Research Article

Digital Noise Generator Design Using Inverted 1D Tent Chaotic Map

Leonardo Palacios-Luengas,¹ Gonzalo Isaac Duchén-Sánchez,¹
José Luis Aragón-Vera,² and Rubén Vázquez-Medina^{1,2}

¹ Instituto Politécnico Nacional, ESIME-Culhuacán, Santa Ana 1000, 04430 Mexico, DF, Mexico

² Centro de Física Aplicada y Tecnología Avanzada, UNAM, Boulevard Juriquilla 3001, Juriquilla 76230, Querétaro, Mexico

Correspondence should be addressed to Rubén Vázquez-Medina, ruvazquez@ipn.mx

Received 19 November 2011; Accepted 30 August 2012

Academic Editor: Wiesław Kuzmicz

Copyright © 2012 Leonardo Palacios-Luengas et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper shows a digital noise generator designed in FPGA, based on a variant of the one-dimensional (1D) chaotic tent map (T-1D). The T-1D map is a piecewise linear 1D chaotic map that defines the statistical behavior of the generated sequences using its control parameter. In this way, the proposed noise generator is a highly competitive alternative in cryptographic systems when the statistical behavior of the sequences is closer to the uniform statistical distribution. The proposed system uses the inverted tent chaotic map (IT-1D), which has the same statistical behavior as the T-1D map. The fundamental algorithm used in this system was developed based on a 64-bit double precision format according to the numerical representation of floating point numbers defined in the IEEE-754 standard. The proposed system is analyzed using mechanical statistic tools and some statistical tests defined in the NIST 800-22SP (USA) standard. The main contribution of this work is the possibility of generating binary sequence of pseudorandom appearance by a procedure implemented in an FPGA device that translates real numbers to natural numbers preserving the statistical properties of sequences of real numbers that can be generated with the tent chaotic map in its original definition domain.

1. Introduction

Chaotic systems have interesting features such as their sensitivity to initial conditions, the use of control parameters, and the ergodicity and mixing properties. These features have a great impact and application on cryptographic systems because they match diffusion and confusion properties of the cryptographic systems required by Information Theory. In this paper, the piecewise linear chaotic maps are the chaotic systems used in the implementation of the proposed digital noise generator. Basically, a chaotic map is defined in a specific interval $I = [a, b] \in \mathbb{R}$ and generates a dispersion process based on its sensitive dependence to initial conditions. This kind of maps can offer uniformly distributed binary sequences and then can be used as deterministic generators of unpredictable appearance sequences. There are some published works related with the application of Chaos Theory to the generation of noise sequences and

cryptographic systems. In 2011, Rojas-Lopez [1] proposed a block cipher using a chaotic map and scaling and discretizing processes. In 2010, Qi et al., [2] proposed a switched chaotic system to generate pseudorandom sequences. In 2011 Dabal and Pelka [3] presented a stream cipher based on a logistic map for real time applications. Also in 2010, Pande and Zambreno [4] presented a stream cipher based on a variant of the logistic map.

On the architecture of the system proposed in this paper the one-dimensional tent chaotic map (T-1D map) was used. This map has been extensively studied [5, 6] due to its simplicity of hardware and software implementations. The T-1D map is generally evaluated on the domain of real numbers, but when it is implemented in a digital system two options must be considered: (1) the scaling and discretizing processes of the T-1D map to be used on the natural numbers domain, and (2) the use a dictionary of binary words related to the real numbers produced by the T-1D map. In

FIGURE 1: Graphical representation of tent chaotic map: (a) T-1D and (b) IT-1D maps.

both alternatives the final sequence must be restricted to the interval $[0, 2^n]$, where n is the number of bits in each resulting binary word.

For the implementation of the T-1D map on an FPGA device it is necessary to take account of different aspects, like the limitation of logic resources in the FPGA device, the efficiency of the designed circuit, and the criteria of cost and velocity. Additionally, a digital noise generator must have good statistic properties to be used in cryptographic systems, that is, its statistical behavior must be similar to the uniform statistical distribution.

In this work a variant of the T-1D map has been proposed to implement the digital noise generator. This variant of T-1D map is named inverted tent map (IT-1D), which has a similar statistic behavior to the T-1D map. IT-1D map has been used due to the possible reduction of the logic elements used to design the digital noise generator in an FPGA device.

The architecture of the proposed digital noise generator is based on four modules: (a) Control System Module (CSM), (b) Arithmetic Module (AM), (c) Register Module (RM), and (d) Constant Register Module (CRM). The functionality of each module is described in Section 3 of this paper.

The algorithm that define the IT-1D map was developed using a 64-bit double precision format, according to the floating point numerical representation defined in the standard IEEE-754 [7] and it is analyzed using mechanical statistic tools such as the bifurcation diagram and the trajectories diagram.

2. Implementation of Inverted Tent Map

The T-1D map family is a subclass of piecewise linear maps (PWL), which can be described according to [8, 9]. But, in this work, this map family can be expressed by

$$f_{\mu}(x_n) = \begin{cases} 2\mu x_n + \left(\frac{1-\mu}{2}\right), & a \leq x_n \leq \frac{1}{2}, \\ -2\mu(x_n - 1) + \left(\frac{1-\mu}{2}\right), & \frac{1}{2} < x_n \leq b. \end{cases} \quad (1)$$

Equation (1) is an iterated function when $x_{n+1} = f_{\mu}(x_n)$, which generates orbits $\mathcal{O}_{\mathfrak{R}} = \{x_0, x_1, x_2, x_3, \dots\}$ that are sequences of real numbers defined at $[a, b] \in \mathfrak{R}$ with $a < b \in [0, 1]$. Each orbit produced by iterating (1) depends on an initial condition x_0 , which is known but arbitrarily selected and a control parameter, μ . Both parameters are defined on the interval $[a, b]$. Figure 1(a) shows the graphic representation for the T-1D map, which has a maximum at $x = 0.5$ and two piecewise linear segments in the intervals, $[0, 0.5]$ and $(0.5, 1]$.

On the other hand, the IT-1D map is expressed by (2) and graphically represented by Figure 1(b) one has

$$f_{\mu}(x_n) = \mu(2|x_n - 0.5|) + \left(\frac{1-\mu}{2}\right). \quad (2)$$

To observe the dispersion process produced by these chaotic maps, (1) and (2) must be evaluated using different values of μ to generate the bifurcation diagram. These diagrams are showed in Figures 2(a) and 2(b) to (1) and (2), respectively. Notice that when $\mu \rightarrow 1$ the sequences are more disperse in both cases.

As mentioned before, there are two alternatives for the implementation of the T-1D and IT-1D maps in a digital electronic circuit: (1), realize the scaling and discretizing processes on the chaotic map, or (2) the use of a dictionary that associates each real number produced by the chaotic map to a binary number of n bits according to some associating rules. Considering the analysis realized by Martínez-Nonthe et al. [10], a discretized and scaled chaotic map is an approximation of the original chaotic map; thus, a new domain interval at the integer numbers appears. In other words, $f_{\mu}(x_n)$ defined at $[0, 1] \in \mathfrak{R}$, is transformed in a new function $\sigma_{\mu}(x_n)$, where n is the iteration number of the chaotic map is transformed in a new function $\sigma_{\mu}(x_n)$ defined at $[0, 2^i] \in \mathbb{N}$, where $\sigma_{\mu}(x_n)$ depends on the precision of the used binary word considering i bits. This alternative produces binary sequences, which do not have a statistical behavior congruent with the original chaotic map, since the numbers rounding produced by the discretization process induces an error. This error is propagated and increased with

FIGURE 2: Bifurcation diagram of tent maps: (a) T-1D and (b) IT-1D maps.

each iteration of the chaotic map, and then the statistical behavior of the resulting sequence of real numbers is strongly affected. In this sense, because T-1D map is a PWL map and the representation of each number has a finite size, an initial condition applied to the chaotic map is really a macroscopic number, which is an ensemble of numbers in the form

$$x = 0 \cdot b_1 b_2 b_3 b_4 \dots b_n. \quad (3)$$

This expression corresponds to the numbers set with cardinality 2^n . Thus, the digital noise generator will produce real numbers, which are represented by a finite set instead of an infinite set. Therefore, it is necessary to define in the FPGA device the registers with the appropriate length. This consideration defines the memory resources required for the FPGA device.

Another alternative to avoid the scaling and discretizing processes of the chaotic maps was described by Martínez-Nonthe et al. in 2011 [11]. This alternative is used in the architecture of the digital noise generator proposed in this work. This alternative for a chaotic map $f_\mu : [a, b] \rightarrow [a, b] \in \mathbb{R}$ defines a regular partition $A = \{A_1, \dots, A_S\}$ of S intervals, in which each interval A_t with $t = 1, \dots, S$ has a length given by

$$L(A_t) = L = \frac{b-a}{S}. \quad (4)$$

In this case, the generated orbits are conformed by natural numbers depending on the subinterval to which the respective real number belongs. This new orbit of natural numbers can be expressed by $\wp_{\mathbb{N}} = \{y_0, y_1, y_2, y_3, \dots\}$, in $[0, 2^i] \in \mathbb{N}$. The relationship between $\wp_{\mathbb{R}}$ and $\wp_{\mathbb{N}}$ is given by

$$y_m = \text{dec}(A_t), \quad \text{if } (I_{A_t}(x_m)) = 1, \quad (5)$$

where $m = 1, 2, 3, 4, \dots$, $t = 1, 2, 3 \dots S$, $\text{dec}(A_t)$ represents the decimal value of the subinterval A_t , and $I_{A_t}(x_m)$ is

the belong function, which indicates if x_m is in A_t according to following expression:

$$I_{A_t}(x) = \begin{cases} 1, & \text{if } x \in A_t \\ 0, & \text{if } x \notin A_t. \end{cases} \quad (6)$$

The second alternative is more complex to be designed than the first, because it considers the implementation of the chaotic map and a specific associating function. According to Martínez-Nonthe et al. in 2011 [11], this second alternative offers better results, since the statistical behavior of the orbits conformed by natural numbers is closest to the statistical behavior of the orbits conformed by real number produced by the chaotic map. Notice that in this alternative the floating point representation of the real numbers is the unique factor that will affect the resulting orbits. Using the same initial condition, the first alternative generates orbits of natural numbers very different from the real orbits due to the discretization process of the chaotic map. Therefore, their statistical behavior will be different.

In order to implement the IT-1D map in Xilinx FPGA devices, it is needed to consider that (2) has four constants: -0.5 , 0.5 , 1 , and 2 , which must be represented using a floating point format according to the digital core design of Xilinx defined on [12]. Therefore, the arithmetic operations must be implemented considering that the length of the constants must be the same as the data length (64 bits). The definition of the constants in the hardware implementation allows the realization of the arithmetic operations directly and, so, the temporary storage of data in some register will be avoided. In this way, there are only two variables, μ and x_n , in (2) that must be considered in calculations. Notice that (1) has the same constants used in (2). In both cases, the term $0.5(1 - \mu)$ must be considered through a sum. Next, the number of operations to perform must be defined. In (2) three sums and three multiplications are made. The absolute value of the first term in (2) is obtained through a logic AND operation between the sum and the known

FIGURE 3: (a) Digital noise generator. (b) Hardware architecture to considering T-1D or IT-1D maps.

TABLE 1: Arithmetic operations required.

Chaotic map	Operations (multiplier module)	Operations (adder module)
T-1D	3	2
	3	3
IT-1D	3	3

value 0x7FFFFFFFFFFFFFFF. This operation changes the most significative bit of the sum to zero. Regarding (1), in first instance, two cases must be identified comparing the value of x_n with the constant 0.5. In the first case, when $x_n \leq 0.5$, three multiplications and two sums are made. On the second case, when $x_n > 0.5$, three sums and three multiplications are required and, additionally, the constants -1 and -2 must be declared.

Due that the proposed system will be considered in a communications system to protect voice signals, we considered that the voice samples are obtained from the PCM frames, which have a length of 13 bits. In this way, it is a requirement to design a digital noise generator that must produce 13-bits binary numbers. Therefore, the expression

in (3) must consider $n = 13$ and in (4) the partition size must be $S = 2^{13}$. In the hardware implementation of the proposed system additional considerations were made and they are described in the following sections. Table 1 shows a comparison of the number of arithmetic operations realized with both chaotic maps. It is remarkable that also the operations number is similar, but in the case of T-1D map some other processes are needed, like a comparator to evaluate if $x_n \leq 0.5$ or $x_n > 0.5$ in (1).

3. FPGA Architecture of the Noise Generator

Figure 3 shows the proposed digital noise generator. In Figure 3(a), the noise generator can be designed using the T-1D or IT-1D map, which must be initialized with 64-bit real numbers but, at the end, the noise generator must produce natural numbers using a 13-bit representation. Figure 3(b) shows the architecture of the proposed noise generator. This architecture is based on (2) and uses a 64-bit data bus for the information exchange. The implementation of (1) uses a similar architecture. However, there are two variants: (a) CSM has additional state controls which are used to an adequate representation; (b) the resulting architecture is based on the use of RAM instead of the registers block.

3.1. Control System Module (CSM). It is based on a finite state machine to communicate with RM using the 128-bit PC register, which introduces together μ and x_0 in parallel form. Next, the $d1$ - $d2$ register separates μ and x_0 in a 64-bit length each one. Additionally, the output of AM is introduced to the register named *evaluation*, which indicates if the operations required in (1) or (2) were completed. If the operations were completed the content of the *evaluation* register is sent to CRM; in other cases, that information is sent to the PC register to be stored in RM.

3.2. Arithmetic Module (AM). It is defined to carry on the sum and multiplication with 64-bit format considering the IEEE-754 standard. These arithmetic operations are described as follows.

3.2.1. Floating Point Sum Module. This module calculates the sum of two floating point numbers considering that two numbers z and w can be defined according to

$$z = m_x 2^{x_e}, \quad w = m_y 2^{y_e}, \quad (7)$$

where $e = E - E_{\text{bias}}$, E is the normalized value of the exponent value, E_{bias} is a constant equal to 1023, which depends on the precision used in the calculations, and m represents the mantissa. Also, this Arithmetic Module must cover the representations defined by

$$z + w = (m_x 2^{x_e - y_e} + m_y) 2^{y_e} = (m_x + m_y 2^{y_e - x_e}) 2^{x_e}. \quad (8)$$

Figure 4 shows the flowchart of the sum module implementation. This algorithm is based on the standard IEEE-754. For the implementation of a floating point sum

FIGURE 4: Algorithm of floating point sum module.

the following considerations must be observed. (1) For each number, exponent and mantissa must be previously separated. (2) A new representation for the original exponent must be obtained using the lower possible integer and then the mantissa must be right shifted the number of positions that correspond, according to the difference between original exponent and its new representation; (3) The exponent of the result obtained for the realized arithmetic operations must be equal to the greater exponent between the exponents of the two involved numbers. (4) The sum and subtraction operations must be performed using the mantissa. (5) If it is necessary, the result obtained from arithmetic operations must be normalized. This process consists of shifting the bits of the mantissa from left to right; thus the exponent value will change. (6) In the calculations of these arithmetic operations the overflow conditions must be checked.

3.2.2. Floating Point Multiplicative Module. The operations made by this module are showed in Figure 5 and when two

variables are considered the arithmetic multiplication can be represented by

$$a * b = m_x 2^{x_e} \times m_y 2^{y_e} = (m_x \times m_y) 2^{x_e + y_e}. \quad (9)$$

This module consists of the following steps. (1) Equal to step (1) in the floating point sum module. (2) Realize the sum or subtraction between exponents of each variable. (3) Multiply the mantissas of the two involved numbers and then the resulting sign must be determined in the operation realized in the step (2). (4) Normalize the result obtained in step (2) according to the procedure indicated in step (1). (5) The overflow conditions must be verified.

Finally, Table 2 shows the time access for the two implemented operations for the IT-1D map.

3.3. Constant Register Module (CRM). The registers defined in this module are 64-bit constants arrays. However, due to the savings of logic elements required on the FPGA device the reduction of constants to 24 bits were made. CRM is showed

FIGURE 5: Floating point multiplier module.

TABLE 2: Latencies and data widths of AM.

Module	Data widths	Latency (clock cycles)
Inputs	a, b (64 bit)	—
Multiplier	64 bit	8
Adder	64 bit	8
Output	Result (64 bit)	—

FIGURE 6: Constant Register Module.

in Figure 6 and its functionality is defined according to the flowchart showed in Figure 7.

Additionally, it must be noticed that the cardinality of the defined domain for the digital noise generator is $2^{13} = 8192$. In this way, in CRM four registers are defined. (a) *BInf*. It is based on an array composed by 33 registers with 24 bits each one. These registers define the limits of 32 subsets, which correspond to the partition of the domain for the digital noise generator. The prefix of the real number labeled as $prefix_{24}(x_n)$ will be searched in *BInf* as a first approximation to the regular partition A. Notice in Figure 7 that j is the auxiliary variable in the searching process on *BInf*. Therefore,

the final searching to find the corresponding number is limited to 256 possibilities for each subset; this searching will be performed using *InterV13b*. (b) *InterV13b*. It defines 32 subsets with 256 elements each one. $prefix_{24}(x_n)$ will be searched now in this register allowing the determination of the particular natural number that must be used to represent each real number. The length of this register is $2^{13} \times 24$ bits. (c) *MMDirInf* and *MMDisSup*. These registers contain the natural numbers in a representation of 13 bits, which are used to generate the output sequence. Notice in Figure 7 that z is the auxiliary variable in the searching process on *InterV13b* and it is used to compare the values in *MMDirInf* with the real number x_n . When the value of x_n is found in *MMDirInf*, z contains the associated natural number, which has 13-bit length. These natural numbers will be associated to the numbers found using *InterV13b*. These registers have a length of 32×13 bits, and they work according to Figure 6, which shows the functionality of these registers by a flowchart. Additionally, two auxiliary variables are defined to compare the real number produced by chaotic map with the found number in the searching process described previously. These auxiliary variables are sup_{64} and inf_{64} , which are conformed concatenating the found 24-bit number and a 40-bit binary word, which contains 40 zeros. In Figure 7, the symbol $\odot$ has used to represent the concatenation function.

4. Results

A chaotic map generates a dispersion process using specific values of the parameter control and the initial condition on a particular interval. This process can be observed using three alternatives: (a) the bifurcation diagram (see Figure 8), (b) the histogram of the produced sequences for different values of parameter control and initial conditions (see Figure 9), and (c) the trajectories diagrams, which show how

FIGURE 7: Flowchart of Constant Register Module.

the iterated chaotic map covers gradually the plane formed by x_n and x_{n+1} depending on the parameter control and initial conditions (see Figure 10).

To observe the dispersion process produced by the chaotic maps the tent map was implemented using an FPGA device SPARTAN 3E XC3S500e [13]. Figures 8(a) and 8(b) show the bifurcation diagrams for the T-1D and IT-1D maps, respectively, considering a 2^{13} domain and natural numbers in the output sequences. Notice that Figure 8 has a similar behavior to Figure 2. The dispersion process produced by chaotic maps occurs when $\mu \geq 0.5$ starting with two bands and then when $\mu \geq 0.73$ the dispersion process occurs in a lonely band until reaching a maximal dispersion when $\mu \rightarrow 1$. The bifurcation diagrams in Figure 8 have been built considering that the control parameter is increased using steps of $\Delta_\mu = 0.0001$ and an initial condition $x_0 = 0.45$. This behavior can be observed also in the histograms of produced sequences (see Figure 9) and the trajectories diagrams (see Figure 10) if the parameter control is changed.

Figures 10(a) and 10(b) show the trajectories diagrams considering 200 iterations, $x_0 = 0.45$ and $\mu = 0.9999$. Notice that the statistical behavior is close to a uniform distribution, and then the produced sequences can be considered pseudo-random sequences. Table 3 shows the hardware requirements when the tent maps are implemented in an FPGA device using (1) and (2).

Table 4 shows the hardware requirements when the IT-1D map is implemented in an FPGA device when the regular partition is defined to translate the real numbers to natural numbers. Figure 11 shows the evidence of the sequences of natural numbers produced by the noise generator proposed in this work. These evidences have been obtained using programs defined with MATLAB and ISIM-Xilinx toolboxes.

TABLE 3: Hardware required by T-1D and IT-1D maps in a FPGA device.

Logic utilization	Available	Utilization (IT-1D)	Utilization (T-1D)
Number of slice flip flops	9,312	2,169 (23%)	2,689 (28%)
Number of 4 input LUTs	9,312	2,376 (25%)	2,674 (28%)
Number of occupied slices	4,656	1,853 (39%)	2,146 (46%)
Total number of 4 inputs LUTs	9,312	2,535 (27%)	2,867 (30%)
Number used as logic	—	2,313	2,575
Number used as a route thru	—	159	193
Number used as shift register	—	63	99
Number of bonded IOBs	232	15 (6%)	194 (83%)
Number of MULT18X18SIOs	20	16 (80%)	16 (80%)
Average fan-out of nonclock nets	—	2.43	2.60
Number of RAMB16s	20	—	1 (5%)

Finally, to show that the sequences produced by the digital noise generator proposed in this work have random appearance, some statistical tests of the NIST 800-22SP

FIGURE 8: Bifurcation diagrams obtained from sequences of natural numbers produced by the tent maps: (a) T-1D map and (b) IT-1D map.

FIGURE 9: Histogram of the sequences produced by tent map considering $\mu = 0.9999$, $x_0 = 0.45$ and 1000 iterations of the chaotic map: (a) T-1D map, and (b) IT-1D map.

TABLE 4: Hardware resources required by IT-1D when the regular partition is defined to translate the reals numbers to natural numbers.

Logic utilization	Available	Utilization (IT-1D)
Number of slice Flip flops	9,312	2,287 (24%)
Number of 4 input LUTS	9,312	2,926 (31%)
Number of occupied slices	4,656	2,167 (46%)
Total number of 4 inputs LUTS	9,312	3,098 (33%)
Number used as logic	—	2,863
Number used as a route-thru	—	172
Number used as shift register	—	63
Number of bonded IOBs	232	15 (6%)
Number of MULT18X18SIOs	20	16 (80%)
Average fan-out of nonclock nets	—	2.66

standard were applied. Table 5 shows the statistical tests used to evaluate the sequences.

TABLE 5: Statistical tests applied to the sequences produced by the digital noise generator.

Test	P value	Result
Monobit Frequency Test	0.2301393404	True
Block Frequency Test	0.2727182697	True
Runs Test	1.9119631770	True
Spectral Test	0.6036850480	True
Maurer's Universal Test	0.4712780830	True
Cumulative Sums Test	0.2471130325	True
Random Excursions Test	0.3140488026	True
Random Excursions VarTest	0.2093446041	True

These results are congruent with the statistical behavior showed in Figure 9 relative to the natural number sequences produced by the digital noise generator proposed in this work. Due to the simplicity of the tent map the statistical tests relative to linear complexity were not considered.

FIGURE 10: Diagram showing the evolution of the trajectories using (a) T-1D map and (b) IT-1D map for 200 iterations with $\mu = 0.9999$ and $x_0 = 0.45$.

FIGURE 11: Sequences of natural numbers produced by noise generator using tent map. (a) simulation results using programs defined in Xilinx and (b) simulation results using programs defined in MATLAB.

5. Conclusions

This paper presents a configurable architecture in a small capacity FPGA device that generates digital noise sequences. The proposed system is optimized and the resultant sequences are used to be combined with 13-bit length data. Some goals that were achieved on the design of the proposed system are the following, the architecture of the digital noise generator proposed in this work was implemented using two versions of the tent chaotic map, T-1D and IT-1D maps. Both alternatives have similar performance characteristics. The experimental work has considered evaluating the functionality of the architecture proposed. Regarding the precision to represent natural and real numbers, it is possible to have similitudes with the processes defined in a personal computer. The results obtained from MATLAB represent the evidence of the behavior of the proposed system. The digital noise generator has been developed using a sequential architecture, where the CSM coordinates the activities of different modules. All registers used in this implementation have been depurated to avoid the delays and excessive clock

cycles. An example of this one is that the Arithmetic Module takes 8 clock cycles for placing data after an operation.

Considering the bifurcation diagrams, it can be assumed that the natural numbers sequences produced by the digital noise generator have a similar statistical behavior as the real numbers sequences produced by the tent chaotic map.

The architecture of the digital noise generator produces noise sequences using the tent map when the control parameter is $\mu \geq 0.73$. This behavior is shown by bifurcations diagrams, trajectories diagrams, and statistical distribution. The precision to represent the natural or real numbers is an important factor when the noise sequences are produced an FPGA device. For this reason, the proposed system was defined using the IEEE-754 standard. If the precision to represent the numbers is increased the demand of logic resources will be increased and the design of the system will be more complicated.

The proposed digital noise generator avoids the scaling and discretization processes on the chaotic map; instead of this an associating rule was defined to relate each real number

with a natural number. The implementation of this function in an FPGA device considers a searching process established in two steps using a 24-bit prefix of each real number. We consider that this process must be optimized because the natural numbers in the noise sequences do not have the same permanence time in the output of the system (see Figure 11).

On the other hand, using some statistical test defined in the NIST 800-22SP standard, the pseudorandomness of the produced sequences was verified. Considering that the tent map is a simple chaotic system, statistical tests relative to the linear complexity were not applied.

Acknowledgments

The authors thank the financial support of the SIP IPN 20120052 and ICYTDF 270/2010 Projects. L. Palacios-Luengas (CVU-373990, #244598) acknowledges the scholarship provided by CONACYT.

References

- [1] C. E. Rojas-Lopez, *Block cipher using the one-dimensional Tent chaotic map* [M.S. dissertation], IPN, Querétaro, Mexico, 2011.
- [2] A. Qi, C. Han, and G. Wang, "Design and FPGA realization of a pseudo random sequence generator based on a switched chaos," in *Proceedings of the International Conference on Communications, Circuits and Systems (ICCCAS '10)*, pp. 417–420, July 2010.
- [3] P. Dabal and R. Pelka, "A chaos-based pseudo-random bit generator implemented in FPGA device," in *Proceedings of the 14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS '11)*, pp. 151–154, April 2011.
- [4] A. Pande and J. Zambreno, "Design and hardware implementation of a chaotic encryption scheme for real-time embedded systems," in *Proceedings of the International Conference on Signal Processing and Communications (SPCOM '10)*, pp. 1–5, July 2010.
- [5] S. Callegari, G. Setti, and P. J. Langlois, "CMOS tailed tent map for the generation of uniformly distributed chaotic sequences," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '97)*, pp. 781–784, Hong Kong, Hong Kong, June 1997.
- [6] H. Nejati, A. Beirami, and Y. Massoud, "A realizable modified tent map for true random number generation," in *Proceedings of the IEEE 51st International Midwest Symposium on Circuits and Systems (MWSCAS '08)*, pp. 621–624, August 2008.
- [7] Institute of Electrical and Electronics Engineers, IEEE754 Standard for Binary Floating-Point Arithmetic, 1984.
- [8] D. Luengo-García, *Estimación óptima de secuencias caóticas con aplicación de comunicaciones* [Ph.D. dissertation], Universidad de Cantabria, Santander, Spain, 2006.
- [9] R. C. Hilborn, *Chaos and Nonlinear Dynamics: An Introduction for Scientists and Engineers*, Oxford University Press, 2nd edition, 2000.
- [10] J. A. Martínez-Ñonthe, L. Palacios-Luengas, M. Cruz Irissón, J. A. Díaz-Méndez, and R. Vázquez-Medina, "Digital Noise Generator using one-dimensional chaotic maps," in *Proceedings of the 7th International Conference on Low Dimensional Structures and Devices*, AIP Conference Proceedings, 2012.
- [11] J. A. Martínez-Ñonthe, J. A. Díaz-Méndez, M. Cruz-Irissón, L. Palacios-Luengas, J. L. Del-Río-Correa, and R. Vázquez-Medina, "Cryptosystem with one dimensional chaotic maps," in *Computational Intelligence in Security for Information Systems*, vol. 6694 of *Lecture Notes in Computer Science*, pp. 191–198, Springer, Berlin, Germany, 2011.
- [12] Digital Core Design, Alliance Core Data Sheets for Floating-Point Arithmetic, 2001, <http://www.xilinx.com/>.
- [13] Xilinx, Spartan-3E FPGA Family: Data Sheet, 2011, <http://www.xilinx.com/>.