INTEGRATION, the VLSI journal 107 (2026) 102622

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

Design insights for implementing a PRNG with fractional Lorenz system on ESP32 and FPGA

Luis Gerardo de la Fraga [a,*], Esteban Tlelo-Cuautle [b]

[a] Cinvestav, Computer Science Department, Av. IPN 2508, Mexico City, 07360, Mexico
[b] INAOE, Department of Electronics, Luis Enrique Erro No. 1, Tonantzintla, Puebla, 72840, Mexico

ARTICLE INFO

Keywords:
Chaos
Fractional order system
Pseudo-random number generator
Fixed point arithmetic
FPGA
32-bit microprocessor

ABSTRACT

A Pseudo Random Number Generator (PRNG) produces a sequence whose randomness is evaluated by statistical tests like NIST and TestU01. The random sequences are deterministic and reproducible when using the same seed value. In this manner, and for cryptographic applications, the key size of a PRNG must be increased to resist brute force attacks. Henceforth, a fractional-order chaotic system, like the Lorenz one, is suitable to be used to design a PRNG, which implementation can be performed by using embedded devices such as the low-cost ESP32 (32-bit LX6 microprocessor) and field-programmable gate array (FPGA). To increase the throughput, the fractional Lorenz system is integrated with an approximated two steps Runge–Kutta method. An analysis in performed to find the domain of attraction for each state variable, and to verify that the PRNG produces non-correlated sequences. The hardware implementation is detailed by establishing the number of bits (or keys) required for the PRNG to guarantee its suitability for cryptographic applications. Finally, the hardware design of a PRNG using the fractional Lorenz system provides a throughput of 4.99 Mbits/s in the ESP32 platform, and 112.96 Mbits/s in the FPGA.

1. Introduction

In [1] was proposed the modified two-stage fractional Runge–Kutta (M2sFRK) method to integrate fractional order systems. This integration method has only two single steps, therefore is very efficient and computationally fast. In [2], the three parameters of the fractional Lorenz system (FLS) were optimized to maximize the maximum Lyapunov exponent and the Kaplan-York dimension of the FLS. The obtained optimized FLS has a behavior more like a map, with big step jumps in its phase space portraits. Because the use of the M2sFRK, the optimization with NSGA-II algorithm was relative fast, taking only a few seconds.

A different result of the optimized FLS showed in [2] is used here to build a Pseudo-Random Number Generator (PRNG). It is shown here in this work how to build a PRNG in software in a low-cost and energy-efficient 32-bit microprocessor, called ESP32, and also how to build the same PRNG in hardware within a FPGA.

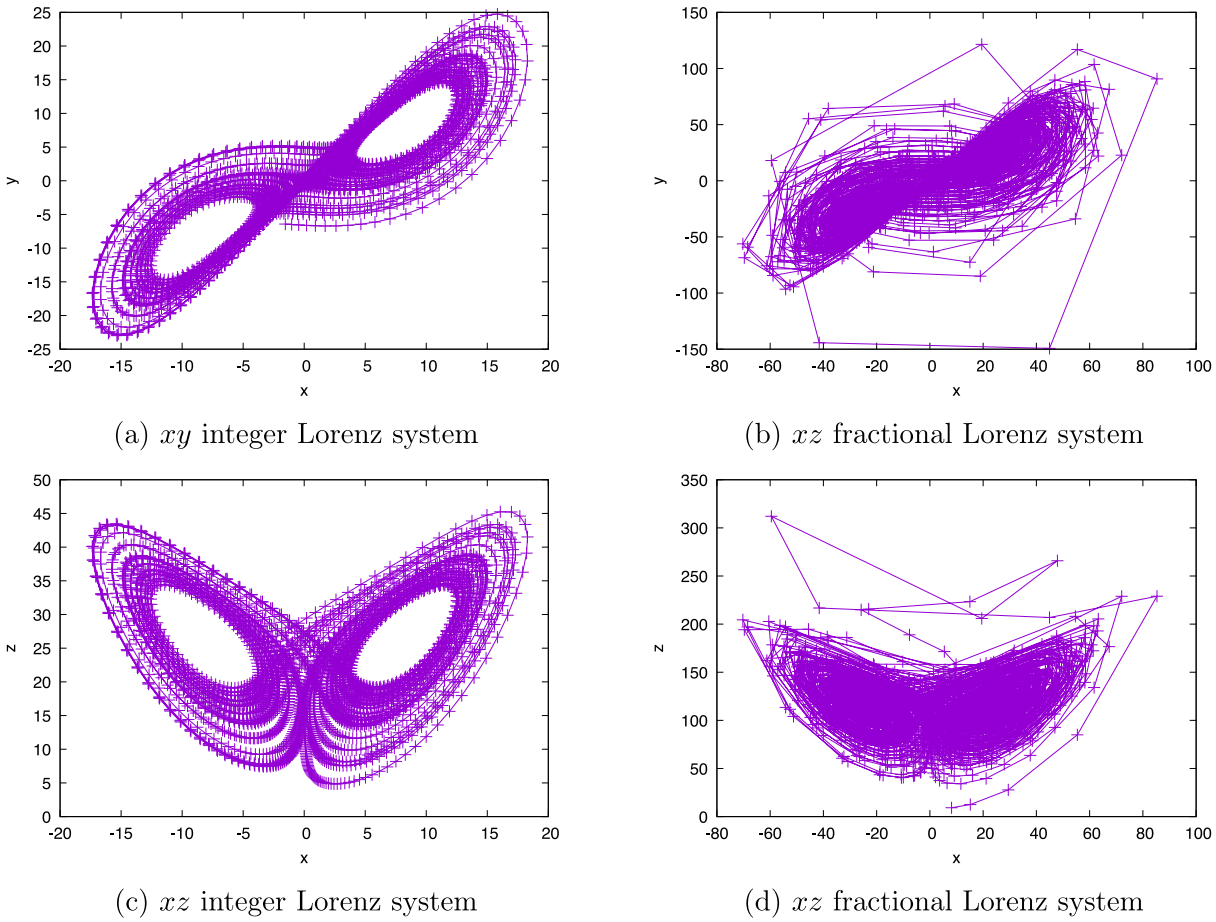
In [3], authors implement two fractional order chaotic systems, a Liu and a V-shape multi-scroll chaotic systems. They use Grünwald–Letnikov definition for the fractional differential. The implementation in FPGA was made to show the acceleration of all the calculations in hardware. Also in [4], authors use the Grünwald–Letnikov method to implement a delayed fractional system in FPGA, in this work is not given an application of their implementation. In the book [5], authors shows the implementation in FPGA of fractional chaotic system using the Grünwald–Letnikov and of Adams–Bashforth-Moulton method, and as application they show the synchronization of these systems. In [6], authors present an implementation in FPGA of a fractional memristive chaotic system, and this system was used to implement a PRNG obtaining a throughput of 0.396 Gbit/s. In [7], authors implement in FPGA several fractional chaotic system based in product-integration rules.

In an excellent review [8], authors analyze the mathematical details and computational complexity to implement fractional order chaotic systems with the Adams–Bashforth-Moulton, Grünwald–Letnikov, and Semi-analytical Adomian Decomposition methods. In this review authors do not show applications with the analyzed systems. In the future challenges identified in this review, authors mention as a open problem how to estimate the optimal bit precision for a proper representation of the chaotic signals. We are proposing here in this work a solution to solve this open problem, according to the authors in [8].

This paper is organized as follows. In Section 2, the used integration method, the M2sFRK is described. In Section 3, the implementation of the designed PRNG in the EPS32 platform is detailed. In Section 4, the PRNG implementation in a FPGA is also detailed. In Section 5,

* Corresponding author.
E-mail address: fraga@cs.cinvestav.mx (L.G. de la Fraga).

https://doi.org/10.1016/j.vlsi.2025.102622
Received 29 October 2025; Received in revised form 16 November 2025; Accepted 28 November 2025
Available online 29 November 2025



(a) xy integer Lorenz system (b) xz fractional Lorenz system

(c) xz integer Lorenz system (d) xz fractional Lorenz system

Fig. 1. Phase space portraits of the integer Lorenz system in (a) and (c), and the fractional Lorenz system in (b) and (d).

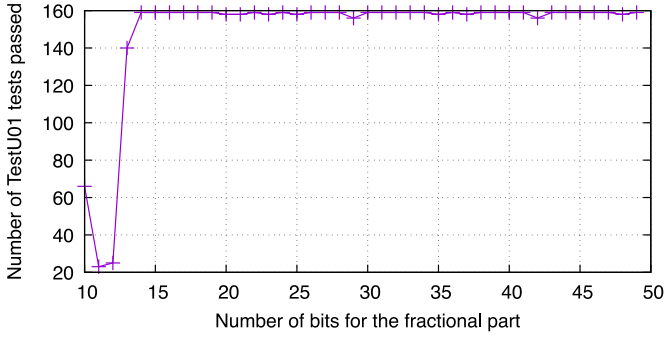


Fig. 2. The number of bits for the fractional part vs. the number of TestU01 statistical tests passed with 10^8 random bits.

a discussion about how could be the way to select the number of bits in the fractional part for the PRNG implementation in fixed-point arithmetic is given. Finally, in Section 6 some conclusions of this work are drawn.

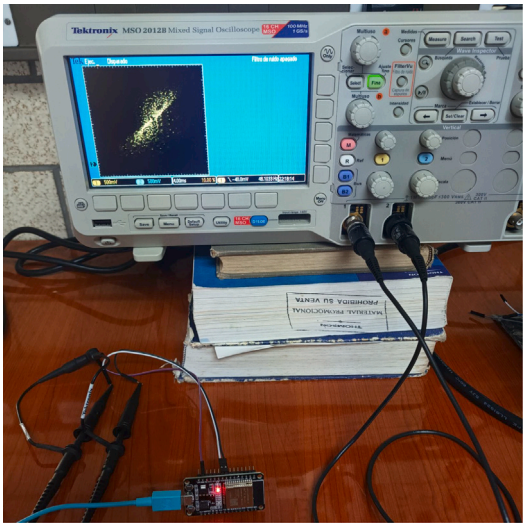


Fig. 3. The fractional Lorenz oscillator working in the STM32.

2. Fractional integration method

The modified two-stage fractional Runge–Kutta (M2sFRK) method was proposed in [1] to integrate fractional order systems. For a fractional function f that does not depend on time, this method as it is described in [1] is

$$\mathbf{v}_{k+1} = \mathbf{v}_k + K_2, \text{ for } k = 0, 1, \ldots M-1, \tag{1}$$

where

$$\begin{aligned} K_1 &= \frac{h^\alpha}{\Gamma(\alpha+1)} f(v_k), \\ K_2 &= \frac{h^\alpha}{\Gamma(\alpha+1)} f(v_k + a_{21} K_1), \\ a_{21} &= \frac{\Gamma(\alpha+1)^2}{\Gamma(2\alpha+1)}, \end{aligned} \tag{2}$$

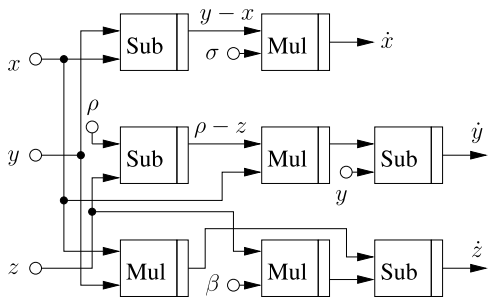


Fig. 4. The Lorenz module. All paths are of size 29 bits. This is the direct implementation of Eq. (4).

h is the integration step, and α is the fractional value of the derivative.

In the calculation of the term $a_{21}K_1$ in (2), is possible to observe that there is only one constant

$$a_{21}K_1 = \frac{\Gamma(\alpha+1)^2}{\Gamma(2\alpha+1)}\frac{h^\alpha}{\Gamma(\alpha+1)}f(v_k), \\ = c_4 f(v_k).$$

Then the M2sFRK method can be arranged as

$$\begin{aligned} K_3 &= \mathbf{v}_k + c_4 f(\mathbf{v}_k), \\ \mathbf{v}_{k+1} &= \mathbf{v}_k + c_2 f(K_3), \end{aligned} \tag{3}$$

where $c_2 = \frac{h^\alpha}{\Gamma(\alpha+1)}$.

The M2sFRK method was used to integrate and optimize a FLS in [2]. The FLS is described by the equation

$$\begin{aligned} D^\alpha x &= \sigma(y-x), \\ D^\alpha y &= x(\rho - z) - y, \\ D^\alpha z &= xy - \beta z, \end{aligned} \tag{4}$$

where x, y, and z are the three state variables, α represents the fractional derivative, and in this case $\alpha < 1$. Common values for the constant parameters (σ, ρ, β) are $(10, 28, 8/3)$ used in the simulations of the integer order Lorenz system.

In [2] the fractional Lorenz system was optimized to maximize the positive maximum Lyapunov exponent (MLE) and its Kaplan-York dimension. One of the obtained fractional Lorenz system with [σ = 29.8022, ρ = 118.9523, β = 6.0606] with MLE = 3.661470, and KYD = 2.048009 is shown in Fig. 1(b) and (d). This fractional system was integrated using the M2sFRK method with $h = 0.01$, and $\alpha = 0.9$. For the integer system was used $h = 0.001$. The initial condition to generate graphs in Fig. 1 is $[x_0, y_0, z_0] = [5, 7, 9]$. 3000 points of both systems are draw in Fig. 1, the samples for the integer system were obtained every 10 integration steps. The optimized fractional Lorenz system in Fig. 1(b) shows a behavior more like a chaotic map, with big jumps in the space portraits, which is the expected behavior because the MLE was maximized. Also the dynamic range of the variables are expanded in the optimized oscillator, e.g. now the dynamic range in x is $[-80, 100]$ and the original dynamic range was $[-20, 20]$.

3. PRNG implementation in the STM32 microprocessor

To implement the fractional Lorenz system with fixed point arithmetic, the number of bits used in the integer part and in the fractional part must be selected. For the integer part, we can see that the ranges of variables in the phase space portraits in Fig. 1 are: $x \in [-80, 100]$, $y \in [-150, 150]$, and $z \in [0, 350]$; also the used values for the constants in Eq. (4) are $\sigma = 29.8022$, $\rho = 118.9523$, and $\beta = 6.0606$. With all these values, the maximum values that the derivatives of variables in Eq. (4) can have are

$$\dot{x} = 6(150 + 80) = 1380,$$

$$\dot{y} = 100(119 - 0) + 150 = 12050,$$

$$\dot{z} = 100(150) - 119(0) = 15000.$$

Thus, the maximum value is 15000, which needs $\lfloor \log_2(15000) + 1 \rfloor = 14$ bits. Then 14 bits are needed for the integer part.

For the selection of the number of bits for fractional part, the analysis is more complicated. A PRNG was implemented using the calculated 14 bits for the integer part, and the bits for the fractional part were selected from 10 to 49 bits. 10^8 random bits were generated and tested with the TestU01 statistical tests. These test were used because is easy to implement scripts to automate the testing process. The graph in Fig. 2 show the result, then the minimum number of bits that can be used for the fractional part is equal to 14, where all the TestU01 tests passed.

In chaotic applications there not exist a mathematical proof that the generated sequences are random. Instead, statistical tests are used to prove if a very long sequence is pseudo random or not, such as the NIST and TestU01 tests.

The TestU01 statistical suite is described in the WEB page http://simul.iro.umontreal.ca/testu01/tu01.html. This is a library in C programming language. The random sequences of bits are stored in single binary file. The tests that can read such file are Rabbit, Alphabit, and BlockAlphabit. With 100 sequences of 10^6 bits, 159 independent tests are applied, Rabbit and Alphabit apply 38 and 17 different statistical tests, respectively, BlockAlphabit applies the Alphabit battery of tests 6 times repeatedly to the binary file after reordering the bits by blocks of sizes 1, 2, 4, 8, 16, and 32 bits [9].

The PRNG with numbers 1:14:14 was finally implemented in the microcontroller ESP32. The measured throughput was 4.99 Mbits/s. In Fig. 3 is shown the fractional oscillator on the oscilloscope screen.

4. PRNG implementation in FPGA

The PRNG was coded also in Verilog and implemented in the FPGA *Alchitry Cu*. This FPGA allow us to use open software for the implementation including apio, version 0.9.5, with Icarus Verilog (verilog compiler) version 13.0, yosys (Yosys Open SYnthesis Suite), and nextpnr-ice40 (Next Generation Place and Route, version 0.8-44).

Fig. 4 shows a diagram of a direct implementation of Eq. (4) when $\alpha = 1$, this is when it calculates the integer order. The implementation uses four multipliers and four substractors. After each operation, multiplication or substraction, a register is used to synchronize the calculations with the clock. E.g. the calculation of $(\dot{x})$ is $\sigma(y - x)$, then first the value of x is subtracted to y and second the result is multiplied by σ, just as it is shown in the top of Fig. 4. The calculation of $(\dot{y})$ of $(\dot{z})$ takes three clock cycles.

The M2sFRK method make two calls to the Lorenz equation with $\alpha = 1$, this is the f function in Eq. (3). The PRNG modules is show in Fig. 5. The output of the Lorenz system is multiplied with c_2 or c_4 constants, the result is stored in a register. This result is added to (x_i, y_i, z_i) and produces the (x_2, y_2, z_2) values. And the next iteration the Lorenz system used these (x_2, y_2, z_2) values as input, the output is multiplied with c_2 and sum up with (x_i, y_i, z_i) values to produce the next $(x_{i+1}, y_{i+1}, z_{i+1})$ values. The process to calculate Eq. (3) takes 10 clock cycles. With the FPGA Alchitry Cu, it was obtained a throughput of 112.96 Mbits/s, this is 22.64 times higher that the software throughput obtained with the ESP32. The design in the FPGA includes a serial port at 100 Mbits/s, thus in practice the throughput is limited by this serial port transmission value. A screenshot of the PRNG working in the FPGA is shown in Fig. 6. The FPGA resources utilization is given in Table 1.

5. Discussion

A comparison with recent works is shown in Table 2. Observing Table II, [10] was the first implementation of a PRNG with a fractional chaotic system, it used real numbers and is not possible to calculate exactly the key space using real numbers. In [11] and [2] used also real numbers. One implementation made in [2] used fixed-point arithmetic but is not given an implementation in FPGA. In our work we can obtain

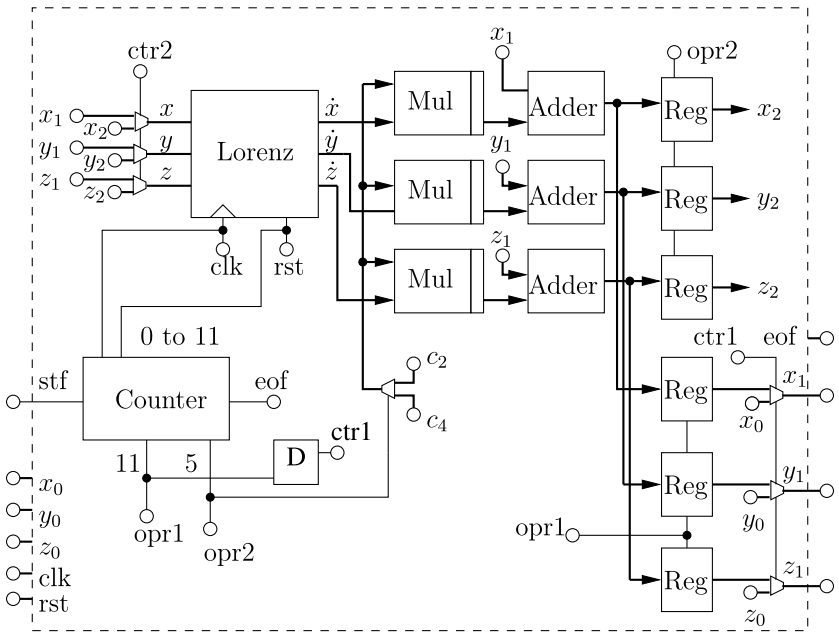


Fig. 5. The PRNG module.

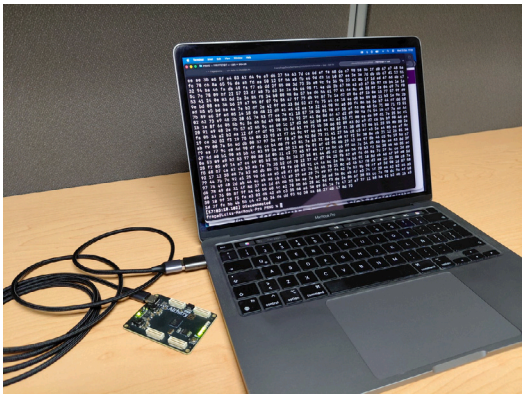


Fig. 6. The PRNG working in the FPGA.

Table 1
Resources utilization of Alchitry Cu FPGA.

	Available	Used
ICESTORM_LC	7680	6923
ICESTORM_PLL	2	0
ICESTORM_RAM	32	0
SB_GB	8	4
SB_IO	256	11
SB_WARMBOOT	1	0

the best fixed-point representation, which at the same time guarantee at least a key space greater or equal to 2^{128}, and also it is implemented in FPGA. The PRNG in [9, Ch 6] is also implemented in FPGA but its key space was not analyzed.

To give one answer to the open problem formulated in [8] to estimate the optimal bit-precision in a fractional system, in our implementation we are using fixed point arithmetic with 1.14.14 number, this is 1 bit for the sign, 14 bits for the integer part, and 14 bits for the fractional part. We are going to analyze here if the used numbers 1.14.14 are correct.

It is necessary to calculate the domain of attraction for the used FLS. The domain of attraction is show in Fig. 7. For this figure were used

Table 2
Comparison with some of the state of art works. N.A. means not applicable or it was not available.

Cite	Integration Method	Representation	Frac. number	Key space	Througput Mbits/sec
[10]	Adomian	Real numbers	N.A.	N.A.	N.A.
[11]	Frequency domain	Real numbers	N.A.	N.A.	4.512
[9, Ch 6]	Memristors	Fixed point	1.5.34	N.A.	5.920
[2]	M2sFRK	Real numbers	N.A.	N.A.	N.A.
[2]	M2sFRK	Fixed point	1.13.50	2^{164}	N.A.
This work	M2sRRK	Fixed point	1.14.36	2^{129}	4.990

the range of values for each variable $x \in [-100, 100]$, $y \in [150, 150]$, and $z \in [34, 162]$ in increments of 16, thus, if we suppose that the shown domains are continuous in z, then is possible to define a zone in $x_0, y_0 \in [-64, 64)$, and $z_0 \in [z_1, z_1 + 128)$, with $z_1 = 34$, where the FLS can be initialized and it will oscillate. With the used 1.14.14 numbers, it will be $(2 \cdot 2^6)^2(2^7)(2^{14})^3 = 2^{14} \cdot 2^7 \cdot 2^{42} = 2^{63}$. This value of 2^{63} is not appropriate for cryptographic applications were the initial values, the key for the PRNG could be guessed by brute force approach. If the key size is equal or greater than 128 bits, then it can resist brute force attacks. Then the number of bits per variable should be $\lceil 128/3 \rceil = 43$, thus $43 - 21 = 22$ bits more per each variable are necessary, then numbers 1.14.36 numbers should be used for a PRNG for cryptographic applications, this give us $(2^7 \cdot 2^{36})^3 = 2^{129}$ different seeds, or keys, than can be used to start the PRNG.

To verify that the PRNG produces non-correlated sequences, three sequences, s_1, s_2, and s_3 were generated with initial conditions $[x_0 = -64, y_0 = -64, z_0 = 34]$, $[x_0 = -64 + \epsilon, y_0 = -64, z_0 = 34]$, and $[x_0 = -64, y_0 = -64 + \epsilon, z_0 = 34]$, with $\epsilon = 2^{-36}$. The produced random s_2 and s_3 are equal. Then, one more bit is necessary to be added in the representation. Finally, with numbers 1.14.37, using the same epsilon $\epsilon = 2^{-36}$, the correlation between sequences s_1-s_2, s_2-s_3, and s_1-s_3, taking the sequences after 1500 initial iterations, the PRNG produces non-correlated sequences as it is shown in Fig. 8.

Table 3 show the results of apply the NIST statistical tests to 100 pseudo-random sequences of 10^6 bits, produced with three different number of bits in the fractional part. It is important to remember that all sequences pass the NIST tests. For 100 sequences, the proportion value in Table 3 must be greater than 96 and the p-value must

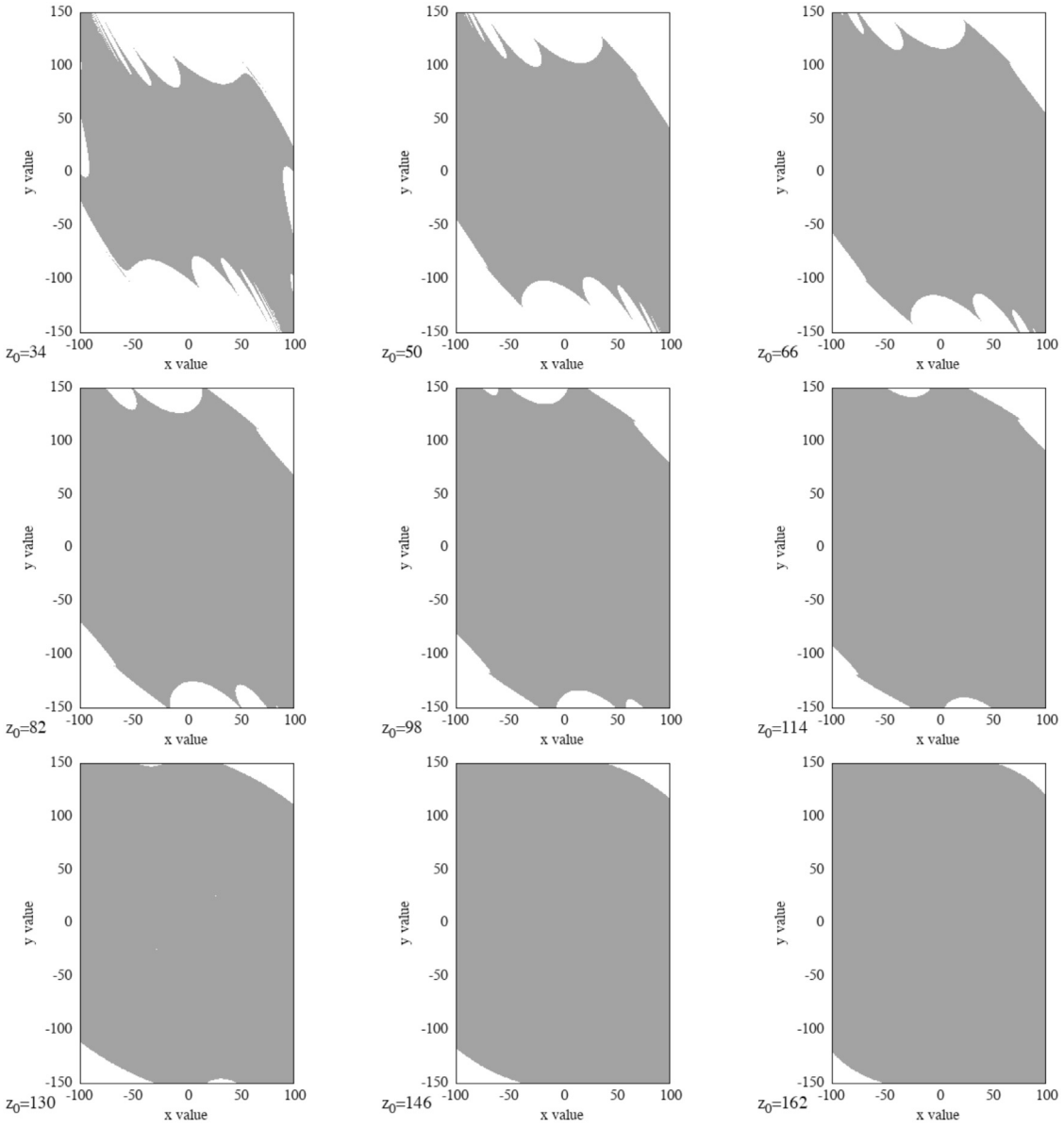


Fig. 7. Domain of attraction for the used FLS.

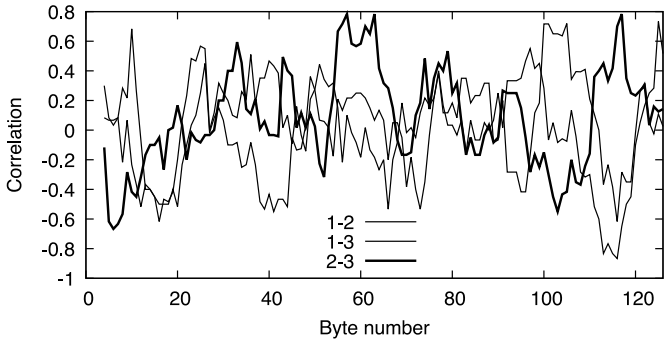


Fig. 8. Correlation among three sequences generated with the proposed PRNG with numbers 1.14.37.

be greater that 0.0001. The sequences produced with 14 bits in the fractional part are pseudo-random but not cryptographic secure. The sequences produced with 36 and 37 bits in the fractional part are pseudo-random and cryptographic secure but they are correlated if the sequences are produced with a change in the last significant bit of the input values. The sequences produced with 37 bits are pseudo-random, cryptographic secure, and non-correlated if a change in the second last significant bit is made.

Therefore, the PRNGs with 14, 36, and 37 bits in the fractional part of the used numbers produce random sequences because all sequences pass the NIST tests. With only 14 bits, generated sequences pass the TestU01 tests. To generate random sequences for an cryptographic application that resist the brute force attack, at least 128 must be used for the PRNG's seed number. This seed is the initial state for the fractional system, thus 36 bits in the fractional part must be used And for our proposed fractional Lorenz system, to generate non-correlated sequences one more bit is necessary in the fractional part, this is, 37 bits. Note that this analysis cannot be made with real numbers because there is not a strait forward mapping from those numbers to its binary representation.

The suggested approach to estimate the optimum number of bits in the fractional part using fixed-point arithmetic is not easy and is very time consuming. The procurement of the graph in Fig. 2 that shows

Table 3
Results of applying the fifteen NIST statistical tests to 100 sequences of 10^6 bits with 15, 36, and 37 bits in the fractional part. The initial state was set equal to $[20, 40, 60]$. Columns show the proportion (prop.) and p-values. All tests passed.

		15 bits		36 bits		37 bits	
	Test name	Prop.	*p*-value	Prop.	*p*-value	Prop.	*p*-value
1	Frequency	100	0.55442	97	0.71975	99	0.26225
2	BlockFrequency	99	0.91141	100	0.73992	100	0.33454
3	CumulativeSums	100	0.85874	98	0.74949	99	0.59098
4	Runs	99	0.88317	97	0.61631	99	0.65793
5	LongestRun	98	0.09659	100	0.97170	100	0.30413
6	Rank	100	0.53415	100	0.71975	99	0.59555
7	FFT	98	0.79814	100	0.16261	99	0.63712
8	NonOverlappingTemplate	99	0.43342	99	0.51306	99	0.44857
9	OverlappingTemplate	98	0.21331	97	0.69931	99	0.69931
10	Universal	100	0.36692	100	0.28967	99	0.08559
11	ApproximateEntropy	100	0.07572	98	0.13728	100	0.71975
12	RandomExcursions	100	0.36883	99	0.64544	99	0.35602
13	RandomExcursionsVariant	99	0.39488	99	0.56672	99	0.23367
14	LinearComplexity	98	0.77919	100	0.61631	99	0.47499
15	Serial	100	0.51470	99	0.61615	100	0.52248

the number of TestU01 statistical tests passed for a given range of bits in the fractional part, and the calculation of the domains of attraction shown in Fig. 7 are both very computational time consuming.

Verilog sources of the designed PRNG are publicly available at https://delta.cs.cinvestav.mx/~fraga/FractionalPRNG.tar.gz.

6. Conclusions

Two implementations of a PRNG, in the STM32 microcontroller and in FPGA, were presented. The implementations give a throughput of 4.99 Mbits/s in software in the EPS32 platform, and 112.96 Mbits/s in hardware in the FPGA using fixed point arithmetic with numbers 1.14.14 (1 bit sign, 14 bits for the integer part, and 14 bits for the fractional part). The generated sequences pass the NIST and TestU01 statistical tests. Analyzing the domain of attraction of the implemented FLA, numbers 1.14.36 must be used to have 2^{129} different seeds for the PRNG (or keys for the PRNG), making the PRNG suitable for cryptographic applications. Finally, one bit more must be added to the fractional part to generate different and non-correlated random binary sequences.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] A. Sai Lekshmi, V. Balakumar, Numerical investigation of fractional order chaotic systems using a new modified runge-kutta method, Phys. Scr. 99 (10) (2024) 105225, http://dx.doi.org/10.1088/1402-4896/ad72b6.
[2] L. de la Fraga, Multi-objective optimization of a fractional-order lorenz system, Fractal & Fract. 9 (3) (2025) http://dx.doi.org/10.3390/fractalfract9030171.
[3] M.F. Tolba, A.M. AbdelAty, N.S. Soliman, L.A. Said, A.H. Madian, A.T. Azar, A.G. Radwan, Fpga implementation of two fractional order chaotic systems, AEU - Int. J. Electron. Commun. 78 (2017) 162–172, http://dx.doi.org/10.1016/j.aeue.2017.04.028.
[4] W.S. Sayed, M. Roshdy, L.A. Said, A.G. Radwan, Chaotic dynamics and fpga implementation of a fractional-order chaotic system with time delay, IEEE Open J. Circuits Syst. 1 (2020) 255–262, http://dx.doi.org/10.1109/OJCAS.2020.3031976.
[5] E. Tlelo-Cuautle, A. Pano-Azucena, O. Guillén-Fernández, A. Silva-Juárez, Analog/Digital Implementation of Fractional Order Chaotic Circuits and Applications, Springer, 2020, http://dx.doi.org/10.1007/978-3-030-31250-3.
[6] S.M. Mohamed, W.S. Sayed, A.H. Madian, A.G. Radwan, L.A. Said, An encryption application and fpga realization of a fractional memristive chaotic system, Electronics 12 (5) (2023) 1219, http://dx.doi.org/10.3390/electronics12051219.
[7] A. Abdelaty, M. Roshdy, L. Said, A. Radwan, Numerical simulations and fpga implementations of fractional-order systems based on product integration rules, IEEE Access 8 (2020) 102093–102105, http://dx.doi.org/10.1109/ACCESS.2020.2997765.
[8] M.P.J. D. Clemente-López, J. Rangel-Magdaleno, A review of the digital implementation of continuous-time fractional-order chaotic systems using fpgas and embedded hardware, Arch. Comput. Methods Eng. 30 (2) (2023) 951–983, http://dx.doi.org/10.1007/s11831-022-09824-6.
[9] L. de la Fraga, J. Rodríguez-Muñoz, E. Tlelo-Cuautle, andOm Number Generators, Springer, 2024, http://dx.doi.org/10.1007/978-3-031-82865-2.
[10] C. Yang, I. Taralova, S. El Assad, et al., Image encryption based on fractional chaotic pseudo-random number generator and dna encryption method, Nonlinear Dynam. 109 (2022) 2103–2127, http://dx.doi.org/10.1007/s11071-022-07534-z.
[11] S.H. AbdElHaleem, S.K. Abd-El-Hafiz, A.G. Radwan, Secure blind watermarking using fractional-order lorenz system in the frequency domain, AEU-Int. J. Electron. Commun. 173.