HENNC: Hardware Engine for Artificial Neural Network-Based Chaotic Oscillators

Mobin Vaziri*, Shervin Vakili†, M.M. Rahimifar‡ and J.M. Pierre Langlois*
* Department of Computer and Software Engineering, Polytechnique Montréal, Canada
† Institut national de la recherche scientifique (INRS), Montréal, Canada
‡ Interdisciplinary Institute for Technological Innovation - 3IT, Université de Sherbrooke, Canada
{mobin.vaziri, pierre.langlois}@polymtl.ca
mohammad.mehdi.rahimifar@usherbrooke.ca
shervin.vakili@inrs.ca

***Abstract*—This paper introduces a framework for automatically generating hardware cores for Artificial Neural Network (ANN)-based chaotic oscillators. The framework starts with training a model to approximate a chaotic system and subsequently conducts design space exploration to identify potential hardware architectures for implementation. From the selected solution, the framework generates synthesizable High-Level Synthesis (HLS) code and a validation testbench tailored for Field-Programmable Gate Arrays (FPGAs). The proposed framework enables a rapid hardware design process for candidate architectures, offering superior hardware cost efficiency and throughput compared to manually designed solutions. The source code is available on GitHub[1].**

***Index Terms*—Chaotic Systems, High-Level Synthesis, Field-programmable Gate Arrays, Artificial Neural Networks.**

I. Introduction

Chaotic systems have found applications across various fields, including cryptography and secure communications [1]. These systems are particularly effective in image encryption, where their inherent unpredictability ensures robust security against cyber-attacks by scrambling pixels in a highly unpredictable manner [2]. For these applications, a high-throughput Pseudo-Random Number Generator (PRNG) is required for real-time operation [3]. Chaotic oscillators are an appealing choice for PRNGs because of their complex dynamics, ergodicity, and sensitivity to initial conditions. Related studies have shown that, compared to traditional PRNGs such as Linear Feedback Shift Registers (LFSRs), chaotic oscillators offer significantly higher resilience against attacks [4], [5]. While numerical methods such as Forward Euler (FE), Runge-Kutta (RK) [6]–[8], and Heun [9] can produce accurate solutions for even highly complex chaotic systems, their hardware implementation is typically resource-intensive [7].

Previous research has demonstrated the feasibility of using Artificial Neural Networks (ANNs) to approximate chaotic systems [10]–[13]. Alcin et al. [11] proposed an ANN for a chaotic system on Field Programmable Gate Arrays (FPGAs). However, the costly implementation of the sigmoid function can significantly impair system performance. Sunny et al. [12] applied Nonlinear Auto-Regressive (NAR) and Nonlinear Auto-Regressive with Exogenous Inputs (NARX) ANNs to model chaotic systems. Nonetheless, NAR and NARX ANNs tend to accumulate feedback errors, which can affect the generated sequences. Al-Musawi et al. [13] introduced an ANN model with fewer hidden neurons; however, this approach resulted in high hardware resource consumption.

This paper presents a comprehensive computer-aided framework for the automated design of efficient hardware architectures for ANN-based chaotic oscillators. By inputting ANN hyperparameters, users receive a list of Pareto-optimal candidate microarchitectures generated by the framework. The framework then produces the corresponding High-Level Synthesis (HLS) model for the selected candidate. The main contributions of the paper are:

- A framework for the automated generation of efficient hardware architectures for chaotic oscillators through a rapid design space exploration process. It provides a list of Pareto-optimal candidate architectures, each offering a distinct trade-off between cost and performance.
- Acceleration of the hardware design process through HLS modeling. HLS directives are employed to define the trade-offs between hardware cost and latency.
- Introduction of novel latency and hardware cost estimation functions to speed up the design space exploration.

The remainder of the paper is structured as follows: Section II provides an overview of chaotic systems and their implementation methods. Section III details our proposed framework, covering its phases and estimation functions. Section IV discusses and compares our experimental results with related studies. Finally, Section V concludes the paper.

II. Background

Fig. 1 illustrates a chaotic oscillator incorporating a multiplexer (MUX) and a chaotic unit. Solving chaotic systems often involves discretization through methods like the RK algorithm. While RK is a robust analytical method for solving differential equations, it can be computationally expensive [7], [8]. ANNs offer a cost-effective alternative for computing non-linear equations, making them suitable for modeling chaotic systems [11]. The performance of such systems largely depends on the implementation approach within the chaotic

[1]https://github.com/INRS-ECCoLe/HENNC

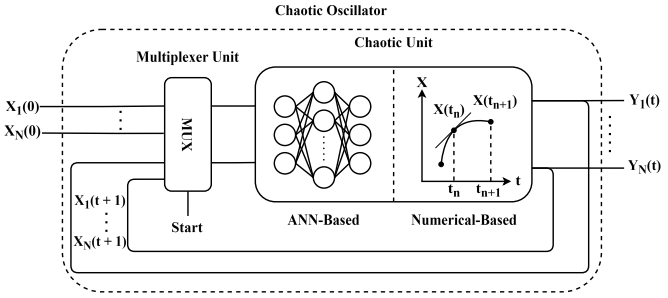

Fig. 1. Architecture of chaotic oscillators.

unit. Key parameters include the number of input and output neurons, which correspond to the dimensionality of the chaotic system, and the number of hidden neurons, which impacts the model's accuracy. Zhang [10] investigated the optimal number of hidden neurons based on the resulting Mean-Square Error (MSE). Their findings indicated that increasing the number of hidden neurons beyond eight did not result in significant improvements in accuracy. This study highlighted the feasibility of using ANNs to effectively model chaotic systems. Further evidence from Yu et al. [14] demonstrated that ANN-based chaotic systems could generate random sequences that successfully pass the stringent randomness tests outlined in the NIST SP 800-22 test suite [15].

Consider a system of *N* differential equations:

$$\frac{dX_i}{dt} = f_i(X_1, X_2, ..., X_N), \tag{1}$$

where X_i are system variables, and *t* denotes time. The RK 4th-order (RK-4) method is calculated two steps: state variable coefficients (2) and variable update (3).

$$\begin{aligned} k_{1i} &= f_i(X_1, X_2, ..., X_N) \\ k_{2i} &= f_i\left(X_1 + \frac{dt}{2} \cdot k_{11}, ..., X_N + \frac{dt}{2} \cdot k_{1N}\right) \\ k_{3i} &= f_i\left(X_1 + \frac{dt}{2} \cdot k_{21}, ..., X_N + \frac{dt}{2} \cdot k_{2N}\right) \\ k_{4i} &= f_i\left(X_1 + dt \cdot k_{31}, ..., X_N + dt \cdot k_{3N}\right) \end{aligned} \tag{2}$$

$$X_{i+1} = X_i + \frac{1}{6} \times (k_{1i} + 2 \times k_{2i} + 2 \times k_{3i} + k_{4i}) \tag{3}$$

The multiplications and additions for RK-4 in (2) and (3) can be divided into static and dynamic terms. Static terms encompass operations that calculate the input arguments in (2). Dynamic terms refer to operations present in the selected chaotic system, f_i. The total number of operations is:

$$\begin{aligned} n_{mul} &= n(mul_{static}) + n(mul_{dynamic}) \\ &= (3 \cdot N^2 + 3 \cdot N) + 4 \cdot n(mul_{dynamic}) \\ n_{add} &= n(add_{static}) + n(add_{dynamic}) \\ &= (3 \cdot N^2 + 4 \cdot N) + 4 \cdot n(add_{dynamic}). \end{aligned} \tag{4}$$

The dynamic terms depend on the selected chaotic system. For instance, for the low-complexity Chen chaotic system [6]:

$$\begin{cases} \frac{dX_1}{dt} &= a \cdot (X_2 - X_1), \\ \frac{dX_2}{dt} &= (c - a) \cdot X_1 - (X_1 \cdot X_3) + c \cdot X_2, \\ \frac{dX_3}{dt} &= X_1 \cdot X_2 - b \cdot X_3 \end{cases} \tag{5}$$

six multiplications and five additions are needed. On the other hand, ANN computation in the layer *l* and i^{th} neuron is given by:

$$Y_i^{(l)} = \phi\left(\sum_{j=1}^{n_{l-1}} W_{ij}^{(l)} X_j + b_i^{(l)}\right) \tag{6}$$

where ϕ is the activation function, n is the number of neurons in the layer, and w and b are the weights and biases of the model, respectively. The number of operations in a general ANN is obtained by:

$$\begin{aligned} n_{mul} &= \sum_{i=2}^{L} n_i \times n_{i-1} \\ n_{add} &= \sum_{i=2}^{L} n_i \times (n_{i-1} + 1). \end{aligned} \tag{7}$$

Table I compares the number of operations to implement a minimal chaotic system using the RK-4 method and an ANN proposed by Zhang [10]. ANNs execute a predetermined number of operations based on their structure. However, the computational complexity of the RK-4 method depends on the specific chaotic system it employs. These chaotic systems rely on trigonometric or other non-linear functions, which are computationally expensive. As a result, while achieving minimal error (as shown in Table II) compared to numerical methods, ANNs can provide more stable and efficient solutions.

TABLE I
NUMBER OF OPERATIONS IN AN ANN WITH 8 NEURONS IN THE HIDDEN LAYER, AND IN RK-4 WITH CHEN CHAOTIC SYSTEM

Method	# Multiplications	# Additions
ANN (3 – 8 – 3) [10]	48	59
RK-4	60	59

III. PROPOSED FRAMEWORK

This section describes the two phases of the HENNC framework, illustrated in Fig. 2.

A. Software Phase

The software phase of the HENNC framework begins with generating a chaotic sequence dataset by numerically solving the defined chaotic system using the *Odeint* function from Python's *SciPy* library. We generate 100 k sequences and use 80% for training and 20% for testing. To create the training dataset, we sample the output of the numeric chaotic system at each time step. Since the output at time step t is the input to the

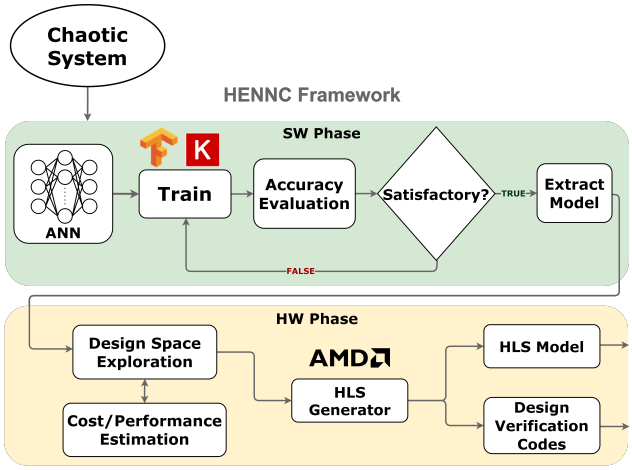

Fig. 2. HENNC framework design flow

TABLE II
HYPERPARAMETERS AND PERFORMANCE OF ANN-BASED MODEL

Hyperparameters				
Loss Function	Optimizer	Learning Rate		
MSE	Adam	1×10^{-4}		
Performance Metrics				
Activation Function	MSE	MAE	RMSE	R^2
ReLU	0.000308	0.011653	0.017547	0.999990
Tanh	0.006976	0.44344	0.083520	0.999819
Sigmoid	0.044117	0.109703	0.210040	0.998751

system at time step $t+1$, each labeled data point consists of the output samples from two consecutive time steps. The dataset in the initial phase is used to train the ANN model, which is built with the *Keras* library, and approximates the chaotic system's function through regression. Users specify model hyperparameters, and the model's performance is evaluated with the Mean Absolute Error (MAE), Mean Square Error (MSE), Root Mean Square Error (RMSE), and R-squared (R^2). Table II presents a hyperparameter set for training the model for Chen's chaotic system. The training process terminates when the model achieves the desired accuracy, and the network parameters are extracted for the hardware phase.

B. Hardware Phase

1) Design Space Exploration: Hardware design begins with the search for Pareto-optimal microarchitectures for the ANN model trained in the software phase. The HENNC framework incorporates a highly-configurable template HLS design with various model hyperparameters and a set of HLS directives that trade off parallelism and resource utilization. This flexibility allows the exploration of various ANN core microarchitectures.

HENNC offers users three options: minimum latency, lowest cost, and Pareto-optimal solutions with a parallelism level *P*. In the lowest cost solution (*P = 0*), the hardware core contains only one adder and one multiplier. Increasing parallelism

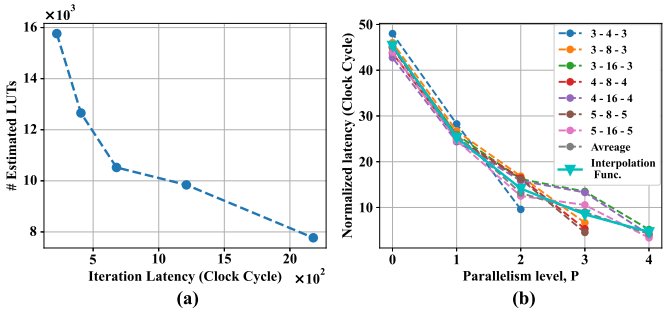

Fig. 3. (a) Estimated cost and latency for the 3-16-3 ANN. (b) Normalized actual latencies and the interpolation curve when utilizing DSP resources.

reduces latency but increases costs. When $P \geq 1$, the number of multipliers and adders is $(2^P \times I)$, with I denoting the number of neurons in the input and output layers. For faster candidate solution characterization, HENNC incorporates a cost and throughput estimation method, described in the next subsection. Fig. 3a depicts the architectural design space, with the estimated cost and latency for a 3-16-3 ANN.

The hardware cores support single-precision floating-point computations and, by default, map these operations onto FPGA Digital Signal Processing (DSP) resources. Users have the option to utilize FPGA Look-Up Tables (LUTs) exclusively. Once the user selects the preferred candidate solution, the HENNC framework generates the corresponding chaotic oscillator HLS model.

2) Estimation functions: Pre-synthesis cost and speed estimation with HLS tools is time-consuming with limited accuracy. To tackle this issue, we developed cost and latency estimation functions with an experimental approach. To estimate latency, we began by measuring the post-synthesis actual latency, in terms of the number of clock cycles, for a range of selected solutions across various ANN sizes and parallelism levels. After normalizing the latency results, we computed the average latency for each parallelism level. By analyzing the average latency values using MATLAB Curve Fitter toolbox, we concluded that a third-degree polynomial approximation provides satisfactory accuracy. Hence, the resulting latency estimation function is as follows:

$$Latency = (I \cdot H) \cdot \left(b_3 P^3 + b_2 P^2 + b_1 P + b_0\right), \quad (8)$$

where I is the number of neurons in the input/output layers, H is the number of neurons in the hidden layer, b_3 to b_0 are constant coefficients, and P is the parallelism level. Due to substantial variations in latency with or without DSP utilization, the measurements and interpolation calculations described above are carried out separately for the two scenarios, yielding distinct value sets for b_3 to b_0. Fig. 3b presents the actual latency results when utilizing DSP resources, shown in terms of the number of clock cycles normalized by dividing them by $I \times H$. The MATLAB Curve Fitter toolbox was used to determine the interpolation function and the coefficients b_3 to b_0. This process was repeated separately for LUT-based solutions.

TABLE III
HARDWARE COSTS FOR HENNC CANDIDATE DESIGNS AND EXISTING WORKS.

HENNC Estimations								Post-Synthesis								
			LUTs		DSPs	Iteration Latency (Clock Cycles)		LUTs		FFs		DSPs	fmax (MHz)		Iteration Latency (ns)	
ANN	P	Solutions	With DSP	No DSP	With DSP	With DSP	No DSP	With DSP	No DSP	With DSP	No DSP	With DSP	With DSP	No DSP	With DSP	No DSP
3 – 4 – 3	2	1	4100	10708	44	169	128	4255	11358	8754	14909	44	510	548	224.25	167.44
	1	2	2288	5592	22	302	253	3683	7089	7383	10334	22	510	548	661.05	516.88
	0	3	2215	2896	5	544	500	2853	3314	5633	5735	5	510	548	1123.20	953.68
3 – 8 – 3	3	1	7988	21204	88	203	165	8051	22310	15020	27439	88	510	548	312.01	227.50
	2	2	5744	12352	44	339	256	5885	12279	9981	14210	44	494	387	816.08	717.24
	1	3	5180	8484	22	605	506	5089	8307	7944	10001	22	494	387	1087.83	1017.70
	0	4	4067	4721	5	1089	1001	3962	4440	6388	6485	5	494	386	1848.10	1880.34
3 – 16 – 3	4	1	15764	42196	176	223	290	15671	43604	28723	53538	176	510	548	483.60	343.98
	3	2	12656	25872	88	407	330	11868	24181	19592	27516	88	494	370	1308.96	1144.80
	2	3	10524	17132	44	678	513	10271	16511	14688	18338	44	494	387	1571.56	1377.72
	1	4	9841	13145	22	1211	1013	9846	12844	16818	19396	22	510	548	2427.75	1914.64
	0	5	7771	8425	5	2178	2003	7726	7795	15051	14759	5	510	548	4212.10	3618.16
3 – 4 – 3 [13]	-	1	-	-	-	-	-	21172	-	96	-	162	7	-	-	-
3 – 8 – 3 [11]	-	1	-	-	-	-	-	87207	-	86329	-	8	266	-	543.75	-

For the cost estimation, we generated and synthesized a range of solutions to gain insights into how LUT utilization varies with hyperparameter values and parallelism. LUT usage was measured as we incremented the H and the I while using two levels of parallelism. There is a semi-linear relationship between LUT utilization and both H and I. However, increasing the level of parallelism exerts a non-linear influence on the hardware core control circuit. we opted to express the estimation of LUT usage as a linear function of I and H for each parallelism level (P), employing the following formulation:

$$\#LUT = (c_1 \cdot I \cdot H) + (c_2 \cdot I) + (c_3 \cdot H) + \beta, \quad (9)$$

where c_1, c_2, c_3, and β are coefficients determined experimentally for each parallelism level.

These coefficients were established through experimentation for each unique parallelism level, with a curve fitting tool. For every potential parallelism level, we conducted linear interpolation of LUT vs. H and LUT vs. I results from actual result curves, such as those depicted in Fig. 4. Subsequently, we determined the coefficient values by equating Eq. 9 with the linear interpolation results. The obtained coefficient values for various parallelism levels were then compiled to create a constant coefficient table. When assessing the LUT cost for a specific candidate architecture, the HENNC framework retrieves the c_1, c_2, c_3, and β values associated with the requested parallelism level from this table. These values are then incorporated into Eq. 9 to derive the estimation function with two variables, I and H.

3) Hardware core generation: In the final step, the HENNC framework generates the HLS design and verification codes for the user-selected solution. Two distinct C++ code files are generated: one for the synthesizable hardware design and another for the testbench to aid in design validation in AMD-Vitis HLS.

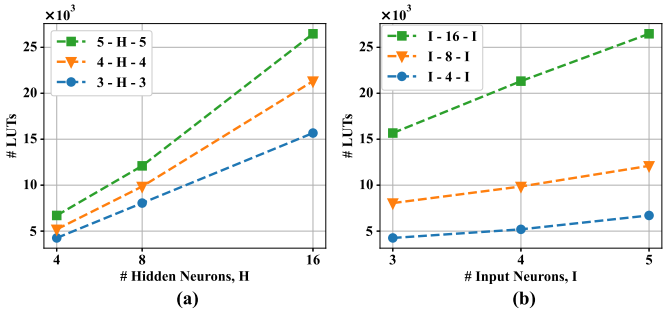

Fig. 4. Post-synthesis number of LUTs as a function of the number of neurons in the (a) hidden layer, (b) input and output layers.

IV. RESULTS

We used AMD-Xilinx Vitis for HLS synthesis, AMD-Xilinx Vivado ML 2023.1 for RTL synthesis, targeting a xcku035 Kintex Ultrascale FPGA with a speed grade of -3. Table III lists the hardware costs for the three most commonly used ANN models for approximating 3-D chaotic systems. It shows that all HENNC solutions require significantly fewer LUTs and operate at considerably higher clock frequencies compared to [11], [13]. In contrast to the architecture proposed by Alcin et al. [11], HENNC's fastest architecture offers reduced latency while consuming nearly 20× fewer LUTs. The other two HENNC solutions have increased latency but lower hardware costs. A primary reason for the substantial decrease in LUT utilization is the adoption of ReLU as an activation function. In contrast, prior works such as [13] employed the exponential function, and [11] utilized numerous floating-point dividers and CORDIC cores to approximate the activation function. For a 3-8-3 ANN, an Intel 12th Gen Intel(R) Core(TM) i7-12700 CPU generates 100 output samples in approximately 3.5 seconds, while the FPGA-based design accomplishes the same task in 31 microseconds.

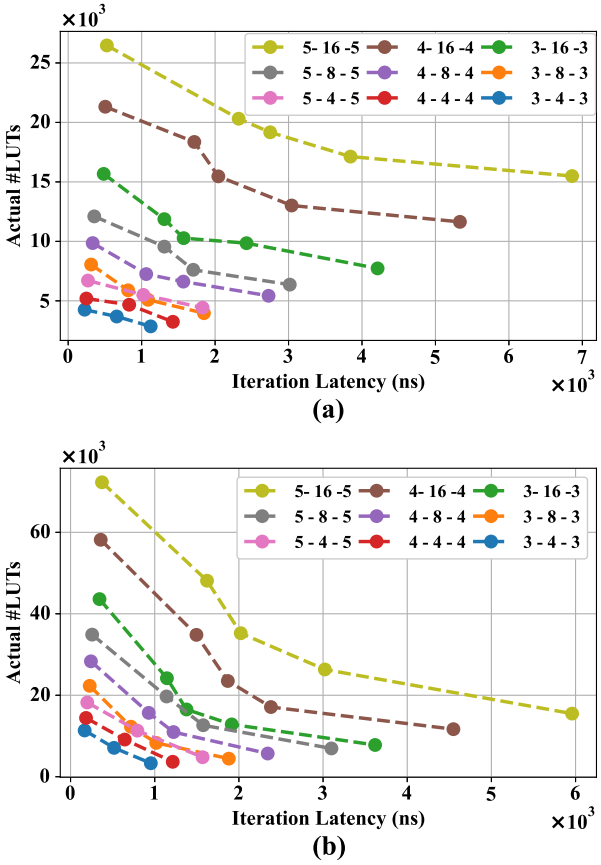

Fig. 5. Post-synthesis hardware cost and latency of the design space explored by HENNC for different ANN sizes in two modes: (a) with DSP utilization, and (b) without DSP utilization.

Fig. 5 represents the HENNC design space for different ANN sizes with and without DSPs. The figure illustrates the trade-offs between post-synthesis LUT utilization and the latency offered by each candidate solution. For each ANN size, the smallest and largest solutions in terms of LUT usage represent the top-speed and cost-optimized solutions, respectively. The top-speed hardware leverages maximum parallelism by employing the maximum number of MAC operators. In contrast, the cost-optimized mode typically deploys only a single MAC unit to calculate all neurons.

V. Conclusion

This paper introduced a framework for the rapid and automated generation of hardware cores for ANN-based chaotic oscillators, with a specific focus on FPGA implementation. The framework explores the hardware design space and offers a range of candidate hardware solutions that trade off hardware costs and throughput. The paper also proposed novel cost and latency estimation functions. The results demonstrate that the proposed framework not only accelerates the hardware design process but also delivers candidate architectures that outperform previous works in terms of efficiency.

References

[1] S. H. Strogatz, *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[2] Z. Hua, Y. Zhou, and H. Huang, "Cosine-transform-based chaotic system for image encryption," *Information Sciences*, vol. 480, pp. 403–419, 2019.

[3] R. Hamza, Z. Yan, K. Muhammad, P. Bellavista, and F. Titouna, "A privacy-preserving cryptosystem for iot e-healthcare," *Information Sciences*, vol. 527, pp. 493–510, 2020.

[4] S. Kalanadhabhatta, D. Kumar, K. K. Anumandla, S. A. Reddy, and A. Acharyya, "Puf-based secure chaotic random number generator design methodology," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 28, no. 7, pp. 1740–1744, 2020.

[5] M. N. Aslam, A. Belazi, S. Kharbech, M. Talha, and W. Xiang, "Fourth order mca and chaos-based image encryption scheme," *IEEE Access*, vol. 7, pp. 66 395–66 409, 2019.

[6] M. S. Azzaz, C. Tanougast, S. Sadoudi, R. Fellah, and A. Dandache, "A new auto-switched chaotic system and its fpga implementation," *Communications in Nonlinear Science and Numerical Simulation*, vol. 18, no. 7, pp. 1792–1804, 2013.

[7] K. Rajagopal, M. Tuna, A. Karthikeyan, İ. Koyuncu, P. Duraisamy, and A. Akgul, "Dynamical analysis, sliding mode synchronization of a fractional-order memristor hopfield neural network with parameter uncertainties and its non-fractional-order fpga implementation," *European Physical Journal Special Topics*, vol. 228, pp. 2065–2080, 2019.

[8] M. Alcin, "The runge kutta-4 based 4d hyperchaotic system design for secure communication applications," *Chaos Theory and Applications*, vol. 2, no. 1, pp. 23–30, 2020.

[9] M. Tuna, M. Alçın, İ. Koyuncu, C. B. Fidan, and İ. Pehlivan, "High speed fpga-based chaotic oscillator design," *Microprocessors and Microsystems*, vol. 66, pp. 72–80, 2019.

[10] L. Zhang, "Artificial neural network model design and topology analysis for fpga implementation of lorenz chaotic generator," in *Canadian Conf. on Electrical and Computer Engineering*. IEEE, 2017, pp. 1–4.

[11] M. Alçın, İ. Pehlivan, and İ. Koyuncu, "Hardware design and implementation of a novel ann-based chaotic generator in fpga," *Optik*, vol. 127, no. 13, pp. 5500–5505, 2016.

[12] J. Sunny, J. Schmitz, and L. Zhang, "Artificial neural network modelling of rossler's and chua's chaotic systems," in *IEEE Canadian Conference on Electrical & Computer Engineering*. IEEE, 2018, pp. 1–4.

[13] W. A. Al-Musawi, W. A. Wali, and M. A. A. Al-Ibadi, "New artificial neural network design for chua chaotic system prediction using fpga hardware co-simulation," *International Journal of Electrical and Computer Engineering*, vol. 12, no. 2, p. 1955, 2022.

[14] F. Yu, Z. Zhang, H. Shen, Y. Huang, S. Cai, J. Jin, and S. Du, "Design and fpga implementation of a pseudo-random number generator based on a hopfield neural network under electromagnetic radiation," *Frontiers in Physics*, vol. 9, 2021.

[15] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert *et al.*, *A statistical test suite for random and pseudorandom number generators for cryptographic applications*. US Department of Commerce, Technology Administration, National Institute of ..., 2001, vol. 22.