHDL coding to develop the model in the Vivado design suite. In the third approach, CORDIC HDL blocks from the Xilinx toolbox are used in Matlab Simulink to implement the exponential function. Finally, the fourth approach utilizes VHDL coding and CORDIC IP core to implement the weighted sum of two neurons in the Vivado design suite.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

For each approach, three separate designs were implemented using 8, 16, and 32-bit fixed-point data format configurations.

The weighted sum of two neurons requires several input parameters, including ω_1, ω_2, θ_1, θ_2, ϕ_1, ϕ_2, and t. However, for the sake of simplicity, we kept all input parameters fixed except for t across all four approaches. Specifically, we used the following fixed values: $\omega_1 = 1$, $\omega_2 = 2$, $\theta_1 = \log(2)$, $\theta_2 = \log(2)$, $\phi_1 = \frac{\pi}{2}$, and $\phi_2 = \frac{\pi}{2}$. We varied the value of the input parameter t within the range of -4 to 4 to cover a complete oscillation cycle.

The outputs of all the approaches were compared with the MATLAB simulation output of equation (5). The accuracy, latency, and resource utilization of each approach were evaluated and compared to determine the most efficient approach for implementing the weighted sum of two neurons on an FPGA.

Vitis Core Development kit version 2021.1, which includes MATLAB R2021a, Xilinx System Generator, and Vivado 2021.1 (64-bit), is used to design, code, and simulate the projects.

3.1.1. BRAM-based design (using SysGen in MATLAB Simulink)

In the FPGA implementation of complex exponential neurons, a crucial task is to implement the function $e^{i\Delta}$. This is achieved by separately implementing the real and imaginary components of the function, which are $\cos\Delta$ and $i\sin\Delta$, respectively. Therefore, the real part of the complex exponential output is given by $\cos\Delta$, while the imaginary part is given by $\sin\Delta$. Here, the Δ is considered as input of the exponential function. In this project, we utilize the periodicity of the cosine and sine functions, which have a period of 2π. Therefore, for any given angle θ, the values of $\cos(\theta)$ and $\cos(\theta + 2\pi)$ are the same. The output of the cosine function is a value between -1 and 1, inclusive. To implement the exponential function in BRAM, we consider the input range to be between -3.14 and 3.14, and the output range to be between -1 and 1, since the maximum and minimum value of real ($\cos(\theta)$) and imaginary ($\sin(\theta)$) output is in between 1 to -1. We set the input resolution to a step size of 0.01, resulting in 629 equally spaced values in this range. This approach simplifies the design and reduces the number of memory elements required to implement the exponential function.

In BRAM-based design using the SysGen approach, to implement the complex exponential function, a lookup table with input versus output data of $\cos and \sin$ is mapped to the block-random-access-memory (BRAM) HDL block in MATLAB Simulink. The block is configured as a fixed point data format representation. Other graphical HDL blocks such as adder, multiplexer, buffer, etc., are used to implement the weighted sum of two complex exponential neurons shown in Figure 3.

Once the design is complete, an IP core is generated using the Xilinx System Generator tool. This IP core is then instantiated in Vivado IDE to perform behavioral simulation and generate a hardware implementation report.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

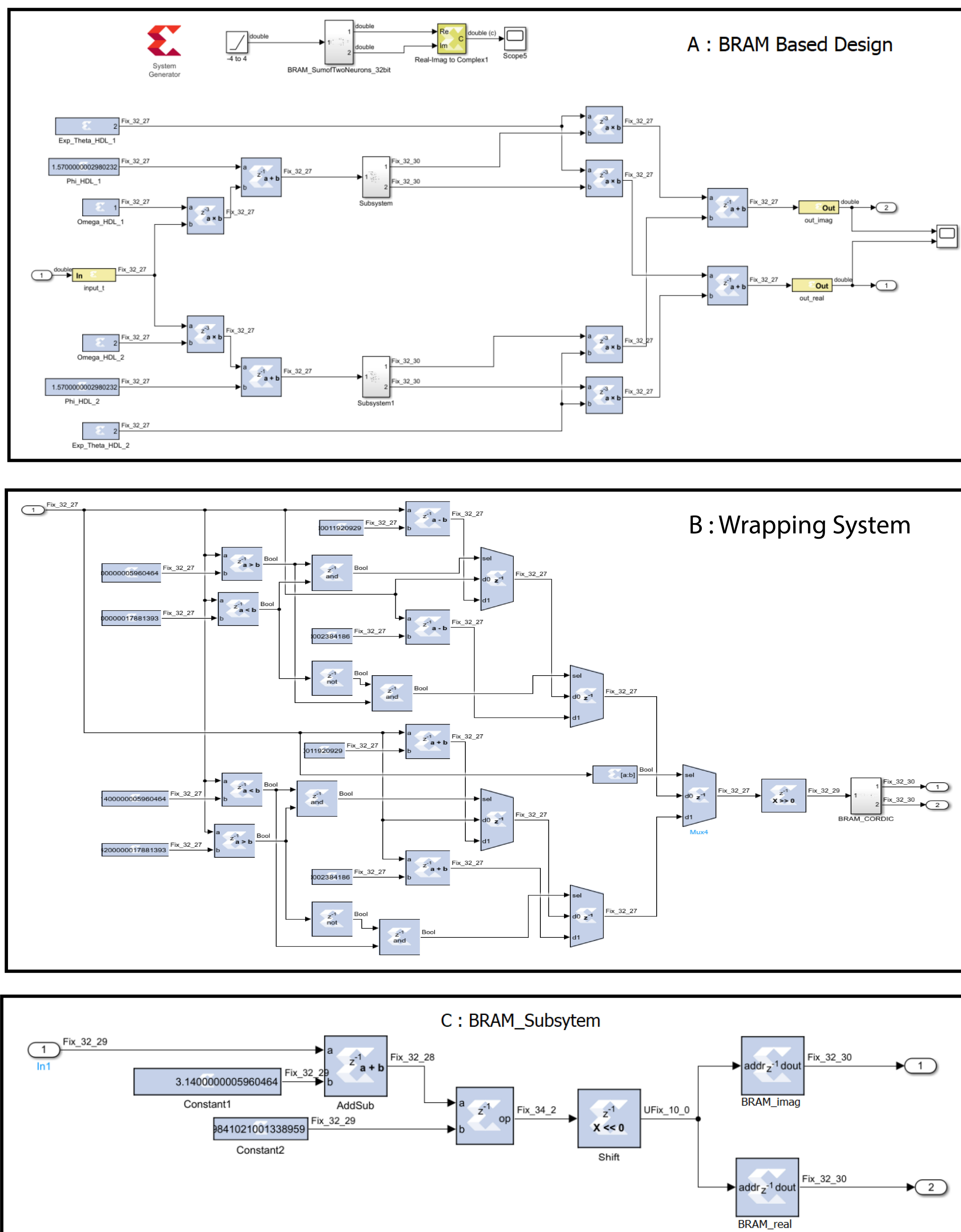

Figure 3. 32-bit implementation of BRAM-based design in MATLAB Simulink.

3.1.2. BRAM-based design in Vivado

In this approach, we utilized the Block Memory Generator tool (version 8.4) of Xilinx LogiCORE to map the Block-Random-Access-Memory (BRAM) to implement the complex exponential function in Vivado IDE. To ensure the same level of precision and accuracy, we used a similar lookup table configuration of the BRAM-based design in MATLAB Simulink for the BRAM mapping.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

For implementing the weighted sum of two neurons model, we opted for a VHDL coding approach and fixed point data format to develop the necessary components, including the adder and multiplexer. These components were then mapped to the BRAM to finally complete the implementation.

3.1.3. CORDIC based design (using SysGen in MATLAB Simulink)

In the CORDIC-based design approach, we take advantage of the built-in Sin_and_Cos function provided by the Xilinx CORDIC 6.0 HDL block in MATLAB Simulink to efficiently implement the complex exponential function. Since the input of the CORDIC block is limited to a range of $-\pi$ to π, a wrapping subsystem is implemented before the CORDIC IP Core to ensure that the input signal falls within this range. This helps to guarantee the accuracy and precision of the final output.

Once the complex exponential function is obtained, it is combined with other components, such as an adder, multiplexer, and buffer, to implement the weighted sum of two neurons. The fixed-point representation is used throughout the design to maintain precision and reduce the computational overhead.

After finishing the design, an IP core is created using the Xilinx System Generator tool. This IP core is then embedded in Vivado IDE, where behavioral simulation is performed and a hardware implementation report is generated.

3.1.4. CORDIC based design in Vivado

CORDIC-based design in Vivado: In the CORDIC-based design approach in Vivado, we utilize the CORDIC (6.0) IP core of Xilinx to implement the complex exponential function using its built-in Sin and Cos functions and parallel architecture.

Similar to the approach in Simulink, we implement a wrapping function, adder, multiplexer, and buffer in VHDL coding to implement the weighted sum of two neurons. Figure 4 shows the schemetic diagram of CORDIC-based design in Vivado IDE as an example.

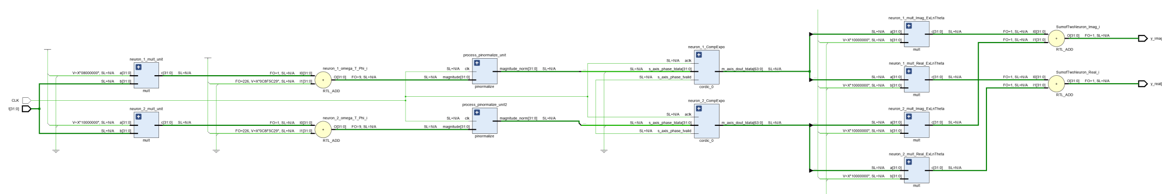

Figure 4. Schematic diagram of 32-bit CORDIC based design in Vivado.

3.1.5. The fixed-point implementation

In the given project, the input range lies between -4 and 4, while the output range is between -1 and 1. To represent the integer part of the input value in binary, only three bits are required as 4 in binary can be represented as 100. One bit is used to represent the sign of the input value, and the remaining bits are reserved for the fractional part.

However, for internal operations such as addition and multiplication, the input values go beyond decimal 16. Therefore, five bits are reserved for the integer part to ensure accurate calculations.

Similarly, for the output values, one bit is used to represent the sign, and the remaining bits are used for the fractional part. Since the maximum and minimum output values are 1 and -1, one bit is reserved for decimal 1, and the remaining bits are used for the fractional part.

By utilizing this fixed-point representation technique, we can allocate more bits to the fractional part, thereby enhancing the precision and accuracy of the final output.

Based on the above conditions, for the Simulink and Vivado-based design, in an 8-bit fixed-point implementation, the input is configured as Fix8_5 data format, with one sign bit, two integer bits and five fractional bits. The output comes with Fix8_6 data format, with one sign bit, one integer bit and six fractional bits.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

In the 16-bit fixed-point implementation, the input is configured as Fix16_13 data format, with one sign bit, two integer bits and 13 fractional bits. The output comes with Fix16_14 data format, with one sign bit, one integer bit and 14 fractional bits.

In the 32-bit fixed-point implementation, the input is configured as Fix32_29 data format, with one sign bit, two integer bits and 29 fractional bits. The output comes with Fix32_30 data format, with one sign bit, one integer bit and 30 fractional bits.

4. Result and Discussion

4.1. Simulation Report

4.1.1. 8-bit implementation result

The graphs presented in Figure 5 depict the output of the FPGA simulation for the weighted sum of complex exponential neurons using four different approaches in 8-bit fixed-point implementation, which are compared to the output of MATLAB simulation.

The graphs in Figure 6 show the absolute difference in the real values of the weighted sum of complex exponential neurons using BRAM-based and CORDIC-based designs in both Simulink and Vivado for 8-bit fixed-point implementation.

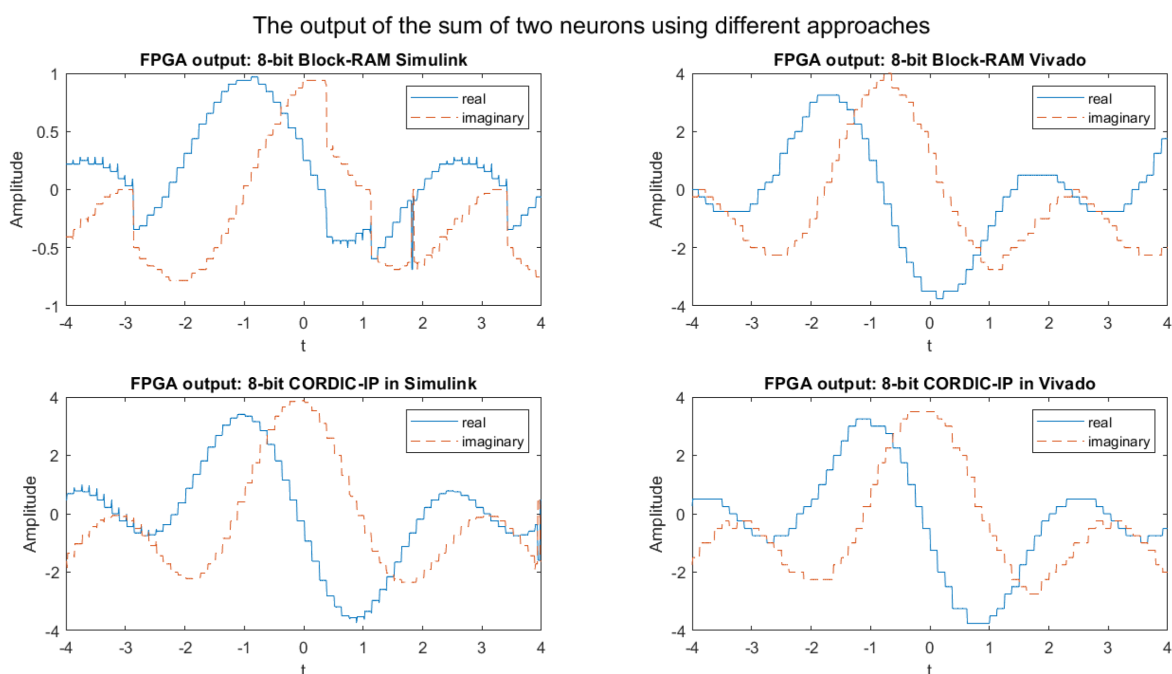

Figure 5. FPGA simulation output graphs of the weighted sum of complex exponential neurons in four different approaches for 8-bit fixed point implementation.

It was observed that while the shape of the output signals produced by the four different approaches were similar to the output produced by MATLAB, the precision of the outputs was very poor when compared to the expected output generated by MATLAB.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

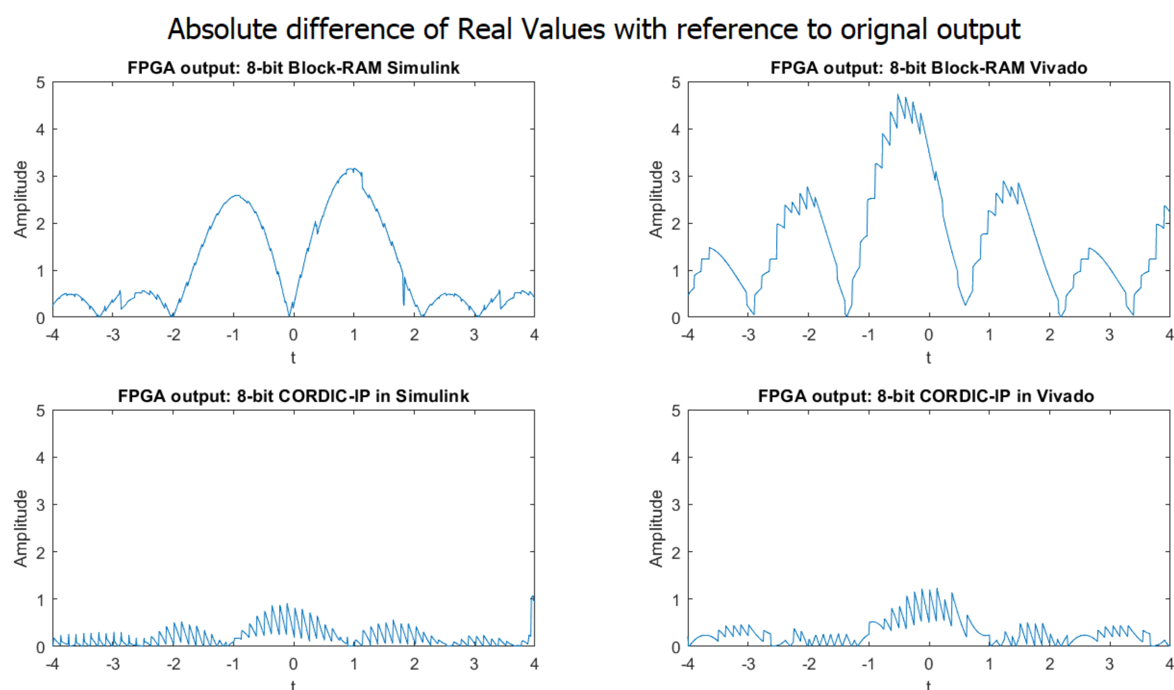

Figure 6. Comparison of absolute difference in real value between MATLAB simulation output and four different approaches in 8-bit implementation.

As the MATLAB simulation output for the weighted sum of two neurons in 16-bit implementation is comparable to the output displayed in Figure 7, we have excluded the graph for MATLAB simulation output of the weighted sum of two neurons.

4.1.2. 16-bit implementation result

The FPGA simulation outputs for the weighted sum of complex exponential neurons using four different approaches in 16-bit fixed point implementation are presented in Figure 7. Figure 8 displays the absolute difference of real values of the weighted sum of two complex exponential neurons for the same four approaches in 16-bit fixed point implementation. The comparison of Figures 7 and 8 indicates that the outputs of the weighted sum of two neurons in 16-bit fixed point implementation for all four approaches are significantly more accurate and closely matched to the output of MATLAB simulation.

From the Figures 5 and 7, we can observe some glitches in the output of the BRAM Simulink-based designs, this might be because of the overflow in the calculation in the wrapping system design. We will fix it in our future research.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

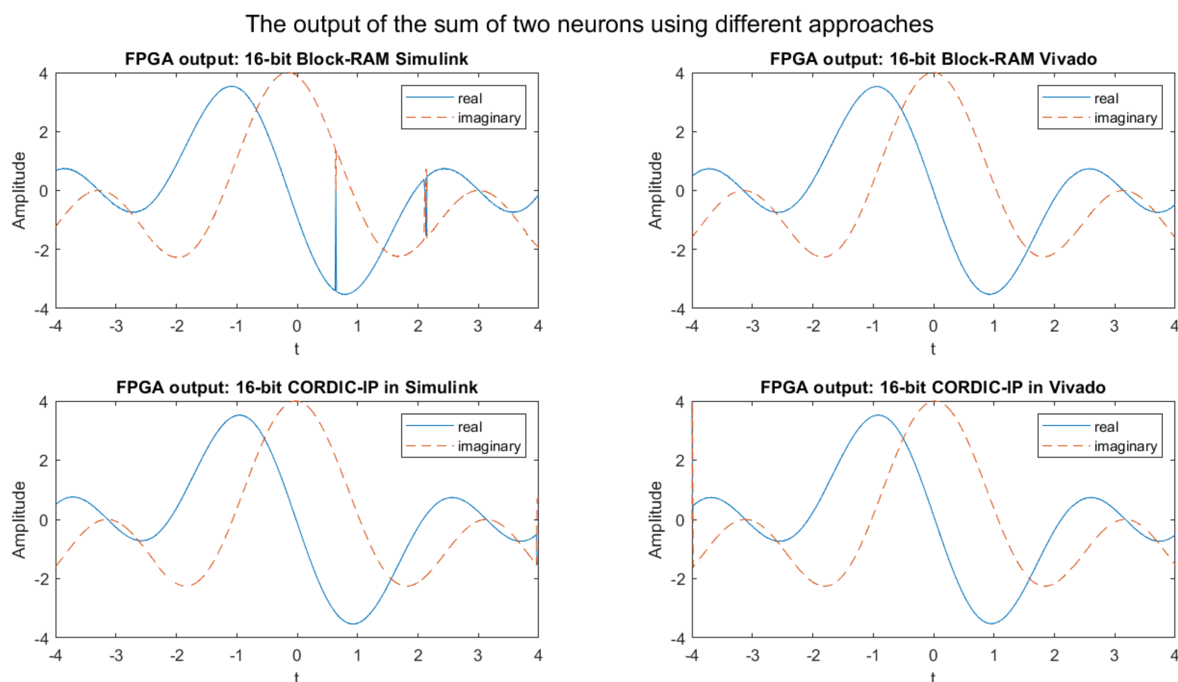

Figure 7. FPGA simulation output graphs of the weighted sum of complex exponential neurons in four different approaches for 16-bit fixed point implementation.

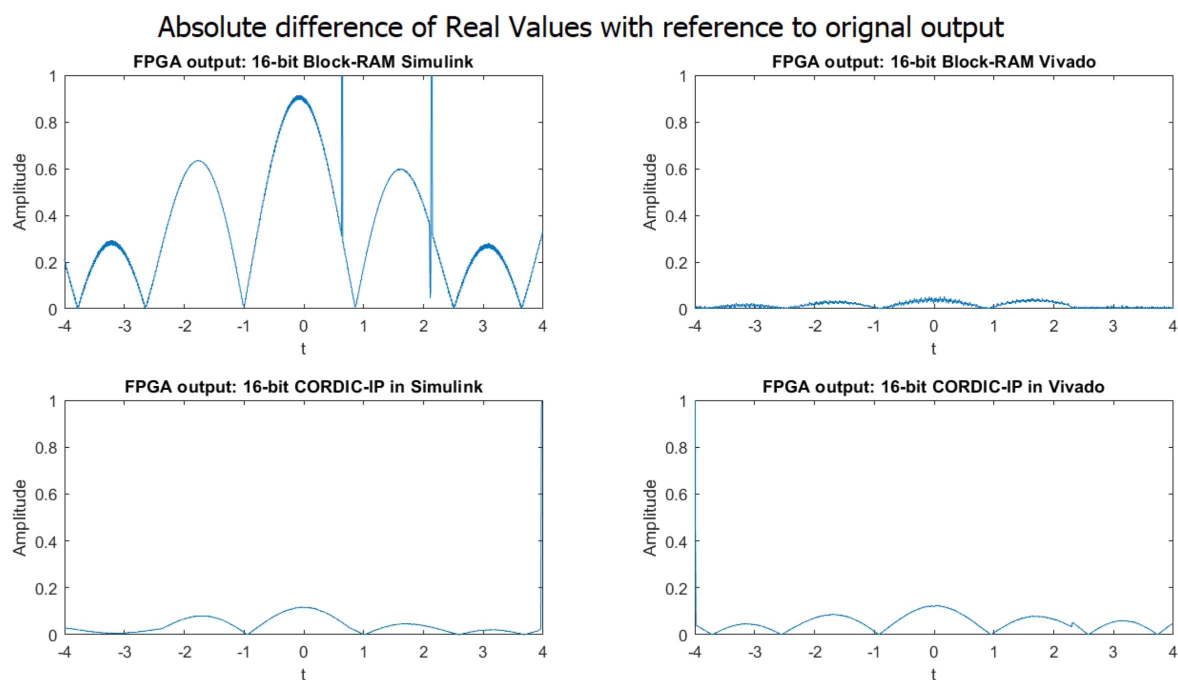

Figure 8. Comparison of absolute difference in real value output between MATLAB simulation and four different approaches in 16-bit implementation.

4.1.3. 32-bit implementation result

By examining the graphs in Figure 9 in the 32-bit implementation presented in the paper, it is evident that the absolute difference between the output generated by the MATLAB simulation and the output produced by the four different design approaches is significantly lower compared to the 16-bit

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

implementation. This implies that the accuracy and precision of the output are greatly improved by using 32-bit implementation.

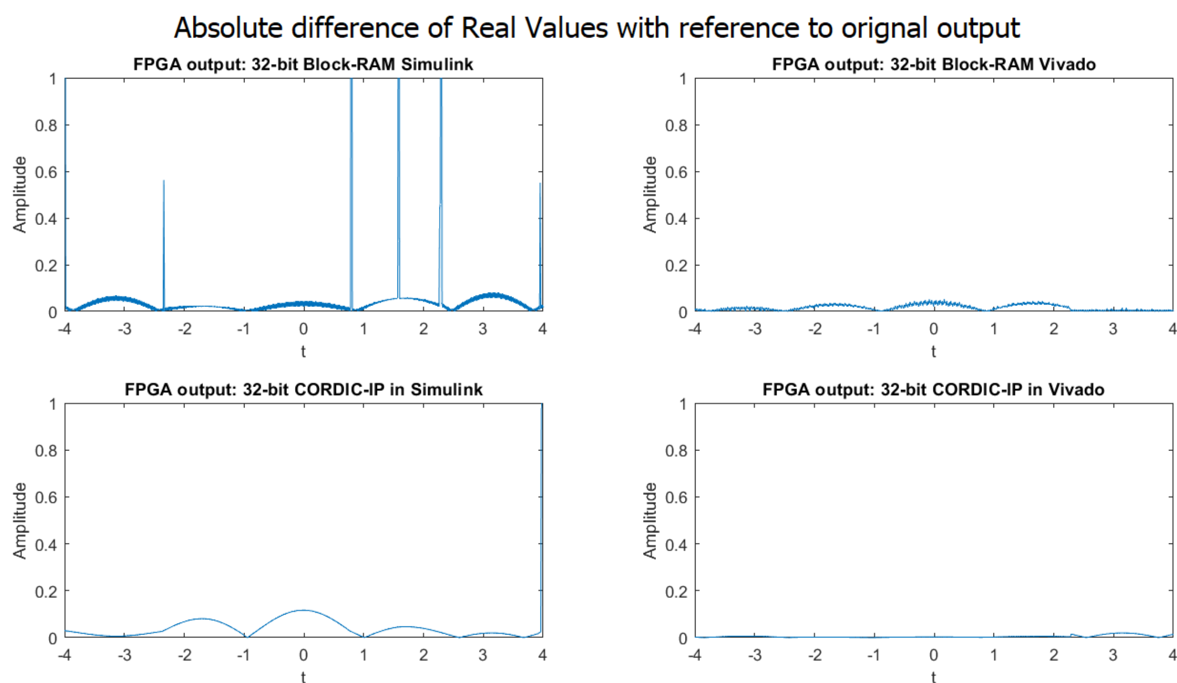

Figure 9. Comparison of absolute difference in real value between MATLAB simulation output and four different approaches in 32-bit implementation.

4.1.4. MAE of the four different implementation approaches

The Mean Average Error graphs in Figure 10 clearly indicate that as the bit size increases, the Mean Average Error becomes minimized for all four implementation approaches. This suggests that the outputs generated by these implementations are becoming more precise and closer to the expected output. Though it is shown in the report that MAE has been calculated using the output of the real value, a similar result is found for the output of the imaginary value.

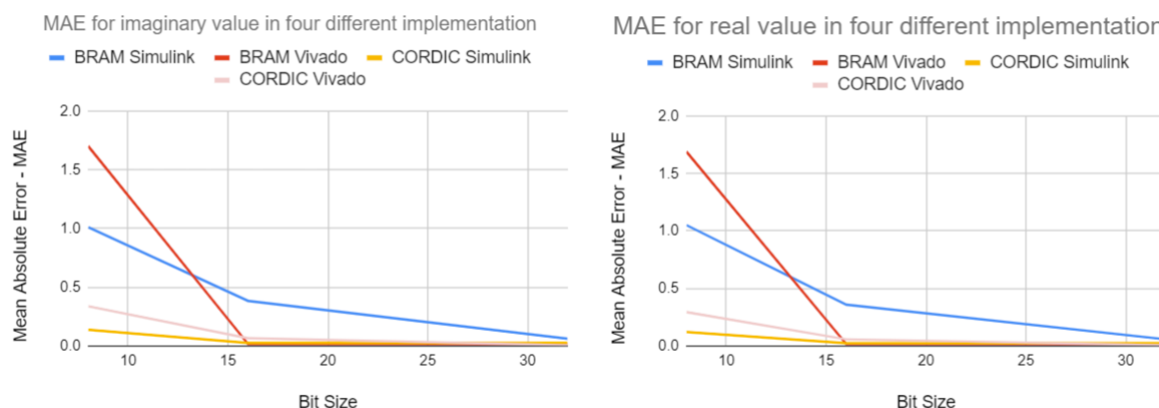

Figure 10. Mean Average Error of the output of four different implementation approaches for real and imaginary values.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

4.1.5. Discussion Summary

The discussed results focus on the performance of different fixed-point implementation approaches of complex exponential neurons in FPGA simulations. The paper compares the output of the FPGA simulations with that of MATLAB simulations. The results show that the 8-bit fixed-point implementation has poor precision compared to the 16-bit and 32-bit implementations. The 16-bit implementation outputs are more accurate and closely matched to the MATLAB simulation outputs. The 32-bit implementation further improves the accuracy and precision of the output, as evident by the significantly lower absolute difference between the output generated by the MATLAB simulation and the output produced by the four different design approaches. The Mean Average Error graphs demonstrate that the accuracy and precision of the output increase as the bit size increases, minimizing the Mean Average Error for all four implementation approaches. Overall, the results suggest that the use of higher bit sizes in fixed-point implementation approaches can improve the accuracy and precision of the output, making them more reliable and useful in practical applications.

4.2. Hardware Implementation Report

The field of FPGA hardware design offers multiple approaches for implementing a design, each with its own set of advantages and disadvantages. This article will focus on comparing four design methods: BRAM-based design in both Simulink and Vivado, and the CORDIC IP-based design in both Simulink and Vivado for 8, 16, and 32-bit implementation. The comparison between these methods will be based on their speed, power requirements, and resource usage.

The BRAM-based implementation uses block RAMs as the primary storage element for the design. The advantage of this approach is that it enables the implementation of designs with high memory requirements, and it can be optimized for power and area. However, this approach may not be optimal for designs with high operating frequencies since the latency of accessing the block RAM can be high, leading to a reduced operating frequency.

The CORDIC IP-core-based implementation involves using pre-designed and pre-verified blocks of digital circuits (IP cores) that can be integrated into a larger design. This approach can reduce development time and simplify the design process. Additionally, IP cores are typically optimized for performance, which can result in high operating frequencies.

This project has utilized the ZYNQ-7 ZC702 Evaluation Board library in Vivado for implementing all the different methods. The board used in this project has a part name of xc7z020clg484-1, and it offers 484 I/O pins, LUTs of 53200, FFs of 106400, and DSPs of 220, among other resources.

4.2.1. WNS Report:

Figure 11 shows the worst negative slack (WNS) in nanoseconds for different clock frequencies for four different types of implementations: BRAM-based in Simulink, BRAM-based in Vivado, CORDIC-based in Simulink, and CORIDC-based in Vivado, with different data widths of 8-bit, 16-bit, and 32-bit.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

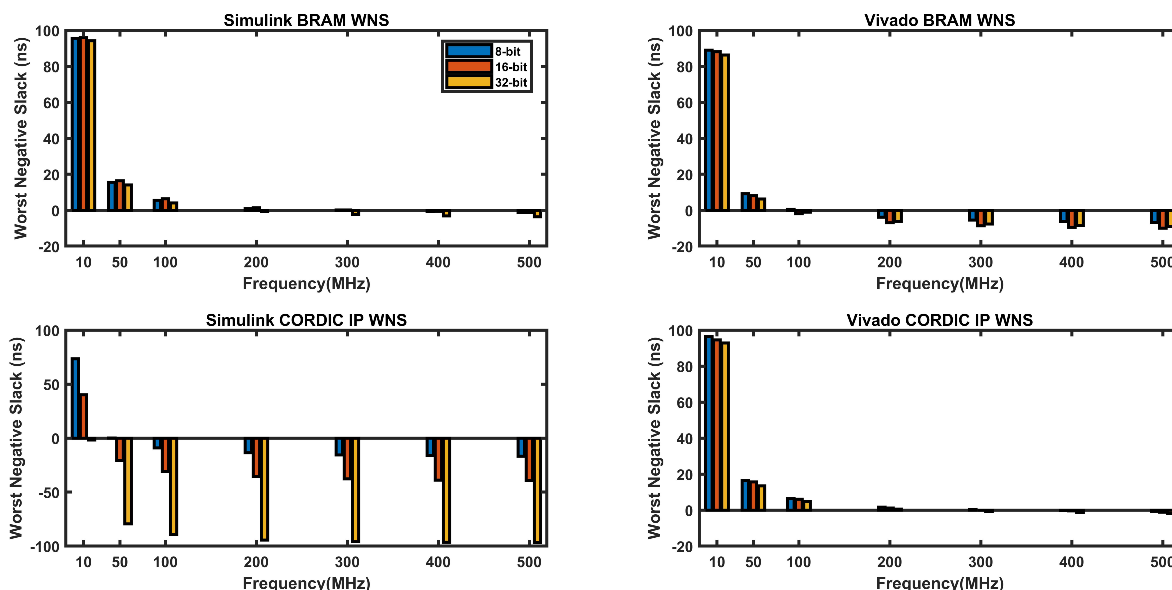

Figure 11. WNS vs. Operating Frequency graphs in four different approaches.

WNS represents the amount of time by which the path with the worst timing violates the clock period, which means that the design fails to meet timing at that frequency. A negative WNS means that the design is failing timing, while a positive WNS means that the design is meeting timing. Ideally, a positive value of nearly zero or zero is considered as a good design for a particular clock frequency.

From Figure 11, it appears that Vivado BRAM and Vivado CORDIC designs have better worst negative slack (WNS) values than Simulink BRAM and Simulink CORDIC designs for all operating frequencies. This means that Vivado designs are meeting timing more easily compared to Simulink designs in the selected frequency range (10Mzh to 500MHz).

Furthermore, it can be observed that as the operating frequency increases, the WNS values become more negative for all designs. This is expected since higher frequencies lead to shorter clock cycles, which gives less time for the signals to propagate through the circuit. As a result, it becomes harder for the designs to meet timing at higher frequencies.

In terms of the design types, it appears that the CORDIC designs have worse WNS values compared to the BRAM designs for all operating frequencies.

It can also be observed that bit size has less significance for all designs except the CORDIC-based design in Simulink, where the increase in bit size leads to a decrease in the WNS value.

Overall, the WNS values provide an indication of the timing performance of the designs, but other factors such as resource usage and power consumption also need to be considered when comparing different design implementations.

4.2.2. Max Operating Frequency

Maximum Operating Frequency is calculated by the formula below:

$$f_{max} = \frac{1}{T_S - WNS} \tag{6}$$

Where f_{max} is the maximum clock frequency, WNS is Worst Negative Slack, T_S is the minimum time required to complete a sequence. In maximum operating frequency, WNS will be nearly 0 and $f_{max} = \frac{1}{T_S}$

If we look at Figure 12, it can be observed that the maximum frequency achieved varies for different bit-sizes and different implementations. For the BRAM-based implementations, the Simulink implementation achieves a higher maximum frequency compared to the Vivado implementation for all bit-sizes. On the other hand, for the CORDIC-based implementations, both the Simulink

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

and Vivado implementations achieve the same maximum frequency of 300 MHz for 8-bit and 16-bit implementations, while for 32-bit implementation, the Vivado implementation achieves a higher maximum frequency of 200 MHz compared to the Simulink implementation.

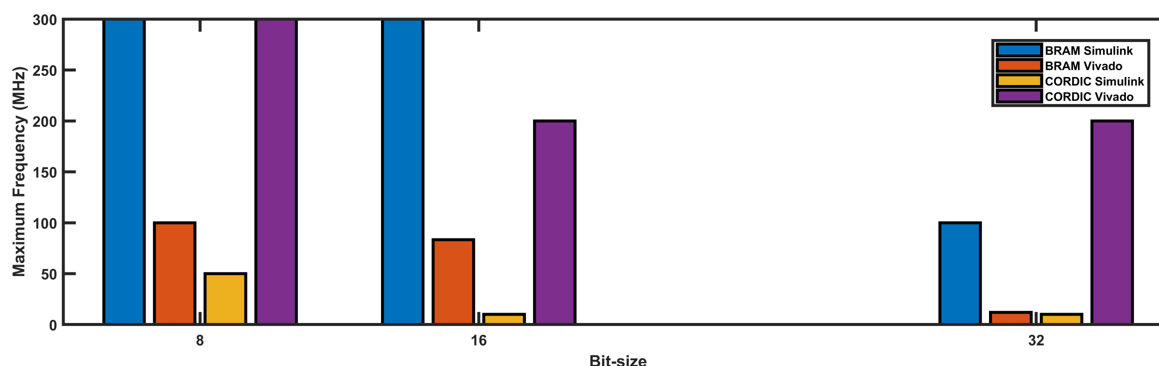

Figure 12. Maximum Operating frequency vs bit-size graph for different implementation approaches.

Overall, the BRAM-based implementations achieve higher maximum frequencies compared to the CORDIC-based implementations. Moreover, increasing the bit-size results in a decrease in the maximum frequency achieved for all implementations, except for the Vivado CORDIC implementation for 32-bit, where it achieves a higher maximum frequency compared to the 16-bit implementation.

4.2.3. Resource usages

Table 1 shows the usage of Lookup tables (LUT) and Flip-flops (FF) for different implementations using different design tools. For the BRAM-based designs, the LUT usage is comparatively low, ranging from 41 to 337 for Vivado and from 286 to 2904 for Simulink, across all bit sizes. The FF usage for the BRAM-based designs is also low, ranging from 0 to 116 for Vivado and from 514 to 5336 for Simulink. On the other hand, the CORDIC-based designs have a higher LUT and FF usage, especially for the higher bit sizes. The LUT usage for the CORDIC-based designs ranges from 612 to 7660 for Simulink and from 548 to 7473 for Vivado. The FF usage for the CORDIC-based designs ranges from 196 to 874 for Simulink and from 516 to 7217 for Vivado.

Overall, it can be observed that the BRAM-based designs require relatively fewer FPGA resources in terms of LUT and FF usage compared to CORDIC-based designs. However, as the bit size increases, the resources required for CORDIC-based designs increase significantly. It is also interesting to note that the usage of FPGA resources is different for different design tools, with Simulink generally requiring more resources than Vivado. These findings can be useful for selecting the appropriate implementation method based on the available FPGA resources and the desired performance requirements.

Table 1. LUT and FF usages for different implementations.

Resource Usages	Bit Size	BRAM Simulink	BRAM Vivado	CORDIC Simulink	CORDIC Vivado
LUT usages	8	286	41	612	548
LUT usages	16	965	202	2160	2094
LUT usages	32	2904	337	7660	7473
FF usages	8	514	0	196	516
FF usages	16	2190	0	412	1944
FF usages	32	5336	116	874	7217

Upon analyzing the Figure 13, it is evident that as the bit-size increases in each implementation method, the IO usage also increases. Additionally, it can be observed that BRAMs are only utilized in the BRAM-based implementations while the DSP units are only used in the Simulink-based implementations. Interestingly, no DSP unit is utilized in the Vivado-based implementation. Overall,

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

the usage of IO, BRAM, and DSP units vary depending on the implementation method and the bit-size used.

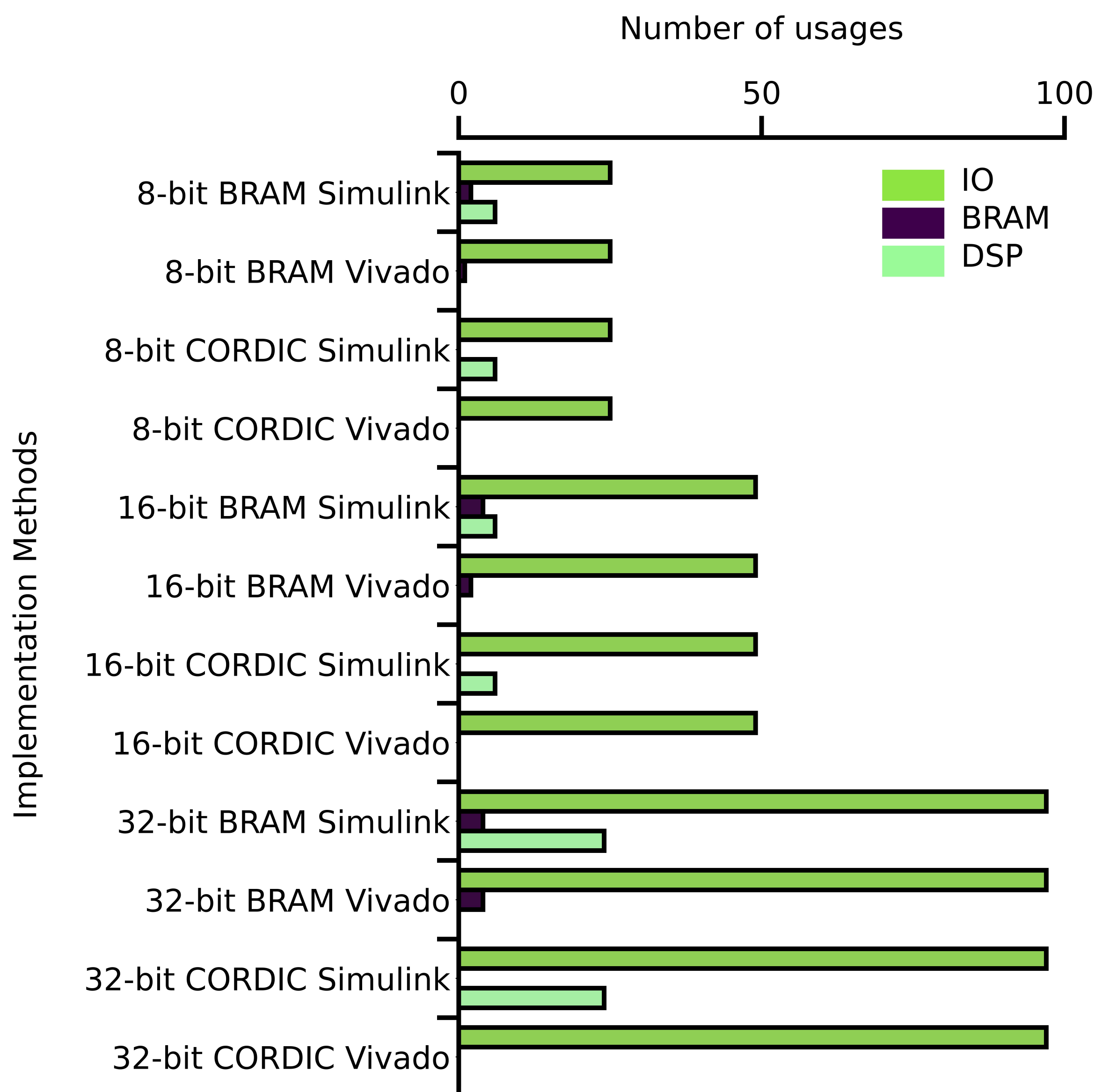

Figure 13. Other resource utilization graph.

4.2.4. Power Requirement

The power requirements in watts for four different types of 16-bit FPGA implementations, namely BRAM Simulink, BRAM Vivado, CORDIC Simulink, and CORDIC Vivado for different operating frequencies are shown in Figure 14. It can be observed that as the operating frequency increases, the power requirement for all four implementations also increases. Among the four implementations, BRAM Simulink requires the least amount of power while CORDIC Simulink and CORDIC Vivado require the highest amount of power. At a frequency of 10 MHz, the power requirement for all four implementations is quite close to each other, ranging from 0.108 watts to 0.114 watts. However, at a frequency of 500 MHz, the power requirement varies significantly among the implementations, ranging from 0.29 watts for CORDIC Simulink to 0.553 watts for CORDIC Vivado. Therefore, it is important to consider the power requirement of different FPGA implementations while selecting the appropriate design for a particular application, especially if the operating frequency is high.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

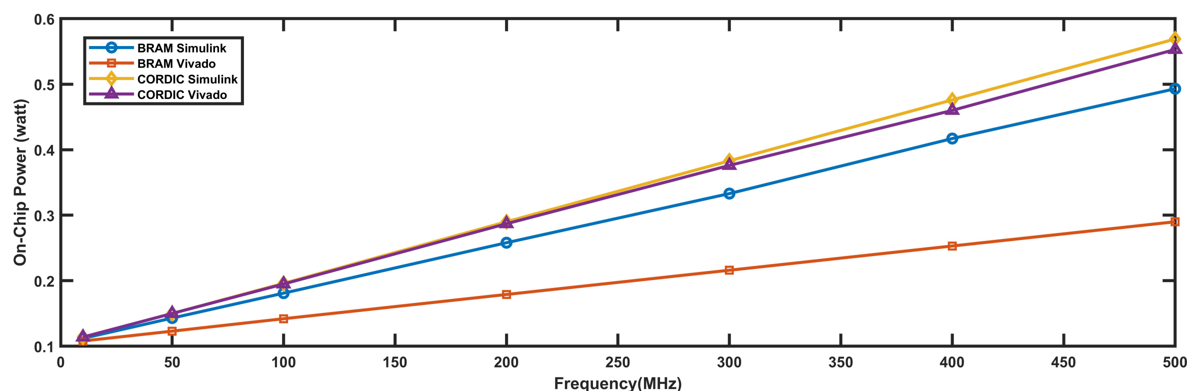

Figure 14. Frequency vs. Power Requirement graph for 16-bit implementation.

4.2.5. Discussion Summary

In this hardware implementation report, four different design methods were compared for implementing a two-neuron network on an FPGA, with a focus on speed, power requirements, and resource usage. The BRAM-based implementation in Vivado achieved better WNS values and higher maximum frequencies compared to the Simulink implementation, while CORDIC-based designs had worse WNS values due to the algorithm's complexity. BRAM-based designs had comparatively lower resource usage, with Vivado implementation requiring fewer resources compared to Simulink.

4.3. Multi-Neuron Implementation Report

After successfully implementing the weighted sum two neurons model, our objective was to evaluate the capabilities of four different implementation methods for real-time computing of the weighted sum of N number of inputs or neurons, where N is 4,8,16,32,64,128. To achieve this, we utilized the ZYNQ-7 ZC702 Evaluation Board and measured how many neurons could be successfully implemented on the FPGA board and how much resources each of the four implementation methods consumed. To achieve real-time computing of the weighted sum of N inputs, we used parallel instantiation of the sum of the two-neuron model. We only considered 16-bit and 32-bit implementations, taking into account output precision.

The Table 2 shows the maximum number of neurons achieved by different implementation methods on the ZYNQ-7 ZC702 Evaluation Board for both 16-bit and 32-bit implementations. For the 16-bit implementation, Simulink BRAM achieved a maximum of 64 neurons while Simulink CORDIC and Vivado CORDIC achieved only 32 neurons each. BRAM-based method in Vivado achieved the highest number of neurons among all the methods with 128 neurons.

Table 2. Maximum number of neurons successfully implemented by the four implementation methods.

8-Neuron Method	Max. Number of Neurons	
	16 bit	32 bit
Simulink BRAM	64	16
Simulink CORDIC	32	8
Vivado BRAM	128	64
Vivado CORDIC	32	8

For the 32-bit implementation, the maximum number of neurons achieved is significantly lower than the 16-bit implementation. Simulink BRAM achieved only 16 neurons while Simulink CORDIC and Vivado CORDIC achieved only 8 neurons each. Vivado BRAM, again, achieved the highest number of neurons with 64. The data shows that the number of neurons that can be successfully implemented on the board varies significantly based on the implementation method and the bit-size used.

Figure 15 displays the LUT and FF resources utilization of the weighted sum of eight neurons for four different implementation methods using a 16-bit and 32-bit implementation. In the case of the 16-bit implementation, the Simulink BRAM and Vivado BRAM implementations consumed the lowest LUT and FF resources, with the latter not utilizing any FF resources. On the other hand, the Simulink CORDIC and Vivado CORDIC implementations consumed significantly higher LUT and FF resources than the BRAM implementations. For the 32-bit implementation, the BRAM implementations (Simulink BRAM and Vivado BRAM) consumed fewer LUT and FF resources than the CORDIC implementations (Simulink CORDIC and Vivado CORDIC), with Vivado CORDIC consuming the most LUT and FF resources. The utilization of LUT and FF resources in the weighted sum of eight neurons implementation was considerably less for the BRAM-based design in Vivado for both 16-bit and 32-bit implementations, when compared to other methods.

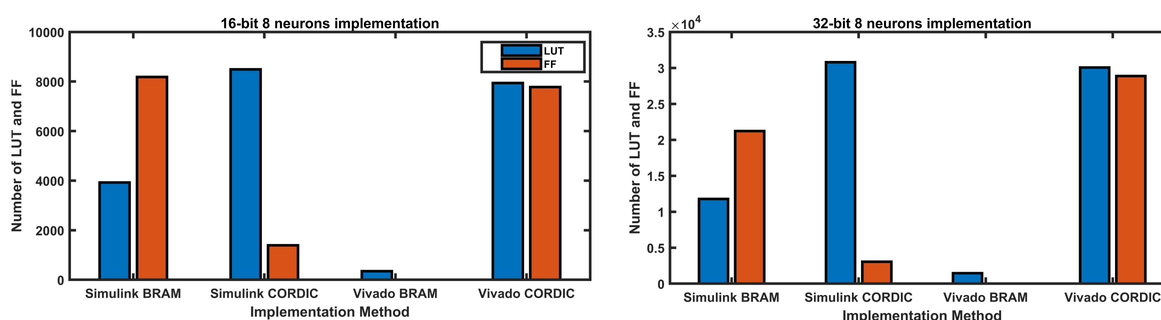

Figure 15. LUT and FF usage in 16-bit and 32-bit implementation for the weighted sum of eight neurons.

From the Table 3, it can be observed that the highest frequency achieved in the 8-neurons implementation for both 16-bit and 32-bit designs was with Vivado CORDIC method, at 200 MHz. The Simulink BRAM and Vivado BRAM implementations achieved the lowest frequencies, with 250 MHz and 50 MHz for 16-bit and 53 MHz for 32-bit, respectively. The Simulink CORDIC implementation had the lowest frequency, achieving only 19 MHz and 9 MHz for 16-bit and 32-bit, respectively.

Table 3. Maximum Frequency achieved by different implementation methods.

8-Neurons Method	Maximum Frequency (MHz)	
	16 bit	32 bit
Simulink BRAM	250	150
Simulink CORDIC	19	9
Vivado BRAM	50	53
Vivado CORDIC	200	200

5. Conclusion and Future Work

In this paper, different design methods were compared for implementing a two-neuron network and a multi-neuron network on an FPGA. The comparison was based on speed, power requirements, and resource usage. The results showed that the BRAM-based implementation in Vivado achieved better WNS values and higher maximum frequencies compared to the Simulink implementation. BRAM-based designs had comparatively lower resource usage, with the Vivado implementation requiring fewer resources compared to Simulink.

For the multi-neuron implementation, the number of neurons that could be successfully implemented on the board varied significantly based on the implementation method and the bit-size used. The Vivado BRAM-based design achieved the highest number of neurons among all the methods with 128 neurons. The utilization of LUT and FF resources in the multi-neuron implementation was considerably less for the BRAM-based design in Vivado for both 16-bit and 32-bit implementations, when compared to other methods.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023
doi:10.20944/preprints202311.0309.v1

Overall, the results show that the choice of implementation method can significantly impact the performance and resource usage of the FPGA design. Therefore, it is important to carefully evaluate different design options and select the one that best meets the specific requirements of the application.

In order to advance the research in this field, the next step would be to create a fully functional complex exponential neural network model. The current work has mainly concentrated on the information encoding aspect of the neural network model. However, in order to develop a complete model, further research is required to establish the decoding method and create a neural network model using complex exponential neurons. Additionally, it would be important to compare the performance of the complex exponential neural network model with other existing neural network models to assess its effectiveness.

Author Contributions: "Conceptualization, M.A. and L.Z.; methodology, M.A. and L.Z.; software, M.A.; validation, M.A., K.T.W.N. and M.E.H.C.; formal analysis, M.A.; investigation, M.A.; resources, K.T.W.N.; data curation, M.A.; writing—original draft preparation, M.A.; writing—review and editing, K.T.W.N. and M.E.H.C.; visualization, M.A.; supervision, L.Z.; project administration, L.Z.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: N/A.

Informed Consent Statement: N/A.

Data Availability Statement: Data sharing not applicable.

Acknowledgments: N/A.

Conflicts of Interest: The authors declare no conflict of interest.

Sample Availability: N/A.

Abbreviations

Abbreviations

The following abbreviations are used in this manuscript:

FPGA	Field-Programmable Gate Array
ASIC	Application-Specific Integrated Circuit
DSP	Digital Signal Processor
HDL	Hardware Description Language
VHDL	VHSIC Hardware Description Language
LUT	Lookup Table
FF	Flip-Flop
SSN	Simultaneous Switching Noise
BRAM	Block RAM (Random Access Memory)
CORDIC	Coordinate Rotation Digital Computer
IO	Input/Output

References

1. Zhang, L. Oscillation Patterns of A Complex Exponential Neural Network. 2022 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT). IEEE, 2022, pp. 423–430.
2. Capra, M.; Bussolino, B.; Marchisio, A.; Shafique, M.; Masera, G.; Martina, M. An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks. *Future Internet* **2020**, *12*, 113.
3. Ghimire, D.; Kil, D.; Kim, S.h. A survey on efficient convolutional neural networks and hardware acceleration. *Electronics* **2022**, *11*, 945.
4. Zhang, J.; Zhang, L. Spiking Neural Network Implementation on FPGA for Multiclass Classification. 2023 IEEE International Systems Conference (SysCon). IEEE, 2023, pp. 1–8.
5. Aizenberg, I. *Complex-valued neural networks with multi-valued neurons*; Vol. 353, Springer, 2011.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 November 2023 doi:10.20944/preprints202311.0309.v1

6. Aizenberg, I.; Herman, J.; Vasko, A. A Convolutional Neural Network with Multi-Valued Neurons: a Modified Learning Algorithm and Analysis of Performance. 2022 IEEE 13th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON). IEEE, 2022, pp. 0585–0591.
7. Javanshir, A.; Nguyen, T.T.; Mahmud, M.P.; Kouzani, A.Z. Advancements in Algorithms and Neuromorphic Hardware for Spiking Neural Networks. *Neural Computation* **2022**, *34*, 1289–1328.
8. Huynh, P.K.; Varshika, M.L.; Paul, A.; Isik, M.; Balaji, A.; Das, A. Implementing spiking neural networks on neuromorphic architectures: A review. *arXiv preprint arXiv:2202.08897* **2022**.
9. Han, J.; Li, Z.; Zheng, W.; Zhang, Y. Hardware implementation of spiking neural networks on FPGA. *Tsinghua Science and Technology* **2020**, *25*, 479–486.
10. Khodamoradi, A.; Denolf, K.; Kastner, R. S2n2: A fpga accelerator for streaming spiking neural networks. The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2021, pp. 194–205.
11. Fang, H.; Mei, Z.; Shrestha, A.; Zhao, Z.; Li, Y.; Qiu, Q. Encoding, model, and architecture: Systematic optimization for spiking neural network in FPGAs. Proceedings of the 39th International Conference on Computer-Aided Design, 2020, pp. 1–9.
12. Gupta, S.; Vyas, A.; Trivedi, G. FPGA implementation of simplified spiking neural network. 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS). IEEE, 2020, pp. 1–4.
13. Kakani, V.; Li, X.; Cui, X.; Kim, H.; Kim, B.S.; Kim, H. Implementation of Field-Programmable Gate Array Platform for Object Classification Tasks Using Spike-Based Backpropagated Deep Convolutional Spiking Neural Networks. *Micromachines* **2023**, *14*, 1353.
14. Guo, W.; Yantır, H.E.; Fouda, M.E.; Eltawil, A.M.; Salama, K.N. Toward the optimal design and FPGA implementation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems* **2021**, *33*, 3988–4002.
15. Hosseiny, A.; Jaberipur, G. Complex exponential functions: A high-precision hardware realization. *Integration* **2020**, *73*, 18–29.
16. Rekha, R.; Menon, K.P. FPGA implementation of exponential function using cordic IP core for extended input range. 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT). IEEE, 2018, pp. 597–600.
17. Wang, D.; Ercegovac, M.D.; Xiao, Y. Complex function approximation using two-dimensional interpolation. *IEEE Transactions on Computers* **2013**, *63*, 2948–2960.
18. Malík, P. High throughput floating point exponential function implemented in FPGA. 2015 IEEE Computer Society Annual Symposium on VLSI. IEEE, 2015, pp. 97–100.
19. Xilinx. Block RAM, 2020. Accessed on April 12, 2023.
20. Xilinx. Introduction to System Generator, 2020.
21. Saidani, T.; Dia, D.; Elhamzi, W.; Atri, M.; Tourki, R.; others. Hardware co-simulation for video processing using xilinx system generator. Proceedings of the World Congress on Engineering, 2009, Vol. 1, pp. 3–7.
22. Xilinx. Introducing the Vivado IDE, 2020. Accessed on April 14, 2023.
23. Susuki, K. Myelin: A Special Membrane for Cell Communication. *Nature education* **2010**, *3*, 59.
24. Michel, H.E.; Awwal, A.A.S. Artificial neural networks using complex numbers and phase encoded weights. *Applied optics* **2010**, *49*, B71–B82.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.