Nonlinear Dyn (2017) 90:1661–1670
DOI 10.1007/s11071-017-3755-z

ORIGINAL PAPER

Hardware implementation of pseudo-random number generators based on chaotic maps

Luis Gerardo de la Fraga · **Esteban Torres-Pérez** · **Esteban Tlelo-Cuautle** · **Cuauhtemoc Mancillas-López**

Received: 22 February 2017 / Accepted: 16 August 2017 / Published online: 30 August 2017

Abstract We show the usefulness of bifurcation diagrams to implement a pseudo-random number generator (PRNG) based on chaotic maps. We provide details on the selection of the best parameter values to obtain high entropy and positive Lyapunov exponent from the bifurcation diagram of four chaotic maps, namely: Bernoulli shift map, tent, zigzag, and Borujeni maps. The binary sequences obtained from these maps are analyzed to implement a PRNG both in software and in hardware. The software implementation is realized using 32 and 64 bits microprocessor architectures, and with floating point and fixed point computer arithmetic. The hardware implementation is done by using a field-programmable gate array (FPGA) architecture. We developed a serial communication interface between the PRNG on the FPGA and a personal computer to obtain the generated sequences. We validate the randomness of the generated binary sequences with the NIST test suite 800-22-a both in floating point and fixed point arithmetic. At the end, we show that those chaotic maps are suitable to implement a PRNG but according to the hardware resources, the one based on the Bernoulli shift map is better. In addition, another advantage is that the required initial value for the sequences can be within the whole interval $[-1, 1]$, including its bounds.

This work is partially supported by CONACyT-Mexico under project 237991.

L. G. de la Fraga (✉) · E. Tlelo-Cuautle · C. Mancillas-López
Cinvestav, Computer Science Department, Av. IPN 2508, 07360 Mexico City, Mexico
e-mail: fraga@cs.cinvestav.mx

E. Torres-Pérez · E. Tlelo-Cuautle
INAOE, Department of Electronics, Luis Enrique Erro No. 1, 72840 Tonantzintla, Puebla, Mexico
e-mail: etlelo@inaoep.mx

1 Introduction

In 1999, Intel introduced a silicon-based random number generator (RNG) that was incorporated across its motherboards, chipsets and desktop systems. That RNG was the first of Intel's family of primitives, released for data and communications protection within the PC hardware [1]. Intel suggested that to define the concept of a RNG, it is first necessary to understand the idea of randomness, which is typically associated with unpredictability, chance and luck. The area of mathematics provides a precise definition of randomness that can then be applied toward the evaluation of a RNG.

The author in [2] defines random numbers as a sequence of independent numbers with a specified distribution and a specified probability of falling in any given range of values. As a result, the ideal RNG will provide a stream of uniformly distributed, nondeterministic, and independent bits over an infinite data set. In addition, as nowadays known, because the mathematical evaluation of randomness is difficult, it is only possible to use statistical analyses on sample data sets to detect characteristics that point to non-randomness,

and one way is by performing tests from the NIST suite [3].

There are many applications that need RNGs, for example [1]: Entertainment, where lotteries and gambling machines are all based on the use of random numbers, video games use random numbers to influence intelligence heuristics or to vary game play, music and graphics composition by interweaving content with random bits, complex scientific and financial models that use random numbers for simulation; artificial intelligence applications that require random data to determine classification accuracy or neural network behavior, software manufacturers use random data to test programs and algorithms to detect bugs, equation-Solving, cryptography, digital signatures, protected communication protocols, and so on.

A RNG is a physical device or software from which a sequence of random binary numbers is obtained. If this generator is a nondeterministic system; then, it is called a truly RNG (TRNG); otherwise, the system is called a pseudo-RNG (PRNG). Because PRNGs employ a mathematical algorithm for number generation, all PRNGs possess the following properties: A seed value is required to initialize the equation, and the sequence will cycle after a particular period. Therefore, application developers must provide unguessable seed value and an algorithm with a period that is sufficiently long, and also they must verify that the PRNG output contains no correlation or bias.

TRNGs was first developed from continuous-time mathematical models and more recently from discrete-time ones [4–6]. However, it is known that map-based TRNGs have many challenges associated with non-ideal effects, which in the majority of cases are inherent to the integrated circuit (IC) fabrication technology process. Those undesirable effects may degrade or even terminate the normal behavior of the chaotic oscillator at the heart of the TRNG, rendering it useless. As proven in [4], the trajectories of the chaotic oscillator may either converge to parasitic stable points or diverge entirely after a determined number of iterations. The most compact hardware implementation and providing good randomness can be obtained from piecewise-linear one-dimensional maps, for which recent research is performed to mitigate non-ideal effects and trying to provide robust operation of the map through improving IC design techniques, and increasing the entropy [7–9]. In a similar direction is the development of true random bit generators, which can also be implemented from continuous-time mathematical models [10]. However, the very big challenge is how to provide high-frequency bit rate, which is quite difficult by using continuous-time mathematical models and then discrete-time ones are preferred, like the chaotic maps that in fact have simple mathematical descriptions and their behavior is quite rich and complex. In addition, this nature is ideal for the development of IC designs and applications.

The authors in [11] provide guidelines on how to determine the conditions from which a piecewise-linear chaotic map guarantees chaotic behavior, large positive or maximum Lyapunov exponent and high entropy. That approach is quite useful to verify from high-level simulation the characteristics of a chaotic map and after that one can spend more time for the electronic implementation, which is also easier than for a continuous-time chaotic oscillator. As one can see, chaotic maps have also a simpler mathematical description, which are relatively easy to be implemented in hardware, for example by using field-programmable gate arrays (FPGAs), which are quite useful for fast prototyping as highlighted in [12].

Among the available chaotic maps, the logistic map has been the most used to develop PRNGs [13,14]; however, other maps can provide better randomness so that one can find applications of multi-modal maps [15], combinations or mixing of chaotic maps [16], quantum chaotic map [17], and so on. Those applications of maps for the development of PRNGs have the same goal: improving randomness, high maximum Lyapunov exponent and entropy. Therefore, in this article, we show that the Bernoulli shift map, tent, and zigzag maps are quite useful for implementing a PRNG. We analyze also the Borujeni map [18] (an extension of logistic map) but, as it will be shown, this map is not suitable for our PRNG implementation. The rest of this article is organized as follows: Sect. 2 describes the four used chaotic maps. The generation of binary sequences from these maps is detailed in Sect. 3. The software and hardware implementations and the NIST test results are listed in Sect. 4. Finally, Sect. 5 summarizes the conclusions.

2 Chaotic maps

This section is devoted to describe the four chaotic maps tested in this work to generate PRNGs: Bernoulli shift map, tent map, zigzag, and Borujeni maps [18].

All these maps share the characteristics to have only a single control parameter, and also a single Lyapunov exponent (LE), which made them easier to analyze. From their model we show their bifurcation diagram, and the theoretical and practical behavior of LE values according to the corresponding control parameter variation.

We show that the bifurcation diagram is quite useful to perform a fast exploration of the control parameter values which produce the output of interest for a specific application of the chaotic map.

2.1 Bernoulli shift map

This map is defined by two linear functions as:

$$x_{n+1} = \begin{cases} bx_n - a, & \text{if} \quad x_n \geq 0, \\ bx_n + a, & \text{if} \quad x_n < 0. \end{cases} \tag{1}$$

Also Eq. (1) can be written as $x_{n+1} = bx_n - a\,\text{sign}(x_n)$, where b (the slope) is the parameter which controls the stochastic properties of the chaotic system, and a is a scale factor which simply increase or decrease the product bx_n, and bound the output values inside the range $[-a, a]$.

The calculated bifurcation diagram for Bernoulli shift map is shown in Fig. 1a for $a = 1$, from which one can see that for b values within interval [0, 1), the map oscillates between two stable fixed points. At $b = 1$ it is an unstable point; for $b \in (1, 1.4]$ there is a non-uniform dispersion in the system output values, which improves when b approaches to 1.4. Finally, for interval [1.4, 2.0), the b values produce the biggest dispersion in the system output values: all output values in Fig. 1 for $b \in [1.4, 2.0)$ cover the whole output range of $[-1, 1]$. For b values greater or equal to 2, the map is unstable and its output tends to infinity for large values of n.

The Lyapunov exponent can be defined as the average rate of exponential divergence or convergence of very near trajectories in the phase space. Any system with at least one positive LE is defined as a chaotic system, and reflecting its magnitude the temporal scale when its dynamics become unpredictable. The LE for a chaotic map can be calculated using the formula:

$$\lambda = \frac{1}{n}\sum_{i=0}^{n-1} \ln\left|\frac{\mathrm{d}x_{n+1}}{\mathrm{d}x_n}\right| \tag{2}$$

For the Bernoulli shift map in Eq. (1), its derivative is equal to b, then the theoretical value for the LE is equal to $\ln(b)$. In Fig. 1a it is shown the theoretical and the estimated LE value calculated with 10,000 samples of the map for $b \in [1.030, 1.995]$ by using the software TISEAN [19]. It is expected that a chaotic system becomes more unpredictable when its positive LE value increases.

2.2 Tent map

This map is defined with the equation:

$$x_{n+1} = \begin{cases} u\ x_n, & \text{for} \quad x_n \in \left[0, \frac{1}{u}\right], \\ \frac{u}{u-1}(1 - x_n), & \text{for} \quad x_n \in \left(\frac{1}{u}, 1\right]. \end{cases} \tag{3}$$

The bifurcation diagram for this map is shown in Fig. 1b. The x_{n+1} is equal to 0 for the control parameter $u \in [0, 1]$. LE is equal to $\lambda = \ln(u) + \ln(|u-1|)(1/u-1)$. For $u \in [0, 1]$, $\lambda < 0$ and it is not shown in Fig. 1b because we are interested in the positive and greater values for λ. As it was computed for Bernoulli shift map, Fig. 1b shows the LE using TISEAN software, for 10,000 samples and $u \in [1.005, 2.000]$.

2.3 Zigzag map

The equation that defines the zigzag map is:

$$x_{n+1} = \begin{cases} -m\left(x_n + \frac{2}{|m|}\right), & \text{for} \quad x_n \in \left(-1, -\frac{1}{|m|}\right], \\ mx_n, & \text{for} \quad x_n \in \left(-\frac{1}{|m|}, \frac{1}{|m|}\right], \\ -m\left(x_n - \frac{2}{|m|}\right), & \text{for} \quad x_n \in \left(\frac{1}{|m|}, 1\right]. \end{cases} \tag{4}$$

The bifurcation diagram for this map is shown in Fig. 1c. Its LE is equal to $\lambda = \ln(|m|)$. For $|m| < 1$ its behavior is not chaotic, while for intervals $m \in (2, 1)$, $(1, 2)$, $[3, 2)$, and $(2, 3]$ its behavior is chaotic. For the last two intervals, x_{n+1} fills completely the output range of $[-1, 1]$. For $|m| = 2$, the map converges to 0.

2.4 Borujeni map

Borujeni map [18] is defined by equation:

$$x_{n+1} = \begin{cases} r x_n (1 - x_n), & \text{for} \quad x_n < 0.5, \\ r(x_n - 0.5)(x_n - 1.5) + \frac{r}{4}, & \text{for} \quad x_n \geq 0.5. \end{cases} \tag{5}$$

This map is a modification of the logistic map presented in [18]. The modification in (5) improves its bifurcation as it is shown in Fig. 1d.

The derivative of Eq. (5) is:

$$\frac{\mathrm{d}x_{n+1}}{\mathrm{d}x_n} = \begin{cases} r - 2rx_n, & \text{for} \quad x_n < 0.5, \\ 2r(x_n - 1), & \text{for} \quad x_n \geq 0.5. \end{cases} \tag{6}$$

Thus, it is not possible to obtain an analytic expression for its LE depending of r using (2). Theoretical LE in Fig. 1d was obtained by summing 10^6 samples in (6) for each independent variable r.

The entropy measurement is a characteristic of a binary sequence. Now we are going to describe in next section how a binary sequence is generated, and also how to analyze its related entropy.

3 Binary sequences

A binary sequence can be generated using Eq. (7):

$$b_{n+1} = \begin{cases} 0 & \text{if} \quad x_{n+1} < q, \\ 1 & \text{if} \quad x_{n+1} \geq q, \end{cases} \tag{7}$$

where x_{n+1} is calculated with any map in Eqs. (1), (3), (4), or (5). The threshold q is 0 for Bernoulli and zigzag maps, and equal to 0.5 for the tent and Borujeni maps. This is a discrete system. Thus at every new value of x_{n+1} a new binary number b_{n+1}, with 0 or 1 values, is generated.

The produced sequence is said to be "pseudo-random" because exactly the same sequence can be reproduced using the same initial conditions. In Eq. (7) the initial condition is the initial value for x_0.

Now, once we have a binary sequence, it is possible to calculate its entropy. The entropy is a measure about how unpredictable a binary sequence is. For a good TRNG or PRNG, the generated bits should be impossible to predict given the previous bits. The entropy can be estimated using the formula:

$$H(b) = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{8}$$

where b is a sequence built in this case with symbols {0,1}, and with probability distributions $\{p_1, p_2, \ldots, p_n\}$. Applying Eq. (8) to a sequence, its entropy is equal to 1 if it is an unpredictable sequence, for a totally predictable sequence its entropy is equal to 0.

Figure 2 shows the entropy measurements for all the four tested chaotic maps. For the Bernoulli shift map in Fig. 2a, its entropy is calculated with two methods: one that uses a long sequence of 10^6 bits and then calculates the histogram [7], the occurrence of small sequences of two, three or more binary symbols of length 10 is computed. In [7] sequences of length 1–16 of a total of 1 million bits were used to calculate those histograms. The histograms form the estimated probabilities to calculate Eq. (8). The taken estimated entropy will converge to the true entropy as the number of taken sequences tend to infinity. The second method uses the *context-tree weighting* (CTW) algorithm [20]. The taken measurement by this algorithm is related with lossless compression schemes. The entropy bounds the performance of the strongest lossless compression, which can be realized in theory by using the typical set or in practice by using Huffman, LempelZiv or arithmetic coding schemes. Then, the performance of existing data compression algorithms is often used as a rough estimate of the entropy of a block of data [8,20]. The study in [9] concludes that the CTW method is the most effective to measure the entropy, with the most accurate and reliable results with much less bits, but it takes also more computation time. For all sequences which entropy was calculated in Fig. 2, $x_0 = 0.2$ was used, also CTW was calculated with sequences of 5000 bits, and the histogram method was applied to sequences of 10^6 bits. As both measure methods are in good agreement, as it can be seen in Fig. 2a, we use the histogram method, which takes lesser computational time, to take the entropy measurements in all other cases with sequences of 10^6 bits.

The sequences obtained directly from any of the maps have not entropy equal to 1, as an unpredictable sequence has (see Fig. 2). Then, a post-processing procedure must be implemented on the generated sequences to increase the entropy of the sequences. We

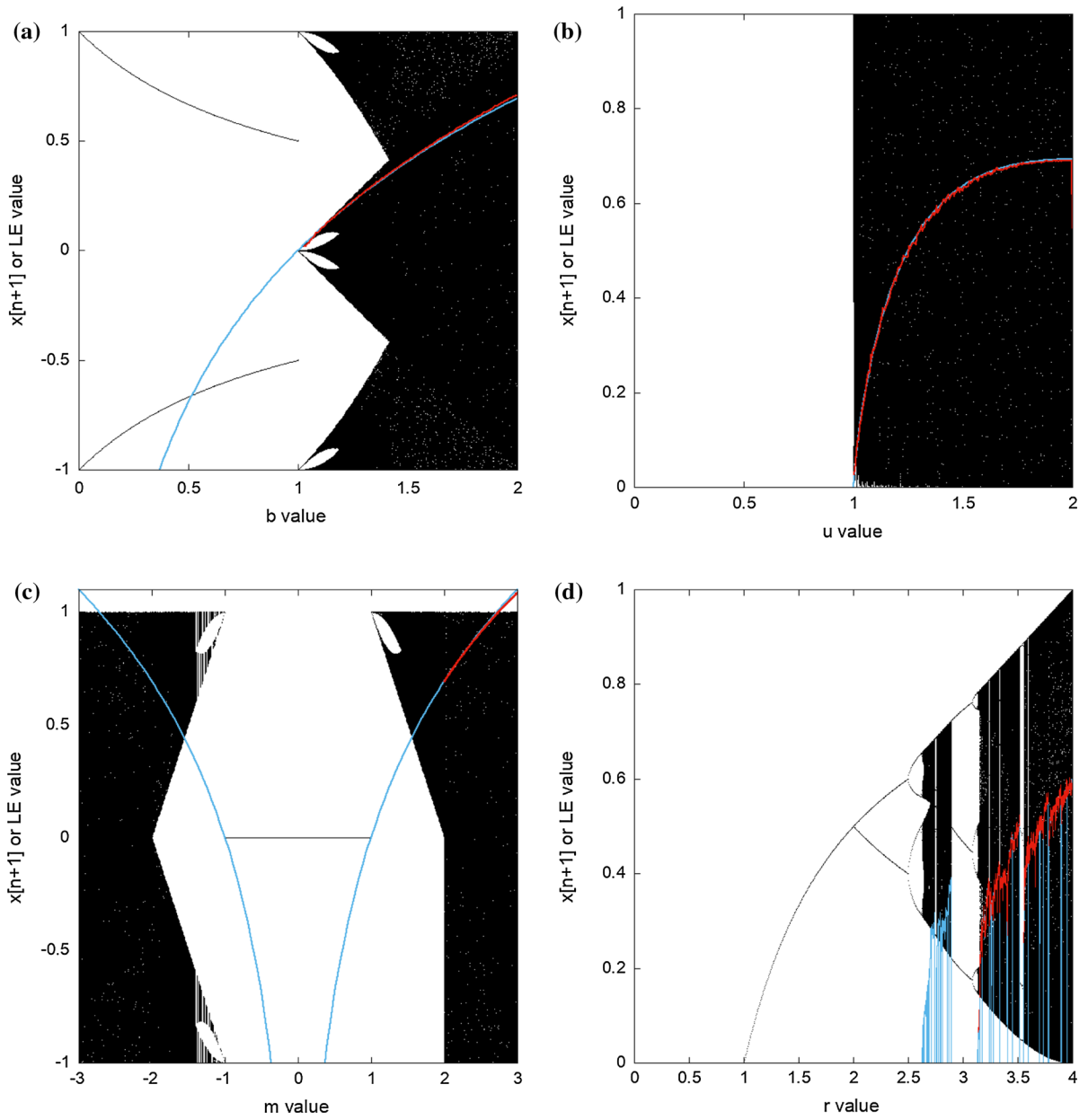

Fig. 1 Bifurcation diagram, in *black*, the four chaotic maps. The *blue line* shows the theoretical value for the Lyapunov exponent (LE). The *red line* shows the LE value calculated from every 10,000 samples of the map with software TISEAN [19]. These experimental LE values are shown only for positive LEs for Bernoulli in (**a**) and tent in (**b**) maps, for zigzag map in (**c**) it was calculated only inside interval $m \in [2, 3]$, and for Borujeni map in the interval $r \in [3, 4]$ **a** Bernoulli shift map. Graph for $a = 1$ in (1), **b** Tent map, **c** Zigzag map **d** Borujeni map. (Color figure online)

use the *bit counting redundancy reduction technique* used in [21]: The original sequence is divided in blocks of 5 bits, then a new bit for a new sequence is generated applying the XOR operation on the 5 bits of each block. In Fig. 2 we can see that the entropy increases to values very near to 1.0, for all the maps, when this technique is applied.

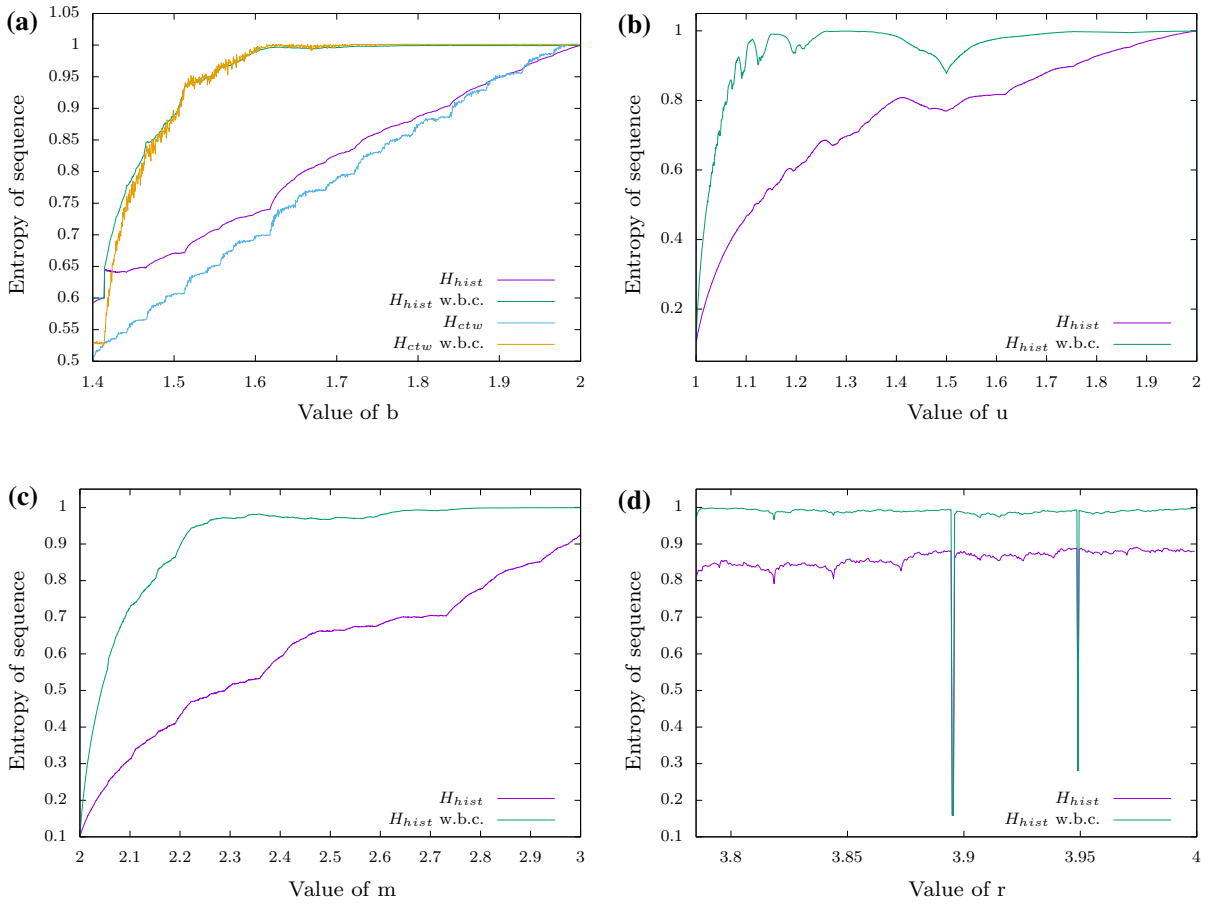

Fig. 2 Entropy of sequences generated with the four chaotic maps for an increment of 0.0005 in the respective independent variable. H_{hist} indicates the entropy calculated with the method based on histograms, and H_{ctw} the entropy calculated with CTW algorithm. w.b.c. means a sequence post-processed with bit counting redundancy reduction technique. **a** Bernoulli shift map, **b** Tent map, **c** Zigzag map, **d** Borujeni map

4 PRNG implementation and NIST test

The National Institute of Standards and Technology (NIST) of the United States of America had created a free software for testing pseudo and random number generators [(P)RNGs]. Specifically, NIST said that the test is for (P)RNGs for cryptographic applications, where randomness is a crucial characteristic. The package is available at, [1] and according to NIST [3, Ch. 5] it will address the problem of evaluating (P)RNGs for randomness. It will be useful in: Identifying (P)RNGs that produce weak (or patterned) binary sequences, designing new (P)RNGs, verifying that the implementations of (P)RNGs are correct, studying (P)RNGs described in standards, and investigating the degree of randomness by currently used (P)RNGs.

We applied the NIST test to our sequences generated using floating point numbers (64 bits, double precision). We used 100 sequences each one 10^6 bits. Results for the Bernoulli shift map are shown in Table 1. The sequences pass all NIST test.

Also, we implement our proposed system in hardware using FPGAs. First we check which is the number of bits necessary to implement the Bernoulli map with fixed point arithmetic and maintain its characteristics, i.e. $H > 0.999$. From Eq. (1), as $a = 1$, $b \in [1.95, 2)$, $x_n \in [-1, 1]$, then the product bx_n will be always lower

[1] http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html.

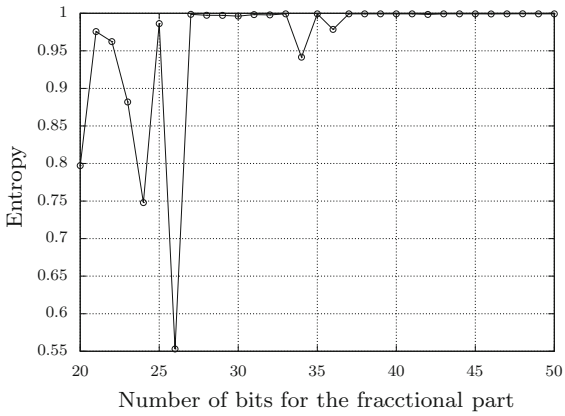

Fig. 3 Entropy versus the number of bits used in the fractional part to implement our PRNG with fixed point arithmetic. This graph is discrete, lines are added for clarity purpose

than 1.0 and greater than -1.0, therefore a single bit is necessary for the integer part. Question here is how many bits are necessary for the integer part. In Fig. 3 one sees the number of bits of the fractional part against the entropy, one can observe in this figure that with 37 or greater number of bits, the entropy value stabilizes. We pass a sequence generated with 37 bits to NIST test, but it fails to pass tests that use the biggest number of bits: *rank*, *random excursion*, *random excursion variant*, *"universal statistical"*, *nonperiodic templates*, and *overlapping templates*. We analyzed this situation and finally sequences generated with 50 bits in the fractional part pass all the test, as can be seen in Table 1.

For the tent map fixed point implementation, $u \in [1.995, 2)$ in (3) must be used, thus the expression $u/(u-1)$ is $1.995/0.995 \approx 2.00502$ and is greater than 2, therefore two bits are necessary to implement the integer part. Also, the same number of bits for the fractional part that for Bernoulli map must be kept. Table 1 lists NIST test results on its integer implementation.

For the zigzag map, $m \in [2.96, 3]$ in (4) must be used, then 2 bits for the integer part must be used, and more bits are required for the fractional part: 54. With these conditions, generated sequences pass all NIST test as it is shown in Table 1.

For the Borujeni map, even using the bit counting technique the sequences have not its entropy bigger than 0.999, for any value of control variable r (see Fig. 2d). Then this map is not suitable for generated sequences that pass NIST test.

A note here, we simulated first our PRNG in software; thus, it is necessary a multiplier of 64 bit numbers, which gives a number of 128 bits. In a machine

Table 1 Results of applying NIST test to our PRNG implemented in floating and fixed point arithmetic

	Test name	Bernoulli shift map				Tent map		Zigzag map	
		Floating point		Fixed arithmetic		Fixed arithmetic		Fixed arithmetic	
		p value	Prop.	p value	Prop.	p value	Prop.	p value	Prop.
1	Frequency	0.419021	0.98	0.699313	0.98	0.911413	1.00	0.514124	0.99
2	Block frequency	0.678686	1.00	0.678686	0.99	0.883171	0.98	0.075719	0.99
3	Cumulative sums+	0.534648	0.98	0.710364	0.98	0.118131	1.00	0.265410	0.97
4	Runs	0.366918	1.00	0.719747	0.99	0.304126	0.99	0.162606	1.00
5	Longest run of ones	0.455937	1.00	0.236810	1.00	0.213309	1.00	0.122325	0.99
6	Rank*	0.001030	0.99	0.935716	0.99	0.911413	1.00	0.971699	0.99
7	Spectral DFT	0.003712	1.00	0.494392	0.99	0.010988	0.99	0.026948	0.99
8	Nonperiodic templates*+	0.474987	0.99	0.488481	0.99	0.441509	0.99	0.479177	0.99
9	Overlapping templates*	0.191687	0.97	0.924076	0.98	0.401199	1.00	0.048716	1.00
10	Universal statistical*	0.455937	0.99	0.964295	0.98	0.437274	1.00	0.534146	0.99
11	Approximate entropy	0.013569	1.00	0.779188	0.99	0.304126	1.00	0.366918	0.99
12	Random excursion*	0.580777	0.99	0.225431	0.98	0.285487	1.00	0.355641	0.99
13	Random excursion variant*	0.525278	0.99	0.445670	0.99	0.309401	1.00	0.224650	0.99
14	Serial+	0.301860	1.00	0.456707	1.00	0.465962	0.99	0.331048	0.99
15	Linear complexity	0.816537	0.97	0.129620	0.98	0.759756	0.99	0.719747	0.98

with 64 bits architecture and using C language and gcc compiler, there exist an integer of 128 bits through type `__int128`. We implement also our PRNG in a 32 bit computer, a RaspBerry Pi 2, for this architecture it was necessary to implement the multiplication of two numbers of 64 bits through the multiplication of four `short int` numbers of 16 bits each one using Knuth algorithm [2, Sec 4.3] modified for two signed numbers. Of course, all implementations with fixed point arithmetic give the same sequences.

In Table 1 symbol '+' at the end of a test name means reported values are the average values; symbol '*' means that these tests used 100 sequences of 10^6 bits each one. The rest of tests uses 100 sequences of 10^4 bits.

The proposed architecture to evaluate the PRNG in hardware using the Bernoulli shift map is shown in Fig. 4a. The arithmetic is performed using also fixed point representation: we use 1 bit for integer part, 50 bits for fractional part, and 1 bit for the sign. The basic blocks are the multiplier and the adder, block labeled as acc-xor computes the parity of the complement of five bits of sign. The multiplexer *mux* selects between input *s_Xi* or feedback from the adder. Finally, the communication interface contains a demultiplexer to feed bytes to the RS-232 interface. The hardware implementations for the other two maps are shown in Fig. 4b for the tent map, and in Fig. 4c for the zigzag map.

In Table 2 we list the results obtained after place and route phase on Spartan 3E (xc3s500e-4fg320) device using ISE 14.7. The amount of logical resources (LUTs and registers) represents only 6% of the available ones, because the fix-point-multiplier was implemented using embedded multipliers. The architecture is more efficient for the Bernoulli shift map: it consumes less FPGA resources and then is faster (see Table 2). Our design could generate pseudo-random bits at the speed of 7.38 Mbps without the RS-232 interface, but with it its speed is limited to 115,200 b/s.

A last question could be how does the initial value for any map affect the generated binary sequences. For Bernoulli shift map, Fig. 5a shows the entropy calculated for sequences of 10^6 bits, with the histogram method, for $x_0 \in [-1, 1]$ with increments of 0.01 units (a value of $b = 1.95$ was used to generate this graph.) For all initial values the entropy is greater than 0.999, and we can observe that x_0 value does not affect the sequence properties. For tent map, $x_0 \in (0, 1)$ was used, the entropy of generated sequences is shown in

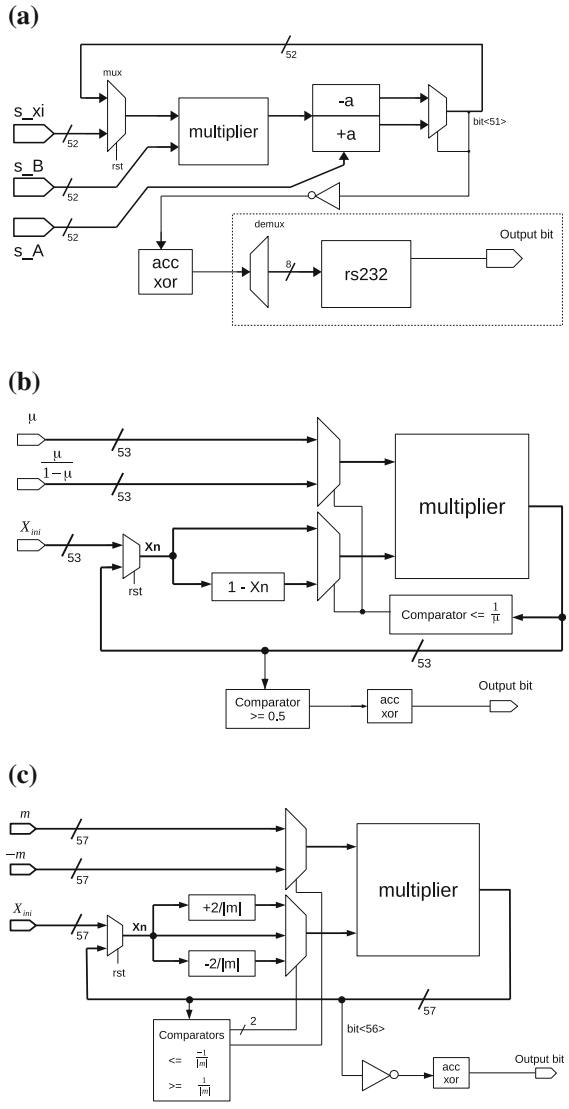

Fig. 4 Architectures of the proposed PRN generator. *Bold lines* represent buses, *thin lines* represent only one bit. The communication module inside *dashed rectangle* in (**a**) is not drawn in (**b**) and (**c**). **a** PRNG implementation using Bernoulli shift map, **b** PRNG with tent map, **c** PRNG with zigzag map

Fig. 5b, and also all they have values above 0.999. Here one must observe that x_0 must be not in the bounds of the interval, this is $x_0 \neq 0$ or 1. The same result is kept for the zigzag map in Fig. 5c, $x_0 \in [-1, 1]$ without consider 0 value; all sequences have entropy greater than 0.999. If $x_0 = 0$, Eq. 4 never changes out this value. Zigzag map collapses if its input value is equal to 0.

All programs and VHDL designs are publicly available at http://cs.cinvestav.mx/~fraga/PRNGs.tar.gz.

Table 2 Implementation resources and results

	Bernoulli	Tent	Zigzag
LUTs	575	769	928
Registers	108	108	108
Slices	315	402	476
Multipliers	9	14	16
Maximum Frequency (MHz)	36.90	33.26	31.33
Throughput (Mb/s)	7.380	6.652	6.266

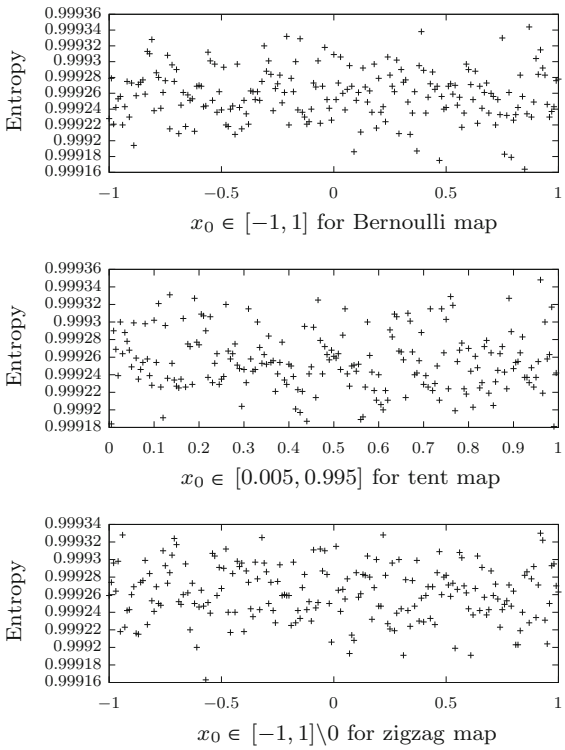

Fig. 5 Entropy of sequences generated with the three maps with fixed point implementation versus the initial value x_0

5 Conclusions

We provided the hardware implementation of a PRNG based on three chaotic maps: the Bernoulli shift map, tent, and zigzag maps. We analyzed first the parameters of each map, their bifurcation diagrams, to compute the positive Lyapunov exponent, and to obtain the greater entropy ($b \in [1.95, 2)$ in Eq. (1), $u \in [1.995, 2)$ in (3), $m \in [2.96, 3]$ in (4)) To increase the entropy of the generated sequences, it was necessary to apply simple post-processing: a sequence bit is generated by calculating the XOR of five bits. These sequences have entropy greater than 0.999. For the sequences generated using the Borujeni map, applying our methodology, it was not possible to obtain entropy greater than 0.999; then, this map is not suitable for generated PRNG with our proposal.

We implemented the PRNG with fixed point arithmetic. The numbers that maintain the properties of sequences generated with floating point numbers are 2.50 (1 sign bit, 1 integer bit, and 50 bits for the fractional part) for the Bernoulli shift map, 3.50 for the tent map, and for the zigzag map was necessary to use 3.54 numbers. The hardware implementations were tested on a FPGA Spartan 3E. A throughput of 7.38 Mb/s was obtained although it is highly limited with the speed of the RS-232 interface.

The binary sequences generated using the Bernoulli shift, tent, and zigzag maps as the core block, passed all NIST tests which prove that our designs are suitable to implement a PRNG.

Finally, it can be appreciated that the best implementation is using the Bernoulli shift map because it requires lesser FPGA resources, provides the highest throughput, and the initial value for the sequences can be within the whole interval $[-1, 1]$, including its bounds.

Acknowledgements Authors would like to thank the anonymous reviewers for their valuable comments which have helped to improve the quality of this article.

References

1. Jun, B., Kocher, P.: The Intel Random Number Generator. Cryptography Research Inc. white paper, San Francisco (1999)
2. Knuth, D.E.: The Art of Computer Programming Volume 2, Seminumerical Algorithms, 3rd edn. Addison Wesley, Boston (1998)
3. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications, 27 April 2010. SP800-22rev1a.pdf file, last date checked: 18 Feb 2016. http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html
4. Valtierra, J.L., Tlelo-Cuautle, E., Rodríguez-Vázquez, Á.: A switched-capacitor skew-tent map implementation for random number generation. Int. J. Circuit Theory Appl. **45**(2), 305–315 (2017)
5. Nejati, H., Beirami, A., Ali, W.H.: Discrete-time chaotic-map truly random number generators: design, implemen-

tation, and variability analysis of the zigzag map. Analog Integr. Circuits Signal Process. **73**(1), 363–374 (2012)
6. Cicek, I., Pusane, A.E., Dundar, G.: A novel design method for discrete time chaos based true random number generators. Integr. VLSI J. **47**(1), 38–47 (2014)
7. Talçin, M.E.: Increasing the entropy of a random number generator using n-scroll chaotic attractors. Int. J. Bifurc. Chaos **17**(12), 4471–4479 (2007)
8. Moddemeijer, R.: On estimation of entropy and mutual information of continuous distributions. Signal Process. **16**(3), 233–246 (1989)
9. Gao, Y., Kontoyiannis, I., Bienenstock, E.: Estimating the entropy of binary time series: methodology, some theory and a simulation study. Entropy **10**, 71–99 (2008)
10. Yalcin, M.E., Suykens, J.A.K., Vandewalle, J.: True random bit generation from a double-scroll attractor. IEEE Trans. Circuits Syst. I Regul. Pap. **51**(7), 1395–1404 (2004)
11. Valtierra Sánchez de la Vega, J.L., Tlelo-Cuautle, E.: Simulation of piecewise-linear one-dimensional chaotic maps by verilog-A. IETE Tech. Rev. **32**(4), 304–310 (2015)
12. Tuncer, T.: Implementation of duplicate trng on fpga by using two different randomness source. Elektronika ir Elektrotechnika **21**(4), 35–39 (2015)
13. Murillo-Escobar, M.A., Cruz-Hernández, C., Cardoza-Avendaño, L., Méndez-Ramírez, R.: A novel pseudorandom number generator based on pseudorandomly enhanced logistic map. Nonlinear Dyn. **87**(1), 407–425 (2017)
14. Wang, Y., Liu, Z., Ma, J., He, H.: A pseudorandom number generator based on piecewise logistic map. Nonlinear Dyn. **83**(4), 2373–2391 (2016)
15. García-Martínez, M., Campos-Cantón, E.: Pseudo-random bit generator based on multi-modal maps. Nonlinear Dyn. **82**(4), 2119–2131 (2015)
16. François, M., Grosges, T., Barchiesi, D., Erra, R.: Pseudo-random number generator based on mixing of three chaotic maps. Commun. Nonlinear Sci. Numer. Simul. **19**(4), 887–895 (2014)
17. Akhshani, A., Akhavan, A., Mobaraki, A., Lim, S.-C., Hassan, Z.: Pseudo random number generator based on quantum chaotic map. Commun. Nonlinear Sci. Numer. Simul. **19**(1), 101–111 (2014)
18. Borujeni, S., Ehsani, M.: Modified logistic maps for cryptographic application. Appl. Math. **6**(5), 773–782 (2015)
19. Hegger, R., Kantz, H., Schreiber, T.: Practical implementation of nonlinear time series methods: the TISEAN package. Chaos **9**, 413–435 (1999)
20. Kennel, M.B., Mees, A.I.: Context-tree modeling of observed symbolic dynamics. Phys. Rev. E **66**, 056209-1–056209-11 (2002)
21. Stojanovski, T., Pihl, J., Kocarev, L.: Chaos-based random number generators. Part II: practical realization. IEEE Trans. Circuits Syst. Fundam. Theory Appl. **48**(3), 382–385 (2001)