http://doi.org/10.35784/iapgos.7583

received: 16.04.2025 | revised: 25.12.2025 | accepted: 04.01.2026 | available online: 30.03.2026

USING FPGA FOR MODELLING AND GENERATING CHAOTIC PROCESSES

Oleksandr V. Osadchuk, Iaroslav O. Osadchuk, Valentyn K. Skoshchuk
Vinnytsia National Technical University, Vinnytsia, Ukraine

Abstract. *This work presents a comprehensive approach to the implementation of two chaotic dynamic systems – Nose–Hoover and Rikitake – on an FPGA platform. Initially, the systems were modeled in a Python environment using the Euler method, which allowed for the verification of chaotic behavior and an assessment of the impact of the integration step on stability. A comparison of time series and phase portraits confirmed the persistent chaotic nature of both systems within a defined parameter range. Next, a Verilog-based implementation using fixed-point arithmetic (Q16.16) was developed and tested in ModelSim. Simulation results showed a close match with the Python models, indicating a correct choice of bit width and Euler method settings. For real-time data transmission, UART modules and intermediate converters were used to scale Q16.16 outputs to an 8-bit format. This approach enables tracking and recording of computations on a personal computer while maintaining real-time chaotic dynamics. The experimental results confirm that the Nose–Hoover and Rikitake systems can be successfully implemented on an FPGA with limited bit width without significant loss of chaotic properties. This approach has potential applications in cryptography, secure communication, and high-performance pseudorandom signal generators, where flexibility, energy efficiency, and real-time processing are essential.*

Keywords: FPGA, Rikitake, Nose–Hoover, Verilog, forward Euler

WYKORZYSTANIE FPGA DO MODELOWANIA I GENEROWANIA PROCESÓW CHAOTYCZNYCH

Streszczenie. *W niniejszej pracy przedstawiono kompleksowe podejście do implementacji dwóch chaotycznych układów dynamicznych – Nose–Hoover i Rikitake – na platformie FPGA. Początkowo układy te zostały zamodelowane w środowisku Python przy użyciu metody Eulera, co pozwoliło na weryfikację zachowań chaotycznych oraz ocenę wpływu kroku integracji na stabilność. Porównanie szeregów czasowych i wykresów fazowych potwierdziło trwały charakter chaotyczny obu układów w określonym zakresie parametrów. Następnie opracowano i przetestowano w programie ModelSim implementację opartą na języku Verilog z wykorzystaniem arytmetyki stałoprzecinkowej (Q16.16). Wyniki symulacji wykazały ścisłą zgodność z modelami w języku Python, co wskazuje na prawidłowy wybór rozdzielczości bitowej i ustawień metody Eulera. W celu transmisji danych w czasie rzeczywistym zastosowano moduły UART oraz konwertery pośrednie w celu skalowania wyników Q16.16 do formatu 8-bitowego. Podejście to umożliwia śledzenie i rejestrowanie obliczeń na komputerze osobistym przy zachowaniu chaotycznej dynamiki w czasie rzeczywistym. Wyniki eksperymentalne potwierdzają, że systemy Nose–Hoover i Rikitake można z powodzeniem zaimplementować na układzie FPGA o ograniczonej liczbie bitów bez znaczącej utraty właściwości chaotycznych. Podejście to ma potencjalne zastosowania w kryptografii, bezpiecznej komunikacji oraz wysokowydajnych generatorach sygnałów pseudolosowych, gdzie elastyczność, efektywność energetyczna i przetwarzanie w czasie rzeczywistym mają zasadnicze znaczenie.*

Słowa kluczowe: FPGA, Rikitake, Nosé–Hoover, Verilog, metoda Eulera w przód

Introduction

Chaotic dynamic systems are attracting increasing attention from scientists and practitioners as they can model complex physical, biological, and technical processes that do not fit within the framework of linear paradigms. In cryptography and information security systems, in particular, they enable the generation of high-quality pseudorandom sequences that are resistant to attacks due to their extreme sensitivity to initial conditions. In computational experiments and machine learning, chaotic systems help analyze multifactor interactions and uncover hidden patterns. At the same time, the application of such systems in real-time devices opens up new possibilities for building more flexible sensor networks, reliable control systems, and intelligent devices with elements of unpredictability.

Despite these advantages, the practical implementation of chaotic systems presents significant challenges. First, nonlinear equations often require high computational precision; otherwise, the characteristic sensitivity of the system may be lost, or artificial periodicity may emerge. Second, computational loads can be substantial, especially when there is a need to process large volumes of real-time data or parallelize computations across multiple sensor channels. Additionally, verification tools and efficient data transmission channels are required for seamless integration with other systems or applications.

Field Programmable Gate Array (FPGA) platforms [10] offer unique advantages in this context. Their parallel architecture accelerates computations compared to traditional processors, while their hardware-level configurability allows fine-tuning of resources to match the specific characteristics of chaotic equations and the required precision. Moreover, modern FPGAs can easily integrate with various peripheral interfaces, simplifying deployment in industrial, scientific, and consumer applications. These features make FPGAs an attractive platform for researching and implementing chaotic dynamic systems, where the combined requirements of performance, flexibility, and scalability are critical.

This paper analyzes and implements two dynamic systems Nosé-Hoover and Rikitake on an FPGA platform. Since these systems have received significantly less attention in the literature compared to classical models such as Lorenz or Rössler, special focus is given to the selection of numerical formats, integration methods, and the preservation of chaotic properties in fixed-point arithmetic. Before hardware implementation, the system will first be validated in a Python environment, after which the results will be transmitted from the FPGA to a personal computer for further analysis. This comprehensive approach will assess the feasibility of FPGA-based signal generation for Nosé-Hoover and Rikitake systems and provide recommendations for designing other, less-studied chaotic systems in digital environments.

1. Analysis of recent research and publications

Over the past decade, research on chaotic dynamic systems implemented on FPGA (Field Programmable Gate Array) platforms has undergone significant advancements, primarily driven by the demand for high-performance pseudorandom sequence generators and real-time applications. Most researchers focus on classical models such as the Lorenz, Rössler, or Chua systems, which are well-studied in the context of cryptography, digital watermarking, and wireless security systems. For instance, L. Zhang [11] concentrated on the Lorenz system, proposing a hardware implementation with both 32-bit and 16-bit fixed-point formats using Xilinx System Generator and analysing how numerical precision affects design performance and the preservation of chaotic behaviour. The results confirmed that even with a 16-bit configuration, it is possible to achieve stable chaotic dynamics suitable for secure real-time data transmission.

R. D. Méndez-Ramírez et al. [4] systematically studied the degradation of multiple chaotic systems, including the Rössler system, in digital implementations. The authors compared several integration methods (Euler, Heun, and Runge-Kutta) using fixed-point arithmetic on both embedded microcontrollers and an FPGA, determining the conditions under which chaotic behaviour is preserved while minimizing hardware costs. These results have proven useful for other researchers working on cryptographic applications since chaotic signals can serve as the basis for generating pseudorandom numbers or dynamic encryption keys.

The practical aspect of hardware implementation has been emphasized in studies by T. Bonny et al. [1] and A. A. Rezk et al. [8], where the development of chaotic signal and pseudorandom sequence generators for communication security systems is described. Specifically, T. Bonny et al. highlighted the potential for parallel processing, achieving throughput up to 1882 Mbit/s in cryptographic systems, while A. A. Rezk et al. demonstrated the high efficiency of chaotic PRNGs for NIST (National Institute of Standards and Technology, USA) tests. More recently, F. N. Kemdoum et al. [5] presented an FPGA-based implementation of a perturbed Chen oscillator for pseudo-random number generation, demonstrating that the generated sequences successfully pass both the NIST SP800-22 and TestU01 test suites, further confirming the viability of chaotic systems for cryptographic applications on FPGA platforms.

Despite the advancements in studying these models, the Nosé-Hoover and Rikitake systems remain underexplored in a hardware context. At a theoretical level, Nosé [6] and Hoover [2] developed an approach for modelling canonical ensembles, which exhibited distinct chaotic dynamics dependent on temperature parameters and a virtual "thermostat". Further studies (e.g., H. A. Posch, W. G. Hoover, and F. J. Vesely [7]) confirmed that the Nosé-Hoover system can exhibit a broad spectrum of behaviors, ranging from regular to fully chaotic, making it particularly interesting for applications requiring fine-tuned chaos control. The Rikitake system, initially proposed by Rikitake [9] to describe geomagnetic dynamo effects, also demonstrates complex nonlinear behaviour, which could be beneficial in theoretical and practical applications related to instability and multimode processes [3]. However, there is a near-total absence of works in the open literature where the Nosé-Hoover or Rikitake systems have been implemented specifically on FPGA.

Given this, the problem arises of porting these specific systems to a digital environment with limited bit-width while preserving their characteristic chaotic properties. Methods applied to other models (such as different integration schemes or variable normalization techniques) do not always directly apply to Nosé-Hoover or Rikitake, as these equations may include additional nonlinear terms, exhibit more complex attractors, or require different numerical precision settings. Additionally, the lack of FPGA-tested solutions for these systems complicates comparisons with classical models like Lorenz or Rössler.

The value of this study lies in proposing a comprehensive approach to implementing the Nosé-Hoover and Rikitake systems on FPGA. This approach allows for:

1. Verifying whether these systems retain their chaotic properties in fixed-point arithmetic.
2. Assessing hardware costs and performance efficiency, aiding in selecting the optimal configuration for specific applications.
3. Demonstrating seamless integration with other modules or external devices, which is crucial for real-world applications in monitoring, control, and security.

Thus, this work not only expands current knowledge on the hardware implementation of chaotic dynamic systems but also lays the groundwork for developing new applied solutions that require powerful and flexible chaotic signal generators. In the future, such FPGA-based solutions may find applications in cryptography, secure communications, scientific experiments, and industrial systems incorporating adaptive control elements.

2. Formulation of the problem

The aim of this work is to develop an approach for modelling and hardware implementation of dynamic systems using two models – Nosé-Hoover and Rikitake - as examples. A key feature of this approach is the combination of software validation (in Python) with subsequent implementation in Verilog [12] and integration into an FPGA. This will enable the creation of a flexible and scalable solution for generating and analyzing chaotic signals in real-time.

To achieve this goal, the following key tasks are outlined:

1. Algorithmic Foundations. Develop or adapt an integration method for the correct modeling of Nosé-Hoover and Rikitake dynamic equations. Ensure that this method can be implemented both in Python (for quick validation) and in Verilog (for hardware implementation).
2. Modeling in Python. Verify the correctness and stability of the selected algorithm by generating plots and analyzing trajectories in Python scripts. Ensure that the system retains its chaotic properties and does not transition into an artificially periodic mode due to computational inaccuracies.
3. Implementation in Verilog and Testing. Create two separate modules (for Nosé-Hoover and Rikitake) in Verilog, incorporating a fixed-point arithmetic mechanism and optimizing bit-width selection. Perform testing in ModelSim, compare the results with the Python model, and visualize them as plots for detailed analysis.
4. Integration into FPGA. Assemble a circuit that transmits computation results via UART to a personal computer. On the PC side, implement real-time visualization and data processing to confirm the system's functionality and its suitability for practical applications.

The proposed approach aims not only to replicate the chaotic behavior of the Nosé-Hoover and Rikitake systems but also to demonstrate the versatility of the chosen methodology. Implementing hardware acceleration on an FPGA with fixed-point arithmetic will contribute to efficient computational resource utilization, while parallel processing will enable high-speed operation and scalability for a broader range of applications – from cryptography to experimental research in nonlinear dynamics.

3. Algorithmic foundations

When modeling dynamic systems in digital devices, the choice of an integration algorithm is critically important. Our work is based on the Forward Euler method, one of the simplest and most intuitive approaches for numerically solving differential equations. Although Euler's method is not the most accurate, it has several advantages: it is easy to implement, requires minimal computations, and runs efficiently even in resource-constrained environments. The essence of this method is that at each iteration step, the current state variable is updated using the formula: $x_{n+1} = x_n + \Delta t \cdot f(x_n, t_n)$ where Δt is the integration step, and $f(x_n, t_n)$ is the right-hand side of the differential equation. For systems with multiple state variables, all equations are computed in a similar manner, but each contains its own function f, which depends on the corresponding variables and parameters.

The Euler method belongs to first-order accuracy schemes, meaning that the local error at each step is proportional to $\theta(\Delta t^2)$, while the global error accumulates as $\theta(\Delta t)$. In complex or nonlinear processes, particularly in chaotic systems, this implies that achieving acceptable accuracy often requires a sufficiently small step Δt. However, if Δt is too small, it significantly increases the total number of iterations, which can be problematic in real-time applications or on devices with limited computational resources. Another critical aspect is stability: if Δt exceeds a certain threshold, Euler's method tends to produce a "diverging" solution, even in cases where the actual system dynamics should remain bounded. This is especially crucial for chaotic systems, as even the slightest change in Δt can drastically affect the quality of attractor reproduction or lead to pseudo-periodic behavior.

As an illustration, consider the simple differential equation: $\frac{dx}{dt} = ax$ with the initial condition $x(0) = x_0$. The analytical solution is: $x(t) = x_0 e^{at}$. Applying the Euler method, at each step n, we obtain: $x_{n+1} = x_n + \Delta t a x_n$, which simplifies to: $x_{n+1} = x_n(1 + a\Delta t)$. For sufficiently small Δt, this approximation accurately reproduces exponential growth or decay. However, if the integration step Δt is increased, the error rapidly accumulates, leading to significant deviations from the analytical curve or even pseudo-stability. Therefore, for Forward Euler to work correctly in chaotic systems, careful selection of Δt and error accumulation control is required.

In our study, Forward Euler is used as the baseline method, considering its simplicity and suitability for hardware implementation. In the next section, we will explore how this integration algorithm can be effectively combined with numerical verification techniques to ensure the correctness of results, as well as how to choose the integration step Δt and fixed-point precision to preserve the chaotic properties of the system.

4. Modelling in the Python environment

This section examines the preliminary (software) modeling of two dynamic systems: Nosé-Hoover and Rikitake. This approach allows for a quick verification of the correctness of the integration algorithm and helps identify key dynamic characteristics before transitioning to hardware implementation.

The Nosé–Hoover system is described by the following equations.

$$\begin{cases} \dot{x} = y \\ \dot{y} = yz - x \\ \dot{z} = 1 - y^2 \end{cases} \tag{1}$$

Here, x, y, z are the state variables, while the right-hand side represents the thermostating mechanism, which was originally used by Nosé and Hoover for modeling canonical ensembles in statistical physics.

The following code (Fig. 1) implements the Nosé-Hoover system using the Forward Euler method. For clarity, a minimal example is provided, which can be easily adapted to different values of the integration step Δt and simulation time T.

```
def solve_nose_hoover(x0=0.0, y0=0.0, z0=0.0,
                      dt=0.001, T=5.0):
    n_steps = int(T / dt)
    # Arrays to store solutions
    x = np.zeros(n_steps + 1)
    y = np.zeros(n_steps + 1)
    z = np.zeros(n_steps + 1)
    t = np.linspace(0, T, n_steps + 1)
    # Initialize with initial conditions
    x[0], y[0], z[0] = x0, y0, z0
    # Forward Euler method
    for i in range(n_steps):
        dx = y[i]
        dy = y[i] * z[i] - x[i]
        dz = 1 - y[i]**2
        x[i + 1] = x[i] + dt * dx
        y[i + 1] = y[i] + dt * dy
        z[i + 1] = z[i] + dt * dz
    return t, x, y, z

if __name__ == "__main__":
    # Example usage:
    dt = 0.01
    T  = 300.0
    x0 = 2.0
    y0 = -2.0
    z0 = 0.5
    # Solve the system
    t, x_sol, y_sol, z_sol = solve_nose_hoover(x0,
y0, z0, dt, T)
```

Fig. 1. Implementation of the Nosé–Hoover system using the Euler method in Python

The graphs (Fig. 2) show non-periodic oscillations of $x(t)$, $y(t)$, $z(t)$ within a bounded range. This indicates a stable chaotic regime: the system does not "diverge", yet it does not settle into a fixed periodic pattern. Small changes in parameters or initial conditions lead to significant differences in trajectories, confirming the sensitivity of the Nosé–Hoover system.

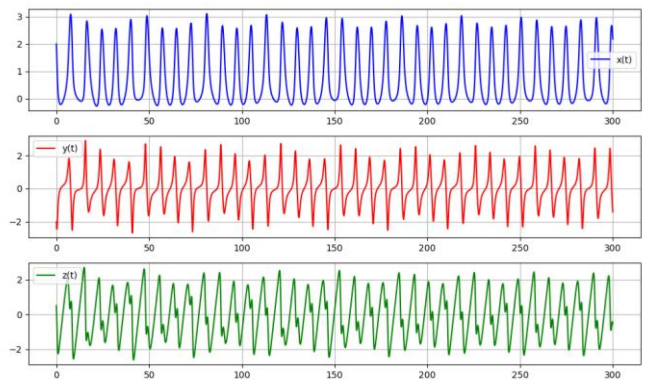


Fig. 2. Simulation results of the Nosé–Hoover system in Python

The Rikitake system is modeled by the equations shown in equation (2).

$$\begin{cases} \dot{x} = -\mu x + yz \\ \dot{y} = -\mu y + x(z - a) \\ \dot{z} = 1 - xy \end{cases} \tag{2}$$

where, μ and a are parameters that determine the nature of chaotic oscillations. Historically, this system was used to explain the geomagnetic dynamo effect, but it is also interesting as a generalized example of nonlinear dynamics.

A similar approach can be used for numerical integration, as shown in Figure 3.

```
def solve_rikitake(mu=0.5, a=0.1,
                        x0=0.0, y0=0.0, z0=0.0,
                        dt=0.001, T=5.0):
    n_steps = int(T / dt)
    # Arrays to store the solutions
    x = np.zeros(n_steps + 1)
    y = np.zeros(n_steps + 1)
    z = np.zeros(n_steps + 1)
    t = np.linspace(0, T, n_steps + 1)

    # Initialize with given initial conditions
    x[0], y[0], z[0] = x0, y0, z0

    # Forward Euler integration
    for i in range(n_steps):
        dx = -mu * x[i] + y[i] * z[i]
        dy = -mu * y[i] + x[i] * (z[i] - a)
        dz = 1 - x[i] * y[i]

        x[i + 1] = x[i] + dt * dx
        y[i + 1] = y[i] + dt * dy
        z[i + 1] = z[i] + dt * dz

    return t, x, y, z


if __name__ == "__main__":
    # Example usage
    mu = 0.9
    a = 0.9
    dt = 0.01
    T  = 300.0
    x0 = 1.0
    y0 = -1.0
    z0 = 0.0

    # Solve the system
    t, x_sol, y_sol, z_sol = solve_rikitake (mu,
a, x0, y0, z0, dt, T)
```

Fig. 3. Implementation of the Rikitake system using the Euler method in Python

The variables $x(t)$, $y(t)$, $z(t)$ also oscillate within a bounded range without a noticeable period. This is a characteristic of a chaotic attractor, which does not degenerate into a stable point or a periodic cycle. The system retains its chaotic nature and does not transition into explosive dynamics or damping, confirming a stable yet unpredictable behavior, as shown in Figure 4.

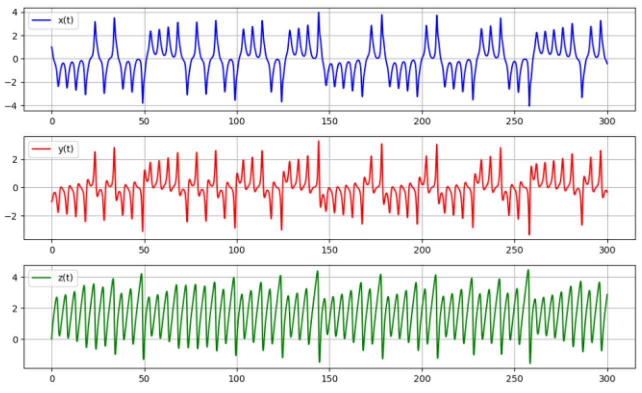


Fig. 4. Simulation Results of the Rikitake system in Python

In conclusion, both systems remain in a stable yet chaotic regime, which implies:

- The system does not diverge – the variables stay within finite amplitude limits.
- No regular period is observed – the curves do not repeat at fixed time intervals.
- The oscillations appear "noise-like" – typical for chaotic processes, where small changes in initial conditions can significantly affect the trajectory.

5. Implementation in Verilog and testing

This section describes the approach to implementing dynamic systems in Verilog using fixed-point arithmetic (Q16.16). After writing the code, both modules (Nosé-Hoover and Rikitake) are simulated in ModelSim. To verify the correctness of the trajectories, the simulation results are saved to files and visualized as graphs, enabling comparison with the Python models. Below is an example of a Verilog module (Fig. 5) that implements the Nosé-Hoover system using the Forward Euler method. The Q16.16 format is used for the state variables x, y, z, system parameters, and the integration step Δt.

```
module nose_hoover_system (
    input wire clk,
    input wire rst,
    output reg signed [31:0] x, // Q16.16
    output reg signed [31:0] y, // Q16.16
    output reg signed [31:0] z  // Q16.16
);
    // Integration step (for example, 0.01 in
Q16.16)
    parameter signed [31:0] dt = 32'h0000028f;  //
0.01
    // 64-bit temporary registers for
multiplications
    reg signed [63:0] mul_temp1, mul_temp2,
mul_temp3, mul_temp4, mul_temp5, mul_temp6,
mul_temp7;
    // Next-state registers (Q16.16)
    reg signed [31:0] x_next, y_next, z_next;
    always @(posedge clk or posedge rst) begin
            x <= x_next;
            y <= y_next;
            z <= z_next;
    end
    // Combinational logic to compute the next
state
    always @(*) begin
        mul_temp1 = $signed(y) * $signed(dt); //
64-bit multiplication
        x_next    = x + (mul_temp1 >>> 16);
        mul_temp4 = mul_temp3 * $signed(dt);
        y_next    = y + (mul_temp4 >>> 16);
        mul_temp6 = (1 <<< 16) - (mul_temp5 >>>
16);
        mul_temp7 = mul_temp6 * $signed(dt);
        z_next    = z + (mul_temp7 >>> 16);
    end
endmodule
```

Fig. 5. Implementation of the Nosé–Hoover system using the Euler method in Python

The code is simulated in ModelSim (Fig. 6) with predefined initial conditions and parameters. The variable values are recorded in a file, after which the results are imported into Python for graph visualization. A typical example of plotting $x(t)$, $y(t)$, $z(t)$ confirms the chaotic behavior and its consistency with the Python model.

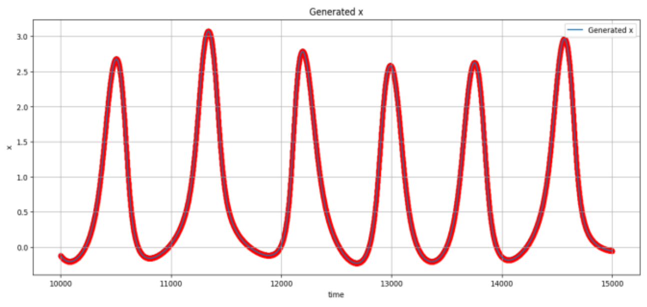


Fig. 6. Simulation results of the Nosé–Hoover system in Verilog

Similarly to the Nosé-Hoover system, the Rikitake system is implemented (Fig. 7). Below is an example of a Verilog module using fixed-point arithmetic:

```
module rikitake_chaos_wave (
    input wire clk,
    input wire rst,
    output reg signed [31:0] x, // Q16.16
    output reg signed [31:0] y, // Q16.16
    output reg signed [31:0] z  // Q16.16
);
    parameter signed [31:0] x0  = 32'h00010000;
    parameter signed [31:0] y0  = 32'hFFFF0000;
    parameter signed [31:0] z0  = 32'h00000000;
    parameter signed [31:0] ONE = 32'h00010000;
    // Intermediate variables for calculations
    reg signed [31:0] dx, dy, dz;               //
Derivatives (Q16.16)
    reg signed [63:0] mul_temp1, mul_temp2;     //
Temporary registers for multiplication
    reg signed [63:0] mul_temp3, mul_temp4;
    reg signed [63:0] dx_mult_dt, dy_mult_dt,
dz_mult_dt;
    reg signed [31:0] x_next, y_next, z_next;
    // Synchronous update (initialization + state
storage)
    always @(posedge clk or posedge rst) begin
        x <= x_next;
        y <= y_next;
        z <= z_next;
    end
    always @(*) begin
        // ========== dx = -mu*x + y*z ==========
//
        mul_temp1 = $signed(mu) * $signed(x);
mul_temp2 = -(mul_temp1 >>> 16);
        mul_temp3 = $signed(y) * $signed(z);
mul_temp4 = (mul_temp3 >>> 16);     // Q16.16
        dx = mul_temp2[31:0] + mul_temp4[31:0];
        dx_mult_dt = $signed(dx) * $signed(dt);
        x_next = x + (dx_mult_dt >>> 16);
        // ========== dy = -mu*y + x*(z - a)
========== //
        mul_temp1 = $signed(mu) * $signed(y);
        mul_temp2 = -(mul_temp1 >>> 16);
        mul_temp3 = $signed(x) * (z - a);
        mul_temp4 = (mul_temp3 >>> 16);
        dy = mul_temp2[31:0] + mul_temp4[31:0];
        dy_mult_dt = $signed(dy) * $signed(dt);
        y_next = y + (dy_mult_dt >>> 16);
        // ========== dz = 1 - x*y ========== //
        mul_temp1 = $signed(x) * $signed(y);
        mul_temp2 = (mul_temp1 >>> 16);
        dz = ONE - mul_temp2[31:0]; // 1 - x*y
        dz_mult_dt = $signed(dz) * $signed(dt);
        z_next = z + (dz_mult_dt >>> 16);
    end
endmodule
```

Fig. 7. Implementation of the Rikitake system using the Euler method in Python

The simulation follows the same principle (Fig. 8):

1. Set initial conditions, parameters (μ and a), and the integration step Δt..
2. Run the module in ModelSim, storing intermediate values x, y, z in a file.
3. Generate graphs to evaluate the system dynamics. The resulting curves confirm the chaotic nature of the Rikitake system for specific values of μ and a.

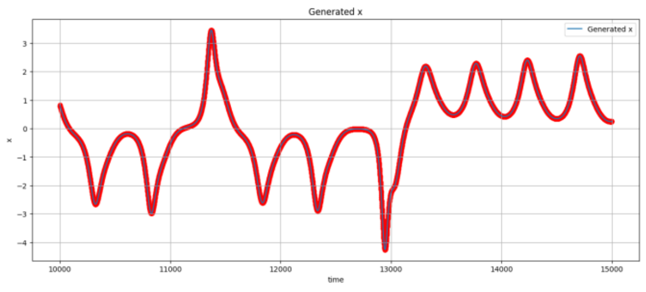


Fig. 8. Simulation results of the Rikitake system in Verilog

Both systems exhibit chaotic behavior similar to the results obtained in Python. The choice of bit-width (Q16.16) and integration step is crucial to preserving the chaotic dynamics. The successful simulation in ModelSim confirms the correct functionality of the Verilog modules and paves the way for further testing directly on FPGA hardware.

5. FPGA Integration

To transition from simulation to a real device, a basic FPGA implementation was designed (Fig. 9).

The schematic includes the following components:

1. Two Chaos Generators. The modules *nose_hoover_system* and *rikitake_chaos_wave* implement the Nosé-Hoover and Rikitake dynamic systems in Q16.16 format. Each generator receives a clock signal and a reset signal (*rst*) and outputs the current values of x, y, z in fixed-point format.
2. Q16.16 to 8-bit Converters. Intermediate components *q16_16_to_8_bits* scale the 32-bit outputs of the chaos generators (Q16.16) to an 8-bit format. This allows transmitting integer values within a limited range, making them suitable for UART communication. Each converter can have predefined minimum and maximum values to define the scaling.
3. UART Converters. Two independent UART blocks transmit data from the FPGA to a PC at a baud rate of 115200. Each UART receives an 8-bit value from the converter and generates a serial bit sequence, which is output via the TX line. On the PC side, the data is received and logged for further visualization.
4. Control and Synchronization. Control signals such as *en* and *start* allow the chaos generators to initiate computation, wait for data readiness, and transmit results via UART. To simplify the architecture, the blocks can operate continuously, transmitting data at a fixed interval.

Thus, the described schematic represents a minimal example of integrating two dynamic systems on an FPGA with real-time data transmission. Further enhancements may include simultaneous output of multiple variables (e.g., x and y), integration of additional chaos generators, or adoption of alternative protocols for result transmission. Below are examples of chaotic signals generated by the Nosé-Hoover system (Fig. 10) and the Rikitake system (Fig. 11) using the FPGA-based implementation.

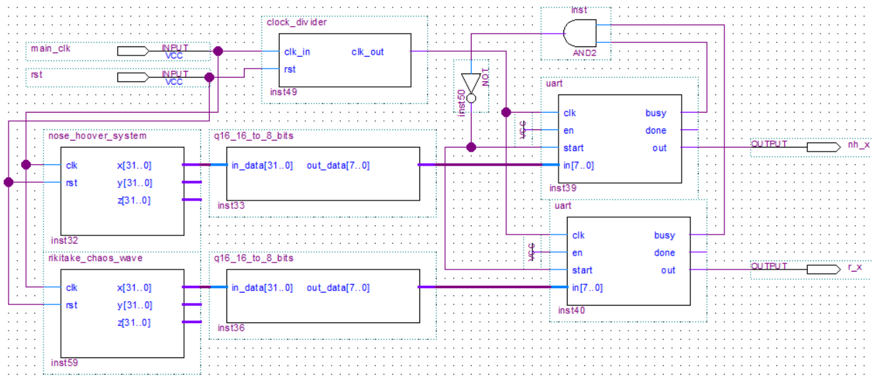
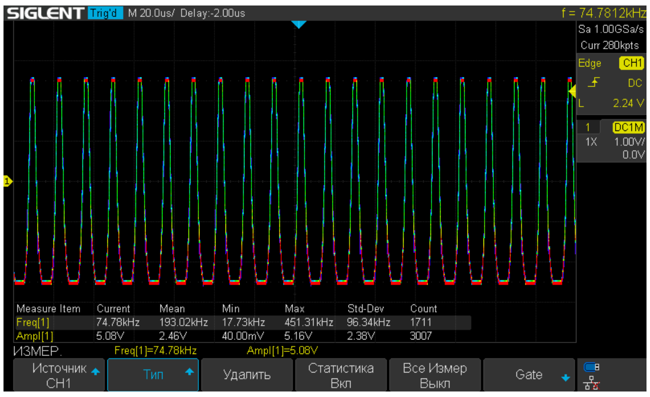
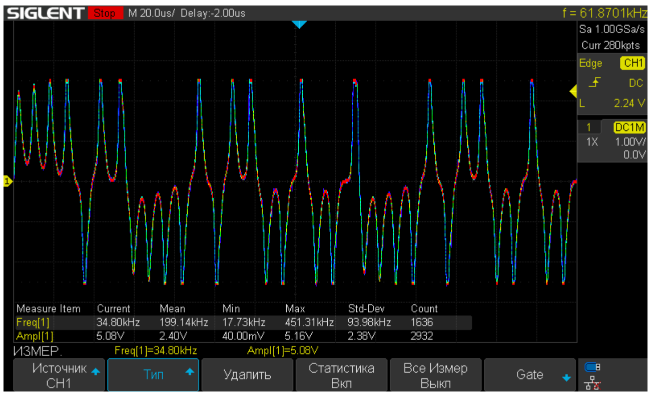


Fig. 9. Schematic of chaotic signal generation for the Nosé–Hoover and Rikitake systems with subsequent transmission via UART



```
Measure Item  Current    Mean        Min        Max         Std-Dev    Count
Freq[1]       74.78kHz   193.02kHz   17.73kHz   451.31kHz   96.34kHz   1711
Ampl[1]       5.08V      2.46V       40.00mV    5.16V       2.38V      3007
ИЗМЕР.        Freq[1]=74.78kHz       Ampl[1]=5.08V
```

Fig. 10. Output of the Nosé–Hoover system on FPGA



```
Measure Item  Current    Mean        Min        Max         Std-Dev    Count
Freq[1]       34.80kHz   199.14kHz   17.73kHz   451.31kHz   93.98kHz   1636
Ampl[1]       5.08V      2.40V       40.00mV    5.16V       2.38V      2932
ИЗМЕР.        Freq[1]=34.80kHz       Ampl[1]=5.08V
```

Fig. 11. Output of the Rikitake system on FPGA

5. Conclusions

In this work, a comprehensive approach to modeling and hardware integration of two chaotic systems – Nose-Hoover and Rikitake – was implemented. Initially, the system dynamics were studied in a Python environment, where the simplicity of the code and clarity of the graphs quickly confirmed their chaotic properties and helped determine optimal integration parameters. Subsequently, both systems were ported to Verilog using fixed-point arithmetic (Q16.16) and verified in ModelSim. The obtained time dependencies matched the results of the Python model, indicating the correctness of the chosen Euler method and the appropriately selected bit-width.

Through FPGA integration and the connection of UART modules, real-time data transmission was successfully established. This opens up possibilities for applying the Nose-Hoover and Rikitake systems in embedded devices, sensor networks, and cryptographic solutions that require high-speed chaotic signal generators. Importantly, key chaotic characteristics (sensitivity to initial conditions, lack of a regular period) were preserved despite the limited bit-width. This result confirms the practical value of the described approach: minimal hardware resources and low latency enable the implementation of chaotic systems in real-time applications across a wide range of fields.

Future research directions may include:

1. Implementing higher-order integration methods (such as Runge-Kutta) to improve accuracy without excessive computational overhead.
2. Analyzing the impact of different parameters and initial conditions on maintaining chaotic behavior in fixed-point arithmetic.
3. Integrating additional interfaces (SPI, Ethernet) for data transmission, allowing the system to be deployed in distributed networks and large-scale projects.

Thus, the primary objective of this study has been achieved: it has been demonstrated that even less-explored systems in hardware environments, such as Nose-Hoover and Rikitake, can be successfully implemented on FPGA while preserving their characteristic chaotic behavior and ensuring reliable and efficient data transmission for further analysis.

References

[1] Bonny, T., Al Debsi, R., Majzoub, S., & Elwakil, A. S. (2019). Hardware Optimized FPGA Implementations of High-Speed True Random Bit Generators Based on Switching-Type Chaotic Oscillators. *Circuits, Systems, and Signal Processing*, *38*(3), 1342–1359. https://doi.org/10.1007/s00034-018-0905-6

[2] Hoover, W. G. (1985). Canonical dynamics: Equilibrium phase-space distributions. *Physical Review A*, *31*(3), 1695–1697. https://doi.org/10.1103/PhysRevA.31.1695

[3] Lorenz, E. N. (1990). Can chaos and intransitivity lead to interannual variability? *Tellus A: Dynamic Meteorology and Oceanography*, *42*(3), 378. https://doi.org/10.3402/tellusa.v42i3.11884

[4] Méndez-Ramírez, R., Arellano-Delgado, A., Murillo-Escobar, M., & Cruz-Hernández, C. (2019). Degradation Analysis of Chaotic Systems and their Digital Implementation in Embedded Systems. *Complexity*, *2019*(1), 9863982. https://doi.org/10.1155/2019/9863982

[5] Nguemo Kemdoum, F., Mboupda Pone, J. R., Bajaj, M., Dzonde Naoussi, S. R., Ayemtsa Kuete, G. P., Louzazni, M., Berhanu Tuka, M., & Kamel, S. (2024). FPGA based implementation of a perturbed Chen oscillator for secure embedded cryptosystems. *Scientific Reports*, *14*(1), 21262. https://doi.org/10.1038/s41598-024-71531-y

[6] Nosé, S. (1984). A molecular dynamics method for simulations in the canonical ensemble. *Molecular Physics*, *52*(2), 255–268. https://doi.org/10.1080/00268978400101201

[7] Posch, H. A., Hoover, W. G., & Vesely, F. J. (1986). Canonical dynamics of the Nosé oscillator: Stability, order, and chaos. *Physical Review A*, *33*(6), 4253–4265. https://doi.org/10.1103/PhysRevA.33.4253

[8] Rezk, A. A., Madian, A. H., Radwan, A. G., & Soliman, A. M. (2019). Reconfigurable chaotic pseudo random number generator based on FPGA. *AEU - International Journal of Electronics and Communications*, *98*, 174–180. https://doi.org/10.1016/j.aeue.2018.10.024

[9] Rikitake, T. (1958). Oscillations of a system of disk dynamos. *Mathematical Proceedings of the Cambridge Philosophical Society*, *54*(1), 89–105. https://doi.org/10.1017/S0305004100033223

[10] Trimberger, S. M. (2015). Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology. *Proceedings of the IEEE*, *103*(3), 318–331. https://doi.org/10.1109/JPROC.2015.2392104

[11] Zhang, L. (2017). System generator model-based FPGA design optimization and hardware co-simulation for Lorenz chaotic generator. *2017 2nd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, 170–174. https://doi.org/10.1109/ACIRS.2017.7986087

[12] *IEEE Standard for Verilog Hardware Description Language*. (2006). IEEE. https://doi.org/10.1109/IEEESTD.2006.99495

Prof. Oleksandr Osadchuk
e-mail: osadchuk.av69@gmail.com

Doctor of Technical Sciences, professor, Head of the Department of Information Radioelectronic Technologies and Systems of Vinnytsia National Technical University, Academician of the Academy Metrology Ukraine.
Author of over 950 publications, including 35 monographs, 18 textbooks, 320 patents for inventions, more than 500 scientific articles in professional journals, of which 75 are in the scientometric databases Scopus and Web of Science.

https://orcid.org/0000-0001-6662-9141

Ph.D. Iaroslav Osadchuk
e-mail: osadchuk.j93@gmail.com

Candidate of Technical Sciences, associate professor of Information Radioelectronic Technologies and Systems of Vinnytsia National Technical University.
Author of more than 250 publications, including 10 monographs, 60 patents for inventions and more than 130 scientific articles in professional journals, of which 32 are in scientometric databases Scopus and Web of Science.

https://orcid.org/0000-0002-5472-0797

M.Sc. Valentyn Skoshchuk
e-mail: skoschuk999@gmail.com

Postgraduate student at the Department of Information Radioelectronic Technologies and Systems of Vinnytsia National Technical University.
Author of 13 publications, including 1 patent for an invention and 5 scientific articles in professional journals.

https://orcid.org/0009-0008-9762-2397