Hardware Accelerators for Softmax Function Approximation Using RTL and HLS Design

Ernani Alexandre Wippel Neto
University of Vale do Itajaí
Itajaí, Brazil
ernani.neto@edu.univali.br

Bruna Henning Pereira
University of Vale do Itajaí
Itajaí, Brazil
bruna.henning@edu.univali.br

Cesar Albenes Zeferino
University of Vale do Itajaí
Itajaí, Brazil
zeferino@univali.br

Abstract

The growing demand for real-time neural network inference has heightened the need for hardware capable of efficiently executing the Softmax function, which relies on exponentials and divisions and is costly, numerically sensitive, and resource-intensive in hardware implementations. This work explores two hardware design approaches—manual Register-Transfer Level design and High-Level Synthesis design—each incorporating distinct approximation strategies to overcome these challenges. By evaluating their cost, performance, and accuracy trade-offs, the study aims to identify the most effective design alternatives and provide practical guidelines for deploying Softmax-based deep learning models on FPGAs in energy and latency-constrained embedded environments.

Keywords

Deep Learining, Softmax function, Digital Design, FPGA

1 Introduction

The increasing use of artificial neural networks in real-time inference applications has driven the search for hardware solutions capable of executing complex mathematical operations with high performance and low energy consumption [1]. Among these operations, the Softmax function plays a fundamental role in multiclass classifiers by transforming the output vector into a probability distribution [2].

Softmax is widely used in the output layers of classification models, as it converts raw values into normalized probabilities, facilitating system interpretation and decision-making. Its mathematical formulation involves exponential and division operations, as described in Section 2.1, which, although conceptually simple, represent a significant challenge when implemented directly in hardware due to their high computational cost.

Implementing Softmax on devices such as FPGAs (Field-Programmable Gate Arrays) and ASICs (Application-Specific Integrated Circuits) poses specific challenges due to its algebraic nature. Part of this complexity arises from the use of exponential functions and divisions, which are not native or efficient operations in digital logic and often require approximations, lookup tables, or auxiliary algorithms to implement them on dedicated hardware [3, 4]. Furthermore, the need to find the maximum value of the input vector and to compute the sum of exponentials introduces data dependencies and parallelism constraints, limiting the achievement of ideal pipeline architectures, particularly in real-time inference applications [5].

Numerical stability issues further aggravate direct implementations. As a result, numerical approximations are often employed, prioritizing optimization and/or ease of design. Additionally, high-speed softmax architectures tend to consume significant amounts of logic area and memory resources, which may be incompatible with the physical and energy constraints of low-power embedded systems [5].

Due to these challenges, alternative approaches—such as mathematical approximations of the exponential function, hardware–software co-design strategies, and low-power integer-based methods—have been proposed to improve Softmax efficiency. These approaches rely on the use of accelerators to reduce both latency and energy consumption without significantly compromising the model's precision [6, 7].

With the growing demand for architectures capable of performing neural network inference with low latency and high energy efficiency, the need to implement hardware accelerators for computationally intensive operations such as Softmax becomes evident. This demand is justified by the fact that general-purpose processors have significant limitations when handling highly parallelizable, high-computational-cost workloads, motivating the use of specialized accelerators [8]. In particular, embedded applications—such as computer vision, autonomous systems, and Internet of Things (IoT) devices—require real-time execution under strict energy constraints, encouraging the use of FPGAs and ASICs [9]. The literature shows that dedicated accelerators can reduce energy consumption by orders of magnitude and substantially increase processing throughput when compared to conventional CPUs (Central Processing Units) and GPUs (Graphic Processing Units) [10]. Moreover, the increasing adoption of larger and more complex models intensifies the need for optimization and mathematical approximation techniques, making accelerators essential to enable efficient execution of critical layers such as Softmax, normalization, and activation functions [5].

Given this context, the present work aims to accelerate the Softmax function processing using two hardware design methodologies with the objective of FPGA synthesis. The first methodology involves manual design at the Register-Transfer Level (RTL) [11]. The second methodology is based on the use of High-Level Synthesis (HLS) tools [12]. The objective is to implement different Softmax approximations using both methodologies and to investigate the trade-offs among cost, performance, and accuracy of the resulting implementations. As a contribution, this work provides guidelines for selecting the best alternatives for implementing deep learning models based on the Softmax function on FPGA.

The remainder of this extended abstract is structured into three sections. Section 2 presents the theoretical background on the Softmax function and the RTL and HLS design methodologies. Section 3 discusses related work, and Section 4 describes the materials and methods that will be used in the development of this work. Section 5 presents the final remarks.

2 Background

2.1 The Softmax function

Considering a vector $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ the Softmax function is formally defined as:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{1}$$

where n is the number of elements in the vector, i denotes the position of the vector element to which the function is applied, and j represents the index iterating over the vector elements in the summation.

This equation ensures that all outputs are always positive and that their total sum is equal to 1, that is:

$$\text{Softmax}(x_i) > 0, \qquad \sum_{i=1}^{n} \text{Softmax}(x_i) = 1 \tag{2}$$

Softmax is sensitive to relative differences between input values, assigning higher probability to the largest element of the vector—a crucial property in classification problems. However, direct computation involves computationally expensive operations, such as exponentiation and normalization, which pose significant challenges for high-performance digital implementations [8]. To avoid numerical overflow issues, a stabilized version of the function is commonly used, in which the maximum value of the vector is subtracted from each element:

$$\text{Softmax}(x_i) = \frac{e^{x_i - \max(x)}}{\sum_{j=1}^{n} e^{x_j - \max(x)}} \tag{3}$$

This reformulation preserves the results while improving numerical stability and is widely adopted in both software and hardware implementations [3]. In digital architectures, especially in programmable logic devices such as FPGAs, exponentiation and division operations remain the main performance bottlenecks, motivating the development of mathematical approximations and specialized accelerators [5].

2.2 RTL and HLS Desing

For this work, two design methodologies are employed. The first methodology is RTL design, which relies on an explicit description of the hardware behavior and internal architecture, specifying the data flow between registers, the associated combinational logic, and the control implemented manually through finite-state machines. In this approach, the hardware designer has complete control over architectural aspects such as pipeline depth, parallelism, allocation of arithmetic operators, signal multiplexing, and timing [11].

The second methodology is HLS design, which involves automatically generating hardware from high-level programming languages such as C, C++, or SystemC. Instead of manually defining registers and combinational logic, the designer specifies the algorithmic behavior, and the HLS tool translates this description into RTL while automatically applying architectural optimizations [12]. Although it does not provide the same level of granularity and fine-grained control available in traditional RTL design, HLS stands out for its productivity, portability, and fast design iteration, constituting a fundamental trade-off between implementation quality and development speed [8], as illustrated in Table 1.

Table 1: Comparison between RTL and HLS approaches

Criterion	RTL Design	HLS Design
Level of control	Full cycle-by-cycle	Partial by optimizations
Performance	Very high, with manual *tuning*	High, depending on the applied pragmas
Development time	Long (months)	Short (weeks)
Implementation flexibility	Low	High
Suitable for	Critical and highly optimized accelerators	Prototyping and complex projects with short deadlines

3 Related Work

The implementation of the Softmax function in hardware has received significant attention in the literature due to its central role in neural networks. Different approaches have been proposed to mitigate the challenges associated with exponentiation, division, and numerical stability.

Works such as Shatravin, Shashev, and Shidlovskiy [3] investigate the direct implementation of the traditional Softmax function using lookup tables to approximate the exponential function and optimized pipelined dividers. The main contribution is demonstrating that well-calibrated approximations can significantly reduce the use of logic resources, making implementation feasible on FPGAs. However, reliance on explicit division operations and large lookup tables limits the solution's scalability, especially when high precision is required.

Among the most innovative proposals, ConSmax [6] stands out by introducing an alternative to Softmax with learnable parameters and greater affinity with parallel architectures. By reducing data dependencies, ConSmax offers clear advantages in latency and energy efficiency. Nevertheless, its practical adoption depends on retraining the models, since it is not directly compatible with previously optimized weights.

Finally, the SoftmAP approach [7] represents a distinct line of research by adopting a fully integer-only implementation. The use of aggressive quantization and the elimination of floating-point operations reduce energy consumption and simplify the hardware, particularly in accelerators associated with the *in-memory computing* paradigm. However, this technique requires an adapted training process and may result in significant performance degradation in models that are sensitive to precision.

In summary, the solutions found in the literature address the problem from different perspectives: mathematical approximations, function reformulation, hardware–software co-design, and replacement with hardware-friendly alternatives.

4 Materials and Methods

Initially, a systematic review of the literature will be conducted, including an analysis of the Softmax function, numerical stabilization

techniques, and classical approximation methods used in hardware implementations. Architectures of neural network inference accelerators will also be studied, with a focus on FPGA devices and RTL and HLS design techniques.

Based on the theoretical foundation, at least two Softmax approximation approaches suitable for hardware implementation will be selected. The selection criteria will include computational complexity, arithmetic operation cost, and numerical stability.

The selected techniques will be initially developed in C/C++ to establish a reference for functionality, accuracy, and performance. This software-based implementation will serve as a baseline for functional validation, numerical and performance comparison, and vector generation. The software versions will also be used as an ideal reference during the validation of the hardware implementations.

After software validation, architectural modeling at the RT level will be performed, defining internal modules, data flow, parallelism, bus widths, and pipeline strategies. The project specification will be documented, including block diagrams and the definition of communication interfaces.

The RTL design of the accelerator will then be implemented in VHDL, encompassing all functional units required to compute the Softmax function or its approximation. The RTL code will be validated through behavioral and post-synthesis simulations using EDA tools suitable for the selected FPGA. The synthesis process will be iteratively refined to optimize performance, logic area, and energy consumption.

In parallel with RTL development, a second version of the accelerator will be generated using HLS techniques. For this purpose, C/C++ code combined with optimization directives provided by the high-level synthesis flow will be used. This approach will enable a direct comparison between the two strategies—RTL and HLS—by evaluating their advantages and limitations in terms of development effort and resulting performance.

The resulting implementations (RTL and HLS) will be synthesized and deployed on a test FPGA platform. The experimental evaluation will include:

- *Logic resource utilization*: number of LUTs, FFs, DSPs, and BRAMs used on the FPGA, according to synthesis and place-and-route reports.
- *Performance*: maximum operating frequency, latency per input vector, and throughput.
- *Energy cost*: total dissipated power, estimated using vendor tools, and energy consumption, computed as the product of latency and dissipated power.
- *Numerical accuracy*: comparison between the hardware results and the software reference, considering absolute error, relative error, and numerical stability.

All tests will be conducted using standardized sets of previously input vectors generated by software.

The results will be analyzed and compared, considering both quantitative metrics (performance, area, power, and precision) and qualitative aspects (implementation complexity, flexibility, and development effort). This analysis will enable identification of the advantages and limitations of traditional RTL and HLS-based flows, as well as determination of which Softmax approximation technique offers the best cost–benefit trade-off for real-time neural inference applications.

5 Conclusion

This project is currently in its initial execution phase, and the next steps include conducting a systematic literature review, selecting Softmax approximation techniques, and implementing and evaluating the proposed accelerators. As outcomes, this work is expected to provide guidelines for implementing Softmax-based deep learning models on FPGA platforms and a library of synthesizable Softmax models using different approximation approaches.

Acknowledgments

This work was supported by the Brazilian National Coordination of Superior Level Staff Improvement – CAPES (Financial Code 001), Research and Innovation Support Foundation of Santa Catarina State – FAPESC (Grant 2025TR001570), and the Brazilian National Council for Scientific and Technological Development – CNPq (PIBITI Program and Grants 408641/2023-1 and 306478/2025-0). During the preparation of this work, the authors used Grammarly and ChatGPT to revise grammar and enhance clarity and conciseness in specific sections of the text. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the article's content. All conceptual, methodological, analytical, and interpretative content remains entirely the authors' own work.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA: MIT Press, 2016. DOI: 10.7551/mitpress/10243.001.0001.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. 4th ed. Cambridge, MA: MIT Press, 2021. DOI: 10.7551/mitpress/13811.001.0001.

[3] Andrey Shatravin, Dmitry Shashev, and Dmitry Shidlovskiy. "Implementation of the SoftMax Activation for Reconfigurable Neural Network Hardware Accelerators". In: *Applied Sciences* 13.23 (2023), p. 12784. DOI: 10.3390/app132312784.

[4] Michele Di Franco et al. "A pseudo-softmax function for hardware-based high speed image classification". In: *Scientific Reports* 11 (2021), p. 4144. DOI: 10.1038/s41598-021-83191-9.

[5] Chi-Hwan Lee et al. "Hardware Accelerator for Approximation-Based Softmax and Layer Normalization in Transformers". In: *Electronics* 14.12 (2025), p. 2337. DOI: 10.3390/electronics14122337.

[6] Shiwei Liu, Guanchen Tao, Yifei Zou, et al. "ConSmax: Hardware-Friendly Alternative Softmax with Learnable Parameters". In: *arXiv preprint* arXiv:2402.10930 (2024). DOI: 10.48550/arXiv.2402.10930. URL: https://arxiv.org/abs/2402.10930.

[7] Mariam Rakka et al. "SoftmAP: Software-Hardware Co-Design for Integer-Only Softmax on Associative Processors". In: *arXiv preprint* arXiv:2411.17847 (2024). DOI: 10.48550/arXiv.2411.17847. URL: https://arxiv.org/abs/2411.17847.

[8] Vivienne Sze et al. "Hardware for machine learning: Challenges and opportunities". In: *Journal of Machine Learning Research* 20.1 (2019), pp. 1–81. DOI: 10.48550/arXiv.1712.05941.

[9] Shahanur Alam et al. "Survey of Deep Learning Accelerators for Edge and Emerging Computing". In: *Electronics* 13.15 (2024), p. 2988. DOI: 10.3390/electronics13152988.

[10] Norman P. Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit". In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 2017, pp. 1–12. DOI: 10.1145/3079856.3080246.

[11] Frank Vahid. *Digital Design with RTL Design, VHDL, and Verilog*. 2nd ed. New York, NY: John Wiley & Sons, 2011. DOI: 10.1002/9781118008883.

[12] *Vivado Design Suite User Guide: High-Level Synthesis (UG902)*. No DOI available. Xilinx. San Jose, CA, 2020.