CSE240D Project Proposal
RTL Implementation of Parametric, Non-Linear Activation Functions, Random Number Generation and Softmax

Rishabh Kumar	**Pranav Raj**	**Yuchuan Li**
ECE	ECE	ECE
A59026395	A59026510	A69026842

Abstract

This project focuses on designing and implementing a hardware-based solution for several parameterized non-linear activation functions, pseudo-random number generation (PRNG) for integer and floating-point numbers, and the softmax function using Register Transfer Level (RTL) design. These operations are critical in neural networks, cryptography, and statistical computations, where real-time performance, low latency, and power efficiency are required. Implementing these components in RTL will provide acceleration and optimize computational resources for real-time systems, embedded applications, and specialized hardware. This proposal outlines the definition of the problem, it's importance, and our methodology for solving it.

1 Problem Definition

Motivation: Non-linear activation functions, random number generation, and the softmax function are essential building blocks in machine learning models. When implemented in software, these operations are computationally expensive, requiring significant processing time and energy. For real-time systems, especially in neural networks, high-performance solutions are critical. By implementing these functions in hardware, we can achieve substantial speedup and improved energy efficiency, allowing for real-time computation in resource-constrained environments.

Key Problem: The challenge is the efficient hardware implementation of complex non-linear activation functions and pseudo-random number generation (PRNG) for both integers and floating-point numbers. These tasks involve precise mathematical operations that can be computationally intensive. In addition, the softmax function, which is heavily used in neural networks, needs to be computed efficiently in real time for classification tasks. Our project will address:

- Efficient hardware implementation of parameterized non-linear activation functions (PReLU, GELU, tanh, sigmoid).
- A robust and high-throughput RTL design for PRNG capable of generating both integer and floating-point numbers.
- Real-time RTL implementation of the softmax function for machine learning inference tasks.

2 Tentative Method

The project will be divided into three primary modules, each addressing a core function:

2.1 Parameterized Non-Linear Activation Functions:

2.1.1 Gaussian Error Linear Unit (GELU)

GELU applies a probabilistic approach using the CDF of the standard normal distribution to activate inputs. It is defined as:

$$f(x) = x \cdot \Phi(x)$$

with an approximation:

$$f(x) \approx 0.5 \cdot x \cdot \left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$$

Properties and Applications GELU is smooth and probabilistic, promoting stable gradients. It is used in:

- **Transformer Models:** Integral to architectures like BERT and GPT.
- **Computer Vision:** Enhances representation stability.

Implementation Approach

The GELU function is implemented using a probabilistic approach that utilizes the cumulative distribution function (CDF) of the standard normal distribution. The output is computed as $\text{GELU}(x) = x \cdot \Phi(x)$, where $\Phi(x)$ is approximated using hardware-friendly methods, such as polynomial approximations or lookup tables to reduce computational complexity. This approach enhances efficiency while maintaining accuracy, making it suitable for rapid inference in deep learning applications [1], [2].

2.1.2 Sigmoid

The sigmoid function compresses inputs to the range (0, 1), defined as:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Implementation Approach

The sigmoid function is implemented by approximating e^{-x} and computing the output as $\sigma(x) = \frac{1}{1+e^{-x}}$. Sigmoid functions can utilize similar approximation techniques as softmax shown in Table 1 to efficiently approximate their outputs in hardware with optimized accuracy and resource usage. This method is scalable to fit within a 16-bit signed integer, making it effective for digital circuits. The design focuses on synthesizability, which is crucial for deployment in real-time systems [3], [4].

2.1.3 Hyperbolic Tangent (tanh)

The tanh function squashes inputs to the range $(-1, 1)$, defined as:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Implementation Approach

The hyperbolic tangent (tanh) function can be efficiently implemented in hardware by leveraging the sigmoid function, using the relationship:

$$\tanh(x) = 2 \cdot \sigma(2x) - 1$$

where $\sigma(x)$ represents the sigmoid function. This approach allows existing hardware optimized for sigmoid calculation in subsection 2.1.2, then compute tanh with minimal additional resources by scaling the input and output.

2.1.4 Parametric ReLU (PReLU)

PReLU introduces a learnable parameter α for negative inputs, defined as:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

Implementation Approach

The PReLU function is implemented using basic conditional logic, allowing the output to adapt based on the input value x. For positive inputs, the output is simply x, while for negative inputs, it is computed as $\alpha \cdot x$ [5], [6].

2.2 Pseudo-Random Number Generator (PRNG)

2.2.1 Integer PRNG

We will use a Linear Feedback Shift Register (LFSR) for generating pseudo-random integers. LFSR is a hardware-efficient method for generating long pseudo-random sequences by shifting and XORing certain bit positions (taps).

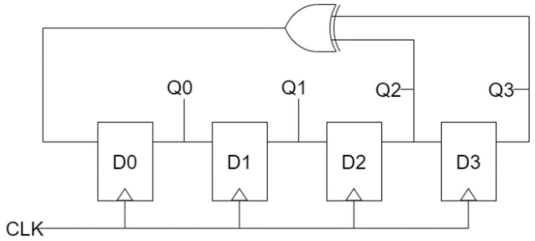

Figure 1: 4-bit LFSR

2.2.2 Floating-Point PRNG

To generate pseudo-random floating-point numbers, we will normalize the integer output of the LFSR-based PRNG by dividing by a large constant, typically 2^{32}. This gives us a random floating-point number in the range $[0, 1)$:

$$\text{random float} = \frac{\text{random integer}}{M}$$

Where M is the normalization factor.

2.3 Softmax Function

The softmax function converts a vector of raw scores (logits) into probabilities, ensuring the outputs sum to 1. It is commonly used in the final layer of neural networks for multi-class classification. The softmax function is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Where z is the input vector and z_i is the i-th component. This function ensures that the output values are between 0 and 1, making them interpretable as probabilities.

Softmax equation implies it contains expensive exponentiation and division, thereby causing the hardware design of softmax layer to suffer from high complexity, long critical path delay and overflow problems [7].

In real-world applications of deep neural networks (DNNs) and transformers, high precision in the Softmax function is often not strictly necessary in most cases. This is because neural networks

are generally robust to small numerical inaccuracies, and various approximation techniques can still provide sufficiently accurate results for the Softmax function without needing full floating-point precision. Table 1 shows the comparison of some of the most popular methods for hardware implementation of the Softmax function

Table 1: Trade-off comparison of Softmax hardware implementation methods

Approximation Method	**Notes**
Lookup Table (LUT)	Simple but can be memory-intensive for high precision; requires interpolation for large ranges.
Piecewise Linear Approx.	Simple to implement with adders and multiplexers; accuracy depends on number of segments.
Piecewise Polynomial Approx.	Better accuracy than linear approximation; requires multipliers for polynomial terms.
Log-Sum-Exp Trick	Improves numerical stability by avoiding overflow; adds extra subtraction step.
Log Softmax	Calculates the log of softmax directly, which reduces the risk of overflow and improves stability.
Taylor Series Expansion	Good accuracy with more terms, but becomes resource-intensive for higher-order expansions.
Newton-Raphson for Exp.	Accurate for exponential computation but requires iterative calculations.

Implementation Approach

We propose a hybrid approach combining Lookup Table (LUT), Log-Softmax, and Log-Sum-Exp approximation to achieve an efficient Softmax implementation.

Use **Log-Sum-Exp** to improve numerical stability by subtracting the maximum logit, avoiding overflow/underflow during exponentiation. Implement **Log-Softmax** to skip direct exponentiation and division by computing probabilities in logarithmic space, reducing computational complexity. Integrate a **LUT** to store precomputed values for logarithms and exponentials, speeding up these operations with minimal hardware resource usage [7].

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \quad \Rightarrow \quad \text{Log-Softmax}(x_i) = x_i - \log\left(\sum_{j=1}^{N} e^{x_j}\right)$$

References

[1] K. Wang et al., "iBERT: A Specialized Hardware for Efficient Inference of Transformer-based Language Models," *IEEE Transactions on Computers*, vol. 71, no. 2, pp. 555-568, 2022. DOI: https://doi.org/10.1109/TC.2021.3083159.

[2] A. Gupta et al., "Towards Efficient Hardware Implementation of Activation Functions in Deep Learning," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 14, no. 4, pp. 1-22, 2021. DOI: https://doi.org/10.1145/3458624.

[3] J. Lee et al., "Efficient FPGA Implementation of the Hyperbolic Tangent Activation Function for Neural Networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 6, pp. 1189-1200, 2021. DOI: https://doi.org/10.1109/TVLSI.2020.3029510.

[4] R. Smith et al., "Optimized Hardware Implementation of Activation Functions for Deep Learning," *Journal of Signal Processing Systems*, vol. 93, no. 4, pp. 455-465, 2021. DOI: https://doi.org/10.1007/s11265-021-01795-7.

[5] M. Zhang et al., "Hardware Implementation of Parametric Rectified Linear Unit (PReLU) in FPGA," *Journal of Hardware and Systems Security*, vol. 5, no. 3, pp. 145-156, 2021. DOI: https://doi.org/10.1007/s41635-021-00123-1.

[6] T. Liu et al., "Design and Implementation of a PReLU Activation Function on a Reconfigurable Platform," *IEEE Access*, vol. 9, pp. 37640-37652, 2021. DOI: https://doi.org/10.1109/ACCESS.2021.3065620.

[7] B. Yuan, "Efficient hardware architecture of softmax layer in deep neural network," *2016 29th IEEE International System-on-Chip Conference (SOCC)*, Seattle, WA, USA, 2016, pp. 323-326, DOI: https://doi.org/10.1109/SOCC.2016.7905501.